Graphics Chips

(Component of Graphics Hardware)

Winter Barrett

First Edition, 2012

ISBN 978-81-323-3058-5

Published by:
Research World
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter 1

Blitter

In a computer system, a **blitter** is a circuit, sometimes as a coprocessor or a logic block on a microprocessor, that is dedicated to the rapid movement and modification of data within that computer's memory. A blitter is capable of copying large quantities of data from one memory area to another relatively quickly, and in parallel with the CPU.

The name comes from the acronym BLIT, which stands for **BL**ock **I**mage **T**ransfer. A typical use for a blitter is the movement of a large bitmap in a 2D computer game or demo.

The historical need for a blitter

In early computers with raster-graphics output, the screen buffer was normally held in main memory and drawn under the control of the CPU. For many simple graphics routines, like sprite support or flood filling polygons, large amounts of memory had to be manipulated. A typical reason to move large data areas arose when drawing bitmaps, such as when drawing the next frame during a computer game or demo. An image that moves smoothly across the screen might give rise to the need for a bitmap (representing the image) to be moved every frame. Game designers had to design their game's graphics so that the total amount of bitmap data transfer required to draw each frame was within the capacity of the CPU.

As graphics hardware became more sophisticated, framebuffers grew larger with higher resolutions and colour depth. Games designers wanted to use larger images to represent game elements, and to maintain acceptable framerates. Graphics operations then required more and more data transfer speed to accomplish these bitmap moves. This work tied down the CPU, preventing it from operating on other tasks or making it infeasible to transfer large images within the finite time allowed for processing between frames.

Blitters in home computing

Computer manufacturers introduced blitters to help lessen this graphics burden on the CPU, or to allow more complex graphics. Several home computers manufactured in the 1980s included a graphics coprocessor that contained a blitter. The CPU would send a description of the necessary bit blit operations to the blitter, which would then carry out the operation much faster than the CPU could, and in parallel.

The Commodore Amiga was the first personal computer to use a full-featured blitter, and the first US patent filing to use the term *blitter* was "Personal computer apparatus for block transfer of bit-mapped image data," assigned to Commodore-Amiga, Inc. On top of the ability to copy and manipulate large areas of graphics, the hardware that contained the Amiga's blitter also included line drawing and area-filling hardware.

Later models of the Atari ST also included a blitter co-processor, which was named in all capitals as the *BLITTER* chip. One story states that manufacturing delays deferred its introduction into the ST line until after the first STs had shipped. Another is that the Atari ST's main competitor, the Amiga, was famous for its blitter, and so Atari introduced one as well. Although Atari planned an upgrade to allow dealers to install the blitter chip, this plan was later dropped. Instead, the BLITTER was introduced on the Mega series, and then also supported on most later machines (except the Atari TT).

Graphics-oriented software (especially games) running on systems that did not have a blitter needed to find other methods of transferring large bitmaps. Some games were written across platforms, some of which contained a blitter and some of which didn't. A typical example would be a game written for the Atari ST and Amiga. Both machines contained similar hardware, including the MC68000 processor, but the Amiga contained a blitter and the early Ataris did not. Such a game could not rely on the presence of a blitter. One approach to handling this was to load all available 68000 data registers with data from the bitmap in memory, and then push the data into the frame buffer in as few operations as possible.

Blitting was not the only solution to providing high-performance graphics in performance-limited machines. A more common solution in early machines was the use of sprites, which used two different graphics pathways to draw images that were then combined in the video display circuitry into a single image. Sprites were small bitmaps that were positioned on the screen independent of the normal bitmap background, allowing them to be moved on-screen by adjusting the values of several timers. The video circuitry started drawing the sprites after the timer had expired, allowing them to be drawn for little cost, and avoiding the need to move memory around to provide the illusion of motion. The downside of this approach is that the sprite systems generally had hard-coded limits to the number of sprites they could display, often between two (the Atari VCS) and eight (Commodore 64). Blitters offered the ability to have any number of objects, limited only by the performance of the blitter and the memory it talked to. As the performance of these circuits increased, the flexibility of the blitter overwhelmed any

performance advantage in the sprite approach, and many sprite systems became blitters in disguise.

Operation

Typically, a computer program would put information into certain registers describing what memory transfer needed to be completed and the logical operations to perform on the data, then trigger the blitter to begin operating. The CPU is then free to begin some other operation while the blitter operates.

The destination for the transfer is usually the frame buffer. However, a blitter can also be used for non-graphics work. For example, an area of memory might be zeroed (filled with zeroes) using a blitter more quickly than can be accomplished with the CPU. Additionally, simple mathematical operations can be built from basic logical operations.

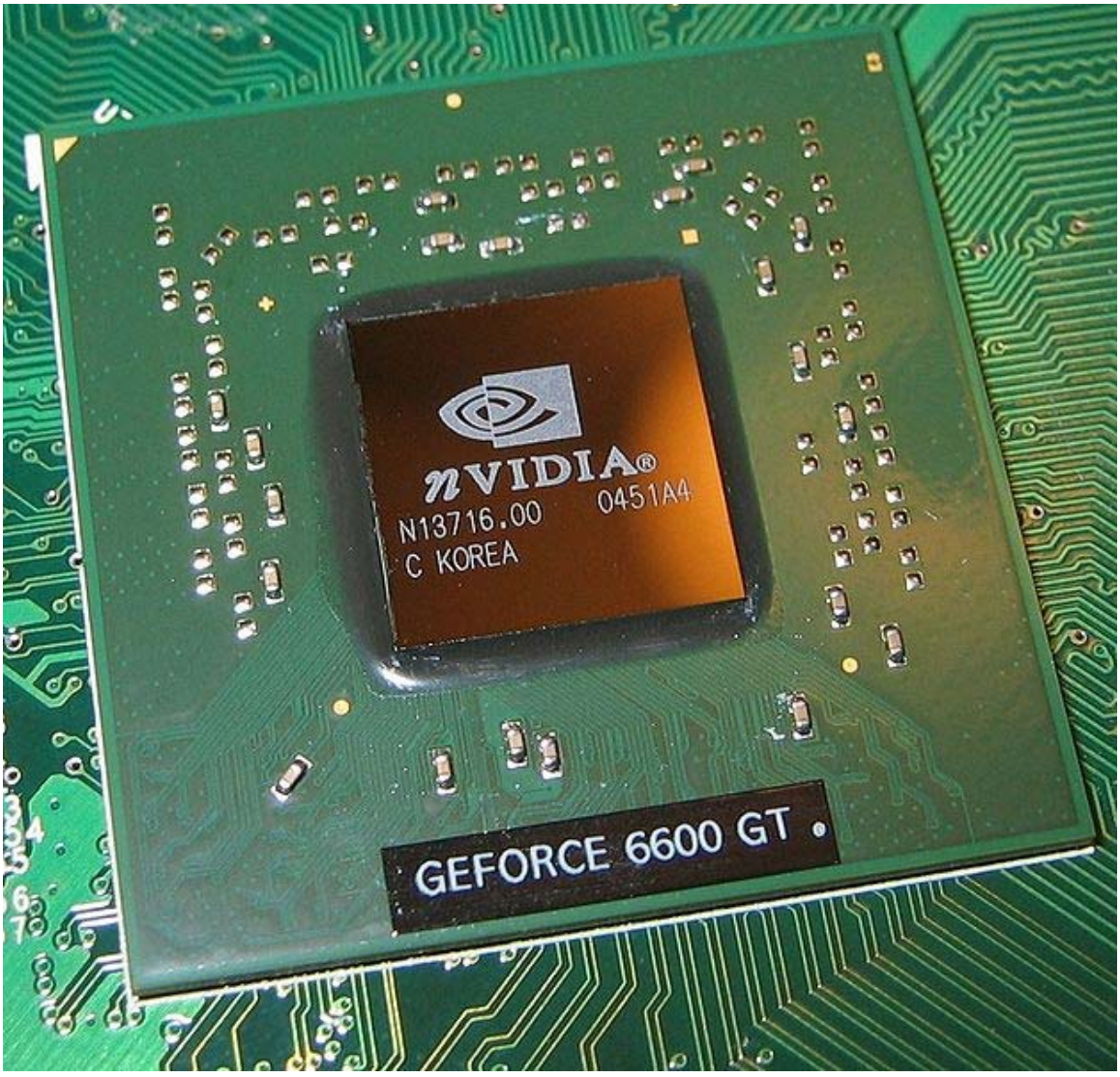
The image at right helps illustrate how a blitter may use a 'mask' to decide which pixels to transfer and which to leave untouched. The mask operates like a stencil, showing which pixels in the source image will be written to destination memory. The logical operation would be Dest = (Background) AND (Mask) OR (Sprite).

Current technology

Blitters have evolved into the modern graphics processing unit. The modern GPU is essentially a very advanced blitter and shares with earlier blitters the goal of rapidly copying, transforming, and writing transformed bitmaps (more generally textures), to a framebuffer. But, while early blitters were limited to performing simple logical operations on the target data, modern GPUs add the ability to modify these bitmaps in mathematically advanced ways that help produce more interesting effects, such as shading to produce illumination effects and blending for transparency. Additionally, modern GPUs can write bitmaps to destination memory in such a way so as to provide the illusion of depth. While earlier blitters were limited to a linear one-to-one mapping of source pixels to destination pixels, modern GPUs can instead map source pixels to destination pixels in a non-linear way to produce the appearance of perspective. Support for more advanced mathematical transformations also allows the modern GPU to transform blocks of coordinates in a 3d space. This ability, combined with shading and non-linear mapping between source and destination pixels helps to create the fluid 3d experience found in many of today's games.

Chapter 2

Graphics Processing Unit



GeForce 6600GT (NV43) GPU

A **graphics processing unit** or **GPU** (also occasionally called **visual processing unit** or **VPU**) is a specialized circuit designed to rapidly manipulate and alter memory in such a way so as to accelerate the building of images in a frame buffer intended for output to a display. This specialization allows a GPU to perform such tasks more rapidly than a less specialized and more general central (micro-)processor. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. In a personal computer, a GPU can be present on a video card, or it can be on the motherboard, or in certain CPUs, on the CPU die. More than 90% of new desktop and notebook computers have integrated GPUs, which are usually far less powerful than those on a dedicated video card.

The term was defined and popularized by Nvidia in 1999, who marketed the GeForce 256 as "the world's first 'GPU', or Graphics Processing Unit, a single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second." Rival ATI Technologies coined the term visual processing unit or VPU with the release of the Radeon 9700 in 2002.

History

Graphics accelerators

A GPU (Graphics Processing Unit) is a graphics coprocessor that offloads graphics tasks from the CPU with the intention of performing them faster than the CPU can perform them. A graphics accelerator incorporates one or more custom microchips which contain special mathematical operations commonly used in graphics rendering. The efficiency and clock speed of the microchips and the algorithms implemented therefore determines the effectiveness of the graphics accelerator. They were once mainly used for playing 3D games or high-end 3D rendering, but now take on more roles and therefore have grown more popular as a result. A GPU implements a number of graphics primitive operations in a way that makes running them much faster than drawing directly to the screen with the host CPU. The most common operations for early 2D computer graphics include the BitBLT operation, combining several bitmap patterns using a raster op, usually in special hardware called a *"blitter"*, and operations for drawing rectangles, triangles, circles, and arcs. Modern GPUs also have support for 3D computer graphics, and typically include digital video–related functions. More recent GPUs now support handling highly parallel tasks that CPUs would take a long time to perform.

1980s

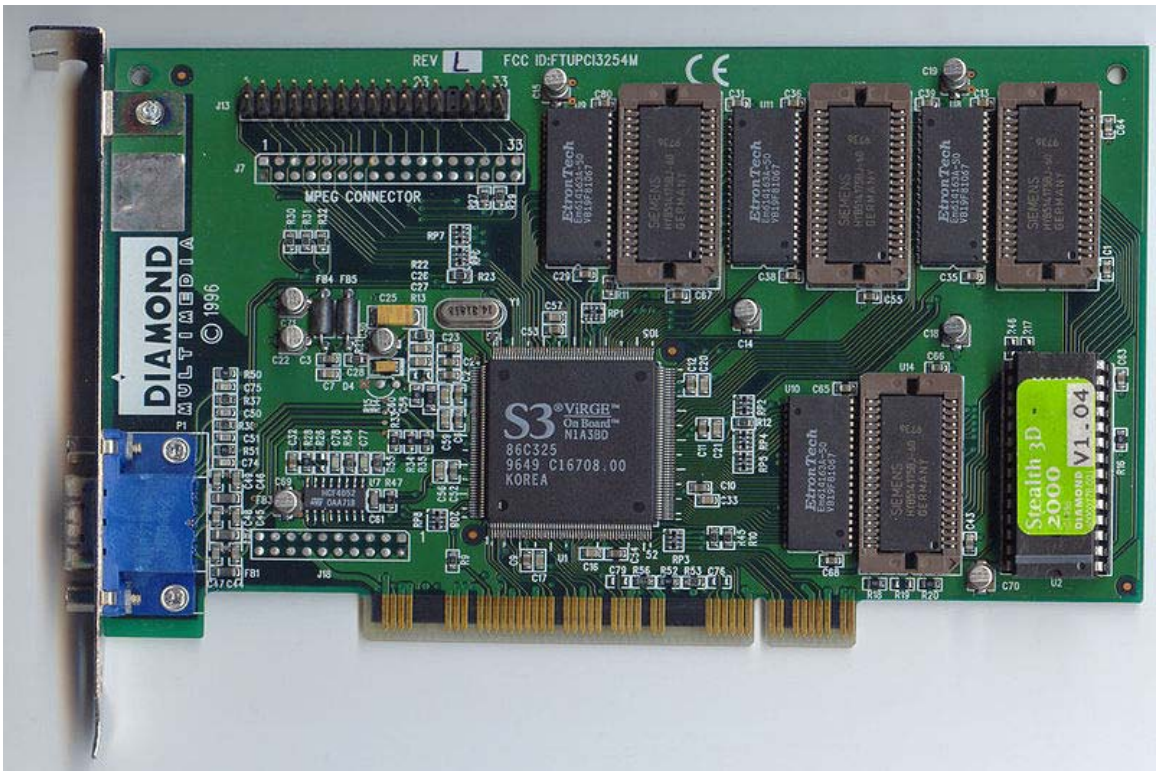
The Commodore Amiga was the first personal computer to use a full-featured GPU, called a blitter, and the first US patent filing to use the term "blitter" was "Personal computer apparatus for block transfer of bit-mapped image data," assigned to Commodore-Amiga, Inc. The Amiga was capable of offloading practically all image

generation functions to hardware, including line drawing, area filling, and block image transfer and manipulation. Also included was a graphics coprocessor with its own (primitive) instruction set. Prior to this and quite some time after, many other systems required a general purpose CPU to handle every aspect of drawing the display.

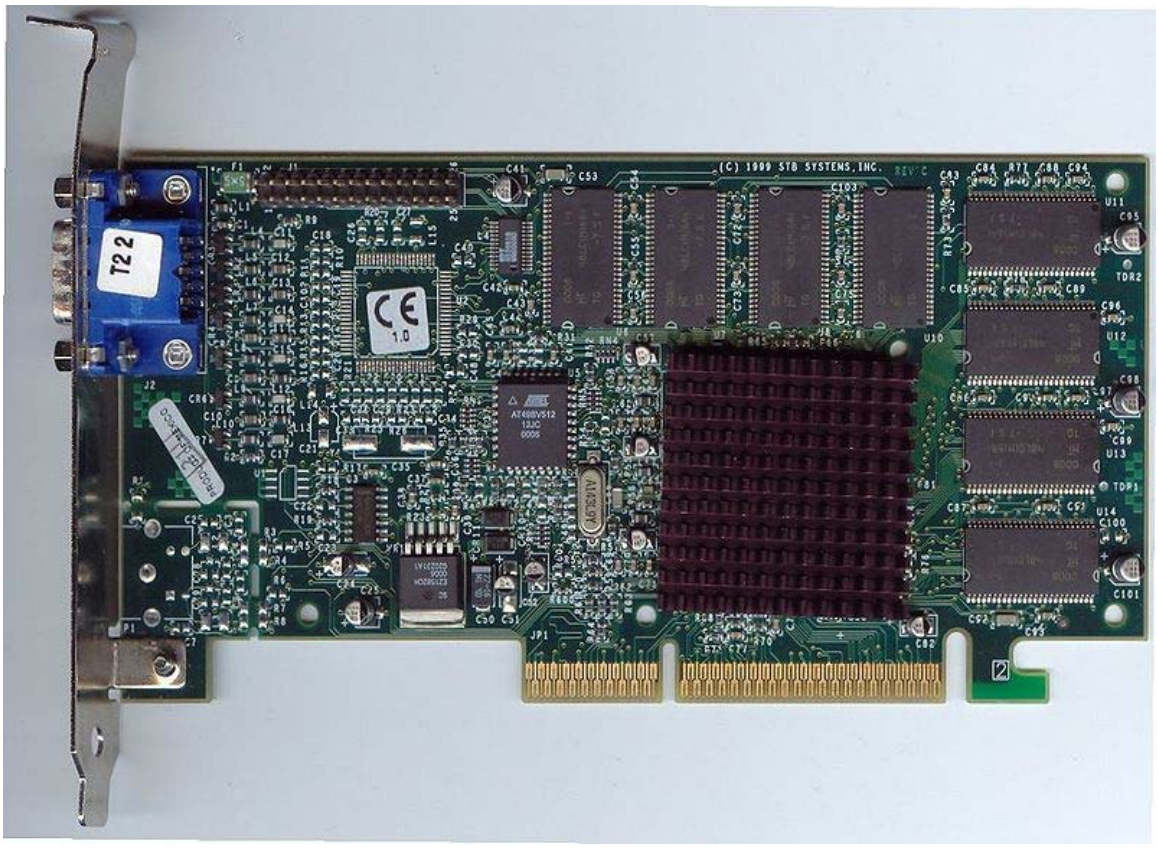
In 1987, IBM's 8514 graphics system was released as one of the first video cards for PC compatibles to implement 2D primitives in hardware.

1990s

Tseng Labs ET4000/W32p



S3 Graphics ViRGE



Voodoo3 2000 AGP card

In 1991, S3 Graphics introduced the *S3 86C911*, which its designers named after the Porsche 911 as an indication of the performance increase it promised. The 86C911 spawned a host of imitators: by 1995, all major PC graphics chip makers had added 2D acceleration support to their chips. By this time, fixed-function *Windows accelerators* had surpassed expensive general-purpose graphics coprocessors in Windows performance, and these coprocessors faded away from the PC market.

Throughout the 1990s, 2D GUI acceleration continued to evolve. As manufacturing capabilities improved, so did the level of integration of graphics chips. Additional application programming interfaces (APIs) arrived for a variety of tasks, such as Microsoft's WinG graphics library for Windows 3.x, and their later DirectDraw interface for hardware acceleration of 2D games within Windows 95 and later.

In the early and mid-1990s, CPU-assisted real-time 3D graphics were becoming increasingly common in computer and console games, which led to an increasing public demand for hardware-accelerated 3D graphics. Early examples of mass-marketed 3D graphics hardware can be found in fifth generation video game consoles such as PlayStation and Nintendo 64. In the PC world, notable failed first-tries for low-cost 3D graphics chips were the S3 *ViRGE*, ATI *Rage*, and Matrox *Mystique*. These chips were essentially previous-generation 2D accelerators with 3D features bolted on. Many were

even pin-compatible with the earlier-generation chips for ease of implementation and minimal cost. Initially, performance 3D graphics were possible only with discrete boards dedicated to accelerating 3D functions (and lacking 2D GUI acceleration entirely) such as the 3dfx *Voodoo*. However, as manufacturing technology again progressed, video, 2D GUI acceleration, and 3D functionality were all integrated into one chip. Rendition's *Verite* chipsets were the first to do this well enough to be worthy of note.

OpenGL appeared in the early 90s as a professional graphics API, but originally suffered from performance issues which allowed the Glide API to step in and become a dominant force on the PC in the late 90s. However these issues were quickly overcome and the Glide API fell by the wayside. Software implementations of OpenGL were common during this time although the influence of OpenGL eventually led to widespread hardware support. Over time a parity emerged between features offered in hardware and those offered in OpenGL. DirectX became popular among Windows game developers during the late 90s. Unlike OpenGL, Microsoft insisted on providing strict one-to-one support of hardware. The approach made DirectX less popular as a stand alone graphics API initially since many GPUs provided their own specific features, which existing OpenGL applications were already able to benefit from, leaving DirectX often one generation behind. (See: Comparison of OpenGL and Direct3D).

Over time Microsoft began to work more closely with hardware developers, and started to target the releases of DirectX with those of the supporting graphics hardware. Direct3D 5.0 was the first version of the burgeoning API to gain widespread adoption in the gaming market, and it competed directly with many more hardware specific, often proprietary graphics libraries, while OpenGL maintained a strong following. Direct3D 7.0 introduced support for hardware-accelerated transform and lighting (T&L) for Direct3D, while OpenGL already had this capability already exposed from its inception. 3D accelerators moved beyond being just simple rasterizers to add another significant hardware stage to the 3D rendering pipeline. The NVIDIA *GeForce 256* (also known as NV10) was the first consumer-level card on the market with hardware-accelerated T&L, while professional 3D cards already had this capability. Hardware transform and lighting, both already existing features of OpenGL, came to consumer-level hardware in the 90s and set the precedent for later pixel shader and vertex shader units which were far more flexible and programmable.

2000 to present

With the advent of the OpenGL API and similar functionality in DirectX, GPUs added programmable shading to their capabilities. Each pixel could now be processed by a short program that could include additional image textures as inputs, and each geometric vertex could likewise be processed by a short program before it was projected onto the screen. NVIDIA was first to produce a chip capable of programmable shading, the *GeForce 3* (code named NV20). By October 2002, with the introduction of the ATI *Radeon 9700* (also known as R300), the world's first Direct3D 9.0 accelerator, pixel and vertex shaders could implement looping and lengthy floating point math, and in general were quickly becoming as flexible as CPUs, and orders of magnitude faster for image-array operations.

Pixel shading is often used for things like bump mapping, which adds texture, to make an object look shiny, dull, rough, or even round or extruded.

As the processing power of GPUs has increased, so has their demand for electrical power. High performance GPUs often consume more energy than current CPUs.

Today, parallel GPUs have begun making computational inroads against the CPU, and a subfield of research, dubbed GPU Computing or GPGPU for *General Purpose Computing on GPU*, has found its way into fields as diverse as oil exploration, scientific image processing, linear algebra, statistics, 3D reconstruction and even stock options pricing determination. Nvidia's CUDA platform is the most widely adopted programming model for GPU computing, with OpenCL also being offered as an open standard.

GPU companies

Many companies have produced GPUs under a number of brand names. In 2008, Intel, NVIDIA and AMD/ATI were the market share leaders, with 49.4%, 27.8% and 20.6% market share respectively. However, those numbers include Intel's integrated graphics solutions as GPUs. Not counting those numbers, NVIDIA and ATI control nearly 100% of the market. In addition, S3 Graphics, VIA Technologies and Matrox produce GPUs.

Computational functions

Modern GPUs use most of their transistors to do calculations related to 3D computer graphics. They were initially used to accelerate the memory-intensive work of texture mapping and rendering polygons, later adding units to accelerate geometric calculations such as the rotation and translation of vertices into different coordinate systems. Recent developments in GPUs include support for programmable shaders which can manipulate vertices and textures with many of the same operations supported by CPUs, oversampling and interpolation techniques to reduce aliasing, and very high-precision color spaces. Because most of these computations involve matrix and vector operations, engineers and scientists have increasingly studied the use of GPUs for non-graphical calculations.

In addition to the 3D hardware, today's GPUs include basic 2D acceleration and framebuffer capabilities (usually with a VGA compatibility mode).

GPU accelerated video decoding

Most GPUs made since 1995 support the YUV color space and hardware overlays, important for digital video playback, and many GPUs made since 2000 also support MPEG primitives such as motion compensation and iDCT. This process of hardware accelerated video decoding, where portions of the video decoding process and video post-processing are offloaded to the GPU hardware, is commonly referred to as "***GPU accelerated video decoding***", "***GPU assisted video decoding***", "***GPU hardware accelerated video decoding***" or "***GPU hardware assisted video decoding***".

More recent graphics cards even decode high-definition video on the card, offloading the central processing unit. The most common API's for GPU accelerated video decoding are DxVA for Microsoft Windows operating-system, and VDPAU, VAAPI, XvMC, and XvBA for Linux and UNIX based operating-system. All except XvMC are capable of decoding videos encoded with MPEG-1, MPEG-2, MPEG-4 ASP (MPEG-4 Part 2), MPEG-4 AVC (H.264 / DivX 6), VC-1, WMV3/WMV9, Xvid / OpenDivX (DivX 4), and DivX 5 codecs, while XvMC is only capable of decoding MPEG-1 and MPEG-2.

Video decoding processes that can be accelerated

The video decoding processes that can be accelerated by today's modern GPU hardware are:

- Motion compensation (mocomp)
- Inverse discrete cosine transform (iDCT)
    - Inverse telecine 3:2 and 2:2 pull-down correction
- Inverse modified discrete cosine transform (iMDCT)
- In-loop deblocking filter
- Intra-frame prediction
- Inverse quantization (IQ)
- Variable-Length Decoding (VLD), more commonly known as slice-level acceleration
- Spatial-temporal deinterlacing and automatic interlace/progressive source detection
- Bitstream processing (CAVLC/CABAC).

GPU forms

Dedicated graphics cards

The GPUs of the most powerful class typically interface with the motherboard by means of an expansion slot such as PCI Express (PCIe) or Accelerated Graphics Port (AGP) and can usually be replaced or upgraded with relative ease, assuming the motherboard is capable of supporting the upgrade. A few graphics cards still use Peripheral Component Interconnect (PCI) slots, but their bandwidth is so limited that they are generally used only when a PCIe or AGP slot is not available.

A dedicated GPU is not necessarily removable, nor does it necessarily interface with the motherboard in a standard fashion. The term "dedicated" refers to the fact that dedicated graphics cards have RAM that is dedicated to the card's use, not to the fact that *most* dedicated GPUs are removable. Dedicated GPUs for portable computers are most commonly interfaced through a non-standard and often proprietary slot due to size and weight constraints. Such ports may still be considered PCIe or AGP in terms of their logical host interface, even if they are not physically interchangeable with their counterparts.

Technologies such as SLI by NVIDIA and CrossFire by ATI allow multiple GPUs to be used to draw a single image, increasing the processing power available for graphics.

Integrated graphics solutions

Integrated graphics solutions, **shared graphics solutions**, or **Integrated graphics processors (IGP)** utilize a portion of a computer's system RAM rather than dedicated graphics memory. Exceptions are AMD's IGPs that use dedicated sideport memory on certain motherboards. Computers with integrated graphics account for 90% of all PC shipments. These solutions are less costly to implement than dedicated graphics solutions, but are less capable. Historically, integrated solutions were often considered unfit to play 3D games or run graphically intensive programs but could run less intensive programs such as Adobe Flash. Examples of such IGPs would be offerings from SiS and VIA circa 2004. However, today's integrated solutions such as AMD's Radeon HD 3200 (AMD 780G chipset) and NVIDIA's GeForce 8200 (nForce 710|NVIDIA nForce 730a) are more than capable of handling 2D graphics from Adobe Flash or low stress 3D graphics. However, most integrated graphics still struggle with high-end video games. Chips like the Nvidia GeForce 9400M in Apple's MacBook and MacBook Pro and AMD's Radeon HD 3300 (AMD 790GX) have an improved performance, but still lag behind dedicated graphics cards. Modern desktop motherboards often include an integrated graphics solution and have expansion slots available to add a dedicated graphics card later.

As a GPU is extremely memory intensive, an integrated solution may find itself competing for the already relatively slow system RAM with the CPU, as it has minimal or no dedicated video memory. System RAM may be 2 Gbit/s to 12.8 Gbit/s, yet dedicated GPUs enjoy between 10 Gbit/s to over 100 Gbit/s of bandwidth depending on the model.

Older integrated graphics chipsets lacked hardware transform and lighting, but newer ones include it.

Hybrid solutions

This newer class of GPUs competes with integrated graphics in the low-end desktop and notebook markets. The most common implementations of this are ATI's HyperMemory and NVIDIA's TurboCache. Hybrid graphics cards are somewhat more expensive than integrated graphics, but much less expensive than dedicated graphics cards. These share memory with the system and have a small dedicated memory cache, to make up for the high latency of the system RAM. Technologies within PCI Express can make this possible. While these solutions are sometimes advertised as having as much as 768MB of RAM, this refers to how much can be shared with the system memory.

Stream Processing and General Purpose GPUs (GPGPU)

A new concept is to use a general purpose graphics processing unit as a modified form of stream processor. This concept turns the massive floating-point computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power, as opposed to being hard wired solely to do graphical operations. In certain applications requiring massive vector operations, this can yield several orders of magnitude higher performance than a conventional CPU. The two largest discrete GPU designers, ATI and NVIDIA, are beginning to pursue this new approach with an array of applications. Both nVidia and ATI have teamed with Stanford University to create a GPU-based client for the Folding@Home distributed computing project, for protein folding calculations. In certain circumstances the GPU calculates forty times faster than the conventional CPUs traditionally used by such applications.

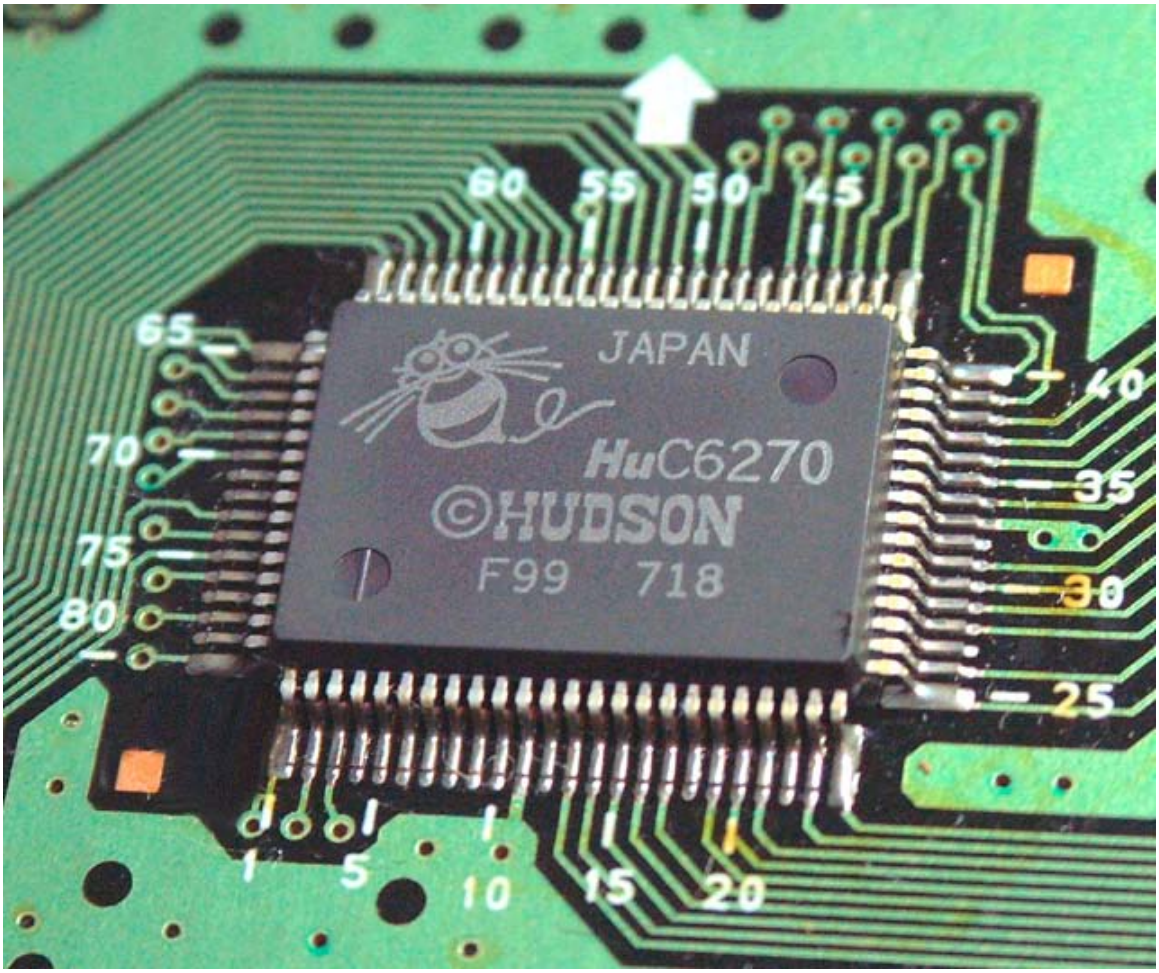
Furthermore, GPU-based high performance computers are starting to play a significant role in large-scale modelling. Three of the 5 most powerful supercomputers in the world take advantage of GPU acceleration. This includes the current leader as of October 2010, Tianhe-1A, which uses the NVIDIA Tesla platform.

Recently NVidia began releasing cards supporting an API extension to the C programming language CUDA ("Compute Unified Device Architecture"), which allows specified functions from a normal C program to run on the GPU's stream processors. This makes C programs capable of taking advantage of a GPU's ability to operate on large matrices in parallel, while still making use of the CPU when appropriate. CUDA is also the first API to allow CPU-based applications to access directly the resources of a GPU for more general purpose computing without the limitations of using a graphics API.

Since 2005 there has been interest in using the performance offered by GPUs for evolutionary computation in general, and for accelerating the fitness evaluation in genetic programming in particular. Most approaches compile linear or tree programs on the host PC and transfer the executable to the GPU to be run. Typically the performance advantage is only obtained by running the single active program simultaneously on many example problems in parallel, using the GPU's SIMD architecture. However, substantial acceleration can also be obtained by not compiling the programs, and instead transferring them to the GPU, to be interpreted there. Acceleration can then be obtained by either interpreting multiple programs simultaneously, simultaneously running multiple example problems, or combinations of both. A modern GPU (*e.g.* 8800 GTX or later) can readily simultaneously interpret hundreds of thousands of very small programs.

Chapter 3

Hudson Soft HuC6270



HuC6270

HuC6270 is a Video Display Controller (VDC) developed and manufactured by Hudson Soft. The VDC was used in the PC Engine game console produced by NEC Corporation in 1987 and in the SuperGrafx and TurboGrafx 16 also developed by NEC.

Besides NEC, the VDC was used in two arcade games. The arcade version of *Bloody Wolf* ran on a custom version of the PC-Engine. Arcade hardware is missing the second 16-bit graphic chip known as the VCE, that is in the PC-Engine. This means the VDC directly accesses palette RAM and builds out the display signals/timing. A rare Capcom quiz-type arcade game also ran on a modified version of the SuperGrafx hardware, which used two VDCs.

The HuC6270 is responsible for building the main image of the system (background and sprites). The chip itself is able to create/generate all the timings needed to build an output signal, but in the PC-Engine is works in slave mode. Where the VCE builds the frame timings for the video signal and the VDC uses the H and V sync lines as input (rather than output) to keep things in sync. But the VDC is still able to define its own video frame inside the VCE's output frame. Both horizontally and vertically. It's possible to define multiple frames both vertically and horizontally. Coryoon does this when you enter in the 4 split screen mode of the sound test option.

While the VCE supplies the H sync signal for the VDC to generate an interrupt for the CPU (for scanline effects/changes), the VDC can also define its own H sync interrupts, causing multiple H interrupts on a single scanline. The VCE also sets the resolution of the VDC. It supplies the divided clock signal, of the master clock. The VDC runs either faster or slower, depending on this 'dot clock' setting. But it doesn't know what speed it's running. It's up to the programmer to correctly set/match up the VDC horizontal settings to be in sync with the VCE.

The VDC has a number of VRAM read/write slots open for the CPU to use during active display. The number of slots are actually faster than what the CPU could possible use/fill. But if the CPU was fast enough, the VDC has a means of stalling the CPU. This is attached to /RDY pin of the HuC6280. This is a unique attribute of the VDC, as other systems of that era (both 8-bit and 16-bit) had extremely restrictive and/or no access to vram during active display (doing so usually corrupts the display). This means the CPU as the option of updating vram outside of the normal and relatively small vblank time.

Sprite pixels are fetched hblank time, not during active scanline like similar systems. Relative to other systems, the VDC both parses and fetches all the sprite pixel data in tiny amount of time at a impressive speed for its era. The VDC has an internal buffer that hold all the sprite pixels for the scanline. This holds a maximum of 256 4-bit pixels. The VDC can not display more than this on a single scanline and any sequentially placed sprites on the line won't be shown. The order of which sprites have priority in the scanline buffer, is direct relative to the top down list of the SAT (sprite attribute table). An alternative to sprite 'drop out' is to cycle between two sets of SAT lists as 60 Hz. This is known as 'flicker'; a common display artifact effect of the time. This isn't automatic, but handled by the programmer. There's a sprite overflow detection flag which is set when sprite drop

out happens. The programmer can then decide what to do about the problem based on the state of this. Most games handle the problem with the flicker method. Some games do not. VRAM read/write by the cpu is off limits when the VDC is filling the sprite scanline buffer during hblank. Depending on how many pixels are fetch, is how long the wait is. But in terms of transfer, it's fairly unnoticeable. If accessed during this time by the CPU, the VDC delays it after the one-WORD FIFO is filled. The if the frame settings are too wide, too much active display is set into the overscan area, this will cause the VDC to not have enough time to fill the 256 pixel buffer and will start dropping off sprite cells per 8 cell segment frame define.

Sprite size are made up of a matrix of 16×16 sprite 'cells'. When sprite dropout happens, it happens on a sprite cell basis; not at a matrix defined level. The sprites sizes are: 16×16, 16×32, 16×64, 32×16, 32×32, 32×64. The larger size sprites have a memory alignment requirement that must be met, else parts of the sprite won't show. The sprite attributes are defined in the SAT (sprite attribute table). The size, the sub palette #, BG priority, and the X/Y flipping options. The subpalette value ranges from 0 to 15 and are separate from the BG subpalette range of 0 to 15. Flipping options apply to the sprite as a whole (matrix), and not individual cell basis. Inter sprite sprite is directly based on first in line. The sprite defined in a top position of the SAT will always show above sprites lower in the attribute table. The BG priority flag tells of the VDC if the sprite should be shown above or below the background layer. No sprite can be made to be shown behind BG color #0 (which is considered its own layer because of this). The priority of the sprite in relation to the BG layer, does not change inter sprite priority. Maybe PC-Engine games use this exploit for force sections of the background to appear above sprites, in a pixel level basis. Ys I & II whenever a sprite appears behind a section of the background. The common method is just to use a simple pixel mask, instead of having to store redundant pixel data of the BG as sprite. This saves space and also allows the mask to be use for multiple BG sections, if the outline is the same but different color/pixel detail. Another game exploits this to get a translucent graphic effect; Jackie chan. The exploit is where the logic of the per sprite priority causes the sprite underneath the BG layer to show instead of the high BG priority sprite. But since the sprite is being masked by BG pixels, the BG pixel data is pushed to show on top of the top-layer sprite - when passing underneath/top of it.

The sprite pixel data is a different format than the BG tile format. The sprite data is stored in sequential 1-bit planes and not interleaved like tile planes. The reason for this is that the VDC is a complete 16-bit memory device. All memory (registers as well as vram) is accessed as WORDs. The VDC fetches 16 pixels at a time, four times, Once from each plane. That's wide the cell size is 16 pixels wide. By comparison, the tile data is 8 pixels wide. So when the VDC fetches a WORD for tile data, it grabs 8 pixels for plane 0 and plane 1 at the same time. Hence the composite nature of the tile format, in contrast to the sprite format.

The VDC outputs 9-bit digital pixel data to the VCEs pixel bus. The 9th bit is set when the sprite pixel is being sent by the VDC, instead of the tile layer or BG color #0 layer. This is how the VCE knows what section of palette ram to assign to have pixel. The logic

is as follows: tile pixel = (palette # * 16)+tile pixel, sprite pixel = (palette # *16)+tile pixel+$100. This assumes tile pixel != 0 and sprite pixel != 0, else output pixel = 0 (BG color #0 layer).

The SAT is located in a special register set in the VDC. But a copy of the SAT must be in vram, in order for the VDC to transfer it. This is referred to as the SATB (Sprite Attribute Table Buffer). If the DCR register has the specific flag set, the SATB to SAT DMA will automatically happen once the defined displayable frame has ended. Otherwise it's not possible to change the internal SAT registers during active display (unless the VDC is setup to display multiple frames). The location of the SATB can be anywhere in vram. The maximum size of SATB is 64 entries or 512 bytes (256 WORDs). The SAT size or sprite scanline buffer do not change with changing of the 'dot clock' resolution, like with the Genesis and other similar systems. Giving it a disadvantage for higher resolutions for sprite to scanline pixel ratio.

Only one background layer can be shown by the VDC. This is the tilemap. It can be configured to a number of sizes; 32x32, 32x64, 128x32, 64x32, 64x64, 128x64. The X and Y size are multiples of 8 pixels. For example, a 32x32 setting is a 256x256 pixel map. The tilemap mirrors/wraps both vertically and horizontally and are independent of the display frame for that given scanline. The tilemap position relative to the display frame, starting from the top, is controlled by the BXR and BYR registers. These registers are latched at the start of every 'VDC' scanline. At using a horizontal scanline interrupt, it is possible to change both the X and Y offset of the tilemap on any scanline. This is how many games divide the display into different scrollable areas. Other effects can be made with this method, including stretching or shrinking the background layer vertically or distortion effects.

The VDC registers are independent are of VRAM access. They also aren't shared for building the display like the Famicom display registers, making mid-frame effects stable. The latch systems facilitates in this as well. But certain registers are latch as specific points. Some are latched per scanline, some are latched per frame end. Horizontal frame build registers are latched per scanline and vertical frame build registers per frame end. All the VDC registers are 16-bit, but registers themselves are buffered with no latch systems. This means either the upper or lower byte can be changed without writing a whole WORD update.

VDC memory range is 64 kWORDs (128 kbytes), but only 32 kWORDS (64 kbytes) are installed. Access to the upper range is open bus. The tilemap entry is 16-bit or a single WORD. 12-bit is the tile address (in 16 WORD offsets), 4-bit is the subpalette assignment. There is not support for tile flipping or per tile priority. The tilemap can address tiles anywhere in vram, including the tilemap location itself, which is fixed at starting point of $0000 and unchangeable.

Lastly, the VDC only has two DMA modes. One is the SATB to SAT DMA. The other is vram to vram DMA. The DMA is separate from the CPU other than the setup/request time. Neither DMA's can happen during active display. The dot clock/speed of the VDC

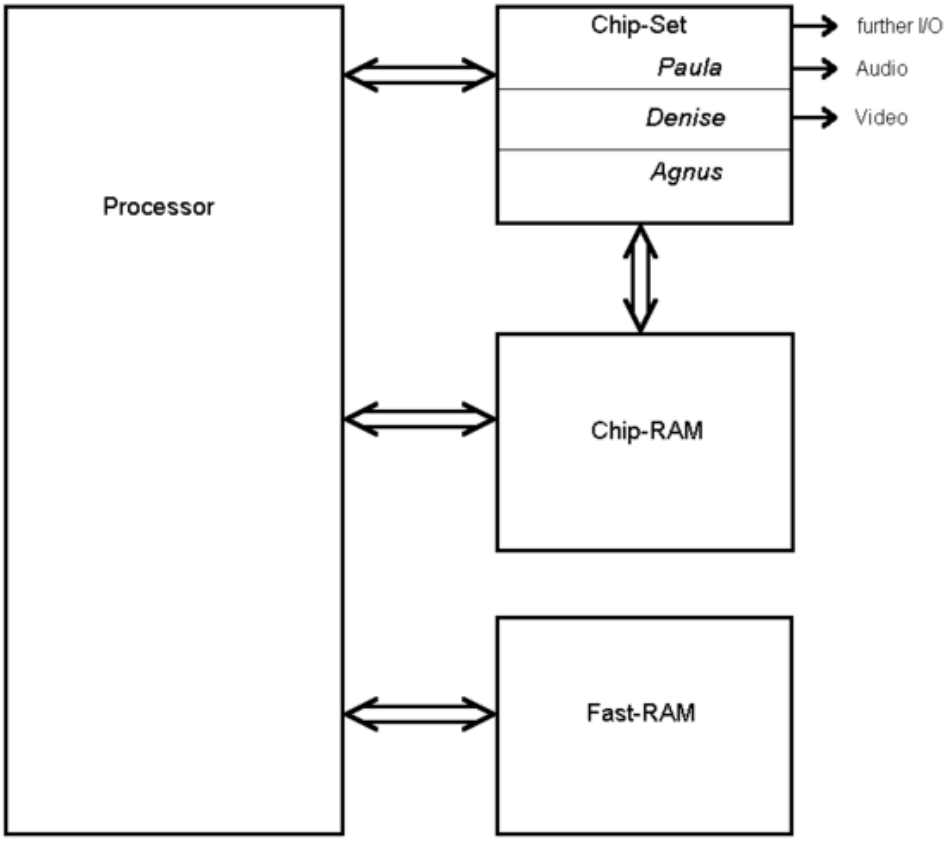
directly effects the speed of these DMA operations. Not only it is possible to switch to any resolution mid screen (on any scanline), but also during vblank area as well.

The VDC builds a total of four graphics layers: G/S'/B/S. G = BG color #0. S' = low BG priority sprite. B = tilemap/background layer. S = high BG priority sprite. There are no known revisions to the HuC6270, like the HuC6280A and HuC6260A.

Chapter 4

Original Chip Set

The **Original Chip Set** (**OCS**) was a chipset used in the earliest Commodore Amiga computers and defined the Amiga's graphics and sound capabilities. It was succeeded by the slightly improved Enhanced Chip Set (ECS) and greatly improved Advanced Graphics Architecture (AGA).



Amiga Chip Set

The original chipset appeared in Amiga models built between 1985 and 1990: the Amiga 1000, Amiga 2000, Amiga CDTV, and Amiga 500.

Overview of chips

The chipset which gave the Amiga its unique graphics features consists of three main "custom" chips; *Agnus*, *Denise*, and *Paula*. Both the original chipset and the enhanced chipset were manufactured using NMOS logic technology by Commodore's chip manufacturing subsidiary, MOS Technology. All three custom chips were originally packaged in 48-pin DIPs; later versions of Agnus, known as Fat Agnus, were packaged in an 84-pin PLCC.

Agnus is the central chip in the design. It controls all access to chip RAM from both the central 68000 processor and the other custom chips, using a complicated priority system. Agnus includes sub-components known as the *blitter* and the *copper*. The original Agnus can address 512 KB of chip RAM. Later revisions, dubbed 'Fat Agnus', added 512 KB pseudo-fast RAM, which for ECS was changed to 1 MB (sometimes called 'Fatter Agnus') and subsequently to 2 MB chip RAM.

Denise is the main video processor. Without using overscan, the Amiga's graphics display is 320 or 640 pixels wide by 200 (NTSC) or 256 (PAL) pixels tall. Denise also supports interlacing, which doubles the vertical resolution. Planar bitmap graphics are used, which splits the individual bits per pixel into separate areas of memory, called bitplanes. In normal operation, Denise allows between 1 and 5 bitplanes, giving 2 to 32 unique colours. These colours are selected from palette of 4096 colours. A 6th bitplane is available for two special video modes: Halfbrite mode and Hold And Modify mode. Denise also supports eight sprites, sub-pixel scrolling, and a "dual playfield" mode. Denise also handles mouse and digital joystick input.

Paula is primarily the audio chip, with 4 independent hardware-mixed 8-bit PCM sound channels, each of which supports 64 volume levels (no sound to maximum volume) and sample rates from roughly 20 Hz to almost 29 kHz. Paula also handles interrupts and various I/O functions including the floppy disk drive, the serial port, and analog joysticks.

There are many similarities - both in overall functionality and in the division of functionality into the three component chips - between the OCS chipset and the much earlier and simpler chipset of the Atari 8-bit family of home computers, consisting of the ANTIC, GTIA and POKEY chips; both chipsets were conceptually designed by Jay Miner, which explains the similarity.

Agnus

The Agnus chip is in overall control of the entire chipset's operation. All operations are synchronised with the output of the video beam. This includes access to the built-in RAM, known as chip RAM because the chipset has access to it. Both the central 68000 processor and other members of the chipset have to arbitrate for access to RAM via

Agnus. In computing architecture terms, this is Direct Memory Access (DMA), where Agnus is the DMA Controller (DMAC).

Agnus has a complex priority-based memory access policy. For example, bitplane data fetches are more important than blitter transfers. As the original 68000 processor in Amigas could only access memory on every second clock cycle, Agnus operated a system where the time-critical custom chips access got the "odd" clock cycle and the CPU got the "even" cycle, thus the CPU did not get locked out of memory access and did not appear to slow down. However, non-time-critical custom chip access, such as *blitter* transfers, can use up any spare odd or even cycles and, if the "BLITHOG" (blitter hog) flag is set, Agnus can lock out the even cycles from the CPU in deference to the *blitter*.

Agnus's timings are measured in "colour clocks" of 280 ns. This is equivalent to two low resolution (140 ns) pixels or four high resolution (70 ns) pixels. Like Denise, these timings were designed for display on household TVs, and can be synchronised to an external clock source.

Blitter

The *blitter* is a sub-component of Agnus. "Blit" is shorthand for "block image transfer" or bit blit. The blitter is a highly parallel memory transfer and logic operation unit. It has three modes of operation: copying blocks of memory, filling blocks (e.g. polygon filling) and line drawing.

The blitter allows the rapid copying of video memory, meaning that the CPU can be freed for other tasks. The blitter was primarily used for drawing and redrawing graphics images on the screen, called "bobs", short for "blitter objects".

The blitter's block copying mode takes zero to three data sources in memory, called A, B and C, performs a programmable boolean function on the data sources and writes the result to a destination area, D. Any of these four areas can overlap. The blitter runs either from the start of the block to the end, known as "ascending" mode, or in reverse, "descending" mode.

Blocks are "rectangular"; they have a "width" in multiples of 16 bits, a height measured in "lines", and a "stride" distance to move from the end of one line to the next. This allows the blitter to operate on any conceivable video resolution. The copy automatically performs a per-pixel logical operation. These operations are described generically using minterms. This is most commonly used to do direct copies (D = A), or apply a pixel mask around blitted objects (D = (C AND B) + A). The copy can also barrel shift each line by 0 to 15 pixels. This allows the blitter to draw at pixel offsets that are not exactly multiples of 16.

These functions allow the Amiga to move GUI windows around the screen rapidly as each is represented in graphical memory space as a rectangular block of memory which may be shifted to any required screen memory location at will.

The blitter's line mode draws single-pixel thick lines using the Bresenham's line algorithm. It can also apply a 16-bit repeating pattern to the line. The blitter's filling mode is used to fill per-line horizontal spans. On each span, it reads each pixel in turn from right to left. Whenever it reads a set pixel, it toggles filling mode on or off. When filling mode is on, it sets every pixel until filling mode is turned off or the line ends. Together, these modes allow the blitter to draw individual flat-shaded polygons, albeit very slowly in comparison to modern 3D graphics chipsets or the CPU of a moderately fast Amiga.

Copper

The *copper* is another sub-component of Agnus; The name is short for "co-processor". The copper is a programmable finite state machine that executes a programmed instruction stream, synchronized with the video hardware.

When it is turned on, the copper has three states; either reading an instruction, executing it, or waiting for a specific video beam position. The copper runs a program called the *copper list* in parallel with the main CPU. The copper runs in sync with the video beam, and it can be used to perform various operations which require video synchronization. Most commonly it is used to control video output, but it can write to most of the chipset registers and thus can be used to set audio registers or interrupt the CPU.

The copper list has three kinds of instructions, each one being a pair of two bytes, four bytes in total:

- The MOVE instruction writes a 16-bit value into one of the chipset's hardware registers.
- The WAIT instruction halts copper execution until a given beam position is reached, thus making possible to synchronize other instructions with respect to screen drawing. It can also wait for a blitter operation to finish.
- The SKIP instruction will skip the following copper instruction if a given beam position has already been reached. This can be used to create copper list loops.

The length of the copper list program is limited by execution time. The copper restarts executing the copper list at the start of each new video frame. There is no explicit "end" instruction; instead, the WAIT instruction is used to wait for a location which is never reached.

Uses of the copper

- The copper is most commonly used to set and reset the video hardware registers at the beginning of each frame.
- It can be used to change video hardware mid-frame. This allows the Amiga to change video configuration, including resolution, between scanlines. This allows the Amiga to display different horizontal resolutions, different colour depths, and entirely different frame buffers on the same screen. The AmigaOS graphical user interface allows two programs to operate at different resolutions in different

buffers, while both are visible on the screen simultaneously. A paint program might use this feature to allow users to draw directly on a low resolution Hold And Modify screen, while offering a high resolution toolbar at the top or bottom of the screen.

- The copper can also change colour registers once per scanline, creating the "raster bars" effect seen commonly in Amiga games. The copper can go further than this and change the background colour often enough to make a blocky graphics display without using any bitmap graphics at all.
- The copper allows "re-use" of sprites; after a sprite has been drawn at its programmed location, the copper can then immediately move it to a new location and it will be drawn again, even on the same scanline.
- The copper can also be used to program and operate the blitter. This is useful for doing several blitter operations in sequence, as the copper can wait for the blitter to finish and then immediately reprogram it for the next operation.
- The copper can be used to produce "sliced HAM", or S-HAM, this consists of building a copper list that switches the palette on every scanline, improving the choice of base colours in Hold And Modify mode graphics.

Denise

Denise controls the video timings, but can also synchronise to an external video signal. Denise is programmed to fetch planar video data from 1 to 5 bitplanes and translate that into a colour lookup. The number of bitplanes is arbitrary, thus if 32 colours are not needed, 2, 4, 8 or 16 can be used instead. The number of bitplanes (and resolution) can be changed on the fly, usually by the copper. This allows for very economical use of RAM. There is also a sixth bitplane, which can be used in three special graphics modes:

In Extra-HalfBrite (EHB), if a pixel is set on the sixth bitplane, the brightness of the regular 32 colour pixel is halved. Early versions of the Amiga 1000 sold in the United States did not have the Extra-HalfBrite mode.

In Hold-and-Modify mode (HAM), each 6-bit pixel is interpreted as 2 control bits and 4 data bits. The 4 possible permutations of control bits are "set", "modify red", "modify green" and "modify blue". With "set", the 4 data bits act like a regular 16-colour display look up. With one of the "modify"s, the red, green or blue component of the previous pixel is modified to the data value, and the other two components are held from the previous pixel. This allows all 4096 colours on screen at once.

In Dual Playfield mode, instead of acting as a single screen, two "playfields" of 8 colours each (3 bitplanes each) are drawn on top of each other. They are independently scrollable and the background colour of the top playfield "shines through" to the underlying playfield.

There are two horizontal graphics resolutions, "lowres" with 140 ns pixels and "hires" with 70 ns pixels. This makes the display 320 or 640 pixels wide without using overscan. Denise supports very wide overscan; there is no need for a border around the graphics as

other computers suffered from. Vertical resolution, without overscan, is 200 pixels for an 60 Hz NTSC Amiga or 256 for a 50 Hz PAL Amiga. This can be doubled using an interlaced display.

Denise can also lay up to 8 sprites on top of the graphics, and detect collisions between sprites and the background, or between sprites. These sprites have 3 visible colours and one transparent colour, however two sprites can be "attached" to make a single 15 colour sprite.

External video timing

Under normal circumstances, the Amiga generates its own video timings, but the chipset also supports synchronising itself to an external signal so as to achieve genlocking with external video hardware. There is also an 1 bit output on this connector that indicates whether the Amiga is outputting background colour or not, permitting easy overlaying of Amiga video onto external video. This made the Amiga particularly attractive as a character generator for titling videos and broadcast work, as it avoided the use and expense of AB roll and chromakey units that would be required without the genlock support. The support of overscan, interlacing and genlocking capabilities, and the fact that the display timing was very close to broadcast standards (NTSC or PAL), made the Amiga the first ideal computer for video purposes, and indeed, it was used in many studios for digitizing video data (sometimes called frame-grabbing), subtitling and interactive video news.

Paula

The Paula chip is mainly used to produce audio output. The chip has 4 DMA-driven 8-bit PCM sample sound channels. Two sound channels are mixed into the left audio output, and the other two are mixed into the right output, producing stereo audio output. The only supported hardware sample format is signed linear 8-bit two's complement. Each sound channel has an independent volume and frequency. Internally, the audio hardware is implemented by four state machines each having eight different states.

Additionally the hardware allows one channel in a channel pair to modulate the other channel's period or amplitude. It is rarely used on the Amiga due to both frequency and volume being controllable in better ways, but could be used to achieve different kinds of tremolo and vibrato, and even rudimentary FM synthesis effects.

With some special programming tricks it is possible to produce 14-bit audio by combining two channels set at different volumes, giving two 14-bit channels instead of four 8-bit channels. This is done by playing the high byte of a 16 bit sample at maximum volume, and the low byte at minimum volume (both ranges overlap, so the low byte needs to be shifted right two bits).

On a regular NTSC or PAL screen display, audio playback, using DMA, is limited to a maximum sampling rate of 28867 Hz (PAL: 28837 Hz), due to the amount of data that

can be fetched from memory in the time allocated to Paula. As explained in the discussion of Agnus, memory access is prioritized and only a few slots for memory access are available to Paula's sound channels. This limit can be overcome in the Enhanced Chip Set by using a higher frequency screen mode, or by using the CPU directly to drive audio output.

The Amiga contains an analog low-pass filter (reconstruction filter) which is external to Paula. The filter is a 12 dB/oct Butterworth low-pass filter at approximately 3.3 kHz. The filter can only be applied globally to all 4 channels. In models after the Amiga 1000, the brightness of the power LED is used to indicate the status of the filter. The filter is active when the LED is at normal brightness, and deactivated when dimmed (on early Amiga 500 models the LED went completely off). Models released before Amiga 1200 also have a static "tone knob" type low pass filter that is enabled regardless of the optional "LED filter". This filter is a 6 dB/oct low pass filter with cutoff frequency at 4.5 or 5 kHz.

Floppy disk controller

The floppy controller is unusually flexible. It can read and write raw MFM or GCR data in any format via DMA or programmed I/O. It also provides a number of convenient features, such as sync-on-word (in MFM coding, $4489 is usually used as the sync word). MFM encoding/decoding is usually done with the blitter — one pass for decode, three passes for encode. Normally the entire track is read or written in one shot, rather than sector-by-sector.

In addition to the native 880 kB 3.5-inch disk format, the controller can handle many foreign formats, such as:

- IBM PC
- Apple II
- Mac 800 kB (requires a Mac drive)
- AMAX Mac emulator (A special floppy of only 200 kB to exchange data between Amiga and Macintosh could be formatted by Amiga, and it could be read and written by floppy drives of both systems)
- Commodore 1541 (requires 5¼" inch drive slowed to 280 rpm)
- Commodore 1581 formatted 3½" floppy for C64 and C128

Serial port

The serial port is rudimentary; programmed I/O only and lacking a FIFO buffer. It does have one positive attribute, which is that virtually any bit rate can be selected, including all the standard rates, MIDI rate, as well as extremely high custom rates.

Origin of the chip names

- The name Agnus is derived from 'Address GeNerator UnitS' since it houses all address registers and controls memory access of the custom chips.

- Denise was called "Daphne" in an early phase.
- Paula was named after the girlfriend of the chip designer.

Amiga graphics chipset roadmap

Released	Acronym	Models that used it
1985	OCS	A1000, A2000, A500
1989	Ranger	Canceled by Commodore and replaced by ECS due to its high price
1990	ECS	A3000, A500+, A600, A2000
1992	AGA	A1200, A4000, CD32
1994	AAA	Canceled by Commodore and replaced by Hombre due to its high price
1994	Commodore AA+ Chipset	Planned but never designed due to lack of financing
1995	Hombre	Never released due to Commodore bankruptcy

Chapter 5

CTIA and GTIA & ANTIC

CTIA and GTIA

Color Television Interface Adaptor (CTIA) and its successor **Graphic Television Interface Adaptor** (GTIA) are custom chips used in the Atari 8-bit family of computers and in the Atari 5200 console. In these systems, a CTIA or GTIA chip works together with ANTIC to produce video display. The chips were designed by George McLeod with technical assistance of Steve Smith.

Color Television Interface Adaptor and *Graphic Television Interface Adaptor* are names of the chips as stated in the Atari field service manual. Various publications named the chips differently, sometimes using the alternative spelling *Adapter* or *Graphics*, or claiming that the "C" in "CTIA" stands for Colleen/Candy and "G" in "GTIA" is for George.

History

The CTIA was designed in 1977 as part of the chipset for use in an improved successor of the Atari 2600 console. The 2600 used a chip known as the Television Interface Adaptor, or TIA. In terms of graphics support, the design of the CTIA followed that of the TIA - it had a "playfield" layer for background graphics along with several "players" and "missiles" (today known as sprites) for moving foreground objects. However, the sprites were improved in number, from two players and two missiles to four of each. A fifth sprite in the TIA, the "ball", was removed, as some of its capabilities were combined into the missiles. The four missiles could alternately be combined to form a fifth player, by setting a register.

During development, the home computer revolution started in earnest in the later half of 1977. In response, Atari decided to release two versions of the new machine, a low-end model as a games console, and a high-end version as a home computer. In either role, a

more complex playfield would be needed, especially support for character graphics in the computer role. For this purpose the new ANTIC chip was introduced to handle the storage and interpretation of a bitmap framebuffer, which the TIA did not support. Under the new model, the ANTIC would feed the CTIA with data, which would then be colored and sent into the video circuitry. This had the added advantage of greatly reducing programming complexity, compared to the "racing the beam" system used in the 2600.

As a result of these changes, the number and selection of graphics modes on the new models was greatly improved over the TIA. Instead of a single playfield mode with 20 or 40 bits of resolution, the CTIA/ANTIC had eight modes with various resolutions and color depths, allowing the programmer to select a mode with the minimum memory needs they required for their display. Resolutions varied from 40 to 320 pixels horizontally, 24 to 192 vertically, and 2 to 4 colors per line. A particularly common mode for gaming purposes was a 160 x 192 x 4 color mode later known as "graphics 7.5".

The original design of the CTIA circuit also included support for three additional 16-color graphics modes. This feature was ready before the computers' November 1979 debut, but was delayed so much in the development cycle that Atari had already ordered a batch of about 100,000 CTIA chips with the graphics modes missing. Not wanting to throw away the already-produced chips, the company decided to use them in the initial release of the Atari 400 and 800 models in the US market. The CTIA-equipped computers, lacking the 3 graphics modes, were shipped until October–November 1981. From this point, all new Atari units were equipped with the new chip, now called GTIA, that supported the new graphics modes.

The Atari computers' operating system ROM supported the 16-color graphic modes from the start, which allowed for easy replacement of the CTIA with the GTIA once it was ready. Atari authorized service centers would install a GTIA chip in CTIA-equipped computers free of charge if the computer was under warranty; otherwise the replacement would cost $62.52.

GTIA was also mounted in all later Atari XL and XE computers and Atari 5200 consoles.

Functions

The CTIA/GTIA is a television interface chip. It converts the digital commands from ANTIC into the signal that goes to the television. It also performs the following additional functions:

- It is responsible for adding color to the display.
- It draws sprites (known as Player/Missiles) over the background graphics (known as playfield).
- It checks for collisions among the sprites as well as between the sprites and the background.

It also performs a few additional minor tasks:

- It is responsible for reading state of joysticks' triggers (bottom buttons only in case of Atari 5200 controllers).
- It contains four input/output pins that are used in different ways depending on the system:
    - In Atari 8-bit computers, three of the pins are used to read state of the console keys (Start/Select/Option). The fourth pin controls the speaker built into the Atari 400/800, used to generate keyboard clicks. On later models there is no internal speaker, but the keyclick is still generated by GTIA and mixed with the regular audio output.
    - In the Atari 5200, the pins are used as part of the process that reads state of joysticks' keyboards.

GTIA enhancements

The GTIA chip was backward compatible with the CTIA, and added 3 new graphics modes. All 3 modes were 80x192 pixels, with the difference being in the colors allowed. With the CTIA chip, the Atari was limited to a maximum of 4 colors in graphics, unless special programming techniques were used. The new modes allowed the following:

- One mode, known as Graphics 9 to BASIC programmers, can display 16 shades of a single hue (there are 16 possible hues).
- The next mode, named Graphics 11, allows for 16 hues with a single shade/luminance value.
- Finally, Graphics 10 allows for 9 colors of any hue/luminance, from a palette of 128 colors.

Of these modes, Graphics 9 is particularly notable. It enabled the Atari to display gray-scale digitized photographs, which despite their low resolution were very impressive at the time. Additionally, by allowing 16 shades of a single hue rather than the 8 available in other graphics modes, it increased the amount of different colors the Atari could display from 128 to 256. Unfortunately this feature was limited for use in this mode only, which due to its low resolution was not widely used.

The GTIA also fixed an error in CTIA that caused graphics to be misaligned by "half a color clock". The side effect of the fix was that programs that relied on color artifacts in high-resolution monochrome modes would reverse their colors.

Atari owners can determine if their machine is equipped with the CTIA or GTIA by executing the BASIC command `POKE 623,64`. If the screen blackens after execution, the machine is equipped with the new GTIA chip. If it stays blue, the machine has a CTIA chip instead.

Versions

by part number

- C012295 — NTSC CTIA
- C014805 — NTSC GTIA
- C014889 — PAL GTIA
- C020120 — French SECAM GTIA (FGTIA)

Atari, Inc. intended to combine functions of the ANTIC and GTIA chips in one integrated circuit to reduce production costs of Atari computers and 5200 consoles. Two such prototype circuits were being developed, however none of them entered production.

- C020577 — CGIA
- C021737 — KERI

Bugs

The last Atari XE computers made for the Eastern European market were built in China. Many if not all have a buggy PAL GTIA chip. The luma values in Graphics 9 and higher are at fault, appearing as stripes. Replacing the chip fixes the problem.

ANTIC

Alphanumeric Television Interface Controller (**ANTIC**) is an early video system chip used in the Atari 8-bit family of microcomputers as well as the Atari 5200 in the 1980s. The chip was patented by Atari, Inc. in 1981. ANTIC corresponded roughly to the graphics chips in other home computers of the era, with the related CTIA/GTIA providing additional support for color modes and sprites.

Features

ANTIC's most notable features are:

- 14 different graphics/text modes
- Display modes can be mixed onscreen
- Non-fixed screen RAM. Allows screen RAM to be located almost anywhere in memory. This allows for easy page-flipping, scrolling and other effects.

ANTIC is a LSI ASIC dedicated to generating 2D computer graphics to be shown on a television screen or computer display. Atari advertised it as a "true microprocessor", in that it has an *instruction set* to run *programs* (called *display lists*) to process *data*. Nonetheless ANTIC has no capacity for writing back computed values to memory, it merely reads data from memory and processes it for output to the screen, therefore it does

not qualify as a Turing machine in the mathematical sense of an abstracted computation device.

The display list

The display list and the display data are written into RAM by a 6502-compatible CPU. The ANTIC retrieves that information from RAM using a technique known as direct memory access (DMA). It processes the higher level instructions in the display list and translates these instructions into a real-time stream of simpler instructions to the CTIA chip, a combination providing for 12 graphics modes. With the more advanced GTIA, 16 modes are available.

ANTIC has four types of instructions:

- Map mode - display colored pixels
- Character mode - display character data
- Blank line - display horizontal blank lines (solid color)
- Two jump instructions - reload ANTIC's program counter (3-byte instructions)

One of the jump instructions is a simple JMP - it transfers the display list execution to another place within the 64k address space. The other one, JVB (Jump on Vertical Blank) instruction is placed at the end of the *display list*. Its role is to get the display list execution in sync with the Vertical Blanking. The JVB's argument usually points to the beginning of the same display list, but it of course can also point to another display list, so that a chain of display lists, executed after consecutive blanking pulses, is formed.

Each instruction has additional options by setting specific bits:

- DLI - Display list interrupt
- Load Memory Scan (LMS) - Loads address of graphics/character data (3-byte instruction)
- Vertical scrolling - Enables vertical fine scrolling
- Horizontal scrolling - Enables horizontal fine scrolling

Combining the bitmap mode instruction with LMS attribute makes it possible to set the screen memory address freely within the 64K address space *independently for each display line*. In other words, the screen memory does not have to be a solid memory area scanned sequentially towards higher addresses - only a single line must cover a continuous area of memory, usually consisting of 32, 40 or 48 bytes.

Limitations

The vertical extent of the entire display can vary between 0 and 240 scanlines - this depends on what number of lines the ANTIC is programmed to display according to the display list. The horizontal width of the screen can be programmed to be 256, 320 or 384 pixels wide (in that last mode, only 352 pixels are really visible).

Although ANTIC's program counter is 16-bit, only 10 bits are changed during normal (i.e. sequential) execution of the display list. This means the *display list* need a JMP (Jump) instruction to cross a 1K boundary. This is not a serious limitation, because the size of a single display list usually varies from 32 to 202 bytes, and virtually never exceeds 720 bytes. Since it can be located anywhere in the memory, there is no trouble in finding a place for it, that does not cross a 1K boundary.

Also the Memory Scan Register, a register addressing the data stored in the screen memory, is 16-bit, but only 12 bits are changed when ANTIC is sequentially scanning the video memory. Thus an LMS (Load Memory Scan) instruction is needed for data crossing a 4K boundary. High resolution graphic modes usually need more than one LMS instruction within the display list to be displayed properly.

The character generator can be located anywhere in the memory, but, depending on the display mode used, it has to be aligned to a 512-byte or a 1K boundary.

Display list example program

Type the following program at the BASIC prompt (typically the word "READY"), terminating each line by pressing the Enter (or Return) key. Run it by typing "RUN" and pressing Enter. When the program has finished running, type "PRINT USR(30720)" and hit Enter. The program should display a green pixel in BASIC graphics mode 3, and lines in BASIC mode 0.

```
10 FOR N=30720 TO 30829
20 READ D
30 POKE N,D
40 NEXT N
50 DATA 169,112,141,0,128,141,1,128
60 DATA 141,2,128,169,66,141,3,128
70 DATA 169,0,141,4,128,169,130
80 DATA 141,5,128,169,2,141,6,128
90 DATA 141,7,128,169,72,141,8,128
100 DATA 169,0,141,9,128,169,140
110 DATA 141,10,128,169,8,141,11,128
120 DATA 141,12,128,141,13,128,141,14
130 DATA 128,169,65,141,15,128,169,0
140 DATA 141,16,128,169,128,141,17,128
150 DATA 169,0,141,48,2,169,128,141
160 DATA 5,141,6,150,169,150,141,7,150
170 DATA 5,141,6,150,169,150,141,7,150
180 DATA 169,2,141,11,140,5,150
```

Chapter 6

Commodore AA+ Chipset

This was the last classic Amiga compatible chipset that Commodore announced in 1992. They planned to release it in 1994 for low end Amiga computers along with AAA.

History

In 1991 Commodore realized that AAA cost was going high, so they postponed it until 1994 and hesitantly designed AGA and released it in 1992 to keep up with the competitors.

Commodore was convinced that even in 1994 AAA systems with their 4 custom chips (6 chips in the 64-bit systems) would be very expensive to use in a low price A1200 like computer or CD32 like console. So unlike Commodore's habit of designing one custom chip for both high end and low end computers to save development costs, Commodore decided to design two custom chips: AAA for the high end computers and AA+ for the low end ones.

In January 1993 at Devcon in Orlando, Florida, Lew Eggebrecht Commodore VP of Engineering at the time stated the following:

"AA+ will be a more profitable version of AA with all the things we wished we'd got in but didn't have time. We have a list of all the problems we currently have at the low end. The serial port, we can't read high density floppies, there isn't enough band width to do 72 Hz screens plus there are no chunky pixel modes for rendering. We listed all those and said, "OK let's go out and fix them as quickly as we can", so AA+ is an extension, not radically new architecture. We're doing the best that we can, taking advantage of advances in technology, significantly reducing the cost and that's the goal."

According to Dave Haynie AA+ only existed on papers and the actual design never started due to Commodore's lack of money at the time. Like AAA and Hombre, Commodore was planning to use AA+ with the Acutiator system that Dave Haynie designed.

A few years later Access Innovations adopted the AA+ name for its BoXeR AGA compatible chipset.

Compatibility

Unlike AAA which was a radical design from ECS and did not support AGA registers, AA+ was built on the foundations of AGA and would be 100% AGA compatible.

Operating System

AA+ systems would be shipped with forthcoming Commodore AmigaOS 4 which added RTG support for chunky pixels.

Chips

To keep costs down, Amiga custom chips would be reduced from 3 (OCS,ECS,AGA) to only two. AA+ would feature two custom chips with 160 - 280 pin packages and each chip would have 100,000 transistors on it. In comparison, AGA Alice has 80,000 while ECS has a total of 60,000. On the other hand, AAA, with its 4 chips, would have a total count of 750,000 transistors, and more than 1,000,000 in its 6 chips 64-bit configuration.

CPU

Commodore stated that AA+ was designed to support ALL 32 bit 680x0 CPUs. For Chunky support,low end systems (A1400) would most likely feature a 68020 with full 32 bit memory addressing (i.e. not 68020EC) or even 68030EC which could handle RTG drivers easiy. Commodore did not add Chunky pixels to AGA because RTG -which was to be part of Commodore forthcoming AmigaOS4- required at least 68020 (not 68020EC in A1200) with 4MB memory at least, while the standard 599$ A1200 only had 2MB and 68020EC CPU.

Memory

AA+ had 8x memory bandwidth over ECS by using 128-bit long memory bus bursts like AAA. Maximum Chip RAM would increase from AGA 2MB to 8MB.

AA+ Systems would use 60 ns DRAM, to add Chunky, AA+ systems would need at least 4MB as a standard to support RTG, most likely A1200 like systems (A1400?) would be shipped with 8 MB which was the standard in 1994 for low end PCs.

Graphics

With 57 MHz pixel clock, AA+ could display progressive rock steady 800 x 600 at 72 Hz resolution in 256 colors easily, or even interlaced 1024 x 768 screens. Perhaps the most significant advancement was the addition of 8 and 16 bit Chunky mode with 24 bit Hybrid mode (3 planes of 8 bit Chunks),although the max resolution for 16 bit pixels would be 640 x 480.

Blitter

A 2X blitter performance over AGA/ECS was promised; compared to AGA 32 bit blitter, AAA blitter had 8x gain in performance because the blitter improved from 16 bit to 32 bit and could use 128 bits long bursts which the AGA blitter couldn't.

The 2x performance increase could be gained by updating AGA blitter from 16 bit to 32 bit without using AAA blitter long bursts which will make cost higher, Commodore never declared whether AA+ would include a 32-bit blitter or utilize 128 bit long bursts like AAA did.

Possibly, the AA+ blitter would stay 16 bit like the AGA one to keep the price down and stay compatible with AGA registers without emulating it like the costly AAA did. Of course Commodore had to add workaround functions to the blitter to improve its speed in 16 bit mode.

A 16 bit performance increase may have also come as a result of using Chunky pixels which the new blitter could handle, AAA blitter was improved to be pixel addressing rather than plane addressing with masks, modulos, and shifts. A new Clip-Rect Line Drawing function was added to AAA blitter for better GUI performance. It is most likely that Commodore would try to copycat some of these new bitter functions to AA+.

By adding Cookie-cut technology to 16 bit AA+ blitter like the Natami team, Commodore made blitting faster by a factor of 2X in comparison to similar planar modes without being forced to use masks like planar modes, just like using hardware sprites in Amiga.

Sound

When asked, Lew Eggebrecht VP of Engineering at Commodore stated that AA+ will support 16 bit sound samples, but it is unclear whether this support would be added by adding a DSP chip, or by improving Paula to something better like AAA, although Lew Eggebrecht stated once that DSP will be integrated in future low end and high end computers for CD-ROM sound support.

More than 4 channels support like AAA was unconfirmed, sample rates remains unknown but most likely it would be synchronized to AGA Paula to keep it compatible with cycle exact software.

Floppy

AA+ would fully support 1.76 HD floppy drives at full speed without using workaround kludges like former Amigas, former HD floppy drives were essentially made to spin at half the speed of standard HD floppy drives to cope with Paula's lack of support of higher bit rates.

Serial Port

AA+ would have two four-byte buffered FIFO serial UARTs like the AAA.

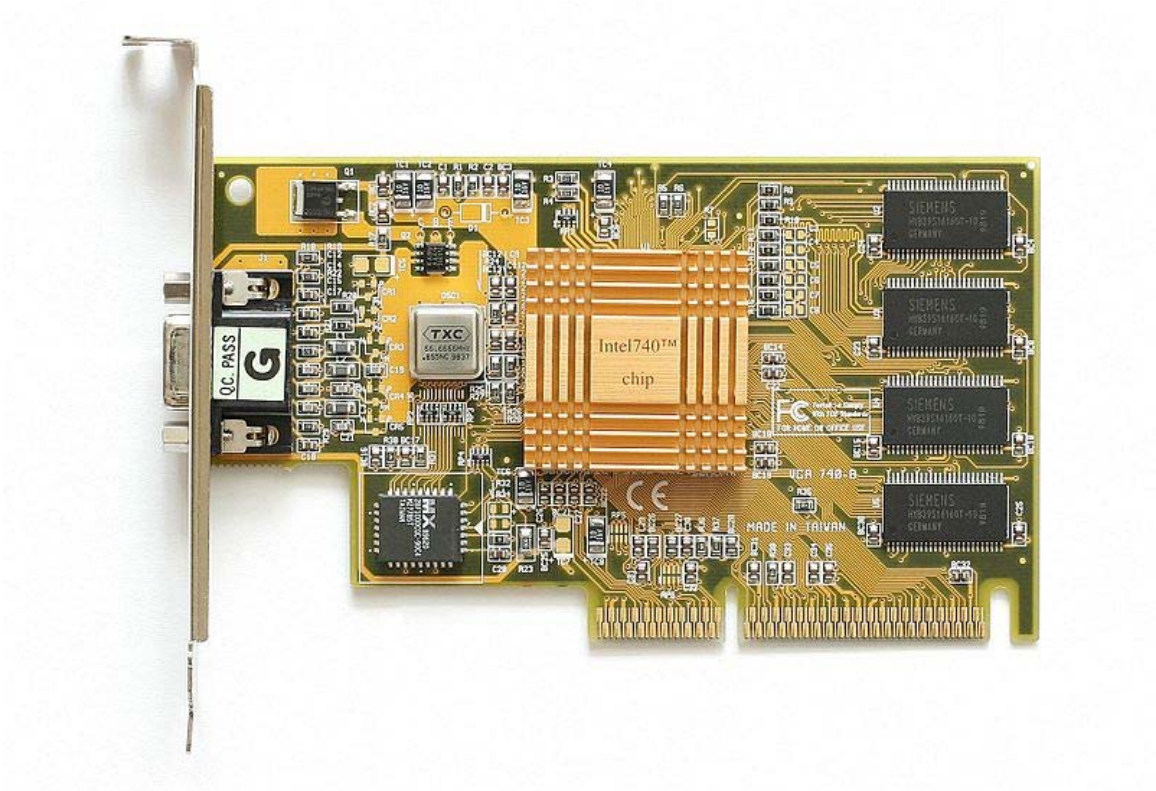
Specifications

- Two Chips with 100k Transistor each.
- synchronous to video clock.
- 160 - 280 pin packages.
- 32 bit DRAM 60ns Page Mode Chip Memory.
- 57 MHz pixel clock.
- ECS & AGA registers compatible.
- 4MB 4Mbps Floppy Controller with Hardware CRC floppy drives using standard technology.
- support for ALL 32 bit 680x0 CPUs.
- 8x memory bandwidth increase over ECS.
- 2x Blitter Performance (gets twice as many clocks as on AGA).
- Rock steady 800x600x8bit Non-Interlace 72Hz Refresh Rate, Larger screens at lower Refresh Rates.
- 16 bit True Color mode (although recent developments with the completion of the first cycle of chipset design indicate that this will actually be a 24 bit True Color Mode)
- FIFO serial ports with large buffer.
- Increased Chip Ram limit up to 8Mb. With 23 bit memory addressing registers instead of AGA's 21 bit registers.

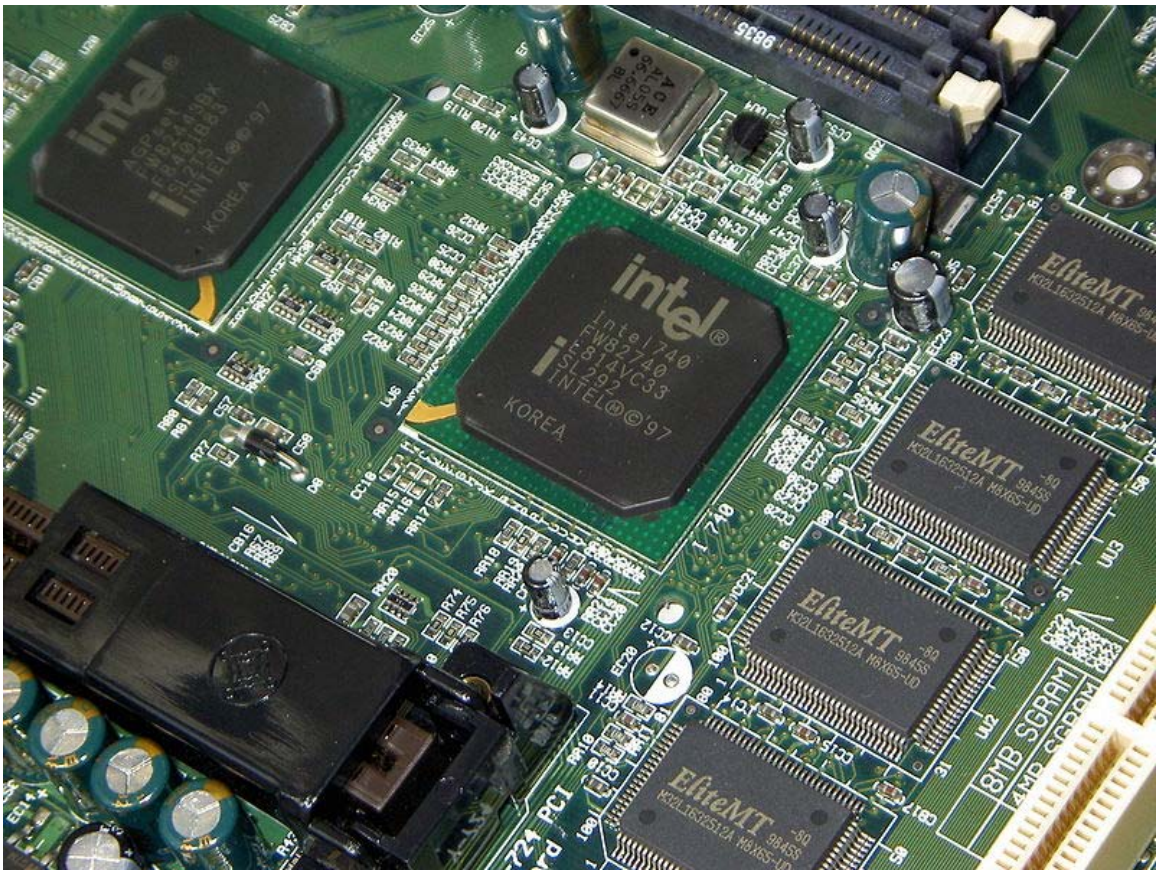
Chapter 7

Intel740 and Picture Processing Unit

Intel740



An Intel740 AGP card.



A mainboard with Intel i740.

The **Intel740**, or **i740**, is a graphics processing unit using an AGP interface released by Intel in 1998. Intel was hoping to use the i740 to popularize the AGP port, while most graphics vendors were still using PCI. Released with enormous fanfare, the i740 proved to have disappointing real-world performance, and sank from view after only a few months on the market. Some of its technology lived on in the Intel GMA systems that continue to be sold to this day.

History

The i740 has a long and storied history that starts at GE Aerospace as part of their flight simulation systems, notable for their construction of the Project Apollo "Visual Docking Simulator" that was used to train Apollo to dock the Command Module and Lunar Module. GE sold their aerospace interests to Martin Marietta in 1992, a victim of Jack Welsh's aggressive downsizing of GE. In 1995, Martin Marietta merged with Lockheed to form Lockheed Martin.

In January 1995, Lockheed Martin re-organized their divisions and formed Real3D in order to bring their 3D experience to the civilian market. Real3D had an early brush with success, providing chipsets and overall design to Sega, who used it in a number of arcade

game boards, the Model 2 and Model 3. They also formed a joint project with Intel and Chips and Technologies (later purchased by Intel) to produce 3D accelerators for the PC market, under the code name "Auburn".

Auburn was designed specifically to take advantage of (and promote) the use of AGP interface, during the time when many competing 3D accelerators (notably, 3dfx Voodoo Graphics) still used the PCI connection. A unique characteristic, which set the AGP version of the card apart from other similar devices on the market, was the use of on-board memory exclusively for the display frame buffer, with all textures being kept in the computer system's main RAM. At the time, most accelerators used the CPU for triangle setup and geometry calculations, then handed the data off to the card to apply texture mapping and bilinear filtering. By leaving this data in main memory, and giving the graphics card a high-speed channel to the data, performance could be improved while also reducing the total amount of memory in the system.

In the lead-up to the i740's introduction, the press widely commented that it would drive all of the smaller vendors from the market. As the introduction approached, rumors of poor performance started circulating. In spite of this, pundits continued to agree that its release would have enormous effects on the market. Peter Glaskowsky noted that "Very few of the manufacturers have the access to the [manufacturing plants] that Intel does, S3 could be the big loser here--it doesn't sell to the performance market. Intel has the resources to beat S3 on those terms and they have the performance". The i740 was released in January 1998, at $34.50 in large quantities. A number of companies had cards to introduce on that day.

The AGP Texture concept soon proved to be a tremendous error in design, because the card had to constantly access the textures over a channel that was upwards of eight times slower than RAM placed on the graphics card itself. Although AGP did indeed improve performance of moving geometry, this was wiped away by the growing use of textures, which were much larger. In real-world use it proved to be much slower than existing solutions like the Voodoo2, and was only able to hold its own with slower 2D/3D cards like the Nvidia RIVA 128. The release of the Nvidia RIVA TNT removed even that advantage. By the end of the year it went largely unmentioned in benchmarks against newer 3D products, already forgotten.

In August 1999, after less than 18 months on the market, Intel withdrew the i740 from the market. In September Lockheed announced a "customer-focused organizational realignment" that shed many of its divisions, and then closed Real3D on 1 October 1999 (following Calcomp in late 1998). Intel purchased the company's intellectual property, part of a series on ongoing lawsuits, but laid off the remaining skeleton staff. Some staff were picked up as contractors within Intel, while a majority were hired by ATI and moved to a new office.

Intel also sold the i740 to 3rd party companies, and some PCI versions of the accelerator also were made. They used an AGP-to-PCI bridge chip and had more on-board memory

for storing textures locally on the card, and were actually faster than their AGP counterparts in some performance tests.

Picture Processing Unit

NES PPU (Ricoh RP2C07) in a PAL NES.

The **PPU (Picture Processing Unit)**, more specifically known as Ricoh RP2C02 (NTSC version) / RP2C07 (PAL version), is the microprocessor in the Nintendo Entertainment System responsible for generating video signals from graphic data stored in memory.

The chip is known for its effective use of memory, using very little memory to store graphical data. It was rather advanced for its time when the Famicom (Japanese version of the Nintendo Entertainment System) was released, sporting full sprite support, movable backgrounds, and many colors on screen at the same time. To compete with other video game systems, like the technically superior Sega Master System, Nintendo also extended the PPU's technical capabilities through the use of mappers, which were placed on the game cartridge. The mappers added more memory or could bank switch data into the PPU's address space, making it possible to create more advanced graphics, using more colors and bigger tile sets.

Key features

- 2KB external RAM for graphics information storage
- 256 bytes for sprite data storage
- 32 bytes for palette storage
- 8 × 8 or 8 × 16 (selectable) sized tiles
- Two 4KB tile sets with space for 256 tiles each
- Up to 64 sprites (movable objects) on screen simultaneously (only 8 visible per scan line)
- 25 colors simultaneously (although more colors are possible using programming tricks) from a hardware color palette of 64 colors
- Picture resolution of 256 × 240 pixels (fully visible on PAL, but cropped to 256 × 224 on most NTSC television sets)

Technical information

The PPU is controlled via eight registers visible in the CPU's address space in the addresses $2000 through $2007. All data and information is passed to the PPU through these, except the raw tile data (there are exceptions, as some games had RAM instead of ROM to store the tile data, and the tiles had to be written each time), which is hardwired to the PPU's address space. The PPU uses the tile graphics data together with information stored by the program in the PPU's RAM, such as color and position, to render the final graphical output to the screen.

The lowest graphical components the PPU operates with are tiles, which are blocks of 8×8 or 8×16 pixels. The tiles are stored in a ROM chip on the game cartridge. The tiles are the basic building blocks, used to create larger moving objects, or large static backgrounds.

Essentially, the PPU supports two different kinds of objects: movable (sprites) and non-movable (background). Both kind of objects are basically a tile, and moreover a sprite and background object can use the same tile. The difference is that a tile used as a sprite can move around, whereas a tile used as a background cannot.

Sprite data is stored in a special memory called the "Sprite-RAM" or "SPR-RAM" for short, which is a 256-byte memory built into the PPU core. The data stored here is 4 bytes; the position, color and tile, for each of the 64 sprites. This data is used by the PPU to place the sprite when it renders the frame. Background objects, however, are stored in a much less exclusive way, which is more like the way characters are stored in text mode on PCs. A background is defined by a simple data structure called a nametable, which is essentially a two dimensional array. The integer value in each array slot corresponds to a tile number, and the index values of this slot correspond to the tile's intended x/y position on screen. The PPU has, without the use of memory mappers, two nametables, so smooth scrolling between backgrounds is possible.

A color palette must be defined in order to show graphics on the screen. It is stored in a separate 32 byte location in RAM, known as "palette-RAM". Each entry here picks a color from the hardware color palette, which are the predefined colors to choose between. 16 colors can be chosen for sprites, and 16 colors for backgrounds. However, bytes 4, 8 and 12 of the sprite palette, and bytes 0, 4, 8, and 12 of the background palette, are not in use by the PPU. Therefore, the number of actually usable colors is reduced to 25 instead of 32. The first byte of the sprite palette also defines the global background color for both sprites and the background.

Chapter 8

Super FX and RAMDAC

Super FX

Super FX-rendered 3D polygon graphics in the SNES game *Star Fox*

The **Super FX** is a coprocessor chip used in select Super Nintendo Entertainment System (SNES) video game cartridges. This custom-made RISC processor was typically programmed to act like a graphics accelerator chip that would draw polygons to a frame buffer in the RAM that sat adjacent to it. For those games, the data in this frame buffer were periodically transferred to the main video memory inside of the console using DMA in order to show up on the television display.

Development history

Some early cartridges of *Star Fox* shipped with a Super FX processor that was marked "MARIO CHIP 1"

The Super FX chip was originally called the MARIO Chip 1 (**M**athematical, **A**rgonaut, **R**otation & **I**/**O**) and was designed by Argonaut Games, who also co-developed (with Nintendo) the 3D space scrolling shooter video game *Star Fox* to demonstrate the additional polygon rendering capabilities the chip brought to the SNES. The hardware designers of the chip were Ben Cheese (formerly of Sinclair and Flare Technology) with Rob Macaulay and James Hakewill. The graphics were considered revolutionary at the time. Although *Star Fox* was capable of rendering polygons, the number of polygons was in the hundreds as opposed to the millions of today's games. *Star Fox* used scaling bitmaps for lasers, asteroids, and other obstacles, but other objects such as ships were rendered with polygons.

In addition to rendering polygons, the chip was also used to assist the SNES in rendering advanced 2D effects. *Super Mario World 2: Yoshi's Island* used it for advanced graphics effects like sprite scaling and stretching, huge sprites that allowed for boss characters to take up the whole screen, and multiple foreground and background parallax layers to give

a greater illusion of depth. While the Super FX is primarily known for graphical enhancements, it's also been used to assist the S-CPU in processing game logic. Yoshi's Island offloads all sprite logic to the Super FX, in addition to the decompression of graphics.

Game cartridges that contain a Super FX chip have additional contacts at the bottom of the cartridge that connect to the extra slots in the cartridge port that were not normally used. Cartridge adapters such as cheat devices made before the release of Super FX games, such as the Game Genie, did not have a connection to these previously unused slots. This meant that Super FX games could not be plugged into these devices. Because of higher manufacturing costs, games that included additional hardware such as the Super FX chips retailed at a higher MSRP than most SNES games.

Versions

GSU-1 chip on Vortex, Wild Trax (Japan)

The first version of the chip, GSU-1, commonly called the Super FX, is clocked with a 21 MHz signal, but an internal clock speed divider halved it to 10.5 MHz. Some early cartridges of *StarFox* shipped with a version of this chip that was marked "MARIO CHIP

1." Later on, the design was revised to become the GSU-2, known as the Super FX 2. Unlike earlier chips, this version was able to reach 21 MHz.

All versions of the Super FX chip are functionally compatible in terms of their instruction set. The differences arise in how they are packaged, their pinout, and their internal clock speed. As a result of changing the package when creating the GSU-2, more external pins were available and assigned for addressing—as a result a larger amount of external ROM or RAM can be accessed.

The technology behind the SuperFX chip would later become the ARC (**A**rgonaut **R**ISC **C**ore) embedded microprocessor.

List of games that use Super FX chips

Super FX 2 chip on *Super Mario World 2: Yoshi's Island*

Games released with the Super FX chip

- *Dirt Racer* (PAL)
- *Dirt Trax FX*
- *Star Fox* (North America/Japan) / *Star Wing* (PAL)
- *Stunt Race FX* (North America/PAL) / *Wild Trax* (Japan)
- *Vortex*

Games released with the Super FX 2 chip

- *Doom*
- *Super Mario World 2: Yoshi's Island*
- *Winter Gold / FX Skiing*

Cancelled/unreleased games

- *Comanche*
- *Powerslide FX*
- *FX Fighter* (released for PC)
- *Star Fox 2* (elements used in Super Mario 64, Star Fox 64 and Star Fox Command)
- *Super Mario FX* (became Super Mario 64)

RAMDAC

Random Access Memory Digital-to-Analog Converter (RAMDAC) is a combination of three fast DACs with a small SRAM used in computer graphics display adapters to store the color palette and to generate the analog signals (usually a voltage amplitude) to drive a colour monitor. The logical colour number from the display memory is fed into the address inputs of the SRAM to select a palette entry to appear on the data output of the SRAM. This entry is composed of three separate values corresponding to the three components (red, green, and blue) of the desired physical colour. Each component value is fed to a separate DAC, whose analog output goes to the monitor, and ultimately to one of its three electron guns (or equivalent in non-CRT displays).

DAC word lengths range usually from 6 to 10 bits. The SRAM's word length is three times the DAC's word length. The SRAM acts as a color lookup table (CLUT). It usually has 256 entries (and thus an 8-bit address). If the DAC's word length is also 8 bits, we have a 256 x 24-bit SRAM which allows a selection of 256 out of 16777216 (16,7 million) possible colours for the display. The contents of this SRAM can be altered when no pixel needs to be generated for transmission to the display. A synchronization pulse is required to maintain vertical picture stability. Therefore a vertical blanking pulse is generated for every frame. This vertical blanking pulse is not visible on the display, nor is any pixel sent. Therefore the D/A is idle and can allow the user to modify the SRAM color lookup table.

The SRAM can usually be bypassed and the DACs can be fed directly by display data, for Truecolor modes. In fact this has become very much the normal mode of operation of a RAMDAC since the mid-1990s, so the programmable palette is mostly retained only as a legacy feature to ensure compatibility with old software. In many newer graphics cards, the RAMDAC can be clocked much faster in true color modes, when the SRAM is not used.

For a quick estimation on the pixel clock for a given output, you can do:

> horizontal pixels x vertical lines x 1.4 (for blankings) x refresh rate
> (based on VESA's GTF calculation sheet)

Usually the RAMDAC rating has to be (quite a bit) better than the pixel clock to produce sharp edges.

As of 2006, the DAC of a modern graphics card runs at a clock rate of 400 MHz. However, video cards based on the XGI Volari XP10 run at 420 MHz DAC. The highest documented DAC frequency ever achieved on a production video card for the PC platform is 550 MHz, set by BarcoMed 5MP2 Aura 76Hz by Barco. However, DOME Md8/PCI supports up to 2560x3200@68Hz over a single output, which would have 810 MHz pixel clock rate under VESA GTF calculation.

History

IMS G171 RAMDAC on the VGA board

The term "RAMDAC" did not enter into common PC-terminology until IBM introduced the IBM VGA display adapter in 1987. The IBM VGA adapter used the INMOS G171 RAMDAC. The INMOS VGA RAMDAC was a separate chip, featured a 256-color (8-bit CLUT) display from a palette of 262,144 possible values, and supported pixel-rates up to ~30MPixel/sec.

As clone manufacturers copied IBM VGA hardware, they also copied the INMOS VGA RAMDAC. Advances in semiconductor manufacturing and PC processing-power allowed RAMDACs to add "direct-color" operation, which is a mode of operation that allows the SVGA-controller to pass a pixel's color-value directly to the DAC-inputs, thereby bypassing the RAM lookup-table. Another innovation was Edsun's CEGDAC, which featured hardware-assisted anti-aliasing for line/vector draw-operations.

By the early 1990s, the PC chip-industry had advanced to the point where RAMDACs were integrated into the display controller chip, thus reducing the number of discrete chips and the cost of video cards. Consequently, the market for standalone RAMDACs

disappeared. Today, RAMDACs are still manufactured and sold for niche applications, but in obviously limited quantity.

In modern PCs, the RAMDAC(s) are integrated into the display controller chip, which itself may be mounted on an add-in-board or integrated into the motherboard core-logic chipset. The original purpose of the RAMDAC, to provide a CLUT-based display-mode, is rarely used, having been supplanted by true-color display-modes. However, many CAD and video editing applications use hardware overlay, combined with the programmable palette, to ensure the user interface does not disrupt the rendering of editing window.

As the use of DVI, HDMI, and other digital interface technology becomes increasingly mainstream, the "DAC" portion of the RAMDAC will likely become obsolete. Digital interfaces use a TMDS module.

Chapter 9

Television Interface Adaptor

Atari 2600

The **Television Interface Adaptor** (**TIA**) is the custom chip that is the heart of the Atari 2600 game console, generating the screen display, sound effects, and reading input controllers. Its design was widely effected by an attempt to reduce the amount of RAM needed to operate the display. The resulting design is notoriously difficult to program, but at the same time offers flexibility well beyond the capabilities of contemporary designs. It remains a key reason that the 2600 maintains a strong homebrew following.

Development of the TIA was led by Jay Miner, later of Amiga fame. Atari later expanded on the design of the TIA for the Atari 400 and Atari 800 with the CTIA and GTIA chips.

Design

Background

Early video games generally used two distinct types of graphics, the "players" which were controlled by the player or the computer (today known as sprites), and the "playfield", or background graphics normally drawn under the players.

The conventional way to draw the playfield is to use a bitmap held in a framebuffer, which TIA doesn't provide one. Each memory location in the framebuffer holds a value that describes pixels on the screen. The display circuitry reads these values out of the buffer and uses it to generate an analog signal for display on a computer monitor. The mapping of the memory to screen locations, or pixels was often limited by the display hardware. On a conventional NTSC color television, maximum resolutions generally fell between 256 and 320 pixels per line, and 192 to 240 lines per screen.

TIA differs partly in that the image on the screen is composed by manipulating five movable graphic objects (2 players, 2 missiles and 1 ball) and a static playfield object on every scan line as they are directly written to their respective registers, unlike as it is in a framebuffer mapped model (Wright, p. 3). Horizontal resolution is not uniform, whose size depends on the particular graphics object; wherein the smallest unit of pixel corresponds to 1 color clock cycle of the chip, of which there are 160 visible ones on a line (p. 4). Playfield object consists of two and a half bytes register (20 bits wide), which can be reflected symmetrically to the right half of the screen (40 bits in total, each bits being 4 color cycles wide) and has only one color register. Two player objects are 8 bits wide registers which can be positioned horizontally with intervals of 15 color cycles (setting the position takes 5 machine cycles, corresponding to 15 color cycles), and fine tuned by 15 pixels precision (-7 to the left, +8 to the right) in order to fill that 15 cycles gaps (p. 8). All movable objects except the ball can be resized up to 8 clock cycles per bits and can be cloned up to 3 copies. Vertical resolution, however, is independent of the chip's abilities and is defined as 192 visible scanline for NTSC, and 228 for PAL television signals (p. 18).

RAM-less design

At the time the 2600 was being designed, RAM memory was extremely expensive, measured in dollars (or cents) per byte. A typical 320 by 200 pixel display with even a single bit per pixel would require 8000 bytes of memory to store the framebuffer. This would not be suitable for a platform that aimed to cost only a few hundred dollars. Even dramatic reductions in the resolution would not reduce the cost of memory to reasonable levels. Instead, the design team decided to remove the memory-based framebuffer entirely.

On the 2600, the display is held in a series of memory registers that describe the output for a single line on the display. The playfield is defined by 20 bits of pixel data, each describing a single pixel, providing a resolution of only 20 pixels per line. However, the

"line" described only the left half of the screen, horizontally. Another register allowed the pattern to be stretched out to fill the screen, or either copied or mirrored on the right half of the screen, increasing the effective resolution to 40-pixels, but requiring the playfield to be symmetric. The color that was drawn if the bit was a 1 or a 0 was selected from a pre-defined palette of up to 128 colors (see below) and held in other registers.

The TIA also supported five separate graphics objects consisting of:

- Two 8-pixel horizontal lines which make up the 'sprites' Player 1 and Player 2. These are single color, can be stretched by a factor of 2 or 4, and can be duplicated or triplicated.
- A 'ball' - a horizontal line that is the same color as the playfield. It can be one, two, four, or eight pixels wide.
- Two 'missiles' - another horizontal line that is the same color as its respective player. It can be one, two, four, or eight pixels wide.

The TIA has hardware collision detection for all of these objects and stores a bitmap of collisions, that are typically read during the VBLANK period. Registers in the TIA allow the programmer to control the positioning of the graphical objects and their color.

The TIA also provides two channels of one-bit sound. Each channel provides for 32 pitch values and 16 possible bit sequences. There is a 4 bit volume control.

Lastly, the TIA has inputs for reading up to four analog paddle controllers using potentiometers and for two joystick triggers.

Drawing the display

As the registers held data for only a single line of the display, creating a full screen required the game program to update the registers on the fly, a process known as "racing the beam".

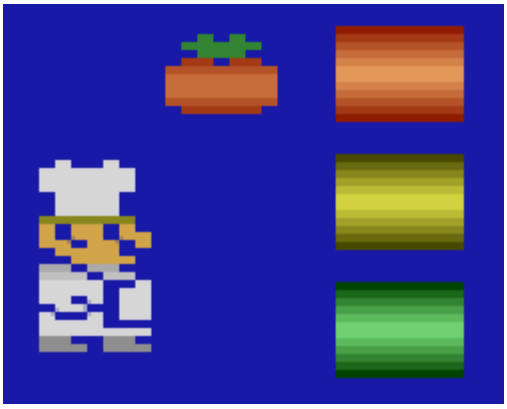
To start the process, the game program running on the MOS Technology 6502-based CPU would load the TIA's registers with the data needed to draw the first line of the display. The TIA would then wait until the television was ready to draw the line (under the command of the TIA's associated analog hardware) and read out the registers to produce a signal for that line. When the television was busy getting ready to draw the next line, the 6502 would quickly change the TIA's registers to the data needed for the next line. This process continued down the screen.

This process was made more difficult by the fact that the MOS Technology 6507 used in the 2600 was a pin-reduced version of the 6502 which had no support for hardware interrupts. Generally the analog side of the display system would generate an interrupt when it finished drawing a line and was getting ready for the next one (as in the Atari home computers, for instance). The interrupt would trigger the code needed up update the screen, and then return to the "main" program. The 6507 left these pins off of the CPU to

save money. This meant that the programmer had to carefully time their programs to run in the exact number of cycles needed for the various screen-related events to take place. Getting it wrong meant the screen would not draw properly, so in addition to the complexity of the drawing the display, the programmer also had to carefully count the number of cycles that their program took to run, moving code around as necessary to ensure it fit cleanly within the limited CPU budget. The program needed to do this was known as the "kernel", and was fantastically complex compared to other systems.

Given this complexity, early games using the system tended to be quite simple in layout, using the TIA to create simple symmetric playfields with players on top. This was the original intention of the system, to be able to run the handful of arcade games Atari had already produced with simple playfields, like *Tank* and *Pong*. In these cases the playfield data was typically laid out in the 2 kB ROM memory in the game cartridge. As each line used 20 bits of data, and there were 192 lines on an NTSC display, a display with a different layout on every line needed only 480 bytes (192 x 20 / 8) of the cartridge's 4 kB to hold a single hard-coded display. In this case the kernel simply advanced 20 bits through ROM for every line as the TIA advanced down the screen, a task that took only a few cycles of CPU time. This could be further reduced by using the same data for multiple lines, either doubling them vertically, or reading one way through the list for the top and then back the other way for the bottom, producing a vertically mirrored display of only 240 bytes.

A key advance was the licensing of *Space Invaders* for the platform, which required many more player graphics to draw the enemy aliens. The solution was to change the player data for every line as the image was being drawn, creating an apparent large number of players. *Space Invaders* was the platform's killer app, quadrupling the system's sales. Another advance was made by (partially) coding the display as CPU instructions instead of storing it as fixed data in ROM. *Adventure* used this concept to produce a wide variety of maps by combining different portions of the data in ROM, jumping back and forth through it during the screen drawing. This allowed the game to have 30 rooms, which would have otherwise required 14 kB of ROM.



Pressure Cooker changes the color every scanline. The chef and tomato use the "players", or "sprites" in more common terminology.

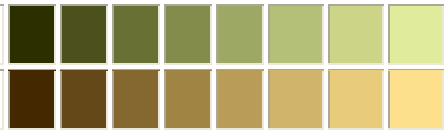
As programmers grew more accustomed to the odd timing needed to get things to work properly on-screen, they began to use the inherent flexibility in the TIA to greatly improve the displays. One common "trick" was to change the color registers that were used to draw the 1 and 0 states of the playfield, resulting in displays with rainbow-like effects - this became a hallmark of the platform. Later games even managed to get the timing to the point where they could safely change the colors while the line was being drawn. Similar effects could be used to modify the playfield mid-line to generate asymmetric patterns and repositioning and changing player sprites mid-screen in order to generate additional sprites on screen. These tricks allowed, in many cases, more graphically rich games than the hardware designers originally anticipated. Programming the TIA remains a real challenge which many homebrew programmers still enjoy today.

TIA Color Capabilities

The TIA uses different color palettes depending on the television signal format used. For NTSC format, a 128-color palette is provided, while only 104 colors are available for PAL. Additionally, the SECAM palette consists of only 8 colors.

NTSC palette

hue / luminance	0	2	4	6	8	10	12	14
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								

14	
15	

PAL palette

hue / luminance	0	2	4	6	8	10	12	14
0,1,14,15								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								

SECAM palette

0	2	4	6	8	10	12	14

Noise/Tone Generator (AUD0/1)

The TIA is capable of generating different flavors of pulse and noise out its two oscillators (or channels) AUD0 and AUD1. Each oscillator has a 5-bit frequency divider and a 4-bit audio control register which manipulates the waveform. There is also a 4-bit volume control register per channel.

Frequency Divider (AUDF0/1)

Frequencies are generated by taking 30kHz and dividing by the 5-bit value supplied. The result is a cheap frequency divider capable of detuned notes and the odd tuned frequency. The TIA is not a musical chip unless the composer works within the frequency limits or modulates between two detuned frequencies to create a vibrato tuned note.

Audio Control (AUDC0/1)

The Audio Control register generates and manipulates a pulse wave to create complex pulses or noise. The following table (with designed duplicates) explains how its tones are generated:

	D7	D6	D5	D4	D3	D2	D1	D0	
HEX									Type of noise or division:
0					0	0	0	0	Set to 1 (volume only)
1					0	0	0	1	4 bit poly
2					0	0	1	0	÷ 15 → 4 bit poly
3					0	0	1	1	5 bit poly → 4 bit poly
4					0	1	0	0	÷ 2
5					0	1	0	1	÷ 2
6					0	1	1	0	÷ 31
7					0	1	1	1	5 bit poly → ÷ 2
8					1	0	0	0	9-bit poly (white noise)
9					1	0	0	1	5-bit poly
A					1	0	1	0	÷ 31
B					1	0	1	1	Set last 4 bits to 1
C					1	1	0	0	÷ 6
D					1	1	0	1	÷ 6
E					1	1	1	0	÷ 93
F					1	1	1	1	5-bit poly ÷ 6

Chapter 10

Video Display Controller

A **Video Display Controller** or **VDC** is an integrated circuit which is the main component in a video signal generator, a device responsible for the production of a TV video signal in a computing or game system. Some VDCs also generate an Audio signal, but in that case it's not their main function.

VDCs were most often used in the old home-computers of the 80s, but also in some early video game systems.

The VDC is always the main component of the video signal generator logic, but sometimes there are also other supporting chips used, such as RAM to hold the pixel data, ROM to hold character fonts, or perhaps some discrete logic such as shift registers were necessary to build a complete system. In any case, it's the VDC's responsibility to generate the timing of the necessary video signals, such as the horizontal and vertical synchronisation signals, and the blanking interval signal.

Most often the VDC chip is completely integrated in the logic of the main computer system, (its video RAM appears in the memory map of the main CPU), but sometimes it functions as a coprocessor that can manipulate the video RAM contents independently

Video Display Controllers vs. Video Display Processors and Graphics processing units

The difference between a **VDC** and the more modern Video Display Processor (**VDP**) is not that the VDCs could not generate graphics, but they did not have the special hardware accelerators to create 2D and 3D images, while a typical 1990s VDP does have at least some form of hardware graphics acceleration. Also VDCs often had special hardware for the creation of "sprites", a function that in more modern VDP chips is done with the "Bit Blitter" using the "Bit blit" function.

One example of a typical Video Display Processor is the "VDP2 32-bit background and scroll plane video display processor" of the Sega Saturn. Another example is the **Advanced Graphics Architecture** (**AGA**) chip that was used for the improved graphics of the later generation Amiga computers.

This said, it is not completely clear when a "Video chip" is a "Video Display Controller" and when it is a "Video Display Processor". For example, the TMS9918 is sometimes called a "Video Display Controller" and sometimes a "Video Display Processor". In general however a "Video Display Processor" has some power to "Process" the contents of the Video RAM (filling an area of RAM for example), while a "Video Display Controller" only controls the timing of the Video synchronisation signals and the access to the Video RAM.

The Graphics processing unit (**GPU**) goes one step further than the VDP and normally also supports 3D functionality. It is the chip that is now used in modern personal computers.

Types of Video Display Controllers

Video Display controllers can be (arbitrarily) divided in several different types (here listed from simple to complex);

- **Video shifters**, or "Video shift register based systems" (there is no generally agreed upon name for these type of devices) are the most simple type of video controllers; they are, (directly or indirectly) responsible for the video timing signals, but they normally do not access the Video RAM directly. They get the video data from the main CPU, a byte at a time, and convert it to a serial bitstream (hence the technical name "Video shifter"). This serial data stream is then used, together with the synchronisation signals, to output a (colour) video signal. The main CPU needs to do the bulk of the work. Normally these chips only support a very low resolution Raster graphics mode.

- A **CRTC**, or Cathode Ray Tube Controller, generates the video timings and reads video data from a RAM attached to the CRTC, to output it via an external character generator ROM, (for text modes) or directly, (for high resolution graphics modes) to the video output shift register. Because the actual capabilities of the video generator depend to a large degree on the external logic, video generator based on a CRTC chip can have a wide range of capabilities. From very simple (text mode only) systems to very high resolution systems supporting a wide range of colours. Sprites however are normally not supported by these systems.

- **Video interface controllers** are much more complex than CRT controllers, and the external circuitry that is needed with a CRTC is embedded in the video controller chip. Sprites are often supported, as are (RAM based) character

generators and video RAM dedicated to colour attributes and pallette registers (Color lookup tables) for the high-resolution and/or text-modes.

- **Video coprocessors** have their own internal CPU dedicated to reading (and writing) their own video RAM, and converting the contents of this video RAM to a video signal. The main CPU can give commands to the coprocessor, for example to change the video modes or to manipulate the video ram contents. The video coprocessor also controls the (most often RAM based) character generator, the colour attribute RAM, Palette registers and the Spite logic (as long as these exist of course).

List of example VDCs

Examples of Video Display Controllers are:

Video shifters

- The RCA CDP1861 was a very simple chip, built in CMOS technology (which was unusual for the mid '70's) to complement the RCA 1802 microprocessor, it was mainly used in the COSMAC VIP. It could only support a very low resolution monochrome graphic mode.
- The "Television Interface Adapter (TIA) is the custom video chip that is the heart of the Atari 2600 games console, a very primitive chip that relied on the 6502 microprocessor to do most of the work, also was used to generate the audio.

CRT Controllers

- The Intel 8275 CRT controller was not used in any mainstream system, but was used in some S100 bus systems.
- The Motorola 6845 is a video address generator first introduced by Motorola and used for the Amstrad CPC, and the BBC Micro. It was later used for almost all the early video adapters for the PC, such as the MDA, CGA and EGA adapters. In all later VGA compatible adapters the function of the 6845 is reproduced inside the Video Chip, so in a sense all current IBM PC compatible PC's still incorporate the logic of the 6845 CRTC.

Video Interface Controllers

- The Signetics 2636 and 2637 are video controllers best known for their use in the Interton VC 4000 and Emerson Arcadia 2001 respectively.
- The MC6847 is a video display generator (VDG) first introduced by Motorola and used in the TRS-80 Color Computer, Dragon 32/64, Laser 200 and Acorn Atom among others.
- The MOS Technology 6560 (NTSC) and 6561 (PAL) are known as the Video Interface Controller (VIC) and used in the Commodore VIC-20.

- The MOS Technology 6567/8562/8564 (NTSC versions) and 6569/8565/8566 (PAL) were known as the VIC-II and were used in the Commodore 64.
- The MOS Technology 8563/8568 was used in the Commodore 128 to create the 80 column text mode, together with the normal VIC-II chip for the C64 compatible video modes.
- The MOS Technology 7360 Text Editing Device (TED) was used in the Commodore Plus/4, Commodore 16 and Commodore 116 computers and had an integrated audio capability.
- The Picture Processing Unit was a video co-processor designed by Ricoh for Nintendo's use in the Famicom and Nintendo Entertainment System. It was connected to 2048 bytes of dedicated video RAM, and had a dedicated address bus that allowed additional RAM or ROM to be accessed from the game cartridge. A scrollable playfield of 256×240 pixels was supported, along with a display list of 64 OBJs (sprites), of which 8 could be displayed per scanline.
- The TMS9918 is known as the Video Display Processor (VDP) and was first designed for the Texas Instruments TI-99/4, but was later also used in systems like the MSX (MSX-1), ColecoVision, Memotech MTX series, and for the Sega SG-1000 and SC-3000.
- The NEC μPD7220. Used in some high-end graphics boards for the IBM PC in the mid 80s, notably in products from Number 9 Computer Company.

Video Coprocessors

- The **ANTIC** (***A**lpha-**N**umeric **T**elevision **I**nterface **C**ircuit*) was an early video system chip used in the Atari 8-bit family of microcomputers. It could read a "Display list" with its own built in CPU and use this data to generate a complex video signal.
- The Yamaha V9938 is an improved version of the TMS9918, and was mainly used in the MSX2.
- The Yamaha V9958 is the Video Display Processor (VDP) mainly used in the MSX 2+ and MSX turbo R computers.

Alternatives to using a VDC chip

Note that many older home-computer did not use a VDP-chip, but built the whole video display controller from a lot of discrete logic chips, (examples are the Apple II, PET, and TRS-80). Because these methods are very flexible the video display generators could be very capable, (or extremely primitive, depending of the quality of the design) but also needed a lot of components.

Others used some form of early programmable logic arrays, (examples include the ZX Spectrum and ZX-81 systems and Elektronika BK-0010). These systems could build a very capable system with relatively few components, but the low transistor count of early programmable logic meant that the capabilities of these systems often were less impressive than those using video interface controllers or video coprocessors.

Later solutions

With Moore's law working, integrated circuits became more and more complex. The simple Video Display Controllers were slowly replaced by chips that had built-in video processing logic such as Blitters and other logic to manipulate the video RAM contents to do things like drawing lines, filling areas, or drawing fonts. Later chips also got special hardware to draw triangles to support 3D images, gained hardware Z-buffers and many other methods to accelerate the drawing of 3D pictures. Current Video generator chips almost always are "Graphics processing units" (**GPU's**) Entry-level PCs today commonly have the video display integrated into the motherboard chipset, which "steals" some system RAM for the display. The performance of such a system is not as good as one with dedicated video hardware.

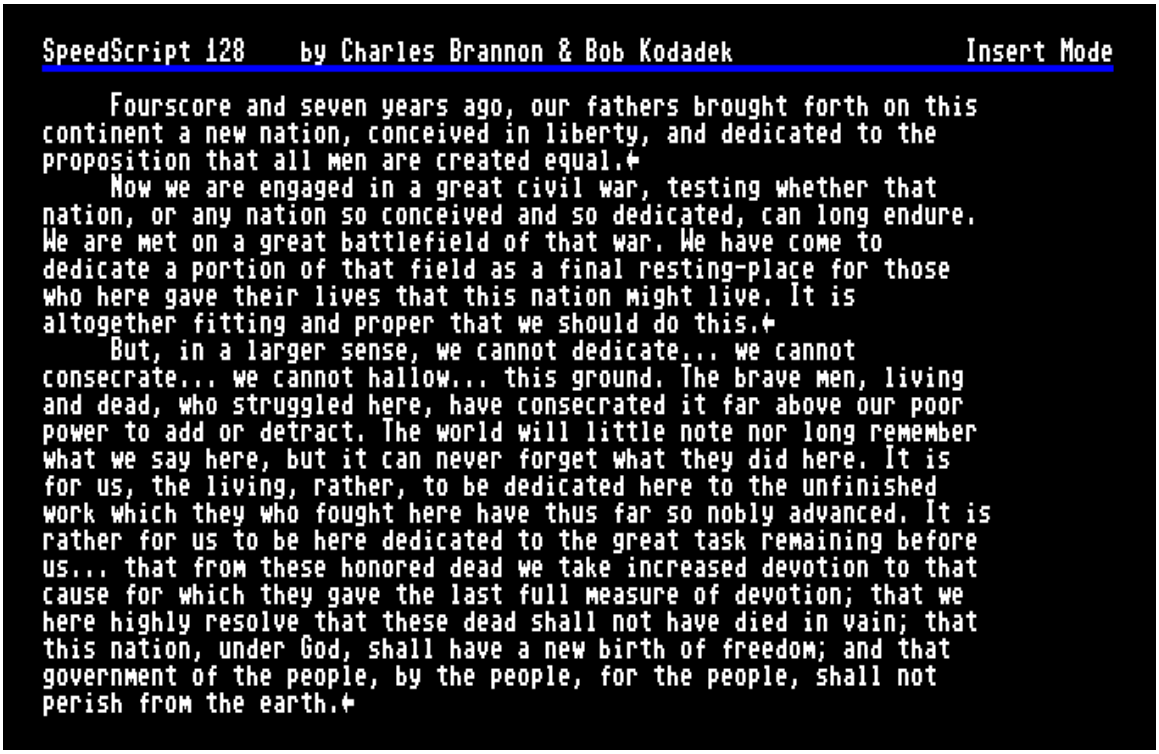
Chapter 11

MOS Technology

MOS Technology 8563

The **8563 Video Display Controller** (**VDC**) was an integrated circuit produced by MOS Technology. It was used in the Commodore 128 computer to generate an 80-column (640×200 pixel) RGB video display. The DCR models (as well as a few D-models) of the C128 used the later and more technically advanced 8568 [D]VDC controller.

History and characteristics



The VDC was designed with office suite applications in mind. Shown here is *SpeedScript 128*, a word processor.

This *Ultra Hi-Res* demo showcases the VDC's blitter capabilities with a simple 3D animation of a wire frame model of a cube.

Originally intended for a planned (but unreleased) UNIX-based business computer, Commodore designed the VDC into several prototype machines. Of these, only the Commodore 128 ever saw production. Unlike earlier MOS video chips such as the popular VIC-II, the VDC had dedicated video RAM, 16 kilobytes (upgradable to 64 kilobytes) in the original or "flat" C128 and 64 kilobytes in the C128DCR. This RAM was not directly accessible by the microprocessor.

The 8563 was more difficult to produce than most of the rest of the MOS Technology line, and initial yields were very low. Also, there were timing issues with the VDC that would cause indirect load and store operations on its registers to malfunction.

Officially, the VDC was a text-only chip, although a careful reading of the technical literature by MOS Technology that was given to the early C128 developers did indicate that a high-resolution bitmap mode was possible—it simply wasn't described in any detail. BASIC 7.0, the Commodore 128's built-in programming language, only supported high-resolution graphics in 40-column mode via the legacy VIC-II chip.

Shortly after the release of the C128 the VDC's bitmap mode was described in considerable detail in a Data Becker book (published in late 1985 in the USA by Abacus Software), and an assembly language program was provided by the authors, in which it was possible to set or clear any pixel or, using BASIC to perform the necessary calculations, generate bitmapped geometric shapes on the 80 column screen. In February

1986, less than a year after the Commodore 128's release, *RUN* magazine published "*Ultra Hi-Res Graphics*", an article describing the VDC's bitmapped mode and including a type-in program (written in 8502 assembly language) that extended BASIC 7.0's capabilities to support 640×200 high-resolution graphics using the 8563. Authors Lou Wallace and David Darus later developed the Ultra Hi-Res utility into a commercial package, *BASIC 8*. One of the most popular third-party utilities for the C128, this offered more advanced VDC high-resolution capabilities to a wide audience of programmers.

Commodore finally offered complete official documentation on the VDC in the *Commodore 128 Programmer's Reference Guide*. VDC bitmap modes were used extensively in the C128 version of the GEOS operating system.

The VDC lacked sprite capabilities, which limited its use in gaming applications. However, it did contain blitting capabilities to autonomously perform small block memory copies within its dedicated video RAM. While the VDC is performing such a copy, the system CPU can continue running code, provided no other VDC accesses are attempted before the copy is finished. These functions were used by the C128's screen editor ROM to rapidly scroll or clear screen sections.

Technical specifications

- RGBI output (RGB plus Intensity) compatible with IBM's CGA video standard.
- 16 or 64 kilobyte address space for display, character shape and display attribute memory (dedicated, separate from system memory).
- Up to 720×700 pixel video resolution in interlaced mode (maximum with 64 kilobyte video ram) . Other image sizes are possible, depending on programmer's needs, such as 640×200 non-interlaced, 640×400 interlaced, etc.
- 80×25 characters text resolution (C128 kernel default); other sizes such as 80×50 or 40×25 are possible.
- 8 colors at 2 intensities.

Programming

Addressing the VDC's internal registers and dedicated video memory must be accomplished by indirect means. First the program must tell the VDC which of its 37 internal registers is to be accessed. Next the program must wait until the VDC is ready for the access, after which a read or write on the selected internal register may be performed. The following code is typical of a register read:

```
        ldx #regnum       ;VDC register to access
        stx $d600         ;write to control register
loop    bit $d600         ;check bit 7 of status register
        bpl loop          ;VDC not ready
        lda $d601         ;read from VDC register
        ...
```

The following code is typical of a register write operation:

```
        ldx #regnum       ;VDC register to write to
        stx $d600         ;write to control register
loop    bit $d600         ;check bit 7 of status register
        bpl loop          ;VDC not ready
        sta $d601         ;write to VDC register
        ...
```

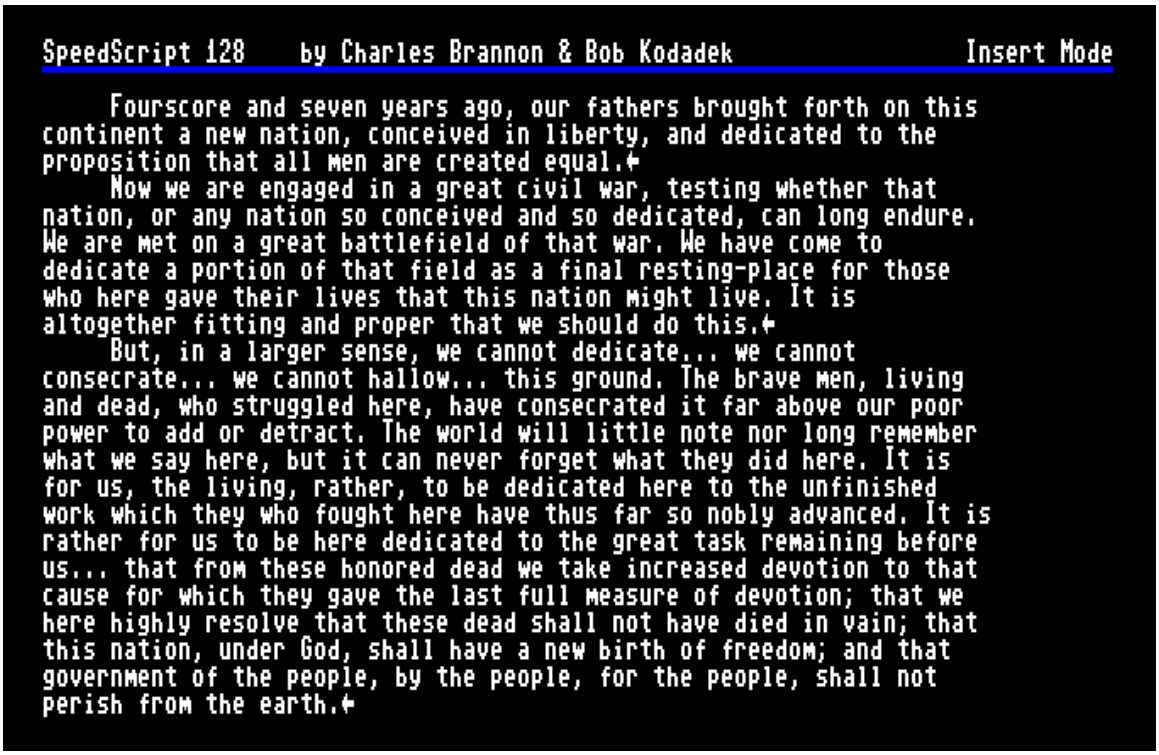
Owing to this somewhat cumbersome method of controlling the VDC, the maximum possible frame rate in bit-mapped mode is generally too slow for arcade-style action video games, in which bit-intensive manipulation of the display is required.

Register Listing

Register	Hexadecimal	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Description
0	$00	HT7	HT6	HT5	HT4	HT3	HT2	HT1	HT0	Horizontal Total
1	$01	HD7	HD6	HD5	HD4	HD3	HD2	HD1	HD0	Horizontal Displayed
2	$02	HP7	HP6	HP5	HP4	HP3	HP2	HP1	HP0	Horizontal Sync Position
3	$03	VW3	VW2	VW1	VW0	HW3	HW2	HW1	HW0	Vertical/Horizontal Sync Width
4	$04	VT7	VT6	VT5	VT4	VT3	VT2	VT1	VT0	Vertical Total
5	$05	--	--	--	VA4	VA3	VA2	VA1	VA0	Vertical Adjust
6	$06	VD7	VD6	VD5	VD4	VD3	VD2	VD1	VD0	Vertical Displayed
7	$07	VP7	VP6	VP5	VP4	VP3	VP2	VP1	VP0	Vertical Sync Position
8	$08	--	--	--	--	--	--	IM1	IM0	Interlace Mode
9	$09	--	--	--	--	CTV4	CTV3	CTV2	CTV1	Character Total Vertical
10	$0A	--	CM1	CM0	CS4	CS3	CS2	CS1	CS0	Cursor Mode, Start Scan
11	$0B	--	--	--	CE4	CE3	CE2	CE1	CE0	Cursor End Scan Line
12	$0C	DS15	DS14	DS13	DS12	DS11	DS10	DS9	DS8	Display Start Address High Byte
13	$0D	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0	Display Start Address Low Byte
14	$0E	CP15	CP14	CP13	CP12	CP11	CP10	CP9	CP8	Cursor Position High Byte
15	$0F	CP7	CP6	CP5	CP4	CP3	CP2	CP1	CP0	Cursor Position Low Byte
16	$10	LPV7	LPV6	LPV5	LPV4	LPV3	LPV2	LPV1	LPV0	Light Pen Vertical Position
17	$11	LPH7	LPH6	LPH5	LPH4	LPH3	LPH2	LPH1	LPH0	Light Pen Horizontal Position
18	$12	UA15	UA14	UA13	UA12	UA11	UA10	UA9	UA8	Update Address High Byte
19	$13	UA7	UA6	UA5	UA4	UA3	UA2	UA1	UA0	Update Address Low Byte
20	$14	AA15	AA14	AA13	AA12	AA11	AA10	AA9	AA8	Attribute Start Address High Byte
21	$15	AA7	AA6	AA5	AA4	AA3	AA2	AA1	AA0	Attribute Start Address Low Byte

22	$16	CTH3	CTH2	CTH1	CTH0	CDH3	CDH2	CDH1	CDH0	Character Total Horizontal, Character Display Horizontal
23	$17	--	--	--	CDV4	CDV3	CDV2	CDV1	CDV0	Character Display Vertical
24	$18	COPY	RVS	CBRATE	VSS4	VSS3	VSS2	VSS1	VSS0	Vertical Smooth Scrolling
25	$19	TEXT	ATR	SEMI	DBL	HSS3	HSS2	HSS1	HSS0	Horizontal Smooth Scrolling
26	$1A	FG3	FG2	FG1	FG0	BG3	BG2	BG1	BG0	Foreground/Background color
27	$1B	AI7	AI6	AI5	AI4	AI3	AI2	AI1	AI0	Address Increment per Row
28	$1C	CB15	CB14	CB13	RAM	--	--	--	--	Character Base Address
29	$1D	--	--	--	UL4	UL3	UL2	UL1	UL0	Underline Scan Line
30	$1E	WC7	WC6	WC5	WC4	WC3	WC2	WC1	WC0	Word Count
31	$1F	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0	Data Register
32	$20	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	Block Start Address High Byte
33	$21	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0	Block Start Address Low Byte
34	$22	DEB7	DEB6	DEB5	DEB4	DEB3	DEB2	DEB1	DEB0	Display Enable Begin
35	$23	DEE7	DEE6	DEE5	DEE4	DEE3	DEE2	DEE1	DEE0	Display Enable End
36	$24	--	--	--	--	DRR3	DRR2	DRR1	DRR0	DRAM Refresh Rate

MOS Technology 8568



The VDC was designed with office suite applications in mind. Shown here is *SpeedScript 128*, a word processor.

This *Ultra Hi-Res* demo showcases the VDC's blitter capabilities with a simple 3D animation of a wire frame model of a cube.

The **8568 Video Display Controller (VDC)** was MOS Technology's graphics processor responsible for the 80 column or RGBI display on D[CR] models of the Commodore 128 personal computer. In the Commodore 128 service manual, this part was referred to as the "80 column CRT controller." The 8568 embodied many of the features of the older 6545E monochrome CRT controller plus RGBI color.

The original ("flat") C128 used the 8563 video controller to generate the 80 column display. The 8568 was essentially an updated version of the 8563, combining the latter's functionality with glue logic that previously was implemented by discrete components in physical proximity to the 8563. Unlike the 8563, the 8568 included an unused active low interrupt request line (`/INTR`), which was asserted when the "ready" bit in the 8568's status register changed from 0 to 1. Reading the control register would automatically deassert `/INTR`. Owing to differences in pin assignments and circuit interfacing, the 8563 and 8568 are not electrically interchangeable.

The Commodore 128 had two video display modes, which were usually used singularly, but could be used simultaneously if the computer was connected to two compatible video monitors. The VIC-II chip, also found in the Commodore 64, was mapped directly into main memory—the video memory and CPUs (the 8502 and Z80A processors) shared a common 128 KB RAM, and the VIC-II control registers were accessed as memory locations (that is, they were memory mapped).

Unlike the VIC-II, the 8568 had its own local video RAM, 64K in the C-128DCR model (sold in North America) and, depending on the date of manufacture of the particular machine, either 16 or 64K in the C-128D model (marketed in Europe). Addressing the VDC's internal registers and dedicated video memory must be accomplished by indirect means. First the program must tell the VDC which of its 37 internal registers is to be accessed. Next the program must wait until the VDC is ready for the access, after which a read or write on the selected internal register may be performed. The following code is typical of a register read:

```
        ldx #regnum       ;VDC register to access
        stx $d600         ;write to control register
loop    bit $d600         ;check bit 7 of status register
        bpl loop          ;VDC not ready
        lda $d601         ;read from VDC register
        ...
```

The following code is typical of a register write operation:

```
        ldx #regnum       ;VDC register to write to
        stx $d600         ;write to control register
loop    bit $d600         ;check bit 7 of status register
        bpl loop          ;VDC not ready
        sta $d601         ;write to VDC register
        ...
```

Owing to this somewhat cumbersome method of controlling the 8568, the maximum possible frame rate in bit-mapped mode is generally too slow for arcade-style action video games, in which bit-intensive manipulation of the display is required.

The final versions of the 8568 had the revision codes R9a or R9b appended to the part number, apparently indicating undocumented improvements.

Features

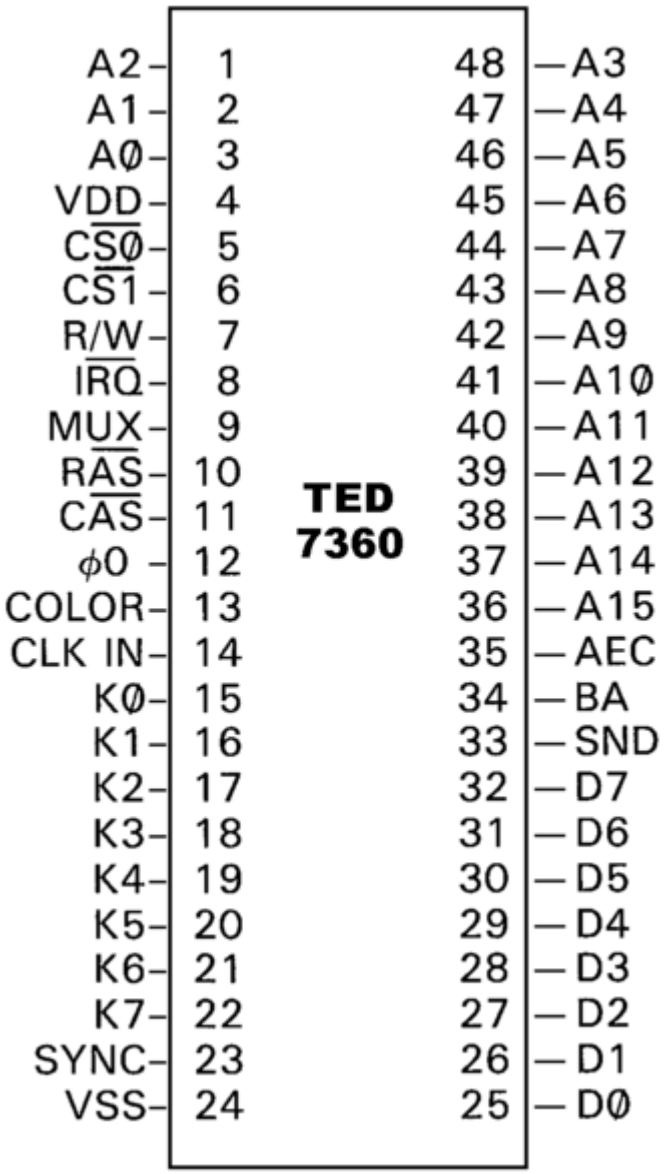
- 80 × 25 characters text resolution
- 640 x 400 pixels maximum video resolution
- Interlaced up to 80 x 50 text, 640H x 480V bitmap
- 3 character modes: standard, semigraphic and graphic, double width & HiRes bitmap.
- Output: digital RGBI with 16 colors or 16 gray shades, plus limited monochrome composite.
- Features: Interlace mode, horizontal & vertical scrolling, Light pen input, hardware cursor, underline, blink, reverse video, 2 character sets of 256 each, update ready interrupt
- Can access 64 KByte of memory, programmable to interface either 4164/4464 or 4416 DRAM
- 48 pins, +5 Volt DC supply.

Register Listing

Register	Hexadecimal	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Description
0	$00	HT7	HT6	HT5	HT4	HT3	HT2	HT1	HT0	Horizontal Total
1	$01	HD7	HD6	HD5	HD4	HD3	HD2	HD1	HD0	Horizontal Displayed
2	$02	HP7	HP6	HP5	HP4	HP3	HP2	HP1	HP0	Horizontal Sync Position
3	$03	VW3	VW2	VW1	VW0	HW3	HW2	HW1	HW0	Vertical/Horizontal Sync Width
4	$04	VT7	VT6	VT5	VT4	VT3	VT2	VT1	VT0	Vertical Total
5	$05	--	--	--	VA4	VA3	VA2	VA1	VA0	Vertical Adjust
6	$06	VD7	VD6	VD5	VD4	VD3	VD2	VD1	VD0	Vertical Displayed
7	$07	VP7	VP6	VP5	VP4	VP3	VP2	VP1	VP0	Vertical Sync Position
8	$08	--	--	--	--	--	--	IM1	IM0	Interlace Mode
9	$09	--	--	--	--	CTV4	CTV3	CTV2	CTV1	Character Total Vertical
10	$0A	--	CM1	CM0	CS4	CS3	CS2	CS1	CS0	Cursor Mode, Start Scan
11	$0B	--	--	--	CE4	CE3	CE2	CE1	CE0	Cursor End Scan Line
12	$0C	DS15	DS14	DS13	DS12	DS11	DS10	DS9	DS8	Display Start Address High Byte
13	$0D	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0	Display Start Address Low Byte
14	$0E	CP15	CP14	CP13	CP12	CP11	CP10	CP9	CP8	Cursor Position High Byte

15	$0F	CP7	CP6	CP5	CP4	CP3	CP2	CP1	CP0	Cursor Position Low Byte
16	$10	LPV7	LPV6	LPV5	LPV4	LPV3	LPV2	LPV1	LPV0	Light Pen Vertical Position
17	$11	LPH7	LPH6	LPH5	LPH4	LPH3	LPH2	LPH1	LPH0	Light Pen Horizontal Position
18	$12	UA15	UA14	UA13	UA12	UA11	UA10	UA9	UA8	Update Address High Byte
19	$13	UA7	UA6	UA5	UA4	UA3	UA2	UA1	UA0	Update Address Low Byte
20	$14	AA15	AA14	AA13	AA12	AA11	AA10	AA9	AA8	Attribute Start Address High Byte
21	$15	AA7	AA6	AA5	AA4	AA3	AA2	AA1	AA0	Attribute Start Address Low Byte
22	$16	CTH3	CTH2	CTH1	CTH0	CDH3	CDH2	CDH1	CDH0	Character Total Horizontal, Character Display Horizontal
23	$17	--	--	--	CDV4	CDV3	CDV2	CDV1	CDV0	Character Display Vertical
24	$18	COPY	RVS	CBRATE	VSS4	VSS3	VSS2	VSS1	VSS0	Vertical Smooth Scrolling
25	$19	TEXT	ATR	SEMI	DBL	HSS3	HSS2	HSS1	HSS0	Horizontal Smooth Scrolling
26	$1A	FG3	FG2	FG1	FG0	BG3	BG2	BG1	BG0	Foreground/Background color
27	$1B	AI7	AI6	AI5	AI4	AI3	AI2	AI1	AI0	Address Increment per Row
28	$1C	CB15	CB14	CB13	RAM	--	--	--	--	Character Base Address
29	$1D	--	--	--	UL4	UL3	UL2	UL1	UL0	Underline Scan Line
30	$1E	WC7	WC6	WC5	WC4	WC3	WC2	WC1	WC0	Word Count
31	$1F	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0	Data Register
32	$20	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	Block Start Address High Byte
33	$21	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0	Block Start Address Low Byte
34	$22	DEB7	DEB6	DEB5	DEB4	DEB3	DEB2	DEB1	DEB0	Display Enable Begin
35	$23	DEE7	DEE6	DEE5	DEE4	DEE3	DEE2	DEE1	DEE0	Display Enable End
36	$24	--	--	--	--	DRR3	DRR2	DRR1	DRR0	DRAM Refresh Rate
37	$25	HSYNC	VSYNC	--	--	--	--	--	--	SYNC Polarity (undocumented)

MOS Technology TED



TED pinout

The **7360 Text Editing Device** (**TED**) was an integrated circuit made by MOS Technology, Inc. It was a video chip that also contained sound generation hardware, DRAM refresh circuitry, interval timers, and keyboard input handling. It was designed for the Commodore Plus/4 and 16. Packaging consisted of a JEDEC-standard 48-pin DIP.

Video capabilities

The video capabilities provided by the TED were largely a subset of those in the VIC-II. The TED supported five video modes:

- Text mode of 40×25 characters with 8×8 pixels
- Multicolor text (4×8 pixels per character, double pixel width in x direction)
- Extended background color mode (8×8 pixels per character)
- Multicolor Graphics 160×200 pixels
- Hi-Res Graphics 320×200 pixels

These were largely unchanged from the corresponding VIC-II modes. However, the TED lacked the sprite capabilities of the VIC-II, making it unsuitable for most action games. The TED did include one feature that the VIC-II lacked: luminance control. Fifteen of its 16 colors (black being the exception) could be assigned one of 8 luminance values, thus making the TED capable of displaying a far wider array of colors than the VIC-II. The full palette of 121 colors is shown below.

hue / luminance	0	1	2	3	4	5	6	7
1 — black	1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7
2 — white	2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7
3 — red	3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7
4 — cyan	4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7
5 — purple	5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7
6 — green	6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7
7 — blue	7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7
8 — yellow	8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7
9 — orange	9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7
10 — brown	10,0	10,1	10,2	10,3	10,4	10,5	10,6	10,7
11 — yellow-green	11,0	11,1	11,2	11,3	11,4	11,5	11,6	11,7
12 — pink	12,0	12,1	12,2	12,3	12,4	12,5	12,6	12,7
13 — blue-green	13,0	13,1	13,2	13,3	13,4	13,5	13,6	13,7
14 — light blue	14,0	14,1	14,2	14,3	14,4	14,5	14,6	14,7
15 — dark blue	15,0	15,1	15,2	15,3	15,4	15,5	15,6	15,7

16 — light green	16,0	16,1	16,2	16,3	16,4	16,5	16,6	16,7

Sound capabilities

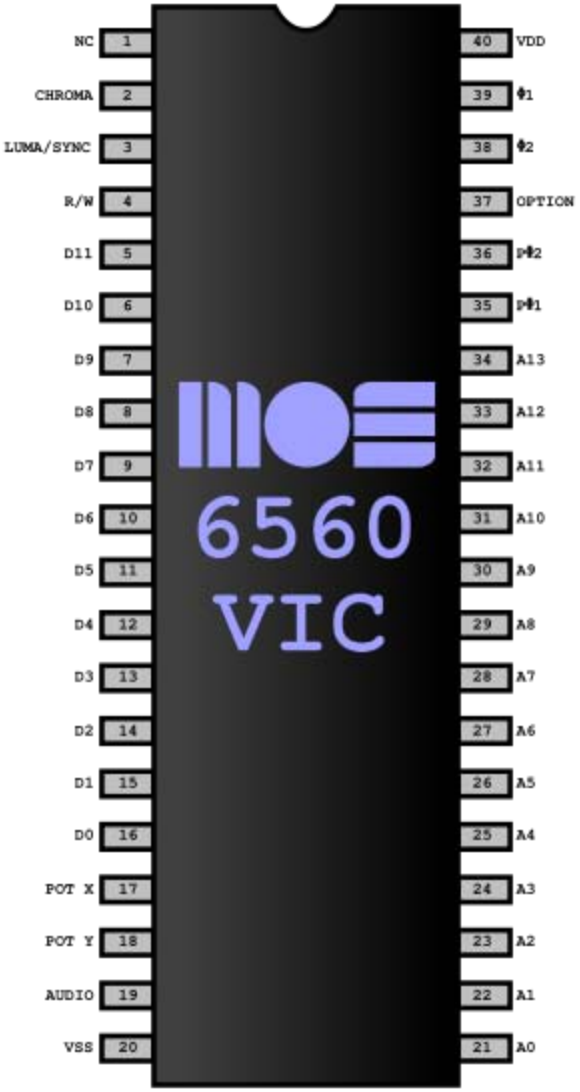
The TED featured a simple tone generator that produced two channels of audio. The first channel produced a square wave, and the second could produce either a square wave or white noise. This tone generator was designed for business applications, and did not provide the extensive sound features found in the SID chip.

Other features

The TED includes three 16-bit interval timers, which consist of down counters operating at the master clock frequency. They can generate IRQs on underflow. The chip also contains an I/O port, which is used on the Plus/4 and 16 to scan the keyboard and joystick. In addition, it handles bank switching, used by the operating system to maximize the amount of RAM available to Commodore BASIC.

An undesirable feature of the chip is its well-known tendency to destroy itself though overheating. To preserve a computer which employs this chip in working order, it is desirable to improve its cooling.

MOS Technology VIC



Pinout diagram of the 6560 version of the MOS VIC chip. This circuit was packaged in a standard 40-pin DIP casing.

The **VIC (Video Interface Chip)**, specifically known as the **MOS Technology 6560** (NTSC version) / **6561** (PAL version), is the integrated circuit chip responsible for generating video graphics and sound in the Commodore VIC-20 home computer. It was originally designed for applications such as low cost CRT terminals, biomedical monitors, control system displays and arcade or home video game consoles.

The chip was designed by Al Charpentier in 1977 but Commodore could not find a market for the chip. In 1979 MOS Technology began work on a video chip named *MOS Technology 6564* intended for the *TOI* computer and had also made some work on another chip, *MOS 6562* intended for a color version of the Commodore PET. Both of these chips failed due to memory timing constraints (both required very fast and thus expensive SRAM, making them unsuitable for mass production). Before finally starting to use the VIC in the VIC-20, chip designer Robert Yannes fed features from the 6562 (a better sound generator) and 6564 (more colors) back to the 6560, so before beginning mass production for the VIC-20 it had been thoroughly revised.

Its features include:

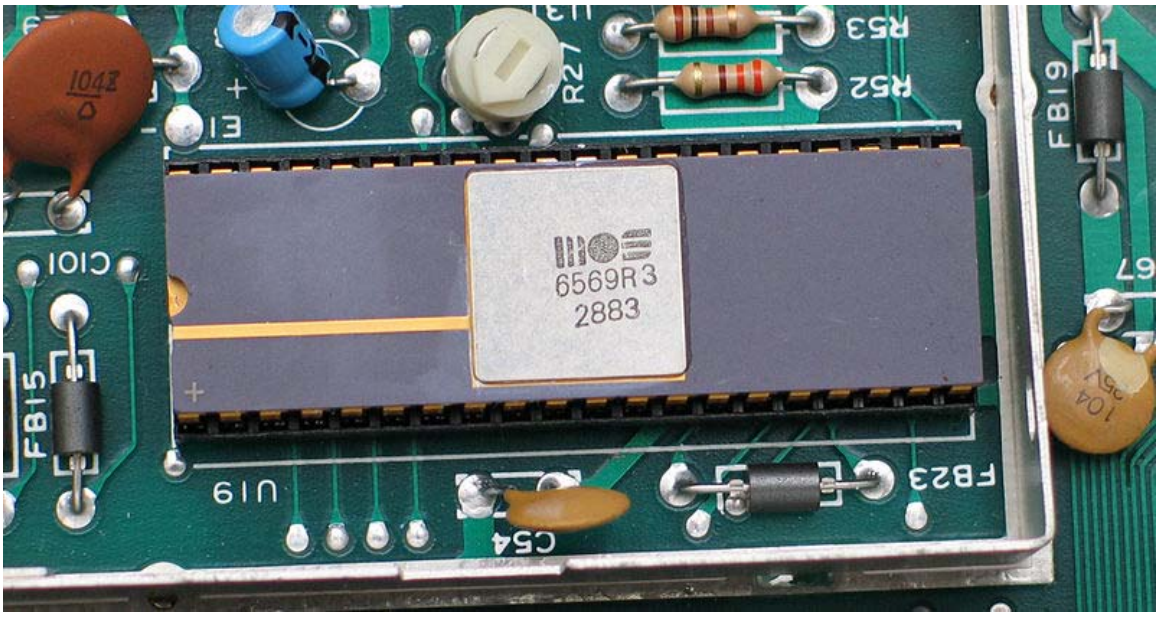
- 16 KB address space for screen, character and color memory (only 5 KB points to RAM on the VIC-20 without a hardware modification)
- 16 colors (the upper 8 can only be used in the global background and auxiliary colors)
- two selectable character sizes (8×8 or 8×16 bits; the pixel width is 1 bit for "hires" characters and 2 bits for "multicolor" characters)
- maximum video resolution depends on the television system (176 × 184 is the standard for the VIC-20 firmware, although at least 224 × 256 is possible on the PAL machine)
- 4 channel sound system (3 square wave + "white" noise + global volume setting)
- on-chip DMA
- two 8-bit A/D converters
- light pen support

The VIC was programmed by manipulating its 16 control registers, memory mapped to the range $9000–$900F in the VIC-20 address space. The on-chip A/D converters were used for dual paddle position readings by the VIC-20, which also used the VIC's lightpen facility. The VIC preceded the much more advanced VIC-II, used by the VIC-20's successors, the C64 and C128.

VIC IC list

- MOS Technology 6560 NTSC
- MOS Technology 6561E PAL Ceramic version, used in early VIC-20's
- MOS Technology 6561-101 PAL

MOS Technology VIC-II



MOS 6569R3 (PAL version) on a C64 main board

The **VIC-II (Video Interface Chip II)**, specifically known as the MOS Technology 6567/8562/8564 (NTSC versions), 6569/8565/8566 (PAL), is the microchip tasked with generating Y/C/composite video graphics and DRAM refresh signals in the Commodore 64 and C128 home computers.

Succeeding MOS's original VIC (used in the VIC-20), the VIC-II was one of the two chips mainly responsible for the C64's success (the other chip being the 6581 SID).

Development history

The VIC-II chip was designed primarily by Al Charpentier and Charles Winterble at MOS Technology, Inc. as a successor to the MOS Technology 6560 "VIC". The team at MOS Technology had previously failed to produce two graphics chips named *MOS Technology 6562* for the Commodore TOI computer, and *MOS Technology 6564* for the Color PET, due to memory speed constraints.

In order to construct the VIC-II, Charpentier and Winterble made a market survey of current home computers and video games, listing up the current features, and what features they wanted to have in the VIC-II. The idea of adding sprites came from the Texas Instruments TI-99/4A computer and its TMS9918 graphics coprocessor. About 3/4 of the chip surface is used for the sprite functionality.

The chip was partly laid out using electronic design automation tools from *Applicon* (now a part of UGS Corp.), and partly laid out manually on vellum paper. The design was

partly debugged by fabricating chips containing small subsets of the design, which could then be tested separately. This was easy since MOS Technology had both its research and development lab and semiconductor plant at the same location.

The work on the VIC-II was completed in November 1981 while Robert Yannes was simultaneously working on the SID chip. Both chips, like the Commodore 64, were finished in time for the Consumer Electronics Show in the first weekend of January 1982.

VIC-II features

- 16 KB address space for screen, character and sprite memory
- 320 × 200 pixels video resolution (160 × 200 in multi-color mode)
- 40 × 25 characters text resolution
- Three character display modes and two bitmap modes
- 16 colors
- Concurrent handling of 8 sprites per scanline, each of 24 × 21 pixels (12 × 21 multicolor)
- Raster interrupt
- Smooth scrolling
- Independent dynamic RAM refresh
- Bus mastering for a 6502-style system bus; CPU and VIC-II accessing the bus during alternating half-clock cycles (the VIC-II will halt the CPU when it needs extra cycles)

Technical details

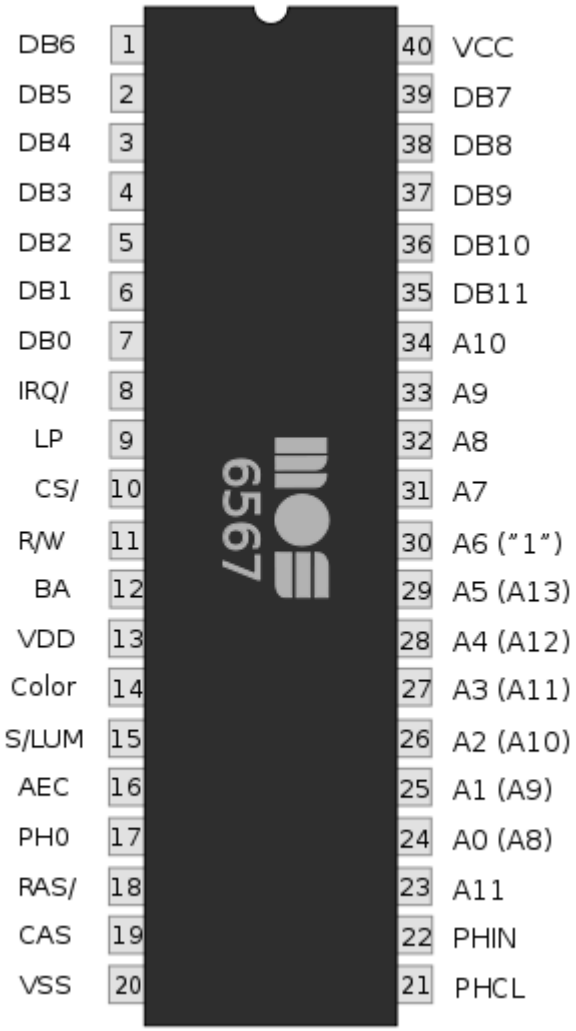
Programming

Supratechnic, a type-in program published by *COMPUTE!'s Gazette* in November 1988, showcases the careful use of raster interrupts to display information outside of the standard screen borders (here: the upper and lower border).

The VIC-II was programmed by manipulating its 47 control registers (up from 16 in the VIC), memory mapped to the range $D000–$D02E in the C64 address space. Of all these registers, 34 dealt exclusively with sprite control (sprites being called MOBs, from Movable Object Blocks, in the VIC-II documentation). Like its predecessor, the VIC-II handled light pen input, and with help from the C64s standard character ROM, provided the original PETSCII character set from 1977 on a similarly dimensioned display as the 40-column PET series.

By reloading the VIC-II's control registers via machine code hooked into the raster interrupt routine (the scanline interrupt), one could program the chip to generate significantly more than 8 concurrent sprites (a process known as sprite multiplexing), and generally give every program-defined slice of the screen different scrolling, resolution and color properties. The hardware limitation of 8 sprites per scanline could be increased further by letting the sprites flicker rapidly on and off. Mastery of the raster interrupt was essential in order to unleash the VIC-II's capabilities. Many demos and some later games would establish a fixed "lock-step" between the CPU and the VIC-II so that the VIC registers could be manipulated at exactly the right moment.

The VIC-II chip has a fixed 16-color palette, shown above.



MOS 6567 VIC-II pinout.

Registers

The VIC-II has 47 read/write registers listed below:

Register	Hexadecimal	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Description
0	D000					M0X				X Coordinate Sprite 0
1	D001					M0Y				Y Coordinate Sprite 0
2	D002					M1X				X Coordinate Sprite 1
3	D003					M1Y				Y Coordinate Sprite 1
4	D004					M2X				X Coordinate Sprite 2

5	D005					M2Y				Y Coordinate Sprite 2
6	D006					M3X				X Coordinate Sprite 3
7	D007					M3Y				Y Coordinate Sprite 3
8	D008					M4X				X Coordinate Sprite 4
9	D009					M4Y				Y Coordinate Sprite 4
10	D00A					M5X				X Coordinate Sprite 5
11	D00B					M5Y				Y Coordinate Sprite 5
12	D00C					M6X				X Coordinate Sprite 6
13	D00D					M6Y				Y Coordinate Sprite 6
14	D00E					M7X				X Coordinate Sprite 7
15	D00F					M7Y				Y Coordinate Sprite 7
16	D010	M7X8	M6X8	M5X8	M4X8	M3X8	M2X8	M1X8	M0X8	MSBs of X coordinates
17	D011	RST8	ECM	BMM	DEN	RSEL		YSCROLL		Control register 1
18	D012				RASTER					Raster counter
19	D013				LPX					Light Pen X
20	D014				LPY					Light Pen Y
21	D015	M7E	M6E	M5E	M4E	M3E	M2E	M1E	M0E	Sprite enabled
22	D016	-	-	RES	MCM	CSEL		XSCROLL		Control register 2
23	D017	M7YE	M6YE	M5YE	M4YE	M3YE	M2YE	M1YE	M0YE	Sprite Y expansion
24	D018	VM13	VM12	VM11	VM10	CB13	CB12	CB11	-	Memory pointers
25	D019	IRQ	-	-	-	ILP	IMMC	IMBC	IRST	Interrupt register
26	D01A	-	-	-	-	ELP	EMMC	EMBC	ERST	Interrupt enabled
27	D01B	M7DP	M6DP	M5DP	M4DP	M3DP	M2DP	M1DP	M0DP	Sprite data priority
28	D01C	M7MC	M6MC	M5MC	M4MC	M3MC	M2MC	M1MC	M0MC	Sprite multicolor
29	D01D	M7XE	M6XE	M5XE	M4XE	M3XE	M2XE	M1XE	M0XE	Sprite X expansion
30	D01E	M7M	M6M	M5M	M4M	M3M	M2M	M1M	M0M	Sprite-sprite collision
31	D01F	M7D	M6D	M5D	M4D	M3D	M2D	M1D	M0D	Sprite-data collision

32	D020	-	-	-	-		EC			Border color
33	D021	-	-	-	-		B0C			Background color 0
34	D022	-	-	-	-		B1C			Background color 1
35	D023	-	-	-	-		B2C			Background color 2
36	D024	-	-	-	-		B3C			Background color 3
37	D025	-	-	-	-		MM0			Sprite multicolor 0
38	D026	-	-	-	-		MM1			Sprite multicolor 1
39	D027	-	-	-	-		M0C			Color sprite 0
40	D028	-	-	-	-		M1C			Color sprite 1
41	D029	-	-	-	-		M2C			Color sprite 2
42	D02A	-	-	-	-		M3C			Color sprite 3
43	D02B	-	-	-	-		M4C			Color sprite 4
44	D02C	-	-	-	-		M5C			Color sprite 5
45	D02D	-	-	-	-		M6C			Color sprite 6
46	D02E	-	-	-	-		M7C			Color sprite 7

Colors

In multicolor bitmap mode (160×200 pixels, which most games used) characters had 4×8 pixels (the characters were still approximately square since the pixels were double width) and 4 colors out of 16 colors. The 4th color was the same for the entire screen (the background color), while the other 3 could be set individually for every such 4×8 pixel area. Two colors were loaded from the active text screen, and the third was loaded from color RAM. Sprites in multicolor mode (12×21 pixels) had three colors: two shared among all sprites and one individual. The artist had to pick shared colors such that the combination with individual colors lead to a colorful impression. Some games reloaded shared colors during the raster interrupt; for example, the game *Turrican II's* underwater area (which was vertically distinct) had different colors. Others, such as Epyx's *Summer Games* and *COMPUTE!'s Gazette*'s *Basketball Sam & Ed*, overlaid two high-resolution sprites to allow two foreground colors to be used without sacrificing horizontal resolution . Of course, this technique reduced the number of available sprites by half.

On PAL C64s, the PAL delay line in the monitor or TV which averages the color hue, but not the brightness, of consecutive screen lines can be (ab)used to create seven nonstandard colors by alternating screen lines showing two colors of identical brightness. There are seven such pairs of colors in the VIC chip.

The C64's team did not spend much time on mathematically computing the 16 color palette. Robert Yannes, who was involved with the development of the VIC-II, said:

I'm afraid that not nearly as much effort went into the color selection as you think. Since we had total control over hue, saturation and luminance, we picked colors that we liked. In order to save space on the chip, though, many of the colors were simply the opposite side of the color wheel from ones that we picked. This allowed us to reuse the existing resistor values, rather than having a completely unique set for each color.

The 1993 game *Mayhem in Monsterland* is an example of what can be done if the VIC-II features are used to the maximum. It uses linewise PAL-colorblending, color interlace, a nonstandard way to achieve very fast scrolling and very sophisticated and extremely colorful character-based graphics and very well drawn sprites, some even with hires overlays, to achieve a level of graphical quality that was almost comparable to 16 bit machines of the era.

The VIC-IIe

The 8564/8566 VIC-IIe in the Commodore 128 used 48 pins rather than 40, as it produced more signals, among them the clock for the additional Zilog Z80 CPU of that computer. It also had two extra registers. One for accessing the added numerical keypad and other extra keys of that computer (this function was added to the VIC merely because that proved to be the easiest place in the computer to add the necessary three extra output pins) and the other for toggling between a 1 MHz and a 2 MHz system clock; at the higher speed the VIC-II's video output is merely displaying every second byte in the code as black hires bit-pattern on the screen, suggesting use of the C128's 80-column mode at that speed (via the 8563 VDC RGB chip). Rather unofficially, the two extra registers were also available in the C128's C64 mode, permitting some use of the extra keys, as well as double-speed-no-video execution of CPU-bound code (such as intensive numerical calculations) in self-made C64 programs. The extra registers were also one source of minor incompatibility between the C128's C64 mode and a real C64 - a few older C64 programs inadvertently wrote into the 2MHz toggle bit, which would do nothing at all on a real C64, but would result in a messed-up display on a C128 in C64 mode.

The VIC-IIe has the little-known ability to create an additional set of colors by manipulating the registers in a specific way that puts the color signal out of phase with what other parts of the chip consider it to be in.

Using the specific behavior of the VIC-IIe's test bit, it is furthermore capable of producing a real interlace picture with a resolution of 320x400 (hires mode) and 160x400 (multicolor mode).

List of VIC-II versions

- PAL
  - MOS Technology 6569 – (PAL-B)
  - MOS Technology 6572 – (PAL-N)
  - MOS Technology 6573 – (PAL-M)

    - MOS Technology 8565 – HMOS-II version for "C64E" motherboards
    - MOS Technology 8566 – VIC-II E (PAL-B) C128 version
    - MOS Technology 8569 – VIC-II E (PAL-N) C128 version

- NTSC
    - MOS Technology 6566 – designed for SRAM/non-muxed address lines (used in the Commodore MAX Machine)
    - MOS Technology 6567 – Original NMOS version
    - MOS Technology 8562 – HMOS-II version
    - MOS Technology 8564 – VIC-II E C128 version

Chapter 12

Motorola 6845 and Motorola 6847

Motorola 6845

Motorola 6845 CRT controller

The **Motorola 6845** (commonly **MC6845**) is a video address generator first introduced by Motorola and used among others in the Videx VideoTerm display cards for the Apple II computers, the MDA and CGA video adapters for the IBM PC, in the Amstrad CPC and the BBC Micro. Its functionality was duplicated and extended by custom circuits in the EGA and VGA PC video adapters. It is related to the later **6545** manufactured by MOS Technology (Commodore Semiconductor Group) and Rockwell (in two variations) and was cloned as the Hitachi **46505** (which was used in Videx's UltraTerm card).

It is also known as the **6845 CRTC** or the **CRTC6845**, meaning "cathode ray tube controller".

Although intended for designs based on the Motorola 6800 CPU and given a related part number, it was more commonly used alongside various other processors.

Overview

The chip generates the signals necessary to interface with a raster display but does not generate the actual pixels, though it does contribute cursor and video-blanking information to the pixel video (intensity) signals. It is used to produce correctly timed horizontal and vertical sync and provide the address in memory from which the next pixel or set of pixels should be read. The process of reading that value, converting it into pixels, and sending it to a CRT is left to other circuits. Because of this, systems using the 6845 may have very different numbers and values of colors, or may not support color at all.

Interlaced and non-interlaced output modes are supported, as is a hardware text cursor. The sync generation includes generation of horizontal and vertical video blanking signals, which are used to condition the external pixel generation circuits. Also, an internal latch is provided which when triggered will duplicate and retain a copy of the video address so that it can later be read back by the CPU. This is useful for light pens and light guns which can function by sending a pulse to the 6845 when the electron beam passes, allowing a running program to read back the location that was pointed at. (Because of this feature, most computer video adapters using a 6845 included a light pen interface, though it was usually an internal connector on the board itself, not on the outside of the computer, and it was usually undocumented in the user manual.)

Because all aspects of video timing are programmable, a single machine can switch between NTSC and PAL timings in software. The 6845 can be used to drive monitors or any other raster display.

Internals

The chip has a total of 18 x 8-bit registers controlling all aspects of video timings. Only two addresses are exposed to external components - one to select which internal register is to be read or written to and another to access that register.

The 6845 is intended for character based displays. Every address it generates is composed of two parts - a 14 bit character address and a 5 bit row address. Using the full address range RA0-RA4:CA0-CA13 the 6845 can address 2^(14+5) = 512 kB of memory.

The character address increases linearly. When the chip signals horizontal sync it increases the row address. If the row address does not equal the programmatically set number of rows per character then the character address is reset to have the same value as it did at the beginning of the current scanline. Otherwise the row address is reset to zero.

If the character address is used to look up a character reference in RAM and the row address to index a table of character graphics in ROM an ordinary text mode display is constructed.

Linear framebuffers

As described above, the 6845 is not ordinarily able to provide large linear framebuffers. A design could use only the 14 bit character address and set the number of rows per character to 1 but it would be constrained to 16 kB of addressable memory.

A solution is found in the Amstrad CPC, which combines the row address and character address to provide linear scanlines within a non-linear buffer. It maps row address RA0-RA2 to memory address MA11-MA13 and character address CA0-CA10 to memory address MA0-MA10. This has the advantages of easier programming for non-character display and easy smooth horizontal scrolling but can impede smooth vertical scrolling.

Differences from the 6545

Although overwhelmingly compatible, a number of small variations exist between the 6845 and 6545.

The biggest difference is that the 6545 may be configured so that it has sole access to the address bus for video memory. Two additional registers are included for setting any address the CPU wishes to read and the chip alternates between outputting addresses for display generation and the display set for CPU access.

Smaller changes are that the MOS Technology and one variation of the Rockwell 6545 lack interlaced output support and all 6545s include an optional address skew, which delays display enable for one character cycle if set. This second feature was incorporated into later variations of the Motorola 6845.

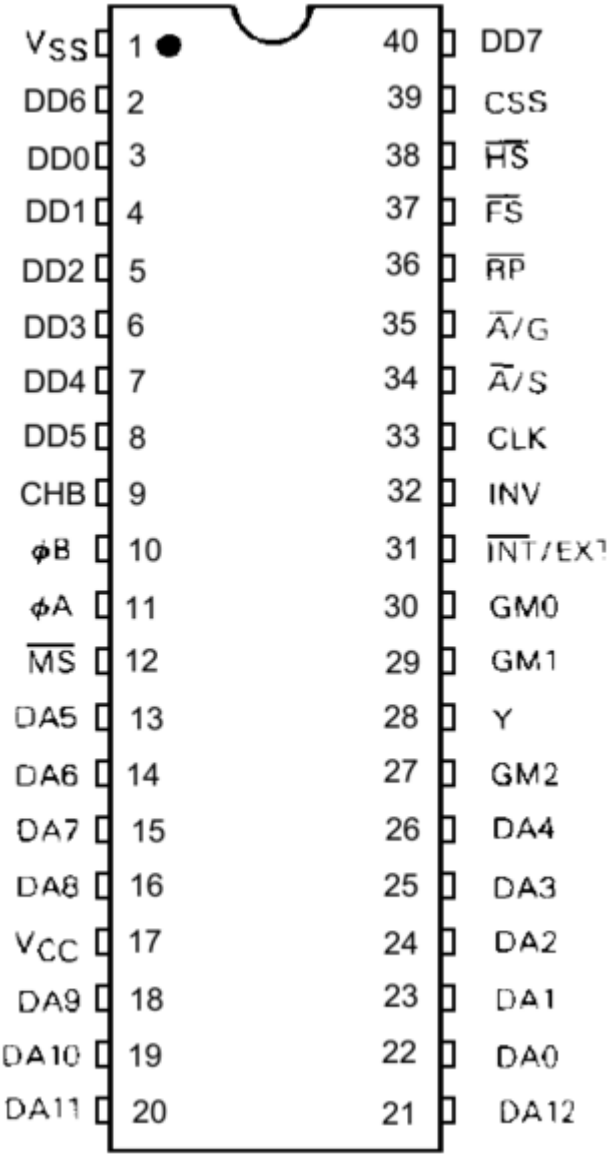
The 6545 may be set to work in linear 14 bit mode using a status bit. On the 6845 the same thing requires adjustment of the character height.

Tricks

The 6845 reads the start address for its display once per frame. However, if the internal timing values on the chip are altered at the correct time it can be made to prepare for a new frame without ending the current one - creating a non-continuous break in generated addresses midway through the display. This is commonly used by games to provide one moving area of the display (usually the play field) and one static (usually a status display).

Vertical scrolling appears constrained because only the character start address can be set and the row address is always zeroed at frame start, but by adjusting border times it is possible to shift the position the framebuffer is shown on the raster display for increments in between whole characters. With drawing of blank pixels at the screen edges, this can be made invisible to the user creating just the illusion of a smooth vertical scroll.

Motorola 6847



The **MC6847** is a video display generator (VDG) first introduced by Motorola and used in the TRS-80 Color Computer, Dragon 32/64, Laser 200 and Acorn Atom among others. It is a relatively simple display generator compared to other display chips of the time. It is capable of displaying text and graphics contained within a roughly square display matrix 256 pixels wide by 192 lines high. It is capable of displaying 9 colors: black, green, yellow, blue, red, buff (almost-but-not-quite white), cyan, magenta, and orange. The low display resolution is a necessity of using television sets as display monitors. Making the display wider risked cutting off characters due to overscan. Compressing more dots into the display window would easily exceed the resolution of the television and be useless.

Video Modes

Video Mode	Resolution	Colours	Bytes
Alphanumeric Internal	32 × 16	1 + Black	512
Alphanumeric External	32 × 16	1 + Black	512
Semigraphics 4	64 × 32	8 + Black	512
Semigraphics 6	64 × 48	4 + Black	512
Color Graphics 1	64 × 64	4	1024
Resolution Graphics 1	128 × 64	1 + Black	1024
Color Graphics 2	128 × 64	4	2048
Resolution Graphics 2	128 × 96	1 + Black	1536
Color Graphics 3	128 × 96	4	3072
Resolution Graphics 3	128 × 192	1 + Black	3072
Color Graphics 6	128 × 192	4	6144
Resolution Graphics 6	256 × 192	1 + Black	6144

Chapter 13

Yamaha V9938 and Yamaha V9958

Yamaha V9938

Yamaha V9938 in a MSX 2

The **Yamaha V9938** is a Video Display Controller (VDC) used in the Geneve 9640 enhanced TI-99/4A clone, as well as MSX 80s home computers (more specifically, the MSX 2).

The Yamaha V9938, also known as MSX-Video or VDP (Video Display Processor), is the successor of the Texas Instruments TMS9918 (used in the MSX1 and various other systems). The V9938 was in turn succeeded by the Yamaha V9958.

Specifications

- Video RAM: 64 KB to 128 KB
- Text modes: 80 x 24, 40 x 24 and 32 x 24
- Resolution: 512 x 212 (16 colours from 512) , 256 x 212 (16 colours from 512) and 256 x 212 (256 colours)
- Sprites: 32, 16 colours, max 8 per horizontal line
- Hardware acceleration for copy, line, fill, etc. Logical operations available.
- Interlacing to double vertical resolution
- Vertical scroll register

Detailed specifications

- Video RAM: 128 kB
    - Optionally 64 kB, in which case screen modes G6 and G7 are not available
    - Optionally 192 kB, where 64 kB is extended-VRAM (only available as backbuffer for G4 and G5 modes)
- Clock: 21 MHz
- Video output frequency: 15 kHz
- Sprites: 32, 16 colours (1 per line), max 8 per horizontal line
- Hardware acceleration, with copy, line, fill etc. With or without logical operations.
- Vertical scroll register
- Capable of superimposition and digitization
- Support for connecting a lightpen and a mouse
- Resolution:
    - Horizontal: 256 or 512
    - Vertical: 192, 212, 384 (interlaced) or 424 (interlaced)
- Colour modes:
    - Paletted RGB: 16 colours out of 512
    - Fixed RGB: 256 colours
- Screen modes
    - Text modes:
        - T1: 40 × 24 with 2 colours (out of 512)
        - T2: 80 × 24 with 4 colours (out of 512)
        - All text modes can have 26.5 rows as well.
    - Pattern modes
        - G1: 256 × 192 with 16 paletted colours and 1 table of 8×8 patterns
        - G2: 256 × 192 with 16 paletted colours and 3 tables of 8×8 patterns

        - G3: 256 × 192 with 16 paletted colours and 3 tables of 8×8 patterns
        - MC: 64 × 48 with 16 paletted colours and 8×2 patterns
        - All modes with 192 lines can have 212 lines as well (similarly 48 → 53 in MC).
    - Bitmap modes:
        - G4: 256 × 212 with 16 paletted colours
        - G5: 512 × 212 with 4 paletted colours
        - G6: 512 × 212 with 16 paletted colours
        - G7: 256 × 212 with 256 fixed-colours
        - All modes with 212 lines can have 192 lines as well (similarly 48 → 53 in MC).
        - All vertical resolutions can be doubled by interlacing

MSX-specific terminology

On MSX, the screen modes are often referred to by their assigned number in MSX-Basic. This mapping is as follows:

Basic mode	VDP mode	MSX system
Screen 0 (width 40)	T1	MSX 1
Screen 0 (width 80)	T2	MSX 2
Screen 1	G1	MSX 1
Screen 2	G2	MSX 1
Screen 3	MC	MSX 1
Screen 4	G3	MSX 2
Screen 5	G4	MSX 2
Screen 6	G5	MSX 2
Screen 7	G6	MSX 2
Screen 8	G7	MSX 2

Yamaha V9958

Yamaha V9958

Yamaha V9958(ceramic package)

The **Yamaha V9958** is a Video Display Controller (VDC) used in MSX 80s home computers. More specifically, the "TIM" upgrade to the TI-99/4A, MSX 2+ and MSX turbo R.

The Yamaha V9958, also known as MSX-Video, is the successor of the Yamaha V9938 (used in the Myarc Geneve 9640 upgrade for the TI-99/4A and the MSX2). It was generally conceived not to be a very major upgrade to its predecessor, which hampered

its adoption. The main new features are three graphical YJK modes (with up to 19268 colours) and horizontal scrolling registers.

Specifications

- Video RAM: 128 KB + 64 KB of expanded VRAM
- Text modes: 80 x 24 and 32 x 24
- Resolution: 512 x 212 (16 colours out of 512) and 256 x 212 (19268 colours)
- Sprites: 32, 16 colours, max 8 per horizontal line
- Hardware acceleration for copy, line, fill, etc.
- Interlacing to double vertical resolution
- Horizontal and vertical scroll registers

Detailed specifications

For detailed specifications, please refer to the Yamaha V9938 page, with the following additions:

- Horizontal scrolling registers
- YJK graphics modes (similar to YUV)
  - G7 + YJK + YAE: 256 x 212, 12499 colours + 16 colour palette
  - G7 + YJK: 256 x 212, 19268 colours
- Ability to execute hardware accelerated commands in non-bitmap screen modes
- Lightpen and mouse functions from V9938 were removed

MSX-specific terminology

On MSX, the screen modes are often referred to by their assigned number in MSX-Basic. This mapping is as follows:

Basic mode	VDP mode	MSX system
Screen 0 (width 40)	T1	MSX 1
Screen 0 (width 80)	T2	MSX 2
Screen 1	G1	MSX 1
Screen 2	G2	MSX 1
Screen 3	MC	MSX 1
Screen 4	G3	MSX 2
Screen 5	G4	MSX 2
Screen 6	G5	MSX 2
Screen 7	G6	MSX 2

Screen 8	G7	MSX 2
Screen 10	G7 with YJK and YAE	MSX 2+ and tR
Screen 11	G7 with YJK and YAE	MSX 2+ and tR
Screen 12	G7 with YJK	MSX 2+ and tR