An Advanced 3D Frame Buffer Memory Controller

Alex Makris, Martin White, Paul Lister

Centre for VLSI and Computer Graphics, School of Engineering, University of Sussex,
Falmer, Brighton BN1 9QT, UK, Email: M.White@sussex.ac.uk

Abstract

This paper details the design of an advanced 32 bit 3D frame buffer memory controller for a 3D Graphics Raster Processor called TAYRA [1]. This memory controller is designed to provide a performance of 33 MPixels/s for read and write cycles, 4 GPixels/s for block write, and 16.5 MPixels/s for read, modify, write cycles (with a pixel size of 4 bytes). This performance is without any interleaving. It has several control modes: S3 shared frame buffer protocol compatibility [2], stand alone 3D buffers, multiplexed 2D/3D buffers, and others. Further, our 3D memory controller is designed to control DRAM, VRAM and WRAM, and EDO versions of these memories. Also, we support up to 4 screen buffers, 16 MBytes of screen memory, and many combinations of memory organisations up to 1600x1280.

1. Introduction

TAYRA is a 3D Graphics Raster Processor which is designed to operate in a mixed 2D/3D rendering environment. This has posed considerable compatibility problems in the design of TAYRA's colour buffer interface. At the start of the design it was decided for various reasons that a VRAM interface would be the most appropriate solution. However, the advent of WRAM gave us the opportunity to design a colour buffer interface which could support both VRAM and WRAM. The design of this memory interface is quite complicated due to a number of factors, not least because of the, as already mentioned, 2D compatibility problems, but also because of the asynchronous nature of these memory technologies, and the added complexity of designing the serial video refresh logic. This can best be appreciated when contrasting the design of this VRAM/WRAM interface with the design of TAYRA's depth and texture interface which controls single port SGRAM and SDRAM. The single port and synchronous nature of the depth and texture buffer memory made these interfaces somewhat easier to design (although by no means trivial). Of course, one could ask the question why not use SGRAM or SDRAM or even FBRAM for the colour buffer. At the time FBRAM was not available, however we have now started on an FBRAM interface, and we decided that performance would be impacted without dual port memories which ruled out the synchronous memories for colour buffer use. Figure 1 illustrates the type of system architecture TAYRA may be expected to work in.

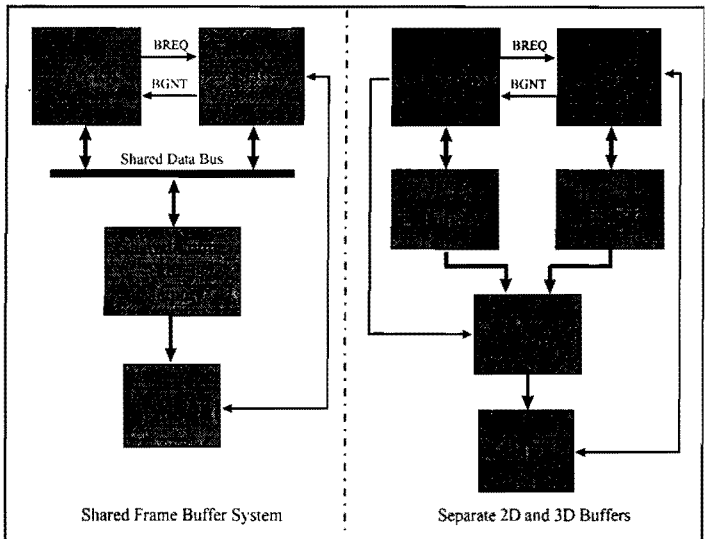


Figure 1 System Configurations

2. Memory Controller Architecture

The memory controller is implemented with six major modules. These modules are: Arbiter module (AM), Main Memory Controller module (MMCM), Timing Generator module (TGM), Microprogram module (MM), Block-Write module (BWM), Serial Interface module (SIM) (see Figure 2).

We describe in detail this architecture and its interfaces in sections 3 and 4.

3. Memory Controller Interfaces

The memory controller has five interfaces: graphics pipeline interface, PCI interface, shared frame buffer interface, memory chip interface, and a video interface.

3.1 Graphics Pipeline Interface

The pipeline signals (see Figure 2) associated with this interface are:

- PIPE_read_port&control: 24-bit address bus and a 32-bit data bus for memory read operations. It also includes all the necessary control signals for communication with the Graphics Pipeline
- PIPE_write_port&control: 24-bit address bus and a 32-bit data for memory write operations. There is also a 4-bit 'byte write enable signal' (as a mask to the 32-bit data). This signal is also supported by the PCI protocol.
- Block_write_control: Signals to control for filling a rectangular area of the screen (buffer clear).

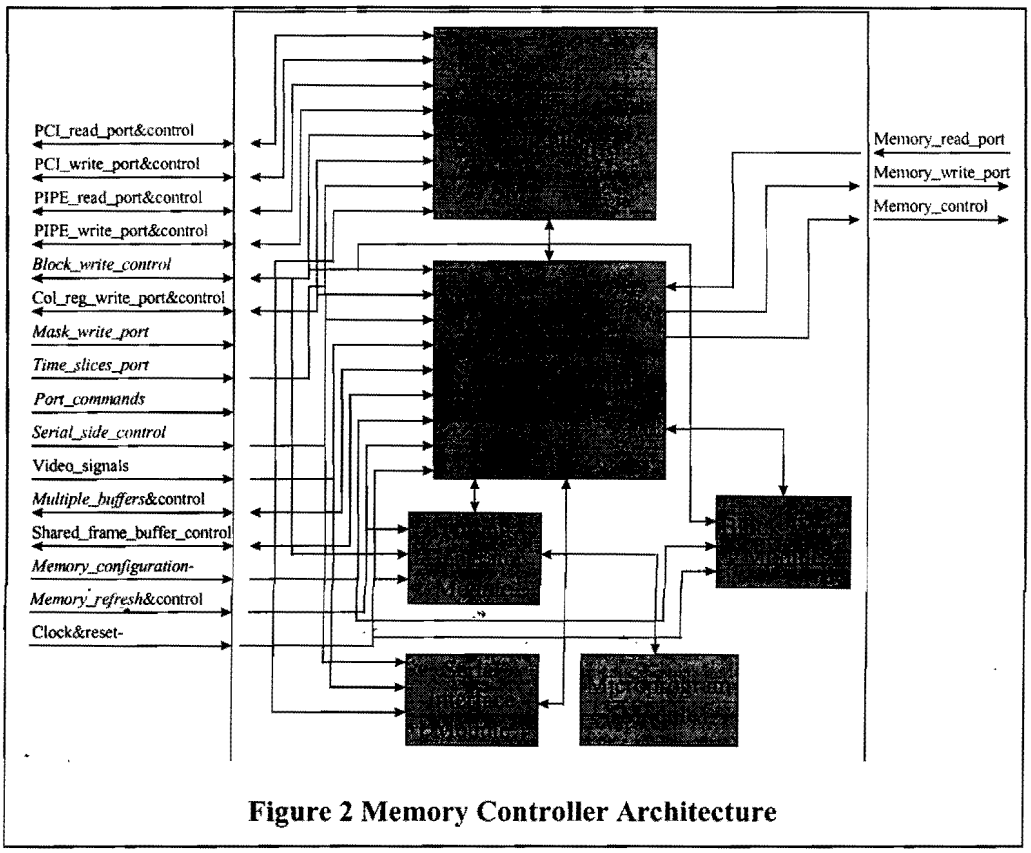


Figure 2 Memory Controller Architecture

- Col_reg_write_port&control: Signals for writing colour data to the memory chip's colour registers.
- Mask_write_port: This 32-bit signal provides the option to mask any data that is stored in memory.
- Multiple_buffers&control: Signals which indicate the organisation of the frame buffers (number of buffers, offset addresses etc.)

3.2 PCI Interface

The memory controller's PCI interface has to interface with TAYRA's chip level PCI interface. Therefore, from now on when we refer to the PCI interface we imply the memory controller PCI interface, i.e. the memory controller communication ports.

The PCI interface is very similar to the Graphics Pipeline interface. The main groups of signals associated with it (see Figure 2) are:

- PCI_read_port&control: 24-bit address bus and a 32-bit data bus for memory read operations.
- PCI_write_port&control: 24-bit address bus and a 32-bit data bus for memory write operations. There is also a 4-bit "byte write enable signal" (as a mask to the 32-bit data). This signal is coming directly from the TAYRA's external PCI interface signals.

3.3 Shared Frame Buffer Interface

The memory controller supports a shared frame buffer interface. An SVGA chip such as those supplied by S3 will be the shared frame buffer bus *master*. There is only one group of signals associated with the shared frame buffer, this is:

- Shared_frame_buffer_control: There are two signals in this group (see

Figure 3). Together they support a simple request-acknowledge protocol.

3.4 Memory Chip Interface

The memory controller supports a variety of memory buffer configurations. The memory controller is able to access different memory chips, e.g. DRAM, VRAM, WRAM, with a wide variety of different sizes and data bus widths. The groups of signals that are mainly used from the memory chip interface are:

- Memory_configuration: These signals provide information such as: memory size, addressing mode, type of memory, memory characteristics (speed, data bus width), etc. Most of these signals do not change during normal operation, and they have to be set after power-up.
- Memory_read_port: This is the memory controller's internal data bus for memory read operations. The memory controller has being designed to support both 32-bit and 64- bit data bus.
- Memory_write_port: This is the memory controller's internal data bus for memory write operations.
- Memory_control: Signals for controlling the memory chips (RAS, CAS, DSF, OE, WE, QSF).

3.5 Video Interface

The memory controller video interface has to interface with an external video display controller. When the memory controller works in 'shared frame buffer' mode the functionality of the video display controller can be provided by the master chip (S3 graphics chip):

- Video_signals: These are the common video signals (Hsync, Vsync, Blank), and they are all input signals.
- Serial_side_control: Registered signals coming from the Graphics Pipeline. These signals provide information about screen configuration, memory serial size registers, etc.

4. Memory Controller Logic Modules

The memory controller consists of six main

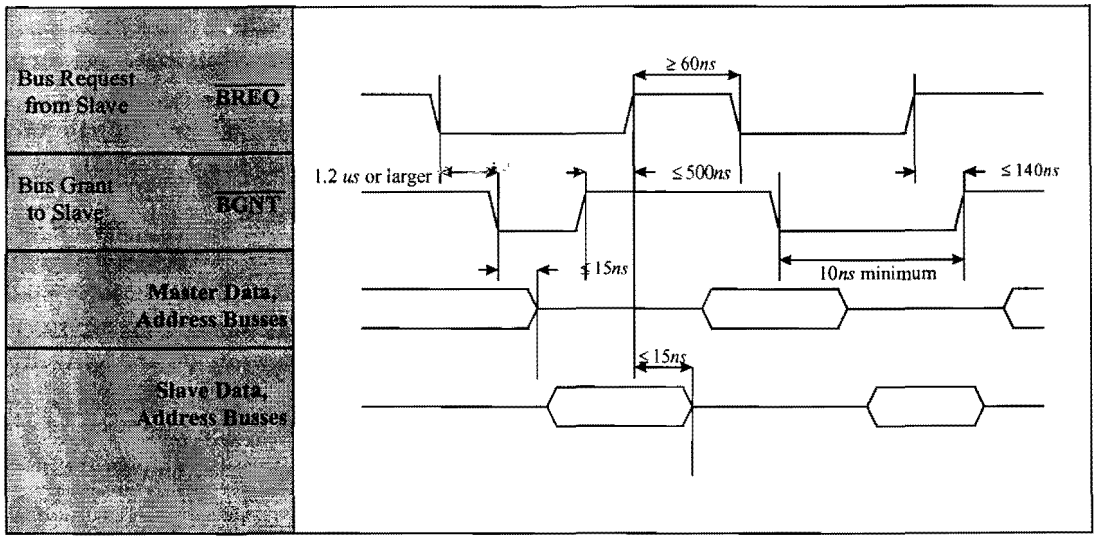


Figure 3 Shared Frame Buffer Protocol

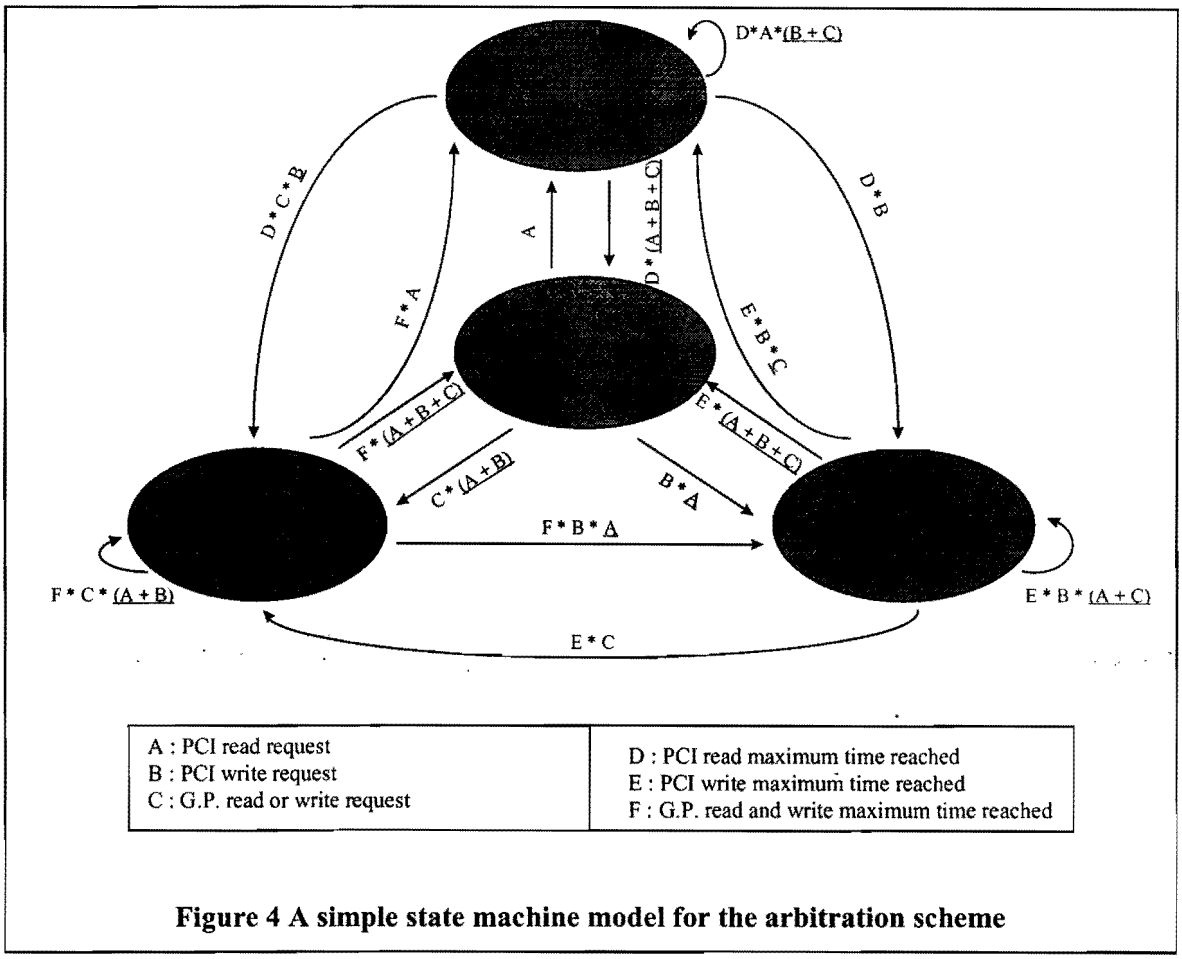


Figure 4 A simple state machine model for the arbitration scheme

modules. These logic modules provide the functionality needed to control the frame buffer.

4.1 Arbiter Module (AM)

This module arbitrates the accesses between the Graphics Pipeline and PCI interface. Each of these ports can access the memory controller for a certain amount of time, which is determined by the Port_time_slices registers. The state machine diagram in Figure 4 explains how this priority scheme works.

From Figure 4 we can see that the Graphics Pipeline read and write ports share the same amount of time. This is because these two ports can read from memory and immediately write to memory data taking advantage of the Read-Modify-Write memory cycles.

The time that each port is permitted to access the memory controller can change at run time by a software driver. This effectively can change the priority between the four ports depending on the needs of the current application.

4.2 Main Memory Controller Module (MMCM)

This module controls many tasks:

- Translates GP's 24 bit physical address (x,y) into memory row and column.
- Serves as a communication interface between four other modules (TGM, SIM, BWM, MM).
- Decides what memory mode to use, e.g. (non-page, page) and memory

cycles (such as Read data, Write data, Block write, Load colour register, refresh, serial transfer).

- Selects which memory cycle the TGM is going to use. The next memory cycle used depends on the following factors:
  - The previous memory cycle executed
  - The new physical address of the data
  - The kind of operation requested

If the I/O ports cannot provide data when the TGM requests them, there can be two possible cases:

- When the previous memory cycle can be continued, e.g. page mode, the MMCM has two options:
  - *Assert wait states.*
  - *Execute an idle cycle.*
- When the previous memory cycle can not continue (non page-mode) or already been in an idle state
  - *Execute the idle state.*

In the shared frame buffer case the memory controller (slave) can use the bus only if the **bus_gnt** signal is low. When the **bus_gnt** signal goes high the controller has to tristate all the external signals to the memory chips (control, data, address busses). The Mode Selector senses this **bus_gnt** signal before starting any memory cycle (or continuing one, e.g. page-mode), and if its high it then brings **bus_rq** high (active low) and tristates every external signal that goes to the memory chips.

The next request of the bus can be asserted after 60 ns, provided that the memory controller is ready to access the memory. In that case the MMCM brings **bus_rq** low and keeps that low until the Master request the bus again. When the frame buffer is used exclusively by the TAYRA chip the **bus_gnt** signal can be set low.

An important interface of the MMCM is the 'Port_commands' group of signals. All of these signals are connected to TAYRA's register set. These registers inform the MMCM what low level memory operation (command) is allocated to each access port. The command registers can be re-programmed at run time. Table 1 shows the low level operation for each different command (cmd) value.

The MMCM includes an Address Translation scheme. The basic function of this module is to use the 24 bit physical address coming from the graphics pipeline and produce memory row, column addresses, even/odd bank selection **ev/odd** signal, **Bmsb** signal (selects between 2 sets of even or odd banks), byte select signals.

cmd 3-bit value	memory cycle (state name)
000	page_mode_write
001	page_mode_read
010	page_mode_block_write
011	wram_reset
100	page_mode_R_M_W_starting_with_read
101	page_mode_R_M_W_starting_with_write
110	load_colour_register
111	load_mask_register

Table 1 Command to memory operation translation

Figure 5 shows one of the eight different memory address configurations, it also illustrates the address translation scheme for that configuration by using a reduced 23-bit address space.

Figure 6 shows an 8 MB memory configuration implemented with 256Kx16-bit VRAM chips. This figure also illustrates the way that the memory control signals are connected to the chips (without including the serial port of the VRAM chips) and the ability to work with a 64-bit data bus.

4.3 Timing Generator Module(TGM)

This module uses the microprogram data and generates the control signals for the memory chips. The control signal generation depends on the data fetched from the microprogram and signals from the 'Memory_configuration' group.

When the TGM is ready to start a memory cycle it requests the microprogram address from the MMCM Controller module. The MMCM then sends this address (if the graphics pipeline is not ready then the address for an idle memory cycle is sent) so that the memory cycle execution can start.

4.3.1 Control Signal Generation

Bearing in mind that the basic memory control signals (**RAS, CAS, DSF, TRG** or **OE, WE**) are provided by the microprogram data, the only function needed for controlling the memory chips is to extend them (in number 2 **RAS**, 8 **CAS**, 1 **DSF**, 2 **TRG**, or **OE**, 2 **WE**), and send them only to the chips that are going to be accessed. Here follows an example for determining the Boolean logic equations of the control signals.

All the signals coming from the MM start with an M (MRAS, MCAS etc.). The signal generation depends on the address translation scheme described in Figure 5 (all the memory configuration schemes of the VIP [3] chip are supported plus others) .

RAS(0) <= MRAS OR Bmsb, RAS(1) <=

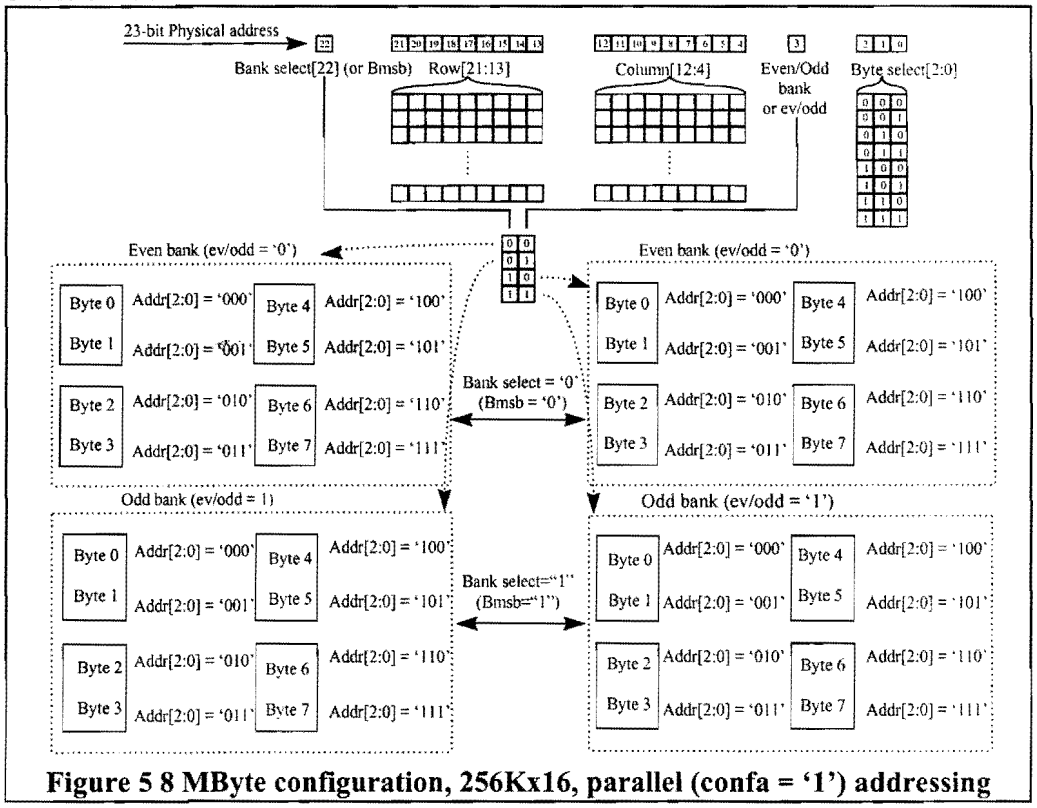


Figure 5 8 MByte configuration, 256Kx16, parallel (confa = '1') addressing

MRAS OR NOT(Bmsb);

CAS(0) <= MCAS OR ((byte_select(2) OR byte_select(1) OR byte_select(0)) AND access_mode='00') OR ((byte_select(2) OR byte_select(1)) AND access_mode='01') OR (byte_select(2) AND access_mode='10');

CAS(1) <= MCAS OR ((byte_select(2) OR byte_select(1) OR NOT(byte_select(0))) AND access_mode='00') OR ((byte_select(2) OR byte_select(1)) AND access_mode='01') OR (byte_select(2) AND access_mode='10');

and so on up to **CAS(7)**

DSF <= MDSF, WE(0) <= MWE OR ev/odd, TRG(0) <= MTRG OR ev/odd;

The **access_mode** effects the generation of the **CAS** signals in such a way that for 8 bit memory access 1 **CAS** line will be enabled (1 out of 8, chosen by the 3 **byte_select** bits).

For 16 bit memory access 2 **CAS** lines will be enabled (2 out of 8, chosen by the 2 msb of **byte_select**).

For 32 bit memory access 4 **CAS** lines will be enabled (4 out of 8, chosen by the msb of **byte_select**).

For 64 bit memory access all the **CAS** lines will be enabled and the **byte_select** data will be ignored.

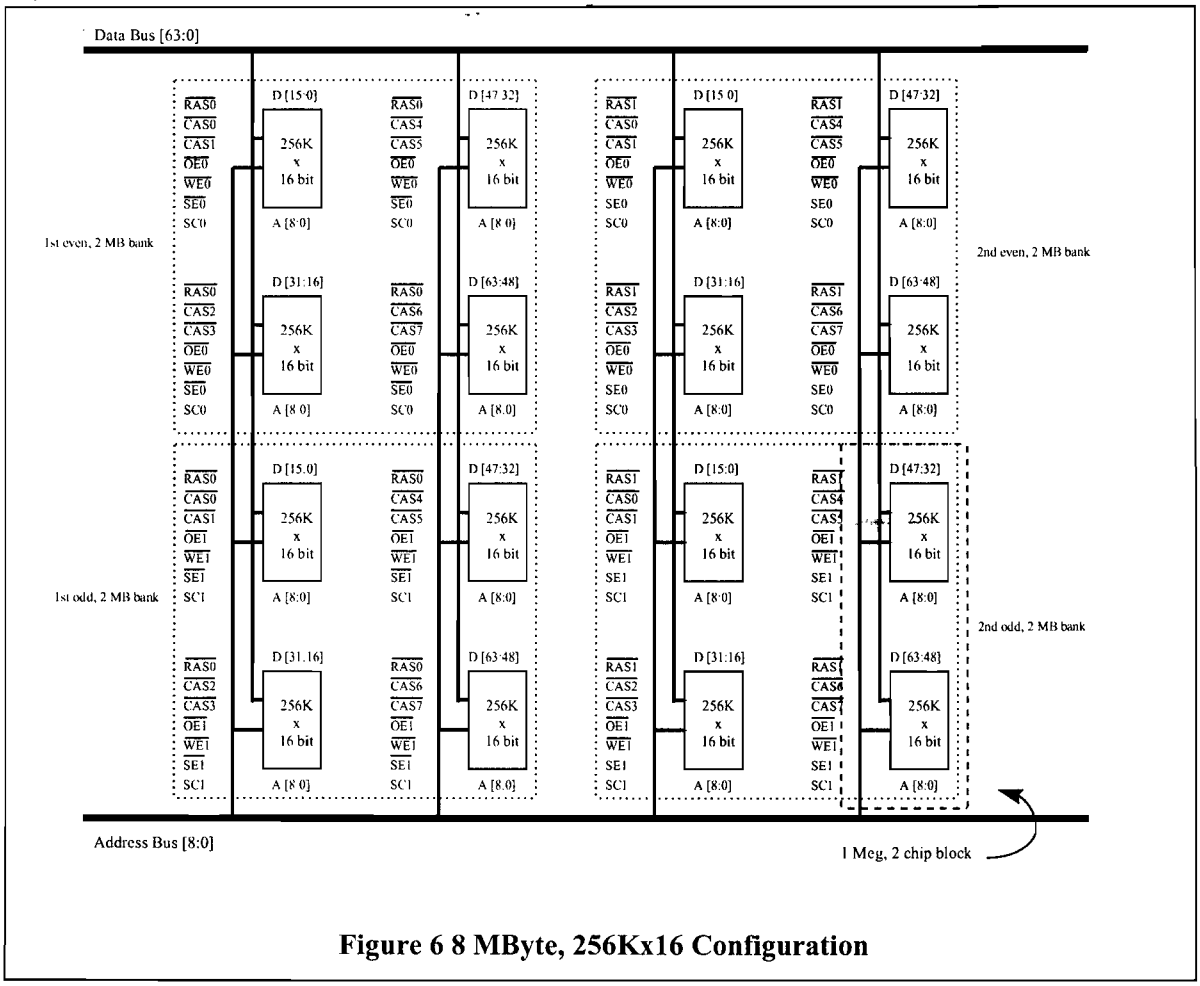


Figure 6 8 MByte, 256Kx16 Configuration

4.3.2 Refresh, 'RAS low', 'CAS low' timers

There are some mechanisms partly implemented in the TGM that check:

- How long it has been since the last memory refresh cycle.
- How long it has been since **RAS** signal brought low (and still remains low).
- How long it has been since **CAS** signal brought low (and still remains low).

The TGM has 3 timers (refresh, cas low, ras low) that start counting (in terms of clock cycles) at proper times in order to prevent any memory timing violations.

When the refresh timer reaches a predetermined value a refresh request is sent to the MMCM which has to finish the current memory cycle and execute a memory refresh cycle. After that the memory controller continues its operation normally.

If the **RAS** low timer has reached its maximum value the **RAS** signals have to go high and therefore the current memory cycle has to finish. If the request becomes from the **CAS** low timer it then depends on the current memory cycle whether the MMCM will finish it or not. When one of these two timers goes the corresponding **RAS** or **CAS** signal will also go high as soon as possible.

4.4 Microprogram Module (MM)

This provides the control signals of a memory cycle to the memory chips, when to read, write (and what to write) from the address translate module and some extra information, e.g. how many times a memory cycle can be executed before a memory refresh occurs. The MM is about 128 bytes.

The module of the memory controller that utilises the microprogram's data most is the TGM. When the TGM makes a decision about which memory cycle to use it sends the start address of the memory cycle to the address bus of the MM and the execution of the memory cycle begins. The MM then sends the 16-bit data to the TGM. The significance of this data is shown in

Figure **8**

The 5 most significant bits of the microprogram address (32 addresses) represent the memory cycle that is going to be executed (read cycle, write cycle, enhanced page mode read cycle, and generally every possible memory cycle up to 32 types). The 4 least significant bits of the microprogram address can access up to 16 memory locations where the data for the selected memory cycle is stored. Some memory cycles (such as write, R-M-W, page-mode write, page-mode R-M-W, block write, page-mode block write) have different variations. For example, the page-mode R-M-W cycle [4] (page 5-292) of TMS55166 has 4 variations:

All 4 variations have to do with the write part of the cycle but all necessary data that indicate which variation to use are sent to the memory at the beginning of the page-mode R-M-W cycle. Fortunately, all possible variations happen to be at the beginning of a memory cycle which means that they can all be stored inside a memory cycle space (16 locations).

The 5 most significant bits of the MM address represent the requested memory cycle and the 4 least significant ones the offset of the requested variation. The page-mode R-M-W memory cycle has 4 variations, where each of them needs 2 address locations. The data of the first variation are stored in address locations represented by offset addresses '0000', '0001', the second variation's data are stored in offset addresses '0010', '0011', etc.

The example in Figure 7 illustrates how the microprogram's data can be interpreted and how the microprogram's flow mechanisms work.

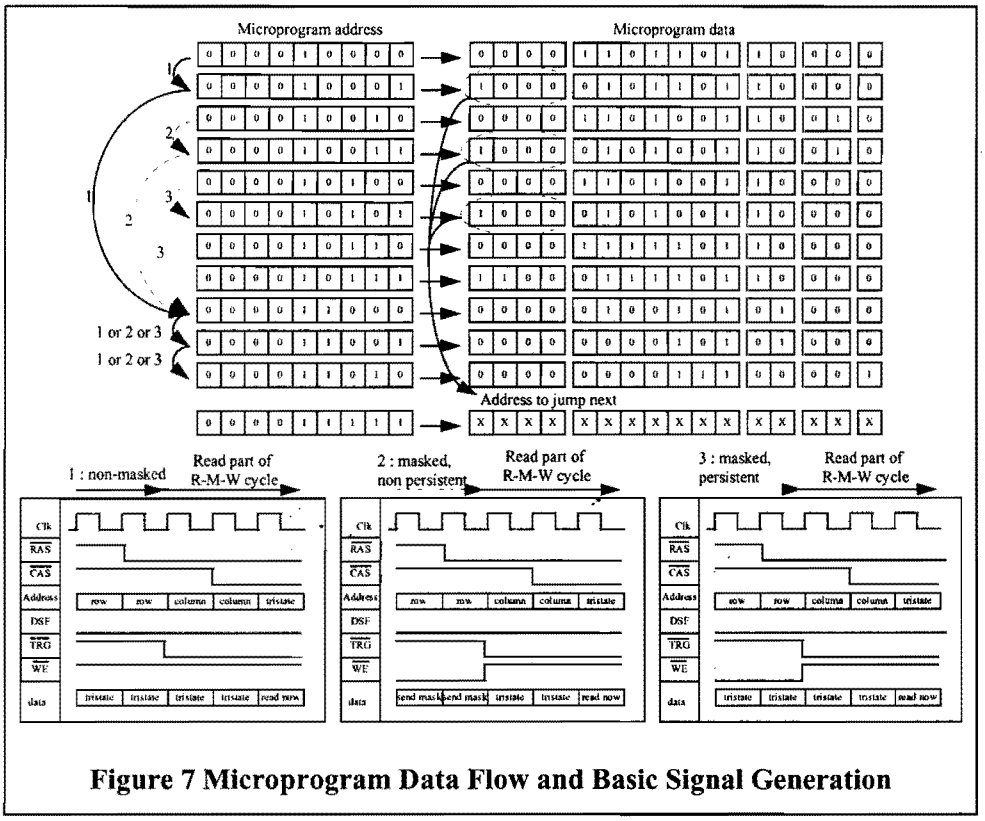


Figure 7 Microprogram Data Flow and Basic Signal Generation

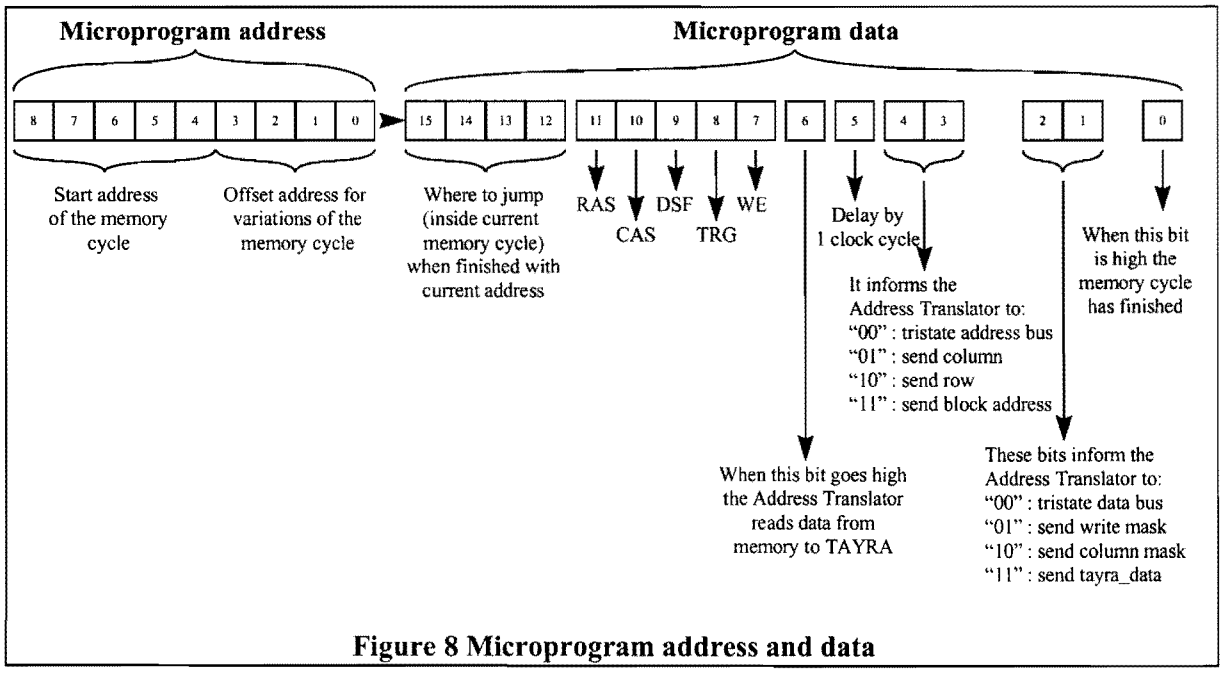


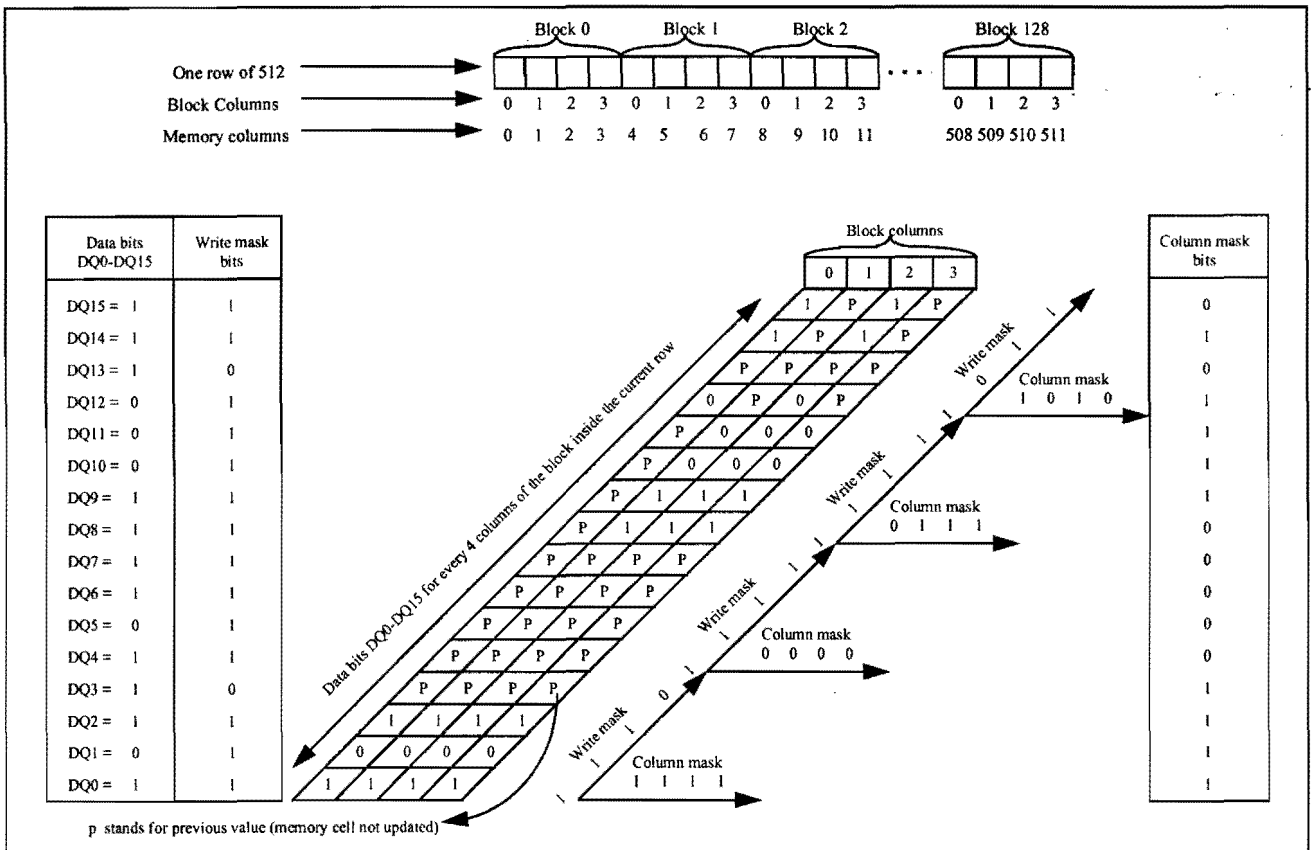
Figure 8 Microprogram address and data

All the memory timings stored in the Microprogram Module can be found in [4] for the TMS55166 Multiport VRAM chip. Although the memory cycles from this data book are given in an asynchronous way they had to be synchronised with the clock (by using states) before they could be stored as a microprogram.

4.5 Block Write Module (BWM)

This is the most complex module of the memory controller. This is the part that initiates the Block Write memory cycles for filling a rectangular area of the screen with a colour value (buffer clear operations). Most of the signals necessary for this module are coming from the Block_write_control group and most of them belong to the register set of the TAYRA chip. When using the Block Write memory cycle for storing data to the memory chips the bandwidth of the memory bus can be improved by a great factor. Typical values are 4 for VRAM, 8 for WRAM [5]. This is because the memory chip can write a colour data value to 4 or 8 locations at the same time. Figure 9 shows an example of the functionality of the Block Write memory operation for 512 rows x 512 columns and a 16-bit word memory chip (such as TMS55165, TMS5166 memory chips from Texas Instruments).

Since the colour data value has been loaded into the memory's colour register the Block Write operations can start (masked or non-masked). For this operation one of the 512 row addresses is selected and then one of 128 possible block addresses (in this row) can be accessed. Each block consists of 4 words (4 columns of the row)



Block writes, internal organisation

Figure 9 Block Write Memory Operation

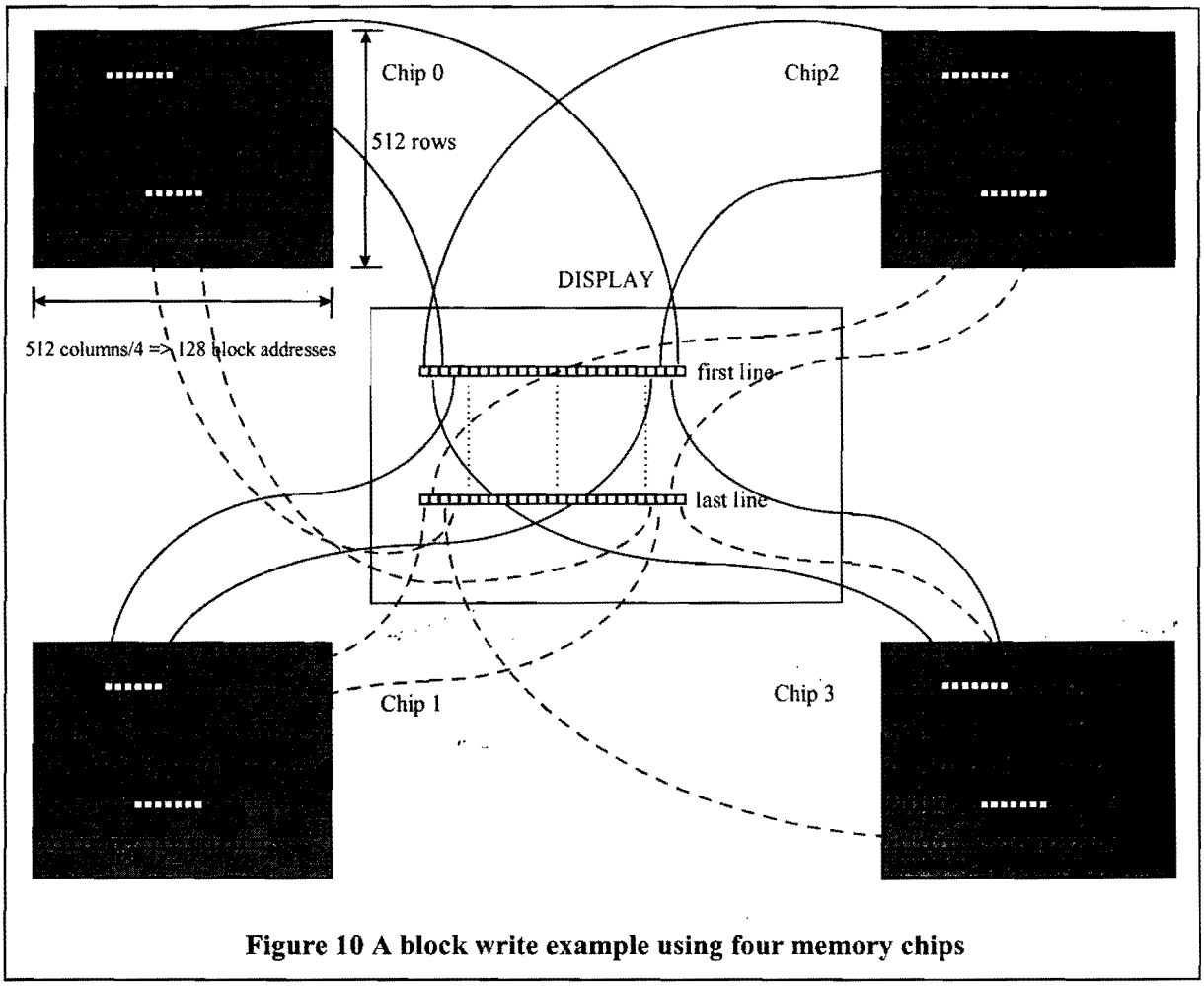


Figure 10 A block write example using four memory chips

and therefore 4 words are stored simultaneously in the memory. The write mask data simply says to the memory which bits of the input data are to be stored. An example of how the column mask works in combination with the write mask is illustrated in the previous figure.

The BWM module has to provide the column mask, the row address and the block address for each chip that has to be accessed (the write mask is not important for the operation of this module). The basic factors to have in mind for the design of a BWM module are:

- A horizontal line of the display usually is not stored sequentially to a memory chip but each pixel's data can be stored from one up to four memory chips (depending on the chip's data bus size).
- Support of different memory configurations and address translation schemes
- Support of chips with different data sizes, different column mask functionality

A simplified example of how the memory controller fills a small block is shown in Figure 10.

In the example of Figure 10 the display memory is shared between four memory chips. Each chip has 512 rows, 512 columns and 16-bit words. The possible block addresses for each row are the number of columns divided by four (128 block

addresses). In order to simplify the example we assume that the size of each pixel is 16-bits.

4.6 Serial Interface Module (SIM)

This part of the memory controller provides the necessary interface for communication with the video signals coming from the video display controller (Hsync, Vsync, Blank). The SIM also uses the 'Serial_side_control' signals. These are hardwired to the TAYRA's register set and provide necessary information to both SIM an MMCM. A part of the SIM identifies the time periods where the scan line has reach the rightmost point of the active display, or the last pixel of the frame. This information is used by the MMCM for programming the serial register of the memory chips, execute refresh operation during horizontal or vertical blank periods, swap to a different memory buffer (double, triple, quadruple buffering).

5. Acknowledgements

TAYRA is part of the ESPRIT Monograph project funded by the European Commission. We are grateful to our Monograph partners (IBM Germany, Universität Tübingen and Caption), for their contributions to this project.

6. References

[1] TAYRA 1.0 Functional Specification, February 1996

[2] Vision964 Graphics Accelerator, January 1995, S3 Incorporated.

[3] VIDEO INTEGRATION PROCESSOR™ — Hardware Specification, Program Reference Versions 1.2, February 17, 1995

[4] MOS MEMORY—Data Book, TI, 1995

[5] GRAPHIC MEMORY — Data Book, SAMSUNG, April 1995