VLSI IMPLEMENTATION OF A CONFIGURABLE MULTIPROCESSOR SYSTEM FOR DSP BEHAVIORAL SIMULATION

by

Alfred K. W. Yeung

Memorandum No. UCB/ERL M90/15

27 February 1990

VLSI IMPLEMENTATION OF A CONFIGURABLE MULTIPROCESSOR SYSTEM FOR DSP BEHAVIORAL SIMULATION

by

Alfred K. W. Yeung

Memorandum No. UCB/ERL M90/15

27 February 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A Perspective

With the advent of digital computers, Digital Signal Processing (DSP) has become a dominant force in the fields of signal processing and communication. Examples of such applications include digital audio, speech synthesis and recognition, telecommunication, image and video processing and robotics. As the complexity of the algorithms increases, the task of verifying and optimizing them becomes formidable. The process often requires high computation throughput and simulation of a large amount of data. For example a computation rate of 800 MOPS or more is typical for High Definition Television (HDTV) algorithms. Furthermore, to verify the behavior of the algorithms, many frames of data have to be simulated. These requirements dictate a hardware solution.

While techniques such as bread-boarding and fast-prototyping can fulfill the requirements, they typically exhibit long development time and offer very little programmability which is important in optimizing the parameters of some algorithms. Some commercial multiprocessor computers are capable of providing high computation power but the high overhead in inter-processor communication, difficulty in mapping the algorithms to the architecture, lack of instructions for supporting DSP applications and the usual high cost of the machines often limit the effectiveness of these machines.

In this report, a dedicated compute-engine called *SMART* (an acronym for Switchable Multiprocessor Architecture supporting Real Time applications) is presented. The machine attempts to speedup simulation of DSP algorithms by at least two orders of magnitude as compared to general purpose computer architectures. The DSP32C from

AT&T Bell Labs, a high performance DSP processor with both floating point and fixed point instructions is used as the core processing unit to provide high computation power, resulting in an order of magnitude in speedup. An additional order of magnitude in speedup is obtained by exploiting the high degree of concurrency, namely *pipelining and parallelism*, present in most signal processing algorithms.

1.2 Organization of Report

This report introduces the SMART architecture using a top-down approach, with the first few chapters presenting an overview of the system architecture and the later chapters explaining the design details and tradeoffs. The emphasis of the report is on the hardware implementation of the VLSI chip set which handles memory accesses and interprocessor communications of the system.

Chapter 2 gives a brief description of the SMART architecture and its special features. Some benchmark results are presented to justify the architecture. Chapter 3 provides a block diagram view of the proposed prototype system, its components, their functions and their interconnections. Design considerations and approaches which ultimately guide the system to its present shape are discussed in chapter 4. Chapter 5 and 6 details the functionalities and designs of the VLSI custom chip set implemented for the system. The next chapter describes the circuit and logic design issues of some macro blocks implemented for the chip set. Chapter 8 discusses the verification and test strategy adopted. Chapter 9 concludes by summarizing some after thoughts about the project. Some future developments on SMART is also outlined.

Chapter 2

SMART Architecture

In this chapter, we will describe the four main features of the SMART architecture: the *Configurable Bus*, the *Distributed Shared Memory*, the *Write Queues* and the *Synchronization Mechanism*. These features, implemented in special custom circuits, are designed to reduce the communication and synchronization overheads significantly.

More details on the architecture can be found in [4] and [5].

2.1 Configurable Bus

In typical DSP algorithms, both types of concurrency, pipelining and parallelism exist. An example is the pitch extractor algorithm shown in Figure 2.1 [3]. The blocks represent the operations to be performed on the data stream and the numbers underneath the blocks indicate the computation load of the corresponding block. Clearly the throughput of the system can be improved by pipelining the entire system between blocks. However, the computation intensive template matching operation becomes a bottleneck. To further enhance the performance a cluster of processors can be allocated to share the computation task in the block, resulting in the data dependence graph in Figure 2.2(a). A natural implementation is to assign one processor for each node and use queues between processors as data buffers. The resulting multi-processor architecture with customized communication pattern is shown in Figure 2.2(b).

Unfortunately, DSP algorithms usually exhibit a large variety of communication patterns and the use of customized multi-processor machines as a simulation engine is grossly inefficient. SMART overcomes this problem by providing a switchable bus that can

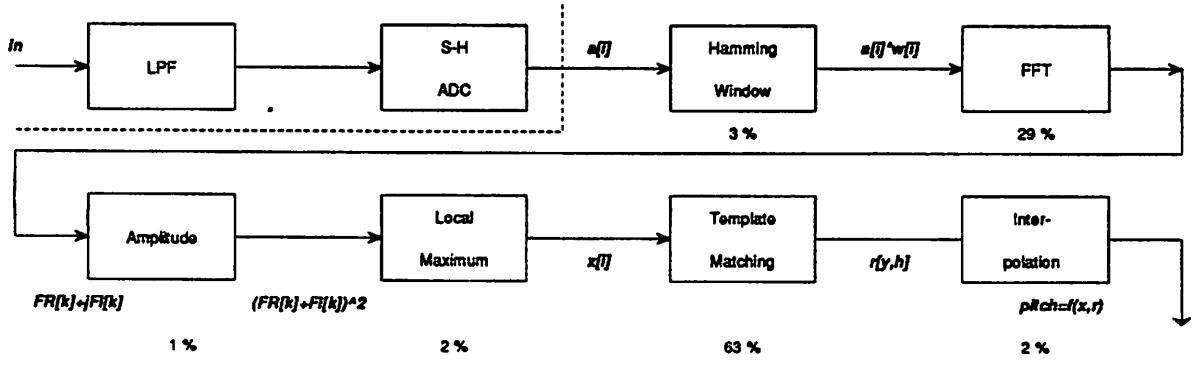


Figure 2.1: Pitch Extractor

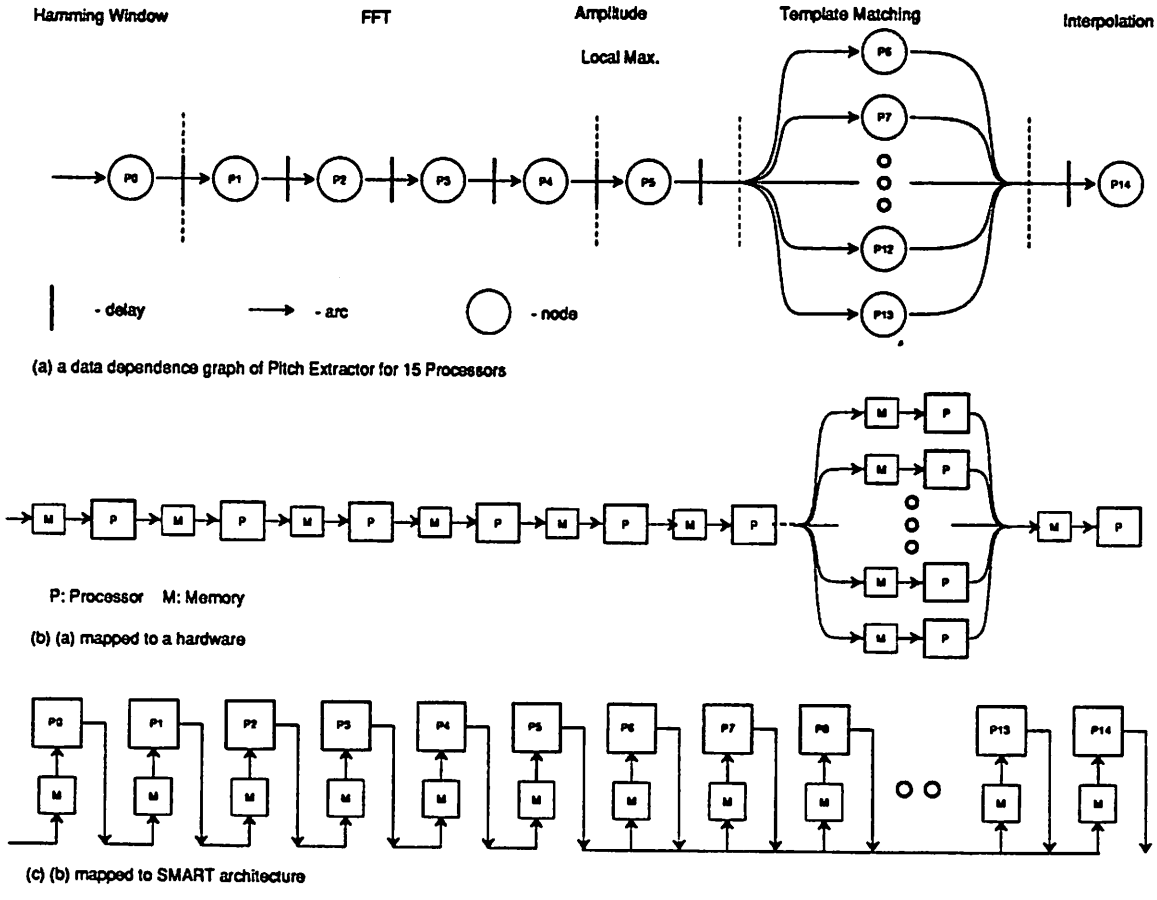


Figure 2.2: Bus Configuration of the Pitch Extractor Algorithm.

be configured by software to mimic the different communication patterns of a wide range of algorithms without too much sacrifice in performance.

The SMART system consists of an array of processing units connected in a linear fashion by a single shared bus. The key feature is that there are *switches* ((S) in Figure 2.3) between all neighboring processors which can be opened or closed to divide the processors into groups. Processors with local communications are put into one group so that they can communicate among themselves independent of activities in other groups, thus boosting the overall communication bandwidth of the bus. In general processers working in parallel are grouped together and processors operating in pipelined fashion are put in different groups. Figure 2.2(c) shows how the bus can be configured to simulate the pitch extractor algorithm.

Sometimes data has to be forwarded to a processor several processor groups away, for example, from processor P2 to processor P8 in Figure 2.2(c). The solution is a hardware supported *bypass unit* ((B) in Figure 2.3) which allows data to go through an open switch without program intervention but at the cost of extra latency. The bypass unit implements global communication by allowing a processor to access the memory of any processor which may be in a different bus group[1]. Thus programs can be written independently of the bus configuration. From the programmer's point of view, the configurable bus is still a single shared bus supporting shared memory. However transfer of data through bypass units incurs higher communication latency.

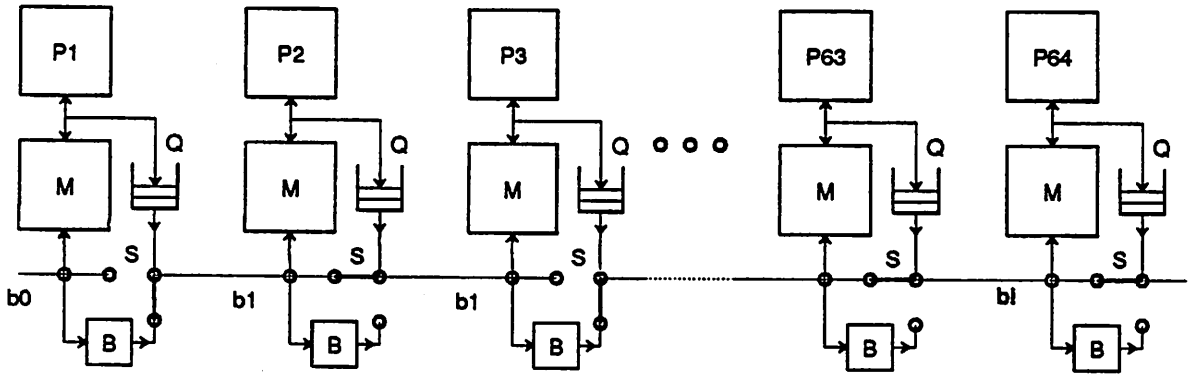
In addition, a *ring connection* is formed by connecting one end of the processor array to the other end to reduce the maximal distance between any two processors in the array.

The configurable bus can simulate a single-shared bus (by closing all switches), a one-dimensional systolic array (by opening all switches) or an irregular communication pattern which is specific to the application algorithm, as shown in Figure 2.4. Optimal bus configuration can be obtained by trading off communication latency and overall bus bandwidth.

2.2 Distributed Shared Memory and Write Queues

The memory architecture is as important as the interconnection scheme in reducing the overhead of interprocessor communication. In traditional shared memory multiprocessor

[1] Global read access is not supported in the first prototype system.



Pi: Processor, M: Memory, S: Switch

bi: Bus, B: ByPass, Q: Write Queue

Figure 2.3: SMART Processor Array with Bypass Units.

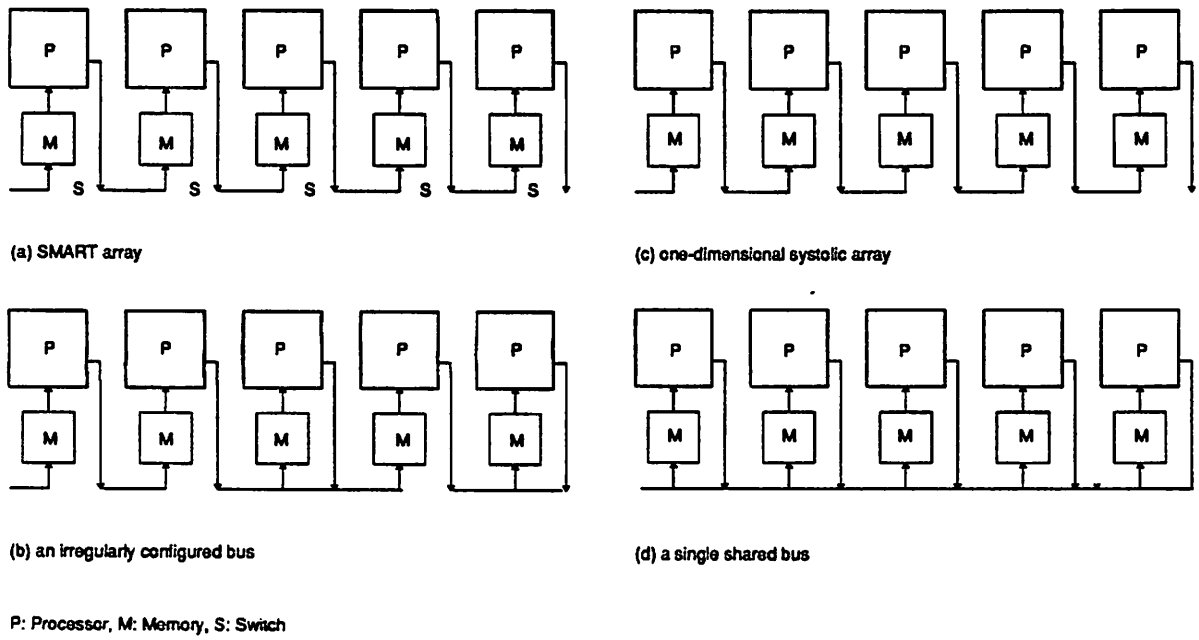


Figure 2.4: Examples of Bus Configuration.

machines where processors are connected to a central memory by some shared connections, e.g. a shared bus, there are heavy penalities in accessing shared data in the memory due to bus contention. The SMART Architecture provides two features, namely the dual-ported distributed shared memory and write queues ((M) and (Q) respectively in Figure 2.3), to alleviate this problem.

In SMART, the large global shared memory is partitioned into small memory units among processors. Each memory unit has dual ports, one of which is connected to its corresponding processor while the other is connected to the shared configurable bus (Figure 2.3). The basic scheme of communicating data between two processors is that of a *source-write and destination-read* scheme, that is, the processor which produces the source data writes the data to the local memory unit of the destination processor through the shared bus. Then the destination processor can read the data via its private port. Using this scheme and a distributed memory system, contention for the shared bus can be relieved by reducing the number of accesses to the shared bus roughly by half.

In case of multiple requests to use the shared bus, some requests will be *stalled*, resulting in communication overhead. SMART provides a write queue for all the write operations so that the processor can immediately resume its computation as soon as data has been written to the queue. By overlapping the computation time with the overhead time due bus arbitration, the effective communication overhead is reduced.

2.3 Synchronization

The synchronization mechanism is an important factor in the performance of a multiprocessor system. There are two frequently found synchronization mechanisms: *mutual exclusive synchronization* and *barrier synchronization*. When several processes try to access a shared resource, the mutual exclusive synchronization guarantees that only one process will get the access right while the others must wait until the resource is available. On the other hand, barrier synchronization is useful to ensure the order of execution by synchronizing all the processes at a certain point of the program. In a typical application, data coherency is maintained by requiring both the process that produces the data and the process that consumes the data to issue a barrier synchronization instruction before proceeding. In most of the applications we have studied, barrier synchronization seems to be the most effective technique.

Implementing synchronization mechanism using software is inherently slow. System performance can be seriously degraded especially in applications where the processors are *tightly coupled* and synchronization between processors is often needed. In SMART, low overhead for barrier synchronization is achieved by providing special hardware supported synchronization instructions. Two semaphore operations are also provided for the mutual exclusive synchronization.

2.4 Benchmark

To show that SMART is powerful enough to simulate a large variety of algorithms, we chose four very different and yet common signal processing algorithms for benchmarking: Fast Fourier Transform, Echo Canceller, Pitch Extractor and Matrix-Vector Multiplication. These algorithms possess different computation complexity, communication patterns and degrees of concurrency and granularity.

To optimize the performance, we used four different bus configurations. For example we used a one-dimensional systolic array to implement the FFT algorithm and a single shared bus to perform the matrix-vector multiplication. This indicates that the bus configuration indeed has a great impact on performance.

The benchmark results show close to ideal speedup and very low communication overhead in all cases (Table 2.1). Speedup is defined as the relative amount of time we saved compared to running the same algorithm on a uniprocessor machine. Communication overhead gives the extra amount of time used in communication compared to an ideal multiprocessor machine in which there is no contention or penalty in accessing any portions of the global memory. The low communication overhead in all test cases confirms the ability of SMART to adapt to different communication patterns. The idle time is a measure of how well the load is partitioned among processors and has a direct effect on speedup. The other figures are included to indicate the nature of the algorithms. The average bus usage simply tells us whether the algorithm is computation intensive or communication intensive. The average number of requests is sort of a time domain measure of the communication pattern. It indicates whether communications are clustered or spread out in time.

These results were obtained with the bus configuration fixed at all time. In some algorithms, we found that we can push the performance further by allowing the bus configuration to change in the middle of the program to take advantage of the changing commu-

	256-pts FFT	Echo Canceller	Pitch Extr.	Matrix-Vector Mult.
Speedup	13.65	14.04	13.29	14.58
Communication Overhead (%)	0.64	0.13	0.03	1.11
Idle Time (%)	8.91	18.52	16.12	1.80
# of Buses	16	3	4	1
Average Bus Usage (%)	32.67	0.65	4.05	14.33
Average # of Bus Requests	1.00	1.59	1.50	8.62

Table 2.1: Benchmark Results for 16 Processors.

nication patterns.

Chapter 3

SMART Prototype System

3.1 System Overview

The major components of the system are the SMART board array, the CPU Board, the Analog/Digital Unit, and the host system, as depicted in Figure 3.1. Computation is performed by the SMART board array which consists of a set of programmable core processors (AT&T's DSP32C) connected to each other via a configurable shared bus. The CPU Board, a single-board micro-computer (HKV2F by Heurikon) running a real-time Unix-like Operating System called *VxWork, Version 4.3*, serves as the *master* of the VME bus[9]. Data can be supplied to and received from the processor array in real-time through a data acquisition board consisting of A/D, D/A and two TMS320 signal processors. The host is an enginerring workstation that provides a UNIX environment for cross-compiling and developing application programs and also serves as a large auxiliary data storage unit. Communication between the host computer and the CPU board is established by the Ethernet which also allows the CPU board to access data files in the file server of the host system directly. In addition to the VME bus an optional custom bus can be implemented to speed up communication between the components. All the components except the host are housed inside a VME cardcage.

3.2 SMART Processor Board

A simplified block diagram of the SMART Processor Board is shown in Figure 3.2. Each SMART board has an array of four processing units[1], input and output first-in-first-out

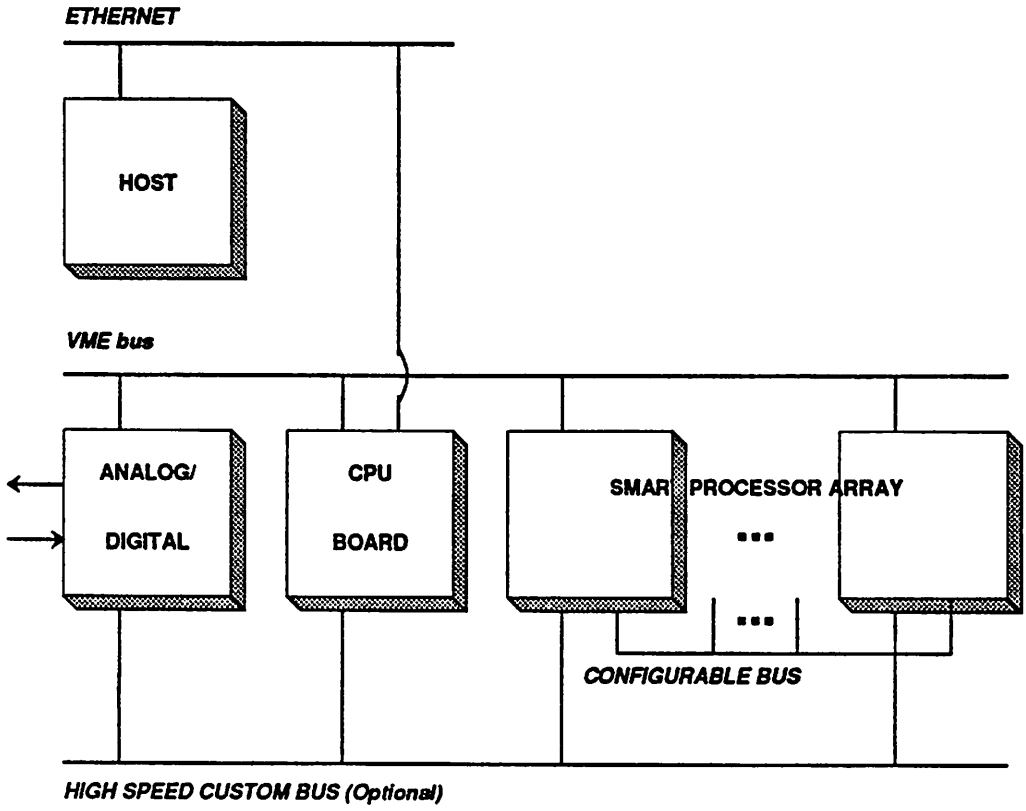


Figure 3.1: SMART System.

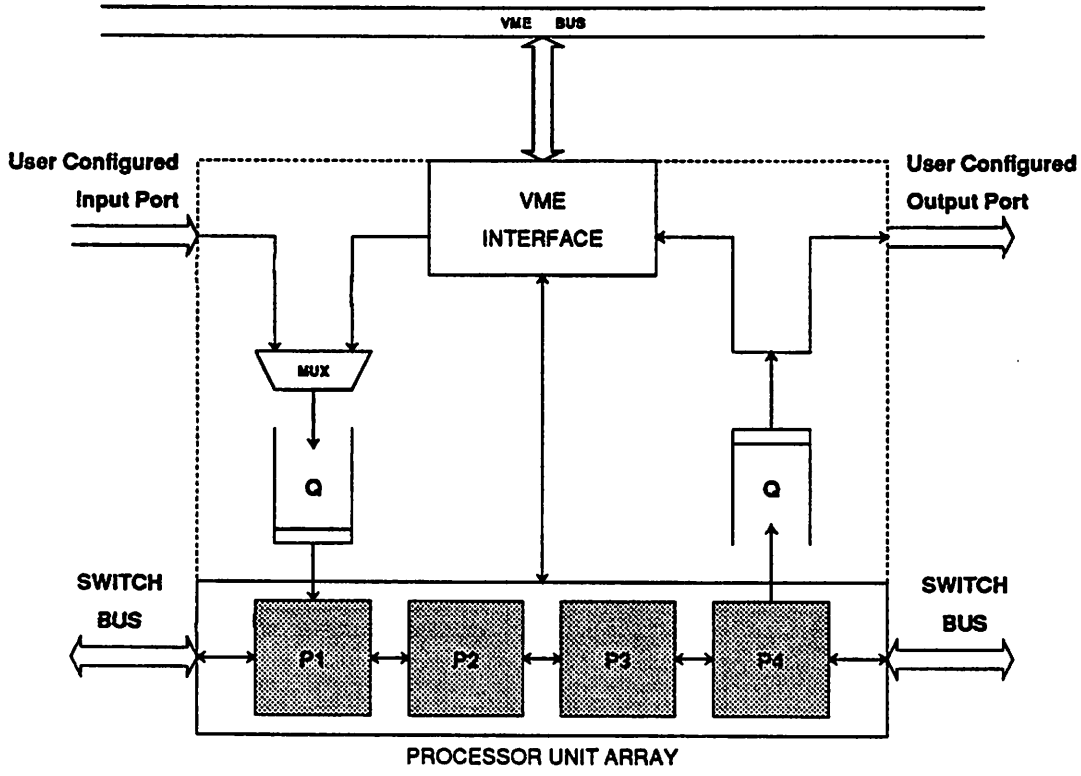


Figure 3.2: Block Diagram of the SMART Processor Board

buffers (FIFO's) and a VME interface. The VME interface is responsible for downloading programs to the core processors, monitoring the internal status of the processors and transferring data in and out of the SMART processor array. It is capable of supporting a peak data rate of 10 Mbytes per second.

In order to take advantage of the high computation power of SMART architecture, SMART provides very flexible and configurable input and output facilities. In addition to the VME interface and the left and right switchable buses (40 Mbytes per second) which transfer data between adjacent boards, the board can also be configured to receive or send data from or to an optional user configurable port if higher bandwidth is desired. For example data to the user configured input port can be fed directly from a dedicated analog to digital board and the user configured output port can be connected to the user configured input port of another SMART processor array for additional computation power. In all cases data to and from the board are buffered with FIFO's which establish a very clean interface between the board and the outside world.

Not shown are some *control registers* that can be programmed by software through the VME interface. These registers are used to generate control signals that are not *time critical.* They implement system reset and the controls for the scan paths on the board for board testing and on-board chip testing.

3.3 SMART Processing Unit

This section takes a detailed look at the SMART processing unit. As shown in Figure 3.3, each processing unit includes an AT&T WE DSP32C processor, two 4KB FIFO's, 256KB of local RAM (*LRAM*), 16KB of Dual-Port RAM (*DPRAM*) and an Access Controller chip set consisted of a Master Access Controller (*MAC*) and a Slave Access Controller (*SAC*). Only the first (*leftmost*) processing unit on the board has its input FIFO connected to the VME interface. Similarly the output FIFO is present only in the last (*rightmost*) processing unit.

The DSP32C is a 32 bit CMOS Digital Signal Processor packaged in a standard 133-pin pin-grid-array (PGA). It offers a unique set of architectural features that include: 32 bit floating point arithmetic, 16 or 24 bit integer arithmetic, 16MB of address space, on-chip ROM and RAM, serial, parallel and external memory I/O ports all equipped with direct

[1]The first prototype system only has two processing units on each board due to board area limitation.

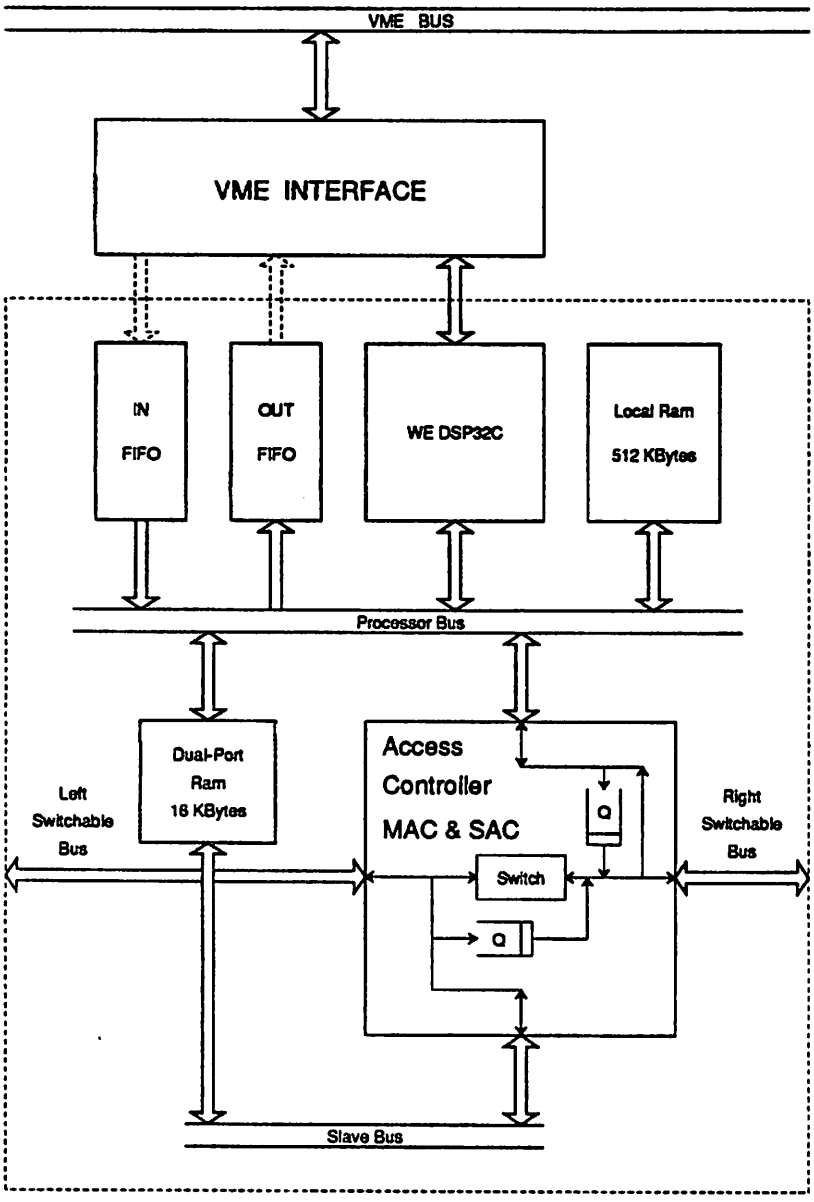


Figure 3.3: Block Diagram of the SMART Processing Unit

memory access capability (DMA), 4 40-bit accumulators and 22 general purpose registers, 2 external and 6 internal individually maskable interrupts. At its maximum operating clock frequency (50MHz), the DSP32 executes 12.5 MIPS and 25 MFLOPS concurrently. In the prototype system, the Rom-less memory configuration which provides 6KB of on-chip RAM is chosen. For further information on the DSP32C please refer to [1]. Each processor in the system is identified by a unique 6-bit *processor identification number (PID)* and therefore a maximum of 64 processors can be supported. The PID's are assigned in such a way that processors on the left have smaller PID numbers than their right-hand-side neighbors.

Communication between the processor and the CPU board is established through the VME interface via the parallel I/O port of the processor. Through the *Processor Bus* the processor can issue commands to the access controller for accessing its local LRAM, DPRAM or FIFO's as well as the DPRAM of another processing unit. A detailed description of the functions of the access controller will be presented in Chapter 5. The left and right switchable buses connect adjacent processing units to form a linear array. Short ribbon cables are used to connect the configurable bus for processing units sitting on different boards. Another bus called the *Slave Bus* connects the Access Controller to the other port of the DPRAM to facilitate concurrent access to the distributed shared memory.

Chapter 4

Design Considerations

4.1 Design Goals

In this section, the design goals of the SMART System are outlined. Their implications on the final design decisions and design methodology will be presented below. The main design goals are as follows:

- High Computation Throughput
- High Communication between Processor and the Access Controller
- Fast Switchable Bus
- Extendible System in terms of Processing Units
- Short Design Cycle

4.2 Design Issues and Methodology

Various design issues and design methodology are discussed in this section. Timing issues include the clock speed of the system, the clocking system used on chip and the problems with clock distribution. Hardware design issues involve the performance of the switchable bus, static design techniques versus dynamic design techniques and power consumption. System issues such as partitioning and extensibility are examined. Design methodology with respect to layout and simulation is discussed. All these considerations had great impacts on the design decisions that ultimately gave the system its present shape.

Clock Speed

On one hand, the maximum computation rate can be achieved by clocking the DSP32C at its peak clock frequency of 50MHz. On the other hand the speed limit of the memory parts restricts the clocking rate unless a wait state is introduced to increase the length of the external memory access cycle from two to three clock periods. Compromising system throughput with communication bandwidth, we decided to clock the system at 40MHz. The lower clock frequency also makes the implementation of the processor interface in the Access Controller less critical speedwise.

System Clocking

A 4-phase non-overlapping clocking strategy is used in SMART (Figure 4.1). The clock signals are all derived from the 40MHz system reference clock using a *Digital Phase-Locked Loop (DPLL)*. A thorough discussion of the hardware implementation of the clock generator will be presented in Section 7.1. While the phi[1-4] clocks are present in both the MAC and SAC, the a1 and a2 clocks exist only in MAC. In order to allow the DSP32C to perform one memory transaction every two cycles, the a1 and a2 clocks which run faster than the phi clocks are used in the MAC and DSP32C interface.

The p1 and b1 clocks define a *bus cycle* for the switchable bus in MAC and SAC respectively. In one bus cycle, one message is transfered from one processor to another in the same processor group. Notice that the bus cycle of SAC lags that of MAC by one phase. That one phase difference essentially accommodates the propagation delays from MAC to SAC and therefore eliminates many time-critical interface signals between MAC and SAC. A longer lag time could be chosen but it would increase the communication latency.

Before each access controller sends out a piece of data to the switchable bus, it must first perform arbitration with all the other access controllers hanging on the same shared bus. To obtain higher throughput on the bus, arbitration and data transfer are pipelined in such a way that arbitration is performed one cycle ahead of the actual data transfer, as depicted in Figure 4.2. Furthermore MAC is responsible for sending the address and some other control signals onto the bus. Hence the address always arrives one phase ahead for decoding before the corresponding data arrive when accessing a remote memory.

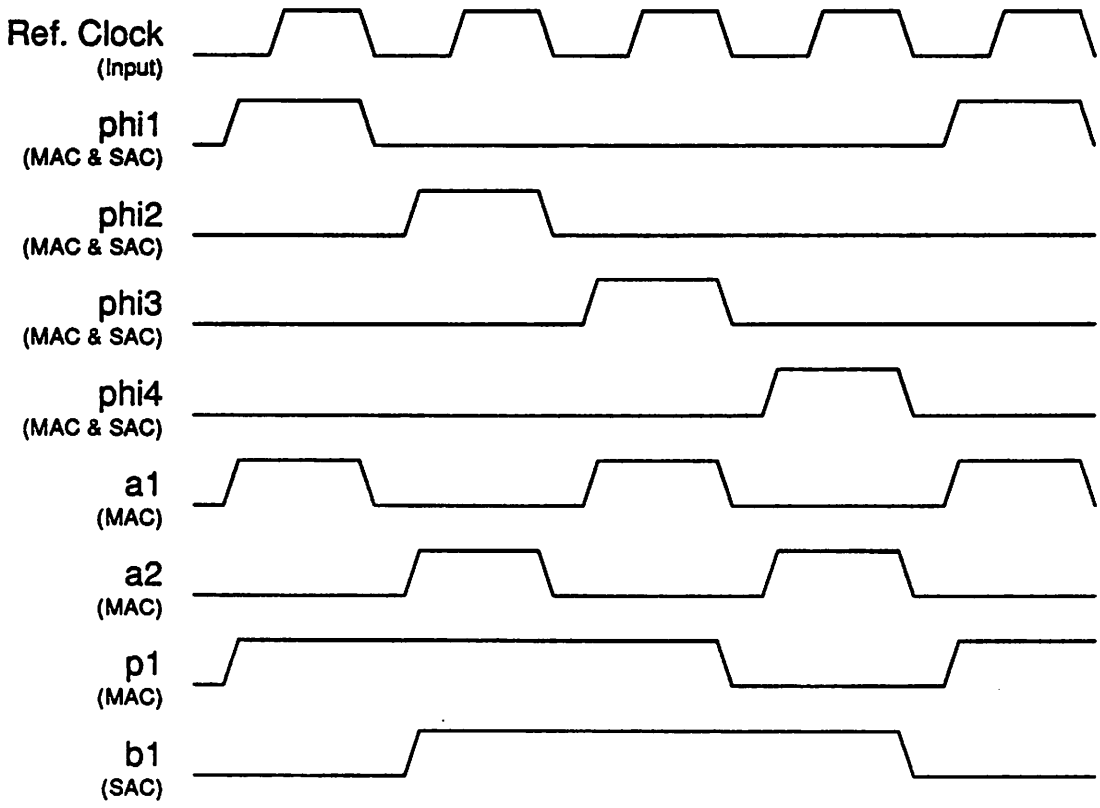


Figure 4.1: Clock Signals in SMART System.

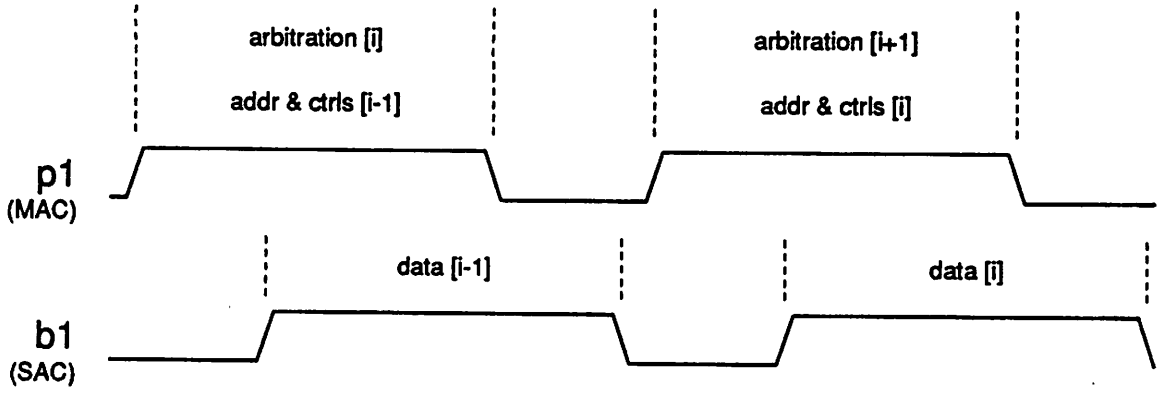


Figure 4.2: Pipelining of Arbitration and Data Transfer.

Clock Distribution

A careful and elaborate clock distribution scheme is adopted to solve the problems with clock skews and transmission line effects when high frequency clock has to be distributed throughout the multi-board system. To shorten wire length and balance clock skews, the system clock is buffered locally on each board. Moreover, terminating resistors are used extensively to alleviate transmission line effects. As a rule of thumb, for the printed circuit board used for our system, transmission line effects become serious when the following inequality is satisfied[8]:

$$t_{tr} \leq C t_d \tag{4.1}$$

The transition time, t_{tr}, is either the rise time or the fall time of the signal. t_d is the intrinsic delay of the signal which depends on the physical medium the signal travels on. C is a constant between $\frac{1}{4}$ and $\frac{1}{3}$.

In the SMART system, processors only communicate with their immediate neighbors. Therefore only clock skew between adjacent processors has to be considered. The worst case occurs between the first and the last processors which are located at opposite ends of the array of processor boards. The solution is to drive the clock from a board in the middle of the array. By balancing the loading of the clock line on both sides of the array, the clock skews experienced by the first and the last processors hopefully will cancel out.

Design of Switchable Bus

In one *bus cycle*, data have to ripple through a series of closed switches to reach its destination. Therefore the speed of the switchable bus is crucial to the performance of the system because it imposes an upper bound on the number of processers working in parallel. To improve this upper bound we can build a fast circuit for the switch and/or increases the length of the bus cycle. However, lengthening the bus cycle results in low effective bus bandwidth.

The decision was to use a *synchronous* bus with a 100ns long bus cycle (equal to 4 system clock cycles). The bus cycle is equally divided into 4 phases, 3 of which is allocated for data propagation on the switchable bus. Estimating the delay through a single switch based on SPICE simulation, we expect the system to support up to 8 processors on a shared bus.

The choice of a synchronous bus design over an asynchronous design deserves some explanation. In addition to simple implementation, the synchronous design also enjoys a speed advantage over its asynchronous counterpart in that handshaking with neighboring access controllers is not necessary. In the asynchronous case each pair of adjacent access controllers must perform handshaking to ensure proper transfer of data.

On the other hand, the synchronous bus design and hence a global system clock presents a big obstacle to the extensibility of the SMART System. Clock skews and transmission line effects as a result of distributing the high frequency system clock over a couple of boards are problems that we must carefully address.

Logic Design vs Circuit Design

An initial estimate revealed that the die sizes of the chips to be implemented are likely to be limited by the number of pins. Thus the need to conserve silicon area has a low priority.[1] The strategy is to eliminate critical paths by redesigning the logic whenever possible even at the expense of more gates and hence area. Circuit design with SPICE simulation is done only for the critical paths or for circuits which must be optimized for system performance. Otherwise the *finger counting technique*, i.e., estimating the total delay by adding the delay of each gate based on a very simple timing model, is used for timing verification.

Static vs Dynamic Design

While dynamic circuits are usually superior to their static counterparts in terms of speed[2] and area, they are also more difficult to integrate into the system due to their timing constraints. A static data-path cell is easier to use and more likely to be reused than a dynamic cell of equivalent function. In addition, the power spikes, increased clock loading and potential timing problems associated with dynamic designs may more than outweigh its advantages. Static design is further favored when there is no urgent need to conserve chip area.

[1] Even though the die size is dictated by the pinouts, the active chip area still has an effect on the process yield.

[2] This is generally true for medium to high clock frequencies where time lost due to clock skew is not a significant portion of the clock period.

Power Consumption

By using a CMOS technology and minimizing the use of circuit techniques that consume static power, power consumption of the Access Controller is estimated to be small (less than 2 Watts for the two-chip chip set) and therefore power is not a big concern in the design process.

System Extensibility

Extensibility is sacrificed to achieve more speed and simplier implementation. For the first prototype system which can accommodate no more than 64 processors, the synchronous bus is *acceptable*. To build an extendible system, we must adopt a design that is immune to clock skews and has more tolerance on variations in propagation delays. An asynchronous bus is a more logical choice.

System Partition

The functional complexity and the large number of inputs and outputs of the Access Controller dictate the need for partitioning into smaller sub-systems for VLSI implementation. Considerations include die size, pinouts, package availability, testability issues and minimization of interface signals and critical paths between sub-systems after partition, etc. In the prototype system, the Access Controller is divided into two chips: the *Master Access Controller (MAC)* and the *Slave Access Controller (SAC)*, each of which contains 208 pins. Their designs and functions will be described in details in Chapter 5 and Chaper 6. Essentially MAC is the *master* of the Access Controller and SAC, being the *slave*, only responses to the control signals generated by MAC.

Computer Aided Design

It is expected that the design of the entire system be completed within a one and a half year period. To accelerate the design process, CAD tools are used extensively to assist layout, circuit and logic simulation and test vectors generation.

Layout Strategy

Manual layout is avoided as much as possible by using the *LAGER* silicon compilation system[2] and a cell-based modular design. Manual layout is only done on the cell level, above that the layout is generated automatically by LAGER. Moreover a bottom-up approach is used in layout so that blocks of layout are first generated and simulated before they are assembled to form more complex blocks. This strategy facilitates modification of the design and hence accelerating the design-simulate-debug cycle. Manual optimization of the layout is performed only after complete verification of the design.

Simulation

Simulation occupies a very significant portion of the design cycle and fast and efficient CAD support is a must. In this design, the physical layout was simulated by *irsim*, an interactive event-driven logic-level simulator for MOS transistor circuits. Using the *linear* simulation model, irsim can also function as a timing verifier. Inputs containing netlists and capacitive loading information to irsim were directly extracted from the layout. With accurate models for the transistors and parasitic capacitance, irsim can be used to verify critical paths of the circuits. SPICE simulations were performed only on critical paths *picked out* by irsim.

Parallel with the physical design, a complete behavioral description of the system in a high level hardware description language called *csl* is also created. Not only does the model allow verification of the architectural features in advance but it also speedups the verification process by providing a cross-checking facility with the physical design. Vectors used in simulating the physical design can be generated conveniently by the model using the high level language.

Chapter 5

Master Access Controller: MAC

The main function of the Access Controller is to *integrate* the individual processors to form the SMART multiprocessor machine. In addition to directing traffics between processors through the configurable bus, it also handles synchronization and arbitrations among processors. Access to local memoris and FIFO's by the processor are also controlled by the Access Controller to eliminate any glue logic.

As explained in Chapter 4, practical considerations dictate the need to partition the Access Controller into two chips: the *Master Access Controller (MAC)* and the *Slave Access Controller (SAC)*. As its name implies, MAC is the *brain* of the Access Controller and therefore its functions are much complicated than those of SAC. The functionality and implementation of MAC will be given in this chapter while SAC will be discussed in the next chapter. Each functional block of the chip will first be explained, followed by the discussion of the circuit design issues of the swithable bus. Finally the floorplanning and layout of MAC will be described. A summary of the MAC instructions and a description of the pinouts of MAC are included in Appendix A and Appendix B.

5.1 Functional Description

The functionality of MAC will be explained in this section. Figure 5.1 shows the functional block diagram of MAC. It includes a master FIFO and a bypass FIFO, a switch block for the configurable bus, master and slave control units, an arbiter for arbitration, logic for synchronization and a clock generator. Each of the blocks will be discussed in details in the following subsections except the clock generator block which will be covered

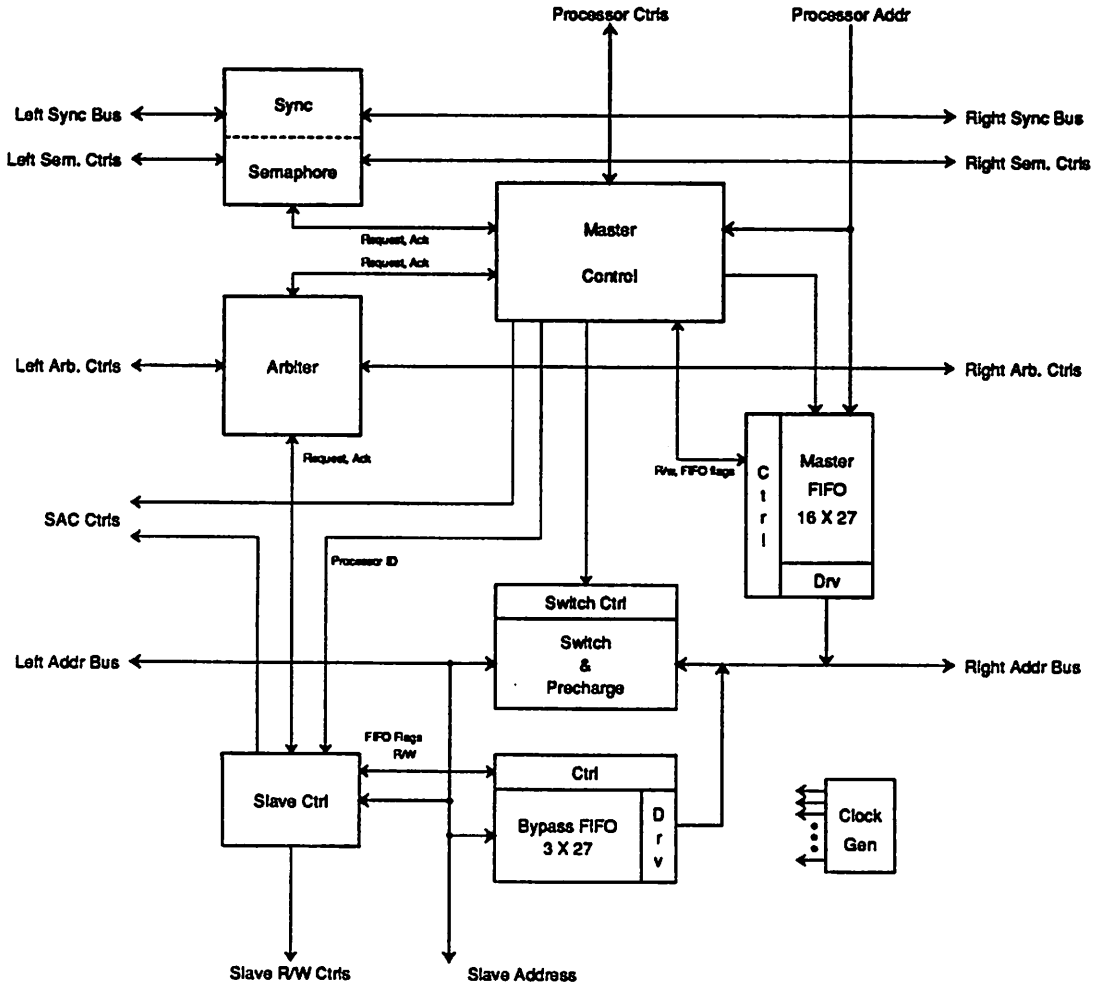


Figure 5.1: Functional Block Diagram of the Master Access Controller.

in Section 7.1.

Master Control

The majority of the operations in MAC is controlled by the *Master Control Unit (MCU)*. The DSP32C issues instructions to MAC by performing an external memory access. MAC decodes the address according to an address map and carries out the necessary actions.

The MAC instructions can be broadly categorized into two groups according to their execution time. Group 1 instructions include instructions whose execution can be completed within a pre-determined period of time and therefore no handshaking is required between MAC and the processor. On the other hand Group 2 instructions usually take a variable amount of time to execute, for example, synchronization or semaphore instructions. In this case DSP32C must *busy-wait* until MAC asserts the system ready signal *(SRDYN)*. For a further description of the address map and the MAC instructions, the reader is referred to Appendix A.

The instructions can also be divided into 4 types according to their functions:

1. Configuration Instructions
2. Local Memory Access Instructions
3. Network Memory Access Instructions
4. Synchronization Instructions

MAC maintains a set of 21 configuration registers accessible to the DSP32C by executing the configuration instructions. Their roles include switch configuration, operation enabling or disabling and identification (pid registers). A list of the configuration registers and their functions is given in Table 5.1. Their precise functions will be explained in more details as they are encountered in the following discussion. As an example, the processor can write the *pid* into the pidReg during system startup or close the switch on the configurable bus by setting *switchState1* high.

The local memory access instructions involve accessing *local* memories which include the LRAM, the processor-port of the DPRAM and the external FIFO's. The MCU simply asserts the corresponding chip enable signal(s) such as *ceF[]*.

Network memory access instructions are used when the processor needs to read or write from a distant processor. In the case of write operation, the MCU enqueues the *write*

Name	Number of Bits	Functions
pidReg	6	Processor Identification Number
syncSwitch	5	Control switches of the synchronization bus
syncDisable	6	Disabling of the synchronization patterns
switchState1	1	Control switch of the configurable bus
switchState2	1	Control switch2 for arbitration
swapState	1	Control Bank Select
bypassEn	1	Enable Bypass Operation

Table 5.1: List of Configuration Registers in MAC.

request into the master FIFO and acknowledges DSP32C the completion of the instruction by asserting *SRDYN* as long as the FIFO is not full yet. Otherwise the acknowledgment is delayed until space in the FIFO is freed up. In the case of read operation, the read request is enqueued as before but the acknowledgment is not issued until the read message is sent out to the network and the data to be read are returned. Read operations are to be avoided whenever possible because the processor must be idle throughout the operation.

It is also possible to issue a Group 1 network memory access instruction with no handshaking between the processor and MAC. In this case the compiler has to guarantee that executing the instruction does not overflow the master FIFO.

The MCU is also responsible for generating the control section of the message to be stored in the master FIFO. Special network memory accesses such as broadcasting are realized by enabling the corresponding control bits *(bc)*.

Synchronization and semaphore instructions are not handled directly by the MCU. Upon receiving a synchronization type instruction from the processor, the MCU sends the appropriate request signals to the *Synchronization Block* or *Semaphore Block* which communicate directly with neighboring Access Controllers. Upon the completion of the instruction, the Synchronization Block or Semaphore Block sends back an acknowledge signal to the MCU which in turns signals to the processor. These operations will be discussed in more details in later sections.

Besides handling instructions, the MCU also monitors the status of the master FIFO. The MCU will send a request for bus usage to the arbiter as long as there are requests stored in the FIFO. An acknowledge from the arbiter to the MCU will trigger a transfer of a request/message from the FIFO to the switchable bus.

Since MAC only handles the control and address portions of the message, for some operations occurred in MAC, *companion* actions that involve the data may be required in SAC. For example whenever a write message is loaded into the master FIFO in MAC, the data to be written must also be stored in the master FIFO in SAC at the same time. MCU controls these companion actions by asserting the appropriate control signals to SAC.

Master FIFO

The master FIFO is a 27 bits wide, 16 words deep first-in-first-out buffer. Its primary function is to provide a temporary storage for messages consisting of a control section from the MCU and an address section from DSP32C address bus. The read or write operations of the FIFO are controlled by the MCU based on two flags (the *Full Flag* and the *One Flag*), generated by the FIFO. The *One Flag* (which indicates that only one piece of data is left in the FIFO) is used instead of the usual *Empty Flag* to eliminate some time critical circuits. After a write operation, the message is dispatched onto the internal dynamic precharged bus by the built-in bus drivers of the FIFO. More discussion on the logic and circuit designs of the FIFO will be given in Section 7.3.

Bus Switch

The Switch Block implements the *switches* and the precharge logic of the configurable bus. The opening and closing of the switches are controlled by the *switchSate1* configuration register. Section 5.2 will address the circuit design issues of the configurable bus.

Slave Control

The *Slave Control Unit (SCU)* controls access to the slave-port of the DPRAM and message bypassing. The SCU constantly monitors traffic on the switchable bus and compares the pid of the incoming message with the local pid. The result of the comparison and the control bits in the control section of the message determine the action to be taken.

If the pid's match or the *broadcast enable bit* in the message is set, a read/write operation to the DPRAM is performed depending on the *rwN* bit. If the destination pid is larger than the local pid, and the switch is open (i.e., the *switchSate1* configuration register

is set low), and bypass operation is enabled (i.e., the *bypassEn* configuration register is set high), a bypass operation is enabled by loading the message into the bypass FIFO.

The *circular bypass enable* bit enables circular bypass operation. Normally, only messages whose destination *pid* is larger than the pid of the processor will be bypassed through an open switch. The circular bypass enable bit forces an bypass operation regardless of the destination pid. The circular enable bit is disabled by hardwiring when the messages is bypassed from the last processing unit (pid 111111) to the first processing unit (pid 000000) so that the message will not keep circulating around the array.

The broadcast operation is enabled by setting the *bc* bit high. When enabled, the message will be broadcasted to groups of processors up to the group which contains the processor whose pid matches the destination pid in the message. The grouping of the processors is determined by the bus configuration. In the example in Figure 5.2, processors P1 to P7 in Group 1, 2 and 3 all receive the broadcasted message.

The reader is referred to Appendix B for a more detailed description of the message format.

Similar to the MCU, the SCU responds to the status flag generated by the bypass FIFO. If the FIFO is not empty, a request is issued to the arbiter logic for bus usage. Unlike the master FIFO, the slave FIFO provides only the *One Flag* but not the *Full Flag* because it is guaranteed that the FIFO will never overflow. This is true because the bypass operation is only enabled if the switch is open, i.e., the processor is the leftmost processor of a bus group which always enjoys the highest priority of all in bus arbitration. The worst case scenario is that the processor with non-empty bypass FIFO makes a request for bus usage when a read request is just granted to another processor. Because of its high priority, the request of the processor is guaranteed to be granted after the read request is serviced which takes three bus cycles. Therefore, a depth of 3 for the FIFO is enough to avoid overflow.

As in the case of the MCU, *companion* operations must be performed in SAC to take care of the data. SCU accomplishes that by generating control signals to SAC.

One of the architectural features of SMART is the ability of the processor to access the processor-port of the shared memory independent of the activities on the configurable bus. Hardware support for this feature is evident from the fact that the SCU can operate concurrently and independently with the MCU.

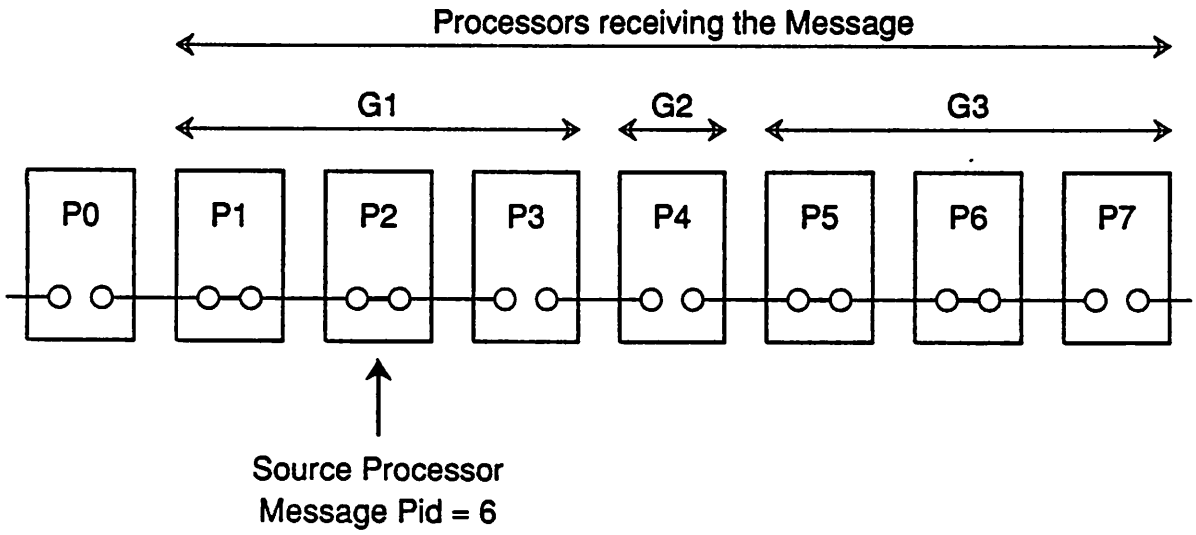


Figure 5.2: Example showing Broadcast Operation.

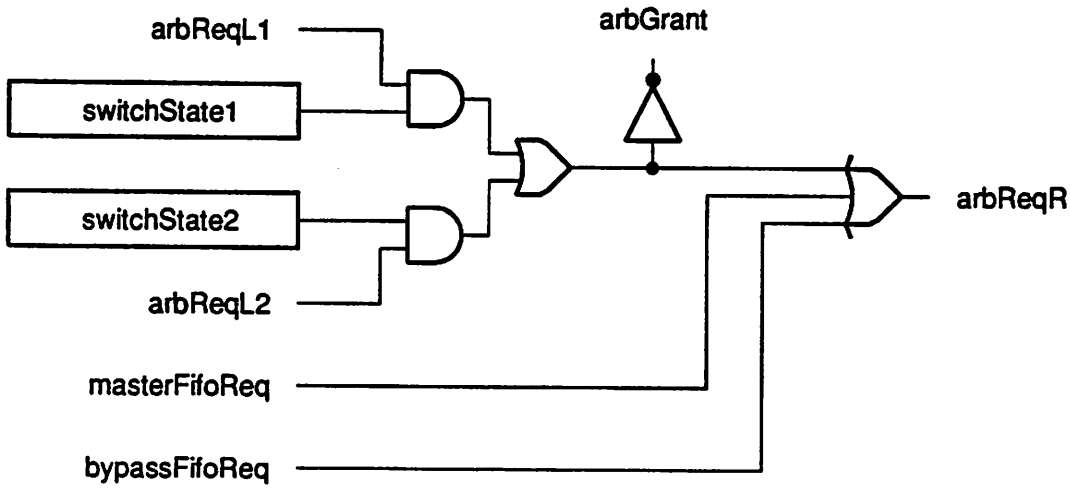


Figure 5.3: Conceptual Logic Diagram of the Arbiter.

Bypass FIFO

The Bypass FIFO is very much like the master FIFO except that it only has a depth of 3 and there is no *Full Flag* being generated. Its function is to provide an alternate path for messages to go through an open switch.

Arbiter

Bus arbitration is necessary when there are more than one processor wanting to use the bus. This important task is performed by the *Arbiter Block* in MAC. Figure 5.3 shows a conceptual logic diagram of the arbiter. The output *arbReqR* is asserted to disable arbitration of the right-hand-side processors (which have lower priority) when there is request coming in from the left (higher priority), or the master FIFO and/or the slave

FIFO are/is not empty. Arbitration is granted (*arbGrant* asserted) when processors on the left do not request for the bus (*arbReqL1* not asserted) and there is(are) pending internal request(s) signalled by the assertation of *masterFifoReq* and/or *bypassFifoReq*. The *switchState1* configuration register is there to block the influence of the processors in a different processor group. The *switchState2* configuration register and the input *arbReqL2* provide the possibility of connecting two MAC's to one but they are not used in the prototype system. A fine detail not shown in Figure 5.3 is that the arbitration process is halted for two bus cycles whenever a read request is granted because unlike a write request, a read request requires 3 bus cycles instead of 1 to finish.

Careful logic and circuit designs are done to minimize the levels of logic and the propagation delay from input of the chip to the output of the chip through the arbiter logic because just like the propagation delay of the switchable bus, this propagation delay also imposes a limit on the number of processors sitting on a shared bus.

Semaphore

Controlled access to a shared resource can be implemented easily using semaphores. In SMART a pair of instructions (*semP* and *semV*) are provided to manipulate the semaphore. The semP instruction is used to acquire the semaphore and the semV instruction releases it.

The semaphore logic is shown in Figure 5.4. The MCU lets the *Semaphore Block* know that a semP instruction has been issued by asserting the *semP* signal. The *semAck* signal is enabled to signal to the MCU the successful acquisition of the semaphore when no processors with higher priority want to obtain the semaphore and the semaphore is not *locked* by any processors. After obtaining the semaphore the semaphore is automatically locked by setting an RS Flip-Flop with the *semAck* signal. The semaphore is not released until it is *unlocked* by executing a semV instruction (resetting the RS Flip-Flop).

As in the arbiter logic, the *switchState1* configuration register is used to isolate the semaphore of one processor group from another. Again the propagation delay is minimized to allow more processors to share the same semaphore.

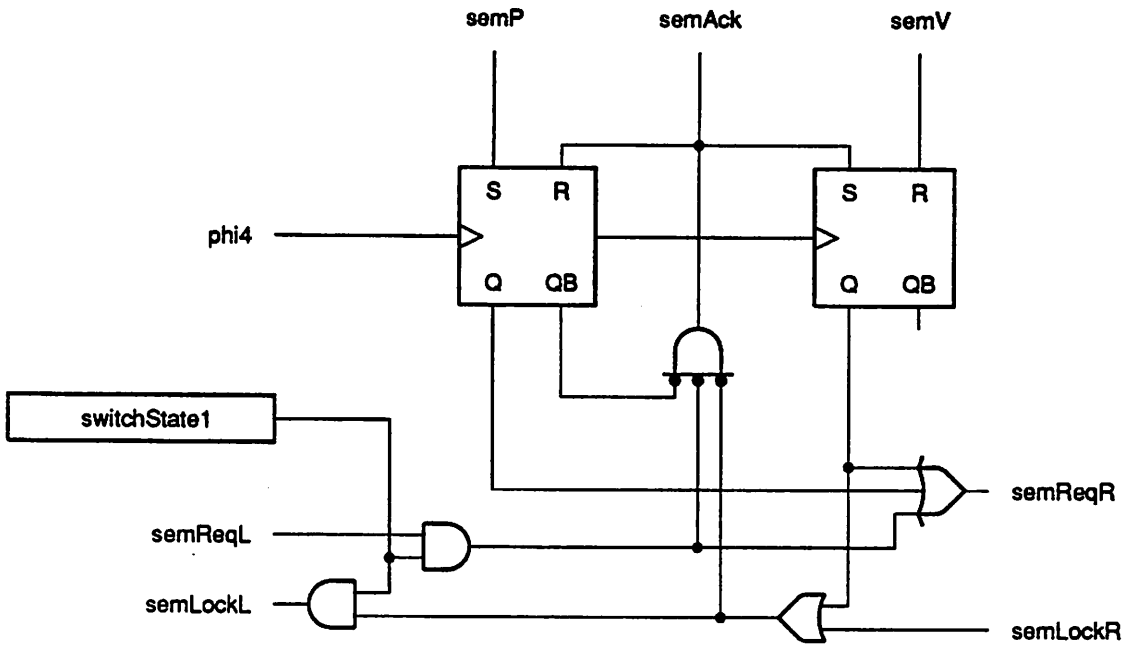


Figure 5.4: Conceptual Logic Diagram of the Semaphore Logic.

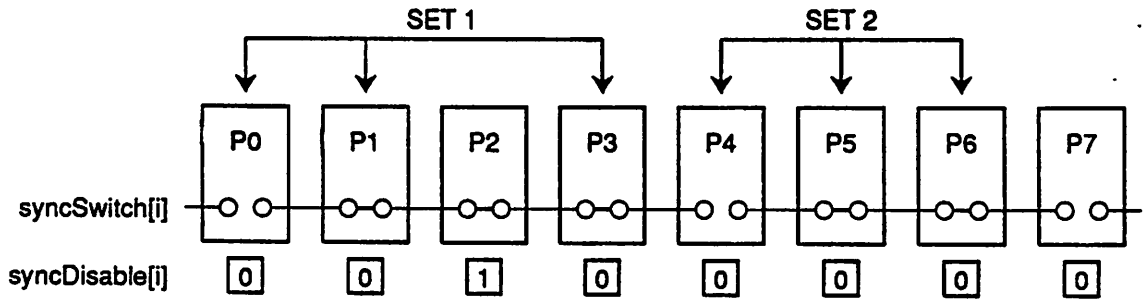


Figure 5.5: Example of an Synchronization Pattern.

Synchronization

The *Synchronization Block* is custom-designed to support the *barrier synchronization mechanism* in SMART. To synchronize a set of processors at a certain point of the program, for example, after finishing the computation on the current sample of data, every processor in the set is required to execute a synchronization instruction at that point of the program. The acknowledge for the completion of the synchronization instruction will not be issued until all processors in the set issue the instruction. A processor which arrives at that point in the program earlier than the other processors must wait. A set of processors are said to be *synchronized* when every processor of set has issued a synchronization instruction.

The processor *set* is defined by the *syncSwitch* and *syncDisable* configuration registers. An example to illustrate set division is given in Figure 5.5. First, adjacent processors

are divided into groups by opening or closing the *syncSwitch.* (P0 to P3 and P4 to P6 in the example.) Next, individual processors within the group (P2) can be disabled by setting the corresponding *syncDisable* configuration registers. In the example P0, P1 and P3 are in one set while P4, P5 and P6 are in another set. Thus the programming of the syncSwitch and syncDisable configuration registers defines a *synchronization pattern.* In each MAC, 5 syncSwitch and 6 syncDisable configuration registers are available to support 5 *fully* programmable and 1 *partially* programmable synchronization patterns. The partially programmable synchronization pattern is called the *Global Synchronization Pattern.* Conceptually we can think of the syncSwitch of this pattern as being closed all the time. By setting the appropriate bits in the synchronization instruction the processor can activate any one or more of the synchronization patterns in one instruction. An acknowledge is not granted until every set of processors corresponding to the activated patterns is synchronized.

Figure 5.6 shows the conceptual logic diagram of the Synchronization Block for one synchronization pattern. Communication between neighboring Synchronization Block is established by the dynamic precharged synchronization buses, *syncR and syncL.* More discussion on the design issues of the dynamic bus will be given in Section 5.2. One important characteristic of this dynamic design is its ability to implement a *configurable wired-and gate.* Referring to Figure 5.5 the portion of the synchronization bus to the right of P4 and to the left of P7 will be low if any processor in *SET 2* pulls down the bus. The fact that the bus stays high means that all the processors in the set are *ready* for synchronization, i.e., each processor has issued a synchronization instruction and no one is pulling the bus low. If a static design was adopted, two pins instead of one would be needed on each side since information has to flow in both directions, thus doubling the pinout requirement. In addition, this design offers high speed and therefore more processors can be grouped into the same set.

A communication pattern is activated by asserting the corresponding *syncReq[i]* bit which sets a RS Flip-flop. The synchronization bus is sampled at the end of the evaluation phase of each bus cycle to determine if all processors in the set are *synchronized.* A high level on the bus resets the flip-flop and generates the *syncAck[i]* bit. The MCU is not acknowledged until all the *syncAck* bits of the activated patterns are set. Setting the *syncDisable* register simply disables the pull-down driver of the synchronization bus and therefore the processor is always *ready.* It is worth pointing out that upon receiving a synchronization instruction, MAC still pull downs the synchronization bus until the FIFO's

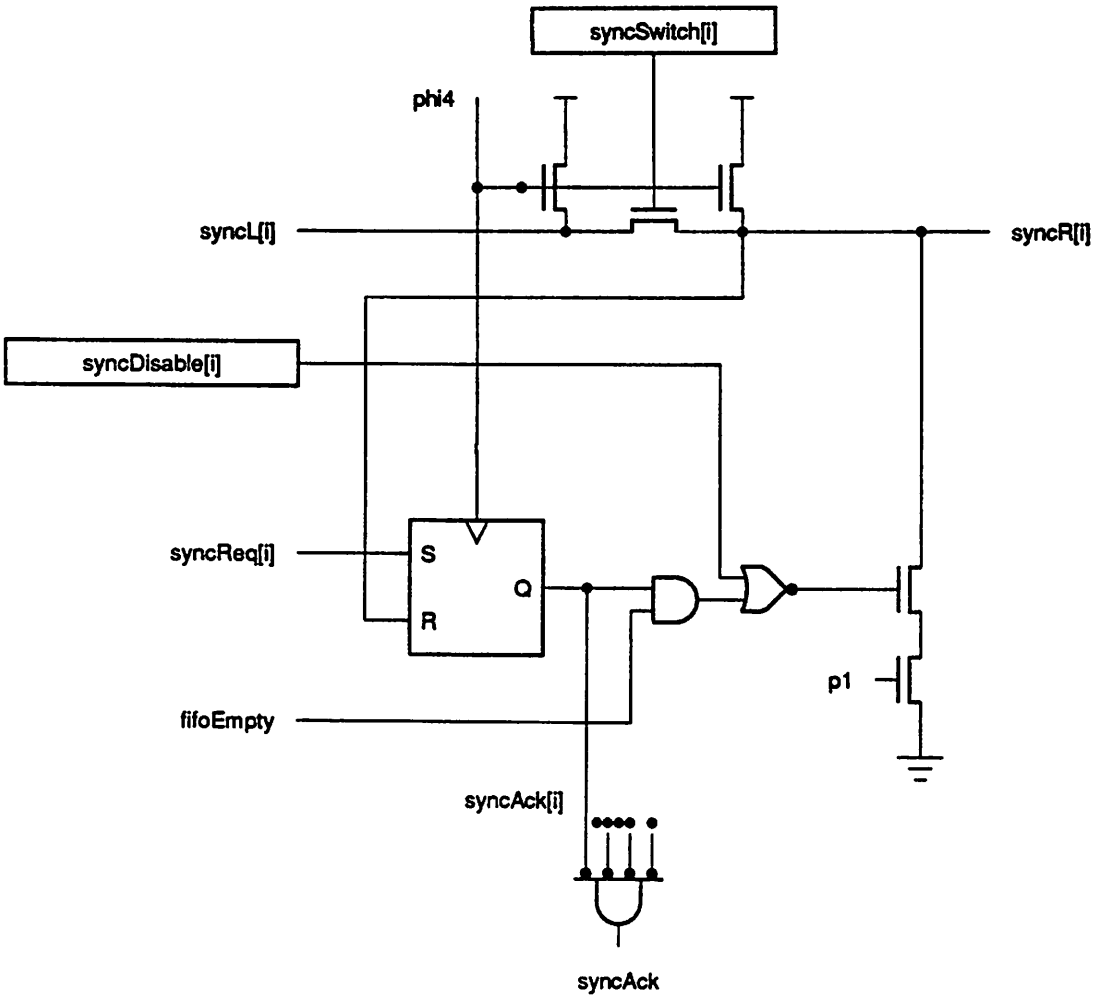


Figure 5.6: Conceptual Logic Diagram of the Synchronization Block.

are empty(fifoEmpty asserted), i.e., there are no pending messages.

5.2 Switchable Bus Design

In one bus cycle data are transferred synchronously from one processor to another sitting on the same shared bus. It is therefore important to keep the delay on the bus through a closed switch as small as possible since it imposes an upper bound on the number of processors which can communicate with each other directly through the shared bus. In this section a dynamic design of the configurable bus is first presented which achieves a low pad-to-pad propagation delay of 6.0ns. The same design is used to realize a configurable WIRED-AND gate for synchronization of processors with half the number of pins normally required using static circuits as discussed in Section 5.1.. The pros and cons of the design are then evaluated. Next a more conservative pseudo-dynamic design which attempts to compromise performance with reliability is introduced.

Dynamic Switchable Bus Design

To optimize speed, both the internal bus and the external bus are implemented using dynamic circuit. The circuit diagram of the dynamic bus driver is shown in Figure 5.7 and Figure 5.8. M2 and M3 are large pull-up and pull-down n-channel transistors while M1 is a small transistor whose function is to safeguard against charge leakage and external capacitive coupling effects during the evaluation phase. The weak holding transistor M1 does not consumes static power because it is turned off once the internal bus is pulled low as a result of the external bus being pulled down. The holding transistor for the external bus and the charge holding circuit (not shown) for the internal bus keep the buses in a high state against leakage. This is important when testing at low clock frequency is performed. To reduce current spikes at the onset of bus precharging, both the p-channel precharge transistor for the internal bus and the n-channel precharge transistor for the external bus are sized as small as possible so that precharging is just accomplished at the end of the precharge phase. Moreover, the driver sizes for the precharge transistors are also minimized so that the rise or fall delays at the gate controls of the precharge transistors are long.

Besides its high speed, the bus design also allows data to flow in *either* direction without the need of any direction control or tristate control. This greatly simplifies the design of the Access Controller.

Despite its numerous advantages, the design is risky especially when it is used in a noisy environment like a high speed printed circuit board. The greatest concern of the

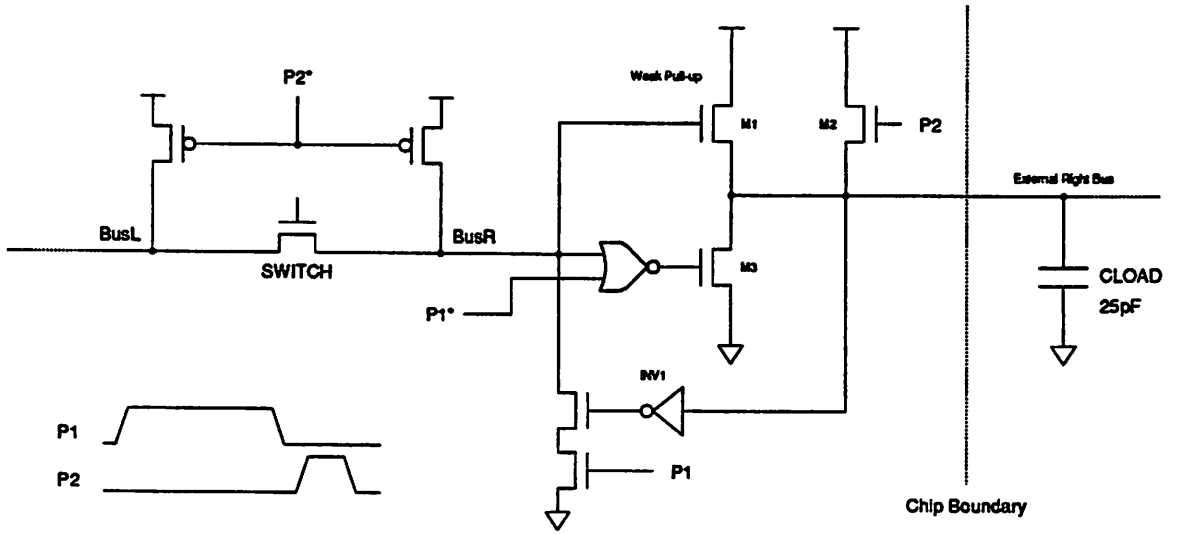


Figure 5.7: Circuit Diagram of the Dynamic Bus Pad Driver.

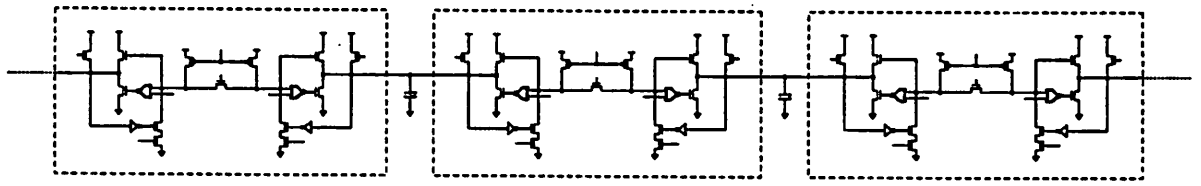


Figure 5.8: Dynamic Configurable Bus.

circuit is its *robustness* against noises on power supplies and signals, external capacitive couplings and transmission line effects. Any accidental discharge of the internal or external dynamic bus beyond some threshold will trigger a *positive feedback* and produce faulty results. Special measures must be taken both on the board level and the chip level to guarantee the functionality of the design.

On the chip level, the size of the holding transistor M1 can be increased to improve the immunity of circuit by increasing the high noise margin (NMH). Since the noise margin low (NML) is not important in precharged circuits, the NMH of the driver can be further enhanced by lowering the trip point of the level-shifting inverter INV1. However, both the forementioned techniques involves trading off speed with safety margins. The design goal is to squeeze out as much safty margin as possible without excessive sacrifice in speed. Also, the routing of the internal dynamic bus must be carefully examined to be sure that on-chip capacitive coupling is not a factor. Wide power buses, preferably in second layer metal, and numerous power supply pins should be provided to reduce supply noises and resistive voltage drop.

On the board level, design techniques such as shielding and minimization of crossovers should be applied to cope with capacitive coupling and crosstalk effects. Wire lengths of the interprocessor busing should be kept short to avoid serious transmission line effects. The dynamic design is a good candidate for new packaging techniques such as Wafer Scale Intergration and Hybrid Wafer Scale Intergration[7] which offer a less noisy environment.

The design was implemented and fabricated using a 2um CMOS technology through the MOSIS facilities. The chip was tested and a pad-to-pad delay of 6.0ns was recorded. (Spice simulation predicts a delay of 6.5ns.) A picture of the oscilloscope trace is shown in Figure 5.9.

Unfortunately the testing also revealed that the design is very susceptible to capacitive coupling effects. The worst case test for coupling effects is when all the bits of the bus except one are pulled down during the evaluation phase of the bus cycle. Test results showed that some of the bits of the bus failed(incorrectly discharged) when subject to the worst case test. Another test was performed in which all bits except a group of 3 consecutive bits are set low. Results showed that the middle bit of the group never failed. This can be explained by hypothesizing that the failure is caused by capacitive coupling. Since the bonding pads, bonding wires and pins of adjacent bits are close to each other and

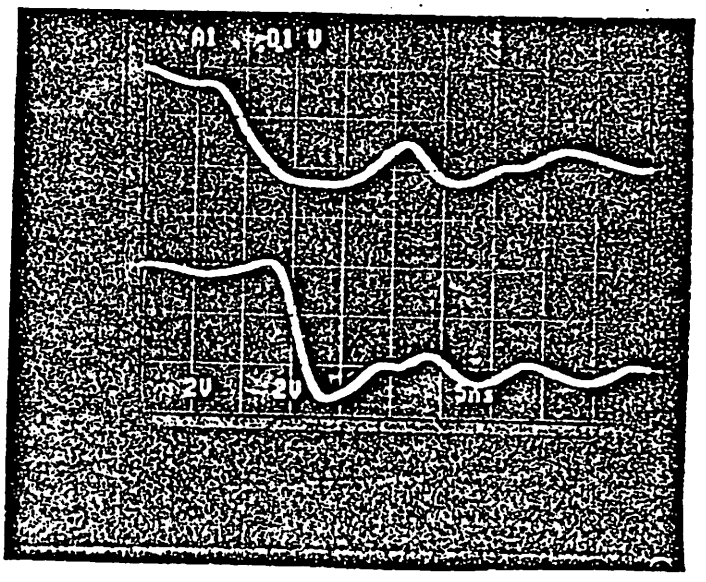


Figure 5.9: Oscilloscope Trace of the Dynamic Configurable Bus. Top trace is the input and the bottom trace is the output.

therefore strongly coupled, the chance of failure due to coupling is reduced if the adjacent bits are kept high.

Another negative effect is the degradation of the *effective* noise margins due to glitches on the power rails of the test board during discharge. The combination of both effects further jeopardises the functionality of the circuit.

Pseudo-dynamic Switchable Bus Design

Although tight control of the environment may improve the chance for the design to work, the decision was to go for a more conservative design. The dynamic design was still used for the synchronization bus because of the small bus width(less coupling and switching noise) and the pin-saving the design offers.

To avoid drastic changes to the design and maintain similar speed, a pseudo-dynamic design is used. The redesigned circuit diagram of the switchable bus is shown in Figure 5.10. The design is very much the same as the previous design except that the output driver and the input driver are both tri-state drivers. By enabling only one of the two drivers, the feedback path can be cut. Moreover the internal bus and the dynamic bus are now driven actively all the time and therefore more robust against noises. An unusually large transistor M5 is used for the inverter so that the delay for pulling down the internal bus is not too much slower than the dynamic design. The penalty of this design is that the arbitor block has to determine the direction the data is going to travel.

The new design was fabricated using the same technology and subsequent measurement recorded a pad-to-pad delay of 6.8ns. A picture of the oscilloscope trace is shown in Figure 5.11.

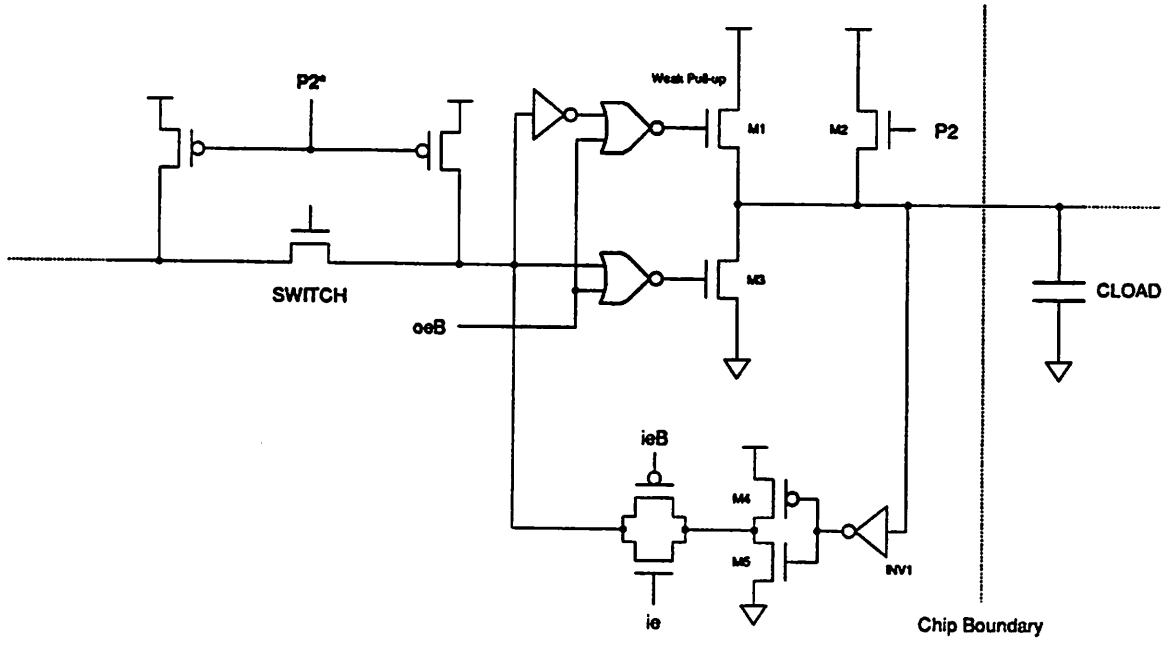


Figure 5.10: Pseudo-dynamic Design of the Configurable Bus.

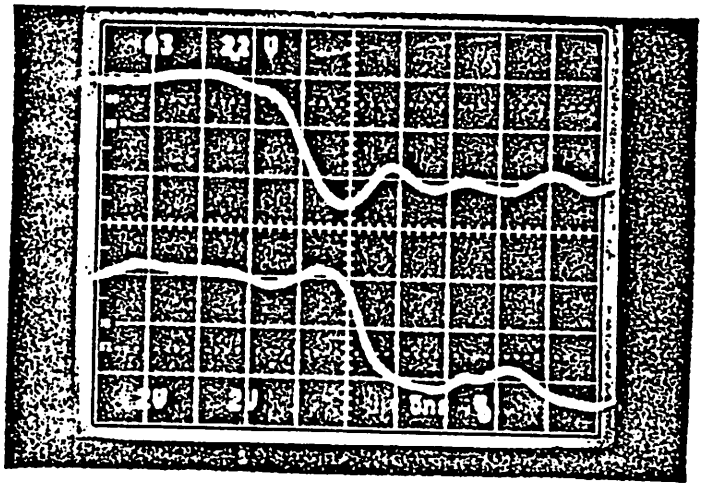


Figure 5.11: Oscilloscope Trace of the Pseudo-dynamic Configurable Bus. Top trace is the input and the bottom trace is the output.

5.3 Floorplan and Layout

The layout process can be roughly divided into three phases. In the first phase individual cells of each block are layouted manually if they are not available in the cell library in LAGER. The cells are then assembled and connected to form blocks. LAGER provides a number of layout generators for different layout styles. A standard cell place-and-route tool set based on *Timberwolfe* is used for generating layout for control logic blocks such as the Master Control Unit and the Slave Control Unit. A datapath compiler called *dpp* generates layouts for the Synchronization and Switch blocks. Layout for macro blocks such as the FIFO's and the pad groups are compiled by the cell tiler *TimLager* in LAGER. Blocks of various sizes can be created at ease by changing some parameters. The MCU is actually divided into three smaller blocks according to the interconnection patterns to achieve more compact layout because the layout generator does not have to deal with a large number of cells. In addition the sub-blocks can be individually placed to take advantage of the communication pattern between blocks.

In the final phase these blocks are placed and routed by an interactive floor-planner and channel router called *Flint*[2]. A hierarchical approach is taken such that the blocks are first routed to form the *core* block and the *clock generator* block. The core, the clock generator and the four pad groups are then routed together to complete the chip layout. This approach allows more control over the actual placement of the blocks. The blocks inside the core block are placed in close proximity of each other since they communicate very often with each other. The clock generator whose analog component is sensitive to noises can be placed near the pads so that power supplies and analog inputs and outputs do not have to suffer large resistive voltage drops and capacitive coupling effects. *Active* area of the chip which has great impact on manufacturing yield is also reduced by clustering blocks together.

Pin assignment is made following two guidelines. Bonding sites which offer the least parasitic inductance (shortest connection to the pins) are first reserved for power supplies to reduce switching noise and voltage spikes on the power lines. Signals are then assigned to minimize interconnects from the core block to the pad groups with priority given to time-critical signals.

The complete layout of MAC is shown in Figure 5.12. The chip has 208 pins, a total of 20883 transistors, of which 12037 are n-channel mosfets and 8846 p-channel mosfets.

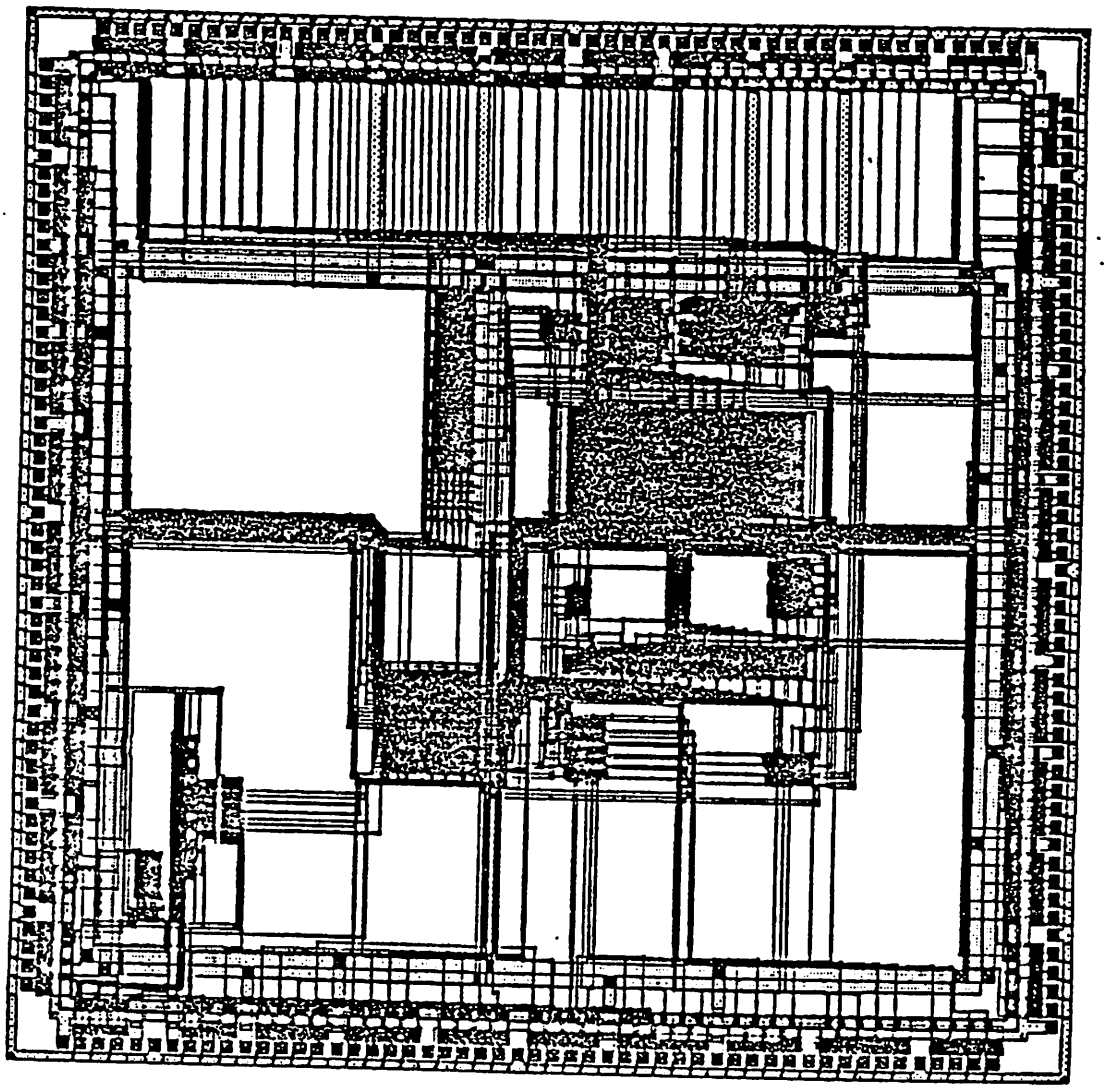


Figure 5.12: Layout of Master Access Controller.

It is fabricated by the MOSIS facilities using the 2um N-well technology. It occupies an area of 10.3 x 10.3 sq. mm and consumes about 0.8 Watts of power.

Chapter 6

Slave Access Controller: SAC

The Slave Access Controller is essentially a slave to the Master Access Controller. It responds to the control signals from MAC and carries out the requested actions. SAC performs actions exclusively on data, transfering them between its four 32-bit buses. Since the chip is very similar to MAC, almost all the circuits used in SAC can be found in MAC.

The functionality and the floorplanning of SAC will be discussed in this chapter. A summary of the pinouts of SAC is given in Appendix C.

6.1 Functional Description

The functionality of SAC will be briefly explained in this section. Figure 6.1 shows the functional block diagram of SAC. It includes a master FIFO and a bypass FIFO, a switch block for the configurable bus and a clock generator.

The master FIFO is a 32 bits wide, 16 words deep first-in-first-out buffer. Its primary function is to provide a temporary storage for the data section of a write request. In case of a read request, dummy data is stored. The read or write operations of the FIFO are controlled respectively by the *masterWr_v* and *masterGrant_c3* signals generated by MAC. No flags are generated by the FIFO since all the status information can be obtained from the master FIFO in MAC. The slave FIFO is very similar to the master FIFO except that it only has a depth of 3 and again there is no flags being generated. Its function is to provide an alternate path for data to go through an open switch. The Switch Block implements the *switches* and the precharge logic of the configurable bus. The opening and closing of the switches are controlled by the *switchSate1* input. The discussion on the clock

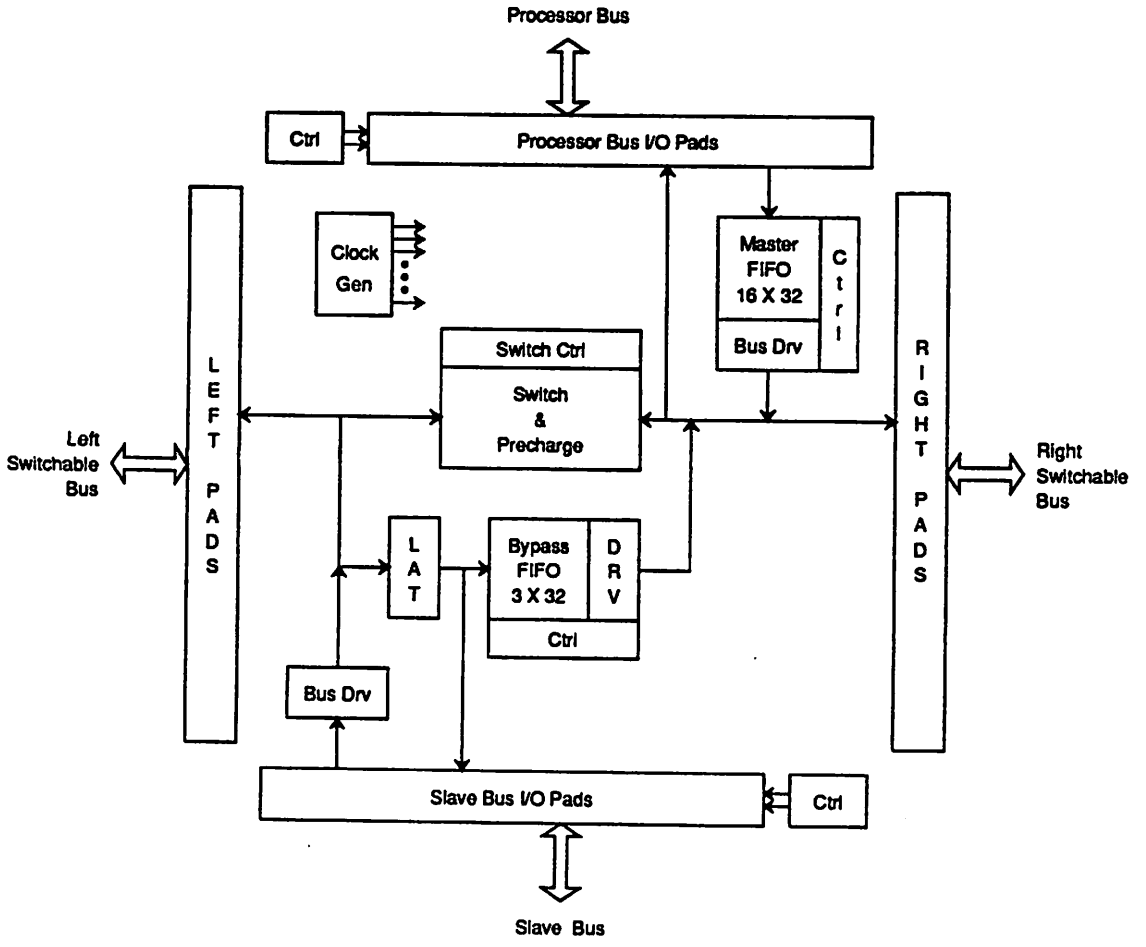


Figure 6.1: Functional Block Diagram of the Slave Access Controller.

generator block will be given in Section 7.1.

6.2 Floorplan and Layout

The layout approach of SAC pretty much follows that of the MAC. The complete layout of SAC is shown in Figure 6.2. Notice that the small control blocks for the FIFO's and the switch block are placed close to the blocks they are controlling. Control signals are delayed, gated with clocks or buffered to increase driving capability in the control blocks. The chip has 208 pins, a total of 21671 transistors, of which 12569 are n-channel mosfets and 9102 p-channel mosfets. It is fabricated by the MOSIS facilities using the 2um N-well technology. It occupies an area of 10.0 x 10.0 sq. mm and consumes about 0.8 Watts of power.

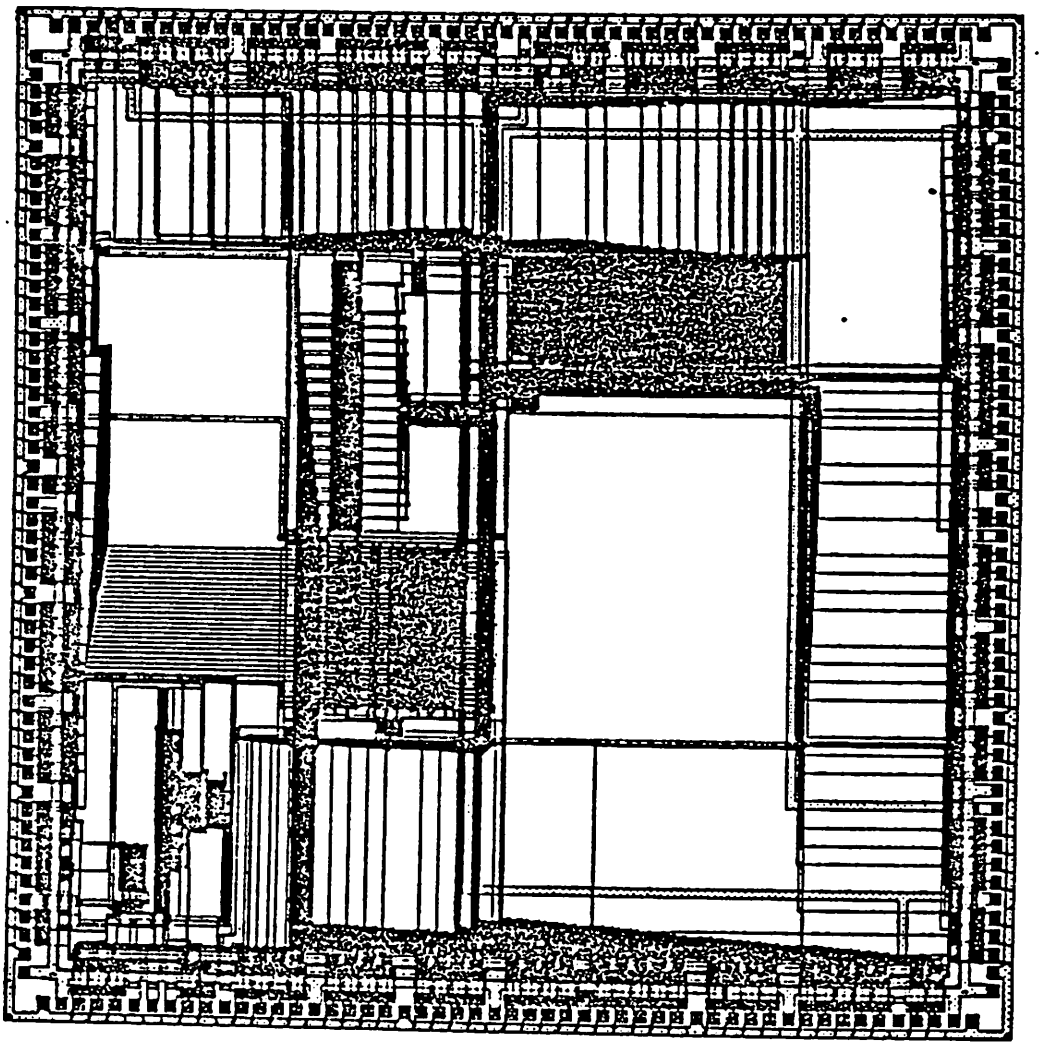


Figure 6.2: Layout of Slave Access Controller.

Chapter 7

Macro Blocks

7.1 Digital Phase Locked Loop

The clock signals used in the Access Controller chip set have been shown in Figure 4.1. The function of the clock generator block is to generate these nonoverlapping clock signals, guaranteeing that the skews between the falling edge of the reference clock and the falling edges of the clock signals on-chip are always constant, as required by the synchronous interfacing scheme used between chips. A simple 4-phase clock generator circuit could have been used but the skews between the reference clock and the internal clocks will vary from chips to chips due to process variations and differences in operating temperature. A design based on *digital phase locked loops* is adopted because of its ability to provide a virtual *zero-delay* clock driver.

In order to save design efforts, a proven design used in the Berkeley SPUR Project[10] is taken with some modifications to suit our needs. Figure 7.1 shows the block diagram of the DPLL clock generator.

Since the clock signals are all derived from the outputs of a ring oscillator, their phase differences are always constant. It is therefore sufficient to maintain phase differences between the reference clock and the internal clocks by *matching* the reference clock with just one of the internal clocks. In our implementation, the falling edge of the signal *phi4* is aligned with that of the reference clock. Because the DSP32C requires an input clock of 40MHz, four times the frequency of the bus cycle, the frequency of the reference clock must be divided by four before feeding into the Phase Frequency Detector. To keep the alignment, a dummy delay matching block is inserted to account for the propagation delay

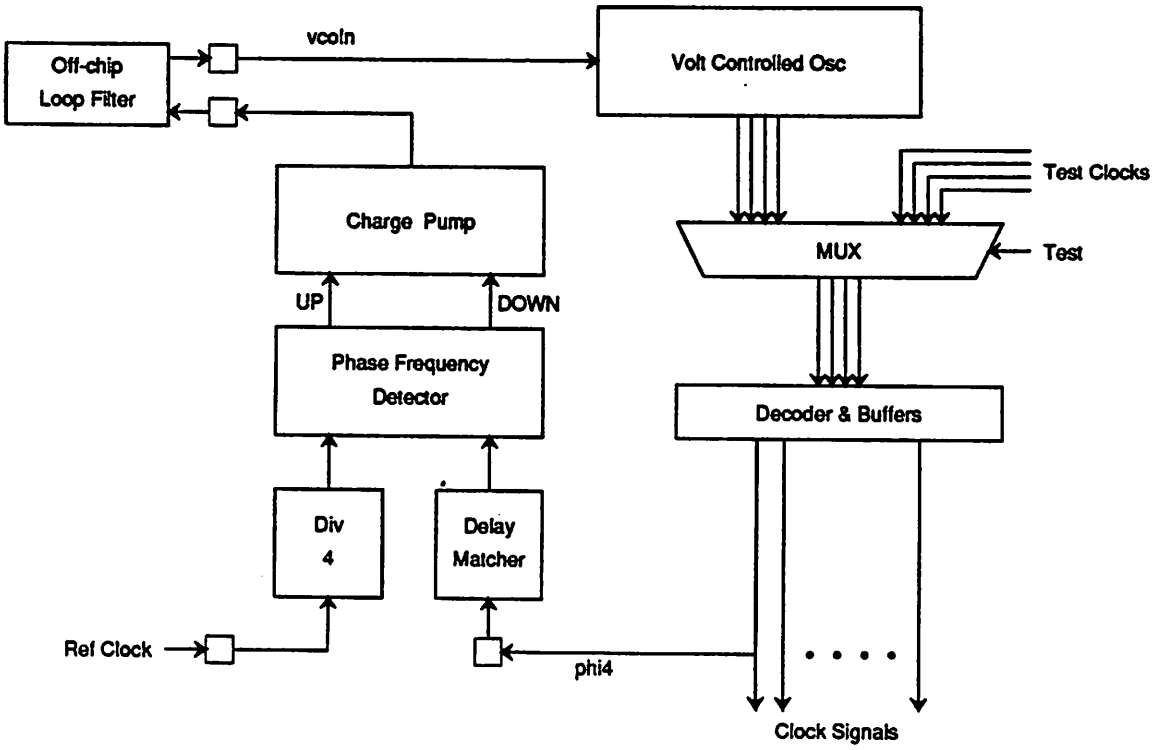


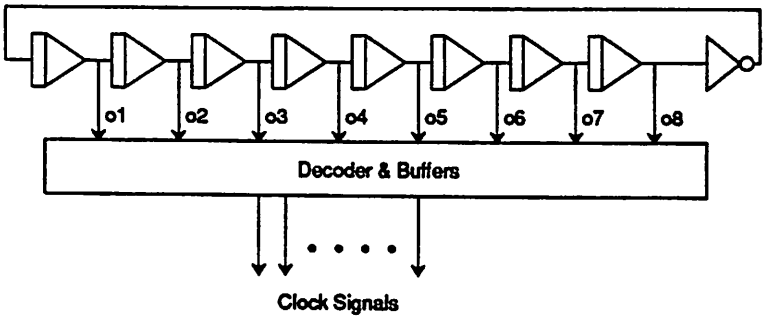
Figure 7.1: Block Diagram of DPLL Clock Generator.

introduced by the divide-by-four circuit. The delay matcher is basically a duplicate of the divide-by-four block (to simulate the same loading on the signal path) except that the input signal is *divided by one.* The propagation delay from the pad through the input buffer to the input of the divide-by-four circuit is also taken into consideration by routing first routing phi4 to a dummy pad with identical input driver before feeding it into the delay matcher.

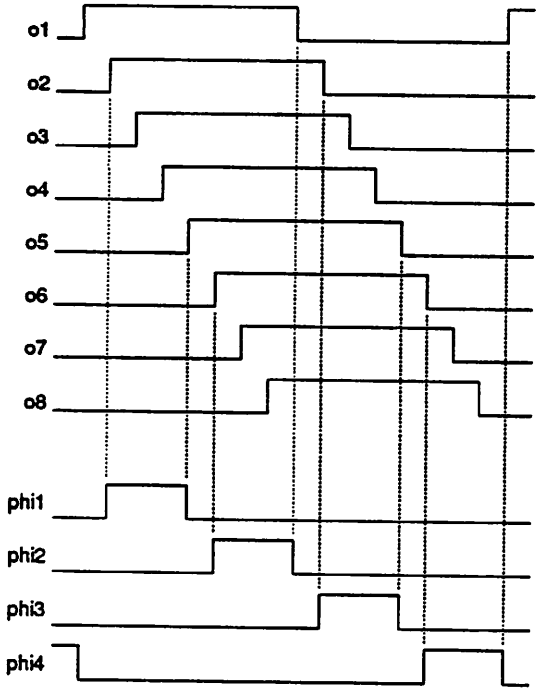
Depending on whether the reference clock *leads* or *lags* phi4, the circuits comprising the phase frequency detector, the charge pump and the loop filter will adjust the delays of the delay cells in the ring oscillator appropriately by raising or lowering the analog input of the oscillator (*vcoIn.* The simplified block diagram of the voltage controlled oscillator (VCO) is shown in Figure 7.2(a). The VCO uses non-inverting delay cells each of which consists of two inverting delay cells used in the original design, to avoid asymmetry due to difference between the *rise delay* and the *fall delay* of the delay cells. Figure 7.2(b) depicts the waveform generation scheme from the outputs of the ring oscillator. Although only four outputs, *o1, o2, o5* and *o6*, are needed to generate the desired clock waveforms, dummy gates are used to ensure the same loadings for each output of the oscillator to improve symmetry of the waveforms. In order to equalize the propagation delays of the decoder for the clock signals as much as possible, the decoder is designed in such a way that each clock signal has to go through the same number of levels of logic. Furthermore, the final buffer stage of each clock signal is sized differently according to the loading extracted from the layout to maintain similar delay for each clock signal.

The oscilloscope traces of the four phase non-overlapping clocks, phi1 to phi4 are displayed in Figure 7.3. The frequency of the reference clock was 30MHz.

To provide a way to test the rest of the circuits on the chip in case the clock generator fails to operate and to allow direct control of the internal clock signals for easy generation of test vectors, a multiplexer is included in the clock generator block so that external test clocks can be *multiplexed in* when the *test* pin is asserted. The four test clocks, corresponding to o1, o2, o5 and o6, are brought onto the chip through bidirectional drivers. When not in the test mode, these drivers are driven by the internal clocks, phi1, phi2, phi3 and phi4 so that they can be observed externally. Performance parameters of DPLL such as jitters and stability can be measured and the loop filter characteristic can be adjusted accordingly.



(a)



(b)

Figure 7.2: Voltage Controlled Oscillator. (a) Simplified block diagram. (b) Waveform generation.

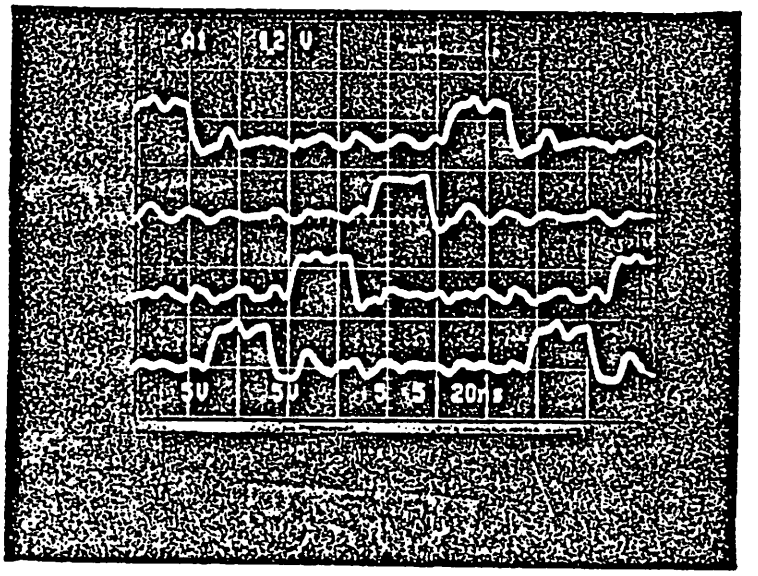


Figure 7.3: Oscilloscope Traces of the Four Phase Non-overlapping Clocks at 30MHz Reference Clock.

7.2 Pads

Since the die size of both MAC and SAC are limited by the large number of pads, it is possible to decrease the overall die size by packing the pads as close to each other as possible, i.e., reducing the width of the pad as much as possible. A new pad library is designed in which the aspect ratio of each pad is such that the width is minimized to 175um, the minimum pad spacing recommended by the MOSIS facility at the time of the design.

Besides the aspect ratio, the new pad library also distinguishes itself from the other pad libraries in LAGER in that an n-channel device instead of a p-channel device is used as the pullup of the last pad buffer stage. Even though the new pads suffer from some degradation in noise margin because the V_{OH} is lowered from 5V to about 3.3V[1], it enjoys many advantages. Lowering the V_{OH} of the driver reduces the power consumed on the external capacitive loads by more than a half. Performing a first order estimate, the amount of off-chip loading is about 5 times the on-chip loading and therefore the power saving can be a significant portion of the total power consumption of the chip.

As far as speed is concerned, a NMOS driver is comparable to a PMOS driver of the same size. Figure 7.4 shows the current driving capabilities versus output voltage for a 200um/2um n-channel pullup and a 200um/2um p-channel pullup. The n-channel device can *source* more current than its p-channel counterpart for output voltage up to about 1.8V. Therefore it is expected the n-channel device can charge up a capacitive load faster initially but beyond some point, the p-channel device is going to catch up. This is confirmed in Figure 7.5 where the transient analysis of the two devices driving a 30pf capacitive load is shown. However, the PMOS driver is still capable of driving more current load than the NMOS driver at high output voltage.

The fact that only big n-channel devices are used in the pad driver means there is no p-well near the large pad driver which generates more substrate current. By keeping the p-well far away, the formation of the parasitic bipolor transistor becomes more difficult, thus improving immunity against CMOS latch-up.

Another consideration in the design of the new pads is the noise injected into the supply lines when the outputs of the pad drivers are switching due to parasitic inductances. To reduce the switching noise, either the size of the output driver can be decreased or the

[1]This figure takes into account the Body Effect of the MOS transistor and depends on the current loading.

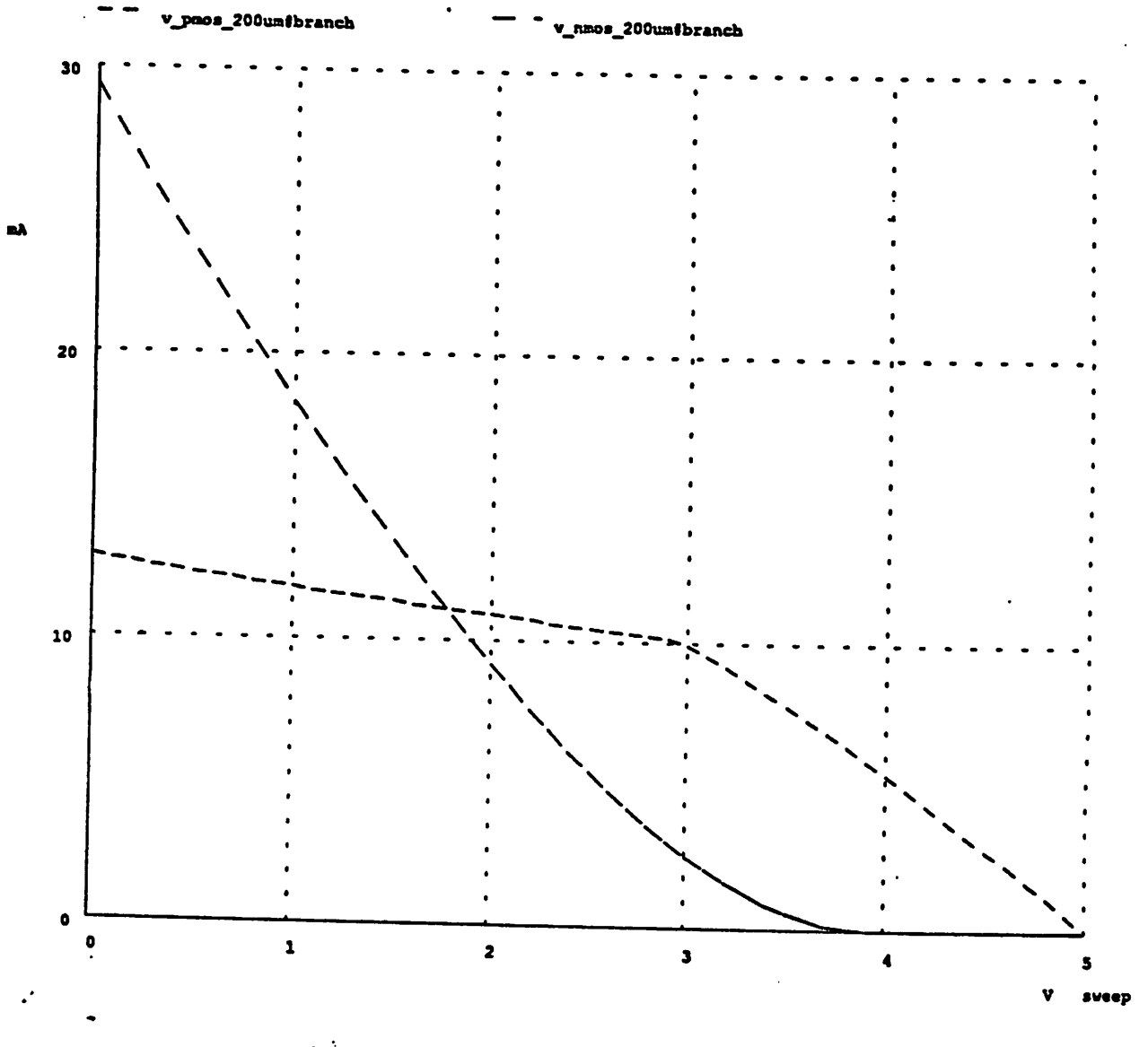


Figure 7.4: Output current driving capabilities versus output voltage for 200um/2um n-channel pullup and 200um/2um p-channel pullup.

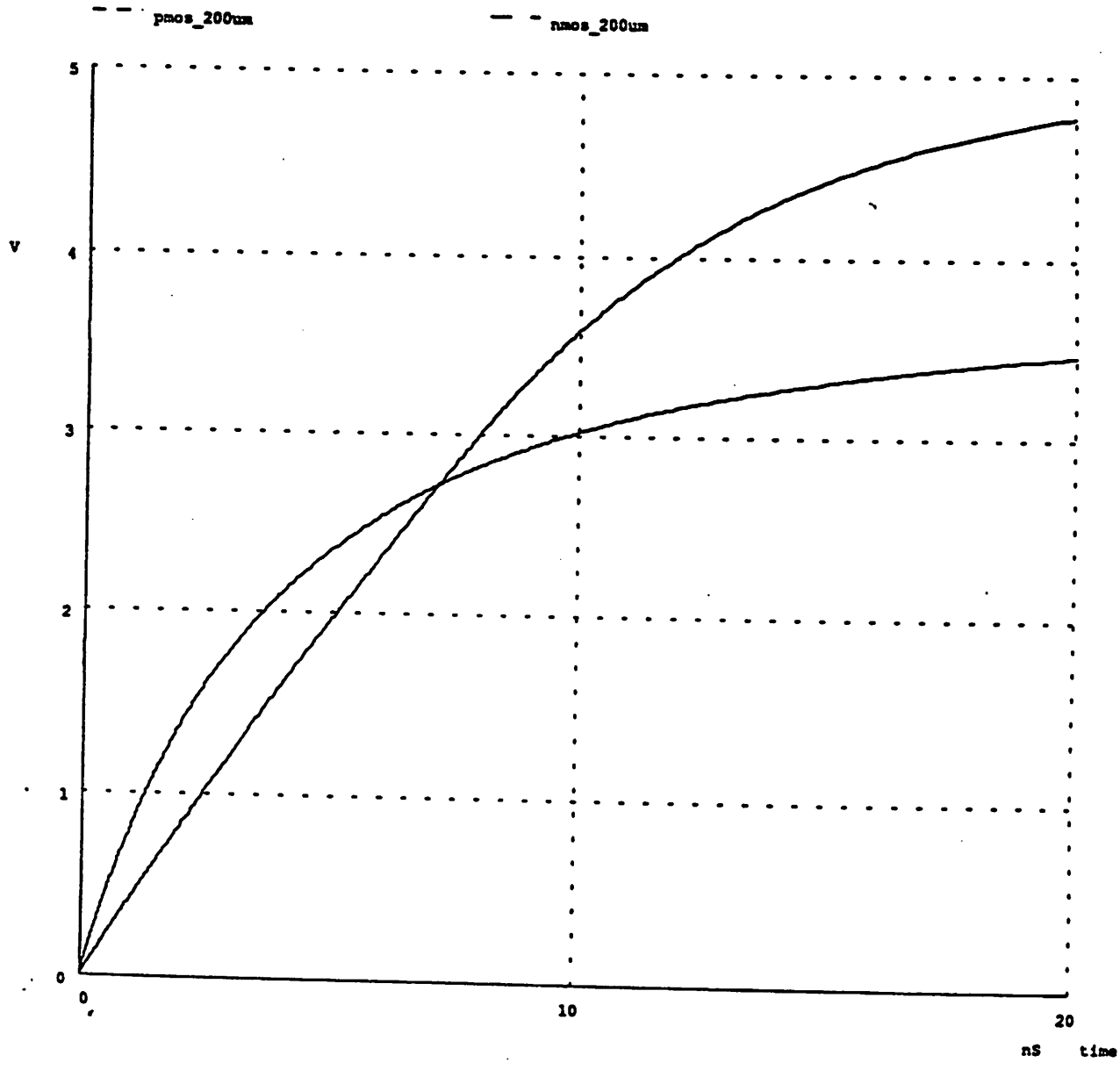


Figure 7.5: Output voltages for 200um/2um n-channel pullup and 200um/2um p-channel pullup vs time. A static load of 30pf is used.

strength of the predriver can be reduced so that the large driver transistor is turned on slowly. However, both techniques increase the propagation delay of the pads. With the aide of SPICE simulations, the driver sizes are adusted to compromise these conflicting requirements such that the magnitudes of the switching noise on the power lines are kept below an arbitrarily chosen level of 0.5V with as little sacrifice in speed as possible. The parasitic inductances used in SPICE simulations are modelled by a 50nH inductor between the ideal voltage supplies and the supplies to the pad driver, assuming one pair of power supply pads are available for five output pads[6].

A second version of the pad library is designed to implement boundary scan registers. More discussion on boundary scan testing will be given in Section 8.3.

The pad library includes input pad, inverted output pad, non-inverted output pad, bidirectional pad and power pads. The circuit diagrams of all the pads except the power pads are included in Appendix D. The size of each pad is 628um x 175um without boundary scan registers and 750um x 175um with boundary scan registers. Spice simulations show that the output pad and the bidirectional pad exhibit 6ns delay for a 50pf static load and the input pad has 2ns delay for a 0.5pf static load using typical process parameters of the MOSIS 2um N-well technology.

7.3 FIFO

The FIFO blocks are the key components of the Access Controller. Since the interfaces with the processor and the switchable bus are both synchronous, it is sufficient to design a synchronous FIFO that allows simultaneous read and write operations. The decision, however, was to design a more general asynchronous FIFO so that it can be reused in other applications in the future. The design specification of the FIFO is that the read and write port of the FIFO must be able to operate independently and asynchronously at 20MHz. This translates to two read operations and/or two write operations in one 100ns bus cycle. (But only one read operation is performed in our design.) It's capacity is 32 bits width and 16 words deep.

The organization of the FIFO is shown in Figure 7.6. The FIFO can be divided into 2 main sections, the data section and the control section. The data section consists of input data buffer(IBUF), cell array and output latches(OLAT), the controls of which are generated in the control section. The control section comprises the read pointer(RPTR),

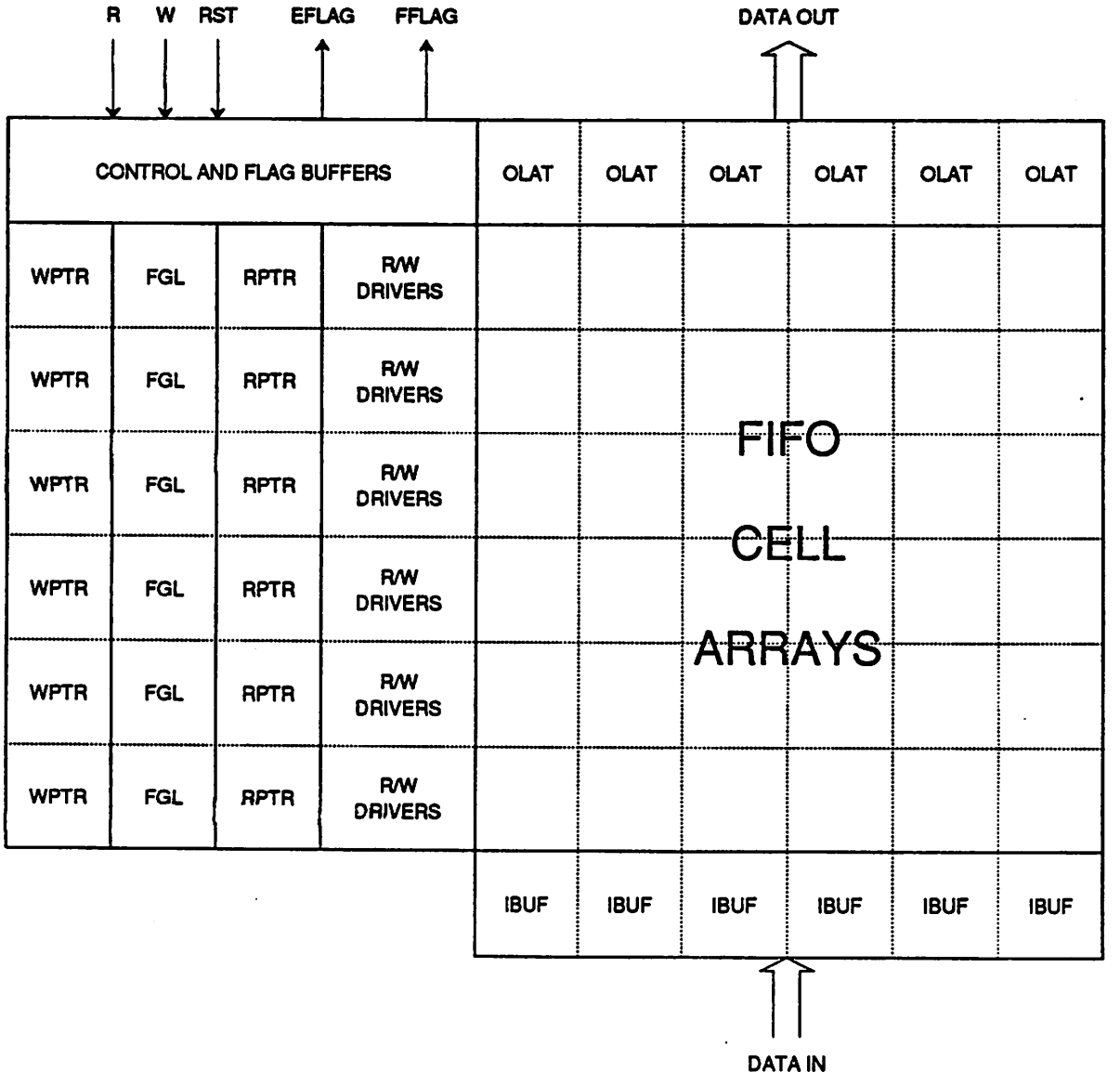


Figure 7.6: Organization of the FIFO.

the write pointer(WPTR), the drivers for the control signals(R/W DRVIERS), the buffers for input control signals and output flags and the flag generation logic(FGL). The read or write pointers are implemented as circular shift registers. After reset, only one register in the shift register is initialized to *One* and the others are all cleared. The word to be read or written is pointed to by the presetted register. After every read/write operation, the corresponding shift register or pointer is shifted to point to the location of the next word. The R/W Driver block and the buffer block serve to provide more driving capabilities for signals and generate the inverse of some signals (such as *WB* in Figure 7.7). The flag generation logic compares the locations pointed to by the read and write pointers to decide the status the FIFO. Besides input data and output data, the FIFO takes 3 control inputs: read, write and reset, and generates 2 output flags. More about flag generation will be discussed below.

Figure 7.7 shows a simplified logic diagram of the FIFO. The data storage cell in the array is implemented as simple static latch with tristate output. The write pointer is *anded* with the write enable signal to generate the latch enable signal for the cells to avoid problems with data hold time and overwriting data stored in the next write location. Figure 7.8 gives a more detailed look at the read/write pointers and the flag generation logic. Each edge-triggered flip flop in the shift register is implemented by two complementary setable/resetable half latches enabled by the read/write signal. Each flag is generated by a *large* and-or gate implemented using pseudo-static circuit. A pair of latches, one from the read pointer and the other from the write pointer, is compared for each word. If the contents of any one pair of latches are both set, the flag is asserted. The size of the p-channel device is kept as small as possible as long as the propagation delay for disasserting the flag is not a problem. The small size enhances noise margin, reduces static power and propagation delay when asserting the flag. To speedup flag generation, we can use some current sense amplifier scheme. But the design usually requires voltage reference and consumes static power.

By picking a different pair of latches from each pointer for comparison, the timing of the flag can be modified to suit our needs. Two examples are given in Figure 7.9 to illustrate this point. The two columns of 1's and 0's represent the contents of the half latches in the two shift registers. As explained before, there will be only two 1's left in each column after initial reset. At a low to high transition of the write(read) signal, the 1's in the write(read) column will shift down 1 notch. The same will happen at a high to

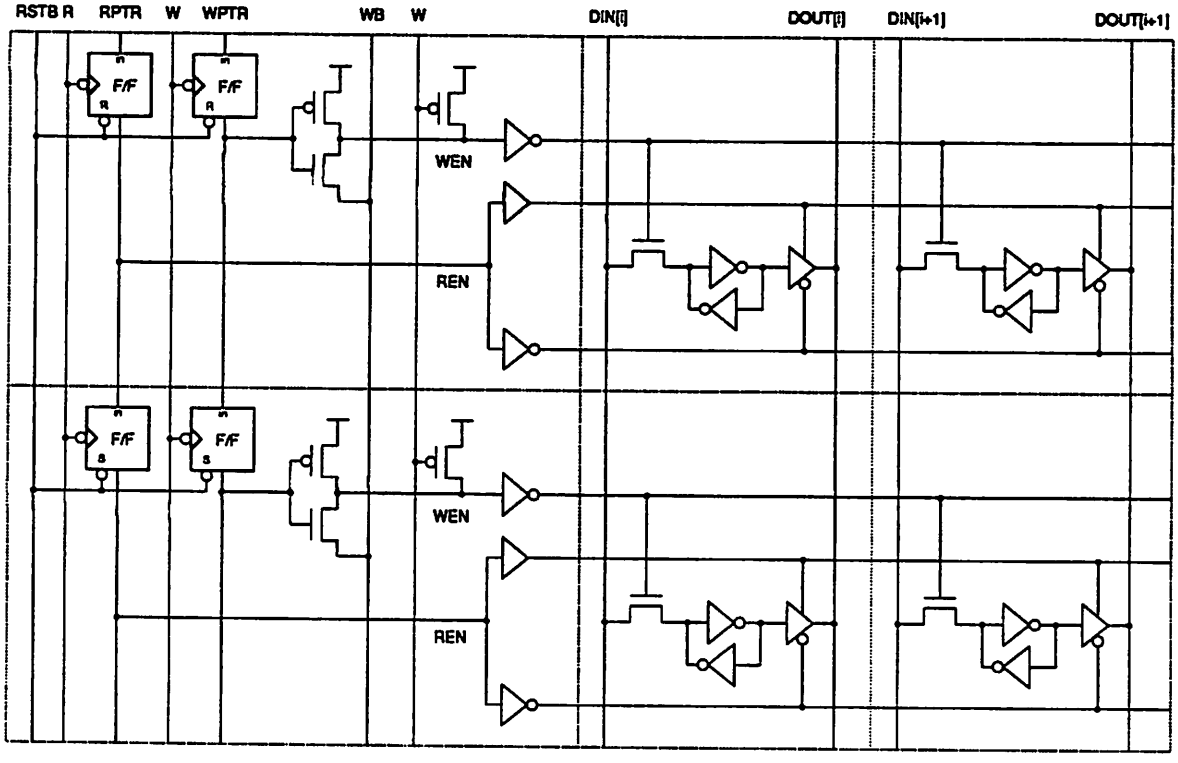


Figure 7.7: Simplified Logic Diagram of the FIFO.

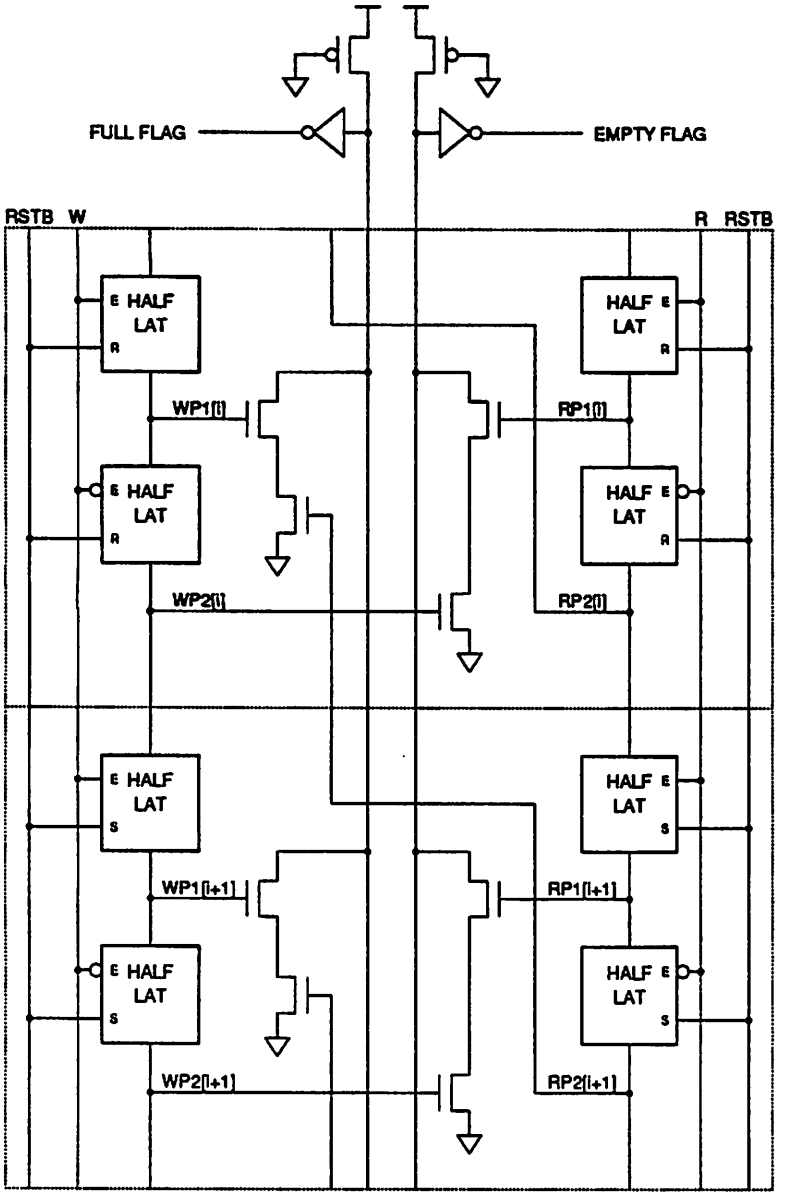


Figure 7.8: Flag Generation Logic of the FIFO.

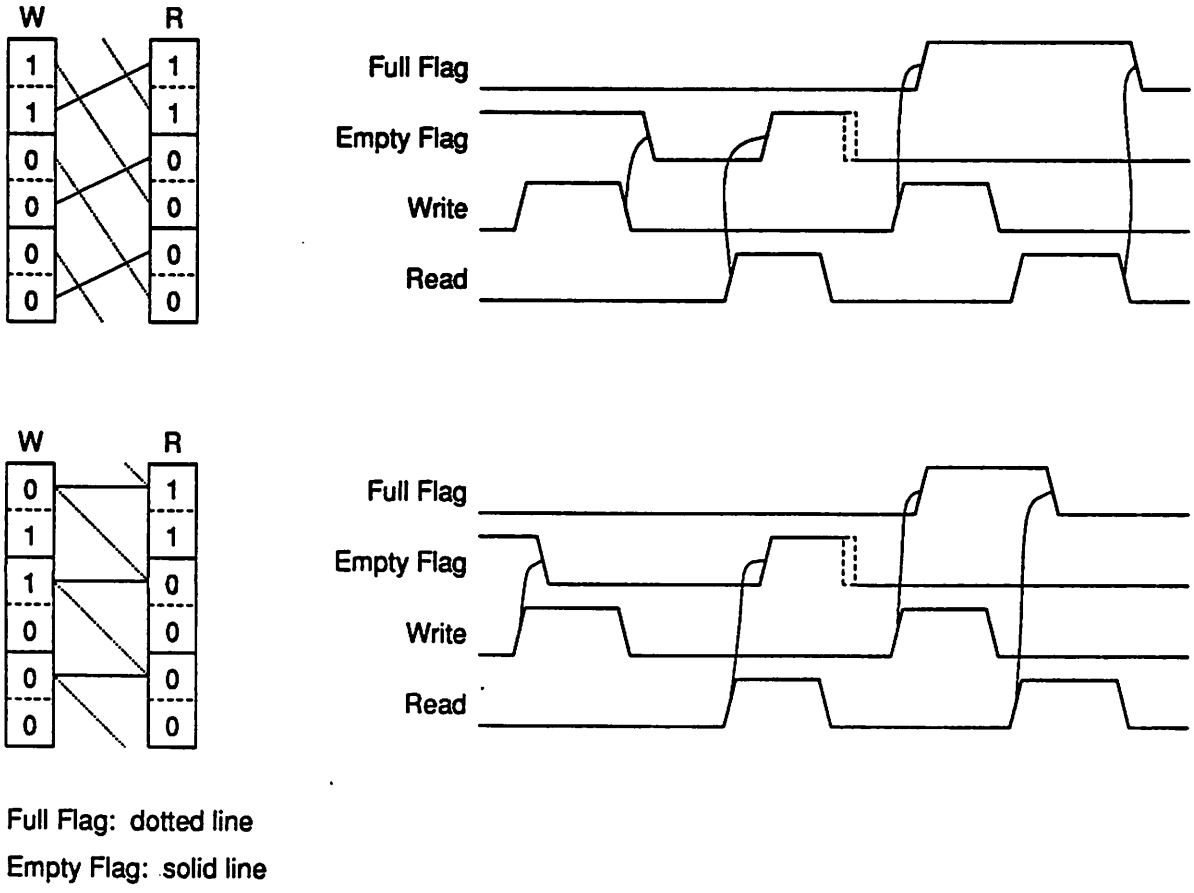


Figure 7.9: Variations of Flag Timing.

low transition. Thus the 1's will shift down two notches (pointing to the next word) after the completion of a write(read) operation. The lines between the columns associate one latch from each column for comparison. The dotted line and the solid line represent the full flag and the empty flag respectively. The timing diagrams show the flag timing with respect to the events that trigger the assertion or disassertion of the flag for the two different comparison scheme. After reset, both the read and write pointers point to the same word and there is no way to distinguish a completely empty FIFO from a completely filled FIFO with combinational logic only. To avoid adding extra logic, the full flag is actually asserted when there is still one empty slot in the FIFO.

By comparing different pairs of latches, it is possible to *program* the flags. For the master FIFO in MAC, a *one* flag is actually generated which is active when there is only one word left in the FIFO. Carrying the idea one step further, any arbitrary flag can be generated if extra shift registers are provided. For instance, a *half full* flag can be implemented by having a shift register with half of its registers presetted.

Chapter 8

Verification and Testing

8.1 Logic Verification with Thor

To assist the task of logic verification, a high level behavioral model of the SMART system was created. Components of the system, including processor, memories and the Access Controller, are described in a high level language called the *C Hardware Description Language (CHDL)*. These models can then be connected together to form the complete system using a *component-oriented* net list language called *CSL*. The entire model can be simulated with a event-driven functional simulator called *THOR*. For our purpose it is enough to abstract the processor as a component that carries out memory transactions to/from some specified memory locations at specified time. Despite its simplicity, the processor model allows us to write very high level commands, such as broadcast write to processor n, to exercise the various functional blocks of the Access Controller.

A cross-reference facility is available in THOR so that signal values in the model can be used as stimuli for an event-driven logic-level simulator called *irsim*. Irsim simulates the actual circuits based on netlist information directly extracted from the physical layout. The *response* of the circuits can again be compared with the corresponding signals in the model to verify functionality of the circuits. Using high level commands for the processor model, simulation patterns can be conveniently generated.

8.2 Testing Strategy

Since many testing strategy such as scan test or built-in self test, requires special hardware support on-chip, it is very important to decide on a testing strategy in the early phase of the design cycle. After close examination of the Access Controller design, we came to the conclusion that the conventional IO testing approach, driving and observing the external pins of the chip, is sufficient. Not only is this approach exempted from the usual hardware overheads and the resulting performance degradation associated with the other testing methods, the approach is also competitive in terms of testability. The reason for that is the high controllability and observability of the Access Controller chip set, as hinted by the large pin counts of the chips. First of all, the chips have relatively few *internal states*, most of which (such as the configuration registers) can be accessed directly by executing the appropriate configuration instructions. Moreover the status of these states can be easily observed at the pins. In addition, there is no large finite state machine type control logic on chip. Control flow is direct and can be monitored easily. Therefore, it is believed that very high fault coverage can still be achieved with a reasonable number of test patterns using this approach.

Test vectors were generated painlessly by writing the high level commands for the processors. Programs were written to automate the process of extracting the test vectors from the simulation patterns, downloading the vectors to the tester, and uploading and verifying the acquired results from the tester after the test is performed.

Unfortunately when it comes to the actual testing of the chips with the tester, the problem lies not in the quality of the test vectors but in the testing setup. Experience acquired during the testing of the first version of the chips revealed that it is very difficult for the tester to access so many pins without some adapter board for the package we are using. It was therefore decided to implement a boundary scan register chain in the pads for the second version of the design. The boundary scan design allows controlling and observing of all the pins with access to only a few pins. Inputs to the chips can be scanned into the chips. The chip is then allowed to run for *one cycle* by advancing the clocks(external test clocks). Outputs of the chips can be latched into the chain and observed externally. By stretching out the test in time, simpler test setup can be used. Furthermore the boundary scan is very useful for debugging interconnections and on-board testing on the printed circuit board. More detailed discussion of the boundary scan scheme will be given in the next section.

8.3 Boundary Scan

A complete boundary scan design should provide the following functions:

Controllable Inputs: Inputs to the chips are controllable externally by *scanning in* the required patterns.

Observable Outputs: Ouputs from the chips are observable externally by *scanning out* the latched chip outputs.

Observable Inputs: Inputs to the chips from other part of the system are observable to verify interconnections.

Controllable Outputs: Outputs of the chips can be controlled by *scanning in* the required patterns.

The first two functions allow isolation of a chip on the board from the rest of the system so that on-board testing can be carried out. The last two capabilities enable direct checking of interconnection on the printed circuit board as well as provide some testing capability for some chips not equipped with boundary scan. The later is accomplished by controlling the outputs and monitoring the inputs of the chips that interface with the chip to be tested. No assumptions on the functionality of the interfacing chips are necessary. In our design, the last function was not implemented to simplify the design. Instead, our custom chips will be tested before they are put on the board so that outputs of the chips can be controlled indirectly by applying test vectors that will result in the desired output patterns.

Extra hardware including three simple latches and a multiplexer is required to implement the boundary scan scheme as shown in the circuit diagram of an input/output pad with boundary scan registers (Figure 8.1). Latch1 and Latch2, enabled by the non-overlapping clocks *mck* and *sck* respectively, formed the master-slave scan register of the scan chain. The *state* of the bonding pad can be loaded into the scan register can enabling the signal *ldpad*. Latch3 is responsible for holding the inputs to the chip steady while scan operation is carried out on the scan chain. Thus inputs to the chips are *updated* only when *ldIn* is active. The *scanEnB* signal decides whether inputs to the chip should come from outside (bonding pad) or from latch3. The multiplexer introduces less than 1ns of extra

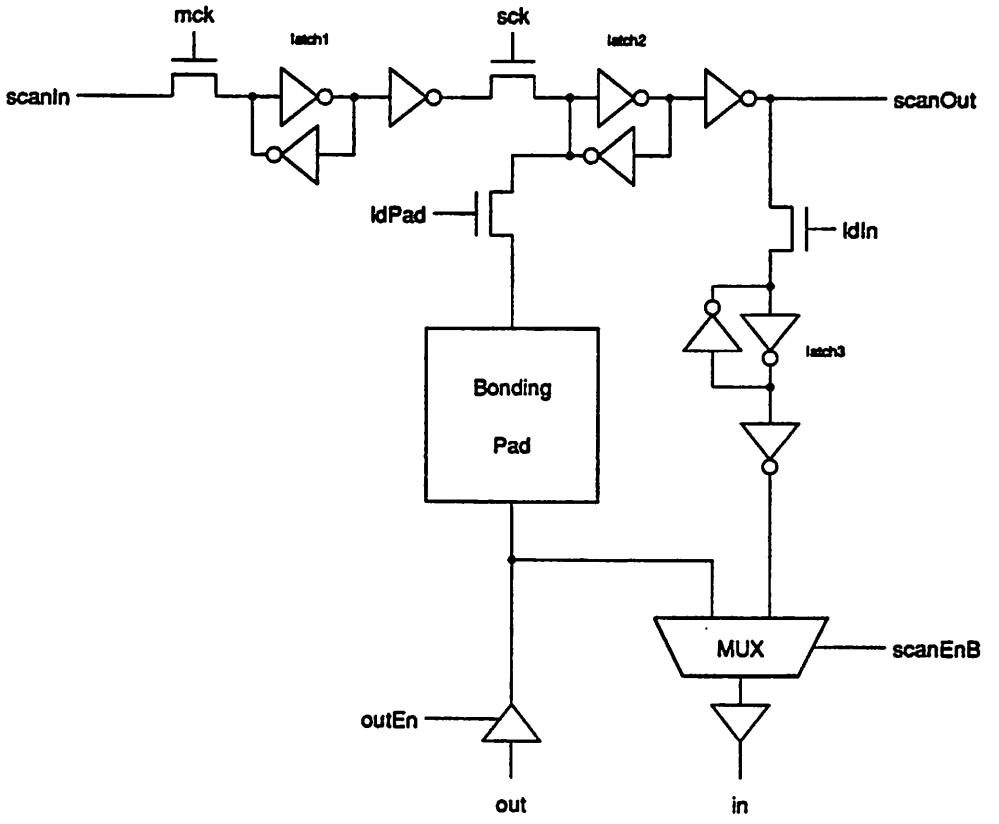


Figure 8.1: Circuit Diagram of an I/O Pad with Boundary Scan Registers.

propagation delay to the input signal path. The extra scan logic increases the chip sizes by about 100um on each side, resulting in a 2% area overhead over the non-scan designs.

Chapter 9

Conclusions

9.1 Summary

The key architectural features of SMART, namely the *Configurable Bus*, the *Distributed Shared Memory*, the *Write Queues* and the *Synchronization Mechanism*, have discussed in this report. By trading off overall communication bandwidth and latency, the SMART system can be configured to achieve high throughput by exploiting concurrency existed in a large variety of algorithms. The system performance is further improved by overlapping the overhead due to communication latency with the computation time of the processors.

The design of the SMART prototype system was presented, emphasizing the design issues of the Access Controller, a custom VLSI chip set implemented to support the forementioned architectural features. By defining the design goals and taking into considerations the various design issues discussed in Chapter 4 early on, we were able to look ahead for potential problems so that measures could be taken to avoid them. Consequently the design process was very *smooth* and the design was completed with very few iterations and modifications. The design of the chip set took about one and a half man-year.

Perhaps the most difficult design decision we had to make was whether to go for the dynamic switchable bus design. On one hand we were excited about the high speed and pin-saving advantages of the design. On the other hand we were concerned about the *robustness* of the design in a noisy environment. Subsequent testing of the design proved that the design was unreliable in the presence of noise and we had to switch to a static design. The final switchable bus design measured a pad-to-pad propagation delay of about

6.8ns. This translates to the capability of rippling data through the closed switches of 8 processors in a single bus group in one 10MHz bus cycle.

Another valuable lesson learned in this project is that every precaution must be taken in the design of the pad drivers to reduce switching noise to an acceptable level, especially in systems where several wide buses may be switching simultaneously.

The availability of a library of CAD tools plays a very important role in speeding up the design process. The LAGER tool sets allow generation of layouts of different styles using a uniform input format. By using a bottom-up approach in layout, blocks of layout can be generated, simulated and modified very quickly with the aid of the LAGER tools. The experience acquired in the process of designing the chips suggest that it is very important to understand the characteristics and limitations of the CAD tools used because very often the design may have to be modified to adapt to the tools. For instance, the simulated annealing algorithm used in the standard cell place-and-route tools is not deterministic and that means the resulting layout generated may vary somewhat from run to run. Therefore it is important to verify any critical paths that may be affected by the layout even though only minor modification is made to the design.

The decision to remove critical paths by redesigning the logic instead of redesigning the circuits proved to be advantageous despite some area overhead. In general there is more confidence in the functionality of the circuits when critical paths are eliminated by reducing the number of levels of logic rather than resizing the transistors. Layout modification with logic redesign only involves changing information about gate connectivities but resizing generally requires laying out new cells. This approach greatly reduces the number of SPICE runs for the entire design. SPICE simulations were needed only for the switchable bus, the FIFO and the pads.

In retrospect the exclusion of scan registers from the design does not seem to make it difficult for test vector generation, nor does it degrade the quality of the test vectors because of the high controllability and observability of the internal nodes in the chips. However, implementation of boundary scan registers is beneficial because it reduces the number of pins that the tester has to access for testing. It would have saved a lot of time in developing the wire-wrapped test board used for interfacing the chips to the tester. While designing scan registers is well supported by the CAD tools, wire wrapping is a manual and tedious (if not impossible) task.

9.2 Future Work

A lot of interesting ideas and alternatives can be suggested to improve the performance of the SMART system.

On implementation issues, it is worthwhile to look into some advance packaging techniques such as *Wafer Scale Integration (WSI)* and *Hybrid WSI*[7] which can improve both the board density and system performances. These techniques also provide new options for system partitioning. It is also interesting to explore some high bandwidth asynchronous bus designs and their impacts on system extensibility.

On architecture, more studies can be done on the communication patterns of DSP algorithms to derive some interconnection schemes that are more general and flexible. Other issues include memory hierarchy, heterogeneous processing elements and self-configuration of buses.

Bibliography

[1] AT&T. *WE DSP32C Digital Signal Processor, Advance Data Sheet*, May 1989.

[2] Electronic Research Laboratory, University of California, Berkeley. *LagerIV Silicon Assembly System Manual*, Distribution 1.0 edition, October 1985.

[3] R. J. Sluyter et. al. A novel method for pitch extraction from speech and a hardware model applicable to vocoder systems. In *International Conference on Acoustics, Speech and signal Processing*, pages pp. 45–48. IEEE, 1980.

[4] W. Koh A. Yeung P. Hoang J. Rabaey. A multiprocessor system for dsp behavioral simulation. In *IEEE Workshop on VLSI Signal Processing, III.* IEEE, 1988.

[5] W. Koh A. Yeung P. Hoang J. Rabaey. A configurable multiprocessor system for dsp behavioral simulation. In *Proceedings of International Symposium on Circuits and Systems.* IEEE, 1989.

[6] J. Keller. Power and ground requirements for a high-speed 32 bit computer chip set. Technical report, University of California, Berkeley, August 1985.

[7] S. K. Tewksbury L. A. Hornak. Wafer level system integration: A review. *IEEE Circuits and Devices Magazine*, September 1989.

[8] Motorola. *MECL System Design Handbook*, Rev 1 edition, 1988.

[9] PRINTEX. *The VMEbus Specification*, Rev C.1 edition, October 1985.

[10] D. K. Jeong G. Borriello D. A. Hodges R. H. Katz. Design of pll-based clock generation circuits. *Journal of Solid-State Circuits*, April 1987.

Appendix A

MAC Instructions

DSP32C issues instructions to MAC by performing external memory accesses. MAC figures out the instruction by decoding the address bus according to an address map.

In DSP32C, the entire address space is divided into 2 banks: bank 0 and bank 1. In one instruction, concurrent access to two different memory banks is allowed. In SMART, the Mode 7 Memory Configuration (ROM-less version) is chosen (Table A.1). Since access to the on-chip RAM's is internal to the DSP32C, this kind of access does not result in issuing of instructions to the MAC.

As explained in Section 5.1, instructions of MAC can be broadly categorized into two groups. Instructions in Group 1 are all mapped to external memory A and instructions in Group 2 to external memory B. Address mapping for the entire address space is shown in Table A.2, Table A.3 and Table A.4. Adr[12-0] is the external word address bus and Msn[3-0] is the byte select of DSP32C. By asserting one or more of the byte select bits, DSP32C can selectively access one or more bytes of the 32-bit word. All instructions in Group 1 are located in Table A.2 and instructions in Group 2 in Table A.3. Table A.4 shows address mapping for memory bank 1. All accesses to this memory bank are internal to DSP32C.

Memory Bank	Byte Address	Memory Assignment
BANK 0	0x000000 0x0007FF	RAM0
	0x000800 0x5FFFFF	External Memory A
	0x600000 0xFFDFFF	External Memory B
BANK 1	0xFFE000 0xFFE7FF	RAM2
	0xFFE800 0xFFF7FF	RESERVED
	0xFFF800 0xFFFFFF	RAM1

Table A.1: DSP32C Memory Configuration, Mode 7 (ROM-less version).

Following is a description of Group 1 instructions (Without Acknowledge):

1. External Local Memory Read/Write (lmemRd/lmemWr):
   MAC responds by asserting chip enable signals for the local memory (LMEM). Adr[12-0] and Msn[3-0] specify the address at which the data are to be read or written.

2. Synchronization Switch Write (wrSyncSwitch):
   The lower order 5 bits of DSP32C address bus are written into the *syncSwitch* configuration registers which controls the switches on the synchronization bus.

3. Synchronization Disable Write (wrSyncDisable):
   The lower order 6 bits of DSP32C address bus are written into the *syncDisable* configuration registers which enable or disable the corresponding synchronization patterns.

4. Semaphore Unlock (semV):
   MAC releases the semaphore.

5. Switch1 Write (wrSwitch):
   Bit 11 of the DSP32C address bus is written into the *switchState* configuration register which controls the switches on the configurable bus.

6. Swap State Write (wrSwap):
   Bit 11 of the DSP32C address bus is written into the *swapState* configuration register which controls the bank select of DPRAM access.

7. Swap State Toggle (toggleSwap):
   Toggle the *swapState* configuration register.

	DSP32C Word Address - Byte Select A A A A A A A A A A A A A A A A A A A A A A M M M M d d d d d d d d d d d d d d d d d d d d d d s s s s r r r r r r r r r r r r r r r r r r r r r r n n n n 2 2 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0-3 2 1 0 1 0 9 8 7 6 5 4 3 2 1 0
0x000000 Start	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
On-chip Ram (2KB)	
0x0007FF End	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1-1 1 1 1
0x000800 Start	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0-0 0 0 0
External Local Memory (256KB)	
0x0407FF End	0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1-1 1 1 1
0x040800 Start	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Reserved for Future Expansion of Local Memory (254KB)	
0x07FFFF End	0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1
0x080000 Start	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Special Instructions (I)	
wrSyncSwitch	0 0 0 0 1 0 0 x x x x x x x x x x u u u u u-x x x x
wrSyncDefault	0 0 0 0 1 0 1 0 x x x x x x x x u u u u u u-x x x x
semV	0 0 0 0 1 0 1 1 0 0 x x x x x x x x x x x x-x x x x
wrSwitch	0 0 0 0 1 0 1 1 0 1 u x x x x x x x x x x x-x x x x
wrSwap	0 0 0 0 1 0 1 1 1 0 u x x x x x x x x x x x-x x x x
wrSwapToggle	0 0 0 0 1 0 1 1 1 1 x x x x x x x x x x x x-x x x x
wrByPassEn	0 0 0 0 1 1 0 0 0 0 u x x x x x x x x x x x-x x x x
wrSwitch2	0 0 0 0 1 1 0 0 0 1 u x x x x x x x x x x x-x x x x
wrProcNum	0 0 0 0 1 1 0 0 1 0 x x x x x x u u u u u u-x x x x
decode	0 0 0 0 1 1 0 0 1 1 x x x x x x x x x x x x-x x x x
0x0FFFFF End	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1
0x100000 Start	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Front-side Memory Access (Read or Write)	
	0 0 0 1 x x x x x s s A A A A A A A A A A A-A A A A
0x1FFFFF End	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1
0x200000 Start	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Bus-side Memory Broadcast (Write Only)	
	0 0 1 p p p p p p s s A A A A A A A A A A A-A A A A
0x3FFFFF End	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1
0x400000 Start	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Bus-side Memory Access (Write Only)	
	0 1 0 p p p p p p s s A A A A A A A A A A A-A A A A
0x5FFFFF End	0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1

LEGENDS:
- x: Don't Care
- u: User Programmed Bits
- s: Dual-port Memory Bank Select Bits
- A: Byte Address of Dual-port Memory
- p: Processor Id of Destination Processor

Table A.2: Address Mapping for Memory Bank 0 (Group 1 Instructions).

	DSP32C Word Address - Byte Select
	A A A A A A A A A A A A A A A A A A A A A A M M M M
	d d d d d d d d d d d d d d d d d d d d d d s s s s
	r r r r r r r r r r r r r r r r r r r r r r n n n n
	2 2 1 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0-3 2 1 0
	1 0 9 8 7 6 5 4 3 2 1 0
0x600000 Start	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
BusSide Memory on Bus Side with Wait	
	0 1 1 p p p p p p s s A A A A A A A A A A A-A A A A
0x7FFFFF End	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1
0x800000 Start	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Special Instructions (II)	
synchronization	1 0 0 0 x x x 0 0 0 0 0 0 0 0 0 u u u u u u-x x x x
semP	1 0 0 0 x x x 1 x x x x x x x x x x x x x x-x x x x
0x9FFFFF End	1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1
0xA00000 Start	1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Broadcast with Wait	
	1 0 1 p p p p p p s s A A A A A A A A A A A-A A A A
0xBFFFFF End	1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1
0xC00000 Start	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Circular ByPass with Wait	
	1 1 0 p p p p p p s s A A A A A A A A A A A-A A A A
0xDFFFFF End	1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1
0xE00000 Start	1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Fifo Read/Write with Wait	
0xFFDFFF End	1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1

LEGENDS:
- x: Don't Care
- u: User Programmed Bits
- s: Dual-port Memory Bank Select Bits
- A: Byte Address of Dual-port Memory
- p: Processor Id of Destination Processor

Table A.3: Address Mapping for Memory Bank 0 (Group 2 Instructions).

	DSP32C Word Address - Byte Select A A A A A A A A A A A A A A A A A A A A A A M M M M d d d d d d d d d d d d d d d d d d d d d d s s s s r r r r r r r r r r r r r r r r r r r r r r n n n n 2 2 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0-3 2 1 0 1 0 9 8 7 6 5 4 3 2 1 0
0xFFE000 Start	1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0-0 0 0 0
On-chip Ram (2KB)	
0xFFE7FF End	1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1-1 1 1 1
0xFFE800 Start	1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0-0 0 0 0
Reserved (4KB)	
0xFFF7FF End	1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1-1 1 1 1
0xFFF800 Start	1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0-0 0 0 0
On-chip Ram (2KB)	
0xFFFFFF End	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1-1 1 1 1

Table A.4: Address Mapping for Memory Bank 1 (Internal Access Only).

8. Bypass Enable Write (wrBypassEn):
   Bit 11 of the DSP32C address bus is written into the *bypassEn* configuration register which enables bypass operation.

9. Switch2 Write (wrSwitch2):
   Bit 11 of the DSP32C address bus is written into the *switchState2* configuration register.

10. PID Write (wrPID):
    The lower order 6 bits of DSP32C address bus are written into the *pidReg* configuration registers which contains the Processor Identification Number of local processor.

11. Interrupt Generate (genIntr):
    The single-phase positive pulse is generated at the *interrupt* output of MAC. This signal can be used to generate an interrupt to the VME interface, thus providing the DSP32C the ability to interrupt the VME bus master.

12. Processor-port DPRAM Read/Write (ppmemRd/ppmemWr):
    MAC responds by asserting chip enable signals for the processor-port of the dual-ported RAM (DPRAM).
    Adr[12-11] together with the swapState specify the memory bank to be accessed. Adr[10-0] and Msn[3-0] specify the address at which the data are to be read or written.

13. Slave-port DPRAM Broadcast Write (spmemBcWr):
    A write request is broadcasted to the slave-ports of groups of processors specified by the pid of the destination processor given in Adr[18-13]. Adr[12-11] together with the

swapState specify the memory bank to be accessed. Adr[10-0] and Msn[3-0] specify the address at which the data are to be read or written.

14. Slave-port DPRAM Write (spmemWr):
A write request is made to the slave-port of a destination processor whose pid is given by Adr[18-13]. Adr[12-11] together with the swapState specify the memory bank to be accessed. Adr[10-0] and Msn[3-0] specify the address at which the data are to be read or written.

Following is a description of Group 2 instructions (With Acknowledge):

1. Slave-port DPRAM Slow Read/Write (spmemSWr/spmemSRd):
Same as spmemWr except that MAC generates an acknowledge and both read and write requests are allowed.

2. Synchronization (syncV):
Any *one's* in the lower order 6 bits of DSP32C address bus activate the corresponding synchronization patterns. MAC generates an acknowledge when every member of the processor set defined by the activated synchronization pattern is *synchronized.*

3. Semaphore Request (semP):
MAC makes a request to acquire the semaphore. After the semaphore is obtained, the semaphore is automatically *locked* and an acknowledge is generated.

4. Slave-port DPRAM Broadcast Slow Write (spmemBcSWr):
Same as spmemBcWr except that MAC generates an acknowledge.

5. Slave-port DPRAM Circular Bypass Slow Write (spmemCBcSWr):
Same as spmemBcSWr except that circular bypass is enabled.

6. External FIFO Read/Write (effRd/effWr):
A read(write) request is made to the external input(output) FIFO. An acknowledge is generated upon the completion of the access. The *empty(full) flag* of the FIFO is checked prior to the access to prevent underflow(overflow) of the FIFO.

Appendix B

MAC Pinout Descriptions

The Master Access Controller Chip has 208 pins, 38 of which are for power and ground pins. The remaining pins can be divided into 6 groups:

- System Interface
- Slave Interface
- SAC Interface
- Network Interface
- Test Interface
- Clocking

Table B.1 gives a summary of the pinouts of MAC divided according to their functionalities.

System Interface

System Interface includes signals needed for interfacing the MAC with DSP32C, LRAM, DPRAM, Input and Output FIFO's. The assertion of *cycleinP* by DSP32C indicates the beginning of a memory transaction. *RwnP* tells whether the transaction is a read or write access. *MwnP* and *mgnP* are the *Write Strobe* and the *Read Strobe* respectively. *Cko* is needed for synchronizing the MAC with the processor. *ResetN* is the system reset signal. The *macEn* signal is provided to allow disabling of the system interface during reset and the possibility of interfacing serval MAC's to one DSP32C. *E_fullN* and *e_emptyN* give the status of the Output FIFO and Input FIFO respectively. These flags have to be checked before any accesses to the FIFO's are made. The addrP[0-21] and msnP[3-0] specify the address of the transaction. *GenIntr* is an output for generating an interrupt to the VME bus interface.

When some instructions are issued to MAC, DSP32C will wait indefinitely until MAC acknowledges the completion of the instruction by asserting *srdynP*. *CeF[7-0]* and *CeL[3-0]* are the chip enable signals for the processor-port of the DPRAM and the LRAM

Signal Group	Input	Output	Bidirectional
System Interface	mwnP, mgnP, rwnP, cycleinP, cko, resetN, macEn, e_fullN, e_emptyN, addrP[21-0], msnP[3-0]	srdynP, ceF[7-0], ceL[3-0], popN, pushN, genIntr	
Slave Interface		oeND[1-0], weND[1-0], addrD[10-0], msnD[3-0]	
SAC Interface		switchState_c4, masterWr_v, byPassEn_c3, masterGrant_c3, bypassGrant_c3, rdReady_c1, wrValid_c3, rdValid_c3	
Network Interface	arbL, arb2L semReqL, semLockR syncGlobalIn	arbR semLockL, semReqR syncGlobalOut	syncL[4-0], syncR[4-0], busL[26-0], busR[26-0]
Test Interface	scanEnN, scanIn mck, sck, ldin, ldpad	scanOut	
Clocking	refClk, vcoIn, gsync, testN	pump	tck[4-1]

Table B.1: MAC Signals divided into Groups.

respectively. *PopN* is the read strobe of the Input FIFO and *pushN* is the write strobe of the Output FIFO.

Slave Interface

Signals for interfacing with the slave-port of the DPRAM form the Slave Interface. *OeND[1-0]* and *weND[1-0]* are the output enable signals and the write enable signals and *addrD[10-0]* and *msnD[3-0]* specify the location of the access.

SAC Interface

SAC Interface contains signals generated by MAC to control the operations in SAC. When *switchState_c4* is high, SAC closes the internal switch on the switchable bus and vice versa. MAC asserts *masterWr_v(byPassEn_c3)* signal to initiate a write operation of the *master FIFO(bypass FIFO)* of SAC. Similarly, asserting the *masterGrant_c3 (byPassGrant_c3)* signal starts a read operation of the master FIFO(bypass FIFO). The process of reading a piece of data from a remote processing unit takes a couple of bus cycles. By asserting *rdReady_c1* MAC signals to SAC that the data to be read arrives on the switchable bus. SAC then responds by latching the data on the bus and transfering it to the processor bus. The *wrValid_c3(rdValid_c3)* indicates to SAC that the *processor ID* of the incoming *message* on the switchable bus matches the local ID and that a write(read) access should be performed to the slave-port of the DPRAM.

Network Interface

Signals in the Network Interface are used to interface with other access controllers in the system. They perform functions such as arbitration, synchronization and message transfer.

ArbL or *arb2L* is asserted when processor(s) on the left with high priority requests for bus usage. MAC in turns indicates to the processing units on the right the intentions of the left processors or itself to use the bus by asserting the output *arbR*. Thus arbitration is performed in a *daisy-chain* fashion with the processors on the left having the highest priority. *Arb2L* is currently unused but is provided to allow hooking up two MAC's to one for future extension of the architecture. The arbitration process will be described in Section 5.1.

Acquiring and releasing of semaphore among a processor group is implemented by the signals *semReqR, semReqL, semLockR* and *semLockL*. Very similar to the daisy-chain implementation of arbitration, *semReqL* is asserted when any processors to the left of MAC want to obtain the semaphore. MAC asserts semReqR when MAC itself or any processors on the left make a request for the semaphore. Once a processor gets hold of the semaphore, *semLockL* is asserted to signal the processors to the left that the semaphore is not yet released. Similarly, the input *semLockR* will be asserted if any processor to the right is holding the semaphore. The logic used to implement semaphore will be discussed in details in Section 5.1.

Global synchronization ,the synchronization of all processors in the system, is implemented by the signals *syncGlobalIn* and *syncGlobalOut*. Local synchronization, the synchronization of processors in the same *bus group*, is provided by *syncL[4-0]* and *syncR[4-0]*, with each bit of the bus handling one *synchronization pattern*.

When the processor issues a global synchronization instruction, MAC will pull *syncGlobalOut* high which is otherwise keeps low. *SyncGlobalIn* is the wired-and of the *syncGlobalOut* signals from all the processors so that global synchronization is not achieved until all processors issue a global synchronization instruction. Similarly, the condition for local synchronization of some patterns is not met if the corresponding bits of the *sync* bus are low. More detailed discussion of the synchronization logic will be covered in Section 5.1.

A *message* or *packet* travels from one processing unit to another on the switchable bus. Each message can be separated into three portions: controls, address and data. The partition is done in such a way that MAC handles the controls and address while SAC takes care of the data. The encoding of the switchable bus, *bus[26-0]*, is given in Table B.2. Bits 23 to 26 constitute the control section of the message. The *validN* bit tells whether a valid message is on the switchable bus. Messages with the *validN* disabled will simply be ignored. The *circular enable* bit enables circular bypass operation. The *rwN* indicates whether a read operation (rwN=0) or write operation (rwN=1) is to be performed at the destination processing unit. The broadcast operation is enabled by setting the *bc* high.

Bits 0 to 22 specify the location from or to which the data are to be read or written. Bits 17 to 22 contain the pid of the destination processor. Bits 15 and 16 (bsel[1-0] together with the state of an internal register called the *swapState* determine which bank of the DPRAM the access is to be made according to the following rule:

bsel=00: Bank 0 is selected.

bsel=01: Bank 1 is selected.

bsel=10: Bank (swapState xor 0) is selected.

bsel=11: Bank (swapState xor 1) is selected.

Bits 0 to 14 specify the byte location(s) to be accessed.

Test Interface

Test Interface consists of signals used in operating the boundary scan circuits on the chip. The scan mode is enabled by pulling *scanEnN* low. *ScanIn* and *scanOut* are the input and output data of the scan chain respectively. *Mck* and *sck* are the two-phase non-overlapping clocks for the scan latches. The *ldpad* signal enables the latching of all signals at the pads into the corresponding scan latches. In scan mode, inputs to MAC is decoupled from the bonding pads and is driven from an auxiliary latch associated with each pad. Data in the scan chain are loaded into these latches when *ldin* is asserted. More information on boundary scan will be provided in Chapter 8.3.

Bit#	Name	Function
26	validN	Valid Enable Low
25	cirBypassEnN	Circular Bypass Enable Low
24	rwN	Read/Write Enable
23	bc	Broadcast Enable
22	pid[5]	Processor ID bit 5
21	pid[4]	Processor ID bit 4
20	pid[3]	Processor ID bit 3
19	pid[2]	Processor ID bit 2
18	pid[1]	Processor ID bit 1
17	pid[0]	Processor ID bit 0
16	bsel[1]	Bank Select bit 1
15	bsel[0]	Bank Select bit 0
14	addr[10]	Address bit 10
13	addr[9]	Address bit 9
12	addr[8]	Address bit 8
11	addr[7]	Address bit 7
10	addr[6]	Address bit 6
9	addr[5]	Address bit 5
8	addr[4]	Address bit 4
7	addr[3]	Address bit 3
6	addr[2]	Address bit 2
5	addr[1]	Address bit 1
4	addr[0]	Address bit 0
3	msN[3]	Byte Select bit 3
2	msN[2]	Byte Select bit 2
1	msN[1]	Byte Select bit 1
0	msN[0]	Byte Select bit 0

Table B.2: Bit Encoding of the Switchable Bus in MAC.

Clocking

Signals in this section are used to generate the four-phase non-overlapping clocks for MAC. Their roles will be presented in more details in Section 7.1. *RefClk* is the 40MHz system reference clock. *Pump* is the analog output of the charge pump and *vcoIn* is the analog input to the on-chip voltage controlled oscillator. The rising edge of the global *gsync* signal is used to ensure the clock generators on different access controllers are in phase. *TestN* is an input which is normally pulled high by a weak internal pull-up. When low, the tck[4-1] bus become inputs and external test clocks can be supplied to generate all the clock signals needed on chip in case the digital phase-locked loop is not functioning. When high, the tck[4-1] bus become outputs and are driven by the internal clocks. This allow easy observation of the internal clock signals.

Appendix C

SAC Pinout Descriptions

The Master Access Controller Chip has 208 pins, 50 of which are for power and ground pins. The remaining pins can be divided into 6 groups:

- System Interface
- Slave Interface
- MAC Interface
- Network Interface
- Test Interface
- Clocking

Table C.1 gives a summary of the pinouts of SAC divided according to their functionalities.

System Interface includes signals needed for interfacing the SAC with DSP32C. *MwnP* enables the latching of the processor data bus *(dataP[31-0])* into a holding register, the contents of which will be written into the master FIFO when MAC asserts the *master_v* signal. The *mgnP* enables the tristate drivers of dataP during a processor read operation. *ResetN* is the system reset signal. The Slave Interface consists of *dataD[31-0]*, the data bus to the slave-port of the DPRAM. The bus contains data to be written to or read from the DPRAM. MAC Interface contains signals generated by MAC to control the operations in SAC. The functions of each control signals have been explained in Appendix B. The two buses, *dataR[31-0]* and *dataL[31-0]* in the Network Interface form the configurable bus for data. The Test Interface signals and Clocking signals are exactly the same as in the MAC. Detailed descriptions of the signals can be found again in Appendix B.

Signal Group	Input	Output	Bidirectional
System Interface	mwnP, mgnP, reset		dataP[31-0]
Slave Interface			dataD[31-0]
MAC Interface	switchState_c4, masterWr_v, byPassEn_c3, rdReady_c3, masterGrant_c3, bypassGrant_c3, wrValid_c3, rdValid_c3		
Network Interface			dataL[31-0], dataR[31-0]
Test Interface	scanEnN, scanIn mck, sck, ldin, ldpad	scanOut	
Clocking	refClk, vcoIn, gsync, testN	pump	tck[4-1]

Table C.1: SAC Signals divided into Groups.

Appendix D

Circuit Diagrams of Pads

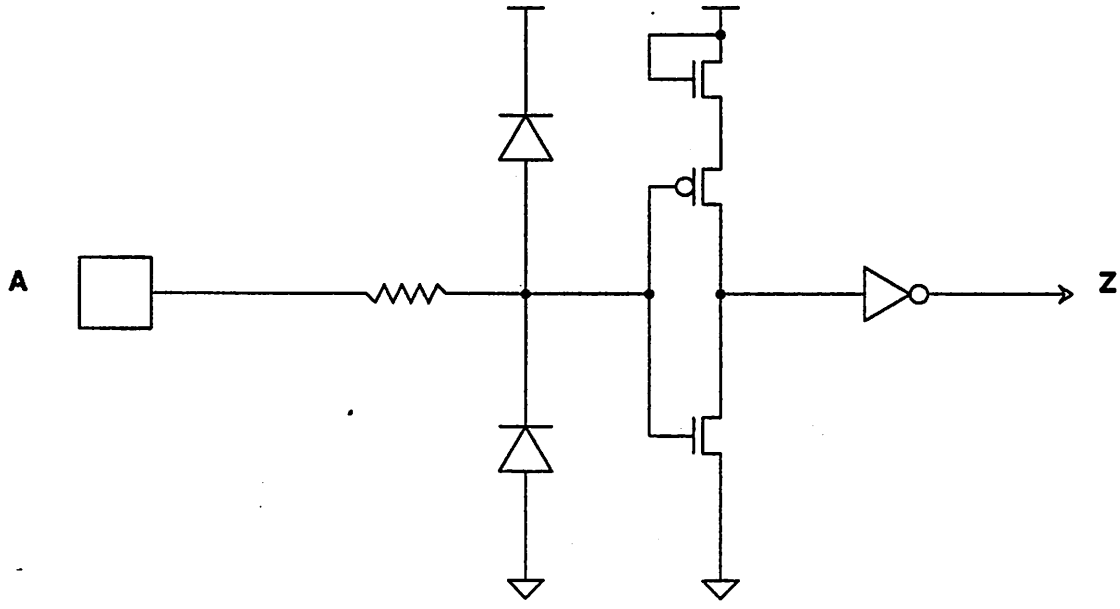


Figure D.1: Input Pad.

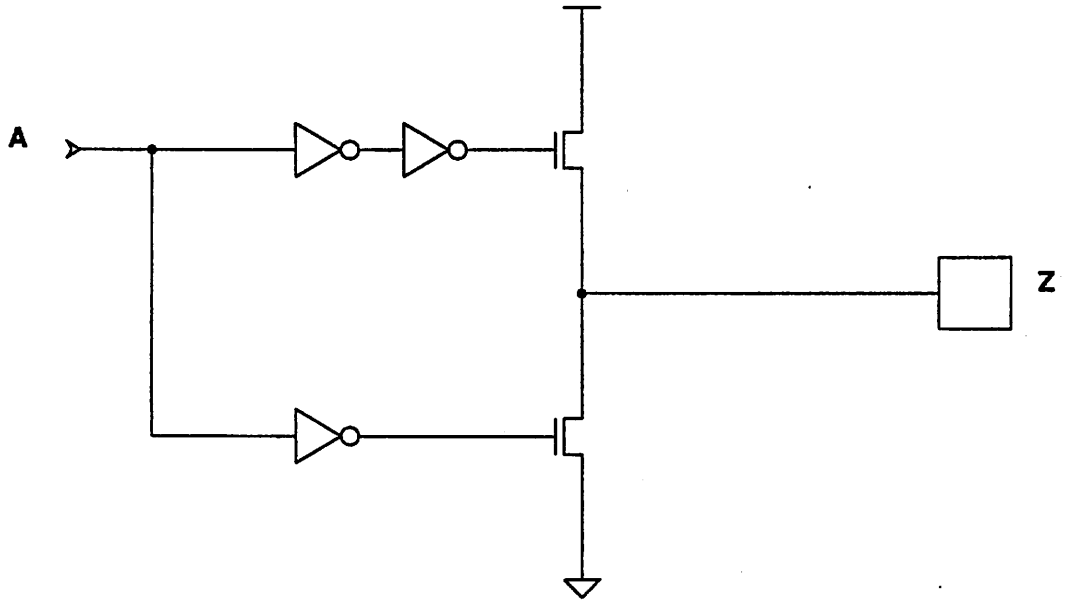


Figure D.2: Non-inverted Output Pad.

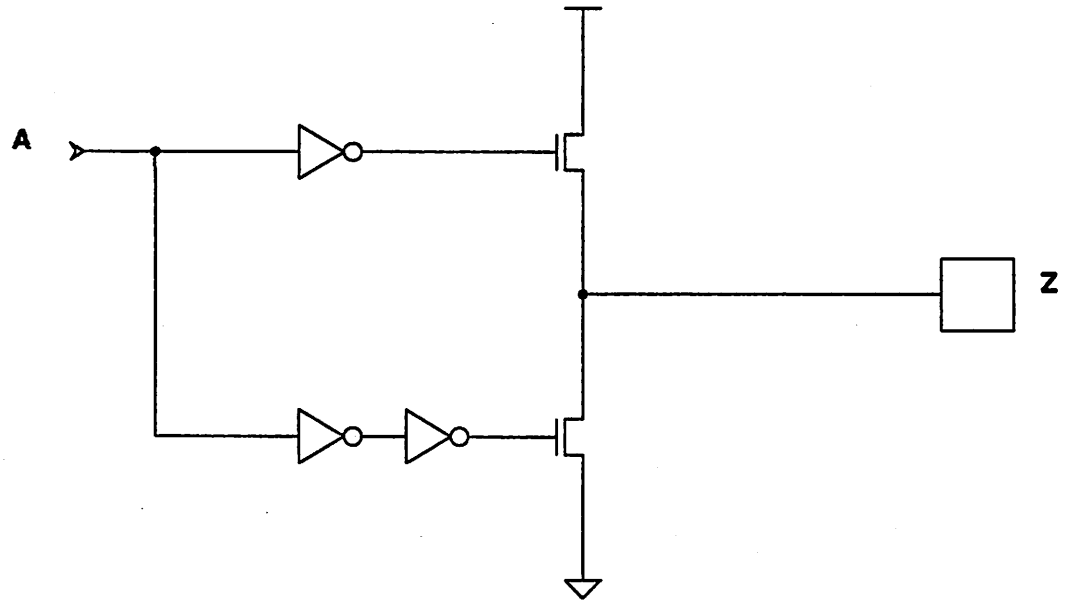


Figure D.3: Inverted Output Pad.

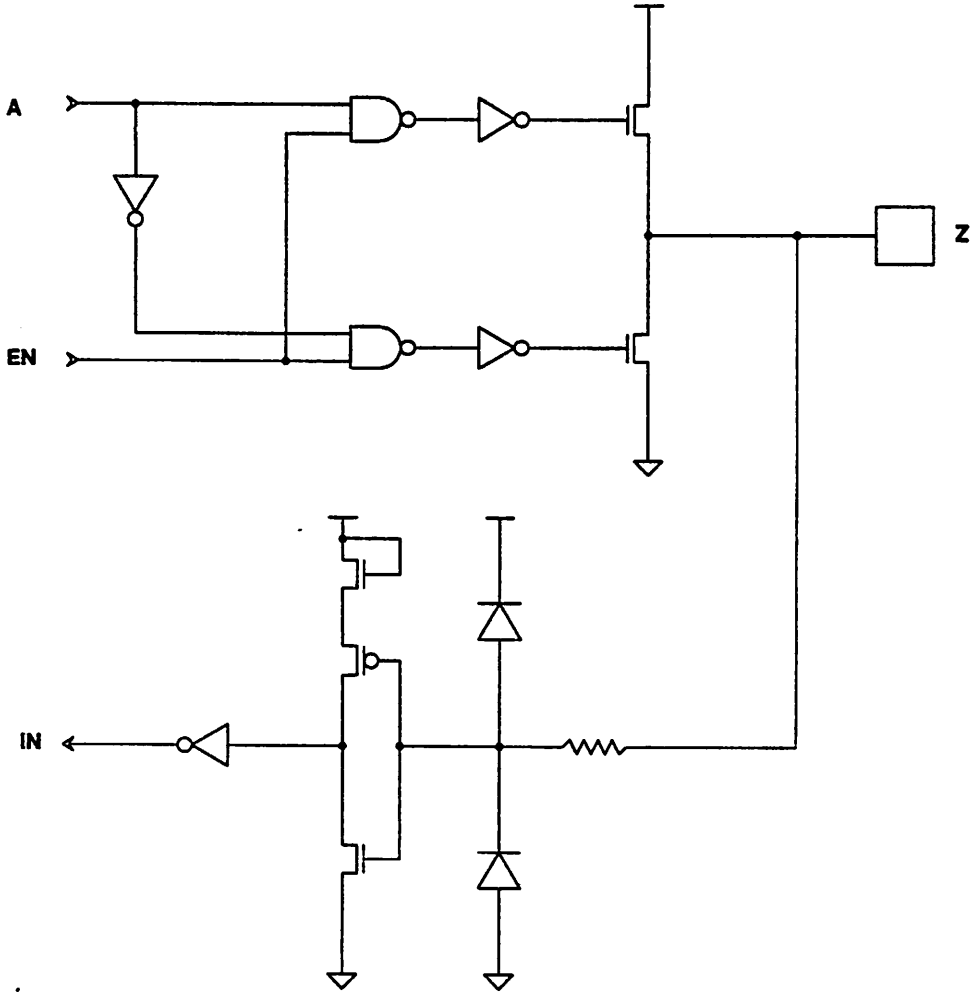


Figure D.4: Bidirection IO Pad.