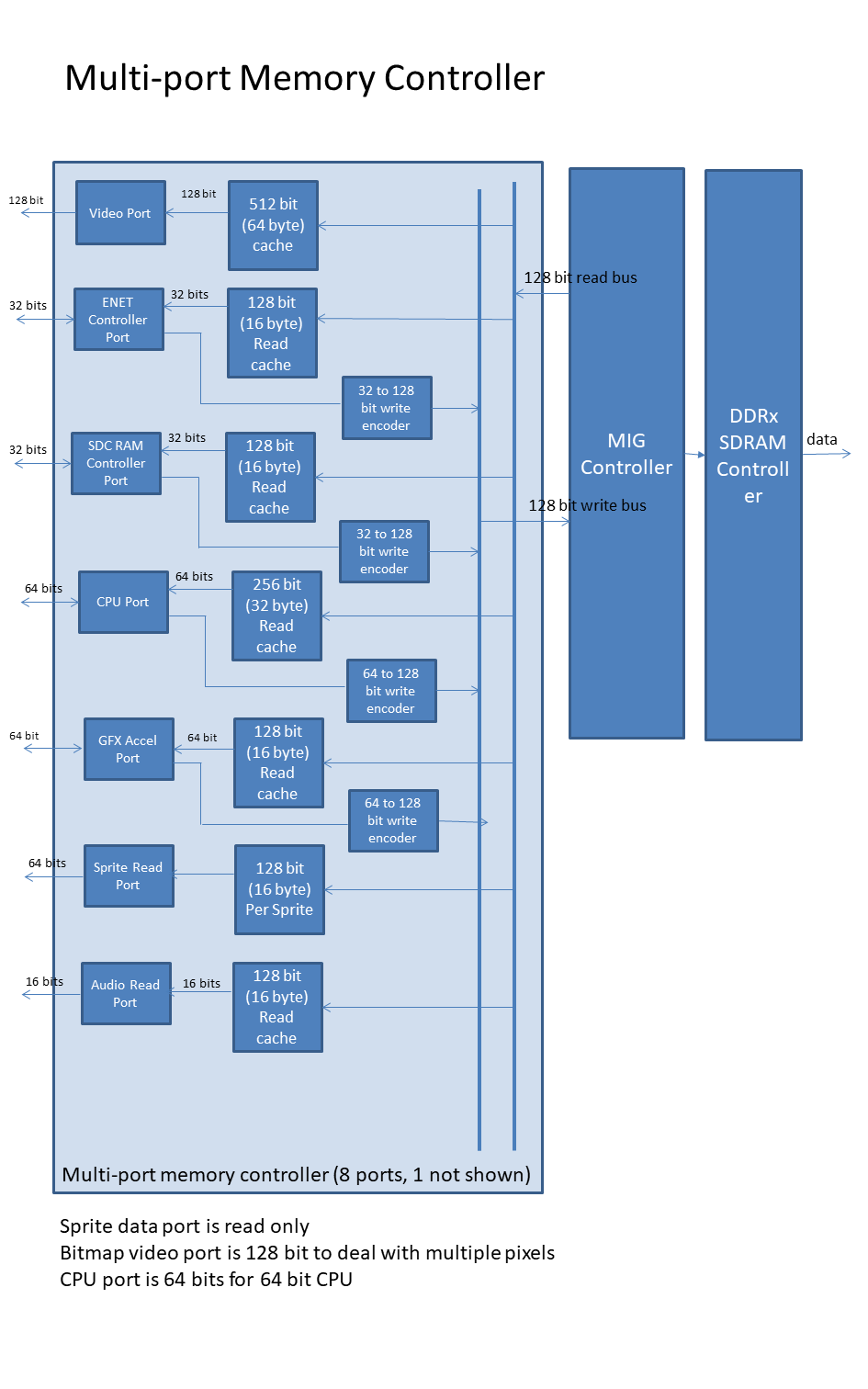
**MPMC7 – Multi-Port Memory Controller #7**

robfinch<remove>@finitron.ca

# Overview

The multi-port memory controller provides eight access ports with small read caches to the ddr3 ram. The multi-port memory controller interfaces between the SoC and a MIG controller. The DDR3 ram controller is outside of the scope of the MPMC.

The ports have pre-designated usages. Ports are specialized to their use. Data widths have been assigned to maximize memory bandwidth. Most devices are filling some sort of cache or buffer from memory or dumping data back to memory. A large number of memory ports are required because the system has only a single physical memory port and everything is in the system on chip.



# Port Assigned Usage

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Port | Use | Port Bits | Cache Bits | Access |
| 0 | Frame Buffer / Bitmap Controller | 128 | 512 | 4 |
| 1 | CPU #1 | 64 | 256 | 4 |
| 2 | Ethernet controller | 32 | 128 | 4 |
| 3 | Audio controller | 16 | 128 | 8 |
| 4 | Graphics controller | 64 | 128 | 2 |
| 5 | Sprite controller (read only) | 64 | 128 | 2 |
| 6 | SD Card (disk) controller | 32 | 128 | 4 |
| 7 | CPU #2 | 64 | 256 | 4 |

# Port Priorities

The ports have a fixed priority arrangement according to the port number. The lowest port number has the highest priority. Periodically, for one access cycle only, port priorities are inverted so that port 7 has the highest priority and port #0 the lowest.

# Overall Organization

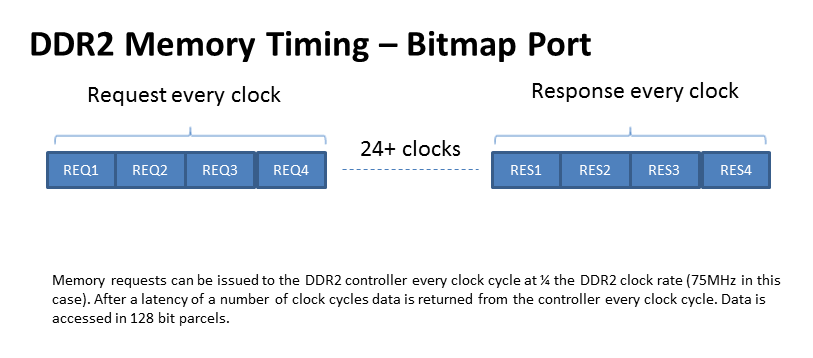
The controller has separate read caches for each of the ports. This allows multiple read accesses by different devices to occur in parallel, provided the data to be read is in the read cache. Some of the latency for memory reads can be hidden in this manner.

# Port Descriptions

## Bitmap Controller Port

Bitmapped graphics can require a high memory bandwidth. In many systems the display memory is separate from the rest of the main memory of the system so that the bandwidth requirements of the display don’t slow the rest of the system down. As an example of bandwidth requirements, the test system uses a 680 x 384 x 16bpp color bitmapped graphics mode (a submultiple of 1366x768 mode). The clock frequency for this mode is about 85MHz. About 85MB/s of data is required. (43MHz \* 16-bit data). Each byte has to be read in about 12ns.

Given that in the test system the main memory of the system is being used for display, it is desirable to make the best use of available bandwidth. In order to achieve better bandwidth memory access for the bitmap controller is organized into a number of strips. One key thing to note is that the bitmap controller basically always reads forwards through memory. It’s addressing is predictable and not random. This makes it easier to achieve high bandwidth access. While the ddr3 controller already does burst access to fetch 128 bit at a time from the memory, performance can be maintained by performing larger burst accesses. Hence the use of memory strips. For the bitmap controller a memory strip is 4 consecutive burst accesses or 512 bits (64 bytes) at a time. The vendor’s controller does not need to wait for a read access to complete before starting a second (or more). However, there is about a 24-cycle latency for memory access. The following diagram shows memory access for the bitmap display.



To perform accesses in this manner takes about 32 clock cycles. Note the display’s effective dot rate is about 43MHz, while the memory controller clock is 75MHz. If performing four consecutive accesses to memory is good wouldn’t it be better to perform even more? It would be except the problem here is that there are other devices in the system that need to access memory sometimes within a limited time-frame. Giving too many consecutive cycles to the bitmap controller would starve other devices. There is also an issue with the fact that part of the last access for a scan line is wasted. More data is fetched than needed because 64 bytes doesn’t divide evenly into a 680-pixel scan line. (680 / 32 = 21.25 strips).

The bitmap controller has a programmable access interval so that it does not continuously access the memory. This gives some time for other devices to access the memory. The interval has to be set carefully or there could be display problems with the bitmap display.

## CPU#1 Port / CPU#2 Port

Memory access for a cpu is somewhat similar to the bitmap controller. Many accesses travel forwards through memory. However, there is a more random aspect to the cpu’s accesses. To improve performance the CPU already has a cache. So, most of the read access required by the cpu is in order to fill a cache line. The CPU only fills one cache line at a time during a miss. Most of the time that’s all that’s required. So, the memory port for the CPU only buffers enough for a single cache line. Timing is similar to the bitmapped display, except that only two consecutive burst accesses are performed.

## Ethernet Controller

The Ethernet controller requires read / write access to memory for network transfers. Ethernet memory transfers are often done by a dedicated DMA controller and not the CPU. Hence, they require a memory port. The current system uses an Ethernet controller that supports only 32-bit accesses. So, the ethernet’s port is only 32 bits wide. However, a read operation from memory always transfers 128 bits, so only those bits that the controller needs are passed back to it. The other bits are cached.

## Graphics Controller

The graphics controller port services memory requests for that device. A graphics accelerator reads and manipulates pixels from memory then writes pixels back. Most of the time the accelerator is dealing with individual pixels. There’s little benefit to fetching or storing large numbers of pixels in this case. Hence the port for the graphics controller uses only single burst accesses.

## Sprite Controller

The sprite controller uses dedicated DMA triggered during the horizontal blanking interval. Sprite images have linear accesses to memory just like a bitmap display. To minimize the number of memory accesses required to display sprite data, each sprite has its own read cache. Thus, the sprite port is special in that the sprite number requesting the read is also input to the controller. The sprite’s read cache is enough to buffer about four scan-lines worth of display data. If the sprites were to share a common cache then the cache would be constantly dumped and reloaded since each sprite’s display data address is different.

## SD Card Controller

The SD card controller has only a 32-bit data path and hence a 32-bit port is used.

# Write Ports

Everything mentioned so far has had to do with read ports. All of the ports with the exception of the sprite port are also capable of writing to memory. While the read ports are somewhat customized to the port owner, write ports all work in the same manner.

There is no write cache. Writes go directly to memory as soon as possible. With the exception of the bitmap controller, they have a higher priority than reads.

All write ports write perform a single 128 bit burst write to memory. Caches are not updated by writes. Instead if there is a cache hit during a write cycle that cache is invalidated. This is a simple means of keeping all the caches in the memory controller coherent.

# Memory Reservations

The MPMC keeps track of up to two reserved memory addresses for the cpu ports. Memory reservations are used to implement atomic memory operations. A conditional store operation by the cpu will be done only if the store address is reserved. The cpu must output signals to indicate that an address is being reserved during a load operation, and to indicate that a conditional store is taking place.

# Conclusion

It’s possible that more ports would be required in the system. For instance, the system may also be interfaced to an EPP parallel port for diagnostic purposes. More CPU cores might be present in the system.

# Parameters

|  |  |  |
| --- | --- | --- |
|  | Default | Description |
| NAR | 2 | Number of outstanding address reservations possible |
| AMSB | 28 | Most significant address bit |