IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
WACO DIVISION

ZENTIAN LTD., Plaintiff, v. APPLE INC. Defendant.	C.A. No. 6:22-cv-122 **JURY TRIAL DEMANDED**

COMPLAINT FOR PATENT INFRINGEMENT

Plaintiff Zentian Ltd. ("Zentian") demands a trial by jury on all issues so triable and, for its complaint against Defendant Apple Inc. ("Apple") alleges as follows:

INTRODUCTION

1. Apple introduced the "Hey Siri" feature in September 2014.

2. "Hey Siri" allows users of Apple devices, including iPhone, iPad, Apple Watch, and Mac personal computers, among others, to engage Apple's virtual assistant by using only their voice and without physically handling their device. Apple's virtual assistant does so by recognizing sounds uttered by the device user and converting those acoustic waveforms into information that can be mapped into actions by the operating system, such as searching the internet, playing a song, making to-do lists, setting alarms, creating a calendar invite, or any number of other tasks. This process is known as automatic speech recognition or simply speech recognition.

3. Apple's implementation of speech recognition that enables users to wake Apple's virtual assistant using the wake phrase "Hey Siri" and/or ask "Siri" questions hands free was invented and patented by Zentian.

4. Zentian is a technology innovator specializing in hardware designs for speech recognition circuits and methods. Zentian was founded in 2000 by Mark Catchpole using funds from the U.K. government's "Smart Award" program—a program that provides funding for innovative projects with the potential for significant growth.

5. In the early 2000s, conventional speech recognition systems were ill-equipped to perform a feature like "Hey Siri" on a mobile electronic device in a fast and power-efficient manner, let alone facilitate the back-and-forth conversational interactions that Apple users have with Siri today. In large part, this shortcoming existed because systems in the art focused on improving algorithms and the models used by speech recognition systems to achieve higher word recognition accuracy and other goals. Such implementation focused on incremental improvements in software that ran on power and memory intensive computers or servers. Another contributing factor was the fact that mobile devices like Apple's iPhone or Apple Watch are typically used in noisier environments, such as on public transportation or near a busy road, requiring a more complex model and more intensive computation to obtain adequate results from the speech recognition system.

6. Zentian, however, focused on speech recognition hardware, including specialized chip designs to enable speech recognition in cost, space, and power constrained environments like mobile electronic devices. Zentian's vision was to study speech recognition systems and to design silicon chips that would accelerate the speech recognition process by off-loading all or part of that processing from software running on a CPU to a bespoke silicon system. This approach differed from that of leading speech recognition companies that were focused on incrementally improving software implementations of speech recognition. Zentian focused on how to design better digital electric circuits (*e.g.*, better hardware).

7. Critically, speech recognition systems can be divided into several discrete processes. First, an audio front-end generates processed speech parameters (referred to as feature or spectral vectors) that describe the characteristics of an audio signal. Second, a calculating means or circuit compares the similarity between feature vectors and a model of predetermined acoustic states. This stage is often referred to as "distance calculations" because the output is a probability of how similar (or different) a feature vector is from a stored acoustic state. And third, a search process attempts to find the most likely set of words spoken from a given vocabulary based on the distance calculations.

8. Among the prototypes Zentian developed was a hardware accelerator for the acoustic stage (audio front end and distance calculation) of the speech recognition process that worked alongside a software implementation of the search stage running on a conventional processor. The accelerator demonstrated a two-times speedup of the speech recognition software compared to a conventional speech recognition software-only solution. Moreover, the prototype proved that segregating the speech recognition process onto bespoke hardware components could help solve the shortcomings of conventional systems and enable more advanced features like Apple's "Hey Siri."

9. In the 2000s, Zentian had a number of conversations about its speech recognition accelerator with leading speech recognition companies, such as Nuance Communications Inc. and IBM, as well as cellular carriers, such as Hutchison 3G UK Limited. But each speech recognition company expressed a desire to focus on software-only solutions and an unwillingness to divert resources to "longer-term" ideas for speech recognition systems. At the 2006 GSM World Congress in Barcelona, a senior executive at Qualcomm, one of the preeminent suppliers of silicon chips for mobile electronic devices, tellingly stated that speech recognition was "not even in my

Amazon / Zentian Limited
Exhibit 1014
Page 3

top ten wish list." Put simply, the industry was not focused on hardware-centric solutions for speech recognition. Zentian was.

10. Zentian's work resulted in several U.S. patents, including U.S. Patent Numbers 7,587,319; 7,979,277; 10,971,140; 10,062,377; and 10,839,789 (collectively, the "Asserted Patents"). These patents have now been widely cited on patents filed by a host of companies that provide speech recognition technology or end-user products that utilize speech recognition technology, including Samsung, Google, Amazon, Nuance Communications Inc., Microsoft, and Voci Technologies Inc.

11. Today, speech recognition capabilities are in all kinds of products from phones to cars to smart speakers. Apple's implementation of speech recognition technology in its products "on-device" involves a combination of general-purpose processors combined with hardware accelerators—the very speech recognition circuit and/or system Zentian invented and claimed in the Asserted Patents.

12. Although Zentian has met with Apple on more than one occasion to discuss its technology, Apple has no license to use Zentian's patented inventions.

13. Nevertheless, Apple's infringing on-device speech recognition features have, on information and belief, generated substantial sales of Apple products. For example, in the 12 months ending on September 25, 2021, Apple reported over $191 billion in revenue for sales of iPhone, over $35 billion in revenue for sales of Mac computers, over $31 billion in revenue for sales of iPad, and over $38 billion in revenue for sales of wearables, Home and Accessories.[1] On average, end users make multiple requests for Siri per day through "Hey Siri." In turn, Siri and the

[1] Apple Inc., *Condensed Consolidated Statements of Operations,* Q4, Fiscal Year 2021, https://www.apple.com/newsroom/pdfs/FY21_Q4_Consolidated_Financial_Statements.pdf.

Amazon / Zentian Limited
Exhibit 1014
Page 4

fluidity of the speech recognition processes that are behind it (facilitated by the hardware architecture on which it runs) have become a material driver of user demand for Apple products.

14. Apple has infringed and is still infringing, directly and indirectly, one or more claims of the Asserted Patents by making, using, selling, and offering to sell in the United States, including in this District, and importing into the United States, products that implement the Asserted Patents' speech recognition technology. Examples of potentially infringing products include Apple's iPhone, iPad, Watch, AirPods, HomePod, HomePod Mini, Apple TV set top box, and Mac computers (including the Mac Pro, which is manufactured in Austin, Texas), in their various models and iterations (the "Accused Products").[2]

15. This list of Accused Products is non-limiting and based on information currently available to Zentian. Zentian reserves the right to modify the list of Accused Products as discovery progresses, including as new products are released during the pendency of this case.

THE PARTIES

16. Zentian is a private corporation incorporated and registered in the United Kingdom, with its registered address at 27 Queens Close, St. Ives, Cambridgeshire, England, PE37 5QD.

17. Zentian is the owner of all rights, title, and interest in the Asserted Patents.

18. On information and belief, Defendant Apple Inc. is a California corporation with a principal place of business at 1 Apple Park Way, Cupertino, California. Apple designs, manufactures, and sells throughout the world a wide range of products, including mobile devices and computers that incorporate Zentian's patented technologies.

[2] *See, e.g.* Devices that support "Hey Siri" https://support.apple.com/en-us/HT209014. In addition to recognition of the wake phrase "Hey Siri," other potentially infringing features include dictation of text messages and notes with integrated applications and on device speech recognition for Siri user requests.

Amazon / Zentian Limited
Exhibit 1014
Page 5

JURISDICTION AND VENUE

19. This is an action for patent infringement arising under the patent laws of the United States of America, Title 35 of the United States Code.

20. This Court has subject matter jurisdiction over the matters asserted in this Complaint under 28 U.S.C. §§ 1331 and 1338(a) and 35 U.S.C. § 271 *et seq*.

21. This Court's exercise of personal jurisdiction over Apple complies with both the Texas long-arm statute and the Due Process Clause of the Fourteenth Amendment. Apple has purposely directed its activities toward Texas and has availed itself of the privileges of conducting activities in this state.

22. Apple conducts continuous and systematic business in this District, including by selling infringing products to residents of this District, by soliciting business from residents in this District, and on information and belief, by manufacturing certain Accused Products in this District.

23. Apple also maintains offices in Austin, Texas, and markets, offers, manufactures, and distributes the Accused Products throughout the United States, including in the state of Texas and in this judicial District.

24. Apple's Austin offices are located at 12545 Riata Vista Cir, Austin TX 78727.

25. Venue is proper under 28 U.S.C. § 1391(b) and (c) and 28 U.S.C. § 1400(b).

26. Apple has committed acts of infringement in this District and has a regular and established place of business here, including offices in Austin, Texas, where it maintains a workforce of more than 7,000 employees.[3]

27. Apple has maintained a physical presence in this District for years, opening its first offices in Austin in the 1990s. And Apple continues to grow its footprint in this District with a

[3] *See Apple Expands in Austin*, Press Release, (Nov. 20, 2019) https://www.apple.com/newsroom/2019/11/apple-expands-in-austin/.

planned $1 billion, 3-million-square-foot, 133-acre campus that will house up to 15,000 employees.[4] The Apple "Austin campus" will include a 192-room hotel,[5] and provide a wide variety of engineering, research and development, operations, finance, sales and customer support jobs.[6] On information and belief, Apple also owns significant real estate in and around Austin, Texas, including for example, a 217,000 square-foot building at 320 S. Capital of Texas Hwy.[7]

28. Apple's new Austin campus is responsible for manufacturing the Mac Pro, one of the Accused Products in this case. The Mac Pro and other Accused Products also contain components from suppliers across the state of Texas, including suppliers in this District. According to Apple's Chief Executive Officer, Tim Cook, "[b]uilding the Mac Pro, Apple's most powerful device ever, in Austin is both a point of pride and . . . deepening our close bond with the city and the talented and diverse workforce that calls it home."[8]

29. Apple also employs individuals in this District who, on information and belief, contribute to Apple's design and sales of infringing products. For example, and without limitation, Apple employs over 100 individuals in the Austin metropolitan area on the "Siri Team" and/or with the title of SoC engineer.[9]

---

[4] *Id*.
[5] Juli Clover, *Apple Adding 192-Room Hotel to Upcoming Austin Campus*, MacRumors (May 20, 2020) https://www.macrumors.com/2020/05/20/apple-austin-campus-hotel/.
[6] Alex Caprariello, Billy Gates, *Apple's Billion-Dollar Austin Campus Nearly Finished, Move-in Date Set*, KXAN (Apr. 26, 2021), https://www.kxan.com/news/business/apples-1b-austin-campus-nearly-finished-move-in-set-for-2022/.
[7] Parimal Rohit, *Apple Buys Austin Office Building on Capital of Texas Hwy*, Austin Business Journal (Aug. 25, 2021), https://www.bizjournals.com/austin/news/2021/08/25/apple-buys-capital-ridge-austin.html.
[8] *Id.*; *see also Apple Expands in Austin*, Press Release (Nov. 20, 2019) https://www.apple.com/newsroom/2019/11/apple-expands-in-austin/.
[9] *See, e.g.* Rubinia Leal Cavazos ("Team Lead, Siri Team at Apple"), LinkedIn profile available online at https://www.linkedin.com/in/rubinialeal/; Nicole Mohrmann ("Team Lead (Siri) at Apple"), LinkedIn profile available online at https://www.linkedin.com/in/nicole-mohrmann-

30. Apple has also sought to recruit individuals for employment within this District through its website, LinkedIn, and other websites. This includes recent postings seeking to fill hundreds of full-time positions for SoC engineers and/or Artificial Intelligence and Machine Learning Annotation Analyst on the Siri Team.[10] Most of these positions require an "On-site" presence in this District.[11]

31. Moreover, as described below, many of the Accused Products implement Apple's infringing speech recognition technology on Intel chips. Intel is a designer and manufacturer of various processors and System on Chips. Intel is headquartered in Austin, Texas, where it employs close to 2,000 people.[12] In pertinent part, this means that on information and belief, components central to Apple's acts of infringement were designed and developed in this District, by individuals who reside and work here.

32. On information and belief, critical documents regarding the Accused Products are also located in this District given the presence of Intel and Apple employees that live in the District and work on Apple's infringing speech recognition technology.

966ab771/; Mortuza Lokhandwala ("SOC Design Verification Engineer at Apple), LinkedIn profile available online at https://www.linkedin.com/in/murtuza-lokhandwala-a8b53785/; Vu Le ("SoC Design Engineer at Apple"), LinkedIn profile available online at https://www.linkedin.com/in/vutle/; Neil Panchal ("SoC Physical Design Manager at Apple"), LinkedIn profile available online at https://www.linkedin.com/in/neil-panchal-194a783/.

[10] *See, e.g.* Apple LinkedIn Job postings, https://www.linkedin.com/jobs/search/?currentJobId=2848276684&geoId=102748797&keywords=siri%20apple&location=Texas%2C%20United%20States; *see also* Careers at Apple: Find your perfect role (search "Austin, TX" in Location field and "SoC Architect" and/or "Siri" in search field), https://jobs.apple.com/en-us/search?search=SoC%20Architect&sort=relevance&location=austin-AST.

[11] *Id.*

[12] *See* Innovating and Investing in Texas, https://www.intel.com/content/www/us/en/corporate-responsibility/intel-in-texas.html.

Amazon / Zentian Limited
Exhibit 1014
Page 8

ZENTIAN BREAKS NEW GROUND

33. Mark Catchpole, Zentian's founder and one of the named inventors on the Asserted Patents, holds a B.Sc. degree in Physics from the University of Edinburgh and a M.Sc. degree in Microelectronics Systems Design from Brunel University in London. After initial stints teaching physics, Catchpole joined Phoenix VLSI and later PixelFusion as an electronics engineer working on Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA) designs. ASICs and FPGAs are two forms of integrated circuits or chips that allow for customization of hardware after manufacturing.

34. Most of the work by Zentian was—and still is—unpaid, as funds proved difficult to raise during the early 2000s recession following the dot-com crash. Engineers worked in exchange for share options in the company, investing their own sweat equity into the potential of a hardware system-on-chip solution for speech recognition. During this time, Zentian was able to build a partial prototype on a programmable hardware emulation platform.

35. Guy Larri joined the Zentian team in 2004 as a technical contributor. Larri holds a B.Sc. degree in Mathematics and Computer Science, as well as an engineering degree in Electrical Engineering from the University of Sydney. Larri has worked as a VLSI (very large-scale integration) chip design engineer, designing audio processor chips, cache, memory controllers, and other microprocessors.

36. After Larri joined, Zentian's efforts shifted to the creation of a hardware accelerator for the acoustic stage of the speech recognition process that would work alongside a separate software implementation of the search stage (presumably running on a conventional mobile processor). Once the design specifications were created, the Zentian acoustic accelerator was loaded onto a programmable gate array (a type of integrated circuit designed to be customized

after manufacturing) and paired with the Sphinx open-source speech recognition software. The Zentian accelerator demonstrated a two-times speedup of the Sphinx software.

37. The implications of the Zentian team's discovery were vast and addressed a number of shortcomings of speech recognition systems in the art that had stymied the technology's commercial development. Speeding up the speech recognition process and reducing the overall power consumption would allow for the speech recognition system to run on a cheaper and/or lower-power processor and memory system. The Zentian accelerator would allow for the overall time the system might take to respond to spoken queries to be significantly reduced to a level more in-line with human conversations. Finally, Zentian's accelerator approach would allow for the use of larger and more sophisticated acoustic models without increasing response time, facilitating better recognition accuracy and a larger recognizable vocabulary.

38. Buoyed by their proof of concept, Catchpole and Larri attempted to market Zentian's acoustic accelerator hardware. They spoke with representatives of leading speech recognition companies, such as Nuance, IBM, Sensory, Inc., and Samsung, as well as cellular carriers, such as Hutchison 3G UK Limited. Catchpole and Larri also attended the 2006 GSM World Congress in Barcelona, where they discussed their novel speech recognition SoC designs with an executive from Qualcomm. Unfortunately, the industry at the time remained focused on software solutions for smaller problems, such as voice-controlled phone dialing.

39. In 2006, Zentian met with a director of Apple's Silicon Engineering Group, the group responsible for the development of multiple classes of chips used in Apple mobile products.[13] The presentation that Zentian shared with Apple identified problems with then-existent

[13] Apple describes its Silicon Technologies group as the team "responsible for the design and manufacture of our next generation of high-performance, power-efficient processor systems on

speech recognition systems, the promise of a dedicated hardware speech recognition accelerator, and the data demonstrating the success of Zentian's own prototype. The presentation also identified that Zentian had filed patents in the United States (and in the United Kingdom) to cover its novel hardware-based approach to speech recognition systems.

40. Apple listened to Zentian's pitch and then ceased communication with the start-up.

ZENTIAN'S PATENTS

41. As a result of its work in the speech recognition field, Zentian has a patent portfolio that covers a range of inventions that improve the performance and efficiency of speech recognition on-device systems, including in the areas of parallel processing and circuit configurations. Five of Zentian's patents, originating from two separate patent families, are at issue in this case.

42. U.S. Patent No. 7,979,277 (the "'277 patent") is entitled "Speech Recognition Circuit and Method" and issued on July 12, 2011. A true and correct copy of the '277 patent is attached as **Exhibit A** to this Complaint.

43. Under 35 U.S.C. § 154(b), the '277 patent is entitled to 1,043 days of Patent Term Adjustment, extending its expiry date until July 2028.

44. Zentian is the owner of all rights, title, and interest in and to the '277 patent, with the full and exclusive right to file suit and enforce the '277 patent, including the right to recover damages for past infringement.

45. The '277 patent is valid and enforceable under U.S. patent laws.

---

chip, as well as the analog IP and high-speed interface evaluation those chips need." *See* Careers at Apple: Hardware, Impossible is our favorite starting point, https://www.apple.com/careers/us/hardware.html.

46. The ’277 patent claims are directed to a patent-eligible, non-abstract idea. They address, among other things, a specific improvement to the way in which conventional software-based speech recognition systems operated on electronic devices. The ’277 patent claims include an architecture for a speech recognition circuit in which data is “pipelined” between different hardware componentry. The claims specify materials (including types of processors, accelerator interfaces, etc.), structures, and configurations of the claimed speech recognition circuit designed to enhance power consumption and efficiency of the system and allow for real-time processing of more complex and robust language models.

47. U.S. Patent No. 7,587,319 (the “’319 patent”) is entitled “Speech Recognition Circuit Using Parallel Processors” and issued on September 8, 2009. A true and correct copy of the ’319 patent is attached as **Exhibit B** to this Complaint.

48. Under 35 U.S.C. § 154(b), the ’319 patent is entitled to 1,137 days of Patent Term Adjustment, extending its expiry date until March 2026.

49. Zentian is the owner of all rights, title, and interest in and to the ’319 patent, with the full and exclusive right to file suit and enforce the ’319 patent, including the right to recover damages for past infringement.

50. The ’319 patent is valid and enforceable under U.S. patent laws.

51. The ’319 patent claims are directed to a patent-eligible, non-abstract idea. They address, among other things, a specific improvement to the way in which conventional software-based speech recognition systems operated on electronic devices. The ’319 patent claims include novel uses of a plurality of lexical tree processors connected in parallel to distribute the task of performing word recognition across processors.[14] This architecture provides for improved and

[14] A “lexical tree” is a data structure that comprises a model of words.

Amazon / Zentian Limited
Exhibit 1014
Page 12

faster processing, solving one of the problems of speech recognition known in the art—the computationally intensive search process that attempts to find the most likely set of words from a given lexicon. The claims specify materials (including types of processors, accelerator interfaces, etc.), structures, and configurations designed to enhance power consumption and efficiency of the system and allow for real-time processing of more complex and robust speech models.

52. U.S. Patent No. 10,971,140 (the "'140 patent") is entitled "Speech Recognition Circuit Using Parallel Processors" and issued on April 6, 2021. A true and correct copy of the '140 patent is attached as **Exhibit C** to this Complaint.

53. Under 35 U.S.C. § 154(b), the '140 patent is entitled to 0 days of Patent Term Adjustment, and its expiry date is in February 2023.

54. Zentian is the owner of all rights, title, and interest in and to the '140 patent, with the full and exclusive right to file suit and enforce the '140 patent, including the right to recover damages for past infringement.

55. The '140 patent is valid and enforceable under U.S. patent laws.

56. The '140 patent claims are directed to a patent-eligible, non-abstract idea. They address, among other things, a specific improvement to the way in which conventional software-based speech recognition systems operated on electronic devices. The '140 patent claims include a speech recognition circuit that has one or more clusters of processors, each of which comprises a plurality of processors and an acoustic model memory for storing acoustic model data. The disclosed and claimed circuit is further configured to generate an initial score for an audio sample, which is then used to determine whether to continue processing to determine a final score by processing a larger amount of model data than that which was processed to generate the initial score. The claims specify materials (including types of processors, accelerator interfaces, etc.),

structures, and configurations of the claimed speech recognition circuit designed to enhance power consumption and efficiency of the system and allow for real-time processing of more complex and robust language models.

57. U.S. Patent No. 10,062,377 (the "'377 patent") is entitled "Distributed Pipelined Parallel Speech Recognition System" and issued on August 28, 2018. A true and correct copy of the '377 patent is attached as **Exhibit D** to this Complaint.

58. Under 35 U.S.C. § 154(b), the '377 patent is entitled to 0 days of Patent Term Adjustment, and its expiry date is in September 2025.

59. Zentian is the owner of all rights, title, and interest in and to the '377 patent, with the full and exclusive right to file suit and enforce the '377 patent, including the right to recover for damages for past infringement.

60. The '377 patent is valid and enforceable under U.S. patent laws.

61. The '377 patent claims are directed to a patent-eligible, non-abstract idea. They address, among other things, a specific improvement to the way in which conventional software-based speech recognition systems operated on electronic devices. The '377 patent claims include, among other things, a speech recognition system comprising a device programmed to calculate a feature vector from a digital audio stream (wherein the feature vector comprises a plurality of extracted and/or derived quantities from the stream during a defined audio time frame); a second device programmed to calculate distances indicating the similarity between the calculated feature vector and a plurality of acoustic states of an acoustic model; and a third device programmed to identify spoken words in the digital audio stream using Hidden Markov Models and/or Neural Networks. The system also includes a search stage for using the calculated distances to identify words within a lexical tree (the lexical tree comprising a model of words). The claims specify

materials (including types of processors, accelerator interfaces, etc.), structures, and configurations of the claimed speech recognition system designed to reduce power consumption and improve efficiency of the system and to allow for real time processing of more complex and robust language models.

62. U.S. Patent No. 10,839,789 (the "'789 patent") is entitled "Speech Recognition Circuit and Method" and issued on November 17, 2020. A true and correct copy of the '789 patent is attached as **Exhibit E** to this Complaint.

63. Under 35 U.S.C. § 154(b), the '789 patent is entitled to 0 days of Patent Term Adjustment, and its expiry date is in September 2025.

64. Zentian is the owner of all rights, title, and interest in and to the '789 patent, with the full and exclusive right to file suit and enforce the '789 patent, including the right to recover for past infringement.

65. The '789 patent is valid and enforceable under U.S. patent laws.

66. The '789 patent claims are directed to a patent-eligible, non-abstract idea. They address, among other things, a specific improvement to the way in which conventional software-based speech recognition systems operated on electronic devices. The '789 patent claims include an acoustic coprocessor for processing data associated with an audio signal that includes an interface for receiving at least one feature vector, a calculating apparatus for calculating distances indicating a similarity between the feature vector and acoustic state of an acoustic model, and a second interface for sending at least one distance calculated by the calculating apparatus. The claims specify materials (including types of processors, accelerator interfaces, etc.), structures, and configurations of acoustic coprocessors designed to enhance power consumption and efficiency of the system and allow for real time processing of more complex and robust language models.

Amazon / Zentian Limited
Exhibit 1014
Page 15

67. For ease of reference, the ’277, ’377, and ’789 patents are part of the same patent family. The ’319 and ’140 patents are part of the same patent family.

APPLE’S UNAUTHORIZED USE OF ZENTIAN’S TECHNOLOGY

68. Apple’s infringing speech recognition solutions are central to shaping the user experience of many Apple products, including in Accused Products like Apple’s iPhone, iPad, Watch, HomePod smart speakers, Apple TV set top boxes, and Mac personal computers.

69. First introduced in 2011, Siri uses speech recognition and natural language processing to enable Apple devices to play music, control smart home accessories, answer general knowledge inquiries, send messages, set clocks and timers, obtain news and weather information, and even get traffic reports and translations. According to Apple, “Siri lets you stay connected without lifting a finger” and is a “faster, easier way to do all kinds of useful things.”[15]

70. Siri fulfills more than 25 billion requests each month, according to Apple’s director of Siri data science and engineering.[16] Siri is, in turn, a core feature built into almost every device Apple sells, and critical to the unique user-device interface that drives Apple’s market share.

71. Even though Siri was introduced in 2011, Apple continues to add features (*e.g.*, enabling the assistant hands free through the wake phrase “Hey Siri” in 2014 (also known as a “voice trigger”) and individualized voice recognition of “Hey Siri” in 2015) and mentions Siri in product demonstrations today. Most recently, during the privacy segment of Apple’s Worldwide

[15] *See* Siri does more than ever. Even before you ask, https://www.apple.com/siri/.
[16] *See* Ry Crist, *Siri’s iOS 14 Makeover Gives You Some of Your iPhone Screen Back*, CNET (June 23, 2020), https://www.cnet.com/tech/mobile/siri-voice-assistant-ios-14-makeover-gives-some-iphone-screen-back-apple-wwdc-2020/.

Amazon / Zentian Limited
Exhibit 1014
Page 16

Developers Conference, Apple revealed that Siri would process user requests "on device," *i.e.*, without using a network connection.[17]

72. On information and belief, on-device voice processing is important to selling Apple's Accused Products because it allays end user privacy concerns and allegedly speeds up performance and reliability.[18] By keeping the "always on" wake phrase function on-device, users don't have to worry about Apple "listening in" on their private conversations. To the extent Apple servers receive any of the user's words or content, it is not until the wake phrase is spoken and the user is directing their speech at the Accused Product.

73. Apple's infringement of Zentian's hardware speech recognition inventions includes among other things: (1) "Wake phrase detection" for Hey Siri; (2) Dictation of text messages, notes, and documents, with integrated applications, such as Apple's Pages and/or Microsoft Word; and (3) On-device speech recognition for Siri user requests, as announced by Apple as a privacy and performance improvement at their WorldWide Developers Conference. As described in detail below, Apple's speech recognition features would not be possible without the hardware-first architectures, methods, circuits, configurations, systems, chips and approaches engineered by Zentian in the early 2000s and claimed in the Asserted Patents.

74. Making matters worse, Apple's infringement is willful. In early 2015, Zentian once again met with Apple, this time with Apple's Patent Acquisitions department. At the meeting,

[17] *See* Alex Hern, *Apple Overhauls Siri to Address Privacy Concerns and Improve Performance*, The Guardian (June 7, 2021), https://www.theguardian.com/technology/2021/jun/07/apple-overhauls-siri-to-address-privacy-concerns-and-improve-performance.

[18] *See* Mitchell Clark, *Watch Apple's Siri Blaze Through Requests with On-device Processing*, The Verge (June 7, 2021), https://www.theverge.com/2021/6/7/22523438/apple-iphone-siri-on-device-processing-ios-15-wwdc-privacy (describing 2019 incident in which Apple contractors were able to listen to Siri requests on Apple servers, requiring Apple to increase transparency by making Siri recordings opt-in).

Zentian and Apple representatives discussed the two patent families at issue, including several of the then-issued Asserted Patents, such as the '277 patent and the '319 patent, as well as then-pending patent applications that kept the two patent families open for future issuances.

75. As a result of this (and the 2006) meeting, Apple was well aware of Zentian's patented speech recognition circuits and methods when it decided to make, use, offer to sell, and/or sell the infringing Accused Products within the United States, making such acts deliberate and intentional at a minimum, and potentially willful, wanton, in bad-faith and/or consciously wrongful.

COUNT ONE: INFRINGEMENT OF U.S. PATENT NO. 7,979,277

76. Zentian incorporates by reference and realleges the allegations in paragraphs 1-75.

77. The '277 patent is valid and enforceable.

78. The '277 patent names Guy Larri, Mark Catchpole, Damian Kelly Harris-Dowsett, and Timothy Brian Reynolds as inventors.

79. Apple has directly infringed, continues to infringe, and has induced or contributed to the infringement of the '277 patent by making, using, selling, offering for sale, and/or importing into the United States, without authority or license, the Accused Products in violation of 35 U.SC. § 271(a).

80. The Accused Products are non-limiting examples that were identified based on publicly available information. Zentian reserves the right to identify additional infringing activities, products, and services, including based on information obtained during discovery.

81. By way of example only, the Accused Products meet all the limitations of at least independent claims 14, 15, and 16 of the '277 patent, either literally or under the doctrine of equivalents. Claim 14 is directed to a speech recognition circuit, claim 15 to a speech recognition

Amazon / Zentian Limited
Exhibit 1014
Page 18

method, and claim 16 to a non-transitory storage medium storing processor implementable code for controlling at least one processor to implement a speech recognition method.

82. Claim 14 of the '277 patent recites:

> A speech recognition circuit, comprising:
>
> [1] an audio front end for calculating a feature vector from an audio signal, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame;
>
> [2] calculating means for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model; and
>
> [3] a search stage for using said calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words;
>
> [4] wherein said audio front end, said calculating means, and said search stage are connected to each other to enable pipelined data flow.

83. Regarding the preamble of claim 14, to the extent the preamble is determined to be limiting, the Accused Products contain a speech recognition circuit that allows for device users to speak the words "Hey Siri" and activate Apple's personal assistant hands-free. The speech recognition circuit of the Accused Products further allows for dictation of text messages, notes, and documents with integrated applications, as well as on device speech recognition for Siri user requests, as announced by Apple recently at their Worldwide Developers Conference.[19]

84. On information and belief, the speech recognition circuit of the Accused Products works with cloud-based programming, natural language interpretation and other services with

[19] *See* Alex Hern, *Apple Overhauls Siri to Address Privacy Concerns and Improve Performance*, The Guardian (June 7, 2021), https://www.theguardian.com/technology/2021/jun/07/apple-overhauls-siri-to-address-privacy-concerns-and-improve-performance.

Amazon / Zentian Limited
Exhibit 1014
Page 19

hardware-assisted on-device processing, creating a combination of hardware and software features.[20]

85. Apple published further evidence of the speech recognition circuit of the Accused Products in October 2017. The Apple Siri Team posted an article on Apple's website entitled "Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant."[21] That article states that a "specialized speech recognizer" incorporated on the local devices themselves "runs all the time and listens" for the words "Hey Siri," which allows for the feature to operate hands-free and respond even when the Accused Product does not have a data connection.[22]

86. As to the first limitation of claim 14 of the '277 patent, the Accused Products calculate a feature vector from an audio signal using an audio front end, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame.

87. In the technical paper published by Apple's Siri Team, Apple describes the Accused Products' calculation of a feature vector from an audio signal in the following way:

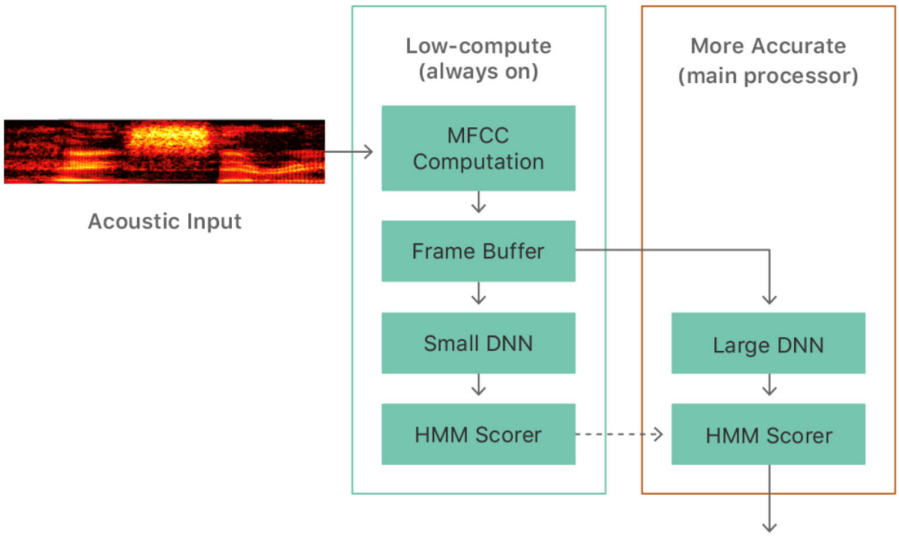
> The microphone in an iPhone or Apple watch turns your voice into a stream of instantaneous waveform samples, at a rate of 16000 per second. A spectrum analysis stage converts the waveform sample stream to a sequence of frames, each describing the sound spectrum of approximately 0.01 sec. About twenty of these frames at a time (0.2 sec of audio) are fed to the acoustic model, a Deep Neural Network (DNN) which converts each of these acoustic patterns into a probability distribution over a set of speech sound classes: those

[20] *See, e.g.*, Christian Ziberg, *New Machine Learning Article from Apple Goes In Depth on How "Hey Siri" Does Its Magic*, iDB (Oct. 18, 2017), https://www.idownload-blog.com/2017/10/18/apple-hey-siri-machine-learning-article/.
[21] Siri Team, *Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant* (Oct. 2017), https://machinelearning.apple.com/research/hey-siri ("*Hey Siri*"). DNN means "Deep Neural Network."
[22] *Id.*

Amazon / Zentian Limited
Exhibit 1014
Page 20

> used in the "Hey Siri" phrase, plus silence and other speech, for a total of about 20 sound classes.[23]

88. As shown below in a figure from the Siri Team's paper, the "Always On Processor (AOP)" of the Accused Products is "a small, low-power auxiliary processor" that has access to "the microphone signal."[24] The AOP calculates feature vectors through an extraction method known as MFCC (or Mel Frequency Cepstrum Coefficient) calculations. MFCCs are a collection of coefficients that represent the short-term power spectrum of a sound.[25]

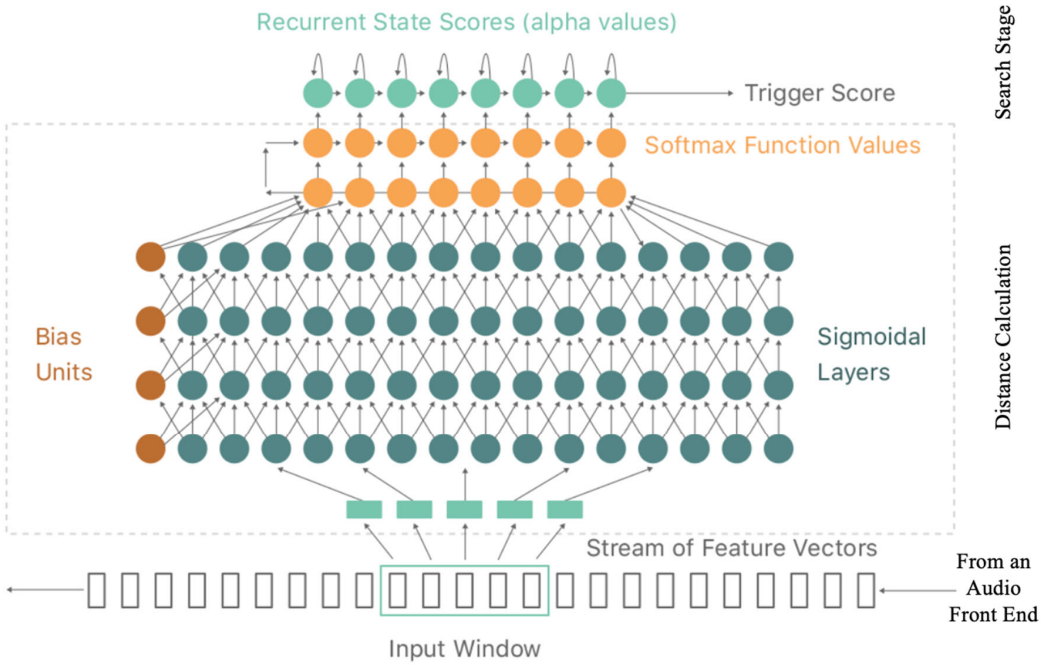


89. The below speech recognizer diagram (annotation added in dark text to the right), also published in the Siri Team's technical paper, illustrates generally how the speech recognizer converts a spoken utterance of the words "Hey Siri" into a function performed by the Accused Product, such as waking the device. Relevant to the first limitation, the "input window" (green box

[23] *Id.*
[24] *Id.*
[25] *See, e.g.*, K.S. Rao and Manjunath K.E., *Speech Recognition Using Articulatory and Excitation Source Features*, Appendix A: MFCC Features at 85, SpringerBriefs in Speech Technology, https://link.springer.com/content/pdf/bbm%3A978-3-319-49220-9%2F1.pdf.

Amazon / Zentian Limited
Exhibit 1014
Page 21

at bottom) receives a "stream of feature vectors" calculated by an audio front end (the MFCC Computation on the AOP).[26]



90. The Accused Products similarly contain the second limitation of claim 14—a calculating means for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model.

91. The acoustic model contained in the Accused Products is a Deep Neural Network (DNN). The DNN converts the "stream of feature vectors" it receives into probability distributions consisting of extracted and derived quantities.[27]

92. In the Siri Team's technical paper, Apple states that the "output of the acoustic model provides a distribution of scores over phonetic classes for every frame. A phonetic class is

[26] *Hey Siri* at Fig. 2 (annotations added).
[27] *Id.*

Amazon / Zentian Limited
Exhibit 1014
Page 22

typically something like 'the first part of an /s/ preceded by a high front vowel and followed by a front vowel.'"[28] Phonetic classes of this type are otherwise known in the art as a triphones.

93. In the same technical paper, Apple describes the DNN acoustic model (shown in the speech recognizer diagram above bounded by a dashed grey box) as follows:

> DNN consists mostly of matrix multiplications and logistic nonlinearities. Each "hidden" layer [e.g. circles connected with arrows identified as sigmoidal layers, bias units, and/or softmax function values] is an intermediate representation discovered by the DNN during its training to convert the filter bank inputs to sound classes. The final nonlinearity is essentially a Softmax function (a.k.a. a general logistic or normalized exponential).[29]

94. The underlying mathematical equation (below) for this "distance" calculation that is the output of the acoustic model is represented by the variable $q_{i,t}$—"the log score for the phonetic class associated with the *ith* state given the acoustic pattern around time *t*"—a further indication that the acoustic model contains predetermined acoustic states.[30]

$$F_{i,t} = \max\{s_i + F_{i,t-1}, m_{i-1} + F_{i-1,t-1}\} + q_{i,t}$$

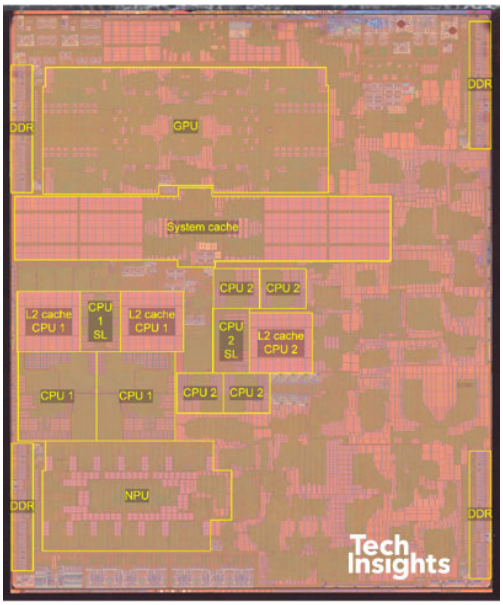
where

- $F_{i,t}$ is the accumulated score for state *i* of the model
- $q_{i,t}$ is the output of the acoustic model—the log score for the phonetic class associated with the *ith* state given the acoustic pattern around time *t*
- s_i is a cost associated with staying in state *i*
- m_i is a cost for moving on from state *i*

[28] *Id.*
[29] *Id.*
[30] *Id.*

Amazon / Zentian Limited
Exhibit 1014
Page 23

95. On information and belief, due to its power and computational requirements, the calculating means for the "Large DNN" (as compared to the "Small DNN" on the AOP) in the Accused Products is a separate processor, such as a Neural Engine with 16-core design processor in the Apple Silicon M1 Mac,[31] or alternatively, for example, in iPhone 12 and iPad Air 4th generation models that use the A14 Bionic SoC, the Graphics Processing Unit (GPU) depicted at the top of the A14 Bionic chip die image to the right.[32]



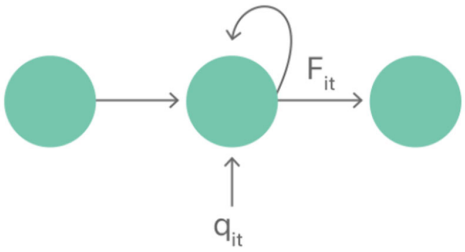
96. The Accused Products contain the third limitation of claim 14—a search stage to identify words within a lexical tree using the calculated distances, with the lexical tree comprising a model of words.

[31] *See* Press Release, *Introducing M1 Pro and M1 Max: The Most Powerful Chips Apple Has Ever Built* (Oct. 18, 2021), https://www.apple.com/newsroom/2021/10/introducing-m1-pro-and-m1-max-the-most-powerful-chips-apple-has-ever-built/.

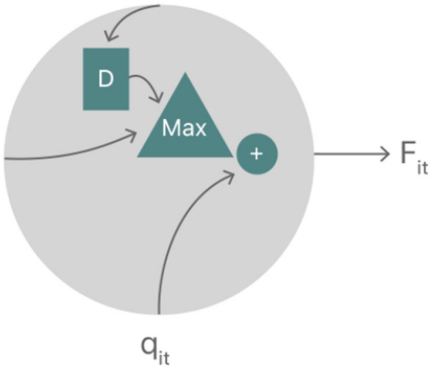
[32] *See, e.g.*, Ben Morris, *Imagination Technologies: A British Technology Success Story*, BBC (May 16, 2012), https://www.bbc.co.uk/news/business-17939014 (quoting Apple GPU provider for certain iPhone models that "Apple's popular voice command system, Siri, is driven by the GPU").

Amazon / Zentian Limited
Exhibit 1014
Page 24

97. In the example of the "Hey Siri" system, the speech recognizer contains a "language-specific phonetic specification of the 'Hey Siri' phrase" that was created with two English variants: one with Siri as in "serious" and two with Siri as in "Syria."[33] According to the Siri Team's technical paper on Apple's website, this model of words was built to "cope with a short break between the two words, especially as the phrase is often written with a comma: 'Hey, Siri.' Each phonetic symbol results in three speech sound classes (beginning, middle, and end) each of which is its own output from the acoustic model."[34]



98. In the speech recognizer diagram, the search stage is depicted by the final top-layer of light green circles outside of the dotted-line box that surrounds the computationally intensive acoustic model. Combining that image with the mathematical equation above, visually shows that the output of the acoustic model, q_{it}, is fed into the search stage, which Apple describes as an "application of dynamic programming, and can be derived based on ideas about Hidden Markov Models—HMMs."[35]



99. Further support for the existence of a search stage to identify words exists in Apple's instructions to application developers looking to incorporate the Siri technology. In the SiriKit for developers, Apple describes a "vocabulary step," consisting of a vocabulary database

[33] *Hey Siri*.
[34] *Id*.
[35] *Id*.

Amazon / Zentian Limited
Exhibit 1014
Page 25

that can be adapted and added to, and that comes into play between when Siri recognizes speech from the user and creating an action that Siri can execute.[36]

100. Further, the Siri Team's technical paper states that "[a]lmost all the computation in the 'Hey Siri' detector is in the acoustic model [*i.e.* Large DNN]. The temporal integration computation is relatively cheap, so we disregard it when assessing size or computational resources."[37]

101. On information and belief, the temporal integration computation that occurs in the search stage occurs on either the AOP or another processor, such as one of the main application processors of the Apple system on chip (SoC). For example, there are six application processor CPU cores on the Apple A14 Bionic chip, as shown on the Tech Insights die photo (two of which are labelled "CPU1" and four of which are labelled "CPU2."[38]

102. On information and belief, the high load of computing the acoustic model outputs explain why those computations are performed by the GPU or Neural Engine (depending on which SoC is in the product), where such calculations can be evaluated more efficiently and quickly, and in parallel with other workloads on the AOP and Application processors.

103. As to the fourth limitation of claim 14 of the '277 patent, the Accused Products are enabled to pipeline data from an audio front end to a calculating means and then to the search stage.

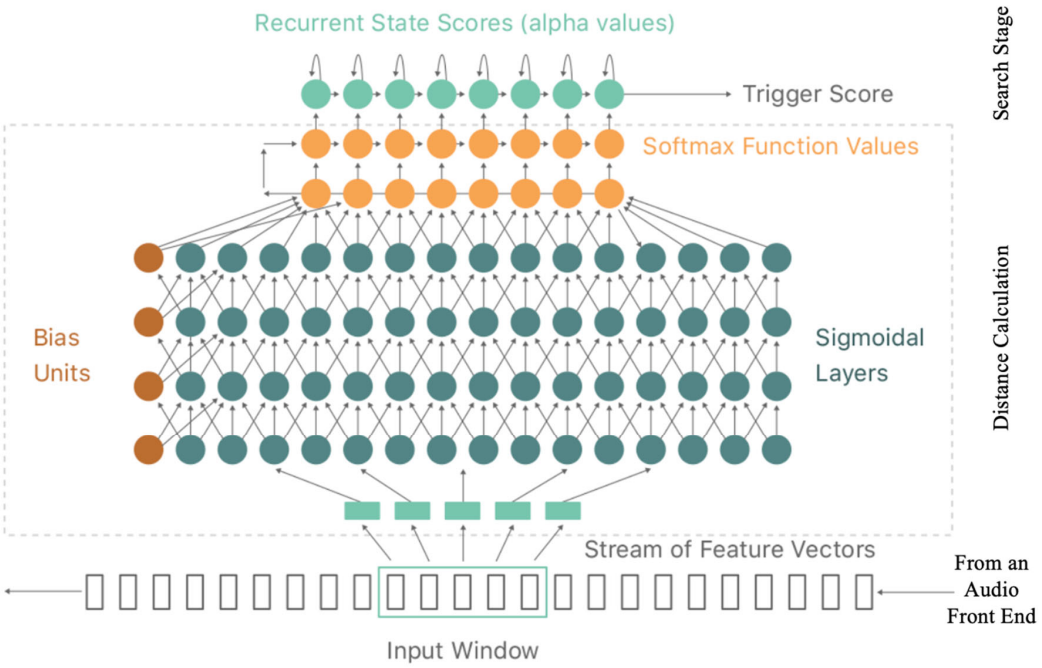
104. The diagram of Apple's speech recognition system (annotated and reproduced again below) describes the flow of data in Apple's speech recognizer. On information and belief,

---

[36] *See* YML, *Learn How to Integrate Siri with Your App Thanks to Apple SiriKit* (June 29, 2016), https://ymedialabs.medium.com/learn-how-to-integrate-siri-with-your-app-thanks-to-apple-siri-kit-c6e570d302cb#1.
[37] *Hey Siri*.
[38] *Id.*

Amazon / Zentian Limited
Exhibit 1014
Page 26

data flows in a pipelined manner from initial feature vector extraction on Apple's AOP (MFCC calculation on the AOP supplying "stream of feature vectors" to the green box labeled "Input Window" showing transmission of feature vectors), to a calculating means that computes the outputs of an acoustic model (circles and arrows within grey dotted-line box) occurring on a second processor, likely a Neural Engine or GPU, to the search stage (top line culminating in "Trigger Score") that identifies corresponding words or phrases from a language model (*i.e.* a lexical tree) that runs on either the AOP or a processor, such as the main application processors on the Apple A14 Bionic chip.[39]



105. The above allegations detail how the Accused Products' operation and implementation of the "Hey Siri" voice trigger meets the limitations of claim 14 of the '277 patent.

[39] *Id.*at Fig. 2 (annotations added).

Amazon / Zentian Limited
Exhibit 1014
Page 27

On information and belief, other speech recognition features of the Accused Products use similar technologies and implementations, are therefore similarly meet the claim limitations.

106. Apple actively, knowingly, and intentionally induces infringement of one or more claims of the ’277 patent under 35 U.S.C. § 271(b) by actively encouraging others to make, use, sell, and offer to sell in the United States, the Accused Products.

107. For example, Apple provides directions, instruction manuals, guides, and/or other materials that encourage and facilitate infringing use by others.

108. Apple has sold and is selling the Accused Products with the knowledge and intent that customers who buy the products will use the products for their infringing use and therefore that customers have been and are directly infringing the ’277 patent.

109. Apple contributes to the infringement of one or more claims of the ’277 patent under 35 U.S.C § 271(c) by offering to sell, selling, or importing into the United States, a component of the Accused Products that constitutes a material part of the inventions, knowing the same to be especially made or especially adapted for use in an infringement of the ’277 patent, and is not a staple article or commodity of commerce suitable for substantial non-infringing use.

110. Apple was aware of the ’277 patent at least from the date of this Complaint.

111. On information and belief, Apple has been aware of the ’277 patent since its issuance on July 12, 2011, and/or since Zentian representatives met with Apple’s patent acquisition department in 2015 and discussed the ’277 patent, among other Asserted Patents.

112. On information and belief, Apple knew that the Accused Products infringe the ’277 patent, or at a minimum believed there was a high probability that the Accused Products were covered by Zentian’s patents, but willfully blinded itself to Zentian’s patents and the infringing nature of the Accused Products.

Amazon / Zentian Limited
Exhibit 1014
Page 28

113. The foregoing allegations are based on publicly available information and a reasonable investigation of the structure and operation of the Accused Products.

114. Zentian reserves the right to modify this description of alleged infringement, including, for example, on the basis of information about the Accused Products that it obtains during discovery.

115. The infringement details provided for claim 1 of the '277 patent are exemplary, and Zentian intends to assert additional claims of the '277 patent beyond claim 14.

116. Apple's infringement has damaged and continues to damage Zentian in an amount to be determined at trial, but in no case less than a reasonable royalty for Apple's unauthorized use of the claimed inventions set forth in the '277 patent.

117. This is an exceptional case. Zentian is entitled to attorneys' fees and costs under 35 U.S.C § 285 as a result of the infringement of the '277 patent by Apple.

COUNT TWO: INFRINGEMENT OF U.S. PATENT NO. 7,587,319

118. Zentian incorporates by reference and realleges the allegations in paragraphs 1-117.

119. The '319 patent is valid and enforceable.

120. The '319 patent names Mark Catchpole as its sole inventor.

121. Apple has directly infringed, continues to infringe, and has induced or contributed to the infringement of the '319 patent by making, using, selling, offering for sale, and/or importing into the United States, without authority or license the Accused Products in violation of 35 U.SC. § 271(a).

122. The Accused Products are non-limiting examples identified based on publicly available information. Zentian reserves the right to identify additional infringing activities, products, and services, including, for example, on the basis of information obtained during discovery.

123. By way of example only, the Accused Products meet all the limitations of at least claim 46 of the '319 patent, either literally or under the doctrine of equivalents, which recites:

A speech recognition circuit, comprising:

[1] an input buffer receiving processed speech parameters:

[2] a plurality of lexical memories containing in combination complete lexical data for word recognition, each lexical memory containing part of said complete lexical data;

[3] a plurality of processors connected in parallel to said input buffer for processing the speech parameters in parallel, said processors being arranged in groups of processors, each group of processors being connected to a lexical memory:

[4] a control processor controlling each processor to process said speech parameters using partial lexical data read from a respective said lexical memory; and

[5] a results memory storing the results of the processing of the speech parameters from said processors.

124. Regarding the preamble of claim 46, to the extent the preamble is determined to be limiting, the Accused Products contain a speech recognition circuit that allows for device users to speak the words "Hey Siri" and activate Apple's personal assistant hands-free. The speech recognition circuit of the Accused Products further allows for dictation of text messages and notes with integrated applications, as well as on device speech recognition for Siri user requests.[40]

125. On information and belief, the speech recognition circuit of the Accused Products works with cloud-based programming, natural language interpretation and other services with hardware-assisted on-device processing, creating a combination of hardware and software features.[41]

[40] *See* Alex Hern, *Apple Overhauls Siri to Address Privacy Concerns and Improve Performance*, The Guardian (June 7, 2021), https://www.theguardian.com/technology/2021/jun/07/apple-overhauls-siri-to-address-privacy-concerns-and-improve-performance.

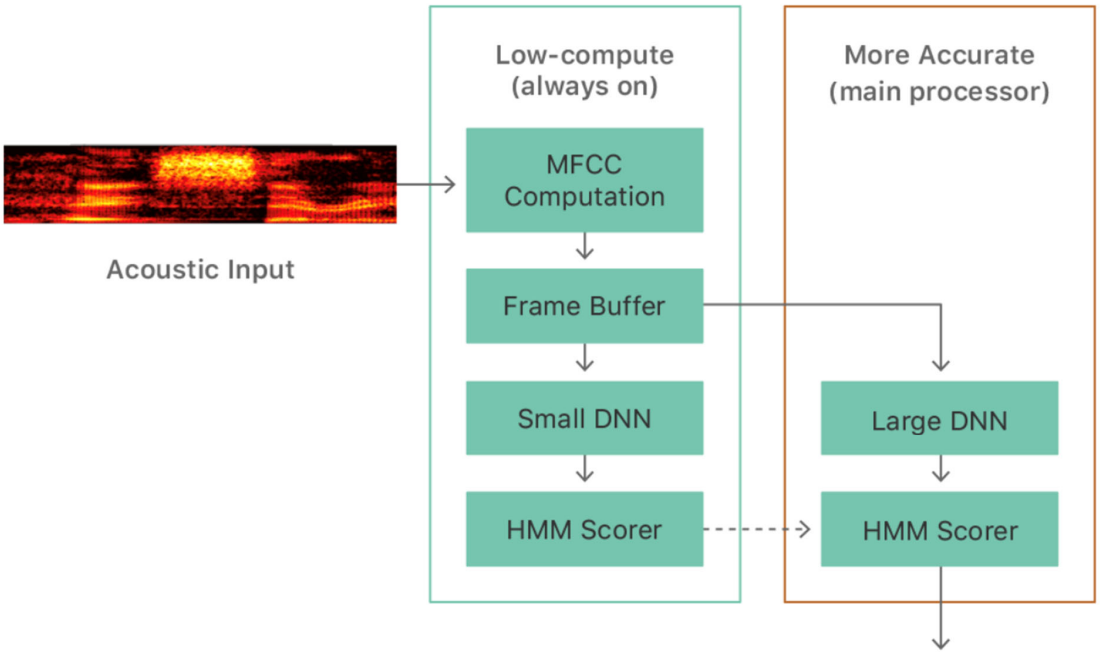
[41] *See, e.g.*, Christian Ziberg, *New Machine Learning Article from Apple Goes In Depth on How "Hey Siri" Does Its Magic*, iDB (Oct. 18, 2017), https://www.idownloadblog.com/2017/10/18/apple-hey-siri-machine-learning-article/.

126. In October 2017, Apple's Siri Team published an article on Apple's website, entitled "Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant."[42] That article states that a "specialized speech recognizer" incorporated on the local devices themselves "runs all the time and listens" for the words "Hey Siri," which allows for the feature to operate hands-free.[43] This feature also allows the Accused Product to respond to "Hey Siri" requests when a data connection is not present.

127. As to the first limitation of claim 46 of the '319 patent, the Accused products contain an input buffer receiving processed speech parameters. Notably, Apple's technical paper describes a "Frame Buffer" in the AOP that stores processed speech parameters (feature vectors). The AOP receives audio input from an acoustic input, such as the microphone[44], and processes the voice audio by performing MFCC computations to produce the feature vectors that are stored in the Frame Buffer. The contents of the Frame Buffer can be received as an input to the so called "main processor," meaning that the Frame Buffer acts as an input buffer for the main processor. This architecture is shown in the figure below.[45]

[42] *Hey Siri*.
[43] *Id.*
[44] *Id.* ("The microphone in an iPhone or Apple Watch turns your voice into a stream of instantaneous waveform samples, at a rate of 16000 per second. A spectrum analysis stage converts the waveform sample stream to a sequence of frames, each describing the sound spectrum of approximately 0.01 sec. About twenty of these frames at a time (0.2 sec of audio) are fed to the acoustic model, a Deep Neural Network (DNN) which converts each of these acoustic patterns into a probability distribution over a set of speech sound classes.")
[45] *Id.*

Amazon / Zentian Limited
Exhibit 1014
Page 31



128. As to the second limitation of claim 46 of the '319 patent, the Accused Products contain a plurality of lexical memories containing (in combination) complete lexical data for word recognition. Notably, each Accused Product contains FLASH memory that acts as a non-volatile memory to store data even when the device is turned off or the battery is depleted.[46] For example, the iPhone 13 Pro A2636 256GB Model contains Kioxia 256 GB NAND Flash.[47]

129. In addition to the FLASH memory, each Accused Product contains a cache memory hierarchy that holds copies of data from the FLASH memory when it is accessed by processing elements, such as the main or application processor referenced in the above figure. On information and belief, each application processor has a private level 1 data cache and shared caches lower in

[46] *See, e.g.,* iPhone 13 specifications, https://www.apple.com/iphone-13/specs/ (describing FLASH memory as "Capacity" or "Storage Capacity").
[47] *See* TechInsights, *Apple iPhone 13 Pro Teardown Briefing*, https://www.techinsights.com/blog/teardown/apple-iphone-13-pro-teardown.

Amazon / Zentian Limited
Exhibit 1014
Page 32

the cache hierarchy, known as level 2 caches, level 3 caches, or "system caches." The FLASH memory and the caches in the cache hierarchy that lead to the Application Processors are, separately or in combination, the lexical memories containing complete lexical data for word recognition, with each containing some part of that data at any given time.

130. On information and belief, the FLASH and cache memories store Apple's speech recognition models, which use lexical data. According to Apple, Siri's "ASR system uses a weighted finite state transducer (WFST)-based decoder, as described by Paulik."[48] Paulik is a reference to a paper published in 2015 by Apple employee Matthias Paulik, who was a senior speech research scientist at the company at the time.[49] Paulik's paper states that "[w]e (presumably referring to Apple) use a WFST based decoder that employs the difference LM principle, similar to [17]. … The decoding dictionary has 523.6K entries and the entropy pruned 4-gram language model has 16 million entries."[50] On information and belief, Apple's decoding dictionaries, with their numerous entries, contain lexical data and information.

131. As to the third limitation of claim 46 of the '319 patent, the Accused Products include more than one processor for processing speech parameters arranged in groups (with each group connected to a lexical memory). For example, the main or application processor in the Accused Products is itself arranged in groups, often referred to as "clusters." These clusters contain

[48] *See* Siri Speech Recognition Team, *Finding Local Destinations with Siri's Regionally Specific Language Models for Speech Recognition* (Aug. 2018), https://machinelearning.apple.com/research/regionally-specific-language-models.

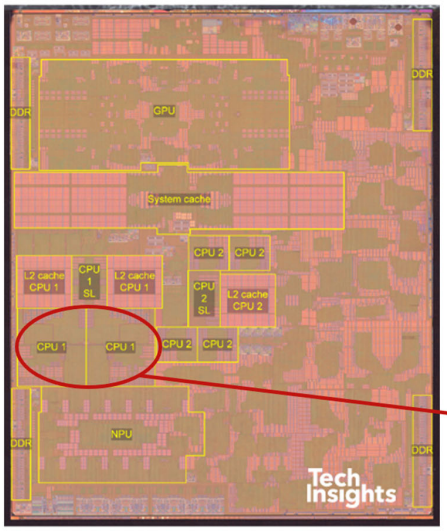
[49] *See* M. Paulik, *Improvements to the Pruning Behavior of DNN Acoustic Models*, Interspeech (2015), https://www.isca-speech.org/archive_v0/interspeech_2015/papers/i15_1463.pdf .

[50] *Id.* (identifying reference [17] as Dolfing H. and Hetherington, I., *Incremental Language Models for Speech Recognition using Finite-State Transducers*, ASRU, (December 2001) Madonna di Campiglio, Trento, Italy).

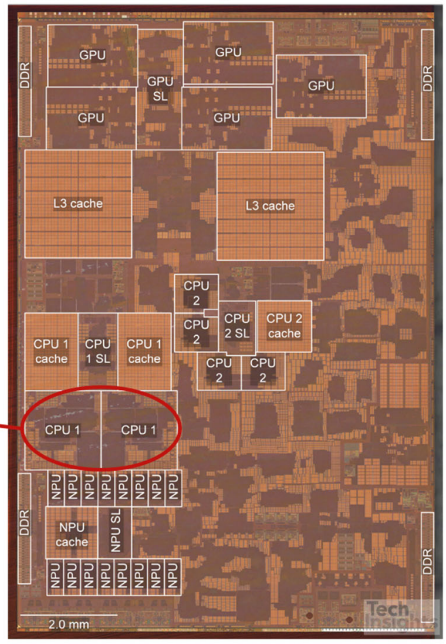
Amazon / Zentian Limited
Exhibit 1014
Page 33

multiple processors. For example, in the Apple A14 and A15 SoC as seen in the die photos below, Cluster 1 contains two application processors labelled CPU1.[51]

Apple A14 SoC die photo and floorplan



Apple A15 SoC die photo and floorplan



132. Moreover, as further shown above, each of the processors within a cluster is connected to cache memory, including a private cache, a level 2 cluster level shared cache, and a level 3 system cache ("L3 cache" in A15 SoC). On information and belief, these caches along with the Accused Products' FLASH memory are sources of lexical memory because they store some part (or all) of the lexical data for word recognition.

133. On information and belief, these processors and clusters are connected in parallel to the input buffer. As described and shown above, during the speech recognition process, the

[51] *See* Tech Insights, *Two New Apple SoCs, Two Market Events: Apple A14 and M1*, https://www.techinsights.com/blog/two-new-apple-socs-two-market-events-apple-a14-and-m1 (A14 SoC die photo); *Apple iPhone 13 Pro Teardown Briefing*, https://www.techinsights.com/blog/teardown/apple-iphone-13-pro-teardown (A15 SoC die photo).

contents of the "Frame Buffer" in the AOP can be received as an input to the so called "main processor" (likely one or more of the clusters of processors described above, such as CPU1 and/or CPU2), meaning that the Frame Buffer acts as an input buffer for the main processor.

134. The Accused Products also contain the fourth limitation of claim 46 of the '319 patent, which is a control processor that controls each processor that is processing speech parameters using lexical data read from stored lexical memory. In a technical paper published in 2017, Apple's Siri Team described one function of Apple's speech recognition circuit—the "Hey Siri" wake phrase detection in the following manner:

> To avoid running the main processor all day just to listen for the trigger phrase, the iPhone's Always On Processor (AOP) (a small, low-power auxiliary processor, that is the embedded Motion Coprocessor) has access to the microphone signal (on 6S and later). We use a small proportion of the AOP's limited processing power to run a detector with a small version of the acoustic model (DNN). When the score exceeds a threshold the motion coprocessor wakes up the main processor, which analyzes the signal using a larger DNN.[52]

135. On information and belief, the AOP controls the main or application processor, as described in the paragraph above. As further described, the main or application processor processes speech parameters using lexical data—the data within the large DNN. And that lexical data is stored in the FLASH and/or cache memory structures connected to and shared by those processors.

136. Finally, the fifth limitation of claim 46 of the '319 is also present in the Accused Products, as each has a results memory storing the results of processed speech parameters. Notably, each of the Accused Products contain SDRAM (Synchronous Dynamic Random Access Memory).[53]

---

[52] *Hey Siri*.

[53] *See, e.g.,* TechInsights, *Apple iPhone 13 Pro Teardown Briefing*, https://www.techinsights.com/blog/teardown/apple-iphone-13-pro-teardown.

137. On information and belief, SDRAM is commonly the main working memory of the computation system in electronic devices like the Accused Products and would store the results of the processed speech parameters from the AOP and main processors.

138. Apple actively, knowingly, and intentionally induces infringement of one or more claims of the ’319 patent under 35 U.S.C. § 271(b) by actively encouraging others to make, use, sell, and offer to sell in the United States, the Accused Products.

139. For example, Apple provides directions, instruction manuals, guides, and/or other materials that encourage and facilitate infringing use by others.

140. Apple has sold and is selling the Accused Products with the knowledge and intent that customers who buy the products will use the products for their infringing use and therefore that customers have been and are directly infringing the ’319 patent.

141. Apple contributes to the infringement of one or more claims of the ’319 patent under 35 U.S.C § 271(c) by offering to sell, selling, or importing into the United States, a component of the Accused Products that constitutes a material part of the inventions, knowing the same to be especially made or especially adapted for use in an infringement of the ’319 patent, and is not a staple article or commodity of commerce suitable for substantial noninfringing use.

142. Apple was aware of the ’319 patent at least from the date of this Complaint.

143. On information and belief, Apple has been aware of the ’319 patent since its issuance on September 8, 2009, and/or since Zentian representatives met with Apple’s patent acquisition department in 2015 and discussed the ’319 patent, among other Asserted Patents.

144. On information and belief, Apple knew that the Accused Products infringe the ’319 patent, or at a minimum believed there was a high probability that the Accused Products were

Amazon / Zentian Limited
Exhibit 1014
Page 36

covered by Zentian's patents, but willfully blinded itself to Zentian's patents and the infringing nature of the Accused Products.

145. The foregoing allegations are based on publicly available information and a reasonable investigation of the structure and operation of the Accused Products.

146. Zentian reserves the right to modify this description of alleged infringement, including, for example, on the basis of information about the Accused Products that it obtains during discovery.

147. The infringement details provided for claim 1 of the '319 patent are exemplary, and Zentian intends to assert additional claims of the '319 patent beyond claim 46.

148. Apple's infringement has damaged and continues to damage Zentian in an amount to be determined at trial, but in no case less than a reasonable royalty for Apple's unauthorized use of the claimed inventions set forth in the '319 patent.

149. This is an exceptional case. Zentian is entitled to attorneys' fees and costs under 35 U.S.C § 285 as a result of the infringement of the '319 patent by Apple.

COUNT THREE: INFRINGEMENT OF U.S. PATENT NO. 10,971,140

150. Zentian incorporates by reference and realleges the allegations in paragraphs 1-149.

151. The '140 patent is valid and enforceable.

152. The '140 patent names Mark Catchpole as its sole inventor.

153. Apple has directly infringed, continues to infringe, and has induced or contributed to the infringement of the '140 patent by making, using, selling, offering for sale, and/or importing into the United States, without authority or license the Accused Products in violation of 35 U.SC. § 271(a).

154. The Accused Products are non-limiting examples identified based on publicly available information. Zentian reserves the right to identify additional infringing activities,

products, and services, including, for example, on the basis of information obtained during discovery.

155. By way of example only, the Accused Products meet all the limitations of at least claim 1 of the '140 patent, either literally or under the doctrine of equivalents, which recites:

> A speech recognition circuit, comprising:
>
> [1] one or more cluster of processors, each of the one or more cluster of processors comprising:
>
> [2] a plurality of processors; and
>
> [3] an acoustic model memory storing acoustic model data, wherein each of the plurality of processors is configured to compute a probability using the acoustic model data in the acoustic model memory, wherein:
>
> [4] the speech recognition circuit is configured to generate an initial score for an audio sample; and
>
> [5] the initial score is used to determine whether to continue processing to determine a final score via processing a larger amount of model data than that was processed to generate the initial score.

156. Regarding the preamble of claim 1, to the extent the preamble is determined to be limiting, the Accused Products contain a speech recognition circuit that allows for device users to speak the words "Hey Siri" and activate Apple's personal assistant hands-free. The speech recognition circuit of the Accused Products further allows for dictation of text messages and notes with integrated applications, as well as on device speech recognition for Siri user requests.[54]

157. On information and belief, the speech recognition circuit of the Accused Products works with cloud-based programming, natural language interpretation and other services with

[54] *See* Alex Hern, *Apple Overhauls Siri to Address Privacy Concerns and Improve Performance*, The Guardian (June 7, 2021), https://www.theguardian.com/technology/2021/jun/07/apple-overhauls-siri-to-address-privacy-concerns-and-improve-performance.

Amazon / Zentian Limited
Exhibit 1014
Page 38

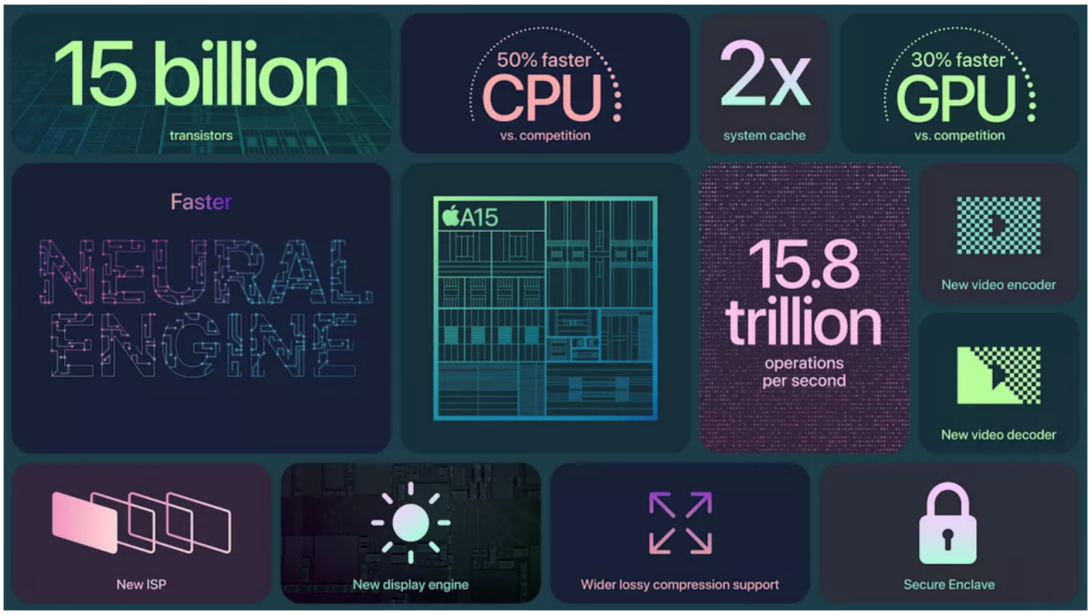
hardware-assisted on device processing, creating a combination of hardware and software features.[55]

158. In October 2017, Apple's Siri Team published an article on Apple's website, entitled "Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant."[56] That article states that a "specialized speech recognizer" incorporated on the local devices themselves "runs all the time and listens" for the words "Hey Siri," which allows for the feature to operate hands-free and respond even when the Accused Product does not have a data connection.[57]

159. As to the first limitation of claim 1 of the '140 patent, the Accused Products contain one or more clusters of processors. For example, the iPhone 13 and sixth generation iPad Mini contain an Apple A15 Bionic System on Chip (SoC). The A15 chip contains two "high performance" cores called Avalanche and four "energy efficient" cores called Blizzard. In addition, the A15 chip contains both Apple's Neural Engine and Apples Graphics Processor Unit (GPU).[58] Each of these engines and/or processors contain within them multiple "cores" and/or processors, such that they can be described, either individually or collectively, as a "clusters of processors."[59]

[55] *See, e.g.*, Christian Ziberg, *New Machine Learning Article from Apple Goes In Depth on How "Hey Siri" Does Its Magic*, iDB (Oct. 18, 2017), https://www.idownloadblog.com/2017/10/18/apple-hey-siri-machine-learning-article/.
[56] *Hey Siri*.
[57] *Id.*
[58] *The New Apple A15 Bionic Chip*, Mac O'Clock (Oct. 12, 2021), https://medium.com/macoclock/the-new-apple-a15-bionic-chip-dc6859d2c90e.
[59] *Id*.

160. These clusters of processors that comprise the claimed speech recognition circuit also were touted by Apple as key features of the A15 chip as seen in the specification diagram below highlighting the 30% faster GPU, faster Neural Engine, and faster CPU.



161. The Accused Products further contain the second limitation of claim 1 of the '140 patent: notably, each cluster of processors comprises a plurality of processors. Notably, Apple's Neural Engine within the Apple A15 SoC contains 16 processor cores and performs more than 15 trillion operations per second.[60] The GPU within the A15 SoC contains 4 or 5 processor cores, depending on the model (5 for the iPhone Pro and iPhone 13 Pro Max).[61] And on information and belief, Apple's AON processor contains more than one processor within it.

[60] *See* Stephen Shankland, *Apple's A15 Bionic Chip Powers iPhone 13 with 15 Billion Transistors, New Graphics and AI*, CNET (Sept. 14, 2021), https://www.cnet.com/tech/mobile/apples-a15-bionic-chip-powers-iphone-13-with-15-billion-transistors-new-graphics-and-ai/.

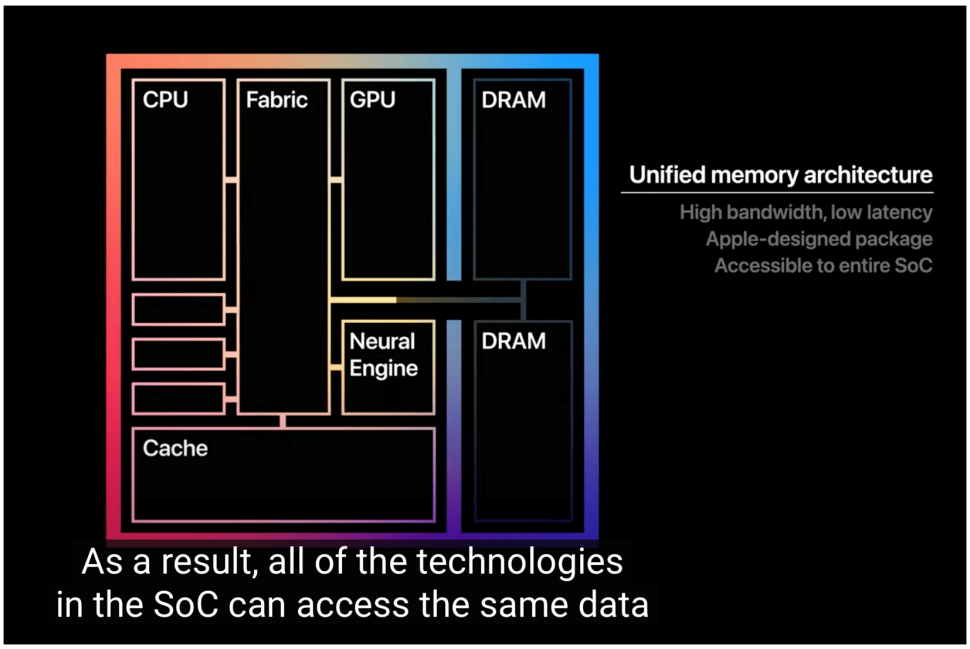
[61] *Id.*; *see also* Filipe Espósito, *iPhone 13 Pro's A15 Bionic Chip Has More Powerful GPU than Regular iPhone 13*, 9 to 5 Mac (Sept. 14, 2021), https://9to5mac.com/2021/09/14/iphone-13-pros-a15-bionic-chip-has-more-powerful-gpu-than-regular-iphone-13/; *see also* Ben Morris, *Imagination Technologies: A British Technology Success Story*, BBC News (May 16, 2012), https://www.bbc.co.uk/news/business-17939014 (noting that Siri "is driven by the GPU").

162. The Accused Products further contain the third limitation of claim 1 of the '140 patent as well—an acoustic model memory storing acoustic model data, wherein each of the plurality of processors is configured to calculate a probability using the acoustic model data in the acoustic model memory. Among other processors, Apple's Neural Engine is used for speech recognition. For example, in November 2020, Apple stated that its M1 SoC contained in the fifth generation iPad Pro, 24-inch iMac (M1), 2020 13-inch MacBrook Pro, MacBook Air, and Mac mini, "features our most advanced Neural Engine with a 16-core design. . . . The entire M1 chip is designed to excel at machine learning, and tasks like . . . voice recognition . . . will have a level of performance never seen before on the Mac."[62]

163. On information and belief, Apple's Neural Engine contains an acoustic model memory storing acoustic model data in the form of on-chip SRAM, DRAM, or cache memory, as part of the M1 chip's unified memory architecture that makes any acoustic model memory available for the Neural Engine, GPU, and/or CPU.[63]

[62] *See* Apple November 10, 2020 Event at 12 minutes 10 seconds, https://www.youtube.com/watch?v=5AwdkGKmZ0I (last visited Jan. 3, 2022).
[63] *See* U.S. Patent Application Publication 2019/0340491 Figures 2 ("[t]he neural engine 314 receives the input data 322, performs multiply-accumulate operations (e.g., convolution operations) on the input data 322 based on stored kernel data")

Amazon / Zentian Limited
Exhibit 1014
Page 41



164. Moreover, the Accused Products contain the fourth and fifth limitations that the speech recognition circuit is configured to generate an initial score for an audio sample and that the initial score is used to determine whether to continue processing to determine a final score via processing a larger amount of model data than what was processed to generate the initial score.

165. In a technical paper published in 2017, Apple's Siri Team described one function of Apple's speech recognition circuit—the "Hey Siri" wake phrase detection in the following manner:

> To avoid running the main processor all day just to listen for the trigger phrase, the iPhone's Always On Processor (AOP) (a small, low-power auxiliary processor, that is the embedded Motion Coprocessor) has access to the microphone signal (on 6S and later). We use a small proportion of the AOP's limited processing power to run a detector with a small version of the acoustic model (DNN). When the score exceeds a threshold the motion coprocessor wakes up the main processor, which analyzes the signal using a larger DNN. In the first versions with AOP support, the first detector used a DNN with 5 layers of 32 hidden units and the second detector had 5 layers of 192 hidden units.[64]

[64] *Hey Siri*.

166. In turn, the "initial score" computed by a the AOP and a small DNN containing a smaller amount of data (5 layers of 32 hidden units) is used to determine whether to continue processing to determine a final score computed by a "larger DNN" that contains a larger amount of model data than what was processed to generate the initial score (5 layers of 192 hidden units).

167. Apple actively, knowingly, and intentionally induces infringement of one or more claims of the '140 patent under 35 U.S.C. § 271(b) by actively encouraging others to make, use, sell, and offer to sell in the United States, the Accused Products.

168. For example, Apple provides directions, instruction manuals, guides, and/or other materials that encourage and facilitate infringing use by others.

169. Apple has sold and is selling the Accused Products with the knowledge and intent that customers who buy the products will use the products for their infringing use and therefore that customers have been and are directly infringing the '140 patent.

170. Apple contributes to the infringement of one or more claims of the '140 patent under 35 U.S.C § 271(c) by offering to sell, selling, or importing into the United States, a component of the Accused Products that constitutes a material part of the inventions, knowing the same to be especially made or especially adapted for use in an infringement of the '140 patent, and is not a staple article or commodity of commerce suitable for substantial noninfringing use.

171. Apple was aware of the '140 patent at least from the date of this Complaint.

172. On information and belief, Apple has been aware of the '140 patent since its issuance on April 6, 2021.

173. On information and belief, Apple knew that the Accused Products infringe the '140 patent, or at a minimum believed there was a high probability that the Accused Products were

covered by Zentian's patents, but willfully blinded itself to Zentian's patents and the infringing nature of the Accused Products.

174. The foregoing allegations are based on publicly available information and a reasonable investigation of the structure and operation of the Accused Products.

175. Zentian reserves the right to modify this description of alleged infringement, including, for example, on the basis of information about the Accused Products that it obtains during discovery.

176. The infringement details provided for claim 1 of the '140 patent are exemplary, and Zentian intends to assert additional claims of the '140 patent beyond claim 1.

177. Apple's infringement has damaged and continues to damage Zentian in an amount to be determined at trial, but in no case less than a reasonable royalty for Apple's unauthorized use of the claimed inventions set forth in the '140 patent.

178. This is an exceptional case. Zentian is entitled to attorneys' fees and costs under 35 U.S.C § 285 as a result of the infringement of the '140 patent by Apple.

COUNT FOUR: INFRINGEMENT OF U.S. PATENT NO. 10,062,377

179. Zentian incorporates by reference and realleges the allegations in paragraphs 1-178.

180. The '377 patent is valid and enforceable.

181. The '377 patent names Guy Larri, Mark Catchpole, Damian Kelly Harris-Dowsett, and Timothy Brian Reynolds as inventors.

182. Apple has directly infringed, continues to infringe, and has induced or contributed to the infringement of the '377 patent by making, using, selling, offering for sale, and/or importing into the United States, without authority or license the Accused Products in violation of 35 U.S.C. § 271(a).

183. The Accused Products are non-limiting examples identified based on publicly available information. Zentian reserves the right to identify additional infringing activities, products, and services, including, for example, on the basis of information obtained during discovery.

184. By way of example only, the Accused Products meet all the limitations of at least independent claim 1 of the '377 patent, and dependent claims 2-6, either literally or under the doctrine of equivalents.

185. Claim 1 of the '377 patent recites:

A speech recognition system, comprising:

> [1] a first programmable device programmed to calculate a feature vector from a digital audio stream, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said digital audio stream during a defined audio time frame;
>
> [2] a second programmable device programmed to calculate distances indicating the similarity between a feature vector and a plurality of acoustic states of an acoustic model wherein said feature vector is received by the second programmable device after it is calculated by the first programmable device; and
>
> [3] a third programmable device programmed to identify spoken words in said digital audio stream using Hidden Markov Model and/or Neural Networks wherein said word identification uses one or more distances that were calculated by the second programmable device, wherein said identification of spoken words uses one or more distances calculated from a first feature vector; and
>
> [4] a search stage for using the calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words.

186. Regarding the preamble of claim 1, to the extent the preamble is determined to be limiting, the Accused Products contain a speech recognition system that allows for device users to speak the words "Hey Siri" and activate Apple's personal assistant hands-free. The speech

Amazon / Zentian Limited
Exhibit 1014
Page 45

recognition system of the Accused Products further allows for dictation of text messages and notes with integrated applications, as well as on device speech recognition for Siri user requests.[65]

187. On information and belief, the speech recognition system of the Accused Products works with cloud-based programming, natural language interpretation and other services with hardware-assisted on device processing, creating a combination of hardware and software features.[66]

188. In October 2017, Apple's Siri Team published an article on Apple's website, entitled "Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant."[67] That article states that a "specialized speech recognizer" incorporated on the local devices themselves "runs all the time and listens" for the words "Hey Siri," which allows for the feature to operate hands-free and respond even when the Accused Product does not have a data connection.[68]

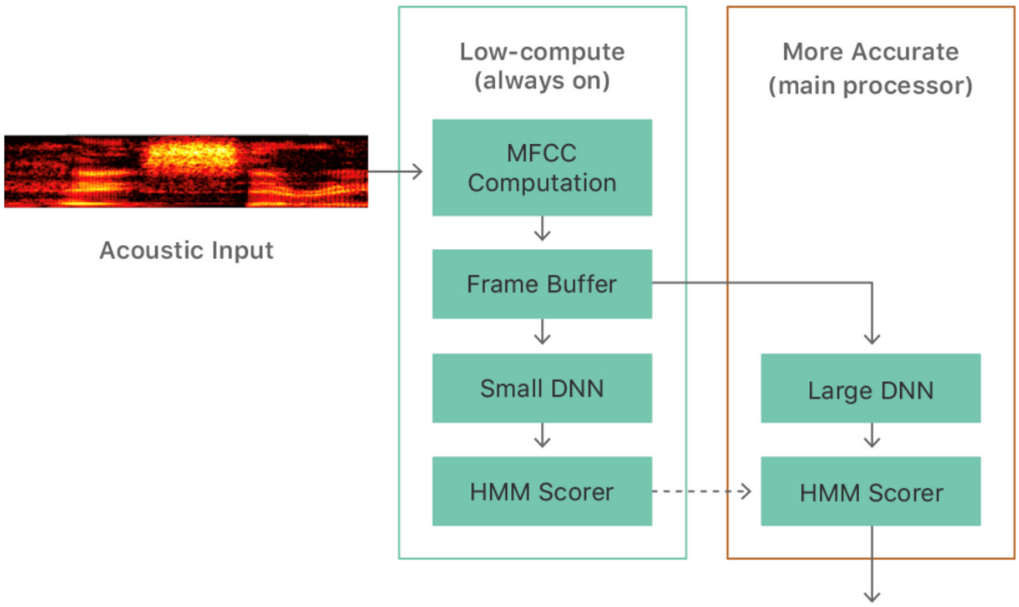
189. As to the first limitation of claim 1 of the '377 patent, the Accused Products contain a first programmable device programmed to calculate a feature vector from a digital audio stream, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said digital audio stream during a defined audio time frame.

190. As shown below in a figure published by Apple's Siri Team, the "Always On Processor (AOP)" of the Accused Products is a programmable device, described as "a small, low-

[65] *See* Alex Hern, *Apple Overhauls Siri to Address Privacy Concerns and Improve Performance*, The Guardian (June 7, 2021), https://www.theguardian.com/technology/2021/jun/07/apple-overhauls-siri-to-address-privacy-concerns-and-improve-performance.
[66] *See, e.g.*, Christian Ziberg, *New Machine Learning Article from Apple Goes In Depth on How "Hey Siri" Does Its Magic*, iDB (Oct. 18, 2017), https://www.idownloadblog.com/2017/10/18/apple-hey-siri-machine-learning-article/.
[67] *Hey Siri*.
[68] *Id.*

Amazon / Zentian Limited
Exhibit 1014
Page 46

power auxiliary processor" that has access to "the microphone signal."[69] The AOP calculates feature vectors through an extraction method known as MFCC (or Mel Frequency Cepstrum Coefficient) calculations. MFCCs are a collection of coefficients that represent the short-term power spectrum of a sound.[70]

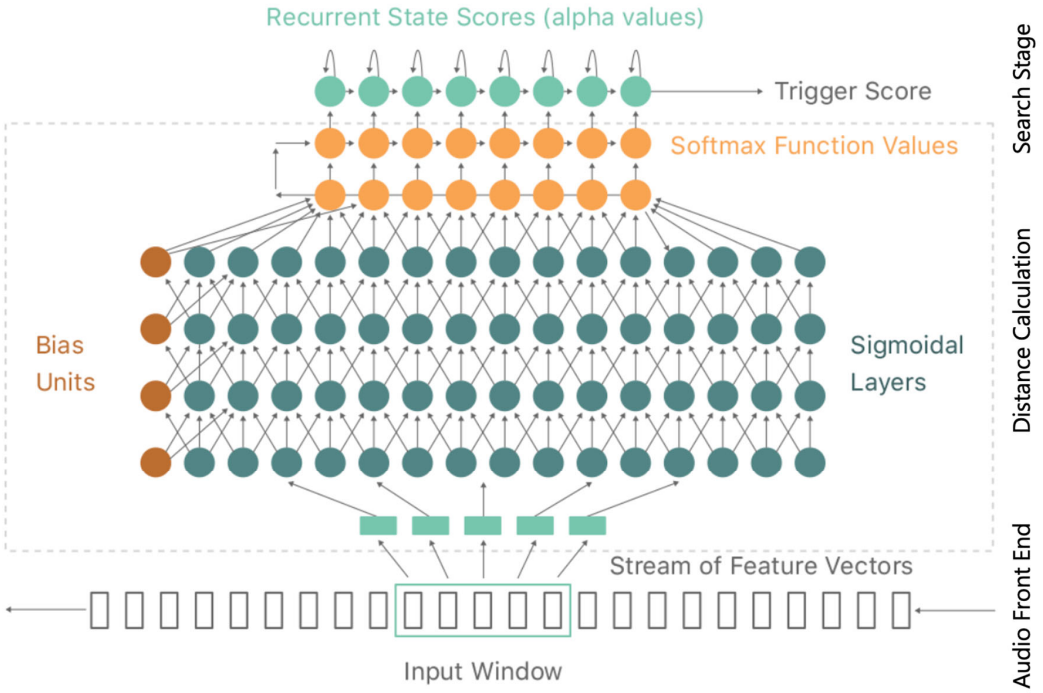


191. Apple's Siri Team further describes the Accused Products' calculation of a feature vector from an audio signal in the following way:

> The microphone in an iPhone or Apple watch turns your voice into a stream of instantaneous waveform samples, at a rate of 16000 per second. A spectrum analysis stage converts the waveform sample stream to a sequence of frames, each describing the sound spectrum of approximately 0.01 sec.[71]

[69] *Id.*
[70] *See, e.g.*, K.S. Rao and Manjunath K.E., *Speech Recognition Using Articulatory and Excitation Source Features,* Appendix A: MFCC Features at 85, SpringerBriefs in Speech Technology, https://link.springer.com/content/pdf/bbm%3A978-3-319-49220-9%2F1.pdf.
[71] *Hey Siri*.

192. The below speech recognizer diagram (annotation added in dark text on the right), also published in the Siri Team's technical paper, illustrates generally how the speech recognition system of the Accused Products converts a spoken utterance of the words "Hey Siri" into a function, such as waking the device. Relevant to the first limitation, the "input window" (green box at bottom) receives a "stream of feature vectors" calculated by an audio front end (the MFCC Computation on the AOP).[72]



193. The Accused Products contain the second limitation of claim 1 of the '377 patent: notably, they contain a second programmable device programmed to calculate distances (*i.e.*, probabilities) indicating the similarity between the feature vector received from the first programable device and a plurality of acoustic states of an acoustic model.

[72] *Id.* at Figure 2 (annotations added).

Amazon / Zentian Limited
Exhibit 1014
Page 48

194. As shown above in the figure published by Apple's Siri Team, the "More Accurate (main processor)" of the Accused Products is a second programmable device designed to calculate probabilities that indicate the similarity between the feature vector received from the first programable device and a plurality of acoustic states of an acoustic model.

195. The acoustic model contained in the Accused Products is a Deep Neural Network (DNN). The DNN converts the 0.2 seconds of audio (approximately twenty frames of 0.01 seconds) it receives into probability distributions consisting of extracted and derived quantities, like the 20 speech sound classes used in "Hey Siri" phrase, plus silence and other speech.[73]

196. In the Siri Team's technical paper, Apple states that the "output of the acoustic model provides a distribution of scores over phonetic classes for every frame. A phonetic class is typically something like 'the first part of an /s/ preceded by a high front vowel and followed by a front vowel.'" Phonetic classes of this type are otherwise known in the art as a triphones.[74]

197. Elsewhere in the same paper, Apple describes the DNN acoustic model (shown in the figure above bounded by a dashed grey box receiving feature vectors within the "input window") as follows:

> "DNN consists mostly of matrix multiplications and logistic nonlinearities. Each 'hidden' layer [e.g. circles connected with arrows identified as sigmoidal layers, bias units, and/or softmax function values] is an intermediate representation discovered by the DNN during its training to convert the filter bank inputs to sound classes. The final nonlinearity is essentially a Softmax function (a.k.a. a general logistic or normalized exponential)."[75]

198. The underlying mathematical equation (below) for this "distance" calculation that is the output of the acoustic model is represented by the variable $q_{i,t}$—"the log score for the

[73] *Id.*
[74] *Id.*
[75] *Id.*

Amazon / Zentian Limited
Exhibit 1014
Page 49

phonetic class associated with the *ith* state given the acoustic pattern around time *t*."[76] In turn, the calculation is a mathematical comparison of the similarity between the feature vector received from the AOP and acoustic states of the DNN.

$$F_{i,t} = \max\{s_i + F_{i,t-1}, m_{i-1} + F_{i-1,t-1}\} + q_{i,t}$$

where

- $F_{i,t}$ is the accumulated score for state *i* of the model
- $q_{i,t}$ is the output of the acoustic model—the log score for the phonetic class associated with the *ith* state given the acoustic pattern around time *t*
- s_i is a cost associated with staying in state *i*
- m_i is a cost for moving on from state *i*

199. On information and belief, due to its power and computational requirements, the "Large DNN" (as compared to the "Small DNN" on the AOP) in the Accused Products runs on a separate (second) processor, which is a programmable device such as in the Apple Silicon M1 Mac, a Neural Engine with 16-core design processor.[77] Or alternatively, for example, in iPhone 12 and iPad Air 4th generation models that use the A14 Bionic system on chip, the Graphics Processing Unit (GPU) depicted at the top of the A14 Bionic chip die image to the right.[78]

200. The Accused Products contain the third limitation of claim 1 of the '377 patent as well— a third programmable device programmed to identify spoken words in a digital audio

[76] *Id.*

[77] *See* Press Release, *Introducing M1 Pro and M1 Max: The Most Powerful Chips Apple Has Ever Built* (Oct. 18, 2021), https://www.apple.com/newsroom/2021/10/introducing-m1-pro-and-m1-max-the-most-powerful-chips-apple-has-ever-built/. In November 2020, Apple stated that its M1 SoC "features our most advanced Neural Engine with a 16-core design. The entire M1 chip is designed to excel at machine learning, and tasks like . . . voice recognition . . . will have a level of performance never before seen on the Mac." Apple November 10, 2020 Event at 12 minutes 10 seconds, https://www.youtube.com/watch?v=5AwdkGKmZ0I.

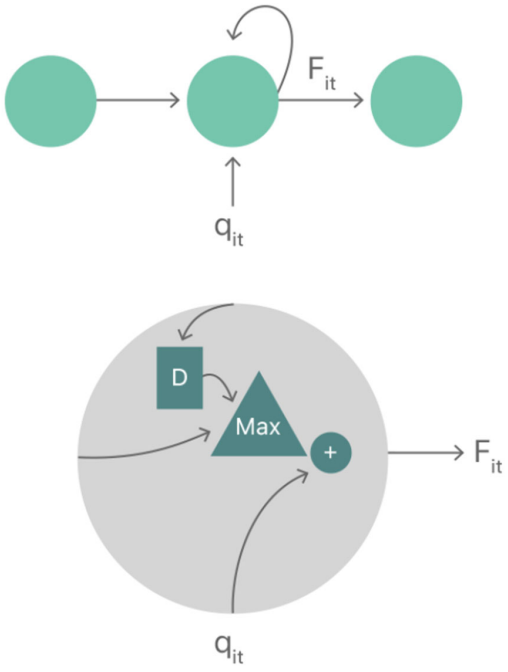
[78] *See, e.g.*, Ben Morris, *Imagination Technologies: A British technology success story*, BBC News (May 16, 2012), https://www.bbc.co.uk/news/business-17939014 (quoting Apple GPU provider for certain iPhone models that "Apple's popular voice command system, Siri, is driven by the GPU").

Amazon / Zentian Limited
Exhibit 1014
Page 50

stream using either a Hidden Markov Model and/or a Neural Network. The identification of spoken words by the third programmable device uses one or more of the distances calculated by the second programmable device, which themselves were calculated from the feature vectors calculated by the first programmable device.

201. On information and belief, the temporal integration computation that occurs in the final stage of the speech recognition process (the search stage) identifies spoken words from the digital audio stream and occurs on a third programmable device, likely another processor, such as one of the main application processors. For example, there are six application processor CPU cores on the Apple A14 Bionic chip, as shown on the die photo above (two of which are labelled "CPU1" and four of which are labelled "CPU2").[79]

202. In the speech recognizer diagram, this search stage is depicted by the final top-layer of light green circles outside of the dotted-line box that surrounds the computationally intensive acoustic model that functions to calculate the "distances" described above. Combining that image with the mathematical equation above shows that the output of the acoustic model, q_{it}, is fed into this last stage, which Apple describes as an "application of dynamic programming, and can be derived based on ideas about Hidden Markov Models—HMMs."[80]



[79] *Hey Siri*.
[80] *Id.*

Amazon / Zentian Limited
Exhibit 1014
Page 51

203. Moreover, the Accused Products contain the fourth limitation of claim 1 of the '377 patent: namely, they include a search stage for using the calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words.

204. The "Hey Siri" system contains a "language-specific phonetic specification of the 'Hey Siri' phrase" that was created with two English variants: one with Siri as in "serious" and two with Siri as in "Syria."[81] According to the Siri Team's technical paper on Apple's website, this model of words (i.e. the lexical tree) was built to "cope with a short break between the two words, especially as the phrase is often written with a comma: 'Hey, Siri.' Each phonetic symbol results in three speech sound classes (beginning, middle and end) each of which as its own output from the acoustic model."[82]

205. Apple's instructions to application developers looking to incorporate the Siri technology provides further support for the existence of a search stage to identify words within a lexical tree. In the SiriKit for developers, Apple describes a "vocabulary step," consisting of a vocabulary database that can be adapted and added to, and that comes into play between when Siri recognizes speech from the user and creating an action that Siri can execute.[83]

206. Apple actively, knowingly, and intentionally induces infringement of one or more claims of the '377 patent under 35 U.S.C. § 271(b) by actively encouraging others to make, use, sell, and offer to sell in the United States, the Accused Products.

207. For example, Apple provides directions, instruction manuals, guides, and/or other materials that encourage and facilitate infringing use by others.

---

[81] *Hey Siri*.
[82] *Id*.
[83] *See* YML Innovation Lab, *Learn How to Integrate Siri with Your App Thanks to Apple SiriKit* (June 29, 2016), https://ymedialabs.medium.com/learn-how-to-integrate-siri-with-your-app-thanks-to-apple-sirikit-c6e570d302cb#1.

Amazon / Zentian Limited
Exhibit 1014
Page 52

208. Apple has sold and is selling the Accused Products with the knowledge and intent that customers who buy the products will use the products for their infringing use and therefore that customers have been and are directly infringing the ’377 patent.

209. Apple contributes to the infringement of one or more claims of the ’377 patent under 35 U.S.C § 271(c) by offering to sell, selling, or importing into the United States, a component of the Accused Products that constitutes a material part of the inventions, knowing the same to be especially made or especially adapted for use in an infringement of the ’377 patent, and is not a staple article or commodity of commerce suitable for substantial noninfringing use.

210. Apple was aware of the ’377 patent at least from the date of this Complaint.

211. On information and belief, Apple has been aware of the ’377 patent since its issuance on August 28, 2018.

212. On information and belief, Apple knew that the Accused Products infringe the ’377 patent, or at a minimum believed there was a high probability that the Accused Products were covered by Zentian’s patents, but willfully blinded itself to Zentian’s patents and the infringing nature of the Accused Products.

213. The foregoing allegations are based on publicly available information and a reasonable investigation of the structure and operation of the Accused Products.

214. Zentian reserves the right to modify this description of alleged infringement, including, for example, on the basis of information about the Accused Products that it obtains during discovery.

215. The infringement details provided for claim 1 of the ’377 patent are exemplary, and Zentian intends to assert additional claims of the ’377 patent beyond claim 1.

Amazon / Zentian Limited
Exhibit 1014
Page 53

216. Apple's infringement has damaged and continues to damage Zentian in an amount to be determined at trial, but in no case less than a reasonable royalty for Apple's unauthorized use of the claimed invention set forth in the '377 patent.

217. This is an exceptional case. Zentian is entitled to attorneys' fees and costs under 35 U.S.C § 285 as a result of the infringement of the '377 patent by Apple.

COUNT FIVE: INFRINGEMENT OF U.S. PATENT NO. 10,839,789

218. Zentian incorporates by reference and realleges the allegations in paragraphs 1-217.

219. The '789 patent is valid and enforceable.

220. The '789 patent names Guy Larri, Mark Catchpole, Damian Kelly Harris-Dowsett, and Timothy Brian Reynolds as inventors.

221. Apple has directly infringed, continues to infringe, and has induced or contributed to the infringement of the '789 patent by making, using, selling, offering for sale, and/or importing into the United States, without authority or license the Accused Products in violation of 35 U.SC. § 271(a).

222. The Accused Products are non-limiting examples identified based on publicly available information. Zentian reserves the right to identify additional infringing activities, products, and services, including, for example, on the basis of information obtained during discovery.

223. By way of example only, the Accused Products meet all the limitations of at least claim 10 of the '789 patent, either literally or under the doctrine of equivalents, which recites:

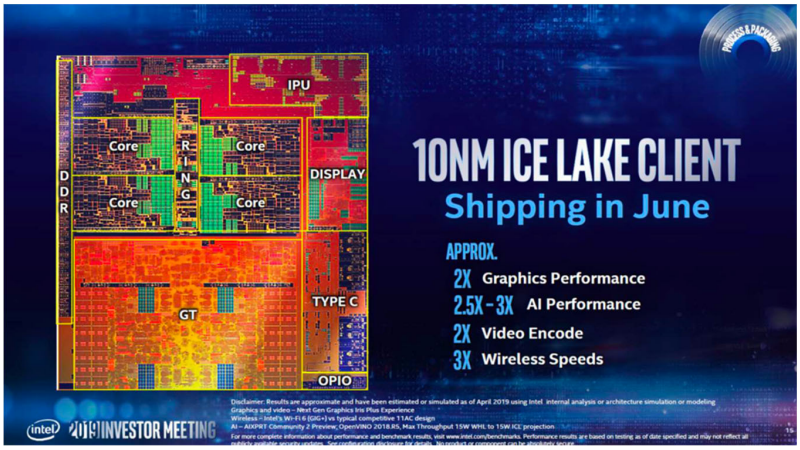
> An acoustic coprocessor, comprising:
>
> [1] a first interface for receiving at least one feature vector;
>
> [2] a calculating apparatus for calculating distances indicating a similarity between the at least one feature vector and at least one acoustic state of an acoustic model read from an acoustic model memory; and

Amazon / Zentian Limited
Exhibit 1014
Page 54

[3] a second interface for sending at least one distance calculated by the calculating apparatus.

224. Regarding the preamble of claim 10, to the extent the preamble is determined to be limiting, the Accused Products contain an acoustic coprocessor. For example, the 2020 MacBook Pro 13-Inch Core-i5 2.0GHz laptop computer contains an Intel Gaussian & Neural Accelerator (GNA). According to Intel, the Intel GNA "delivers a dedicated engine for background workloads such as voice processing and noise suppression at ultra-low power, for maximum battery life."[84]

225. The Intel GNA is located on the "Ice Lake" Core-i5 processor on the 2020 MacBook Pro 2.0GHz, as marketed by Intel upon its release in 2019.[85]



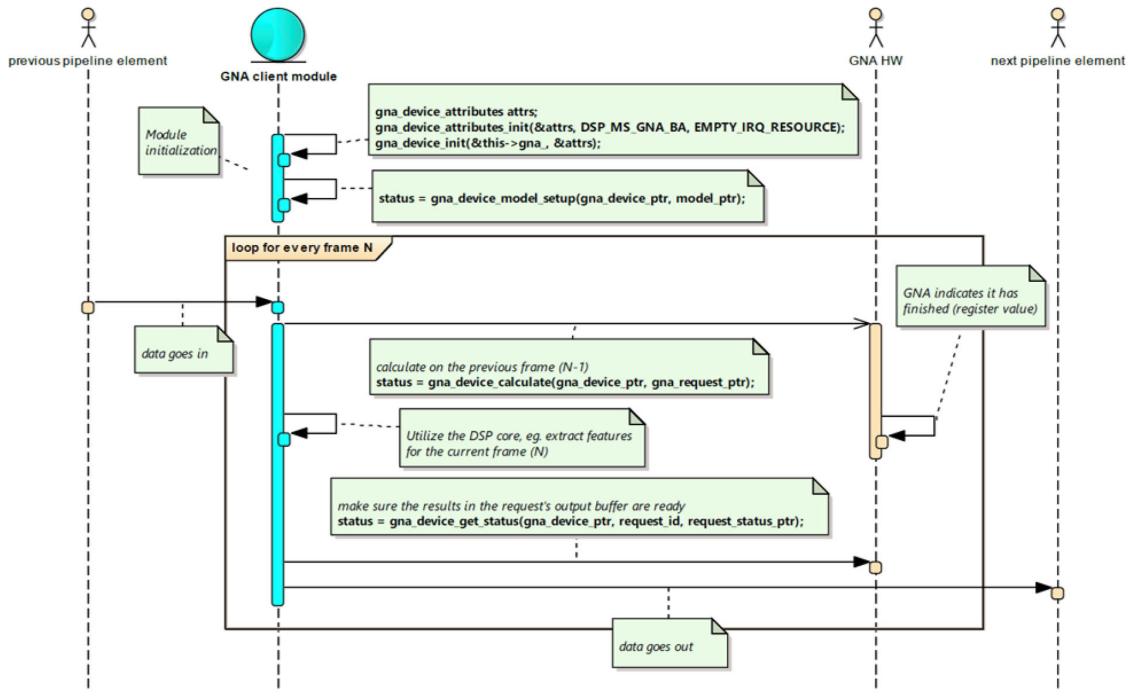
226. The Intel GNA is a "flexible, low-power, high performance streaming co-processor that offloads neural network inference operations . . . ."[86] As explained by Intel in the programming guide excerpted below, the GNA is capable of utilizing extracted features from the DSP core that

[84] *See* Intel Newsroom, *Intel Launches First 10th Gen Intel Core Processors: Redefining the Next Era of Laptop Experiences* (Aug. 1, 2019), https://newsroom.intel.com/news/intel-launches-first-10th-gen-intel-core-processors-redefining-next-era-laptop-experiences/#gs.iwvy00.

[85] *See, e.g.,* Apple MacBook Pro 13-Inch "Core i5" 2.0 2020 4 TB 3 Specs, https://everymac.com/systems/apple/macbook_pro/specs/macbook-pro-core-i5-2.0-quad-core-13-2020-4-thunderbolt-3-ports-scissor-specs.html (identifying Intel® Core™ i5-1038NG7 Processor); Intel Product Specification, Intel® Core™ i5-1038NG7 Processor, https://ark.intel.com/content/www/us/en/ark/products/196594/intel-core-i51038ng7-processor-6m-cache-up-to-3-80-ghz.html (identifying GNA in "Advanced Technologies").

[86] *See* G. Stemmer et al., *Speech Recognition and Understanding on Hardware-Accelerated DSP*, Interspeech (2017).

are calculated for frame (N) while calculating the acoustic model for the previous frame (N-1), such that the GNA is processing one frame behind the DSP.[87]



In other words, the Intel GNA processes feature vectors (i.e. data) associated with audio signals received from the microphones (via the DSP) in the 2020 MacBook Pro 2.0GHz and is, in turn, an acoustic coprocessor.

[87] *See* Intel® Quark™ SoC S1000 Firmware Programming Guide, https://www.intel.com/content/www/us/en/developer/articles/guide/firmware-programming-guide-s1000.html.

227. As to the first limitation of claim 10 of the '789 patent, the Accused Products contain a first interface for receiving at least one feature vector. Specifically, the block diagram at right, shows the Intel GNA with an interface for receiving feature vectors from an audio signal, shown as the end point of arrows from data arrays and network parameter arrays to input data buffers designed to "hold inputs…of each layer."[88]

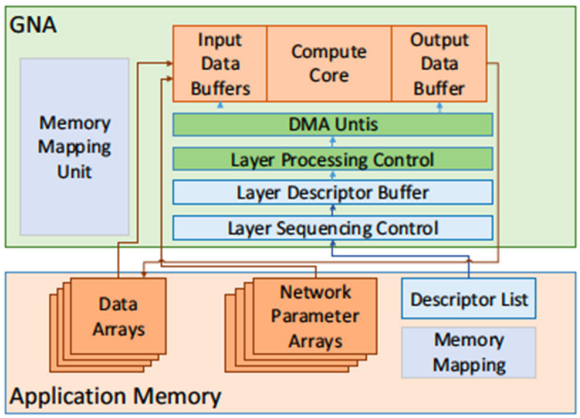


Figure 2: *GNA block diagram and memory organization*

228. The Accused Products also contain the second limitation of claim 10 of the '789 patent. On information and belief, the "Compute Core" of the Intel GNA, depicted in the figure above between the "Input Data Buffer" and "Output Data Buffer" is a calculating apparatus for calculating distances indicating a similarity between feature vectors and acoustic states of an acoustic model, read from an acoustic memory.

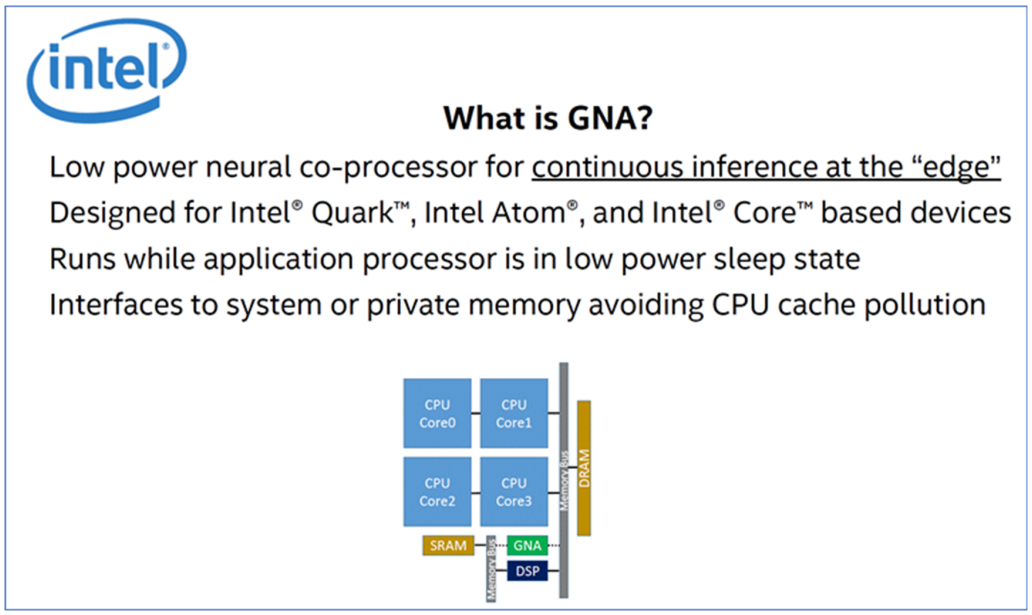
229. At the 2017 International Conference of Acoustics, Speech, and Signal Processing (ICASSP), representatives of Intel demonstrated the GNA performing "continuous acoustic model likelihood scoring" using a 6-layer DNN with 2048 hidden nodes acoustic model. The demonstration showed the utility of the Intel GNA for highly accurate, low power, speech recognition compared to the application processor itself.[89]

---

[88] *See* G. Stemmer et al., *Speech Recognition and Understanding on Hardware-Accelerated DSP*, Interspeech (2017) at Figure 2.

[89] *See* Michael Deisher and Andrzej Polonsky, *Implementation of Efficient, Low Power Deep Neural Networks on Next-Generation Intel Client Platforms*, ICASSP (Apr. 12, 2017),

230. Moreover, the compute core of the GNA retrieves the acoustic model from an acoustic model memory, as shown below in a poster presented at ICASSP 2017 by Intel employees.[90]



231. The Accused Products further contain the third limitation of claim 10 of the '789 patent, a second interface for sending at least one distance calculated by the calculating apparatus. Returning to the GNA block diagram and memory organization figure for the Intel GNA, a second interface for sending distances calculated by the Compute Core is shown as the starting point of an arrow from the output data buffer to the "Data Arrays."[91] The Output Data Buffers hold the outputs of each layer.[92]

---

https://sigport.org/documents/implementation-efficient-low-power-deep-neural-networks-next-generation-intel-client.

[90] *See* Michael Deisher, et al., ICASSP GNA poster presentation "Implementation of Efficient, Low Power Deep Neural Networks on Next-Generation Intel Client Platforms," (ICASSP 2017), https://sigport.org/sites/all/modules/pubdlcnt/pubdlcnt.php?fid=1610 (noting that GNA can be programmed to compute a range of mathematical functions which are used to implement Neural Network based acoustic models, such as Affine, Convolutional, etc.).

[91] *Id.*

[92] *Id.*

Amazon / Zentian Limited
Exhibit 1014
Page 58

232. Apple actively, knowingly, and intentionally induces infringement of one or more claims of the '789 patent under 35 U.S.C. § 271(b) by actively encouraging others to make, use, sell, and offer to sell in the United States, the Accused Products.

233. For example, Apple provides directions, instruction manuals, guides, and/or other materials that encourage and facilitate infringing use by others.

234. Apple has sold and is selling the Accused Products with the knowledge and intent that customers who buy the products will use the products for their infringing use and therefore that customers have been and are directly infringing the '789 patent.

235. Apple contributes to the infringement of one or more claims of the '789 patent under 35 U.S.C § 271(c) by offering to sell, selling, or importing into the United States, a component of the Accused Products that constitutes a material part of the inventions, knowing the same to be especially made or especially adapted for use in an infringement of the '789 patent, and is not a staple article or commodity of commerce suitable for substantial noninfringing use.

236. Apple was aware of the '789 patent at least from the date of this Complaint.

237. On information and belief, Apple has been aware of the '789 patent since its issuance on November 17, 2020.

238. On information and belief, Apple knew that the Accused Products infringe the '789 patent, or at a minimum believed there was a high probability that the Accused Products were covered by Zentian's patents, but willfully blinded itself to Zentian's patents and the infringing nature of the Accused Products.

239. The foregoing allegations are based on publicly available information and a reasonable investigation of the structure and operation of the Accused Products.

Amazon / Zentian Limited
Exhibit 1014
Page 59

240. Zentian reserves the right to modify this description of alleged infringement, including, for example, on the basis of information about the Accused Products that it obtains during discovery.

241. The infringement details provided for claim 10 of the '789 patent are exemplary, and Zentian intends to assert additional claims of the '789 patent beyond claim 10.

242. Apple's infringement has damaged and continues to damage Zentian in an amount to be determined at trial, but in no case less than a reasonable royalty for Apple's unauthorized use of the claimed inventions set forth in the '789 patent.

243. This is an exceptional case. Zentian is entitled to attorneys' fees and costs under 35 U.S.C § 285 as a result of the infringement of the '789 patent by Apple.

DEMAND FOR A JURY TRIAL

244. Zentian requests a trial by jury of all issues so triable under Federal Rule of Civil Procedure 38.

PRAYER FOR RELIEF

WHEREFORE, Zentian respectfully requests the following:

A. That Judgment be entered that Apple has infringed one or more claims of each of the Asserted Patents, directly and indirectly, literally and/or under the doctrine of equivalents;

B. An award of damages sufficient to compensate Zentian for Apple's infringement under 35 U.S.C. § 284, including an enhancement of damages on account of Apple's egregious willful infringement;

C. That the case be found exceptional under 35 U.S.C. § 285 and that Zentian be awarded its reasonable attorneys' fees;

D. Costs and expenses in this action;

E. An award of prejudgment and post-judgment interest; and

Amazon / Zentian Limited
Exhibit 1014
Page 60

F. Such other and further relief as the Court may deem just and proper.

Dated: February 2, 2022

By: */s/ Jonathon K. Hance*

Jonathon K. Hance (TX Bar No. 24065364)
Lee Andrew Taggart (TX Bar No. 24093238)
BRACEWELL LLP
711 Louisiana St., Suite 2300
Houston, Texas 77002
Tel: (713) 223-2300
Fax: (800) 404-3970
jonathon.hance@bracewell.com
drew.taggart@bracewell.com

Nevin M. Gewertz (*pro hac vice* to be filed)
Rebecca T. Horwitz (*pro hac vice* to be filed)
BARTLIT BECK LLP
54 West Hubbard Street
Chicago, Illinois 60654
Tel: (312) 494-4400
Fax: (312) 494-4440
nevin.gewertz@bartlitbeck.com
rebecca.horwitz@bartlitbeck.com

John M. Hughes (*pro hac vice* to be filed)
Katherine L.I. Hacker (TX Bar No. 24074338)
BARTLIT BECK LLP
1801 Wewatta Street, Suite 1200
Denver, Colorado 80202
Tel: (303) 592-3100
Fax: (303) 592-3140
john.hughes@bartlitbeck.com
kat.hacker@bartlitbeck.com

Attorneys for Plaintiff Zentian Limited

Exhibit A

US007979277B2

(12) **United States Patent**
Larri et al.

(10) **Patent No.: US 7,979,277 B2**
(45) **Date of Patent: Jul. 12, 2011**

(54) **SPEECH RECOGNITION CIRCUIT AND METHOD**

(75) Inventors: **Guy Larri**, Cambridge (GB); **Mark Catchpole**, Cambridge (GB); **Damian Kelly Harris-Dowsett**, Coventry (GB); **Timothy Brian Reynolds**, Cambridge (GB)

(73) Assignee: **Zentian Limited**, Cambridge (GB)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1043 days.

(21) Appl. No.: **11/662,704**

(22) PCT Filed: **Sep. 14, 2005**

(86) PCT No.: **PCT/GB2005/003554**

§ 371 (c)(1),
(2), (4) Date: **Mar. 14, 2007**

(87) PCT Pub. No.: **WO2006/030214**

PCT Pub. Date: **Mar. 23, 2006**

(65) **Prior Publication Data**

US 2008/0255839 A1 Oct. 16, 2008

(30) **Foreign Application Priority Data**

Sep. 14, 2004 (GB) 0420464.0

(51) **Int. Cl.**
G10L 15/00 (2006.01)

(52) **U.S. Cl.** **704/238**

(58) **Field of Classification Search** ..... None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,881,312 A 3/1999 Dulong
7,328,153 B2 * 2/2008 Wells et al. 704/231

OTHER PUBLICATIONS

Glinski et al., "Spoken Language Recognition on a DSP Array Processor" IEEE Transactions on Parallel and Distributed Systems, vol. 5, No. 7, pp. 697-703, Jul. 1, 1994.

* cited by examiner

Primary Examiner — Susan McFadden
(74) *Attorney, Agent, or Firm* — Dickstein Shapiro LLP

(57) **ABSTRACT**

A speech recognition circuit comprising a circuit for providing state identifiers which identify states corresponding to nodes or groups of adjacent nodes in a lexical tree, and for providing scores corresponding to said state identifiers, the lexical tree comprising a model of words; a memory structure for receiving and storing state identifiers identified by a node identifier identifying a node or group of adjacent nodes, said memory structure being adapted to allow lookup to identify particular state identifiers, reading of the scores corresponding to the state identifiers, and writing back of the scores to the memory structure after modification of the scores; an accumulator for receiving score updates corresponding to particular state identifiers from a score update generating circuit which generates the score updates using audio input, for receiving scores from the memory structure, and for modifying said scores by adding said score updates to said scores; and a selector circuit for selecting at least one node or group of adjacent nodes of the lexical tree according to said scores.

16 Claims, 27 Drawing Sheets

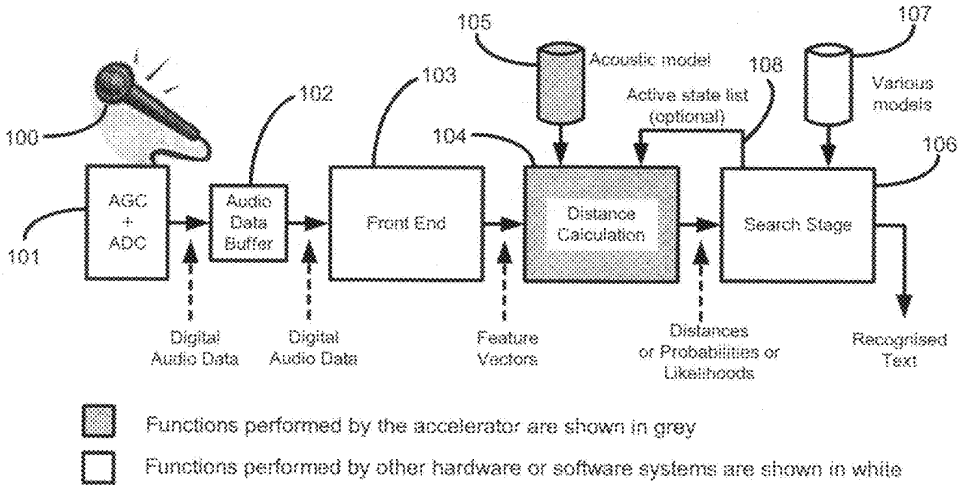


Figure 1a

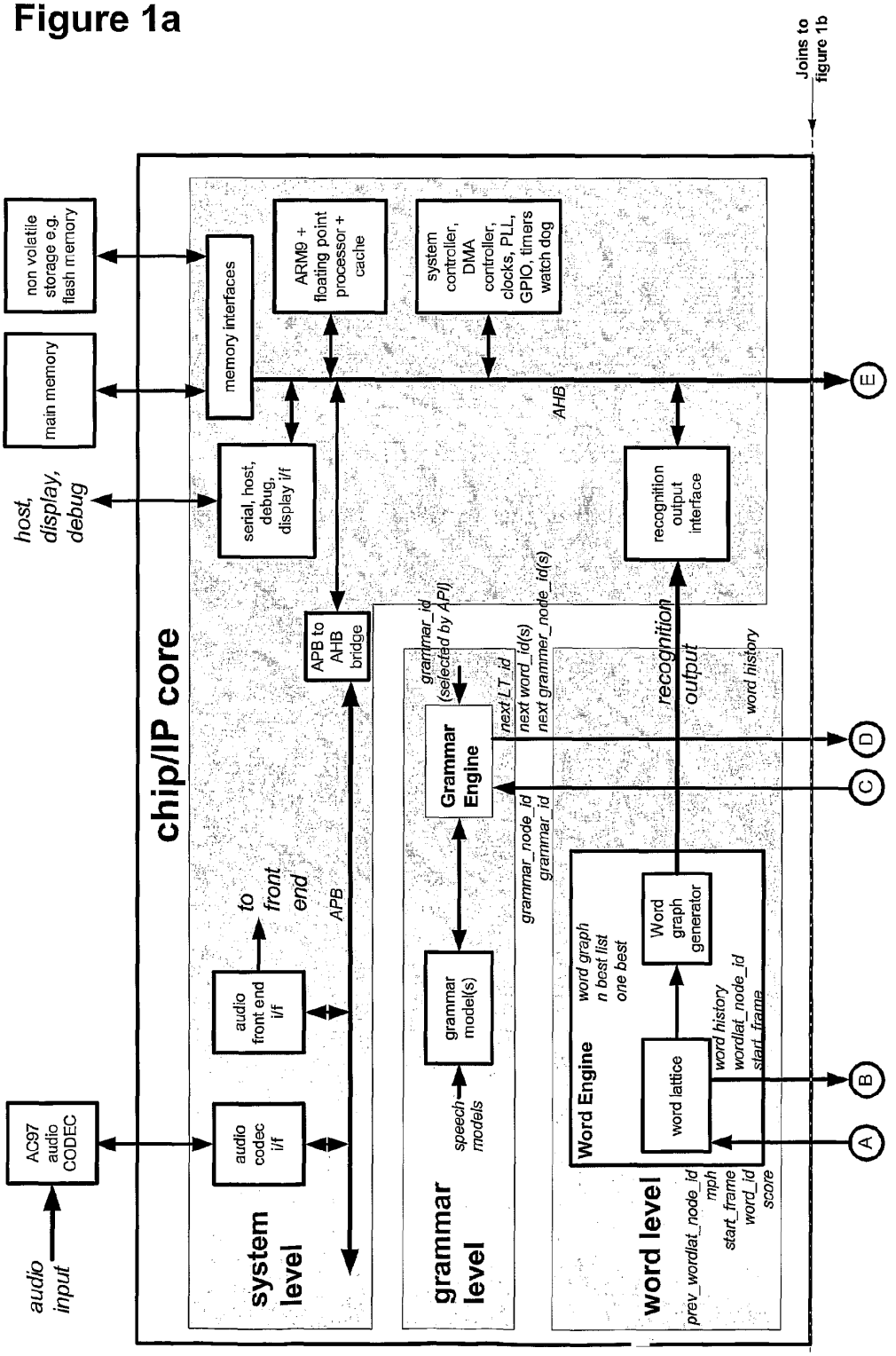

Joins to figure 1b
non volatile storage e.g. flash memory
main memory
host, display, debug
audio input
AC97 audio CODEC
chip/IP core
memory interfaces
ARM9 + floating point processor + cache
system controller, DMA controller, clocks, PLL, GPIO, timers watch dog
AHB
serial, host, debug, display i/f
recognition output interface
APB to AHB bridge
audio codec i/f
audio front end i/f
to front end
APB
system level
grammar_id (selected by API)
next LT_id
next word_id(s)
next grammar_node_id(s)
Grammar Engine
recognition output
word history
grammar_node_id
grammar_id
speech models
grammar model(s)
grammar level
Word Engine
word graph
n best list
one best
Word graph generator
word lattice
word history
wordlat_node_id
start_frame
prev_wordlat_node_id
mph
start_frame
word_id
score
word level
A
B
C
D
E

Figure 1b

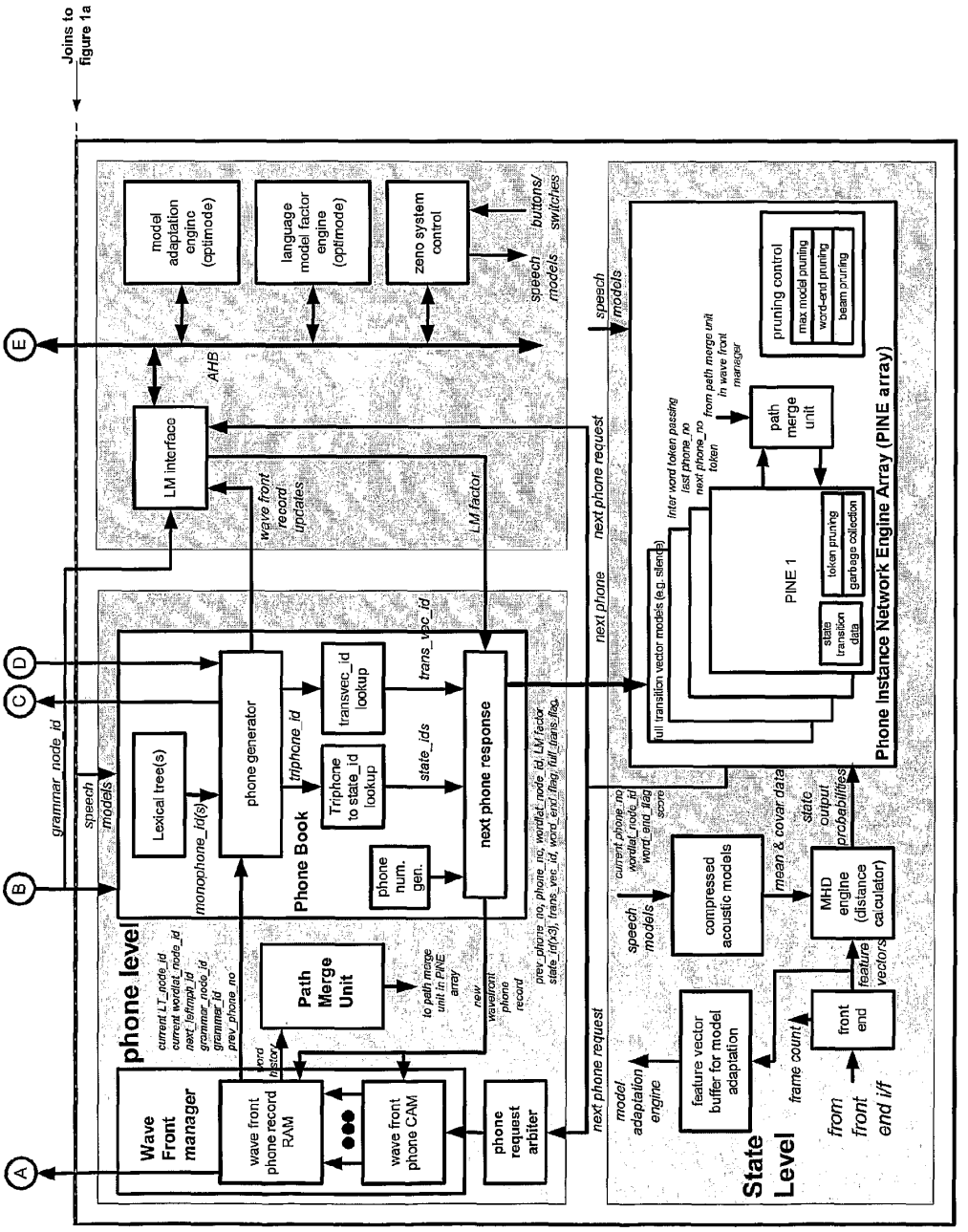

Joins to figure 1a
model adaptation engine (optimode)
language model factor engine (optimode)
zeno system control
buttons/ switches
speech models
AHB
LM interface
wave front record updates
LM factor
next phone request
next phone
Phone Instance Network Engine Array (PINE array)
pruning control
max model pruning
word-end pruning
beam pruning
path merge unit
inter word token passing
last phone_no
next phone_no
token
from path merge unit in wave front manager
PINE 1
token pruning
garbage collection
state transition data
full transition vector models (e.g. silence)
grammar_node_id
speech models
Lexical tree(s)
monophone_id(s)
phone generator
triphone_id
transvec_id lookup
Triphone to state_id lookup
state_ids
trans_vec_id
phone num. gen.
Phone Book
next phone response
current phone_no
wordlet_node_id
word_end_flag
score
speech models
compressed acoustic models
mean & covar data
state output probabilities
MHD engine (distance calculator)
feature vectors
front end
frame count
from front end i/f
feature vector buffer for model adaptation
model adaptation engine
State Level
phone level
current LT_node_id
current wordlet_node_id
next_lexinput_id
grammar_node_id
grammar_id
prev_phone_no
Path Merge Unit
to path merge unit in PINE array
new wavefront phone record
prev_phone_no, phone_no, wordlet_node_id, LM factor
state_id(x3), trans_vec_id, word_end_flag, full_trans_flag
word history
Wave Front manager
wave front phone record RAM
wave front phone CAM
phone request arbiter
next phone request
A
B
C
D
E

Figure 2a

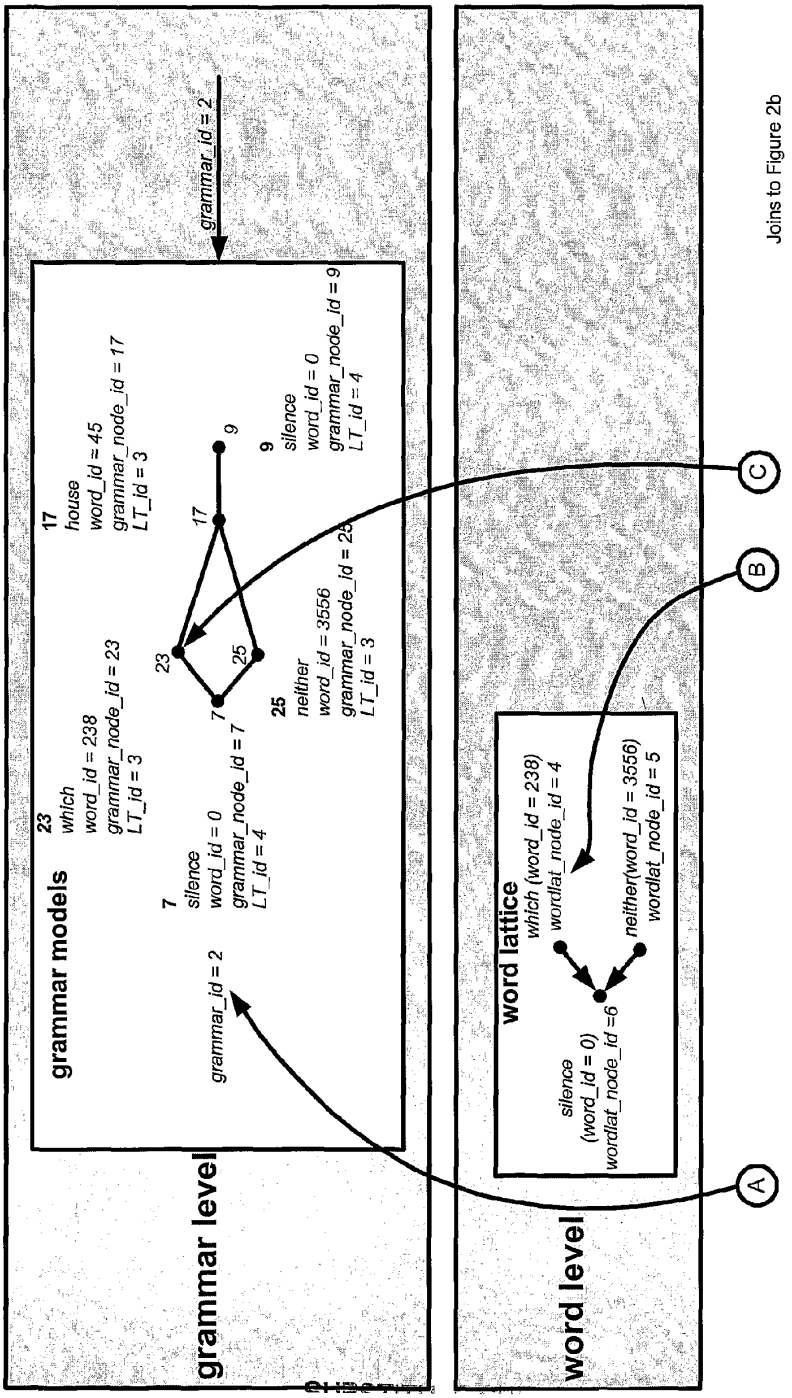

grammar level
word level
grammar models
grammar_id = 2
23
which
word_id = 238
grammar_node_id = 23
LT_id = 3
17
house
word_id = 45
grammar_node_id = 17
LT_id = 3
7
silence
word_id = 0
grammar_node_id = 7
LT_id = 4
25
neither
word_id = 3556
grammar_node_id = 25
LT_id = 3
9
silence
word_id = 0
grammar_node_id = 9
LT_id = 4
word lattice
which (word_id = 238)
wordlat_node_id = 4
silence
(word_id = 0)
wordlat_node_id =6
neither(word_id = 3556)
wordlat_node_id = 5
A
B
C
Joins to Figure 2b

Figure 2b

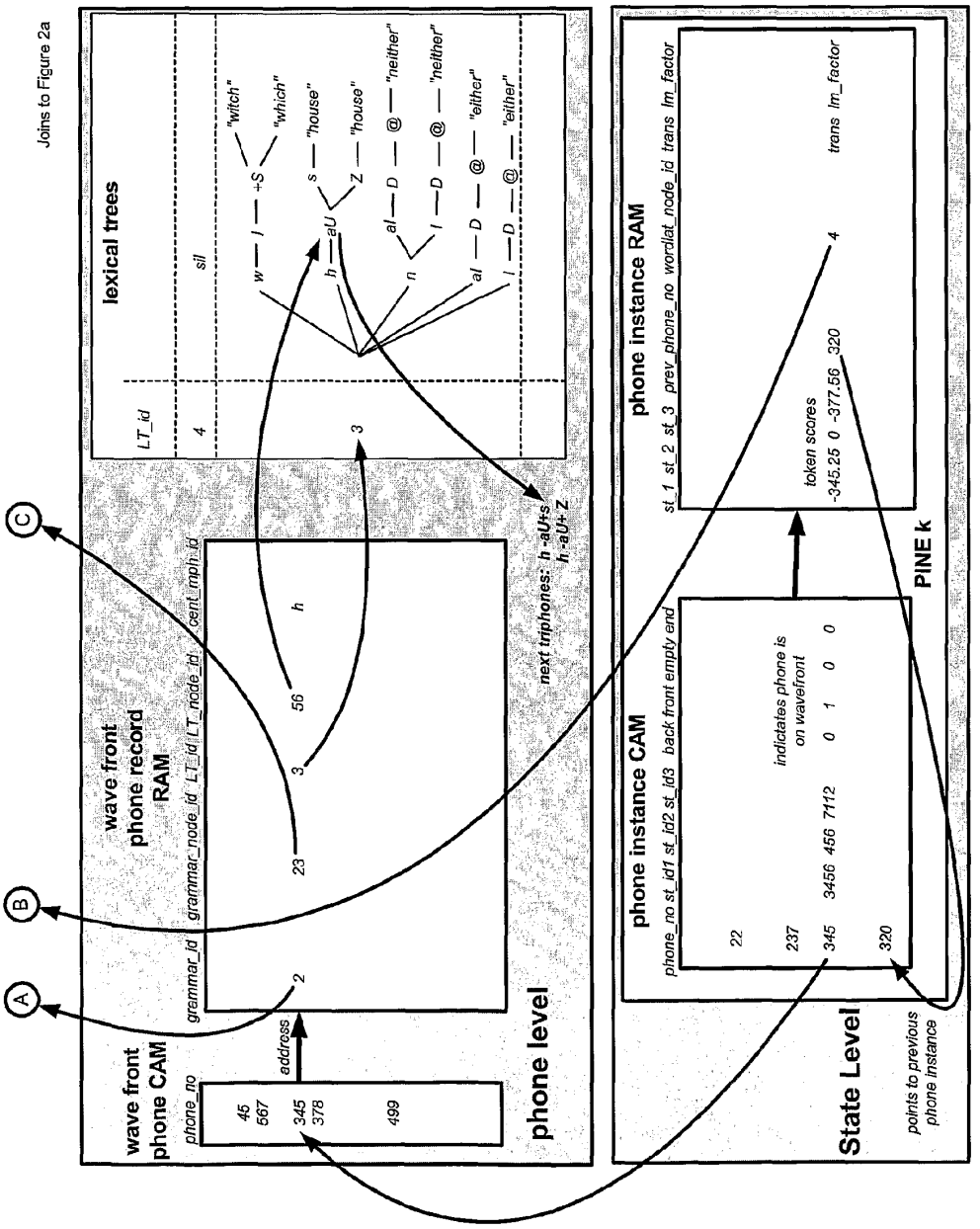
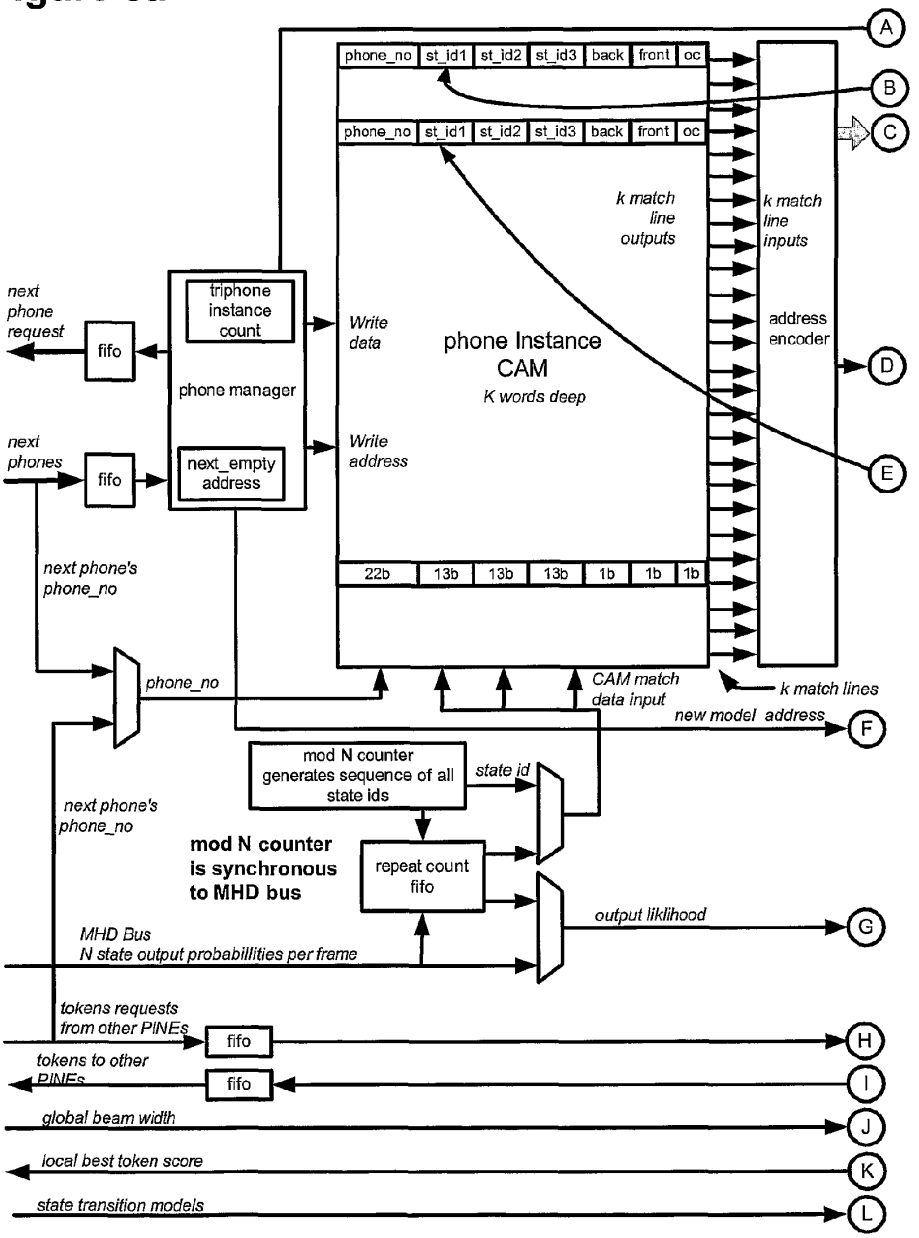

Joins to Figure 2a
lexical trees
LT_id
4
sil
"witch"
"which"
"house"
"house"
"neither"
"neither"
"either"
"either"
wave front phone record RAM
grammar_id
grammar_node_id
LT_id
LT_node_id
cent_mph_id
next triphones: h -aU+s h -aU+ Z
wave front phone CAM
phone_no
address
45 567
345 378
499
phone level
phone instance CAM
phone_no st_id1 st_id2 st_id3 back front empty end
22
237
345
320
3456 456 7112
indicates phone is on wavefront
0 1 0 0
phone instance RAM
st_1 st_2 st_3 prev_phone_no wordlat_node_id trans lm_factor
token scores
-345.25 0 -377.56 320
4
trans lm_factor
PINE k
State Level
points to previous phone instance
A
B
C

Figure 3a



CAM abbreviations

phone_no - unique phone instance indentifier
st_id1 - state_identifier1
back(wave_back_flag) - phone is at back of wave
front(wave_front_flag) - phone is at the front of wave
end(word_end_flag) - phone is the last in a word
oc - occupy - indicates that location has valid phone instance (it may not have any tokens)

RAM abbreviations

st_dum - dummy state to hold incoming token
st_1 - location for token in state 1, includes updated flag
st_2 - location for token in state 2, includes updated flag
st_3 - location for token in state 3, includes updated flag
tmp - location for temporary storage of token
prev_phone_no - pointer to previous phone instance
trans (trans_vec_id) - pointer to state transition vector in state transition RAM
LM_factor - language model factor
end - word end phone

Figure 3b

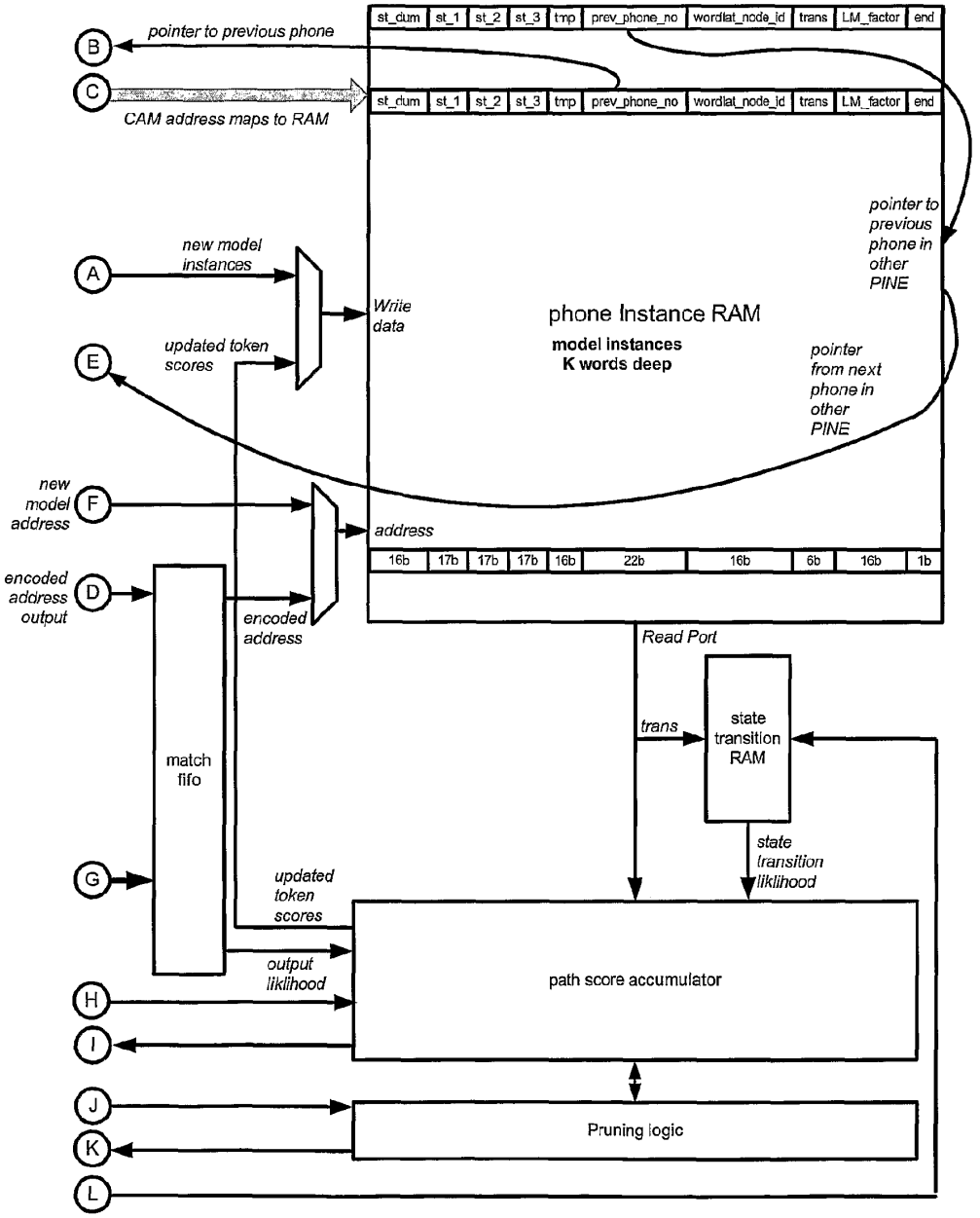

st_dum
st_1
st_2
st_3
tmp
prev_phone_no
wordlat_node_id
trans
LM_factor
end
pointer to previous phone
CAM address maps to RAM
new model instances
updated token scores
Write data
phone Instance RAM
model instances
K words deep
pointer to previous phone in other PINE
pointer from next phone in other PINE
new model address
address
16b
17b
17b
17b
16b
22b
16b
6b
16b
1b
encoded address output
encoded address
Read Port
trans
state transition RAM
state transition liklihood
match fifo
updated token scores
output liklihood
path score accumulator
Pruning logic
A
B
C
D
E
F
G
H
I
J
K
L

Figure 4

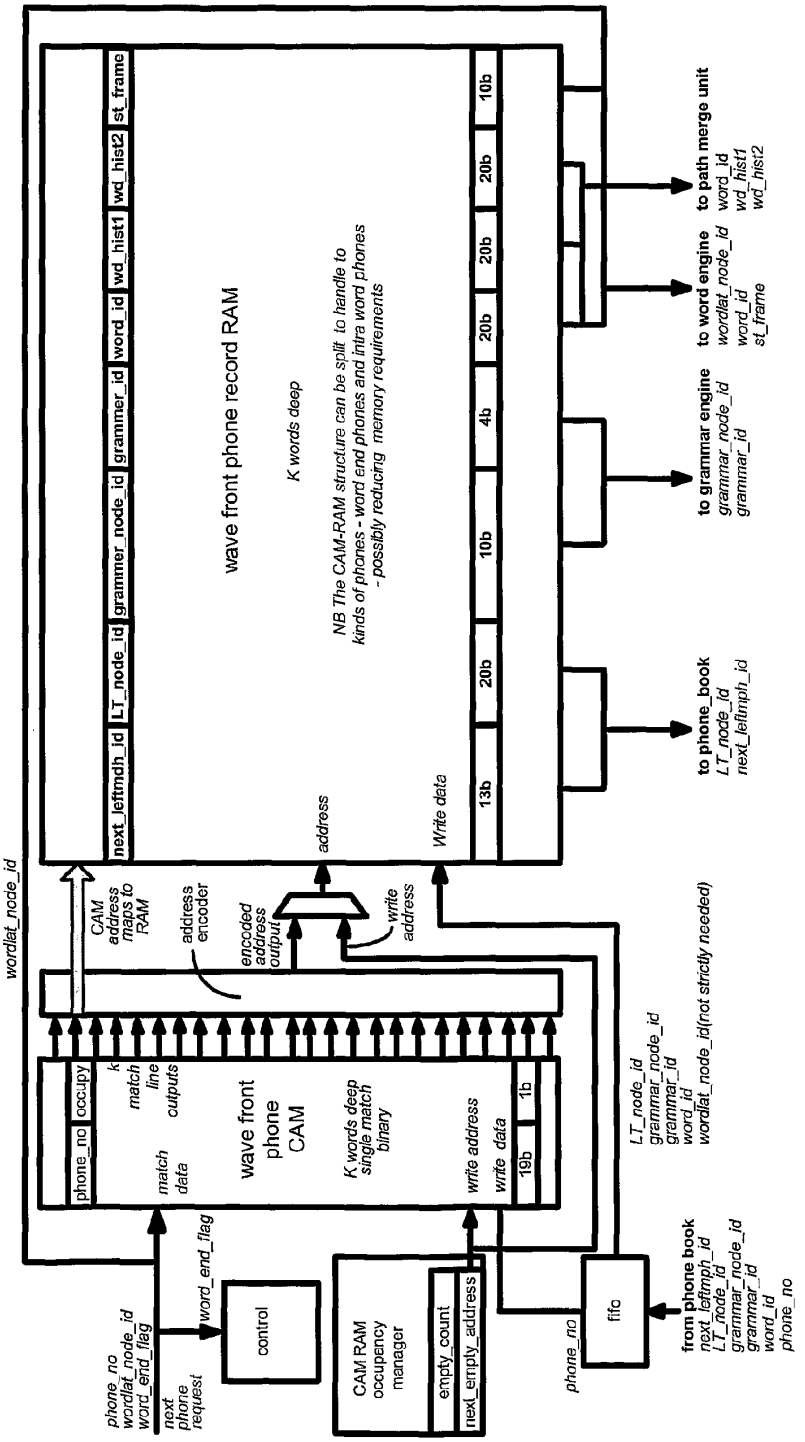

wave front phone record RAM
K words deep
NB The CAM-RAM structure can be split to handle to kinds of phones - word end phones and intra word phones - possibly reducing memory requirements
next_leftmph_id
LT_node_id
grammar_node_id
grammar_id
word_id
wd_hist1
wd_hist2
st_frame
13b
20b
10b
4b
20b
20b
20b
10b
address
Write data
to path merge unit
word_id
wd_hist1
wd_hist2
to word engine
wordlat_node_id
word_id
st_frame
to grammar engine
grammar_node_id
grammar_id
to phone_book
LT_node_id
next_leftmph_id
wordlat_node_id
CAM address maps to RAM
address encoder
encoded address output
write address
phone_no
occupy
k match line outputs
match data
wave front phone CAM
K words deep single match binary
write address
write data
19b
1b
LT_node_id
grammar_node_id
grammar_id
word_id
wordlat_node_id(not strictly needed)
phone_no
wordlat_node_id
word_end_flag
next phone request
control
CAM RAM occupancy manager
empty_count
next_empty_address
fifo
from phone book
next_leftmph_id
LT_node_id
grammar_node_id
grammar_id
word_id
phone_no

Figure 5

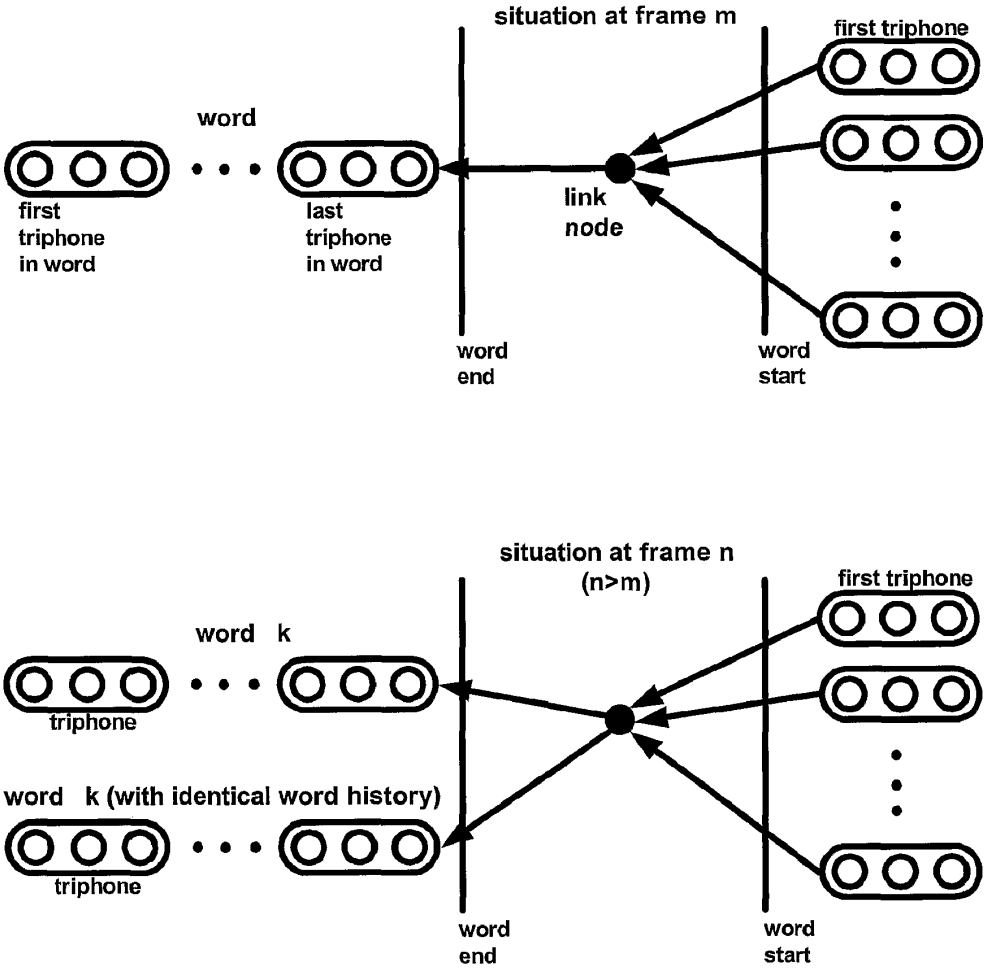

situation at frame m
first triphone
word
first triphone in word
last triphone in word
link node
word end
word start
situation at frame n (n>m)
first triphone
word k
triphone
word k (with identical word history)
triphone
word end
word start

Figure 6

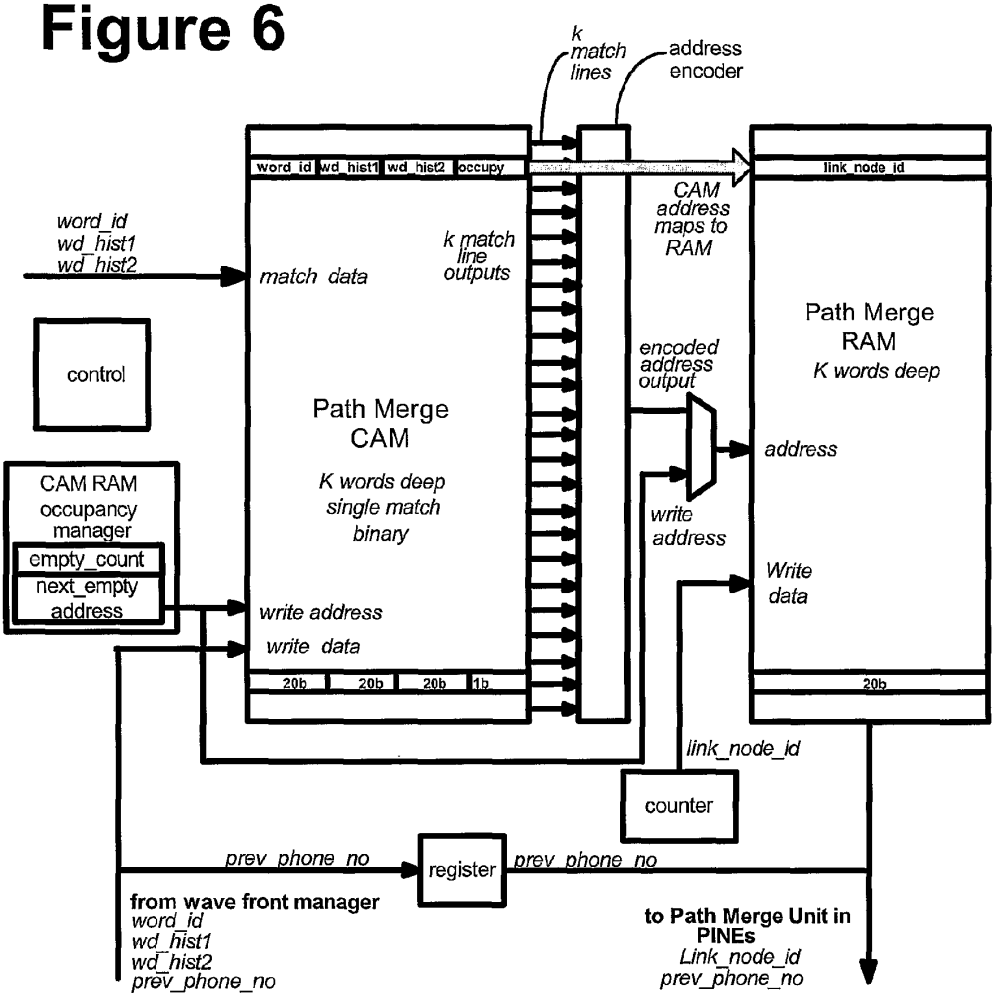

k match lines
address encoder
word_id
wd_hist1
wd_hist2
occupy
link_node_id
CAM address maps to RAM
word_id
wd_hist1
wd_hist2
match data
k match line outputs
Path Merge RAM
K words deep
control
encoded address output
Path Merge CAM
K words deep
single match
binary
address
CAM RAM
occupancy
manager
empty_count
next_empty address
write address
write data
Write data
write address
write data
20b
20b
20b
1b
20b
link_node_id
counter
prev_phone_no
register
prev_phone_no
from wave front manager
word_id
wd_hist1
wd_hist2
prev_phone_no
to Path Merge Unit in PINEs
Link_node_id
prev_phone_no

Figure 7

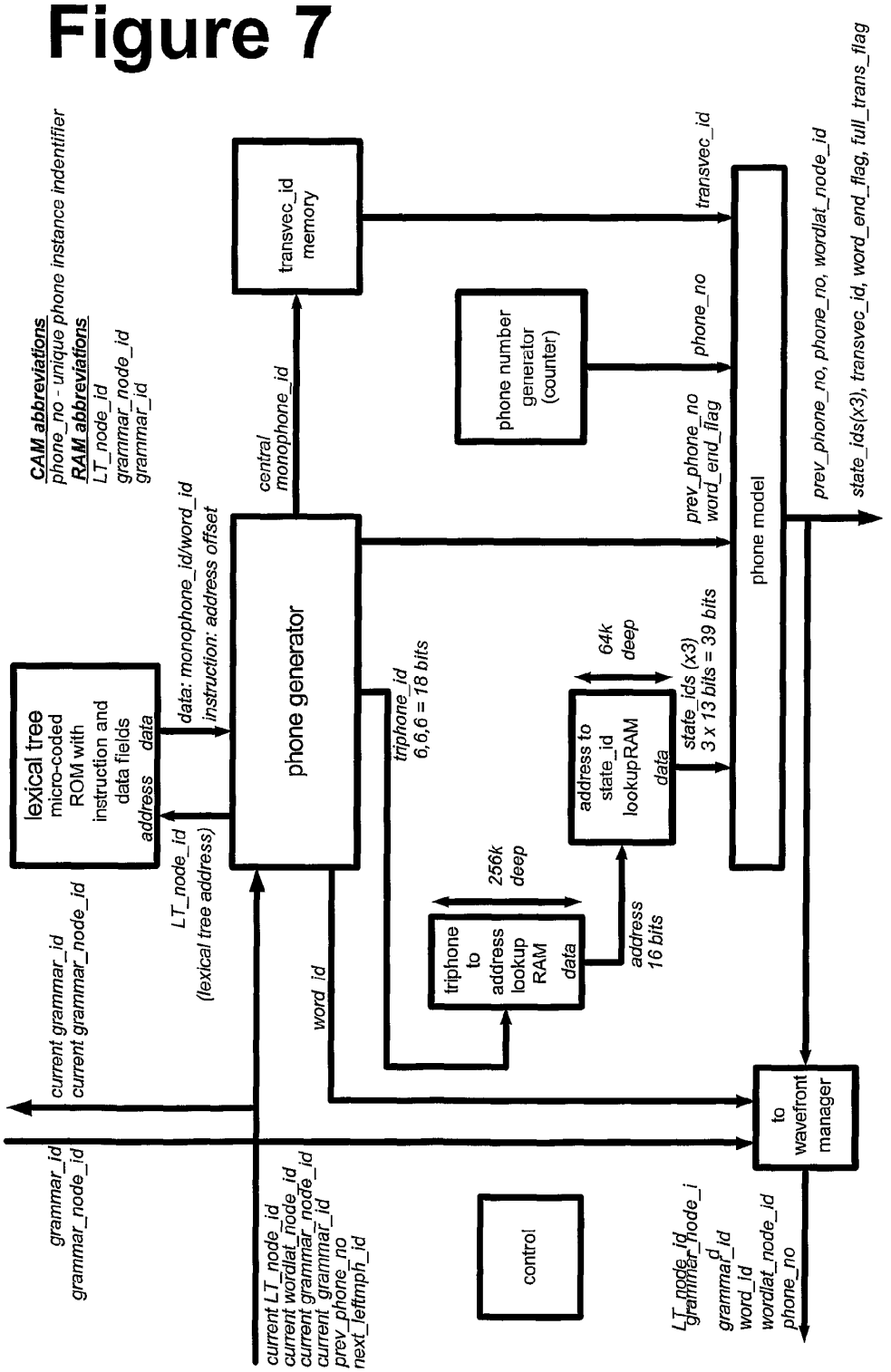

CAM abbreviations
phone_no - unique phone instance indentifier
RAM abbreviations
LT_node_id
grammar_node_id
grammar_id
transvec_id memory
phone number generator (counter)
phone model
phone generator
lexical tree micro-coded ROM with instruction and data fields
address data
data: monophone_id/word_id
instruction: address offset
LT_node_id (lexical tree address)
central monophone_id
prev_phone_no word_end_flag
transvec_id
phone_no
prev_phone_no, phone_no, wordlat_node_id
state_ids(x3), transvec_id, word_end_flag, full_trans_flag
triphone_id 6,6,6 = 18 bits
64k deep
state_ids (x3) 3 x 13 bits = 39 bits
address to state_id lookupRAM data
256k deep
triphone to address lookup RAM data
address 16 bits
word_id
current grammar_id
current grammar_node_id
grammar_id
grammar_node_id
to wavefront manager
control
current LT_node_id
current wordlat_node_id
current grammar_node_id
current grammar_id
prev_phone_no
next_leftmph_id
LT_node_id
grammar_id
word_id
wordlat_node_id
phone_no

Figure 8a

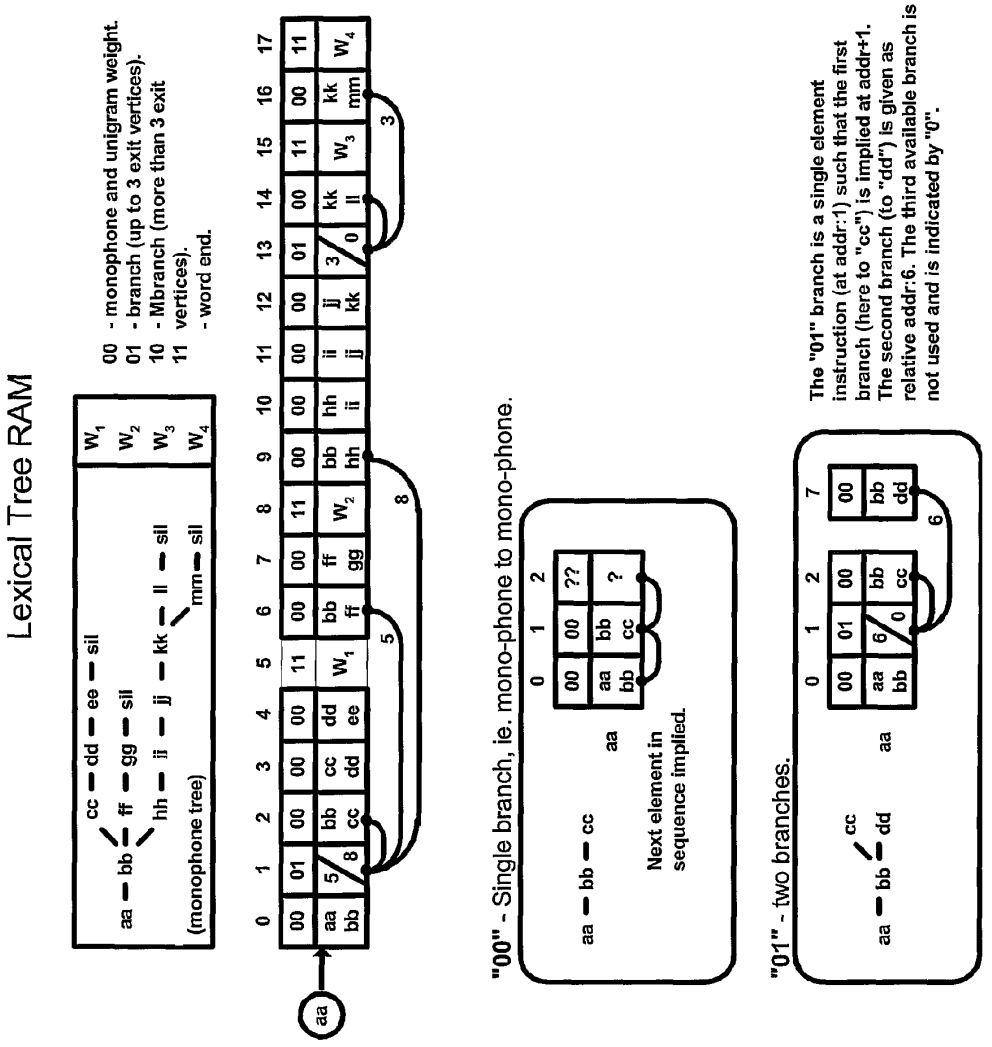

Lexical Tree RAM
(monophone tree)
00 - monophone and unigram weight.
01 - branch (up to 3 exit vertices).
10 - Mbranch (more than 3 exit vertices).
11 - word end.
"00" - Single branch, ie. mono-phone to mono-phone.
Next element in sequence implied.
"01" - two branches.
The "01" branch is a single element instruction (at addr:1) such that the first branch (here to "cc") is implied at addr+1. The second branch (to "dd") is given as relative addr:6. The third available branch is not used and is indicated by "0".

Figure 8b

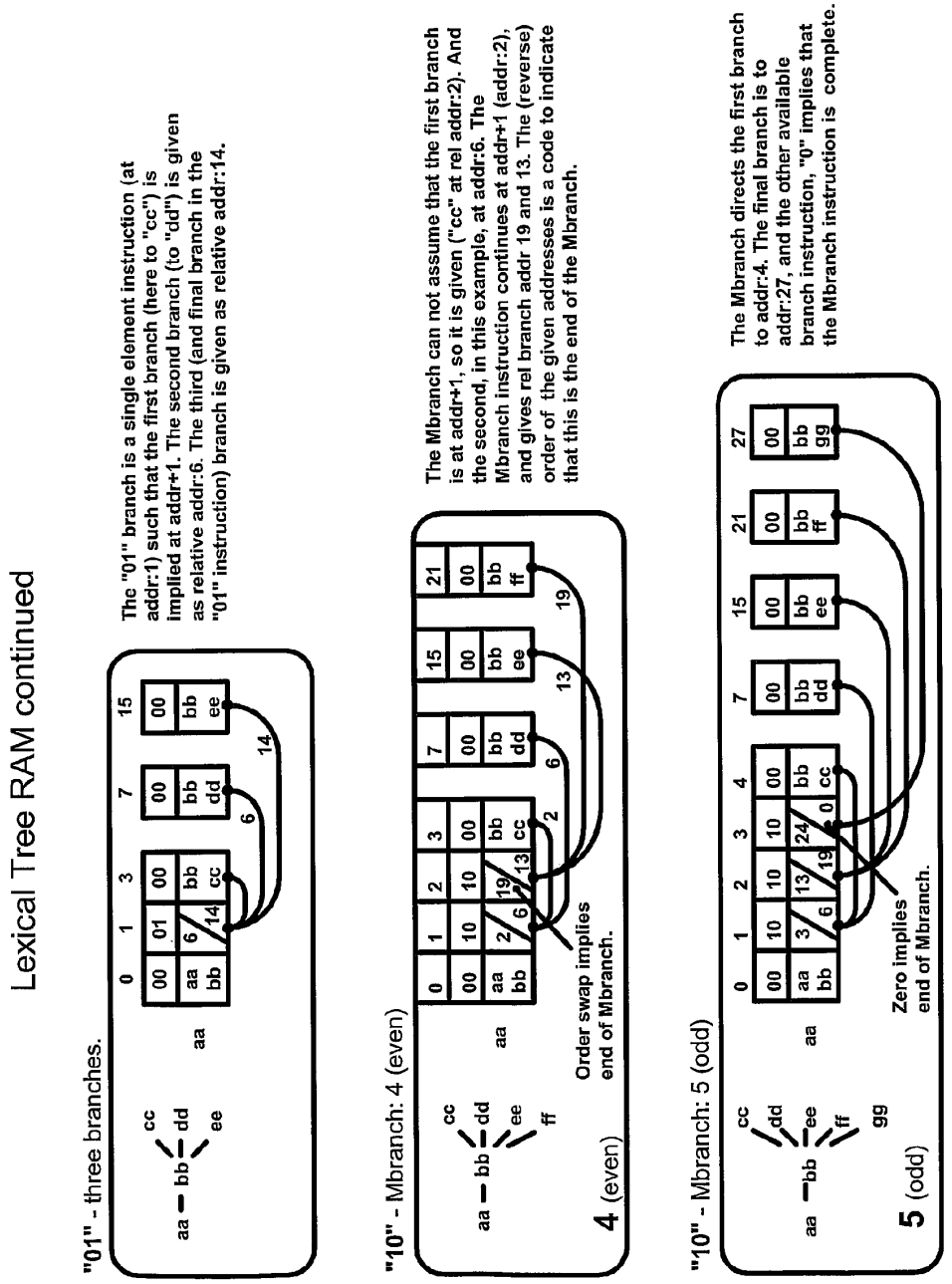
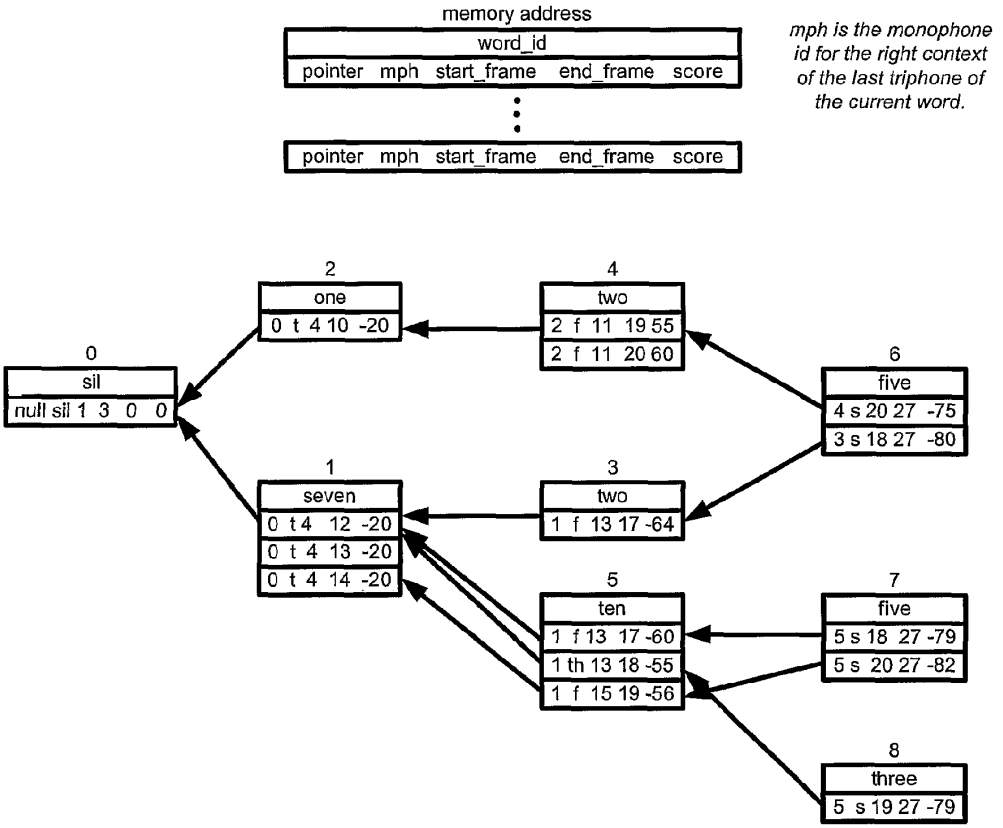

Lexical Tree RAM continued
"01" - three branches.
The "01" branch is a single element instruction (at addr:1) such that the first branch (here to "cc") is implied at addr+1. The second branch (to "dd") is given as relative addr:6. The third (and final branch in the "01" instruction) branch is given as relative addr:14.
"10" - Mbranch: 4 (even)
The Mbranch can not assume that the first branch is at addr+1, so it is given ("cc" at rel addr:2). And the second, in this example, at addr:6. The Mbranch instruction continues at addr+1 (addr:2), and gives rel branch addr 19 and 13. The (reverse) order of the given addresses is a code to indicate that this is the end of the Mbranch.
Order swap implies end of Mbranch.
4 (even)
"10" - Mbranch: 5 (odd)
The Mbranch directs the first branch to addr:4. The final branch is to addr:27, and the other available branch instruction, "0" implies that the Mbranch instruction is complete.
Zero implies end of Mbranch.
5 (odd)

Figure 9

data structure for word link record to enable word n-best decoding



Notes

---

1. each record represents the passage of a single token through a word with a particular right monophone context.
2. each word's acoustic model with a particular right context monophone can only generate one token per frame. Thus frame number can be used to distinguish trace back paths with the same end_frame numbers but different right context monophones
3. the end_frame doesn't need to be passed to the PINEs because it is simply the current frame number
4. scores may be absolute or relative
5. the address is referred to as the wordlat_node_id elsewhere

Traceback

---

for the n-best tokens

1. using the associated wordlat_node_ids look up each word link record
2. look at associated records and use the end_frame number and monophone id to select associated pointer and score.
3. trace back recursively and select n-best paths

Figure 10

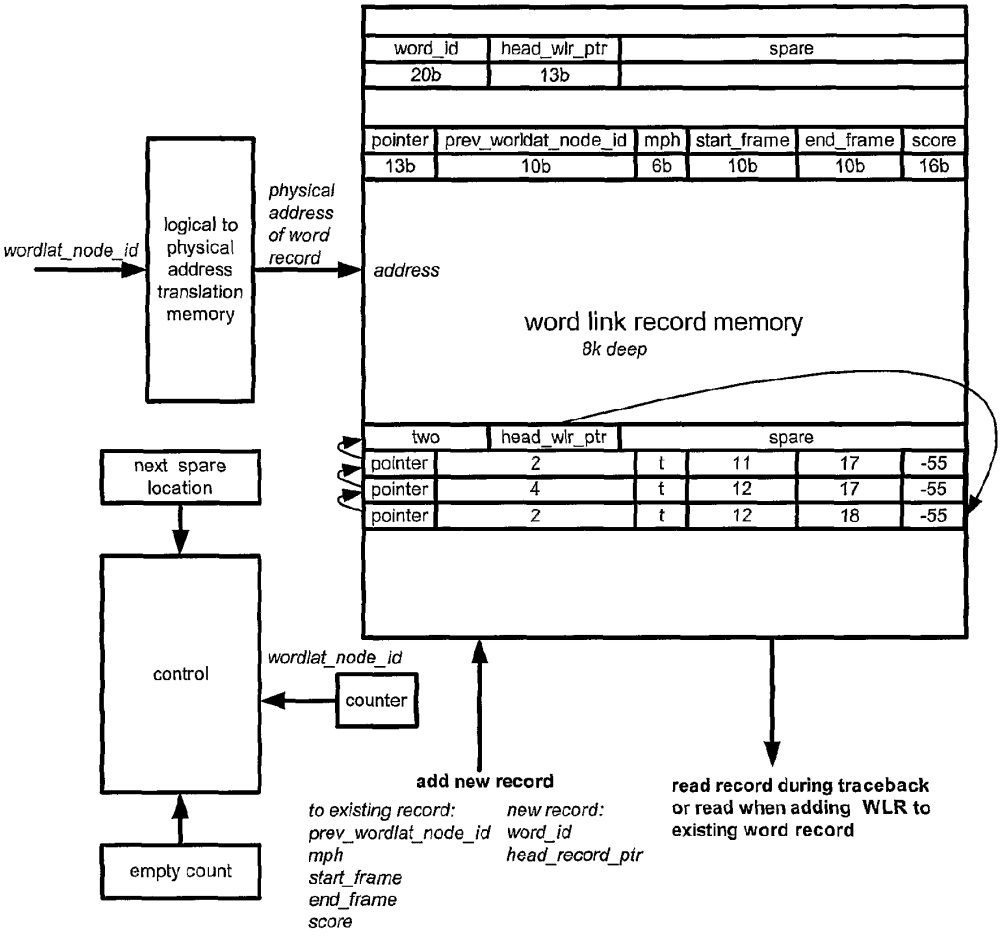

word_id
head_wlr_ptr
spare
20b
13b
pointer
prev_worldat_node_id
mph
start_frame
end_frame
score
13b
10b
6b
10b
10b
16b
wordlat_node_id
logical to physical address translation memory
physical address of word record
address
word link record memory
8k deep
two
head_wlr_ptr
spare
pointer
2
t
11
17
-55
pointer
4
t
12
17
-55
pointer
2
t
12
18
-55
next spare location
control
wordlat_node_id
counter
empty count
add new record
to existing record:
prev_wordlat_node_id
mph
start_frame
end_frame
score
new record:
word_id
head_record_ptr
read record during traceback or read when adding WLR to existing word record

Figure 11

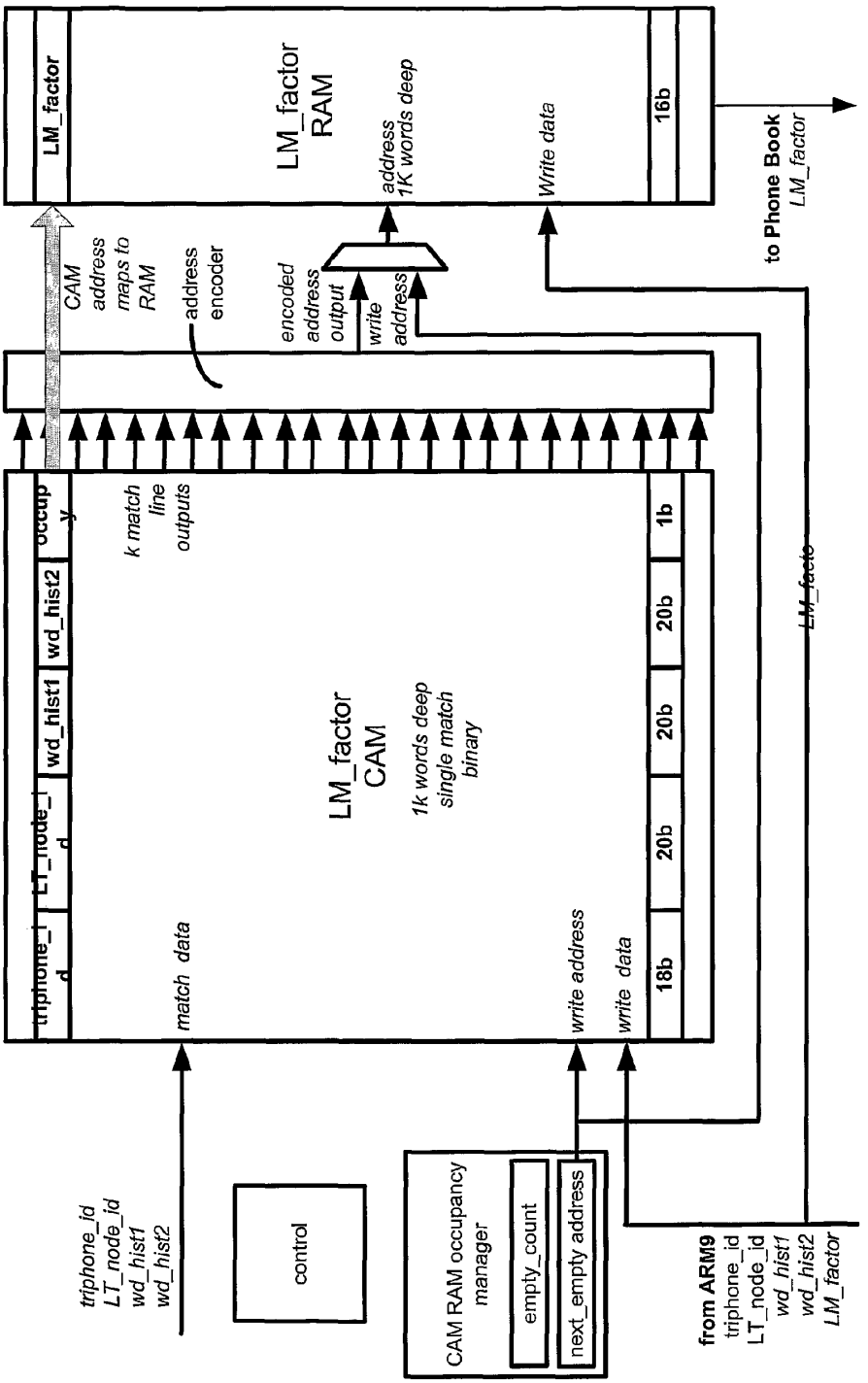

LM_factor
LM_factor RAM
address 1K words deep
Write data
16b
to Phone Book LM_factor
CAM address maps to RAM
address encoder
encoded address output
write address
triphone_id
LT_node_id
wd_hist1
wd_hist2
occupy
k match line outputs
LM_factor CAM
1k words deep single match binary
1b
20b
20b
20b
18b
match data
write address
write data
LM_factor
triphone_id
LT_node_id
wd_hist1
wd_hist2
control
CAM RAM occupancy manager
empty_count
next_empty address
from ARM9
triphone_id
LT_node_id
wd_hist1
wd_hist2
LM_factor

Figure 12

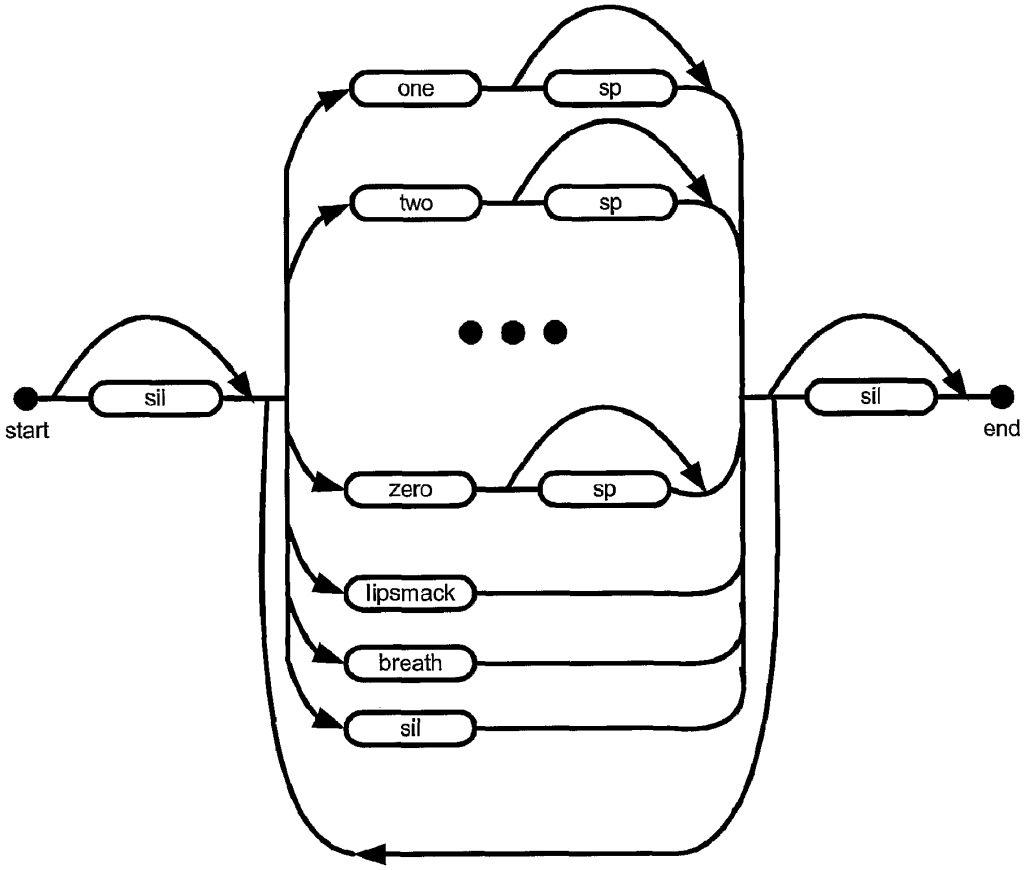


Figure 13

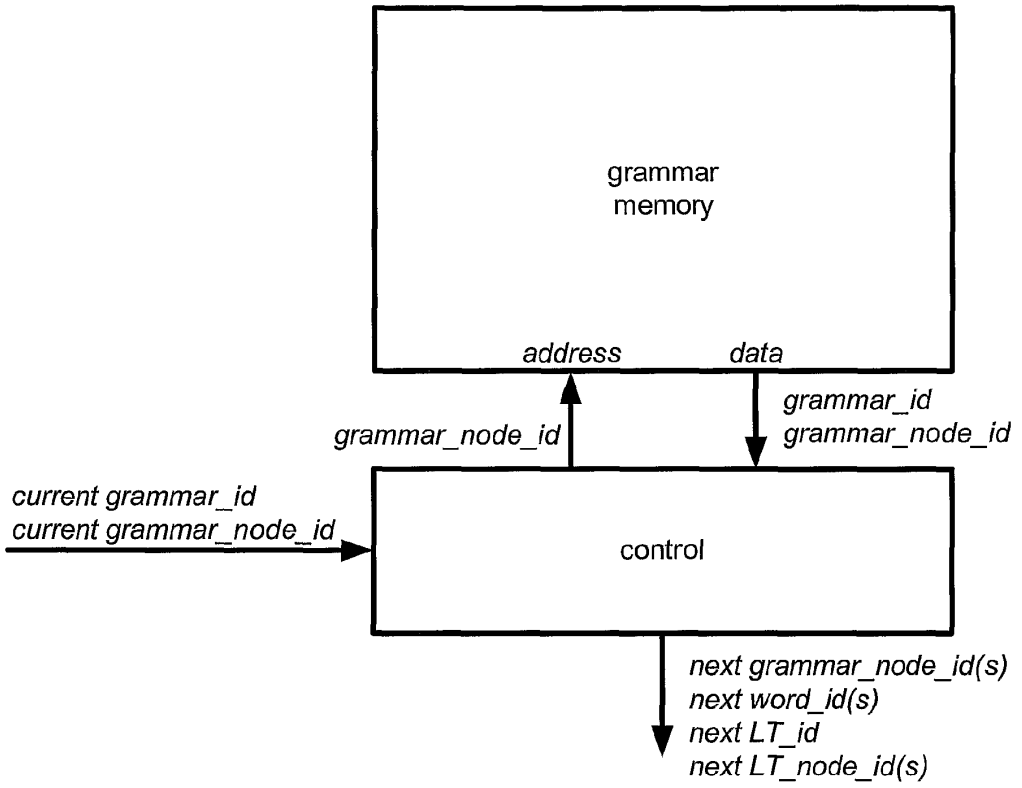

grammar
memory
address
data
grammar_id
grammar_node_id
grammar_node_id
current grammar_id
current grammar_node_id
control
next grammar_node_id(s)
next word_id(s)
next LT_id
next LT_node_id(s)

Figure 14

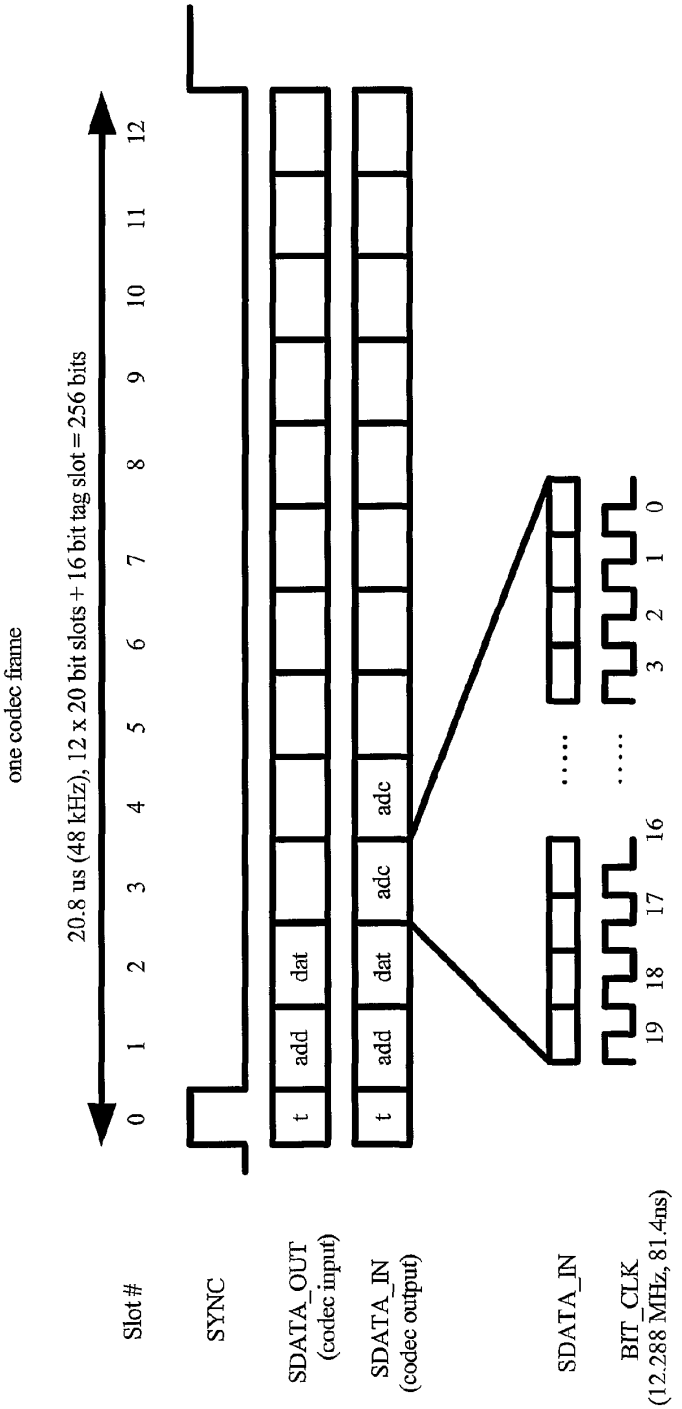

one codec frame
20.8 us (48 kHz), 12 x 20 bit slots + 16 bit tag slot = 256 bits
0 1 2 3 4 5 6 7 8 9 10 11 12
Slot #
SYNC
SDATA_OUT (codec input)
SDATA_IN (codec output)
SDATA_IN
BIT_CLK (12.288 MHz, 81.4ns)
t
add
dat
adc
0 1 2 3 16 17 18 19

Figure 15

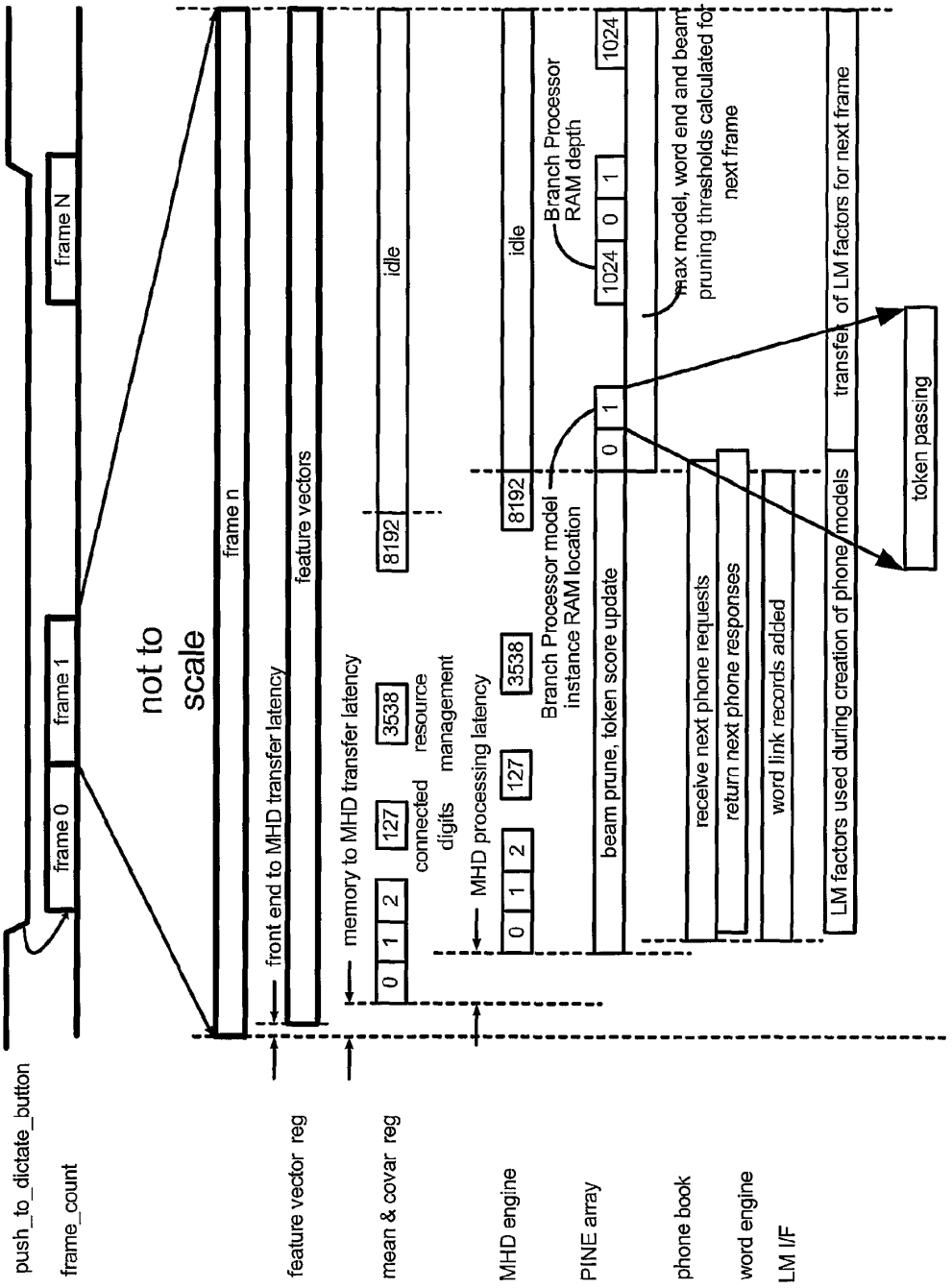

push_to_dictate_button
frame_count
feature vector reg
mean & covar reg
MHD engine
PINE array
phone book
word engine
LM I/F
frame 0
frame 1
frame N
not to scale
frame n
front end to MHD transfer latency
feature vectors
memory to MHD transfer latency
connected digits
resource management
3538
127
8192
idle
MHD processing latency
Branch Processor model instance RAM location
Branch Processor RAM depth
1024
beam prune, token score update
receive next phone requests
return next phone responses
word link records added
max model, word end and beam pruning thresholds calculated for next frame
LM factors used during creation of phone models
transfer of LM factors for next frame
token passing

Figure 16

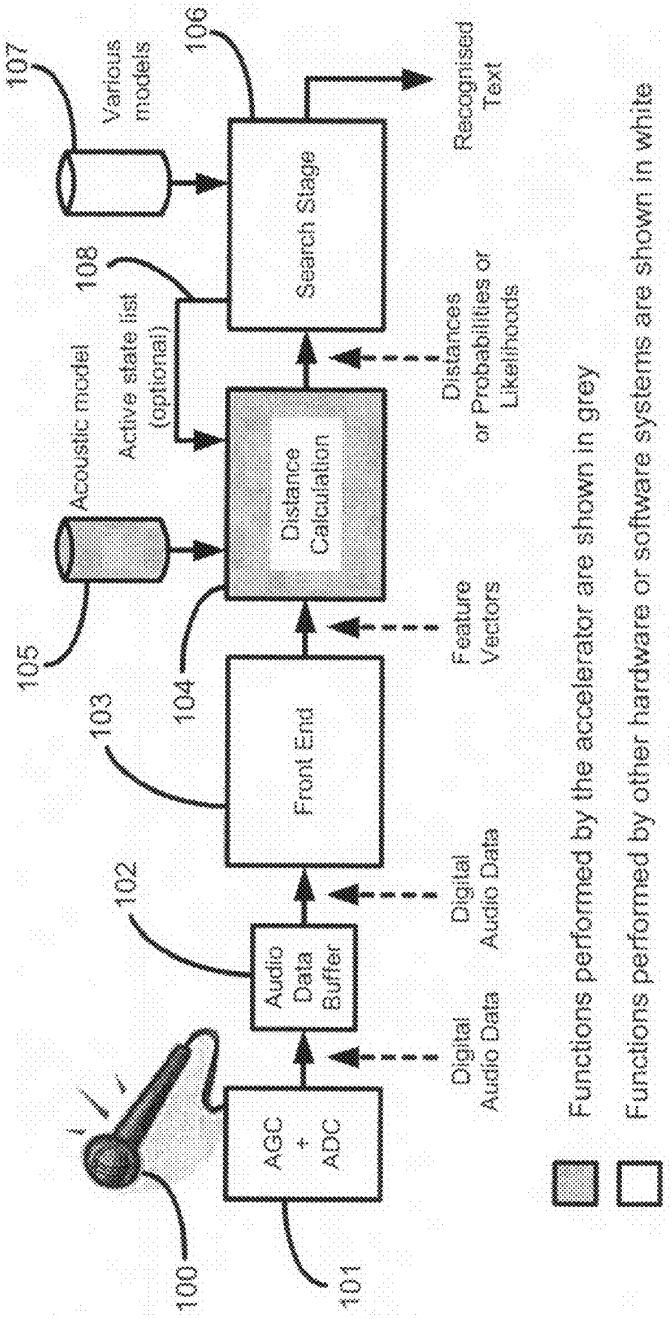

100
101
AGC + ADC
Digital Audio Data
102
Audio Data Buffer
Digital Audio Data
103
Front End
104
Feature Vectors
105
Acoustic model
Distance Calculation
108
Active state list (optional)
Distances or Probabilities or Likelihoods
107
Various models
106
Search Stage
Recognised Text
Functions performed by the accelerator are shown in grey
Functions performed by other hardware or software systems are shown in white

FIGURE 17

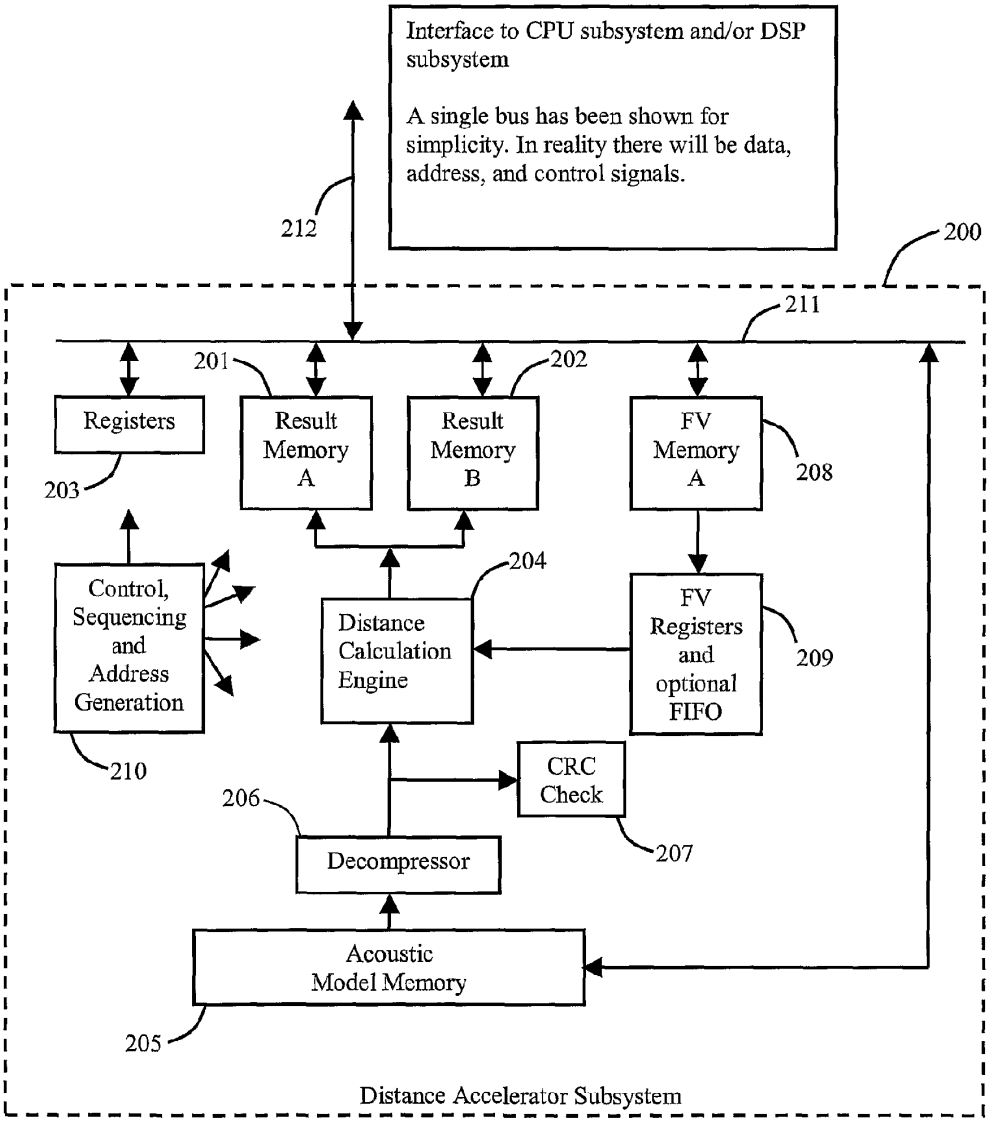

Interface to CPU subsystem and/or DSP subsystem
A single bus has been shown for simplicity. In reality there will be data, address, and control signals.
212
200
211
201
202
Registers
203
Result Memory A
Result Memory B
FV Memory A
208
Control, Sequencing and Address Generation
210
204
Distance Calculation Engine
FV Registers and optional FIFO
209
CRC Check
207
206
Decompressor
Acoustic Model Memory
205
Distance Accelerator Subsystem

Figure 18

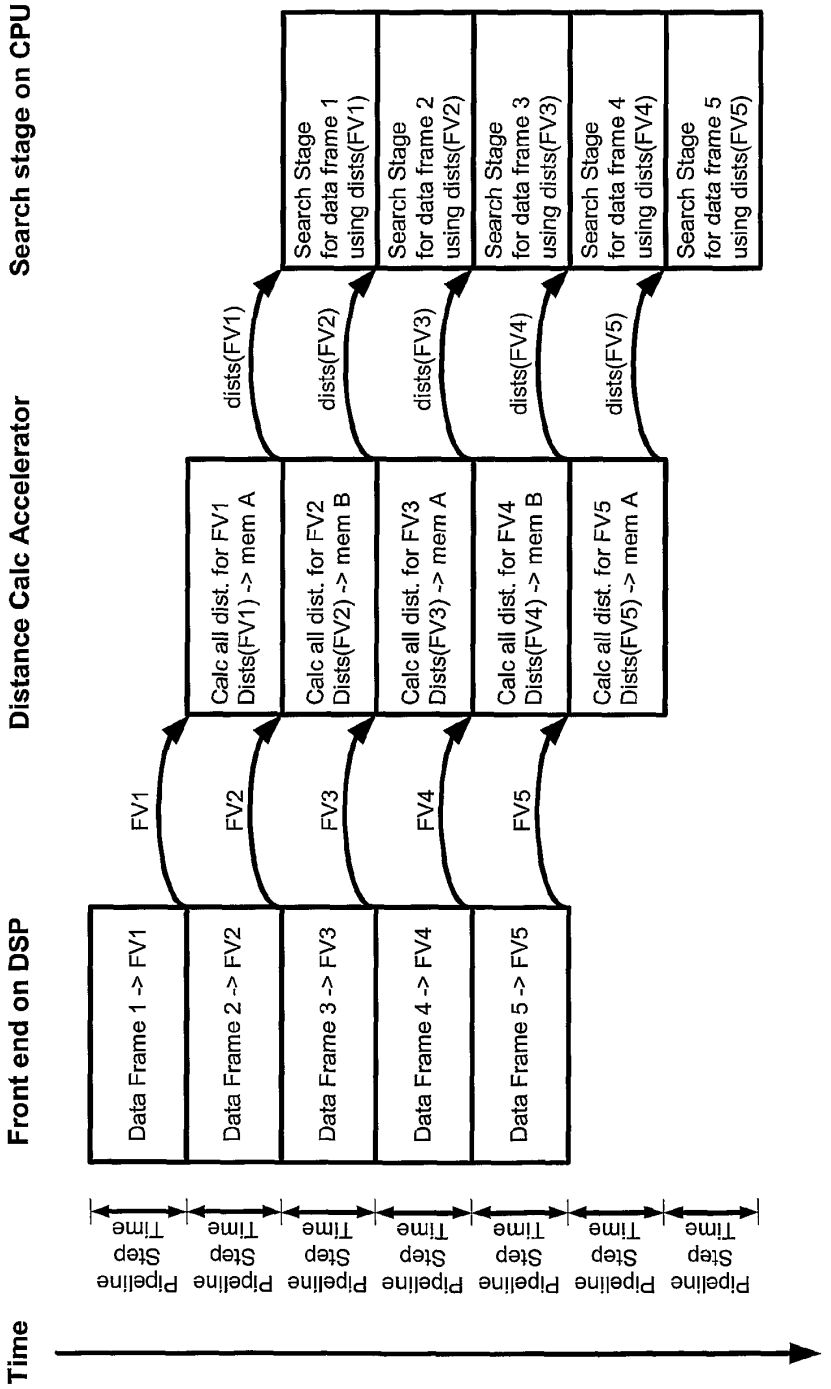

Search stage on CPU
Distance Calc Accelerator
Front end on DSP
Time
Data Frame 1 -> FV1
Data Frame 2 -> FV2
Data Frame 3 -> FV3
Data Frame 4 -> FV4
Data Frame 5 -> FV5
FV1
FV2
FV3
FV4
FV5
Calc all dist. for FV1 Dists(FV1) -> mem A
Calc all dist. for FV2 Dists(FV2) -> mem B
Calc all dist. for FV3 Dists(FV3) -> mem A
Calc all dist. for FV4 Dists(FV4) -> mem B
Calc all dist. for FV5 Dists(FV5) -> mem A
dists(FV1)
dists(FV2)
dists(FV3)
dists(FV4)
dists(FV5)
Search Stage for data frame 1 using dists(FV1)
Search Stage for data frame 2 using dists(FV2)
Search Stage for data frame 3 using dists(FV3)
Search Stage for data frame 4 using dists(FV4)
Search Stage for data frame 5 using dists(FV5)
Pipeline Step Time

Figure 19

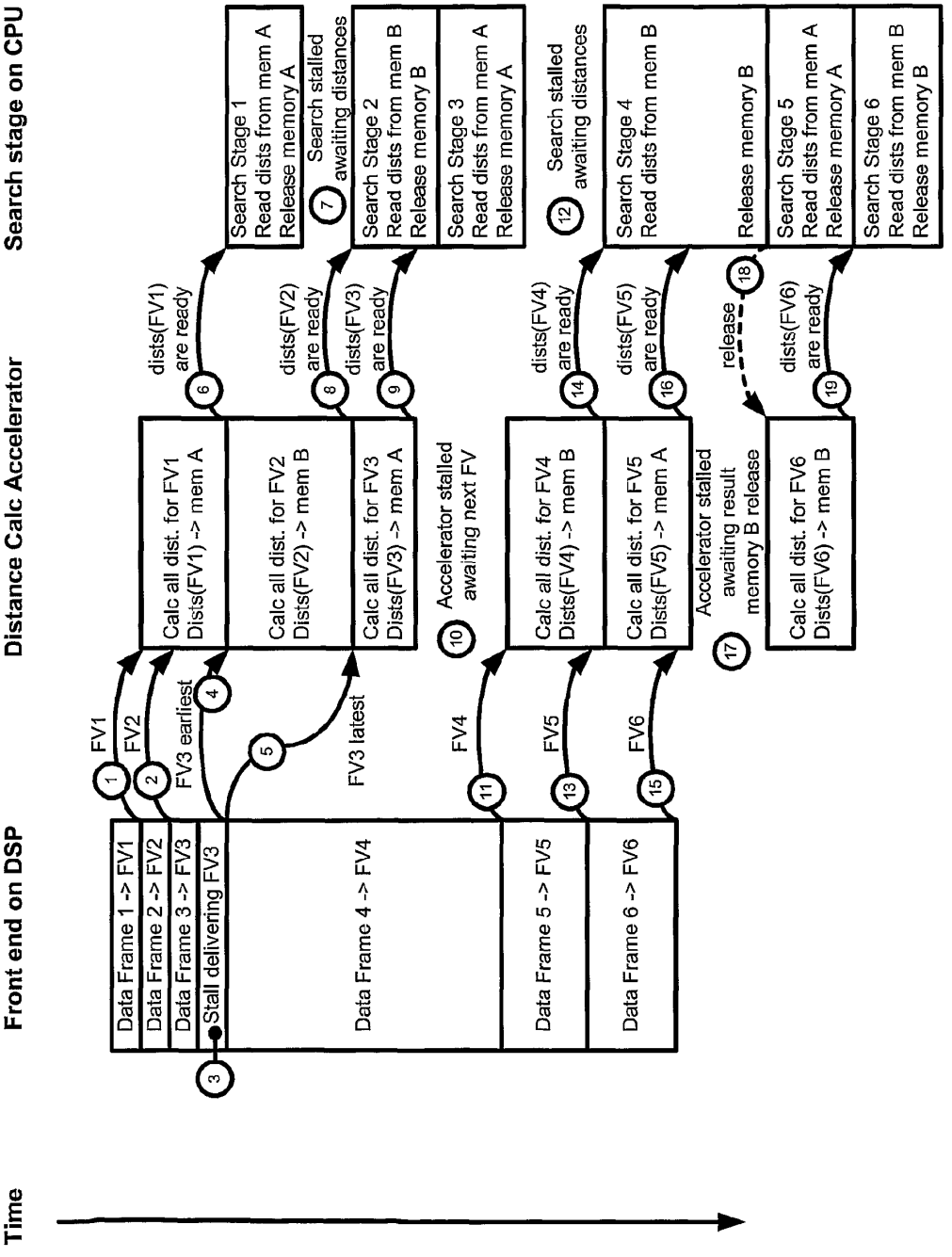

Time
Front end on DSP
Distance Calc Accelerator
Search stage on CPU
Data Frame 1 -> FV1
Data Frame 2 -> FV2
Data Frame 3 -> FV3
Stall delivering FV3
Data Frame 4 -> FV4
Data Frame 5 -> FV5
Data Frame 6 -> FV6
FV1
FV2
FV3 earliest
FV3 latest
FV4
FV5
FV6
Calc all dist. for FV1 Dists(FV1) -> mem A
Calc all dist. for FV2 Dists(FV2) -> mem B
Calc all dist. for FV3 Dists(FV3) -> mem A
Calc all dist. for FV4 Dists(FV4) -> mem B
Calc all dist. for FV5 Dists(FV5) -> mem A
Calc all dist. for FV6 Dists(FV6) -> mem B
Accelerator stalled awaiting next FV
Accelerator stalled awaiting result memory B release
dists(FV1) are ready
dists(FV2) are ready
dists(FV3) are ready
dists(FV4) are ready
dists(FV5) are ready
dists(FV6) are ready
release
Search Stage 1 Read dists from mem A Release memory A
Search stalled awaiting distances
Search Stage 2 Read dists from mem B Release memory B
Search Stage 3 Read dists from mem A Release memory A
Search stalled awaiting distances
Search Stage 4 Read dists from mem B
Release memory B
Search Stage 5 Read dists from mem A Release memory A
Search Stage 6 Read dists from mem B Release memory B
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

Figure 20

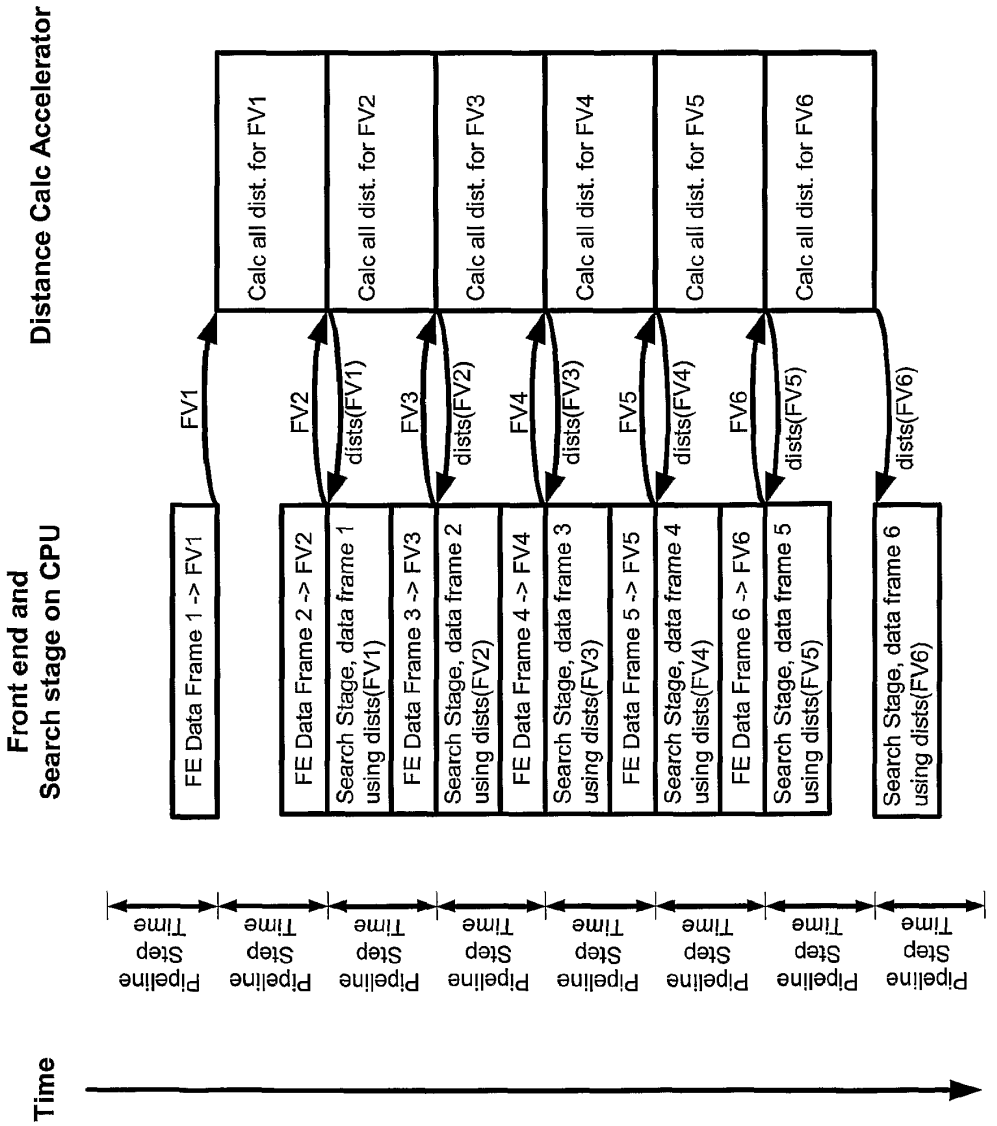
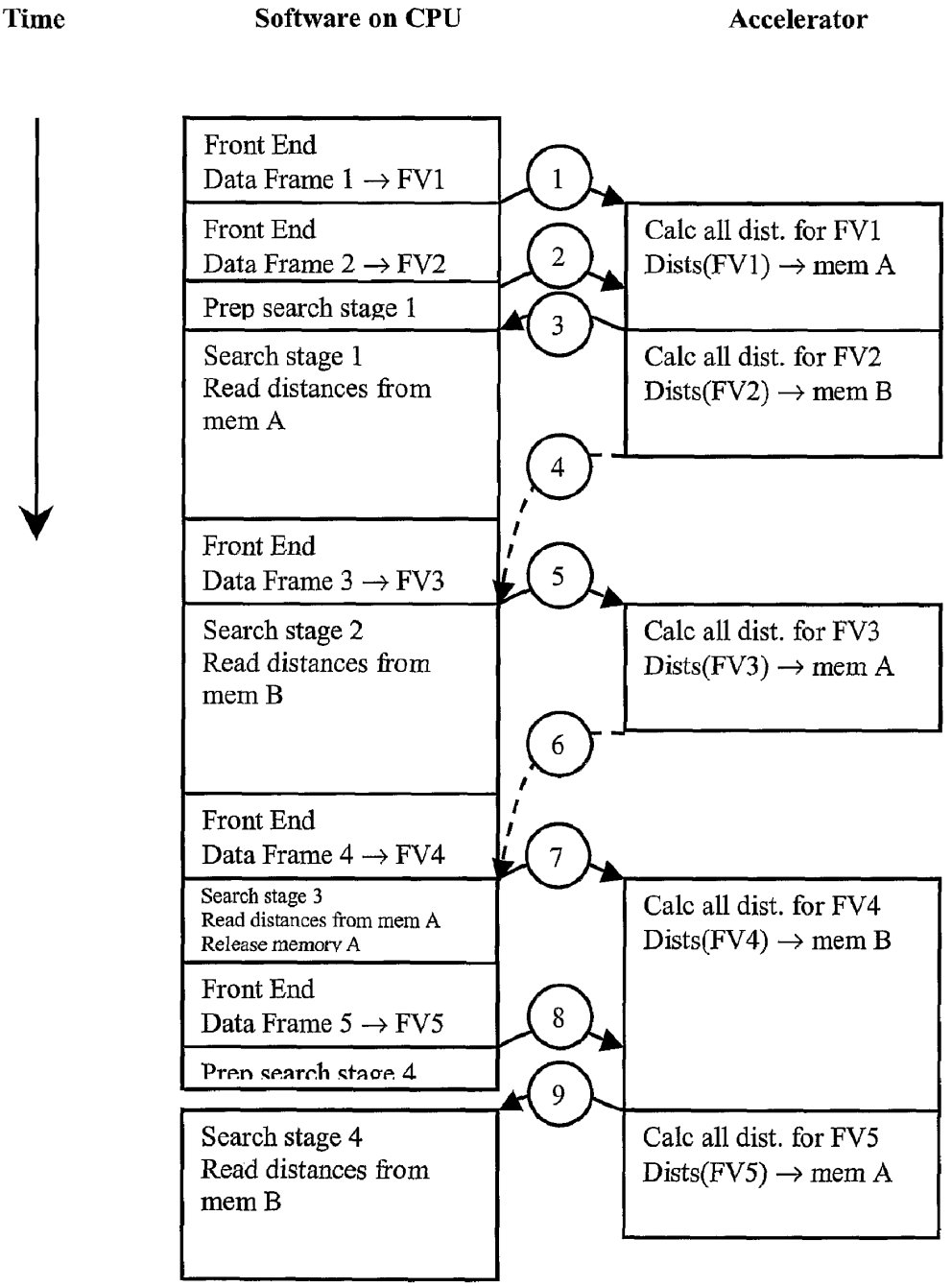

Distance Calc Accelerator
Front end and Search stage on CPU
Time
Pipeline Step Time
FE Data Frame 1 -> FV1
FE Data Frame 2 -> FV2
FE Data Frame 3 -> FV3
FE Data Frame 4 -> FV4
FE Data Frame 5 -> FV5
FE Data Frame 6 -> FV6
Search Stage, data frame 1 using dists(FV1)
Search Stage, data frame 2 using dists(FV2)
Search Stage, data frame 3 using dists(FV3)
Search Stage, data frame 4 using dists(FV4)
Search Stage, data frame 5 using dists(FV5)
Search Stage, data frame 6 using dists(FV6)
Calc all dist. for FV1
Calc all dist. for FV2
Calc all dist. for FV3
Calc all dist. for FV4
Calc all dist. for FV5
Calc all dist. for FV6
FV1
FV2
FV3
FV4
FV5
FV6
dists(FV1)
dists(FV2)
dists(FV3)
dists(FV4)
dists(FV5)
dists(FV6)

FIGURE 21


Time
Software on CPU
Accelerator
Front End
Data Frame 1 → FV1
Front End
Data Frame 2 → FV2
Prep search stage 1
Search stage 1
Read distances from
mem A
Front End
Data Frame 3 → FV3
Search stage 2
Read distances from
mem B
Front End
Data Frame 4 → FV4
Search stage 3
Read distances from mem A
Release memory A
Front End
Data Frame 5 → FV5
Prep search stage 4
Search stage 4
Read distances from
mem B
1
2
3
4
5
6
7
8
9
Calc all dist. for FV1
Dists(FV1) → mem A
Calc all dist. for FV2
Dists(FV2) → mem B
Calc all dist. for FV3
Dists(FV3) → mem A
Calc all dist. for FV4
Dists(FV4) → mem B
Calc all dist. for FV5
Dists(FV5) → mem A

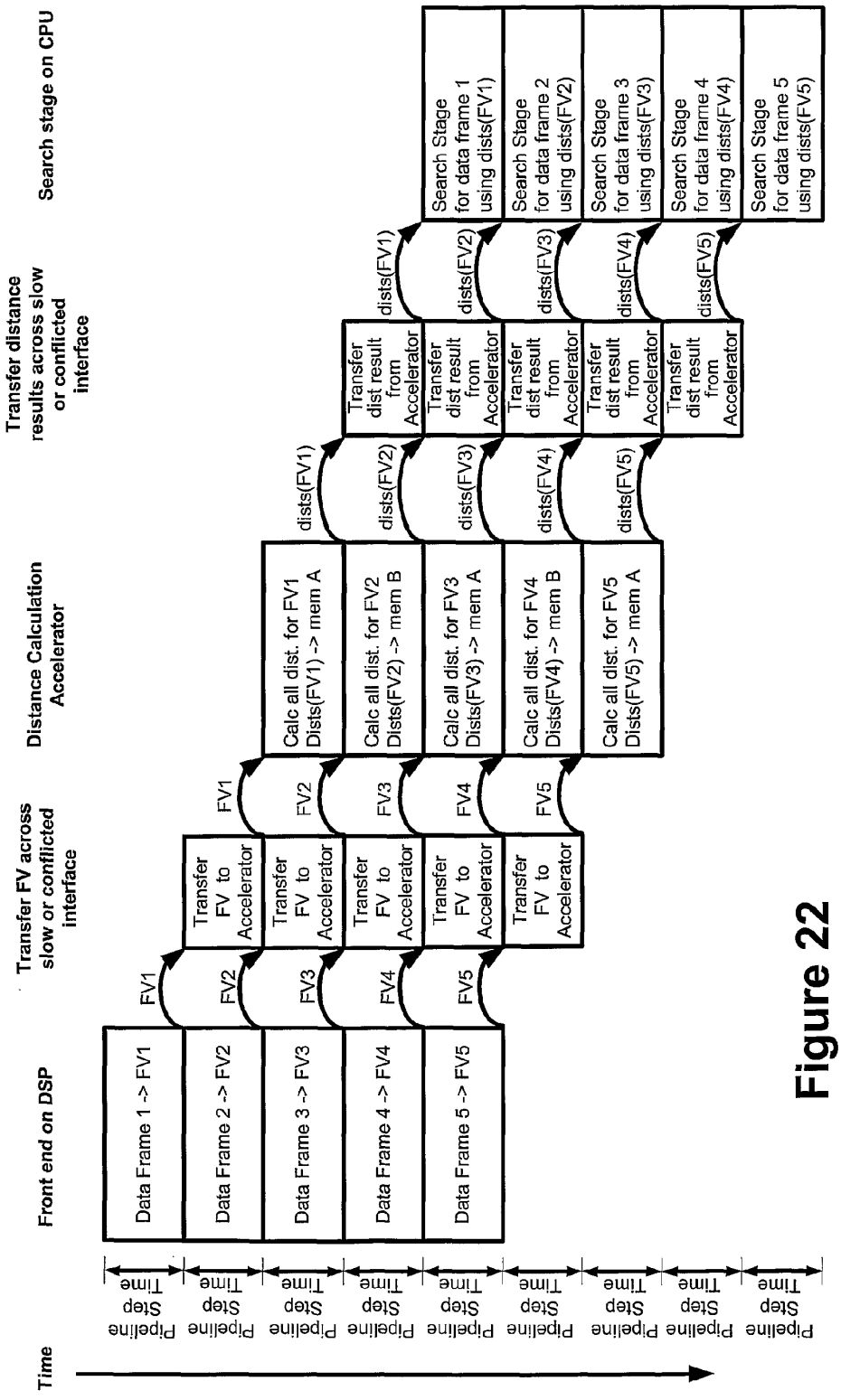
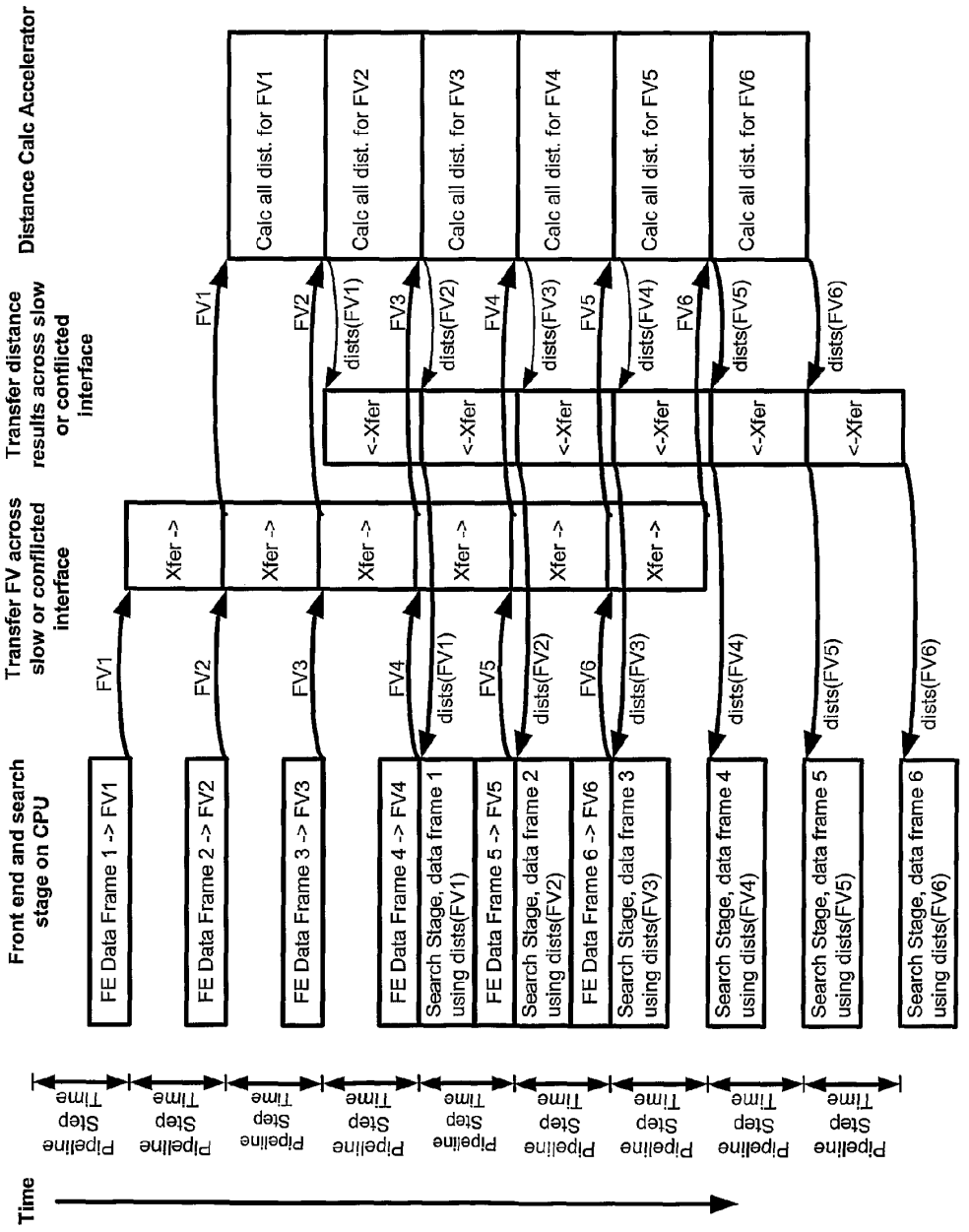

Time
Pipeline Step Time
Front end on DSP
Data Frame 1 -> FV1
Data Frame 2 -> FV2
Data Frame 3 -> FV3
Data Frame 4 -> FV4
Data Frame 5 -> FV5
Transfer FV across slow or conflicted interface
FV1
FV2
FV3
FV4
FV5
Transfer FV to Accelerator
Distance Calculation Accelerator
Calc all dist. for FV1 Dists(FV1) -> mem A
Calc all dist. for FV2 Dists(FV2) -> mem B
Calc all dist. for FV3 Dists(FV3) -> mem A
Calc all dist. for FV4 Dists(FV4) -> mem B
Calc all dist. for FV5 Dists(FV5) -> mem A
dists(FV1)
dists(FV2)
dists(FV3)
dists(FV4)
dists(FV5)
Transfer distance results across slow or conflicted interface
Transfer dist result from Accelerator
Search stage on CPU
Search Stage for data frame 1 using dists(FV1)
Search Stage for data frame 2 using dists(FV2)
Search Stage for data frame 3 using dists(FV3)
Search Stage for data frame 4 using dists(FV4)
Search Stage for data frame 5 using dists(FV5)

Figure 22

Figure 23


Front end and search stage on CPU
Transfer FV across slow or conflicted interface
Transfer distance results across slow or conflicted interface
Distance Calc Accelerator
FE Data Frame 1 -> FV1
FE Data Frame 2 -> FV2
FE Data Frame 3 -> FV3
FE Data Frame 4 -> FV4
FE Data Frame 5 -> FV5
FE Data Frame 6 -> FV6
Search Stage, data frame 1 using dists(FV1)
Search Stage, data frame 2 using dists(FV2)
Search Stage, data frame 3 using dists(FV3)
Search Stage, data frame 4 using dists(FV4)
Search Stage, data frame 5 using dists(FV5)
Search Stage, data frame 6 using dists(FV6)
Xfer ->
<-Xfer
Calc all dist. for FV1
Calc all dist. for FV2
Calc all dist. for FV3
Calc all dist. for FV4
Calc all dist. for FV5
Calc all dist. for FV6
FV1
FV2
FV3
FV4
FV5
FV6
dists(FV1)
dists(FV2)
dists(FV3)
dists(FV4)
dists(FV5)
dists(FV6)
Pipeline Step Time
Time

SPEECH RECOGNITION CIRCUIT AND METHOD

The present invention relates to speech recognition circuits and methods. These circuits and methods have wide applicability, particularly for devices such as mobile electronic devices.

There is growing consumer demand for embedded speech recognition in mobile electronic devices, such as mobile phones, dictation machines, PDAs (personal digital assistants), mobile games consoles, etc. For example, email and text message dictation, note taking, form filling, and command and control applications are all potential applications of embedded speech recognition.

However, when a medium to large vocabulary is required, effective speech recognition for mobile electronic devices has many difficulties not associated with speech recognition systems in hardware systems such as personal computers or workstations. Firstly, the available power in mobile systems is often supplied by battery, and may be severely limited. Secondly, mobile electronic devices are frequently designed to be as small as practically possible. Thus, the memory and resources of such mobile embedded systems tends to be very limited, due to power and space restrictions. The cost of providing extra memory and resources in a mobile electronic device is typically much higher than that for a less portable device without this space restriction. Thirdly, the mobile hardware may be typically used in a noisier environment than that of a fixed computer, e.g. on public transport, near a busy road, etc. Thus, a more complex speech model and more intensive computation may be required to obtain adequate speech recognition results.

These restrictions have made it difficult to implement effective speech recognition in mobile devices, other than with very limited vocabularies.

Some prior art schemes have been proposed to increase the efficiency of speech recognition systems, in an attempt to make them more suitable for use in mobile technology.

In an article entitled "A low-power accelerator for the SPHINX 3 speech recognition system", in University of Utah, International conference on Compilers, Architectures and Synthesis for Embedded Systems, November 2003, Davis et al have proposed the idea of using a special purpose co-processor for up-front calculation of the computationally expensive Gaussian output probabilities of audio frames corresponding to particular states in the acoustic model.

In an article entitled "Hardware Speech Recognition in Low Cost, Low Power Devices", University of California, Berkeley, CS252 Class Project, Spring 2003, Sukun Kim et al describe using special purpose processing elements for each of the nodes in the network to be searched. This effectively implies having a single processing element for each phone in the network. An alternative suggested by Sukun Kim et al is to provide a processor for each state in the network.

In an article entitled "Dynamic Programming Search for Continuous Speech Recognition" in IEEE Signal Processing Magazine, September 1999, Ney et al discuss language model lookahead. Language model lookahead involves computation of a language model factor for each node (i.e. phone) in the lexical tree. This technique is also known as smearing. Each phone instance in the search network can be given a language model factor when it is used in the lexical tree search. Ney et al show that for an example bigram language model, the average number of states per 10 ms frame can be reduced from around 168,000 states with no language model lookahead to around 8,000 states when language model lookahead is used. They also show that bigram language model lookahead requires about a quarter of the states compared with unigram language model lookahead.

Although these prior art documents provide improvements to speech recognition in embedded mobile technology, further improvement is still needed to provide a larger vocabulary and better accuracy.

One aspect of the present invention provides a speech recognition circuit including a circuit for providing state identifiers which identify states corresponding to nodes or groups of adjacent nodes in a lexical tree, and for providing scores corresponding to said state identifiers. The lexical tree includes a model of words. The speech recognition circuit also has a memory structure for receiving and storing state identifiers identified by a node identifier identifying nodes or groups of adjacent nodes, the memory structure being adapted to allow lookup to identify particular state identifiers, reading of the scores corresponding to the state identifiers, and writing back of the scores to the memory structure after modification of the scores. An accumulator is provided for receiving score updates corresponding to particular state identifiers from a score update generating circuit which generates the score updates using audio input, for receiving scores from the memory structure, and for modifying said scores by adding said score updates to said scores. A selector circuit is used for selecting at least one node or group of nodes of the lexical tree according to said scores.

One suitable type of hardware for the memory structure includes a content addressable memory (CAM). A CAM is a memory unit which stores a series of data items using a series of addresses. However, the memory is accessed by specifying a data item, such that the CAM returns the corresponding address. This contrasts with a random access memory (RAM) in which the memory is accessed by specifying an address, such that the RAM returns the corresponding data item.

However, the memory structure is not limited to including a CAM. Other types of hardware are also possible, to provide this functionality. For example, a single chip which operates in the same way as a CAM and RAM may be used instead.

Embodiments of the present invention provide a solution to the problem of how to map a lexical tree search to a CAM system architecture. The realisation by the present inventors that certain speech recognition data structures can be mapped into the CAMs allows a lexical tree search to be performed using a CAM system architecture.

Further embodiments of the invention include a counter for sequentially generating state identifiers, and using said generated state identifiers to sequentially lookup said states in the memory structure.

The node identifier may comprise a direct reference to the lexical tree. However, in some embodiments, the node identifier for at least some of the states includes a pointer to a node identifier for another state. For example, a state corresponding to the furthest part of the search path in the lexical tree may be referenced by a node identifier which directly links to a particular node or group of nodes in the lexical tree. In a lexical tree comprising phones, using a state model of triphones, the node identifier may indicate the position of a triphone in the lexical tree.

However, in this example, for states occurring further back in the search path, instead of supplying a node identifier linking directly to the lexical tree, instead a pointer to a node identifier of another state may be supplied. E.g. a triphone instance may have a pointer to another triphone instance, which has a pointer to another triphone instance, which has a pointer to a node or group of nodes in the lexical tree. Chains of reference may be set up in this way, where only the last state in the chain has a direct pointer to the lexical tree.

There may not be a one-to-one correspondence between the nodes of the lexical tree and the node identifiers. This will occur for a branched lexical tree, where the nodes represent monophones, but the acoustic model states represent triphones, i.e. groups of three adjacent monophones. Then, paths of three monophones will have unique identifiers to be stored in the memory structure, rather than single monophones having unique identifiers.

Phone instance numbers may be generated, and used to uniquely label each phone instance. They can be generated sequentially, using a counter. The phone instance numbers may be used as pointers between phone instances to assist in node identification. It is thus not essential to provide a direct node identifier for each phone instance to directly indicate a location in the lexical tree. The dynamic network of phone instances provided in the memory structure may thus include both direct and relative references to the lexical tree.

The memory structure may be divided into one part which stores phone instance identifiers and direct references to the lexical tree, and a second part which stores phone instance identifiers and corresponding states. This can speed up the processing, by only storing the phone instances which are furthest on in the lexical tree in the first part of the memory structure.

The memory structure may also be divided into separately accessible units, to reduce the amount to data in each unit, thereby decreasing the chance of finding the same two states identifiers in different phone instances in any single memory unit, and increasing the chance of some state identifiers being completely absent from any single memory unit. This makes it easier to deal with the situation when the same two state identifiers are found, because a spare time slot is available for processing when a state identifier is not present.

A further aspect of the invention provides a distance calculation engine within a speech recognition system. The distance calculation engine may be included within an accelerator. The accelerator may include logic to interface with other parts of a speech recognition circuit, in addition to the distance engine, although this is not essential. For example, the accelerator may include one or more results memories for storing distances calculated by the distance calculation engine. The accelerator may also include at least one of a memory for storing one or more acoustic models, a decompressor for decompressing acoustic data that has been stored in a compressed format, a memory for storing feature vectors, a checksum or data signature calculation means, buffers for data storage, and data registers. The accelerator may be implemented in software or in hardware, or in a combination. It may be physically separate to the rest of the speech recognition circuit, although this is not essential.

The distance calculation engine may calculate one or more of a wide range of distance metrics and probability distributions. The distances may represent the likely correspondence of feature vectors to states in an acoustic model. In other words, the distances can indicate the similarity of an audio data frame to each possible state in an acoustic model

There are a wide variety of probability distributions that can be used for the distance calculation stage of a speech recogniser, and a wide variety of distance metrics used. These are widely documented in the literature. A point is a simple example of a probability distribution.

A common choice is to use Gaussian Distributions and correspondingly the Mahalanobis Distance metric. The Gaussian probability distribution is then defined by a mean vector, which defines centre point in the N-dimensional space, and a Covariance matrix which defines the shape of the probability distribution. It is common to restrict the Covariance matrix to be a diagonal matrix (only N non-zero values along the diagonal of the N×N matrix) which significantly lowers the implementation cost by reducing the number of arithmetic operations.

In particular embodiments, the distance calculated is a Mahalanobis distance. Particular examples of this are described later in the specification.

In one embodiment, the distance engine autonomously computes all of the distances associated with a given feature vector. This may comprise computing distances for every state in the lexicon. The distance engine may operate in a pipelined manner with other stages of the recognition process. In this context a distance is an indication of the probability or likelihood that a feature vector corresponds to a particular state. An important class of distance computation in speech recognition is the calculation of output state probabilities in recognisers using Hidden Markov Models. Another possible use is in recognisers using Neural Networks.

The distance engine reads data from the acoustic models to use as parameters in the calculation. The acoustic models may be optionally stored in a compressed format. The distance engine may read and de-compress the acoustic models one (or more) times for each feature vector processed. Each reading of the acoustic models may require reading the entire acoustic model, or various optimisations may be implemented to avoid reading parts of the acoustic model that are not required to be used for calculations with the current feature vector. The distance engine may use a de-compression method where the de-compression is sign or zero extension or may otherwise convert data of narrow or variable width to a wider data format. The distance engine may use a de-compression method where the de-compression is sign or zero extension or may otherwise convert data of narrow or variable width to IEEE standard single or double precision floating point format. The distance engine may use a decompression method where decompression is a codebook decompression of a binary bitstream, where the codebook is stored as part of the acoustic model data. The distance engine may use a decompression method where the decompression is decompression of a Huffman or Lempel-Ziv compressed stream. The distance engine may use a decompression method where decompression is decompression of run length encoded data. The distance engine may use a decompression method where decompression is decompression of difference encoded data. The distance engine may use a decompression method using any well known lossy or lossless compression scheme. The distance engine may use a decompression method using subspace distribution clustering. The distance engine may use a decompression method comprising any combination of the above described decompression types.

The distance engine may read the acoustic models from a dedicated on-chip memory. The distance engine may read the acoustic models from a dedicated off-chip memory. The distance engine may read the acoustic models from a shared on-chip memory. The distance engine may read the acoustic models from a shared off-chip memory. Any of these acoustic models may be compressed.

The distance engine may compute a CRC or checksum or similar signature as it reads in the acoustic model and compares this to a stored CRC, checksum, or signature, in order to check that the acoustic model has not been corrupted, and signals an error condition if such corruption is detected. The stored CRC, Checksum, or signature may had been pre-computed and stored in the model data, or it may be computed at the time the model data is loaded into the Acoustic Model Memory. It may be held in Acoustic Model Memory, or it may be loaded into a register or another memory from where it can

be accessed and compared when the CRC/checksum/signature is computed each time the Acoustic Model is loaded.

The distance engine may support the pass-through of data from the front-end to the search-stage. The data to be passed through will be supplied to the distance engine as an adjunct to the feature vector, and the distance engine will pass it to the search stage as an adjunct to the distance results. This provides a simple mechanism for passing frame-specific data that is not involved in the distance calculation through to the search stage, and keeping it associated with the correct frame, which may otherwise be complex in pipelined systems with multiple processors. The data passed through may be for any purpose. Examples might include silence detected, end-of-audio-stream detected, a frame number, information that some intervening frames have been dropped, or data from another input device such as a button or keyboard in a multimodal interface.

The distance engine may be implemented in hardware, software, or a combination. Other stages may be implemented in hardware, software, or a combination. The distance engine may be implemented with any number representation format including fixed point or floating point arithmetic, or any mixture of number representation formats.

In particular, the other stages may be implemented on a CPU, or on a DSP and CPU. The "DSP" and "CPU" may each be implemented as software programmable devices.

The distance engine may implement one or more additional pipeline stages to overcome delays introduced by low bandwidth, high latency, or conflicted bus interfaces. The distance engine may also implement additional pipeline stages to maintain the same throughput while allowing more time for each distance calculation. Particular embodiments of the invention may include one or more of the above aspects.

A further aspect of the invention comprises a speech recognition circuit, comprising: an audio front end for calculating a feature vector from an audio signal, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame; calculating circuit for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model; and a search stage for using said calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words; a buffer memory between the calculating circuit and the search stage, for receiving data passing from the calculating circuit to the search stage, wherein a processor in the search stage has higher bandwidth and/or lower latency access to the buffer compared to the bandwidth and/or latency of direct transfer between the calculating circuit and the search stage. The data transfer from the calculating circuit to the buffer memory and/or from the buffer memory to the search stage may be performed as one or more sequential bursts. The data transfer to the buffer memory may be performed in parallel with data transfer to the calculating circuit and/or in parallel with data transfer to the search stage. A second buffer memory may be provided between the audio front end and the calculating circuit.

A further aspect of the invention comprises a speech recognition circuit, comprising: an audio front end for calculating a feature vector from an audio signal, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame; calculating circuit for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model; and a search stage for using said calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words; comprising an elastic buffer between at least one of the front end and calculating circuit, or the calculating circuit and search stage, and/or for buffering said audio signal.

A further aspect of the invention comprises an accelerator for a speech recognition circuit, the accelerator comprising: calculating means for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model, wherein the feature vector comprises a plurality of extracted and/or derived quantities from an audio signal during a defined audio time frame; means for comparing a first version of a stored checksum of data representing said acoustic model and a second version of said stored checksum, wherein the second version is obtained from an updated measurement and calculation of the checksum; and means for indicating an error status if the checksums do not match.

A further aspect of the invention comprises an accelerator for a speech recognition circuit, the accelerator comprising: calculating means for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model, wherein the feature vector comprises a plurality of extracted and/or derived quantities from an audio signal during a defined audio time frame; wherein said accelerator is configured to autonomously compute distances for every acoustic state defined by the acoustic model.

A further aspect of the invention comprises an accelerator for calculating distances for a speech recognition circuit, the accelerator comprising: calculating circuit for calculating distances indicating the similarity between a feature vector and a plurality of predetermined acoustic states of an acoustic model, wherein the feature vector comprises a plurality of extracted and/or derived quantities from an audio signal during a defined audio time frame; first and second storage circuit, which may be referred to as result memories, each for storing calculated distances for at least one said audio time frame, and for making said stored distances available for use by another part of the speech recognition circuit; control circuit for controlling read and write access to the first and second storage circuit, said control means being configured to allow writing to one said storage means while the other said storage means is available for reading, to allow first calculated distances for one audio time frame to be written to one said storage means while second calculated distances for an earlier audio time frame are made available for reading from the other said storage means.

Embodiments of the invention may comprise means for generating a checksum or computed signature for the acoustic model data stored in the memory, and means for comparing checksums or computed signatures that have been calculated at different times, to indicate an error status if the checksums do not match, one possible cause of such mismatch being that the acoustic model data has been overwritten by said other data and said error status being used to indicate that the acoustic model should be re-loaded into the said memory.

A further aspect of the invention comprises a speech recognition circuit comprising: lexical memory containing lexical data for word recognition, said lexical data comprising a lexical tree data structure comprising a model of words; means for accessing a state model corresponding to each phone or each group of phones in the lexical tree; a content addressable memory for storing content addressable data for each phone or group of phones, including states corresponding to said phone or group of phones, and for storing an address value for each said phone or group of phones; a RAM configured to store accumulated scores for each said phone or group of phones, the accumulated scores being addressable

by said address value for each said phone or group of phones; means to obtain scores that each of a plurality of frames of an audio signals corresponds to each of a plurality of said states; a counter to sequentially search for each said state in the content addressable memory, to obtain the corresponding address value if the state is found in the content addressable memory; means to use said address value to access an accumulated likelihood and an accumulator to add said likelihood to the accumulated likelihood; means to use the phones or groups of phones with the highest accumulated scores to obtain a plurality of next phones from the lexical tree which correspond to the next phone; and output means for outputting a lexical tree path of highest likelihood.

A further aspect of the invention comprises speech recognition apparatus comprising: a lexical tree having a corresponding state model; means for obtaining scores of an audio input corresponding to each of a plurality of states in said state model; a content addressable memory for storing a marker indicating a part of the lexical tree, and one or more states associated with said part of the lexical tree; a random access memory addressable by the CAM output, to output accumulated scores for states corresponding to said parts of the lexical tree; adder means for adding likelihood to said accumulated likelihood, to be stored back in the RAM.

A further aspect of the invention comprises speech recognition apparatus, comprising: a CAM-RAM arrangement for storing records including pointers to a lexical tree, and accumulative scores for states within the lexical tree; input means for obtaining scores that an audio frame corresponds to a particular state in the lexical tree; an accumulator for calculating the updated scores and modifying the records in the CAM-RAM accordingly; output means for outputting a path of highest likelihood in the lexical tree.

A further aspect of the invention comprises a speech recognition method comprising: storing state identifiers which identify states corresponding to nodes or groups of adjacent nodes in a lexical tree, and scores corresponding to said state identifiers in a memory structure, the lexical tree comprising a model of words and the memory structure being adapted to allow lookup to identify particular state identifiers, reading of the scores corresponding to the state identifiers, and writing back of the scores to the memory structure after modification of the scores; repeating the following sequence of steps for each of a plurality of incoming frames of an audio signal; obtaining score updates corresponding to the likelihoods that said frame of the audio signal corresponds to each of a plurality of said states; accessing said memory structure to obtain scores, updating the scores by adding score updates to the scores, and writing back the updated scores to the memory structure; determining if scores for states furthest on in the lexical tree correspond to a significant likelihood, and if so, then accessing the lexical tree to determine the next set of possible states; and storing the next set of possible states and said scores of significant likelihood in the memory structure.

A further aspect of the invention comprises speech recognition circuit comprising: a circuit for providing state identifiers which identify states corresponding to phones or groups of adjacent phones in a lexical tree, and for providing scores corresponding to said state identifiers, the lexical tree comprising a model of words; a memory structure for receiving and storing state identifiers and phone instance identifiers uniquely identifying instances of phones or groups of phones in the lexical tree; said memory structure being adapted to allow lookup to identify particular state identifiers, reading of the scores corresponding to the state identifiers, and writing back of the scores to the memory structure after modification of the scores; an accumulator for receiving score updates corresponding to particular state identifiers from a score update generating circuit which generates the score updates using audio input, for receiving scores from the memory structure, and for modifying said scores by adding said score updates to said scores; and a selector circuit for selecting at least one phone instance identifier according to said scores.

A speech recognition apparatus according to the invention may be embedded in or included in a mobile electronic device such as a mobile telephone, PDA (personal digital assistant), etc.

Embodiments of the present invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

FIG. **1** is a block diagram of the system architecture for a speech recognition apparatus according to an embodiment of the invention;

FIG. **2** is a block diagram showing the main data structures used in the speech recognition apparatus of FIG. **1**;

FIG. **3** shows a block diagram of the Single Phone Instance Network Engine (PINE) of FIG. **1**;

FIG. **4** is a block diagram showing the Wave Front Manager architecture;

FIG. **5** shows the use of link nodes to permit path merging;

FIG. **6** shows a block diagram of a Path Merge Unit in the Phone Level;

FIG. **7** shows the Phone Book architecture;

FIG. **8** shows Lexical tree storage;

FIG. **9** shows the data structure for the word link record;

FIG. **10** shows a block diagram of Word Engine;

FIG. **11** shows a Language Model interface architecture;

FIG. **12** shows an example of a simple Loop Grammar;

FIG. **13** shows a block diagram of grammar engine;

FIG. **14** shows a single frame of the Audio Codec link protocol; and

FIG. **15** illustrates the system timing which may be used in the embodiment of FIG. **1**;

FIG. **16** is a block diagram showing a speech recognition circuit according to an embodiment of the invention, and illustrating data flow between parts of the circuit;

FIG. **17** is a block diagram showing a distance calculation engine according to an embodiment of the invention;

FIG. **18** shows ideal data flow over several time steps within a speech recognition circuit having separate front end and search stages, in an embodiment of the invention;

FIG. **19** shows an example of actual data flow over several time steps within a speech recognition circuit having separate front end and search stages, in a further embodiment of the invention;

FIG. **20** shows ideal data flow over several time steps within a speech recognition circuit having a shared processor for front end and search stages, in an embodiment of the invention;

FIG. **21** shows an example of actual data flow over several time steps within a speech recognition circuit having a shared processor for front end and search stages, in an embodiment of the invention;

FIG. **22** shows ideal data flow in a speech accelerator pipeline with two data transfer stages, according to an embodiment of the invention; and

FIG. **23** shows an example of actual data flow in a speech accelerator pipeline with two data transfer stages according to an embodiment of the invention.

FIG. **1** shows a speech recognition system architecture for an embodiment of the invention. The speech recognition system is divided into several levels of operation: a state level, a phone level, a word level, a grammar level, and a system level.

Each level is shown in FIG. **1** as a grey shaded box, interconnected to other levels of the system.

The information passed between different levels or subcircuits of FIG. **1** is indicated on the FIG. **1** using the following identifiers.

Identifier	Definition
grammar_id	Identifies which grammar model is currently being used
grammar_node_id	Identifies a node in the grammar model
word_id	Uniquely identifies a word
wordlat_node_id	Identifies a node in the word lattice
LT_id	Lexical tree identifier. Identifies the lexical tree to be used
LT_branch_id	Identifies a unique branch in a lexical tree. A branch is defined as all of the monophones subsequent to and including the first monophone in a tree.
LT_node_id	Identifies a node in a lexical tree
monophone_id	Identifies a monophone
transvec_id	Identifies which transition vector to use
state_id	Identifies the state
phone_no	Unique phone instance identifier
prev_phone_no	Phone number of previous phone in network
triphone_id	Triphone identifier
LML_factor	Language model lookahead factor. The smeared language model score that must be added

The system level includes a general purpose processor and cache, a system controller, a DMA (direct memory access) controller, clocks, PLL (phased locked loop), GPIO (general purpose I/O interface), timer watch dog, memory interfaces to interface to main memory and non volatile storage, serial, host, debug and display interfaces. The system level also includes a model adaptation engine, a language model factor engine, a speech recognition system controller, a language model interface, and a recognition output interface.

The processor organises the transfer of the various models from the non volatile storage to the various model memories. It handles the language model factor requests. Language model factors are used to by the state level to apply the language model score. The language model (LM) interface provides the mechanism for obtaining LM factors as and when they are needed. The processor keeps a copy of the wave front in memory which is updated simultaneously with that in the wave front manager. Using its copy of the lexical tree, the processor can determine which language model scores may be needed in the coming frames. Once calculated, the language model factors can be kept in memory for use in subsequent frames. The system level is also used for forming speech model adaptation, and acting as an overflow for the various buffers used by the architecture.

The system level includes the audio codec interface and the audio front end interface, connected to the processor via a APB (advanced peripheral bus) to AHB (advanced high performance bus) bridge.

The grammar level includes a grammar engine and grammar model(s). The grammar level sets out the rules of the particular recognition task at the highest level. During system initialisation a number of grammar models might be loaded, for example a command grammar and a dictation grammar. Command mode grammar is a restricted grammar which includes only a very limited lexical tree with command specific words. Some branches of this lexical tree may include multiple words together, where a command has such a structure, e.g. "open file menu". Typically, only a single instance of the lexical tree will be needed. Command mode is used because it can obtain a higher accuracy, due to the lexical tree constraint. In dictation mode grammar, a full lexical tree structure is used.

The grammar may be selected by a user during the recognition process. For example, the command grammar may be selected when the user presses a command button. The architecture supports fast swapping of grammar models. For example, a user may be in the middle of dictating a message when a "punctuation button" is pressed and the phrase "question mark" uttered. At this point, a punctuation grammar could be loaded, in order to increase recognition accuracy.

The word level comprises a word engine, which includes a word lattice and a word graph generator. The word level has two main functions, of keeping a record of the search path at the word level using a word lattice, and interfacing to the language model via a cache. The word lattice is a record of all of the words that have been searched up to the current frame. As new words are searched, they are appended into the lattice at the appropriate place. After the final frame in the current sentence, the n-best word lattice node identifiers from the n-best scoring tokens are passed back to the word lattice. This enables the n-best sentences to be traced back through the lattice. The word lattice can also be post processed to produce a word graph which is essentially a compact version of the word lattice.

At the word level, a record is kept of all tokens that have passed through a word end. This results in a lattice of words. When the final feature vector has been processed, the best scoring token is identified and used to trace back the search path back to the start of the search.

The phone level includes a "phone book" for generating new triphone instances, a wave front manager for sending requests to the phone book to generate new phone instances, a phone request arbiter, and a path merge unit.

The phone book includes a lexical tree, which comprises a vocabulary of words, sounds and/or phrases organised according to the phones or monophones making up the sound. A lexical tree is commonly used in speech recognition rather than a linear lexicon, because it greatly reduces the amount of storage space required to store a word vocabulary, by avoiding duplication of common word beginnings. It is also easier to search when the word beginning is known.

Although it is possible to search a lexical tree based on monophones, it is well known that better results can be achieved in speech recognition by basing the lexical tree search on groups of more than one monophone, since the sound of one syllable commonly affects the pronunciation of the next syllable in the same word, or the first syllable in the next word. In this embodiment, groups of three monophones are used, and these are known as triphones. It is alternatively possible to use a different number of monophones for each group. The phone generator in the "phone book" uses data from the lexical tree to generate triphones.

In one embodiment of the invention, approximately 45 different cued mono-phones are considered. This gives of order 60,000 tri-phones. Each triphone in this embodiment is modelled by a Hidden Markov Model (HMM) with three states. There is no correspondence between individual states and individual phones of the triphone.

Alternatively, it is possible to model each triphone by a single state, or by any other number of states.

The phone level is responsible for generating the search space according to the word structure specified in the grammar model and the phone structure specified by the lexical tree. The search space can be envisaged as a wave moving through an acousto-linguistic search space. In order that the wave can be dynamically created, a record of the wave front

phone instances must be maintained. This is done in a wave front phone CAM/RAM structure, within the wave front manager. When a token occurs in the final state of a phone instance in the PINE array at the front of the wave, a new phone instance must be generated at the phone level and passed to the PINE array.

The phone instance includes a phone_no, which is a phone instance identifier to identify the phone uniquely in the current search. The phone instance also includes a word lattice node identifier (labelled "word_lat_node_id") to identify the phone's position in the word lattice, state identifiers (labelled "state_ids") for the HMM, storage to hold the tokens as they move through the HMM and LM_factor which is used to by the PINE to apply the language model score. To handle the connectivity between phone instances, a phone instance points to a previous phone instance using the phone_no of the previous phone. This allows tokens to be passed between phones. A "token" is a hypothesised search path, represented by a non-zero score at a particular state, and tokens are propagated through the HMM with each time frame.

The phone instance network can be visualised as a wave with new phone instances being added to the front of the wave, and empty phone instances being removed from the back of the wave. In some circumstances, the back of the wave may be in the same position as the wave front. The back of the wave moves on through the lexical tree as tokens are either moved on or pruned.

Tokens, each representing a particular hypothesis, hop through the states in the HMMs, each hop consuming a feature vector. Each token explores all possible next states which typically involves placing a token in the next state and keeping one in the current state. When there is more than one token in a state, the least likely token can be removed because both have the same word history. When a token reaches the last state of a phone instance at the from of the wave, the next phone instance(s) are requested from the phone level. The phone level uses the lexical tree to create the new phones. As well as looking up the state identification numbers for each HMM state, it also incorporates the correct language model factor.

The language model factor engine is used to smear the language model scores across the lexical tree. This operates in two ways depending on whether the factor is needed for a word start phone(s) or word internal phones. For word internal phones, LM factors can be computed as needed, provided that the LM scores are available in cache. This is because in general the storage requirements for the LM scores are less than the storage of the LM factors for all of the factors in the branch. However when a token is in a word end phone there may a significant number of word start phones e.g. up to $(\text{monophones})^2$ requiring a large amount of computation. To avoid this the LM engine will pre compute word start LM factors when it seems highly likely that a token will be passed to the word starts. These word start LM factors will be held in the LM factors cache. The phone level is responsible for generating the search space according to the word structure specified in the grammar model and the phone structure specified by the lexical tree.

The state level shown in FIG. **1** includes a phone instance network engine (PINE) array, a MHD (Mahalanobis distance) engine, an audio front end, a feature vector buffer for model adaptation, and compressed acoustic models. The state level is the lowest level, in which a network of phones is dynamically created.

The audio input for speech recognition is input to the front end unit of the state level. This audio input may be digital audio, or it may be analog audio which is converted to digital audio using an analog to digital converter. The audio input is divided into time frames, each time frame typically being of the order of 10 ms.

For each audio input time frame, the audio signal is converted into a feature vector. This may be done by splitting the audio signal into spectral components. For example, in the present embodiment, the audio input signal is split into 12 frequency bands and one energy band, and a feature vector is constructed using these 13 components, plus their first and second derivatives, a total of 39 components.

The feature vector is passed to the MHD (Mahalanobis distance) engine. This computes all possible state output likelihoods for the current frame as the state mean and covariance data is read from the acoustic model memory.

The likelihood of each state is determined by the distance between the feature vector and each state. To reduce the number of states, a technique known as "state tying" may be used. Multiple states are grouped together in 39D space, according to their proximity to one another. An average or central value from each group is used to represent the entire group. Thus, the effective number of states may be reduced from 40,000 to 8,000. The accuracy may be improved using a language model in addition to the word recognition lexical tree. A language model is based on the probability of sequential words. The above figure of 8,000 states gives best recognition results when used with a language model.

The MHD (Mahalanobis Distance) is a distance between two N-dimensional points, scaled by the statistical variation in each component. In embodiments of the invention, the Mahalanobis distance between the feature vector and each state is calculated, to determine similarity of the feature vector to each state.

The Feature Vector represents a point in an N-dimensional space. Different embodiments of the invention may use different values of N. Generally we see values of N in the range of 20 to 39, although values outside this range are not excluded.

The distance calculator stage of the recognition process computes a probability or likelihood that a feature vector corresponds to a particular state. Each state has an associated probability distribution in the N-dimensional space. The distance calculator computes the distance in the N-dimensional space from the Feature Vector to the probability distribution for each state. In some embodiments, the calculation is performed only when the search stage algorithms determine that the distance for that particular state is needed by the search stage. In some embodiments, the distances for all of the states are calculated and made available to the search stage, which allows pipelined operation. Embodiments that do the calculations on demand are preferable.

The Mahalanobis Distance (MHD) is extensively described in the literature. It may be calculated using the formula

$$\text{MHD value}=\ln(P(x))=-0.5[N*\ln(2*pi)+\ln|Cov|)]+(-0.5*[(x-u)'*\text{inverse}(Cov)*(x-u)])$$

where

N=number of dimensions in the space

x=feature vector, a point in the N dimensional space

u=mean vector

Cov=covariance matrix

|Cov|=determinant of covariance matrix

(a)'=transpose of matrix a

Different embodiments may use variations on this equation, for example the base for the logarithm is not always the natural base e that is used in the above example. The log of the probability is preferably calculated, rather than the probabil-

ity, as it simplifies the implementation. Not all embodiments do this. In alternative embodiments, the likelihoods rather than probabilities are computed as it leads to simpler implementations of computations in the search stage.

The feature vector is used to calculate 8,000 MHD distances for each time frame, i.e. one distance for each of the 8,000 states. It is beneficial, although not essential, to perform the MHD calculations separately in a pre-processor, known as a MHD engine, because MHD calculation is a computationally intensive process.

Due to the 10 ms frame length, a feature vector arrives at the MHD engine of the state level every 10 ms. The MHD engine calculates the likelihood of each feature vector corresponding to each of a number of states derived from an acoustic model. Each state is also a 39 dimensional vector, having the same spectral components as the feature vector. The MHD engine calculates the difference in 39-D space between the feature vector and each state to obtain the likelihood for that state. A parallel MHD or a single MHD arrangement may be used.

The feature vectors may be sent to a feature vector buffer for model adaptation, which determines whether the observed features fit well with the acoustic model, and may initiate an adaptation of the model if the fit is not sufficiently good.

Scores indicating the state likelihoods or probabilities are output from the MHD engine to the PINE array. It is preferable to use likelihoods rather than probabilities, because likelihoods can be added together to give a total likelihood. Probabilities must be multiplied, which can give rise to rounding errors. In this embodiment, eight separate pines are used, each of 4 kilobytes, thus storing 4000 phones per pine.

During recognition, the Phone Instance Network Engine (PINE) Array will contain a network of phone instances. Tokens representing search hypotheses in the form of a probability(likelihood) will exist in some of the HMM states of the phone instances. The tokens are propagated into all available states and updated with the corresponding state output probabilities. Maximum model, word end and beam pruning are applied to the tokens to ensure the search space is kept bounded. When a token is in the final state of a phone model and has avoided being pruned, the PINE makes a next phone request to the Phone level. The PINE array may receive a number of new phone models from the phone level which must be appended to the appropriate existing phone.

FIG. **2** shows the data structures stored in the PINE array, and other main data structures used during the acoustic search. Each PINE includes a phone instance CAM and a phone instance RAM. The phone instance CAM stores the phone instance number (labelled "phone_no") which uniquely identifies each phone instance. The phone instance CAM also stores the state IDs of the three HMM states used to model the phone, and flags indicating the position of the phone. The "back" flag indicates if the phone is at the back of the wavefront. The "front" flag indicates if the phone is at the front of the wavefront. The "empty" flag indicates if there are no tokens in any of the three states for that phone. The "end" flag indicates if the phone is at the end of a word.

The phone instance RAM stores the token scores for each of the three states corresponding to each phone. The st_dum field indicates a dummy state, which is used to represent a state prior to the first state of the first phone. The "tmp" field is used for passing a token from a previous phone in the lexical tree to the current phone. This will be explained in more detail later. The prev_phone_no field is for storing a pointer to the previous phone instance in the network. The wordlat_node_id identifies the position of the phone instance on the word lattice node, and the LM_factor stores the language model factor for that phone instance.

At the phone level a record of all phones on the wave front must be maintained. If a phone instance is on the wave front, the phone_no stored in the phone instance CAM will point to the details in the wave front phone CAM and RAM.

The wave front phone CAM and RAM keeps pointers into the grammar and lexical tree. When a token in a wave front phone reaches the last state, the PINE array makes a request to the phone level by passing back the phone_id. The phone_id is used as a key to determining the token's grammer_node_id and LT_node_id within the wave front CAM/RAM structure. If the token is at a word end the grammar model is checked. A grammar identifier points to the type of grammar.

At the word level, FIG. **2** shows an example word lattice structure, with silence having a word_id of 0, and a wordlat_node_id of 6. This is shown as connected to two word nodes—"which" has a word_id of 238 and a wordlat_node_id of 4, and "neither" has a word_id of 3556 and a wordlat_node_id of 5.

At the highest level, i.e. the grammar level, a simple grammar model is shown which will allow the phrase 'silence which house silence' or 'silence neither house silence' to be spoken. Whenever a word end is reached during the search, the grammar model must be consulted to determine the next possible word(s) or lexical tree(s). This allows the word lattice at the word level, consisting of a network of words, to be assembled. Whenever a word end is reached, the word is added to the lattice and the wordlat_node_id updated accordingly.

FIG. **3** shows a single phone instance network engine (PINE). At system power up the state transition RAM will be loaded with the appropriate state transition vector but the CAM and RAM memories will be empty. When the system is initialised an initial set of phone model(s) will be instantiated with a token in the dummy state and a path score of zero. The phone models are received from the phone level.

When the audio data is processed, a stream of all possible state output probabilities will be received from the MHD engine. A counter that is synchronous to the stream of state output probabilities from the MHD produces a concurrent stream of state_ids. The job of the CAM is to perform a search to see if the state_id is instantiated anywhere in the CAM state_id fields. If it exists and the location doesn't have its empty flag set then this indicates that a token may need to be updated. The match signal from the CAM must be encoded into an address for the RAM. The corresponding RAM address can then be read. Any token that will end up in the identified state can then have it's path score updated and including any transition vector likelihood.

It is possible that a single state_id generates multiple CAM matches. To handle this situation the CAM and RAM are effectively decoupled by means of a match fifo and repeat count fifo. The state_id and corresponding state_output_probability are stored in the repeat count fifo. During a subsequent frame when there is no match, this state_id can be revisited, ignoring the matches already dealt with.

All searched phone instances corresponding to words in the current possible lexical tree are stored in the CAM-RAM. This includes phone instances at the beginning of the word, middle of the word and end of the word. These phone instances are connected by the "previous phone instance" field, which allows traceback through an entire word within the CAM-RAM.

A mechanism is thus required to pass tokens from one phone instance to the next. This mechanism is provided by the tmp field in the RAM.

The tmp field in the RAM holds tokens during token passing.

At the start of each time frame, the audio input is converted into a feature vector, the MHD values calculated, and the MHD values used to update the token scores. After update of all tokens, 2 passes are made through the CAM-RAM, firstly for pruning, and then for token passing. The pruning in this example removes tokens of below a threshold score, although other types of pruning are also possible. The pruning may involve removing a predetermined proportion of the tokens, chosen as those of lowest likelihood. A third possible method of pruning is removing all tokens with scores below a first predetermined value, but at word ends removing all tokens with scores below a second predetermined value, the second predetermined value being greater than the first predetermined value.

The token passing works as follows. Each phone instance points back to a previous phone, due to the prev_phone_no field stored in the RAM. During the token passing, the third token of a previous phone is grabbed and put into the tmp field of the current phone.

In other words, if a token is on the third state, and it is not pruned, it must enter the next state of the next phone. To do this, the tokens that lie in the third state are moved across to the tmp field.

For example, phone no. 345 needs to look at phone no. 320 to see if any tokens are in the third state. If yes, it takes these tokens and puts them in the tmp field.

One field in the CAM is labelled "front". This indicates whether the phone is on the wave front. If the phone is not on the wave front, then another phone will have picked up the token in the third state. However, if the phone is on the wave front, then no other phone will have picked up the token on the third state. Therefore, if there is a token in the third state, and the phone is on the wavefront, it is necessary to generate a next phone request.

The current phone number, word lattice identifier and word end flag are sent to the wave front manager, to request a next phone. If the word end flag indicates a word end, then the score is also passed to the wave front manager. The score can be passed in any case, but it is only required at a word end.

The phone level generates the next phone or phones. A new phone instance starts, and all of state**1**, state**2** and state**3** have scores pre-set to zero. The old score is provided in the tmp field.

The old phone is still within the wave, but it is unlikely to be at the back of the wave at this stage. When it gets to the back of the wave, it is deleted. Until it gets there, it remains in the CAM-RAM as before. The third token remains where it was, since tokens can either stay in the third state or move on.

Note that it is possible to end up with more that one token in each state. If this occurs, then the token with the poorer score is deleted. This does not affect the result of the lexical tree search, because both tokens on the same state have exactly the same word history.

FIG. **4** shows a block diagram of the wave front manager. The wave front manager is responsible for maintaining a record of the set of phone models that exist at the front of the acoustic search space. This set of phone models is referred to as the "wave front." The key requirement for this block as to minimise the time between receiving a "next phone request" and generating the appropriate responses. The wave front manager receives the next phone requests from the PINE array and generates appropriate responses for the Phone Book, Word Engine, Path Merge Unit and Grammar engine The wave front manager also receives new wave front phone models from the Phone Book and updates the wave front.

The wave front manager receives the next phone request from the PINE array, via the phone request arbiter. The next phone request consists of a phone_no, word_end_flag and wordlat_node_id. If this phone is at the end of a word, the word_end_flag will be set.

The wave front manager uses the phone_no as input to the wave front phone CAM. As each phone_no is unique, it is guaranteed that only a single match will be generated. The wave front manager uses the CAM generated address to access the RAM. It then sends retrieved data from the RAM to the phone book, grammar engine, Path Merge Unit and word engine.

The wave front manager removes phone instance from CAM and RAM by writing 0's into CAM and RAM, and increments the empty count by one.

Sometime later, the wave front manager receives new phone instances from the phone book and place them in a fifo. The phone instances are taken from the fifo and written into the CAM and RAM at the next_empty_address. The empty_count is decremented by one. A match is run on the "occupy" flag, and the address is placed into next_empty_address register.

Both the phone level and the state level make use of a path merge unit to reduce the number of possible paths, by path merging. FIG. **5** shows the use of link nodes to permit path merging. In the present embodiment, all word end phone instances are linked to word start triphones with a link node. During each frame the path merge unit looks for word end tokens that can be recombined. This is done by looking for word end phones that have the same word_id and word history. When they are found a link is added using the path merge unit in the PINE array.

FIG. **6** shows a diagram of the architecture at the Phone level, showing the path merge unit. The path merge unit includes a path merge CAM and a path merge RAM. The path merge CAM stores word_id, word history for two words, and an "occupy" field. The path merge RAM stores a link_node_id. A CAM RAM occupancy manager keeps track of the empty addresses in the CAM and RAM. The path merge unit is only passed information from the wave front manager if the phone is a word end phone. For each frame, there is a check to see if there is an identical word with the same history, and if there is, the link_node_id is passed to the path merge unit. If there isn't an identical word with the same history, a new record is written into the Path merge CAM-RAM.

FIG. **7** is a block diagram showing the Phone Book architecture. The Phone Book is tasked with generating the next phone models when given a pointer into the lexical tree. The key requirement for this block as to minimise the time between receiving a "lexical tree address" and generating the appropriate responses, in particular the next phone model instance(s).

The lexical tree is used to build the next triphones based on the current context.

Current triphone	Next triphone	comments
Left context = a Central context = b Right context = c	discarded Left context = b Central context = c Right context = ?	next_leftmph_id stored in wave front manager current LT_node_id stored in wave front manager must be looked up. Large (<45) for new word start

The LT_node_id, next_left_mph_id and prev_phone_no are received at the phone generator from the wavefront manager. The LT_node_id is then used to address the lexical tree store and return the central monophone_id. One or more further accesses to memory are performed using the micro-coded instructions, each memory access returning the right monophone_id.

A two stage lookup is accessed using the three monophone_ids (giving a triphone), the associated state_ids. The phone model is assembled, and returned to the PINE array. The wavefront manager is then updated by sending it relevant parts of the phone model.

The lexical tree store is a micro-coded ROM with both instruction and data fields. The lexical tree store holds a number of lexical trees. Each lexical tree may hold single words or multiple words. Each lexical tree comprises lexical tree elements, and each element has three fields. For now, the fields will be integers and will be converted to std_logic_vector later. The following is an example of the lexical tree store, with the LT data in word_id order.

ZERO	z iy r ow	19	8	12	11	0	0	0	0	sp	0
ONE	w ah n	18	2	10	0	0	0	0	0	sil	1
TWO	t uw	14	16	0	0	0	0	0	0	ah	2
THREE	th r iy	15	12	8	0	0	0	0	0	ay	3
FOUR	f ow r	6	11	12	0	0	0	0	0	eh	4
FIVE	f ay v	6	3	17	0	0	0	0	0	ey	5
SIX	s ih k s	13	7	9	13	0	0	0	0	f	6
SEVEN	s eh v ih n	13	4	17	7	10	0	0	0	ih	7
EIGHT	ey t	5	14	0	0	0	0	0	0	iy	8
NINE	n ay n	10	3	10	0	0	0	0	0	k	9
OH	ow	11	0	0	0	0	0	0	0	n	10
The following shows the LT data in MP (monophone) start order.										ow	11
										r	12
EIGHT	ey t	5	14	0	0	0	0	0	0	s	13
FIVE	f ay v	6	3	17	0	0	0	0	0	t	14
FOUR	f ow r	6	11	12	0	0	0	0	0	th	15
NINE	n ay n	10	3	10	0	0	0	0	0	uw	16
OH	ow	11	0	0	0	0	0	0	0	v	17
SEVEN	s eh v ih n	13	4	17	7	10	0	0	0	w	18
SIX	s ih k s	13	7	9	13	0	0	0	0	z	19
TWO	t uw	14	16	0	0	0	0	0	0		
THREE	th r iy	15	12	8	0	0	0	0	0		
ONE	w ah n	18	2	10	0	0	0	0	0		
ZERO	z iy r ow	19	8	12	11	0	0	0	0		

The following is an example of the three fields of a lexical tree element. The first field is an instruction field, and the second and third fields are data fields.

Fields:	1st instr	2nd data	3rd data	
	2b	6b	6b	
	00	MP	MP	raw data field.
	01	br	br	branch address.
	10	br	br	branch address.
	11	word_id		double field.

Note that there might be no difference between "01" and "10"

To build the "ey" tree, we only have to consider "EIGHT", because this is the only word in the lexical tree which starts with "ey". EIGHT is represented as "ey t", with phone_nos 5 14 0 0 0 0 0 0. The first lexical tree element stores the raw data specifying the phone_nos, the second lexical tree element is an end field, and the subsequent lexical tree element is set to zero.

0	00	5	14	raw data
1	11	8	0	end field
2	00	0	0	

To build the "f" tree, we have:

FIVE	f ay v			6	3	17	0	0	0	0	0
FOUR	f ow r			6	11	12	0	0	0	0	0
0	01	4	0	branch forward four places as well							
1	00	6	3	as the implied other brach of next (one).							
2	00	3	17								
3	11	5	0	all addresses are relative offsets.							
4	00	6	11								
5	00	11	12								
6	11	4	0								
7	00	0	0								

To build the "n" tree, we have:

NINE	n ay n	10	3	10	0	0	0	0	0
0	00	10	3						
1	00	3	10						
2	11	9	0						
3	00	0	0						

To build the "ow" tree, we have "ow" in both the first and second data fields, as no tree is going to be a single monophone long.

OH	ow	11	0	0	0	0	0	0	0
0	11	11	0						
1	00	0	0						

To build the "s" tree, we have:

SEVEN	s eh v ih n			13	4	17	7	10	0	0	0
SIX	s ih k s			13	7	9	13	0	0	0	0
0	01	6	0	branch forward six places.							
1	00	13	4								
2	00	4	17								
3	00	17	7								
4	00	7	10								
5	11	7	0								
6	00	13	7								
7	00	7	9								
8	00	9	13								
9	11	6	0								
10	00	0	0								

To build the "t" tree, we have:

TWO	t uw	14	16	0	0	0	0	0	0
0	00	14	16						
1	11	2	0						
2	00	0	0						

To build the "th" tree, we have:

THREE	th r iy	15	12	8	0	0	0	0	0
0 00 15 12									
1 00 12 8									
2 11 3 0									
3 00 0 0									

To build the "w" tree, we have:

ONE	w ah n	18	2	10	0	0	0	0	0
0 00 18 12									
1 00 12 10									
2 11 1 0									
3 00 0 0									

To build the "z" tree, we have:

ZERO	z iy r ow	19	8	12	11	0	0	0	0
0 00 19 8									
1 00 8 12									
2 00 12 11									
3 11 0 0									
4 00 0 0									

Once all of the branches have been built, the next stage is to concatenate the tree. It appears that four elements make a page, and each element is two bytes.

														page
ey:	00	5	14	11	8	0	00	0	0	00	0	0	5
f:	01	4	0	00	6	3	00	3	17	11	5	0	6
	00	6	11	00	11	12	11	4	0	00	0	0	7

-continued

													page
n:	00	10	3	00	3	10	11	9	0	00	0	0	8
ow:	11	11	0	00	0	0	00	0	0	00	0	0	9
s:	01	6	0	00	13	4	00	4	17	00	17	7	10
	00	7	10	11	7	0	00	13	7	00	7	9	11
	00	9	13	11	6	0	00	0	0	00	0	0	12
t:	00	14	16	11	2	0	00	0	0	00	0	0	13
th:	00	15	12	00	12	8	11	3	0	00	0	0	14
w:	00	18	12	00	12	10	11	1	0	00	0	0	15
z:	00	19	8	00	8	12	00	12	11	11	0	0	16

Then, the start_lookup header is defined. Each row (page) is 8 bytes and hence the header takes 3 rows. The end_lookup header is two rows. Therefore, the lexical tree store proper starts at the 6th (#5) page.

sp	sil	ah	ay	eh	ey	f	ih	MP
0	1	2	3	4	5	6	7	MP_id
3	0	0	0	5	0	6	0	data stored.
iy	k	n	ow	r	s	t	th	
8	9	10	11	12	13	14	15	
0	0	8	9	0	10	13	14	
uw	v	w	z	.	.	.	.	
16	17	18	19	.	.	.	.	
0	0	15	16	0	0	0	0	

The end_lookup header is defined. 20 is out of bounds.

ey	f	n	ow	s	t	th	w
5	6	10	11	13	14	15	18
z	.	.	.	.	.	.	.
19	20	20	20	20	20	20	20

To put the above information all together:

START	sp:ih	3		0	0	0	5	0	6	0	0
LOOKUP	iy:th	0		0	8	9	0	10	13	14	1
	uw: v	0		0	15	16	0	0	0	0	2
END	ey:w	5		6	10	11	13	14	15	18	3
LOOKUP	z	19		20	20	20	20	20	20	20	4
LEXICAL	ey:	00	5	14	11	8	0	00	0	0	00	0	0	5
TREE	f:	01	4	0	00	6	3	00	3	17	11	5	0	6
STORE		00	6	11	00	11	12	11	4	0	00	0	0	7
	n:	00	10	3	00	3	10	11	9	0	00	0	0	8
	ow:	11	11	0	00	0	0	00	0	0	00	0	0	9
	s:	01	6	0	00	13	4	00	4	17	00	17	7	10
		00	7	10	11	7	0	00	13	7	00	7	9	11
		00	9	13	11	6	0	00	0	0	00	0	0	12
	t:	00	14	16	11	2	0	00	0	0	00	0	0	13
	th:	00	15	12	00	12	8	11	3	0	00	0	0	14
	w:	00	18	12	00	12	10	11	1	0	00	0	0	15
	z:	00	19	8	00	8	12	00	12	11	11	0	0	16
		3		0	0	0	5	0	6	0				
		0		0	8	9	0	10	13	14				
		0		0	15	16	0	0	0	0				
		5		6	10	11	13	14	15	18				
		19		20	20	20	20	20	20	20				
		00	5	14	11	8	0	00	0	0	00	0	0	
		01	4	0	00	6	3	00	3	17	11	5	0	
		00	6	11	00	11	12	11	4	0	00	0	0	
		00	10	3	00	3	10	11	9	0	00	0	0	
		11	11	0	00	0	0	00	0	0	00	0	0	
		01	6	0	00	13	4	00	4	17	00	17	7	
		00	7	10	11	7	0	00	13	7	00	7	9	
		00	9	13	11	6	0	00	0	0	00	0	0	
		00	14	16	11	2	0	00	0	0	00	0	0	

-continued

00	15	12	00	12	8	11	3	0	00	0	0
00	18	12	00	12	10	11	1	0	00	0	0
00	19	8	00	8	12	00	12	11	11	0	0

This information is converted to hexidecimal. The first five are byte fields.

The rest are:

2bit—uCode.

6bit—data.

2bit—null.

6bit—data

03	00	00	00	05	00	06	00
00	00	08	09	00	0A	0D	0E
00	00	0F	10	00	00	00	00
05	06	0A	0B	0D	0E	0F	12
13	14	14	14	14	14	14	14
05	0E	C8	00	00	00	00	00
84	00	06	03	03	11	C5	00
06	0B	0B	0C	C4	00	00	00
0A	03	03	0A	C9	09	00	00
CB	00	00	00	00	00	00	00
86	00	0D	04	04	11	11	07
07	0A	C7	00	0D	07	07	09
00	0D	C6	00	00	00	00	00
0E	10	C2	00	00	00	00	00
0F	0C	0C	08	C3	00	00	00
12	0C	0C	0A	C1	00	00	00
13	08	08	0C	0C	0B	C0	00

The above is the micro code for the digit lexical tree. It allows branching, and it holds the word_id. It knows which lexical trees it contains and where they are housed. It also stores the possible end monophone list.

FIG. **8** shows a further example of the Lexical Tree RAM, giving examples of different types of branching in the lexical tree, and how these are represented in the RAM.

FIG. **9** is a block diagram showing the data structure that is supported for the word link record, to enable word n-best decoding. The word engine maintains a record of the words that have been currently searched by the PINE array. It is used to determine the word history and allow trace back during or at the end of search. The word history is needed to determine the language model score and whether particular paths can be recombined or merged.

FIG. **10** shows a block diagram of the word engine. This is designed to allow fast access to the word history (thus allowing rapid phone creation at the phone level), and to allow reasonably fast trace back. The Word Engine is primarily a memory store that fills incrementally. The decoupling of the wordlat_node_id and the physical address will allow the possibility of being able to keep the WLRs for a given word in contiguous addresses, thus avoiding the need for the pointer field.

The language model interface is designed to allow the Phone Book to quickly obtain the appropriate LM factor for the phone model that it is currently constructing. Based on the phones in the wave front, all possible next LM factors can be determined. This will be done in software together with the language model factor engine. The ARM 9 will maintain its own record of the wavefront using the same next phone requests received by the wave front manager and its own copy of the lexical tree.

FIG. **11** shows the architecture of the LM interface. This allows the Phone book to quickly access the LM factor for the phone that is currently being generated. Prior to the phone generation phase LM_factors are computed for all possible phones that may be generated in the next frame.

The grammar engine is designed to give the Phone Book a quick response to requests for the next grammar node(s). In the case of a task grammar where the lexicon may be limited to a few words (e.g. yes, no), it may be desirable to use the same lexical tree. In this case it will be necessary for the grammar, to provide lexical tree pointers such that the phone generator can jump to the appropriate part of the lexical tree.

For a speech recognition system it is also necessary to specify the grammar. Again we will use HTK's Standard Lattice Format (SLF) to specify the grammar. In practice it will be convenient to use HTK's HParse format to specify the grammar using a high level grammar notation. This can then be converted into using the HParse tool. Consider having to specify a grammar for a digit recognition task. The HParse format for this is as follows:

```
$digit = one | two | three | four | five | six | seven | eight | nine | zero
$digit_sp = $digit sp
$noise = lipsmack | breath | sil
( [sil] < $digit_sp | $noise > [sil] )
```

where | denotes alternatives, < > denotes one or more repetitions, { } denotes zero or more repetitions, [ ] denotes that item is optional, and ( ) contains the grammar.

FIG. **12** shows the network that is defined by this grammar. The network starts and ends with a "sil" phone, representing silence. Between the start and end, a number of possible routes correspond to further silence, a breath, a lipsmack sound, or a detected word.

FIG. **13** shows the grammar engine architecture. When a next phone request arrives, the next grammar node(s) must be retrieved from memory. In the case of a task grammar, the phone book should be given the lexical tree memory addresses to allow the phone to be determined.

FIG. **14** shows a single frame of the Audio Codec (AC) link protocol used in some embodiments of the invention. The Advanced Audio CODEC Interface (AACI) is an ARM Primecell that provides an interface to off-chip AC97 audio codecs. AC97 is Intel's audio codec specification, initially created for the PC market. The AACI is an AMBA slave block that connects to the Advanced Peripheral bus (APB). The AACI connects to the off-chip codec using the AC-link protocol.

The AC-link protocol is a bi-directional, fixed clock rate, serial digital stream. AC-link handles multiple input and output PCM audio streams as well as register control accesses. It employs a time division multiplexed (TDM) scheme that divides each audio frame into 12 outgoing and 12 incoming data streams each with up to 20 bit sample resolution. The tag slots are used to indicate which of the subsequent slots contain valid data. Add and dat are the control register address and data slots for read and writes. The adc slots contain the outputs from the analog to digital converters. Assuming that the ADC sample rate is set to 16 kHz then the AC-link controller will receive 16 bits of data every third codec frame.

The AACI has 4 channels that can be operated simultaneously. They are accessed through DMA.

FIG. **15** shows the system timing. Each frame is divided into three sections:

Token update (MHD wash)

First pass—inter phone token passing

Second pass—garbage collection, max model pruning

The horizontal direction of the figure indicates elapsing time, and various subcircuits and processes are indicated along the vertical direction of the figure. The top part of the diagram shows frame count, and push to dictate button, and the lower part of the diagram shows various subcircuits and processes, within a single time frame, whilst the push to dictate button is engaged. These subcircuits and processes are language model interface, word engine, phone book, PINE array, MHD engine, mean & covar reg, and feature vector reg.

At the start of the time frame, the feature vector is generated, and the MHD values are calculated. The MHD values are then passed to the PINE array, which engages in beam pruning and token score updating. The phone book receives next phone requests and returns next phone responses. The word engine adds word link records, and the language model interface supplies language model factors.

In the second part of the frame, pruning thresholds are calculated for the PINE array, for the next frame, and the PINE array engages in token passing. The language model interface transfers language model factors for the next frame.

Whilst the "push to dictate" button remains activated, the above process repeats in a sequence of frames.

The embodiments of the invention which are now described relate to a distance engine for calculation of Mahalanobis Distances.

FIG. **16** shows a block diagram of a speech recognition system according to an embodiment of the invention. The system includes a microphone **100**, which outputs an audio signal to a combined AGC (automatic gain control) and ADC (analogue to digital converter) **101**. In alternative embodiments, the AGC and ADC may be separate. In this embodiment, the incoming data comes from a microphone, and is then converted to digital format. However, it is also possible that the digital audio data may be received from a network, or may be read from storage means such as a memory, an optical disk, or a hard disk drive.

The digital audio data output from the ADC is input to an audio data buffer **102**, and may be held here until required. The data is then output from the audio data buffer **102** to a front end circuit **103**. The incoming audio data is divided into frames, as described above. In cases where the system is extremely busy, it is possible to drop some audio frames, to allow the system to catch up with any backlog.

In the audio front end circuit **103**, the audio is split into spectral components to generate a feature vector, as described above. For example, the feature vector may have a number of frequency bands, and an energy band, plus their first and second derivatives. The feature vector is then output from the front end **103** to the distance calculation engine **104**.

The distance calculation engine **104** converts feature vectors into Mahalanobis distances. The acoustic model may be chosen from a plurality of acoustic models, e.g. according to the language used, and may be stored in a storage means **105** such as a memory, a disk drive, etc. The distances calculated by the distance calculation engine are then transferred to the search stage **106** of the speech processing circuit, which uses models such as one or more word models and/or language models to generate and output recognised text. Such word and language models may be stored in a separate storage means **107**.

In one embodiment, the distance calculation engine uses Mahalanobis Distance (MHD) calculations to determine distance values for output to the search stage **106**. In further embodiments, different types of distance calculations may be used, e.g. as described above.

In some embodiments of the invention, the distance calculation engine is designed as a speech accelerator, to operate as a loosely bound co-processor for a CPU running speech recognition software. This has the advantage of reducing computational load on the CPU, and reducing memory bandwidth load for the CPU.

In some embodiments, the accelerator interface is memory-mapped. Other types of interface are also possible to support different system configurations, for example removable accelerator cards with SDIO (secure digital input/output), CardBus, or USB interfaces.

The speech recognition apparatus may also operate with a CPU and DSP (digital signal processor), where the DSP implements the "front-end" signal processing to produce a feature vector, and the CPU implements the search stage, reading the distance results from the distance calculation engine.

The system may be implemented in any combination of hardware or software.

In some embodiments of the invention the search stage **106** provides the distance calculation engine **104** with an active state list **1**XX that indicates the subset of states for which the search stage requires distances to be calculated by the distance calculation engine **104**. This is an optimization that may reduce computation time and/or power consumption.

In preferred embodiments of the invention, the front end **103**, distance calculation engine **104** and the search stage **106** operate in a pipelined manner. When operating in a pipelined manner it is unlikely that the search stage **106** will be able to provide the active state list **1**XX early enough for the distance calculation engine **104** to implement the optimization of computing only the distances that will be required by the search stage **106**. The distance calculation circuit **104** may calculate the MHD values for every state in the lexicon, per frame, whether it is subsequently required by the search stage or not. This allows the accelerator and software system to operate in a concurrent pipelined manner, which maximizes the system throughput.

The distance calculation engine **104** may have a plurality of memory areas for storing feature vectors, to allow a feature vector for one frame to be transferred to one memory area in the distance calculation engine **104**, while another feature vector for another frame is being read from a second memory area, for calculation of a distance. The memory areas used for reading and writing may alternate with subsequent frames.

FIG. **17** shows a block diagram of a distance calculation accelerator **200**, used in an embodiment of the invention. A feature vector is received at input **212** of the accelerator **200**. The feature vector is loaded into feature vector (FV) memory A **208** via a bus **211**. In reality, there may be more than a single bus, but for simplicity, only one bus **211** is shown in the figure.

The registers **203** are discussed further at the end of this description. The registers **203** may include, for example, an accelerator status register, for indicating whether the acoustic model data is correctly stored in memory, as described in more detail below.

The FV registers **209** hold the feature vector whose distances are currently bring computed by the distance calculation engine **204**. If the distance calculation engine **204** is implemented in software, the FV registers might be implemented as memory attached to the processor running the software. If the distance calculation engine **204** is imple-

mented in hardware the FV registers might be implemented as logic or as a memory block. The FV registers **209** are loaded from the FV memory **208** when the Distance Calculation Engine **204** has completed distance calculations for the current feature vector and a subsequent feature vector is available in the FV memory **208**. In some embodiments there is an optional FIFO associated with FV registers **209** so that the FIFO is loaded from the FV memory **208** when there is a subsequent feature vector in the FV memory **208** and there is a space in the FIFO. In these embodiments the FV Registers **209** are loaded from the associated optional FIFO **209** when the Distance Calculation Engine **204** has completed calculations for the current feature vector and there is a subsequent feature vector in the associated optional FIFO **209**. The presence of the FIFO **209** provides for a looser coupling between the Front End **103** and the accelerator **200** which improves performance in some system configurations.

Each feature vector is transferred to a distance calculation engine circuit **204**, to obtain distances for each state of the acoustic model. The acoustic model is stored in an acoustic model memory **205**, which is connected to the bus **211**, for control of reading/writing to the memory. In some embodiments, this also allows acoustic models to be changed or updated via the bus. The acoustic model may be stored in compressed form in the acoustic model memory **205**. A decompressor **206** is provided on the data output of the acoustic model memory. The decompressor **206** can decompress the results as they are needed, and thus allows a much smaller size of acoustic memory to be used, than with uncompressed data. From the decompressor **206**, the decompressed acoustic model states are sent to the distance calculation engine **204**.

The distance calculation engine **204** converts feature vectors and acoustic model states to distances. The accelerator **200** includes two separate results memories for storing these calculated distances, labelled as "result memory A" **201** and "result memory B" **202**.

The control, sequencing, and address generation **210** is logic, state machines, and/or software programs that provide overall control of the other elements of the accelerator, taking inputs from the registers **203**, and all the other elements and interfaces to determine when to start and stop the Distance Calculation Engine **204**. The control **210** also produces addresses and control signals to control the reading and writing of the memories **201**, **202**, **205** and **208**, to control the flow of data between the various elements of the accelerator **200**, the operation of the Decompressor **206**, CRC check **207** and Distance Calculation Engine **204**, and to control the setting or clearing or status bits in the Registers **203** and to control signalling to the front end **103** and search stage **106**. The control **210** is implemented using any of the widely known and used methods of controlling and sequencing in digital systems including logic gates, state machines, and embedded software systems. The control **210** might be a centralized control, or it might be distributed in some or all of the other elements of the accelerator **200**, or some combination of these.

Each time a feature vector is loaded into the accelerator, the accelerator computes the distances for all states for that feature vector, and stores the results alternately in the A or B Results Memory. The results are written to one of the two memories **201**, **202**, while the other memory is available for reading. Thus, while the distance calculation engine **204** is storing the distances calculated for one audio frame, the other result memory may make available the distances calculated for an earlier audio frame, to the search stage of the speech recognition system. For non-real time processing, "earlier" audio frame may refer to an audio frame that has been processed earlier, rather than being limited to an audio frame that corresponds to speech at an earlier time.

Thus, the search stage software reads results alternatively from the A or B result memory, while the accelerator writes results into the other result memory, allowing concurrent pipelined operation of the accelerator and the CPU [and the DSP if present].

In some implementations the A and B results memories may be exposed to the software interface as separate memories appearing at different locations in the CPU and/or DSP memory map. Another implementation is to map each memory alternatively to the same range of addresses in the CPU and/or DSP memory map, so that the software always accesses results at the same location, and the memory containing the set of results the software currently needs to access is the one mapped there.

The accelerator may autonomously load and decompress an acoustic model, which is stored in a compressed format in the acoustic model memory **205**. The Acoustic Model may be loaded into the Acoustic Model Memory by software running on the CPU prior to the first use of the Accelerator. The Acoustic Model Memory may be an on-chip memory, to maximize bandwidth and minimize power consumption of reading in the entire Acoustic Model every frame, which means 50 to 100 times per second, depending on the system design.

The Acoustic Model Memory may be a dedicated memory attached to, or incorporated within, the accelerator. Alternatively the Acoustic Model Memory may be a shared resource that is used by other system functions or applications when speech recognition is not running. To allow detection of situations where the acoustic model has been overwritten by another application, a pre-calculated CRC (cyclic redundancy check) signature may be stored with the acoustic model. Each time the acoustic model is read into the accelerator, a CRC may be computed and compared with the signature. If the comparison fails, an error bit is set in the accelerator status register.

Some embodiments of the invention relate to speech recognition systems comprising separate front end **103** and search stage **106** processors. These separate processors perform the front-end and search stages of the recognition process, with the accelerator **200** performing the distance calculations. Typically the front-end processing would be performed on a DSP and search stage would be performed on a CPU, although this description holds for any type of processors, even dedicated hardware implementations. As shown in FIG. **17**, the front end **103** supplies the distance accelerator with feature vectors, and the search stage **106** uses distances calculated by the accelerator.

The accelerator is designed to operate in a concurrent pipelined manner with the other processors, allowing a high level of parallelism in the system.

FIG. **18** shows how this pipelined operation works in an embodiment of the invention with ideal data flow, i.e. where each of the three recognition stages require exactly the same time to process each data frame, and the processors are never delayed, stalled, or diverted to other tasks.

The time axis is divided into equal "pipeline step times". If the step time is the same as a frame time then the system is processing audio in real time. If the step time is short than a frame time, the system is operating faster than real time, which may be valuable for batch processing of audio data, or catching up after some type of delay. If the step time is loner than a frame time, the system is operating slower than real time, which may be acceptable in some applications, for

example where the person dictating pauses occasionally allowing the system to "catch up".

Once the pipeline has filled (which takes two step times), the DSP, the Accelerator, and the CPU are all operating in parallel. In this idealized example, the total time available for processing a frame is three step times—one step time in the DSP, one step time in the accelerator, and one step time in the search stage. There is a latency of three step times from the time the audio data enters the front end to the completion of the search stage for that frame, and the processing of one frame finishes each step time. If the system were not pipelined, then the processing of all stages for one frame would have to complete in one step time in order to achieve the same overall throughput of frames per second. Thus pipelining in this way can achieve three times the throughput with the same processing elements, compared to a non-pipelined system. A non-pipelined system would be one in which the processing of frame n+1 would not start until all stages of the processing of frame n was already complete.

If, for example, in a system running in real time with a frame time and step time of $1/100^{th}$ of a second, which is 10 ms, each of the three processing elements has the entire 10 ms available to complete the processing of one stage of the recognition task, giving 30 ms of processing time overall for each frame. If the system were not pipelined, real time operation would require that all three processing elements would be complete in 10 ms, allowing only 3.3 ms for each of the three processing elements.

At the end of each pipeline step time, the DSP provides a feature vector to the Accelerator, and the accelerator provides a set of distance results to the CPU. When the CPU is performing the search stage for frame n, the Accelerator is computing the distances for frame n+1, and the DSP is performing the front-end stage for frame n+2. During each pipeline step, the front end processing consumes one frame of digital audio data (although this is not indicated explicitly on the diagram).

Real world systems are more complex that the idealized system in many ways. The processing time for a frame is often highly variable and data dependent, especially in the search stage, and the processing time for each processing stage is likely to be quite different. The DSP and CPU are likely from time to time to be diverted to run other software tasks, either to service interrupts, or under control of an operating system scheduling regime. Communications between the processing elements may be stalled due to competing bus traffic, for example for DMA transfers.

Recognition software may implement "frame dropping" and other advanced techniques, and as a result not all frames are processed through all of the recognition stages. Processing elements cannot respond instantly to the availability of data from the preceding processing stage, resulting in delays.

Finite buffer space for information storage between stages means that an earlier stage may be stalled waiting for a space to store its output data when following processing stages are "falling behind".

The accelerator has been carefully designed to mitigate these problems. FIG. **19** shows a pipelined system with real world complications added. The following text describes the interactions shown in the diagram. The time for the processing of various stages has been varied to show difference examples of buffering of data between stages and stalling of processing elements. The numbering corresponds to the numbered bubbles on the diagram. The text also makes reference to memories used to pass information into and out of the accelerator, which are shown in FIG. **17**.

The front end has computed the feature vector for the first frame, FV**1**. It is loaded into the accelerator, causing the accelerator to commence computation of the distances for the frame.

The feature vector for the second frame, FV**2**, is passed to the accelerator. The accelerator is still busy computing the distances for the first frame, so this feature vector is held in the accelerators FV memory.

The front end has computed the feature vector for the third frame, FV**3**, but it cannot be delivered to the accelerator as the FV memory is still full, holding FV**2**. The time period labeled "stall delivering FV**3**" is the period between FV**3** being computed and when it can first be delivered. Of course the DSP could be diverted to some other useful task, or it could buffer FV**3** in its own memory and commence computation of FV**4**.

The arrow labeled "FV**3** earliest" shows the earliest time that the feature vector FV**3** can be delivered to the accelerator, which is immediately after the accelerator loads FV**2** into its internal registers and frees the memory.

The arrow labeled "FV**3** latest" shows the latest time that the feature vector FV**3** can be delivered to the accelerator without causing the accelerator to stall for lack of a feature vector to work on. This example shows that there is a wide tolerance in the system to when FV**3** can be delivered without affecting system performance—an entire accelerator frame processing time. The accelerator provides an interrupt and a status bit that can be polled to indicate the state of the FV memory. Regardless of which control method the DSP uses, the system tolerates long latency without reducing performance.

The accelerator signals to the search stage that the distances for frame **1** are available in result memory A. The search stage commences processing for frame **1**, reading distances from result memory A.

The search stage has finished its processing for frame **1** before the accelerator has completed computing the distances for frame **2**, so the search stage stalls waiting for the accelerator. Of course the CPU could be diverted to other useful tasks, including any search stage tasks that can be completed before distances are available. Again, an interrupt and status bits are provided by the accelerator to control this interaction between the accelerator and the CPU.

The accelerator completes the distance calculation for frame **2**, and signals to the CPU that it can commence reading the distances from result memory B.

The accelerator has completed the computation of distances for frame **3** before the search stage has finished processing frame **2**. As soon as the search stage signals to the accelerator that it has completed frame **2** by releasing memory B, the accelerator signals that the distances for frame **3** are available, allowing the search stage to immediately move on to processing stage **3**.

The accelerator is stalled waiting for feature vector FV**4**, due to the long processing time in the front end. In real systems this may be caused by a number of factors, including the DSP being diverted to another task for a period of time, or simply due to the DSP performance being insufficient to keep up with the accelerator's throughput.

The accelerator restarts as soon as FV**4** is delivered by the front end.

The search stage is stalled waiting for distances for frame **4**, a follow on consequence of the late delivery of FV**4** to the accelerator.

FV**5** is delivered to the accelerator while it is still computing the distances for the previous frame, allowing it to move onto processing frame **5** without stalling when it completes frame **4**.

The accelerator signals that the distances for frame **4** are available and the search stage commences processing frame **4**.

FV**6** is delivered to the accelerator while it is still computing the distances for frame **5**.

The availability of distances for frame **5** is signaled by the accelerator to the CPU.

Even though the accelerator has already received FV**6**, it is stalled waiting for a free result memory it can write the distances into.

The search stage releases result memory B at the end of its processing of frame **4**. This allows the accelerator to start computing distances for frame **6**.

The accelerator completes computing distances for frame **6** and signals to the CPU that the results are ready.

Further embodiments of the invention relate to systems with a Front End and Search Stage on a single processor. This single processor performs the front-end and search stages of the recognition process, with the accelerator performing the distance calculations. This configuration is likely to be the most common configuration where the accelerator is added to an embedded system such as a PDA or mobile phone that has a high-performance embedded application processor (but no DSP) available to run speech recognition software.

In this configuration the accelerator still operates in a concurrent pipelined manner with the front-end and search stages; it is just that the front-end and search stages are running on a single processor.

CPUs in embedded systems are generally relatively simple compared to desktop PCs and servers. Embedded CPUs in mobile phones, PDAs, in-car navigation and telematics systems, and other consumer electronics devices are typically "simple uni-processors" by which we mean not multi-processors and not supporting hardware multi-threading support—i.e. they can only execute a single thread of instructions (although they can be diverted to a different thread under the control of interrupts, exceptions, or an operating system). For this discussion, except where explicitly stated otherwise, we are talking about embedded processors that are "simple uni-processors". We are also using the term "CPU" and "processor" interchangeably.

FIG. **20** shows how the system can be pipelined with a single embedded processor working with the accelerator, in an idealized system. In this idealized system, the time taken for the processor to perform the front-end and search stage processing is identical to the time the accelerator takes to perform the distance calculations, and the processor is never delayed, stalled, or diverted to other tasks.

The time axis is divided into equal "pipeline step" times. If the step time is the same as a frame time then the system is processing audio in real time. If the step time is short than a frame time, the system is operating faster than real time, which may be valuable for batch processing of audio data, or catching up after some type of delay. If the step time is loner than a frame time, the system is operating slower than real time, which may be acceptable in some applications, for example where the person dictating pauses occasionally allowing the system to "catch up".

The first two pipeline step times are used to fill the pipeline. By the third pipeline step time, the pipeline is full, and all three stages of the recognition process are processed every pipeline step time, with each stage working on the processing of a different data frame.

During the first pipeline step time, the CPU completes the front-end processing for frame **1**, and the feature vector passed to the accelerator so that distance calculations can start in the accelerator.

During the second pipeline step time, the front-end processing for frame **2** is completed by the CPU, in parallel with the accelerator computing the distances for frame **1**.

During the third pipeline step time, the CPU runs the search stage for frame **1** reading distance results from the accelerator, and then runs the front-end processing for frame **3**. The accelerator runs in parallel computing the distances for frame **2** using the feature vectors computed in the previous pipeline step.

Each subsequent pipeline step time is similar to the third pipeline step, with the frame numbers incremented. So for pipeline step n=4, 5, 6, etc. the CPU runs the search stage for frame n−2 and then runs the front-end for frame n. In parallel the accelerator computes the distances for frame n−1.

Note that with this situation the production of feature vectors and the consumptions of results run in lock step—one feature vector is produced every time the search stage consumes one frame of distance results. This considerably reduces the need for elastic buffering between the distance accelerator and the software system.

Again, real world systems are more complex than the idealized system in many ways. For example, the processing time for a frame is often highly variable and data dependent, especially in the search stage, and the processing time for each processing stage is likely to be quite different. The CPU is likely from time to time to be diverted to run other software tasks, either to service interrupts, or under control of an operating system scheduling regime. Communications between the CPU and accelerator may be stalled due to competing bus traffic, for example for DMA transfers.

Recognition software may implement "frame dropping" and other advanced techniques, and as a result not all frames are processed through all of the recognition stages. Processing elements cannot respond instantly to the availability of data from the preceding processing stage, resulting in delays

Finite buffer space for information storage between stages means that an earlier stage may be stalled waiting for a space to store its output data when following processing stages are "falling behind".

The accelerator has been carefully designed to mitigate these problems. FIG. **21** shows a pipelined system with real world complications added. The following text describes the interactions shown in the diagram. The time for the processing of various stages has been varied to show difference examples of buffering of data between stages and stalling of processing elements. The numbering corresponds to the numbered bubbles on the diagram. The text also makes reference to memories used to pass information into and out of the accelerator, which are shown in FIG. **21**.

At the end of front-end processing for frame **1**, the CPU passes the feature vectors for frame **1** to the accelerator, allowing distance calculations to start.

To fill the pipeline, the CPU immediately performs the front-end processing for frame **2** and passes the feature vectors to the accelerator. They are stored in the accelerator so that the accelerator can load them into its internal engine for processing as soon as the distance calculations for frame **1** are complete.

The CPU is now ready to start the search stage for frame **1**, however the distance computation in the accelerator has not completed. The box labeled "Prep search stage **1**" indicates a period where the CPU could do any preparatory search stage processing that does not require distances. Alternatively the CPU could be diverted to another task, or it can stall waiting for the distance results.

The accelerator signals that the distance results for frame **1** are now available. The search stage can commence on the

CPU, reading distance results from the accelerator's Result Memory A. The accelerator automatically starts processing the feature vector for frame **2** that was delivered earlier.

The accelerator signals to the CPU that it has completed the distance calculations for frame **2**. The CPU is still busy running the search stage for frame **1**, and has still to run the front end for frame **3**, so the completion signal is ignored until the CPU has need of it, when it starts running the search stage for frame **2**. As the accelerator has not received the feature vector for frame **3**, it has no task it can perform and so it stalls.

The CPU delivers the feature vector for frame **3**, restarting the accelerator, which writes results into Result Memory A. Note that Result Memory A was previously "released" at the end of the processing of search stage **1**, informing the accelerator that it can overwrite the old results held in that memory with new results. The CPU then runs the search stage for frame **2**, reading distance results from the accelerator's Result Memory B.

The accelerator signals to the CPU that the distance results for frame **3** are complete in Result Memory A. Again, the CPU ignores this signal until it is ready to commence the search stage for frame **3**, and the accelerator stalls waiting for the next feature vector. The CPU delivers the feature vector for frame **4**, restarting the accelerator. It them starts search stage **3** reading distances from the accelerator's result memory A.

Search stage **3** and front end **5** were completed quickly, so that the feature vector for frame **5** is delivered to the accelerator while it is still running. The CPU performs all the preparatory work it can for the search stage, and then stalls waiting for distances to be available.

The accelerator signals that the distances for frame **4** are available. The CPU starts search stage **4**, reading the distances from the accelerator's result memory B. The accelerator automatically starts computing distances for frame **5** using the feature vector delivered earlier.

Note that with the pipelining scheme just presented which alternates front-end and search stage processing, the accelerator can never stall waiting for a result memory to be released because the feature vector the accelerator needs as an input is not produced until after the memory has been cleared. The CPU task schedule could be described with the shorthand F**1** F**2** S**1** F**3** S**2** F**4** S**3** F**5** S**4**.

It can seen readily from FIG. **21** and the accompanying description above, that with this CPU task schedule, that the operation of the CPU and accelerator are tightly coupled and whenever the processing time for a pipeline step on the CPU is unequal to the processing time on the accelerator, then it quickly results in one of them stalling.

In a system where either the CPU or the accelerator is always (or at least in most instances) slower than the other in completing its processing task, stalling is inevitable and throughput may not be improved by adding additional buffering (e.g. additional result memories or FIFO depth on the feature vector interface). [Unless there are other difficulties to mitigate such as very low-bandwidth buses linking the CPU to the accelerator, or competing bus traffic such as DMA transfers that can significantly reduce performance.

If the processing time for a pipeline step is highly variable on either the CPU or Accelerator, such that there are regular changes in which one of them takes the longer time, then significant enhancements in throughput may be obtained by adding additional buffering between them. This may have the disadvantage of adding additional latency through the system. This buffering may be implemented by adding additional depth to the Feature Vector FIFO, and/or adding additional Result Memories. Alternatively it may be by implementing elastic buffers in software.

If the accelerator has sufficient buffering on the feature vector interface, e.g. by including the optional FIFO shown in FIG. **21**, then increasing the pipeline depth by computing extra front end frames up front will improve throughput at a cost of extra latency. This will result in a CPU task schedule of F**1** F**2** F**3** S**1** F**4** S**2** F**5** S**3** F**6** S**4** or F**1** F**2** F**3** F**4** S**1** F**5** S**2** F**6** S**3** F**7** S**4**, etc.

Of course it would also be possible to dynamically schedule whether the CPU should run the front-end or search stage code based on the availability, or not, of distance results and/or the space available to store more feature vectors and distance results. Given the fundamental relationship that one set of distance results is produced by the accelerator for every feature vector it consumes, and the restriction stated at the start of this section that we are discussing single-threaded CPUs, dynamic scheduling is likely to add little performance improvement while being complex to implement.

On a CPU supporting multi-threaded operation (which is a technique for improving performance in the presence of high memory system latency), or on a multi-processor, the highest system performance is likely to be achieved by running the front-end and search stages concurrently as separate threads, rather than alternating between the front-end and search stages. The system operation is then very similar to that described above, if each thread is thought of as a separate virtual processor. In some embodiments of the invention, elastic buffers may be used between stages, to accommodate varying time delays between the processing of frames at each stage. Elastic buffers in the interfaces between the three recognition stages may also significantly enhance performance of such a system.

Operation of the accelerator and the software system is not constrained to a rigid repetitive frame time. Instead the timing of the system is controlled dynamically by the availability of feature vectors into the accelerator, and the consumption of results out of the accelerator.

This allows the recognition process to fall behind real-time operation and to "catch up" again. It also allows the system to be robust in complex systems where interrupts, DMA bus traffic, memory caches, operating system task switching, and competing real-time software tasks all make guaranteeing rigid timing constraints impractical.

Elastic buffers at the major data transfer points between processing elements maximize system performance by allowing one element to continue useful processing whenever it can, regardless of whether other processing elements have been stalled or diverted to other tasks. Elastic buffers are useful for the following applications:

(1) For buffering raw audio data arriving in real time from the microphone(s) The data would typically be moved into the buffer by a DMA and/or high priority interrupt driven software process to ensure that hard real time constraints are met.

(2) For buffering feature vectors output from the front end. This is especially true if the front-end is implemented on a DSP or other processing element independent of the search stage.

(3) For buffering distance results from the accelerator. Again this is especially true if the front-end and search stages are implemented on independent processing elements. This buffering is also especially valuable if the interface between the search stage processor and the accelerator is very slow, has high latency, or is subject to competing bus traffic.

The Feature Vector and Results interfaces on the accelerator already provide a level of elasticity that should be suffi-

cient in systems where the Front End and Search stages are running on the same CPU, and the accelerator has a high-bandwidth interface to the CPU with low level of competition from other bus traffic such as DMA.

Other embodiments may have deeper FIFOs. In one embodiment, the two result memories are an implementation of a FIFO where the implementation is exposed across the interface. It would be possible to have additional memories and use them in a round-robin fashion, or to use another implementation of a FIFO that hides the FIFO depth by only exposing one result memory at a time to the interface. An example of this would be to map each memory in turn to a memory range that is always used to read results, so that at any one time the "current" result memory is mapped to that location. This would hide the FIFO depth behind the interface so that the pipeline depth can be changed without changing the software (or hardware) that accesses the interface.

Slow or Conflicted Bus Interfaces may occur. In the system configurations described above, the search stage reads distances directly from the Accelerator's Result Memories. The result memories allow the search stage to randomly access the distance results, as they are needed. This will only give a high performance system if the interface or bus between the CPU and Accelerator provides the CPU with low latency, high bandwidth access to the Accelerator Result Memories.

In some systems it is not practical to provide such a low latency, high bandwidth interface between the CPU and the Accelerator. This may be for many reasons, including:

- The interface is a bus shared with other devices and/or processors that generate bus traffic that keeps the bus busy such that the CPU access to the accelerator is delayed waiting for the bus to be free. DMA transfers are one example of conflicting bus traffic.
- The interface is implemented as a serial bus, or narrow parallel bus, for cost or compatibility reasons, yielding low bandwidth and high latency
- The interface crosses an industry standard interface such as PCI, USB, SDIO, Cardbus, Firewire, PCIExpress etc. that introduces protocol overheads and/or latency as would be the case where the accelerator is in a plug-in peripheral to a system containing a CPU (or CPU and DSP).

In these situations it is possible to add another pipeline stage during which the result data is copied from the Accelerator into another memory (which we will call a "buffer memory") to which the CPU does have high bandwidth, low latency access. This copying can be performed by a DMA controller, or by software running as an interrupt handler, or a separate software thread on the CPU, or a software thread on another processor, or by other commonly known means. The device controlling the transfer may be on the CPU side of the interface or on the accelerator side of the interface, depending on the capabilities of the interface. This extra pipeline stage then allows an entire pipeline step time to copy the data from the accelerator to the other memory. The data copy can optionally be performed as a sequential burst, or a number of sequential bursts, to optimize transfers across the interface between the Accelerator and the buffer memory. The search stage, running on the CPU, can then randomly access the results in the buffer memory with low latency.

Similarly an extra pipeline stage can be added for transferring the Feature Vectors to the Accelerator, although given that the feature vectors are a much smaller object than the distance results, and that they are not accessed randomly across the interface but delivered once to the Accelerator for each frame, there are fewer situations where the performance benefit from this extra pipeline stage is as significant.

FIGS. **22** and **23** show idealized systems with the two extra pipeline stages for transfers added. The first shows a system with a CPU, DSP, and Accelerator with the two extra pipeline stages. The second shows a system with a CPU and DSP with the two pipeline stages added.

Other embodiments may add additional pipeline stages so that the distance computations are performed over two or more pipeline stages rather than being restricted to a single pipeline stage. This allows more time for the computation of the distance results for each frame, while maintaining the same throughput of distance results. This is similar in concept to the well-known pipelining techniques used to implement RISC microprocessors.

The Acoustic Models may occupy many megabytes of storage space. In embedded systems they will typically held in a non-volatile memory such as FLASH memory, or other types of non-volatile memory as is the appropriate currently economic technology. The Accelerator may read the acoustic model directly from the non-volatile memory, or more likely the acoustic model will be copied from the non-volatile memory into a higher bandwidth RAM memory of some type (e.g. SRAM or DRAM, SDRAM, etc.). The RAM memory may be part of the same integrated circuit as the Accelerator (i.e. "on chip"), or attached to the integrated circuit containing the Accelerator. ("off chip"). The having the RAM memory on-chip is particularly advantageous, as this configuration would give the lowest power consumption and highest bandwidth. An alternative solution would be for the acoustic model to be stored on a disk drive, or an another device accessible via a wired or wireless bus or network, and for the acoustic model to be loaded into the RAM memory from the disk drive, or over the wired or wireless bus or network.

On-chip RAM memory is however likely to come at a cost-premium to off-chip commodity memory devices. Many embedded systems face extreme cost pressures in the market place, especially high volume devices supplied by many manufacturers such as mobile phones. For a speech recognizer to be incorporated into such systems, the overall recognizer solution much be delivered at very low cost. There is therefore a significant economic benefit from reducing the cost of the RAM memory and the non-volatile memory. One way to reduce the cost of these memories is to compress the acoustic models and store the compressed model in the memory.

The Accelerator computes all of the distances associated with each Feature Vector that is delivered to it by the front end. This means that the accelerator must use the data from the entire Acoustic Model each time a new Feature Vector is presented to the Accelerator (with the exception of some specific optimizations that may determine that some data values need not be used in the distance computations). It is therefore possible to arrange for the accelerator to read through the acoustic data sequentially once for each Feature Vector, using each data value as it read from the Acoustic Model.

Reading through the entire Acoustic Model sequentially is ideally suited to placing a decompressor between the distance calculation engine and the RAM memory, because many compression schemes depend on the decompressor reading sequentially through the compressed data, and do not readily support random access into the compressed data.

Thus this configuration allows storing compressed data in the RAM memory (and the non-volatile memory), which can therefore be smaller and lower cost. There is no requirement

for a memory that is large enough to hold the entire decompressed acoustic model, as the decompressed data is consumed as it is decompressed.

The decompression scheme could be any combination of:

sign or zero extension or otherwise conversion of narrow or variable width data to a wider data format.

sign or zero extension or otherwise conversion of narrow or variable width data to IEEE standard single or double precision floating point format

codebook decompression of a binary bitstream, where the codebook is stored as part of the acoustic model data

decompression of a Huffman or Lempel-Ziv compressed stream

decompression of run length encoded data

decompression of difference encoded data

any well known compression scheme.

Additionally the model size can be reduced by using subspace distribution clustering and the decompressor and distance calculation engine designed to operate with such models.

The following description provides further information on the visible registers in some embodiments of the invention.

The register bank is designed to allow simple connection with an 8-bit data bus. Accesses to the register bank should be made with byte-read and byte-write accesses. To facilitate this, only the least significant 8 bits of the data bus are used.

The address decoder should respond so to byte accesses to byte **0** or byte **3** within the designated word. This way the same code can be used on an ARM processor regardless of whether the processor is configured for little endian or big endian operation.

REGBASE=Base address of register bank

Register **0** Product Version

8-bit Product version ID

This register value is changed when the functionality of the hardware charges in a way that changes the hardware/software interface.

Address=REGBASE (and REGBASE+3 for big endian)

Read only. Reads with value 0x01

Writes ignored

Register **1** Implementation Version

8-bit Implementation version ID

This register value is changed for each implementation to facilitate bug tracking and version control.

Address=REGBASE+4 (and REGBASE+4+3 for big endian)

Read only. Reads with value 0x01

Writes ignored

Register **2** Interrupt Enable

8-bit Interrupt Enable register

Address=REGBASE+4*2 (and REGBASE+4*2+3 for big endian)

Readable and writeable.

Value following reset is 0 for bits **0**, **1**, **2**, **3**

Bit **0** I 1=Enable FV memory empty interrupt on Int0 output.

Bit **1** J 1=Enable Results complete interrupt on Int0 output.

Bit **2** K 1=Enable FV memory empty interrupt on Int1 output.

Bit **3** L 1=Enable Results complete interrupt on Int1 output.

Bit **4** Future use. Read value is not defined. Writes ignored.

Bit **5** Future use. Read value is not defined. Writes ignored.

Bit **6** Future use. Read value is not defined. Writes ignored.

Bit **7** Future use. Read value is not defined. Writes ignored.

Software should use a read-modify-write code sequence when setting this register so that future use bits are written with the value read from them.

Register **3** Status

Address=REGBASE+4*3 (and REGBASE+4*3+3 for big endian)

Read-only register. Writes are ignored.

Value following reset is 0 for bits **0**, **1**, **2**, **3**

Bit **0** FVMB Feature Vector Memory is Busy

0=OK to write new value to FV memory (FV memory is empty)

1=Don't write new value to FV memory (FV memory is busy)

Bit **1** CRM Current Result Memory

0=Result memory A

1=Result memory B

This bit toggles every time the Release Result Memory (RRM) command is issued to the accelerator (register **3**). It indicates which memory software should read results from, and which result memory the RRM command should release next time it is issued. Note: Software should also check that results are available in the indicated result memory by checking the value of the MA and MB bits.

Bit **2** MA 1=Results are available is result memory A

The MA bit is set to 1 when the compute engine completes writing all of the distances for a frame into result memory A. This value is set to zero when the Release Result Memory command is issued and CRM=0.

Bit **3** MB 1=Results are available is result memory B

The MB bit is set to 1 when the compute engine completes writing all of the distances for a frame into result memory B. This value is set to zero when the Release Result Memory command is issued and CRM=1.

Bit **4** Dfault 1=Acoustic data CRC fault. Indicates that the CRC value at the end of the acoustic data did not match the CRC value computed while the acoustic data was read in.

Bit **5** Future use. Value undefined.

Bit **6** Future use. Value undefined.

Bit **7** Future use. Value undefined.

Register **4** Command

Address=REGBASE+4*4 (and REGBASE+4*4+3 for big endian)

Write-only register.

Software should perform byte writes to this register with a single bit set to 1 and all other bits set to zero to issue a command to the accelerator.

Behaviour is not defined if more than one command is issued within a single write (i.e. if more than 1 bit is a 1).

Value on read is not defined.

Bit **0** LFV Load Feature Vector from the FV memory.

Tells the accelerator that a new feature vector has been placed into FV memory ready for use by the accelerator. The FV will be loaded into the compute engine when the compute engine completes its current task (or immediately if the compute engine is idle). Computation of distances for this FV will commence when the FV is loaded AND there is a released result memory. (At reset, both result memories are released). The results for the first, third, fifth etc. feature vectors loaded are written to Result Memory A. The results for the second, fourth, sixth etc. feature vectors loaded are written to Result Memory B.

Bit **1** RRM Release Result memory. Indicates that the results in the current result memory can now be overwritten with new results. The current result memory is indicated by the CRM bit in the status register.

Bit **2** CFVI Clear the FV memory empty interrupt

Bit **3** CRI Clear the Results Complete interrupt

Bit **4** Future use—write 0

Bit **5** Future use—write 0

Bit **6** Future use—write 0

Bit **7** Future use—write 0

The present invention may be included in a mobile phone, or in another consumer electronics device, a PC, Laptop or tablet PC, in-car electronics system, or plug-in card for an electronic device.

The methods and apparatus according to the invention are not limited to embedded mobile systems, but may be used with any type of computing device, including PCs, workstations, laptop computers, notebook computers, handheld computing devices, dictation machines, PDAs (personal digital assistants), personal organisers, mobile phones, games consoles, video players, etc. Embedded computers according to the present invention may be embedded in a wide range of apparatus and machinery, such as electronic entertainment products, domestic appliances, factory machinery, vehicles, etc.

Embodiments of the present invention may be implemented as an IP core in a System on Chip (SOC) or a discrete device. Multiple chips or bare die may be connected to provide a larger vocabulary system.

Although the apparatus may include all or some of an input for accepting audio input, a circuit for digitising the audio input, a circuit for dividing the audio input into a plurality of time-frames, and a circuit for analysing the spectral components of each time frame to characterise the audio in that frame, this is not essential. The apparatus may also include a circuit for calculating a score indicating the probability or likelihood that each audio frame corresponds to each of a plurality of different possible states, but again, this is not essential. Instead, the apparatus may include an input for receiving the pre-calculated scores.

In the described embodiments, n-gram language models are used. For example, the Bigram language model is based on outputting the probability that a given second word directly follows a given first word. However, it is possible to use alternative language models instead. It is possible to use a mixture of different models. For example, it is possible to run a word-dependent n-best decode applying a bigram language model, and the resulting word link record or word graph could then be rescored using a trigram language model.

The Viterbi beam search algorithm may be used in embodiments of the invention, to perform the search. A word level search may be performed. Dummy states may be used for the start and end of an utterance

The grammar level model may be used to add distance elements into the likelihood calculation. This is not essential, but tends to give much better results.

The HMM models described in the embodiments comprise self connections to each states and single connections to a neighbouring state. However, the present invention is not limited to such HMM models, and for example, HMM models with additional connections between states may also be used. Alternatively, embodiments of the invention may use alternative types of acoustic models, rather than HMM models.

The system may comprise a plurality of language models (e.g. English, French). A model of words may comprise actual words, phrases, silence, lipsmack sounds, and/or other sounds for recognition by the apparatus.

As previously discussed, the phone level has been found to be the best level for lexical tree searching, rather than the state level or the word level. However, this invention is not limited solely to phone level searching.

Pipelining may comprises processing data in different parts of a circuit at the same time, i.e. parallel processing.

In the following claims, the calculation means may comprise the distance calculation engine described above.

Further embodiments of the invention include the methods carried out in the accelerator, and the methods carried out in the speech recognition circuit.

While the invention has been described in terms of what are at present its preferred embodiments, it will be apparent to those skilled in the art that various changes can be made to the preferred embodiments without departing from the scope of the invention, which is defined by the claims.

The invention claimed is:

1. A speech recognition circuit, comprising:

an audio front end for calculating a feature vector from an audio signal, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame;

a calculating circuit for calculating distances indicating the similarity between a feature vector and a plurality of predetermined acoustic states of an acoustic model; and

a search stage for using said calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words;

wherein said audio front end and said search stage are implemented using a first processor, and said calculating circuit is implemented using a second processor, and wherein data is pipelined from the front end to the calculating circuit to the search stage.

2. A speech recognition circuit as claimed in claim **1**, wherein the pipelining comprises alternating of front end and search stage processing on the first processor.

3. A speech recognition circuit as claimed in claim **1**, comprising dynamic scheduling whether the first processor should run the front end or search stage code, based on availability or unavailability of distance results and/or availability of space for storing more feature vectors and/or distance results.

4. A speech recognition circuit as claimed in claim **1**, wherein the first processor supports multi-threaded operation, and runs the search stage and front ends as separate threads.

5. A speech recognition circuit as claimed in claim **1**, wherein the said calculating circuit is configured to autonomously calculate distances for every acoustic state defined by the acoustic model.

6. The speech recognition circuit of claim **1**, comprising control means adapted to implement frame dropping, to discard one or more audio time frames.

7. The speech recognition circuit of claim **1**, wherein the feature vector comprises a plurality of spectral components of an audio signal for a predetermined time frame.

8. The speech recognition circuit of claim **1**, wherein the processor is configured to divert to another task if the data flow stalls.

9. The speech recognition circuit of claim **1**, wherein the speech accelerator has an interrupt signal to inform the front end that the accelerator is ready to receive a next feature vector from the front end.

10. The speech recognition circuit of claim **1**, wherein the accelerator signals to the search stage when the distances for a new frame are available in a result memory.

11. The speech recognition circuit of claim **1**, comprising increasing the pipeline depth by computing extra front frames in advance.

12. The speech recognition circuit of claim **1**, wherein the audio front end is configured to input a digital audio signal.

13. A speech recognition circuit of claim **1**, wherein said distance comprises a Mahalanobis distance.

14. A speech recognition circuit, comprising:

an audio front end for calculating a feature vector from an audio signal, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame;

calculating means for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model; and

a search stage for using said calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words;

wherein said audio front end, said calculating means, and said search stage are connected to each other to enable pipelined data flow.

15. A speech recognition method, comprising:

calculating a feature vector from an audio signal using an audio front end, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame;

calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model using a calculating circuit; and

using a search stage to identify words within a lexical tree using said calculated distances, the lexical tree comprising a model of words;

wherein data is pipelined from the front end, to the calculating circuit, and to the search stage.

16. A non-transitory storage medium storing processor implementable code for controlling at least one processor to implement a speech recognition method, the code comprising:

code for controlling the processor to calculate a feature vector from an audio signal, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame;

code for controlling the processor to calculate a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model; and

code for controlling the processor to identify words within a lexical tree using said calculated distances, the lexical tree comprising a model of words,

wherein data is pipelined by the processor pursuant to the code from the feature calculation, to the distance calculation, and to the word identification.

* * * * *

Exhibit B

US007587319B2

(12) **United States Patent**
Catchpole

(10) **Patent No.: US 7,587,319 B2**
(45) **Date of Patent: Sep. 8, 2009**

(54) **SPEECH RECOGNITION CIRCUIT USING PARALLEL PROCESSORS**

(75) Inventor: **Mark Catchpole**, Cambridgeshire (GB)

(73) Assignee: **Zentian Limited**, Cambridge (GB)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 823 days.

(21) Appl. No.: **10/503,463**

(22) PCT Filed: **Feb. 4, 2003**

(86) PCT No.: **PCT/GB03/00459**

§ 371 (c)(1),
(2), (4) Date: **May 24, 2005**

(87) PCT Pub. No.: **WO03/067572**

PCT Pub. Date: **Aug. 14, 2003**

(65) **Prior Publication Data**

US 2005/0251390 A1 Nov. 10, 2005

(30) **Foreign Application Priority Data**

Feb. 4, 2002 (GB) 0202546.8

(51) **Int. Cl.**
G10L 15/00 (2006.01)

(52) **U.S. Cl.** **704/235**; 704/242; 704/243; 704/258

(58) **Field of Classification Search** 704/235, 704/242, 243, 256, 258
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,293,213 A 3/1994 Klein et al.
5,832,428 A * 11/1998 Chow et al. 704/254

(Continued)

FOREIGN PATENT DOCUMENTS

GB 2 112 194 A 7/1983

(Continued)

OTHER PUBLICATIONS

S. Glinski et al., "Spoken Language Recognition on a DSP Array Processor", IEEE Transactions on Parallel and Distributed Systems, Jul. 5, 1994, No. 7, New York, USA, pp. 697-703.

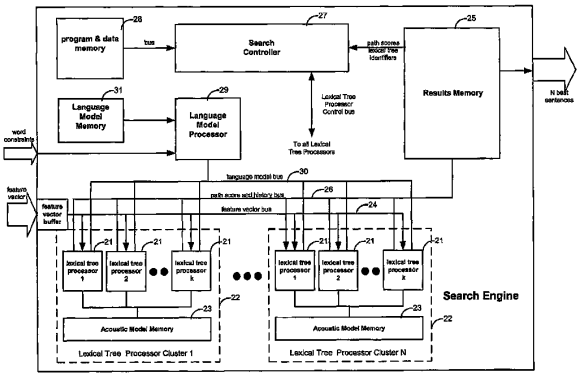
(Continued)

Primary Examiner—Daniel D Abebe
(74) *Attorney, Agent, or Firm*—Dickstein Shapiro LLP

(57) **ABSTRACT**

A speech recognition circuit comprises an input buffer for receiving processed speech parameters. A lexical memory contains lexical data for word recognition. The lexical data comprises a plurality of lexical tree data structures. Each lexical tree data structure comprises a model of words having common prefix components. An initial component of each lexical tree structure is unique. A plurality of lexical tree processors are connected in parallel to the input buffer for processing the speech parameters in parallel to perform parallel lexical tree processing for word recognition by accessing the lexical data in the lexical memory. A results memory is connected to the lexical tree processors for storing processing results from the lexical tree processors and lexical tree identifiers to identify lexical trees to be processed by the lexical tree processors. A controller controls the lexical tree processors to process lexical trees identified in the results memory by performing parallel processing on a plurality of said lexical tree data structures.

69 Claims, 6 Drawing Sheets



U.S. PATENT DOCUMENTS

5,881,312	A	3/1999	Dulong	
5,995,930	A *	11/1999	Hab-Umbach et al.	704/254
6,047,283	A *	4/2000	Braun	707/3
7,035,802	B1 *	4/2006	Rigazio et al.	704/256
7,120,582	B1 *	10/2006	Young et al.	704/255
7,174,037	B2	2/2007	Arnone et al.	
2001/0011218	A1 *	8/2001	Phillips et al.	704/256
2003/0178584	A1	9/2003	Arnone et al.	
2008/0255839	A1	10/2008	Larri et al.	

FOREIGN PATENT DOCUMENTS

GB	2 331 392	A	5/1999
GB	2 347 835	A	9/2000
GB	2 380 920	A	4/2003
WO	WO 00/50859		8/2000
WO	WO 03/067572	A2	8/2003

OTHER PUBLICATIONS

S. Chatterjee et al., "Connected Speech Recognition on a Multiple Processor Pipeline", ICASSP 89, May 23, 1989, Glasgow, UK, pp. 774-777.

S-H. Chung et al., "A Parallel Phoneme Recognition Algorithm Based on Continuous Hidden Markov Model", Proceedings of 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, 1999, Los Alamitos, CA, USA, pp. 453-457.

S-H. Chung et al., "A Parallel Computational Model for Integrated Speech and Natural Language Understanding", IEEE Transactions on Computers, New York, USA, vol. 42, No. 10, Oct. 1, 1993, pp. 1171-1183.

N. Deshmukh et al., "Hierarchical Search for Large-Vocabulary Conversational Speech Recognition: Working Toward a Solution for the Decoding Problem", IEEE Signal Processing Magazine, vol. 16, No. 5, Sep. 1999, pp. 84-107.

R.M. Woodward et al., "Tissue Classification Using Terahertz Pulsed Imaging", Proceedings of the SPIE—The International Society for Optical Engineering SPIE-INT. Soc. Opt. Eng. USA, vol. 5318, No. 1, 2004, pp. 23-33. Abstract also attached.

V.P. Wallace et al., "Biomedical Applications of THz Imaging", Microwave Symposium Digest, 2004, IEEE MTT-S International Fort Worth, TX, USA, Jun. 6-11, 2004, Piscataway, NJ, USA, IEEE, vol. 3, Jun. 6, 2004, pp. 1579-1581.

V.P. Wallace et al., "Biomedical Applications of Terahertz Pulse Imaging", Second Joint EMBS-BMES Conference 2002. Conference Proceedings. 24th Annual International Conference of the Engineering in Medicinal and Biology Society. Annual Fall Meeting of the Biomedical Engineering Society, Houston, TX, Oct. 23-26, 2002, Annual vol. 1 of 3. Conf. 24, Oct. 23, 2002, pp. 2333-2334.

A.J. Fitzgerald et al., "Terahertz Imaging of Breast Cancer, a Feasibility Study", Conference Digest of the 2004 Joint 29th International Conference on Infrared and Millimeter Waves and 12th International Conference on Terahertz Electronics IEEE Piscataway, NJ, USA, 2004, pp. 823-824.

B.E. Cole et al., "Terahertz Imaging and Spectroscopy of Human Skin, In-vivo", Proceedings of SPIE, vol. 4276, pp. 1-10.

* cited by examiner

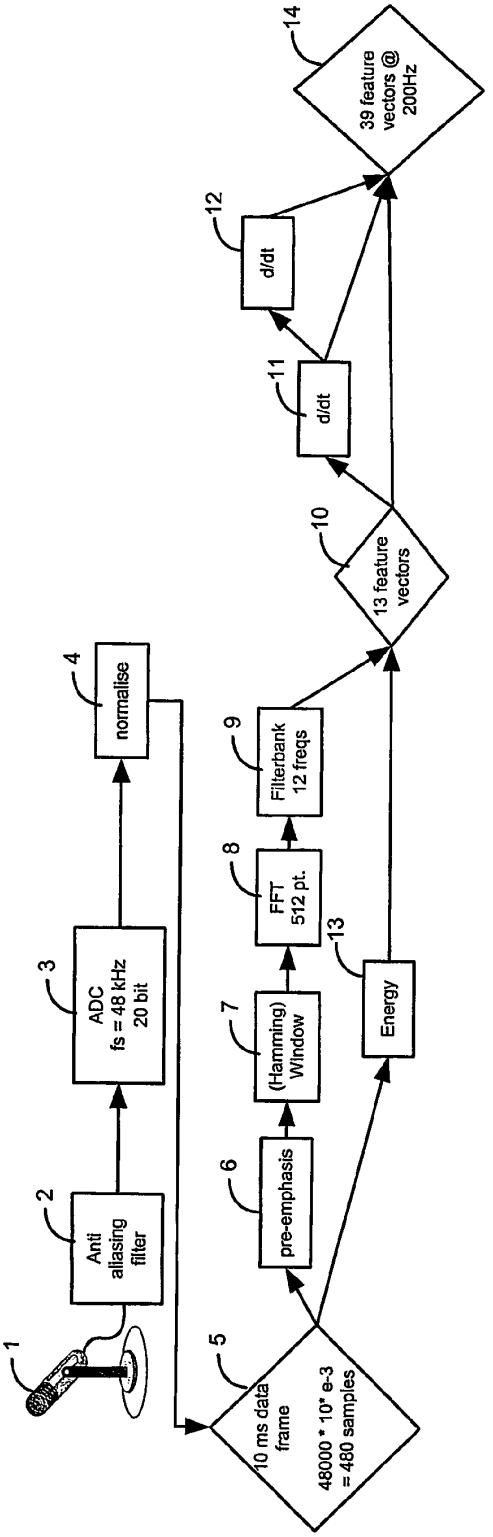

1
2
Anti aliasing filter
3
ADC fs = 48 kHz 20 bit
4
normalise
5
10 ms data frame 48000 * 10* e-3 = 480 samples
6
pre-emphasis
7
(Hamming) Window
8
FFT 512 pt.
9
Filterbank 12 freqs
10
13 feature vectors
11
d/dt
12
d/dt
13
Energy
14
39 feature vectors @ 200Hz

Fig 1

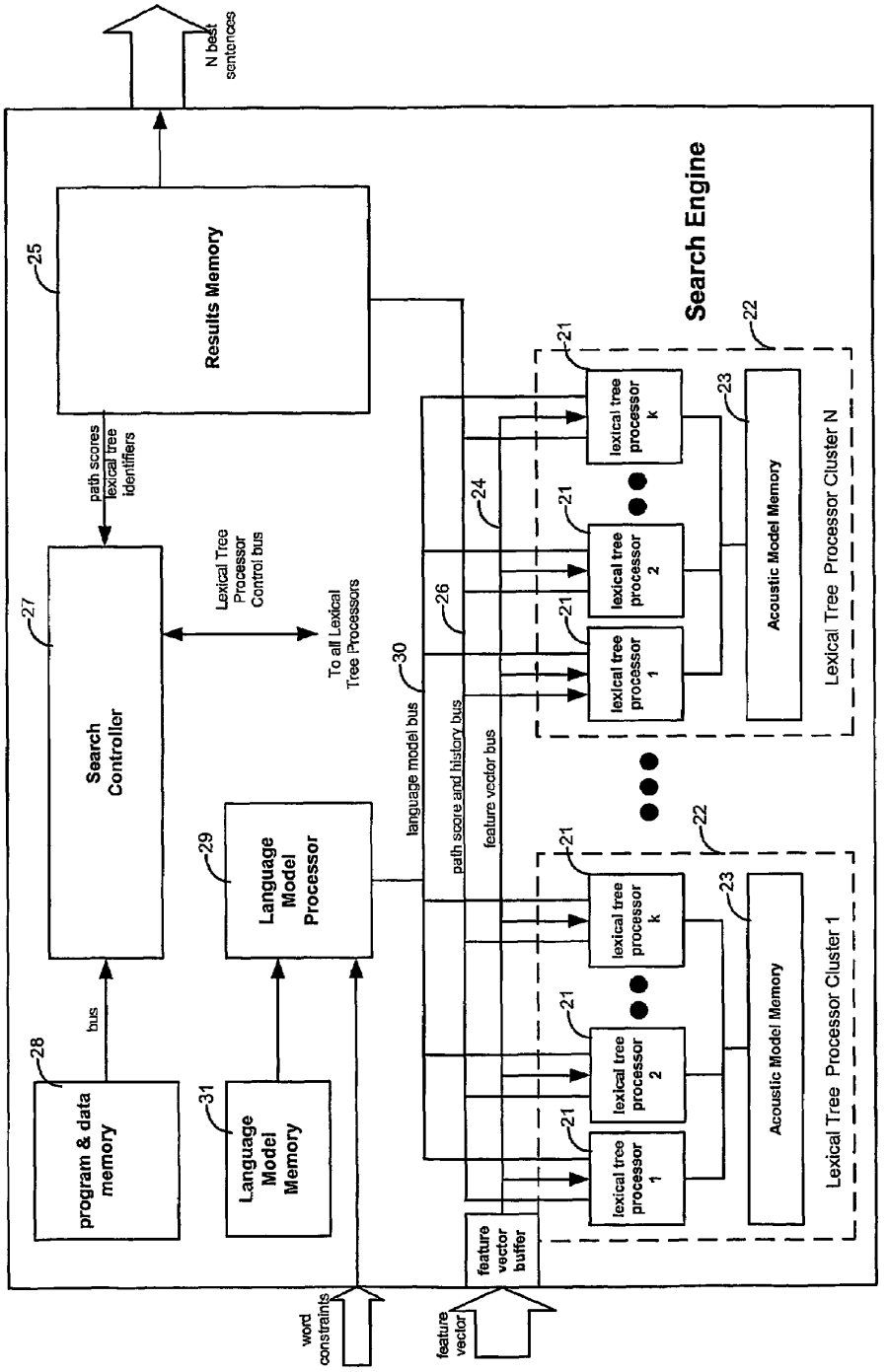

N best sentences
Search Engine
Results Memory
25
path scores lexical tree identifiers
Search Controller
27
Lexical Tree Processor Control bus
To all Lexical Tree Processors
Language Model Processor
29
language model bus
30
path score and history bus
feature vector bus
24
26
bus
program & data memory
28
Language Model Memory
31
feature vector buffer
word constraints
feature vector
lexical tree processor k
lexical tree processor 2
lexical tree processor 1
21
Acoustic Model Memory
23
Lexical Tree Processor Cluster N
Lexical Tree Processor Cluster 1
22

Fig 2

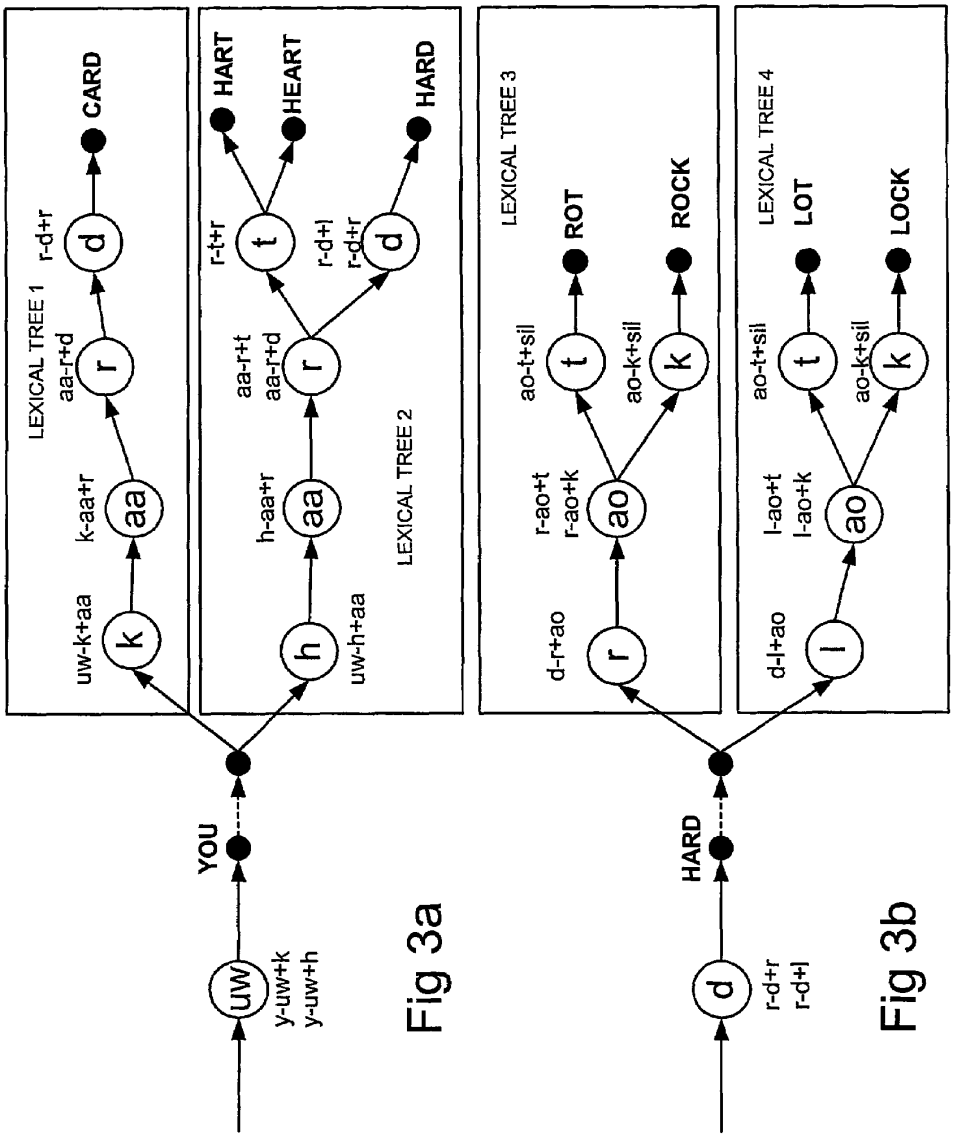

LEXICAL TREE 1
uw-k+aa
k-aa+r
aa-r+d
r-d+r
CARD
LEXICAL TREE 2
uw-h+aa
h-aa+r
aa-r+t
aa-r+d
r-t+r
r-d+|
r-d+r
HART
HEART
HARD
YOU
uw
y-uw+k
y-uw+h

Fig 3a

LEXICAL TREE 3
d-r+ao
r-ao+t
r-ao+k
ao-t+sil
ao-k+sil
ROT
ROCK
LEXICAL TREE 4
d-l+ao
l-ao+t
l-ao+k
ao-t+sil
ao-k+sil
LOT
LOCK
HARD
d
r-d+r
r-d+|

Fig 3b

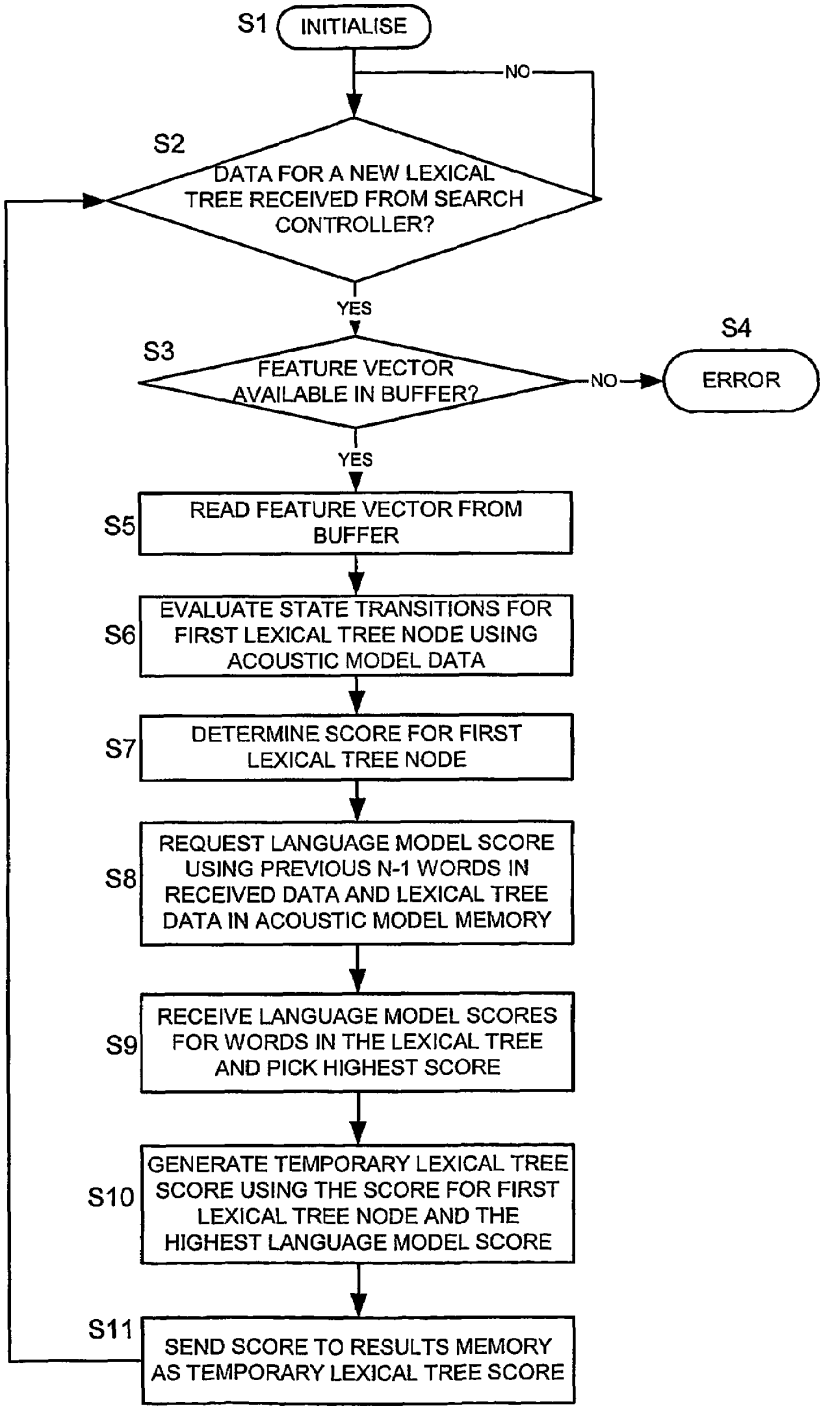

S1
INITIALISE
NO
S2
DATA FOR A NEW LEXICAL TREE RECEIVED FROM SEARCH CONTROLLER?
YES
S3
FEATURE VECTOR AVAILABLE IN BUFFER?
NO
S4
ERROR
YES
S5
READ FEATURE VECTOR FROM BUFFER
S6
EVALUATE STATE TRANSITIONS FOR FIRST LEXICAL TREE NODE USING ACOUSTIC MODEL DATA
S7
DETERMINE SCORE FOR FIRST LEXICAL TREE NODE
S8
REQUEST LANGUAGE MODEL SCORE USING PREVIOUS N-1 WORDS IN RECEIVED DATA AND LEXICAL TREE DATA IN ACOUSTIC MODEL MEMORY
S9
RECEIVE LANGUAGE MODEL SCORES FOR WORDS IN THE LEXICAL TREE AND PICK HIGHEST SCORE
S10
GENERATE TEMPORARY LEXICAL TREE SCORE USING THE SCORE FOR FIRST LEXICAL TREE NODE AND THE HIGHEST LANGUAGE MODEL SCORE
S11
SEND SCORE TO RESULTS MEMORY AS TEMPORARY LEXICAL TREE SCORE

Fig 4

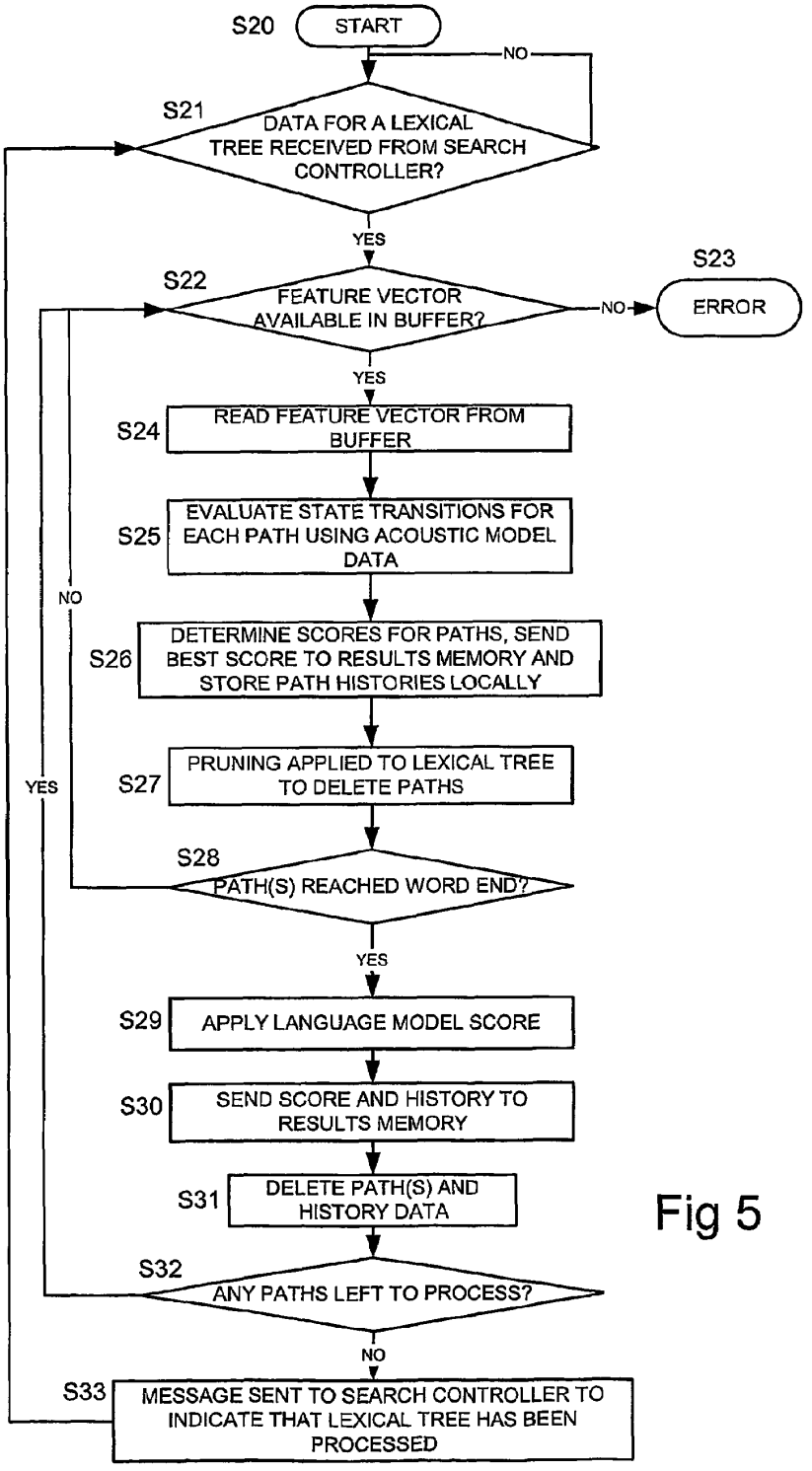

S20
START
NO
S21
DATA FOR A LEXICAL TREE RECEIVED FROM SEARCH CONTROLLER?
YES
S22
FEATURE VECTOR AVAILABLE IN BUFFER?
NO
S23
ERROR
YES
S24
READ FEATURE VECTOR FROM BUFFER
S25
EVALUATE STATE TRANSITIONS FOR EACH PATH USING ACOUSTIC MODEL DATA
NO
S26
DETERMINE SCORES FOR PATHS, SEND BEST SCORE TO RESULTS MEMORY AND STORE PATH HISTORIES LOCALLY
S27
PRUNING APPLIED TO LEXICAL TREE TO DELETE PATHS
YES
S28
PATH(S) REACHED WORD END?
YES
S29
APPLY LANGUAGE MODEL SCORE
S30
SEND SCORE AND HISTORY TO RESULTS MEMORY
S31
DELETE PATH(S) AND HISTORY DATA
S32
ANY PATHS LEFT TO PROCESS?
NO
S33
MESSAGE SENT TO SEARCH CONTROLLER TO INDICATE THAT LEXICAL TREE HAS BEEN PROCESSED

Fig 5

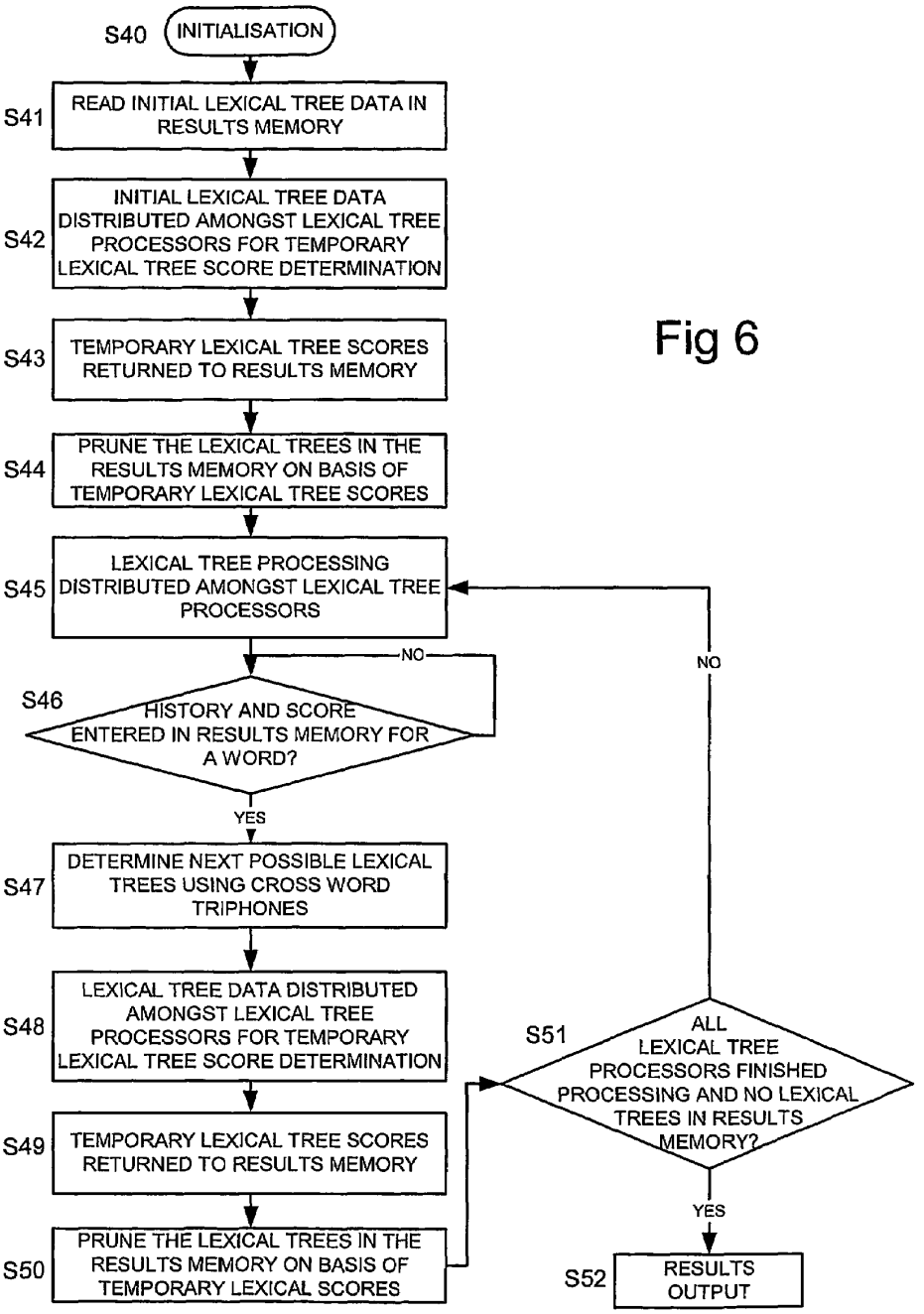

S40
INITIALISATION
S41
READ INITIAL LEXICAL TREE DATA IN RESULTS MEMORY
S42
INITIAL LEXICAL TREE DATA DISTRIBUTED AMONGST LEXICAL TREE PROCESSORS FOR TEMPORARY LEXICAL TREE SCORE DETERMINATION
S43
TEMPORARY LEXICAL TREE SCORES RETURNED TO RESULTS MEMORY
S44
PRUNE THE LEXICAL TREES IN THE RESULTS MEMORY ON BASIS OF TEMPORARY LEXICAL TREE SCORES
S45
LEXICAL TREE PROCESSING DISTRIBUTED AMONGST LEXICAL TREE PROCESSORS
S46
HISTORY AND SCORE ENTERED IN RESULTS MEMORY FOR A WORD?
NO
YES
S47
DETERMINE NEXT POSSIBLE LEXICAL TREES USING CROSS WORD TRIPHONES
S48
LEXICAL TREE DATA DISTRIBUTED AMONGST LEXICAL TREE PROCESSORS FOR TEMPORARY LEXICAL TREE SCORE DETERMINATION
S49
TEMPORARY LEXICAL TREE SCORES RETURNED TO RESULTS MEMORY
S50
PRUNE THE LEXICAL TREES IN THE RESULTS MEMORY ON BASIS OF TEMPORARY LEXICAL SCORES
S51
ALL LEXICAL TREE PROCESSORS FINISHED PROCESSING AND NO LEXICAL TREES IN RESULTS MEMORY?
NO
YES
S52
RESULTS OUTPUT

Fig 6

SPEECH RECOGNITION CIRCUIT USING PARALLEL PROCESSORS

The present invention generally relates to a speech recognition circuit which uses parallel processors for processing the input speech data in parallel.

Conventional large vocabulary speech recognition can be divided into two processes: front end processing to generate processed speech parameters such as feature vectors, followed by a search process which attempts to find the most likely set of words spoken from a given vocabulary (lexicon).

The front end processing generally represents no problem for current processing systems. However, for large vocabulary, speaker independent speech recognition, it is the search process that presents the biggest challenge. An article by Deshmukh et al entitled "Hierarchical Search for Large-Vocabulary Conversational Speech Recognition" (IEEE Signal Processing Magazine, September 1999, pages 84 to 107), the content of which is hereby incorporated by reference, discusses the general concepts of large vocabulary speech recognition. As discussed in this paper, one algorithm for performing the search is the Viterbi algorithm. The Viterbi algorithm is a parallel or breadth first search through a transition network of states of Hidden Markov Models. An acoustic model for words in a lexicon are represented as states of Hidden Markov Models. These states represent phones or n phones in a phone model of the words. The search requires the evaluation of possible word matches. It is known that such a search is computationally intensive.

In order to speed up the processing performed during such a search in a speech recognition system, parallel processing has been explored. In an article by M K Ravishankar entitled "Parallel Implementation of Fast Beam Search for Speaker-Independent Continuous Speech Recognition" (Indian Institute of Science, Bangalor, India, Jul. 16, 1993) a multi-threaded implementation of a fast beam search algorithm is disclosed. The multi-threading implementation requires a significant amount of communication and synchronization among threads. In an MSC project report by R Dujari entitled "Parallel Viterbi Search Algorithm for Speech Recognition" (MIT, February 1992) the parallel processing of input speech parameters is disclosed in which a lexical network is split statically among processors.

It is an object of the present invention to provide an improved circuit which can perform parallel processing of speech parameters.

In accordance with a first embodiment of the present invention, a speech recognition circuit comprises an input port such as input buffer for receiving parameterized speech data such as feature vectors. A lexical memory arrangement is provided which contains lexicon data for word recognition. The lexical data comprises a plurality of lexical tree data structures representing a plurality of lexical trees. Each lexical tree data structure comprises a model of words having common prefix components and an initial component which is unique as an initial component for lexical trees. A plurality of lexical tree processors are connected in parallel to the input port and perform parallel lexical tree processing for word recognition by accessing the lexical data in the lexical memory arrangement. A results memory arrangement is connected to the lexical tree processors for storing processing results from the lexical tree processors and lexical tree identifiers to identify lexical trees to be processed by the lexical tree processors. A controller controls the lexical tree processors to process lexical trees identified in the results memory arrangement by performing parallel processing of a plurality of lexical tree data structures.

Thus in accordance with this embodiment of the present invention, the processing in order to perform word recognition is distributed across the processors by controlling the processors to perform processing on different lexical trees. The controller controls the processor by the processes to provide for efficient process management by distributing lexical processing to appropriate processors.

The lexical tree data structure can comprise a phone model of words, wherein the components comprise phones. For reduced storage, the lexical tree data structure can comprise a mono phone lexical tree. The mono phone lexical tree can be used to generate context dependent phone models dynamically. This enables the use of context dependent phone models for matching and hence increased accuracy whilst not increasing memory requirements. Alternatively, the lexical tree data structure can comprise context dependent phone models.

The processing performed by each processor in one embodiment comprises the comparison of the speech parameters with the lexical data, e.g. phone models or data derived from the lexical data (e.g. dynamically generated context dependent phone models) to identify words as a word recognition event and to send information identifying the identified words to the results memory as the processing results. In this embodiment a language model processor arrangement can be provided for providing a language model output for modifying the processing results at a word recognition event by a lexical tree processor. The modification can either take place at each lexical tree processor, or at the language model processing arrangement.

In one embodiment each lexical tree processor determines an output score for words in the processing results at word recognition events. Thus in this embodiment the language model processing arrangement can modify the score using a score for a language model for n preceding words, where n is an integer.

In one embodiment the controller instructs a lexical tree processor to process a lexical tree by passing a lexical tree identifier for the lexical tree and history data for a recognition path associated with the lexical tree from the results memory. The history data preferably includes an accumulated score for the recognition path. This enables a score to be determined based on the score for the recognition path to accumulate a new score during recognition carried out using the lexical tree data structure. The scores can be output in the processing results to the results memory during the processing of the speech parameters so that the scores can be used for pruning.

In one embodiment of the present invention, each lexical tree processor operates on more than one lexical tree at the same time, e.g. two lexical trees represented by two different lexical tree data structures, or two lexical trees represented by the same data structure but displaced in time (which can be termed to instances of the same lexical tree).

At word recognition events, the controller determines new lexical tree identifiers for storing in the results memory for words identified in the results memory for respective word events. In order to reduce the processing, the controller can prune the new lexical tree identifiers to reduce the number of lexical trees which are required to be processed. This pruning can be achieved using context dependant n phones to reduce the number of possible next phones. The number can be further reduced by using a language model look ahead technique.

In one embodiment of the present invention, the lexical tree processors are arranged in groups or clusters. The lexical memory arrangement comprises a plurality of partial lexical memories. Each partial lexical memory is connected to one of

the groups of lexical tree processors and contains part of the lexical data. Thus a group of lexical tree processors and a partial lexical memory form a cluster. Each lexical tree processor is operative to process the speech parameters using a partial lexical memory and the controller controls each lexical tree processor to process a lexical tree corresponding to partial lexical data in a corresponding partial lexical memory.

In another embodiment of the present invention the lexical memory arrangement comprises a plurality of partial lexical memories. Each partial lexical memory being connected to one of the lexical tree processors and containing part of the lexical data. Each lexical tree processor processes the speech parameters using a corresponding partial lexical memory and the controller is operative to control each lexical tree processor to process a lexical tree corresponding to partial lexical data in a corresponding partial lexical memory.

In one embodiment of the present invention the lexical memory arrangement stores the lexical tree data structures as Hidden Markov Models and the lexical tree processors are operative to perform the Viterbi search algorithm using each respective lexical tree data structure. Thus in this way, this embodiment of the present invention provides a parallel Viterbi lexical tree search process for speech recognition.

The first aspect of the present invention is a special purpose circuit built for performing the speech recognition search process in which there are a plurality of processors for performing parallel lexical tree processing on individual lexical tree processors.

In another aspect of the present invention a speech recognition circuit comprises an input port such as an input buffer for receiving parameterized speech data such as feature vectors. A plurality of lexical memories are provided which contain in combination complete lexical data for word recognition. Each lexical memory contains part of the complete lexical data. A plurality of processors are provided connected in parallel to the input port for processing the speech parameters in parallel. The processors are arranged in groups in which each group is connected to a corresponding lexical memory to form a cluster. A controller controls each processor to process the speech parameters using partial lexical data read from a corresponding lexical memory. The results of processing the speech parameters are output from the processors as recognition data.

Thus this aspect of the present invention provides a circuit in which speech recognition processing is performed in parallel by groups of processors operating in parallel in which each group accesses a common memory of lexical data. This aspect of the present invention provides the advantage of parallel processing of speech parameters and benefits from a limited segmentation of the lexical data. By providing a plurality of processors in a group with a common memory, flexibility in the processing is provided without being bandwidth limited by the interface to the memory that would occur if only a single memory were used for all processors. The arrangement is more flexible than the parallel processing arrangement in which each,processor only has access to its own local memory and requires fewer memory interfaces (i.e. chip pins). Each processor within a group can access the same lexical data as any other processor in the group. The controller can thus control the parallel processing of input speech parameters in a more flexible manner. For example, it allows more than one processor to process input speech parameters using the same lexical data in a lexical memory. This is because the lexical data is segmented into domains which are accessible by multiple processors.

In a preferred embodiment this aspect of the present invention is used in combination with the first aspect of the present invention. In such an arrangement each processor performs lexical tree processing and the lexical data stored in each lexical memory comprises lexical tree data structures which each comprise a model of words having common prefix components and an initial component that is unique.

In preferred embodiments of the second aspect of the present invention, the preferred embodiments of the first aspect of the present invention are incorporated.

Embodiments of the present invention will now be described with reference to the accompanying drawings in which:

FIG. **1** is a diagram of a speech data processing circuit for generating parameterized speech data (feature vectors);

FIG. **2** is a diagram of a speech recognition circuit in accordance with an embodiment of the present invention;

FIGS. **3***a* and **3***b* are schematic diagrams illustrating lexical tree structures;

FIG. **4** is a flow diagram illustrating the process performed by a lexical tree processor to determine a temporary lexical tree score in accordance with an embodiment of the present invention;

FIG. **5** is a flow diagram illustrating the process performed by the lexical tree processor for processing the input feature vectors in accordance with an embodiment of the present invention; and

FIG. **6** is a flow diagram illustrating the process performed by the controller in accordance with an embodiment of the present invention.

FIG. **1** illustrates a typical circuit for the parameterization of input speech data. In this embodiment the parameters generated are speech vectors.

A microphone **1** records speech in an analogue form and this is input through an anti-aliasing filter **2** to an analogue-to-digital converter **3** which samples the speech at 48 kHz at 20 bits per sample. The digitized output signal is normalized (**4**) to generated a 10 millisecond data frame every 5 milliseconds with 5 milliseconds overlap (**5**). A pre-emphasis operation **6** is applied to the data followed by a hamming window **7**. The data is then fast Fourier transformed (FFT) using a 512 point fast Fourier transform (**8**) before being filtered by filter bank **9** into 12 frequencies. The energy in the data frame **5** is also recorded (**13**) as an additional feature and together with the 12 frequency outputs of the filter bank **9**, 13 feature vectors (**10**) are thus produced and these are output as part of the 39 feature vectors **14**. First and second derivatives (**11** and **12**) are taken of the 13 feature vectors **10** to complete the generation of the 39 feature vectors **14**.

The arrangement illustrated in FIG. **1** is purely given for illustration. The present invention encompasses any means by which speech and data can be parameterized to a suitable form for input to the search process as will be described in more detail hereinafter.

FIG. **2** is a schematic diagram of a speech recognition circuit in accordance with an embodiment of the present invention for performing the search process. The parameterized speech data, which in this embodiment comprise feature vectors are input to a feature vector buffer **20**. The feature vector buffer **20** is provided to buffer the incoming feature vectors to allow lexical tree processors **21** to read and process the feature vectors in the buffer **20** via a feature vector bus **24**. A plurality k of lexical tree processors **21** are arranged in a respective lexical tree processor cluster **22**. Each lexical tree processor cluster **22** has an acoustic model memory **23** in which is stored lexical data for use by the lexical tree processors **21** within the lexical tree processor cluster **22**. Each lexical tree processor **21** in the lexical tree processor cluster **22** are connected to the acoustic model memory **23** within the

lexical tree processor **22**. There are N lexical tree processor clusters and thus there are Nk lexical tree processors **21** connected by the feature vector bus **24** to the feature vector buffer **20**. Each lexical tree processor **21** is capable of processing a different lexical tree and thus Nk lexical trees can be processed in parallel. The acoustic model memories **23** store as a whole a complete set of lexical data, i.e. lexical tree data structures for use in the lexical tree processing by the lexical tree processors **21**. Each acoustic model memory **23** contains part or a segment of the lexical tree data. Since lexical tree processors **21** in a lexical tree processor cluster **22** access the same acoustic model memory **23**, it is possible for more than one lexical tree processor **21** to process the same lexical data. This provides for some degree of flexibility in the controlling of the processing by the lexical tree processors **21**. Further, the acoustic model memories **23** need not contain only one copy of the lexical data. It is possible to build in a redundancy in the data to further enhance the flexibility. This avoids any bottleneck in the processing due to the search processing focusing on a small number of lexical trees.

A results memory **25** is provided for storing processing results from the lexical tree processors **21** which are received over the path score and history bus **26**. The results memory **25** also stores information on lexical trees to identify which lexical trees are to be processed. A search controller **27** is provided to control the processing performed by the lexical tree processors **21** in dependence upon a program and data stored in program and data memory **28**. The search controller reads the path scores and lexical tree identifiers from the results memory and controls the lexical tree processors accordingly. A language model processor **29** is provided which is connected to each lexical tree processor **21** by a language model bus **30**. The language model processor **29** accesses a language model memory **31** to read language model data for provision to lexical tree processors **21** in response to language model data requests. External control of the language model memory **31** is provided by a word constrains input. The language model processor **29** determines a score for a word occurring following N previous words using N grams. When a lexical tree processor requires a language model score a request is sent to the language model processor **29** over the language model bus **30** identifying the current word and the N−1 previous words. A language model score for the N gram can be returned to the lexical tree processor **21** for the modification of the score at the end of a branch of lexical tree processing. The lexical tree processor can modify the score in accordance with the language model and output a score to the results memory **25** for a word at the end of a branch of the lexical tree processing. Thus the results memory stores the results as an ordered list of scores for words together with their histories.

The results memory **25** stores the following data:

1. initial lexical tree data. This comprises pointers to an initial set of lexical trees. No history data is associated with the initial set of lexical trees. The initial set of lexical trees is predetermined and stored in the results memory **25** based on the most likely initial phones of an utterance. This initial lexical tree data is required to initialize the search process.
2. History data for search results. This comprises a record of a recognition path through the lexical tree recognition process performed by the lexical tree processors **21**. The history data includes the current word, the previous N−1 words, the current accumulated score, the phone history (for use in the determination of likely next lexical trees using cross word context dependent tri-phones), and an identifier or pointer to the lexical tree used for identifying the word.
3. Best scores for best paths being processed by each lexical tree processor **21**. This information enables the search controller **27** to monitor the processing being performed by lexical tree processors **21** to determine whether a global pruning strategy should be applied in order to reassign processing performed by a lexical tree processor if its best score for its best path is below a threshold or well below the best scores for the paths being processed by other lexical tree processors **21**.
4. Temporary lexical tree scores. These comprise tree scores which are determined as temporary scores to prune the next lexical trees to be processed at word ends. The temporary lexical tree scores include lexical tree identifiers or pointers to identify the next lexical trees to be processed. The scores enable the pruning of this list.
5. Pruning threshold. This can be a global threshold value for use in the pruning of the lexical trees globally, or a local threshold value for use by a lexical processor for locally pruning the processing performed by the lexical processor **21**.

The acoustic model memory **23** stores a Hidden Markov Model for acoustically modelling words as lexical trees. The acoustic model memory **23** stores a plurality of lexical tree data structures. Each lexical tree data structure comprises an n phone model of a number of words having common prefix phones. The first node of the lexical tree (the root) comprises a common n phone to all words in the lexical tree and uniquely identifies the lexical tree.

Each lexical tree processor **21** includes on-board memory (or local memory) to be used during the lexical tree processing. This working memory has to store all of the parameters currently working on including current scores for all paths being processed within the lexical tree, and previous N−1 words for the lexical tree. The local storage of the previous N−1 words enables the lexical tree processor **21**, when a word end is reached along a branch of the lexical tree, to send a request for the language model score for an N gram by sending the identity of the N−1 previous words together with the word identified at the end of the branch.

FIG. **3***a* and **3***b* schematically illustrate lexical trees which can be processed by the lexical tree processor **21** during the recognition of the two words HARD ROCK. In FIG. **3***a* a previous lexical tree terminated at a branch recognizing the word YOU and terminating with the mono phone uw which is associated with two context dependent tri-phones y−uw+k and y−uw+h. Thus the context dependent tri-phone associated with the last phone in the lexical tree word model for YOU indicates two possible next lexical trees, i.e. the lexical trees beginning with the mono phone k and h and having a left context phone of uw. As can be see in FIG. **3***a* this word end YOU therefore leads to two possible next lexical trees. These two lexical trees are traversed in parallel by two different lexical tree processors accumulating scores for matching of input feature vectors into Hidden Markov Models of the context dependent tri-phones associated with each node in the tree. When the end of the branch is reached, a word end event is reached and a word is recognized. As can be seen in FIG. **3***a*, in this example since four words are of similar phone length, it is possible for the search strategy based on inputting feature vectors in parallel to simultaneously reach a number of possible word ends. These possible word ends are sent as processing results to the results memory. The results memory stores the accumulated score at the word end together with phone history to identify the last phone and its associated context dependent tri-phones. In this example, considering the branch where the word HARD is recognized, the last phone is d which has associated with it two context dependent

tri-phones r−d+1 and r−d+r. Thus the search controller **27** can identify next possible lexical trees using the next phone in the context dependent tri-phone. In this case, as can be seen in FIG. **3***b*, the next possible lexical trees begin with the phones r and **1** and have associated with them context dependent tri-phones d−r+ao and d−1+ao respectively. Thus these are the next lexical trees that require processing following the end node or word event detected in the processing lexical tree **2**. FIG. **3***b* thus represents the processing that is required at the end of FIG. **3***a* at the word node for HARD in processing the second lexical tree.

As can be seen in FIG. **3***b*, the two lexical trees are processed in parallel by traversing through the branches to reach word ends by sequentially entering in feature vectors in parallel to lexical tree processors processing the respective lexical trees. When the word end is reached, the accumulated score is output to the results memory as described before. In this case since it is the last word, the important context dependent tri-phone associated with the final phone has silence (sil) as the following phone. The final phone can in fact have 50 context dependent tri-phones associated with it if there are 50 possible next phones (including silence). FIG. **3***b* only illustrates the relevant one for the end of the utterance.

FIG. **4** is a flow diagram illustrating the processing performed by a lexical tree processor **21** in order to determine a temporary lexical tree score for a lexical tree.

When a word end is identified by a lexical tree processor, the processing results are sent to the results memory **25**. The results memory identifies the last phone of the recognized words and thus enables the search controller **27** to identify possible next lexical trees using context dependent tri-phones as illustrated in FIG. **3***b*. Although in theory if there are 50 phones there are 50^2 possible lexical trees, due to the use of the context dependant triphones, only 100-200 lexical trees are likely to be identified as possible lexical trees. The search controller needs to further prune out the lexical trees which are unlikely to generate likely paths with high scores. In order to do this, instructions are sent to the lexical tree processors to determine a temporary lexical tree score which can be used to prune out lexical trees from the processing which have a low score.

The processing by the lexical tree processor to generate the temporary lexical tree score is shown in FIG. **4** and for this process, following initialization, a lexical tree processor **21** awaits receipt of data for a new lexical tree from the search controller **27** (step S**2**). The data comprises a lexical tree pointer on an identifier to lexical tree data stored in the acoustic model memory **23**, i.e. the lexical tree data structure, previous N−1 words (for use in language model score determination by the language model processor **29**), and the current path score. The previous N−1 words includes the previously recognized word for which a score has been accumulated.

The lexical tree processor **21** then attempts to read the next feature vector from the feature vector buffer **20** (step S**3**) and if this is not available, an error occurs (step S**4**). If the feature vector is available in the feature vector buffer **20**, the lexical tree processor **21** reads the feature vector from the buffer **20** (step S**5**) and evaluates the state transitions for the first lexical tree node using the acoustic model data in the acoustic model memory **23** (step S**6**). Since the state transitions for the first node will require several feature vectors to complete, a score is determined for a state transition of the first node in the lexical tree (step S**7**). The lexical tree processor **21** then sends a request to the language model processor **29** for a language model score. The request includes the previous N−1 words and all of the possible words represented by the lexical tree data structure in the acoustic model memory **23** (step S**8**). The language model processor returns scores for each of the N grams ending in the words represented by the lexical tree. The lexical tree processor thus receives language model scores for the words in the lexical tree and picks the highest score (step S**9**). Alternatively, the language model processor **29** can select the highest n gram score and return this to the lexical tree processor for the determination of the temporary lexical tree score. The temporary lexical tree score for the lexical tree is then generated using the score determined for the first state transition of the first lexical tree node and the highest language model score (step S**10**). The temporary lexical tree score is then sent by the lexical tree processor **21** to the results memory **25** over the bus **26** (step S**11**). In this mode of processing, the processor then awaits the next data of the new lexical tree from the search controller (step S**2**).

This process is just one of the processes performed by the lexical tree processor. The main processing performed by the lexical tree processor is the lexical tree evaluation. This process will now be described with reference to the flow diagram of FIG. **5**.

After the start of the process (step S**20**) a lexical tree processor awaits receipt of data for a lexical tree from the search controller (step S**21**). The data comprises a lexical tree pointer to a lexical tree data structure stored in the acoustic model memory **23**, the previous N−1 words in the recognition path, and the current accumulated score.

The lexical tree processor then attempts to read the next feature vector from the feature vector buffer **20** (step S**22**). If this is not available, an error occurs (step s**23**). If the feature vector is available in the feature vector buffer **20**, the feature vector is read from the feature vector buffer **20** by the lexical tree processor **21** (step S**24**) and the lexical tree processor **21** evaluates the state transitions for each path using the acoustic model data in the acoustic model memory **23** (step S**25**). Scores for the paths are determined in accordance with the conventional Viterbi search technique and the best score for the best path is sent to the results memory **25** while the path histories are stored locally in the on-board memory of the lexical tree processor **21** (step S**26**). Pruning is applied to the lexical tree by the lexical tree processor to delete paths in order to keep the breadth of the search manageable (step S**27**). The pruning applied locally by the lexical tree processor can be purely on the basis of a local threshold, which can be provided by the search controller **27**, or it can be determined on a relative basis dependent upon the range of scores for paths being processed within the lexical tree processor **21**. If the path has not reached a word end (step S**28**) the lexical tree processor **21** attempts to read the next feature vector from the feature vector buffer **20** (step S**22**). If the path reaches a word end (step S**28**) the score must be modified by the language model score (step S**29**). There are two ways in which this can be done, in this embodiment the lexical tree processor **21** sends a request to the language model processor **29** for a language model score. The request includes the current word and the N−1 previous words. The language model processor thus returns the language model score and the language model score is used to modify the current accumulated score at the word end. Alternatively, the lexical tree processor can send the language model processor **29** not just the current word and the N−1 previous words, but also the current accumulated score. The language model processor then determines the language model score using the language model memory **31** and modifies the accumulated score using the language model score. The language model processor **29** can then return the modified score to the lexical tree processor **21** which can then pass it to the results memory **25**, or a connec-

tion between the language model processor **29** and the results memory **25** (not shown) can enable the language model processor **29** to send the score directly to the results memory **25** for the lexical tree processor **21**. In this latter case, the language model processor **29** must also receive a lexical tree pointer to identify the lexical tree for which the score applies.

Assuming in this embodiment that the lexical tree processor calculates the modified score (step S**29**), the score and history is then sent by the lexical tree processor **21** to the results memory **25** (step S**30**). The history data sent to the results memory **25** comprises the lexical tree pointer identifying the lexical tree, the modified score at the word end, the phone history identifying at least the last phone to allow for context dependent tri-phone determination of next lexical trees, and the identity of the word identified in the evaluation process.

The lexical tree processor then deletes the path and history data in its on-board memory (step S**31**) and determines if there are any paths still left to be processed (step S**32**). If so, the lexical tree processor **21** tries to access the next feature vector available in the feature vector buffer **20** (step S**22**). If there are no paths left to be processed, a message is sent by the lexical tree processor to the search controller **27** to indicate that the lexical tree has been processed (step S**33**). In this level of processing the lexical tree processor will then await the next data for a lexical tree from the search controller **27**.

The flow diagram of FIG. **5** illustrates the processing of a single instance of a lexical tree. The single point in time a lexical tree can start to be processed from a feature vector, i.e. lexical tree processing starts at a time frame. However, due to a word end returned from lexical tree processing in another lexical tree processor, the lexical tree processor may also be asked during a subsequent time frame to start processing another instance of the lexical tree with a different path history. Thus a lexical tree processor **21** can be processing multiple instances of the lexical tree with different paths having different path histories. In this instance, the pruning applied in step S**27** needs to take into account all paths currently being processed in all instances of lexical tree processing by the lexical tree processor **21**. Thus although pruning is applied locally, it is applied over more than one instance of the lexical tree.

The operation of the search controller **27** will now be described with reference to the flow diagram of FIG. **6**.

After initialization (step S**40**) initial lexical tree data in the results memory **25** is read (step S**41**). The search controller distributes the initial lexical tree data amongst the lexical tree processors for temporary lexical tree score determination (step S**42**) as described with reference to FIG. **4**. Temporary lexical tree scores are returned by the lexical tree processors **21** to the results memory **25** (step S**43**) and the search controller reads the temporary lexical tree scores in the results memory **25** and prunes the lexical trees in the results memory **25** on the basis of the read temporary lexical scores (step S**44**). Thus the set of initial lexical trees can be quite large and this requires a large number of temporary lexical tree score determinations to be performed by the lexical tree processors **21**. However, since the lexical tree processors only need to determine a score for a single feature vector, this can be performed quite quickly. Thus the number of lexical trees for which full evaluation by the lexical tree processors **21** is required can thus be reduced by the selection of the lexical trees with the highest temporary lexical tree score. The search controller than distributes the lexical tree processing among the lexical tree processors **21** by sending each lexical tree processor data including a lexical tree pointer to a lexical tree data structure in the acoustic model memory **23** in the cluster **22**. The search controller **27** then awaits the entry of processing results, i.e. history and scores in the results memory **25** for words (step S**46**). When history and scores for words are entered in the results memory, the search controller **27** determines next possible lexical trees using cross-word tri-phones (step S**47**). For example, as illustrated in FIGS. **3***a* and **3***b*, when the word HARD is entered in the results memory, the search controller identifies the final phone of the word as being d and thus next possible tri-phones can comprise any tri-phones having the left phone d. If there are 50 possible phones, this results in 50^2 possible tri-phones. In practice, only 1 to 200 are likely. However, this may still represent too many for full evaluation and thus the lexical tree data for the next possible lexical trees are distributed amongst the lexical tree processors for temporary lexical tree score determination (step S**48**). Temporary lexical tree scores are returned to the results memory **25** by the lexical tree processors **21** which execute the process as described with reference to FIG. **4** (step S**49**).

The search controller **27** prunes the list of next lexical trees in the results memory **25** on the basis of the temporary lexical scores (step S**50**). The search controller then determines if the words in the results memory have been processed and there are no lexical trees left to be processed and whether or not lexical tree processes are finished (step S**51**). If not, the lexical tree processing is distributed amongst the lexical tree processors (step S**45**). When the end of the utterance is reached, next possible lexical trees will all have been processed and the results can be output from the results memory **25** (step S**52**).

It can be seen from the foregoing description that the embodiment of the invention provides for an efficient parallel processing system incorporating a parallel lexical tree processing architecture and a segmented lexicon storage architecture. Each lexical tree processor performs lexical tree evaluation and language model look ahead for evaluation of company scores for lexical trees to assist in pruning of the number of potential lexical trees that require processing. The architecture is ideally suited to synchronous breadth first searching e.g. a Viterbi search (i.e. real time recognition). The breadth first search is performed synchronously by the parallel processors on the input search parameters when the language model look ahead takes place, it is possible for the feature vector buffer **20** to store a small number, e.g. one or two, feature vectors whilst the language model look ahead is performed on the feature vector so that synchronous breadth first searching can be carried out by re-reading the feature vector out of the buffer **20** when it is determined which lexical trees are to be evaluated by the lexical trees processors **21** after a word end. Thus, the feature vector buffer **20** provides a method of decoupling the search parameterisation circuitry from the search circuit.

If the buffer causes processing to get too far behind, e.g. there are a lot of word end language model look ahead operations to be performed, it is possible for the lexical tree processors to be controlled to temporarily halve their local beam-widths (i.e. more vigorously prune) and thus complete their processing more quickly. This would allow the content of the buffer to be decreased as the processors catch up.

Although the present invention has been described hereinabove with reference to a specific embodiment, it will be apparent to a skilled person in the art that modifications lie within the spirit and scope of the present invention. Although in the embodiment described hereinabove, the acoustic model memories **23** were described as storing parts of the lexical data, it is also possible for more than one copy of the lexical data to be stored in different acoustic model memories **23**. Thus, more than one lexical tree processor cluster **22** can have

access to the same lexical data. This increases the flexibility and allows the controller to avoid any bottlenecks in processing which, for example, can occur if the highest pass scores all require processing by one or a number of lexical tree data structures residing in one of the acoustic model memories **23**.

Although in the embodiments described hereinabove, the memory is partitioned in to a plurality of acoustic model memories **23**, the present invention is not limited to the partitioning of the lexical data and a plurality of lexical tree processors can access a single memory containing lexical data.

The lexical data stored can comprise mono phone models of words. When the lexical data is partitioned, these can be distributed evenly across the memories. If there are 50 mono phones, there will be 50 lexical tree data structures having a unique initial model phone data node. Alternatively, the lexical tree data structures can comprise a context dependent n phone word models. If there are 50 mono phones, there are potentially 50^n lexical tree data structures. For tri-phones, this comprises 50^3 (i.e. 125,000). These are distributed so that trees with the same left context tri-phone, are eventually spread across memories when the lexical data is partitioned. Thus in this way the word end where the left context tri-phone is fixed i.e. the same, the next possible lexical trees will inherently be distributed across the memories, thus ensuring efficient parallel processing of the next lexical trees.

Although the embodiment described hereinabove uses a single results memory accessible by all processors, the present invention encompasses a plurality of results memories. For example, a results memory can be associated with each processor to store the processing results and the controller can read the results in the plurality of results memories in order to control the processors.

Although in the embodiment described above the input speech parameters are described as being feature vectors, the present invention encompasses any type of parameterization of speech data as an input to the circuit. Such parameterization methods are well known in the art.

Although in the embodiment described above there is only one language model processor, more than one can be used and these can be used to operate in parallel for processing scores at word boundaries.

The invention claimed is:

1. A speech recognition circuit comprising:

input means for receiving processed speech parameters;

lexical memory means containing lexical data for word recognition, said lexical data comprising a plurality of lexical tree data structures, each lexical tree data structure comprising a model of words having common prefix components, an initial component of each lexical tree data structure being unique;

a plurality of lexical tree processors connected in parallel to said input means for processing the speech parameters in parallel to perform parallel lexical tree processing for word recognition by accessing said lexical data in said lexical memory means;

results memory means connected to said lexical tree processors for storing processing results from said lexical tree processors and lexical tree identifiers to identify lexical trees to be processed by said lexical tree processors; and

control processor means for controlling said lexical tree processors to process lexical trees identified in said results memory means by performing parallel processing on a plurality of said lexical tree data structures.

2. A speech recognition circuit according to claim **1**, wherein said lexical memory means stores said lexical tree data structure as an n phone model of words, where n is an integer, and said components comprise n phones.

3. A speech recognition circuit according to claim **2**, wherein said lexical memory means stores each lexical tree data structure as mono phone models of words, and said lexical tree processors are arranged to use said mono phone models to generate context dependant phone models of words dynamically for use in processing the speech parameters.

4. A speech recognition circuit according to claim **1**, wherein each lexical tree processor is operative to compare the processed speech parameters with said lexical data or data derived from said lexical data to identify words as a word recognition event and to send information identifying the identified words to said results memory means as the processing results.

5. A speech recognition circuit according to claim **4**, including language model means for providing a language model output for modifying the processing results at a word recognition event by a said lexical tree processor.

6. A speech recognition circuit according to claim **5**, wherein each lexical tree processor is operative to receive said language model output to modify the processing result at a word recognition event determined by said lexical tree processor and to output the modified processing result to said results memory means.

7. A speech recognition circuit according to claim **4**, wherein each lexical tree processor is operative to determine and output scores for words in the processing results at word recognition events.

8. A speech recognition circuit according to claim **1**, wherein said lexical tree processors are operative to process said speech parameters by comparing said speech parameters with said lexical data or data derived from said lexical data in said lexical memory means.

9. A speech recognition circuit according to claim **1**, wherein said control processor means is operative to instruct a said lexical tree processor to process a lexical tree by passing a lexical tree identifier for the lexical tree and history data for a recognition path associated with the lexical tree from said results memory means.

10. A speech recognition circuit according to claim **9**, wherein said history data includes an accumulated score for the recognition path.

11. A speech recognition circuit according to claim **1**, wherein said control processor means is operative to pass a lexical tree identifier to a said lexical tree processor, said lexical tree processor is operative to read lexical data for the identified lexical tree from said lexical memory means and to process said speech parameters using said read lexical data.

12. A speech recognition circuit according to claim **1**, wherein said lexical tree processors are operative to determine and output scores in the processing results during the processing of said speech parameters.

13. A speech recognition circuit according to claim **12**, wherein said lexical tree processors are operative to prune the processing of the lexical tree in dependence upon the determined scores.

14. A speech recognition circuit according to claim **1**, wherein said lexical tree processors are operative to process more than one lexical tree at the same time.

15. A speech recognition circuit according to claim **4**, wherein said control processor means is operative to determine new lexical tree identifiers for storing in said results memory means for words identified in said results memory means.

16. A speech recognition circuit according to claim 15, wherein said control processor means is operative to prune the new lexical tree identifiers.

17. A speech recognition circuit according to claim 16, wherein each lexical tree processor is operative to determine and output scores for words in the processing results at word recognition events, and said control processor means is operative to prune said new lexical tree identifiers on the basis of the scores for the words stored in said results memory means.

18. A speech recognition circuit according to claim 1, wherein said input means is arranged to receive said speech parameters as feature vectors.

19. A speech recognition circuit according to claim 1, wherein said input means is arranged to receive said speech parameters as n phones, where n is an integer.

20. A speech recognition circuit according to claim 1, wherein said lexical tree processors are arranged in groups, said lexical memory means comprises a plurality of partial lexical memory means, each partial lexical memory means being connected to one of said groups of lexical tree processors and containing part of the lexical data, each lexical tree processor is operative to process said speech parameters using a said partial lexical memory means, and said control processor means is operative to control each lexical tree processor to process a lexical tree corresponding to partial lexical data in a corresponding partial lexical memory means.

21. A speech recognition circuit according to claim 1, wherein said lexical memory means comprises a plurality of partial lexical memory means, each partial lexical memory means being connected to one of said lexical tree processors and containing part of the lexical data, each lexical tree processor is operative to process said speech parameters using a said partial lexical memory means, and said control processor means is operative to control each lexical tree processor to process a lexical tree corresponding to partial lexical data in a corresponding partial lexical memory means.

22. A speech recognition circuit according to claim 1, wherein said lexical memory means stores said lexical tree data structures as Hidden Markov Models, and said lexical tree processors are operative to perform the Viterbi search algorithm using each lexical tree data structure.

23. A speech recognition circuit comprising:

an input buffer for receiving processed speech parameters;

lexical memory containing lexical data for word recognition, said lexical data comprising a plurality of lexical tree data structures, each lexical tree data structure comprising a model of words having common prefix components, the initial component of each lexical tree data structure being unique;

a plurality of lexical tree processors connected in parallel to said input buffer for processing the speech parameters in parallel to perform parallel lexical tree processing for word recognition by accessing said lexical data in said lexical memory;

results memory connected to said lexical tree processors for storing processing results from said lexical tree processors and lexical tree identifiers to identify lexical trees to be processed by said lexical tree processors; and

a control processor for controlling said lexical tree processors to process lexical trees identified in said results memory by performing parallel processing on a plurality of said lexical tree data structures.

24. A speech recognition circuit according to claim 23, wherein said lexical memory stores said lexical tree data structure as an n phone model of words, where n is an integer, and said components comprise n phones.

25. A speech recognition circuit according to claim 24, wherein said lexical memory stores said lexical tree data structure as a mono phone model of words, and each lexical tree processor generates context dependant phone models of words dynamically for use in processing the speech parameters.

26. A speech recognition circuit according to claim 23, wherein each lexical tree processor compares the processed speech parameters with the lexical data or data derived from the lexical data to identify words as a word recognition event and sends information identifying the identified words to said results memory as the processing results.

27. A speech recognition circuit according to claim 26, including a language model processor for providing a language model output for modifying the processing results at a word recognition event by a said lexical tree processor.

28. A speech recognition circuit according to claim 27, wherein each lexical tree processor receives said language model output to modify the processing result at a word recognition event determined by said lexical tree processor and outputs the modified processing result to said results memory.

29. A speech recognition circuit according to claim 26, wherein each lexical tree processor determines and outputs scores for words in the processing results at word recognition events.

30. A speech recognition circuit according to claim 23, wherein said lexical tree processors process said speech parameters by comparing said speech parameters with said lexical data in said lexical memory.

31. A speech recognition circuit according to claim 23, wherein said control processor instructs a said lexical tree processor to process a lexical tree by passing a lexical tree identifier for the lexical tree and history data for a recognition path associated with the lexical tree from said results memory.

32. A speech recognition circuit according to claim 31, wherein said history data includes an accumulated score for the recognition path.

33. A speech recognition circuit according to claim 23, wherein said control processor passes a lexical tree identifier to a said lexical tree processor, said lexical tree processor reads lexical data for the identified lexical tree from said lexical memory and processes said speech parameters using said read lexical data.

34. A speech recognition circuit according to claim 23, wherein said lexical tree processors determine and output scores in the processing results during the processing of said speech parameters.

35. A speech recognition circuit according to claim 34, wherein said lexical tree processors prune the processing of the lexical trees in dependence upon the determined scores.

36. A speech recognition circuit according to claim 23, wherein said lexical tree processors process more than one lexical tree at the same time.

37. A speech recognition circuit according to claim 26, wherein said control processor determines new lexical tree identifiers for storing in said results memory for words identified in said results memory.

38. A speech recognition circuit according to claim 37, wherein said control processor means is operative to prune the new lexical tree identifiers.

39. A speech recognition circuit according to claim 38, wherein each lexical tree processor determines and outputs scores for words in the processing results at word recognition events, and said control processor prunes said new lexical tree identifiers on the basis of the scores for the words stored in said results memory means.

40. A speech recognition circuit according to claim 23, wherein said input buffer receives said speech parameters as feature vectors.

41. A speech recognition circuit according to claim 23, wherein said input buffer receives said speech parameters as n phones, where n is an integer.

42. A speech recognition circuit according to claim 23, wherein said lexical tree processors are arranged in groups, said lexical memory comprises a plurality of partial lexical memories, each partial lexical memory being connected to one of said lexical tree processors and containing part of the lexical data, each lexical tree processor processes said speech parameters using a said partial lexical memory, and said control processor controls each lexical tree processor to process a lexical tree corresponding to partial lexical data in a corresponding partial lexical memory.

43. A speech recognition circuit according to claim 23, wherein said lexical memory comprises a plurality of partial lexical memories, each partial lexical memory is connected to one of said lexical tree processors and containing part of the lexical data, each lexical tree processor processes said speech parameters using a said partial lexical memory, and said control processor controls each lexical tree processor to process a lexical tree corresponding to partial lexical data in a corresponding partial lexical memory.

44. A speech recognition circuit according to claim 23, wherein said lexical memory stores said lexical tree data structures as Hidden Markov Models, and each lexical tree processor performs the Viterbi search algorithm using a respective said lexical tree data structure.

45. A speech recognition circuit comprising:

input means for receiving processed speech parameters;

a plurality of lexical memory means containing in combination complete lexical data for word recognition, each lexical memory means containing part of said complete lexical data;

a plurality of processors connected in parallel to said input means for processing the speech parameters in parallel, said processors being arranged in groups of processors, each group of processors being connected to a lexical memory means;

control processor means for controlling each processor to process said speech parameters using partial lexical data read from a said lexical memory means; and

receiving means for receiving the results of the processing of the speech parameters from said processors, wherein

each lexical memory means stores lexical data as at least one lexical tree data structure, each lexical tree data structure comprises a model of words having common prefix components, and an initial component of each lexical tree data structure is unique, and

wherein said processors are operative to use lexical data corresponding to more than one lexical tree at the same time.

46. A speech recognition circuit comprising:

an input buffer receiving processed speech parameters;

a plurality of lexical memories containing in combination complete lexical data for word recognition, each lexical memory containing part of said complete lexical data;

a plurality of processors connected in parallel to said input buffer for processing the speech parameters in parallel, said processors being arranged in groups of processors, each group of processors being connected to a lexical memory;

a control processor controlling each processor to process said speech parameters using partial lexical data read from a respective said lexical memory; and

a results memory storing the results of the processing of the speech parameters from said processors.

47. A speech recognition circuit according to claim 46, wherein each lexical memory stores lexical data as at least one lexical tree data structure, each lexical tree data structure comprises a model of words having common prefix components, and an initial component of each lexical tree data structure is unique.

48. A speech recognition circuit according to claim 47, wherein each lexical memory stores said lexical tree data structure as an n phone model of words, where n is an integer, and said components comprise n phones.

49. A speech recognition circuit according to claim 48, wherein each lexical memory stores said lexical tree data structure as a mono phone model of words, and said lexical tree processors use said mono phone models to generate context dependant phone models of words dynamically for use in processing the speech parameters.

50. A speech recognition circuit according to claim 46, wherein each processor compares the processed speech parameters with the lexical data in a corresponding lexical memory to identify words as a word recognition event and sends information identifying the identified words to said results memory as the processing results.

51. A speech recognition circuit according to claim 50, a including language model processor providing a language model output for modifying the processing results at a word recognition event by a said processor.

52. A speech recognition circuit according to claim 51, wherein each processor receives said language model output to modify the processing result at a word recognition event determined by said processor and outputs the modified processing result to said results memory.

53. A speech recognition circuit according to claim 51, wherein each processor determines and outputs scores for words in the processing results at word recognition events.

54. A speech recognition circuit according to claim 46, wherein said processors process said speech parameters by comparing said speech parameters with said lexical data or data derived from lexical data in respective said lexical memories.

55. A speech recognition circuit according to claim 47, wherein said control processor instructs a said processor to process a lexical tree by passing a lexical tree identifier for the lexical tree and history data for a recognition path associated with the lexical tree to said processor.

56. A speech recognition circuit according to claim 55, wherein said history data includes an accumulated score for the recognition path.

57. A speech recognition circuit according to claim 47, wherein said control processor passes a lexical tree identifier to a said processor, said processor reads lexical data for the identified lexical tree from a corresponding said lexical memory and processes said speech parameters using said read lexical data.

58. A speech recognition circuit according to claim 46, wherein said processors determine and output scores in the processing results during the processing of said speech parameters.

59. A speech recognition circuit according to claim 58, wherein said processors prune the processing of the speech parameters in dependence upon the determined scores.

60. A speech recognition circuit according to claim 47, wherein said processors use lexical data corresponding to more than one lexical tree at the same time.

61. A speech recognition circuit according to claim 50, wherein said control processor identifies lexical data in said

lexical memories required for processing next words at word recognition events, and controls said processors to process the speech parameters using the identified lexical data.

62. A speech recognition circuit according to claim **61**, wherein said control processor prunes the identified lexical data.

63. A speech recognition circuit according to claims **62**, wherein each processor determines and outputs scores for words in the processing results at word recognition events, and said control processor prunes the identified lexical data on the basis of the scores for the words.

64. A speech recognition circuit according to claim **46**, wherein said input buffer is arranged to receive said speech parameters as feature vectors.

65. A speech recognition circuit according to claim **46**, wherein said input buffer is arranged to receive said speech parameters as n phones, where n is an integer.

66. A speech recognition circuit according to claim **46**, wherein said control processor controls each processor to process a lexical tree corresponding to partial lexical data in a corresponding said lexical memory.

67. A speech recognition circuit according to claim **46**, wherein said lexical memories store said lexical data as Hidden Markov Models, and each processor performs the Viterbi search algorithm using a respective part of said lexical data.

68. A speech recognition method comprising:

receiving processed speech parameters;

storing lexical data for word recognition, said lexical data comprising a plurality of lexical tree data structures, each lexical tree data structure comprising a model of words having common prefix components, an initial component of each lexical tree data structure being unique;

passing the speech parameters in parallel to a plurality of lexical tree processors to perform parallel lexical tree processing for word recognition by accessing said lexical data;

storing processing results from said lexical tree processors and lexical tree identifiers to identify lexical trees to be processed by said lexical tree processors; and

controlling said lexical tree processors to process lexical trees identified by the stored identifiers by performing parallel processing on a plurality of said lexical tree data structures.

69. A speech recognition method comprising:

receiving processed speech parameters;

storing complete lexical data for word recognition in a plurality of lexical memory means, each lexical memory means containing part of said complete lexical data;

processing the speech parameters in parallel using a plurality of processors connected in parallel, said processors being arranged in groups of processors, each group of processors being connected to a said lexical memory means;

controlling each processor to process said speech parameters using partial lexical data read from a said lexical memory means; and

storing the results of the processing of the speech parameters from said processors.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 7,587,319 B2
APPLICATION NO. : 10/503463
DATED : September 8, 2009
INVENTOR(S) : Mark Catchpole

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

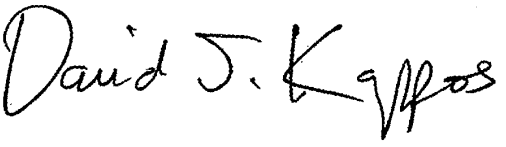
On the Title Page:

The first or sole Notice should read --

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1137 days.

Signed and Sealed this

Twenty-first Day of September, 2010



David J. Kappos
Director of the United States Patent and Trademark Office

Exhibit C

US010971140B2

(12) **United States Patent**
Catchpole

(10) **Patent No.: US 10,971,140 B2**
(45) **Date of Patent: Apr. 6, 2021**

(54) **SPEECH RECOGNITION CIRCUIT USING PARALLEL PROCESSORS**

(71) Applicant: **Zentian Limited**, Cambridge (GB)

(72) Inventor: **Mark Catchpole**, Prickwillow (GB)

(73) Assignee: **Zentian Limited**, Cambridge (GB)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **16/266,265**

(22) Filed: **Feb. 4, 2019**

(65) **Prior Publication Data**

US 2019/0172447 A1 Jun. 6, 2019

Related U.S. Application Data

(63) Continuation of application No. 15/392,396, filed on Dec. 28, 2016, now Pat. No. 10,217,460, which is a (Continued)

(30) **Foreign Application Priority Data**

Feb. 4, 2002 (GB) 0202546

(51) **Int. Cl.**
G10L 15/32 (2013.01)
G10L 15/187 (2013.01)
(Continued)

(52) **U.S. Cl.**
CPC ***G10L 15/187*** (2013.01); ***G10L 15/05*** (2013.01); ***G10L 15/34*** (2013.01)

(58) **Field of Classification Search**
CPC G10L 15/32; G10L 15/34
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,977,599 A * 12/1990 Bahl G10L 15/02 704/256.4
4,980,918 A * 12/1990 Bahl G10L 15/08 704/240
(Continued)

FOREIGN PATENT DOCUMENTS

GB 2 112 194 A 7/1983
GB 2 331 392 A 5/1999
(Continued)

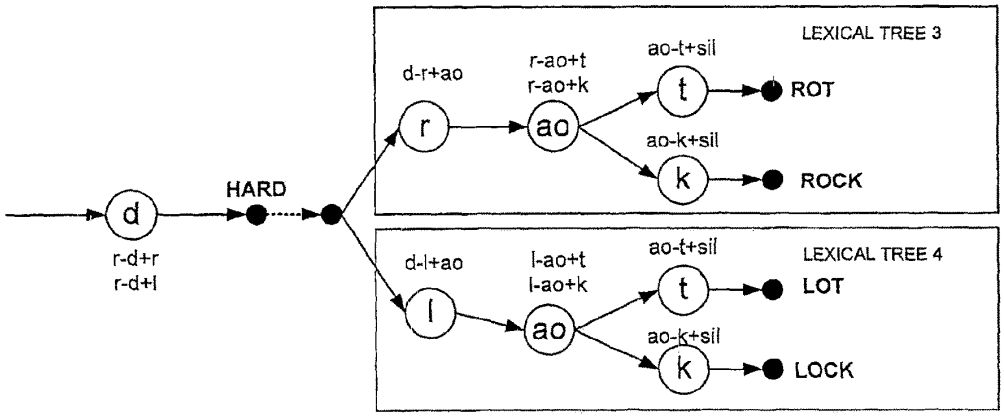
OTHER PUBLICATIONS

R.M. Woodward et al., "Tissue Classification Using Terahertz Pulsed Imaging", Proceedings of the SPIE—The International Society for Optical Engineering SPIE-INT. Soc. Opt. Eng. USA, vol. 5318, No. 1, 2004, pp. 23-33. Abstract also attached.
(Continued)

Primary Examiner — Daniel Abebe
(74) *Attorney, Agent, or Firm* — Finnegan Henderson Farabow Garrett & Dunner LLP

(57) **ABSTRACT**

A speech recognition circuit comprises an input buffer for receiving processed speech parameters. A lexical memory contains lexical data for word recognition. The lexical data comprises a plurality of lexical tree data structures. Each lexical tree data structure comprises a model of words having common prefix components. An initial component of each lexical tree structure is unique. A plurality of lexical tree processors are connected in parallel to the input buffer for processing the speech parameters in parallel to perform parallel lexical tree processing for word recognition by accessing the lexical data in the lexical memory. A results memory is connected to the lexical tree processors for storing processing results from the lexical tree processors and lexical tree identifiers to identify lexical trees to be processed by the lexical tree processors. A controller controls the lexical tree processors to process lexical trees
(Continued)



identified in the results memory by performing parallel processing on a plurality of said lexical tree data structures.

8 Claims, 6 Drawing Sheets

Related U.S. Application Data

continuation of application No. 14/309,476, filed on Jun. 19, 2014, now Pat. No. 9,536,516, which is a continuation of application No. 13/253,223, filed on Oct. 5, 2011, now Pat. No. 8,768,696, which is a continuation of application No. 12/554,607, filed on Sep. 4, 2009, now Pat. No. 8,036,890, which is a continuation of application No. 10/503,463, filed as application No. PCT/GB03/00459 on Feb. 4, 2003, now Pat. No. 7,587,319.

(51) **Int. Cl.**
G10L 15/34 (2013.01)
G10L 15/05 (2013.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,293,213	A	3/1994	Klein et al.	
5,349,645	A	9/1994	Zhao	
5,457,768	A	10/1995	Tsuboi	
5,579,436	A *	11/1996	Chou	G10L 15/063 704/236
5,621,859	A	4/1997	Schwartz	
5,710,866	A *	1/1998	Alleva	G10L 15/10 704/246
5,832,428	A	11/1998	Chow	
5,881,312	A	3/1999	Dulong	
5,960,395	A	9/1999	Tzirkel-Hanock	
5,983,180	A *	11/1999	Robinson	G10L 15/142 704/254
5,995,930	A	11/1999	Hab-Umbach et al.	
6,047,283	A	4/2000	Braun	
6,374,222	B1	4/2002	Kao	
6,526,380	B1	2/2003	Thelen	
7,024,359	B2 *	4/2006	Chang	G10L 15/065 704/251
7,035,802	B1	4/2006	Rigazio	
7,065,488	B2 *	6/2006	Yajima	G10L 15/20 704/226
7,120,582	B1	10/2006	Young et al.	
7,174,037	B2	2/2007	Arnone et al.	
7,899,669	B2	3/2011	Gadbois	
2001/0011218	A1	8/2001	Phillips et al.	
2002/0143531	A1	10/2002	Kahn	
2003/0178584	A1	9/2003	Arnone et al.	
2005/0119883	A1 *	6/2005	Miyazaki	G10L 15/142 704/231
2008/0255839	A1	10/2008	Larri et al.	

FOREIGN PATENT DOCUMENTS

GB	2 347 835 A	9/2000
GB	2 380 920 A	4/2003
WO	WO 00/50859	8/2000
WO	WO-03/067572 A2	8/2003

OTHER PUBLICATIONS

V.P. Wallace et al., "Biomedical Applications of THz Imaging", Microwave Symposium Digest, 2004, IEEE MTT-S International Fort Worth, TX, USA, Jun. 6-11, 2004, Piscataway, NJ, USA, IEEE, vol. 3, Jun. 6, 2004 (Jun. 6, 2004), pp. 1579-1581.

V.P. Wallace et al., "Biomedical Applications of Terahertz Pulse Imaging", Second Joint EMBS-BMES Conference 2002. Conference Proceedings. 24th Annual International Conference of the Engineering in Medicinal and Biology Society. Annual Fall Meeting of the Biomedical Engineering Society, Houston, TX, Oct. 23-26, 2002, Annual vol. 1 of 3. Conf. 24, Oct. 23, 2002 (Oct. 23, 2002), pp. 2333-2334.

A.J. Fitzgerald et al., "Terahertz Imaging of Breast Cancer, a Feasibility Study", Conference Digest of the 2004 Joint 29th International Conference on Infrared and Millimeter Waves and 12th International Conference on Terahertz Electronics IEEE Piscataway, NJ, USA, 2004, pp. 823-824.

B.E. Cole et al., "Terahertz Imaging and Spectroscopy of Human Skin, In-vivo", Proceedings of SPIE, vol. 4276, pp. 1-10.

S. Glinski et al., "Spoken Language Recognition on a DSP Array Processor", IEEE Transactions on Parallel and Distributed Systems, Jul. 5, 1994, No. 7, New York, USA, pp. 697-703.

S. Chatterjee et al., "Connected Speech Recognition on a Multiple Processor Pipeline", ICASSP 89, May 23, 1989, Glasgow, UK, pp. 774-777.

S. H. Chung et al. "A Parallel Phoneme Recognition Algorithm Based on Continuous Hidden Markov Model", Proceedings 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, IPPS/SPDP 1999, Proceedings of 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Pro, IEEE Comput. Soc. pp. 453-457 (1999).

S.H. Chung et al. "A Parallel Computation Model for Integrated Speech and Natural Language Understanding", IEEE Transactions on Computers, vol. 42, No. 10, Oct. 1, 1993.

N. Deshmukn et al. "Hierarchical Search for Large-Vocabulary Conversational Speech Recognition: Working Toward a Solution to the Decoding Problem", IEEE Signal Processing Magazine, vol. 16, No. 5, pp. 84-107 (Sep. 1999).

* cited by examiner

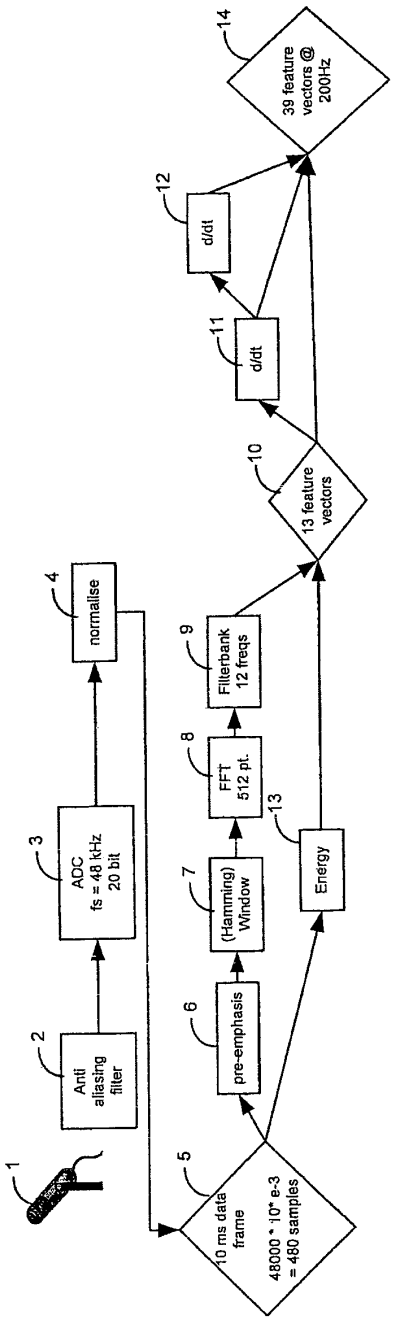

1
2
Anti aliasing filter
3
ADC fs = 48 kHz 20 bit
4
normalise
5
10 ms data frame 48000 * 10* e-3 = 480 samples
6
pre-emphasis
7
(Hamming) Window
8
FFT 512 pt.
9
Filterbank 12 freqs
10
13 feature vectors
11
d/dt
12
d/dt
13
Energy
14
39 feature vectors @ 200Hz

Fig 1

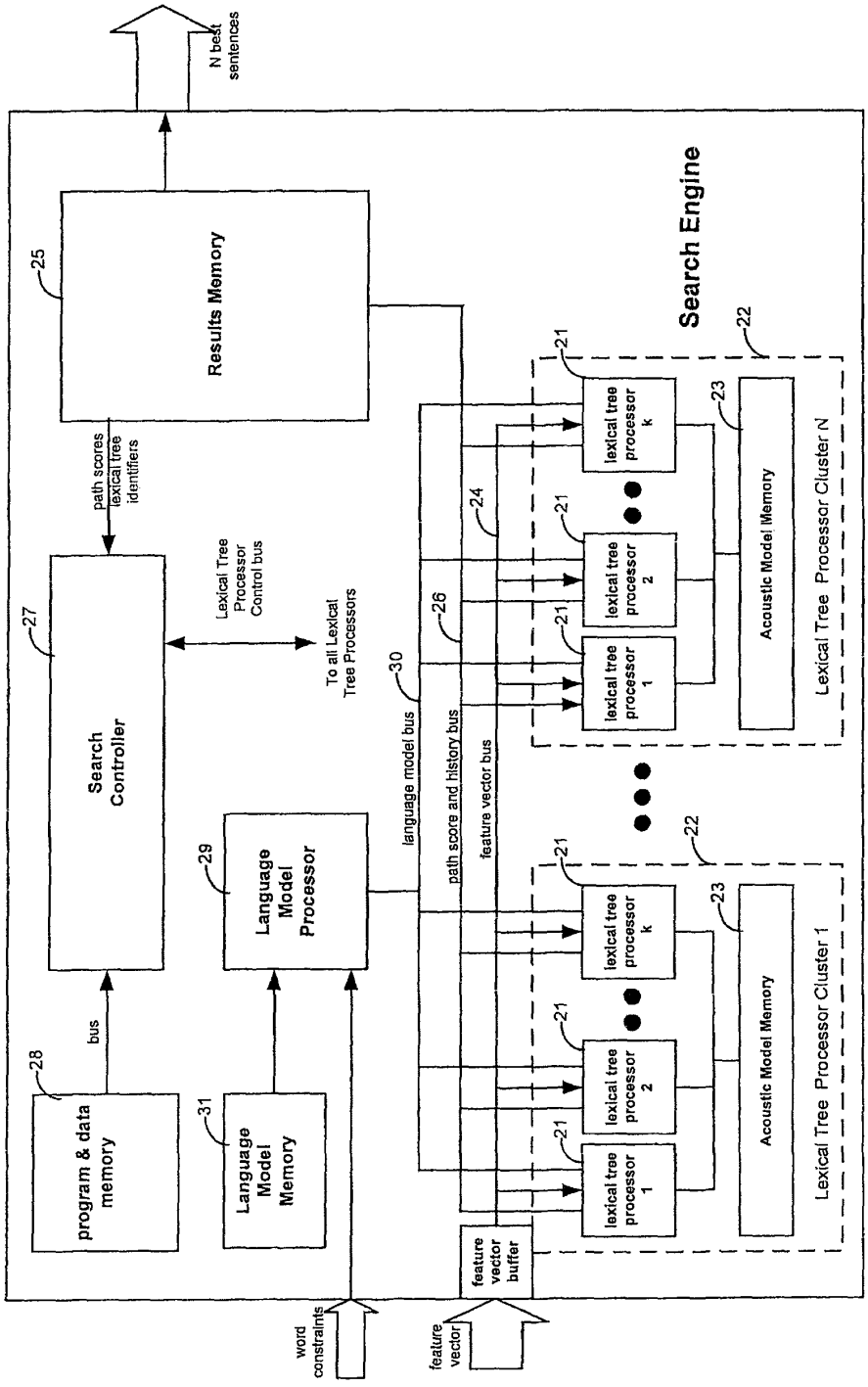

N best sentences
Search Engine
Results Memory
25
path scores lexical tree identifiers
Lexical Tree Processor Control bus
To all Lexical Tree Processors
Search Controller
27
Language Model Processor
29
language model bus
path score and history bus
feature vector bus
24
26
30
bus
program & data memory
28
Language Model Memory
31
feature vector buffer
word constraints
feature vector
lexical tree processor k
lexical tree processor 2
lexical tree processor 1
21
Acoustic Model Memory
23
Lexical Tree Processor Cluster N
Lexical Tree Processor Cluster 1
22

Fig 2

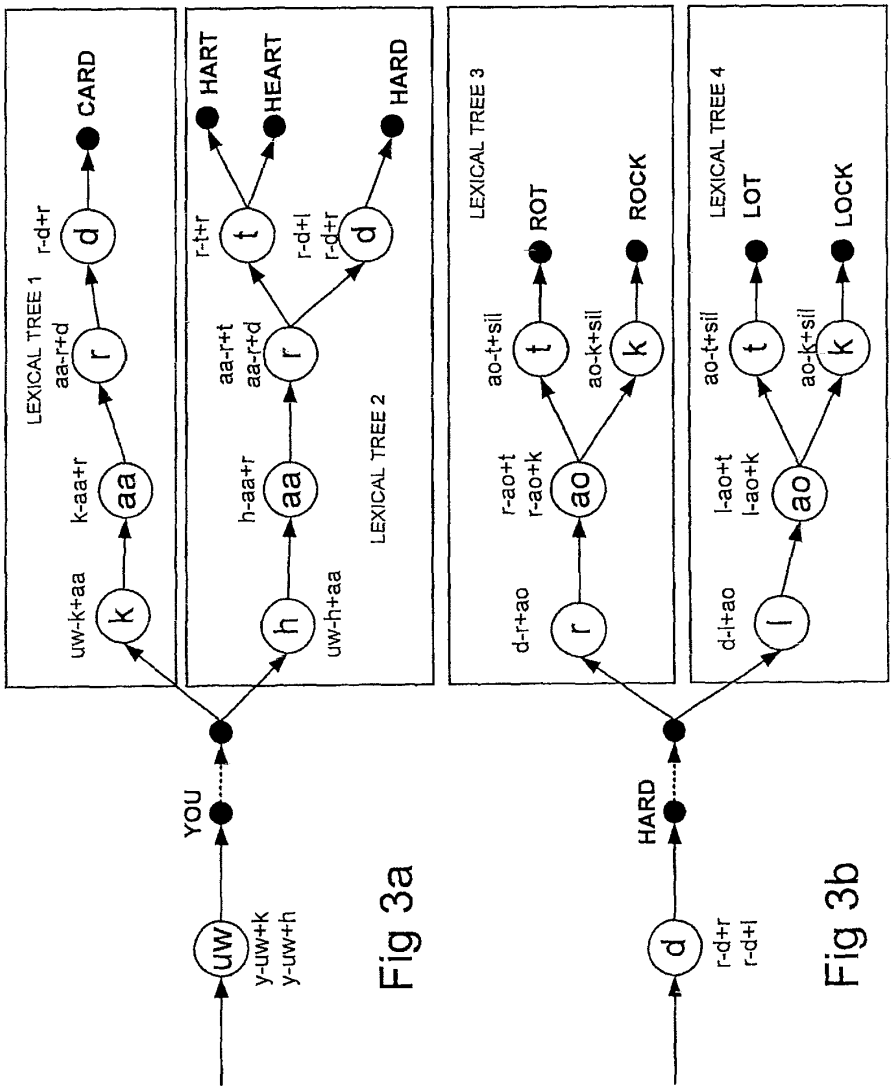

YOU
uw
y-uw+k
y-uw+h
LEXICAL TREE 1
uw-k+aa
k-aa+r
aa-r+d
r-d+r
k
aa
r
d
CARD
LEXICAL TREE 2
uw-h+aa
h-aa+r
aa-r+t
aa-r+d
r-t+r
r-d+l
r-d+r
h
aa
r
t
d
HART
HEART
HARD
HARD
r-d+r
r-d+l
LEXICAL TREE 3
d-r+ao
r-ao+t
r-ao+k
ao-t+sil
ao-k+sil
r
ao
t
k
ROT
ROCK
LEXICAL TREE 4
d-l+ao
l-ao+t
l-ao+k
ao-t+sil
ao-k+sil
l
ao
t
k
LOT
LOCK

Fig 3a

Fig 3b

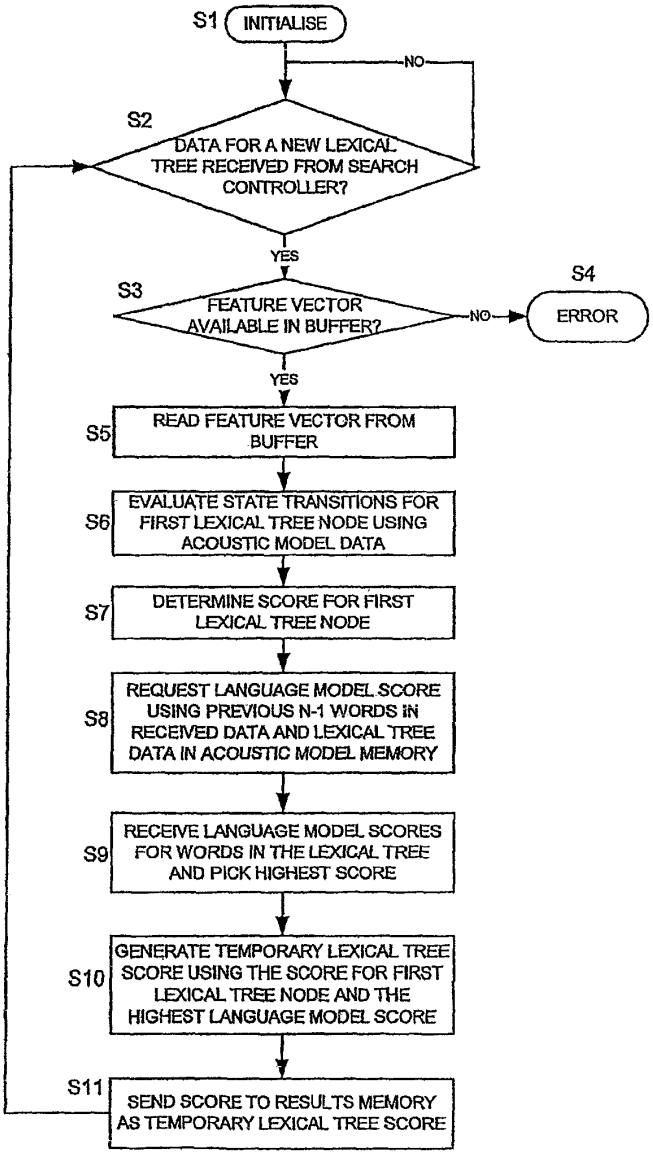

S1
INITIALISE
NO
S2
DATA FOR A NEW LEXICAL TREE RECEIVED FROM SEARCH CONTROLLER?
YES
S3
FEATURE VECTOR AVAILABLE IN BUFFER?
NO
S4
ERROR
YES
S5
READ FEATURE VECTOR FROM BUFFER
S6
EVALUATE STATE TRANSITIONS FOR FIRST LEXICAL TREE NODE USING ACOUSTIC MODEL DATA
S7
DETERMINE SCORE FOR FIRST LEXICAL TREE NODE
S8
REQUEST LANGUAGE MODEL SCORE USING PREVIOUS N-1 WORDS IN RECEIVED DATA AND LEXICAL TREE DATA IN ACOUSTIC MODEL MEMORY
S9
RECEIVE LANGUAGE MODEL SCORES FOR WORDS IN THE LEXICAL TREE AND PICK HIGHEST SCORE
S10
GENERATE TEMPORARY LEXICAL TREE SCORE USING THE SCORE FOR FIRST LEXICAL TREE NODE AND THE HIGHEST LANGUAGE MODEL SCORE
S11
SEND SCORE TO RESULTS MEMORY AS TEMPORARY LEXICAL TREE SCORE

Fig 4

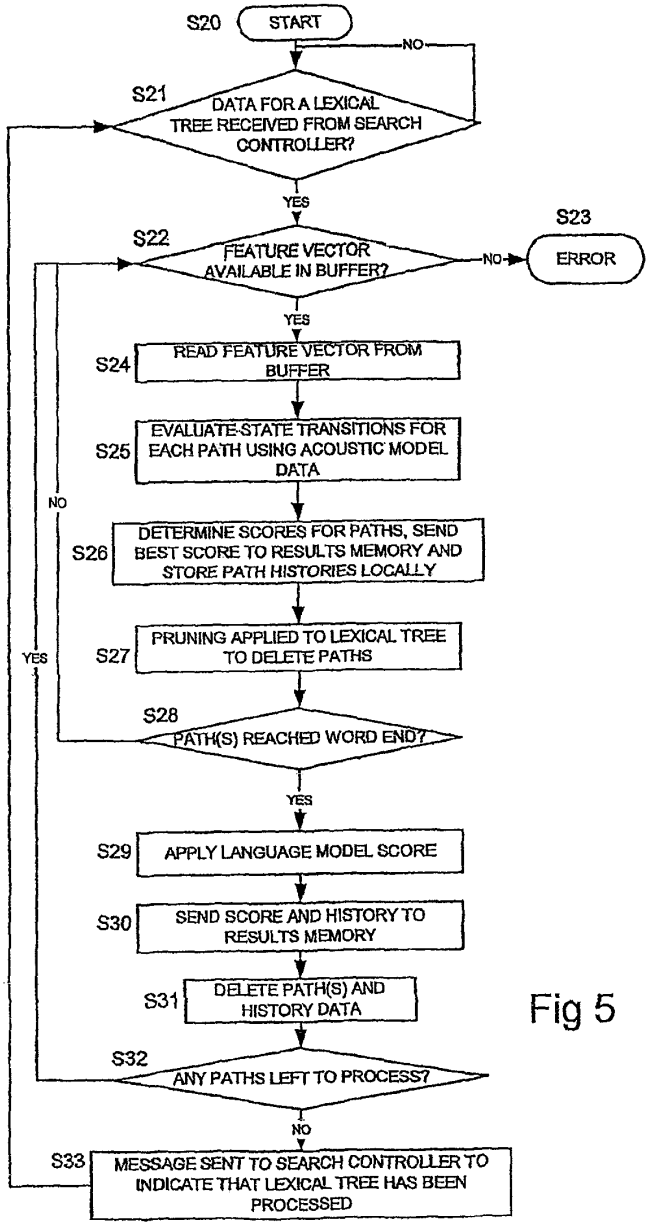

S20
START
NO
S21
DATA FOR A LEXICAL TREE RECEIVED FROM SEARCH CONTROLLER?
YES
S22
FEATURE VECTOR AVAILABLE IN BUFFER?
NO
S23
ERROR
YES
S24
READ FEATURE VECTOR FROM BUFFER
S25
EVALUATE STATE TRANSITIONS FOR EACH PATH USING ACOUSTIC MODEL DATA
NO
S26
DETERMINE SCORES FOR PATHS, SEND BEST SCORE TO RESULTS MEMORY AND STORE PATH HISTORIES LOCALLY
S27
PRUNING APPLIED TO LEXICAL TREE TO DELETE PATHS
YES
S28
PATH(S) REACHED WORD END?
YES
S29
APPLY LANGUAGE MODEL SCORE
S30
SEND SCORE AND HISTORY TO RESULTS MEMORY
S31
DELETE PATH(S) AND HISTORY DATA
S32
ANY PATHS LEFT TO PROCESS?
NO
S33
MESSAGE SENT TO SEARCH CONTROLLER TO INDICATE THAT LEXICAL TREE HAS BEEN PROCESSED

Fig 5

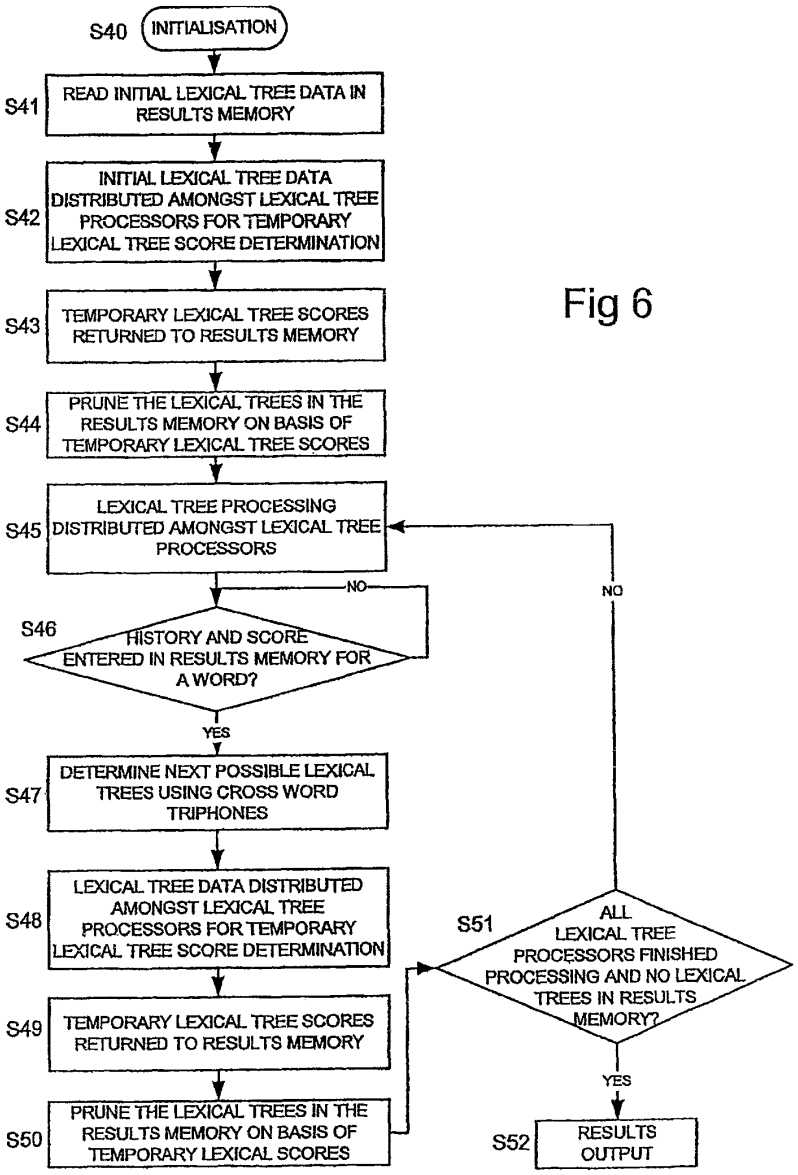

S40
INITIALISATION
S41
READ INITIAL LEXICAL TREE DATA IN RESULTS MEMORY
S42
INITIAL LEXICAL TREE DATA DISTRIBUTED AMONGST LEXICAL TREE PROCESSORS FOR TEMPORARY LEXICAL TREE SCORE DETERMINATION
S43
TEMPORARY LEXICAL TREE SCORES RETURNED TO RESULTS MEMORY
S44
PRUNE THE LEXICAL TREES IN THE RESULTS MEMORY ON BASIS OF TEMPORARY LEXICAL TREE SCORES
S45
LEXICAL TREE PROCESSING DISTRIBUTED AMONGST LEXICAL TREE PROCESSORS
S46
HISTORY AND SCORE ENTERED IN RESULTS MEMORY FOR A WORD?
NO
YES
S47
DETERMINE NEXT POSSIBLE LEXICAL TREES USING CROSS WORD TRIPHONES
S48
LEXICAL TREE DATA DISTRIBUTED AMONGST LEXICAL TREE PROCESSORS FOR TEMPORARY LEXICAL TREE SCORE DETERMINATION
S49
TEMPORARY LEXICAL TREE SCORES RETURNED TO RESULTS MEMORY
S50
PRUNE THE LEXICAL TREES IN THE RESULTS MEMORY ON BASIS OF TEMPORARY LEXICAL SCORES
S51
ALL LEXICAL TREE PROCESSORS FINISHED PROCESSING AND NO LEXICAL TREES IN RESULTS MEMORY?
NO
YES
S52
RESULTS OUTPUT

Fig 6

SPEECH RECOGNITION CIRCUIT USING PARALLEL PROCESSORS

This is a continuation of application Ser. No. 15/392,396, filed Dec. 28, 2016, which is a continuation of application Ser. No. 14/309,476, filed Jun. 19, 2014, now U.S. Pat. No. 9,536,516, which is a continuation of application Ser. No. 13/253,223, now U.S. Pat. No. 8,768,696, filed Oct. 5, 2011, which is a continuation of application Ser. No. 12/554,607, now U.S. Pat. No. 8,036,890, filed Sep. 4, 2009, which is continuation of application Ser. No. 10/503,463, now U.S. Pat. No. 7,587,319, filed May 24, 2005, which is a 371 of International Application No. PCT/GB2003/000459, filed Feb. 4, 2003, which claims foreign priority to United Kingdom Application No. 0202546.8 filed Feb. 4, 2002, the disclosures of which are incorporated herein by reference in their entireties.

The present invention generally relates to a speech recognition circuit which uses parallel processors for processing the input speech data in parallel.

Conventional large vocabulary speech recognition can be divided into two processes: front end processing to generate processed speech parameters such as feature vectors, followed by a search process which attempts to find the most likely set of words spoken from a given vocabulary (lexicon).

The front end processing generally represents no problem for current processing systems. However, for large vocabulary, speaker independent speech recognition, it is the search process that presents the biggest challenge. An article by Deshmukh et al entitled "Hierarchical Search for Large-Vocabulary Conversational Speech Recognition" (IEEE Signal Processing Magazine, September 1999, pages 84 to 107), the content of which is hereby incorporated by reference, discusses the general concepts of large vocabulary speech recognition. As discussed in this paper, one algorithm for performing the search is the Viterbi algorithm. The Viterbi algorithm is a parallel or breadth first search through a transition network of states of Hidden Markov Models. An acoustic model for words in a lexicon are represented as states of Hidden Markov Models. These states represent phones or n phones in a phone model of the words. The search requires the evaluation of possible word matches. It is known that such a search is computationally intensive.

In order to speed up the processing, performed during such a search in a speech recognition system, parallel processing has been explored. In an article by M K Ravishankar entitled "Parallel Implementation of Fast Beam Search for Speaker-Independent Continuous Speech Recognition" (Indian Institute of Science, Bangalor, India, Jul. 16, 1993) a multi-threaded implementation of a fast beam search algorithm is disclosed. The multi-threading implementation requires a significant amount of communication and synchronization among threads. In an MSC project report by R. Dujari entitled "Parallel Viterbi Search Algorithm for Speech Recognition" (MIT, February 1992) the parallel processing of input speech parameters is disclosed in which a lexical network is split statically among processors.

It is an object of the present invention to provide an improved circuit which can perform parallel processing of speech parameters.

In accordance with a first embodiment of the present invention, a speech recognition circuit comprises an input port such as input buffer for receiving parameterized speech data such as feature vectors. A lexical memory arrangement is provided which contains lexicon data for word recognition. The lexical data comprises a plurality of lexical tree data structures representing a plurality of lexical trees. Each lexical tree data structure comprises a model of words having common prefix components and an initial component which is unique as an initial component for lexical trees. A plurality of lexical tree processors are connected in parallel to the input port and perform parallel lexical tree processing for word recognition by accessing the lexical data in the lexical memory arrangement. A results memory arrangement is connected to the lexical tree processors for storing processing results from the lexical tree processors and lexical tree identifiers to identify lexical trees to be processed by the lexical tree processors. A controller controls the lexical tree processors to process lexical trees identified in the results memory arrangement by performing parallel processing of a plurality of lexical tree data structures.

Thus in accordance with this embodiment of the present invention, the processing in order to perform word recognition is distributed across the processors by controlling the processors to perform processing on different lexical trees. The controller controls the processor by the processes to provide for efficient process management by distributing lexical processing to appropriate processors.

The lexical tree data structure can comprise a phone model of words, wherein the components comprise phones. For reduced storage, the lexical tree data structure can comprise a mono phone lexical tree. The mono phone lexical tree can be used to generate context dependent phone models dynamically. This enables the use of context dependent phone models for matching and hence increased accuracy whilst not increasing memory requirements. Alternatively, the lexical tree data structure can comprise context dependent phone models.

The processing performed by each processor in one embodiment comprises the comparison of the speech parameters with the lexical data, e.g. phone models or data derived from the lexical data (e.g. dynamically generated context dependent phone models) to identify words as a word recognition event and to send information identifying the identified words to the results memory as the processing results. In this embodiment a language model processor arrangement can be provided for providing a language model output for modifying the processing results at a word recognition event by a lexical tree processor. The modification can either take place at each lexical tree processor, or at the language model processing arrangement.

In one embodiment each lexical tree processor determines an output score for words in the processing results at word recognition events. Thus in this embodiment the language model processing arrangement can modify the score using a score for a language model for n preceding words, where n is an integer.

In one embodiment the controller instructs a lexical tree processor to process a lexical tree by passing a lexical tree identifier for the lexical tree and history data for a recognition path associated with the lexical tree from the results memory. The history data preferably includes an accumulated score for the recognition path. This enables a score to be determined based on the score for the recognition path to accumulate a new score during recognition carried out using the lexical tree data structure. The scores can be output in the processing results to the results memory during the processing of the speech parameters so that the scores can be used for pruning.

In one embodiment of the present invention, each lexical tree processor operates on more than one lexical tree at the same time, e.g. two lexical trees represented by two different lexical tree data structures, or two lexical trees represented

by the same data structure but displaced in time (which can be termed to instances of the same lexical tree).

At word recognition events, the controller determines new lexical tree identifiers for storing in the results memory for words identified in the results memory for respective word events. In order to reduce the processing, the controller can prune the new lexical tree identifiers to reduce the number of lexical trees which are required to be processed. This pruning can be achieved using context dependant n phones to reduce the number of possible next phones. The number can be further reduced by using a language model look ahead technique.

In one embodiment of the present invention, the lexical tree processors are arranged in groups or clusters. The lexical memory arrangement comprises a plurality of partial lexical memories. Each partial lexical memory is connected to one of the groups of lexical tree processors and contains part of the lexical data. Thus a group of lexical tree processors and a partial lexical memory form a cluster. Each lexical tree processor is operative to process the speech parameters using a partial lexical memory and the controller controls each lexical tree processor to process a lexical tree corresponding to partial lexical data in a corresponding partial lexical memory.

In another embodiment of the present invention the lexical memory arrangement comprises a plurality of partial lexical memories. Each partial lexical memory being connected to one of the lexical tree processors and containing part of the lexical data. Each lexical tree processor processes the speech parameters using a corresponding partial lexical memory and the controller is operative to control each lexical tree processor to process a lexical tree corresponding to partial lexical data in a corresponding partial lexical memory.

In one embodiment of the present invention the lexical memory arrangement stores the lexical tree data structures as Hidden Markov Models and the lexical tree processors are operative to perform the Viterbi search algorithm using each respective lexical tree data structure. Thus in this way, this embodiment of the present invention provides a parallel Viterbi lexical tree search process for speech recognition.

The first aspect of the present invention is a special purpose circuit built for performing the speech recognition search process in which there are a plurality of processors for performing parallel lexical tree processing on individual lexical tree processors.

In another aspect of the present invention a speech recognition circuit comprises an input port such as an input buffer for receiving parameterized speech data such as feature vectors. A plurality of lexical memories are provided which contain in combination complete lexical data for word recognition. Each lexical memory contains part of the complete lexical data. A plurality of processors are provided connected in parallel to the input port for processing the speech parameters in parallel. The processors are arranged in groups in which each group is connected to a corresponding lexical memory to form a cluster. A controller controls each processor to process the speech parameters using partial lexical data read from a corresponding lexical memory. The results of processing the speech parameters are output from the processors as recognition data.

Thus this aspect of the present invention provides a circuit in which speech recognition processing is performed in parallel by groups of processors operating in parallel in which each group accesses a common memory of lexical data. This aspect of the present invention provides the advantage of parallel processing of speech parameters and benefits from a limited segmentation of the lexical data. By providing a plurality of processors in a group with a common memory, flexibility in the processing is provided without being bandwidth limited by the interface to the memory that would occur if only a single memory were used for all processors. The arrangement is more flexible than the parallel processing arrangement in which each processor only has access to its own local memory and requires fewer memory interfaces (i.e. chip pins). Each processor within a group can access the same lexical data as any other processor in the group. The controller can thus control the parallel processing of input speech parameters in a more flexible manner. For example, it allows more than one processor to process input speech parameters using the same lexical data in a lexical memory. This is because the lexical data is segmented into domains which are accessible by multiple processors.

In a preferred embodiment this aspect of the present invention is used in combination with the first aspect of the present invention. In such an arrangement each processor performs lexical tree processing and the lexical data stored in each lexical memory comprises lexical tree data structures which each comprise a model of words having common prefix components and an initial component that is unique.

In preferred embodiments of the second aspect of the present invention, the preferred embodiments of the first aspect of the present invention are incorporated.

Embodiments of the present invention will now be described with reference to the accompanying drawings in which:

FIG. **1** is a diagram of a speech data processing circuit for generating parameterized speech data (feature vectors);

FIG. **2** is a diagram of a speech recognition circuit in accordance with an embodiment of the present invention;

FIGS. **3***a* and **3***b* are schematic diagrams illustrating lexical tree structures;

FIG. **4** is a flow diagram illustrating the process performed by a lexical tree processor to determine a temporary lexical tree score in accordance with an embodiment of the present invention;

FIG. **5** is a flow diagram illustrating the process performed by the lexical tree processor for processing the input feature vectors in accordance with an embodiment of the present invention; and

FIG. **6** is a flow diagram illustrating the process performed by the controller in accordance with an embodiment of the present invention.

FIG. **1** illustrates a typical circuit for the parameterization of input speech data. In this embodiment the parameters generated are speech vectors.

A microphone **1** records speech in an analogue form and this is input through an anti-aliasing filter **2** to an analogue-to-digital converter **3** which samples the speech at 48 kHz at 20 bits per sample. The digitized output signal is normalized (**4**) to generated a 10 millisecond data frame every 5 milliseconds with 5 milliseconds overlap (**5**). A pre-emphasis operation **6** is applied to the data followed by a hamming window **7**. The data is then fast Fourier transformed (FFT) using a 512 point fast Fourier transform (**8**) before being filtered by filter bank **9** into 12 frequencies. The energy in the data frame **5** is also recorded (**13**) as an additional feature and together with the **12** frequency outputs of the filter bank **9**, **13** feature vectors (**10**) are thus produced and these are output as part of the **39** feature vectors **14**. First and second derivatives (**11** and **12**) are taken of the **13** feature vectors **10** to complete the generation of the **39** feature vectors **14**.

The arrangement illustrated in FIG. **1** is purely given for illustration. The present invention encompasses any means by which speech and data can be parameterized to a suitable form for input to the search process as will be described in more detail hereinafter.

FIG. **2** is a schematic diagram of a speech recognition circuit in accordance with an embodiment of the present invention for performing the search process. The parameterized speech data, which in this embodiment comprise feature vectors, are input to a feature vector buffer **20**. The feature vector buffer **20** is provided to buffer the incoming feature vectors to allow lexical tree processors **21** to read and process the feature vectors in the buffer **20** via a feature vector bus **24**. A plurality k of lexical tree processors **21** are arranged in a respective lexical tree processor cluster **22**. Each lexical tree processor cluster **22** has an acoustic model memory **23** in which is stored lexical data for use by the lexical tree processors **21** within the lexical tree processor cluster **22**. Each lexical tree processor **21** in the lexical tree processor cluster **22** is connected to the acoustic model memory **23** within the lexical tree processor **22**. There are N lexical tree processor clusters and thus there are Nk lexical tree processors **21** connected by the feature vector bus **24** to the feature vector buffer **20**. Each lexical tree processor **21** is capable of processing a different lexical tree and thus Nk lexical trees can be processed in parallel. The acoustic model memories **23** store as a whole a complete set of lexical data, i.e. lexical tree data structures for use in the lexical tree processing by the lexical tree processors **21**. Each acoustic model memory **23** contains part or a segment of the lexical tree data. Since lexical tree processors **21** in a lexical tree processor cluster **22** access the same acoustic model memory **23**, it is possible for more than one lexical tree processor **21** to process the same lexical data. This provides for some degree of flexibility in the controlling of the processing by the lexical tree processors **21**. Further, the acoustic model memories **23** need not contain only one copy of the lexical data. It is-possible to build in a redundancy in the data to further enhance the flexibility. This avoids any bottleneck in the processing due to the search processing focusing on a small number of lexical trees.

A results memory **25** is provided for storing processing results from the lexical tree processors **21** which are received over the path score and history bus **26**. The results memory **25** also stores information on lexical trees to identify which lexical trees are to be processed. A search controller **27** is provided to control the processing performed by the lexical tree processors **21** in dependence upon a program and data stored in program and data memory **28**. The search controller reads the path scores and lexical tree identifiers from the results memory and controls the lexical tree processors accordingly. A language model processor **29** is provided which is connected to each lexical tree processor **21** by a language model bus **30**. The language model processor **29** accesses a language model memory **31** to read language model data for provision to lexical tree processors **21** in response to language model data requests. External control of the language model memory **31** is provided by a word constrains input. The language model processor **29** determines a score for a word occurring following N previous words using N grams. When a lexical tree processor requires a language model score a request is sent to the language model processor **29** over the language model bus **30** identifying the current word and the N−1 previous words. A language model score for the N gram can be returned to the lexical tree processor **21** for the modification of the score at the end of a branch of lexical tree processing. The lexical tree processor can modify the score in accordance with the language model and output a score to the results memory **25** for a word at the end of a branch of the lexical tree processing. Thus the results memory stores the results as an ordered list of scores for words together with their histories.

The results memory **25** stores the following data:

1. Initial lexical tree data. This comprises pointers to an initial set of lexical trees. No history data is associated with the initial set of lexical trees. The initial set of lexical trees is predetermined and stored in the results memory **25** based on the most likely initial phones of as utterance. This initial lexical tree data is required to initialize the search process.

2. History data for search results. This comprises a record of a recognition path through the lexical tree recognition process performed by the lexical tree processors **21**. The history data includes the current word, the previous N−1 words, the current accumulated score, the phone history (for use in the determination of likely next lexical trees using cross word context dependent tri-phones), and an identifier or pointer to the lexical tree used for identifying the word.

3. Best scores for best paths being processed by each lexical tree processor **21**. This information enables the search controller **27** to monitor the processing being performed by lexical tree processors **21** to determine whether a global pruning strategy should be applied in order to reassign processing performed by a lexical tree processor if its best score for its best path is below a threshold or well below the best scores for the paths being processed by other lexical tree processors **21**.

4. Temporary lexical tree scores. These comprise tree scores which are determined as temporary scores to prune the next lexical trees to be processed at word ends. The temporary lexical tree scores include lexical tree identifiers or pointers to identify the next lexical trees to be processed. The scores enable the pruning of this list.

5. Pruning threshold. This can be a global threshold value for use in the pruning of the lexical trees globally, or a local threshold value for use by a lexical processor for locally pruning the processing performed by the lexical processor **21**.

The acoustic model memory **23** stores a Hidden Markov Model for acoustically modelling words as lexical trees. The acoustic model memory **23** stores a plurality of lexical tree data structures. Each lexical tree data structure comprises an n phone model of a number of words having common prefix phones. The first node of the lexical tree (the root) comprises a common n phone to all words in the lexical tree and uniquely identifies the lexical tree.

Each lexical tree processor **21** includes on-board memory (or local memory) to be used during the lexical tree processing. This working memory has to store all of the parameters currently working on including current scores for all paths being processed within the lexical tree, and previous N−1 words for the lexical tree. The local storage of the previous N−1 words enables the lexical tree processor **21**, when a word end is reached along a branch of the lexical tree, to send a request for the language model score for an N gram by sending the identity of the N−1 previous words together with the word identified at the end of the branch.

FIGS. **3***a* and **3***b* schematically illustrate lexical trees which can be processed by the lexical tree processor **21** during the recognition of the two words HARD ROCK. In FIG. **3***a* a previous lexical tree terminated at a branch recognizing the word YOU and terminating with the mono phone uw which is associated with two context dependent tri-phones y−uw+k and y−uw+h. Thus the context dependent tri-phone associated with the last phone in the lexical

tree word model for YOU indicates two possible next lexical trees, i.e. the lexical trees beginning with the mono phone k and h and having a left context phone of uw. As can be see in FIG. **3***a* this word end YOU therefore leads to two possible next lexical trees. These two lexical trees are traversed in parallel by two different lexical tree processors accumulating scores for matching of input feature vectors into Hidden Markov Models of the context dependent tri-phones associated with each node in the tree. When the end of the branch is reached, a word end event is reached and a word is recognized. As can be seen in FIG. **3***a*, in this example since four words are of similar phone length, it is possible for the search strategy based on inputting feature vectors in parallel to simultaneously reach a number of possible word ends. These possible word ends are sent as processing results to the results memory. The results memory stores the accumulated score at the word end together with phone history to identify the last phone and its associated context dependent tri-phones. In this example, considering the branch where the word HARD is recognized, the last phone is d which has associated with it two context dependent tri-phones r−d+l and r−d+r. Thus the search controller **27** can identify next possible lexical trees using the next phone in the context dependent tri-phone. In this case, as can be seen in FIG. **3***b*, the next possible lexical trees begin with the phones r and l and have associated with them context dependent tri-phones d−r+ao and d−l+ao respectively. Thus these are the next lexical trees that require processing following the end node or word event detected in the processing lexical tree **2**. FIG. **3***b* thus represents the processing that is required at the end of FIG. **3***a* at the word node for HARD in processing the second lexical tree.

As can be seen in FIG. **3***b*, the two lexical trees are processed in parallel by traversing through the branches to reach word ends by sequentially entering in feature vectors in parallel to lexical tree processors processing the respective lexical trees. When the word end is reached, the accumulated score is output to the results memory as described before. In this case since it is the last word, the important context dependent tri-phone associated with the final phone has silence (sil) as the following phone. The final phone can in fact have 50 context dependent tri-phones associated with it if there are 50 possible next phones (including silence). FIG. **3***b* only illustrates the relevant one for the end of the utterance.

FIG. **4** is a flow diagram illustrating the processing performed by a lexical tree processor **21** in order to determine a temporary lexical tree score for a lexical tree. When a word end is identified by a lexical tree processor, the processing results are sent to the results memory **25**. The results memory identifies the last phone of the recognized words and thus enables the search controller **27** to identify possible next lexical trees using context dependent tri-phones as illustrated in FIG. **3***b*. Although in theory if there are 50 phones there are 50^2 possible lexical trees, due to the use of the context dependant triphones, only 100-200 lexical tees are likely to be identified as possible lexical trees. The search controller needs to further prune out the lexical trees which are unlikely to generate likely paths with high scores, hi order to do this, instructions are sent to the lexical tree processors to determine a temporary lexical tree score which can be used to prune out lexical trees from the processing which have a low score.

The processing by the lexical tree processor to generate the temporary lexical tree score is shown in FIG. **4** and for this process, following initialization, a lexical free processor **21** awaits receipt of data for a new lexical free from the search controller **27** (step S**2**). The data comprises a lexical tree pointer on an identifier to lexical tree data stored in the acoustic model memory **23**, i.e. the lexical tree data structure, previous N−1 words (for use in language model score determination by the language model processor **29**), and the current path score. The previous N−1 words include the previously recognized word for which a score has been accumulated.

The lexical tree processor **21** then attempts to read the next feature vector from the feature vector buffer **20** (step S**3**) and if this is not available, an error occurs (step S**4**). If the feature vector is available in the feature vector buffer **20**, the lexical tree processor **21** reads the feature vector from the buffer **20** (step S**5**) and evaluates the state transitions for the first lexical tree node using the acoustic model data in the acoustic model memory **23** (step S**6**). Since the state transitions for the first node will require several feature vectors to complete, a score is determined for a state transition of the first node in the lexical tree (step S**7**). The lexical tree processor **21** then sends a request to the language model processor **29** for a language model score. The request includes the previous N−1 words and all of the possible words represented by the lexical tree data structure in the acoustic model memory **23** (step S**8**). The language model processor returns scores for each of the N grams ending in the words represented by the lexical tree. The lexical tree processor thus receives language model scores for the words in the lexical tree and picks the highest score (step S**9**). Alternatively, the language model processor **29** can select the highest n gram score and return this to the lexical tree processor for the determination of the temporary lexical tree score. The temporary lexical tree score for the lexical tree is then generated using the score determined for the first state transition of the first lexical tree node and the highest language model score (step S**10**). The temporary lexical tree score is then sent by the lexical tree processor **21** to the results memory **25** over the bus **26** (step S**11**). In this mode of processing, the processor then awaits the next data of the new lexical tree from the search controller (step S**2**).

This process is just one of the processes performed by the lexical tree processor. The main processing performed by the lexical tree processor is the lexical tree evaluation. This process will now be described with reference to the flow diagram of FIG. **5**.

After the start of the process (step S**20**) a lexical tree processor awaits receipt of data for a lexical tree from the search controller (step S**21**). The data comprises a lexical tree pointer to a lexical tree data structure stored in the acoustic model memory **23**, the previous N−1 words in the recognition path, and the current accumulated score.

The lexical tree processor then attempts to read the next feature vector from the feature vector buffer **20** (step S**22**). If this is not available, an error occurs (step s**23**). If the feature vector is available in the feature vector buffer **20**, the feature vector is read from the feature vector buffer **20** by the lexical tree processor **21** (step S**24**) and the lexical tree processor **21** evaluates the state transitions for each path using the acoustic model data in the acoustic model memory **23** (step S**25**). Scores for the paths are determined in accordance with the conventional Viterbi search technique and the best score for the best path is sent to the results memory **25** while the path histories are stored locally in the on-board memory of the lexical tree processor **21** (step S**26**). Pruning is applied to the lexical tree by the lexical tree processor to delete paths in order to keep the breadth of the search manageable (step S**27**). The pruning applied locally by the lexical tree processor can be purely on the basis of a

local threshold, which can be provided by the search controller **27**, or it can be determined on a relative-basis dependent upon the range of scores for paths being processed within the lexical tree processor **21**. If the path has not reached a word end (step S**28**) the lexical tree processor **21** attempts to read the next feature vector from the feature vector buffer **20** (step S**22**). If the path reaches a word end (step S**28**) the score must be modified by the language model score (step S**29**). There are two ways in which this can be done, in this embodiment the lexical tree processor **21** sends a request to the language model processor **29** for a language model score. The request includes the current word and the N−1 previous words. The language model processor thus returns the language model score and the language model score is used to modify the current accumulated score at the word end. Alternatively, the lexical tree processor can send the language model processor **29** not just the current word and the N−1 previous words, but also the current accumulated score. The language model processor then determines the language model score using the language model memory **31** and modifies the accumulated score using the language model score. The language model processor **29** can then return the modified score to the lexical tree processor **21** which can then pass it to the results memory **25**, or a connection between the language model processor **29** and the results memory **25** (not shown) can enable the language model processor **29** to send the score directly to the results memory **25** for the lexical tree processor **21**. In this latter case, the language model processor **29** must also receive a lexical tree pointer to identify the lexical tree for which the score applies.

Assuming in this embodiment that the lexical tree processor calculates the modified score (step S**29**), the score and history is then sent by the lexical tree processor **21** to the results memory **25** (step S**30**). The history data sent to the results memory **25** comprises the lexical tree pointer identifying the lexical tree, the modified score at the word end, the phone history identifying at least the last phone to allow for context dependent tri-phone determination of next lexical trees, and the identity of the word identified in the evaluation process.

The lexical tree processor then deletes the path and history data in its on-board memory (step S**31**) and determines if there are any paths still left to be processed (step S**32**). If so, the lexical tree processor **21** tries to access the next feature vector available in the feature vector buffer **20** (step S**22**). If there are no paths left to be processed, a message is sent by the lexical tree processor to the search controller **27** to indicate that the lexical tree has been processed (step S**33**). In this level of processing the lexical tree processor will then await the next data for a lexical tree from the search controller **27**.

The flow diagram of FIG. **5** illustrates the processing of a single instance of a lexical tree. The single point in time a lexical tree can start to be processed from a feature vector, i.e. lexical tree processing starts at a time frame. However, due to a word end returned from lexical tree processing in another lexical tree processor, the lexical tree processor may also be asked during a subsequent time frame to start processing another instance of the lexical tree with a different path history. Thus a lexical tree processor **21** can be processing multiple instances of the lexical tree with different paths having different path histories. In this instance, the pruning applied in step S**27** needs to take into account all paths currently being processed in all instances of lexical tree processing by the lexical tree processor **21**. Thus although pruning is applied locally, it is applied over more than one instance of the lexical tree.

The operation of the search controller **27** will now be described with reference to the flow diagram of FIG. **6**.

After initialization (step S**40**) initial lexical tree data in the results memory **25** is read (step S**41**). The search controller distributes the initial lexical tree data amongst the lexical tree processors for temporary lexical tree score determination (step S**42**) as described with reference to FIG. **4**. Temporary lexical tree scores are returned by the lexical tree processors **21** to the results memory **25** (step S**43**) and the search controller reads the temporary lexical tree scores in the results memory **25** and prunes the lexical trees in the results memory **25** on the basis of the read temporary lexical scores (step S**44**). Thus the set of initial lexical trees can be quite large and this requires a large number of temporary lexical tree score determinations to be performed by the lexical tree processors **21**. However, since the lexical tree processors only need to determine a score for a single feature vector, this can be performed quite quickly. Thus the number of lexical trees for which full evaluation by the lexical tree processors **21** is required can thus be reduced by the selection of the lexical trees with the highest temporary lexical tree score. The search controller than distributes the lexical tree processing among the lexical tree processors **21** by sending each lexical tree processor data including a lexical tree pointer to a lexical tree data structure in the acoustic model memory **23** in the cluster **22**. The search controller **27** then awaits the entry of processing results, i.e. history and scores in the results memory **25** for words (step S**46**). When history and scores for words are entered in the results memory, the search controller **27** determines next possible lexical trees using cross-word tri-phones (step S**47**). For example, as illustrated in FIGS. **3***a* and **3***b*, when the word HARD is entered in the results memory, the search controller identifies the final phone of the word as being d and thus next possible tri-phones can comprise any tri-phones having the left phone d. If there are 50 possible phones, this results in 50^2 possible tri-phones. In practice, only 1 to 200 are likely. However, this may still represent too many for full evaluation and thus the lexical free data for the next possible lexical trees are distributed amongst the lexical tree processors for temporary lexical tree score determination (step S**48**). Temporary lexical tree scores are returned to the results memory **25** by the lexical tree processors **21** which execute the process as described with reference to FIG. **4** (step S**49**).

The search controller **27** prunes the list of next lexical trees in the results memory **25** on the basis of the temporary lexical scores (step S**50**). The search controller then determines if the words in the results memory have been processed and there are no lexical trees left to be processed and whether or not lexical tree processes are finished (step S**51**). If not, the lexical tree processing is distributed amongst the lexical tree processors (step S**45**). When the end of the utterance is reached, next possible lexical trees will all have been processed and the results can be output from the results memory **25** (step S**52**).

It can be seen from the foregoing description that the embodiment of the invention provides for an efficient parallel processing system incorporating a parallel lexical tree processing architecture and a segmented lexicon storage architecture. Each lexical tree processor performs lexical tree evaluation and language model look ahead for evaluation of company scores for lexical trees to assist in pruning of the number of potential lexical trees that require processing. The architecture is ideally suited to synchronous breadth

first searching e.g. a Viterbi search (i.e. real time recognition). The breadth first search is performed synchronously by the parallel processors on the input search parameters when the language model look ahead takes place, it is possible for the feature vector buffer **20** to store a small number, e.g. one or two, feature vectors whilst the language model look ahead is performed on the feature vector so that synchronous breadth first searching can be carried out by re-reading the feature vector out of the buffer **20** when it is determined which lexical trees are to be evaluated by the lexical trees processors **21** after a word end. Thus, the feature vector buffer **20** provides a method of decoupling the search parameterisation circuitry from the search circuit.

If the buffer causes processing to get too far behind, e.g. there are a lot of word end language model look ahead operations to be performed, it is possible for the lexical tree processors to be controlled to temporarily halve their local beamwidths (i.e. more vigorously prune) and thus complete their processing more quickly. This would allow the content of the buffer to be decreased as the processors catch up.

Although the present invention has been described hereinabove with reference to a specific embodiment, it will be apparent to a skilled person in the art that modifications lie within the spirit and scope of the present invention. Although in the embodiment described hereinabove, the acoustic model memories **23** were described as storing parts of the lexical data, it is also possible for more than one copy of the lexical data to be stored in different acoustic model memories **23**. Thus, more than one lexical tree processor cluster **22** can have access to the same lexical data. This increases the flexibility and allows the controller to avoid any bottlenecks in processing which, for example, can occur if the highest pass scores all require processing by one or a number of lexical tree data structures residing in one of the acoustic model memories **23**.

Although in the embodiments described hereinabove, the memory is partitioned in to a plurality of acoustic model memories **23**, the present invention is not limited to the partitioning of the lexical data and a plurality of lexical tree processors can access a single memory containing lexical data.

The lexical data stored can comprise mono phone models of words. When the lexical data is partitioned, these can be distributed evenly across the memories. If there are 50 mono phones, there will be 50 lexical tree data structures having a unique initial model phone data node. Alternatively, the lexical tree data structures can comprise a context dependent n phone word models. If there are 50 mono phones, there are potentially 50^n lexical tree data structures. For tri-phones, this comprises 50^3 (i.e. 125,000). These are distributed so that trees with the same left context tri-phone, are eventually spread across memories when the lexical data is partitioned. Thus in this way the word end where the left context tri-phone is fixed i.e. the same, the next possible lexical trees will inherently be distributed across the memories, thus ensuring efficient parallel processing of the next lexical trees.

Although the embodiment described hereinabove uses a single results memory accessible by all processors, the present invention encompasses a plurality of results memories. For example, a results memory can be associated with each processor to store the processing results and the controller can read the results in the plurality of results memories in order to control the processors.

Although in the embodiment described above the input speech parameters are described as being feature vectors, the present invention encompasses any type of parameterization of speech data as an input to the circuit. Such parameterization methods are well known in the art.

Although in the embodiment described above there is only one language model processor, more than one can be used and these can be used to operate in parallel for processing scores at word boundaries.

What is claimed is:

1. A speech recognition circuit comprising:

one or more clusters of processors, each of the one or more clusters of processors comprising:

a plurality of processors; and

an acoustic model memory storing acoustic model data, wherein each of the plurality of processors is configured to compute a probability using the acoustic model data in the acoustic model memory, wherein:

the speech recognition circuit is configured to generate an initial score for an audio sample; and

the initial score is used to determine whether to continue processing to determine a final score via processing a larger amount of model data than that was processed to generate the initial score.

2. The speech recognition circuit of claim **1**, wherein the probability is an input to an evaluation of a state transition of a model of states.

3. The speech recognition circuit of claim **2**, wherein the model is a Hidden Markov Model.

4. The speech recognition circuit of claim **2**, wherein the probability is computed from a Gaussian mixture model and one or more feature vectors.

5. The speech recognition circuit of claim **1**, further comprising:

a buffer for storing one or more feature vectors coupled to at least one of the plurality of processors of the one or more clusters of processors.

6. The speech recognition circuit of claim **5**, further comprising:

a search controller coupled to the at least one of the plurality of processors of the one or more clusters of processors, the search controller capable of controlling the at least one of the plurality of processors of the one or more clusters of processors to initiate speech recognition processing in at least a subset of processors of the one or more clusters of processors.

7. The speech recognition circuit of claim **1**, wherein:

the one or more clusters of processors comprises a first cluster of processors and a second cluster of processors;

the first cluster comprises a first acoustic model memory; and

the second cluster comprises a second acoustic model memory that is distinct and separate from the first acoustic model memory.

8. The speech recognition circuit of claim **1**, wherein:

the one or more clusters of processors comprises a first cluster of processors; and

the first cluster comprises a first acoustic model coupled to all processors of the first cluster.

* * * * *

Exhibit D

US010062377B2

(12) **United States Patent**
Larri et al.

(10) **Patent No.: US 10,062,377 B2**
(45) **Date of Patent: *Aug. 28, 2018**

(54) **DISTRIBUTED PIPELINED PARALLEL SPEECH RECOGNITION SYSTEM**

(71) Applicant: **Zentian Limited**, Cambridge (GB)

(72) Inventors: **Guy Larri**, Cambridge (GB); **Mark Catchpole**, Prickwillow (GB); **Damian Kelly Harris-Dowsett**, Conventry (GB); **Timothy Brian Reynolds**, Cambridge (GB)

(73) Assignee: **Zentian Limited**, Cambridge (GB)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **14/788,164**

(22) Filed: **Jun. 30, 2015**

(65) **Prior Publication Data**

US 2016/0005397 A1 Jan. 7, 2016

Related U.S. Application Data

(60) Continuation of application No. 13/735,091, filed on Jan. 7, 2013, now Pat. No. 9,076,441, which is a (Continued)

(30) **Foreign Application Priority Data**

Sep. 14, 2004 (GB) 0420464.0

(51) **Int. Cl.**
G10L 15/10 (2006.01)
G10L 15/08 (2006.01)
(Continued)

(52) **U.S. Cl.**
CPC ***G10L 15/08*** (2013.01); ***G10L 15/02*** (2013.01); ***G10L 15/06*** (2013.01); ***G10L 15/10*** (2013.01); ***G10L 15/285*** (2013.01)

(58) **Field of Classification Search**
CPC G10L 15/10
(Continued)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,881,312 A 3/1999 Dulong
6,374,220 B1 4/2002 Kao
(Continued)

FOREIGN PATENT DOCUMENTS

EP 1 178 466 A2 2/2002
GB 2 333 172 A 7/1999
(Continued)

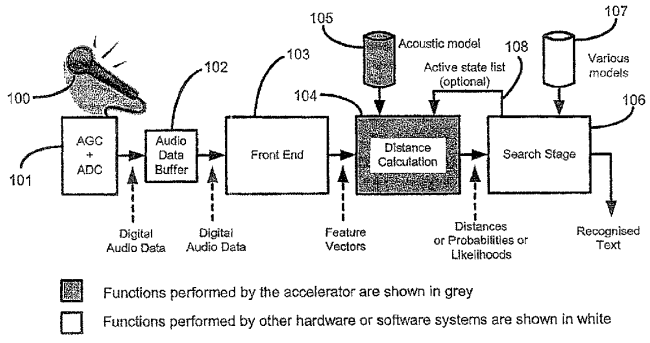
OTHER PUBLICATIONS

Glinski et al., "Spoken Language Recognition on DSP Array Processor" IEEE Transactions on Parallel and Distributed Systems, vol. 5, No. 7, pp. 697-703, Jul. 1, 1994.
(Continued)

Primary Examiner — Susan McFadden
(74) *Attorney, Agent, or Firm* — Blank Rome LLP

(57) **ABSTRACT**

A speech recognition circuit comprising a circuit for providing state identifiers which identify states corresponding to nodes or groups of adjacent nodes in a lexical tree, and for providing scores corresponding to said state identifiers, the lexical tree comprising a model of words. The circuit includes: a memory structure for receiving and storing state identifiers identified by a node identifier identifying a node or group of adjacent nodes, the memory structure being adapted to allow lookup to identify particular state identifiers, reading of the scores corresponding to the state identifiers, and writing back of the scores to the memory structure after modification of the scores; an accumulator for receiving score updates corresponding to particular state identifiers from a score update generating circuit which generates the score updates using audio input, for receiving
(Continued)



scores from the memory structure, and for modifying said scores by adding said score updates to said scores; and a selector circuit for selecting at least one node or group of adjacent nodes of the lexical tree according to said scores.

6 Claims, 27 Drawing Sheets

Related U.S. Application Data

continuation of application No. 13/162,128, filed on Jun. 16, 2011, now Pat. No. 8,352,262, which is a division of application No. 11/662,704, filed as application No. PCT/GB2005/003554 on Sep. 14, 2005, now Pat. No. 7,979,277.

(51) **Int. Cl.**

G10L 15/28	(2013.01)
G10L 15/06	(2013.01)
G10L 15/02	(2006.01)

(58) **Field of Classification Search**
USPC 704/238, 251
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,328,153	B2	2/2008	Wells et al.
7,979,277	B2	7/2011	Larri et al.
8,036,890	B2	10/2011	Catchpole
8,352,262	B2	1/2013	Larri et al.
8,612,227	B2	12/2013	Kato
9,076,441	B2	7/2015	Larri et al.
2003/0061046	A1	3/2003	Zhao et al.
2004/0111264	A1	6/2004	Wang et al.

FOREIGN PATENT DOCUMENTS

GB	2 391 679 A	2/2004
WO	WO 01/48737 A2	7/2001
WO	WO 01/75862 A2	10/2001
WO	WO 03/067572 A2	8/2003

OTHER PUBLICATIONS

S. Chatterjee et al., "Connected Speech Recognition on a Multiple Processor Pipeline." ICASSP 89, pp. 774-777, May 23, 1989.

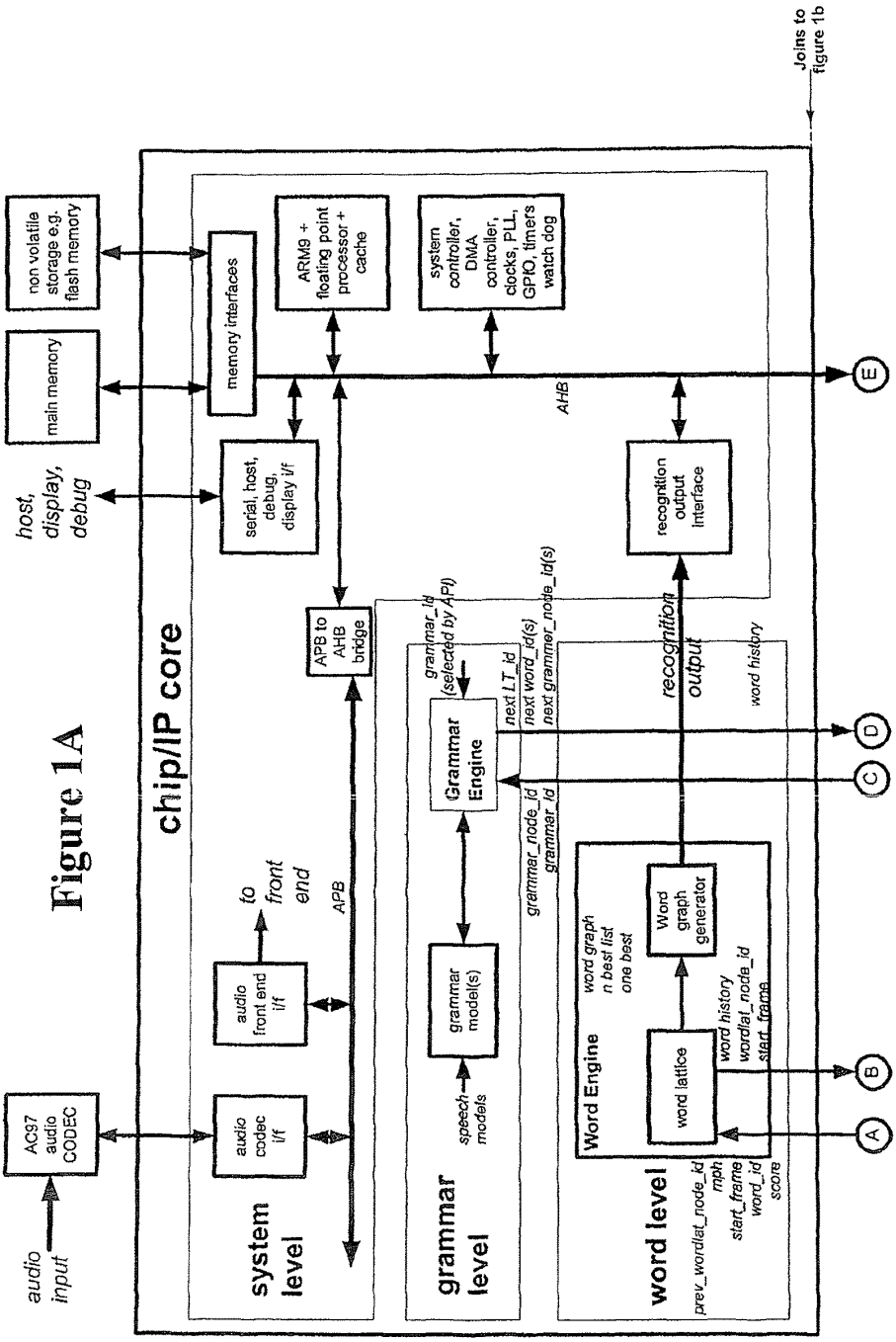
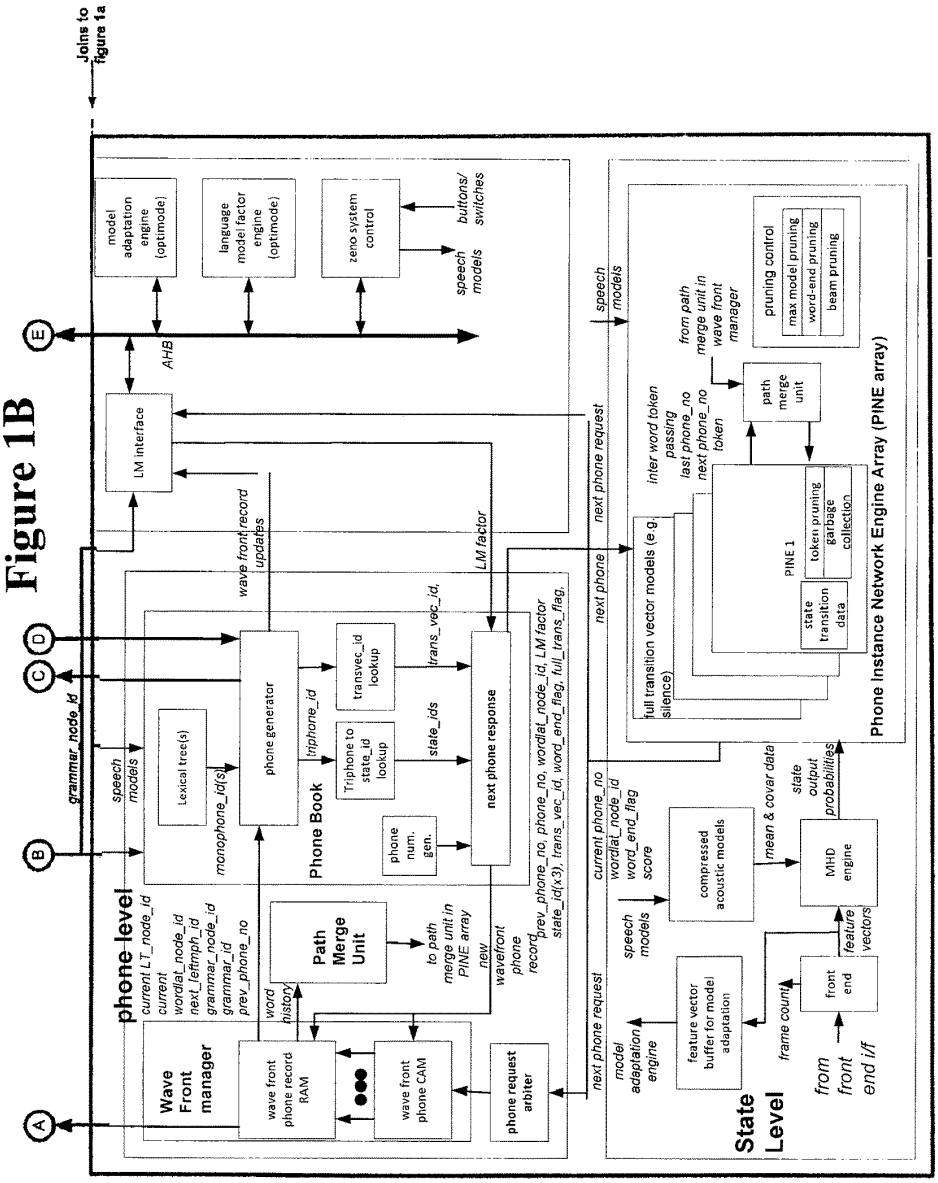

Joins to figure 1b
non volatile storage e.g. flash memory
main memory
host, display, debug
AC97 audio CODEC
audio input
chip/IP core
memory interfaces
ARM9 + floating point processor + cache
system controller, DMA controller, clocks, PLL, GPIO, timers watch dog
AHB
serial, host, debug, display i/f
recognition output interface
APB to AHB bridge
audio codec i/f
audio front end i/f
to front end
APB
system level
grammar_id (selected by API)
Grammar Engine
next LT_id
next word_id(s)
next grammar_node_id(s)
grammar_node_id
grammar_id
grammar model(s)
speech models
grammar level
recognition output
word history
Word Engine
word graph
n best list
one best
Word graph generator
word lattice
word history
wordlat_node_id
start_frame
prev_wordlat_node_id
mph
start_frame
word_id
score
word level
A
B
C
D
E

Figure 1A

Figure 1B


Joins to figure 1a
A
B
C
D
E
model adaptation engine (optimode)
language model factor engine (optimode)
zeno system control
buttons/ switches
speech models
AHB
LM interface
grammar_node_id
speech models
Lexical tree(s)
monophone_id(s)
phone generator
wave front record updates
triphone_id
transvec_id lookup
Triphone to state_id lookup
Phone Book
phone num. gen.
trans_vec_id,
state_ids
LM factor
next phone response
next phone request
next phone
speech models
inter word token passing
last phone_no
next phone_no
token
from path merge unit in wave front manager
path merge unit
pruning control
max model pruning
word-end pruning
beam pruning
full transition vector models (e.g. silence)
PINE 1
token pruning
garbage collection
state transition data
Phone Instance Network Engine Array (PINE array)
phone level
current LT_node_id
current wordlat_node_id
next_leftmph_id
grammar_node_id
grammar_id
prev_phone_no
Wave Front manager
wave front phone record RAM
wave front phone CAM
word history
Path Merge Unit
to path merge unit in PINE array
new wavefront phone record
prev_phone_no, phone_no, wordlat_node_id, LM factor
state_id(x3), trans_vec_id, word_end_flag, full_trans_flag.
phone request arbiter
current phone_no
wordlat_node_id
word_end_flag
score
speech models
model adaptation engine
next phone request
compressed acoustic models
mean & cover data
state output probabilities
MHD engine
feature vector buffer for model adaptation
frame count
front end
feature vectors
from front end i/f
State Level

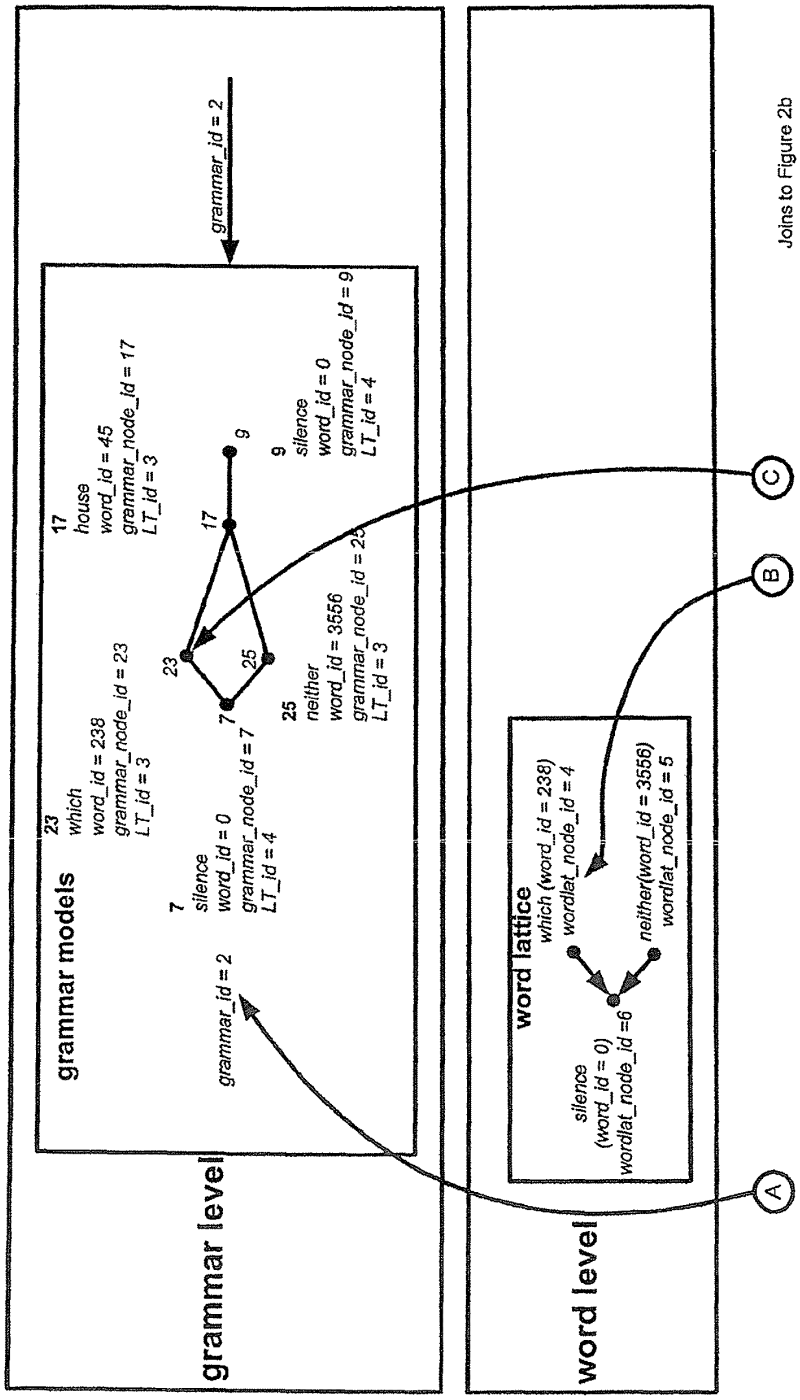

grammar level
word level
grammar models
grammar_id = 2
23
which
word_id = 238
grammar_node_id = 23
LT_id = 3
17
house
word_id = 45
grammar_node_id = 17
LT_id = 3
7
silence
word_id = 0
grammar_node_id = 7
LT_id = 4
25
neither
word_id = 3556
grammar_node_id = 25
LT_id = 3
9
silence
word_id = 0
grammar_node_id = 9
LT_id = 4
grammar_id = 2
word lattice
which (word_id = 238)
wordlat_node_id = 4
neither(word_id = 3556)
wordlat_node_id = 5
silence
(word_id = 0)
wordlat_node_id =6
A
B
C
Joins to Figure 2b

Figure 2A

Figure 2B

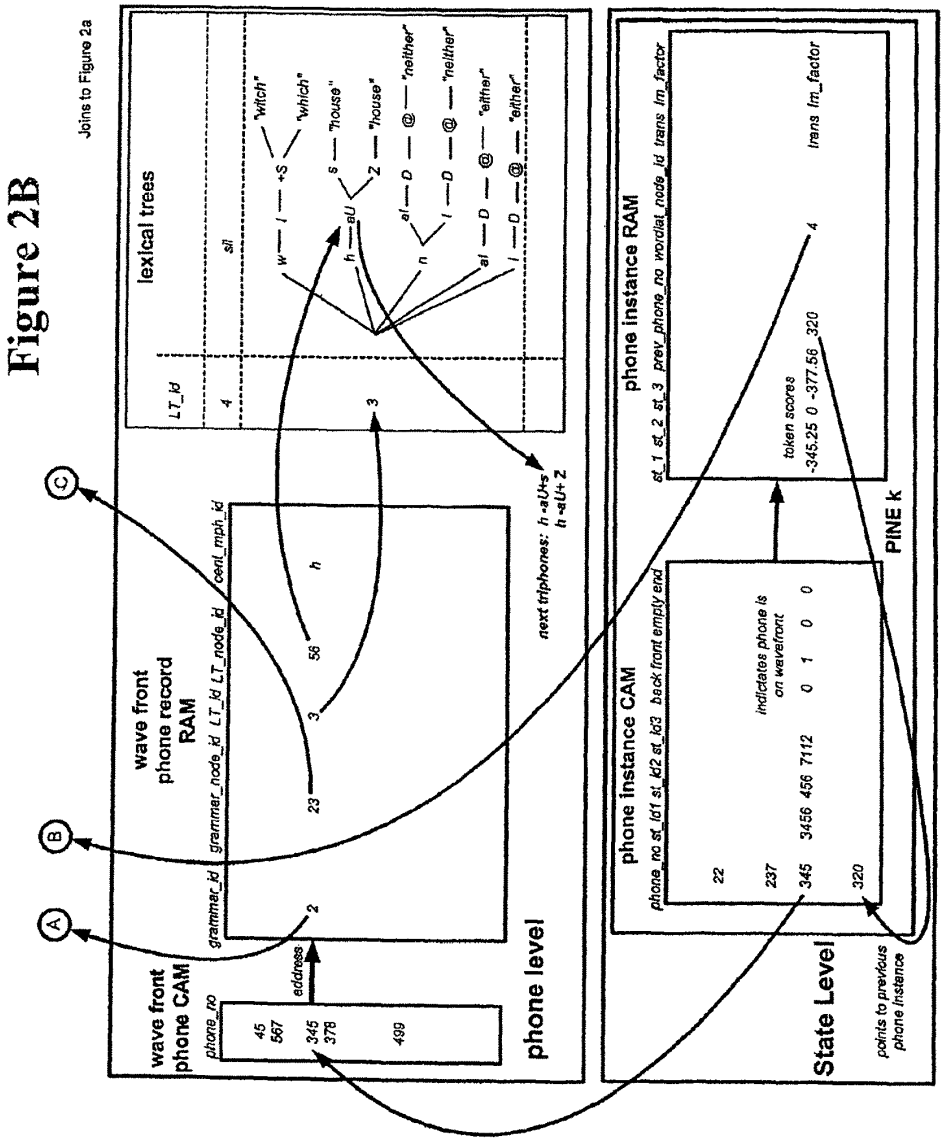

Joins to Figure 2a
lexical trees
LT_id
4
sil
"witch"
"which"
"house"
"house"
"neither"
"neither"
"either"
"either"
wave front phone record RAM
grammar_id
grammar_node_id
LT_id
LT_node_id
cent_mph_id
next triphones: h -aU+s
h -aU+ Z
wave front phone CAM
phone_no
address
45
567
345
378
499
phone level
phone instance RAM
phone_no st_id1 st_id2 st_id3 back front empty end
st_1 st_2 st_3 prev_phone_no wordlet_node_id trans lm_factor
token scores
-345.25 0 -377.56 320
trans lm_factor
4
indicates phone is on wavefront
22
237
345
3456 456 7112
0 1 0 0
320
PINE k
State Level
points to previous phone instance
A
B
C

Figure 3A

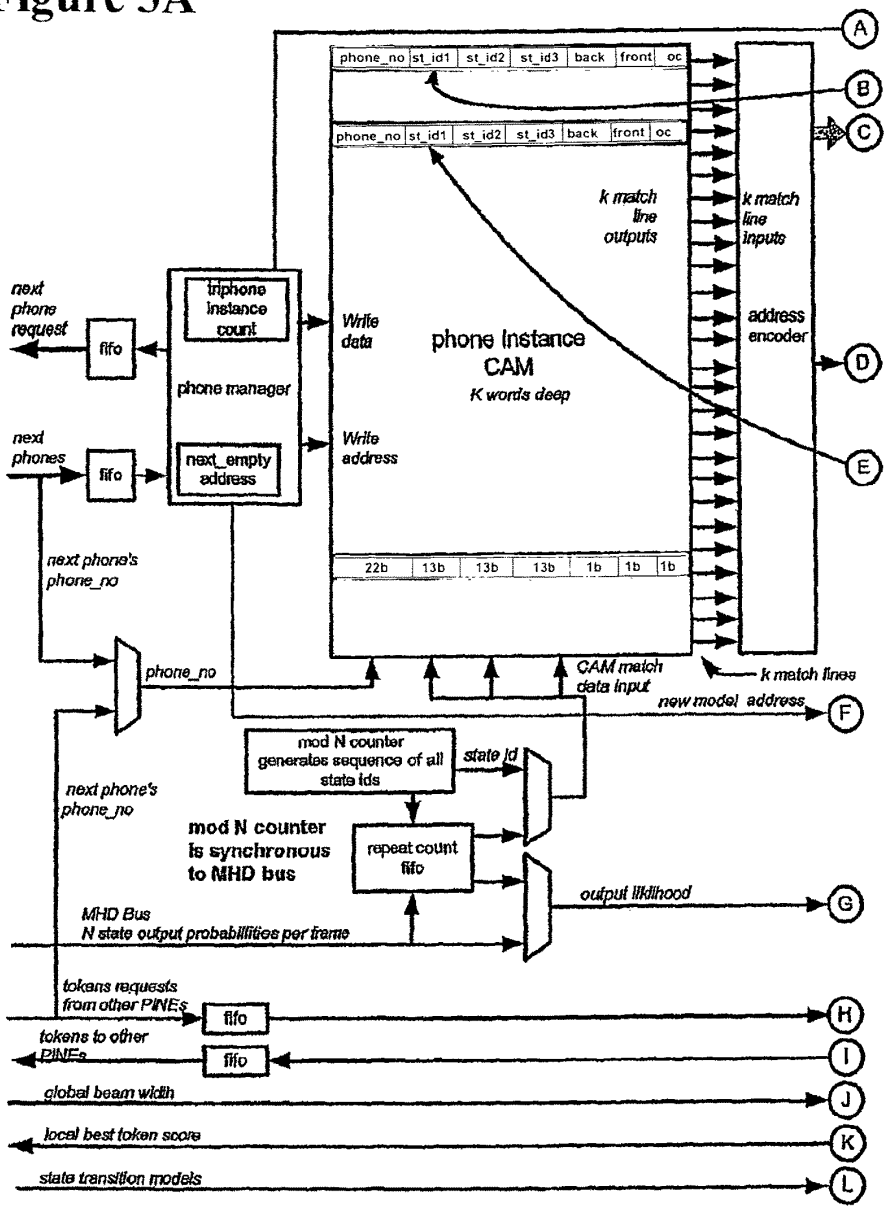
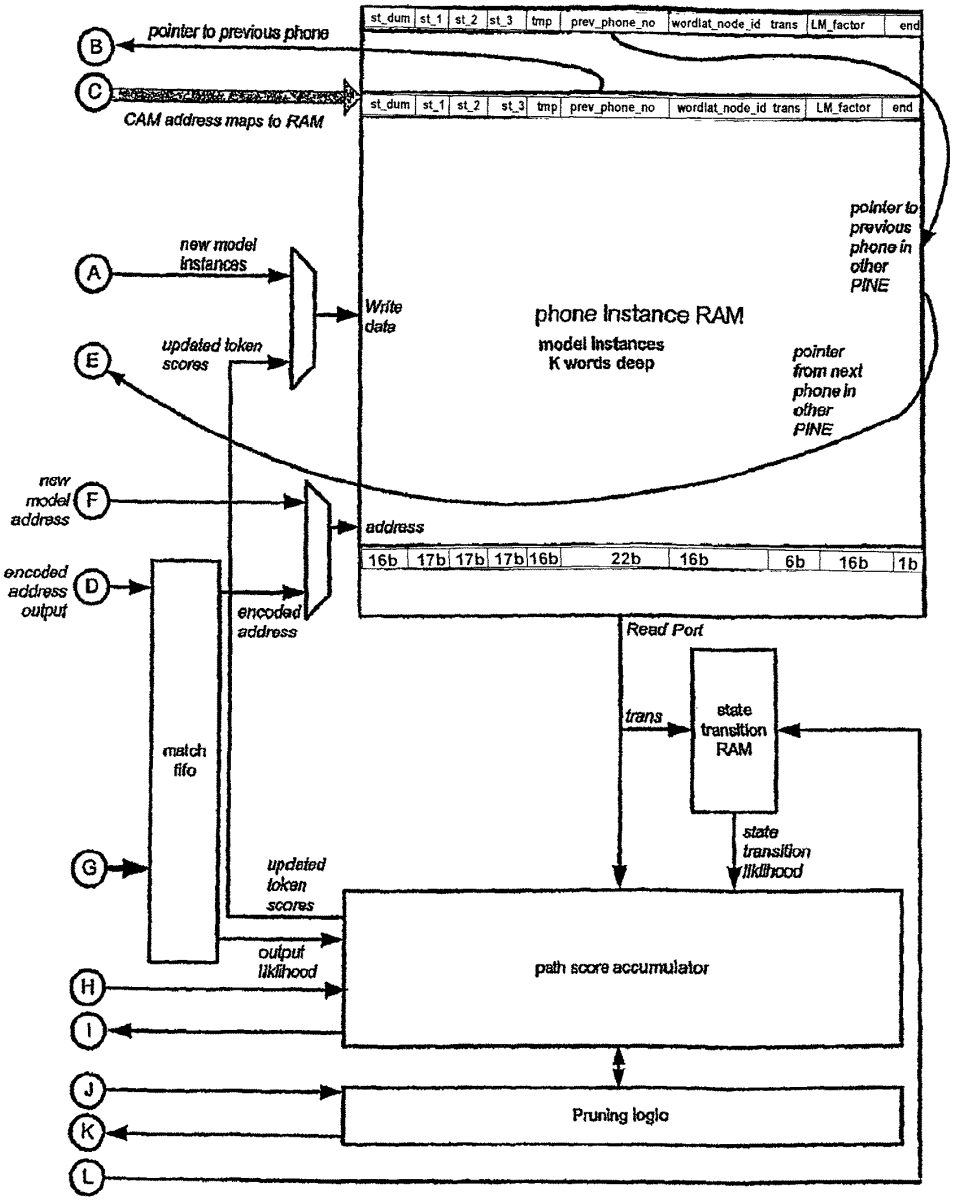

phone_no
st_id1
st_id2
st_id3
back
front
oc
k match line outputs
k match line inputs
address encoder
phone instance CAM
K words deep
Write data
Write address
triphone instance count
phone manager
next_empty address
next phone request
fifo
next phones
fifo
next phone's phone_no
22b
13b
13b
13b
1b
1b
1b
phone_no
CAM match data input
k match lines
new model address
mod N counter generates sequence of all state ids
state id
next phone's phone_no
mod N counter is synchronous to MHD bus
repeat count fifo
output likilhood
MHD Bus
N state output probabilities per frame
tokens requests from other PINEs
fifo
tokens to other PINEs
fifo
global beam width
local best token score
state transition models
A
B
C
D
E
F
G
H
I
J
K
L

Figure 3B


st_dum st_1 st_2 st_3 tmp prev_phone_no wordlat_node_id trans LM_factor end
pointer to previous phone
B
C
CAM address maps to RAM
pointer to previous phone in other PINE
new model instances
A
Write data
phone instance RAM
model instances
K words deep
updated token scores
E
pointer from next phone in other PINE
new model address
F
address
16b 17b 17b 17b 16b 22b 16b 6b 16b 1b
encoded address output
D
encoded address
Read Port
trans
state transition RAM
match fifo
state transition likilhood
G
updated token scores
output likilhood
path score accumulator
H
I
J
K
Pruning logic
L

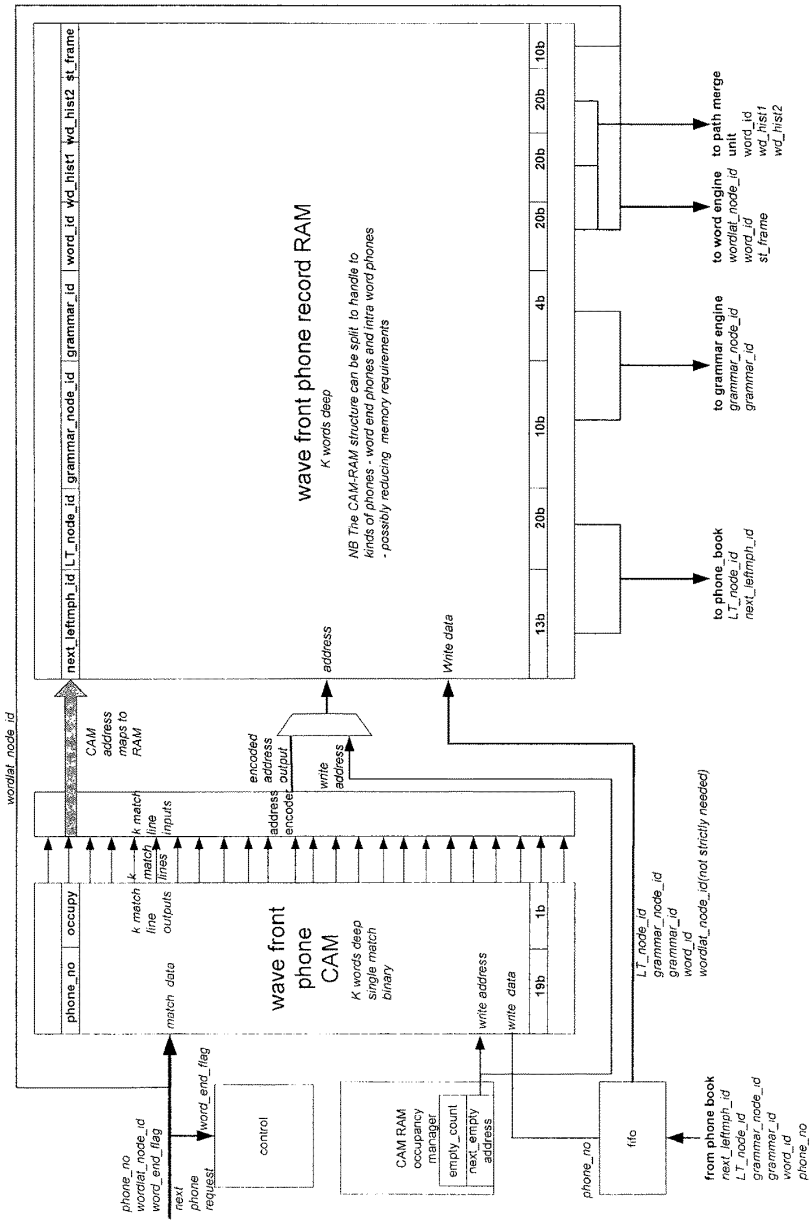
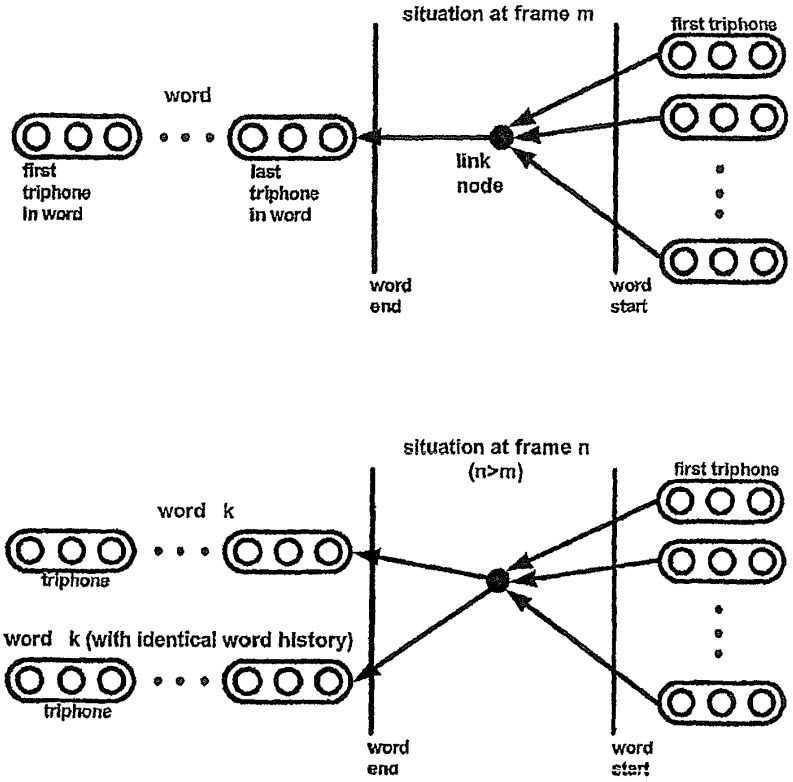

wave front phone record RAM
K words deep
NB Trie CAM-RAM structure can be split to handle to kinds of phones - word end phones and intra word phones - possibly reducing memory requirements
next_leftmph_id
LT_node_id
grammar_node_id
grammar_id
word_id
wd_hist1
wd_hist2
st_frame
13b
20b
10b
4b
20b
20b
20b
20b
address
Write data
wave front phone CAM
K words deep
single match binary
phone_no
occupy
match data
write address
write data
1b
19b
k match line outputs
k match line inputs
CAM address maps to RAM
encoded address output
address encoder
write address
wordlat_node_id
LT_node_id
grammar_node_id
grammar_id
word_id
wordlat_node_id(not strictly needed)
to phone_book
LT_node_id
next_leftmph_id
to grammar engine
grammar_node_id
grammar_id
to word engine
wordlat_node_id
word_id
st_frame
to path merge unit
word_id
wd_hist1
wd_hist2
control
CAM RAM occupancy manager
empty_count
next_empty address
fifo
phone_no
word_end_flag
next phone request
from phone book
next_leftmph_id
LT_node_id
grammar_node_id
grammar_id
word_id
phone_no

Figure 4

Figure 5


situation at frame m
first triphone
word
first triphone in word
last triphone in word
link node
word end
word start
situation at frame n (n>m)
first triphone
word k
triphone
word k (with identical word history)
triphone
word end
word start

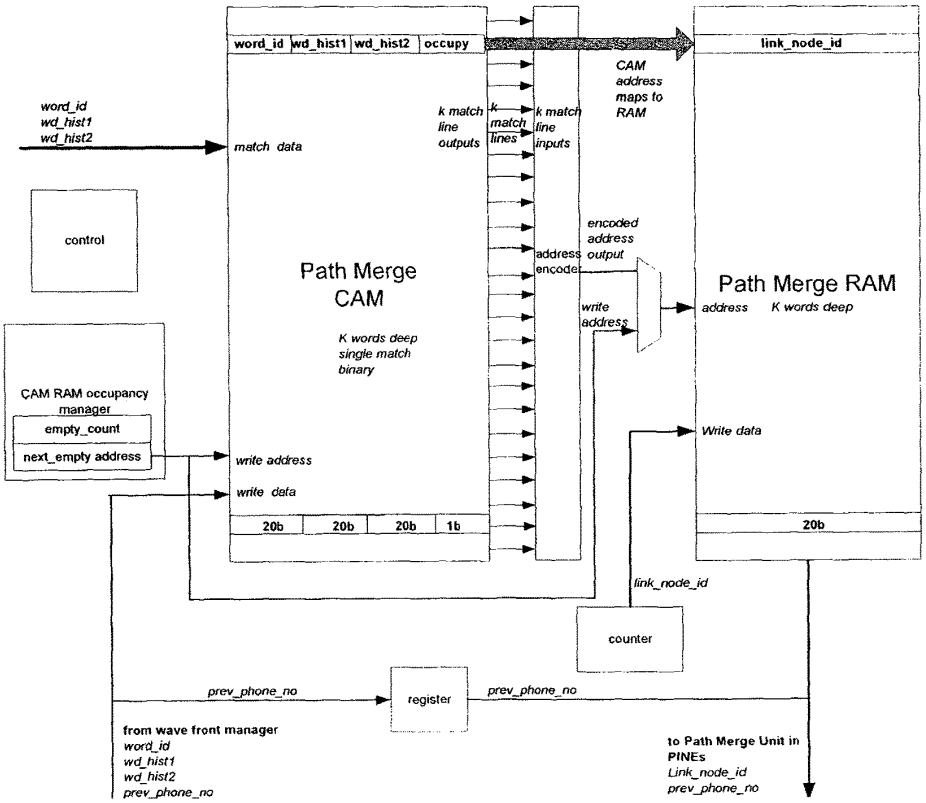

word_id
wd_hist1
wd_hist2
occupy
link_node_id
CAM address maps to RAM
word_id
wd_hist1
wd_hist2
match data
k match line outputs
k match lines
k match line inputs
control
encoded address output
address encoder
Path Merge CAM
K words deep
single match
binary
write address
Path Merge RAM
address
K words deep
CAM RAM occupancy manager
empty_count
next_empty address
write address
write data
Write data
20b
20b
20b
1b
20b
link_node_id
counter
prev_phone_no
register
prev_phone_no
from wave front manager
word_id
wd_hist1
wd_hist2
prev_phone_no
to Path Merge Unit in PINEs
Link_node_id
prev_phone_no

Figure 6

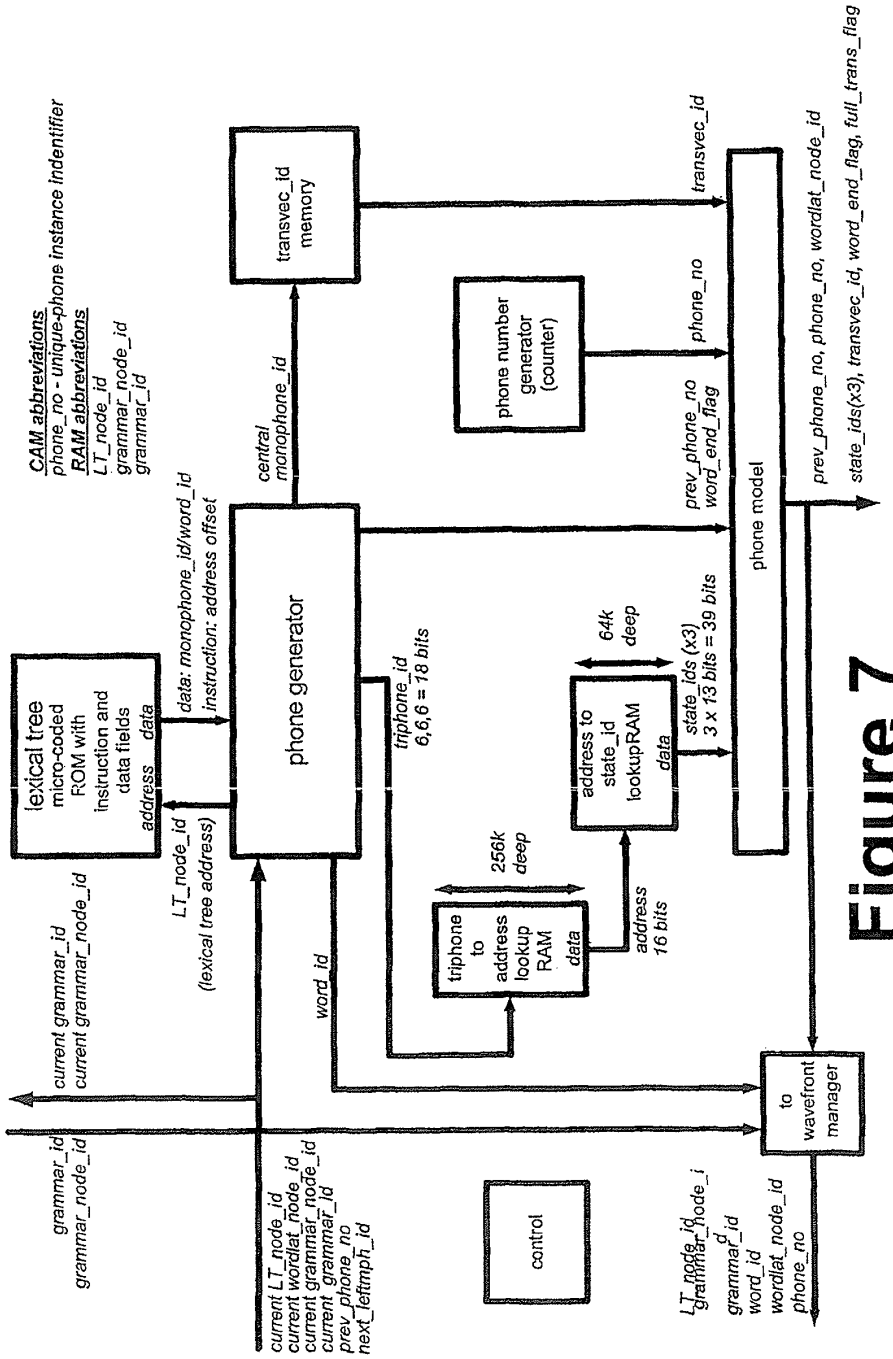

CAM abbreviations
phone_no - unique-phone instance indentifier
RAM abbreviations
LT_node_id
grammar_node_id
grammar_id
lexical tree
micro-coded
ROM with
instruction and
data fields
address data
data: monophone_id/word_id
instruction: address offset
LT_node_id
(lexical tree address)
phone generator
central
monophone_id
transvec_id
memory
phone number
generator
(counter)
phone_no
prev_phone_no
word_end_flag
phone model
transvec_id
prev_phone_no, phone_no, wordlat_node_id
state_ids(x3), transvec_id, word_end_flag, full_trans_flag
triphone_id
6,6,6 = 18 bits
256k
deep
triphone
to
address
lookup
RAM
data
address
16 bits
address to
state_id
lookupRAM
data
64k
deep
state_ids (x3)
3 x 13 bits = 39 bits
word_id
current grammar_id
current grammar_node_id
grammar_id
grammar_node_id
current LT_node_id
current wordlat_node_id
current grammar_node_id
current grammar_id
prev_phone_no
next_leftmph_id
control
to
wavefront
manager
LT_node_id
grammar_id
grammar_id
word_id
wordlat_node_id
phone_no

Figure 7

Figure 8A

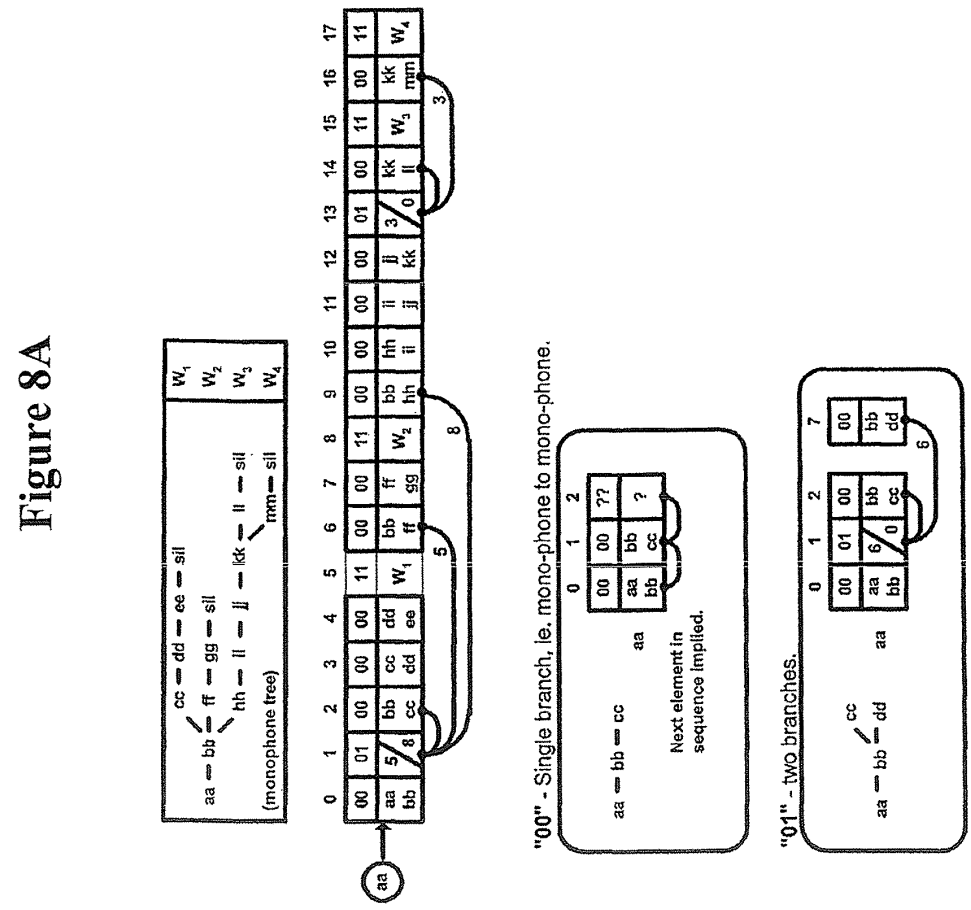

aa
bb
aa
bb
00
01
5
8
bb
cc
cc
dd
00
00
dd
ee
00
11
W1
5
bb
ff
00
00
ff
gg
00
11
W2
8
bb
hh
00
00
hh
ii
00
00
ii
jj
00
00
jj
kk
00
01
3
0
kk
ll
00
11
W3
3
kk
mm
00
11
W4
W1
W2
W3
W4
cc — dd — ee — sil
aa — bb
ff — gg — sil
hh — ll — jj — kk — ll — sil
(monophone tree)
mm — sil
"00" - Single branch, ie. mono-phone to mono-phone.
aa — bb — cc
aa
Next element in sequence implied.
00
aa
bb
00
bb
cc
??
?
"01" - two branches.
cc
aa — bb — dd
aa
00
aa
bb
01
6
0
bb
cc
00
bb
dd
6

Figure 8B

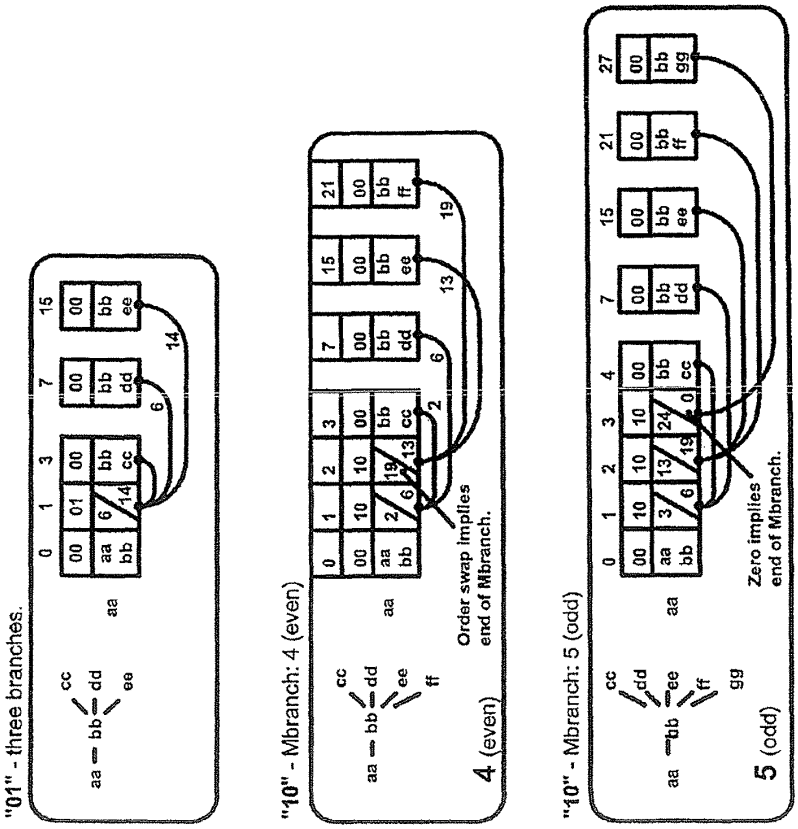

"01" - three branches.
"10" - Mbranch: 4 (even)
Order swap implies end of Mbranch.
4 (even)
"10" - Mbranch: 5 (odd)
Zero implies end of Mbranch.
5 (odd)

Figure 9

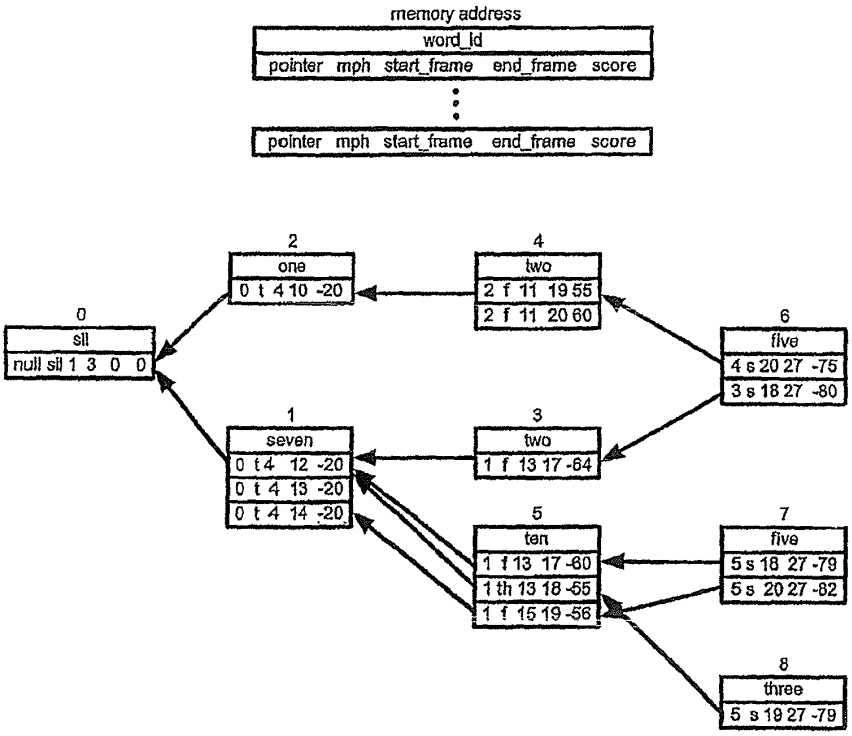
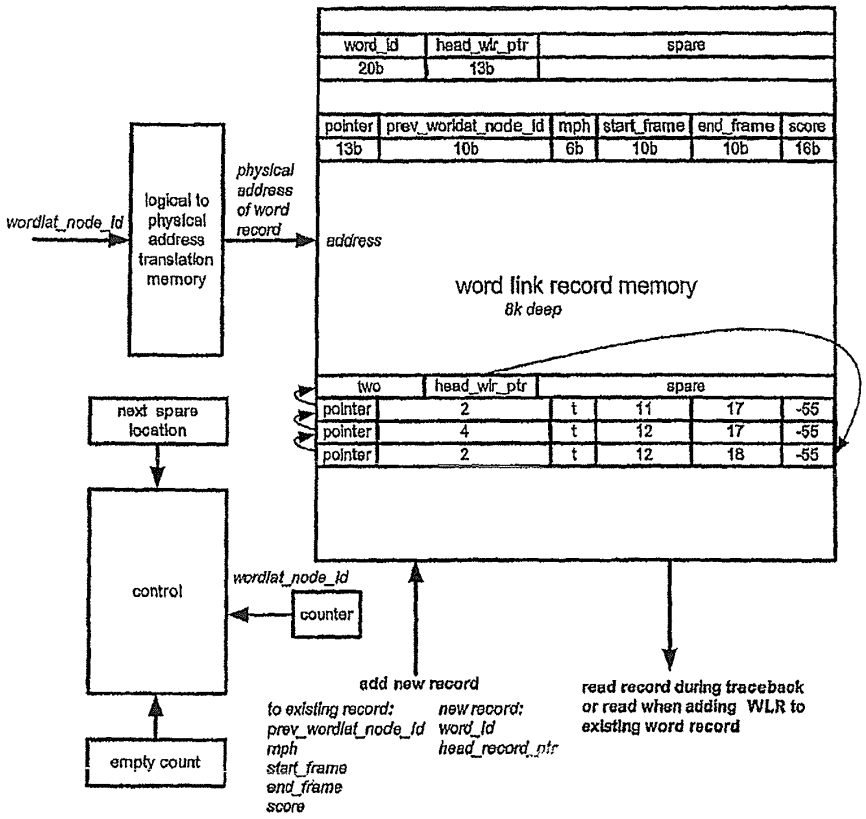

memory address
word_id
pointer mph start_frame end_frame score
pointer mph start_frame end_frame score
0
sll
null sll 1 3 0 0
2
one
0 t 4 10 -20
4
two
2 f 11 19 55
2 f 11 20 60
6
five
4 s 20 27 -75
3 s 18 27 -80
1
seven
0 t 4 12 -20
0 t 4 13 -20
0 t 4 14 -20
3
two
1 f 13 17 -64
5
ten
1 f 13 17 -60
1 th 13 18 -55
1 f 15 19 -56
7
five
5 s 18 27 -79
5 s 20 27 -82
8
three
5 s 19 27 -79

Figure 10


word_id
head_wlr_ptr
spare
20b
13b
pointer
prev_worldat_node_id
mph
start_frame
end_frame
score
13b
10b
6b
10b
10b
16b
wordlat_node_id
logical to physical address translation memory
physical address of word record
address
word link record memory
8k deep
two
head_wlr_ptr
spare
pointer
2
t
11
17
-55
pointer
4
t
12
17
-55
pointer
2
t
12
18
-55
next spare location
control
wordlat_node_id
counter
empty count
add new record
to existing record:
prev_wordlat_node_id
mph
start_frame
end_frame
score
new record:
word_id
head_record_ptr
read record during traceback or read when adding WLR to existing word record

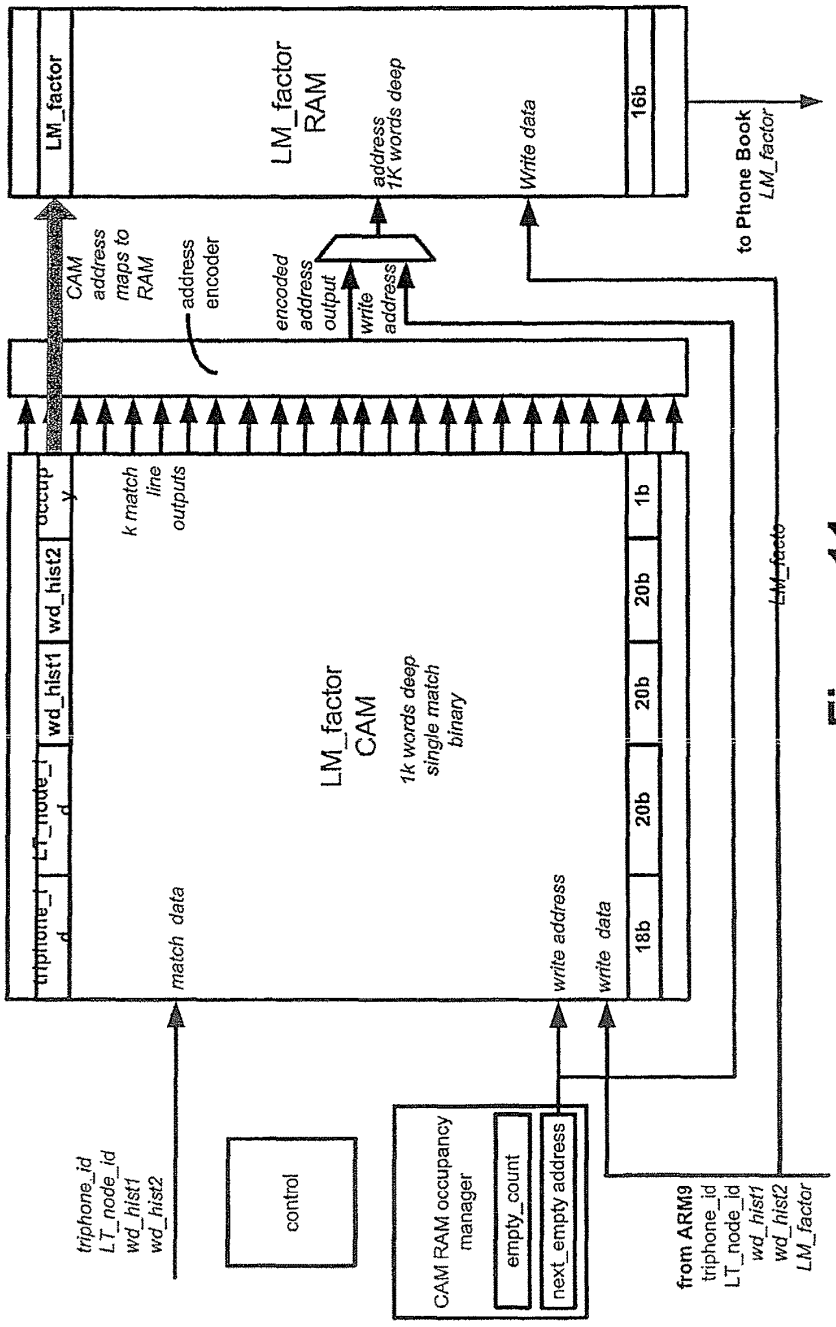

LM_factor
LM_factor RAM
address 1K words deep
Write data
16b
to Phone Book LM_factor
CAM address maps to RAM
address encoder
encoded address output
write address
triphone_id
LT_node_id
wd_hist1
wd_hist2
occupancy
k match line outputs
LM_factor CAM
1k words deep single match binary
1b
20b
20b
20b
18b
match data
write address
write data
LM_factor
triphone_id
LT_node_id
wd_hist1
wd_hist2
control
CAM RAM occupancy manager
empty_count
next_empty address
from ARM9
triphone_id
LT_node_id
wd_hist1
wd_hist2
LM_factor

Figure 11

Figure 12

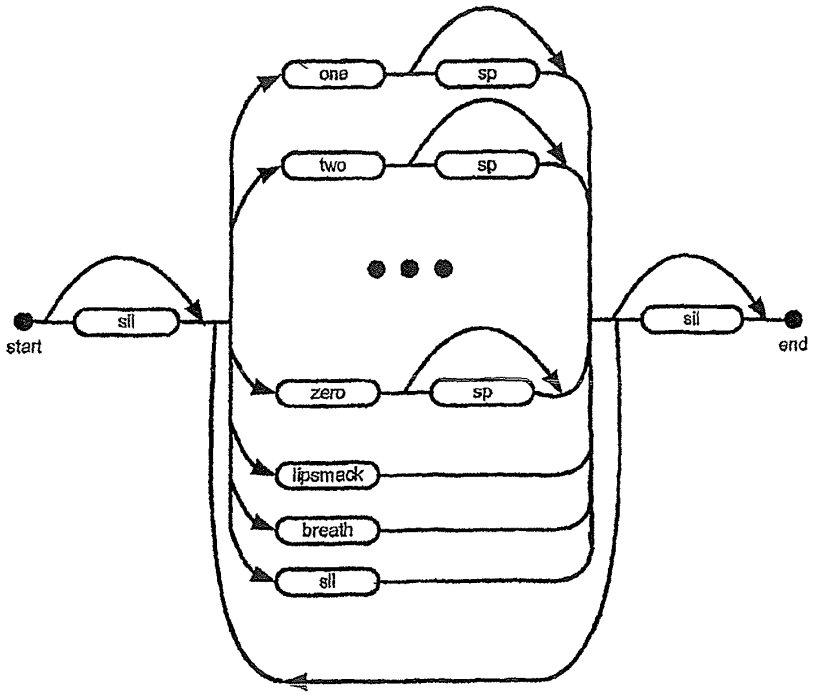
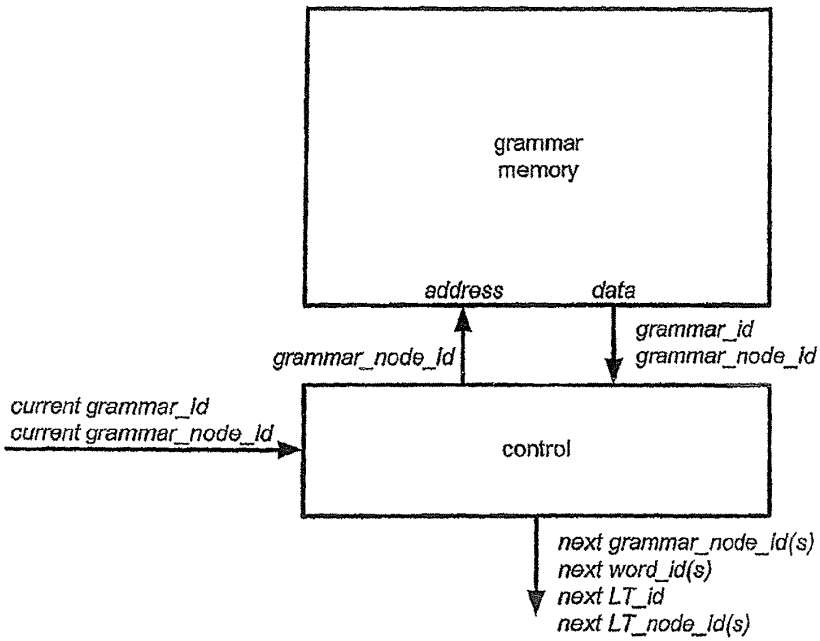


Figure 13


grammar
memory
address
data
grammar_id
grammar_node_id
grammar_node_id
current grammar_id
current grammar_node_id
control
next grammar_node_id(s)
next word_id(s)
next LT_id
next LT_node_id(s)

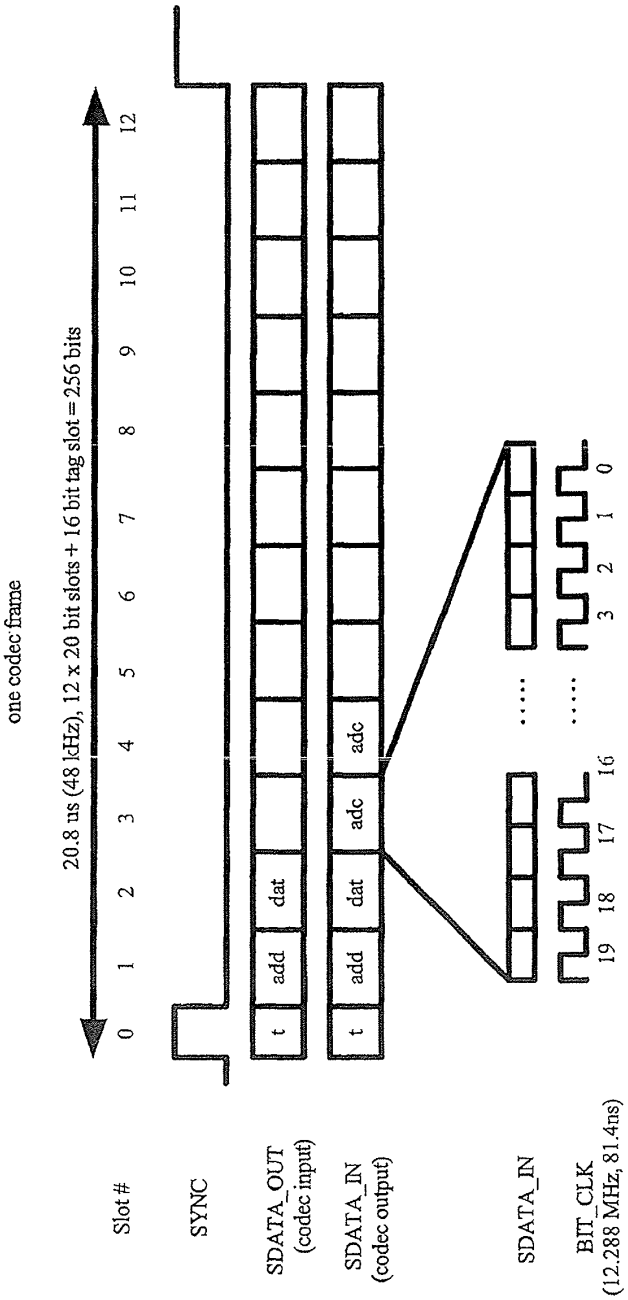
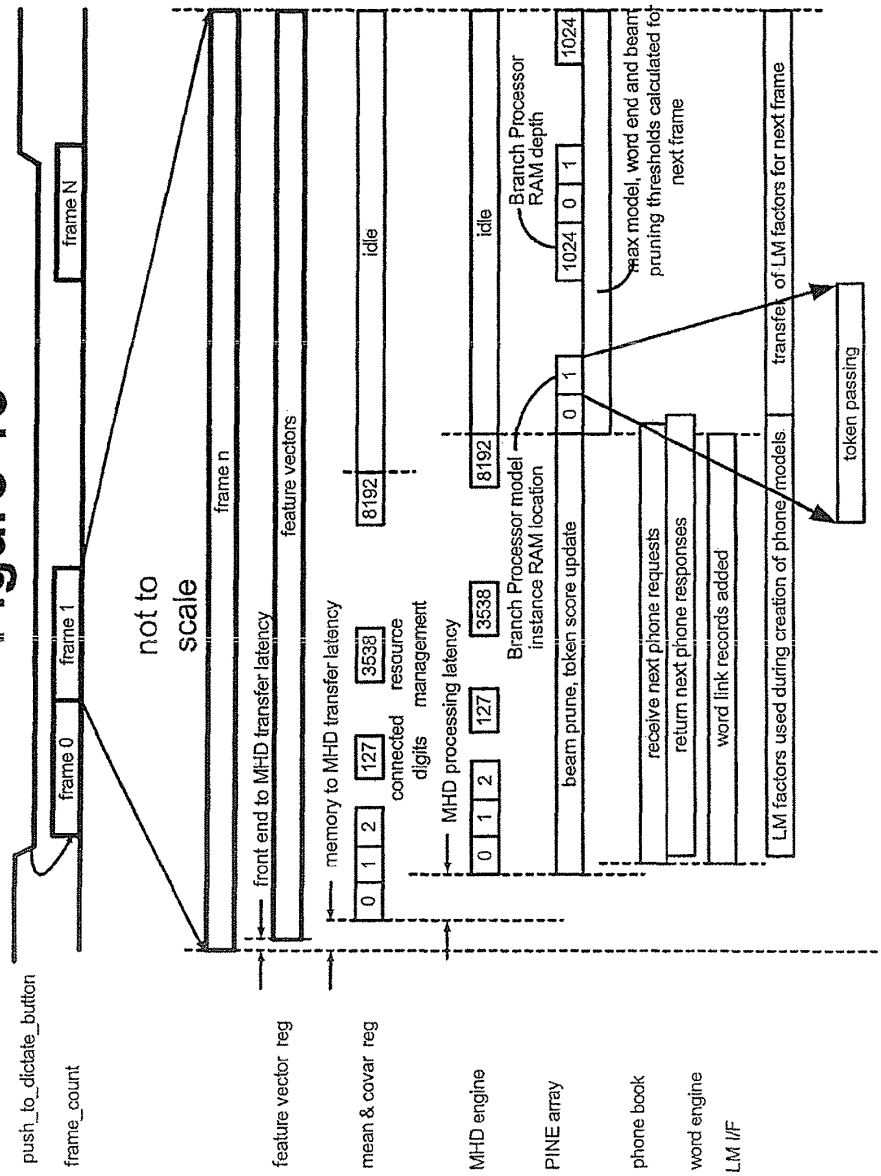

one codec frame
20.8 us (48 kHz), 12 x 20 bit slots + 16 bit tag slot = 256 bits
Slot #
SYNC
SDATA_OUT (codec input)
SDATA_IN (codec output)
SDATA_IN
BIT_CLK (12.288 MHz, 81.4ns)
t
add
dat
adc

Figure 14

Figure 15


not to scale
frame 0
frame 1
frame N
frame n
feature vectors
front end to MHD transfer latency
memory to MHD transfer latency
connected digits
resource management
MHD processing latency
Branch Processor model instance RAM location
Branch Processor RAM depth
beam prune, token score update
receive next phone requests
return next phone responses
word link records added
LM factors used during creation of phone models
transfer of LM factors for next frame
token passing
idle
idle
push_to_dictate_button
frame_count
feature vector reg
mean & covar reg
MHD engine
PINE array
phone book
word engine
LM I/F

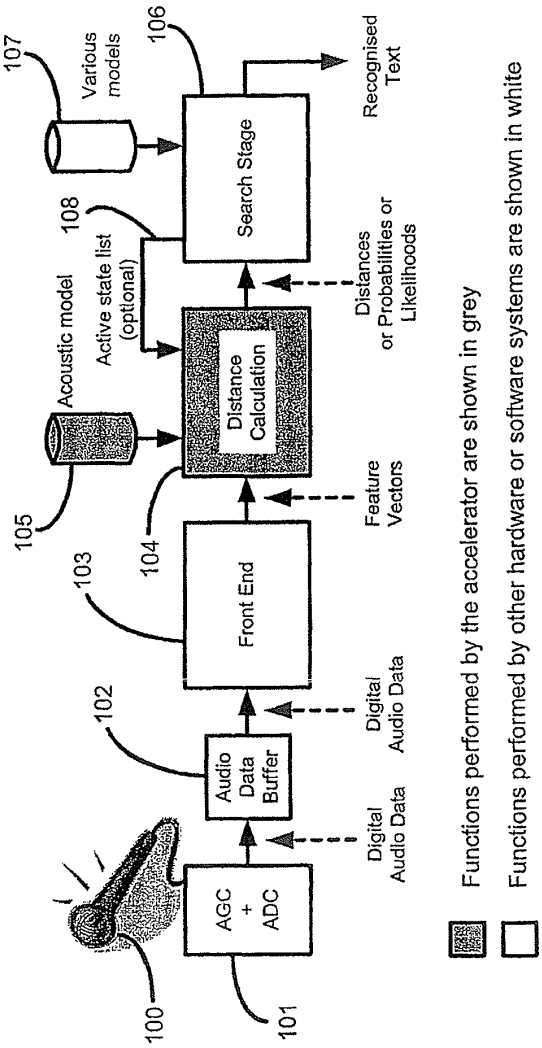
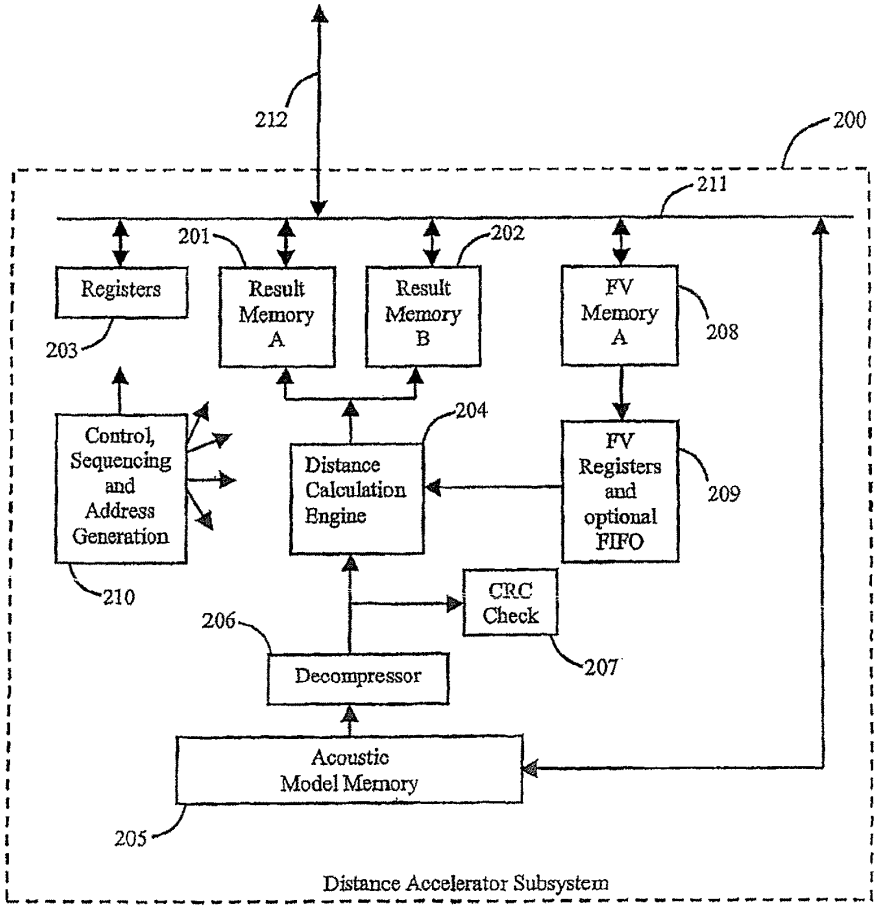

100
101
AGC + ADC
Digital Audio Data
102
Audio Data Buffer
Digital Audio Data
103
Front End
104
Feature Vectors
105
Acoustic model
Active state list (optional)
108
Distance Calculation
Distances or Probabilities or Likelihoods
107
Various models
106
Search Stage
Recognised Text
Functions performed by the accelerator are shown in grey
Functions performed by other hardware or software systems are shown in white

Figure 16

FIGURE 17


212
200
211
201
202
Registers
Result Memory A
Result Memory B
FV Memory A
208
203
Control, Sequencing and Address Generation
204
Distance Calculation Engine
FV Registers and optional FIFO
209
210
CRC Check
206
207
Decompressor
Acoustic Model Memory
205
Distance Accelerator Subsystem

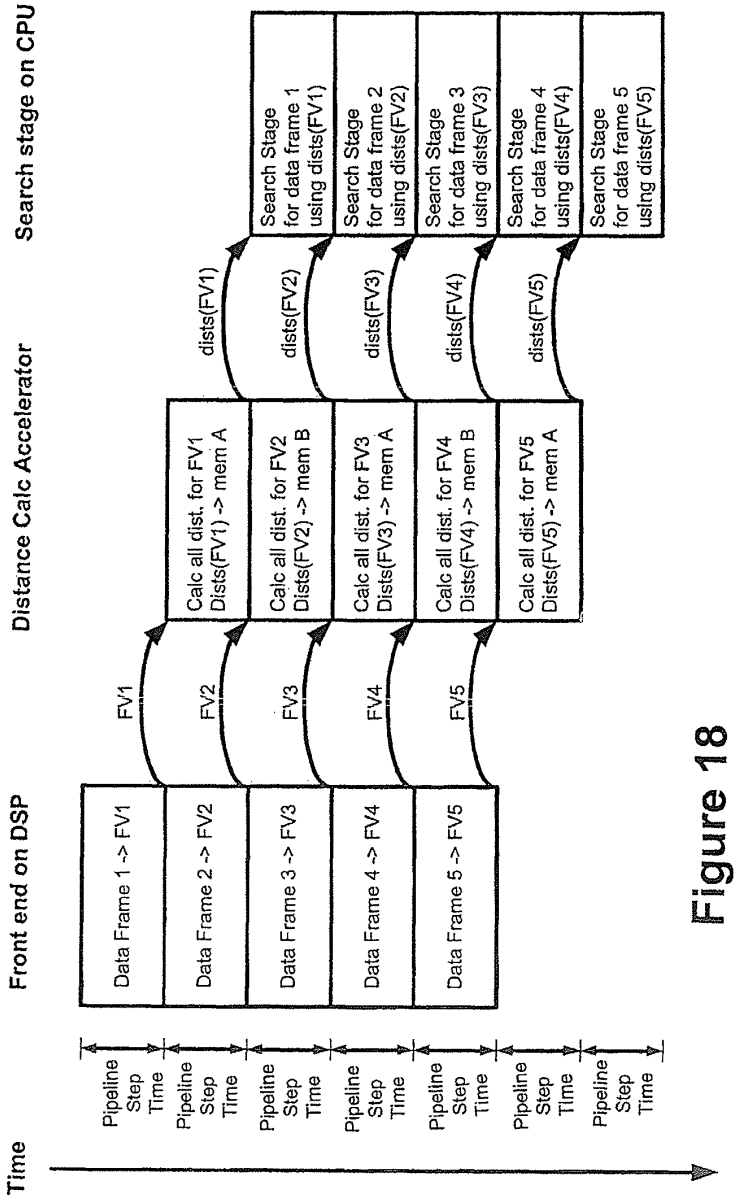

Time
Front end on DSP
Distance Calc Accelerator
Search stage on CPU
Pipeline Step Time
Data Frame 1 -> FV1
Data Frame 2 -> FV2
Data Frame 3 -> FV3
Data Frame 4 -> FV4
Data Frame 5 -> FV5
FV1
FV2
FV3
FV4
FV5
Calc all dist. for FV1 Dists(FV1) -> mem A
Calc all dist. for FV2 Dists(FV2) -> mem B
Calc all dist. for FV3 Dists(FV3) -> mem A
Calc all dist. for FV4 Dists(FV4) -> mem B
Calc all dist. for FV5 Dists(FV5) -> mem A
dists(FV1)
dists(FV2)
dists(FV3)
dists(FV4)
dists(FV5)
Search Stage for data frame 1 using dists(FV1)
Search Stage for data frame 2 using dists(FV2)
Search Stage for data frame 3 using dists(FV3)
Search Stage for data frame 4 using dists(FV4)
Search Stage for data frame 5 using dists(FV5)

Figure 18

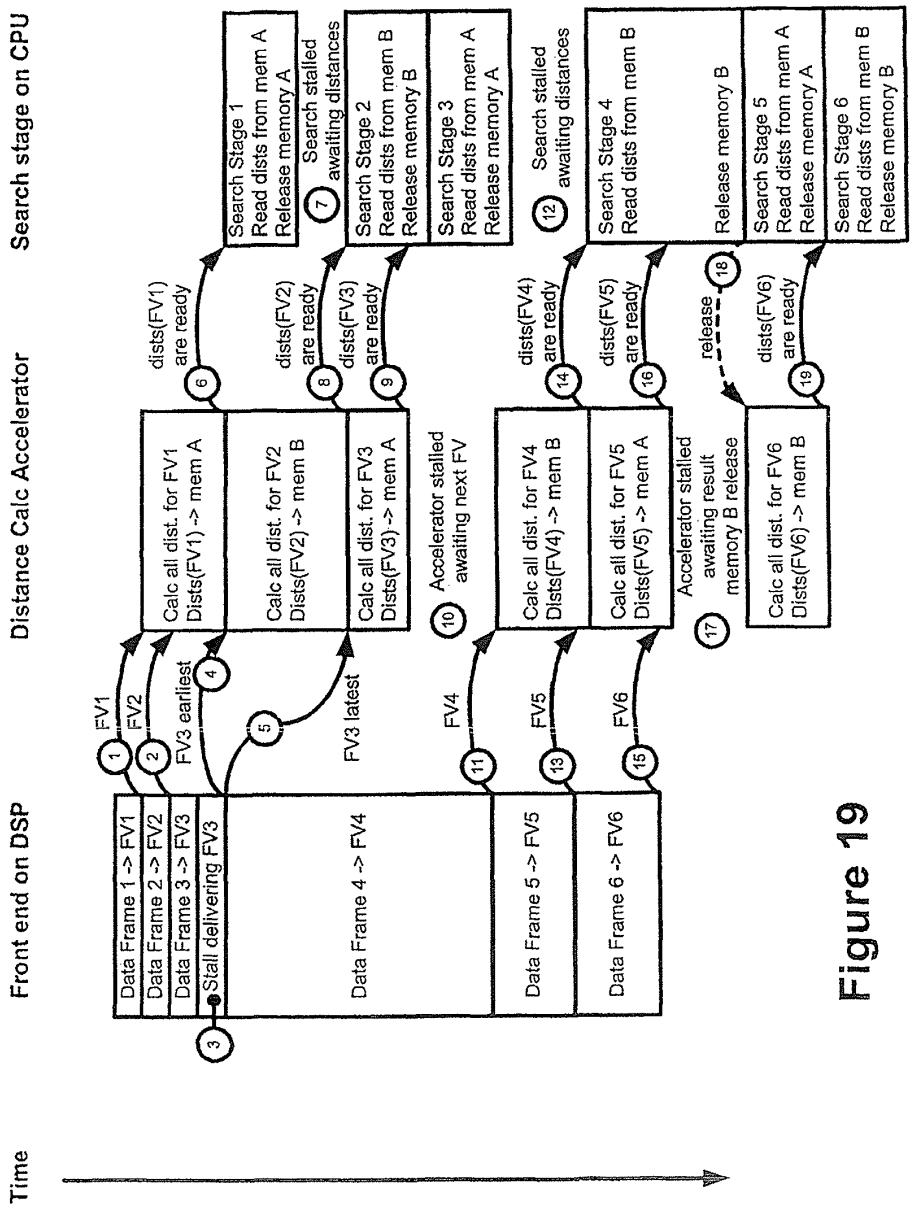

Time
Front end on DSP
Distance Calc Accelerator
Search stage on CPU
Data Frame 1 -> FV1
Data Frame 2 -> FV2
Data Frame 3 -> FV3
Stall delivering FV3
Data Frame 4 -> FV4
Data Frame 5 -> FV5
Data Frame 6 -> FV6
FV1
FV2
FV3 earliest
FV3 latest
FV4
FV5
FV6
Calc all dist. for FV1 Dists(FV1) -> mem A
Calc all dist. for FV2 Dists(FV2) -> mem B
Calc all dist. for FV3 Dists(FV3) -> mem A
Accelerator stalled awaiting next FV
Calc all dist. for FV4 Dists(FV4) -> mem B
Calc all dist. for FV5 Dists(FV5) -> mem A
Accelerator stalled awaiting result memory B release
Calc all dist. for FV6 Dists(FV6) -> mem B
dists(FV1) are ready
dists(FV2) are ready
dists(FV3) are ready
dists(FV4) are ready
dists(FV5) are ready
release
dists(FV6) are ready
Search Stage 1 Read dists from mem A Release memory A
Search stalled awaiting distances
Search Stage 2 Read dists from mem B Release memory B
Search Stage 3 Read dists from mem A Release memory A
Search stalled awaiting distances
Search Stage 4 Read dists from mem B
Release memory B
Search Stage 5 Read dists from mem A Release memory A
Search Stage 6 Read dists from mem B Release memory B
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

Figure 19

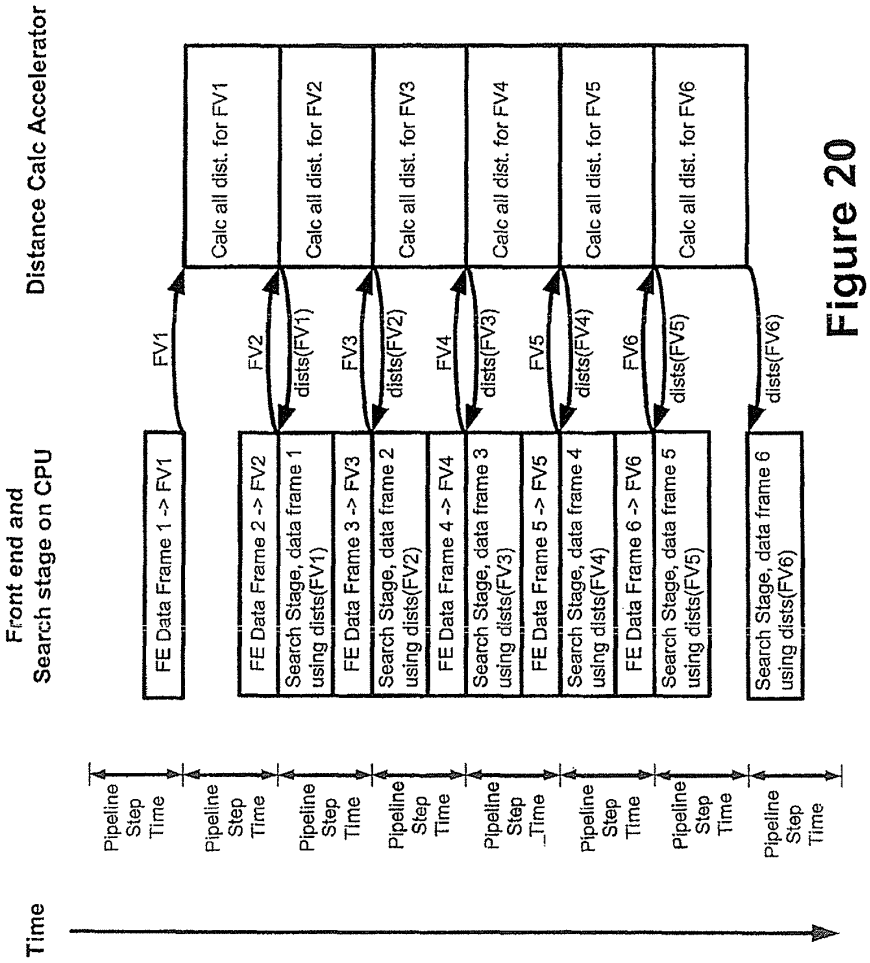
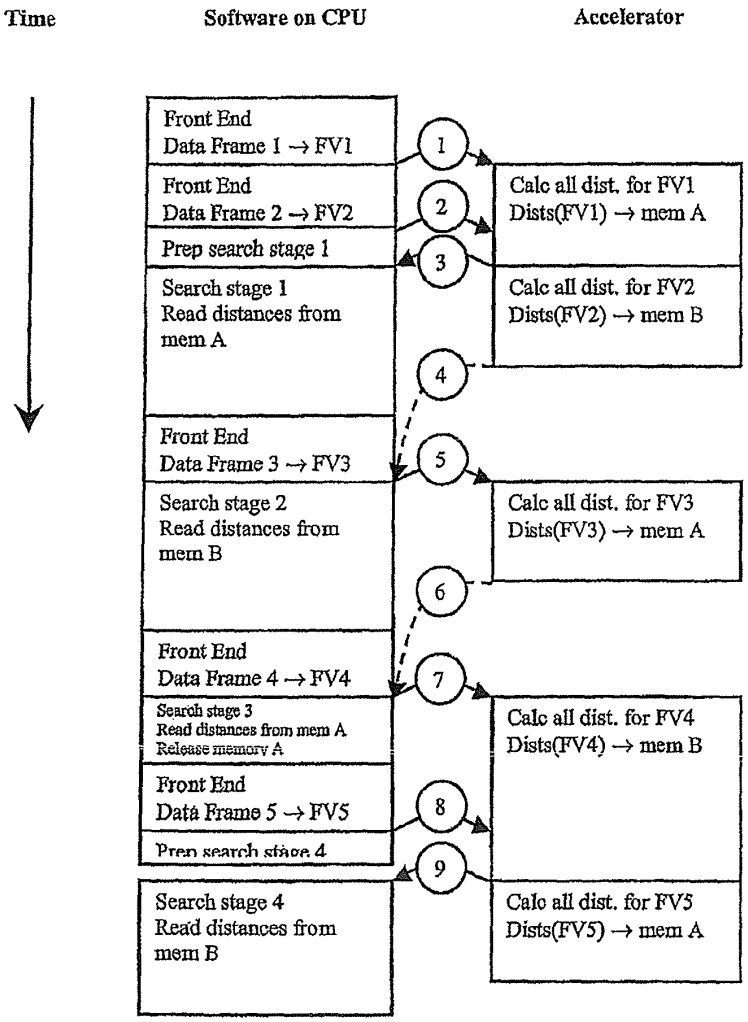

Front end and Search stage on CPU
Distance Calc Accelerator
Time
Pipeline Step Time
FE Data Frame 1 -> FV1
FE Data Frame 2 -> FV2
Search Stage, data frame 1 using dists(FV1)
FE Data Frame 3 -> FV3
Search Stage, data frame 2 using dists(FV2)
FE Data Frame 4 -> FV4
Search Stage, data frame 3 using dists(FV3)
FE Data Frame 5 -> FV5
Search Stage, data frame 4 using dists(FV4)
FE Data Frame 6 -> FV6
Search Stage, data frame 5 using dists(FV5)
Search Stage, data frame 6 using dists(FV6)
FV1
FV2
FV3
FV4
FV5
FV6
dists(FV1)
dists(FV2)
dists(FV3)
dists(FV4)
dists(FV5)
dists(FV6)
Calc all dist. for FV1
Calc all dist. for FV2
Calc all dist. for FV3
Calc all dist. for FV4
Calc all dist. for FV5
Calc all dist. for FV6

Figure 20

FIGURE 21


Time
Software on CPU
Accelerator
Front End
Data Frame 1 → FV1
Front End
Data Frame 2 → FV2
Prep search stage 1
Search stage 1
Read distances from
mem A
Front End
Data Frame 3 → FV3
Search stage 2
Read distances from
mem B
Front End
Data Frame 4 → FV4
Search stage 3
Read distances from mem A
Release memory A
Front End
Data Frame 5 → FV5
Prep search stage 4
Search stage 4
Read distances from
mem B
1
2
3
4
5
6
7
8
9
Calc all dist. for FV1
Dists(FV1) → mem A
Calc all dist. for FV2
Dists(FV2) → mem B
Calc all dist. for FV3
Dists(FV3) → mem A
Calc all dist. for FV4
Dists(FV4) → mem B
Calc all dist. for FV5
Dists(FV5) → mem A

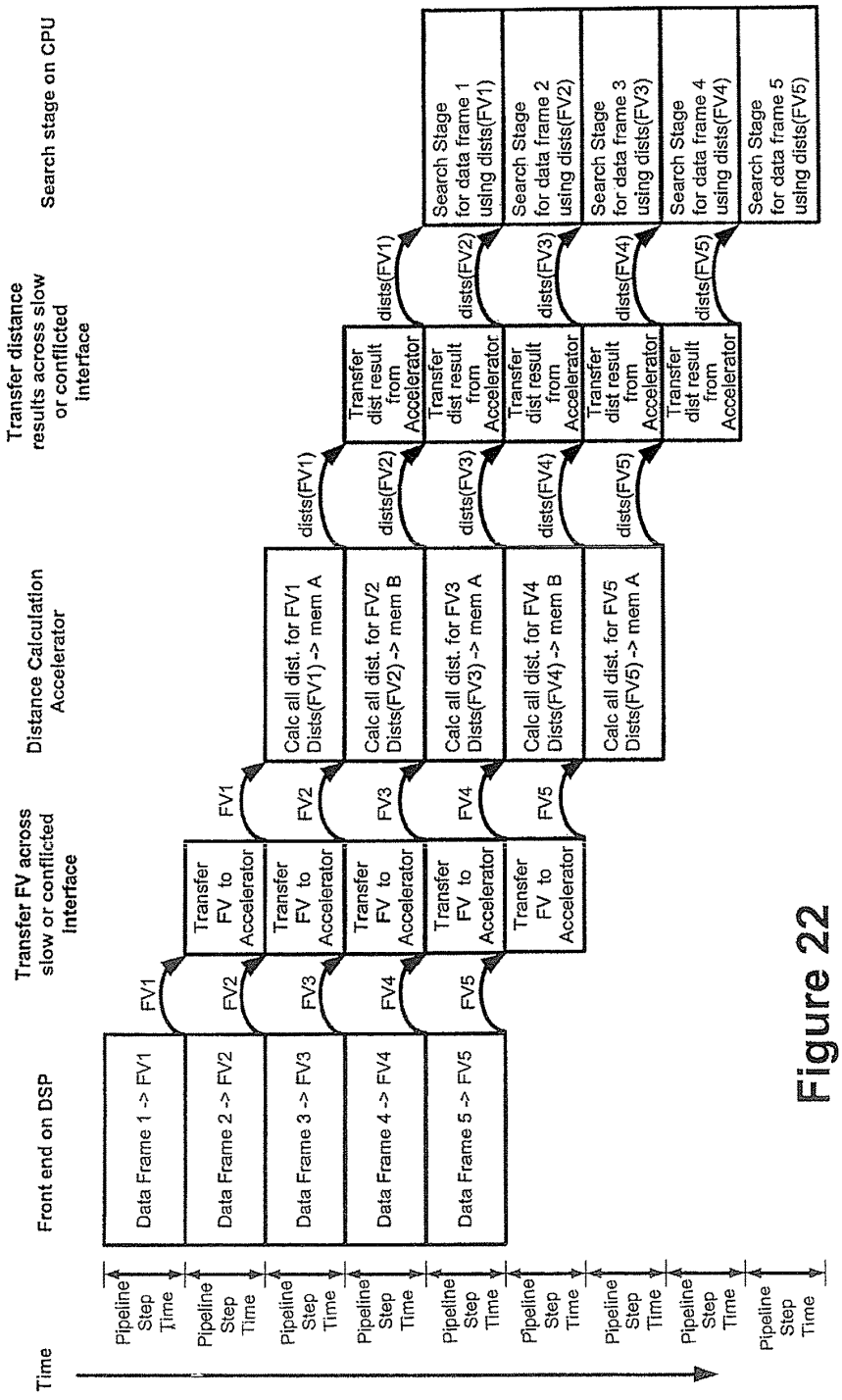

Time
Front end on DSP
Transfer FV across slow or conflicted interface
Distance Calculation Accelerator
Transfer distance results across slow or conflicted interface
Search stage on CPU
Pipeline Step Time
Data Frame 1 -> FV1
Data Frame 2 -> FV2
Data Frame 3 -> FV3
Data Frame 4 -> FV4
Data Frame 5 -> FV5
FV1
FV2
FV3
FV4
FV5
Transfer FV to Accelerator
Calc all dist. for FV1 Dists(FV1) -> mem A
Calc all dist. for FV2 Dists(FV2) -> mem B
Calc all dist. for FV3 Dists(FV3) -> mem A
Calc all dist. for FV4 Dists(FV4) -> mem B
Calc all dist. for FV5 Dists(FV5) -> mem A
dists(FV1)
dists(FV2)
dists(FV3)
dists(FV4)
dists(FV5)
Transfer dist result from Accelerator
Search Stage for data frame 1 using dists(FV1)
Search Stage for data frame 2 using dists(FV2)
Search Stage for data frame 3 using dists(FV3)
Search Stage for data frame 4 using dists(FV4)
Search Stage for data frame 5 using dists(FV5)

Figure 22

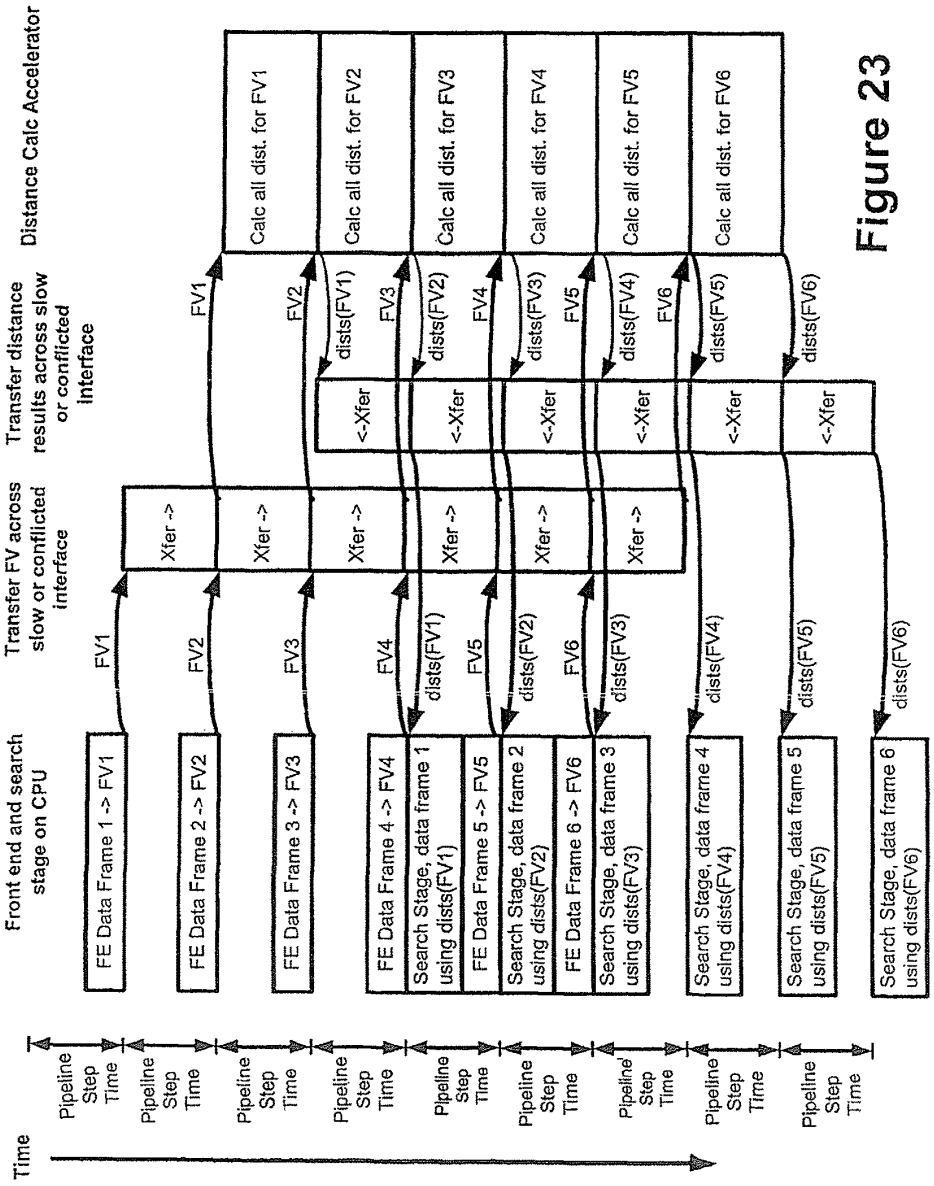

Front end and search stage on CPU
Transfer FV across slow or conflicted interface
Transfer distance results across slow or conflicted interface
Distance Calc Accelerator
Time
Pipeline Step Time
FE Data Frame 1 -> FV1
FE Data Frame 2 -> FV2
FE Data Frame 3 -> FV3
FE Data Frame 4 -> FV4
FE Data Frame 5 -> FV5
FE Data Frame 6 -> FV6
Search Stage, data frame 1 using dists(FV1)
Search Stage, data frame 2 using dists(FV2)
Search Stage, data frame 3 using dists(FV3)
Search Stage, data frame 4 using dists(FV4)
Search Stage, data frame 5 using dists(FV5)
Search Stage, data frame 6 using dists(FV6)
FV1
FV2
FV3
FV4
FV5
FV6
Xfer ->
<-Xfer
dists(FV1)
dists(FV2)
dists(FV3)
dists(FV4)
dists(FV5)
dists(FV6)
Calc all dist. for FV1
Calc all dist. for FV2
Calc all dist. for FV3
Calc all dist. for FV4
Calc all dist. for FV5
Calc all dist. for FV6

Figure 23

DISTRIBUTED PIPELINED PARALLEL SPEECH RECOGNITION SYSTEM

This is a continuation of application Ser. No. 13/735,091, filed Jan. 7, 2013, now U.S. Pat. No. 9,076,441, which is a continuation of application Ser. No. 13/162,128, filed Jun. 16, 2011, now U.S. Pat. No. 8,352,262, which is a divisional of application Ser. No. 11/662,704, filed Mar. 14, 2007, now U.S. Pat. No. 7,979,277, which is a 371 of PCT/GB2005/003554, filed Sep. 14, 2005, which claims priority to UK Application No. 0420464.0, filed Sep. 14, 2004, the disclosures of which are hereby incorporated by reference in their entireties.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to speech recognition circuits and methods. These circuits and methods have wide applicability, particularly for devices such as mobile electronic devices.

2. Description of the Related Art

There is growing consumer demand for embedded speech recognition in mobile electronic devices, such as mobile phones, dictation machines, PDAs (personal digital assistants), mobile games consoles, etc. For example, email and text message dictation, note taking, form filling, and command and control applications are all potential applications of embedded speech recognition.

However, when a medium to large vocabulary is required, effective speech recognition for mobile electronic devices has many difficulties not associated with speech recognition systems in hardware systems such as personal computers or workstations. Firstly, the available power in mobile systems is often supplied by battery, and may be severely limited. Secondly, mobile electronic devices are frequently designed to be as small as practically possible. Thus, the memory and resources of such mobile embedded systems tends to be very limited, due to power and space restrictions. The cost of providing extra memory and resources in a mobile electronic device is typically much higher than that for a less portable device without this space restriction. Thirdly, the mobile hardware may be typically used in a noisier environment than that of a fixed computer, e.g. on public transport, near a busy road, etc. Thus, a more complex speech model and more intensive computation may be required to obtain adequate speech recognition results.

These restrictions have made it difficult to implement effective speech recognition in mobile devices, other than with very limited vocabularies.

Some prior art schemes have been proposed to increase the efficiency of speech recognition systems, in an attempt to make them more suitable for use in mobile technology.

In an article entitled "A low-power accelerator for the SPHINX 3 speech recognition system", in University of Utah, International conference on Compilers, Architectures and Synthesis for Embedded Systems, November 2003, Davis et al have proposed the idea of using a special purpose co-processor for up-front calculation of the computationally expensive Gaussian output probabilities of audio frames corresponding to particular states in the acoustic model.

In an article entitled "Hardware Speech Recognition in Low Cost, Low Power Devices", University of California, Berkeley, CS252 Class Project, Spring 2003, Sukun Kim et al describe using special purpose processing elements for each of the nodes in the network to be searched. This effectively implies having a single processing element for each phone in the network. An alternative suggested by Sukun Kim et al is to provide a processor for each state in the network.

In an article entitled "Dynamic Programming Search for Continuous Speech Recognition" in IEEE Signal Processing Magazine, September 1999, Ney et al discuss language model lookahead. Language model lookahead involves computation of a language model factor for each node (i.e. phone) in the lexical tree. This technique is also known as smearing. Each phone instance in the search network can be given a language model factor when it is used in the lexical tree search. Ney et al show that for an example bigram language model, the average number of states per 10 ms frame can be reduced from around 168,000 states with no language model lookahead to around 8,000 states when language model lookahead is used. They also show that bigram language model lookahead requires about a quarter of the states compared with unigram language model lookahead.

Although these prior art documents provide improvements to speech recognition in embedded mobile technology, further improvement is still needed to provide a larger vocabulary and better accuracy.

SUMMARY OF THE INVENTION

One aspect of the present invention provides a speech recognition circuit including a circuit for providing state identifiers which identify states corresponding to nodes or groups of adjacent nodes in a lexical tree, and for providing scores corresponding to said state identifiers. The lexical tree includes a model of words. The speech recognition circuit also has a memory structure for receiving and storing state identifiers identified by a node identifier identifying nodes or groups of adjacent nodes, the memory structure being adapted to allow lookup to identify particular state identifiers, reading of the scores corresponding to the state identifiers, and writing back of the scores to the memory structure after modification of the scores. An accumulator is provided for receiving score updates corresponding to particular state identifiers from a score update generating circuit which generates the score updates using audio input, for receiving scores from the memory structure, and for modifying said scores by adding said score updates to said scores. A selector circuit is used for selecting at least one node or group of nodes of the lexical tree according to said scores.

One suitable type of hardware for the memory structure includes a content addressable memory (CAM). A CAM is a memory unit which stores a series of data items using a series of addresses. However, the memory is accessed by specifying a data item, such that the CAM returns the corresponding address. This contrasts with a random access memory (RAM) in which the memory is accessed by specifying an address, such that the RAM returns the corresponding data item.

However, the memory structure is not limited to including a CAM. Other types of hardware are also possible, to provide this functionality. For example, a single chip which operates in the same way as a CAM and RAM may be used instead.

Embodiments of the present invention provide a solution to the problem of how to map a lexical tree search to a CAM system architecture. The realisation by the present inventors that certain speech recognition data structures can be mapped into the CAMs allows a lexical tree search to be performed using a CAM system architecture.

Further embodiments of the invention include a counter for sequentially generating state identifiers, and using said generated state identifiers to sequentially lookup said states in the memory structure.

The node identifier may comprise a direct reference to the lexical tree. However, in some embodiments, the node identifier for at least some of the states includes a pointer to a node identifier for another state. For example, a state corresponding to the furthest part of the search path in the lexical tree may be referenced by a node identifier which directly links to a particular node or group of nodes in the lexical tree. In a lexical tree comprising phones, using a state model of triphones, the node identifier may indicate the position of a triphone in the lexical tree.

However, in this example, for states occurring further back in the search path, instead of supplying a node identifier linking directly to the lexical tree, instead a pointer to a node identifier of another state may be supplied. E.g. a triphone instance may have a pointer to another triphone instance, which has a pointer to another triphone instance, which has a pointer to a node or group of nodes in the lexical tree. Chains of reference may be set up in this way, where only the last state in the chain has a direct pointer to the lexical tree.

There may not be a one-to-one correspondence between the nodes of the lexical tree and the node identifiers. This will occur for a branched lexical tree, where the nodes represent monophones, but the acoustic model states represent triphones, i.e. groups of three adjacent monophones. Then, paths of three monophones will have unique identifiers to be stored in the memory structure, rather than single monophones having unique identifiers.

Phone instance numbers may be generated, and used to uniquely label each phone instance. They can be generated sequentially, using a counter. The phone instance numbers may be used as pointers between phone instances to assist in node identification. It is thus not essential to provide a direct node identifier for each phone instance to directly indicate a location in the lexical tree. The dynamic network of phone instances provided in the memory structure may thus include both direct and relative references to the lexical tree.

The memory structure may be divided into one part which stores phone instance identifiers and direct references to the lexical tree, and a second part which stores phone instance identifiers and corresponding states. This can speed up the processing, by only storing the phone instances which are furthest on in the lexical tree in the first part of the memory structure.

The memory structure may also be divided into separately accessible units, to reduce the amount to data in each unit, thereby decreasing the chance of finding the same two states identifiers in different phone instances in any single memory unit, and increasing the chance of some state identifiers being completely absent from any single memory unit. This makes it easier to deal with the situation when the same two state identifiers are found, because a spare time slot is available for processing when a state identifier is not present.

A further aspect of the invention provides a distance calculation engine within a speech recognition system. The distance calculation engine may be included within an accelerator. The accelerator may include logic to interface with other parts of a speech recognition circuit, in addition to the distance engine, although this is not essential. For example, the accelerator may include one or more results memories for storing distances calculated by the distance calculation engine. The accelerator may also include at least one of a memory for storing one or more acoustic models, a decompressor for decompressing acoustic data that has been stored in a compressed format, a memory for storing feature vectors, a checksum or data signature calculation means, buffers for data storage, and data registers. The accelerator may be implemented in software or in hardware, or in a combination. It may be physically separate to the rest of the speech recognition circuit, although this is not essential.

The distance calculation engine may calculate one or more of a wide range of distance metrics and probability distributions. The distances may represent the likely correspondence of feature vectors to states in an acoustic model. In other words, the distances can indicate the similarity of an audio data frame to each possible state in an acoustic model

There are a wide variety of probability distributions that can be used for the distance calculation stage of a speech recogniser, and a wide variety of distance metrics used. These are widely documented in the literature. A point is a simple example of a probability distribution.

A common choice is to use Gaussian Distributions and correspondingly the Mahalanobis Distance metric. The Gaussian probability distribution is then defined by a mean vector, which defines centre point in the N-dimensional space, and a Covariance matrix which defines the shape of the probability distribution. It is common to restrict the Covariance matrix to be a diagonal matrix (only N non-zero values along the diagonal of the N×N matrix) which significantly lowers the implementation cost by reducing the number of arithmetic operations.

In particular embodiments, the distance calculated is a Mahalanobis distance. Particular examples of this are described later in the specification.

In one embodiment, the distance engine autonomously computes all of the distances associated with a given feature vector. This may comprise computing distances for every state in the lexicon. The distance engine may operate in a pipelined manner with other stages of the recognition process. In this context a distance is an indication of the probability or likelihood that a feature vector corresponds to a particular state. An important class of distance computation in speech recognition is the calculation of output state probabilities in recognisers using Hidden Markov Models. Another possible use is in recognisers using Neural Networks.

The distance engine reads data from the acoustic models to use as parameters in the calculation. The acoustic models may be optionally stored in a compressed format. The distance engine may read and de-compress the acoustic models one (or more) times for each feature vector processed. Each reading of the acoustic models may require reading the entire acoustic model, or various optimisations may be implemented to avoid reading parts of the acoustic model that are not required to be used for calculations with the current feature vector. The distance engine may use a de-compression method where the de-compression is sign or zero extension or may otherwise convert data of narrow or variable width to a wider data format. The distance engine may use a de-compression method where the de-compression is sign or zero extension or may otherwise convert data of narrow or variable width to IEEE standard single or double precision floating point format. The distance engine may use a decompression method where decompression is a codebook decompression of a binary bitstream, where the codebook is stored as part of the acoustic model data. The distance engine may use a decompression method where the decompression is decompression of a Huffman or Lempel-Ziv compressed stream. The distance engine may use a

decompression method where decompression is decompression of run length encoded data. The distance engine may use a decompression method where decompression is decompression of difference encoded data. The distance engine may use a decompression method using any well known lossy or lossless compression scheme. The distance engine may use a decompression method using subspace distribution clustering. The distance engine may use a decompression method comprising any combination of the above described decompression types.

The distance engine may read the acoustic models from a dedicated on-chip memory. The distance engine may read the acoustic models from a dedicated off-chip memory. The distance engine may read the acoustic models from a shared on-chip memory. The distance engine may read the acoustic models from a shared off-chip memory. Any of these acoustic models may be compressed.

The distance engine may compute a CRC or checksum or similar signature as it reads in the acoustic model and compares this to a stored CRC, checksum, or signature, in order to check that the acoustic model has not been corrupted, and signals an error condition if such corruption is detected. The stored CRC, Checksum, or signature may had been pre-computed and stored in the model data, or it may be computed at the time the model data is loaded into the Acoustic Model Memory. It may be held in Acoustic Model Memory, or it may be loaded into a register or another memory from where it can be accessed and compared when the CRC/checksum/signature is computed each time the Acoustic Model is loaded.

The distance engine may support the pass-through of data from the front-end to the search-stage. The data to be passed through will be supplied to the distance engine as an adjunct to the feature vector, and the distance engine will pass it to the search stage as an adjunct to the distance results. This provides a simple mechanism for passing frame-specific data that is not involved in the distance calculation through to the search stage, and keeping it associated with the correct frame, which may otherwise be complex in pipelined systems with multiple processors. The data passed through may be for any purpose. Examples might include silence detected, end-of-audio-stream detected, a frame number, information that some intervening frames have been dropped, or data from another input device such as a button or keyboard in a multi-modal interface.

The distance engine may be implemented in hardware, software, or a combination. Other stages may be implemented in hardware, software, or a combination. The distance engine may be implemented with any number representation format including fixed point or floating point arithmetic, or any mixture of number representation formats.

In particular, the other stages may be implemented on a CPU, or on a DSP and CPU. The "DSP" and "CPU" may each be implemented as software programmable devices.

The distance engine may implement one or more additional pipeline stages to overcome delays introduced by low bandwidth, high latency, or conflicted bus interfaces. The distance engine may also implement additional pipeline stages to maintain the same throughput while allowing more time for each distance calculation. Particular embodiments of the invention may include one or more of the above aspects.

A further aspect of the invention comprises a speech recognition circuit, comprising: an audio front end for calculating a feature vector from an audio signal, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame; calculating circuit for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model; and a search stage for using said calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words; a buffer memory between the calculating circuit and the search stage, for receiving data passing from the calculating circuit to the search stage, wherein a processor in the search stage has higher bandwidth and/or lower latency access to the buffer compared to the bandwidth and/or latency of direct transfer between the calculating circuit and the search stage. The data transfer from the calculating circuit to the buffer memory and/or from the buffer memory to the search stage may be performed as one or more sequential bursts. The data transfer to the buffer memory may be performed in parallel with data transfer to the calculating circuit and/or in parallel with data transfer to the search stage. A second buffer memory may be provided between the audio front end and the calculating circuit.

A further aspect of the invention comprises a speech recognition circuit, comprising: an audio front end for calculating a feature vector from an audio signal, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said audio signal during a defined audio time frame; calculating circuit for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model; and a search stage for using said calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words; comprising an elastic buffer between at least one of the front end and calculating circuit, or the calculating circuit and search stage, and/or for buffering said audio signal.

A further aspect of the invention comprises an accelerator for a speech recognition circuit, the accelerator comprising: calculating means for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model, wherein the feature vector comprises a plurality of extracted and/or derived quantities from an audio signal during a defined audio time frame; means for comparing a first version of a stored checksum of data representing said acoustic model and a second version of said stored checksum, wherein the second version is obtained from an updated measurement and calculation of the checksum; and means for indicating an error status if the checksums do not match.

A further aspect of the invention comprises an accelerator for a speech recognition circuit, the accelerator comprising: calculating means for calculating a distance indicating the similarity between a feature vector and a predetermined acoustic state of an acoustic model, wherein the feature vector comprises a plurality of extracted and/or derived quantities from an audio signal during a defined audio time frame; wherein said accelerator is configured to autonomously compute distances for every acoustic state defined by the acoustic model.

A further aspect of the invention comprises an accelerator for calculating distances for a speech recognition circuit, the accelerator comprising: calculating circuit for calculating distances indicating the similarity between a feature vector and a plurality of predetermined acoustic states of an acoustic model, wherein the feature vector comprises a plurality of extracted and/or derived quantities from an audio signal during a defined audio time frame; first and second storage circuit, which may be referred to as result memories, each for storing calculated distances for at least one said audio time frame, and for making said stored distances available

for use by another part of the speech recognition circuit; control circuit for controlling read and write access to the first and second storage circuit, said control means being configured to allow writing to one said storage means while the other said storage means is available for reading, to allow first calculated distances for one audio time frame to be written to one said storage means while second calculated distances for an earlier audio time frame are made available for reading from the other said storage means.

Embodiments of the invention may comprise means for generating a checksum or computed signature for the acoustic model data stored in the memory, and means for comparing checksums or computed signatures that have been calculated at different times, to indicate an error status if the checksums do not match, one possible cause of such mismatch being that the acoustic model data has been overwritten by said other data and said error status being used to indicate that the acoustic model should be re-loaded into the said memory.

A further aspect of the invention comprises a speech recognition circuit comprising: lexical memory containing lexical data for word recognition, said lexical data comprising a lexical tree data structure comprising a model of words; means for accessing a state model corresponding to each phone or each group of phones in the lexical tree; a content addressable memory for storing content addressable data for each phone or group of phones, including states corresponding to said phone or group of phones, and for storing an address value for each said phone or group of phones; a RAM configured to store accumulated scores for each said phone or group of phones, the accumulated scores being addressable by said address value for each said phone or group of phones; means to obtain scores that each of a plurality of frames of an audio signals corresponds to each of a plurality of said states; a counter to sequentially search for each said state in the content addressable memory, to obtain the corresponding address value if the state is found in the content addressable memory; means to use said address value to access an accumulated likelihood and an accumulator to add said likelihood to the accumulated likelihood; means to use the phones or groups of phones with the highest accumulated scores to obtain a plurality of next phones from the lexical tree which correspond to the next phone; and output means for outputting a lexical tree path of highest likelihood.

A further aspect of the invention comprises speech recognition apparatus comprising: a lexical tree having a corresponding state model; means for obtaining scores of an audio input corresponding to each of a plurality of states in said state model; a content addressable memory for storing a marker indicating a part of the lexical tree, and one or more states associated with said part of the lexical tree; a random access memory addressable by the CAM output, to output accumulated scores for states corresponding to said parts of the lexical tree; adder means for adding likelihood to said accumulated likelihood, to be stored back in the RAM.

A further aspect of the invention comprises speech recognition apparatus, comprising: a CAM-RAM arrangement for storing records including pointers to a lexical tree, and accumulative scores for states within the lexical tree; input means for obtaining scores that an audio frame corresponds to a particular state in the lexical tree; an accumulator for calculating the updated scores and modifying the records in the CAM-RAM accordingly; output means for outputting a path of highest likelihood in the lexical tree.

A further aspect of the invention comprises a speech recognition method comprising: storing state identifiers which identify states corresponding to nodes or groups of adjacent nodes in a lexical tree, and scores corresponding to said state identifiers in a memory structure, the lexical tree comprising a model of words and the memory structure being adapted to allow lookup to identify particular state identifiers, reading of the scores corresponding to the state identifiers, and writing back of the scores to the memory structure after modification of the scores; repeating the following sequence of steps for each of a plurality of incoming frames of an audio signal; obtaining score updates corresponding to the likelihoods that said frame of the audio signal corresponds to each of a plurality of said states; accessing said memory structure to obtain scores, updating the scores by adding score updates to the scores, and writing back the updated scores to the memory structure; determining if scores for states furthest on in the lexical tree correspond to a significant likelihood, and if so, then accessing the lexical tree to determine the next set of possible states; and storing the next set of possible states and said scores of significant likelihood in the memory structure.

A further aspect of the invention comprises speech recognition circuit comprising: a circuit for providing state identifiers which identify states corresponding to phones or groups of adjacent phones in a lexical tree, and for providing scores corresponding to said state identifiers, the lexical tree comprising a model of words; a memory structure for receiving and storing state identifiers and phone instance identifiers uniquely identifying instances of phones or groups of phones in the lexical tree; said memory structure being adapted to allow lookup to identify particular state identifiers, reading of the scores corresponding to the state identifiers, and writing back of the scores to the memory structure after modification of the scores; an accumulator for receiving score updates corresponding to particular state identifiers from a score update generating circuit which generates the score updates using audio input, for receiving scores from the memory structure, and for modifying said scores by adding said score updates to said scores; and a selector circuit for selecting at least one phone instance identifier according to said scores.

A speech recognition apparatus according to the invention may be embedded in or included in a mobile electronic device such as a mobile telephone, PDA (personal digital assistant), etc.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the present invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

FIGS. **1**A-**1**B are a block diagram of the system architecture for a speech recognition apparatus according to an embodiment of the invention;

FIGS. **2**A-**2**B are a block diagram showing the main data structures used in the speech recognition apparatus of FIGS. **1**A-**1**B;

FIGS. **3**A-**3**B show a block diagram of the Single Phone Instance Network Engine (PINE) of FIG. **1**B;

FIG. **4** is a block diagram showing the Wave Front Manager architecture;

FIG. **5** shows the use of link nodes to permit path merging;

FIG. **6** shows a block diagram of a Path Merge Unit in the Phone Level;

FIG. **7** shows the Phone Book architecture;

FIGS. **8**A-**8**B show Lexical tree storage;

FIG. **9** shows the data structure for the word link record;

FIG. **10** shows a block diagram of Word Engine;

FIG. **11** shows a Language Model interface architecture;

FIG. **12** shows an example of a simple Loop Grammar;

FIG. **13** shows a block diagram of grammar engine;

FIG. **14** shows a single frame of the Audio Codec link protocol; and

FIG. **15** illustrates the system timing which may be used in the embodiment of FIGS. **1**A-**1**B;

FIG. **16** is a block diagram showing a speech recognition circuit according to an embodiment of the invention, and illustrating data flow between parts of the circuit;

FIG. **17** is a block diagram showing a distance calculation engine according to an embodiment of the invention;

FIG. **18** shows ideal data flow over several time steps within a speech recognition circuit having separate front end and search stages, in an embodiment of the invention;

FIG. **19** shows an example of actual data flow over several time steps within a speech recognition circuit having separate front end and search stages, in a further embodiment of the invention;

FIG. **20** shows ideal data flow over several time steps within a speech recognition circuit having a shared processor for front end and search stages, in an embodiment of the invention;

FIG. **21** shows an example of actual data flow over several time steps within a speech recognition circuit having a shared processor for front end and search stages, in an embodiment of the invention;

FIG. **22** shows ideal data flow in a speech accelerator pipeline with two data transfer stages, according to an embodiment of the invention; and

FIG. **23** shows an example of actual data flow in a speech accelerator pipeline with two data transfer stages according to an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIGS. **1**A-**1**B show a speech recognition system architecture for an embodiment of the invention. The speech recognition system is divided into several levels of operation: a state level, a phone level, a word level, a grammar level, and a system level. Each level is shown in FIGS. **1**A-**1**B as a, interconnected to other levels of the system.

The information passed between different levels or subcircuits of FIGS. **1**A-**1**B is indicated on FIGS. **1**A-**1**B using the following identifiers.

Identifier	Definition
grammar_id	Identifies which grammar model is currently being used
grammar_node_id	Identifies a node in the grammar model
word_id	Uniquely identifies a word
wordlat_node_id	Identifies a node in the word lattice
LT_id	Lexical tree identifier. Identifies the lexical tree to be used
LT_branch_id	Identifies a unique branch in a lexical tree. A branch is defined as all of the monophones subsequent to and including the first monophone in a tree.
LT_node_id	Identifies a node in a lexical tree
monophone_id	Identifies a monophone
transvec_id	Identifies which transition vector to use
state_id	Identifies the state
phone_no	Unique phone instance identifier
prev_phone_no	Phone number of previous phone in network
triphone_id	Triphone identifier
LML_factor	Language model lookahead factor. The smeared language model score that must be added

The system level includes a general purpose processor and cache, a system controller, a DMA (direct memory access) controller, clocks, PLL (phased locked loop), GPIO (general purpose I/O interface), timer watch dog, memory interfaces to interface to main memory and non volatile storage, serial, host, debug and display interfaces. The system level also includes a model adaptation engine, a language model factor engine, a speech recognition system controller, a language model interface, and a recognition output interface.

The processor organises the transfer of the various models from the non volatile storage to the various model memories. It handles the language model factor requests. Language model factors are used to by the state level to apply the language model score. The language model (LM) interface provides the mechanism for obtaining LM factors as and when they are needed. The processor keeps a copy of the wave front in memory which is updated simultaneously with that in the wave front manager. Using its copy of the lexical tree, the processor can determine which language model scores may be needed in the coming frames. Once calculated, the language model factors can be kept in memory for use in subsequent frames. The system level is also used for forming speech model adaptation, and acting as an overflow for the various buffers used by the architecture.

The system level includes the audio codec interface and the audio front end interface, connected to the processor via a APB (advanced peripheral bus) to AHB (advanced high performance bus) bridge.

The grammar level includes a grammar engine and grammar model(s). The grammar level sets out the rules of the particular recognition task at the highest level. During system initialisation a number of grammar models might be loaded, for example a command grammar and a dictation grammar. Command mode grammar is a restricted grammar which includes only a very limited lexical tree with command specific words. Some branches of this lexical tree may include multiple words together, where a command has such a structure, e.g. "open file menu". Typically, only a single instance of the lexical tree will be needed. Command mode is used because it can obtain a higher accuracy, due to the lexical tree constraint. In dictation mode grammar, a full lexical tree structure is used.

The grammar may be selected by a user during the recognition process. For example, the command grammar may be selected when the user presses a command button. The architecture supports fast swapping of grammar models. For example, a user may be in the middle of dictating a message when a "punctuation button" is pressed and the phrase "question mark" uttered. At this point, a punctuation grammar could be loaded, in order to increase recognition accuracy.

The word level comprises a word engine, which includes a word lattice and a word graph generator. The word level has two main functions, of keeping a record of the search path at the word level using a word lattice, and interfacing to the language model via a cache. The word lattice is a record of all of the words that have been searched up to the

current frame. As new words are searched, they are appended into the lattice at the appropriate place. After the final frame in the current sentence, the n-best word lattice node identifiers from the n-best scoring tokens are passed back to the word lattice. This enables the n-best sentences to be traced back through the lattice. The word lattice can also be post processed to produce a word graph which is essentially a compact version of the word lattice.

At the word level, a record is kept of all tokens that have passed through a word end. This results in a lattice of words. When the final feature vector has been processed, the best scoring token is identified and used to trace back the search path back to the start of the search.

The phone level includes a "phone book" for generating new triphone instances, a wave front manager for sending requests to the phone book to generate new phone instances, a phone request arbiter, and a path merge unit.

The phone book includes a lexical tree, which comprises a vocabulary of words, sounds and/or phrases organised according to the phones or monophones making up the sound.

A lexical tree is commonly used in speech recognition rather than a linear lexicon, because it greatly reduces the amount of storage space required to store a word vocabulary, by avoiding duplication of common word beginnings. It is also easier to search when the word beginning is known.

Although it is possible to search a lexical tree based on monophones, it is well known that better results can be achieved in speech recognition by basing the lexical tree search on groups of more than one monophone, since the sound of one syllable commonly affects the pronunciation of the next syllable in the same word, or the first syllable in the next word. In this embodiment, groups of three monophones are used, and these are known as triphones. It is alternatively possible to use a different number of monophones for each group. The phone generator in the "phone book" uses data from the lexical tree to generate triphones.

In one embodiment of the invention, approximately 45 different cued mono-phones are considered. This gives of order 60,000 tri-phones. Each triphone in this embodiment is modelled by a Hidden Markov Model (HMM) with three states. There is no correspondence between individual states and individual phones of the triphone. Alternatively, it is possible to model each triphone by a single state, or by any other number of states.

The phone level is responsible for generating the search space according to the word structure specified in the grammar model and the phone structure specified by the lexical tree. The search space can be envisaged as a wave moving through an acousto-linguistic search space. In order that the wave can be dynamically created, a record of the wave front phone instances must be maintained. This is done in a wave front phone CAM/RAM structure, within the wave front manager. When a token occurs in the final state of a phone instance in the PINE array at the front of the wave, a new phone instance must be generated at the phone level and passed to the PINE array.

The phone instance includes a phone_no, which is a phone instance identifier to identify the phone uniquely in the current search. The phone instance also includes a word lattice node identifier (labelled "word_lat_node_id") to identify the phone's position in the word lattice, state identifiers (labelled "state_ids") for the HMM, storage to hold the tokens as they move through the HMM and LM_factor which is used to by the PINE to apply the language model score. To handle the connectivity between phone instances, a phone instance points to a previous phone instance using the phone_no of the previous phone. This allows tokens to be passed between phones. A "token" is a hypothesized search path, represented by a non-zero score at a particular state, and tokens are propagated through the HMM with each time frame.

The phone instance network can be visualised as a wave with new phone instances being added to the front of the wave, and empty phone instances being removed from the back of the wave. In some circumstances, the back of the wave may be in the same position as the wave front. The back of the wave moves on through the lexical tree as tokens are either moved on or pruned.

Tokens, each representing a particular hypothesis, hop through the states in the HMMs, each hop consuming a feature vector. Each token explores all possible next states which typically involves placing a token in the next state and keeping one in the current state. When there is more than one token in a state, the least likely token can be removed because both have the same word history. When a token reaches the last state of a phone instance at the from of the wave, the next phone instance(s) are requested from the phone level. The phone level uses the lexical tree to create the new phones. As well as looking up the state identification numbers for each HMM state, it also incorporates the correct language model factor.

The language model factor engine is used to smear the language model scores across the lexical tree. This operates in two ways depending on whether the factor is needed for a word start phone(s) or word internal phones. For word internal phones, LM factors can be computed as needed, provided that the LM scores are available in cache. This is because in general the storage requirements for the LM scores are less than the storage of the LM factors for all of the factors in the branch. However when a token is in a word end phone there may a significant number of word start phones e.g. up to $(\text{monophones})^2$ requiring a large amount of computation. To avoid this the LM engine will pre compute word start LM factors when it seems highly likely that a token will be passed to the word starts. These word start LM factors will be held in the LM factors cache. The phone level is responsible for generating the search space according to the word structure specified in the grammar model and the phone structure specified by the lexical tree.

The state level shown in FIG. **1**B includes a phone instance network engine (PINE) array, a MHD (Mahalanobis distance) engine, an audio front end, a feature vector buffer for model adaptation, and compressed acoustic models. The state level is the lowest level, in which a network of phones is dynamically created.

The audio input for speech recognition is input to the front end unit of the state level. This audio input may be digital audio, or it may be analog audio which is converted to digital audio using an analog to digital converter. The audio input is divided into time frames, each time frame typically being of the order of 10 ms.

For each audio input time frame, the audio signal is converted into a feature vector. This may be done by splitting the audio signal into spectral components. For example, in the present embodiment, the audio input signal is split into 12 frequency bands and one energy band, and a feature vector is constructed using these 13 components, plus their first and second derivatives, a total of 39 components.

The feature vector is passed to the MHD (Mahalanobis distance) engine. This computes all possible state output likelihoods for the current frame as the state mean and covariance data is read from the acoustic model memory.

The likelihood of each state is determined by the distance between the feature vector and each state. To reduce the number of states, a technique known as "state tying" may be used. Multiple states are grouped together in 39D space, according to their proximity to one another. An average or central value from each group is used to represent the entire group. Thus, the effective number of states may be reduced from 40,000 to 8,000. The accuracy may be improved using a language model in addition to the word recognition lexical tree. A language model is based on the probability of sequential words. The above figure of 8,000 states gives best recognition results when used with a language model.

The MHD (Mahalanobis Distance) is a distance between two N-dimensional points, scaled by the statistical variation in each component. In embodiments of the invention, the Mahalanobis distance between the feature vector and each state is calculated, to determine similarity of the feature vector to each state.

The Feature Vector represents a point in an N-dimensional space. Different embodiments of the invention may use different values of N. Generally we see values of N in the range of 20 to 39, although values outside this range are not excluded.

The distance calculator stage of the recognition process computes a probability or likelihood that a feature vector corresponds to a particular state. Each state has an associated probability distribution in the N-dimensional space. The distance calculator computes the distance in the N-dimensional space from the Feature Vector to the probability distribution for each state. In some embodiments, the calculation is performed only when the search stage algorithms determine that the distance for that particular state is needed by the search stage. In some embodiments, the distances for all of the states are calculated and made available to the search stage, which allows pipelined operation. Embodiments that do the calculations on demand are preferable.

The Mahalanobis Distance (MHD) is extensively described in the literature. It may be calculated using the formula

$$MHD\ \text{value}=\ln(P(x))=-0.5[N*\ln(2*\text{pi})+\ln|Cov|)]+(-0.5*[(x-u)'*\text{inverse}(Cov)*(x-u)])$$

where

N=number of dimensions in the space

x=feature vector, a point in the N dimensional space

u=mean vector

Cov=covariance matrix

|Cov|=determinant of covariance matrix

(a)'=transpose of matrix a

Different embodiments may use variations on this equation, for example the base for the logarithm is not always the natural base e that is used in the above example. The log of the probability is preferably calculated, rather than the probability, as it simplifies the implementation. Not all embodiments do this. In alternative embodiments, the likelihoods rather than probabilities are computed as it leads to simpler implementations of computations in the search stage.

The feature vector is used to calculate 8,000 MHD distances for each time frame, i.e. one distance for each of the 8,000 states. It is beneficial, although not essential, to perform the MHD calculations separately in a pre-processor, known as a MHD engine, because MHD calculation is a computationally intensive process.

Due to the 10 ms frame length, a feature vector arrives at the MHD engine of the state level every 10 ms. The MHD engine calculates the likelihood of each feature vector corresponding to each of a number of states derived from an acoustic model. Each state is also a 39 dimensional vector, having the same spectral components as the feature vector. The MHD engine calculates the difference in 39-D space between the feature vector and each state to obtain the likelihood for that state. A parallel MHD or a single MHD arrangement may be used.

The feature vectors may be sent to a feature vector buffer for model adaptation, which determines whether the observed features fit well with the acoustic model, and may initiate an adaptation of the model if the fit is not sufficiently good.

Scores indicating the state likelihoods or probabilities are output from the MHD engine to the PINE array. It is preferable to use likelihoods rather than probabilities, because likelihoods can be added together to give a total likelihood. Probabilities must be multiplied, which can give rise to rounding errors. In this embodiment, eight separate pines are used, each of 4 kilobytes, thus storing 4000 phones per pine.

During recognition, the Phone Instance Network Engine (PINE) Array will contain a network of phone instances. Tokens representing search hypotheses in the form of a probability (likelihood) will exist in some of the HMM states of the phone instances. The tokens are propagated into all available states and updated with the corresponding state output probabilities. Maximum model, word end and beam pruning are applied to the tokens to ensure the search space is kept bounded. When a token is in the final state of a phone model and has avoided being pruned, the PINE makes a next phone request to the Phone level. The PINE array may receive a number of new phone models from the phone level which must be appended to the appropriate existing phone.

FIG. 2B shows the data structures stored in the PINE array, and other main data structures used during the acoustic search. Each PINE includes a phone instance CAM and a phone instance RAM. The phone instance CAM stores the phone instance number (labelled "phone_no") which uniquely identifies each phone instance. The phone instance CAM also stores the state IDs of the three HMM states used to model the phone, and flags indicating the position of the phone. The "back" flag indicates if the phone is at the back of the wavefront. The "front" flag indicates if the phone is at the front of the wavefront. The "empty" flag indicates if there are no tokens in any of the three states for that phone. The "end" flag indicates if the phone is at the end of a word.

The phone instance RAM stores the token scores for each of the three states corresponding to each phone. The st_dum field indicates a dummy state, which is used to represent a state prior to the first state of the first phone. The "tmp" field is used for passing a token from a previous phone in the lexical tree to the current phone. This will be explained in more detail later. The prev_phone_no field is for storing a pointer to the previous phone instance in the network. The wordlat_node_id identifies the position of the phone instance on the word lattice node, and the LM_factor stores the language model factor for that phone instance.

At the phone level a record of all phones on the wave front must be maintained. If a phone instance is on the wave front, the phone_no stored in the phone instance CAM will point to the details in the wave front phone CAM and RAM.

The wave front phone CAM and RAM keeps pointers into the grammar and lexical tree. When a token in a wave front phone reaches the last state, the PINE array makes a request to the phone level by passing back the phone_id. The phone_id is used as a key to determining the token's grammer_node_id and LT_node_id within the wave front

CAM/RAM structure. If the token is at a word end the grammar model is checked. A grammar identifier points to the type of grammar.

At the word level, FIGS. **2**A-**2**B show an example word lattice structure, with silence having a word_id of 0, and a wordlat_node_id of 6. This is shown as connected to two word nodes—"which" has a word_id of 238 and a wordlat_node_id of 4, and "neither" has a word_id of 3556 and a wordlat_node_id of 5.

At the highest level, i.e. the grammar level, a simple grammar model is shown which will allow the phrase 'silence which house silence' or 'silence neither house silence' to be spoken. Whenever a word end is reached during the search, the grammar model must be consulted to determine the next possible word(s) or lexical tree(s). This allows the word lattice at the word level, consisting of a network of words, to be assembled. Whenever a word end is reached, the word is added to the lattice and the wordlat_node_id updated accordingly.

FIGS. **3**A-**3**B show a single phone instance network engine (PINE). At system power up the state transition RAM will be loaded with the appropriate state transition vector but the CAM and RAM memories will be empty. When the system is initialised an initial set of phone model(s) will be instantiated with a token in the dummy state and a path score of zero. The phone models are received from the phone level.

When the audio data is processed, a stream of all possible state output probabilities will be received from the MHD engine. A counter that is synchronous to the stream of state output probabilities from the MHD produces a concurrent stream of state_ids. The job of the CAM is to perform a search to see if the state_id is instantiated anywhere in the CAM state_id fields. If it exists and the location doesn't have its empty flag set then this indicates that a token may need to be updated. The match signal from the CAM must be encoded into an address for the RAM. The corresponding RAM address can then be read. Any token that will end up in the identified state can then have it's path score updated and including any transition vector likelihood.

It is possible that a single state_id generates multiple CAM matches. To handle this situation the CAM and RAM are effectively decoupled by means of a match fifo and repeat count fifo. The state_id and corresponding state_output_probability are stored in the repeat count fifo. During a subsequent frame when there is no match, this state_id can be revisited, ignoring the matches already dealt with.

All searched phone instances corresponding to words in the current possible lexical tree are stored in the CAM-RAM. This includes phone instances at the beginning of the word, middle of the word and end of the word. These phone instances are connected by the "previous phone instance" field, which allows traceback through an entire word within the CAM-RAM.

A mechanism is thus required to pass tokens from one phone instance to the next. This mechanism is provided by the tmp field in the RAM.

The tmp field in the RAM holds tokens during token passing.

At the start of each time frame, the audio input is converted into a feature vector, the MHD values calculated, and the MHD values used to update the token scores. After update of all tokens, 2 passes are made through the CAM-RAM, firstly for pruning, and then for token passing. The pruning in this example removes tokens of below a threshold score, although other types of pruning are also possible. The pruning may involve removing a predetermined proportion of the tokens, chosen as those of lowest likelihood. A third possible method of pruning is removing all tokens with scores below a first predetermined value, but at word ends removing all tokens with scores below a second predetermined value, the second predetermined value being greater than the first predetermined value.

The token passing works as follows. Each phone instance points back to a previous phone, due to the prev_phone_no field stored in the RAM. During the token passing, the third token of a previous phone is grabbed and put into the tmp field of the current phone.

In other words, if a token is on the third state, and it is not pruned, it must enter the next state of the next phone. To do this, the tokens that lie in the third state are moved across to the tmp field.

For example, phone no. 345 needs to look at phone no. 320 to see if any tokens are in the third state. If yes, it takes these tokens and puts them in the tmp field.

One field in the CAM is labelled "front". This indicates whether the phone is on the wave front. If the phone is not on the wave front, then another phone will have picked up the token in the third state. However, if the phone is on the wave front, then no other phone will have picked up the token on the third state. Therefore, if there is a token in the third state, and the phone is on the wavefront, it is necessary to generate a next phone request.

The current phone number, word lattice identifier and word end flag are sent to the wave front manager, to request a next phone. If the word end flag indicates a word end, then the score is also passed to the wave front manager. The score can be passed in any case, but it is only required at a word end.

The phone level generates the next phone or phones. A new phone instance starts, and all of state1, state2 and state3 have scores pre-set to zero. The old score is provided in the tmp field.

The old phone is still within the wave, but it is unlikely to be at the back of the wave at this stage. When it gets to the back of the wave, it is deleted. Until it gets there, it remains in the CAM-RAM as before. The third token remains where it was, since tokens can either stay in the third state or move on.

Note that it is possible to end up with more that one token in each state. If this occurs, then the token with the poorer score is deleted. This does not affect the result of the lexical tree search, because both tokens on the same state have exactly the same word history.

FIG. **4** shows a block diagram of the wave front manager. The wave front manager is responsible for maintaining a record of the set of phone models that exist at the front of the acoustic search space. This set of phone models is referred to as the "wave front." The key requirement for this block as to minimise the time between receiving a "next phone request" and generating the appropriate responses. The wave front manager receives the next phone requests from the PINE array and generates appropriate responses for the Phone Book, Word Engine, Path Merge Unit and Grammar engine The wave front manager also receives new wave front phone models from the Phone Book and updates the wave front.

The wave front manager receives the next phone request from the PINE array, via the phone request arbiter. The next phone request consists of a phone_no, word_end_flag and wordlat_node_id. If this phone is at the end of a word, the word_end_flag will be set.

The wave front manager uses the phone_no as input to the wave front phone CAM. As each phone_no is unique, it is

guaranteed that only a single match will be generated. The wave front manager uses the CAM generated address to access the RAM. It then sends retrieved data from the RAM to the phone book, grammar engine, Path Merge Unit and word engine.

The wave front manager removes phone instance from CAM and RAM by writing 0's into CAM and RAM, and increments the empty count by one.

Sometime later, the wave front manager receives new phone instances from the phone book and place them in a fifo. The phone instances are taken from the fifo and written into the CAM and RAM at the next_empty_address. The empty_count is decremented by one. A match is run on the "occupy" flag, and the address is placed into next_empty_address register.

Both the phone level and the state level make use of a path merge unit to reduce the number of possible paths, by path merging. FIG. **5** shows the use of link nodes to permit path merging. In the present embodiment, all word end phone instances are linked to word start triphones with a link node. During each frame the path merge unit looks for word end tokens that can be recombined. This is done by looking for word end phones that have the same word_id and word history. When they are found a link is added using the path merge unit in the PINE array.

FIG. **6** shows a diagram of the architecture at the Phone level, showing the path merge unit. The path merge unit includes a path merge CAM and a path merge RAM. The path merge CAM stores word_id, word history for two words, and an "occupy" field. The path merge RAM stores a link_node_id. A CAM RAM occupancy manager keeps track of the empty addresses in the CAM and RAM. The path merge unit is only passed information from the wave front manager if the phone is a word end phone. For each frame, there is a check to see if there is an identical word with the same history, and if there is, the link_node_id is passed to the path merge unit. If there isn't an identical word with the same history, a new record is written into the Path merge CAM-RAM.

FIG. **7** is a block diagram showing the Phone Book architecture. The Phone Book is tasked with generating the next phone models when given a pointer into the lexical tree. The key requirement for this block as to minimise the time between receiving a "lexical tree address" and generating the appropriate responses, in particular the next phone model instance(s).

The lexical tree is used to build the next triphones based on the current context.

Current triphone	Next triphone	comments
Left context = a Central context = b Right context = c	discarded Left context = b Central context = c Right context = ?	next_leftmph_id stored in wave front manager current LT_node_id stored in wave front manager must be looked up. Large (<45) for new word start

The LT_node_id, next_left_mph_id and prev_phone_no are received at the phone generator from the wavefront manager. The LT_node_id is then used to address the lexical tree store and return the central monophone_id. One or more further accesses to memory are performed using the micro-coded instructions, each memory access returning the right monophone_id.

A two stage lookup is accessed using the three monophone_ids (giving a triphone), the associated state_ids. The phone model is assembled, and returned to the PINE array. The wavefront manager is then updated by sending it relevant parts of the phone model.

The lexical tree store is a micro-coded ROM with both instruction and data fields. The lexical tree store holds a number of lexical trees. Each lexical tree may hold single words or multiple words. Each lexical tree comprises lexical tree elements, and each element has three fields. For now, the fields will be integers and will be converted to std_logic_vector later. The following is an example of the lexical tree store, with the LT data in word_id order.

ZERO	z iy r ow	19 8 12 11 0 0 0 0	sp 0
ONE	w ah n	18 2 10 0 0 0 0 0	sil 1
TWO	t uw	14 16 0 0 0 0 0 0	ah 2
THREE	th r iy	15 12 8 0 0 0 0 0	ay 3
FOUR	f ow r	6 11 12 0 0 0 0 0	eh 4
FIVE	f ay v	6 3 17 0 0 0 0 0	ey 5
SIX	s ih k s	13 7 9 13 0 0 0 0	f 6
SEVEN	s eh v ih n	13 4 17 7 10 0 0 0	ih 7
EIGHT	ey t	5 14 0 0 0 0 0 0	iy 8
NINE	n ay n	10 3 10 0 0 0 0 0	k 9
OH	ow	11 0 0 0 0 0 0 0	n 10
			ow 11
The following shows the LT data in MP (monophone) start order.			r 12
			s 13
			t 14
EIGHT	ey t	5 14 0 0 0 0 0 0	th 15
FIVE	f ay v	6 3 17 0 0 0 0 0	uw 16
FOUR	f ow r	6 11 12 0 0 0 0 0	v 17
NINE	n ay n	10 3 10 0 0 0 0 0	w 18
OH	ow	11 0 0 0 0 0 0 0	z 19
SEVEN	s eh v ih n	13 4 17 7 10 0 0 0	
SIX	s ih k s	13 7 9 13 0 0 0 0	
TWO	t uw	14 16 0 0 0 0 0 0	
THREE	th r iy	15 12 8 0 0 0 0 0	
ONE	w ah n	18 2 10 0 0 0 0 0	
ZERO	z iy r ow	19 8 12 11 0 0 0 0	

The following is an example of the three fields of a lexical tree element. The first field is an instruction field, and the second and third fields are data fields.

Fields:	1st instr 2b	2nd data 6b	3rd data 6b	
	00	MP	MP	- raw data field.
	01	br	br	- branch address.
	10	br	br	- branch address.
	11	word_id		- double field.

Note that there might be no difference between "01" and "10"

To build the "ey" tree, we only have to consider "EIGHT", because this is the only word in the lexical tree which starts with "ey". EIGHT is represented as "ey t", with phone_nos 5 14 0 0 0 0 0 0. The first lexical tree element stores the raw data specifying the phone_nos, the second lexical tree element is an end field, and the subsequent lexical tree element is set to zero.

0 00 5 14	- raw data
1 11 8 0	- end field
2 00 0 0	

To build the "f" tree, we have:

FIVE	f ay v	6 3 17 0 0 0 0 0
FOUR	f ow r	6 11 12 0 0 0 0 0

-continued

```
0 01 4 0     -- branch forward four places as well
1 00 6 3     -- as the implied other brach of next (one).
2 00 3 17
3 11 5 0     -- all addresses are relative offsets.
4 00 6 11
5 00 11 12
6 11 4 0
7 00 0 0
```

To build the "n" tree, we have:

```
NINE          n ay n          10 3 10 0 0 0 0 0
0 00 10 3
1 00 3 10
2 11 9 0
3 00 0 0
```

To build the "ow" tree, we have "ow" in both the first and second data fields, as no tree is going to be a single monophone long.

```
OH            ow              11 0 0 0 0 0 0 0
0 11 11 0
1 00 0 0
```

To build the "s" tree, we have:

```
SEVEN         s eh v ih n     13 4 17 7 10 0 0 0
SIX           s ih k s        13 7 9 13 0 0 0 0
0 01 6 0 -- branch forward six places.
1 00 13 4
2 00 4 17
3 00 17 7
4 00 7 10
5 11 7 0
6 00 13 7
7 00 7 9
8 00 9 13
9 11 6 0
10 00 0 0
```

To build the "t" tree, we have:

```
TWO           t uw            14 16 0 0 0 0 0 0
0 00 14 16
1 11 2 0
2 00 0 0
```

To build the "th" tree, we have:

```
THREE         th r iy         15 12 8 0 0 0 0 0
0 00 15 12
1 00 12 8
2 11 3 0
3 00 0 0
```

To build the "w" tree, we have:

```
ONE           w ah n          18 2 10 0 0 0 0 0
0 00 18 12
1 00 12 10
2 11 1 0
3 00 0 0
```

To build the "z" tree, we have:

```
ZERO          z iy r ow       19 8 12 11 0 0 0 0
0 00 19 8
1 00 8 12
2 00 12 11
3 11 0 0
4 00 0 0
```

Once all of the branches have been built, the next stage is to concatenate the tree. It appears that four elements make a page, and each element is two bytes.

		page
ey:	00 5 14 11 8 0 00 0 0 00 0 0	5
f:	01 4 0 00 6 3 00 3 17 11 5 0	6
	00 6 11 00 11 12 11 4 0 00 0 0	7
n:	00 10 3 00 3 10 11 9 0 00 0 0	8
ow:	11 11 0 00 0 0 00 0 0 00 0 0	9
s:	01 6 0 00 13 4 00 4 17 00 17 7	10
	00 7 10 11 7 0 00 13 7 00 7 9	11
	00 9 13 11 6 0 00 0 0 00 0 0	12
t:	00 14 16 11 2 0 00 0 0 00 0 0	13
th:	00 15 12 00 12 8 11 3 0 00 0 0	14
w:	00 18 12 00 12 10 11 1 0 00 0 0	15
z:	00 19 8 00 8 12 00 12 11 11 0 0	16

Then, the start lookup header is defined. Each row (page) is 8 bytes and hence the header takes 3 rows. The end_ lookup header is two rows. Therefore, the lexical tree store proper starts at the 6th (#5) page.

sp	sil	ah	ay	eh	ey	f	ih	MP
0	1	2	3	4	5	6	7	MP_id
3	0	0	0	5	0	6	0	data stored.
iy	k	n	ow	r	s	t	th	
8	9	10	11	12	13	14	15	
0	0	8	9	0	10	13	14	
uw	v	w	z	.	.	.	.	
16	17	18	19	.	.	.	.	
0	0	15	16	0	0	0	0	

The end_lookup header is defined. **20** is out of bounds.

ey	f	n	ow	s	t	th	w
5	6	10	11	13	14	15	18
z	.	.	.	.	.	.	.
19	20	20	20	20	20	20	20

To put the above information all together:

START	sp: ih	3 0	0 0	5 0	6 0	0
LOOKUP	iy: th	0 0	8 9	0 10	13 14	1
	uw: v	0 0	15 16	0 0	0 0	2
END	ey: w	5 6	10 11	13 14	15 18	3
LOOKUP	z	19 20	20 20	20 20	20 20	4
LEXICAL	ey:	00 5 14	11 8 0	00 0 0	00 0 0	5
TREE	f:	01 4 0	00 6 3	00 3 17	11 5 0	6
STORE		00 6 11	00 11 12	11 4 0	00 0 0	7
	n:	00 10 3	00 3 10	11 9 0	00 0 0	8
	ow:	11 11 0	00 0 0	00 0 0	00 0 0	9
	s:	01 6 0	00 13 4	00 4 17	00 17 7	10
		00 7 10	11 7 0	00 13 7	00 7 9	11
		00 9 13	11 6 0	00 0 0	00 0 0	12

-continued

t:	00 14 16	11 2 0	00 0 0	00 0 0 13
th:	00 15 12	00 12 8	11 3 0	00 0 0 14
w:	00 18 12	00 12 10	11 1 0	00 0 0 15
z:	00 19 8	00 8 12	00 12 11	11 0 0 16

3 0	0 0	5 0	6 0
0 0	8 9	0 10	13 14
0 0	15 16	0 0	0 0
5 6	10 11	13 14	15 18
19 20	20 20	20 20	20 20
00 5 14	11 8 0	00 0 0	00 0 0
01 4 0	00 6 3	00 3 17	11 5 0
00 6 11	00 11 12	11 4 0	00 0 0
00 10 3	00 3 10	11 9 0	00 0 0
11 11 0	00 0 0	00 0 0	00 0 0
01 6 0	00 13 4	00 4 17	00 17 7
00 7 10	11 7 0	00 13 7	00 7 9
00 9 13	11 6 0	00 0 0	00 0 0
00 14 16	11 2 0	00 0 0	00 0 0
00 15 12	00 12 8	11 3 0	00 0 0
00 18 12	00 12 10	11 1 0	00 0 0
00 19 8	00 8 12	00 12 11	11 0 0

This information is converted to hexidecimal. The first five are byte fields.

The rest are:

2bit - uCode. 6bit - data. 2bit - null. 6bit - data			
03 00	00 00	05 00	06 00
00 00	08 09	00 0A	0D 0E
00 00	0F 10	00 00	00 00
05 06	0A 0B	0D 0E	0F 12
13 14	14 14	14 14	14 14
05 0E	C8 00	00 00	00 00
84 00	06 03	03 11	C5 00
06 0B	0B 0C	C4 00	00 00
0A 03	03 0A	C9 09	00 00
CB 00	00 00	00 00	00 00
86 00	0D 04	04 11	11 07
07 0A	C7 00	0D 07	07 09
00 0D	C6 00	00 00	00 00
0E 10	C2 00	00 00	00 00
0F 0C	0C 08	C3 00	00 00
12 0C	0C 0A	C1 00	00 00
13 08	08 0C	0C 0B	C0 00

The above is the micro code for the digit lexical tree. It allows branching, and it holds the word_id. It knows which lexical trees it contains and where they are housed. It also stores the possible end monophone list.

FIGS. **8**A-**8**B show a further example of the Lexical Tree RAM, giving examples of different types of branching in the lexical tree, and how these are represented in the RAM.

FIG. **9** is a block diagram showing the data structure that is supported for the word link record, to enable word n-best decoding. The word engine maintains a record of the words that have been currently searched by the PINE array. It is used to determine the word history and allow trace back during or at the end of search. The word history is needed to determine the language model score and whether particular paths can be recombined or merged.

FIG. **10** shows a block diagram of the word engine. This is designed to allow fast access to the word history (thus allowing rapid phone creation at the phone level), and to allow reasonably fast trace back. The Word Engine is primarily a memory store that fills incrementally. The decoupling of the wordlat_node_id and the physical address will allow the possibility of being able to keep the WLRs for a given word in contiguous addresses, thus avoiding the need for the pointer field.

The language model interface is designed to allow the Phone Book to quickly obtain the appropriate LM factor for the phone model that it is currently constructing. Based on the phones in the wave front, all possible next LM factors can be determined. This will be done in software together with the language model factor engine. The ARM **9** will maintain its own record of the wavefront using the same next phone requests received by the wave front manager and its own copy of the lexical tree.

FIG. **11** shows the architecture of the LM interface. This allows the Phone book to quickly access the LM factor for the phone that is currently being generated. Prior to the phone generation phase LM_factors are computed for all possible phones that may be generated in the next frame.

The grammar engine is designed to give the Phone Book a quick response to requests for the next grammar node(s). In the case of a task grammar where the lexicon may be limited to a few words (e.g. yes, no), it may be desirable to use the same lexical tree. In this case it will be necessary for the grammar, to provide lexical tree pointers such that the phone generator can jump to the appropriate part of the lexical tree.

For a speech recognition system it is also necessary to specify the grammar. Again we will use HTK's Standard Lattice Format (SLF) to specify the grammar. In practice it will be convenient to use HTK's HParse format to specify the grammar using a high level grammar notation. This can then be converted into using the HParse tool. Consider having to specify a grammar for a digit recognition task. The HParse format for this is as follows:

```
$digit = one | two | three | four | five | six | seven | eight | nine | zero
$digit_sp = $digit sp
$noise = lipsmack | breath | sil
( [sil] < $digit_sp | $noise > [sil] )
```

where | denotes alternatives, < > denotes one or more repetitions, { } denotes zero or more repetitions, [ ] denotes that item is optional, and ( ) contains the grammar.

FIG. **12** shows the network that is defined by this grammar. The network starts and ends with a "sil" phone, representing silence. Between the start and end, a number of possible routes correspond to further silence, a breath, a lipsmack sound, or a detected word.

FIG. **13** shows the grammar engine architecture. When a next phone request arrives, the next grammar node(s) must be retrieved from memory. In the case of a task grammar, the phone book should be given the lexical tree memory addresses to allow the phone to be determined.

FIG. **14** shows a single frame of the Audio Codec (AC) link protocol used in some embodiments of the invention. The Advanced Audio CODEC Interface (AACI) is an ARM Primecell that provides an interface to off-chip AC97 audio codecs. AC97 is Intel's audio codec specification, initially created for the PC market. The AACI is an AMBA slave block that connects to the Advanced Peripheral bus (APB). The AACI connects to the off-chip codec using the AC-link protocol.

The AC-link protocol is a bi-directional, fixed clock rate, serial digital stream. AC-link handles multiple input and output PCM audio streams as well as register control accesses. It employs a time division multiplexed (TDM) scheme that divides each audio frame into 12 outgoing and

12 incoming data streams each with up to 20 bit sample resolution. The tag slots are used to indicate which of the subsequent slots contain valid data. Add and dat are the control register address and data slots for read and writes. The adc slots contain the outputs from the analog to digital converters. Assuming that the ADC sample rate is set to 16 kHz then the AC-link controller will receive 16 bits of data every third codec frame.

The AACI has 4 channels that can be operated simultaneously. They are accessed through DMA.

FIG. **15** shows the system timing. Each frame is divided into three sections:

Token update (MHD wash)

First pass—inter phone token passing

Second pass—garbage collection, max model pruning

The horizontal direction of the figure indicates elapsing time, and various subcircuits and processes are indicated along the vertical direction of the figure. The top part of the diagram shows frame count, and push to dictate button, and the lower part of the diagram shows various subcircuits and processes, within a single time frame, whilst the push to dictate button is engaged. These subcircuits and processes are language model interface, word engine, phone book, PINE array, MHD engine, mean & covar reg, and feature vector reg.

At the start of the time frame, the feature vector is generated, and the MHD values are calculated. The MHD values are then passed to the PINE array, which engages in beam pruning and token score updating. The phone book receives next phone requests and returns next phone responses. The word engine adds word link records, and the language model interface supplies language model factors.

In the second part of the frame, pruning thresholds are calculated for the PINE array, for the next frame, and the PINE array engages in token passing. The language model interface transfers language model factors for the next frame.

While the "push to dictate" button remains activated, the above process repeats in a sequence of frames.

The embodiments of the invention which are now described relate to a distance engine for calculation of Mahalanobis Distances.

FIG. **16** shows a block diagram of a speech recognition system according to an embodiment of the invention. The system includes a microphone **100**, which outputs an audio signal to a combined AGC (automatic gain control) and ADC (analogue to digital converter) **101**. In alternative embodiments, the AGC and ADC may be separate. In this embodiment, the incoming data comes from a microphone, and is then converted to digital format. However, it is also possible that the digital audio data may be received from a network, or may be read from storage means such as a memory, an optical disk, or a hard disk drive.

The digital audio data output from the ADC is input to an audio data buffer **102**, and may be held here until required. The data is then output from the audio data buffer **102** to a front end circuit **103**. The incoming audio data is divided into frames, as described above. In cases where the system is extremely busy, it is possible to drop some audio frames, to allow the system to catch up with any backlog.

In the audio front end circuit **103**, the audio is split into spectral components to generate a feature vector, as described above. For example, the feature vector may have a number of frequency bands, and an energy band, plus their first and second derivatives. The feature vector is then output from the front end **103** to the distance calculation engine **104**.

The distance calculation engine **104** converts feature vectors into Mahalanobis distances. The acoustic model may be chosen from a plurality of acoustic models, e.g. according to the language used, and may be stored in a storage means **105** such as a memory, a disk drive, etc. The distances calculated by the distance calculation engine are then transferred to the search stage **106** of the speech processing circuit, which uses models such as one or more word models and/or language models to generate and output recognised text. Such word and language models may be stored in a separate storage means **107**.

In one embodiment, the distance calculation engine uses Mahalanobis Distance (MHD) calculations to determine distance values for output to the search stage **106**. In further embodiments, different types of distance calculations may be used, e.g as described above.

In some embodiments of the invention, the distance calculation engine is designed as a speech accelerator, to operate as a loosely bound co-processor for a CPU running speech recognition software. This has the advantage of reducing computational load on the CPU, and reducing memory bandwidth load for the CPU.

In some embodiments, the accelerator interface is memory-mapped. Other types of interface are also possible to support different system configurations, for example removable accelerator cards with SDIO (secure digital input/output), CardBus, or USB interfaces.

The speech recognition apparatus may also operate with a CPU and DSP (digital signal processor), where the DSP implements the "front-end" signal processing to produce a feature vector, and the CPU implements the search stage, reading the distance results from the distance calculation engine.

The system may be implemented in any combination of hardware or software.

In some embodiments of the invention the search stage **106** provides the distance calculation engine **104** with an active state list **108** that indicates the subset of states for which the search stage requires distances to be calculated by the distance calculation engine **104**. This is an optimization that may reduce computation time and/or power consumption.

In preferred embodiments of the invention, the front end **103**, distance calculation engine **104** and the search stage **106** operate in a pipelined manner. When operating in a pipelined manner it is unlikely that the search stage **106** will be able to provide the active state list **108** early enough for the distance calculation engine **104** to implement the optimization of computing only the distances that will be required by the search stage **106**. The distance calculation circuit **104** may calculate the MHD values for every state in the lexicon, per frame, whether it is subsequently required by the search stage or not. This allows the accelerator and software system to operate in a concurrent pipelined manner, which maximizes the system throughput.

The distance calculation engine **104** may have a plurality of memory areas for storing feature vectors, to allow a feature vector for one frame to be transferred to one memory area in the distance calculation engine **104**, while another feature vector for another frame is being read from a second memory area, for calculation of a distance. The memory areas used for reading and writing may alternate with subsequent frames.

FIG. **17** shows a block diagram of a distance calculation accelerator **200**, used in an embodiment of the invention. A feature vector is received at input **212** of the accelerator **200**. The feature vector is loaded into feature vector (FV)

memory A **208** via a bus **211**. In reality, there may be more than a single bus, but for simplicity, only one bus **211** is shown in the figure.

The registers **203** are discussed further at the end of this description. The registers **203** may include, for example, an accelerator status register, for indicating whether the acoustic model data is correctly stored in memory, as described in more detail below.

The FV registers **209** hold the feature vector whose distances are currently bring computed by the distance calculation engine **204**. If the distance calculation engine **204** is implemented in software, the FV registers might be implemented as memory attached to the processor running the software. If the distance calculation engine **204** is implemented in hardware the FV registers might be implemented as logic or as a memory block. The FV registers **209** are loaded from the FV memory **208** when the Distance Calculation Engine **204** has completed distance calculations for the current feature vector and a subsequent feature vector is available in the FV memory **208**. In some embodiments there is an optional FIFO associated with FV registers **209** so that the FIFO is loaded from the FV memory **208** when there is a subsequent feature vector in the FV memory **208** and there is a space in the FIFO. In these embodiments the FV Registers **209** are loaded from the associated optional FIFO **209** when the Distance Calculation Engine **204** has completed calculations for the current feature vector and there is a subsequent feature vector in the associated optional FIFO **209**. The presence of the FIFO **209** provides for a looser coupling between the Front End **103** and the accelerator **200** which improves performance in some system configurations.

Each feature vector is transferred to a distance calculation engine circuit **204**, to obtain distances for each state of the acoustic model. The acoustic model is stored in an acoustic model memory **205**, which is connected to the bus **211**, for control of reading/writing to the memory. In some embodiments, this also allows acoustic models to be changed or updated via the bus. The acoustic model may be stored in compressed form in the acoustic model memory **205**. A decompressor **206** is provided on the data output of the acoustic model memory. The decompressor **206** can decompress the results as they are needed, and thus allows a much smaller size of acoustic memory to be used, than with uncompressed data. From the decompressor **206**, the decompressed acoustic model states are sent to the distance calculation engine **204**.

The distance calculation engine **204** converts feature vectors and acoustic model states to distances. The accelerator **200** includes two separate results memories for storing these calculated distances, labelled as "result memory A" **201** and "result memory B" **202**.

The control, sequencing, and address generation **210** is logic, state machines, and/or software programs that provide overall control of the other elements of the accelerator, taking inputs from the registers **203**, and all the other elements and interfaces to determine when to start and stop the Distance Calculation Engine **204**. The control **210** also produces addresses and control signals to control the reading and writing of the memories **201**, **202**, **205** and **208**, to control the flow of data between the various elements of the accelerator **200**, the operation of the Decompressor **206**, CRC check **207** and Distance Calculation Engine **204**, and to control the setting or clearing or status bits in the Registers **203** and to control signalling to the front end **103** and search stage **106**. The control **210** is implemented using any of the widely known and used methods of controlling and sequencing in digital systems including logic gates, state machines, and embedded software systems. The control **210** might be a centralized control, or it might be distributed in some or all of the other elements of the accelerator **200**, or some combination of these.

Each time a feature vector is loaded into the accelerator, the accelerator computes the distances for all states for that feature vector, and stores the results alternately in the A or B Results Memory. The results are written to one of the two memories **201**, **202**, while the other memory is available for reading. Thus, while the distance calculation engine **204** is storing the distances calculated for one audio frame, the other result memory may make available the distances calculated for an earlier audio frame, to the search stage of the speech recognition system. For non-real time processing, "earlier" audio frame may refer to an audio frame that has been processed earlier, rather than being limited to an audio frame that corresponds to speech at an earlier time.

Thus, the search stage software reads results alternatively from the A or B result memory, while the accelerator writes results into the other result memory, allowing concurrent pipelined operation of the accelerator and the CPU [and the DSP if present].

In some implementations the A and B results memories may be exposed to the software interface as separate memories appearing at different locations in the CPU and/or DSP memory map. Another implementation is to map each memory alternatively to the same range of addresses in the CPU and/or DSP memory map, so that the software always accesses results at the same location, and the memory containing the set of results the software currently needs to access is the one mapped there.

The accelerator may autonomously load and decompress an acoustic model, which is stored in a compressed format in the acoustic model memory **205**. The Acoustic Model may be loaded into the Acoustic Model Memory by software running on the CPU prior to the first use of the Accelerator. The Acoustic Model Memory may be an on-chip memory, to maximize bandwidth and minimize power consumption of reading in the entire Acoustic Model every frame, which means 50 to 100 times per second, depending on the system design.

The Acoustic Model Memory may be a dedicated memory attached to, or incorporated within, the accelerator. Alternatively the Acoustic Model Memory may be a shared resource that is used by other system functions or applications when speech recognition is not running. To allow detection of situations where the acoustic model has been overwritten by another application, a pre-calculated CRC (cyclic redundancy check) signature may be stored with the acoustic model. Each time the acoustic model is read into the accelerator, a CRC may be computed and compared with the signature. If the comparison fails, an error bit is set in the accelerator status register.

Some embodiments of the invention relate to speech recognition systems comprising separate front end **103** and search stage **106** processors. These separate processors perform the front-end and search stages of the recognition process, with the accelerator **200** performing the distance calculations. Typically the front-end processing would be performed on a DSP and search stage would be performed on a CPU, although this description holds for any type of processors, even dedicated hardware implementations. As shown in FIG. **17**, the front end **103** supplies the distance accelerator with feature vectors, and the search stage **106** uses distances calculated by the accelerator.

The accelerator is designed to operate in a concurrent pipelined manner with the other processors, allowing a high level of parallelism in the system.

FIG. **18** shows how this pipelined operation works in an embodiment of the invention with ideal data flow, i.e. where, each of the three recognition stages require exactly the same time to process each data frame, and the processors are never delayed, stalled, or diverted to other tasks.

The time axis is divided into equal "pipeline step times". If the step time is the same as a frame time then the system is processing audio in real time. If the step time is short than a frame time, the system is operating faster than real time, which may be valuable for batch processing of audio data, or catching up after some type of delay. If the step time is loner than a frame time, the system is operating slower than real time, which may be acceptable in some applications, for example where the person dictating pauses occasionally allowing the system to "catch up".

Once the pipeline has filled (which takes two step times), the DSP, the Accelerator, and the CPU are all operating in parallel. In this idealized example, the total time available for processing a frame is three step times—one step time in the DSP, one step time in the accelerator, and one step time in the search stage. There is a latency of three step times from the time the audio data enters the front end to the completion of the search stage for that frame, and the processing of one frame finishes each step time. If the system were not pipelined, then the processing of all stages for one frame would have to complete in one step time in order to achieve the same overall throughput of frames per second. Thus pipelining in this way can achieve three times the throughput with the same processing elements, compared to a non-pipelined system. A non-pipelined system would be one in which the processing of frame n+1 would not start until all stages of the processing of frame n was already complete.

If, for example, in a system running in real time with a frame time and step time of $\frac{1}{100}^{th}$ of a second, which is 10 ms, each of the three processing elements has the entire 10 ms available to complete the processing of one stage of the recognition task, giving 30 ms of processing time overall for each frame. If the system were not pipelined, real time operation would require that all three processing elements would be complete in 10 ms, allowing only 3.3 ms for each of the three processing elements.

At the end of each pipeline step time, the DSP provides a feature vector to the Accelerator, and the accelerator provides a set of distance results to the CPU. When the CPU is performing the search stage for frame n, the Accelerator is computing the distances for frame n+1, and the DSP is performing the front-end stage for frame n+2. During each pipeline step, the front end processing consumes one frame of digital audio data (although this is not indicated explicitly on the diagram).

Real world systems are more complex that the idealized system in many ways. The processing time for a frame is often highly variable and data dependent, especially in the search stage, and the processing time for each processing stage is likely to be quite different. The DSP and CPU are likely from time to time to be diverted to run other software tasks, either to service interrupts, or under control of an operating system scheduling regime. Communications between the processing elements may be stalled due to competing bus traffic, for example for DMA transfers.

Recognition software may implement "frame dropping" and other advanced techniques, and as a result not all frames are processed through all of the recognition stages. Processing elements cannot respond instantly to the availability of data from the preceding processing stage, resulting in delays.

Finite buffer space for information storage between stages means that an earlier stage may be stalled waiting for a space to store its output data when following processing stages are "falling behind".

The accelerator has been carefully designed to mitigate these problems. FIG. **19** shows a pipelined system with real world complications added. The following text describes the interactions shown in the diagram. The time for the processing of various stages has been varied to show difference examples of buffering of data between stages and stalling of processing elements. The numbering corresponds to the numbered bubbles on the diagram. The text also makes reference to memories used to pass information into and out of the accelerator, which are shown in FIG. **17**.

The front end has computed the feature vector for the first frame, FV1. It is loaded into the accelerator, causing the accelerator to commence computation of the distances for the frame.

The feature vector for the second frame, FV2, is passed to the accelerator. The accelerator is still busy computing the distances for the first frame, so this feature vector is held in the accelerators FV memory.

The front end has computed the feature vector for the third frame, FV3, but it cannot be delivered to the accelerator as the FV memory is still full, holding FV2. The time period labelled "stall delivering FV3" is the period between FV3 being computed and when it can first be delivered. Of course the DSP could be diverted to some other useful task, or it could buffer FV3 in its own memory and commence computation of FV4.

The arrow labelled "FV3 earliest" shows the earliest time that the feature vector FV3 can be delivered to the accelerator, which is immediately after the accelerator loads FV2 into its internal registers and frees the memory.

The arrow labelled "FV3 latest" shows the latest time that the feature vector FV3 can be delivered to the accelerator without causing the accelerator to stall for lack of a feature vector to work on. This example shows that there is a wide tolerance in the system to when FV3 can be delivered without affecting system performance—an entire accelerator frame processing time. The accelerator provides an interrupt and a status bit that can be polled to indicate the state of the FV memory. Regardless of which control method the DSP uses, the system tolerates long latency without reducing performance.

The accelerator signals to the search stage that the distances for frame 1 are available in result memory A. The search stage commences processing for frame 1, reading distances from result memory A.

The search stage has finished its processing for frame 1 before the accelerator has completed computing the distances for frame 2, so the search stage stalls waiting for the accelerator. Of course the CPU could be diverted to other useful tasks, including any search stage tasks that can be completed before distances are available. Again, an interrupt and status bits are provided by the accelerator to control this interaction between the accelerator and the CPU.

The accelerator completes the distance calculation for frame 2, and signals to the CPU that it can commence reading the distances from result memory B.

The accelerator has completed the computation of distances for frame 3 before the search stage has finished processing frame 2. As soon as the search stage signals to the accelerator that it has completed frame 2 by releasing

memory B, the accelerator signals that the distances for frame 3 are available, allowing the search stage to immediately move on to processing stage 3.

The accelerator is stalled waiting for feature vector FV4, due to the long processing time in the front end. In real systems this may be caused by a number of factors, including the DSP being diverted to another task for a period of time, or simply due to the DSP performance being insufficient to keep up with the accelerator's throughput.

The accelerator restarts as soon as FV4 is delivered by the front end.

The search stage is stalled waiting for distances for frame 4, a follow on consequence of the late delivery of FV4 to the accelerator.

FV5 is delivered to the accelerator while it is still computing the distances for the previous frame, allowing it to move onto processing frame 5 without stalling when it completes frame 4.

The accelerator signals that the distances for frame 4 are available and the search stage commences processing frame 4.

FV6 is delivered to the accelerator while it is still computing the distances for frame 5.

The availability of distances for frame 5 is signalled by the accelerator to the CPU.

Even though the accelerator has already received FV6, it is stalled waiting for a free result memory it can write the distances into.

The search stage releases result memory B at the end of its processing of frame 4. This allows the accelerator to start computing distances for frame 6.

The accelerator completes computing distances for frame 6 and signals to the CPU that the results are ready.

Further embodiments of the invention relate to systems with a Front End and Search Stage on a single processor. This single processor performs the front-end and search stages of the recognition process, with the accelerator performing the distance calculations. This configuration is likely to be the most common configuration where the accelerator is added to an embedded system such as a PDA or mobile phone that has a high-performance embedded application processor (but no DSP) available to run speech recognition software.

In this configuration the accelerator still operates in a concurrent pipelined manner with the front-end and search stages; it is just that the front-end and search stages are running on a single processor.

CPUs in embedded systems are generally relatively simple compared to desktop PCs and servers. Embedded CPUs in mobile phones, PDAs, in-car navigation and telematics systems, and other consumer electronics devices are typically "simple uni-processors" by which we mean not multi-processors and not supporting hardware multi-threading support—i.e. they can only execute a single thread of instructions (although they can be diverted to a different thread under the control of interrupts, exceptions, or an operating system). For this discussion, except where explicitly stated otherwise, we are talking about embedded processors that are "simple uni-processors". We are also using the term "CPU" and "processor" interchangeably.

FIG. **20** shows how the system can be pipelined with a single embedded processor working with the accelerator, in an idealized system. In this idealized system, the time taken for the processor to perform the front-end and search stage processing is identical to the time the accelerator takes to perform the distance calculations, and the processor is never delayed, stalled, or diverted to other tasks.

The time axis is divided into equal "pipeline step" times. If the step time is the same as a frame time then the system is processing audio in real time. If the step time is short than a frame time, the system is operating faster than real time, which may be valuable for batch processing of audio data, or catching up after some type of delay. If the step time is loner than a frame time, the system is operating slower than real time, which may be acceptable in some applications, for example where the person dictating pauses occasionally allowing the system to "catch up".

The first two pipeline step times are used to fill the pipeline. By the third pipeline step time, the pipeline is full, and all three stages of the recognition process are processed every pipeline step time, with each stage working on the processing of a different data frame.

During the first pipeline step time, the CPU completes the front-end processing for frame 1, and the feature vector passed to the accelerator so that distance calculations can start in the accelerator.

During the second pipeline step time, the front-end processing for frame 2 is completed by the CPU, in parallel with the accelerator computing the distances for frame 1.

During the third pipeline step time, the CPU runs the search stage for frame 1 reading distance results from the accelerator, and then runs the front-end processing for frame 3. The accelerator runs in parallel computing the distances for frame 2 using the feature vectors computed in the previous pipeline step.

Each subsequent pipeline step time is similar to the third pipeline step, with the frame numbers incremented. So for pipeline step n=4, 5, 6, etc. the CPU runs the search stage for frame n−2 and then runs the front-end for frame n. In parallel the accelerator computes the distances for frame n−1.

Note that with this situation the production of feature vectors and the consumptions of results run in lock step—one feature vector is produced every time the search stage consumes one frame of distance results. This considerably reduces the need for elastic buffering between the distance accelerator and the software system.

Again, real world systems are more complex than the idealized system in many ways. For example, the processing time for a frame is often highly variable and data dependent, especially in the search stage, and the processing time for each processing stage is likely to be quite different. The CPU is likely from time to time to be diverted to run other software tasks, either to service interrupts, or under control of an operating system scheduling regime. Communications between the CPU and accelerator may be stalled due to competing bus traffic, for example for DMA transfers.

Recognition software may implement "frame dropping" and other advanced techniques, and as a result not all frames are processed through all of the recognition stages. Processing elements cannot respond instantly to the availability of data from the preceding processing stage, resulting in delays.

Finite buffer space for information storage between stages means that an earlier stage may be stalled waiting for a space to store its output data when following processing stages are "falling behind".

The accelerator has been carefully designed to mitigate these problems. FIG. **21** shows a pipelined system with real world complications added. The following text describes the interactions shown in the diagram. The time for the processing of various stages has been varied to show difference examples of buffering of data between stages and stalling of processing elements. The numbering corresponds to the numbered bubbles on the diagram. The text also makes

reference to memories used to pass information into and out of the accelerator, which are shown in FIG. **21**.

At the end of front-end processing for frame 1, the CPU passes the feature vectors for frame 1 to the accelerator, allowing distance calculations to start.

To fill the pipeline, the CPU immediately performs the front-end processing for frame 2 and passes the feature vectors to the accelerator. They are stored in the accelerator so that the accelerator can load them into its internal engine for processing as soon as the distance calculations for frame 1 are complete.

The CPU is now ready to start the search stage for frame 1, however the distance computation in the accelerator has not completed. The box labeled "Prep search stage 1" indicates a period where the CPU could do any preparatory search stage processing that does not require distances. Alternatively the CPU could be diverted to another task, or it can stall waiting for the distance results.

The accelerator signals that the distance results for frame 1 are now available. The search stage can commence on the CPU, reading distance results from the accelerator's Result Memory A. The accelerator automatically starts processing the feature vector for frame 2 that was delivered earlier.

The accelerator signals to the CPU that it has completed the distance calculations for frame 2. The CPU is still busy running the search stage for frame 1, and has still to run the front end for frame 3, so the completion signal is ignored until the CPU has need of it, when it starts running the search stage for frame 2. As the accelerator has not received the feature vector for frame 3, it has no task it can perform and so it stalls.

The CPU delivers the feature vector for frame 3, restarting the accelerator, which writes results into Result Memory A. Note that Result Memory A was previously "released" at the end of the processing of search stage 1, informing the accelerator that it can overwrite the old results held in that memory with new results. The CPU then runs the search stage for frame 2, reading distance results from the accelerator's Result Memory B.

The accelerator signals to the CPU that the distance results for frame 3 are complete in Result Memory A. Again, the CPU ignores this signal until it is ready to commence the search stage for frame 3, and the accelerator stalls waiting for the next feature vector.

The CPU delivers the feature vector for frame 4, restarting the accelerator. It them starts search stage 3 reading distances from the accelerator's result memory A.

Search stage 3 and front end 5 were completed quickly, so that the feature vector for frame 5 is delivered to the accelerator while it is still running. The CPU performs all the preparatory work it can for the search stage, and then stalls waiting for distances to be available.

The accelerator signals that the distances for frame 4 are available. The CPU starts search stage 4, reading the distances from the accelerator's result memory B. The accelerator automatically starts computing distances for frame 5 using the feature vector delivered earlier.

Note that with the pipelining scheme just presented which alternates front-end and search stage processing, the accelerator can never stall waiting for a result memory to be released because the feature vector the accelerator needs as an input is not produced until after the memory has been cleared. The CPU task schedule could be described with the shorthand F1 F2 S1 F3 S2 F4 S3 F5 S4.

It can seen readily from FIG. **21** and the accompanying description above, that with this CPU task schedule, that the operation of the CPU and accelerator are tightly coupled and whenever the processing time for a pipeline step on the CPU is unequal to the processing time on the accelerator, then it quickly results in one of them stalling.

In a system where either the CPU or the accelerator is always (or at least in most instances) slower than the other in completing its processing task, stalling is inevitable and throughput may not be improved by adding additional buffering (e.g. additional result memories or FIFO depth on the feature vector interface). [Unless there are other difficulties to mitigate such as very low-bandwidth buses linking the CPU to the accelerator, or competing bus traffic such as DMA transfers that can significantly reduce performance.

If the processing time for a pipeline step is highly variable on either the CPU or Accelerator, such that there are regular changes in which one of them takes the longer time, then significant enhancements in throughput may be obtained by adding additional buffering between them. This may have the disadvantage of adding additional latency through the system. This buffering may be implemented by adding additional depth to the Feature Vector FIFO, and/or adding additional Result Memories. Alternatively it may be by implementing elastic buffers in software.

If the accelerator has sufficient buffering on the feature vector interface, e.g. by including the optional FIFO shown in FIG. **21**, then increasing the pipeline depth by computing extra front end frames up front will improve throughput at a cost of extra latency. This will result in a CPU task schedule of F1 F2 F3 S1 F4 S2 F5 S3 F6 S4 or F1 F2 F3 F4 S1 F5 S2 F6 S3 F7 S4, etc.

Of course it would also be possible to dynamically schedule whether the CPU should run the front-end or search stage code based on the availability, or not, of distance results and/or the space available to store more feature vectors and distance results. Given the fundamental relationship that one set of distance results is produced by the accelerator for every feature vector it consumes, and the restriction stated at the start of this section that we are discussing single-threaded CPUs, dynamic scheduling is likely to add little performance improvement while being complex to implement.

On a CPU supporting multi-threaded operation (which is a technique for improving performance in the presence of high memory system latency), or on a multi-processor, the highest system performance is likely to be achieved by running the front-end and search stages concurrently as separate threads, rather than alternating between the front-end and search stages. The system operation is then very similar to that described above, if each thread is thought of as a separate virtual processor. In some embodiments of the invention, elastic buffers may be used between stages, to accommodate varying time delays between the processing of frames at each stage. Elastic buffers in the interfaces between the three recognition stages may also significantly enhance performance of such a system.

Operation of the accelerator and the software system is not constrained to a rigid repetitive frame time. Instead the timing of the system is controlled dynamically by the availability of feature vectors into the accelerator, and the consumption of results out of the accelerator.

This allows the recognition process to fall behind real-time operation and to "catch up" again. It also allows the system to be robust in complex systems where interrupts, DMA bus traffic, memory caches, operating system task switching, and competing real-time software tasks all make guaranteeing rigid timing constraints impractical.

Elastic buffers at the major data transfer points between processing elements maximize system performance by

allowing one element to continue useful processing whenever it can, regardless of whether other processing elements have been stalled or diverted to other tasks. Elastic buffers are useful for the following applications:

(1) For buffering raw audio data arriving in real time from the microphone(s) The data would typically be moved into the buffer by a DMA and/or high priority interrupt driven software process to ensure that hard real time constraints are met.

(2) For buffering feature vectors output from the front end. This is especially true if the front-end is implemented on a DSP or other processing element independent of the search stage.

(3) For buffering distance results from the accelerator. Again this is especially true if the front-end and search stages are implemented on independent processing elements. This buffering is also especially valuable if the interface between the search stage processor and the accelerator is very slow, has high latency, or is subject to competing bus traffic.

The Feature Vector and Results interfaces on the accelerator already provide a level of elasticity that should be sufficient in systems where the Front End and Search stages are running on the same CPU, and the accelerator has a high-bandwidth interface to the CPU with low level of competition from other bus traffic such as DMA.

Other embodiments may have deeper FIFOs. In one embodiment, the two result memories are an implementation of a FIFO where the implementation is exposed across the interface. It would be possible to have additional memories and use them in a round-robin fashion, or to use another implementation of a FIFO that hides the FIFO depth by only exposing one result memory at a time to the interface. An example of this would be to map each memory in turn to a memory range that is always used to read results, so that at any one time the "current" result memory is mapped to that location. This would hide the FIFO depth behind the interface so that the pipeline depth can be changed without changing the software (or hardware) that accesses the interface.

Slow or Conflicted Bus Interfaces may occur. In the system configurations described above, the search stage reads distances directly from the Accelerator's Result Memories. The result memories allow the search stage to randomly access the distance results, as they are needed. This will only give a high performance system if the interface or bus between the CPU and Accelerator provides the CPU with low latency, high bandwidth access to the Accelerator Result Memories.

In some systems it is not practical to provide such a low latency, high bandwidth interface between the CPU and the Accelerator. This may be for many reasons, including:

- The interface is a bus shared with other devices and/or processors that generate bus traffic that keeps the bus busy such that the CPU access to the accelerator is delayed waiting for the bus to be free. DMA transfers are one example of conflicting bus traffic.
- The interface is implemented as a serial bus, or narrow parallel bus, for cost or compatibility reasons, yielding low bandwidth and high latency
- The interface crosses an industry standard interface such as PCI, USB, SDIO, Cardbus, Firewire, PCIExpress etc. that introduces protocol overheads and/or latency as would be the case where the accelerator is in a plug-in peripheral to a system containing a CPU (or CPU and DSP).

In these situations it is possible to add another pipeline stage during which the result data is copied from the Accelerator into another memory (which we will call a "buffer memory") to which the CPU does have high bandwidth, low latency access. This copying can be performed by a DMA controller, or by software running as an interrupt handler, or a separate software thread on the CPU, or a software thread on another processor, or by other commonly known means. The device controlling the transfer may be on the CPU side of the interface or on the accelerator side of the interface, depending on the capabilities of the interface. This extra pipeline stage then allows an entire pipeline step time to copy the data from the accelerator to the other memory. The data copy can optionally be performed as a sequential burst, or a number of sequential bursts, to optimize transfers across the interface between the Accelerator and the buffer memory. The search stage, running on the CPU, can then randomly access the results in the buffer memory with low latency.

Similarly an extra pipeline stage can be added for transferring the Feature Vectors to the Accelerator, although given that the feature vectors are a much smaller object than the distance results, and that they are not accessed randomly across the interface but delivered once to the Accelerator for each frame, there are fewer situations where the performance benefit from this extra pipeline stage is as significant.

FIGS. **22** and **23** show idealized systems with the two extra pipeline stages for transfers added. The first shows a system with a CPU, DSP, and Accelerator with the two extra pipeline stages. The second shows a system with a CPU and DSP with the two pipeline stages added.

Other embodiments may add additional pipeline stages so that the distance computations are performed over two or more pipeline stages rather than being restricted to a single pipeline stage. This allows more time for the computation of the distance results for each frame, while maintaining the same throughput of distance results. This is similar in concept to the well-known pipelining techniques used to implement RISC microprocessors.

The Acoustic Models may occupy many megabytes of storage space. In embedded systems they will typically held in a non-volatile memory such as FLASH memory, or other types of non-volatile memory as is the appropriate currently economic technology. The Accelerator may read the acoustic model directly from the non-volatile memory, or more likely the acoustic model will be copied from the non-volatile memory into a higher bandwidth RAM memory of some type (e.g. SRAM or DRAM, SDRAM, etc.). The RAM memory may be part of the same integrated circuit as the Accelerator (i.e. "on chip"), or attached to the integrated circuit containing the Accelerator. ("off chip"). The having the RAM memory on-chip is particularly advantageous, as this configuration would give the lowest power consumption and highest bandwidth. An alternative solution would be for the acoustic model to be stored on a disk drive, or an another device accessible via a wired or wireless bus or network, and for the acoustic model to be loaded into the RAM memory from the disk drive, or over the wired or wireless bus or network.

On-chip RAM memory is however likely to come at a cost-premium to off-chip commodity memory devices. Many embedded systems face extreme cost pressures in the market place, especially high volume devices supplied by many manufacturers such as mobile phones. For a speech recognizer to be incorporated into such systems, the overall recognizer solution much be delivered at very low cost. There is therefore a significant economic benefit from reduc-

ing the cost of the RAM memory and the non-volatile memory. One way to reduce the cost of these memories is to compress the acoustic models and store the compressed model in the memory.

The Accelerator computes all of the distances associated with each Feature Vector that is delivered to it by the front end. This means that the accelerator must use the data from the entire Acoustic Model each time a new Feature Vector is presented to the Accelerator (with the exception of some specific optimizations that may determine that some data values need not be used in the distance computations). It is therefore possible to arrange for the accelerator to read through the acoustic data sequentially once for each Feature Vector, using each data value as it read from the Acoustic Model.

Reading through the entire Acoustic Model sequentially is ideally suited to placing a decompressor between the distance calculation engine and the RAM memory, because many compression schemes depend on the decompressor reading sequentially through the compressed data, and do not readily support random access into the compressed data.

Thus this configuration allows storing compressed data in the RAM memory (and the non-volatile memory), which can therefore be smaller and lower cost. There is no requirement for a memory that is large enough to hold the entire decompressed acoustic model, as the decompressed data is consumed as it is decompressed.

The decompression scheme could be any combination of:

- sign or zero extension or otherwise conversion of narrow or variable width data to a wider data format.
- sign or zero extension or otherwise conversion of narrow or variable width data to IEEE standard single or double precision floating point format
- codebook decompression of a binary bitstream, where the codebook is stored as part of the acoustic model data
- decompression of a Huffman or Lempel-Ziv compressed stream
- decompression of run length encoded data
- decompression of difference encoded data
- any well known compression scheme.

Additionally the model size can be reduced by using subspace distribution clustering and the decompressor and distance calculation engine designed to operate with such models.

The following description provides further information on the visible registers in some embodiments of the invention.

The register bank is designed to allow simple connection with an 8-bit data bus. Accesses to the register bank should be made with byte-read and byte-write accesses. To facilitate this, only the least significant 8 bits of the data bus are used.

The address decoder should respond to byte accesses to byte 0 or byte 3 within the designated word. This way the same code can be used on an ARM processor regardless of whether the processor is configured for little endian or big endian operation.

REGBASE=Base address of register bank

Register 0 Product Version

8-bit Product version ID
This register value is changed when the functionality of the hardware charges in a way that changes the hardware/software interface.
Address = REGBASE (and REGBASE+3 for big endian)
Read only. Reads with value 0x01
Writes ignored

Register 1 Implementation Version

8-bit Implementation version ID
This register value is changed for each implementation to facilitate bug tracking and version control.
Address = REGBASE+4 (and REGBASE+4+3 for big endian)
Read only. Reads with value 0x01
Writes ignored

Register 2 Interrupt Enable

8-bit Interrupt Enable register
Address = REGBASE+4*2 (and REGBASE+4*2+3 for big endian)
Readable and writeable.
Value following reset is 0 for bits 0,1,2,3
Bit 0 I 1 = Enable FV memory empty interrupt on Int0 output.
Bit 1 J 1 = Enable Results complete interrupt on Int0 output.
Bit 2 K 1 = Enable FV memory empty interrupt on Int1 output.
Bit 3 L 1 = Enable Results complete interrupt on Int1 output.
Bit 4 Future use. Read value is not defined. Writes ignored.
Bit 5 Future use. Read value is not defined. Writes ignored.
Bit 6 Future use. Read value is not defined. Writes ignored.
Bit 7 Future use. Read value is not defined. Writes ignored.

Software should use a read-modify-write code sequence when setting this register so that future use bits are written with the value read from them.

Register 3 Status

Address = REGBASE+4*3 (and REGBASE+4*3+3 for big endian)
Read-only register. Writes are ignored.
Value following reset is 0 for bits 0,1,2,3
Bit 0 FVMB Feature Vector Memory is Busy
0 = OK to write new value to FV memory (FV memory is empty)
1 = Don't write new value to FV memory (FV memory is busy)
Bit 1 CRM Current Result Memory
0 = Result memory A
1 = Result memory B

This bit toggles every time the Release Result Memory (RRM) command is issued to the accelerator (register 3). It indicates which memory software should read results from, and which result memory the RRM command should release next time it is issued. Note: Software should also check that results are available in the indicated result memory by checking the value of the MA and MB bits.

Bit 2 MA 1=Results are available is result memory A

The MA bit is set to 1 when the compute engine completes writing all of the distances for a frame into result memory A. This value is set to zero when the Release Result Memory command is issued and CRM==0.

Bit 3 MB 1= Results are available is result memory B

The MB bit is set to 1 when the compute engine completes writing all of the distances for a frame into result memory B. This value is set to zero when the Release Result Memory command is issued and CRM==1.

Bit 4 Dfault 1=Accoustic data CRC fault. Indicates that the CRC value at the end of the acoustic data did not match the CRC value computed while the acoustic data was read in.
Bit 5 Future use. Value undefined.

-continued

Bit 6 Future use. Value undefined.
Bit 7 Future use. Value undefined.

Register 4 Command

Address = REGBASE+4*4 (and REGBASE+4*4+3 for big endian)
Write-only register.

Software should perform byte writes to this register with a single bit set to 1 and all other bits set to zero to issue a command to the accelerator.

Behaviour is not defined if more than one command is issued within a single write (i.e. if more than 1 bit is a 1).

Value on read is not defined.

Bit 0 LFV Load Feature Vector from the FV memory.

Tells the accelerator that a new feature vector has been placed into FV memory ready for use by the accelerator. The FV will be loaded into the compute engine when the compute engine completes its current task (or immediately if the compute engine is idle). Computation of distances for this FV will commence when the FV is loaded AND there is a released result memory. (At reset, both result memories are released). The results for the first, third, fifth etc. feature vectors loaded are written to Result Memory A. The results for the second, fourth, sixth etc. feature vectors loaded are written to Result Memory B.

Bit 1 RRM Release Result memory. Indicates that the results in the current result memory can now be overwritten with new results. The current result memory is indicated by the CRM bit in the status register.
Bit 2 CFVI Clear the FV memory empty interrupt
Bit 3 CRI Clear the Results Complete interrupt
Bit 4 Future use - write 0
Bit 5 Future use - write 0
Bit 6 Future use - write 0
Bit 7 Future use - write 0

The present invention may be included in a mobile phone, or in another consumer electronics device, a PC, Laptop or tablet PC, in-car electronics system, or plug-in card for an electronic device.

The methods and apparatus according to the invention are not limited to embedded mobile systems, but may be used with any type of computing device, including PCs, workstations, laptop computers, notebook computers, handheld computing devices, dictation machines, PDAs (personal digital assistants), personal organisers, mobile phones, games consoles, video players, etc. Embedded computers according to the present invention may be embedded in a wide range of apparatus and machinery, such as electronic entertainment products, domestic appliances, factory machinery, vehicles, etc.

Embodiments of the present invention may be implemented as an IP core in a System on Chip (SOC) or a discrete device. Multiple chips or bare die may be connected to provide a larger vocabulary system.

Although the apparatus may include all or some of an input for accepting audio input, a circuit for digitising the audio input, a circuit for dividing the audio input into a plurality of time-frames, and a circuit for analysing the spectral components of each time frame to characterise the audio in that frame, this is not essential. The apparatus may also include a circuit for calculating a score indicating the probability or likelihood that each audio frame corresponds to each of a plurality of different possible states, but again, this is not essential. Instead, the apparatus may include an input for receiving the pre-calculated scores.

In the described embodiments, n-gram language models are used. For example, the Bigram language model is based on outputting the probability that a given second word directly follows a given first word. However, it is possible to use alternative language models instead. It is possible to use a mixture of different models. For example, it is possible to run a word-dependent n-best decode applying a bigram language model, and the resulting word link record or word graph could then be rescored using a trigram language model.

The Viterbi beam search algorithm may be used in embodiments of the invention, to perform the search. A word level search may be performed. Dummy states may be used for the start and end of an utterance

The grammar level model may be used to add distance elements into the likelihood calculation. This is not essential, but tends to give much better results.

The HMM models described in the embodiments comprise self connections to each states and single connections to a neighbouring state. However, the present invention is not limited to such HMM models, and for example, HMM models with additional connections between states may also be used. Alternatively, embodiments of the invention may use alternative types of acoustic models, rather than HMM models.

The system may comprise a plurality of language models (e.g. English, French). A model of words may comprise actual words, phrases, silence, lipsmack sounds, and/or other sounds for recognition by the apparatus.

As previously discussed, the phone level has been found to be the best level for lexical tree searching, rather than the state level or the word level. However, this invention is not limited solely to phone level searching.

Pipelining may comprises processing data in different parts of a circuit at the same time, i.e. parallel processing.

In the following claims, the calculation means may comprise the distance calculation engine described above.

Further embodiments of the invention include the methods carried out in the accelerator, and the methods carried out in the speech recognition circuit.

While the invention has been described in terms of what are at present its preferred embodiments, it will be apparent to those skilled in the art that various changes can be made to the preferred embodiments without departing from the scope of the invention, which is defined by the claims.

What is claimed is:

1. A speech recognition system comprising:

a first programmable device programmed to calculate a feature vector from a digital audio stream, wherein the feature vector comprises a plurality of extracted and/or derived quantities from said digital audio stream during a defined audio time frame;

a second programmable device programmed to calculate distances indicating the similarity between a feature vector and a plurality of acoustic states of an acoustic model wherein said feature vector is received by the second programmable device after it is calculated by the first programmable device; and

a third programmable device programmed to identify spoken words in said digital audio stream using Hidden Markov Models and/or Neural Networks wherein said

word identification uses one or more distances that were calculated by the second programmable device, wherein said identification of spoken words uses one or more distances calculated from a first feature vector; and

a search stage for using the calculated distances to identify words within a lexical tree, the lexical tree comprising a model of words.

2. The speech recognition system of claim **1** wherein the time period taken for said identification of spoken words overlaps the time period taken for said second programmable device to calculate one or more distances from a second feature vector.

3. The speech recognition system of claim **2** wherein the first programmable device, the second programmable device, and third programmable device are physically distinct processing circuits.

4. The speech recognition system of claim **3** wherein said physically distinct processing circuits are all formed on a single integrated circuit.

5. The speech recognition system of claim **2** wherein the first programmable device and the second programmable device comprise one physical uni-processor or multi-processor, and the second programmable device is a physically distinct processing circuit.

6. The speech recognition system of claim **5** wherein said one physical uni-processor or multi-processor and said second programmable device are all formed on a single integrated circuit.

* * * * *