

U.S. Patent No. 11,184,628

TCL / Realtek Product

U.S. Patent No. 11,184,628: Claim 7(pre)
“A computer processing system comprising:”

7(pre). A computer processing system comprising:

The TCL 50S535 television, graphics systems incorporated in the same, and Realtek components thereof (hereinafter, the “TCL / Realtek Product”), include a computer processing system.



See <https://www.tcl.com/us/en/products/home-theater/5-series/50-class-5-series-4k-qled-hdr-smart-roku-tv-50s535>.

For example, the TCL / Realtek Product includes a printed circuit board assembly comprising a Realtek RTD2873 system on chip (“SoC”) (hereinafter, the “Realtek SoC”).



See TCL 50S535 Television Product Photo (Chip).

The Realtek SoC is a “Smart TV SoC (System on a Chip).”

Realtek Ex. 1019

4K Android Smart TV SoC with Advanced Picture/Audio
Quality (RTD2851/RTD2873)

The RTD2851/RTD2873 Smart TV SoC (System on a Chip) supports Android Pie and Google Assistant TV voice control. It also supports all HDR formats, including Dolby Vision™, HDR10, Technicolor Prime, HLG, and HDR10+. It provides theater-like sound effects using the built-in TV speakers through decoding Dolby ATMOS™, and offers advanced picture and video processing for internet streaming contents.

See <https://www.realtek.com/en/press-room/news-releases/item/realtek-to-demonstrate-full-range-of-connectivity-multimedia-and-consumer-electronics-solutions-at-2019-ces>.

“a processor and a memory, wherein the processor is configured to:”

7(a). a processor and a memory, wherein the processor is configured to:

The TCL / Realtek Product includes a processor and a memory.

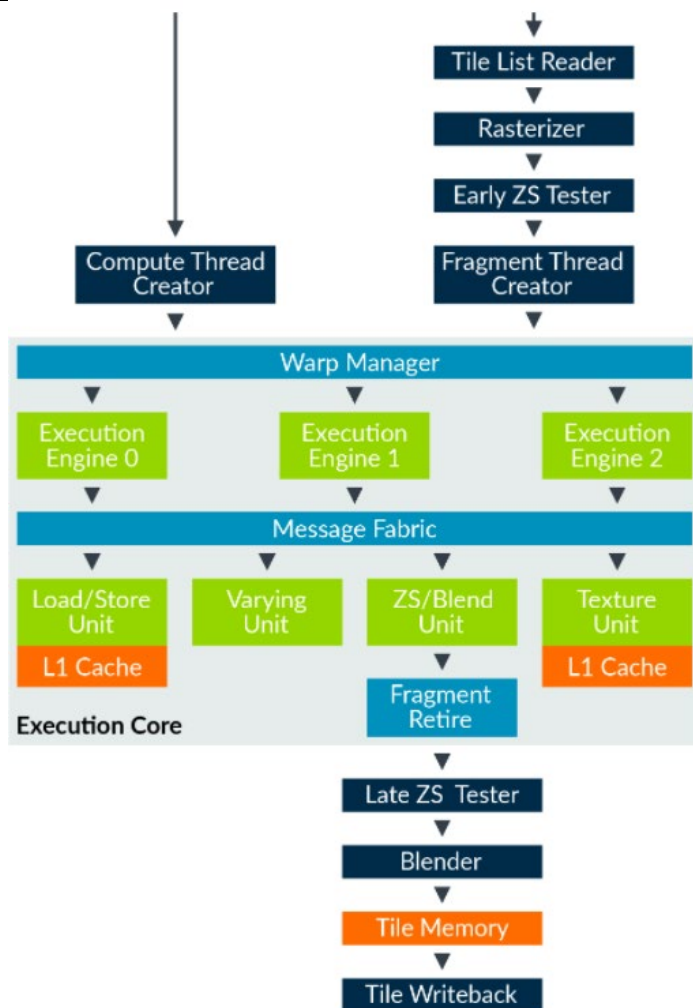
For example, the Realtek SoC contains four ARM Cortex-A55 Processor cores and an ARM Mali-G31 GPU (hereinafter, the “Mali GPU”).

Jeu de puces	CPU	GPU
Realtek RTD2873 RTD2873 RTD2873	4x ARM Cortex-A55 (1100 Mhz)	ARM Mali G31 (700 Mhz)

See https://twitter.com/AndroidTV_Rumor/status/1091304992907161602/photo/2.

Further, the TCL / Realtek Product includes at least L1 caches, L2 caches, tile memory, local storage, and system memory.

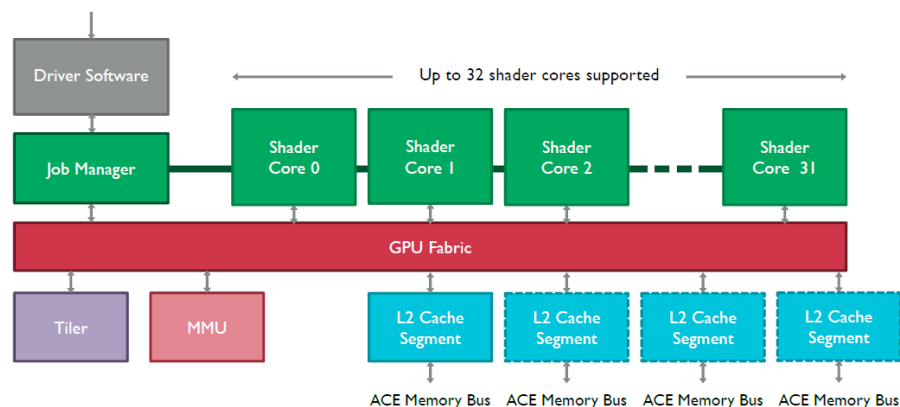
“a processor and a memory, wherein the processor is configured to:”



See The Bifrost Shader Core, <https://developer.arm.com/solutions/graphics-and-gaming/developer-guides/learn-the-basics/the-bifrost-shader-core/the-bifrost-shader-core>.

“a processor and a memory, wherein the processor is configured to:”

Bifrost GPU design



See <https://community.arm.com/cfs-file/key/telligent-evolution-components-attachments/01-2066-00-00-00-00-71-76/HotChips-Presentation-Jem-Davies.pdf>, slide 7.

ZS and blend unit

The ZS and color blend units are both responsible for handling all OpenGL ES and programmatic accesses to the tile-buffer for functionality like:

- [EXT_shader_pixel_local_storage]
- [ARM_shader_framebuffer_fetch]
- [ARM_shader_framebuffer_fetch_depth_stencil]

See The Bifrost Shader Core, <https://developer.arm.com/solutions/graphics-and-gaming/developer-guides/learn-the-basics/the-bifrost-shader-core/the-bifrost-shader-core>.

This extension provides a way for applications to pass information between fragment shader invocations covering the same pixel by introducing the concept of pixel local storage. Pixel local storage is an on-chip memory storage that can be efficiently accessed by fragments being processed by the GL. The format of data stored in the pixel local storage is independent of the format of the currently attached framebuffer. The data in pixel local storage is not written back to main memory. Access to pixel local storage is controlled via glEnable and glDisable. If commands that implicitly or explicitly flush the GL command stream are issued when pixel local storage is enabled then the contents of the pixel local storage becomes undefined for subsequent commands.

See https://www.khronos.org/registry/OpenGL/extensions/EXT/EXT_shader_pixel_local_storage.txt, EXT_shader_pixel_local_storage,

Realtek Ex. 1019

“a processor and a memory, wherein the processor is configured to:”



See TCL 50S535 Television Product Photo (Chip and Memory).

Motherboard	merlin4
Memory	2.00 GB
Processor Information	
Name	ARM rtd287o
Topology	1 Processor, 4 Cores
Identifier	ARM implementer 65 architecture 7 variant 1 part 3333 revision 0
Base Frequency	1.60 GHz

See <https://browser.geekbench.com/v4/cpu/15689207>.

KIOXIA

TC58NVG2S0HBAI4

MOS DIGITAL INTEGRATED CIRCUIT SILICON GATE CMOS

4 GBIT (512M × 8 BIT) CMOS NAND E²PROM

DESCRIPTION

The TC58NVG2S0HBAI4 is a single 3.3V 4Gbit (4,563,402,752 bits) NAND Electrically Erasable and Programmable Read-Only Memory (NAND E²PROM) organized as (4096 + 256) bytes × 64 pages × 2048 blocks. The device has two 4352-byte static registers which allow program and read data to be transferred between the register and the memory cell array in 4352-byte increments. The Erase operation is implemented in a single block unit (256 Kbytes + 16 Kbytes: 4352 bytes × 64 pages).

The TC58NVG2S0HBAI4 is a serial-type memory device which utilizes the I/O pins for both address and data input/output as well as for command inputs. The Erase and Program operations are automatically executed, making the device most suitable for applications such as solid-state file storage, voice recording, image file memory for still cameras and other systems which require high-density non-volatile memory data storage.

See TC58NVG2S0HBAI4 Datasheet, <https://www.digikey.com/en/products/detail/kioxia-america-inc/TC58NVG2S0HBAI4/5226322>.

“receive a compressed texture block, wherein the compressed texture block includes two or more disjoint subsets into which data in the compressed texture block is to be unpacked, wherein the two or more disjoint subsets include a first subset having a first set of color endpoints and a second subset having a second set of color endpoints; and”

7(b). receive a compressed texture block, wherein the compressed texture block includes two or more disjoint subsets into which data in the compressed texture block is to be unpacked, wherein the two or more disjoint subsets include a first subset having a first set of color endpoints and a second subset having a second set of color endpoints; and

The TCL / Realtek Product includes a processor configured to receive a compressed texture block, wherein the compressed texture block includes two or more disjoint subsets into which data in the compressed texture block is to be unpacked, wherein the two or more disjoint subsets include a first subset having a first set of color endpoints and a second subset having a second set of color endpoints.

The TCL / Realtek Product includes a processor configured to receive a compressed texture block. For example, the Mali GPU features at least “Adaptive Scalable Texture Compression (ASTC).”



Arm Mali-G31 GPU

Features	Value	Description
Anti-Aliasing	4x MSAA 8x MSAA 16x MSAA	4x Multi-Sampling Anti-Aliasing (MSAA) with minimal performance drop.
API Support	OpenGL® ES 1.1, 2.0, 3.2 Vulkan 1.2 OpenCL™ 1.1, 1.2, 2.0 Full Profile, RenderScript	Full support for next-generation and legacy 2D/3D graphics applications.
Bus Interface	AMBA® 4 ACE-LITE and AXI	Compatible with a wide range of bus interconnect and peripheral IP.
L2 Cache	Configurable	8KB for MP1. 32KB-64KB for MP2.
Memory System	Virtual Memory	Built-in Memory Management Unit (MMU) to support virtual memory.
Scalability	1 uni-pixel core or dual-pixel core	Optimized for high area and energy efficiency to address mainstream device requirements.
Adaptive Scalable Texture Compression (ASTC)	Low dynamic range (LDR) and high dynamic range (HDR). Supports both 2D and 3D images	ASTC offers a number of advantages over existing texture compression schemes by improving image quality, reducing memory bandwidth and thus energy use.
Arm Frame Buffer Compression (AFBC)	Version 1.2 4x4 pixel block size	AFBC is a lossless image compression format that provides random access to pixel data to a 4x4 pixel block granularity. It is employed to reduce memory bandwidth both internally within the GPU and externally throughout the SoC.
Transaction Elimination	16x16 pixel block size	Transaction Elimination spots the identical pixel blocks between two consecutive render targets and performs a partial update to the frame buffer with the changed pixel blocks only, which reduces memory bandwidth and thus energy.

See <https://developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus/mali-g31-gpu>.

The Mali GPU and/or related software produces “high-quality compressed textures at low bit rates in many different image configurations.”

“receive a compressed texture block, wherein the compressed texture block includes two or more disjoint subsets into which data in the compressed texture block is to be unpacked, wherein the two or more disjoint subsets include a first subset having a first set of color endpoints and a second subset having a second set of color endpoints; and”

Adaptive Scalable Texture Compression

Adaptive Scalable Texture Compression (ASTC) produces high-quality compressed textures at low bit rates in many different image configurations.

This section contains the following subsections:

- [3.2.1 About textures on page 3-35.](#)
- [3.2.2 About ASTC on page 3-35.](#)

See <https://developer.arm.com/documentation/100959/0101/>, pg. 3-35.

Block compression

Compression formats for real-time graphics need the ability to quickly and efficiently make random samples into a texture. This places two technical requirements on any compression format. It must be possible to do the following:

- Compute the address of data in memory given only a sample coordinate.
- Decompress random samples without decompressing too much surrounding data.

See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

The Mali GPU receives compressed texture blocks, wherein during the compression process, the texture is “divide[d] into fixed-size blocks of texels, each of which is compressed into a fixed number of output bits.” This “makes it possible to access texels quickly, in any order, and with a well-bounded decompression cost.”

The standard solution for this used by all contemporary real-time formats, including ASTC, is to divide the image into fixed-size blocks of texels, each of which is compressed into a fixed number of output bits. This feature makes it possible to access texels quickly, in any order, and with a well-bounded decompression cost.

See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

The compressed texture block includes two or more disjoint subsets. For example, the Mali GPU and/or related software “allows a block to define up to four distinct color gradients, called partitions, and can assign each texel to a single partition.”

Blocks often contain a complex distribution of colors, for example a red ball sitting on green grass. In these scenarios, a single-color gradient cannot accurately represent all different texel color values. ASTC allows a block to define up to four distinct color gradients, called partitions, and can assign each texel to a single partition. For our example, we require two partitions, one for the red ball texels and one for the green grass texels. The following diagram shows how the partition index specifies which color gradient to use for each texel:

See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

“receive a compressed texture block, wherein the compressed texture block includes two or more disjoint subsets into which data in the compressed texture block is to be unpacked, wherein the two or more disjoint subsets include a first subset having a first set of color endpoints and a second subset having a second set of color endpoints; and”

The two or more disjoint subsets include a first subset having a first set of color endpoints and a second subset having a second set of color endpoints. Furthermore, the compressed texture block includes two or more disjoint subsets into which data in the compressed texture block is to be unpacked. For example, each color gradient comprises a first and second endpoint (“Endpoint 1” and “Endpoint 2”). Furthermore, the “color data for a block is encoded as a gradient between two color endpoints.” “Each texel selects a position along that gradient, which is then interpolated during decompression.”

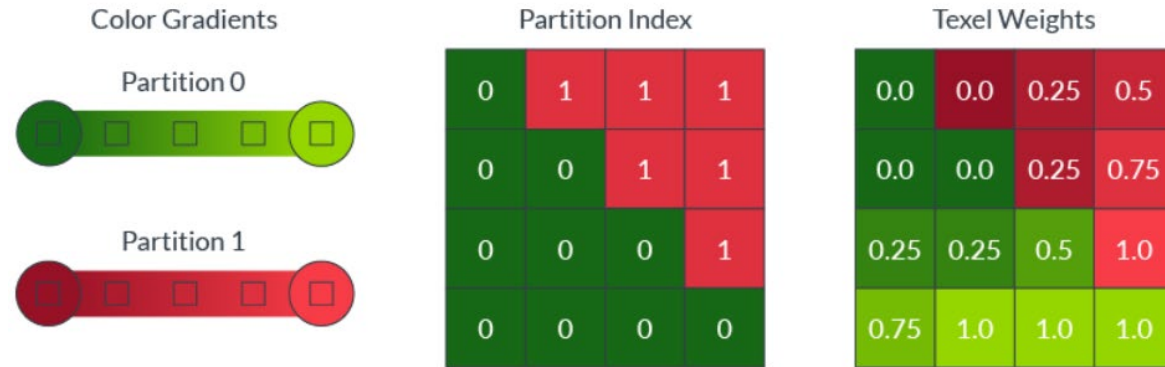
ASTC uses gradients to assign the color values of each texel. Each compressed block stores the end-point colors for a gradient, and an interpolation weight for each texel. During decompression, the color value for each texel is generated by interpolating between the two end-point colors, based on the per-texel weight. The following diagram shows this interpolation for a variety of texel weights:

See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.



See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

“receive a compressed texture block, wherein the compressed texture block includes two or more disjoint subsets into which data in the compressed texture block is to be unpacked, wherein the two or more disjoint subsets include a first subset having a first set of color endpoints and a second subset having a second set of color endpoints; and”



See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

Color endpoints

The color data for a block is encoded as a gradient between two color endpoints. Each texel selects a position along that gradient, which is then interpolated during decompression. ASTC supports 16 color endpoint encoding schemes, known as endpoint modes.

See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

Furthermore, during the compression process, the compression software “algorithmically generates a range of patterns, using the partition index as a seed value.”

Directly storing the partition assignment for each texel would need a lot of decompressor hardware to store it for all block sizes. Instead, ASTC algorithmically generates a range of patterns, using the partition index as a seed value. The compression process selects the best pattern match for each block. The block then only needs to store the index of the best matching pattern. The following image shows the generated patterns for two (top section of image), three (middle section of image), and four (bottom section of image) partitions for the 8x8 block size:

See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

“individually decompress each of the two or more disjoint subsets to form texels, wherein the decompressing includes:”

7(c). individually decompress each of the two or more disjoint subsets to form texels, wherein the decompressing includes:	The TCL / Realtek Product individually decompresses each of the two or more disjoint subsets to form texels. For example, “[d]uring decompression, the color value for each texel is generated by interpolating between the two end-point colors, based on the per-texel weight.” ASTC uses gradients to assign the color values of each texel. Each compressed block stores the end-point colors for a gradient, and an interpolation weight for each texel. During decompression, the color value for each texel is generated by interpolating between the two end-point colors, based on the per-texel weight. The following diagram shows this interpolation for a variety of texel weights: See https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm.
---	---

“determining a first texel color for a first texel that is part of the first disjoint subset by interpolating between color endpoints of the first set of color endpoints based on a first index value for the first texel; and determining a second texel color for a second texel that is part of the second disjoint subset by interpolating between color endpoints of the second set of color endpoints based on a second index value for the second texel.”

7(d). determining a first texel color for a first texel that is part of the first disjoint subset by interpolating between color endpoints of the first set of color endpoints based on a first index value for the first texel; and
determining a second texel color for a second texel that is part of the second disjoint subset by interpolating between color endpoints of the second set of color endpoints based on a second index value for the second texel.

The decompressing includes determining a first texel color for a first texel that is part of the first disjoint subset by interpolating between color endpoints of the first set of color endpoints based on a first index value for the first texel; and determining a second texel color for a second texel that is part of the second disjoint subset by interpolating between color endpoints of the second set of color endpoints based on a second index value for the second texel.

For example, “[d]uring decompression, the color value for each texel is generated by interpolating between the two end-point colors, based on the per-texel weight.”

ASTC uses gradients to assign the color values of each texel. Each compressed block stores the end-point colors for a gradient, and an interpolation weight for each texel. During decompression, the color value for each texel is generated by interpolating between the two end-point colors, based on the per-texel weight. The following diagram shows this interpolation for a variety of texel weights:



See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

Color endpoints

The color data for a block is encoded as a gradient between two color endpoints. Each texel selects a position along that gradient, which is then interpolated during decompression. ASTC supports 16 color endpoint encoding schemes, known as endpoint modes.

See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

“determining a first texel color for a first texel that is part of the first disjoint subset by interpolating between color endpoints of the first set of color endpoints based on a first index value for the first texel; and determining a second texel color for a second texel that is part of the second disjoint subset by interpolating between color endpoints of the second set of color endpoints based on a second index value for the second texel.”

In addition, “[t]he number of partitions and the partition index can be chosen on a per-block basis, and a different color endpoint mode can be chosen per partition.”

The number of partitions and the partition index can be chosen on a per-block basis, and a different color endpoint mode can be chosen per partition.

See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

“Each texel requires a weight, which defines the relative contribution of each color endpoint.” To decode one texel, the processor “interpolate[s] color endpoints using weight” from a “stored weight grid during decompression using bilinear interpolation.”

Texel weights

Each texel requires a weight, which defines the relative contribution of each color endpoint when interpolating the color gradient.

For smaller block sizes, we can choose to store the weight directly, with one weight per texel. However, for the larger block sizes there are not enough bits of storage to do this. To work around this issue, ASTC allows the weight grid to be stored at a lower resolution than the texel grid. The per-texel weights are interpolated from the stored weight grid during decompression using a bilinear interpolation.

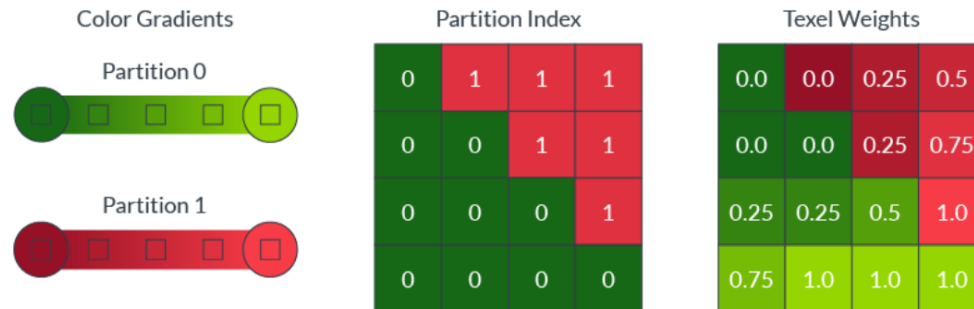
Both the number of texel weights, and the weight value BISE quantization level, can be chosen on a per-block basis.

See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

“The following diagram shows how the partition index specifies which color gradient to use for each texel.”

“determining a first texel color for a first texel that is part of the first disjoint subset by interpolating between color endpoints of the first set of color endpoints based on a first index value for the first texel; and determining a second texel color for a second texel that is part of the second disjoint subset by interpolating between color endpoints of the second set of color endpoints based on a second index value for the second texel.”

Blocks often contain a complex distribution of colors, for example a red ball sitting on green grass. In these scenarios, a single-color gradient cannot accurately represent all different texel color values. ASTC allows a block to define up to four distinct color gradients, called partitions, and can assign each texel to a single partition. For our example, we require two partitions, one for the red ball texels and one for the green grass texels. The following diagram shows how the partition index specifies which color gradient to use for each texel:



See <https://developer.arm.com/documentation/102162/0002/The-ASTC-algorithm>.

Furthermore, “which gradient a specific texel is taken from comes from a second layer, an index pattern which tells it per texel which gradient to pick from.”

That probably sounds like it might lead to blurry looking images with blocky colour leaking, but that’s only half the story. Because the block can have up to 4 different colour gradients, each with a start and end point. There’s a single set of samples for where in the gradient the texel lies, but which gradient a specific texel is taken from comes from a second layer, an index pattern which tells it per texel which gradient to pick from. The great thing is, this set of indices doesn’t have any interpolation so it gives nice crisp edges on contrasting colours between the soft parts of the gradients.

See <https://community.arm.com/arm-community-blogs/b/graphics-gaming-and-vr-blog/posts/astc-does-it>.