NEC μPD7220/72120 Related Public Documents

- ISSCC (International Solid-State Circuit Conference)
- μPD7220A User's Manual
- Nikkei Electronics Magazine (μPD7220)
- Transistor Gijutsu (Technology) Magazine (μPD7220)
- Transistor Gijutsu (Technology) Magazine (μPD7220A)
- Nikkei Electronics Magazine (μPD72120)
- μPD72120 User's Manual

Go to https://www.oguchi-rd.com/LSI%20products.php to get more detailed NEC μPD7220/72120 related information such as;

“Logic Schematics”, “Design Notes”, “Evaluation Board Schematics”, “Evaluation Software”, “Silicon Die Photos”, “Newspaper”, “Magazine”, and so forth.

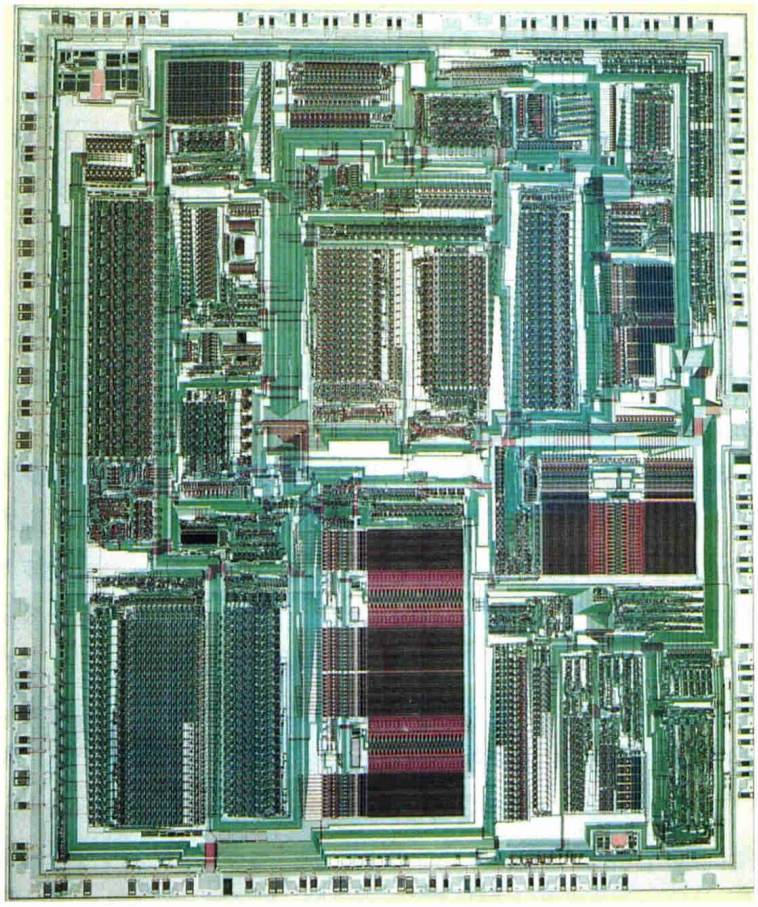
Go to https://www.oguchi-rd.com/patents.php to get patent information including NEC μPD7220/72120 related patents.

Article: A Graphics Display Controller LSI (Successor of μPD7220) with Enhanced Copy and Paint Functions

Tetsuji Oguchi, Mitsuro Ouchi,
Ryuji Horiguchi, Toshikazu Chiba ... NEC Microcomputer Product Division

Keishi Uno ..... NEC 1st LSI Division

NEC has developed a new graphics display controller LSI -- AGDC (μPD72120). Based on its predecessor GDC (μPD7220), which was placed on the market in 1982, the features of μPD72120 include an enhanced bit map processing function, a copy function of rectangular areas which include enlargement, shrinkage and rotation and a fill function of triangles and trapezoids. Within the LSI, a preprocessor for executing draw preprocessing is mounted in addition to a drawing processor. The CPU, preprocessor and drawing processor are arranged for 3-stage pipeline processing to enhance system performance. Capacity of the display memory can be increased to 32 M bytes (max.) using dual port memory chips. μPD72120 operates with clocks up to 8 MHz. (NIKKEI)



The article was translated from the Japanese in the "Nikkei Electronics / Feb. 23 , 1987".

A Graphics Display Controller LSI (Successor of μPD7220) with Enhanced Copy and Paint Functions

NEC announced the graphics draw/display LSI μPD7220 (GDC) at the ISSCC in February of 1981. Since then, this LSI has been used in a wide area of applications, such as OA equipment, fish-finder, photocomposers and satellite terminals. The GDC can directly control a display memory of up to 512 K bytes which is separated from the system bus. This LSI is equipped with functions to draw straight lines, arcs, graphics characters, etc. in high speeds. (Ref. 1) Although speeds of drawing one dimensional objects such as straight lines and arcs are thus increased, the GDC is not equipped with a two dimensional draw function (i.e. that of areas), such as paint and character font expansion for the bit map memory. The CPU then needs to perform their supplementary functions.

The newly developed μPD72120 (AGDC) aims to minimize supplementation by the CPU. Rich draw functions are added to the graphics controller and the drawing speeds are increased. As far as graphic draw functions are concerned, differences between the GDC and AGDC are not remarkable except that an ellipse, an arc, a sector and a chord can now be drawn by using single commands.

During the design of the AGDC, it was attempted to mount as many bit map processing functions (which were missing in the GDC) as possible: data transfer between display memories (copy or bitblt function), paint function in an arbitrary closed area and unconditional fill functions in a circle, an ellipse, a triangle and a trapezoid. (See Figure 1 (a).) Among these functions, the copy function plays an important role in expanding character fonts and displaying the window. Also, during a copy, a graphic can be slanted or rotated through an arbitrary angle and, under some restrictions, enlarged or shrinked by an arbitrary factor. (See Figure 1 (b).) Together with the enhanced bit map functions, a put function and a get function, which are used when one dimensional data stored on the main storage are expanded in a two dimensional manner on the display memory or vice versa, are also added.

Although these functions can also be offered using a general processor, it is then not practical to expect sufficient drawing speeds. (Ref. 4) Typical drawing speeds of the AGDC, when it is operated at a clock frequency of 8 MHz are given below. While drawing a straight line, the AGDC takes 500 ns to draw a bit or 1 pixel. If a copy function is used, 13,640 characters of 24 x 24 dot character font can be expanded in one second.

Rapid Shift From Character Display To Graphics Display:

As the memory price has lowered these years, there have been rapid changes in the display technology, that is, from character display to graphics display. Even for word processors, it now seems that handling of graphics data is a minimum condition. But graphics display requires a large memory capacity and its control is not easy. To change a character on a character display screen, for example, it is only necessary to rewrite one byte or one word of the character code memory contents. On a graphics display screen, however, it becomes necessary to expand the character font on the display memory again. If this is processed by the CPU, the rewriting speed is too slow. And painting a complex figure is too great a burden for the CPU.

For this reason, there have been attempts to implement high-speed bit map functions using special gate array circuits. But, due to restraints on the integration level and speed of gate arrays, it has not been possible to execute macro drawing functions like the filling of a triangle. So, at each of these small steps, the CPU needed to participate in the drawing. If the drawing speed is evaluated in small-step units, the speed of the array circuit is high. But from the viewpoint of an overall system, its performance often fails to meet expectations.

From this background, since the GDC appeared, many graphics controller LSI's with enhanced drawing functions have been announced. Table 1 lists some typical controller LSI's. These products show different features depending on design objectives of the manufacturers. Am95C60 (of Advanced Micro Devices, Inc., U.S.A.) and AGDC aim to implement high-speed drawing functions by means of hardware, while HD63484 (of Hitachi) tries to emphasize its display control function. 82786 (of Intel, U.S.A.) mainly tries to enrich its display control function and TMS34010 (of Texas Instruments Inc, U.S.A.) is equipped with a built-in processor so that drawing software can be created freely.

Another important move to be noted concerning graphics display is the appearance of dual port memory chips with a line buffer. The first product was a 64 K dynamic RAM chip having a serial memory type built-in line buffer. (Ref. 2) After this, 256 K dual port memories with a random access type line buffer which are equipped with real-time data transfer function, pointer control function and write per bit function were announced.

The use of dual port memory chips markedly increases possible draw timings, to make the best use of the potential power of a continuous drawing controller.

Improve a System Performance

The AGDC is equipped with dual port memories. The AGDC has been designed to improve the system performance through the combined graphics control LSI and the CPU. Its internal architecture adopts a multi-processor configuration. Separate from the drawing processor, a processor especially for pre-drawn processings (called a preprocessor) is included.

The preprocessor performs pre-drawn processings such as address conversion from X-Y coordinates to absolute addresses. As the command interface, the CPU directly writes command parameters into the built-in register of the preprocessor. The preprocessor operates independently of the drawing processor and, as a result, the CPU can execute its pre-drawn processings such as command interpretation of CGI and BASIC and driving of the AGDC; the preprocessor can perform its pre-drawn

processings and the drawing processor can draw in a 3-stage pipeline manner when viewed as a system.

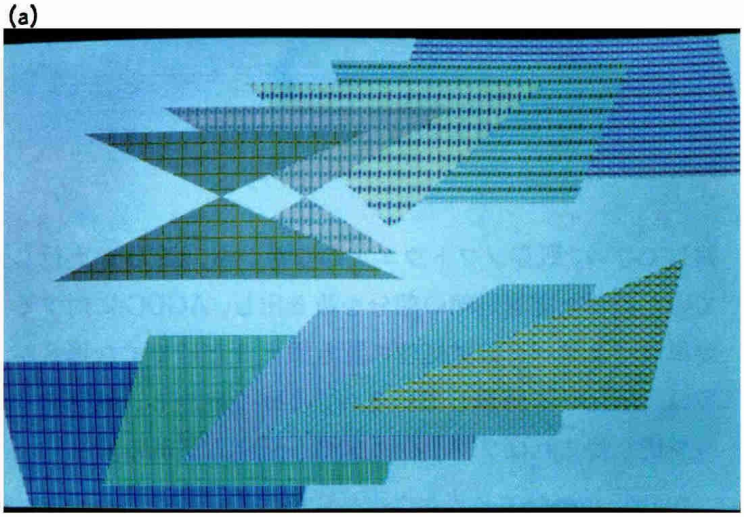
In addition, during the research of the drawing algorithm, a special hardware suiting the algorithm was designed to improve the drawing speed. The drawing processor was designed to tightly control these hardware components and, even when drawing types change, adapt to the various applications. Note that, in each step to be executed in a clock cycle of 125 ns (at 8MHz), up to 6 commands and direct branches can be executed. The execution of each command can use a command prefetch function as pipelining and, so, the operation of the processor itself includes both modes of parallel processing and pipeline processing.

Non-Compatible with GDC:

We greatly improved the functions rather than maintain compatibility with the GDC at the command and function levels. To maintain compatibility, it would have been necessary to use the GDC as the master, as was done in the past, and add an AGDC to improve the drawing functions. In this case, the existing software can be used but the functions of the AGDC are not available. However, it is not difficult to add routines which make use of the AGDC to the existing software. In many cases, in fact, it is simply enough to extract the complex drawing parts (executed by the CPU in the past) from among the existing software and replace them with simple parameter transfer processings for the AGDC. By judging whether or not an AGDC is mounted and, when mounted, switching the control flow, unification of the entire software is also possible.

We hope that the AGDC will be used in every application field which requires graphics functions, such as personal computers, word processors, work stations and laser printers.

(a)



(b)

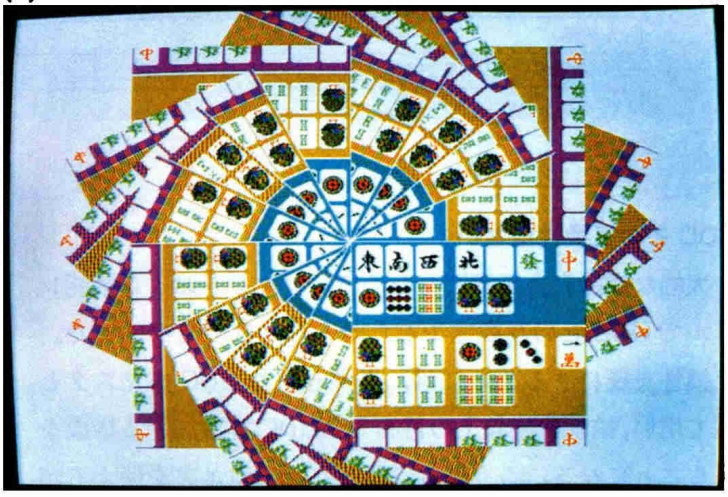


Figure 1

Drawings Using 640 x 480 Dot Display Screens

(a) A single command with its parameters as coordinates of apexes has painted the trapezoids.

(b) The result of a rotational copy of a rectangle. A reduce processing is executed as the rotation angle increases. This is because if the rectangle is rotated with a fixed magnification factor, the length of each side changes.

Table 1 Functional Comparison of Typical Graphics Controllers

	HD63484 (Hitachi)	Am95C60 (Advanced Micro Devices, Inc.)	TMS34010 (Texas Instruments Inc.)	82786 (Intel Corp.)	μPD72120 (NEC)
o Drawing function					
Straight line, rectangle	◎	○	○	◎	◎
Circle, arc	◎	◎	-	◎	◎
Ellipse, elliptic arc	◎	x	-	x	◎
Sector, chord	x	x	-	x	◎
Arbitrary closed area painting	△	◎	-	x	◎
Rectangle fill	◎	◎	-	x	◎
Circle/ellipse fill	x	◎	-	x	◎
Triangle fill	x	◎	-	x	◎
Trapezoid fill	x	x	-	x	◎
o Display memory data transfer					
Normal copy	◎	◎	◎	◎	◎
90 deg. rotation copy	◎	◎	-	◎	◎
Slant copy	x	x	-	x	◎
Enlarged/shrinked copy	x	○	-	x	◎
Enlarged/shrinked copy including arbitrary angle rotation	x	x	-	x	◎
o Hardware clipping	◎	◎	x	◎	◎
o Put, get	x	◎	-	x	◎
o Drawing position designation	X-Y coordinates	X-Y coordinates	X-Y coordinates	X-Y coordinates, absolute addresses	X-Y coordinates, absolute addresses
o Display memory configuration (maximum capacity)	Pixel type (2 M bytes)	Plane type (8 M bytes)	Pixel type (128 M bytes)	Pixel type (4 M bytes)	Plane type, pixel type (32 M bytes)
o Mapping of display memory into CPU	x	x	x	◎	◎
o Dual port memory control	x	◎	◎	◎	◎
o Separation of drawing from pre-processing	x	x	x	x	◎
o Separation of clock between display and drawing	x	◎	x	x	◎
o Display functions					
Screen partition	◎	x	x	◎	x
Enlarged display	◎	x	x	◎	x
Hardware window	△	△	x	◎	x
Cursor output	◎	x	x	◎	◎

In the table below,

△ : shows that the function is available but not sufficiently practical.
o : shows that only a part of the function is available.
- : shows that the function can be implemented by software.

Internal Architecture with A Multiprocessor Configuration

The AGDC is internally divided into the following five functional blocks. (See Figure 2.)

1 Preprocessor
2 Drawing processor
3 Display processor
4 Synchronization signal generator
5 CPU interface

The important feature of this architecture is its multiprocessor configuration consisting of three types of processors for drawing, displaying and preprocessing. And this configuration enables pipelining by the CPU, preprocessor and drawing processor. The drawing processor is equipped with a special hardware to improve its drawing speed.

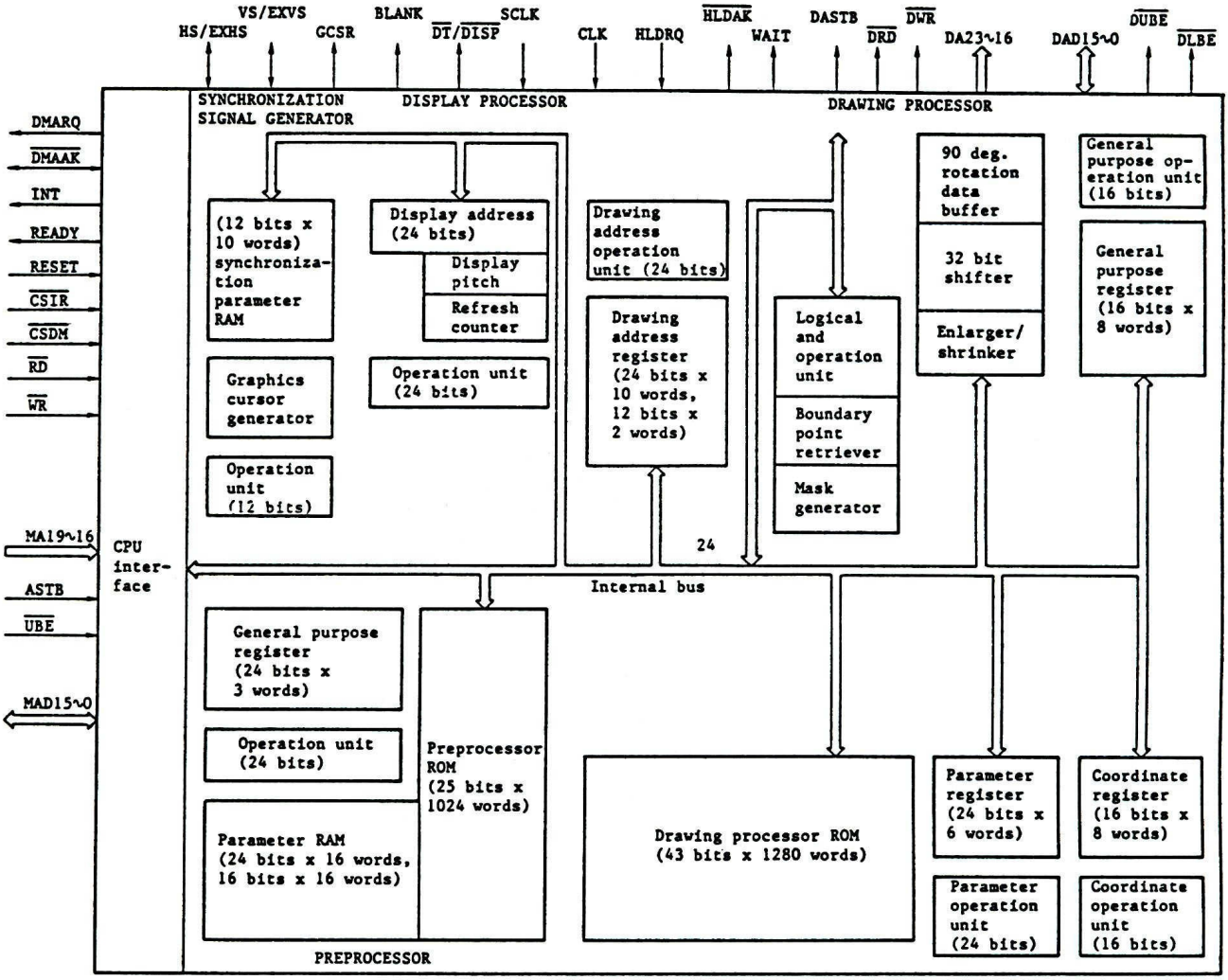


Figure 2 Internal Configuration

The AGDC consists of five functional blocks: synchronization signal generator, display processor, preprocessor, drawing processor and CPU interface. These blocks are connected by a 24-bit internal bus. The feature of this architecture is that the preprocessor and drawing processor perform pipelining, and that the drawing processor is equipped with hardware to perform high-speed painting and enlargement/shrinkage.

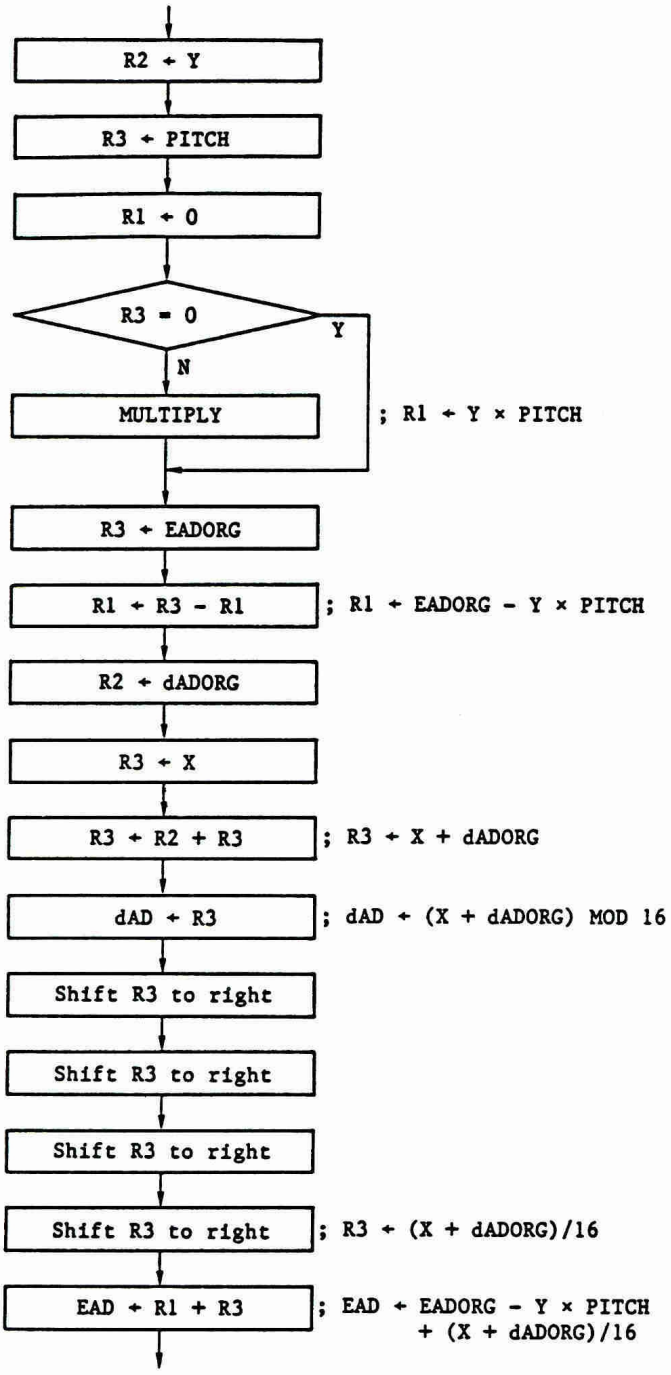


Figure 3 Preprocessor's Processing Procedure to Calculate An Absolute Address From X-Y Coordinates

Each step is executed in one clock cycle (125 ns at 8 MHz). The multiplication (MULTIPLY above) is conducted using the adder-subtractor within the preprocessor. Introduction of the preprocessor has made the pre-drawn processing about 10 times faster than to its previous processing by the CPU.

Preprocessor to Execute Pre-drawn Processing

The GDC had a built-in FIFO to receive commands from the CPU. This was done to adjust the difference between the drawing speed and the pre-processing speed. This FIFO had only a passive role to temporarily store command parameters. With improvements made on LSI integration levels, however, the general tendency now is to let the graphics controller perform pre-processings such as calculation of absolute memory addresses from X-Y coordinates. If this function is added to the drawing processor, it becomes impossible to perform pre-processing and drawing in parallel. So, a decision was made to let the AGDC have a built-in preprocessor to execute parts of the pre-drawn processing at high speeds. In addition, a method to directly write parameters into the register managed by the preprocessor rather than the parameter writing into a passive FIFO was adopted. As soon as a command code was written into the register, the preprocessor starts its operation.

The preprocessor consists of a 24-bit adder/subtractor, a general purpose register, a control ROM and a parameter RAM. (See Figure 2.) The main functions of the preprocessor are as follows:

① Conversion from X-Y coordinates to absolute addresses

② Interpretation of commands

③ Generation of parameters required for drawing graphics

④ Calculation of paint pattern extracted position based on Y coordinate

⑤ Sorting of coordinates used to paint triangles

⑥ Error checking of user-set parameters

⑦ Data exchange with the drawing processor, and start control of the drawing processor

High-Speed Calculation of Physical Address from X-Y Coordinates:

The preprocessor calculates a display memory's absolute address from the X-Y coordinates. Using the formulas shown below, an absolute word address EAD and a dot address dAD are calculated from coordinates (X-Y).

$$EAD = EADORG - Y \times PITCH + \{(X + dADORG)/16\} \quad (1)$$

$$dAD = (X + dADORG) \text{ MOD } 16 \quad (2)$$

EADORG and dADORG show absolute addresses (a word address and a dot address respectively) which correspond to the origin (0, 0) of X-Y coordinates. PITCH shows the number of words of the display memory in the horizontal direction.

Based on the above formulas, the preprocessor executes address calculations with the minimum of hardware at high speeds. For this purpose, we did not adopt an easy method of using a general-purpose processor, because it seemed necessary to design a new optimum processor which is free of unnecessary circuits and can process at high speeds. If the 24-bit adder-subtractor of the preprocessor is used, for example, multiplication between a 16-bit multiplier and a 16-bit multiplicand (whose product is 24 bits long) can be executed with a single clock per bit. To realize this speed, a right shifter is added to a register and a left shifter to another among the three arithmetic registers. And an operation end-detection circuit is added to the adder-subtractor. These devices can also be used to perform integer divisions at high speeds.

Figure 3 shows the flowchart of address calculation. In this figure, MULTIPLY (Y x PITCH) shows the multiplication by the adder-subtractor. When the horizontal resolution of the display screen is 1024 pixels, PITCH showing the number of words is 64 (7 bits actually). Therefore, the multiplication can be executed in 7 clock cycles. And conversion from X-Y coordinates to a physical address is therefore executed in 22 clock cycles which are 7 clock cycles of multiplication plus clock cycles of other steps (in Figure 3).

Hardware Mounted on Drawing Processor for Enlargement/Shrinkage, Boundary Point Retrieval:

The preprocessor is equipped with a single arithmetic unit. And the drawing processor is equipped with more than one arithmetic units, shifters and mask generator. These circuits are controlled to be executed simultaneously within one cycle. A command is 43 bits wide

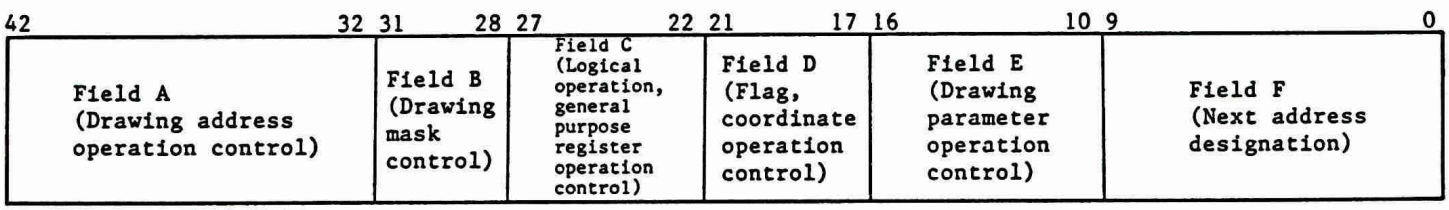


Figure 4 Command Format of Drawing Processor

Consisting of 6 fields (A to F), each command is 43 bits long. Each field can be executed independently of other fields. Because the number of arithmetic units and shifters is larger than the number of fields, a single field can control two or more operations.

consisting of six fields. Each field is assigned a different role of hardware control. (See Figure 4.) A 4-address branch based on judgment results of up to two types can also be executed within a single cycle. Therefore, all address branches and arithmetic commands can be executed in one clock cycle (125 ns). For example, a 24-bit register read, an operation, a rewrite and judgment of the operation all end in one clock cycle.

To achieve the above mentioned processing speed, the control storage is designed so that its asynchronous ROM gives its output as soon as the access address is input. In order to read the command synchronously with the address cycle, a synchronous type temporary storage is added to the command decode output driver. Under this circuit configuration, the control ROM can always be accessed within one cycle.

A tightly connected group of registers is assigned to the 24-bit drawing address operation unit, drawing parameter operation unit, 16-bit coordinate operation unit, and general purpose operation unit of the drawing processor. In addition, the drawing processor is equipped with a display data logic and operation unit, a 90 deg. rotation data buffer, a 32-bit shifter, a data enlarger/shrinker, a drawing mask generator and a boundary point retriever. (See Figure 2.)

To generate coordinates required for a slant copy, a fill-in of a triangle or a trapezoid or a rotational copy of an arbitrary angle, and a fill-in of a circle or an ellipse, a single straight line generator, two straight line generators and a single circle/ellipse generator are used respectively. These coordinate generators are not implemented by fixed hardware wired logic. In fact, only by changing software, it is possible to effectively use the common hardware of the drawing processor in various applications. For example, a straight line generator consists of three parameter registers and one working register.

The AGDC is equipped with functions to draw on the basis of both a pixel or dot and a word. A new drawing mask operator has been developed so that the AGDC can cope with different drawings without any inconvenience. (A clipping generator is also included.) Mask generation and clipping control operate in parallel with other draw processings. So, even when a clipping operation is designated, the drawing speed is not lowered.

Complex Drawings are Partitioned and Pipelined by the Preprocessor and Drawing Processor:

The drawing processor and the preprocessor are each equipped with a coordinate register and a drawing address register. For this reason, when the preprocessor has ended its processing, changing the register contents of the preprocessor does not affect the drawing operation. That is, the preprocessor and the drawing processor can then perform their own processings independently. As a result, while the drawing processor is in operation, the preprocessor can execute the preprocessing of the next drawing.

The preprocessor is equipped with a start designate function as well as a status detect function of the drawing processor. So, a single drawing command can designate an operation of the drawing processor in two or more separate times. For example, drawing a circular sector is actually processed in the following way.

① Calculates drawing parameters of an arc (Preprocessor)

② Draws the arc (Drawing processor)

③ Calculates parameters for drawing a straight line between the end point and the center after a start point and an end point of the drawing have been read from the drawing processor. (Preprocessor)

④ Draws the straight line between the end point and the center. (Drawing processor)
Calculates the parameters for drawing a straight line between the center and the start point. (Preprocessor)

⑤ Draws the straight line between the center and the start point. (Drawing processor)

When drawing an arc or an elliptic arc, the AGDC defines the arc in terms of the coordinates of its start point and end point. In business graphics applications, however, an arc is normally defined in terms of an angle. So, if the trigonometric function for calculating coordinates is simplified the results may designate coordinates which do not exist on the actually drawn arc due to some errors. Even if this kind of erroneous setting is made, the AGDC is ready to compensate the coordinates.

Lets draw an arc in the 1st quadrant as an example. (See Figure 5.) When the slope of the tangent is smaller than 135 degrees, even if the user-set start point does not exist on the actual arc, its Y coordinate at least needs to be equal to that of the actual start point. And when the slope of the tangent is greater than 135 degrees, the X coordinate at least needs to be equal. When the slope is 135 degrees, a small/large comparison using both X and Y coordinates is made (to judge if, in the case of the first quadrant, the drawing position is smaller than the user-set position). As a result, wherever the coordinates are set, the start point or end point of the drawing can be compensated. Therefore, even if the coordinates are designated very roughly, no runaway of the AGDC processing occurs. In the processing flow of the preprocessor and drawing processor described above, the preprocessor reads correct coordinate values in step 3 .

No Hardware Window Keeps the Display Processor Simple

Those parts other than the drawing processor and the preprocessor are explained below. Considering the application records of the GDC, the enlarged display function and screen partition function are removed from the display processor of the AGDC. These functions can partition and scroll the screen at high speeds only by changing the display addresses. But it is also known that these functions have restrained the general use and function of the software in such a way that the scroll range and partition method depend on the size of the display memory mounted. Partially this is why no hardware window has been mounted. As a result, the display processor section has been simplified. The multiwindow display can be realized using the copy function of the drawing processor. And the display processor is equipped with a dynamic RAM refresh function which, during refresh cycles, outputs a 13-bit refresh address as lower bits of the display memory address output.

The synchronization signal generator outputs horizontal/vertical synchronization signals (HS/VS) and a display delete signal (BLANK) according to the user-set parameter values. And it also provides a timing output (GCSR) for displaying the graphics cursor which does not depend on display memory addresses. With the top left corner of the display screen as its home position, the cursor can be displayed at any position. The form of the cursor is either a cross or a block.

The CPU interface controls the timings of the signal exchanges between the system bus and the AGDC. This interface is equipped with an address expansion function which operates when the CPU accesses the display memory, and with an arbitration function of the display bus.

The AGDC has input terminals of two clock systems: CLK and SCLK. The SCLK clock is supplied to the synchronization signal generator and the display processor. The frequency of the SCLK clock is determined depending on the specification of the display unit used and its display resolution. Therefore, it may become necessary to set the clock frequency lower than the maximum operating frequency (fmax) of the controller. If that happens, the drawing speed of the controller cannot be utilized effectively. Therefore, the AGDC is designed so that, different from the SCLK clock, another system of clock CLK can be supplied to the preprocessor and the drawing processor. As a result, the preprocessor and drawing processor can always operate at the maximum operating frequency.

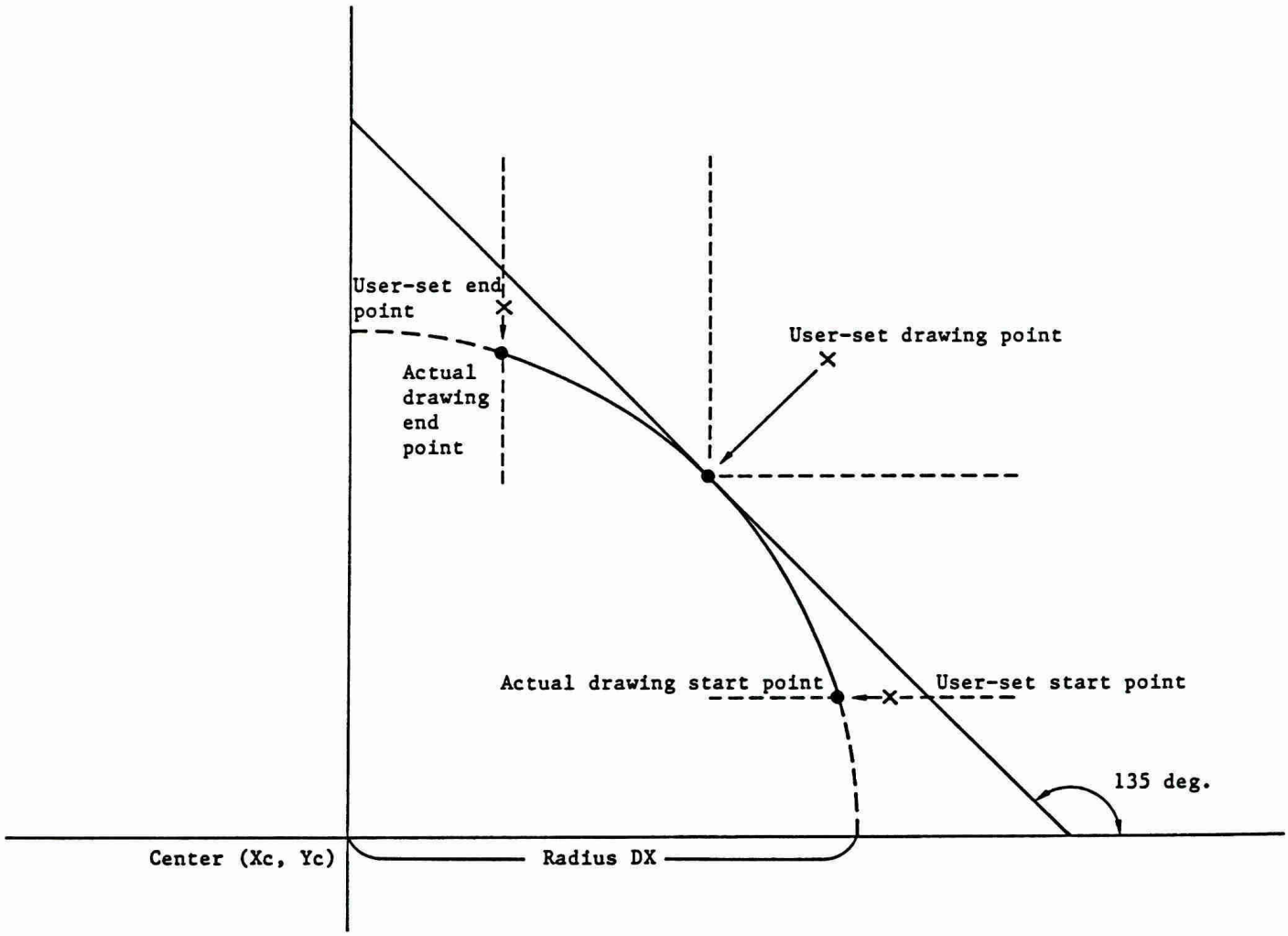


Figure 5 Drawing of An Arc

Given a start point and an end point, the AGDC draws an arc. When the user-designated point does not exist on the actual arc, the AGDC compensates the point.

Up to 32 M Bytes Display Memory Consisting of Dual Port Memory Chips Can Be Controlled

The AGDC directly controls the display memory bus which is separate from the system bus. The AGDC has terminals for outputting a 24-bit word address (DA23-16 and DAD15-0) and inputting/outputting its lower 16 bits (DAD15-0). The capacity of the display memory can be up to 32 M bytes which correspond to 256 memory planes having 1024 x 1024 bits each. It is not necessary to assign all of the memory as the display frame buffer. If the ROM and RAM storing Kanji font patterns and dynamic graphic patterns are mounted as a part of the display memory area, it becomes possible to perform high-speed character font expansion and dynamic graphic control using the copy function.

Kanji font memory (Kanji ROM) can be mounted, as it is, on the display memory bus without attaching anything like an address counter of line direction. Each Kanji JIS code is 14 bits long. And in order to output 32 x 32 dot data onto the 16-bit display memory data bus, it is only necessary to supply 5 row address lines and 1 column address line. That is, if a 20-bit word address (14 bits for Kanji code plus 6 bits for the font) is given, it is possible to mount the first and second level Kanji ROM (of 32 x 32 dot configuration). (The AGDC's word address is 24 bits long.)

In the case of a 16-bit CPU having 20 byte address lines, it is not possible even to directly map the 32 x 32 dot Kanji ROM onto the main storage. If an attempt is made to construct a system such that a character font memory is mounted at the system bus side to transfer font data to the controller, much time will be wasted reading and transferring the font data.

Display Memory is Used to Store Paint Patterns and as a Stack:

With many graphics controllers, each time a drawing is executed, the required paint patterns and character font data are set from the system bus into the built-in register. But because the storage capacity of the built-in register is limited, the data must sometimes be rewritten while drawing or complex paint patterns cannot be selected.

The AGDC can use the display memory as a working area with litle restraint on the storage capacity. Similar to character font and dynamic graphic patterns, paint patterns are stored in the display memory before the painting is executed. And while paint patterns are stored in the display memory, it is not necessary to transfer the patterns. Even when paint patterns are stored in the main storage, they can be transferred using a single block transfer command of the CPU.

The display memory is also used as a stack area for painting data. During the execution of a boundary point retrieval to determine an area to be painted, if a partial closed area has been determined, instead of interrupting the retrieval, the data required to express the closed area are temporarily saved onto the stack. (Details of this operation are described later.) The built-in register may be used for the data stacking but, also in this case, the storage capacity of the register causes an upper limit on the stack level. Although it depends on the algorithm of the boundary point retrieval, as drawing graphics becomes complex, the possibility of the stack level exceeding the upper limit increases.

In order to set a working area in the display memory, it is necessary to give its head absolute address and its size to the AGDC. When a defined stack area overflows, the AGDC stops its operation and informs the CPU of a drawing processor error.

CPU Can Directly Access the Display Memory and the Internal Registers of the AGDC:

For the CPU to be able to access the display memory in the same way as the main storage, it is necessary to map the display memory, as a part of the main storage, on the CPU memory addresses. For this purpose, the AGDC is equipped with address terminals to which 20 byte address lines (MA19-16 and MAD15-0) of the system bus are connected, and with a terminal ($\overline{CSDM}$) which is kept at low level when the CPU accesses the display memory.

The display memory bus has 24-bit word address lines. To extend the byte address of the CPU into a 24-bit word address, an 8-bit bank register is mounted in the AGDC. Figure 6 shows the address extension method. In the extension example, 20H (□) is set in the bank register and a byte address 24DA8H or 24DA9H (□) is given. The display memory lower byte enable signal ($\overline{DLBE}$) (□) is equal to the LSB of the byte address. The data bus width at the system side may be 8 bits. In that case, the upper byte enable signal ($\overline{UBE}$) (□) is always high.

It is possible for the CPU to access the internal registers (i.e. coordinate register and others) of the AGDC. The parameter registers and others of the AGDC can be selected by the 8 lower bits of the CPU's byte address. When the chip select signal ($\overline{CSIR}$) is low, the internal registers can be accessed. Because the display memory and the internal registers under the control of the AGDC can thus be directly accessed by the CPU, it is possible to dump and rewrite their contents using a general-purpose software debugger.

Built-in Bus Arbiter:

In relation to the mapping function, the AGDC has a built-in bus arbiter which gives priority to the use of the display memory bus. It also outputs a ready signal (READY) to request the CPU for a wait operation. Thus, it is not necessary to construct a bus arbiter, as an external circuit, whose timing designs are difficult. Priority sequence for the use of the display memory bus (from high to low) is as follows:

① Generation of display addresses and refresh addresses by the AGDC

② Access of the display memory by other processors with permission of using the display memory bus

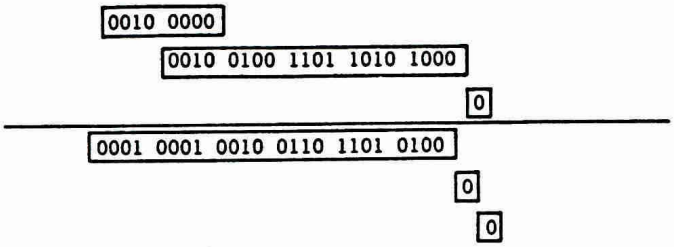
③ Direct access of the display memory by the CPU through the AGDC

④ Generation of drawing addresses by the AGDC

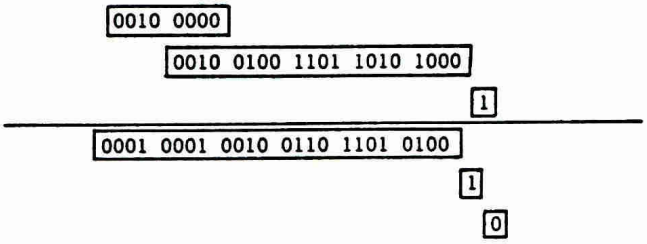
The AGDC generates display addresses with high priority and refresh addresses. For this reason, the AGDC normally operates as the bus master of the display memory bus. Considering the possibility that other external processors will use the display memory bus, the AGDC is equipped with an input terminal ($\overline{HLDRQ}$) for the signal which requests the use of the display memory bus. Accepting this signal, the AGDC puts the address and data output in the display memory bus in floating status and, if some drawing is then in progress, temporarily halts the drawing. And by lowering the acknowledge signal through the output terminal ($\overline{HLDAK}$), the AGDC shows the request source that its request has been accepted. Because the address supply for the display by the AGDC and refreshing are given the

highest priority, the AGDC, by raising the HLDAK signal, reversely requests the right of using the bus to be returned.

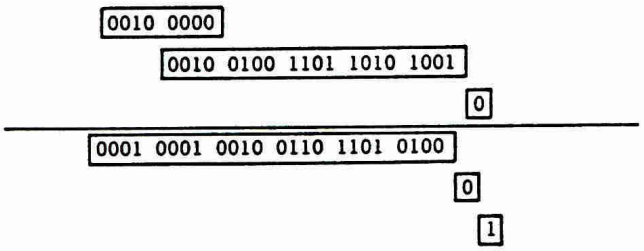
(a) Word access by a 16-bit CPU



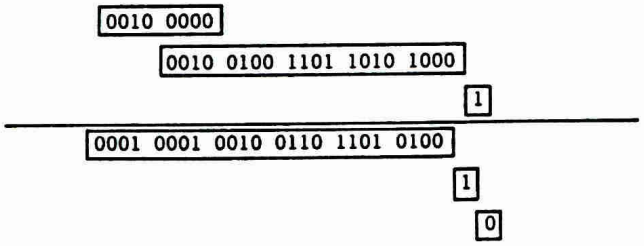
(b) Lower byte access by a 16-bit CPU



(c) Upper byte access by a 16-bit CPU



(d) Access of an even address by an 8-bit CPU



(e) Access of an odd address by an 8-bit CPU

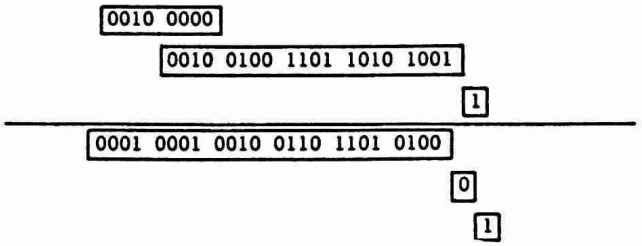


Figure 6 Display Memory Access Method by the CPU

☐ : The content of the built-in bank register

☐ : The byte address input from the system bus

☐ : $\overline{UBE}$ signal which is kept low during an upper byte access through the system bus

☐ : The display memory word address which the AGDC outputs

☐ : $\overline{DUBE}$ signal which is kept low during an upper byte access through the display memory bus

☐ : $\overline{DLBE}$ signal which is kept low during a lower byte access

The address extension examples above show a word access, a lower byte access and an upper byte access by a CPU with a 16-bit data bus width, and a byte access by an 8-bit CPU.

Four Types of Address Cycles are Prepared:

The AGDC offers four display memory bus address cycles as follows (See Figure 7.):

① Drawing address read cycle

② Drawing address write cycle

③ Display address cycle

④ Refresh address cycle

The GDC's drawing address cycle always consists of four clock cycles used for a read-modify-write operation. With the AGDC, one drawing cycle consists of two CLK-synchronized clock cycles, and each display and refresh cycles consists of two SCLK-synchronized clock cycles. If the execution of a refresh operation for dynamic RAM is selected, a refresh address cycle is inserted during the horizontal synchronization period. Two drawing address cycles, read (2 clocks) and write (2 clocks), may be executed continuously or 1 clock cycle of idle time may be inserted between two consecutive drawing cycles.

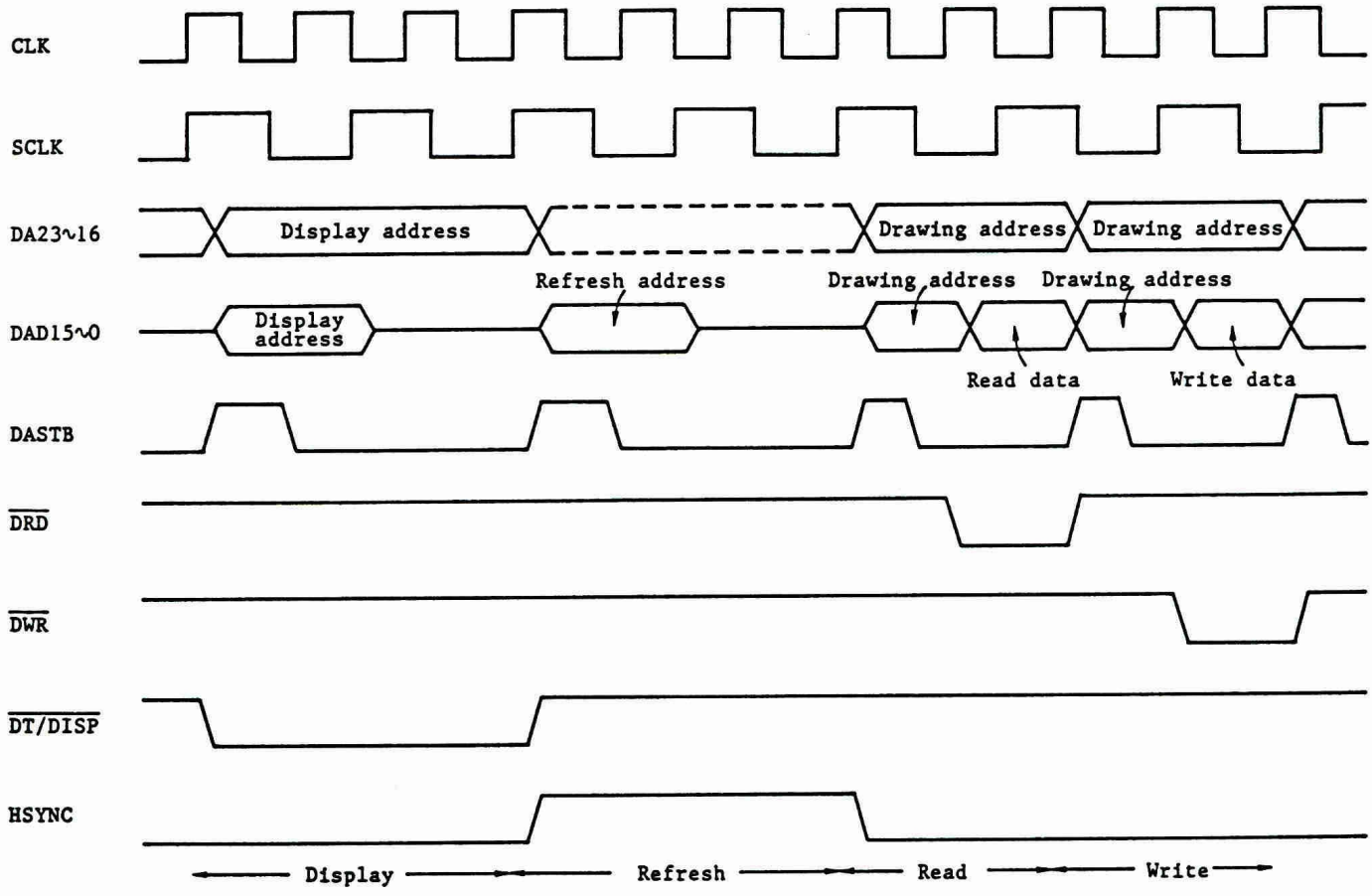


Figure 7 Display Memory Address Cycles

When the display memory is accessed, the address strobe signal (DASTB) is always output. The address strobe signal is synchronized with SCLK during display and refresh cycles, and with CLK during read and write cycles.

Because the address strobe signal (DASTB) is always output at the head of each address cycle, the external circuits can correctly know the address cycle. This DASTB signal is used as the reference timing signal to extract the address alone from the address/data multiplexed terminals or to generate RAS signal and CAS signal supplied to the dynamic RAM.

Because, during the drawing address read cycle, the read signal ($\overline{DRD}$) is output, direction control of the data bus is possible. During the drawing address write cycle, the write signal ($\overline{DWR}$) is output. These signals, after being appropriately delayed by external circuits, are supplied to the display memory. During the display and refresh cycles, only the DASTB signal is output. Whether or not it is a display cycle can be judged by seeing the output signal from the terminal $\overline{DT}/\overline{DISP}$ which is used for both the data transfer timing signal and display cycle timing signal. Whether or not it is a refresh cycle can be judged by the horizontal synchronization signal (HSYNC).

Data Transfer Signal is Output to the Dual Port Memory:

Considering the use of a dual port memory, two drawing timing operation modes are offered as follows:

① Data transfer mode

② Memory cycle steal mode

If a dual port memory is used as the display memory, the data transfer mode is selected. In this mode, clocks of different frequencies may be supplied to the CLK and SCLK terminals. Because of the need by the circuit which synchronizes signals related to the different frequency clocks, the following relationship must be met:

fmax (maximum operating frequency) $\geq$ CLK $\geq$ SCLK

In this status, during refresh cycle and data transfer cycle, no drawing can be performed. With CLK as the unit, during the previous two clock cycles and the following one clock cycle in addition, it may not be possible to perform drawing. (See Figure 8 (a).)

When the same clock signal is supplied to CLK and SCLK, the synchronization circuit can be bypassed by means of a flag setting. In this status, drawing can be performed in all timings except for refresh cycle and data transfer cycle. (See Figure 8 (b).)

If the drawing timing selection is set to data transfer mode, the data transfer timing signal ($\overline{DT}$) is output to the dual port memory. The timing for generating the $\overline{DT}$ signal is selected as follows:

① At the start of displaying each scanning line and when 8 lower bits of the display address become zero.

② At the start of displaying each screen and when 8 lower bits of the display address become zero.

If horizontal direction of the display memory is defined to be larger than the display screen to enable horizontal scrolling, ① above is selected. If the horizontal direction of the display memory is equal to the display screen to perform noninterlace scanning, because continuity of the display address is maintained from the right end of each scanning line to the left end of the next scanning line on the screen, ② above is selected. Because frequency of generating the $\overline{DT}$ signal is lower by ② than by ① , possible timings for drawing increase accordingly. Use of a dual port memory thus enables to cope with a high-resolution display unit whose dot frequency is as high as several hundred megahertz.

As resolution of the display screen becomes high, display data to be read per unit time increases. For this reason, the generation interval of the $\overline{DT}$ signal needs to be shortened. To cope with various display units, from low resolution types to high resolution types, while maintaining high clock frequencies, the AGDC offers eight modes of incrementing the display address, that is, << 1/4, 1/2, 1, 2, 4, 8, 16 and 32 >>. Conceptually, in the case of a 640 x 400 dot noninterlace display, <<1>> can be applied. If the resolution is lower than this, a smaller value is selected. And with higher resolution, a larger value is selected.

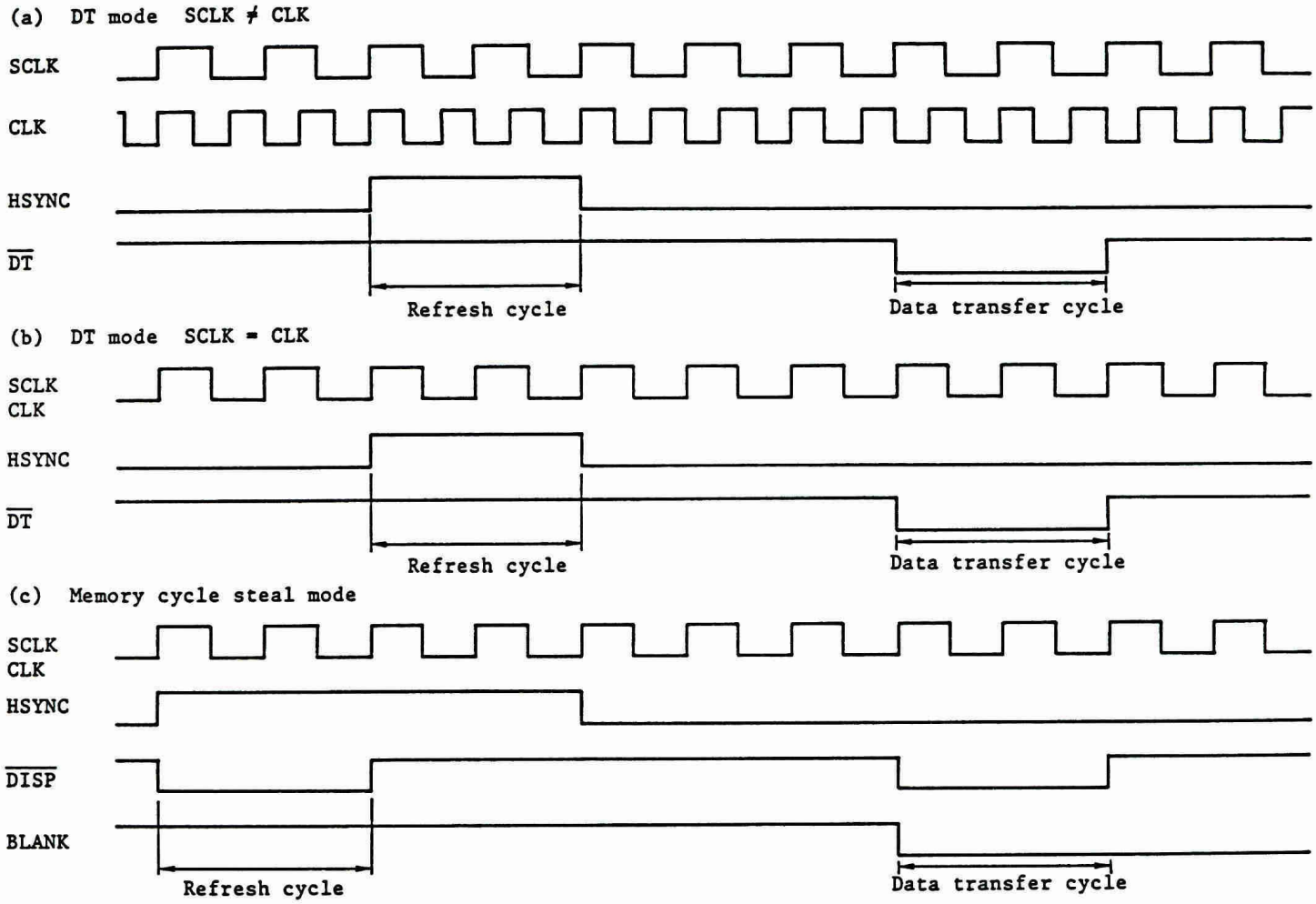


Figure 8 No Drawing - Timings

In those periods indicated by [ ], no drawing can be executed. These timings vary among the DT mode (SCLK ≠ CLK) in which two different clock sources are used, the DT mode (SCLK = CLK) in which the same clock signal is supplied, and the cycle steal mode.

Memory Cycle Steal is Also Possible:

When it is desired to keep the cost as low as possible or when no display is particularly required like a laser printer, normal RAM memory can be used as the display memory instead of dual port memory. For this purpose, memory cycle steal mode is offered. In this mode, the same source for clock signals must be supplied to CLK and SCLK.

During refresh and display periods, a display cycle which includes a refresh address generation and a drawing cycle in which drawing can be performed are generated alternately, 2 clocks by 2 clocks. (See Figure 8 (c).) During these periods, therefore, possible timings for drawing are 50% per unit time. Because, except for the display period, drawing can be executed at all timings excluding refresh cycles, this value increases a little. However, the restraint of memory cycle time determines an upper limit of resolution and, so, this cycle steal mode can only be applied to display units of relatively low resolution. If a standard display unit of 24 k Hz as its horizontal scanning frequency is used, data bus width of the display memory is set to be 16 bits and cycle time of the dynamic RAM used is 300 ns, the upper limit will be on around 640 x 400 dot of interlace display.

Command Interface Uses Parameter Direct Transfer Method

Now that the drawing processor is equipped with high-speed drawing functions, it is important to use them effectively at the system level. That is, the procedure of exchanging commands between the CPU and the graphics controller (i.e. command interface) is seen to determine the actual performance of the system. From the start of development, we fully studied this method.

Generally speaking, command interface methods can be summarized as follows:

① Data transfer method on the system bus

- (a) Parameter direct transfer
- (b) Parameter indirect transfer

② Detection method of the controller status

- (a) Status flag
- (b) Interrupt request
- (c) Wait request
- (d) DMA request
- (e) Hold request

③ Storing method of transfer data

- (a) Indirect address designation
- (b) Direct address designation

④ Storing destination of transfer data

- (a) Register
- (b) FIFO

The AGDC has adopted the combination of ① (a), ② (a), (b) (c), ③ (b) and ④ (a).

Parameter indirect transfer method (① (a)) was not adopted because the data generation time seemed to lengthen the total drawing time (described later). Closely related to this method, DMA request and hold request (② (d), (e)) are also put aside. Regarding the storing method of the transfer data, direct address designation has been adopted. With an indirect address designation method, if adopted, a command and parameters or an address and data must be separated during assembling transfer data. And the number of transferred bytes obviously would increase compared to the direct address designation method by which an address and data can be transferred simultaneously. The storing destination of transfer data has been set to be registers. Even if FIFO method was adopted, it causes an upper limit on the storage capacity because a re-read operation becomes necessary.

The registers in the AGDC are mapped on the CPU's address space. So, for the CPU to give commands to the AGDC, it is enough to execute normal data move commands for the main storage. (See Figure 9.) Those parts indicated by [ ] in Figure 9 correspond to the writing into the parameter RAM in the AGDC. Thirty-two word registers in the parameter RAM are assigned roles respectively corresponding to their addresses. As soon as a command is written, the preprocessor starts its pre-drawn processing.

Three status detection methods are provided (② (a), (b), (c)). These methods may be used being switched as often as required depending on the drawing types. If a system is constructed using a CPU devoted to parameter generation and the AGDC, hardware handshaking using the wait request method (② (c)) is most appropriate. The CPU does not then need to detect the status of the AGDC and can handle accesses to the AGDC in the same way as accesses to the main storage. But when the CPU is in wait status, no interrupt processing can be executed.

If software handshaking by means of status flag method (② (a)) is selected, however, even when the CPU is in a loop being incapable of transferring parameters due to busy status of the controller, interrupt processing can be performed.

Whatever status detection method is adopted, the wait control signal (READY) must be connected to the wait control terminal of the CPU. Occurrence interval of the READY signal differs under various conditions. Take the CPU accessing the display memory as an example. During a read, a wait signal occurs only for five clock periods starting from the fall of the system bus's read signal. And during a write, no wait signal is generated.

```
CHAR:   MOV WORD PTR PITCHS, 0002H; Transfer source address, pitch transfer:  2
        MOV WORD PTR DHH,    0017H; Horizontal direction dot count transfer:  24
        MOV WORD PTR DV,     0017H; Vertical direction dot count transfer:  24
        MOV WORD PTR EAD2H,  0000H; Transfer source upper word absolute address
                                    transfer:  0
        MOV AX, ES : WORD PTR BUF2; Transfer destination Y coordinate
                                    initialization
        MOV BX, ES : WORD PTR BUF3; Transfer destination X coordinate
                                    initialization
        MOV DI, ES : WORD PTR BUF1; Color information initialization
        MOV DX, 0D000H            ; Transfer source lower word absolute address
                                    initialization
        MOV SI, ES : WORD PTR BUF8; Command flag setting
CHAR_2: MOV CX, 041AH             ; Loop count initialization:  26 characters,
                                    4 lines
CHAR_1: MOV PLANES, DI            ; Color information transfer
        MOV EAD2L,  DX            ; Transfer source lower word absolute address
                                    transfer
        MOV X,      BX            ; Transfer destination X coordinate transfer
        MOV Y,      AX            ; Transfer destination Y coordinate transfer
        MOV WORD PTR COM, SI      ; Command, flag transfer
        ADD DX, +30H              ; Character code change
        ADD BX, +18H              ; Transfer destination X coordinate change
        INC DI                    ;
        AND DI, 0007H             ; Color information change
        DEC CL                    ;
        JNZ CHAR_1                ; Is the drawing of 1 line/26 characters ended?
        SUB AX, 0018H             ; Transfer destination Y coordinate change
        MOV BX, 0000H             ; Transfer destination X coordinate change
        MOV CL, 1AH               ; Character loop count initialization
        DEC CH                    ;
        JNZ CHAR_1                ; Is the drawing of 4 character ended?
        DEC WORD PTR BUF7         ; 26 characters/4 lines drawing
        JNZ CHAR_2                ;                                 × 12 times ended?
        ;
```

Figure 9 Part of the Assembler Program Which Expands 1280 Characters (of 24 x 24 Dot Configuration)

The CPU is the 80286. As shown above, command parameters can be easily set only by move commands (indicated by □).

Parameter Indirect Transfer Method Has Problems

Among the parameter indirect transfer methods, there are the receiving command parameters by DMA transfer and (i.e. linked list) the controller itself endeavors to read the command parameter lists which have been generated beforehand on the main storage. By these methods, transfer data strings which include command codes and transfer address information depending on the controller are generated from the original data-like coordinate strings. For this reason, when compared to the parameter direct transfer method which, as soon as coordinate data are generated, transfers them immediately, the data generate processings get in the way. So, operating time of the CPU increases accordingly and software for the parameter transfer becomes complex.

When a series of draw processings is clearly seen through, such as in the case of stroke character drawing which draws characters by means of many straight-line drawings, the indirect transfer method which does not suspend the CPU's pre-drawing depending on the controller's status may be advantageous. But, with DMA transfer method, how to optimize the size of data blocks being transferred becomes a big problem. To fix the size of a transfer block is basically unnatural because no data can be transferred until the fixed amount is ready. And as the transfer block is divided into smaller units, processing time required for the generation of transfer data and operation settings of the DMA controller increases, thus lowering the total drawing speed.

If the controller is of linked list method type to read command lists, this kind of problem does not occur. But the control software will be more complex than that of DMA transfer method because the CPU needs to manage all recognitions of the created command parameter strings and transferred parameter strings, linking to the next parameter strings, and defining the generate positions of the new parameter strings. Setting this information into the controller is also required. And because the controller performs hold control to exchange the fight of using the system bus, this controller may limit CPUs which can be put on the interface.

The Relation Between Parameter Generation Time and Drawing Time Is Important

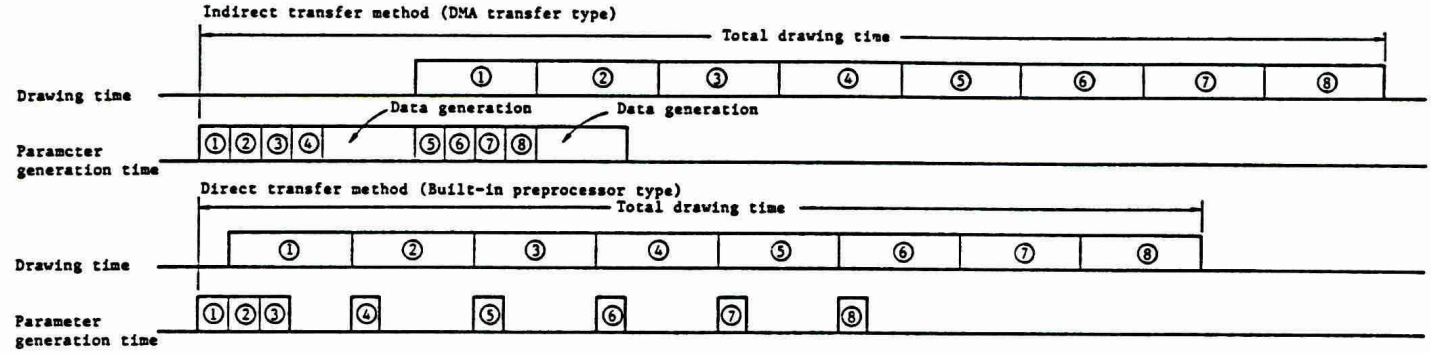
The parameter indirect transfer method aims to execute as far as possible without stopping the CPU processing the parameter generation. In the actual use, however, we judge that it does not have any advantage when compared to the parameter direct transfer method. When comparing these transfer method, the relationship between parameter generation time by the CPU and drawing time by the controller becomes an important factor.

When drawing time is slower than parameter generation time such as in paint drawing, the total drawing time is approximately equal to drawing execution time. But the indirect transfer method is slower than the direct transfer method by as much as the time taken to generate the first data. (See Figure 10 (a).)

If an extreme case of drawing only 1 dot is taken as an example, the total drawing time depends on the parameter generation time. (See Figure 10 (b).) However, as the indirect transfer method generates parameter strings beforehand, the drawing can be executed continuously without depending on the parameter generate processing. This is, of course, limited to the drawing which uses the previously generated data blocks. Although the direct transfer method, as it depends on parameter generation time, cannot continuously draw, the total drawing always ends within a short time.

That is, because the drawing time in a block does not depend on parameter generation time as the direct transfer method does, the indirect transfer method seems faster. In fact, its total drawing time is longer by as much as the time taken to generate data.

(a) Drawing time ≧ Parameter generation time



(b) Drawing time < Parameter generation time

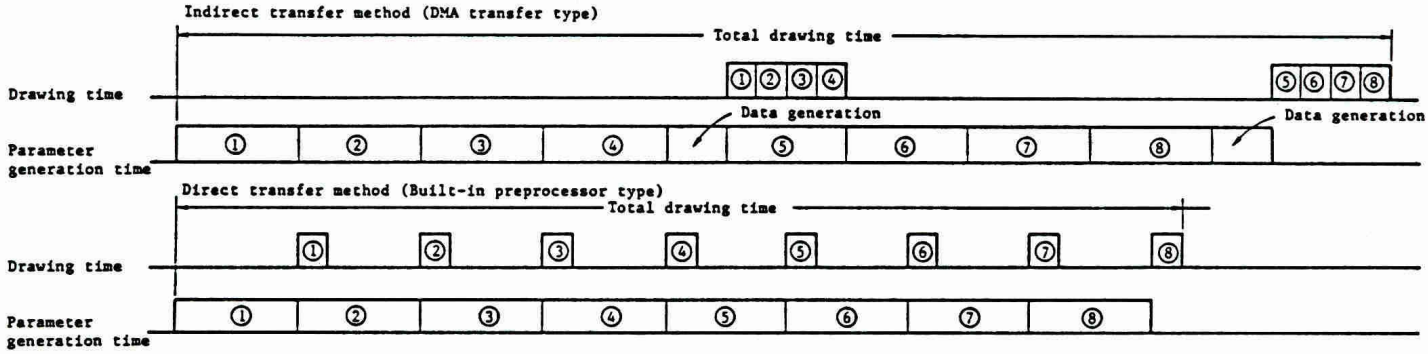


Figure 10 Comparison of Actual Drawing Times

Between parameter direct transfer method and parameter indirect transfer method, the operations of the CPU and the controller are compared. Regarding case (a) in which drawing time is longer than parameter generation time and its opposite case (b), total drawing time is evaluated. Parameter indirect transfer method causes the total drawing time to be longer than direct transfer method by as much as the time taken to generate data.

Drawing Processor Used to Execute High-Speed processings of Enlargement, Shrinkage, Rotation and Painting

Nowadays, we often see display units which offer multiwindow functions in order to display two or more processing results on the same screen and help create new documents while referring to different document contents. Some controllers which enable multiwindow function, display only by changing display addresses without transferring display memory contents. (Ref. 5) This is called hardware window function.

A hardware window operates fast because it does not need to transfer any data at all. To display a window having bit boundaries, however, it is necessary to take display data once into the controller chip. In the case of a display unit with high resolution, a real-time control corresponding to a dot frequency which is as high as several hundred megahertz is required. Because, in general, controllers can only guarantee their operations at frequencies 1 digit lower than dot frequency, this restrains the resolution of display screen. And there is also an upper limit on the number of windows which can be displayed. Although it is desired to increase the number of windows at higher resolution, the number is limited to smaller one on the contrary. This is because all of the change positions of display addresses occurring on each display line and the head display address must be transferred during the horizontal retrace period. Although scrolling in windows then becomes faster, this is also limited by the capacity of the window memory.

Clipping Function is Used for Window Display and Picking

We decided not to mount a hardware window function which is advantageous from the viewpoint of speed but causes considerable restraints during the stage of unit design. The copy function is used instead to transfer display data and form windows. Different control methods are adopted between the pop-up menu whose display contents are fixed and windows. Window control requires a window memory which has the same memory configuration as the frame buffer, but menu display data need not be two dimensional. Only when required, it is enough to expand the original menu data placed on 1-dimensional continuous addresses in a 2-dimensional manner. This fact has enabled to expand 1-dimensional data into 2-dimensional ones or convert 2-dimensional data back to 1-dimensional data.

In order to efficiently form a multi-window, a hardware clipping function is also provided. A rectangle clipping area is defined by designating two diagonal coordinate points. Whether only the inside is drawn, only the outside is drawn, or no clip operation is performed can be set. (See Figure 11.) This function can be used effectively for a window positioned at the front of the display. In the case of painting, definition of this clipping area is used as the outer frame designation of a boundary point retrieve area.

If the clipping function is used, the pick operation specifies and extracts a particular graphic drawn on the screen. If the entire drawing is left to the controller, unless the CPU traces the drawing positions, no pick operation is possible. So, this function is added to the AGDC. Using a mouse or a digitizing tablet, a clipping area is moved to the intended position beforehand and an area as small as 2 x 2 dots is defined. A drawing area is set outside the clipping area, a logic operation mode which does not cause the display results to change is selected, and drawing is executed

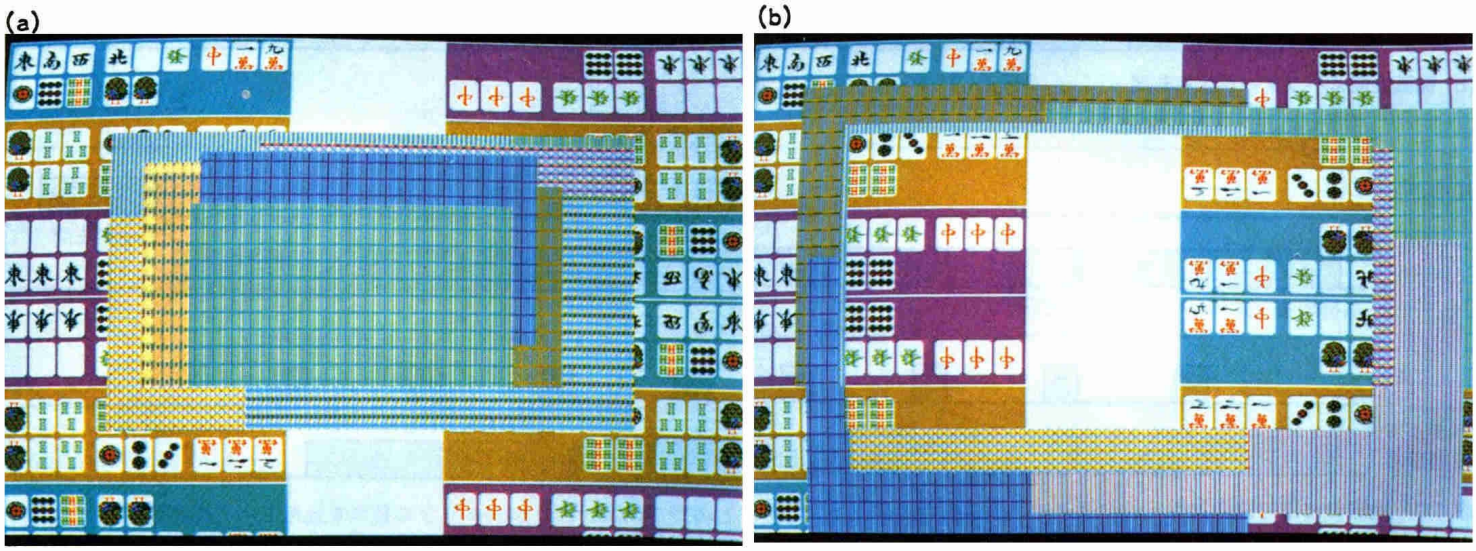


Figure 11 Clipping Examples

Clipping areas are designated, mode of painting inside (a) or outside (b) is selected, and a rectangle fill is executed.

again. By means of an interrupt or a flag, the CPU knows that clipping has occurred and picks the graphic which was then being drawn.

Plane Type and Pixel Type Display Memory Can Be Constructed

Table 1 lists the main drawing functions of the AGDC. But all of their combinations are not necessarily available. (See Table 2.) Functions which are not built in the AGDC, such as drawing of an ellipse whose axes are not parallel to X and Y axes, must be supplemented by the CPU's software. For this reason, if some AGDC-executed functions like clipping are used in other drawings, it is desirable to be able to apply them in the drawing by the CPU. In this case, the CPU can draw the ellipse using the bit mapped method which enables to designate relative positions. In drawing into the frame buffer (in the display memory) which corresponds to the display screen, it is possible to designate positions by means of X-Y coordinates. In data referencing concerning the Kanji ROM and working areas, it is possible to designate positions by means of absolute addresses.

There are two methods to store color information as follows:

① Color plane method

② Packed pixel method

The plane method stores one pixel of color information at addresses a color plane from each other. So, in the case of one-pixel-to-one-pixel drawing like that of a straight line, it is necessary to sequentially execute all planes of drawing bit by bit. But the plane method is suitable for copying, painting, rotating, enlarging and shrinking which would be faster if neighboring bit strings were processed as a whole. Packed pixel method, however, because it packs all pixel information in the smallest access unit (1 word) of the display memory, can access one pixel simultaneously. In graphics drawing which draws one pixel as the unit, the packed pixel method is faster than the plane method. But if the number of dots constituting one

pixel is changed, it will cause such hardware as drawing flow and drawing mask to change, thus being less flexible.

The AGDC can apply all of its drawing functions to the plane method of display memory. In the case of bit mapped drawing, the AGDC can cope well with the pixel method of display memory. Because this mode can be dynamically selected between plane method and pixel method during drawings, it is possible to mount a display memory with two configurations (plane method and pixel method) to make full use of their respective features.

Unlike the GDC, the AGDC executes color drawing by issuing a single drawing command even for plane method of display memory. In graphics drawing or painting, up to two logical operations can be designated for two or more planes at the drawing destination. With four memory planes, for example, a different logical operation can be executed for every two planes. In copy and put/get operations, similar processings can be executed for transfer source and transfer destination.

It is easy to output a hard copy of the color screen to a dot-matrix color printer. This is done by executing logical operations appropriate for each ink ribbon among the transfer source data which are stored in two or more planes and issuing once a 90 degree rotation get command which reads 90-degree rotated data into the system bus. This operation takes out the dot strings (from the AGDC) which are used to directly drive the print head pin for 16 dots vertically and 1 line horizontally.

Table 2 Possible Combinations of Drawing Functions

	Pixel configuration	Hardware clipping	Selection of enlargement, shrinkage and line type	Selection of line type	Selection of paint pattern	Transfer destination (drawing destination) position designation	Transfer source (line type, paint pattern) position designation
Dot	O	O	x	O	-	Coordinate	Built-in register
Straight line	O	O	Enlargement only	O	-	Coordinate	Built-in register
Rectangle	O	O	Enlargement only	O	-	Coordinate	Built-in register
Circle, arc	O	O	x	O	-	Coordinate	Built-in register
Ellipse, elliptic arc	O	O	x	O	-	Coordinate	Built-in register
Paint	x	O	-	-	O	Coordinate	Built-in register, head address
Rectangle fill	O	O	-	-	O	Coordinate, absolute address	Built-in register, head address
Circle fill	x	O	-	-	O	Coordinate	Built-in register, head address
Trapezoid fill	x	O	-	-	O	Coordinate	Built-in register, head address
Triangle fill	x	O	-	-	O	Coordinate	Built-in register, head address
Normal copy	O	O	O	-	-	Coordinate, absolute address	Coordinate, absolute address
90 deg. rotation copy	x	O	x	-	-	Coordinate, absolute address	Coordinate, absolute address
Slant copy	x	O	x	-	-	Coordinate, absolute address	Coordinate, absolute address
Arbitrary angle rotation copy	x	O	O	-	-	Coordinate, absolute address	Coordinate, absolute address
Normal put/get	O	O	x	-	-	Coordinate, absolute address	-
90 deg. rotation get	x	O	x	-	-	-	Coordinate, absolute address
CPU direct drawing	O	x	x	-	-	Coordinate	Arbitrary

Enlargement Factor of 16/N, Shrinkage Factor of N/16:

Enlarged and shrinked copy operations are set to 16/N for enlargement and N/16 for shrinkage (N is an integer from 1 to 16). The fact that many factors can be selected near the original size (i.e. factor of 1) is practical. The restraints on enlargement and shrinkage factors were required to enable simple hardware to achieve high-speed enlarging and shrinking copy operations. Let's take 15/16 reduction as an example. This is just done by extracting one predetermined bit from among the 16 bits read from the display memory. In cases of shrinkage the extracted bits are predetermined depending on the shrinkage factors.

Shrinked data are written again into the display memory after being processed through shift, mask and other operations. In the case of a last word, the number of bits whose enlargement or shrinkage has not been completed may not be 16 bits. In this kind of processing, drawing mask operation is difficult. The AGDC, based on the factor and the number of bits whose drawing has not ended, generates masks by referring to the table in which mask formats have been entered beforehand.

Because vertical enlargement and reduction are not sensitive to processing speeds, they are handled on a count basis by software. For doubling, the same line is simply drawn twice and, for shrinkage, lines are just thinned out. In graphics dealing with characters and figures, thinning a single line and leaving no gaps may cause visual information to drastically decrease. It now seems necessary to consider some sort of supplementary processings for shrinked data depending on their applications, such as a straight line drawn not to delete a ruled line.

Three reference methods for paint patterns are shown below.

① Built-in register reference

② Display memory reference

(a) Patterns common to all planes are referenced.

(b) Independent patterns of each plane are referenced.

① is the so-called "solid patterns" which are repeated in vertical direction and in plane's direction. This will be used in clear operation. If reference method ② (b) is adopted, as different patterns can be referenced in each of the horizontal, vertical and plane directions, painting a different color at each dot (i.e. tiling) becomes possible. The AGDC is equipped with functions used to calculate initial values of and change the reference positions of paint patterns, depending on the Y coordinates of the line being painted.

High-Speed Boundary Point Retrieval Leads to Fast Painting:

As far as painting of an area with the same size is concerned, filling is faster than painting. Coordinates used to execute a fill-in can be generated at high speed. And in painting, to specify the area to be painted, read contents of the display memory and retrieve boundary points are necessary. For speeding up painting, to end this boundary point retrieval in a short time is a task of the highest priority. Immediately after 16-bit retrieve data are given, our newly developed boundary point retriever can return the following information to the drawing processor.

① Does a boundary point exist?

② If a boundary point exists, where is its dot position?

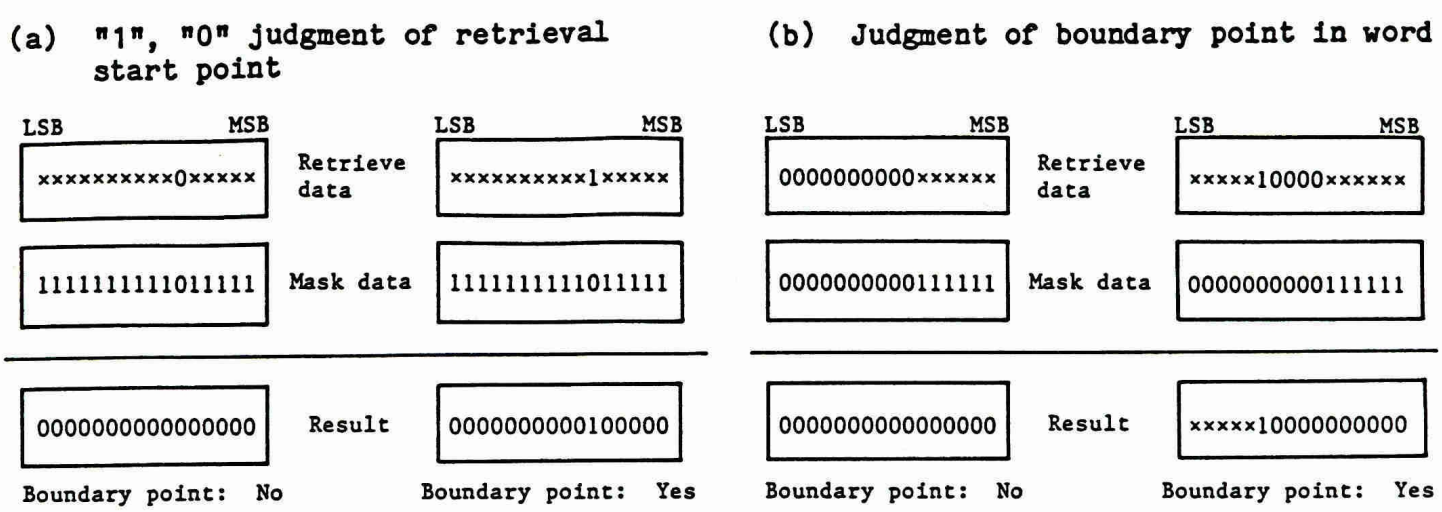


Figure 12 Judgment of Boundary Point

From the result obtained by masking retrieve data, it is known whether a boundary point exists in the word or not. If it exists, its dot position is known. In the above figure, "X" shows an arbitrary value ("0" or "1").

If no boundary point exists within the retrieve data, the content of the next neighboring word is read and, from there, retrieve data are created. In the case of a color display, the boundary point is given as a boundary color. If there are three planes, data of the three planes are continuously read, two clock cycles per plane, and logical operations are executed between the plane data according to the boundary color designation, thus generating retrieve data. When reading the data of all the planes has ended, generation of 16-pixel (word) retrieve data ends.

To specify an area as a retrieving object, it is possible to mask the retrieve data. (See Figure 12.) In Figure 12, only those bits for which mask data are "0" are retrieving objects. To judge whether or not the retrieval start point itself is included within the boundary, mask data are given so that only the retrieval start point's data are taken out ((a)). If the result is "0", it shows that no boundary point exists in the word. When continuing to retrieve to the left or to the right from a point within the word, those retrieve data from the point to the word boundary are masked so that the once-retrieved position will not be a retrieving object again ((b)). When there are many boundary points in the word, it is just enough to change only the mask for the same retrieve data. Position data output from the boundary point retriever is used not only in calculating the size of a painting area but in generating retrieve mask data within the word.

Upper and Lower Lines are Read, and Sneak Points of Area are Extracted:

Figure 13 (a) shows the sequence of painting a closed area and how its boundary area data are pushed and popped. The retrieval starts from a position marked X and the area is painted in the sequence from ① to ⑭.

By the sneak points, the closed area is divided into ⑭ smaller areas. The arrow by the side of each number shows the direction of painting to be executed.

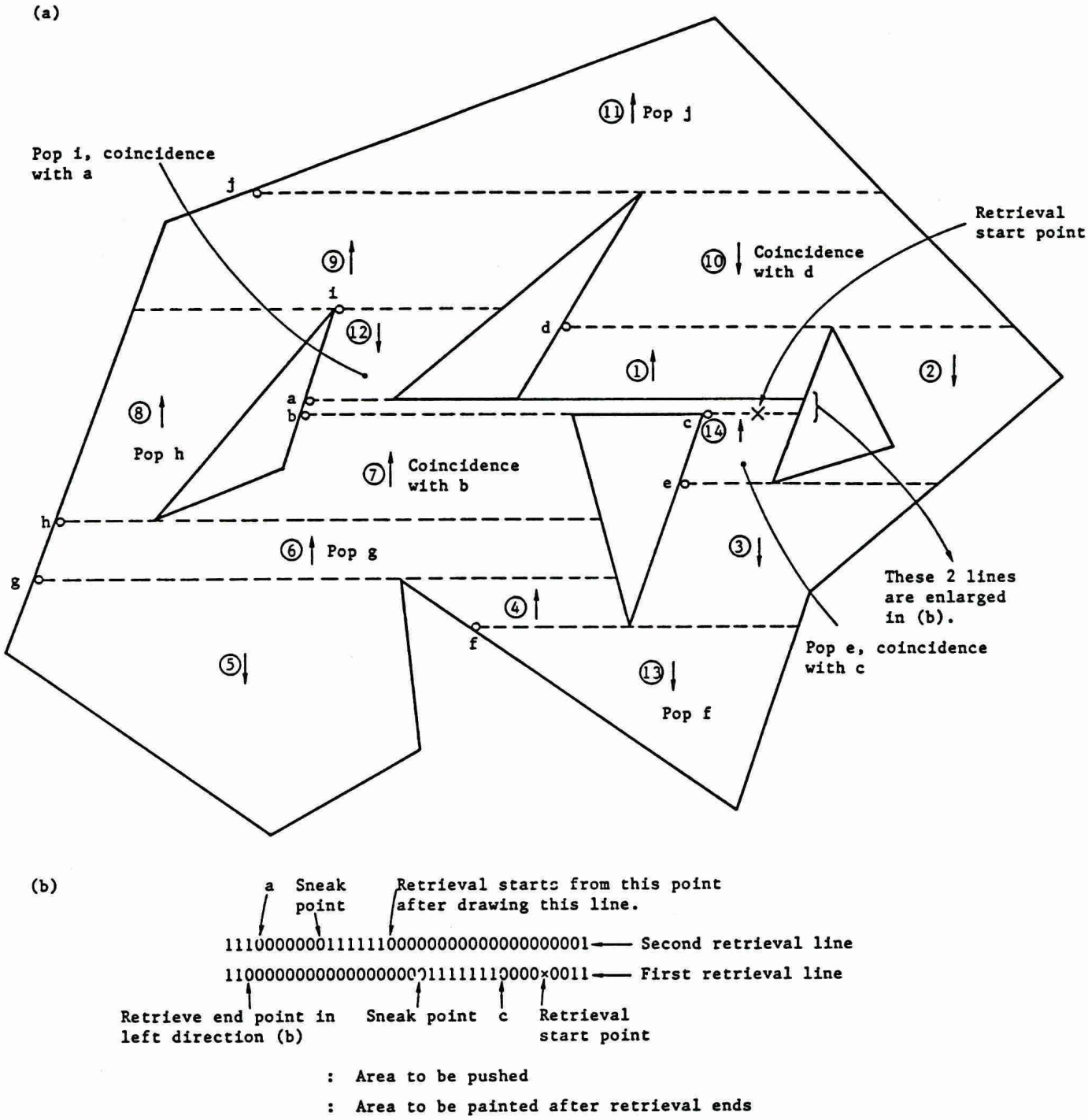


Figure 13 Paint Operation Procedure

An arbitrary closed area is divided into two or more smaller area and painted ((a)), from ① to ⑭, in the direction shown by each arrow. A part of the closed area in (a) is enlarged in (b) to show the procedure of retrieving boundary points. The boundary point retrieve processing includes specifying a closed area based on retrieved change points from "0" to "1" or from "1" to "0", extraction of sneak points, retrieval-end judgment, push and pop of the closed area information and actual painting inside the closed area.

Painting which includes boundary point retrieval proceeds in the following way. Retrieval starts from the retrieval start point shown in the figure. The line where the retrieval start point lies and its upper line are retrieved at the same time. When, during the retrieval of one line, boundary points to the left and right are found, those parts' information (i.e. closed area information) are pushed onto the stack. In this example, a, b and c are pushed. In area ①, painting is executed from ___ and operation proceeds to retrieval of its upper line. When, at the end of the retrieval in ①, a sneak point into ② occurs, d is pushed and painting of ② starts. While the execution of painting thus continues, closed area information is piled on the stack in the display memory.

When painting of ⑤ has ended, a, b, c, ..., g have been pushed onto the stack. In area ⑥, the stack pops and retrieval and painting start from g which was pushed at the end of retrieval in ④. And in ⑥, h is pushed later.

When post-retrieval painting is executed, it is always checked to see if the closed area information specified by retrieval coincides with the stack contents. In painting of ⑦, as the coincidence with b is obtained, flag is raised at b's closed area information so that no retrieval will be executed again after popping.

In the boundary point retrieval described above, the display memory contents of two (upper and lower) lines are read alternately. This method enables to make clear the flow of boundary point retrieve processing (Ref. 7), that is, extraction of sneak points around the area to be painted, stack operation when the area is determined during the retrieval, and retrieval-end judgment. Two lines of data are read and the results of respective boundary point retrievals are compared to retrieve boundary points. When an area to be painted is determined during retrieval, as described before, six words of information including position and size of the area is pushed onto the predetermined stack area on the display memory.

A part of the figure (a) is enlarged in part (b) of Figure 13. First, that the retrieval start point itself is not included within the boundary is confirmed and, then, the change point from "0" to "1" is retrieved. As the result, point c is obtained. Next, boundary point judgment is made on the point one line higher than c. If the point is "1", as it means that the boundary in the left direction is found, retrieval proceeds to right. In this example, "0" is first obtained, so retrieval proceeds to left. When leftward retrieval starts, change point from "1" to "0" on the first retrieval line and from "0" to "1" on the second retrieval line is retrieved.

Generally speaking, a point changing from "0" to "1" becomes a boundary point and, if its upper or lower point is "1", the point is the retrieval end point. With "0", it may be a sneak point. Therefore, a point of change from "1" to "0" in further is retrieved. The processing continues until the retrieval end point is found.

After the retrieval end point b in left direction is found, rightward retrieval starts. Then, during retrieval, the closed area information of a and b have already been pushed. During the rightward retrieval, the closed area of c is first determined and its information is pushed. When the rightward retrieval has ended, the line from which painting is executed is determined. With the left end point of this line as the retrieval start point, retrieval for the rest of area ① begins.

90 Degree Rotation Is Executed Using the Buffer Register:

The built-in 16-word buffer register is designed to speed up the 90 degree rotation operation. This buffer is structured to read rows written data is columns, by the address bus and data bus connection. In addition, a flow control is adopted so that, by judging the mutual relationship between the bit positions of transfer source areas and those of transfer destination areas, no garbage occurs during the data transfer to the buffer. This buffer also functions as an FIFO to temporarily store the transfer data in get/put operations.

Arbitrary angle rotation copy which is accompanied by enlargement/reduction is executed by means of straight line drawing. Generally, when a figure is rotated, some points of no drawing appear in the figure drawing area. If the rotation angle with relation to an axis reaches 45 degrees, this phenomenon is usually noted. Judging this kind of singular points, it is possible to select whether or not drawing of the neighboring points are also executed similarly for these singular points.

Evaluation of Execution Speed by Various Benchmark Tests

The evaluation method of drawing speeds of graphic controllers differs very much among different manufacturers. The environmental conditions for evaluation are often not very clear. Therefore, it is difficult to compare the drawing speeds of different controllers simply based on their specifications. The performance of some controllers is expressed in such a way as "some nanoseconds per dot" or "some vectors per second". But these values are something like the maximum instantaneous wind speed, which does not consider such system factors as time ratio showing how long the display memory can be occupied for drawing or command interface. Therefore, the actual drawing speed often comes to differ considerably.

Values useful for unit designers who judge if particular controllers can be suitably adopted, are the drawing speeds available when the intended systems are established. For that purpose, the method of calculating the number of straight lines or the number of characters which can be drawn in one second based on the time taken to draw a single straight line or a single 24 x 24 dot character is not good either. Because, when drawing operations are continuously executed, how far the respective drawing operations are from each other is not included in the evaluation method.

Table 3 Number of Drawing Cycles

Drawing contents	No. of drawing cycles
Straight line	4 clocks/dot (pixel)
Horizontal straight line of rectangle (of plane configuration)	4 clocks/word
Circle, ellipse	6 clocks/dot (pixel)
Copy	
Data replacement alone	4 clocks/word
Logical operation execution	6 clocks/word
Fill	
Built-in register reference, data replacement alone	2 clocks/word
Built-in register reference, logical operation execution	4 clocks/word
Display memory reference, logical operation execution	6 clocks/word

The total drawing speed should not be based on these values alone. For painting, the number of cycles varies depending on the reference patterns. However, it will affect little the total paint drawing speed.

We decided, in cases of drawing graphics and graphic characters, to evaluate an overall drawing speed available when the same type of drawing is continuously executed. With coordinate values given first, we actually measured, using a logic analyzer, the time taken by the CPU from its starting of pre-drawn processing to its ending of the last drawing. In the case of controllers using the parameter indirect transfer method, the expressed time includes the creation and processing of DMA transfer data and command lists. Get and put which depend on the CPU's data transfer speed are excluded. Dual port memory chips are used as the display memory and, as its refreshing consumes 6%, 93.7% of possible drawing timings are secured.

One Word Processing in 2 to 6 Clock Cycles

Before showing the measured values, Table 3 lists the number of clock cycles (i.e. theoretical value) required for the execution of a drawing. A straight line is drawn in the same number of clock cycles as with the GDC, but circle and ellipse are rather slow. This is because the drawing algorithm of an ellipse is based on circle, and that an entire circle can be drawn by a single command. When drawing a rectangle for the plane configured display memory, 16 dots can be drawn in one drawing cycle on the horizontal line. A copy which can designate both a transfer source and a transfer destination on bit boundary is executed at the speed of 6 clock cycles per word while executing logical operations for the transfer source or transfer destination. If the existing data at the transfer destination are ignored and replaced by the transfer source data, 1 word of copy ends in 4 clock cycles.

As for painting, execution cycles differ between the case of referencing paint patterns stored in the built-in register and the case of referencing patterns on the display memory. Compared to the built-in register reference, referencing patterns on the display memory needs more time (as much as 2 read clock cycles). As for rectangle painting, if the setting of replacement alone is made similarly to the case of copy, 16 bits can be painted in 2 clock cycles.

Fast Paining of Arbitrary Closed Area

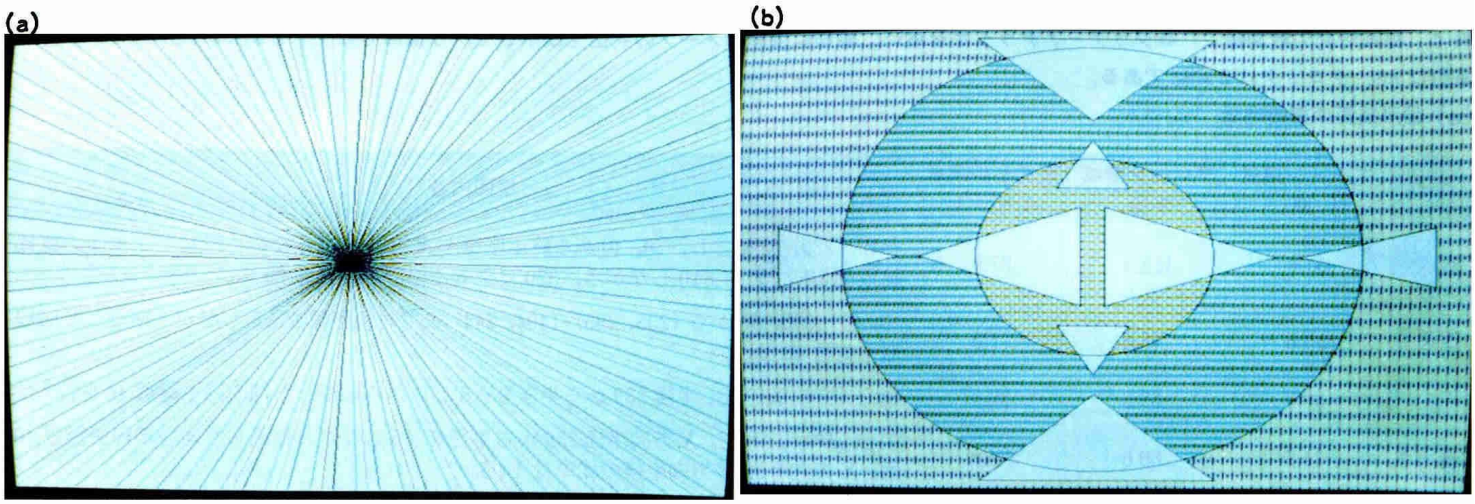
Execution times of graphic drawing, painting and copy are shown in Table 4 (a), (b) and (c), respectively.

If the drawing time of one display memory plane alone is compared with the drawing time of 3 planes (8 colors), in the case of straight line (Figure 14 (a)), it increases three times. In the case of circle and ellipse, however, it increases to a little over two times (Table 4 (a)). This is because, among 6 clock cycles required for drawing a circle or an ellipse, only the execution time of 4 clock cycles needed to draw 1 plane increases three times to 12 clock cycles.

Screen clear shown in Table 4 (b) is an example of painting. By referencing the paint patterns stored in the built-in register, the rectangle is painted. If the left side and right side of a rectangle are bit boundaries, painting of the word requires mask processing and a read-modify-write operation is executed. But, this screen clear has no bit boundary and 115,200 words are executed within 28.8 ms. This fact shows that 1 word is cleared in 2 clock cycles. Painting speed, except for that

Figure 14 Benchmark Processings Used To Measure Drawing Times

(a) Straight lines (in Table 4 (a))
(b) Paint (in Table 4 (b))



of rectangle depends little on the reference method of paint patterns. Although read time differs depending on the reference method of paint patterns, this is because it takes long to calculate coordinates for securing paint areas and retrieve boundary points.

Arbitrary closed area painting is very fast (Table 4 (b)). As shown in Figure 14 (b), about 80% of the screen is painted. However, it is a little less than two times compared to the time taken to copy the whole screen consisting of three planes to another three planes. Copy just does the calculation of simple drawing addresses, but paint includes such complex processing as boundary point retrieval. This speed is the result of the previously described hardware to execute boundary point retrieval at high speeds and the retrieval algorithm. And there is not much difference in the execution time of drawing between 90 deg. rotation copy and normal copy. It can be assumed that enlargement and shrinkage take about two and three times as much time as normal copy, respectively.

Drawing time of expanding the 24 x 24 dot character font has been evaluated separately from the copy of a large area. The font is drawn not on word boundary but on bit boundary. There is little difference of execution time observed between slant expansion which calculates drawing start coordinates using a straight line generator and normal expansion. This proves that parallel processing and pipeline processing are conducted efficiently. Calculating from the evaluation results of this continuous graphic character drawing, it is understood that 13,640 characters in black and white or 8,040 24 x 24 dot characters in 8 colors can be drawn in one second. This value seems to be fairly low compared to the one which is calculated based on the number of drawing clock cycles, but is a practical figure.

Table 4 Measurement Values of Drawing Time

Using a logic analyzer, the time taken from the first generation of drawing parameters to the ending of drawing is measured (unit : ms, clock frequency : 8 MHz).

(a) Graphic drawing time

	Plane configuration : 1 plane	Plane configuration : 3 planes	Pixel configuration : 4 bits	Drawing contents
Straight line	37.8	113.4	37.8	With (319, 239) given as the start point, 128 240-pixel straight lines are drawn with their end coordinates starting from (0, 0), which is followed by ± 10 in X direction. And 96 320-pixel straight lines are drawn with ± 10 in their end coordinates in Y direction.
Rectangle	15.6	46.8	37.8	From (0, 0) - (639, 479) to (235, 235) - (404, 244), 48 rectangles are drawn with their diagonal coordinates changing at the rate of ± 5.
Circle	30.0	69.0	30.0	With (319, 239) as their fixed center, 46 concentric circles are drawn with their radii changing from 239 at the rate of -5.
Ellipse 1	34.8	80.6	34.8	With (319, 239) as their fixed center, 46 concentric ellipses are drawn with their X-direction radii changing from 239 at the rate of -5. Ratio of their X-direction radii to Y-direction radii is 4 : 3.
Ellipse 2	45.0	104.2	49.8	The drawing conditions are the same as above (ellipse 1) except that the ratio of their X-direction radii to Y-direction radii is 3 : 4.

(b) Paint drawing time

	Solid pattern	Tiling 1	Tiling 2	Drawing contents
Screen clear	28.8	-	-	Clearing of 640 x 480 dots x 3 planes x 2 sets (115,200 words).
Rectangle fill	7.1	13.5	13.8	Logical operation-added painting of a 400 x 300 rectangular area.
Circle fill	9.3	9.3	9.3	Logical operation-added painting of a circular area whose radius is 150 dots.
Ellipse fill 1	6.7	6.7	6.7	Logical operation-added painting of an elliptic area whose X-direction radius is 150 dots and ratio of X-direction radius to Y-direction radius is 4 : 3.
Ellipse fill 2	12.3	12.3	12.3	Logical operation-added painting of an elliptic area whose X-direction radius is 150 dots and ratio of X-direction radius to Y-direction radius is 3 : 4.
Triangle fill	3.0	3.0	3.0	Logical operation-added painting of a triangular area which is enclosed by 3 points of (152, 419), (320, 240) and (459, 320).
Trapezoid fill	5.2	5.2	5.2	Logical operation-added painting of a trapezoid area which is enclosed by 4 points of (0, 150), (300, 150), (30, 0) and (270, 0).
Paint	94.5	96.7	97.8	Painting of any of the 3 closed areas formed by 2 circles whose center is (320, 240) and radii are 100 and 220, a rectangle of (0, 0) - (639, 479) and 8 triangles.

(c) Copy drawing time

	1 plane → 1 plane	3 planes → 1 plane	1 plane → 3 planes	3 planes → 3 planes	Drawing contents
Normal copy 1	12.0	29.2	24.3	36.3	No logical operation-added transfer of a 640 x 480 dot rectangular area.
Normal copy 2	18.0	29.2	40.5	52.1	Logical operation-added transfer of a 640 x 480 dot rectangular area.
90 deg. rotation copy	20.2	29.2	43.1	61.1	Logical operation-added, 90 deg. rotation-accompanied transfer of a 480 x 640 dot rectangular area.
Enlarged copy	34.1	-	56.6	102.3	Logical operation added transfer of data enlarged by a factor 16/13 to a 640 x 480 dot rectangular area.
Shrinked copy	63.3	-	95.6	153.0	Logical operation-added transfer of 640 x 480 dot data shrinked by a factor 13/16.
Arbitrary angle rotation enlarged/shrinked copy	35.2	-	71.6	103.1	Logical operation-added transfer of 200 x 200 dot data shrinked by a factor 14/16 and rotated by $\pi/8$ with relation to X and Y axes.

	1 plane → 1 plane	1 plane → 3 planes	Drawing contents
Character font normal expansion	91.5	155.2	Normal expansion of 1248 characters of 24 x 24 dot configuration font.
Character font slant expansion	119.2	196.5	Slanted expansion of 1248 characters of 24 x 24 dot configuration font.

Conformity to CGI and Upgrading to Higher Speeds

The final stage of research on graphics interface standards, such as VDI or its simplified version CGI, has been progressing. Every time a new graphics controller is announced, how the controller is designed to cope with these standards is a matter of our concern. If we may come to the conclusion first, it will be long before we really see a controller which can directly interpret CGI-defined commands. Only when the standards are established and received well and the software is mature, attempts to put the controller circuits into an LSI will start. Some controllers already claim that they conform to CGI standards, but it is an arguable statement. A CGI interpreter can be roughly divided into a command interpreting section and a device driver section. Because the command interpreting section does not depend on controllers, a different controller can even be used as long as the CPU remains the same. And even when the CPU is different, if the software has been written in C language, no problem occurs. Any difference caused by the used controller occurs only in the device driver section. Therefore, we think that how easily this device driver can be created determines the degree of the controller's conformity to CGI.

The AGDC has, as its built-in functions, almost all of the drawing functions of the CGI (Table 5). Therefore, it is enough to just create parameters which the AGDC requires. And it is not necessary to develop the software to control the complex drawing address generation and the display memory. Only when complementing drawing functions which are not built in the AGDC, software creation work needs to be considered.

Upgrading Toward 3-Dimensional Display and Image Processing:

The AGDC did not depend on the design base built during the development of the GDC. Because the GDC, which was designed when LSI integration level was no match for that of the present, had to achieve drawing functions being then limited by hardware wired logic. Under a completely new system concept, however, all of the drawing algorithms, special hardware for faster drawing functions, a preprocessor which effectively controls them, a drawing processor, individual hardware and software had to be developed and piled up little by little from the status of almost no design base. We groped in the dark for a long time. Finally the AGDC was developed as a completely new 2-generation graphics controller. For this reason, every detail regarding an optimum controller from the viewpoints of unit designers was taken into consideration.

Presently, the maximum operating frequency is 8 MHz but this will be speeded up as the size of the LSI chip is reduced. It is the natural flow of technology that, after a certain function is established, higher functions are sought for. It is now possible to imagine a leap to 3-dimensional graphics and an expansion to image processing in the near future. In addition, it will be studied how to allot functions more effectively while making better use of the functions of dual port memory and adding more functions. Anyway, products to be developed from now will be based on the AGDC, with more functions added to it. And there will less barriers than those we experienced during the development of the AGDC.

References:

1) Oguchi, "Graphic Controller LSI for Raster Scan Type CRT Which Can Draw 1 dot in 800 ns," Nikkei Electronics, Oct. 12, 1981, no. 275, pp. 186-209.

2) Pinkham, R., Novak, M. and Guttag, K., "Video RAM Excels at Fast Graphics," Electronic Design, Aug. 18, 1983, pp. 161-171

3) Kobayashi, "Development of 256 K Bit Dual Port Memory for Frame Buffer, Which Enables Continuous Serial Output," Nikkei Electronics, Aug. 12, 1985, no. 375, pp. 211-240

4) Wientjes, B., Guttag, K. and Roskell, D., "First Graphics Processor Takes Complex Orders to Run Bit-maped Displays," Electronic Design, Jan. 23, 1986, pp. 73-81

5) Mihokawa, Ueno, Yoshida, Takeda, Maejima, Katsura, "CRT Controller That Can Designate Drawing Positions by Means of Coordinates and Has Rich Commands Like Paint and Copy," Nikkei Electronics, May 21, 1984, no. 343, pp. 221-254

6) Inaba, "Comparison of Multiwindow Display Methods of Work Stations," nikkei Electronics, July 29, 1985, no. 374, pp. 141-161

7) Oguchi, Minamino, Higuchi, "Graphics Display Controller," Transistor Technology, Jan., 1983, pp. 320-343

Abbreviations are listed in Table 5 of the next page.

Table 5 List of AGDC Commands

Commands starting with A or R indicate absolute position designation or relative position designation respectively. Command names ending with A, C, D or M mean absolute address designation, coordinate designation, drawing based on the drawing processor's coordinates and drawing with newly set coordinates respectively.

Command name	Setting parameters and operation contents
READ_DP	Transfers X and Y coordinates of the drawing processor to the preprocessor.
READ_COL	Transfers color information of the designated coordinates to the preprocessor.
DOT_D	Draws 1 dot at (X#, Y#).
A_DOT_M	Draws 1 dot at (X, Y).
R_DOT_M	Draws 1 dot at (X+DX, Y+DY).
A_LINE_M	Draws a straight line between (X, Y) and (XE, YE).
A_LINE_D	Draws a straight line between (X#, Y#) and (XE, YE).
R_LINE_M	Draws a straight line between (X, Y) and (X+DX, Y+DY).
R_LINE_D	Draws a straight line between (X#, Y#) and (X+DX, Y+DY).
A_REC	Draws a rectangle defined by (X, Y) and (XS, YS).
R_REC	Draws a rectangel defined by (X, Y) and (X+DX, Y+DY).
CRL	Draws a circle defined by the center (XC, YC) and radius DX.
ARC	Draws a circular arc defined by the center (XC, YC), radius DX, start point (XS, YS) and end point (XE, YE).
CSEC	Draws a circular sector defined by the center (XC, YC), radius DX, start point (XS, YS) and end point (XE, YE).
CSEG	Draws a circular chord defined by the center (XC, YC), radius DX, start point (XS, YS) and end point (XE, YE).
ELPS	Draws an ellipse defined by the center (XC, YC), Y-direction radius DY and ratio of squared radii, DH : DV, in X and Y directions.
EARC	Draws an elliptic arc defined by the center (XC, YC), X-direction radius DX, Y-direction radius DY, ratio of squared radii, DH : DV, in X and Y directions, start point (XS, YS) and end point (XE, YE).
ESEC	Draws an elliptic sector defined by the center (XC, YC), X-direction radius DX, Y-direction radius DY, ratio of squared radii, DH : DV, in X and Y directions, start point (XS, YS) and end point (XE, YE).

Command name	Setting parameters and operation contents
ESEG	Draws an elliptic chord defined by the center (XC, YC), X-direction radius DX, Y-direction radius DY, ratio of squared radii, DH : DV, in X and Y directions, start point (XS, YS) and end point (XE, YE).
PAINT	Paints an arbitrary closed area by means of boundary point retrieval start point (X, Y) and boundary point color designation DX. Paints an arbitrary closed area with a boundary color other than that of boundary point retrieval start point (X, Y).
A_REC_FILL_A	Paints a rectangle defined by the start absolute address EAD1, horizontal direction dot count DH and vertical direction dot count DV.
A_REC_FILL_C	Paints a rectangle defined by (X, Y) and (XS, YS).
R_REC_FILL	Paints a rectangle defined by (X, Y) and (X+DX, Y+DY).
CRL_FILL	Paints a circle defined by the center (XC, YC) and radius DX.
ELPS_FILL	Paints an ellipse defined by the center (XC, YC), Y-direction radius DY and ratio of squared radii, DH : DV, in X and Y directions.
A_TRI_FILL	Paints a triangle defined by (X, Y) (XS, YS) and (XC, YC).
A_TRA_FILL	Paints a trapezoid defined by (X, Y) (XS, Y), (YS, YE) and (XE, YE).
R_TRA_FILL	Paints a trapezoid defined by left point (X, Y) and right point (XS, Y) on the upper side, relative distance DY between the upper side and lower side, relative distance DX from X of the 3rd point and relative distance XC from XS of the 4th point.
A_COPY_AA	Area-to-area transfer defined by the transfer start word address EAD2 and dot address dAD2 of a transfer source, the transfer start word address EAD1 and dot address dAD1 of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
A_COPY_AC	Area-to-area transfer defined by the transfer start word address EAD2 and dot address dAD2 of a transfer source, the transfer start point (X, Y) of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
A_COPY_CA	Area-to-area transfer defined by the transfer start point (XS, YS) of a transfer source, transfer start word address EAD1 and dot address dAD1 of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.

Command name	Setting parameters and operation contents
A_COPY_CC	Area-to-area transfer defined by the transfer start point (XS, YS) of a transfer source, transfer start point (X, Y) of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
R_COPY_CC	Area-to-area transfer defined by the transfer start point (XS, YS) of a transfer source, transfer start point (XS+XC, YS+YC) of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
PUT_A	Transfer from the main storage to a display memory area defined by the transfer start word address EAD1 and dot address dAD1 of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
PUT_C	Transfer from the main storage to a display memory area defined by the transfer start point (X, Y) of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
GET_A	Transfer from a display memory area defined by the transfer start word address EAD1 and dot address dAD1 of a transfer source, horizontal direction dot count DH and vertical direction dot count DV, to the main storage.
GET_C	Transfer from a display memory area defined by the transfer start point (X, Y) of a transfer source, horizontal direction dot count DH and vertical direction dot count DV, to the main storage.

Abbreviations (in alphabetical order)

AGDC: Advanced Graphics Display Controller

bitblt: bit block transfer

CAS: column address strobe

CGI: computer graphics interface

CPU: central processing unit

FIFO: first-in first-out

GDC: Graphics Display Controller

ISSCC: International Solid State Circuits Conference

JIS: Japan Industrial Standard

LSI: large scale integrated circuit

OA: office automation

RAM: random access memory

RAS: row address strobe

ROM: read only memory

VDI: virtual device interface