AN INPUT-OUTPUT SYSTEM FOR THE CAM-8 MACHINE

DAVID I. S. HARNANAN

June 18, 1999

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cellular automata are a special class of discrete dynamic systems. A cellular automaton consists of a universe of cells laid out in a regular pattern. Each cell in the grid is characterized by a certain state which changes in time. The evolution of a cell's state is determined by its current state and the states of its neighbors. A cellular automata rule specifies exactly how the new state is derived from the old one.

Much of the research on cellular automata has focussed on the development of the rules that are iteratively applied to the cellular automata universe of cells. It turns out that when the system is run with carefully chosen rules, its evolution (as determined by the changing states of its cells) can accurately model a large class of physical phenomena. For example, diffusion, crystallization, gas models, reflection and refraction of waves, as well as numerous other processes have been investigated through cellular automata models.

Conventional general-purpose computers are not suited to the requirements of cellular automata simulations. To be effective, many cellular automata experiments call for the execution of billions or trillions of cell-updating operations. At best, sequential computers can perform an updating operation in a few microseconds. Hence they would take several days to complete a single cellular automata run!

The limitations of conventional computers and the nature of the cellular automata process call for the development of specialized computer hardware to perform cellular automata simulations. Two characteristics of cellular automata are particularly useful in directing hardware development — locality and parallelism. Cell updates in cellular automata depend only on the states of a cell and its neighbors. Furthermore, at each temporal iteration, all cells in the cellular automata universe are updated in parallel. This suggests that cellular automata machines (CAMs) should consist of arrays of parallel processors operating in a SIMD-like[1] fashion. This has indeed been the general architecture of the CAMs that have been developed to date.

The Information Mechanics group at the MIT Laboratory for Computer Science has pioneered the development of cellular automata machines. After much experimentation, this group successfully developed and marketed a general-purpose cellular automata machine (CAM-6).

CAM-6 allowed for the simulation of a "toroidal space"[2] consisting of 256x256 cells. Each cell was assigned 4 bits of state. The processors used to apply the temporal rule consisted of suitably programmed SRAM lookup tables. A cell update in CAM-6 involved using a subset of the 36 state bits associated with the cell and its 8 neighbors as a pointer to the SRAM lookup table. The value contained at that table location was taken as the new state of the cell. Cellular automata rules could be programmed into CAM-6 by downloading suitable data into the lookup tables.

The cell states of CAM-6 were mapped as pixels on a CRT screen. This allowed graphic visual output from the system — the operation of the cellular automata rule could be viewed on a monitor as the iterations progressed. CAM-6 was implemented on a circuit board with ordinary memory chips and standard TTL components. It was plugged into a host IBM-PC from which it could be programmed and run. The book *Cellular Automata Machines: A New Environment for Modeling* [**?**] describes CAM-6 in more detail and provides many examples of experiments which have been performed with its hardware.

Encouraged by the success of CAM-6, the Information Mechanics group at MIT has begun work on the development of a much more powerful cellular automata machine, CAM-8. CAM-8 will allow for the updating of billions of 16-bit cells which are arranged in a multidimen-

[1]SIMD is short for Single Instruction Multiple Data computer.

[2]Toroidal space means that the cells on the left and right edges of the space are considered to be neighbors, as are the cells on the top and bottom edges.

-bb-error = = □

Figure 1.1: The Input-Output System of the CAM-8 Machine.

sional array. Thousands of lookup tables will be used, each of which will service a million cells. The proposed architecture will provide neighborhoods spread over a large radius: a cell will see a neighborhood consisting of up to 16 bits chosen from a few million neighboring cells.

Unlike CAM-6, the new CAM-8 is based on the principle of partitioning cellular automata — a procedure whereby cells are divided into blocks and subjected to block updates. By changing the block partitions between iterations, the effect of conventional cell-based cellular automata can be achieved. This leads to a much neater and more flexible hardware realization than that of CAM-6. Indeed, the CAM-8 architecture promises to significantly extend the size, speed and versatility of cellular automata simulations.

The CAM-8 project is the source of this Master's thesis. The thesis involves the design of an input-output (I/O) system for the CAM-8 machine. The input-output system connects the CAM-8 hardware to a host computer and serves as a communication medium over which the host can control the operation of CAM-8. In addition, the statistical data gathered from CAM-8 runs may be transmitted to the host over the I/O interface. The following section describes the input-output system in greater detail.

1.1 The Input-Output System

Figure 1.1 shows the main elements which compose the CAM-8 input-output system. A Sun Sparcstation host sits at one end and is responsible for controlling the operation of CAM-8 by communicating with the CAM-8 hardware over the I/O system. The host initializes the state of the cellular automata universe modelled by CAM-8 and generates and downloads lookup tables to the processing modules before a simulation. The lookup table contents will depend on the particular rule that

CAM-8 is to apply to the cellular automata universe during the experiment. Thereafter, as the simulation progresses, the host may change control parameters, lookup tables and/or gather statistical information over the I/O system.

The modules which make up CAM-8 are located at the other end of the I/O system. Each space module consists of 16 CAM-8 controller chips; 16 DRAMS which store the state of an area of the cellular automata universe; and two SRAMs which serve as the lookup table processors through which the cellular automata rule is applied.

The host controls the machine by writing to special registers on the controller chips. Statistical data gathered during a simulation and stored in other chip registers may also be accessed by the host. The thesis includes the design of the I/O circuitry of the controller chips. Such circuitry provides for the update of the control registers; the reading of status registers; the initialization and possible update of the lookup tables; and the initialization of the cell states.

The optimum performance of CAM-8 involves operating the machine at near memory bandwidth. To facilitate this, a suitably pipelined tree bus is used to pass control and data information to the space modules. As illustrated in Figure 1.1, the pipelined bus or PBus links the space modules to an interface board which is connected to the host computer. The root of the PBus is connected to the interface board and the leaves of the bus are tied to the CAM-8 space modules. Bus nodes are located at the origin of each branch of the PBus. The tree-like structure of the bus gives the I/O system a high degree of fanout to the numerous modules that may compose a CAM-8 machine. The pipelined structure, which involves placing registers at each bus node, supports the fanout and allows the system to be run at the required throughput. The PBus operates at the CAM-8 clocking rate.

The Sun Sparcstation host computer communicates with the outside world over the Sun SBus. As described above, however, the CAM-8 controller chips communicate with the PBus using the PBus protocol. Conversion circuitry is therefore required for data and control transmission from the SBus to the PBus and vice versa. This circuitry is implemented in the interface board shown in Figure 1.1. The interface board is more than just a protocol converter. It possesses some intelligence which allows it to interpret certain low level commands from the host. The interface board therefore reads CAM-8 control programs stored in the host memory, interprets them and generates the correct PBus signals that are needed to execute these programs on the CAM-8 hardware.

The following chapters of this thesis describe the input-output protocols used in the CAM-8 I/O system as well as the hardware designs of the major sections described above. Chapter 2 discusses the typical data transfers needed to run different types of cellular automata experiments on CAM-8. Chapter 3 describes the design of the pipelined PBus and its associated protocol.

Chapters 4 and 5 of the thesis present the design and operation of the input-output section of the CAM-8 controller chip. Chapter 6 provides an in-depth discussion of the interface board. The command format, system hardware and state machine controllers of the board are described. Chapter 7 explains the present state of the implementation of the CAM-8 input-output system, and indicates what future work still needs to be done.

Chapter 2

Sample Situations for the CAM-8 I/O Interface

In this chapter we examine some situations which the input-output interface of the CAM-8 machine must be prepared to handle. It is assumed that there are a large number of space modules in the given CAM-8 machine. These modules generally listen to the same control information from the host, but will return different status information.

2.1 Situation 1

This is the simplest case for the I/O system to handle. It also probably represents the most typical usage of the CAM-8 machine. In Situation 1, the I/O system must initialize the DRAM and SRAM chips with data and lookup table information respectively.

Thereafter, the host prompts the modules with a 'GO' command after every sweep is completed. Each such command represents a particular step in the cellular automata experiment that is being conducted. When the experiment is done, the final results that have been accumulated in the module registers and any other relevant data must be read by the host.

No major timing problems appear here. The host is time constrained neither at initialization nor at the conclusion of the experiment when it reads the results. The 'GO' prompt that must be issued between steps is executed by writing to a special bit register in the controller chip register bank.

2.2 Situation 2

Situation 2 is a bit more complicated than Situation 1 described above. Here a count of the number of 1's in each of two selected bit planes for a given sweep must be accumulated. As in Situation 1, the modules are initialized with control and data. The appropriate event source select registers are written to specify the required accumulation in the chosen bit planes.

During normal operation, the host issues a 'GO' command after every sweep. While a sweep is proceeding, the host reads the double buffered event counters of each module associated with the monitored bit planes. Of course, at the end of a sweep, the modules must update these double buffered registers with data from that sweep.

2.3 Situation 3

Situation 3 involves running a repetitive cycle. In such a cycle, the lookup table changes for each 'phase'. However, each phase lookup table is small enough to occupy only a portion of the entire lookup table space. All phase tables are stored in the SRAM. By keeping the phase address bit to the lookup table fixed, a given phase lookup table may be selected.

In this situation, the experiment is again started by initializing all data and control information. The lookup table source select register for a given bit plane is set to a constant to specify the first phase. The host then issues a series of 'GO' commands to execute that phase. In between sweeps at the end of that phase, the host changes the source select and other parameters to implement the next phase. It also reads any status information that may be required. Then the host issues some more 'GO' commands to complete the following phase. This procedure is repeated until the experiment is done. Note that only control information must be changed between sweeps here (if at all).

The lookup table remains fixed throughout the experiment.

2.4 Situation 4

In Situation 4 control information and/or lookup tables may need to be changed between sweeps. The topology of the system itself may be changed, although this would probably be quite rare. Situation 4 places the tightest constraints on the CAM-8 I/O system.

The host would attempt to update the double buffered lookup tables as well as the control information between sweeps. If a total update is not possible, as much of it is done and the following sweep is held back long enough to allow for the required updates.

Chapter 3

The Pipelined Input-Output Bus

3.1 An Overview of the PBus

The pipelined input-output bus, or PBus connects the I/O interface board to the CAM-8 space modules. The PBus is a synchronous 16-bit bus which operates at the CAM-8 clocking rate. It has a high degree of fanout in order to interface to a large number of space modules which may compose a giant CAM-8. Indeed, the most characteristic feature of the PBus is that it is designed to provide for the fanout of control and data to an unlimited number of CAM-8 space modules. Figure 3.1 shows a top level diagram of the tree-like PBus structure with binary fanout.

The PBus begins as a root which consists of a single level of its data and control lines. Thereafter, the bus divides into several branches which themselves further divide into more branches. Each branch contains a replica of the control and data lines associated with the root level. Bus branches are connected by bus nodes. The bus fanout is defined as the number of output branches associated with a node. The terminal nodes of the PBus are called bus leaves. The depth of the bus is the number of nodes encountered by a direct path from the root to a leaf. In the CAM-8 input-output system, the interface board forms the root of the PBus and the CAM-8 space modules occupy the leaf

-bb-error = = □

Figure 3.1: The Pipelined Input-Output Bus with binary fanout.

-bb-error = = □

Figure 3.2: Operation of a PBus node in write and read modes.

positions. The depth of the PBus is the same for all leaves, and is proportional to the logarithm of the number of leaves supported by the bus. The fanout of the PBus is variable, and depends on the drive capability of the bus nodes. A larger fanout supports more CAM-8 space modules for a given bus depth. However, a larger fanout requires stronger line drivers on each PBus node. As mentioned above, Figure 3.1 is an example of a PBus with binary fanout per node.

The PBus branches simply consist of wires over which the control and data signals travel. The bus direction of a branch, defined by the direction of data transfer over that branch, is controlled by the leftmost node attached to the branch. In addition to controlling branch direction, a node contains storage elements to hold data in transit over the bus when the pipeline from root to leaf is held up. Although the direction of data flow over the PBus may change, control information to CAM-8 is always transmitted from the root to the leaves. The PBus has a minimum latency which is proportional to the bus depth. The bus can provide a maximum throughput equal to its clocking rate.

During a bus write, all PBus branches are configured to transmit data away from the root. Nodes broadcast data from their incoming to outgoing branches. Data and control information entering the pipeline at the root eventually makes its way to the space modules where it is absorbed by active modules. Should the controller chips be unable to absorb the incoming data (an unlikely event), the pipeline is held up and data is stored in the bus nodes. When pipeline flow is resumed, the stored data is propagated to the leaves.

In a read cycle, all bus branches are configured for data flow from the leaves to the root. Nodes propagate valid data on any of their input branches towards the root. During a read, only one CAM space module drives valid data on the PBus. All other modules in the system transmit the default data value of 1 on every PBus data line. At each node, the data on all incoming branches is logically ANDed, and the result is sent towards the root. The result is that the data transmitted from the active CAM module at a bus leaf is propagated to the interface board where it is absorbed. If the interface board is unable to absorb data at the PBus rate, the pipeline is held up until data can be accepted. The operation of a typical PBus node in write and read modes is shown in Figure 3.2. The branches labelled Data in this figure are branches which contain valid CAM data. The No Data branch contains the default value of all ones on its PBus data lines.

A major challenge of using a pipelined bus structure as shown in Figure 3.1 is that when the bus direction needs to be changed from read to write, there is potential for contention since branches closer to the root are stimulated to change bus direction before those nearer the leaves. Thus one can envision a node in the system where both the input and output branches are configured to transmit data into the node. In the PBus architecture, this contention is avoided by always prefacing a bus direction change with a propagation wave of "broken branches" through the system. A "broken branch" is a bus branch which is in a high impedance state and transmits data in neither direction. Instead of changing branch directions immediately, a branch is first put into the broken state before assuming the new direction. By propagating broken branches from the root to the leaves, there is never the case of all branches pointing into a node. As a result, contention is avoided.

3.2 The PBus Protocol

The PBus protocol is almost entirely defined by the operation of the bus nodes which tie the bus branches together. Since the root and the leaves of the bus are special bus nodes, they also execute the protocol. Figure 3.3 shows a block diagram of a PBus node. There is one PBus control line, the RD/WR line, which is propagated from the interface board to configure the PBus direction. Adjacent nodes also communicate status information with each other over two status lines which connect them together. In addition, the RD control signal generated at a bus node is used to enable the tri-state buffers of that node's right

-bb-error = = □

Figure 3.3: Block Diagram of a PBus node.

neighbors. The only other control line is the CAM-8 I/O Select line which is unused by the PBus protocol.

The Node Control Logic is responsible for the execution of the bus protocol. It directs the storage and transmission of PBus data and control information at the node. The storage circuitry is represented by the Data/Control Register block. The RD and WR signals generated by the Node Control Logic determine the direction of the branch between a node and the one to its immediate right. If RD is active, the branch transfers data into the node. If WR is active, the branch transmits data out of the node. If neither RD nor WR is active, the branch is broken and assumes the high impedance state.

Figure 3.3 shows some of the circuitry associated with each node. RD and WR are the read and write signals generated by the node; RD' is the read signal from the node's leftmost neighbor. Node fanout is also illustrated in this figure. For a fanout of n, the 16 PBus data lines are replicated n times and driven by tri-state drivers controlled by the WR control line of a node. The replicated data lines are logically ANDed as required during a PBus read to generate 16 bits of valid bus data. The buffering of data at a node occurs in the IN-OUT block of Figure 3.3. A 16-bit wide multiplexor selects the source of the input to this block — the incoming branch data for a bus write or the logically ANDed data for a bus read. The output of the IN-OUT block is either fanned out during a write or transmitted to the root during a read. The I/O Select line is pipelined so that it remains synchronized with its associated data.

A bus node contains several D-type flip-flops. Two such flip-flops register data coming along each of the I/O data lines. Another register is used to determine the direction of the bus branch between a node and the one immediately to its right. Finally, every node contains two status bits which indicate whether the node is full or empty. A node is defined to be full if it is unable to buffer more information without loss of data. Since there are two storage elements per data line, this will occur if a node is buffering two bits of data line information. An

-bb-error = =

Figure 3.4: A portion of the PBus path from root to leaf during a RD to WR transition.

empty node is one that does not contain any valid data. In a flowing bus pipeline, the nodes will be neither empty nor full — each node will be temporarily storing exactly one valid data bit per line.

A bus node also reads the status bits of the data flip-flops of the nodes immediately to its left and right. This information, together with its own status information, allows a node to execute the required bus protocol. A portion of the PBus during a read to write transition is shown in Figure 3.4.

3.2.1 Data and Status Bit Updating

Under the PBus protocol, a data flip-flop is updated whenever it can accommodate new data and such data is available.[1] This will occur by default except under the following conditions when the data flip-flops retain their old data:

1. The node is full and the node to its right in the pipeline is also full.

2. The node to the left is empty.

These conditions are illustrated in Table 3.1. The empty and full status bits are represented by E and F respectively. The status bits of neighboring nodes are labelled EL (Empty Left) and FR (Full Right). E' and F' are the new values of the node's empty and full bits. RCRC is a signal which indicates that data contained in the node is stored for the clock cycle. The table assumes that data in the pipeline flows from left to right. For a PBus write, "left" corresponds to the direction of the bus root and "right" corresponds to the direction of the bus leaves. For a PBus read operation, these directions are interchanged — the bus

[1]A data flip-flop update corresponds to a flow of data along the PBus.

EL	E	F	FR	E'	F'	RCRC
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	0	X
0	1	1	1	1	0	X
1	0	0	0	1	0	X
1	0	0	1	0	0	1
1	0	1	0	0	0	1
1	0	1	1	0	1	1
1	1	0	0	1	0	X
1	1	0	1	1	0	X
1	1	1	0	1	0	X
1	1	1	1	1	0	X

Table 3.1: Transition Table for a PBus node.

leaves are to the left and the bus root is to the right. A node should never enter a state where it is both full and empty. If this state is ever encountered, the node reverts to the empty state on the following clock cycle.

3.2.2 Changing Bus Direction

The change of bus direction is always initiated at the bus root. The interface board simply transmits the bus direction bit with every clock cycle along the one-directional, pipelined, RD/WR line. The bus direction protocol is as follows. If the RD/WR bit in one clock cycle agrees with its value on the previous cycle, then the direction of the branch between the node and the one to its immediate right is set to that direction. If the present and previous RD/WR bits do not agree, then the same branch assumes the high impedance state. As described above,

-bb-error = =

Figure 3.5: Circuitry for transfer of PBus node status signals.

this ensures that there is never any contention caused by adjacent nodes trying to drive the same line in different directions.

The bus direction should not change from Write to Read until all previously transmitted valid data has been accepted by the space modules. This is ensured by only propagating the RD/WR bit through a node if the node was previously in write mode and it is presently empty, or if the node was previously in read mode. Otherwise, the old bus direction associated with that node is maintained.

3.3 Node Circuitry

This section describes some of the circuitry that is used to implement the PBus protocol described above.

3.3.1 Transfer of Status Information

For the proper operation of the PBus protocol, a node must show the value of its empty and full flip-flops to its neighboring nodes. It must also read the empty and full flip-flops of its neighbors. The transition equations described earlier indicate that a node need only read the Full bit of the node to its immediate right in the logic pipeline[2] and the Empty bit of the node to its left.

These restrictions give rise to the circuitry shown in Figure 3.5. Two wires are used between adjacent nodes. Each of these wires carries data in one direction from one node to the other. In Read mode, a node passes its empty bit to the node at its left, and its full bit to the node at its right. It accepts the empty bit from its right neighbor and the full bit from its left neighbor. During Write mode these directions are reversed. As illustrated in the diagrams, the circuitry essentially

[2]The logic pipeline goes from root to leaf in Write mode and from leaf to root in Read mode.

-bb-error = = □

Figure 3.6: Data Register Circuit for a PBus node.

consists of four 2-to-1 multiplexors correctly wired. The status input signal from the right node is actually the logic OR of the Right Status In signals from all adjacent right nodes. Since the Empty or Full status signals are active low, and inactive CAM modules never hold up the bus, transfers should proceed only when all fanned out nodes can send or receive data. This is achieved by the logic ORing of the right status inputs.

3.3.2 The Data and Status Flip-Flops

Figure 3.3 indicated that the data buffering circuitry of a PBus node is concentrated in the IN-OUT block. Figure 3.6 shows the circuitry contained in this block. The Data register circuit consists of two D-type flip-flops and three 2-to-1 multiplexors per data line. The top flip-flop is used to pipeline the data bit. The lower flip-flop normally stores a copy of the last bit that has been advanced to the right in the logic pipeline. This prevents loss of data in the case when that bit was not accepted by the neighboring right nodes because of a pipeline holdup. In such a case, the Data register circuit stores two consecutive data bits until the pipeline resumes normal flow.

The I/O Select control line to CAM-8 passes through an identical circuit. The only difference between this line and the data lines is that the I/O Select line always goes from the root to the leaves of the PBus. The data lines, however, are bidirectional.

The logic equations for the control signals of the Data register circuit are derived from Table 3.1:

```
RCRC  = EL + F*FR

WHICH = F
```

The status bits are held in Node Control Logic D-flip-flops which are updated according to the following equations[3]:

```
F := (FR*/E*/EL + F*/E*/EL + FR*/E*F)*(RD + WR)

E := E*EL + EL*/FR*/F + E*F + /RD*/WR
```

The equations for E and F also contain terms for the bus direction. This is necessary to initialize the bus to the empty state on a switch of direction.

3.3.3 The Bus Direction Registers

The bus direction register circuit is shown in Figure 3.7. This circuit contains two serially configured D-flip-flops which hold the present and previous values of the RD/WR bit that is propagated from the host. The logic AND of these signals is used to configure the bus branch to write mode while the logic NOR of the signals generates the read mode.

The top flip-flop of Figure 3.7 is a synchronizing flip-flop. Its purpose is to pipeline the Rd/Wr line of the PBus. The lower flip-flop is needed to provide the delayed version of the Rd/Wr signal. When the present and delayed versions of Rd/Wr do not agree, a change in bus direction is required. The RD and WR control lines of the node are toggled accordingly.

A 2-to-1 multiplexor chooses the input of the first of the series flip-flops. If the node was previously in write mode but is now empty, or if the previous mode was read, then the flip-flop is fed with incoming Read/Write data. Otherwise, the old value of the RD/WR bit is recirculated through the multiplexor. This procedure ensures that a Write to Read transition of the bus lags behind any previously transmitted valid data.

3.4 Conclusion

The PBus protocol described in this chapter has several advantages. When the bus is read as fast as data is put into it, none of the nodes

[3]These were derived from the transition table using Karnaugh maps to simplify the expressions.

will be full or empty. Thus data advances from node to node with each clock cycle, at the optimum rate. The buffering effect of the bus also ensures that no valid data is lost because of a pileup along the pipeline. Also, because the bus only responds to valid data[4], any invalid data generated by a slow bus driver is automatically thrown out of the pipeline to make way for valid information. Thus, invalid data bits do not hold up the pipeline. Because the bus itself indicates that data is valid, a separate Valid data line to the space modules is not needed.

[4] Data is defined as valid if a node is not empty.

-bb-error = =

Figure 3.7: Bus Direction Circuitry for a PBus node.

Chapter 4

The Input-Output Section of the CAM-8 Controller Chip

This chapter describes the top level design of the input-output section of the CAM-8 controller chip. The I/O section of the chip is responsible for allowing the host to initialize and update the control registers, lookup tables and system state. It also allows the host to read any status information that has been accumulated in the module as a result of an experimental run.

A top-level block diagram of the I/O circuitry is shown in Figure 4.1. In the following sections, the function and interconnection of each block of the figure will be explained.

4.1 The Register Bank

The Register Bank consists of a series of 1-bit and 8-bit registers which store the control and status information associated with each controller chip. This is the set of registers which the host needs to access. All

-bb-error = =

Figure 4.1: Block Diagram of the I/O Section of the CAM-8 controller chip.

registers are shift registers with their inputs tied to the serial I/O data line and their outputs connected to the same line through tri-state buffers. The Register Select block enables one of the registers in the register bank for I/O access.

In Read mode, the tri-state associated with the selected register is turned on so that the data contained in the register may be serially transmitted out along the I/O data line. In Write mode, this tri-state is turned off[1] so that the host can drive the I/O data line with the data that needs to be written to the selected register.

The registers of the Register Bank are wired to other parts of the chip so that they may control the logic of these other sections.

4.2 The Register Select Block

The Register Select Block is responsible for enabling one of the registers of the Register Bank for I/O access. It takes a Data Register enable signal from the Enable Logic block and a register number from the Opcode counter circuitry. If the enable line is active, the register specified by the counter is enabled. Thus the Register Select block acts as a decoder for the registers in the bank. However, it is not realized simply as a giant decoder. Such a design requires too many wires running to the register bank. Instead, this block takes the form of combinational logic ANDs located in the proximity of the individual registers in the bank. The inputs of these ANDs are tapped off a bus consisting of the register number and the enable signal.

4.3 The Test/Verify Block

The Test/Verify block contains circuitry that compares bits stored in a selected register with incoming bits from the host. This allows the host to verify the data that it has written to the Register Bank. If there is any discrepancy between written and transmitted data, then an interrupt line is activated to indicate the error.

[1] The tri-state may actually be turned on depending on the implementation of the Write-Verify logic.

It is thought that it would be optimum to always compare data that is being written with the previous data in the selected register. The interrupt line, however, may only be set when the Write-Verify operation is actually being performed.

4.4 The Opcode Register Circuitry

The Opcode Register circuitry consists of a 7-bit opcode shift register, a 7-bit counter and some combinational logic that enables the counter.

The register that the host wishes to select for an I/O operation is written to the opcode register when the I/O Select line is low. The Enable Logic block ensures that the opcode register is enabled in this case. The number of the selected register is then written with the least significant bit sent last. This form of transfer allows some registers to be specified without writing all 7 bits into the opcode register.

When the I/O Select line goes from high to low, the opcode register is cleared by the Enable Logic. Register one may be specified by sending a single '1' before bringing the I/O Select line high again. At this point, the counter is loaded with the opcode register data and the selected register number is provided to the Register Select block.

The Counter Logic is used to implement an auto-increment feature for register selection. If the selected register is in the 1-bit register space, then the counter is incremented with the arrival of every valid bit of data. If an 8-bit register is selected, then the increment occurs after the arrival of 8 valid bits. This allows registers to be selected in a sequential manner for I/O operations.

Some registers are defined to be non-auto incrementable. The counter logic will inhibit the auto-increment feature on these registers.

4.5 The Enable Logic

The Enable Logic is responsible for decoding a global module enable signal provided by the Global Module Enable block into an opcode or data enable signal. It also generates the signal to clear the opcode shift register and to load the opcode counter.

The Enable Logic uses the I/O Select line to generate its control signals. If there is a Global enable and the I/O Select line changes from high to low, then the clear signal to the opcode shift register is generated. Under the same enable, if this signal changes from low to high, then the load counter signal is activated. A global module enable on an I/O select line that is low enables the opcode shift register, while this enable on a high I/O select line enables the data registers.

4.6 The Module Select Block

This block is responsible for indicating whether the given module is in fact selected for I/O purposes. It contains comparison logic to determine whether the host selected module or group matches the ID or group number associated with the module. It may therefore access some of the registers contained in the register bank.

The module select block also takes as input the broadcast line from the host. Once this line is high, all modules are selected for a write operation. An individual module may be selected by initiating a broadcast and writing the ID or group number to the ID or group select registers of all the modules. When the broadcast line is brought low, the module(s) corresponding to the selected ID or group will be selected by the comparison logic of the module select block.

4.7 The Global Module Enable Block

The Global Module Enable block generates a global module enable signal if a module is selected and the I/O bus is providing or expecting data. The enable signal merely means that the chip should be either sending or receiving data in that clock cycle.

Essentially, this block acts as the interface between the chip and the PBus. It turns on the chip when the bus is ready to accept or send data, and it turns off the chip when the bus is full or empty.

4.8 The Module Status Block

The Module Status block is responsible for providing the Empty or Full status of the chip to the PBus node adjacent to the controller chip. Because of the nature of the I/O design, the controller chip is normally always ready to accept or send data. Thus, by default, the chip status is defined to be neither empty nor full. However, for such functions as the update of the DRAM chip, the I/O needs to interact with the scan circuitry, which may not always be able to process data in one clock cycle. In these cases, the module status may be affected by the operation of the scan circuitry. This circuitry is therefore allowed to set and clear the full and empty bits.

Chapter 5

The Controller Chip I/O Protocols

This chapter describes some of the standard input-output protocols that are used by the CAM-8 controller chip. Although a sound understanding of the chip hardware is not assumed, the reader may find it helpful to browse over the hardware schematics when reading this chapter.

5.1 The Register Bank Updating Protocol

This section describes the protocol that is used to update the register bank of the input-output section of the CAM-8 controller chip. The typical unit of stimulation and update in CAM-8 is the space module. As described previously, a space module contains 16 controller chips, each with its own register bank. In a CAM-8 register access, the identical bit sequence is first transmitted to all controller chips in a space module to select the same input-output register in all of these chips. Thereafter, during a write operation, different data bits may be sent to the register in each chip. For a read operation, different data contents can be expected from each controller chip.

In order to simplify the protocol and reduce the number of wires that must be routed to the controller chips, information to each chip is serially transmitted and received. Two input-output lines are used by the interface board for register update and readback — the I/O Select line and the I/O Data line. A single I/O Select line is connected to all controller chips in a space module. Each controller chip is tied to a unique I/O Data line of the 16 bit PBus.

The I/O Select line is used to differentiate the updating of the Opcode register from that of the data registers of the register bank. A low on this line indicates that the Opcode register is to be accessed and a high indicates that the data registers are addressed. The Opcode register stores the address of the currently accessed register of the Register Bank. The I/O Data line serially transmits the contents of the Opcode register as well as the actual data bits that are to be stored in the registers of the register bank. The protocol described in this chapter indicates which hardware control signals are asserted for given changes in the I/O Select line. It also indicates at what times the I/O Data line must be guaranteed to contain specific data.

5.1.1 The Protocol

Based on the hardware design of the input-output section of the controller chip, three distinct operations must be completed to facilitate register access:

1. The Opcode register must be cleared.

2. The Opcode register must be loaded with address data.

3. The selected register must be enabled.

The first two operations load the selected address into the Opcode register and the final operation enables the chosen register for input-output access.

Clearing the Opcode Register

The Opcode register is cleared to erase any previous contents that it may have contained. Since the address is transmitted with the least significant bit sent last, low addresses may be loaded in fewer than the seven clock cycles that it would normally take to fill the 7-bit Opcode register. For example, register 0 may be selected by simply clearing the Opcode register.

The Opcode register is cleared when the I/O Select line transitions from high to low. As shown in Figure 5.1, the clearing occurs during the clock cycle immediately following the fall of the I/O Select line. During this clock cycle, the parity word is sent to the controller chip. The chip checks that the parity bits are of the expected values, and generates an interrupt if there is some error. Immediately after reset, the Opcode register of the controller chip is automatically cleared.

Loading the Opcode Register

Once the Opcode register has been cleared, it must be loaded with the address of the register to be accessed. This is done by serially shifting address data into the Opcode register. If the I/O Select line is low in a given clock cycle and it was low in the previous clock cycle, then the Opcode register is enabled. The interface board must hold the I/O Select line low for at least one clock cycle and for no more than eight clock cycles.[1]

The I/O Data line must contain parity data on the clock cycle immediately following the high to low transition of the I/O Select line. It must contain valid Opcode data from the second clock cycle after I/O Select has gone from high to low until the clock cycle immediately preceding the low to high transition of the I/O Select line. This data must be the serially transmitted register bank address to be selected.

Once the Opcode register has been loaded and the I/O Select line goes high, the Opcode register acts as a counter to implement the auto-increment feature for register access. The enable line for the counting is derived in the Counter logic of the controller chip.

Enabling the Selected Register

After the correct address has been loaded into the Opcode register, the Enable logic block “turns on” the selected register. This enabling

[1] One clock cycle clears the Opcode register; eight clock cycles clear and fully load it with new address data.

-bb-error = =

Figure 5.1: Time Line for the Register Bank Updating Protocol.

occurs if the I/O Select line is high in a given clock cycle and the chip is enabled.

An enabled register programmed for write expects valid register data to be serially transmitted on the I/O Data line. Thus, valid data must be present on this line by the clock edge immediately after the low to high transition of the I/O Select line. The interface board must transmit just the right number of data bits to match the size of the register that it has selected. Of course, in auto-increment mode more data can be sent to update the contents of sequentially addressed registers.

If the register is in read mode, then the I/O Data line will contain the high order bit of the selected register on the first clock edge after the low to high transition of the I/O Select line. The remaining contents of the register are serially transmitted out on subsequent clock cycles.

5.1.2 Additional Details

A time line diagram illustrating the protocol described above is shown in Figure 5.1. The discussion has assumed the continual enabling of the controller chip. In reality, the chip is enabled only in those clock cycles during which it has been selected for I/O and the PBus node connected to it can send or receive data. The protocol still holds, however, so long as a "clock cycle" is taken to mean a chip enabled clock cycle.

5.2 Initialization of the ID Registers of the Space Modules

Before an experiment can be run on CAM-8, every space module in the machine must be assigned a unique identification number or ID. This allows the software to uniquely select each space module for register access. In addition, space modules may be tagged together to form groups of modules. The host may then simultaneously write the same data to all modules in a given group. This saves the overhead of having to sequentially select all modules in a group only to write the identical data to all of them.

The ID and Group labelling of the space modules is programmed by the CAM-8 software. This is advantageous since it eliminates the need for hardwiring the modules. Software control also provides the flexibility of changing the system configuration from experiment to experiment. Since a complete space module is the basic unit of selection in CAM-8, the ID and Group numbers associated with a module are distributed over all 16 controller chips which make up the module. Each space module has an associated 16 bit ID register and a 16-bit Group register. One bit of these registers is stored in each controller chip of the space module. A corresponding ID select and Group select register is also associated with each component controller chip. A space module is normally selected for I/O when the ID select or Group select registers of all the controller chips in the module match the programmed ID or Group values. Communication across the component chips of the space module for the match comparison occurs through the use of a wired AND implementation. All system modules may be selected in unison by the activation of a PBus Broadcast line which is routed to every CAM-8 controller chip.

The assignment of ID and Group numbers presents the host with the problem of selecting individual modules for initialization in the absence of any unique information with which to turn on a module! Indeed, upon reset, all of the space modules in the system appear identical to the host. This problem is overcome as described below.

A module may be enabled for I/O if there is either a match or mismatch between its programmed and selected ID or Group registers. An Invert bit in the register bank of the chip determines whether the module is selected for an ID or Group match (Invert Bit = 1) or mismatch (Invert Bit = 0). In addition, a module may be selected if the incoming glue line to that module is at a specified logic level. As mentioned above, normal operation corresponds to an Invert bit value of 1. However, the mismatch and glue line selection are utilized in the initialization routine as follows.

The host first broadcasts to all modules and writes an ID number of 0 to them. It then sets the Invert bit value of '0' as the selection condition. The hardware is arranged so that the host is tied to the open collector network of one of the space modules. Now the host selects on the module with ID number of 0. Of course, all modules are deselected since their open collector wired AND lines will remain high as a result of the match between the programmed and selected IDs. The host, however, pulls the open collector line that it is connected to low and forcibly selects one of the space modules. The correct ID number is written to this module and the host instructs the module to send a '1' across its glue line to the module to its right.

Next, a broadcast occurs, and the host indicates a selection of the module which receives a '1' on the glue line from its left neighbor. The Invert bit is reset to 1 for normal operation and the ID of the newly selected module is written. Now the host broadcasts that all modules should send a '0' over their glue lines. The host reselects the last ID updated module using its programmed ID and configures it to write a '1' to its right neighbor. A selection is now done on the module receiving a '1' from its left neighbor, and the ID number of that module is written. The procedure is repeated until every module in the entire sea of space modules is uniquely labelled.

5.3 The Fast Sequential Readback Mode

A typical operation of the CAM-8 machine involves running an experiment on it and then reading back the accumulated sums in the event counters. Under normal I/O operation, reading the event counters would require that the host sequentially select each space module and then perform a read operation. The host must transmit the module ID to select a module and then read the event counter. This process is inefficient because it involves transmitting sequential name IDs and changing the bus direction twice for every event counter access. Changing the bus direction requires a holdup equal to the pipeline latency of the bus. It was therefore preferable that a special mode be designed to handle this situation more efficiently. The Fast Sequential Readback mode involves using the Lookup Table address counter value to select a space module for readback. There is no conflict of operation since this counter will not be in use when an event counter readback is to occur.

A fast sequential readback is initiated by first loading the Opcode register with the address of the event counter. The Module Select hardware is programmed to enable a space module when its ID register matches the value of the address counter of its Lookup Table. In this mode, special circuitry increments the Lookup Table counters after 25 bits of data have been successfully transmitted over the PBus.[2] The address counters of all of the Lookup Tables in the system are cleared and enabled. The system is then allowed to run. The result is the sequential transmission of event counter information from the space modules to the host. Although the Lookup Table counters in all modules count simultaneously, only the selected space module will drive data to the host. As soon as one space module has completed sending its event counter data, it automatically becomes deselected. The module whose ID is one higher then starts to transmit its event counter data to the host. This continues until all event counters in the system have been read by the host.

[2]The event counter is 25 bits in length, so the Lookup Table counter is incremented once for every event counter read.

Chapter 6

The CAM-8 Input-Output Interface Board

The input-output interface board of the CAM-8 machine connects the high speed pipelined bus of the CAM-8 space modules to the SBus of the host Sun Sparcstation. The board coordinates the transfer of data from the host memory to the pipelined bus during a CAM write cycle and a reverse transfer during a read.

The interface board is designed to conform to the protocols of both the SBus and the pipelined input-output PBus to which it is connected. The SBus runs at a clock rate of 20MHz in the Sparcstation. The PBus, however, runs at the CAM-8 clocking rate of 25MHz. The interface board ensures proper data transmission between these separately clocked systems.

The interface board has a unique board address in the SBus system. It acts as an SBus slave to allow the CPU to program it to start a data transfer. However, the board primarily behaves as an SBus master as it reads from and writes to the host memory. In typical operation, the board reads and executes a 'Display List' which is located somewhere in the host memory. The Display List is a sequence of commands which controls the operation of CAM-8. For example, a DMA write to or read from CAM-8 may be specified. Also, execution of the Display List may be programmed to halt until the occurance of a given event such as the CAM-8 End-of-Scan interrupt or a video HSYNC or VSYNC signal.

The interface board operates as independent of the host as possible.

Although it controls memory transfers, the board does not function as a conventional DMA chip: it does not interrupt the host after every data transfer.

All accesses by the board as a master involves 16 byte burst transfers. Thus, the burst mode is used for reading the Display List, reading data from the host memory, and for writing data to the host memory. For slave operation, only SBus word transfers are supported.

The following sections describe in detail the design and implementation of the CAM-8 input-output Interface Board. It is assumed that the reader is familiar with both the SBus Specification and the PBus protocol for communicating with CAM-8.

6.1 The Display List Command Format

This section describes the format of the Display List commands that are used to program the interface board. All commands consist of 16 bytes which are partitioned into four words as follows:[1]

Opcode Reg; Flags; Cond	Start Address of XFER	Length in Half Words	Pointer to Next Disp Item

The board operation will be discussed with reference to the four fields which define its instruction format.

6.1.1 Opcode Reg; Flags; Condition Field

This field contains the primary information that controls the board operation. The Flag bits specify whether a write to or read from CAM-8 is to be executed. They also determine the levels on the control lines to be sent to CAM-8 while data is being written to or read from the CAM-8 registers.

The contents to be written to the CAM-8 Opcode Register is also included in this field. The interface board accesses a CAM-8 register by first generating the correct control lines to write the address of the selected register into the CAM-8 Opcode Register. Once this register has been selected and written to, any following data is written to or read starting from that register. Auto-increment from a register is achieved by reading or writing long streams of data after selecting a register. The flags indicate whether data is immediate or to be read from the host memory.

The condition bits specify whether the interface board should wait on the video HSYNC, VSYNC, End-of-Scan, or the host before continuing to execute the Display List.

[1]Note that the 16 byte command format is logical since the board as a master only supports 16 byte burst transfers.

6.1.2 Start Address of Transfer

The Start Address field specifies the host memory address for the start of the burst transfer operation. When reading from memory, this is the first address accessed by the interface board. When writing to memory, Start Address specifies the first address written to by the board.

In the special case where immediate data is being specified by the instruction word, the Start Address field stores this immediate data representation.

6.1.3 Length Field

The length field defines the length of the DMA transfer to or from CAM-8 in terms of 16-bit half-words. During a read or write operation, its value is decremented for each half-word transmission over the PBus. When Length has been decremented to zero, the I/O transfer is complete. The interface board then fetches the next instruction in the Display List to begin a new execution.

6.1.4 Pointer to Next Display Item

This field contains the address of the next Display List item to be read and executed by the interface board. For sequential operation of the system, when a transfer is complete, the board reads the instruction at this address and begins the new execution.

Upon system initialization, the board is programmed to wait on the host. The host accesses the board as a slave and writes the address of the first instruction of the Display List in a corresponding slave register. If the host has written the Display List instruction address to the interface board, then the slave register is used instead of the Next Display Item field of the instruction word to access the next Display List instruction. This Next Display Item pointer must be correctly aligned on a 16-byte boundary of the host memory for proper operation.

-bb-error = =

Figure 6.1: Block Diagram of the Interface Board.

6.2 The Interface Board System

This section provides a system level overview of the interface board hardware organization. A top level block diagram of the board is shown in Figure 6.1. For master operation, two finite state controllers are utilized: an SBus FSM (SFSM), which runs at the SBus clock rate; and a PBus FSM (PFSM), which operates at the CAM-8 clock rate. These controllers oversee the loading and unloading of the Control, Datain and Dataout registers. A Slave FSM which operates at the SBus clock rate controls the board in slave mode. This involves loading and driving the Slave Register Bank contents. The interface board also contains a timeout counter and interrupt hardware. These allow the board to interrupt the host should a timeout or error condition be encountered. A Parity Generator on the board produces serial odd parity bits for data transfers over each of the 16 PBus data lines.

To transfer data from the SBus to the PBus, the data must first be loaded into the Control or Datain registers by the SFSM using the SBus protocol. Subsequently, data is driven from the Datain registers onto the PBus by the PFSM, which uses the PBus protocol. A reverse data transfer uses the Dataout registers to hold the data in transit. These registers are loaded by the PFSM and driven by the SFSM.

In the following subsections, the functionality of the different blocks shown in the system block diagram will be explained in further detail.

6.2.1 The Control Registers

The Control Registers store the present Display List instruction word that is being executed. The instruction word parameters were described in detail in the previous section. The Control Registers are loaded by the SFSM in accordance with the SBus protocol.

Both the SFSM and PFSM interpret the contents of the Control registers for processing the data. For example, the PFSM loads the transfer length from these registers in order to transfer the correct number of PBus words during a CAM read or write. It also obtains the contents of the Opcode register from the Control registers before it starts a data transfer to CAM-8. The SFSM uses the Start Address of Transfer and Next Display Item pointers in the Control Register word to further access memory for subsequent data and instruction transfers. The SFSM also uses the conditional information of the Control Registers to coordinate waits between Display List instructions.

The contents of the Start Address or the Next Display pointer registers are driven on the data lines of the SBus by the SFSM when the board initiates a data transfer over the SBus.

6.2.2 The Datain Registers

The Datain Registers consist of four 32-bit registers which are loaded from the SBus by the SFSM. These registers hold any data that is in transit from the SBus to the PBus. Once the Datain registers have been loaded with data, the PFSM is triggered to transfer their contents over the PBus. Of course, during the PBus transfers, the Datain Registers are read as 16-bit CAM-8 data words. The registers may be reloaded with new data until the entire DMA transfer to CAM-8 has ended.

6.2.3 The Dataout Registers

The Dataout Registers consist of eight 16-bit registers. They are loaded from the PBus by the PFSM and contain data that is in transit from the PBus to the SBus. Once the Dataout registers have been loaded, the SFSM is triggered to transfer their contents over the SBus. The registers are driven as four 32-bit registers on the SBus and may be reloaded with new data until the entire DMA transfer from CAM-8 has ended.

6.2.4 The SFSM

The SFSM is a finite state machine controller which is clocked at the SBus clocking rate. It interprets and executes the 'program' that is stored in the Display List. This involves correctly loading the Datain Registers; driving the Dataout Registers; and holding up program execution until specified conditions are met.

6.2.5 The PFSM

The PFSM is a finite state machine controller that is clocked at the PBus or CAM-8 clocking rate. When a new instruction is read into the Control Registers, the PFSM generates the sequence of PBus control signals to transfer the Opcode Register contents over the PBus. For a write to CAM-8, the PFSM then proceeds to drive the contents of the Datain Registers over the PBus. For a read from CAM-8, it loads the Dataout Registers with the requested data.

The PFSM is also responsible for decrementing a counter that contains the remaining length of the data transfer. When that value has been decremented to zero, the SFSM is informed that the data transfer is complete.

6.2.6 The Slave Register Bank

The Slave Register Bank consists of three 32-bit registers and several 32-bit hardwired values. The three registers are used to store the pointer to the Next Display List item supplied by the host, the address of the current instruction being executed, and the interrupt polling bits to be read by the host. These are required registers for the proper operation of the interface system. The remaining hardwired values are those required by Sun for any SBus slave. They include the board ID number and other slave tokens. The contents of any item in the Slave Register bank may be driven by the Slave FSM over the SBus. The pointer to the Next Display List item is loaded from the host, but the current instruction address is loaded from the board by the SFSM. This is explained in further detail in a later section. The interrupt polling bits are set by the Interrupt Hardware and read by the SBus Controller.

6.2.7 The Slave FSM

The Slave FSM operates at the SBus clock rate and coordinates the board as an SBus slave. It monitors the SBus control signals and generates correctly timed load and drive pulses to the Slave Register Bank. The Slave FSM operates independent of the other two FSMs in the interface board system.

6.2.8 The Timeout Counter

The timeout counter is an 8-bit counter which is used to generate an interrupt to the Interrupt Hardware if there is no response from CAM-8 within 256 clock cycles of a CAM read operation.

6.2.9 The Interrupt Hardware

The Interrupt Hardware is responsible for activating an SBus interrupt to request CPU intervention on the board. Interrupts may be generated if data transfer errors or timeouts occur, or if the Display List instruction word specifically requires the host to write to the Host Next Pointer register.

6.2.10 The Parity Generator

The Parity Generator produces parity bits for PBus data transfers. Parity generation begins with the transmission of the first bit of the Opcode register over the PBus and continues until the current data transfer is complete. A serial odd parity bit is generated for each of the 16 PBus data lines. The parity word is transmitted to CAM at the end of the current data transfer. This word is used by the controller chips of CAM-8 to determine whether there were any transmission errors over the PBus.

-bb-error = = □

Figure 6.2: The Control Registers of the Interface Board.

6.3 Datapath Specifications

This section describes in detail the datapaths and hardware organization of the interface board. The Control, Datain and Dataout register banks are discussed. In addition, the necessary synchronization circuitry, timeout counter, and interrupt and parity generation hardware for the board are presented in this section.

6.3.1 The Register Banks

The Control, Datain and Dataout registers are arranged in banks each consisting of four 32-bit registers. Each bank is separately addressed by the SCTR and PCTR counters. The SFSM accesses the register that is pointed to by the SCTR while the PFSM accesses the register addressed by the PCTR. Thus all SFSM control signals to the register banks activate the register pointed to by the SCTR, while all PFSM signals affect the PCTR addressed register.

6.3.2 The Control Registers

The Control Registers logic diagram is shown in Figure 6.2. These registers store the four 32-bit SBus words which make up a single Display List instruction. The Control Registers are labelled R0 thru R3. Their contents are as follows:

R0 : Flags, Condition Bits, Opcode Contents
R1 : Start Address of Transfer or Immediate Data
R2 : Length of Transfer in Half Words
R3 : Pointer to Next Display Item

Each of the four Control Registers will now be examined in greater detail.

The R0 Control Register

The high 16 bits of the R0 register contain flags such as WR; the immediate flag, which says whether the data is immediate; as well as the condition bit encoding which declares what condition, if any, the board should wait on before executing the next Display List item. The low 16 bits include the 7-bit Opcode contents which declares the CAM-8 register to which the data transfer is directed.

Both the SFSM and the PFSM read the information contained in R0. A shift register clocked by the PBus clock loads the Opcode contents from R0 when LDCTL is active. The shift register allows the PFSM to shift the Opcode data to CAM-8 using the PBus protocol. Shifting is controlled by the PSH line of the PFSM. A '0' is always shifted into the shift register. Thus, when all the shift register bits are zero, no further shifting to CAM-8 is needed for Opcode specification. At this point, PSELDONE becomes active. The serial output of the shift register is fanned out to the 16 PBus data lines through tristate drivers. These drivers are enabled when both PIOSEL and DRPTY are low. Thus, whenever this is the case, the PBus data lines are driven with the required Opcode data.

The R1 Control Register

The contents of the R1 register depends on the value contained in the Immediate flag bit of R0. If this flag is active, then R1 contains the value of the immediate data to be sent to CAM-8. This data would have been loaded into the first Datain Register at the same time that R1 was loaded.

When the immediate flag is not active, R1 contains the starting virtual address of the DMA transfer. In the case of a write to CAM-8, this is the address of the first 32-bit word to be sent as data to the selected CAM-8 register. For a read from CAM-8, the address represents the start address to which data from CAM-8 should be written.

The R1 register also acts as a counter during DMA transfers. Since all transfers are 16 byte burst transfers, INCADR is asserted by the SFSM to increment the virtual address by 16 bytes upon the completion of each successful burst. This ensures that the correct address is driven onto the SBus for the subsequent burst transfer.

The R2 Control Register

This register contains the length in CAM-8 words (or SBus half-words) of the data transfer. The LDCTL signal loads the transfer length into a length counter clocked by the PFSM. This counter is decremented with each PBus word transfer. When the counter value becomes zero, the DONE flag is set to indicate that the transfer is complete. The flip-flop structure that is gated to generate DONE serves to turn the DONE flag on or off. This is useful to avoid interpreting the DONE from a previous transfer as valid for the present transfer. Essentially, the PFSM sets the DONE lookout flag on by asserting LDCTL when it loads the length counter with new data. The flag is taken off by the SFSM when a data transfer has been completed. This is achieved by the assertion of CLRDONE.

The R3 Control Register

The R3 Control register contains the pointer to the next item of the Display List. When the current instruction has been executed, the contents of this register is driven onto the SBus as the virtual address of the next intruction of the Display List to be executed. There is one exception, however. If there is a Wait-for-Host condition and the HOSTFLG has been set, then the next instruction pointer is driven from the slave Host Next Pointer register. In this case, the drive line to R3 is inhibited.

Strictly speaking, if there is no required branching in the Display List execution, this portion of the Display List word would not be needed. The program counter could simply be incremented as each instruction is executed.

The loading of R3 sets the instruction ready (SR3DAV) flag. This informs the PFSM that a valid instruction has been loaded into the Control Registers. The PFSM acknowledges this by clearing the flag with LDCTL when it loads the instruction data into the shift register and length counter.

-bb-error = = □

Figure 6.3: The Datain Registers of the Interface Board.

-bb-error = = □

Figure 6.4: The Dataout Registers of the Interface Board.

6.3.3 The Datain Registers

The Datain Registers logic diagram is shown in Figure 6.3. These registers are loaded as four 32-bit registers from the SBus and driven as 16-bit registers onto the PBus. As in the case of the Control Registers, the SCTR and PCTR counters select the register to which load or drive signals are applied.

The loading of each 32-bit Datain Register from the SBus sets a data available flag (labelled SR4DAV thru SR7DAV). These flags inform the PFSM that valid data is present in the registers. Thus the PFSM may begin transferring data from R4 to CAM-8 even before register R7 is loaded. This reduces the transfer time for data to be written to CAM-8. The data available flags are reset by the PFSM with the assertion of PCWR.

When the PFSM has driven the last CAM-8 data word onto the PBus,[2] the PDRW flag is set. This informs the SFSM that all data written to the Datain registers have been processed. These registers may now be refilled with new data.

6.3.4 The Dataout Registers

The Dataout Registers are shown in Figure 6.4. Their setup is very similar to that of the Datain Registers described above. Here, however, each register is loaded as two 16-bit registers from the PBus and driven as a 32-bit register onto the SBus. The Dataout Registers are clocked at the CAM-8 rate.

During a read from CAM-8, the Dataout Registers are loaded with data from the PBus as quickly as possible. Once the final half-word of R7_OUT is loaded, the buffer becomes full and the PDRD flag is set.

[2]This will occur when the last half-word of R7 is driven.

-bb-error = = □

Figure 6.5: Synchronization of the PDAV signal.

-bb-error = = □

Figure 6.6: Synchronization of SFSM input signals.

This stimulates the SFSM to drive the contents of the registers onto the SBus. When the SBus transfer is complete, the SFSM clears the flag by asserting SCRD. The PFSM may now refill the registers with new PBus data.

The OR gate that logically ORs the PDRD and DONE signals requires some explanation. If the number of CAM-8 words to be read is not a multiple of eight, then the Length counter would be decremented to zero before the Dataout buffer is full. Once the length has gone to zero, however, the SBus transfer should be started since all required data has been read from CAM-8. Thus the data available flag to the SFSM is the logic OR of DONE and PDRD. Note that it does not matter that data is always transmitted in groups of eight CAM-8 words. The host which processes the data read from CAM-8 should just read the correct number of words that it expects from the transfer. The contents of the remaining words are meaningless to the host in this context.

6.3.5 Synchronization Hardware

The interface board contains two completely asynchronous clocked subsystems — the SBus clocked SFSM and the PBus clocked PFSM. Status bits from one system to the other must be correctly synchronized to the clock of the controller to which they are input. The synchronization is illustrated in Figures 6.5 and 6.6.

The data available bits from the SFSM to the PFSM are multiplexed to generate a single S-synchronized data available line. This line passes through an S-to-P synchronizer which synchronizes it to the PFSM. The output of the synchronizer is the PDAV signal. As shown in Figure 6.5,

-bb-error = =

Figure 6.7: The Timeout Counter of the Interface Board.

the PCTR determines which of the SRDAV lines is to be interpreted as the PDAV signal at any given time.

Figure 6.6 shows that the PDRD1, PDRW and PFSM generated DONE signals are synchronized to the SFSM to generate the DRD, DRW and DONE inputs respectively. In addition, the selected condition bit is synchronized to the SBus clock before it is input to the SFSM.

The S-to-P and P-to-S synchronization are achieved by passing the signal to be synchronized through two serially connected D-type flip-flops clocked at the synchronizing clock rate. The output of this arrangement is a correctly synchronized input signal.

6.3.6 The Timeout Counter

The interface board contains an 8-bit Timeout counter which is clocked at the PBus or CAM-8 clocking rate. It serves to interrupt the host if there is no response from CAM-8 within the first 256 clock cycles of a CAM read. This would indicate that an invalid module select ID had been written to CAM, or that there is some malfunction in the CAM-8 hardware. In either case, an interrupt line must be generated to the host so that the situation can be further investigated.

As shown in Figure 6.7, the Timeout counter is cleared by the LD-CTL line from the PFSM. Counting is enabled if EOFIN is high and the PBUSDIR line is low — that is, during a CAM read. If the EOFIN line becomes low before the timeout, then CAM has responded to the read request, and the counter is disabled. The ripple carry out of the counter is deglitched through a flip-flop to produce the TIMEOUT signal to the Interrupt Hardware.

-bb-error = =

Figure 6.8: The Interrupt Hardware of the Interface Board.

-bb-error = =

Figure 6.9: The Edge Detection FSM of the Parity Generation Hardware.

6.3.7 The Interrupt Hardware

Figure 6.8 shows the realization of the Interrupt Hardware of the interface board. This circuitry is responsible for triggering an interrupt to the host CPU if a timeout, error condition, or programmed interrupt is encountered.

As mentioned above, a Timeout interrupt is generated if a CAM read is requested and CAM does not respond with data within 256 CAM clock cycles. An Error interrupt is generated by the SFSM if either an Error Acknowledgment or Late Error is encountered during an SBus transaction. A "soft" interrupt is a programmed interrupt to the host. It is initiated if the Interrupt bit in the Flag field of the instruction word has been set. The "soft" interrupt is primarily used to stimulate the host to write a new Host Next Pointer to the interface board.

As shown in Figure 6.8, if an interrupt condition occurs, the SBus IntReq(7)* line is pulled low. At the same time, SPOLREG is generated to set the polling register of the Slave Register Bank. The IntReq(7)* line is kept asserted until the host reads the polling register. At that time, the interrupt line to the host is unasserted.

6.3.8 The Parity Generation Hardware

The Parity Generation hardware of the interface board is implemented in state machines programmed into an Altera MAX5032 chip. An edge detection FSM is used to ascertain when the PBus I/O Select line transitions from high to low. In accordance with the PBus protocol, the parity word should be transmitted at that time. Sixteen parity bit generation machines produce the serial parity bits which are transmitted over the PBus.

Figure 6.9 shows the bubble diagram for the edge detection FSM. Upon reset, the system enters the LOW state. It remains in this state until PIOSEL goes high. At that time, there is a transition to the HIGH state. The edge detection FSM remains in HIGH until PIOSEL goes low, when there is a transition to the TRANS state. In the TRANS state, the tri-state drivers on the parity generation chip are enabled by DRPTY so the current parity word is driven on the PBus. The system remains in TRANS until both EOFIN and EOFOUT are asserted. This indicates that the PBus is sampling the parity word. On the following clock cycle, there is a transition back to the LOW state.

The parity generation FSM is shown in Figure 6.10. This two state machine is replicated 16 times in the parity generation chip. A parity generation FSM is associated with each of the 16 PBus data lines. Upon reset, or one clock cycle after the successful transmission of the last parity word over the PBus (DRPTY_Q), the parity generation FSM goes to the default PAR1 state. In this state, the parity bit (PARBIT) is 1. The PBus data line is monitored for data transmissions. If there is a successful non-parity transmission over the PBus (/EOFIN*/(EOFOUT+DRPTY)) and a logic 1 is transmitted, there is a transition to the PAR0 state where the PARBIT is 0. The system stays in PAR0 until a logic 1 is successfully transmitted as data over the PBus. There is then a transition back to the PAR1 state. This procedure continues until the CAM data transfer is complete. The current state of the parity generation machines is then transmitted as the parity word, and all of the machines are automatically reset to the PAR1 state.

Figure 6.11 is a timing diagram for a typical write over the PBus. The PBus is always enabled in this example (EOFIN is never high). The Opcode, Dataout and parity tristate drivers of the PBus are never turned on in the same clock cycle that another driver is turning off. This eliminates contention in the system.

-bb-error = =

Figure 6.10: The Parity Generation FSM.

-bb-error = =

Figure 6.11: Time Line Diagram for a typical PBus write.

6.4 The SFSM of the CAM-8 Interface Board

This section describes the SFSM that is used on the interface board. The SFSM controller operates at the SBus clocking rate and performs the following functions:

- Fetches each Display List instruction from the host memory and loads it into the Control Registers for execution.
- Sequentially loads the Datain Registers with data from the host memory during a write to CAM.
- Sequentially drives the Dataout Registers over the SBus when they are filled by the PFSM during a read from CAM.
- Halts execution of the Display List instructions when required to wait on a specified event.
- Handshakes with the PFSM for the most efficient operation of the board as a whole.

The SFSM acts as a DVMA master whenever it interacts with the SBus. Because of the complexity of the SFSM, it was necessary to break this finite state machine into four separate finite state machines which all run in parallel. A Control FSM is responsible for interpreting the Display List instructions and setting the pointers that are used by the other FSMs. A Parameter Drive FSM drives the initial virtual address and the SBus Size(2:0) and Read lines at the required times. A CAM-Write FSM loads data from the SBus into the Datain registers during a write to CAM. Finally, a CAM-Read FSM drives data from the Dataout registers over the SBus during a read from CAM. State transition bubble diagrams for the operation of these finite state machines are shown in Figures 6.12, 6.13 and 6.14. The remainder of this section describes the operation of the SFSM in terms of these diagrams.

6.4.1 The Control FSM

As mentioned above, the Control FSM is responsible for interpreting the Display List program contained in the host memory and setting pointers to be used by the other finite state machines in the SFSM system. Figure 6.12 shows the 11 state implementation of the Control FSM.

The INIT State

On the assertion of the system reset, the Control FSM enters the INIT state. An initial instruction flag is set by the assertion of SINST, and the FSM goes to the WAITST.

The WAITST State

In the WAITST state, the Control FSM waits for the occurance of a specified event before continuing its interpretation of the Display List. Possible events include the assertion of the video HSYNC or VSYNC signal, the CAM-8 End-of-Scan interrupt or the host write signals. Once the event has been detected, there is a transition to either the INSTRLD state or the MAINPRC state.

An INSTRLD transition occurs if the initial instruction flag is set. This will generally only occur immediately after the system reset. SETWR is asserted to set the SBus Read line high, CINST clears the initial instruction flag, and 011 is loaded into the SCTR so that it points to the address of the next Display List instruction. In addition, the FST flag is set so that the instruction may be loaded by the CAM-Write FSM into the Control registers.

A MAINPRC transition occurs if the initial instruction flag is not set ie. when the WAITST was entered from MAINPRC. This indicates a required wait for a condition before proceeding to interpret the remainder of a previously fetched instruction. The CLRENWT signal is asserted to clear the enable wait flag so that the system does not re-enter the WAITST after it transitions to MAINPRC.

-bb-error = =

Figure 6.12: The Control FSM of the SFSM System.

The INSTRLD State

In the INSTRLD state, the Control FSM prepares to do an instruction load from the host memory. It requests the SBus and waits until it is granted mastership of the bus. When the Grant* line becomes unasserted, the Request* line is asserted by the Control FSM. Once the board is granted the SBus, there is a transition to the INSTRPRC state. The SCTR is loaded with 000 so that it points to the first Control register to be loaded from the SBus.

The INSTRPRC State

In the INSTRPRC State, the Control FSM waits until the CAM-Write FSM has loaded the instruction into the Control registers. If a Rerun acknowledgment is detected, then the board must retry the transfer. A transition is made to the RRNWT1 state in this case. If there is no Rerun acknowledgment, the ENWT flag is set in INSTRPRC. The ENWT flag allows the system to wait for a condition before proceeding to execute the remainder of the instruction that has been fetched.

A successful instruction load is indicated by the unassertion of Grant* without any Rerun acknowledgments. In this case, there is a transition to the MAINPRC state. The SCTR is loaded with 001 so that the virtual address of the host memory location to be accessed for a CAM read or write is available to be driven on the SBus by the Parameter Drive FSM.

The RRNWT1 State

In this state, the Control FSM simply waits for the SBus controller to unassert Grant* to indicate an end to the present bus cycle. Since a Rerun acknowledgment has occurred, 011 is reloaded into the SCTR so that the virtual address of the instruction to be read can be driven on the SBus by the Parameter Drive FSM. The system reenters INSTRLD when Grant* is unasserted.

The MAINPRC State

The MAINPRC state is the main interpretive state of the Control FSM. It is in this state that Control Register and PFSM flag data are interpreted to direct the operation of the Control FSM. If ERROR is detected[3],there is a transition to INIT so that the host can investigate the situation. ERRINT is asserted to inform the host of the error condition.

If the ENWT flag is set and WAIT is high, then there is a transition to WAITST so that the system can wait on the required condition before continuing to execute the instruction.

If the DONE line is high, then the instruction has been executed, and there is an transition back to INSTRLD so that a new instruction may be loaded. CLRDONE is asserted to clear the DONE line, SETWR is asserted to reset the SBus direction if it had been changed, and 011 is loaded into the SCTR so that the address of the next Display List instruction is driven by the Parameter Drive FSM. The FST flag is also set to allow an instruction load by the CAM-Write FSM.

If there is no bus grant and a non-immediate CAM write is to be executed, then Request* is asserted to obtain mastership of the SBus. If a CAM read is to occur and either DRD or FST is set, then Request* is also asserted. In this case, the SBus direction is changed to allow a write from the board to the host memory by the assertion of CLRWR.

If Grant* is detected and the WR flag is set, then the system has been given permission to do a write from the memory to the board. The Control FSM transitions to WRPRC1 to process the write. To increase bus bandwidth, all data writes to the interface board occur as atomic SBus transactions. Thus, the Control FSM continues to assert Request* when given this initial Grant*. The SCTR is loaded with 100 so that the first Datain register may be loaded from the SBus.

If Grant* is detected and the WR flag is not set, then a CAM read is required. The system transitions to the RDPRC state. The SCTR is loaded with 100 so that the first Dataout register may be driven on the SBus.

If none of the above conditions hold, the system transitions back to

[3]This will indicate either a Late Error or Error acknowledgment in a previous SBus cycle.

the MAINPRC state and waits for one of them to be met. This will usually be the case for a long immediate write to CAM-8, when the system waits in MAINPRC until the PFSM completes the transfer and sets the DONE flag.

The WRPRC1 State

In the WRPRC1 state, the Control FSM waits for the completion of the first burst CAM write of the atomic transaction. Request* is asserted in this state in accordance with the SBus requirement that this line be asserted until the Grant* for the second burst is detected.

If a Rerun acknowledgment is detected, there is an immediate transition to the RRNWT2 state, from which the transaction will be retried. If there is no Rerun acknowledgment, the system waits until Grant* is unasserted. There is then a transition to WRPRC2. The FST flag is cleared and the SCTR is loaded with the value 001 so that the starting virtual address of the next burst of data may be driven by the Parameter Drive FSM.

The WRPRC2 State

In the WRPRC2 state, the Control FSM waits until it receives the bus grant for the second cycle of the atomic transaction. While Grant* remains unasserted, Request* is asserted as required for the atomic protocol, and the Control FSM remains in the WRPRC2 state. Once Grant* is asserted, there is a transition to the WRPRC3 state. The Request* line is no longer asserted, and the SCTR is loaded with the value 100 so that the first Datain register is addressed.

The WRPRC3 State

In the WRPRC3 state, the Control FSM waits for the completion of the second burst CAM write of the atomic transaction. If a Rerun acknowledgment is detected, then there is a transition to RRNWT2. Otherwise, the system remains in WRPRC3 until Grant* is deasserted, at which point it transitions back to MAINPRC.

-bb-error = = □

Figure 6.13: The Parameter Drive and CAM-Write FSMs of the SFSM System.

The RDPRC State

In the RDPRC state, the Control FSM waits for the completion of a burst transfer from the Dataout registers to the host memory. If a Rerun acknowledgment is detected, there is a transition to RRNWT2 from which the transfer may be retried. Otherwise, the system remains in RDPRC until Grant* is deasserted, at which point it transitions back to MAINPRC. CLRFST is asserted to clear the FST flag.

The RRNWT2 State

In the RRNWT2 state, the Control FSM simply waits for the SBus controller to unassert Grant* to indicate an end to the present bus cycle. Since a Rerun acknowledgment has occurred during either a data read or write operation, 001 is reloaded into the SCTR so that the virtual address of the host memory address to be accessed can be driven on the SBus by the Parameter Drive FSM when the transfer is retried. Note that since DONE will not be set at this time, there is bound to be retry of the transaction when the system reenters the MAINPRC state.

6.4.2 The Parameter Drive FSM

As mentioned above, the Paramater Drive FSM is responsible for driving the virtual address at the beginning of an SBus transaction. It also asserts the correct values on the SBus Size(2:0) and Read lines throughout the transfer. As shown in Figure 6.13, this FSM is a simple two state machine.

The WAITBG State

In the WAITBG state, the Parameter Drive FSM awaits the assertion of the Grant* line by the SBus controller. Once this is detected, the FSM immediately asserts SDR_S to drive the required virtual address onto the SBus data lines. It also asserts DRPARAM to drive the Size(2:0) and Read lines. There is then a transition to the DRIVE state.

The Control FSM will have correctly set the value of the SCTR on a previous clock cycle so that when SDR_S is asserted as described above, the correct virtual address is placed on the SBus. In like fashion, the Control FSM would have set the value to be placed on the SBus Read line in a previous clock cycle.

The WAITBG state is the reset state for the Parameter Drive FSM.

The DRIVE State

In the DRIVE state, the Parameter Drive FSM simply awaits the deassertion of the Grant* line. While this line remains asserted, DRPARAM must be asserted so that the SBus Size(2:0) and Read lines are driven during the present bus cycle. As soon as Grant* becomes unasserted, DRPARAM is unasserted, and the system transitions back to the WAITBG state.

6.4.3 The CAM-Write FSM

The CAM-Write FSM loads burst data from the SBus into the Control or Datain registers on the Interface board. As shown Figure 6.13, it is implemented as a three state machine.

The IDLE State

In the IDLE state, the CAM-Write FSM waits for the presence of data to be loaded into either the Control or Datain registers. This is indicated if the WR flag is set and there is a bus grant and word acknowledgment. When these conditions are met, there are two possible operations of the CAM-Write FSM.

If the previous contents of Datain registers have been driven to CAM-8; if this is the first update of these registers; or if an instruction is being read, then the data is loaded from the SBus. Here either the DRW or FST flags are active, and there is a transition to CAMWR. SLD is asserted to load the first word of the transfer into the Datain or Control register. SINC increments the SCTR value so that the next

SBus word will be loaded in the following register location. Finally, SCWR is asserted to clear the DRW flag.

If the Datain registers are being updated and the previous contents of these registers have not been cleared by the PFSM, then the interface board must spill the data that is read from the SBus. In this case, neither FST nor DRW are active. The CAM-Write FSM transitions to SPILL to spill the arriving data.

The IDLE state is the reset state of the CAM-Write FSM.

The CAMWR State

The CAMWR state is used to load SBus data into either the Control or Datain registers. If a word acknowledgment is detected, the SLD and SINC signals are asserted to load the data into the presently addressed register and to increment the value of the SCTR pointer.

The CAM-Write FSM remains in the CAMWR state until either Grant* is deasserted or an error acknowledgment or Late error is detected. The deassertion of Grant* indicates the end of the burst transfer. The other two conditions imply that an error condition has been encountered. Whenever any of these conditions are met, there is a transition to the IDLE state. INCADR is asserted if the burst transfer has been successful to increment the Start Address of the data transfer by 16 bytes.

The SPILL State

The SPILL state is used to complete the required master obligations of the interface board when data is ready to be sent to it but the previous contents of the Datain Registers have not yet been sent over the PBus. The CAM-Write FSM merely waits in the SPILL state until Grant* has been deasserted. At that time, the host will have sent the desired data but it would not have been stored anywhere. The CAM-Write FSM transitions back to the IDLE state. Note that the Start Address has not been incremented. Hence the following SBus read will provide the same data. Hopefully, the next time the data arrives, the Datain Registers will be able to accept it.

-bb-error = = □

Figure 6.14: The CAM-Read FSM of the SFSM System.

6.4.4 The CAM-Read FSM

The CAM-Read FSM drives data from the Dataout registers on the interface board to the host memory over the SBus. Figure 6.14 shows that this FSM is implemented as a 6-state machine.

The START State

The START state is the reset state of the CAM-Read FSM. The machine waits in this state until it detects a bus grant with the write flag unasserted. This would indicate that a read from CAM-8 is required. When these conditions are met, and provided that the error flag is not set, a transition to the CAMRD1 state occurs. If the error flag is set or the above conditions are not satisfied, the system remains in the START state.

The CAMRD1 State

In the CAMRD1 state, SDR is asserted to drive the contents of the first Dataout register onto the SBus data lines. If an Error acknowledgment or Late Error is detected, there is a transition back to the START state. If a Rerun acknowledgment occurs, the system goes to the CAMRD5 state. Finally, if a word acknowledgment occurs, SINC is asserted to increment the SCTR, and there is a transition to the CAMRD2 state. If none of these hold, the system remains in the CAMRD1 state.

The CAMRD2 and CAMRD3 States

These states are very similar to the CAMRD1 state, except for the fact that there is no check for Rerun acknowledgment.[4] The CAMRD2 and CAMRD3 states drive the contents of the second and third Dataout registers onto the SBus.

[4]This is justified since the SBus specification indicates that a Rerun acknowledgment can only occur as the first acknowledgment of a burst transfer.

The CAMRD4 State

In the CAMRD4 state, the contents of the final Dataout register is driven onto the SBus. As above, if an Error acknowledgment or Late Error is detected, there is a transition back to START. In the CAMRD4 state, however, SDR is asserted only if a word acknowledgment is not detected. This is necessary to conform to the SBus protocol requirement that the SBus data lines be tristated on the cycle immediately following the final word acknowledgment.

When a word acknowledgment is detected in CAMRD4, there is a transition to the CAMRD5 state. INCADR is asserted to increment the Start Address of the data transfer by 16 bytes, and SCRD is asserted to clear the DRD flag.

The CAMRD5 State

In the CAMRD5 state, the CAM-Read FSM simply waits for the SBus Grant* line to be deasserted. This formally indicates the end of the bus cycle. The system then transitions back to the START state to await a new cycle.

6.4.5 Implementation of the SFSM

A practical implementation of the logic for the SFSM machine presented a challenging problem. A programmable logic implementation was preferred so that any bugs or changes to the interface board could be easily fixed. Initially, a single SFSM was designed to perform the functions of the four state machines described above. The single SFSM was programmed into the fastest available Xilinx programmable gate array[5]. Simulation of the chip, however, proved that it could not operate at the required SBus 20MHz rate because of the complexity of the logic and the high interconnect time of the Xilinx chip. Indeed, the worst case delays through the system was well over 60ns versus the maximum permissable delay of 50ns for a 20MHz system.

An implementation in the Altera MAX5064 programmable chip was also tried. This appeared to work, although the worst case de-

[5]The Xilinx 3090 100MHz chip was used.

lay through the controller was only a few nanoseconds short of the required operating period of 50ns. This did not leave much margin for the addition of the datapaths of the interface board.

At this point, the SFSM was partitioned into the four relatively independent systems described above. It was possible to enter the Control FSM in a single MAX5032 chip. The Parameter Drive, CAM-Write and CAM-Read FSMs could all be integrated into a second MAX5032 chip. The MAX5032 part, apart from being smaller than the MAX5064, is also a much faster chip. Simulation of the partitioned SFSM with the Altera MAX simulator proved that the required SBus rate of 20MHz could be readily achieved with the MAX5032 parts. The Altera Hardware Description Language (AHDL) text files for the four finite state machines described above are shown in Appendix C.

SIGNAL	TYPE	FUNCTION
sb_clk	Input	SBus clock line which serves as the clock for the SFSM.
rst	Input	Reset line for all FSMs in the SFSM system.
sb_bg	Input	Active low SBus Grant* line to the interface board.
ack(2:0)	Input	SBus acknowledge lines. Used to decode word, rerun and error acknowledgments.
wr	Input	Flag input from the instruction word. Indicates a write to or read from CAM.
done	Input	Indicates that the present instruction has been executed. This means that the data transfer specified by the instruction has been completed.
immed	Input	Flag input from the instruction word. Indicates an immediate write to CAM.
wait	Input	Flag input from the instruction word. Indicates that execution must halt until a given wait condition has been met.
cond	Input	Multiplexed condition input specified for an instruction wait.
drd	Input	Indicates that the Dataout registers have been loaded with new data.
drw	Input	Indicates that the Datain registers can be loaded with new data.
error	Input	Indicates that either an Error acknowledgment or Late Error has occurred.
instr	Flag	Indicates processing of first instruction after reset.
bsdir	Flag	Determines direction of SBus data transfer.
enwt	Flag	Enables halt of processing for a wait condition.
fst	Flag	Enables the loading of the Control registers. Also permits the Datain or Dataout registers to be initially loaded when DRW or DRD is not set.
ld	Output	Loads the SCTR counter with the values specified on sc(2:0).
sc(2:0)	Output	Value to be loaded in the SCTR counter. This counter points to the Control, Datain or Dataout register addressed by the SFSM.
clrdone	Output	Clears the DONE input to the SFSM.
br	Output	The SBus Request* line for the interface board in master mode.
errint	Output	Interrupt line to signal that an error has occurred on the interface board.
drparam	Output	Drives the SBus Size(2:0) and Read lines.
sinc	Output	Increments the SCTR counter.
sdr	Output	Drives the contents of the Dataout register addressed by the SCTR onto the SBus Data(31:0) lines.
sdr_s	Output	Drives the contents of the Control register addressed by the SCTR onto the SBus Data(31:0) lines.
sld	Output	Loads data on the SBus Data(31:0) lines into either the Control or Datain registers.
scwr	Output	Clears the DRW flag input to the SFSM.
scrd	Output	Clears the DRD flag input to the SFSM.
incadr	Output	Increments the virtual address of the start of a data transfer by 16 bytes.

Table 6.1: Signals of the SFSM System.

6.5 The PFSM of the CAM-8 Interface Board

This section describes the PFSM that is used on the interface board. The PFSM controller operates at the CAM-8 clocking rate and performs the following functions:

- Writes the Opcode Register of CAM-8 with the correct value.
- Sequentially drives the contents of the Datain Registers to CAM-8 over the PBus during a write to CAM.
- Sequentially loads the Dataout Registers from the PBus during a read from CAM.
- Decrements a length counter with each successful transfer over the PBus and sets a flag when the entire data transfer is complete.
- Handshakes with the SFSM for the most efficient operation of the board as a whole.

As in the case of the SFSM, the functionality of the PFSM was distributed over a few small finite state machine controllers. State transition diagrams of these FSMs are shown in Figures 6.15, 6.16, 6.17 and 6.18. A PControl FSM detects the arrival of a new instruction word and shifts the Opcode data over the PBus to CAM-8. If a CAM write is required, the PWrite FSM then takes over and transfers data from the Datain registers to CAM. If a CAM read is to be executed, a PRead FSM changes the PBus direction and then performs the transfer from the PBus to the Dataout registers. A PEdge FSM synchronizes the operation of the PFSM to that of the parity generation chip. This allows the PFSM to assert the EOFOUT line when the parity chip is ready to drive the parity word onto the PBus. The remainder of this section describes the operation of these finite state machines in terms of their state transition bubble diagrams.

-bb-error = =

Figure 6.15: The PControl FSM of the PFSM System.

6.5.1 The PControl FSM

The PControl FSM is shown in Figure 6.15. This FSM is responsible for the initial phase of the execution of a data transfer. All data transfers to or from CAM-8 are preceded by an Opcode register write. This specifies the CAM-8 register which is to be accessed when the data transfer begins. The PControl FSM loads the Opcode value contained in the Control registers into a P-clocked shift register. It then shifts this value over the PBus while correctly toggling the I/O Select line as required by the PBus protocol.

The PControl FSM also loads the transfer length counter from the Control registers into a PFSM controlled decrementable counter. In addition, it enables the DONE flag and resets the instruction ready flag.

The PINIT State

Upon reset, the PControl FSM enters the PINIT state. In this state, LD is asserted and PC is set to 1. This loads the value 011 into the PCTR on the next clock edge. This means that the PCTR selects the last Control register. Thus a high on PDAV would indicate that an instruction has been loaded into the Control registers by the SFSM. The PBus direction is also set so that the interface board can write to CAM-8. There is an unconditional transfer from the PINIT to the PIDLE state.

The PIDLE State

The PControl FSM waits in PIDLE until PDAV is asserted and the parity flag is clear. This indicates that the previous parity word has been transmitted to CAM and a instruction has been loaded into the Control registers. When this occurs, the PCTR is loaded with the value 100 so that the first Datain or Dataout register may be accessed

-bb-error = =

Figure 6.16: The PWrite FSM of the PFSM System.

by the other FSMs. LDCTL is asserted to load Opcode and transfer length data into P-controlled registers and to clear the instruction ready (PDAV) flag, enable the DONE flag and clear the Timeout counter. There is then a transition to the OPSHIFT state.

The OPSHIFT State

In the OPSHIFT state, the contents of the CAM-8 Opcode register is shifted out over the PBus. The system waits in OPSHIFT until DONE becomes asserted, EOFIN goes low or PSELDONE goes high. A low on EOFIN indicates that the PBus can accept valid data. While EOFIN is low and PSELDONE is unasserted, the contents of the Opcode register is shifted over the PBus by the assertion of PSH and EOFOUT. Once PSELDONE goes high, the shifting is complete, and the OK flag is set to initiate the other system FSMs. The PIOSEL line is set high by SIOSEL for the subsequent transmission of data over the PBus. The PControl FSM remains in OPSHIFT until the DONE line is set by the PRead or PWrite FSM. When this happens, PCWR is asserted to clear the Datain register data valid flags; the OK flag is cleared; the I/O Select line is set low as required by the PBus protocol; and the system transitions to the PINIT state.

6.5.2 The PWrite FSM

The PWrite FSM transfers data from the Datain registers to CAM-8 over the PBus. Its operation is started with the assertion of the OK flag by the PControl FSM. Data writes continue until the entire data transfer from the interface board to CAM-8 is complete. The PWrite FSM is shown in Figure 6.16.

The WRWAIT State

The WRWAIT state is the reset state of the PWrite FSM. The system remains in this state until it detects that the OK flag has been set and that a CAM write is required ie. that WR is active. Once these conditions are met, there is a transition to the PWR1 state.

The PWR1 State

In the PWR1 state, the PWrite FSM drives the high order 16 bits of available data from the presently addressed Datain register over the PBus by asserting PDRDATA. When data is available (PDAV); the transfer is not complete (/DONE); and the PBus can accept data (/EOFIN), EOFOUT is asserted. The transfer length is decremented by PLDEC, and there is a transition to the PWR2 state. If the transfer is complete (DONE asserted), then there is a transition back to the WRWAIT state. Otherwise, the system remains in PWR1 until one of these conditions are met.

The PWR2 State

In the PWR2 state, the PWrite FSM drives the low order 16 bits of available data from the presently addressed Datain register over the PBus. This is done by the assertion of PDRDATA and PDR2. When the data transfer is not complete (/DONE) and the PBus can accept data (/EOFIN), EOFOUT is asserted.[6] The transfer length is decremented by PLDEC, and there is a transition to the PWR1 state. If a non-immediate write is occurring, then PINC is asserted to increment the PCTR counter. In addition, PCWR is asserted if PRCO is active so that the Data Ready inputs from the Datain registers are cleared. These inputs will be enabled again when the SFSM loads the Datain registers with new data.

If the transfer is complete (DONE asserted), then there is a transition back to the WRWAIT state. Otherwise, the PWrite FSM remains in the PWR2 state.

---

[6]PDAV does not need to be checked since it was already checked on the transition into PWR2.

-bb-error = =

Figure 6.17: The PRead FSM of the PFSM System.

6.5.3 The PRead FSM

The PRead FSM is responsible for reading data from the PBus into the Dataout registers during a read from CAM-8. Like the PWrite FSM, operation of the PRead FSM is started with the assertion of OK by the PControl FSM. Data is read until the entire transfer from CAM-8 to the interface board is complete. Figure 6.17 shows the bubble diagram for the PRead FSM.

The RDWAIT State

The RDWAIT state is the reset state of the PRead FSM. The system remains in RDWAIT until the OK flag is detected and the WR input is not high. There is then a transition to the PDIR1 state.

The PDIR1 and PDIR2 States

These states are used to change the PBus direction from write to read. The bus direction was write when the Opcode value was written to CAM-8. A change of direction is therefore necessary in order to read the contents of the selected CAM-8 register. The PRead FSM remains in PDIR1 until the PBus can accept new data. It then sets the PBus RD/WR line to read and transitions to the PDIR2 state. The setting of the RD/WR line enables counting of the Timeout counter.

The system waits in PDIR2 for EOFIN to become unasserted as an indicator that the bus direction has changed to read. The explanation is as follows. If the bus direction is write and the EOFIN has become low,[7] then the same line cannot go back high again since EOFOUT is unasserted and no data is being sent over the PBus. When EOFIN does go high again, it must therefore mean that the PBus node adjacent to

[7] This represents the conditional transition into PDIR2.

the interface board has assumed the default empty state that occurs with a bus transition from write to read.

When EOFIN goes high, a transition is made to the PRD1 state in preparation for reading the required data from CAM-8.

The PRD1 and PRD2 States

The PRD1 and PRD2 states are used to read data from the PBus into the Dataout registers. Unless a timeout occurs, the system remains in these states until the entire read transfer is complete, at which time DONE becomes active. When this happens, there is a transition back to the RDWAIT state. If a timeout occurs before the first assertion of /EOFIN, then the PFSM is reset, and a host interrupt is activated. The timeout is necessary to prevent the system from hanging when an invalid module select ID has been sent to CAM.

When DONE is unasserted, the system remains in PRD1 until the PBus contains valid data and the Dataout registers are clear of valid data (/EOFIN*/PDRD). In this case, the PBus data is loaded into the high order 16 bits of the presently addressed Dataout register by the assertion of PLD1. The length counter is decremented with PLDEC and the PBus data is advanced by the assertion of EOFOUT. The system then enters the PRD2 state. Here the PRead FSM waits for more valid PBus data (/EOFIN). When this data becomes available, it is loaded into the low order 16 bits of the presently addressed Dataout register by the assertion of PLD2. The data transfer length counter is decremented by PLDEC, and the PCTR is incremented with PINC. EOFOUT is asserted to advance the PBus data. A transition is then made back to PRD1.

The loading of the final 16 bit Dataout register sets the DRD flag. This informs the SFSM that the Dataout buffer is full and that a burst write to the host memory should occur. Once this burst is complete, the SFSM clears the PDRD flag so that the PFSM can resume loading the Dataout registers with more data. The assertion of DONE immediately sets the DRD flag to the SFSM.

-bb-error = = □

Figure 6.18: The PEdge FSM of the PFSM System.

6.5.4 The PEdge FSM

The PEdge FSM is responsible for asserting the EOFOUT line of the interface board when the PBus can accept data and the Edge FSM of the parity generation chip is driving the parity word onto the PBus. In addition, this FSM sets a parity flag which prevents the PControl FSM from processing a new instruction before the parity word of the last PBus data transfer has been sent to CAM-8.

Figure 6.18 shows that the PEdge FSM is very similar to the Edge FSM of the parity hardware. Upon reset, the PEdge FSM enters the LOW state. The system remains in LOW until PIOSEL goes high. At this point, the parity flag is set and there is a transition to the HIGH state. The PEdge FSM remains in the high state until PIOSEL goes low again, when there is a transition to TRANS. The system waits in TRANS until EOFIN goes low. This indicates that the PBus can accept new data. When EOFIN goes low, EOFOUT is asserted low by the PEdge FSM. At this time, the parity hardware will be driving the parity word onto the PBus. The parity flag is cleared and there is a transition back to the LOW state.

6.5.5 Implementation of the PFSM

All of the finite state machines of the PFSM were integrated into a single Altera MAX5032 programmable chip. Simulation of this chip with the MAX System simulator showed that the required PFSM operation rate of 25MHz could be met with the implementation. The Altera Hardware Description Language (AHDL) text file for the PFSM is shown in Appendix C.

SIGNAL	TYPE	FUNCTION
pb_clk	Input	PBus clock line which serves as the clock for the PFSM.
rst	Input	Reset line for all FSMs in the PFSM system.
pdav	Input	Multiplexed data available signal to the PFSM. Indicates that the Control or Datain register referenced by the PCTR has valid data.
wr	Input	Flag input from the instruction word. Indicates a write to or read from CAM.
done	Input	Indicates that the present instruction has been executed. This means that the data transfer specified by the instruction has been completed.
immed	Input	Flag input from the instruction word. Indicates an immediate write to CAM.
pdrd	Input	Indicates that the Dataout registers contain valid data and may not be loaded.
pseldone	Input	Indicates that all relevant Opcode register bits have been sent to CAM.
prco	Input	Indicates that the PCTR contains the value 11. This means that the final Datain or Dataout register is addressed.
eofin	Input	EOF status input from the neighboring PBus node. This active low line also stops the timeout counter.
ptyflg	Flag	Ensures that the PFSM does not shift new Opcode data over the PBus until the parity word from the last data transfer has been transmitted to CAM.
ok	Flag	Indicates that the Opcode data has been shifted out over the PBus.
ldctl	Output	Loads the Opcode data from the Control registers into a PFSM controlled shift register. Also loads the transfer length into a PFSM decrementable counter. In addition, this signal clears the instruction ready flag, enables the DONE flag, and clears the timeout counter.
ld	Output	Loads the PCTR counter with the value [/pc,pc,pc].
pc	Output	Determines the value to be loaded into the PCTR counter. The PCTR points to the Control, Datain or Dataout register addressed by the PFSM.
pinc	Output	Increments the PCTR counter.
pcwr	Output	Clears the Data Ready inputs from the Datain registers. This effectively clears the PDAV line when the PCTR counter points to a Datain register.
piosel	Output	The I/O Select line of the PBus.
eofout	Output	The interface board EOF status output to its neighboring PBus node.
psh	Output	The shift enable line for the shift register which shifts the Opcode data out over the PBus data lines.
pdrdata	Output	Drives the contents of the presently addressed Datain register as a 16 bit word on the PBus.
pdr2	Output	Selects between the high and low order 16 bits of the currently addressed Datain register.
pld1	Output	Loads the high order 16 bits of the currently addressed Dataout register with PBus data.
pld2	Output	Loads the low order 16 bits of the currently addressed Dataout register with PBus data.
pldec	Output	Decrements the length of data transfer counter with each successful PBus data word transfer.
pbusdir	Output	The RD/WR line of the PBus. This determines the direction of data transfer over the PBus. Enables the timeout counter in read mode.

Table 6.2: Signals of the PFSM System.

-bb-error = =

Figure 6.19: Datapaths for the interface board slave hardware.

6.6 The Slave Hardware of the Interface Board

The SFSM and PFSM described earlier are used to control the interface board when it acts as a DVMA master on the SBus. In order to function effectively in the CAM-8 input-output system, however, the board must also be able to act as an SBus slave. In slave mode, the pointer to the next Display List item may be computed by the host and written to the interface board. Thus the Wait-for-Host condition can be satisfied. When an error acknowledgment occurs, the host needs to be able to read from the interface board the address of the instruction which caused the error. This also requires the board to function as an SBus slave.

The slave operation of the interface board is completely independent of its master operation. Slave response is required once the board is correctly selected as a slave by the SBus Controller. This can occur when the board is no longer an SBus master but is internally processing a transfer to or from CAM-8. To prevent any conflicts that may arise due to simultaneous slave and master processing, the slave controller and datapaths are designed to be relatively independent of the master hardware. This section describes the hardware and finite state machine controller needed to allow the interface board to function as an SBus slave.

6.6.1 The Slave Datapath Hardware

The datapath hardware for the slave system is shown in Figure 6.19. An address decoder selects a Slave Register Bank location for host access based on the SBus physical address lines PA(27:0). The SVLD and SVDR signals generated by the slave FSM are used to activate the correct register load and drive signals in the register bank.

The host needs to write only to the single 32-bit Host Next Pointer register in the Slave Register Bank. This register stores the virtual address of the Display List instruction to be executed when the next Wait-for-Host condition is encountered during normal Display List execution. The loading of the Host Next Pointer register sets the HOST-FLG, which is the Wait-for-Host condition flag. The register may be driven during slave operation; or by the master when the HOSTFLG is set, there is a Wait-for-Host condition, and the Next Display List address is being driven on the bus.[8] In the latter case, the Control Register at location 011 is inhibited from driving the SBus.

The Slave Register address location zero is reserved for the hard-wired board ID provided by Sun. Subsequent address locations may also be used for other required tokens of an SBus slave. The Slave Register Bank also contains two Status registers, which are loaded by the board as a master through the activation of the LDPTR signal whenever the Next Display List address is driven on the SBus. Status register #2 is loaded from Status Register #1, and Status Register #1 is loaded from the SBus at that time. There is no conflict between master and slave operation in this case since the board is explicitly an SBus master in possession of the bus when it drives the Next Display List address. Thus it cannot be a slave unless the driven address is in the interface board slave space. The Status Registers keep track of the present Display List instruction that is being executed as well as the previous instruction. Their contents may be driven on the SBus when the host attempts to resolve an error condition.

The Slave Register Bank contains one other register — the interrupt polling register. This register is set by the Interrupt Hardware whenever the board activates an SBus interrupt.[9] It is read by the host when the latter is attempting to determine the source of the interrupt. As explained in the section describing the Interrupt Hardware, the reading of the polling register resets the interrupt line.

[8]Under these conditions the DRHNP line becomes active.

[9]The SPOLREG signal sets the polling register.

-bb-error = = □

Figure 6.20: The Slave FSM of the interface board.

-bb-error = = □

Figure 6.21: Generation of the ON input for the Slave FSM.

6.6.2 The Slave FSM Controller

The three state finite state machine shown in Figure 6.20 is used to control the slave operation of the interface board. Upon reset, the slave FSM enters the SVIDLE state. When the board is selected as an SBus slave, the ON signal is activated, and there is a transition to the SV1 state. Figure 6.21 shows that ON is generated if the Size(2:0) lines indicate a word transfer and there is a high to low transition in the logic OR of AS* and Sel*.

In SV1 the Ack(2)* line is asserted to indicate a word acknowledgment. If the CPU is writing to the board (/RD), SVLD is asserted to load the data into the selected register. There is an unconditional transition from SV1 to the SV2 state.

In state SV2, the Ack(2)* line is driven high as required by the SBus protocol. Since the interface board as a slave issues only idle and word acknowledgments, only the Ack(2)* line needs to be driven by the board. Error acknowledgments are not issued. In the case of an error due to invalid transfer size, ON does not become asserted and there is a bus timeout. In state SV2, SVDR is asserted if the slave is being read by the CPU. There is always a jump back to the IDLE state from SV2.

The finite state machine of Figure 6.20 is implemented in the following manner. The SVIDLE, SV1 and SV2 states are assigned the state bits 00, 01 and 11 respectively. This leads to the following equations for the control signals:

$$S0 := (/S1*S0 + /S1*ON)*/RST$$

$$S1 := /S1*S0*/RST$$

$$ACKDR = S0$$

$$ACK(2)* = S1$$

SVDR = RD*S1

SVLD = /RD*S0*/S1

A hardware implementation based on these equations will be programmed into an Altera MAX5032 chip to realize the Slave FSM of the interface board.

Chapter 7

Implementation of the CAM-8 Input-Output System

The previous chapters of this thesis have dealt with the design of the CAM-8 input-output system. Several factors have prevented a direct implementation of the design from being realized at the present time. In this chapter we explain the implementation status of each part of the system, what problems have been encountered, and any tests and simulations which have been completed. Future system level simulations are also discussed.

7.1 The Input-Output Section of the CAM-8 Controller Chip

The input-output section of the controller chip described in Chapter 4 was schematically entered using the VLSI Technology ASIC design software. Appendix A shows the schematics of the various blocks described in Chapter 4. The circuitry was developed using the VLSI Technology 1.0μ CMOS gate array library primitives. Design entry of the input-output section of the controller chip was completed long before circuitry for the other blocks of the chip had been developed. As a result, it was impossible to predict all of the registers which would be required for the final design. An eight register input-output block was therefore developed for testing purposes.

Each sub-block of the controller chip I/O system was simulated using the VLSI gate array simulator to ensure that it met the system requirements. Once this was completed, the blocks were wired up together and the entire eight register test version of the input-output block of the controller chip was simulated. Test code and output for these simulations are shown in Appendix B. The results were encouraging. The eight register test version of the circuit appeared to meet the goals of its design. When the control signals to the input-output section of the chip were toggled as they would in the final chip, the registers of the test version were correctly read and written. In addition, timing analysis revealed that the test version of the chip correctly operated at clocking rates above 25MHz, as required for the CAM-8 system.

It was hoped that the design of the controller chip would have been completed by December 1989. However, by January 1990, only the preliminary eight register test version of the I/O block described above had been entered into the system. Work on the remainder of the chip is only now progressing. It was therefore impossible to obtain a fabricated chip to test at the time of writing. It is hoped that the controller chip would be fabricated in the coming months. Only then can the I/O circuitry shown in the schematics of Appendix A be truly tested.

7.2 The Pipelined Input-Output Bus

The pipelined input-output bus (PBus) described in Chapter 3 has not been physically built. However, both the controller chip and interface board circuitry have been designed to be compatible with the PBus protocol. The initial CAM-8 hardware will probably consist of only two CAM-8 space modules. The PBus for such a system will take the form of a series of connecting wires — the depth of the bus will be zero. Indeed, it will probably be several years before a CAM-8 machine of sufficient size is developed to require a PBus with any real tree-like structure. The physical implementation of the PBus was therefore given low priority in the project. This is not to say that the PBus protocol would not be tested. Since PBus compatible circuitry has been built into the controller chip and interface board, even the initial CAM-8 machine will exercise the design of the PBus protocol. The circuitry contained in a PBus node is relatively straightforward, and should not present a major implementation problem when it is required.

7.3 The Interface Board

The design and implementation of the interface board proved to be the most challenging part of the input-ouput system. The problem involved interfacing the system to the high speed (20MHz) SBus of the Sun Sparcstation. Economic and practical considerations dictated that the interface board be implemented with some combination of programmable logic devices. After all, only one interface board is needed for a CAM-8 machine of any size. Also, especially in the initial phase of development, it was advantageous to have an implementation which could easily be modified should the need arise.

Most of the programmable logic devices, however, proved to be too slow in their implementation of the complex logic required by the SFSM of the interface board. Initially, the SFSM was designed as a single eight state machine. This machine appeared to be logically correct and efficient. Boolean equations for the eight state SFSM were entered using the Xilinx Programmable Gate Array logic entry system. The design was compiled, optimized, placed and routed on the fastest Xilinx 3090 programmable gate array available.[1] Timing analysis of the routed chip indicated that even without pad connections or the remaining system datapaths, the logic could operate at best only at about 15MHz. This was unacceptable for the interface board.

The SFSM was restructured in many ways in attempts to speed up the circuit. The architecture of the Xilinx 3090 programmable gate array was studied, and the state realizations and equations were modified specifically for this architecture. Repeated attempts succeeded in reducing the worst case delays by several nanoseconds at best. The problem was the complexity of logic — which could only be reduced at the loss of interface board functionality. Each level of logic in the Xilinx realization required a fixed delay time through the chip. Routing delays

[1] The 100MHz toggle rate chip was used.

were also unpredictable and excessively high, sometimes of the order of 15 to 25 nanoseconds. After about a month of frustrating trials, it was concluded that the Xilinx chip was inadequate for the purposes of the interface board.

Other programmable logic families were investigated. Discussions were held with representatives of Actel and Altera Corporations. The Actel engineers were skeptical that a 20MHz SFSM could be implemented even with their fastest chip. During this period, plans were made to attempt to enter the control circuitry for the interface board into a Mosis 2.0μ standard cell, which could be fabricated in several weeks at a relatively low cost. The SFSM and PFSM were entered into the VLSI standard cell system using the VLSI tools state machine entry language. Some simulation of the state machines were done. However, an extensive timing analysis was not completed.

By this time, the Altera MAX programmable entry software was acquired on a trial basis. The SFSM controller was entered into the system using the Altera Hardware Description Language (AHDL) and the design was compiled into the Altera MAX5064 chip. Simulation of the design using the MAX simulator proved that the SFSM timing was only just met. This was encouraging, but still unsatisfactory, since datapath delays would need to be included in the system timing. The SFSM was restructured into its present implementation so that it could be compiled into two smaller and faster Altera MAX5032 chips.

Simulation of this design on the MAX simulator was successful. The SFSM operating rate of 20MHz could be comfortably realized with the fastest grade MAX5032-1 parts. Test vectors were written to exercise the possible situations with which the SFSM would be required to cope. The SFSM seemed to operate correctly in all of the test cases.

The PFSM was structured in a similar manner and compiled into a single Altera MAX5032-1 chip. Simulations of the PFSM under worst case conditions proved that it could correctly operate at the required 25MHz CAM-8 clock rate. The AHDL text description files for the SFSM and PFSM are shown in Appendix C. Compiled pinouts for the Altera chips are in Appendix D and test vectors and simulation output are shown in Appendix E.

The parity generation hardware was also targeted into an Altera MAX5032-1 chip. It was compiled and tested using the MAX software.

Appendix C shows the AHDL text file for this chip. Simulation output for the parity chip can be found in Appendix E. The parity chip simulated at the PBus clock rate and appeared to behave in the correct manner.

The problems of the interface board implementation were not over, however. Because the board uses SBus word transfers, 32-bit wide registers were required for the Control, Datain, Dataout and slave registers. A physical realization of these registers presented a major problem. Discrete logic components could be used, but they would require a large board area and numerous interconnecting traces. This would easily exceed the maximum SBus double width board area (6.6in by 5.58in) and capacitive loading (20pF per signal). A programmable logic implementation was tried. However, the large number of registers meant that the realization would require several programmable chips. There was also the problem of driving the SBus lines to meet the SBus setup time of 15ns. At the present time, other alternatives for the interface board register banks are being investigated, including possibly a specialized Mosis chip containing all of the registers. The Mosis chip is a real possibility since it would be compact and there is a low risk of failure since relatively simple logic would be used on the chip.

It is thought that once the register bank problem is overcome, a prototype of the interface board should be easy to produce — the slave FSM and remaining glue logic could be programmed into another Altera chip. Several Altera MAX5032-1 chips have been special ordered[2] and are expected soon.

It is the author's belief that the many problems encountered in the implementation of the interface board is a direct result of trying to use a technology unsuited to the target device. The interface board contains logic that is almost as complicated as that of the CAM-8 controller chip itself. The stringent requirements which must be met by SBus expansion devices only serve to complicate the practical realization of this logic. According to *The SBus Specification*, the SBus is "intended for use as a chip-level interconnect between components." The specification also states "From an electrical and performance perspective, the setup, hold and delay requirements of the bus are compatible with

[2] These chips are not presently available commercially.

modern CMOS families, such as FCT, and the current and upcoming generations of CMOS gate arrays (such as LSI Logic 9K and 100K families)."

Virtually every board in the Sparcstation which interfaces to the SBus contains some sort of high performance CMOS gate array controller which performs the interfacing. Clearly the CAM-8 interface board should really be implemented in a high performance gate array as well. Such a realization will easily meet SBus card size, timing and electrical requirements as dictated by *The SBus Specification*. Unfortunately, the CAM-8 project has reserved gate array development for the controller chips rather than the interface board. This is quite reasonable given the practical and economic considerations. The author's point is simply that there is a better and more efficient way to implement the CAM-8 interface board which could overcome many of the problems that have plagued the current realization. It is hoped that once the preliminary programmable interface board has been fully debugged and tested, its logic can be integrated onto a gate array in later months.

7.4 System Level Simulations

Once the design of the CAM-8 controller chip is completed, a thorough system level software simulation of the entire CAM-8 machine will be conducted. This simulation will be performed with the Verilog simulator. A behavioral model of the SBus has already been acquired from Sun Microsystems for this simulation. Behavioral models for the interface board, the PBus and the controller chip are currently being developed.

Several simple CAM-8 experiments will be executed on the system level simulator. The performance of all of the CAM elements, including the input-output system, will be carefully monitored. The Verilog simulations should uncover any unforeseen bugs in the machine. Necessary modifications to the interface board or controller chip will be made before the chip is fabricated. Fabrication of the controller chip and final modifications to the interface board hardware will take place only after the CAM-8 system has performed satisfactorily under the

Verilog simulations.

Chapter 8

Conclusion

An input-output system for connecting the CAM-8 cellular automata machine to a host Sun Sparcstation has been designed. This system provides for the downloading of data and control information from the Sparcstation memory to CAM-8. It also allows for the transfer of CAM-8 status information in the opposite direction. The input-output system is still in the process of being thoroughly simulated and built.

Some of the input-output circuitry of the controller chip described in this document may be amended as certain features of the CAM-8 I/O are dropped or added. It is hoped that a preliminary version of the controller chip will be fabricated by August 1990. At that time, a simple two module CAM-8 machine will be built and tested. Real cellular automata experiments will be run on the system to determine whether the goals of the initial design have indeed been met.

A special version of the interface board described in this thesis will be wired up to exercise the SBus protocol and to gather certain statistical information on the operation of the SBus. This should aid in any last minute modifications to the interface board hardware. It will also help to direct the development of software for the CAM-8 system.

An attempt will be made to try to integrate the logic for the interface board on the controller chip itself, in parallel with its programmable logic implementation. Selector pins on the chip will allow it to function as a CAM controller chip; an interface board controller chip; an interface board control registers chip; or an interface board data registers chip. Successful implementation of this logic on the gate array will easily meet all SBus electrical, timing and size requirements for the interface board. The reader may recall that this was one of the major problems with implementing the interface board in programmable logic. The author is presently conducting active work in this area.

Printed circuit boards to house the entire CAM-8 system still have to be fabricated. The high speeds of system operation mean that simple wire wrapped test boards will probably not function properly. Thus printed circuit technology is needed even for the preliminary test versions.

Once a two module CAM-8 machine has been built and tested, larger machines will be developed. However, the input-output system should remain the same for these bigger machines. As the CAMs grow in size, the depth of the PBus will grow accordingly. This would require that PBus nodes be physically built into the system. Future CAMs may include second or third generation versions of the CAM-8 controller chip with added features. It is hoped that the CAM-8 machine can one day be industrially manufactured and distributed to scientists around the world. As a scientific tool, CAM-8 will allow physicists to run cellular automata experiments to model very large universes of cells. Such experiments can ultimately lead to a better understanding of many statistical phenomena of physics and matter.

It is the author's hope that this thesis forms an integral part of the final working version of the CAM-8 cellular automata machine and be useful in research on other machines with similar architectures as well.