

Reconfigurable Memory Module in the RAMP System for Stream Processing

Vason P. Srin

Data Flux Systems Inc., 1678 Shattuck Ave.
MS 292, Berkeley, CA 94709

John Thendean and Jan M. Rabey

University of California, EECS Dept., Berkeley,
CA 94720

Summary

Realtime audio/video communication and image processing using wired and wireless networks involve handling data coming in streams at a high rate from sensors, cameras, and networks. Temporary storage and fast access to data streams are important to keep the overall system performance at a high level. A memory module with builtin logic and controller to support random access (RAM), FIFO access (FIFO), delay line implementation (DELAY), and lookup table implementation (LUT) is described in this paper. The memory module can be reconfigured to be in one of four (RAM, FIFO, DELAY, LUT) modes and is part of a system containing reconfigurable clusters of memory and processors, called RAMP.

RAMP is a low-power high performance reconfigurable parallel processing system with the needed partitioning, placement, and routing (PPR) software so that the above applications can be mapped to processors on one or more chips. Each cluster in RAMP has a memory module, four computation processors, and an I/O processor. The architecture and design of the reconfigurable memory module is the focus of this paper. All the units within a cluster communicate data with a delay of one cycle using a simple send/receive handshake protocol. By limiting the number of clusters to 16 in a chip, intercluster communication delay can also be kept at one cycle without slowing the processor. The implementation of the protocol using a single wire, buffers, and queue controllers in sending and receiving units is discussed.

Since the memory module can be configured to be in one of the four modes, control logic is needed to

switch between these modes and also to perform the operations in each of the modes. Four finite state machines (FSMs) have been designed to support the four operating modes of the memory modules. The details of the FSMs and other control parts of the memory module are described. The storage array for the memory module is based on a self timed low-power SRAM design. The programming and reconfiguration of the memory module is supported using a scan chain register and control signals. The datapath, controllers, and storage array of the memory module have been integrated and simulated at the VHDL level and transistor level to verify functionality and timing.

This research was partially funded by AFRL, Eglin AFB, FL. Contract No. F08630-95-C-101

1.0 Introduction

High performance and low-power microprocessors are needed for many of the media applications involving images, sound, and video. Processors such as Philips' TriMedia, TI's TMS320C6000, Analogic's Sharc, and HP/ST's Lx have been introduced to meet the performance requirements. The datapaths in these processors are optimized to provide extremely high performance on certain kinds of arithmetic-intensive algorithms. However, a powerful datapath is only a part of the solution for high performance. To keep datapath fed with data and to store the results of datapath operations, the processors require the ability to move large amounts of data to and from memory quickly. Thus, the organization of memory, latency, and its interconnection with the processor's datapath are critical factors in determining processor performance. It is known that any off-

chip communication is expensive in terms of power, delay, and area. To drive a signal off-chip, large current is required to drive the capacitor on the I/O pads and any external capacitors. Therefore, having to move data to and from chip to off-chip memory will not be an efficient implementation. Many high performance DSP and media processors have on-chip memory. Commercial general purpose digital signal processors, for example, Motorola's DSP56000 family and TI's TMS320C6000 family have separate on-chip memory for program and data. With on-chip memory, the memory bandwidth can be increased and off-chip traffic can be reduced. FIR implementation is the simplest example to clearly illustrate the memory requirements of DSP processors. A single FIR tap computation requires four accesses to memory - instruction fetch, read data from delay line, read the appropriate filter coefficient, and write data to the next location in the delay line to shift data through the delay line.

In this paper we discuss the architecture and design of a memory module that is part of a reconfigurable clusters of memory and processor (RAMP) system capable of delivering tens of giga operations per second and uses low-power techniques. The RAMP system combines on a single chip many stream data processors, memory modules with builtin logic and controllers, I/O processors, and a conventional superscalar processor. The parallel stream data processors compute on the multiple data streams using a MIMD model of computation and the superscalar processor performs the coordination of parallel tasks, sequential computations, and communication with the memory system. The execution time for any instruction is one cycle.

RAMP has its foundations in PADDI-2 [1] chip containing 48 nanoprocessors. A nanoprocessor is used to interface the PADDI-2 chip and the external memory. The nanoprocessor handles the single cycle interprocessor communication protocol. In the RAMP system, memory is on-chip and partitioned so that a part of it is available in each cluster to support locality of data in applications. Based on analyzing DSP algorithms [1-3], simulation studies,

and experiments [4], it was determined that performance could be improved if memory can be organized to fit the four commonly used memory accesses in signal processing - RAM, Look-up Table, FIFO, and delay line. We decided to enhance the storage array in a memory block with logic and controllers to support the four commonly used accesses and the single cycle interprocessor communication protocol.

The rest of the paper is organized in seven sections. Section 2 contains an overview of the cluster architecture in the RAMP. The architecture of the memory module is described in Section 3. The blocks of the datapath are described in Section 4. The storage array of the memory module is also described in this section. The hardware support for data communication between memory module and other processors, and buffering are also described in Section 4. The FSM controllers used in the control part of the memory module are described in Section 5. A brief description of the blocks used for communicating data and control tokens is included in Section 6. The programming of the memory module as well as other processors is done using a scan chain register in each cluster and reconfiguration is achieved using the scan chain register after bringing the processors in a cluster to a freeze state. The 55 bits forming the scan chain register of a memory module is explained in Section 7. The bits of the scan chain register are used by the blocks described in Sections 4 to 6. Forward references to Section 7 appears throughout the paper. The detailed timing diagram resulting from functional and timing simulation is described in Section 8.

2.0 Cluster Architecture

A cluster containing processors, memory, and I/O interface is the building block of a RAMP system. A block diagram of a cluster in RAMP is shown in Figure 1. It contains four processors that can do video, graphics, image processing, and digital communication computations on streams of data, called vgiprocessors [5, 6]. A memory module containing a storage array (256x16 words) and controllers for

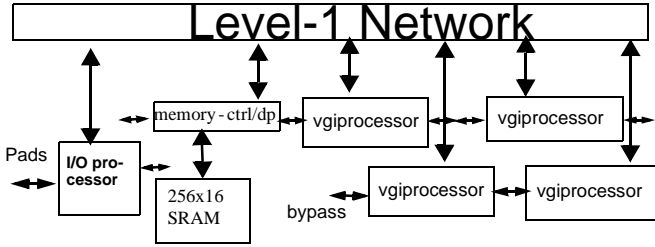


FIGURE 1. Cluster architecture with memory

implementing the four commonly used memory accesses is the second major part of the cluster. The third part of the cluster is an I/O processor that sends and receives data from external devices using a two cycle communication delay protocol. The I/O processor communicates with vgiprocessors and memory modules using a single cycle communication delay protocol. All six processors in a cluster have a three stage pipeline in their datapaths and there is a system clock. Although the prototype memory module has a 256 X 16 SRAM, it is not a limitation. The storage array can be increased in size and the controllers can be resynthesized by changing the parameters of delay lines, queue length, and counters.

Communication between the processors in a cluster is done through six data token buses (16 bits) and four control token buses (2 bits) in the Level-1 communication network and the associated handshake lines. The single cycle delay communication protocol for memory module and I/O processor is the same as that of the vgiprocessor. From Figure 1, we can see that memory module's control and datapath is the interface between other processors and the storage array. Memory module handles all the timing requirement to read and write into the storage array. It also handles data addressing in FIFO and delay line mode.

Using Level-1 network, the processors in a cluster can communicate with other clusters on a RAMP chip using an intercluster network, called Level-2 interconnection network, containing 16 data token buses and eight control token buses. This is done by setting the switches (50 switches in all) between Level-2 network and Level-1 network. Adjacent processors in a cluster can communicate data using

the bypass buses and thus leaving the resources in Level-1 network for intercluster and non adjacent processor communication.

One of the novel features of the RAMP architecture is the single cycle communication protocol between processors in a chip using a send/receive handshake protocol. This protocol is implemented using a bidirectional handshake line for each data token bus and control token bus and the two phases of the system clock. During the execution stage, a processor wanting to communicate data to another processor pulls the handshake line high when clock is low. If the receiver is not ready it can pull down the handshake line when the clock goes high on the next cycle (communication stage). At the end of clock high the success of the transaction is evaluated. If the handshake line is high then the transaction is successful. Otherwise it is unsuccessful. A sender can retry when the clock is low. The details of the protocol and the implementation details are scattered in Sections 4 and 5.

3.0 Memory Module Architecture

The memory module has a datapath and a control section. A block diagram of the memory module is shown in Figure 2. The datapath has a three stage pipeline similar to that of the vgiprocessor and I/O processor comprising fetch, memory access, and communicate. The fetch and communicate stages are needed to synchronize the memory module with a vgiprocessor. This avoids the need for special protocol when a vgiprocessor communicates with the memory. The additional two stages impose a two clock latency when a read is requested and one clock latency for a write request. Since requests are usually pipelined, the latency occurs only on the first request.

In the following sections the architecture of the memory module is explained. The datapath structure is described first and then the controller block.

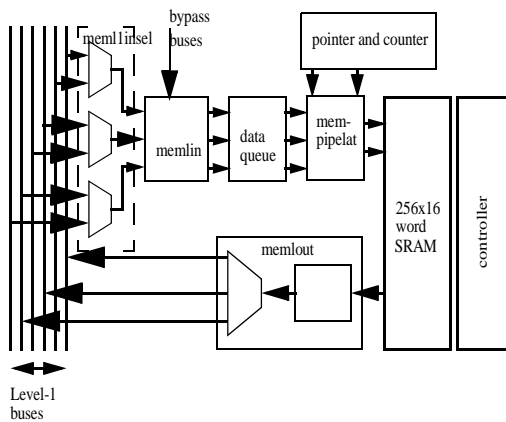


FIGURE 2. Block Diagram of Memory Module

4.0 Datapath Structure

The memory module's datapath consists of the following blocks: meml1insel, memlin, data queue, mempipelat, storage array, memlout, address pointer and counter, as shown in Figure 2. We have retained some of the cryptic names for the blocks to maintain consistency with the names in design schematics and netlists. The following subsections describe in detail the functions of each block.

4.1 Meml1insel Block

Meml1insel is the interface between buses in Level-1 network and memory module. It consists of three multiplexers that select 3 out of 6 Level-1 buses. The three outputs of this block are read address, write address, and write data. The bus assignment is done by setting the associated Level-1 configuration switches in the scan chain register (bits memscni<5:0> in the scan register of memory module). Figure 3 shows the Level-1 configuration switches.

The buses in Level-1 network are assigned in the following manner:

- bus (d0 or d1) or bus (d2 or d3) can become read address source.
- bus (d0 or d1) or bus (d2 or d3) can become write address source.
- only bus d4 or d5 can become write data source.

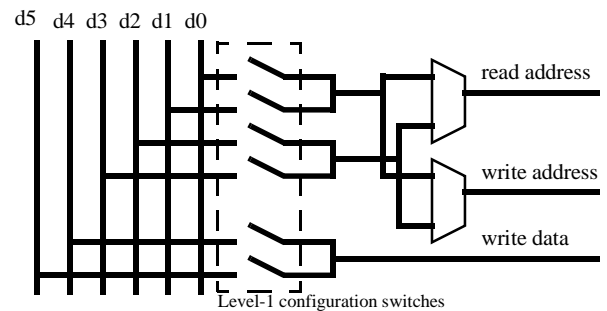


FIGURE 3. Meml1insel

4.2 Memlin Block

This block consists of two 8-bit registers for read and write address and one 16-bit register for write data. During the communicate cycle, the sender drives the output data on Level-1 bus. On the same cycle, the receiver copies the data into the master of the memlin register. During fetch cycle, the data in memlin will be transferred into registers in data queue block. Memlin is the communication data buffer at the receiver.

In memlin block, data and address source can be selected from the bypass buses or the Level-1 buses. To select the address or data source from bypass buses, the associated memsbpi bits in the scan chain register (memsbpi<2:0>) should be asserted.

4.3 Data Queue

Data queue consists of three two-deep queues, one for read address, one for write address and the other for write data. The queues are important because of the two cycle delay for a request to be processed. For example, when a read is requested, whether the read data can be sent out or not is not known until two cycles after the request is received. During those two cycles, additional requests are being accepted and processed. If the memory module fails to send the read data the processor will stall; data queues are used to buffer the requests that have been received before the stall comes.

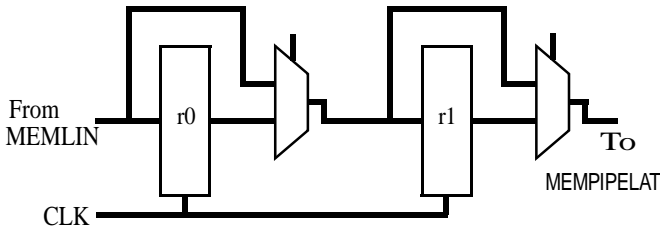


FIGURE 4. Schematic of one queue register. Data queue block consists of three such queue register; they are read address queue, write

The schematic of the queue is shown in Figure 4, where r0 and r1 are queue registers. When there is no pipeline stall, data will always be placed to the head of the queue, r1. The 2-to-1 mux between r0 and r1 is used to bypass r0, to allow data from memlin to appear directly to r1. The second 2-to-1 mux is used to allow data from memlin to go directly into the pipeline register (mempipelat). Therefore, when no stall pipe occurs, data from memlin can be fetched into r1 and pipeline register at the same time.

4.4 Pipeline register (mempipelat)

The pipeline register plays two major roles. During program execution, mempipelat is used to supply stable address and data to the storage array block, shown in Figure 2. It is the pipeline register of the memory module. Like the data queue, mempipelat consists of two 8-bit register for read and write address and one 16-bit register for write data. The read address register and write data register are also part of the scan chain register and they are used to load data into the storage array during scan in process and to read out the memory content during scan out process. Data register is also used to initialize the read and write pointer for FIFO and delay line operation.

The two major rolls of the mempipelat block are explained in the next two subsections using the control signals of a cluster and two scan chain register bits.

4.4.1 Mempipelat as scan chain register.

During the scan process data can be loaded into the storage array (e.g. sine or cosine table) or read out the memory content and initialize the address pointers. To store data in the storage array, data and address are scanned into data and address registers. Read/Write enable (rwen) bit and memory update (memupdate) bit of the scan chain register should be asserted as well. When scan update (supdate) control signal comes, data in the scan register will be loaded into the location specified by the address register. Table 1 summarizes the operation of mempipelat as scan chain register

To read out the content of a memory location, first address has to be scanned in, read/write enable bit (rwen) and memory update (memupdate) bit should be disabled. When supdate signal comes, data in the address location will be loaded into data register which can be scanned out later.

Data register is also used to initialize read and write pointer for FIFO and delay line operation. Read and write pointers are 8 bits wide each, while data register is 16 bits wide. Cascading read and write pointers, we get 16 bits wide data. Therefore, by connecting the upper byte of data register to write pointer register and the lower byte to read pointer register, we can initialize the pointer registers. Memory update (memupdate) bit of the scan chain is used to distinguish this operation from loading data into storage array. When pointer initialization is intended, memupdate bit is not asserted. Read/Write enable (rwen) bit, however, has to be asserted.

TABLE 1. Summary of mempipelat as scan chain register.

rwen	memupdate	Operation
0	0	Read the content of a location in storage array.
0	1	Read the content of counter, read and write pointer

TABLE 1. Summary of mempipelat as scan chain register.

rwen	memupdate	Operation
1	0	Initialize counter, read and write pointer
1	1	Store data into a location in storage array.

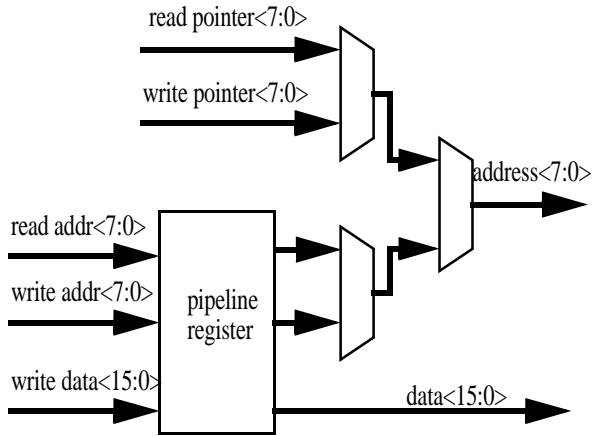


FIGURE 5. Mempipelat schematic as pipeline register. Address source can be read/write address

4.4.2 Mempipelat as pipeline register

The schematic of mempipelat register when operated as a pipeline register is shown in Figure 5. As pipeline register, mempipelat is used to supply stable address and data to the storage array unit. Whether data in mempipelat can advance or not is controlled by advance signal which comes from queue controller block. Advance signal is asserted when neither stall pipe nor save signal is asserted. Save signal comes from access controller and it is asserted when the access controller cannot serve a request. The section on access controller explains more about the function of this signal.

Address output of mempipelat can come from two sources. In the RAM and Look-up table operation, this address comes from Level-1 buses. In FIFO and Delay line operation the address comes from read or write pointer.

4.5 Storage Array Block

The 256x16 words SRAM in Figure 2 is the storage array of memory module. This storage array is the self-timed SRAM designed by Burstein [7]. It has separate buses for input data and output data and a single address bus. The storage consists of memory cells and a dummy circuit which mimic the worst case delay in the memory block. This dummy circuit generates a done signal which indicates that a read/write has been completed. This signal is intended for asynchronous design. In this design, only the memory cells is used. The dummy circuit is ignored. Although the prototype has a 256 X 16 SRAM, a generator is available [8] for extending the size of the storage array.

4.6 Memlout Block

The memlout consists of a data register to hold data read from storage array and switches that connect to Level-1 buses. Data in the register can be driven to three of the six Level-1 buses so that it can be sent to other processors. Level-1 output switches are configured using scan chain register bits (mem-scno<2:0>). When a receiver is not ready to receive data in the memlout register, then memlout block will preserve its data to be resent the next cycle.

Output of memlout is also extended to bypass buses. Therefore, if receiving processor is next to the memory module, the data transfer can be done through the bypass buses which free up the Level-1 buses for other processors.

4.7 Pointer and Counter

This block, shown in Figure 6, consist of two up-counters for read pointer and write pointer, one up/down-counter to count the number of data stored in

Advance signal is generated using the following logic:

$$advance = \overline{(stallpipe + save)}$$

where save is a signal from access controller that tells the queue controller that the data at the head of the queue has not been serviced because another process is being executed. As an example consider a situation where read and write requests arrive simultaneously at the access controller. If the priority is given to write over read, then write will be performed and save will be generated for read address queue controller.

5.3 Memory Access Controller

The memory access controller is the key block of the memory module. The four modes of the memory module are implemented in this block. The operation mode of memory access controller is configured using configuration (config<1:0>) bits of the scan chain register. Memory access controller communicates with the queue controller using the save signal. Whenever a request (read/write) cannot be serviced by memory access controller, it will assert the save signal. This will result in advance signal being deasserted and the data in data queue and pipeline register blocks will be preserved. For example, if the memory access controller receives simultaneously read and write requests, depending on the priority, either read or write will be selected and the other request has to wait. If write is performed then save signal for read queue controller will be asserted and vice versa for read.

The memory access controller communicates with the output controller through the out signal. When a read is performed, out signal is asserted, signaling the output controller to prepare to send out the data read from storage array. The four modes of the memory access controller are described using FSMs in the next four subsections. The control signals shown in Figure 7 are used in deriving inputs that cause state transitions.

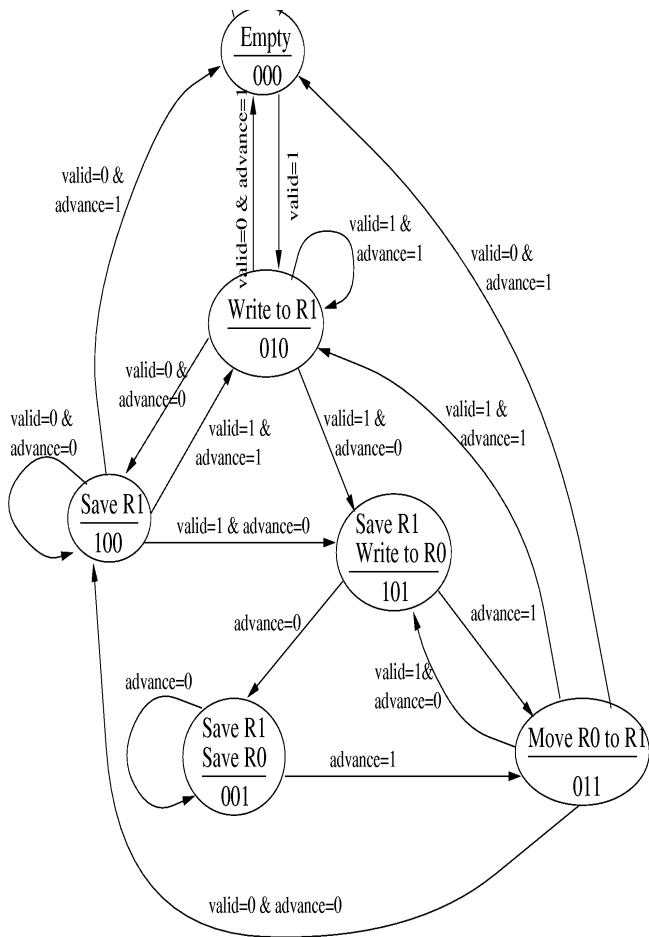


FIGURE 8. Queue controller state transition diagram Memory module controller consists of three controllers of this type - read address queue, write address queue, and write data queue.

5.2 Queue Controller

The queue controller decides where the data in the memlin register will be stored in the data queue block. If there is no stall pipe, data will be placed at the head of the queue. There are 3 queue controllers, one for read address, one for write address and the other for write data. Queue controllers also decide whether an incoming data can be accepted. When a queue is full, the queue controller will pull down the handshake line to indicate to the sender that no data can be accepted. The states of the queue controller are shown in Figure 8. Please refer to Figure 4 in reading the state diagram.

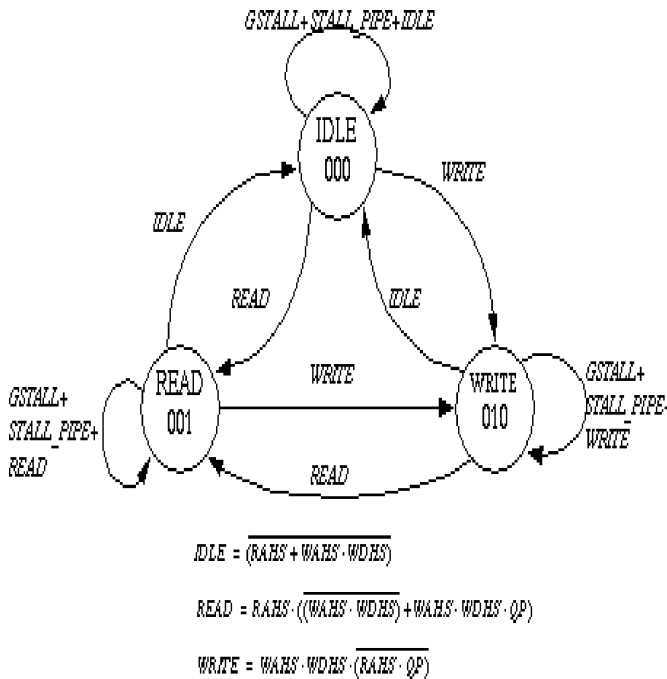


FIGURE 9. RAM mode state transition diagram.

5.3.1 RAM Mode

When configured in RAM mode, memory access controller treats the storage array as a random access memory. It can accept any read and write operation. Simultaneous read and write request is resolved using queue priority bit of scan chain register. This means that the user determines the read or write priority. When queue priority bit is asserted, priority is given to read request and when the bit is low, priority is given to write request. For a write request to be executed, both write address and write data have to be available to the access controller. Figure 9 shows the state diagram of access controller when configured in RAM mode.

5.3.2 Look-up Table Mode

The Look-up Table (LUT) mode is similar to RAM mode except that write is not allowed during execution. In the LUT mode, storage array is loaded with a table during the scan in process. The procedure for scanning is described in mempipelat section



FIGURE 10. Look-up Table state transition

(Section 4.4). Figure 10 shows the state transition diagram of Look-up Table operation.

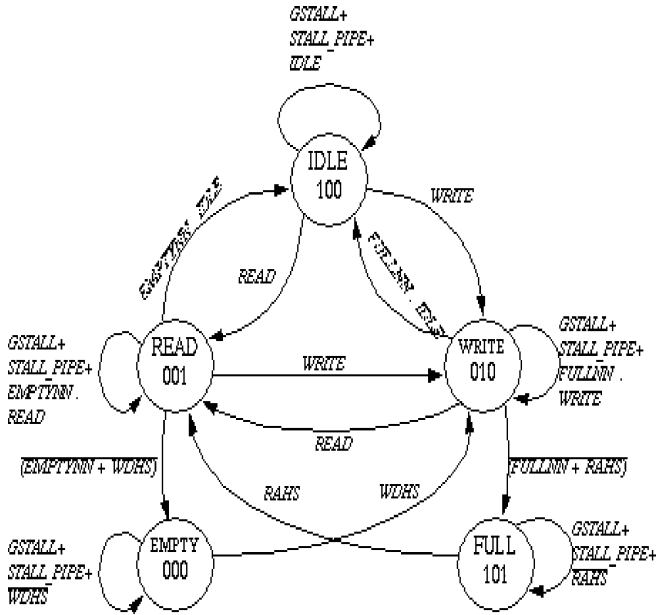
5.3.3 FIFO Mode

In FIFO mode, addresses for read and write come from read and write pointer blocks. Only the write data comes from Level-1 buses. Read request is communicated using read handshake line and write request is communicated using write data handshake. Since write address comes from write pointer it is assumed to be always ready. So, in FIFO mode, write address handshake signal is ignored.

Simultaneous read and write request can occur in FIFO mode. This is resolved using the queue priority bit like in the RAM mode. Fullnn and emptynn signal from memflag block (Figure 6) signal the access controller when the FIFO is full or empty. Full occurs when the counter value reaches the number of delay in the scan chain register. Empty occurs when the counter value reaches one and the access controller is in read mode. Figure 11 shows the state transition diagram of access controller in FIFO mode. One thing to notice in the state diagram above is that unlike other operation modes, idle state in FIFO controller is encoded as 100 and empty is encoded as 000. Therefore, when the memory module is reset, FIFO controller will assume that the FIFO is initially empty.

5.3.4 Delay line mode

Like the FIFO controller, the sources for the addresses of delay line controller are read and write pointer blocks (Figure 6). What makes delay line



$$IDLE = \overline{(RAHS + WDHS)}$$

$$READ = RAHS \cdot (\overline{FULLNN} + \overline{WDHS} + OP)$$

$$WRITE = (\overline{EMPTYNN} \cdot RAHS \cdot OP)$$

FIGURE 11. FIFO state transition diagram.

different from FIFO is that the delay line controller does not accept read requests. Read will be executed automatically once the delay line is full. This is indicated by fullnn signal from memflag block (Figure 6). Write address is assumed ready every time write data is available. So, write request is communicated using write data handshake signal only. Since simultaneous read and write is not possible when the controller is operating in delay line mode, the queue priority bit is simply ignored. Figure 12 shows the state diagram for delay line controller.

5.4 Output Controller

The output controller, shown in Figure 7, receives the out signal from memory access controller every time access controller is in read state. When the request comes, the controller will assert the output handshake line to signal the receiver of outgoing data during clock low. When clock high comes, the output controller observes the handshake line to

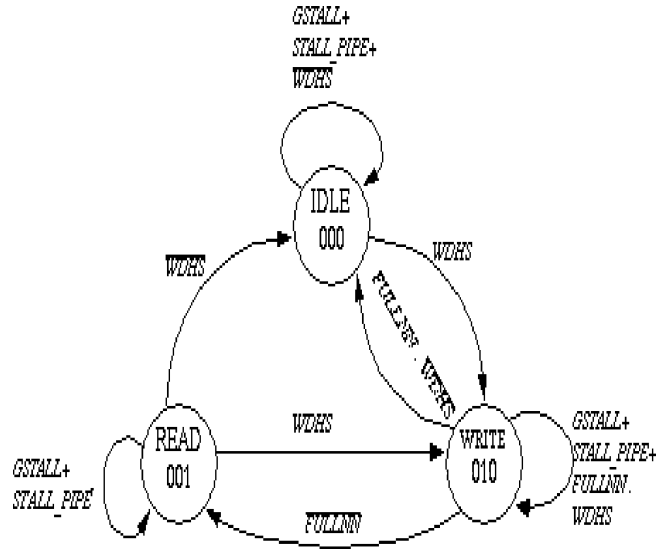


FIGURE 12. Delay line state transition diagram.

determine if the send is successful. If the handshake line is pulled down any time during clock high, the send is unsuccessful and the controller will try to resend the data the next cycle. In this case stall_pipe signal is asserted for the memory module which will stall other activities.

6.0 Communication / Handshake Blocks

These blocks are the Level-1 network's handshake lines that interface with the memory module controller. At the input side we have input handshake block (memhsin) and at the output side we have output handshake block (memhsout) as shown in Figure 7.

6.1 Input Handshake Block

Like the memllinsel block in Figure 2, input handshake block consists of Level-1 configuration switches which select three out of six handshake lines from Level-1 handshake lines. These switches are configured using the same scan chain bits used to configured Level-1 switches in memllinsel (memsnci<5:0>). The three output handshake lines are read address handshake (rahs), write address

handshake (wahs) and write data handshake (wdhs). The order of handshake assignments is the same as the address and data assignment in meml1insel block (see Figure 3). Input handshake block receives full signal from queue controller. When full is asserted, it will pull the associated handshake low during clock high.

6.2 Output Handshake Block

The output handshake block consists of Level-1 output configuration switches like those in memlout block (see Figure 2). When there is data to be sent out, output controller will assert send signal (see Figure 7). The output handshake block will drive this signal across the communication network to other processors. Output handshake block can fan out this signal to at most three other processors.

7.0 Scan Chain Register

The memory module's scan chain register is 55 bits long. It is divided into three blocks, memscni (26 bits), mempipelat (26 bits), and memscno (3 bits). The physical location of the bits of the scan chain register is scattered among the blocks of the memory module. Memscni contains 6 bits for selecting Level-1 network's six buses, 3 bits for activating bypass buses to supply data tokens, 2 bits for selecting read and write address source, 2 bits for operation mode selection (00 = RAM mode, 01 = look-up table mode, 10 = FIFO mode, 11 = Delay line mode), 1 bit for queue priority assignment, and 8 bits for FIFO or delay line size. Scan chain register in mempipelat consists of 8 bit address register, 16 bit data register, read/write enable (rwen) bit, and memupdate bit. Address and data register are used to load storage array with data or to read out content of memory. The second function of the data register is to initialize read and write pointer register in the datapath's counter unit. Read/Write (rwen) bit indicates that the data in data register is to be loaded either into memory or pointer register. Together with read/write bit, memupdate indicates that the data in data register is to be loaded into

storage array. Memscno block contains 3 bits to control Level-1 output select switches.

The operation of the scan chain register is controlled by two cluster level control signals sen and supdate that are generated from primary input signals gstall, tms, reset, and clk.

8.0 Design Verification

All the blocks of the memory module have been designed and functionally simulated at the RTL level using a VHDL simulator. Many blocks in the control part of the memory module have been synthesized. The remaining blocks have been custom designed. The physical layout of the entire memory module has been completed using custom layout tools and it is included at the end of the paper. The storage array occupies the two quadrant on the right side of the layout. The datapath occupies the bottom left quadrant and the control occupies the top left quadrant. The physical layout has been extracted, verified, and a transistor level netlist generated for use with Timemill simulation. The operation of the memory module has been verified using Timemill simulations.

The following example illustrates the data propagation in a memory module. We assume the memory module is configured in RAM mode (memconfig = 00). There are three requests that arrive consecutively to the memory module. They are named 1, 2 and 3 in Figure 13. Request 1 is a read operation, request 2 is a write operation and request 3 is a read operation. The three requests can come from a vgi processor, another memory module, or an I/O processor. For read request, output data is sent out to another processor and the receiving processor is assumed ready to receive the data. Figure 13 shows the timing diagram of data through the memory module's datapath.

During the communication cycle, data 1, are driven onto Level-1 buses to the input of memory module. These data are copied by memlin and driven to data queue in the fetch cycle of memory module. Data 1

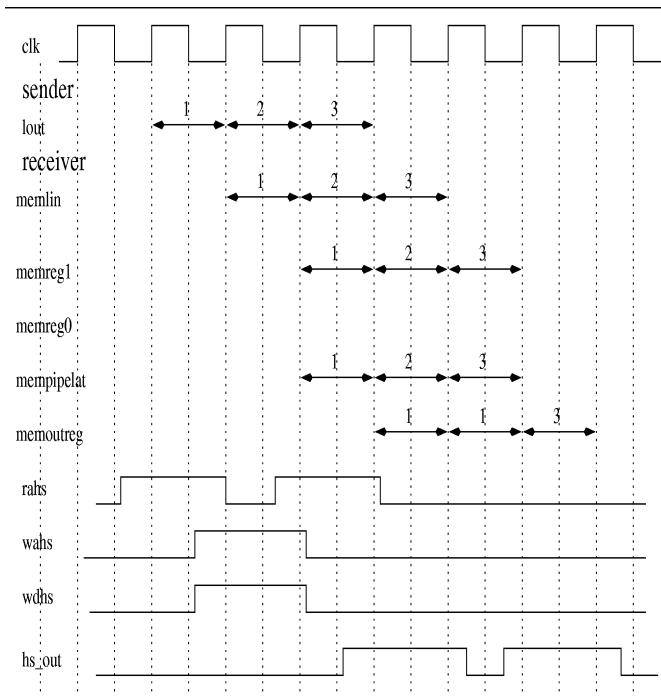


FIGURE 13. RAM Data propagation through memory module datapath.

is a read request as indicated by the rabs line. Since no data are yet stored in the queue, data 1 will advance to memreg1 and mempipelat. At the same time, memlin is accepting the next data 2. Data 2 is write request as indicated by both wabs and wdhs line. For write request, both wabs and wdhs have to be high to be processed. The next cycle, mempipelat drives data 1 request to storage array and memory read is executed. Data 2 are advanced to memreg1 and the third read request, data 3 are communicated. At the beginning of the next cycle, data 1 result appear at the output register. These data are being driven on Level-1 bus to a receiving processor. Notice that output handshake line (hs_out) has been pulled high during clock low in the previous cycle to tell the receiver processor of the arrival data 1 result. At the same time, write request of data 2 is executed and data 2 operation is completed. Data 3 are transferred to memreg1 and mempipelat during this clock cycle. Since data 2 do not produce any output, hs_out is pulled low by the output con-

troller at the next clock cycle to tell the receiver that no data is being communicated. On the same cycle data 3 is executed and the output is stored in the output register. The output handshake line again is asserted when clock goes low to inform the receiver of the availability of data. During the next cycle, data 3 results are communicated to the receiver.

In this example, we can clearly see the latency of read and write requests. A read request, indicated by data 1 in the example, incurs two clock cycle latency. When data 1 appear at the output of memlin during the first clock cycle, data 1 is transferred to mempipelat. At the second clock cycle, the memory read is performed. At the end of this clock cycle, the data 1 result is available at the output register which will be sent to the receiver when the next clock cycle comes. A write request, indicated by data 2, only incurs one clock cycle latency. When data 2 is available at the output of memlin during the first cycle, data 2 is transferred into mempipelat. At the second clock cycle, the memory write is performed and the request is completed.

Acknowledgement

The authors wish to thank Brian Richards, Matt Armstrong, and Roy Sutton for their comments and Bob Brodersen for his support with the infrastructure at Berkeley and BWRC. We also acknowledge the encouragement from Robb Brown and Pat Coffield.

9.0 References

1. A. K. W. Yeung, "PADDI-2 Architecture and Implementation," Ph.D. Thesis, University of California, Berkeley, CA, June, 1995.
2. P. Lapsley, J. Bier, A. Shoham, E.A. Lee, "DSP Processor Fundamentals: Architectures and Features," Berkeley Design Technology, Inc, Berkeley, CA, 1994, Chapter 5.
3. E. C. Ifeachor, B. W. Jervis, "Digital Signal Processing: A Practical Approach," AddisonWesley, Wokingham, England, 1993.

4. V. P. Srin, R. A. Sutton, J. M. Rabaey, "Multiple Processor DSP System using PADDI-2", Proceedings of Design Automation conference, San Francisco, CA June, 1998.

5. D. M. Pini, "A Parallel Processor for High Performance Digital Signal Processing," Master's Thesis, University of California, Berkeley, CA, May, 1997.

6. V. P. Srin, J. Thendean, S. Ueng, J. M. Rabaey, "A Parallel DSP with Memory and I/O Processors", Proceedings of the SPIE Conference, San Diego, CA, July, 1998

7. A. J. Burstein, "Speech Recognition for Portable Multimedia Terminals", ERL Memorandum No. UCB/ERL M97/14, Feb. 1997, University of California, Berkeley, CA.

8. N. Walker, "Integrated Circuit Module Generator", Master's Thesis, University of California, Berkeley, CA, May, 1999.

