

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CADENCE DESIGN SYSTEMS, INC.,

Petitioner,

v.

SEMICONDUCTOR DESIGN TECHNOLOGIES, LLC,

Patent Owner.

IPR2024-00024

U.S. Patent No. 7,971,167

PETITION FOR *INTER PARTES* REVIEW

OF U.S. PATENT NO. 7,971,167

TABLE OF CONTENTS

EXHIBIT LIST	4
I. BACKGROUND	6
II. The '167 Patent.....	10
A. Prosecution History	13
B. Patent Owner's Allegations.....	13
C. Priority Date	14
III. Identification of Challenge	14
A. Statutory Grounds.....	14
B. Relied-Upon Prior Art.....	15
1. U.S. Patent Application Publication 2006/0069538A1 ("Harada").....	15
2. U.S. Patent Application Publication 2006/0225022A1 ("Ezaki").....	16
3. CECS Publication TR03-42 ("Shin")	17
4. PCT Patent Publication WO2005/119528A2 ("Gutberlet")	18
5. U.S. Patent Application Publication 2002/0152453A1 ("Rittman").....	19
C. Standing.....	19
D. Discretionary Analysis for Review	19
IV. Field and Level of Ordinary Skill.....	21
V. Claim Construction.....	22
A. "Block"	22

B.	“Motion Level”.....	23
C.	“Latency Analyzer”	23
VI.	Challenged Claims.....	24
A.	Grounds 1 and 2	24
1.	Motivation to Combine (“MTC”)	24
2.	Claim 1	29
3.	Claim 2.....	64
B.	Ground 3.....	67
1.	MTC	67
2.	Claim 1	71
3.	Claim 2.....	84
C.	Grounds 4-6.....	85
1.	MTC	86
2.	Claim 3	87
3.	Claim 4.....	91
VII.	MANDATORY NOTICES AND FEES	92

EXHIBIT LIST

Exhibit No.	Description
1001	U.S. Patent No. 7,971,167 to Tsukamoto (“the ’167 Patent”)
1002	Prosecution History of U.S. Patent No. 7,971,167
1003	Declaration of Bill Lin
1004	U.S. Patent Application Publication No. 2006/0069538A1 to Harada (“ <i>Harada</i> ”)
1005	International Patent Application Publication No. WO2005119528A2 to <i>Gutberlet et al.</i> (“ <i>Gutberlet</i> ”)
1006	CECS Technical Report TR 03-42 (“ <i>Shin</i> ”)
1007	U.S. Patent Application Publication No. US2006/0225022A1 to Ezaki (“ <i>Ezaki</i> ”)
1008	U.S. Patent Application Publication No. US2002/0152453A1 to Rittman (“ <i>Rittman</i> ”)
1009	SDT's 1st Amended Complaint (NDCA 3:23-cv-01001-41)
1010	Declaration of R. Dömer
1011	Declaration of N. E. Frank-White for the Internet Archive
1012	Japanese Patent Application Publication No. JP2008204111A to Tsukamoto (annotated translation)
1013	SDT's Opposition to Motion to Dismiss (NDCA 3:23-cv-01001-46)
1014	So, “Using estimates in HLS for DSE”
1015	U.S. Patent No. 7,370,293 to Yamagata
1016	SDT's Infringement Contentions under PLR 3-3 (NDCA 3:23-cv-01001)
1017	SDT's Infringement Claim Chart for '167 patent (Ex B-1 to Ex-1016) (NDCA 3:23-cv-01001)
1018	<i>McFarland et al.</i> , “High Level Synthesis of Digital Systems”
1019	<i>Hsu et al.</i> , “Advanced Techniques for RTL Debugging”
1020	U.S. Patent No. 5,862,149 to Carpenter <i>et al.</i>
1021	U.S. Patent No. 5,910,897 to Dangelo <i>et al.</i>
1022	U.S. Patent No. 7,607,116 to Kwon
1023	U.S. Patent No. 5,752,002 to Naidu <i>et al.</i>
1024	U.S. Patent No. 7,080,365 to Broughton <i>et al.</i>
1025	U.S. Patent Application Publication No. 2006/0015314A1 to Roesner <i>et al.</i>
1026	Japanese Patent Application Publication No. JPH11149489A to Matsumoto (translation)

Exhibit No.	Description
1027	Cadence's Motion to Dismiss (NDCA 3:23-cv-01001-42)
1028	Cadence's Reply to SDT's Opposition to Motion to Dismiss (NDCA 3:23-cv-01001-50)

The '167 patent is directed to a methodology for chip design called behavioral synthesis. The patent purports to add a “correspondence table” and refer to it during conventional behavioral synthesis and chip-design steps. During prosecution, the Applicant emphasized the correspondence table as a point of novelty over the prior art, leading to allowance. However, correspondence tables were already well-known in the art.

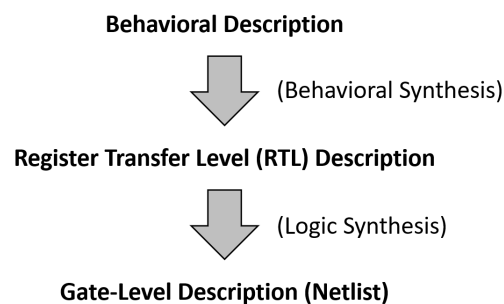
I. BACKGROUND

Electronic Design Automation (“EDA”) techniques have been used for decades to design integrated circuits (“ICs”), including logic synthesis, logic simulation, and latency analysis. Engineers frequently mixed-and-matched different EDA tools and techniques to suit their workflow. Ex-1003 ¶3.

Engineers designed ICs by writing code in a hardware description language (“HDL”), describing the design at the register-transfer level (“RTL”) meaning it described the transfer of signals between hardware registers, synchronized to the IC’s clock. Ex-1021, 3:54-60; Ex-1018, 302. Logic synthesis converted RTL code into a gate-level “netlist that includes a list of components in the circuit and the interconnections between the components.” Ex-1005, 1:17-27. Ex-1003 ¶4.

In the 1980s and 1990s, behavioral synthesis methodologies were developed to allow engineers to design ICs at a higher layer of abstraction than RTL. Engineers wrote behavioral code, often in the C programming language or

variations such as SystemC, that described IC behavior without lower-level RTL details such as the exact registers used or the timing of clock signals. *See, e.g.,* Ex-1005, 1:17-27, 5:33-7:17. Ex-1003 ¶5. Behavioral code was then synthesized into RTL which, in turn, was converted by logic synthesis into a netlist in the same manner as when engineers directly wrote RTL code. Subsequent steps of conventional design/manufacturing processes converted the netlist into a layout and eventually a physical chip.



Ex-1021, 9:13-10:6, Fig. 12; Ex-1005, 1:17-27; Ex-1003 ¶¶6-7.

Behavioral synthesis used a Control Data Flow Graph (“CDFG”) and state machine to convert behavioral code into RTL. A state machine is a conceptual model that moves from one state to the next, e.g., a stop light moving between its three states: green, yellow, and red. Ex-1003 ¶¶8-9. The synthesized RTL code included states from the finite state machine (“FSM”). *See* Ex-1018, §III.A, §III.B, §IV.D. Ex-1003 ¶10.

During synthesis, a “table of correspondence” was often saved to track which portions of behavioral code resulted in which states in the RTL. E.g., Ex-

1007, ¶1. This allowed engineers to check the corresponding behavioral and RTL code with ease. Ex-1004, ¶75; Ex-1007, ¶27; Ex-1006, ¶40. These types of tables were common. E.g., Ex-1007, Fig. 7; Ex-1004, Fig. 8; Ex-1006, Fig. 7:

FIG . 7

NODES OF RTL DESCRIPTION	CFG NODES	ROW NUMBERS OF BEHAVIORAL DESCRIPTION	STATES	CONDITIONS
MULTIPLIER 1	MULTIPLICATION 1	8	1	1
	MULTIPLICATION 2	11	1	2
	MULTIPLICATION 3	14	2	1,2
COMPARATOR 1	COMPARATIVE ARITHMETIC PROCESSING 1	7	1	1,2
ADDER 1	ADDITION 1	9	1	1
	ADDITION 2	15	2	1,2
i1	i1	8	1	1
		11	1	2
i2	i2	8	1	1
		11	1	2
i3	i3	9	1	1
i4	i4	14,15	2	1,2
s	s	7	1	1,2
REGISTER 1	r1	13	1	1,2

FIG . 8

#StateNm	StateID	CondID	Source:line
ST1	1	0	main.bdl:11
ST2	2	1	main.bdl:12,13
ST2	2	2	main.bdl:14,15
ST3	3	3	main.bdl:16
ST4	4	4	main.bdl:16

State Operations Table							Estimation	
State	NS	CS	Target operand	oper	operand1	operand2	# op	delay
S0	S0	0	a	=	in1		!= 1	1.6 ns
		0	b	=	in2			
		0	_status_	!=	start	true		
S1	S2	0	t1	abs	a		abs 2 max 1 min 1 >> 2 + 1 - 1	66.5 ns
		0	t2	abs	b			
		1	x	max	t1	t2		
		1	y	min	t1	t2		
		2	t3	>>	x	3		
		2	t4	>>	y	1		
		3	t5	-	x	t3		
		4	t6	+	t4	t5		
		5	t7	max	t6	x		
S2	S0	0	done	=	false			2.5 ns
		0	result	=	t7			

(a)

(b)

Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

Ex-1003 ¶11.

Behavioral synthesis was an iterative process where errors detected during simulation (discussed below) were fixed in behavioral code, which was then re-synthesized and the process repeated. *See, e.g.,* Ex-1020, 2:51-55 (“[E]rrors may be detected during the simulation and testing phases of the design cycle and then

fixed in the behavioral description.”); Ex-1018, §III.B.1. When synthesized RTL malfunctioned or was too slow, engineers traced the RTL back to the original behavioral code to address the root cause. Ex-1019, 362-364; Ex-1004, ¶¶75-76. Ex-1003 ¶12.

It was necessary to check the timing of the RTL code because behavioral synthesis introduced clock cycles into the design at the RTL level. RTL states (from the FSM) were scheduled to run at certain times, synchronized to a clock signal. Ex-1003 ¶13. Timing errors in RTL could result in erroneous processing and excessive delay. Ex-1003 ¶14. Therefore, RTL logic simulation was used to analyze and debug timing/latency issues. E.g., Ex-1019, 362-364; Ex-1005, 34:3:20; Ex-1004, ¶¶75-76. Ex-1003 ¶15.

Logic simulation, including at the RTL level, was viewed as an indispensable step. Ex-1018, §III.B.2. RTL simulation simulated the operation and signal flow described in the RTL code based on inputs provided to the simulation. E.g., Ex-1023, 1:21-33, 2:23-2:63; Ex-1024, 2:11-13; Ex-1025, ¶¶5-7, ¶¶10-11; Ex-1026, ¶66. This included, for example, simulating the operation of RTL code for each clock cycle and tracking the number of executions/clock cycles spent in each RTL state. E.g., Ex-1022, 10:45-12:58; Ex-1004, ¶¶131-137, Figs. 16-17; Ex-1005, 2:11-19, Table 38. Ex-1003 ¶¶16-17.

After RTL logic simulation, latency data was correlated back to the original behavioral code. It was known in the art, for example, to analyze RTL logic simulation data to report latency for each block of behavioral code. E.g., Ex-1004, ¶¶131-137, Figs. 16-17; Ex-1005, 2:11-19, Table 38 (listing latency for “for” loop blocks):

Process	Loop	Iterations	C-Steps	Latency	Duration
/fir_filter/fir_filter_proc	fir_filter_proc	Infinite	0	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_main	Infinite	2	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_for	8	1	8	160.00 ns
/fir_filter/fir_filter_proc	fir_filter_for_0	8	3	24	480.00 ns

Table 38

Ex-1003 ¶18.

II. THE '167 PATENT

The '167 patent does not purport to invent behavioral synthesis. Its specification admits behavioral synthesis, logic simulation, and latency analysis were known in the art. Ex-1001, 1:33-55. The original assignee Ricoh was an office equipment vendor that used existing EDA tools to design chips inside its products. Ricoh did not provide EDA software itself. Accordingly, the '167 patent relies on known EDA methodologies. *Id.*, 1:25-62, 3:19-26; Ex-1003 ¶21.

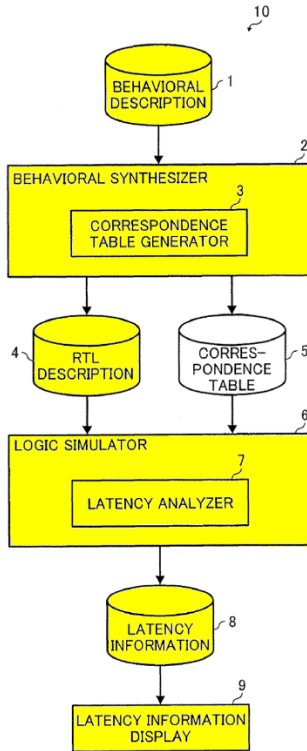
According to the '167 patent, prior art behavioral synthesis generates RTL from a behavioral description that “describes an algorithm of processing to be executed by the hardware directly in a motion level, and does not recognize a

concept of clock cycles.” Ex-1001, 1:36-47. “On the other hand, the RTL description recognizes a concept peculiar to the hardware, such as register and clock synchronism.” *Id.*; Ex-1003 ¶22.

The ’167 patent admits that, in the prior art, “[w]hen the behavioral synthesis tool generates the RTL description, a latency in the RTL description needs to be checked.” *Id.*, 1:48-55. Since “latency generally varies depending on type of the input data...a logic simulation (e.g., an RTL simulation) is performed on the RTL description to calculate the latency based on the simulation result.” *Id.* If logic simulation and latency analysis revealed that a portion of the RTL was too slow, “the behavioral description is checked to determine which part of the behavioral description consumes a great number of clock cycles (e.g., which part of the behavioral description has a great latency).” *Id.*, 1:56-62; Ex-1003 ¶¶23-25.

In short, the ’167 patent admits that nearly every component of its behavioral synthesis workflow, illustrated in Figure 1, was already known. Ex-1001, 1:36-62, Fig. 1 (highlights added to admitted prior art):

FIG. 1



Ex-1003 ¶24.

According to the '167 patent, when going from RTL back to behavioral code, “known methods may not check the behavioral description effectively.” *Id.*, 1:56-62. The '167 patent purports to solve this difficulty by writing down the correspondence between RTL and behavioral blocks in a “correspondence table” and then referring to it when calculating latency. *Infra* §II.A, Ex-1001, Fig. 3:

FIG. 3

BLOCK NAME	5	
	START STATE	FINISH STATE
file1_102_if	S10	S21
	S12	S30
file1_104_else	S10	S24
	S12	S29

Ex-1003 ¶¶25-28.

However, as explained above, correspondence tables were already known and used for correlating latency information from RTL logic simulation back to each block of behavioral code. *Supra* §I; Ex-1003 ¶29.

Claims 3-4 add limitations for common manufacturing steps. The '167 specification does not provide details, beyond a barebones recitation of language similar to the claims, instead referring to known manufacturing steps in general terms. Ex-1001, 6:38-44, 2:13-38; Ex-1003 ¶30.

A. Prosecution History

The '167 patent was rejected in two Office Actions. In response to the first, the Applicant argued the “correspondence table” and “latency analyzer” were missing in the cited art. Ex-1002, 00130-00132. In the second Office Action, the Examiner found the “correspondence table” argument persuasive but rejected the latency analyzer argument. *Id.*, 00143. The Applicant canceled the rejected claims and amended the dependent claims (which recited the correspondence table) into independent form (*id.*, 00209), leading to allowance. *Id.*, 00216; Ex-1003 ¶31.

B. Patent Owner's Allegations

In district court, SDT argued that a behavioral synthesis tool infringes because it allegedly “enable[s] tracing RTL descriptions to the originating behavioral descriptions” and “supports graphical analysis of the RTL with links to

source code.” Ex-1009, ¶¶89-92; Ex-1017, 13-19. SDT also alleged that displaying latency satisfies or implies calculating latency from logic simulation results. Ex-1009, ¶¶86-87; Ex-1017, 6-12; Ex-1003 ¶35.

C. Priority Date

Petitioner demonstrates invalidity as of February 19, 2007 regardless of whether the patent is entitled to that priority date. Ex-1003 ¶¶33-34.

III. IDENTIFICATION OF CHALLENGE

A. Statutory Grounds

Claims 1-2 are obvious under Grounds 1-3 because any differences between the claimed subject matter and prior art are such that the subject matter, as a whole, was obvious at the time the alleged invention. *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 406 (2007). Ex-1003 ¶38.

Claims 3-4 further recite conventional manufacturing steps. Grounds 4-6 combine *Rittman* with Grounds 1-3 to render those steps obvious. *Id.*

Petitioner requests *inter partes* review and cancelation of the challenged claims on the following grounds:

Grounds	Claims	Statutory Basis	Prior Art
1	1-2	§103	<i>Harada</i> and <i>Gutberlet</i>
2	1-2	§103	<i>Harada</i> , <i>Gutberlet</i> , and <i>Shin</i>
3	1-2	§103	<i>Ezaki</i> , <i>Gutberlet</i> , and <i>Shin</i>
4	3-4	§103	Ground 1 (<i>Harada</i> , <i>Gutberlet</i>) in view of <i>Rittman</i>
5	3-4	§103	Ground 2 (<i>Harada</i> , <i>Gutberlet</i> , and <i>Shin</i>) in view of <i>Rittman</i>
6	3-4	§103	Ground 3 (<i>Ezaki</i> , <i>Gutberlet</i> , and <i>Shin</i>) in view of <i>Rittman</i>

Ex-1003 ¶¶39-40.

B. Relied-Upon Prior Art¹

1. U.S. Patent Application Publication 2006/0069538A1 (“*Harada*”)

Harada was published March 30, 2006 and is §102(a)/(b)/(e) prior art.

Harada is directed to behavioral synthesis, including, e.g., a “correspondence table” for analyzing “a clock-based simulation” and correlating RTL simulation data back to behavioral code (Source:line). Ex-1004, ¶2, ¶117, ¶98, Fig. 8:

FIG . 8

#StateNm	StateID	CondID	Source:line
ST1	1	0	main.bdl:11
ST2	2	1	main.bdl:12,13
ST2	2	2	main.bdl:14,15
ST3	3	3	main.bdl:16
ST4	4	4	main.bdl:16

Ex-1003 ¶¶41-43.

¹ Pre-AIA 35 U.S.C. §102 applies. MPEP §2159.02.

Harada is in the same field of endeavor as the '167 patent (*infra* §IV), because it teaches a behavioral synthesis process for designing semiconductor ICs to “make the circuit design more efficient and easier.” Ex-1004, ¶¶75-76, ¶¶1-2; Ex-1003 ¶45.

2. U.S. Patent Application Publication 2006/0225022A1 (“*Ezaki*”)

Ezaki was published October 5, 2006 and is §102(a) prior art. *Ezaki* is directed to behavioral synthesis and teaches a “table of correspondence” between RTL and behavioral code, for “determining the relationship of correspondence between the RTL description and the behavioral description.” Ex-1007, Abstract, ¶¶23-24, Fig. 7:

FIG . 7

NODES OF RTL DESCRIPTION	CDFG NODES	ROW NUMBERS OF BEHAVIORAL DESCRIPTION	STATES	CONDITIONS
MULTIPLIER 1	MULTIPLICATION 1	8	1	1
	MULTIPLICATION 2	11	1	2
	MULTIPLICATION 3	14	2	1,2
COMPARATOR 1	COMPARATIVE ARITHMETIC PROCESSING 1	7	1	1,2
ADDER 1	ADDITION 1	9	1	1
	ADDITION 2	15	2	1,2
i1	i1	8	1	1
		11	1	2
i2	i2	8	1	1
		11	1	2
i3	i3	9	1	1
i4	i4	14,15	2	1,2
s	s	7	1	1,2
REGISTER 1	r1	13	1	1,2

Ex-1003 ¶¶46-48.

Ezaki is in the same field of endeavor (*infra* §IV) because it teaches a behavioral synthesis process for “logical circuit verification or simulation” of semiconductor ICs. Ex-1007, ¶¶56, ¶¶23-24, ¶42, ¶45; Ex-1003 ¶49.

3. CECS Publication TR03-42 (“*Shin*”)

Shin is a paper from the lab of Dr. Gajski—a prominent professor known for pioneering work in behavioral synthesis—and was published by Dr. Gajski’s Center for Embedded and Cyber-physical Systems (“CECS”) at UC Irvine. Ex-1010, ¶¶3-8. *Shin* was published and publicly available on the Internet by at least April 2004, as confirmed by co-author Dr. Dömer (Ex-1010, ¶¶9-17) and the Internet Archive (Ex-1011, 00004, 00008-00028). By April 2004, *Shin* was easily located by searching the CECS webpage; it was also linked and listed on the CECS list of Technical Reports and Dr. Gajski’s and Dr. Dömer’s UC Irvine webpages. Ex-1010, ¶¶9-17. Therefore, *Shin* was publicly available by April 2004 and is §102(a)/(b) art. Ex-1010, ¶¶6-17; Ex-1003 ¶50-51.

Shin teaches behavioral synthesis, including a “state operations table” showing each block of behavioral code and corresponding RTL states. Ex-1006, ¶40, ¶1, Fig. 7:

		State Operations Table						Estimation	
State	NS	CS	Target operand	oper	operand1	operand2	# op	delay	
S0	S0	0	a	=	in1				
		0	b	=	in2				
	S1	0	_status_	!=	start	true	!= 1		1.6 ns
S1	S2	0	t1	abs	a				
		0	t2	abs	b				
		1	x	max	t1	t2			
		1	y	min	t1	t2			
		2	t3	>>	x	3			
		2	t4	>>	y	1			
		3	t5	-	x	t3			
		4	t6	+	t4	t5			
		5	t7	max	t6	x			
							abs 2 max 2 min 1 >> 2 + 1 - 1		66.5 ns
S2	S0	0	done	=	false				
		0	result	=	t7				2.5 ns

(a)

(b)

Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

Ex-1003 ¶52.

Shin is in the same field of endeavor (*infra* §IV) because it teaches an interactive IC design process using behavioral synthesis. Ex-1006, ¶4, ¶20, ¶38, ¶44, Abstract; Ex-1003 ¶53.

4. PCT Patent Publication WO2005/119528A2 (“*Gutberlet*”)

Gutberlet was published December 15, 2005 and is §102(a)/(b) art.

Gutberlet is directed at behavioral synthesis and using logic simulation to determine the latency for each block of behavioral code. Ex-1005, 1:10-32, 34:3-20.

Process	Loop	Iterations	C-Steps	Latency	Duration
/fir_filter/fir_filter_proc	fir_filter_proc	Infinite	0	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_main	Infinite	2	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_for	8	1	8	160.00 ns
/fir_filter/fir_filter_proc	fir_filter_for_0	8	3	24	480.00 ns

Table 38

Ex-1003 ¶¶54-56.

Gutberlet is in the same field of endeavor (*infra* §IV) because it teaches behavioral synthesis techniques for IC design. Ex-1005, 5:33-7:6, Fig. 1. Ex-1003 ¶57.

5. U.S. Patent Application Publication 2002/0152453A1 (“*Rittman*”)

Rittman was published October 17, 2002 and is §102(a)/(b)/(e) art. *Rittman* teaches conventional steps for manufacturing an IC, including layout and photomask creation. Ex-1008, ¶7, ¶¶49-58, Fig. 6; Ex-1003 ¶¶58-60.

Rittman is in the same field of endeavor (*infra* §IV) because is directed to layout/photomask techniques for IC manufacturing. Ex-1008, ¶2, ¶¶7-12, ¶¶58-59; Ex-1003 ¶61.

C. Standing

Petitioner certifies Petitioner is not barred or estopped from requesting this *inter partes* review and the ’167 patent is eligible for IPR.

D. Discretionary Analysis for Review

This Petition’s merits are compelling, which “alone demonstrates that the PTAB should not discretionarily deny institution under *Fintiv*.” See June 21, 2022 Director’s Memo, 3-5. The Grounds present compelling obviousness theories that, “if unrebutted in trial, would plainly lead to a conclusion that one or more claims are unpatentable by a preponderance of the evidence.” *Id.*, 4.

Even if the Board were to address *Fintiv*, the factors favor institution. There has not been an initial case management conference; there is no case schedule; claim construction has not been briefed; no trial date is set. The Court has not ruled on Petitioner’s pending motion to dismiss and scheduled the hearing for January 2024.

Denial under §325(d) is likewise unwarranted because none of the art from Grounds 1-6 were relied upon during the ’167 patent’s prosecution. Ex-1003, ¶¶40-41. *Harada*, *Gutberlet*, *Shin*, and *Rittman* were not cited or considered. *Ezaki* was cited, but the file history does not include any substantive analysis of *Ezaki*. *Supra* §II.A. Grounds based on references cited, but not relied upon, indicating they were not considered at any length, should not serve as a basis for denying institution under §325(d). *Becton, Dickinson & Co. v. B. Braun Melsungen AG*, IPR2017-01586, Paper 8 at 17-18 (PTAB Dec. 15, 2017) (Informative). This is the only IPR filed against the challenged claims to date.

Additionally, the Examiner erred by overlooking *Ezaki*'s teachings. The Applicant argued, and the Examiner agreed, that the prior art lacked a "correspondence table." Ex-1002, 00130-00132, 00143. But the Examiner did not analyze *Ezaki* in any Office Action and overlooked *Ezaki*'s teaching of a "table of correspondence." Ex-1007, ¶¶23-24, ¶48, ¶34, Fig. 7; Ex-1003 ¶32.

IV. Field and Level of Ordinary Skill

The field of the patent is IC design and manufacturing, including EDA tools and behavioral synthesis. That is the focus of the claims and specification, and it is how the patent describes its field and related art. Ex-1001, 1:17-2:37; Ex-1003 ¶62.

A POSITA would have a bachelor's degree in electrical engineering or a similar field with at least two years of experience in the field; or a master's degree with specialization in the field. A person with less education but more relevant practical experience may also meet this standard. Ex-1003 ¶¶63-64.

A POSITA with this background would have had the knowledge and skills to:

- Design/analyze/modify IC designs;
- Use known EDA techniques/tools, including behavioral synthesis, logic synthesis, RTL logic simulation, and latency analysis;
- Apply known behavioral synthesis techniques, including in conjunction with known EDA tools/techniques such as those listed above;

- Formulate and modify EDA scripts for use in semiconductor IC design and manufacturing processes;
- Program software to read, write, and modify data in a table, including by adding columns and rows to tables;
- Design and implement data files, databases, and interfaces to support interoperability and data exchange between EDA tools; and
- Apply known techniques to convert logical circuit designs to physical layouts

Ex-1003 ¶65.

V. CLAIM CONSTRUCTION

A. “Block”

Claim 1 defines a block as “representing an operation in a predetermined unit in the behavioral description.” Here, operation refers to behavior,² written in a programming language. Therefore, a block represents behavior, described by a sequence of code in a predetermined unit (between matching delimiters). *Id.*, ¶68-69.

The specification describes a block as representing motion (behavior) in a predetermined unit. Ex-1001, 3:25-26, 3:50-51, 4:26-49. For the sole embodiment,

² The ’167 patent uses “behavior,” “operation,” and “motion” interchangeably—the different terminology stems from a translation artifact for the same Japanese word. Ex-1003, ¶69, ¶77; *see also* Ex-1013, 4.

the block is a sequence of C code surrounded by matching curly braces. *Id.*, 3:37-42, Figure 2; *see also* Ex-1002, 00108; Ex-1003, ¶¶70-71.

Therefore, “block” means a sequence of code surrounded by matching delimiters, which are curly braces (“{}”) in the C language. Ex-1003 ¶¶68-72.

B. “Motion Level”

“Motion level” refers to the behavioral level. *Id.*, ¶¶73-78. The ’167 patent uses these terms interchangeably. *Supra* n. 2. The specification contrasts “behavioral” and RTL levels of abstraction and uses “motion” to refer to the behavioral level (and behavioral languages). E.g., Ex-1001, 1:40-45, 7:1-6, 2:3-31, 3:25-51.

C. “Latency Analyzer”

In district court, SDT argued that “‘latency analyzer’ should be construed to exclude latency analysis that does not provide any insight as to latency associated with a behavioral description.” Ex-1013, 00011; Ex-1003, ¶80.

However, no construction is needed because, as conceded by SDT (*id.*), the claim already states what it requires: “a latency analyzer configured...to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description[.]” Regardless, SDT’s construction is irrelevant to this IPR because each Ground renders the claim language obvious and teaches latency

analysis that provides insight to the latency associated with a behavioral description. *Infra* §VI.A.2[1c], §VI.B.2[1c]; Ex-1003 ¶79-82.

VI. CHALLENGED CLAIMS

A. Grounds 1 and 2

1. Motivation to Combine (“MTC”)

Ground 1 combines *Harada* with *Gutberlet*. Ground 2 further adds Shin’s teachings. A POSITA would have been motivated to combine for numerous reasons. Ex-1003 ¶85; ¶100.

The combination uses known techniques to improve similar devices and methods in the same way. *Harada* and *Gutberlet* share a similar purpose and architecture: both teach behavioral synthesis of C code to RTL (Ex-1004, ¶98; Ex-1005, 1:10-27); both simulate RTL (Ex-1004, ¶¶1-2; Ex-1005, 7:3-6, 34:13-20); and both report latency metrics for each behavioral block (Ex-1004, ¶¶72-73, Figs. 16-17; Ex-1005, 2:11-19, Table 38). *Shin* likewise teaches behavioral synthesis and latency analysis. Ex-1006, ¶¶20-24, ¶¶45-47, ¶¶55-58. *Shin* and *Harada* both generate correspondence tables. *Id.*, ¶¶40-43, Figs. 5, 7, 8; Ex-1004, ¶33, ¶112, Fig. 8. Therefore, a POSITA would naturally have looked to these similar references and their interrelated teachings. Ex-1003 ¶86, ¶101.

Grounds 1-2 were combinations of prior art elements according to known methods yielding predictable results. *Harada* teaches a process including steps for synthesizing behavioral code into RTL; performing RTL logic simulation; and

analyzing the results, along with a table showing correspondence between RTL and behavioral code, to calculate latency for each behavioral block. Ex-1004, ¶¶98, ¶¶117-118, ¶¶132-134. *Gutberlet* further teaches using logic simulation to calculate latency for each behavioral block, measured in clock cycles/ns, and reported in a table. Ex-1005, Table 38, 2:11-19. Ground 1 applies these teachings in a conventional manner, yielding predictable results (e.g., latency in clock cycles/ns, reported in tables), as explained below. Ex-1003 ¶87. Ground 2 adds *Shin*, which teaches a correspondence table (Ex-1006, Fig. 7) with a column for delay (*id.*, ¶22, Fig. 7). Using *Shin*'s "table format" (*id.*, ¶40), or adding *Shin*'s columns to *Harada* yields predictable results: *Shin*'s table or a table with additional columns, calculated using arithmetic calculations, consistent with *Gutberlet*'s teachings. Ex-1003 ¶103.

Adding *Gutberlet*'s teachings for logic simulation and calculating/reporting latency to *Harada* would have applied a known technique to a known device ready for improvement, yielding predictable results. It would have been natural to calculate the number of clock cycles needed to process input data for the simulation because *Gutberlet* teaches this known technique (e.g., Ex-1005, Table 38, 2:11-19), and *Harada* already counted steps on a clock-cycle basis, using simulation, including the time needed for each block. E.g., Ex-1004, ¶¶131-137,

Figs. 16-17; Ex-1003 ¶¶88-89. This was also a simple substitution of Gutberlet's logic simulation calculations and reporting into *Harada*. Ex-1003 ¶92.

Harada and *Gutberlet* seek to debug or edit behavioral code based on latency analysis. Ex-1005, 2:11-19, 7:3-6, Fig. 3; Ex-1004, ¶¶74-75, ¶122. As known in the art, this was a common practice when designing ICs with behavioral synthesis. *Supra* §I; Ex-1020, 2:51-55 (“[E]rrors may be detected during the simulation and testing phases of the design cycle and then fixed in the behavioral description.”). *Shin* facilitates the combination by including columns of behavioral code in a table, with timing metrics, lined up with behavioral code and RTL states. Ex-1003 ¶102.

The references provide express motivation to modify their teachings. Ex-1005, 35:32-36:2; Ex-1004, ¶78, ¶¶137-139. For example, *Gutberlet* teaches modifying its embodiments and a “data structure” for mapping behavioral code. Ex-1005, 6:30-32, 35:32-36:2. Therefore, a POSITA would naturally look to related teachings of similar references, as explained above. Ex-1003 ¶90.

Using *Gutberlet*'s teachings with *Harada* has several advantages. For example, *Gutberlet*'s timing report provides latency in clock cycles/nanoseconds needed for each behavioral block and adds several metrics to *Harada*'s simulation monitor and display, allowing access to whichever is more convenient for measuring the runtime impact of design choices. *Id.*, ¶91. A POSITA would have

been motivated to apply *Harada*'s "correspondence table" teachings to *Gutberlet*'s "intermediate database" because they perform analogous functions for tracking behavioral code to RTL. Applying *Harada*'s approach to *Gutberlet* simplifies the identification of block boundaries. Ex-1003 ¶93. The combination with *Shin*'s teachings further makes it easier for designers to identify behavioral blocks with excessive delay, which is a stated goal. Ex-1004, ¶¶74-75, ¶122; Ex-1005, 2:11-19, 6:33-7:6, Fig. 3. Furthermore, the combinations aid the design process by presenting blocks of related functionality that impact circuit path timing, improving visualization, and facilitating the designer's comprehension of the issue. Ex-1006, ¶20; Ex-1003 ¶104.

A POSITA was motivated to combine *Gutberlet*'s embodiments because they comprise complementary functionality for a single EDA process/interface and were meant to be used together to provide more convenience and allow designers to test design algorithms. Ex-1005, 3:18-4:4; Ex-1003 ¶94. Likewise, a POSITA was motivated to combine *Harada*'s embodiments, which were meant to be used together and describe aspects of the same process; *Harada* suggests doing so because "[t]he basic system configuration of the present embodiment is the same" in all three embodiments. Ex-1004, ¶95, ¶127, ¶131, ¶136; Ex-1003 ¶95.

Grounds 1 and 2 would not have changed the principle of operation for any reference because it uses their teachings in the manner as taught by each, with

Harada's behavioral synthesis process and logic simulator using *Harada's* teachings for carrying out a clock-cycle accurate simulation on the RTL description, and *Gutberlet's* RTL logic simulation teachings used to determine latency for each block, and a table reporting such details. *Shin's* table teachings would be used to aid a behavioral synthesis process and report results, consistent with *Shin*. Ex-1006, ¶40, ¶¶42-47, Figs. 5, 7-8. Furthermore, the references teach similar behavioral synthesis processes, and the combination would have used known techniques with simple arithmetic and already-existing data, while formatting the results into a table. Ex-1003 ¶96, ¶105.

A POSITA would have had a reasonable expectation of success for Grounds 1 and 2 because the references provide complimentary teachings, combined in a conventional manner as taught by each reference. The combination is based on compatible devices and techniques in similar behavioral synthesis processes for all three references. The combination and any necessary modifications used known logic simulation and simple calculations to present additional information, based on information at hand, to facilitate a stated purposes of both references (e.g., to debug or edit the behavioral description based on the latency analysis). *Id.*, ¶97, ¶106.

Behavioral synthesizers, logic simulators, and latency analysis were all well-known. The combination of Ground 1 is directed to simpler aspects of combining

known techniques, and correlating information collected or generated during the course of practicing these known techniques by using simple tables to record and retrieve such information. *Id.*, ¶98. The combination of Ground 2 adds table columns and uses arithmetic to convert latency metrics, thus repackaging information, which would have been straightforward to implement. *Id.*, ¶¶107-108. These types of activities were common in the field. *Supra* §IV; Ex-1003 ¶99.

2. Claim 1

1[pre]. A semiconductor design support device for designing a semiconductor integrated circuit, the device comprising:

Ground 1 teaches the preamble (e.g., a computer with *Harada*'s teachings). For example, *Harada* teaches a device “for creating a simulation model through behavioral synthesis” and “executing simulation using a simulation model” (Ex-1004, ¶1) to support “a circuit design process” (*id.*, ¶8). Ex-1003 ¶109. *Harada* further teaches a number of component devices (Ex-1004, ¶¶13-15, ¶¶20-23, ¶97, Fig. 1) and implements its teachings as software on computers, which was common for EDA and obvious to a POSITA (*id.*, ¶55, ¶114, ¶136; *supra* §I; Ex-1003, ¶¶110-111).

Gutberlet teaches 1[pre].

Gutberlet further teaches the preamble. *Supra* §VI.A.1 (explaining MTC). *Gutberlet* teaches a semiconductor design support device, including a “behavioral

synthesis tool [that] may be executed on any desired general purpose computer” “for generating an integrated circuit.[.]” where the “design can be optimized.” Ex-1005, 5:33-6:22, Fig. 1; Ex-1003 ¶112. This includes a computer running *Gutberlet’s* behavioral synthesis tool and teachings, used for designing semiconductor ICs. Ex-1005, 1:16-27, 1:11-13, 4:4-4:11, 3:18-4:3, 6:18-32, 5:33-6:17, Fig. 2; Ex-1003 ¶¶113-116.

Shin teaches [1pre].

Shin further teaches the preamble. *Supra* §VI.A.1 (explaining MTC). *Shin* teaches a semiconductor design support device, comprising “an RTL design environment” and “RTL synthesis tool” for designing semiconductor ICs by synthesizing RTL from behavioral code. Ex-1006, ¶¶6-7, ¶1, ¶3. *Shin* teaches or suggests implementing its teachings as software on a computer (*id.*, ¶7, ¶93), which would have been obvious for EDA software. Ex-1003, ¶111, ¶117.

Grounds 1 and 2 teach a semiconductor design support device comprising the below limitations. *Infra* §VI.A.2[1a]-[1e]; Ex-1003 ¶118.

[1a] a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level;

Ground 1 teaches [1a]. For example, *Harada* teaches a behavioral description configured to describe an algorithm of processing performed by hardware (e.g., logic circuit) in a motion (behavioral) level. *Harada’s* “source file”

is a “behavior level description (*algorithm description*) as the hardware description.”³ Ex-1004, ¶2, ¶14. The behavior level description is “of a logic circuit[.]” Id., ¶28, ¶32; Ex-1003 ¶119.

Harada’s exemplary source files describe an algorithm of processing in a motion (behavioral) level, written in C (a typical behavioral level language) for describing an algorithm to compute input values. E.g., Ex-1004, ¶82, ¶88, ¶100, ¶124, Figs. 4, 10.

FIG . 4

```

1 : in ter(0:1) i1;
2 : in ter(0:1) i2;
3 : in ter(0:1) i3;
4 : out reg(0:1) o1;
5 :
6 : process main(){
7 :   var(0:1) A;
8 :   var(0:1) B;
9 :   var(0:1) C;
10:
11:  A=i2; B=i3;
12:  if ( i1 != 0 ) {
13:    C = A + 10;}
14:  else {
15:    C = B + 20;}
16:  o1 = A+B+C;
17: }
```

FIG . 10

```

1 : in ter(0:1) i1, i2, i3, i4;
2 : out reg(0:1) o1;
3 :
4 : process main(){
5 :   var(0:1) A;
6 :   var(0:1) B;
7 :   var(0:1) C;
8 :   var(0:1) D;
9 :
10:  A=i2; B=i3;
11:  if ( i1 != 0 ) {
12:    if ( i4 != 0 ){
13:      C = A + 10;}
14:    else {
15:      C = A + 5;}}
16:  else {
17:    C = B + 20;}
18:  D = A + 1;
19:  o1 = A+B+C+D;
20: }
```

Ex-1003 ¶¶120-121.

³ All emphasis/annotations added unless stated otherwise.

Gutberlet teaches [1a].

The combination with *Gutberlet* further teaches limitation [1a]. *Supra* §VI.A.1 (explaining MTC). For example, *Gutberlet*'s "source code" is "a behavioral description of an integrated circuit." Ex-1005, 5:33-6:17, Fig. 1. The "behavior or functionality" (*id.*, 1:16-27) is described with high-level behavioral languages and is configured to describe IC algorithms in a motion (behavioral) level. *Id.*, 5:33-06:17; Ex-1003 ¶¶122-123.

Gutberlet includes source code examples that describe the "behavior or functionality of the hardware system" for exemplary semiconductor IC designs. Ex-1005, 1:16-27. These examples include, e.g., behavioral descriptions written using "HDL source code" (*id.*, 1:28-32, Tables 1-2) and "embodiments involving code written in the C language." (*id.*, 6:1-13, Tables 3-13, 18, 20, 22-24, 27-37). For example, Table 33 describes a processing algorithm to compute data. *Id.*, 29:4-6, Table 33:

```
int inp[16], tmp[14], out[12];

Loop1: for ( i = 0; i < 14; ++i ) {
    // Body1
    tmp[i] = inp[i]*3 + inp[i+1]*5 + inp[i+2]*3;
}

Loop2: for ( j = 1; j < 13; ++j ) {
    // Body2
    out[j-1] = - tmp[j-1] + tmp[j]*3 - tmp[j+1]*3;
}
```

Table 33

Ex-1003 ¶124. These examples, including Table 33, describe behavior using C and do not have to account for physical hardware details such as registers and clock cycles. *Id.*, ¶125.

Shin teaches [1a].

The combination with *Shin* (Ground 2) further teaches [1a]. *Supra* §VI.A.1 (explaining MTC). *Shin* teaches a behavioral description, which it refers to as a “specification model,” that describes the system that is being designed at a “behavior” level. Ex-1006, ¶17, Fig. 1; Ex-1003 ¶126. *Shin* starts with “the behavioral description of custom hardware in C/C++” which is written at the behavioral level. Ex-1006, ¶21, ¶33, ¶54, Fig. 3, Fig. 7:

```
behavior sra(  
  in  int in1,  
  in  int in2,  
  out int result,  
  in  bool start,  
  out bool done) {  
  
  while (true) {  
    do {  
      a = in1;  
      b = in2;  
    } while (start != true);  
  
    t1 = abs(a);  
    t2 = abs(b);  
    x = max(t1, t2);  
    y = min(t1, t2);  
    t3 = x >> 3;  
    t4 = y >> 1;  
    t5 = x - t3;  
    t6 = t4 + t5;  
    t7 = max(t6, x);  
  
    done = true;  
    result = t7;  
  }  
}
```

(a)

Ex-1003 ¶¶127-128.

[1b] an RTL description generated by reading the behavioral description, the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware;

An RTL description generated by reading the behavioral description.

Harada teaches an RTL “simulation model” that is “generated through behavioral synthesis of a behavior model description composed of a plurality of lines, the simulation model including a finite state machine structure[.]” Ex-1004, ¶¶21, ¶¶14-19, ¶¶40-43. *Harada*’s simulation model is an RTL description because it is the output of the behavioral synthesis process (*id.*, ¶¶97, ¶14, ¶28) and it describes the design at the register transfer level, including descriptions of registers and RTL states. *Id.*, ¶¶99, 97, Figs. 5-7. *Harada* teaches reading the behavioral description to generate the RTL simulation model. E.g., *Id.*, ¶102, ¶14, Figs. 1-2. Ex-1003 ¶129.

The RTL description configured to recognize a concept including register and clock synchronism particular to the hardware. *Harada* teaches exemplary RTL simulation models in Figures 7 and 11 that show code at the register transfer level, including the flow of data to and from the particular registers for the hardware (“Reg_A”-“Reg_D”) and other particular hardware features such as input ports (“in ter(0:1) i1, i2, i3, i4”) and output ports (“out reg(o:1) o1”). Ex-1004, Figs. 7, 11; Ex-1003 ¶130.

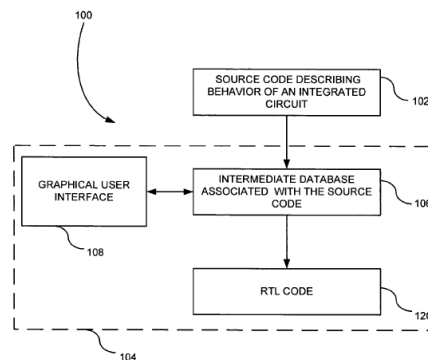
Harada further teaches the RTL simulation model that is configured to recognize a concept including clock synchronism particular to the hardware: for

example, the “cycle-based simulator...executes clock-based simulation” of the hardware using *Harada*’s RTL description. Ex-1004, ¶117, ¶2 (“performing *clock-level simulation*”), ¶98 (“...performs simulation preferably on a *cycle basis*.”).⁴ Ex-1003 ¶131.

Gutberlet teaches [1b].

The combination with *Gutberlet* further teaches [1b]. *Supra* §VI.A.1 (explaining MTC). *Gutberlet* teaches “RTL code” generated from “source code describing behavior of an integrated circuit.” Ex-1005, 1:16-27, Fig. 1:

FIG. 1



Ex-1003 ¶132.

To “generate RTL code,” *Gutberlet*’s “behavioral synthesis tool 104 reads in the source code file 102” which contains the behavioral description. Ex-1005, 5:33-6:17, 38:26-27, Fig. 3 (steps 304, 312); Ex-1003 ¶¶133-134.

⁴ *Harada*’s teachings are consistent with the ’167 patent, which admits that, in the prior art, an “RTL description recognizes a concept peculiar to the hardware, such as register and clock synchronism.” Ex-1001, 1:40-45.

Gutberlet teaches RTL code is synthesized to accommodate “clock” parameter constraints set by the designer. Ex-1005, 1:16-27. It is therefore configured to recognize a concept including clock synchronism particular to the hardware. *Id.*; Ex-1003 ¶135.

Additionally, RTL stands for “Register-Transfer Level” meaning it describes hardware in terms of registers and clock cycles. *Supra* §I; *see also* Ex-1004, ¶¶97-98, ¶2, ¶117; Ex-1005, 1:16-27. It was obvious for the RTL code to be configured to recognize a concept including register and clock synchronism particular to the hardware. Ex-1003 ¶136.

***Shin* teaches [1b].**

The combination with *Shin* further teaches [1b]. *Supra* §VI.A.1 (explaining MTC). *Shin* teaches a process with several steps for synthesizing RTL from the behavioral description. Ex-1006, Figs. 1-2; Ex-1003 ¶137. This starts with reading “the behavioral description” during preprocessing, refining it into a finite state machine (“FSMD”) and CDFG, and then generating an RTL description (e.g., a “structural RTL model”). Ex-1006, ¶¶23-¶27, ¶18, ¶33, ¶62, Fig. 11. The behavioral description is written in a C language (“SpecC” or “C/C++”). *Id.*, ¶90, ¶¶16-17; Ex-1003 ¶¶138-140.

Shin teaches “a cycle-accurate, structural description of the RTL” that reflects structural elements such as “registers.” Ex-1006, ¶19; ¶34-36, ¶44 (“cycle

accurate model”). Therefore, *Shin* teaches that the synthesized RTL description is configured to recognize a concept of registers and clock synchronism particular to the hardware. *Id.*; Ex-1003 ¶141.

[1c] a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description;

Ground 1 teaches [1c]. For example, *Harada* teaches a latency analyzer, referred to as a “simulation monitor,” that is configured to analyze RTL logic simulation results to calculate a latency in each block, explained below. *Harada* emphasizes “perform[ing] simulation *preferably on a cycle basis*” of RTL. Ex-1004, ¶98. *Harada*’s cycle-based simulator “executes clock-based simulation” with the simulation model. *Id.*, ¶117, ¶2. It is “a *simulator that receives a simulation model for executing simulation...*, the simulation model being generated through behavioral synthesis of a behavior model description composed of a plurality of lines, the simulation model including a finite state machine structure.” *Id.*, ¶21, ¶35; Ex-1003 ¶142.

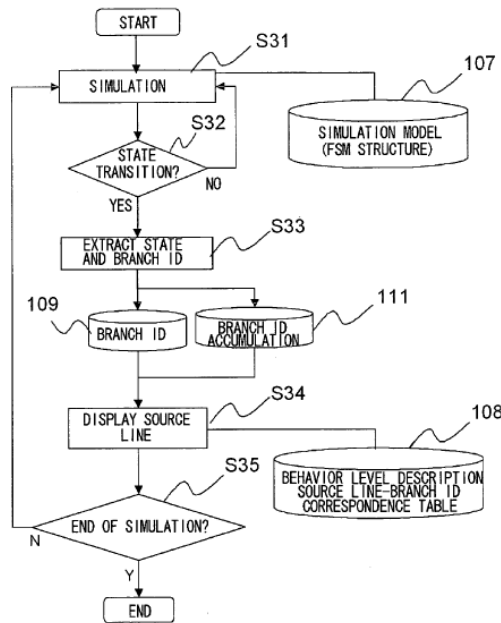
Harada monitors and analyzes the results of the clock-based simulation to compare back with the corresponding behavioral description. *Id.* ¶98; *infra*

§VI.A.3 (further explaining *Harada*’s latency analysis).⁵ A “simulation monitor (numeral 15 in FIG. 1) monitors the execution result of...the simulation.” Ex-1004, ¶117, ¶23, ¶36, ¶44-54. The simulation monitor analyzes the results of the simulation to “recognize[] that a state transition has occurred[.]” *Id.*, ¶118. The simulation monitor then refers to *Harada*’s “behavior level description source line—branch ID correspondence table” to identify the corresponding block in the source file, and “highlights (displays in reverse video) the source line corresponding to the branch ID....” *Id.*, ¶119-120, Figs. 9, 13; Ex-1003 ¶143.

Harada further teaches a latency analyzer that is configured to look at the results of a clock-cycle-based logic simulation and calculate latency by analyzing the number of executions for each block of behavioral code. Ex-1004, ¶76 (“The present invention allows the user to set a breakpoint on each branch ID or *to understand the accumulated number of execution of a branch ID*, further making the circuit design efficient and easier.”), Fig. 15 (step 111):

⁵ *Harada*’s latency analyzer and simulation teachings are consistent with the embodiments of the ’167 patent. Indeed, *Harada*’s teachings include the same overall algorithm and steps as the ’167 patent’s latency analyzer discussion. *See* Ex-1001, 4:39-62.

FIG. 15



Harada associates blocks with branch IDs, as explained further below. Ex-1003

¶145.

This calculation refers to the results of the logic simulation and the correspondence table. *Infra* §VI.A.2[1d] (explaining correspondence table). For each execution of a block, an execution count is “incremented by one and is accumulated in a branch ID accumulation unit” (Ex-1004, ¶132, ¶26, ¶39) where *Harada* correlates the branch ID with blocks in the example of figures 10, 16-17. *Id.*, ¶15, ¶133-134.

FIG . 16

BRANCH ID	NO. OF EXECUTIONS
0	5
1	5
2	1
3	0
4	5
5	5
6	5

Ex-1003 ¶146.

Harada displays the execution count “beside the line number of the behavior level description source (highlighted) for the branch ID.” Ex-1004, ¶135, Fig. 17:

FIG . 17

1 : in ter(0:1) i1, i2, i3, i4;	
2 : out reg(0:1) o1;	
3 :	
4 : process main(){	
5 : var(0:1) A;	
6 : var(0:1) B;	
7 : var(0:1) C;	
8 : var(0:1) D;	
9 :	
10: A=i2; B=i3;	
11: if (i1 != 0) {	
12: if (i4 != 0){	
13: C = A + 10;}	
14: else {	1
15: C = A + 5;}}	
16: else {	
17: C = B + 20;}	
18: D = A + 1;	5
19: o1 = A+B+C+D;	
20: }	

Ex-1003 ¶147.

“[T]he accumulated number of executions of each branch ID is displayed...at the completion of simulation, in the behavior level description source lines corresponding to all branch IDs[.]” Ex-1004, ¶136. Here, *Harada* teaches calculating latency for each block because the number of executions is a measure of latency indicative of the amount of time required during *Harada*’s clock-cycle simulation. This was one known way for calculating latency, in clock cycles over multiple iterations. *See, e.g.*, Ex-1014, 00003 (“To derive Cycles(d), the compiler multiplies C-steps(d), by the number of iterations of the loops in the nest.”). Converting to another metric, such as the clock cycles needed for each block to complete processing for the simulation’s input data (which *Gutberlet* teaches, *see infra* subsection on *Gutberlet*), would have been obvious because *Harada* already simulates each component for every clock cycle, for every circuit path. Thus, *Harada*’s simulation provides the necessary information and converting from one metric to another would have been a matter of simple arithmetic, e.g., by multiplying ratios. Ex-1003 ¶148.

Harada applies the above latency analyzer teachings for **each block** of behavioral code. For example, *Harada* teaches a latency analyzer configured to do so, including for the examples it teaches in Figures 10, 12, and 16. Figure 12 shows a “correspondence table” with blocks of behavioral code cited by line number (under “Source:line”), correlated to one or more branch IDs (“CondID”); *Harada*

teaches calculating latency, in terms of number of executions, for each behavioral block, shown by cross-referencing Figures 12 and 16, where each behavioral block is on the same row as a latency calculation showing the number of executions for that block. Ex-1004, Figs. 12, 16:

FIG . 12

#StateNm	StateID	CondID	Source:line
ST1	1	0	main2.bdl:10
ST2	2	1	main2.bdl:11-13
ST2	2	2	main2.bdl:14-15
ST2	2	3	main2.bdl:16-17
ST2	2	4	main2.bdl:18
ST3	3	5	main2.bdl:19
ST4	4	6	main2.bdl:19

FIG . 16

BRANCH ID	NO. OF EXECUTIONS
0	5
1	5
2	1
3	0
4	5
5	5
6	5

Ex-1003 ¶149.

The source behavioral code is shown in Figure 10. Each block in Figure 10 is represented in Figure 12, which correlates these blocks to one or more states and branch IDs:

FIG . 12

#StateNm	StateID	CondID	Source:line
ST1	1	0	main2.bdl:10
ST2	2	1	main2.bdl:11-13
ST2	2	2	main2.bdl:14-15
ST2	2	3	main2.bdl:16-17
ST2	2	4	main2.bdl:18
ST3	3	5	main2.bdl:19
ST4	4	6	main2.bdl:19

FIG . 10

```

1 : in ter(0:1) i1, i2, i3, i4;
2 : out reg(0:1) o1;
3 :
4 : process main(){
5 :   var(0:1) A;
6 :   var(0:1) B;
7 :   var(0:1) C;
8 :   var(0:1) D;
9 :
10:  A=i2; B=i3;
11:  if ( i1 != 0 ) {
12:    if ( i4 != 0 ){
13:      C = A + 10;}
14:    else {
15:      C = A + 5;}}
16:  else {
17:    C = B + 20;}
18:  D = A + 1;
19:  o1 = A+B+C+D;
20: }

```

Thus, Ground 1 teaches a latency analyzer configured to analyze the results of an RTL logic simulation to calculate a latency in each block of the behavioral code in Figure 10. Ex-1004, Figs. 10, 12; Ex-1003 ¶150.

The same concepts apply to Figure 4, where each behavioral block is correlated to the state and branch IDs in Figure 8. Ex-1004, Figs. 4, 8:

FIG . 4

```

1 : in ter(0:1) i1;
2 : in ter(0:1) i2;
3 : in ter(0:1) i3;
4 : out reg(0:1) o1;
5 :
6 : process main(){
7 :   var(0:1) A;
8 :   var(0:1) B;
9 :   var(0:1) C;
10:
11:  A=i2; B=i3;
12:  if ( i1 != 0 ) {
13:    C = A + 10;}
14:  else {
15:    C = B + 20;}
16:  o1 = A+B+C;
17: }

```

FIG . 8

#StateNm	StateID	CondID	Source:line
ST1	1	0	main.bdl:11
ST2	2	1	main.bdl:12,13
ST2	2	2	main.bdl:14,15
ST3	3	3	main.bdl:16
ST4	4	4	main.bdl:16

Ex-1003 ¶151.

Each block in *Harada*'s behavioral code is a sequence of C code surrounded by matching curly brace delimiters. Ex-1004, Figs. 4, 10. Ex-1003 ¶152. *Harada*'s correspondence tables (e.g., Figs. 8 and 12) correlate blocks of behavioral code with states and branch IDs. Ex-1004, ¶15, ¶97, ¶¶17-18, ¶154, Figs. 4, 10. Ex-1003 ¶154.

Harada teaches that one block can correspond to multiple branch IDs and RTL states.⁶ *E.g.*, Ex-1004, Figs. 10, 12 (multiple rows indicating the last block corresponds to multiple states and branch IDs), 16; Ex-1003 ¶155.

A POSITA would have been motivated to apply *Harada*'s teachings for the code in Figures 4 and 10 because *Harada* expressly applies its behavioral synthesis teachings to those examples. Ex-1004, ¶100, ¶124. Additionally, a POSITA would have recognized this code to have value, at a minimum, as learning and educational examples, for example to learn about *Harada*'s behavioral synthesis teachings. Ex-1003 ¶153.

Harada teaches **each block representing an operation in a predetermined unit in the behavioral description**. *Supra* §V.A (explaining how this refers to behavior in a block). As explained above, *Harada* teaches each block is a sequence of behavioral code, written in C, and representing the operation (behavior) in between matching curly braces. *E.g.*, Ex-1004, ¶100, Figs. 4, 10. These blocks are predetermined when the code is written because their boundaries are delimited using curly braces in the code. For any subsequent person (or tool) that reads the behavioral description, the blocks in the behavioral description are predetermined units delimited by curly braces. Ex-1003 ¶157.

⁶ This is consistent with the '167 patent, where blocks can have multiple rows in a correspondence table corresponding to multiple states. Ex-1001, Fig. 3.

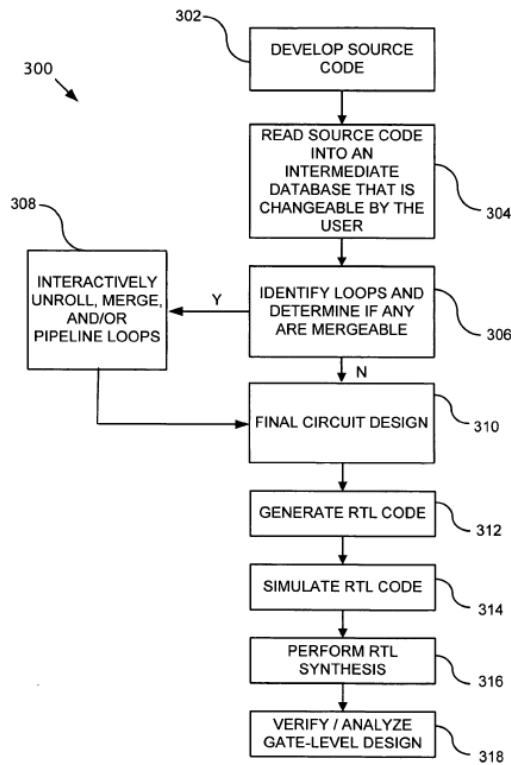
Each block represents behavior in a predetermined unit. For example, *Harada*'s source file is “composed of multiple lines” of code that describe an “algorithm,” in-between braces, in the behavioral description. Ex-1004, ¶97; Ex-1003 ¶158.

Gutberlet teaches [1c].

The combination with *Gutberlet* further teaches this limitation. *Supra* §VI.A.1 (explaining MTC). *Gutberlet* provides complementary teachings that reinforce those of *Harada* while expanding on the types of latency reported. *See id.*; Ex-1003 ¶159.

Gutberlet further teaches **a latency analyzer configured to analyze a result of logic simulation to calculate latency in each block**. *Gutberlet* “simulate[s] RTL code” and verifies and analyzes the synthesis results. Ex-1005, 6:3-7, 7:3-6, Fig. 3 (steps 314, 318):

FIG. 3



Ex-1003 ¶160.

Gutberlet teaches a framework that includes conventional RTL methodologies to “simulate and design” hardware, which was typically used to analyze latency: “the designer may run the source code through the synthesis tool” which typically “generates a report that the designer can use to analyze the performance of the circuit” including “*latency* of the circuit.” Ex-1005, 2:11-26. *Gutberlet* reports latency, including the number of clock cycles needed for each block to complete processing for input data. *Id.*, Table 38 (under “Latency”). This analysis identifies “what impact [its processing] within a design might have on the rest of the design.” *Id.*, 34:3-7. *Gutberlet* describes these analysis steps as part of

its behavioral synthesis process (*id.*, 6:18-19, 6:22-23) and explains these latency calculations are provided to the user “in response to a specific request” or automatically at the end of the process. *Id.*, 34:3-20. Therefore, *Gutberlet* teaches or suggests configuring a latency analyzer to analyze the results of the RTL logic simulation to calculate latency. Ex-1003 ¶¶161-162.

Additionally, this would have been obvious because *Gutberlet* performs RTL logic simulation, which was known to simulate clock cycles including the number of cycles spent in portions of RTL (*supra* §I; Ex-1022, 10:45-12:58; Ex-1023, 2:23-2:63, Figs. 6a-b; Ex-1025, ¶¶10-11, Figs. 5A-B; Ex-1026, ¶66; Ex-1004, ¶¶117, ¶2), and then presents timing reports with latency information regarding the number of clock cycles (under “Latency”) spent in each behavioral block. Ex-1005, 2:11-19, 34:3-20, Table 38. *Gutberlet*’s report is based on synthesized RTL because it includes clock cycles. It also includes the number of iterations spent in each loop which teaches, or at least suggests, that latency information from the RTL simulation was used in the timing reports. This would have been the most natural, straightforward, and obvious way to interpret and implement *Gutberlet*’s teachings, particularly in view of *Harada*’s teachings of determining the number of executions (iterations) of each block from the results of a clock-based RTL simulation. Ex-1004, ¶76, ¶¶132-135, Figs. 16-17. Since *Gutberlet* likewise calculates the number of iterations for each block, it would have been obvious to

use RTL logic simulation to calculate the latency information in *Gutberlet's* report.

Ex-1005, Table 38. Ex-1003 ¶163.

Gutberlet generates timing reports, based on RTL simulation, for timing analysis of whether the circuit functions correctly, and how fast/slow the processing is carried out. E.g., Ex-1005, 2:11-19, 34:13-15, Table 38. This was commonly analyzed for synthesized RTL, where performance depends on clock-cycle implementation in the RTL. Ex-1003 ¶164.

Process	Loop	Iterations	C-Steps	Latency	Duration
/fir_filter/fir_filter_proc	fir_filter_proc	Infinite	0	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_main	Infinite	2	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_for	8	1	8	160.00 ns
/fir_filter/fir_filter_proc	fir_filter_for_0	8	3	24	480.00 ns

Table 38

For the timing reports, *Gutberlet's* tool is configured to calculate a latency in each block of behavioral code, including for its example of Table 38. The corresponding behavioral code, listed in Table 33, includes two blocks: the first block for “Loop1” and the second block for “Loop2.” Ex-1005, 29:6-31:13.

```

int inp[16], tmp[14], out[12];

Loop1: for ( i = 0; i < 14; ++i ) {
    // Body1
    tmp[i] = inp[i]*3 + inp[i+1]*5 + inp[i+2]*3;
}

Loop2: for ( j = 1; j < 13; ++j ) {
    // Body2
    out[j-1] = - tmp[j-1] + tmp[j]*3 - tmp[j+1]*3;
}

```

Table 33

For the Table 38 timing report, *Gutberlet* calculated latency of 8 clock cycles (160ns) for the first “for” loop (Loop1) and 24 clock cycles (480ns) for the second (Loop2). *Id.* These numbers indicate the time needed to complete processing for the input data of *Gutberlet*’s logic simulation, which was used to evaluate whether the loop configuration was acceptable.⁷ *Id.*, 34:3-15, 2:11-19; Ex-1003 ¶165.

In Table 38, Loop 1 and 2 are labelled “fir_filter_for” and “fir_filter_for_0” respectively. The “Loop” column includes an entry for the first “for” loop, labeled with the file name “fir_filter” and “for,” indicating it is the block for the first “for” loop. The next entry is named “for_0,” indicating it is the block for the next (second) “for” loop. This approach of appending numbers was common for distinguishing repeating items (e.g., Microsoft Windows appending “Copy,” “Copy (1),” “Copy (2),” and on). *Gutberlet* starts numbering with 0, which was common for C because it uses zero-based numbering. Consistent with C parlance, where execution begins with the main function, Table 38 also lists “proc” and “main,” which encompass the entirety of the C code in Table 33 and report total latency including, e.g., initial startup time. Ex-1003 ¶167. In short, to generate the Table 38 timing report, *Gutberlet* calculates latency for the behavior described in

⁷ Table 38’s timing report shows the impact of not merging “for loops,” as compared with Table 39 where the loops have been merged. Ex-1005, 34:13-15. Table 33 shows the original behavioral code for this “Loop Merging Example.” *Gutberlet* includes a second example, in Table 36, that uses “while” loops instead of the “for” loops indicated in Tables 33 and 38. *Id.*, 29:6-30:6; Ex-1003 ¶166.

each of the two blocks (“for” Loops 1 and 2) in Table 33’s behavioral description. Ex-1003, ¶172.

A POSITA would have been motivated to apply *Gutberlet’s* teachings for the code in Figure 33 and timing report in Figure 33 because *Gutberlet* expressly applies its behavioral synthesis teachings to those examples. Additionally, this code provides value as an FIR filter (for DSP) and as educational examples for *Gutberlet’s* behavioral synthesis teachings, including for studying and observing the impact of loop merging and unrolling. Ex-1005, 20-34; Ex-1003 ¶168.

A POSITA would have found it obvious to combine these teachings, by using clock-based logic simulation on RTL code to calculate latency for each block of behavioral code, as *Harada* and *Gutberlet* teach, and using that data to create a timing report with latency information in terms of clock cycles and delay, as *Gutberlet* teaches. *Supra* §VI.A.1. The combination adds information, e.g., in columns, to the timing report and tables taught by *Harada* and *Gutberlet*, for the number of clock cycles and nanoseconds needed to process input data for a logic simulation. Since *Harada’s* simulation was based on clock cycles (Ex-1004, ¶98, ¶¶117-118), it would have been obvious to calculate time measured in clock cycles. Additionally, *Gutberlet* teaches reporting delay in nanoseconds (Ex-1005, Table 38), and converting between seconds and clock cycles was simple arithmetic. Ex-1003, ¶148, ¶169. In short, it would have been obvious based on the

combined teachings to configure a latency analyzer to generate reports with calculated latency for each block, expressed in clock cycles and delay, for *Harada's* simulation monitor because the extra information would be a simple, but useful addition to *Harada's* teachings of displaying latency in terms of number of executions. *Supra* §VI.A.1; Ex-1003 ¶173.

Gutberlet further teaches **each block representing an operation in a predetermined unit in the behavioral description**. For example, *Gutberlet* Table 33 includes behavioral code “written in the C language.” Ex-1005, 6:9-13. Following C convention, each block (Loop1 and Loop2) is a sequence of code surrounded by matching delimiter curly braces. *Id.*; Ex-1003 ¶170.

Each block represents behavior, written in C at the behavioral level, in the behavioral description. Loops 1 and 2 each represent the operation (behavior) in a predetermined unit, between matching braces for calculating values and assigning output for their respective equations. These blocks are predetermined for the same reasons explained above for *Harada*, which likewise teaches C code delimited by curly braces. Ex-1003 ¶¶171-172, ¶157.

Shin

The combination with *Shin* further teaches **a latency analyzer configured to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description**. For example, *Shin* teaches the

use of a “performance analysis tool” that “produces quality metrics for RTL design such as the delay.” Ex-1006, ¶22, Fig. 2. Among its functions, “[p]ost-synthesis estimation reflects synthesis decisions to the RTL design and *calculates delay...of each state and area of the design.*” *Id.*, ¶47. The results are produced in the “state operations table format” which is displayed, e.g., in a “scheduling & binding window.” Ex-1006, ¶40, Fig. 5 (estimation column):

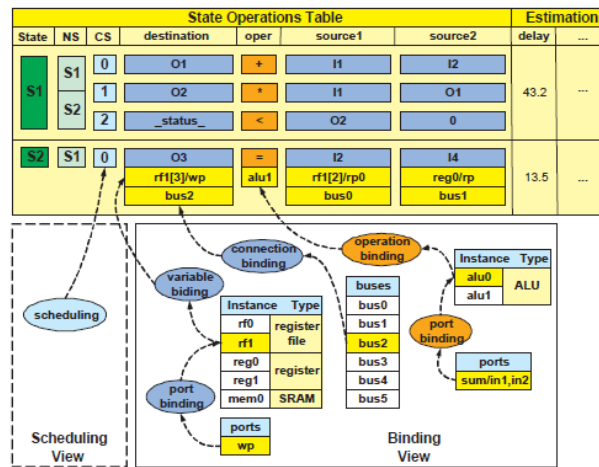
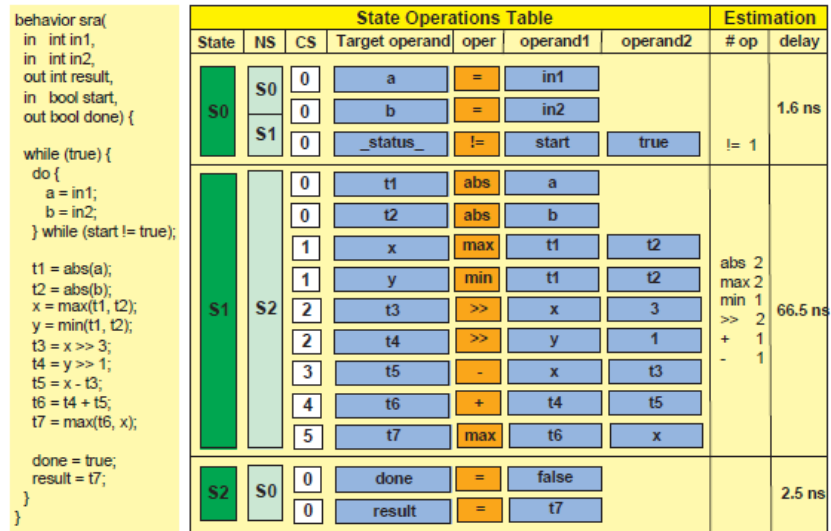


Figure 5. Scheduling & Binding window for an SF-SMD

Ex-1003 ¶174.

Shin provides additional teachings to Grounds 1-3 for reporting latency metrics in a table for each block of behavioral code. For example, *Shin* teaches calculating latency in each block of behavioral code, including for the example of Figure 7, reported in a table that includes each block of behavioral code, aligned with corresponding RTL states and delay information. Ex-1006, Fig. 7:



(a) (b)

Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

Ex-1003 ¶176.

Figure 7 reports latency for each block, shown in red below. Like *Harada* and *Gutberlet*, *Shin* teaches blocks of behavioral C code delimited by curly braces. Therefore, *Shin*'s blocks are analyzed in the same fashion, and *Shin*'s teachings are directed to each block of behavioral code for the same reasons explained above for *Harada* and *Gutberlet*. E.g., Ex-1006, Fig. 7:

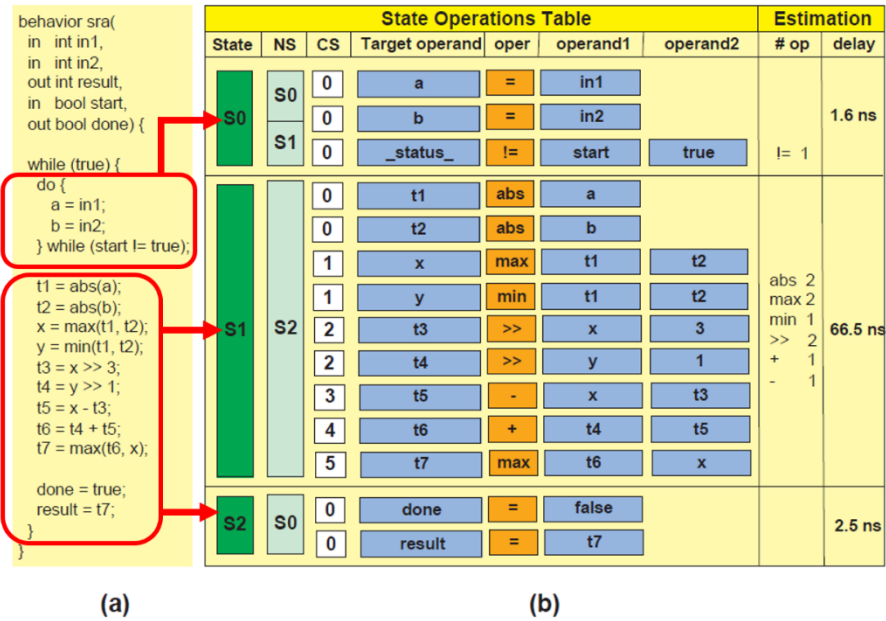


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

Ex-1003 ¶177.

A POSITA would have been motivated to use *Shin*'s table or add *Shin*'s columns to *Harada*'s correspondence table and to calculate latency in terms of clock cycles and nanoseconds. *Supra* §VI.A.1 (explaining MTC); Ex-1003 ¶175. *Gutberlet* teaches this (Ex-1005, Table 38), and converting from one latency metric to another was a simple matter. Ex-1003 ¶148. *Shin* teaches several reasons for displaying latency in a convenient manner, displayed for corresponding blocks, which complements *Harada*'s similar teachings. *See, e.g.*, Ex-1006, ¶¶22-23, ¶63, Figs. 4-5, 7, 9; Ex-1004, Fig. 17. Therefore, it would have been obvious to use *Harada* and *Gutberlet*'s teachings, with respect to performing an RTL logic simulation based on clock cycles, to calculate latency in clock cycles as taught by

Gutberlet, and to report the results in tables as taught by all three references including *Shin*. Ex-1003 ¶179.

A POSITA would have been motivated to apply *Shin*’s teachings for the code in Figure 7 because *Shin* expressly applies its behavioral synthesis teachings to that code. Additionally, the code provides value as educational examples, to learn about *Shin*’s teachings and behavioral synthesis generally. Ex-1003 ¶178.

[1d] a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description; and

Ground 1 teaches [1d]. For example, *Harada* generates a “behavior level description source line—branch ID *correspondence table*” “in which a correspondence between branch IDs and source line numbers in the behavior level description is stored.” Ex-1004, ¶33, Abstract. The “behavior level description source line—branch ID correspondence table” is in “a table format.” *Id.*, ¶16. Ex-1003 ¶180.

Harada includes exemplary correspondence tables in Figs. 8 and 12; each table includes information on “correspondences between *a state number (StateNm)*, *a state ID (StateID)*” in the RTL description, and “a source line (*Source: line*)” in the behavioral code. Ex-1004, ¶112, Figs. 8, 12:

FIG . 8

#StateNm	StateID	CondID	Source:line
ST1	1	0	main.bdl:11
ST2	2	1	main.bdl:12,13
ST2	2	2	main.bdl:14,15
ST3	3	3	main.bdl:16
ST4	4	4	main.bdl:16

FIG . 12

#StateNm	StateID	CondID	Source:line
ST1	1	0	main2.bdl:10
ST2	2	1	main2.bdl:11-13
ST2	2	2	main2.bdl:14-15
ST2	2	3	main2.bdl:16-17
ST2	2	4	main2.bdl:18
ST3	3	5	main2.bdl:19
ST4	4	6	main2.bdl:19

The “source line/state position correspondence table contain[s] the correspondence between algorithm description source lines and the state positions” generated by the “finite state machine (FSM)/data path allocation unit[.]” *Id.*, ¶99; Ex-1003 ¶181.

Each block in the behavioral description corresponds to a state in the RTL. For example, each block of behavioral code in *Harada*’s Figure 4 (highlighted below) is cited by line number in the correspondence table of Figure 8 aligned with a state (StateNm or StateID) in the RTL description. Ex-1004, ¶113, This also applies to Figures 10 and 12:

FIG . 4

```

1 : in ter(0:1) i1;
2 : in ter(0:1) i2;
3 : in ter(0:1) i3;
4 : out reg(0:1) o1;
5 :
6 : process main(){
7 :   var(0:1) A;
8 :   var(0:1) B;
9 :   var(0:1) C;
10:
11:  A=i2; B=i3;
12:  if ( i1 != 0 ) {
13:    C = A + 10;}
14:  else {
15:    C = B + 20;}
16:  o1 = A+B+C;
17: }

```

FIG . 8

#StateNm	StateID	CondID	Source:line
ST1	1	0	main.bdl:11
ST2	2	1	main.bdl:12,13
ST2	2	2	main.bdl:14,15
ST3	3	3	main.bdl:16
ST4	4	4	main.bdl:16

FIG . 10

```

1 : in ter(0:1) i1, i2, i3, i4;
2 : out reg(0:1) o1;
3 :
4 : process main(){
5 :   var(0:1) A;
6 :   var(0:1) B;
7 :   var(0:1) C;
8 :   var(0:1) D;
9 :
10:  A=i2; B=i3;
11:  if ( i1 != 0 ) {
12:    if ( i4 != 0 ){
13:      C = A + 10;}
14:  else {
15:    C = A + 5;}}
16:  else {
17:    C = B + 20;}
18:  D = A + 1;
19:  o1 = A+B+C+D;
20: }

```

FIG . 12

#StateNm	StateID	CondID	Source:line
ST1	1	0	main2.bdl:10
ST2	2	1	main2.bdl:11-13
ST2	2	2	main2.bdl:14-15
ST2	2	3	main2.bdl:16-17
ST2	2	4	main2.bdl:18
ST3	3	5	main2.bdl:19
ST4	4	6	main2.bdl:19

The behavioral blocks were explained above (*supra* §VI.A.2[1c]) along with these examples and the motivations to apply them (*supra* §VI.A.2[1c]; §VI.A.1). Ex-1003 ¶183.

Gutberlet teaches [1d].

Gutberlet further teaches or suggests including information for each block of the behavioral description. See §VI.A.1 (explaining MTC); §VI.A.2[1c] (explaining *Gutberlet*'s blocks and motivation to use its examples). *Gutberlet* also suggests a “data structure” for “mapping” back to behavioral code. Ex-1005, 6:30-32; Ex-1003 ¶184.

A POSITA would have been motivated to apply *Gutberlet*'s teachings and include each block of the behavioral description in *Harada*'s correspondence table, because *Gutberlet* teaches the advantages to allow a circuit designer to exercise “[c]omplete or enhanced control over how a design is synthesized to hardware.” Ex-1005, 3:7-13. *Gutberlet*'s insight, provided by block-level latency information, facilitates “precise control of a hardware implementation, allowing for interactive convergence on significantly better quality designs in much less time.” *Id.*, 3:7-13. *Harada* already analyzes the execution of blocks and stores information about latency in terms of number of executions of each block. Hence, *Gutberlet* reinforces *Harada*'s own teachings, which would result in easier identification of

information and facilitate the collection and presentation of other useful analysis results corresponding to each block. Ex-1003 ¶185.

Shin teaches [1d].

The combination with *Shin* further teaches a correspondence table in which **each block in the behavioral description corresponds to a state in the RTL description**. *Shin* provides additional teachings for organizing information in tables (e.g., Fig. 7) showing how each behavioral block corresponds to an RTL state, as indicated in the State and CS⁸ columns (e.g., S0-0, S1-0, S1-1, S2-0, etc.). A POSITA would have been motivated to apply *Shin*'s teachings by using *Shin*'s table format or adding *Shin*'s columns to the tables of *Harada* and *Gutberlet* to show correspondence relationships and report latency information. *Supra* §VI.A.1 (explaining MTC); Ex-1003 ¶186.

Shin's Figure 7 teaches a behavioral description that has two blocks of code delimited by curly braces. *Supra* §VI.A.2[1c] (explaining *Shin*'s blocks and motivation to use its examples). The inner do-while block corresponds to state S0-0; the remainder of the outer while block corresponds to states S1-0 through S2-0.

⁸ “CS” refers to control steps, which correspond to RTL clock cycles. Ex-1006, ¶¶41-43.

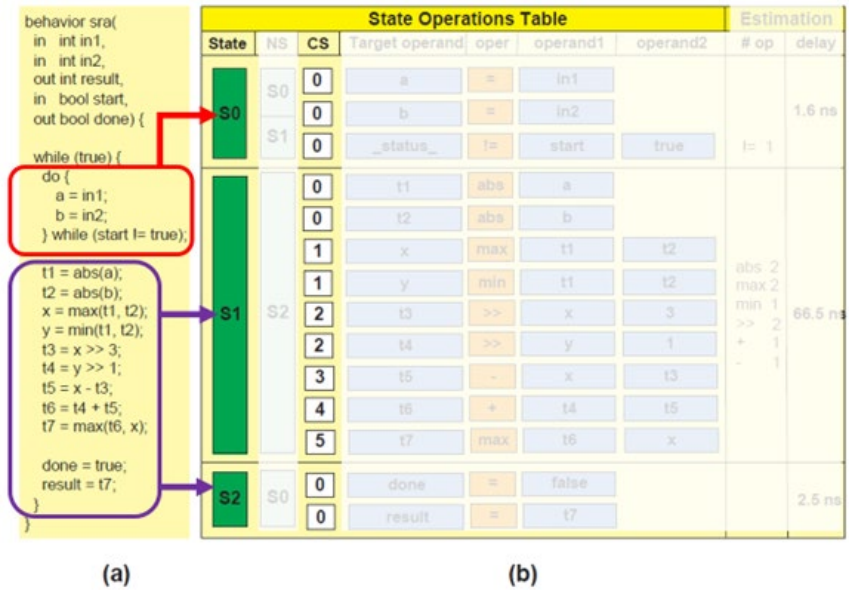
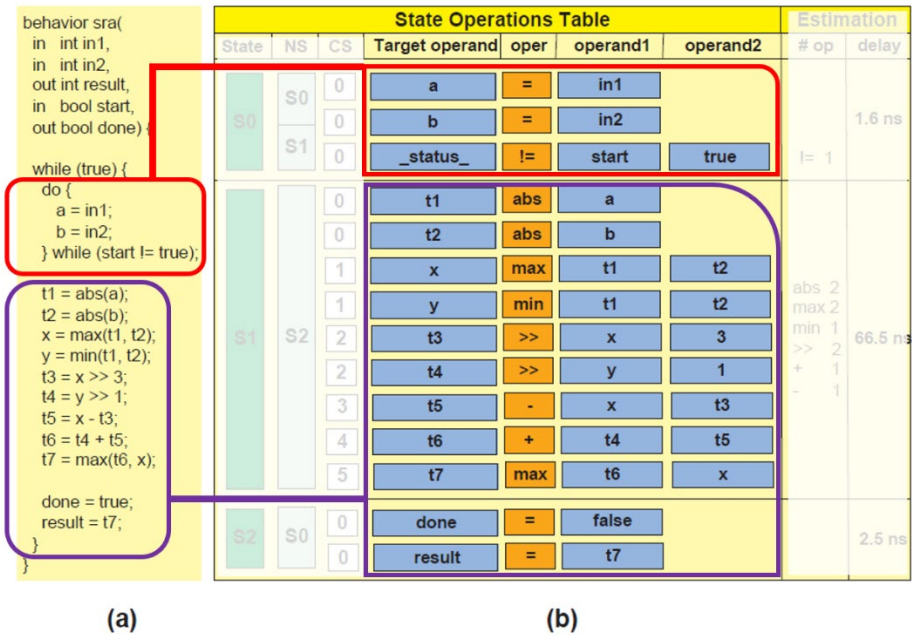


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

Ex-1003 ¶187.

Figure 7 teaches a correspondence table that includes each block of code in the behavioral description, where each block corresponds to at least one RTL state.⁹

⁹ See *supra* n. 6 (explaining correspondence between a block and multiple states).



Shin’s table format indicates correspondence by presenting behavioral code in the same row as corresponding RTL states. The operations within every line of each block are included in the table, in the row(s) of the state(s) corresponding to each block. Ex-1007, Fig. 7; Ex-1003 ¶¶188-189.

[1e] a correspondence table generator configured to generate the correspondence table.

Ground 1 teaches [1e]. For example, *Harada*’s “simulation model generation device” is a correspondence table generator configured to generate the correspondence table. Ex-1004, Abstract. “The simulation model generation unit... *generates* a behavior level description source line—branch ID *correspondence table* 13, which is a correspondence table containing the correspondence between

the assigned branch IDs and the behavior level description source lines.” Ex-1004, ¶97, ¶¶42-43, ¶16, ¶32, Fig. 2 (step 106); Ex-1003 ¶190.

Harada teaches steps taken for generating the correspondence table, including, e.g., identifying branch IDs and states (Ex-1004, ¶¶17-19) and combining those with a column for each behavioral block (*id.*, ¶99). Ex-1003 ¶191.

Gutberlet teaches [1e].

Gutberlet further teaches [1e]. *Supra* §VI.A.1 (explaining MTC). *Gutberlet* similarly teaches its behavioral synthesis tool generating an intermediate database. Ex-1005, 6:22-32. It would have been obvious for the database tables to include behavioral source line, branch ID, and RTL state correspondence information, e.g., as taught by *Harada*, because *Gutberlet* uses it to track RTL data back to behavioral blocks for its timing report. Ex-1004, 6:4-12. Ex-1003 ¶192.

Shin teaches [1e].

The combination with *Shin* further teaches [1e] (*supra* §VI.A.1 (explaining MTC)) with additional teachings of a correspondence table generator configured to generate a table showing the corresponding RTL states as indicated in the State and CS columns (e.g., S1-0), for each behavioral block along with latency. Ex-1006, ¶22, Fig. 7. Ex-1003 ¶¶193-196. For example, *Shin* teaches a scheduling & binding window that generates and displays a correspondence table (“state operations

table’’). Ex-1006, ¶¶40-42; Ex-1003 ¶194. *Shin*’s Figure 7 further teaches generating a table that includes corresponding blocks of behavioral code. Ex-1006, Figs. 5, 7:

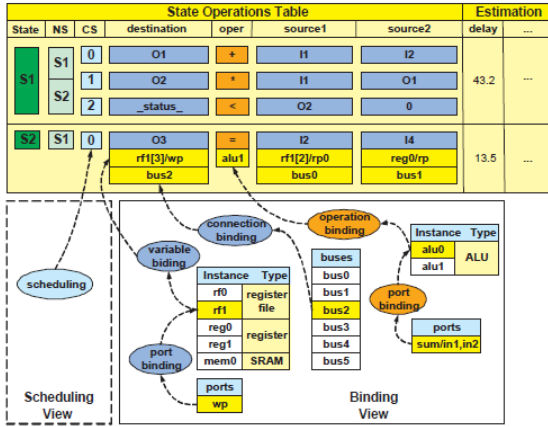


Figure 5. Scheduling & Binding window for an SF-SMD

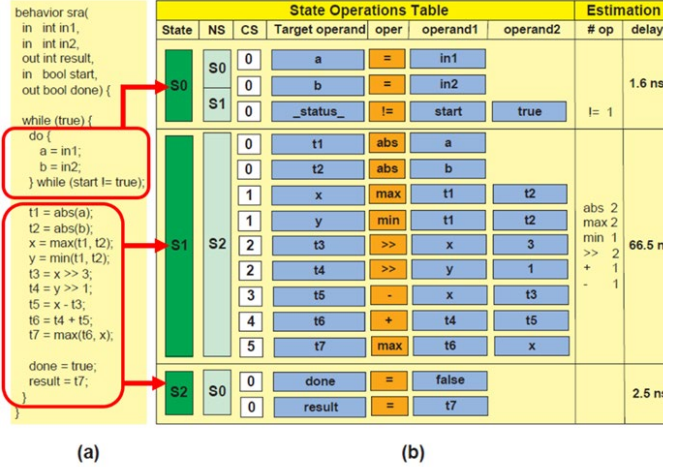


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

Ex-1003 ¶195.

A POSITA would have been motivated to combine these teachings from *Shin* with *Harada* and *Gutberlet* because the respective teachings perform analogous functions with respect to the correspondence table: they describe devices that generate a correspondence table as part of a behavioral synthesis process. *Supra* §VI.A.1. When combining teachings from each reference, as explained for [1a]-[1d], a POSITA would have been motivated to combine the respective teachings for devices that generate the correspondence table. Ex-1003 ¶196.

3. Claim 2

2. **The semiconductor design support device according to claim 1, wherein the latency analyzer refers to the result of the logic simulation and the correspondence table generated by the correspondence table generator to calculate the latency in each block in the behavioral description.**

Ground 1 teaches a semiconductor design support device according to claim 1. *Supra* §VI.A.2; Ex-1003 ¶197.

Ground 1 teaches claim 2. Ex-1003 ¶¶197-206. For example, *Harada* teaches a latency analyzer (referred to as “simulation monitor”) that calculates latency in each behavioral block by referring to the result of the RTL logic simulation (*supra* §VI.A.2[1c]) and the correspondence table (*supra* §VI.A.2[1d]-[1e]). *Harada*’s latency analyzer monitors state transitions based on the RTL logic simulation and refers to the correspondence table to correlate the execution back to blocks of behavioral code using the correspondence between the states, branch ID, and blocks of behavior-level code (indicated by source:line numbers).¹⁰ Ex-1004, ¶98 (“[S]imulation monitor 15 ... ***obtains the source line number of a behavior level description*** corresponding to the extracted branch ID from a behavior level description source line—branch ID ***correspondence table*** 13.”), Fig. 3; Ex-1003 ¶198.

¹⁰ This is consistent with the embodiments of the ’167 patent, where the latency analyzer likewise monitors RTL state transitions based logic simulation and refers to the correspondence table to correlate the execution back to behavioral blocks. Ex-1001, 4:44-62; *supra* n. 5.

Harada refers to the logic simulation results and the correspondence table together to calculate latency by counting the number of executions for each block. Ex-1004, ¶39, ¶26; *supra* §VI.A.2[1c]; Ex-1003 ¶199. *Harada* teaches the execution count being updated “beside the line number of the behavior level description source (highlighted) for the branch ID” during simulation. Ex-1004, ¶135, Fig. 17. The execution count for “each branch ID is displayed.” *Id.*, ¶136. Ex-1003 ¶200.

Latency measurements were explained above, including the obviousness of converting between them. *Supra* §VI.A.2.[1c]; Ex-1003 ¶201.

Gutberlet teaches Claim 2.

The combination with *Gutberlet* further teaches Claim 2. *Supra* §VI.A.1 (explaining MTC); Ex-1003 ¶¶202-206. *Gutberlet* teaches a latency analyzer configured to analyze a result of an RTL logic simulation to calculate latency in each block. *Supra* §VI.B.2[1c]. Therefore, *Gutberlet* teaches or suggests the latency analyzer refers to the result of the logic simulation to calculate the latency in each block—after all, *Gutberlet*’s Table 38 correlates RTL logic simulation data back to behavioral blocks. Ex-1005, 34:3-15; 7:4-6; 2:20-26; Ex-1003 ¶203.

To perform this correlation (Ex-1005, 34:3-7, 6:3-7, 7:3-6, Table 38), it would have been obvious to refer to *Harada*’s correspondence table, which lists RTL states and corresponding behavioral blocks for this purpose (to track

correspondence). Ex-1004, ¶33, ¶¶112-113, ¶¶119-120, Figs. 4, 8, 10, 12; Ex-1003 ¶204; *supra* §VI.A.2[1d]. The idea of referring to the table, e.g., by looking at a row and reading its columns, was obvious because that is how tables have been used since antiquity. Therefore, a POSITA would have found it obvious, based on the *Harada* and *Gutberlet*'s teachings, to refer to *Harada*'s correspondence table while calculating latency for each block. Ex-1003 ¶205.

The correspondence table is generated by the correspondence table generator. *Supra* §VI.A.2[1e]; Ex-1003 ¶206.

B. Ground 3

1. MTC

A POSITA would have been motivated to combine *Ezaki*, *Gutberlet*, and *Shin*. Ex-1003, ¶¶210-223. The combination uses known techniques to improve similar devices and methods in the same way. For example, all three references teach behavioral synthesis processes, starting with behavioral code in C and synthesizing RTL. *Ezaki* and *Gutberlet* both correlate latency back to behavioral code for timing analysis. Ex-1005, 2:11-19, 7:3-6, Fig. 3; Ex-1007, ¶42, ¶8, ¶¶47-49, ¶57, Fig. 1. *Ezaki* and *Shin* both generate correspondence tables to aid this endeavor. *Id.*, Abstract, ¶1, ¶48, ¶34; Ex-1006, ¶33, ¶112, Abstract, Fig. 8; Ex-1003, ¶210.

This was a combination of prior art elements according to known methods, yielding predictable results. Ground 3 starts with *Ezaki*'s behavioral synthesis teachings, including steps for synthesizing RTL code; analyzing latency for circuit paths; generating a timing report; and making/using a "table of correspondence." Ex-1007, ¶¶47-49, ¶¶56-57, Fig. 1. Ground 3 adds *Gutberlet*'s known teachings of using RTL logic simulation to determine/report latency for behavioral blocks. Ex-1005, Table 38, 2:11-19. *Shin* facilitates the combination with a table for correlating RTL latency data to behavioral blocks. Ex-1006, ¶22, Fig. 7. This combination yields predictable results (e.g., logic simulation latency reported for behavioral code, with *Shin*'s table or table columns). Ex-1003, ¶211.

Adding *Gutberlet*'s teachings would have applied a known technique (e.g., RTL logic simulation and latency analysis) to a known device (e.g., *Ezaki*'s behavioral synthesis process) yielding predictable results. Ex-1005, 7:3-6, Table 38. *Gutberlet* applied its teachings to behavioral synthesis, which *Ezaki* teaches (and thus was ready for improvement). *Id.*, 1:1-27, 3:18-4:13. Adding *Shin*'s teachings would also have applied a known technique (e.g., *Shin*'s correspondence table), which the prior art provides express teaching, suggestion, and motivation to combine. E.g., Ex-1004, ¶33, ¶¶112-113, Figs. 4, 8, 10, 12; Ex-1003, ¶¶212-214.

The references provide motivation. *Ezaki* suggests "a variety of changes or substitutions" to enhance its teachings for "exploiting the information generated in

the process of behavioral synthesis to render it possible to determine the sites of the behavioral description corresponding to the RTL description.” Ex-1007, ¶63. *Gutberlet* likewise teaches modifying its embodiments and a “data structure” for mapping behavioral code. Ex-1005, 6:30-32, 35:32-36:2. This would have been a simple substitution of *Gutberlet*’s logic simulation and latency analysis teachings into *Ezaki*, using the type of information suggested by *Ezaki*. Ex-1003, ¶¶215-217.

The combination provides numerous advantages. Using *Gutberlet*’s RTL logic simulation provides a fuller, more accurate latency picture based on actual simulation of every logic gate, providing cycle-accurate insight into circuit operation in action for a given set of input data. *See* Ex-1005, 2:11-19, 34:3-20, Table 38; Ex-1007, ¶8, ¶57. RTL logic simulation was common by the 1990s and often accelerated. *Supra* §I. Furthermore, the combination aids design by flagging behavioral blocks and facilitating the designer’s comprehension with improved visualization. Ex-1006, ¶20. Ex-1003, ¶216.

POSITAs were motivated to combine *Ezaki*’s two embodiments because they operate on the same principle “of taking the correspondence between the RTL description and the behavioral description” (Ex-1007, ¶58) and differ only in intermediate data used to generate a correspondence table. Therefore, teachings for the two embodiments were meant to be used together and result in the same underlying functionality. Ex-1003, ¶218. Similarly, POSITAs were motivated to

combine *Gutberlet's* embodiments, which comprise complementary aspects of a single EDA tool and were meant to be used together. Ex-1005, 3:18-4:4. The combination provides more functionality and flexibility. Ex-1003, ¶219.

The combination would not have changed the principle of operation, given the similarities between the references (discussed above), and because it uses the respective teachings in the manner as taught by each reference. *Ezaki's* teachings are still used for behavioral synthesis and correlating RTL information back to behavioral code. *Gutberlet's* RTL logic simulation is still used to determine latency, reported in a table. *Shin's* teachings are still used for behavioral synthesis and to report the correspondence of RTL and behavioral code, along with delay information, in a table. Ex-1006, ¶¶40-42, ¶¶45-47, Fig. 7. Ex-1003, ¶220.

A POSITA would have had a reasonable expectation of success when combining *Ezaki*, *Gutberlet*, and *Shin*. As explained above, their respective teachings are complimentary and combined in a conventional manner that does not change their principle of operation. The combination is based on compatible techniques in similar processes. Logic simulation, latency analysis, and correspondence tables were all known and used with behavioral synthesis. *Supra* §I. And *Shin's* teachings facilitate the combination, making it even easier to combine. Ex-1003, ¶221.

Additionally, the combination is directed to simpler aspects of combining known techniques, e.g., correlating information from known behavioral synthesis, logic simulation, and latency analysis by using simple tables to record and retrieve information. Ex-1003, ¶222. A POSITA would have been fully capable of implementing and modifying the elements and components taught by *Shin*, *Gutberlet* and *Ezaki*. *Supra* §IV. This was common for graduate students. Ex-1003, ¶223.

2. Claim 1

1[pre]. A semiconductor...:

Ground 3 teaches the preamble. *See* §VI.B.1 (explaining MTC); Ex-1003, ¶224. For example, *Ezaki* teaches a semiconductor design support device, comprising “a behavioral synthesis device, a logical synthesis device and a path reflecting device” (Ex-1007, ¶42, ¶24, Fig. 1), for designing a semiconductor IC, by “generating, from the behavioral description and the synthesis constraint, the register transfer level description” (*id.*, ¶23), which “is then transformed, by logical synthesis, into the gate-level description” of an IC (*id.*, ¶8). Ex-1003, ¶225-226. *Ezaki* applies its teachings to software on a computer (Ex-1007, ¶25), which would have been obvious for EDA. Ex-1003 ¶111, ¶227.

Ezaki supports the design process by “determining the relationship of correspondence between the register transfer level description and the behavioral

description” of “a semiconductor integrated circuit.” Ex-1007, ¶1, Abstract, *Id.*, ¶23; Ex-1003, ¶228. *Ezaki* “improv[es] the productivity in designing” ICs. Ex-1007, ¶56. Ex-1003, ¶229.

As explained above for Grounds 1-2, *Gutberlet* and *Shin* further teach the preamble. *Supra* §VI.A.2.1[pre] (explained in subsections for *Gutberlet* and *Shin*; that analysis applies here and is incorporated by reference). Ex-1003, ¶230.

Ground 3 teaches a semiconductor design support device comprising the below-discussed limitations. *Infra* §VI.B.2[1a]-[1e]; Ex-1003, ¶231.

[1a] a behavioral...;

Ground 3 teaches [1a]. *Supra* §VI.B.1 (explaining MTC). For example, *Ezaki* teaches a “behavioral description” including the example of Fig. 2. Ex-1007, ¶4, ¶29, Fig. 2; Ex-1003, ¶232. The behavioral description is configured to describe an algorithm of processing, including arithmetic steps and data and control flow, to be performed by the designed IC hardware in a motion (behavioral) level: “the behavioral description, which directly *describes the algorithm of the processing of interest by the behavioral level.*” Ex-1007, ¶3, ¶5, ¶4; Ex-1003, ¶233.

As explained above, *Gutberlet* and *Shin* further teach [1a]; that analysis applies here. *Supra* §VI.A.2.[1a] (*Gutberlet*, *Shin* subsections); Ex-1003, ¶234.

[1b] an RTL...

Ground 3 teaches [1b]. *Supra* §VI.B.1 (explaining MTC). Ex-1003, ¶¶235-240. *Gutberlet* and *Shin* further teach [1b]; that analysis applies here. *Supra* §VI.A.2[1b] (*Gutberlet*, *Shin* subsections); Ex-1003, ¶240.

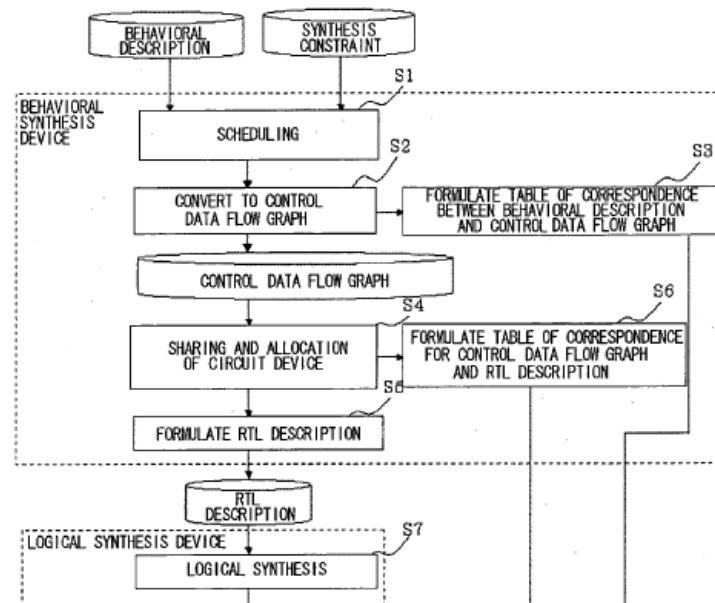
An RTL description generated by reading the behavioral description.

Ezaki “synthesizes the ... RTL description ... from the behavioral description[.]”

Ex-1007, ¶3, ¶4, ¶7. Ex-1003, ¶236. *Ezaki*’s behavioral synthesis process generates RTL by reading the behavioral description to obtain “the processing described on the behavioral level” and to extract information from the behavioral description.

Ex-1007, ¶5, ¶45, Fig. 1 (reading behavioral description stored top left):

FIG . 1



Ex-1003, ¶237.

The RTL description configured to recognize a concept including register and clock synchronism particular to the hardware. *Ezaki's* “RTL description” “takes account of the concept proper to hardware, such as registers or synchronization by clocks.” Ex-1007, ¶3. Ex-1003, ¶238. *Ezaki* teaches a graphical depiction of an “RTL description” and CDFG generated from the behavioral description,¹¹ which includes registers. Ex-1007, ¶¶30-31, Figs. 3-4; Ex-1003, ¶239.

[1c] a latency...;

Ground 3 teaches [1c]. Ex-1003, ¶¶241-253. Ground 3 adds *Gutberlet's* RTL logic simulation and latency analysis teachings to *Ezaki*; *Shin* provides additional teachings that facilitate the combination. *Supra* §VI.B.1 (explaining how and why the teachings would be combined).

Ground 3 teaches **a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block.** For example, *Gutberlet* teaches a behavioral synthesis tool that “simulate[s] RTL code” (Ex-1005, 6:3-7, 7:3-6, Fig. 3) and generates a report that calculates “latency of the circuit” including the clock cycles and nanoseconds

¹¹ *Ezaki's* teachings are consistent with the '167 patent's discussion of a behavioral synthesis process that uses a CDFG to generate an RTL description. Ex-1001, 4:21-38, 5:4-21.

needed for each block to complete its processing of input data (*id.*, 2:11-26, 34:3-15, Table 38). *Gutberlet* teaches limitation [1c] as explained above; that analysis applies here. *Supra* §VI.A.2[1c] (*Gutberlet* subsection); *see also, e.g.*, Ex-1005, 6:22-7:6, 34:3-20, 6:3-7, 2:11-26, Figs. 3, 33, 38:

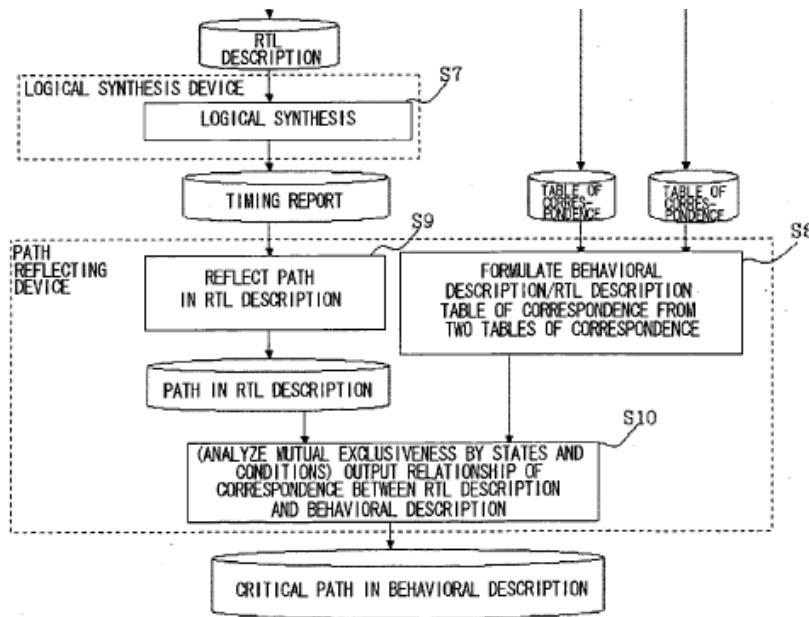
Process	Loop	Iterations	C-Steps	Latency	Duration
/fir_filter/fir_filter_proc	fir_filter_proc	Infinite	0	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_main	Infinite	2	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_for	8	1	8	160.00 ns
/fir_filter/fir_filter_proc	fir_filter_for_0	8	3	24	480.00 ns

Table 38

Ex-1003, ¶242.

Ezaki's teachings reinforce the obviousness of limitation [1c] and provide additional motivation to combine. *Ezaki* teaches a path reflecting device¹² configured to analyze the timing of circuit paths through the RTL description, in terms of the time needed for a circuit path to complete processing of input data (delay through the path); this is used to generate a "timing report" with latency information. Ex-1007, ¶42, ¶8, ¶¶47-49, ¶61, Fig. 1 (steps S7, "timing report", S9):

¹² *Ezaki* refers to the "path reflecting device" interchangeably as the "circuit designing assisting device" and the "path determining device." Ex-1007, ¶¶48-¶49, ¶69, Abstract.



Ex-1003, ¶243.

Ezaki tracks the correspondence between RTL and the original behavioral description (*infra* §VI.B.2[1d]) and refers back to the original behavioral description when latency exceeds design conditions: when “the delay is not in meeting with design conditions, or...there is not sufficient allowance for delay” “it may become necessary to correct the original behavioral description.” Ex-1007, ¶8. *Ezaki* thus expands on a prior art “method for correlating the circuit delay information in a timing report” “including logical block coordinates, net names, delay time, instance names of logical blocks and instance names of cells included in the logical blocks” of the gate description. *Id.*, ¶10-11. Ex-1003, ¶244.

Ezaki teaches or suggests conducting its latency analysis for each block because “the relationship of correspondence with the behavioral description may readily be taken *for any optional paths* of the RTL description.” Ex-1007, ¶57. At

the very least, this was obvious because *Ezaki* calculates timing through RTL paths and checks the correspondence between RTL and behavioral code for both optional and critical paths. *Id.*, ¶42, ¶8, ¶¶47-49, ¶57, Fig. 1; Ex-1003, ¶245.

Moreover, *Gutberlet* teaches a latency analyzer configured to analyze the result of logic simulation performed on RTL to calculate latency in each behavioral block, including for the example of Tables 33 and 38. *Supra* §VI.A.2[1c] (*Gutberlet* subsection). For Ground 3, it would have been obvious to use *Gutberlet*'s logic simulation teachings for analyzing latency, measured in terms of clock cycles and nanoseconds needed for each behavioral block to complete its processing. Ex-1005, 7:4-6, 34:3-20, 2:11-19, Fig 3, Tables 33, 38; Ex-1003, ¶¶246-247; *supra* §VI.B.1 (explaining MTC).

Shin's teachings facilitate the combination. *Shin* teaches organizing each block of behavioral code in a table along with its associated delay. Ex-1006, ¶47, Fig. 7:

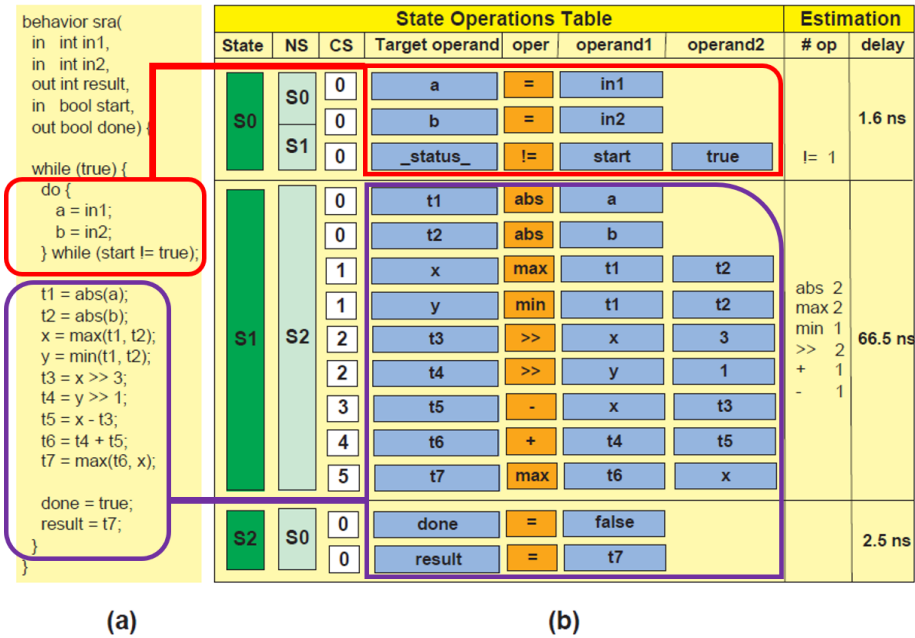


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

Ex-1003, ¶248.

Applying *Shin's* teachings to *Ezaki* and *Gutberlet*, it would have been obvious to analyze latency for each block and present the results in a table for each block (as *Shin* teaches). See Ex-1006, Fig. 7; *supra* §VI.B.1. *Shin* teachings for [1c] were explained above and apply here. *Supra* §VI.A.2[1c] (*Shin* subsection).

Ex-1003, ¶249-250.

Each block representing an operation in a predetermined unit in the behavioral description. *Gutberlet* and *Shin* teach this, as explained above; that analysis applies here. *Supra* §VI.A.2.[1c] (*Gutberlet*, *Shin* subsections); Ex-1003, ¶¶251-253.

Ezaki also teaches a behavioral description as a sequence of code delimited by matching curly braces. Ex-1007, ¶4, Fig. 2. Each of *Ezaki*'s blocks represents operation (behavior), written in behavioral C code, bounded by curly braces. *Id.* The analysis explained above for blocks in *Harada*, *Gutberlet*, and *Shin* also apply to *Ezaki*, which likewise uses C. *Supra* §VI.A.2[1c]; Ex-1003, ¶251.

Each block represents behavior in a predetermined unit. *Ezaki*'s behavioral description “describes the algorithm of the processing of interest by the behavioral level” as code in-between braces. Ex-1007, ¶3, Fig. 2; *Supra* §V.A.; Ex-1003, ¶252.

[1d] a correspondence...

Ground 3 teaches [1d]. *Supra* §VI.B.1 (explaining MTC); Ex-1003, ¶254.

A correspondence table in which...the behavioral description corresponds to...the RTL description. *Ezaki* teaches a path determining device “generat[ing] *a table of correspondence* determining the relationship of *correspondence between the RTL description and the behavioral description.*”

Ex-1007, Abstract, ¶1, ¶¶48-49, ¶34, Fig. 7:

FIG . 7

NODES OF RTL DESCRIPTION	CDFG NODES	ROW NUMBERS OF BEHAVIORAL DESCRIPTION	STATES	CONDITIONS
MULTIPLIER 1	MULTIPLICATION 1	8	1	1
	MULTIPLICATION 2	11	1	2
	MULTIPLICATION 3	14	2	1,2
COMPARATOR 1	COMPARATIVE ARITHMETIC PROCESSING 1	7	1	1,2
ADDER 1	ADDITION 1	9	1	1
	ADDITION 2	15	2	1,2
i1	i1	8	1	1
		11	1	2
i2	i2	8	1	1
		11	1	2
i3	i3	9	1	1
i4	i4	14,15	2	1,2
s	s	7	1	1,2
REGISTER 1	r1	13	1	1,2

Id., ¶44, ¶39, Fig. 12:

FIG . 12

INPUTS OUTPUTS OF NODES OF RTL DESCRIPTION	INPUTS OUTPUTS OF CDFG NODES	ROW NUMBERS OF BEHAVIORAL DESCRIPTION	STATES	CONDITIONS
MULTIPLIER 1_1	MULTIPLICATION 1_1	8	1	1
	MULTIPLICATION 2_1	11	1	2
	MULTIPLICATION 3_1	14	2	1,2
MULTIPLIER 1_2	MULTIPLICATION 1_2	8	1	1
	MULTIPLICATION 2_2	11	1	2
	MULTIPLICATION 3_2	14	2	1,2
COMPARATOR 1_1	COMPARATIVE ARITHMETIC PROCESSING 1_1	7	1	1,2
ADDER 1_1	ADDITION 1_1	9	1	1
	ADDITION 2_1	15	2	1,2
ADDER 1_2	ADDITION 1_2	9	1	1
	ADDITION 2_2	15	2	1,2
REGISTER 1_1	r1_1	13	1	1,2

Ex-1003, ¶255-256.

Ezaki's table of correspondence (“correspondence table”) allows “the relationship of correspondence to be uniquely determined, using the above table of correspondence, based on the states of the finite state machine of the RTL description and on the information contained in the behavioral description.” Ex-1007, ¶¶23-24, ¶56, ¶61; Ex-1003, ¶257, ¶264.

Shin also teaches a correspondence table, as explained above; that analysis applies here. *Supra* §VI.A.2[1d] (*Shin* subsection); Ex-1003, ¶258.

Each block in the behavioral description corresponds to a state in the RTL description. *Ezaki* teaches that “the RTL description and the behavioral description are in one-to-one correspondence with each other” in the correspondence table. Ex-1007, ¶50, ¶52, ¶62. *Gutberlet* suggests organizing a table around blocks of behavioral code because it correlates RTL simulation data back to behavioral blocks and teaches a “data structure” for mapping behavioral code. Ex-1005, 6:30-32; *supra* §VI.B.2[1c]. Therefore, organizing *Ezaki*’s correspondence table around behavioral blocks provides a mechanism for implementing *Gutberlet*’s teachings. *Supra* §VI.B.1; §VI.A.2[1d] (explaining obviousness of modifying a correspondence table). *Gutberlet* teaches the advantages of reporting information, including latency, for each block to allow a circuit designer to exercise “[c]omplete or enhanced control over how a design is synthesized to hardware.” Ex-1005, 3:7-13. The insight provided by block level information would facilitate the “precise control of a hardware implementation, allowing for interactive convergence on significantly better quality designs in much less time.” *Id.*; Ex-1003, ¶¶259-260.

Moreover, *Shin* teaches the mechanism suggested by *Gutberlet*: a table with columns for each block of behavioral code aligned in rows with corresponding states in the RTL code. Ex-1006, Fig. 7:

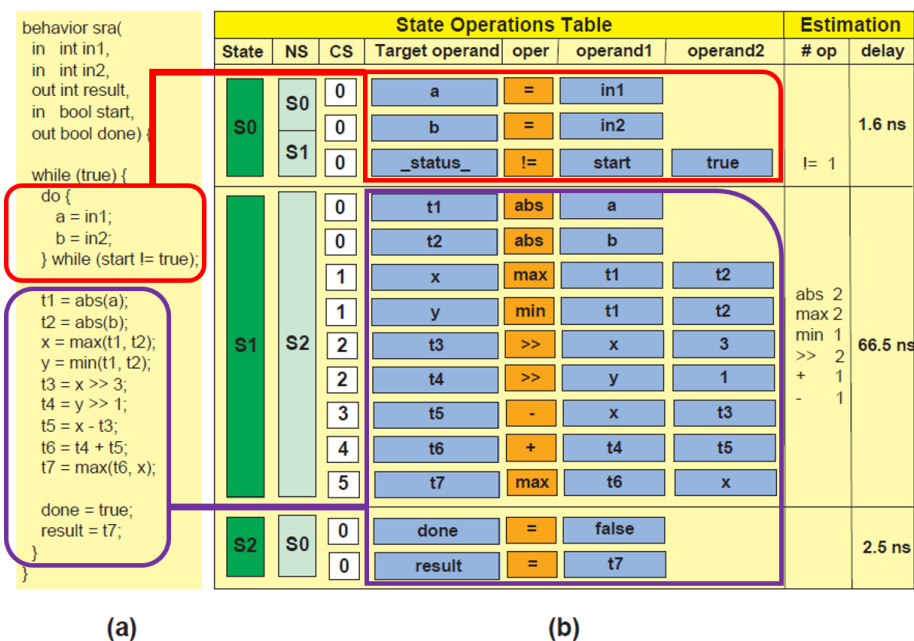


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

Ex-1003, ¶261.

A POSITA would have been motivated to apply *Shin*'s teachings to implement (and complement) *Gutberlet*'s logic simulation and latency analysis teachings. *Supra* §VI.B.1. *Shin* teaches [1d], as explained above; that analysis applies here. *Supra* §VI.A.2[1d] (*Shin* subsection); Ex-1003, ¶262.

[1e] a correspondence

Ground 3 teaches [1e]. *Supra* §VI.B.1 (explaining MTC); Ex-1003, ¶263.

For example, *Ezaki*'s teaches configuring components (e.g., “behavioral synthesis device” and “path reflecting device”) to generate the correspondence table together. Ex-1007, ¶¶23-24, ¶49, ¶61, Abstract:

The behavioral synthesis device then formulates a table of correspondence to show the relationship of correspondence between the RTL description and the CDFG and...between the CDFG and the behavioral description. A path determining device couples these tables of correspondence to ***generate a table of correspondence determining the relationship of correspondence between the RTL description and the behavioral description.***

Ex-1003, ¶264.

Ezaki provides examples of intermediate steps and resulting correspondence table. Ex-1007, ¶¶37-39, ¶¶59-60, ¶¶32-34, ¶¶44-48; Figs. 5-7, 10-12. The correspondence table can be generated by either referring to the nodes (operational units) or the input and outputs of nodes in the RTL description. *Id.*, ¶58, ¶48, ¶60. Here, a POSITA would have looked to *Ezaki*'s teachings regarding the “third table of correspondence” (Ex-1007, ¶56) for limitations [1d] and [1e] because it includes correspondence information between the behavioral description and the RTL description. Ex-1003, ¶265.

Gutberlet and *Shin* both teach, or suggest, using components of behavioral synthesis processes to generate a correspondence table. This was explained above and applies here. *Supra* §VI.A.2[1e] (*Gutberlet*, *Shin* subsections); Ex-1003, ¶266.

The respective teachings of *Ezaki*, *Gutberlet*, and *Shin* perform analogous functions with respect to the correspondence table: they describe devices that generate a correspondence table as part of a behavioral synthesis process.

Therefore, when combining their teachings, a POSITA would have been motivated to combine teachings from *Ezaki*, *Gutberlet*, and *Shin* for components that generate the correspondence table, particularly given the similar purpose and architecture of all three references. *Supra* §VI.B.1; Ex-1003, ¶267.

3. Claim 2

Ground 3 teaches a semiconductor design support device according to claim 1. *Supra* §VI.B.2.

Ground 3 teaches claim 2. *Supra* §VI.B.1 (explaining MTC); Ex-1003 ¶¶268-273. As explained above, *Gutberlet* teaches a latency analyzer configured to analyze (and therefore refer to) the results of RTL logic simulation and calculate latency in each behavioral block, e.g., as reported in Table 38. *Supra* §VI.B.2[1c], §VI.A.2[1c]; Ex-1003 ¶269.

To correlate RTL latency data back to behavioral blocks (as *Gutberlet* teaches), it was obvious to refer to a correspondence table, which *Ezaki* teaches for

this purpose. Ex-1007, Abstract, ¶1, ¶49, ¶23, ¶27; *supra* §VI.B.2.[1c]. For example, *Ezaki* identifies paths in the behavioral description based on RTL timing information, for the purpose of editing the behavioral description. Ex-1007, ¶¶8-11. *Ezaki* teaches correlating RTL timing information, for any paths of the RTL description, back to the behavioral description by referring to the correspondence table, which correlates RTL states with behavioral code. *Id.*, ¶47-49, ¶62 (“Hence, ***a path on the RTL description can be determined on the behavioral description using this method of employing the table of correspondence.***”), ¶56-57, Fig. 1 (steps S9-S10); Ex-1003 ¶270.

Shin facilitates the combination by teaching a table with behavioral blocks organized in rows with corresponding RTL states. *Supra* §VI.B.1, §VI.B.2[1c]-[1d] (explaining *Shin*’s table); Ex-1006, Fig. 7. When correlating RTL simulation data back to behavioral code (as *Gutberlet* teaches), it was obvious to refer to *Shin*’s table for its known purpose: to look up the block of behavioral code that correlates to an RTL state. Ex-1005, 7:4-6, 2:11-16, 34:3-20. Ex-1003, ¶271.

Gutberlet and *Shin* teach claim 2, as explained above; that analysis applies here. *Supra* §VI.A.3 (*Gutberlet*, *Shin* subsections); Ex-1003 ¶272.

The correspondence table is generated by the correspondence table generator. *Supra* §VI.B.2[1e]; Ex-1003 ¶273.

C. Grounds 4-6

1. MTC

Grounds 1-3 (*supra* §VI.A.1, §VI.B.1) teach IC design processes that, e.g., generate RTL designs of an IC. *Rittman* teaches known steps for manufacturing an IC. A POSITA would have been motivated to combine *Rittman*'s teachings with Grounds 1-3 because the next step after designing an IC is to manufacture it. Ex-1003 ¶275.

The prior art provides express motivation. *Gutberlet* teaches “[f]urther processing is then performed on the RTL code to ultimately generate the 15 integrated circuit.” Ex-1005, 6:13-15. *Ezaki* teaches “designing the hardware” and then manufacturing it. Ex-1007, ¶3. Ex-1003 ¶276. Grounds 1-3 teach steps, following behavioral synthesis, for converting RTL into a netlist, which was used to create layouts for photolithography masks in semiconductor manufacturing. Ex-1005, 1:24-26; Ex-1006, ¶23; Ex-1007, ¶8. Ex-1003 ¶277.

Rittman teaches the rest of the manufacturing process that Grounds 1-3 refer to. *Rittman* teaches generating a layout, photomask, and ICs on wafers. Ex-1008, ¶47, ¶¶49-58, Fig. 6. Combining *Rittman*'s teachings with Grounds 1-3 would have combined prior art elements according to known methods to yield predictable results (e.g., manufacturing the ICs designed in Grounds 1-3). ICs are designed to be manufactured. Ex-1003, ¶¶278-279.

The combinations with *Rittman* would not have changed the principle of operation for any reference because Grounds 1-3 focus on IC design while *Rittman* describes subsequent manufacturing steps. Grounds 4-6 simply take IC designs and carry out those subsequent steps. Ex-1003 ¶280.

A POSITA would have had a reasonable expectation of success because *Rittman* teaches conventional and common steps for manufacturing ICs, which had been used to manufacture ICs for decades. For example, Logic synthesis translated an RTL description into a gate-level circuit design. Ex-1020, 2:34-40. The netlist would then be converted into a physical layout. Ex-1008, ¶¶49-57, ¶4; Ex-1018, 307 (“Lower level tools such as logic synthesis and layout synthesis complete the design”); Ex-1020, 3:7-12. The layout would next be transferred to a photomask, that was used to process semiconductors with light and chemicals to etch the substrate and dope the materials that make up the circuit onto a semiconductor wafer. Ex-1008, ¶58, ¶4. Ex-1003 ¶¶19-20; ¶281.

2. Claim 3

3[pre]. A manufacturing method for manufacturing a semiconductor integrated circuit, the method comprising:
--

Grounds 4-6 each teach the preamble, including a method comprising limitations discussed below. *Supra* §VI.C.1 (explaining MTC); Ex-1003 ¶284.

For example, *Rittman* teaches a method for manufacturing a semiconductor IC including “photomasks” used for imaging (Ex-1008, ¶2, Title, ¶4, ¶29),

resulting in the “layers of *the integrated circuit have been fabricated on a semiconductor wafer*” (*id.*, ¶58; Ex-1003 ¶282).

Ezaki (Ground 6) further teaches using behavioral synthesis for “designing the hardware” that is manufactured into physical hardware. Ex-1007, ¶3; Ex-1003 ¶283.

[3a] describing an algorithm...to generate a behavioral description;
--

Limitation [3a] shares language with [1a] and is satisfied for the same reasons. *Supra* §VI.A.2[1a] (for Grounds 4-5), §VI.B.2[1a] (for Ground 6); Ex-1003 ¶286.

Grounds 4-6 teach the additional language of “generat[ing]” a behavioral description. *Supra* §VI.C.1 (explaining MTC). *Rittman* teaches that IC designs including “behavioral descriptions” are “generated[.]” Ex-1008, ¶47; Ex-1003 ¶287.

Grounds 1-3 teach behavioral descriptions. Ex-1004, ¶97; Ex-1005, 5:33-6:2; Ex-1006, ¶17; Ex-1007, ¶¶3-5. Therefore, Grounds 4-6 respectively teach or suggest generating (e.g., writing) one. *See id.*; Ex-1003 ¶288.

[3b] reading the behavioral description to generate an RTL description for recognizing...;

Limitation [3b] shares language with [1b] and is satisfied for the same reasons. *Supra* §VI.A.2[1b] (for Grounds 4-5), §VI.B.2[1b] (for Ground 6); Ex-1003 ¶¶289-290.

Grounds 4-6 each teach the additional language where the RTL description is generated “for” recognizing the recited concepts because the prior art teaches that RTL descriptions are “configured to” recognize those concepts and are for describing circuits at the register-transfer level based on hardware clock signals. *Supra* §VI.A.2[1b], §VI.B.2[1b]; *supra* §VI.C.1 (explaining MTC); Ex-1003 ¶¶291-293.

[3c] ...

Limitation [3c] appears in limitation [1c] and is satisfied for the same reasons. *Supra* §VI.A.2[1c] (for Grounds 4-5), §VI.B.2[1c] (for Ground 6); *supra* §VI.C.1; Ex-1003 ¶294.

[3d] designing a circuit based on the calculated latency;

Grounds 4-6 each teach [3d]. *Supra* §VI.C.1 (explaining MTC); Ex-1003 ¶295. Each ground teaches latency calculations for chip design. *Supra* §VI.A.2[1c] (for Grounds 4-5), §VI.B.2 (for Ground 6). *Gutberlet* teaches calculating latency for each block, presenting latency information to the designer, and using latency

calculations to guide the IC design. Ex-1005, Table 38, 34:3-15, 6:8-7:6; 2:11-3:4. *supra* §VI.B.2[1c]. *Rittman* additionally teaches designing a circuit. Ex-1008, Fig. 6; Ex-1003 ¶298.

For Grounds 4-5, *Harada* teaches that latency analysis of simulation results is used for debugging a circuit design, which is part of the process for designing an IC. Ex-1004, ¶11, ¶75, ¶77, ¶122; Ex-1003 ¶296.

For Ground 6, *Ezaki* teaches reducing the delay of the critical path in the design, and “correct[ing] the original behavioral description.” Ex-1007, ¶8. Ex-1003 ¶297.

It was obvious to use the calculated latency of Grounds 4-6 for designing a circuit because each reference applies its teachings to circuit design—that is the purpose of their latency teachings. *Supra* §VI.A.2[1c], §VI.B.2[1c]; Ex-1003 ¶299.

[3e] creating a layout of the circuit;
--

Grounds 4-6 teach [3e]. *Supra* §VI.C.1 (explaining MTC); Ex-1003 ¶300. *Gutberlet* and *Shin* teach synthesizing RTL into a netlist of components and interconnections. Ex-1005, 1:24-26; Ex-1006, ¶23; Ex-1003 ¶301. Ground 6, *Ezaki* teaches transforming RTL into a netlist. Ex-1007, ¶8; Ex-1003 ¶303.

Rittman teaches generating a netlist and then a “*layout* block 54 that represents the integrated circuit.” Ex-1008, ¶49, ¶58 (“mask layout file”), Fig. 6 (step 90); Ex-1003 ¶302.

[3f] generating a photomask based on the layout of the circuit;

Grounds 4-6 each teach [3f]. *Supra* §VI.C.1 (explaining MTC). *Rittman* teaches “creat[ing] a photomask for each layer of an integrated circuit” from the “mask layout file.” Ex-1008, ¶58, Fig. 6 (steps 92-110), ¶14, Fig. 1. The mask layout file “contains data regarding the geometry and interconnections of the integrated circuit represented in layout block 54.” *Id.*, ¶50; Ex-1003 ¶304-306.

[3g] transferring a pattern of the photomask onto a semiconductor wafer;

Grounds 4-6 each teach [3g]. *Supra* §VI.C.1 (explaining MTC). *Rittman* teaches using “a lithography system to image the pattern of the photomask on to a semiconductor wafer.” Ex-1008, ¶58, Fig. 6 (steps 112-114). Photolithography has been a standard IC manufacturing method for decades. Ex-1003 ¶307.

[3h] ...

Limitation [3h] appears in [1e] and [1d] and is satisfied for the same reasons. *Supra* §VI.A.2[1d]-[1e] (Grounds 4-5), §VI.B.2[1d]-[1e] (Ground 6); Ex-1003 ¶¶308-310; *supra* §VI.C.1 (explaining MTC).

3. Claim 4

Grounds 4-6 each teach a manufacturing method according to claim 3. *Supra* §VI.C.2[pre]-[3h]. Ex-1003 ¶311. Beyond the preamble, the remainder of claim 4 is identical to claim 2 and satisfied for the same reasons. *Supra* §VI.A.3 (Grounds 4-5), §VI.B.3 (Ground 6), §VI.C.1 (explaining MTC); Ex-1003 ¶312.

VII. MANDATORY NOTICES AND FEES

Real Party In Interest: The real party-in-interest here is Cadence Design Systems, Inc. No other parties directed, controlled, or funded this Inter Partes Review proceeding (IPR).

Related Matters: NDCA 3-23-cv-01001

Lead and Backup Counsel: Pursuant to 37 C.F.R. §42.8(b)(3) and 42.10(a), Petitioner designates Harper Batts (Reg. No. 56,160) as lead counsel and Chris Ponder (Reg. No. 77,167) and Jeffrey Liang (Reg. No. 69,043) as back-up counsel, each of Sheppard, Mullin, Richter & Hampton LLP.

Service Information: Petitioner consents to service by electronic mail addressed to:

HBatts@sheppardmullin.com;
CPonder@sheppardmullin.com; and
JLiang@sheppardmullin.com

Petitioner's counsel may also be served by mail or hand delivery at Sheppard, Mullin, Richter & Hampton LLP, 1540 El Camino Real Suite 120, Menlo Park, CA 94025. Petitioner's counsel may be reached by telephone at (650) 815-2673 and by facsimile at (650) 815-4668.

Fees: The Office is authorized to charge fees for this Petition to Deposit Account 50-4561, Ref. 041Y-372191.

Respectfully submitted,

SHEPPARD, MULLIN,
RICHTER & HAMPTON LLP

/Harper Batts/

Harper Batts (Reg. 56,160)

Attorneys for Petitioner

CLAIM APPENDIX

1[pre]. A semiconductor design support device for designing a semiconductor integrated circuit, the device comprising:

[1a] a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level;

[1b] an RTL description generated by reading the behavioral description, the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware;

[1c] a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description;

[1d] a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description; and

[1e] a correspondence table generator configured to generate the correspondence table.

2. The semiconductor design support device according to claim 1, wherein the latency analyzer refers to the result of the logic simulation and the correspondence table generated by the correspondence table generator to calculate the latency in each block in the behavioral description.

3[pre]. A manufacturing method for manufacturing a semiconductor integrated circuit, the method comprising:

[3a] describing an algorithm of processing performed by hardware in a motion level to generate a behavioral description;

[3b] reading the behavioral description to generate an RTL description for recognizing a concept including register and clock synchronism particular to the hardware;

[3c] analyzing a result of a logic simulation performed on the RTL description to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description;

[3d] designing a circuit based on the calculated latency;

[3e] creating a layout of the circuit;

[3f] generating a photomask based on the layout of the circuit;

[3g] transferring a pattern of the photomask onto a semiconductor wafer;
and

[3h] generating a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description.

4. The manufacturing method according to claim 3, further comprising:

referring to the result of the logic simulation and the correspondence table to calculate the latency in each block in the behavioral description.

CERTIFICATE OF SERVICE

The undersigned hereby certifies that on October 20, 2023 true and correct copies of the foregoing Petition for *Inter Partes* Review of U.S. Patent No. 7,971,167 at the following addresses of record as listed on PAIR, via Priority Mail Express®:

Szymon Gurda
903 Commerce Drive
Suite 310
Oak Brook, IL 60523
Official Correspondence Address

Semiconductor Design Technologies
LLC
1000 N. West Street
Suite 1200
Wilmington, DE 19801
Patent Owner Address

The undersigned certifies that true and correct copies of the Petition for *Inter Partes* Review of U.S. Patent No. 7,971,167 and Exhibits 1001-1028 were also sent via electronic mail to the attorneys of record for Plaintiff in the concurrent litigation matter(s):

David L. Alberti
dalberti@krameralberti.com
Hong Lin
hlin@krameralberti.com
Robert Y. Xie
rxie@krameralberti.com
Russell Steven Tonkovich
rtonkovich@krameralberti.com

Sal Lim
slim@krameralberti.com
James P. Barabas
jbarabas@krameralberti.com
Robert C Mattson
rmattson@krameralberti.com
Robert F. Kramer
rkramer@krameralberti.com

SHEPPARD, MULLIN,
RICHTER & HAMPTON LLP

/Harper Batts/

Harper Batts (Reg. No. 56,160)
Attorneys for Petitioner

Date: October 20, 2023

1540 El Camino Real Suite 120
Menlo Park, CA 94025
Tel: (650) 815-2673

**CERTIFICATE OF COMPLIANCE WITH
TYPE-VOLUME LIMITATION, TYPEFACE REQUIREMENTS,
AND TYPE STYLE REQUIREMENTS**

1. This Petition complies with the type-volume limitation of 14,000 words, comprising 13,497 words, as counted using the Microsoft Word software that was used to prepare this paper, excluding the parts exempted by 37 C.F.R. § 42.24(a).

2. This Petition complies with the general format requirements of 37 C.F.R. § 42.6(a) and has been prepared using Microsoft® Word in 14-point Times New Roman.

Respectfully submitted,

SHEPPARD, MULLIN,
RICHTER & HAMPTON LLP

/Harper Batts/
Harper Batts (Reg. No. 56,160)
Attorney for Petitioner

Date: October 20, 2023

1540 El Camino Real Suite 120
Menlo Park, CA 94025
Tel: (650) 815-2673