A Field Programmable Gate Array Based Stream Processor for the Cheops Imaging System

by

Ross Anthony Yu

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Masters of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 16, 1996

Author .....
Electrical Engineering and Computer Science
August 16, 1996

Certified by .....
V. Michael Bove, Jr.
Thesis Supervisor

Accepted by .....
Frederick Morgenthaler
Chairman, Department Committee on Graduate Theses

A Field Programmable Gate Array Based Stream Processor for the Cheops Imaging System

by

Ross Anthony Yu

Submitted to the Department of Electrical Engineering and Computer Science on August 16, 1996, in partial fulfillment of the requirements for the degree of Masters of Engineering in Electrical Engineering and Computer Science

Abstract

We discuss the design and implementation of a new reconfigurable stream processor for the Cheops Imaging System. Cheops suffers a substantial performance degradation when performing a computation for which there does not exist a stream processor. This gives rise to a need for a reconfigurable stream processor. This thesis explores the use of reconfigurable logic in the Cheops system, and discusses a new kind of stream processor called the State Machine. This new processor joins a general purpose microprocessor with two SRAM-based FPGAs to realize reconfigurable logic. The State Machine is designed, built, and tested with results reported.

This research is supported by the Television of Tomorrow Research Consortium at the MIT Media Laboratoy.

Thesis Supervisor: V. Michael Bove, Jr.
Title: Associate Professor, Media Arts and Sciences

Acknowledgements

First of all, my thanks to my thesis advisor, Professor V. Michael Bove, Jr for giving the chance to work on this project, and for being understanding and patient when things did not go well.

Next, my thanks to John 'Wad' Watlington for not only his all-encompassing expertise and answering my questions, but also for dragging me out to softball to get some sun.

Jeff Wong for 'showing me the ropes' on my first day of work (and every day ever since), for keeping me up-to-date on all the latest personal electronics, and for being a fellow Cheops hardware guy. We shall miss the hum of Cheops.

Matt Antone for sharing Simpsons and Sean Connery impressions with me for a more stress-free life, not to mention his willingness to answer questions no matter how busy he was.

Eugene Lin for his years of friendship, and all those McDonald's trips.

Thanks to Beth Kelley and the rest of the folks at Lucent Technologies help-line for all the help on the ORCA FPGA configuration.

Last but certainly not least, my gratitude and love to my parents and sisters over the years. This thesis marks the end of my long academic career. I have finally finished school!!

Table of Contents

Chapter 1
Introduction

With the changing costs of different hardware elements, the balance between general purpose hardware and dedicated custom hardware has swung between the two schools of architecture. When processor speed was much slower, custom hardware was the norm at the sacrifice of versatility. In the past decade, microprocessor speed has grown exponentially due in no small part to the integration of additional dedicated hardware execution units. However, along with this increased performance, the software demands placed on the hardware have increased as well. This rekindles the need for custom hardware. Programmable logic offers a good solution for some target applications not requiring a large number of complex mathematical operations. The recent improvements in FPGA reconfiguration time and in gate density have made them practical for in-circuit reconfigurable processing elements. This allows the custom hardware to be reconfigurable and therefore not dedicated. The following thesis describes the background of reconfigurable hardware, the Cheops Imaging System, the role of stream processors within Cheops, and finally the State Machine stream processor, which implements reconfigurable custom hardware using FPGAs.

Chapter 2
Background

The introduction of SRAM-based Field Programmable Gate Arrays has rekindled an interest in application-specific hardware to complement general purpose processors. These new devices offer in-circuit reconfiguration, with shorter reprogramming times and an infinite number of reprogramming cycles.

2.1 Reconfigurable systems

One of the first reconfigurable custom computing architectures was the Anyboard, a PC plug-in board used as a rapid prototyping environment for digital systems development. This utilized five Xilinx XC3042 FPGAs (21,000 gates total) and took over five seconds to configure. Although this system was not used to complement a general purpose processor, it did serve to pioneer work as a reconfigurable platform for custom computing. [11]

Another notable project was SPLASH, constructed at the IDA Supercomputing Research Center. The architecture allowed up to 16 SPLASH boards to be connected to a Sun Sparcstation via a separate interface board. Each SPLASH board employed sixteen Xilinx XC4010 FPGAs (16,000 gates, total) connected by a 16x16 full cross-bar switch. The notable aspects of this project is that the FPGAs were of high density (10,000 gates) and had a reconfiguration time several orders of magnitude faster than the Anyboard. In addition, a Hardware Description Language was employed to facilitate design entry on a behavioral level. The current revision of this system, the SPLASH2, has been built with seventeen Xilinx XC4010 FPGAs connected via a crossbar network with sixteen 36-bit bidirectional ports. [6] Currently, Abbot, Athanas, and Tarmaster at the Bradley Depart-

ment of Electrical Engineering at the Virginia Polytechnic Institute and State University are implementing real-time median and morphological filtering of images on the SPLASH2.[28]

While the previous two architectures act as peripheral coprocessors to a host computer, a more integrated approach is the PRISM-II machine built by Athanas, Agarwal, and Silverman, *et. al.* at Brown University. This system employed three Xilinx 4010 FPGAs, closely coupled with an AM29050 processor. A compiler identified computationally intensive inner program loops, and automatically synthesized application specific hardware implemented on the FPGAs to complement the general purpose processors computational abilities.[27]

In addition to adding custom hardware to complement processors, the use of FPGAs may also reduce the size of a system through run-time reconfiguration. The system by Villasenor, Jones, and Schoner [29] experimented with reconfigurable hardware to time-multiplex a set of algorithms to implement far more logic than possible in combination. Their system utilized a 5K gate CLAy 31 (Configurable Logic Array) a 64K EPROM for configuration data, SRAM, a frame grabber for input/output, and an Altera EP600 EPLD to house the control state machine. The system was most useful in applications in which rapid changes to the processing logic enhanced performance.

2.2 Cheops Imaging System

The Cheops system is a scalable modular processor for video coding that is being developed by the Information and Entertainment Systems Group at the MIT Media Laboratory. The system is able to perform digital signal processing for the generation of high-resolution graphics at real-time rates. Cheops has three main types of modules: processor, input/memory, and output. Data transfer between the three types of modules are handled by the high-speed Nile bus. A lower speed Global bus is also available for control signals.

The processor module, called the P2, performs almost all of the signal processing. It delegates regular operations (*e.g.* matrix algebra, correlations, and convolutions) to specialized *stream processors*. Figure 2.1 shows that the P2 can contain up to eight stream processors which are connected to banks of high-speed Video RAM via a full crosspoint switch forming the hub. Thus, data may be streamed from a memory bank to a stream processor and

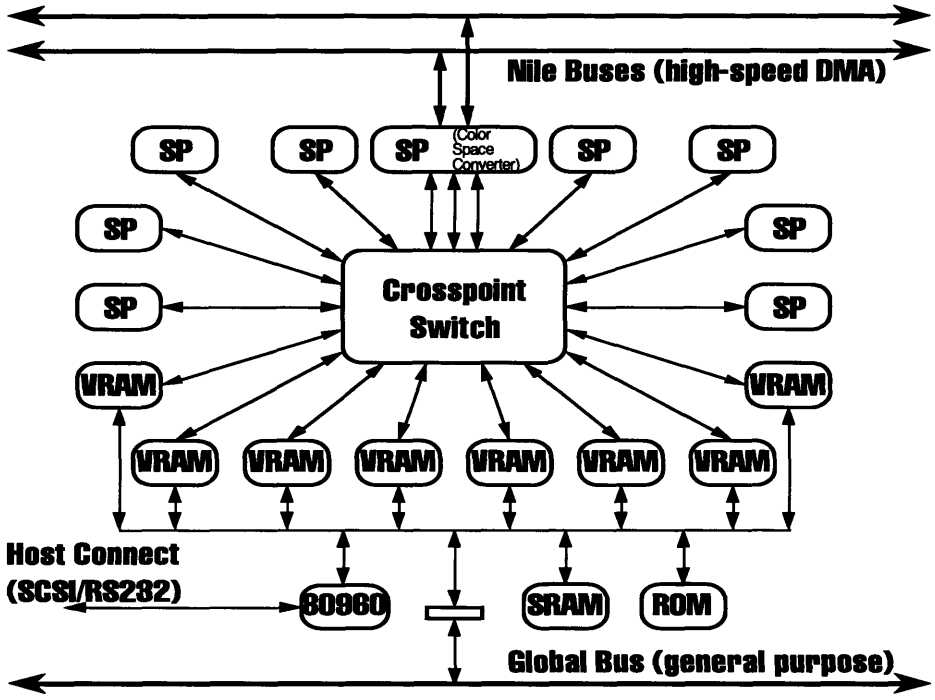


Figure 2.1: Cheops P2 Block Diagram. Blocks labeled SP represent stream processors. Blocks with VRAM represent memory banks with DMA controllers.

vise versa. The Intel 80960CF general purpose processor is responsible for sequencing and synchronizing the streaming of data through the functional elements, as well as performing computations for which a stream processor is not present.

If the stream processors can perform the regular operations with high efficiency, Cheops is able to achieve real-time processing rates. However, the absence of an appropriate stream processor forces the computation to be delegated to the i960 general purpose processor (GPP), causing a degradation in system performance. This generates a the need for a reconfigurable stream processor whose behavior can be redefined in-circuit. In this

thesis we describe a stream processor called the State Machine which will employ a general purpose processor and two FPGAs to complement its computational capabilities.

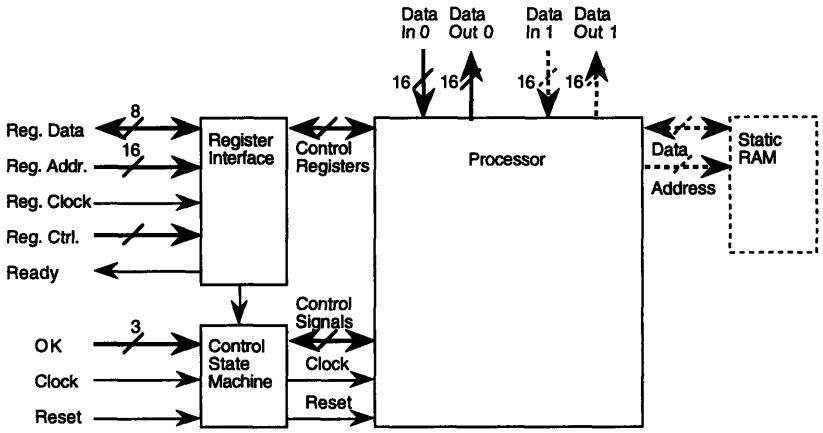


Figure 2.2: Generic Stream Processor Block Diagram. All Cheops stream processors to date have conformed to the above block diagram. Data paths are shown at the top, while control signals are to the left. SRAM exist only on two-phase stream processors.

2.2.1 Cheops Stream Processors

All of the stream processors constructed to date have conformed to a general block diagram as illustrated in figure 2. A stream processor in Cheops communicates to the surrounding system via a control port called the Register Interface, and via two Stream Data ports for video data.

The processor has access to the hub via the stream interface, located at the top of the diagram. Typically, a single stream processor card may contain up to two separate stream processors each of which would require a separate stream port. Note that each processor contains an input line and a separate output line. This accommodates single-phase operations in which processor simultaneously streams data in and out. Two-phase processors contain memory to store the stream data while it is being processed, and are necessary when the stream operation is long latency, or requires access to a set of values (*e.g.* matrix multiplication).

The register interface is directly accessible from the i960 GPP on each P2 module. It allows the i960 to configure the stream processor and update parameters. During normal operation, this contains both status and control registers accessible to both the i960 and any internal processor that may be on-board. Due to arbitration and the fact that the register interface is only eight bits wide, transfer is slow. Thus, long data transfers should be cached on board the stream processor and processed by the internal processor.

2.3 Organization of Thesis

This thesis describes the hardware and the software of the State Machine reconfigurable stream processor. The next chapter explains the hardware implementation. Chaper 4 describes the software environment in which applications run. Finally, Chaper 5 discusses State Machine applications.

Chapter 3
Hardware Implementation

In the previous chapter, we discussed the growing trend of closely coupling reconfigurable logic with a microprocessor, then illustrated how the Cheops Imaging System may benefit from a stream processor with such a design. In this chapter, we discuss the hardware implementation of the State Machine. It begins with explaining the design objectives of the board, then discusses the major components of the board and how they meet those constraints. After acquainting the reader with the main parts, board control is then discussed. Finally, we conclude this chapter with details on the flood interface by which the State Machine has access to the stream data.

3.1 Design Objectives

The three major influences on the State Machine design were conformity to the Cheops stream processor model, the ability to perform three different classes of operation, and the processing power to provide a substantial improvement over current Cheops hardware. These design objectives manifest themselves in both the determination of datapaths as well as the selection of components.

The State Machine must fit within the general stream processor block diagram that was discussed in the previous chapter. Thus, the i960 aboard the P2 must have direct access to the State Machine in such a way that it may configure the card via this access. This strongly affected the design of the PC603 bus, discussed below.

The second interface that was specified in the general stream processor description is the flood interface. In order to take advantage of the two stream data ports per stream processor card, the State Machine must be able to perform in three distinct classes of opera-

tion. It must be able to operate as two separate stream processors each using its own stream port. Next, it must be able to function as a single stream processor with the option of using one or both stream ports. Finally, the State Machine should be able to function as two separate stream processors, but working in tandem. To meet the demands placed on the board by these requirements, the State Machine was designed with two symmetrical halves, each having an FPGA, an SRAM to support two-phase operation, as well as a Look-Up Table SRAM to aid in computations. The two halves, named Boswell and Johnson, can be seen in the State Machine block diagram below (Figure 3.1). Their components are discussed in the Major Components section below.

The general stream processor model places restrictions on not only the datapaths, but also the components themselves. A Cheops stream processor is a daughter card to the P2 processing board. As a result, all stream processors are subject to size restrictions so that they may physically fit into the Cheops system. This limits the number of components that may be on the board. Given that the State Machine will contain a microprocessor, and two FPGAs, the board design becomes very dense. This makes heat dissipation and thus power consumption a concern and affects the selection of the major components.

Another influence on component selection is the fact that the State Machine computational elements must provide substantial processing power. Since the i960 processor aboard the P2 is integer-based, the State Machine must utilize a microprocessor the functional units to complement the i960. In addition, to reduce the latency on stream operations, the microprocessor should have high instruction throughput. As described in more detail in Section 3.2.1.1 below, the PowerPC603 is the best choice given these requirements.

Additional computational elements of the State Machine are the FPGAs. A simple requirement for the FPGAs is that they must be in-circuit reconfigurable and have a fast

configuration clock speed to make real-time reconfiguration practical within Cheops. Further, the FPGAs should have as high of a usable gate count as possible so that they may realize more complex functions. This count is a function of not only the total number of logic gates available, but also of the percentage of utilization. In turn, this percentage of utilization increases with better internal FPGA routing resources. As will be explained below in Figure 3.2.1.2, given these parameters, the AT&T ORCA FPGAs were a good choice

Finally the computational power of the State Machine can be augmented by the ability to cache functionality descriptions on-board. This allows the 603 to handle FPGA configuration, leaving the i960 to perform other tasks. As a result, the program memory for the 603 should be large enough to fit not only executable code, but also, configuration bitstreams for the FPGAs

3.2 Major Components

Now that the design objectives have been discussed, we turn to a discussion of the major components of the State Machine to explain how they meet those requirements. In addition, details are given of each component to lay down background information for mechanisms explained later. The following section is divided into processing elements, storage elements, and interconnection elements.

3.2.1 Processing Elements

The processing elements aboard the State Machine consist of a general purpose processor (GPP) as well as reconfigurable logic. The reconfigurable logic consists of not only

field programmable gate arrays, but also of SRAM Look-Up Tables which aid the FPGAs in computations.

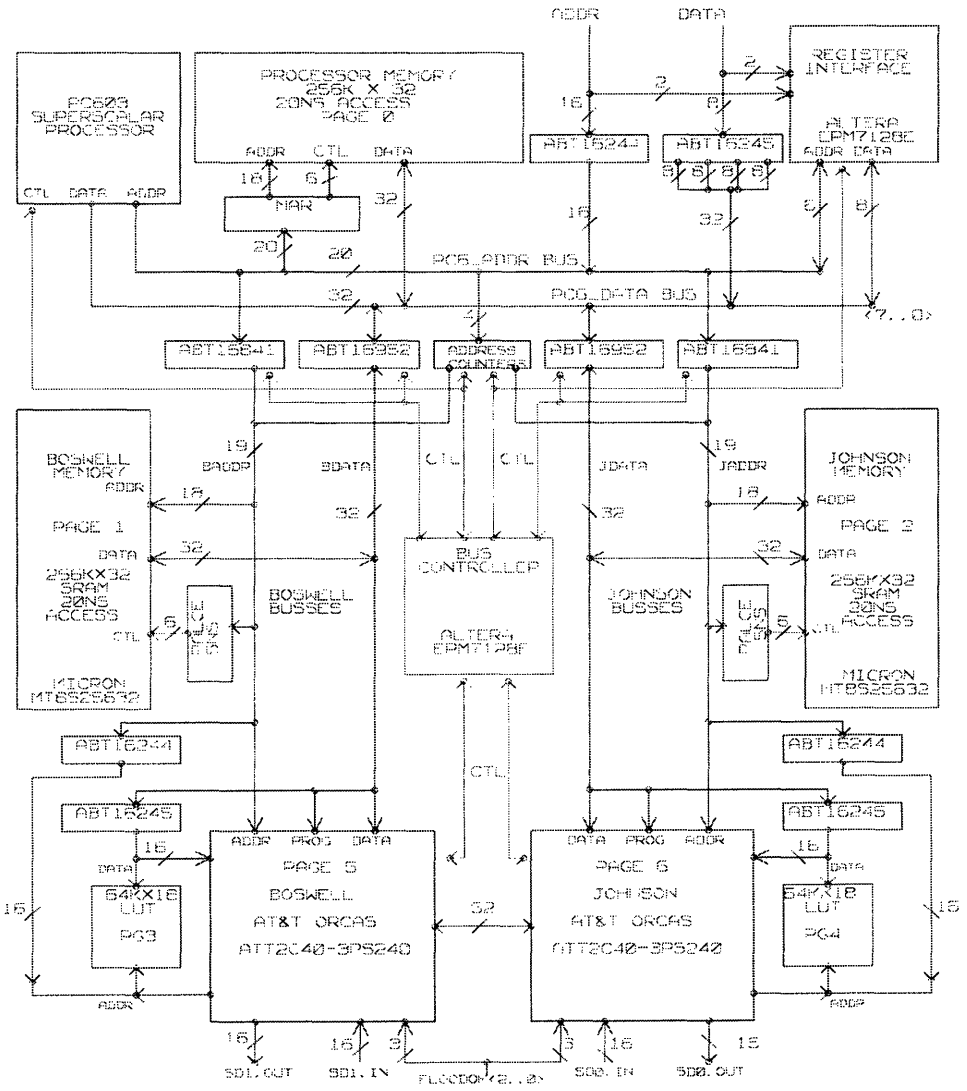


Figure 3.1: State Machine Block Diagram

3.2.1.1 Power PC603 Microprocessor

The general purpose processor is realized by a PowerPC603, a state of the art RISC microprocessor. It has 32-bit addressing and 64 bit data bus which will be run in the optional 32-bit mode. The PowerPC family of processors was chosen for its superscalar

performance, as it is capable of issuing and retiring as many as three instructions per clock. It supports out of order execution to maximize instruction throughput despite long-latency operations such as bus transactions. The 603 implementation of the PowerPC architecture has multiple execution units: an integer unit, a floating point unit, a branch processing unit, a load-store unit, and a system register unit. The performance of these units are described briefly below.[5]

The integer unit (IU) has a one cycle latency for most computations. The floating point unit (FPU) has a longer latency, but is pipelined and issues up to one instruction per clock cycle. The branch processing unit (BPU) performs static branch prediction and can often process zero-cycle branches. The load/store unit (LSU) enforces in order issue and translation of load/store instructions, though the memory accesses may occur out-of-order. The programmer may enforce strict ordering through the use of synchronizing instructions. Finally the system register unit (SRU) performs system-level control instructions, such as moves to and from special purpose registers.

Another advantageous feature of the 603 is its low-power consumption. It is capable of four power-saving modes as well as dynamic power management. The first power saving mode is sleep mode where all of the functioning units are powered down. Nap mode is where all of the functioning units are powered down except for the time base and decrementer units. Doze mode maintains data cache coherency as well as the time base and decrementer units. The 603 can exit nap or doze mode by implementing a time-out in the decrementer. The final power mode is of course full-power mode, where everything in the 603 is powered up. Dynamic power management (DPM) automatically powers down unused execution units. Since CMOS circuits consume negligible power when not switching, the execution units can be turned off by withholding the clock signal. To power on the execution unit, the 603 would just enable the clock signal, making power-up time of an

execution unit negligible. Thus, DPM operates transparently from software and external hardware.

An added benefit of the 603 is its memory address translation facilities which will be used to provide flexibility in relocating application code. In particular, block address translation (BAT) will be used to translate the effective addresses of the application code to the correct physical address. This allows the application code to be compiled to effective addresses starting with zero, regardless of the code location in physical memory. In addition to translation, the BAT facilities will also provide memory protection for the operating system.

3.2.1.2 AT&T Optimized Reconfigurable Cell Array (ORCA) Field Programmable Gate Array

The reconfigurable logic of the State Machine board will be realized by a pair of AT&T 2c40 ORCA field programmable gate arrays (FPGAs). The ORCAs represent the state of the art in FPGAs at the time of the board construction. Some of the industry standard features of the ORCAs are the 0.5 micron CMOS process technology, low power consumption, and high-frequency system clock operation (33Mhz-80Mhz). What distinguishes the AT&T ORCAs over other FPGAs is its higher usable gate count. At the time of board construction, the ORCAs boasted the highest gate count of 40,000 logic gates. In such a large die size, routing resources become even more critical to overcome increased propagation delay. However, the ORCAs' routing resources are also better designed than those of other FPGAs. A more detailed discussion of the ORCA architecture is as follows.

The ORCA architecture consists of an array of programmable logic cells (PLCs) which realize the logic with programmable input/output cells (PICs) around the perimeter. The ORCA 2c series divides the PLC array into four equal quadrants. Each quadrant is arranged into blocks of 4x4 PLCs, called sub-quad arrays.

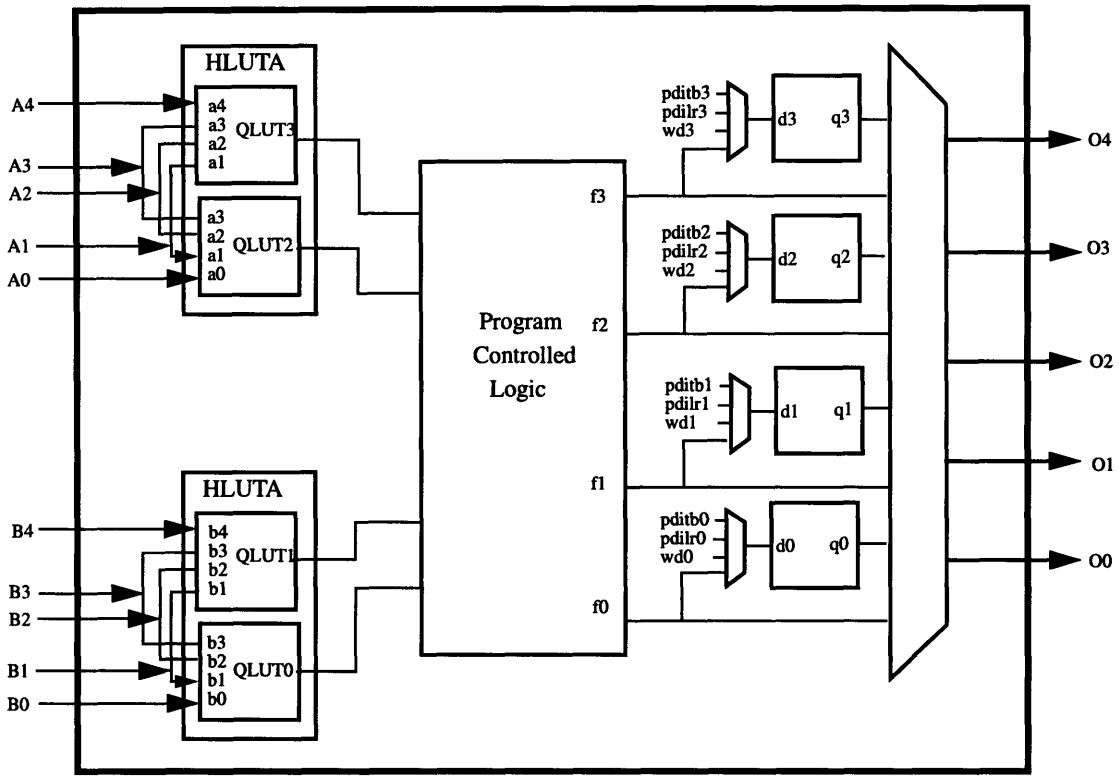


Figure 3.2: Simplified PFU Diagram

Each PLC consists of a programmable function unit (PFU) and internal routing resources. As can be seen in Figure 3.2, a PFU consists of four look-up tables (LUTs) and four latches/flip-flops for logic implementation. The LUTs realize the combinatorial logic and can be configured to operate in one of three modes (Combinatorial logic mode, ripple mode, memory mode). The latches/flip-flops realize the sequential logic. They can function as positive or negative level sensitive latches, or positive or negative edge-triggered flip-flops. Their input may come from either directly from PFU input or from the LUT. The flip-flops themselves can operate in different modes to realize synchronous/asynchronous, inverted/noninverted, or purely combinatorial logic. To speed up signals which feed external logic, the outputs of the FFs feed directly to the I/O pads for each PLC that is adjacent to a PIC.

The PICs consist of I/O buffers to interface with bond pads as well as routing resources to connect bond pads to/from PLCs. Although PICs have no user-definable register logic they do have direct pathways to the nearest PLC registers. This gives the ORCAs more versatility for over Altera FLEX architecture which have dedicated input/ output registers which cannot be used for any other function.

The routing resources of the ORCA FPGAs are implemented on various levels. Routing resources consist of switching circuitry and metal interconnect segments. The switching circuitry connects the metal interconnects, and provide either signal switching, amplification, or isolation. The ORCA routing resources are symmetric in vertical and horizontal directions, and can be classified into different levels of scope.

Within each PCL, intra-PLC routing resources are used to connect the inputs and outputs of the PFU to and from the PLCs. Inter-PLC routing resources are used to route between PLCs. The 2C40 model of ORCAs has added sub-quad routing resources as well. These routing resources stay within the bounds of the PLC 4-by-4 arrays, and are used to increase routing of the longer inter-PLC routing resources. The inter-quad routing resources are designed to connect PLCs in different quadrants, but particularly in non-adjacent quadrants. As mentioned above, PIC routing connect the bond pads to and from the PLCs and are designed to route nibble-wide data efficiently.

The metal interconnects or R-nodes, occur in different lengths. For example, inter-PLC R-nodes (metal interconnects) can occur in four types which differ in their lengths: x1, which spans one PLC; x4, spans four PLCs; xH, spans half the length/height of a PLC array; xL which spans the full length/height of the PLC array. In each PLC there are 16 x1 R-nodes (8 vertical, 8 horizontal), four sets of 4 x4 R-nodes 8 xL R-nodes (4 vertical, 4 horizontal) four horizontal and four vertical xH R-nodes run in each column and row of

the a PLC array. The R-nodes are connected by either configurable interconnect points (CIPs) or by bidirectional buffers.

The varying lengths of R-nodes provide another advantage over other FPGAs such as the Altera FLEX 8000 architecture. The FLEXes metal connections, called FastTrack Interconnects run the entire length of the device. While it saves time by avoiding switching circuits, it reduces the number of signals which can use this interconnect. That is, while the ORCAs can route many local signals through their shorter length R-nodes, the FLEXes can only route one signal through their FastTrack's regardless of the length of path.

Another advantage that the ORCAs have is the routability of clock and global signals. In order to provide a fast and low-skew clock or other global routing, dedicated clock R-nodes are provided. They run the entire length of the PLC and run two horizontal/vertical R-nodes per column/row. This increases routability of clock signal to the point that any of the I/O pins may serve as the clock input without significant clock skew.

As is common in many modern FPGAs, the ORCAs can be reconfigured in-circuit by a microprocessor. During configuration, the FPGA latches in configuration data which specifies its logical behavior. Specifically, the ORCAs will be configured in Synchronous Peripheral mode. In this mode, an external device must generate a configuration clock which the ORCA uses to serialize byte-wide configuration data. Every eight clock cycles, the microprocessor must drive a new configuration byte to the ORCA. The configuration clock is generated by the Register Interface, described below. Synchronization between the configuring microprocessor and the configuration clock is performed by the bus controller.

3.2.1.1 SRAM Look Up Tables (LUTs) 64Kbyte by 18bits

In addition to the ORCAs, the SRAM Look-Up Tables, or LUTs, realize a form of reconfigurable logic in that they can too can be reprogrammed to realize combinatorial logic. During stream data computations, the LUTs provide a one-to-one mapping for stream data coming in from the hub. The LUTs are realized by NEC UPD431018le-15 SRAM which has just a 15 nanosecond access time. It is fully programmed by either the 603 or the i960 and can be read by the its dedicated ORCA.

3.2.2 Storage Elements

The major storage elements aboard the State Machine are three SRAM modules of one megabyte each. These are used to store program code, FPGA configuration data, as well as stream data and are described below.

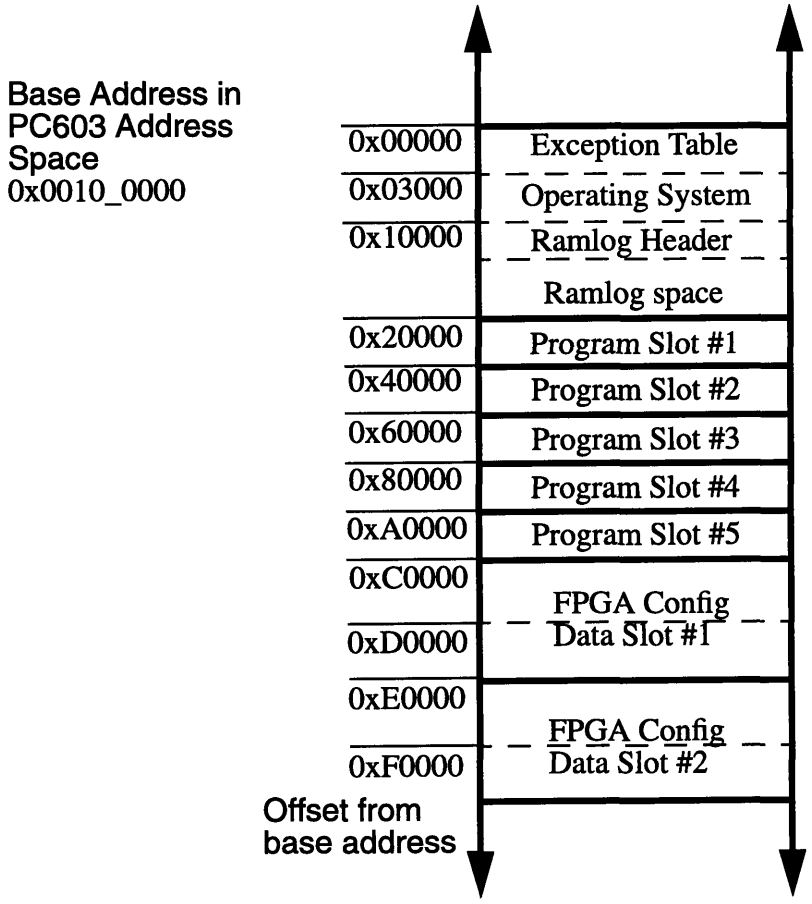


Figure 3.3: Page 0 SRAM memory space

3.2.2.1 Page0 SRAM

The Page 0 SRAM holds both 603 code, and ORCA configuration information. It is accessible by both the 603 and the i960. The 603 may access the SRAM in burst transfers which it typically uses in instruction cache line fills. The i960 writes the code and configuration information during WRITE board mode. The SRAM is partitioned by the 603 BAT facilities into 128 Kbyte slots as follows:

The first slot is reserved for the operating system, and can only be accessed while in supervisor level. The next six slots are used for storing application code. The last two slots are reserved for caching FPGA configuration data.

3.2.2.2 Boswell/Johnson SRAM

The Boswell and Johnson SRAM hold stream data flooded in through their respective FPGA. These SRAM are accessible from the 603 as well, allowing the microprocessor to perform operations on the stream data. Both the ORCA and the 603 may access the SRAM in byte, short (two bytes), and word (four bytes) lengths. The 603 may access the Boswell and Johnson SRAMs in burst transfers as well for data cache line fills.

3.2.3 Interconnections and Communications

3.2.3.1 PC603 bus

The PC603 bus is the main bus of both the PowerPC603 and the i960. It connects both the 603 and the i960 to the Page 0 SRAM (program memory) as well as the Register interface. In addition, it allows access to both the Boswell and Johnson busses, described below.

The PC603 data bus is 32-bits wide to accommodate the 32-bit architecture of the PowerPC. In order for the i960 to access this bus through the eight-bit register interface,

data bus exchangers are used to align bytes of the i960 bus within the 32-bit wide 603 data bus. The width of the address bus was determined by the 603 as well. The 603 is operating in 32-bit addressing mode with the upper-order address bits used to select devices on the board. The remaining lower-order address bits are used to drive the 20-bit wide PC603 address bus. Since the i960 can only assert 16 address lines, the remaining four upper-order address bits are asserted by the page register in the register interface whenever the i960 is driving the bus.

3.2.3.2 Boswell/Johnson bus

The Boswell and Johnson busses are symmetrical to each other and primarily connect each FPGA to its respective stream data SRAM. In addition, the Look Up Table SRAM may also be written through bus. This bus interfaces with the PC603 bus, and allows the PC603 to access the stream data SRAM and to configure the FPGA as well as program the LUTs. Configuration and LUT programming from the i960 is also supported. Note that the Boswell and the Johnson address busses may drive each other when the two FPGAs are working in tandem.

3.2.3.3 Intra-FPGA bus

The IntraFPGA bus is a 52-bit wide connection which directly connects the Boswell and Johnson ORCAs. This bus is meant to be used in applications where the two ORCAs are working together to realize a single, large function. Depending on the specific application, the ORCAs may pass signals such as data bits, state bits, or handshake signals.

3.2.3.4 Transfer Sizes

While the i960 is only capable of single beat transfers of 8-bits wide, the 603 is capable of initiating single-beat, two-beat, or burst (eight-beat) transfers. Within a single-beat transfer, the 603 may initiate one, two, three, or four-byte wide transactions. The 603 uses

burst mode transfers for cache line fills. In this mode, the 603 negotiates for the address and data busses once for eight beats of data transfer, thus reducing arbitration overhead. The addresses must be generated external to the 603 as described below in Section 3.3.5.1 The design does not support 2-beat transfers which are only used for a double-word aligned load- or store-double operation to or from the floating-point GPRs. This decision was made to simplify the bus controller logic. To cancel any 2-beat transactions that the 603 starts, the bus controller asserts /TEA which will cause a machine check exception

The ORCAs may initiate virtually any length transfer. Once it is granted its bus, the Boswell or Johnson ORCA may retain the bus by asserting the BUSLOCK signal. This was designed to allow long transactions to the stream data SRAM during flood transfers. As with the Page 0 SRAM, the Boswell and Johnson SRAM MARs support one, two, three, and four-byte wide transactions.

3.3 Board Control

Thus far, we have discussed the computational components, memory components as well as the data paths. In this section, we discuss the board control devices which keep track of board state as well as control bus transactions.In addition, we explain the hand-shake protocols that are available to the 603.

3.3.4 Register Interface

The register interface is an EPLD with registers for board control, and returning board status. It is accessible by both the i960 and the 603, and whose address space is described below.

i960 address (binary)	603 address (binary)	Meaning
00000	00011	State Register (read only)
10001	10011	General Purpose Register (read/write)
01010	01011	LPI interrupt
11011	11011	LP to State Machine Interrupt
00100	00111	Bus Priority Register (write only)
10101	10111	Helper/Configuration Register (write only)
01110	01111	FPGA interrupt store
11111	11111	Page Register (write only)

Table 3.1: Register Interface Address Space

3.3.4.1 State Register

The status register is readable by both the i960 and the 603, and returns status bits bout the State Machine board.Together, start mode bit and the configuration mode bit determine the board mode. The settings are described below. Note that the board mode values in this

Register Bit	Meaning
Q1	Board Config Mode
Q2	Board Start Mode
Q3	Boswell Configuration Done Signal
Q4	Boswell /INIT signal
Q5	Johnson Configuration Done Signal
Q6	Johnson /INIT signal
Q7	Configuration Clock Enable
Q8	Configuration Synchronization Clock

Table 3.2: State Register

register are read only. They are a mirror of the writable values found in the bus priority register, below. The Boswell/Johnson /INIT signal is used to detect bit stream errors in the configuration data. The Boswell/Johnson Configuration Done indicates when configuration for the FPGA has completed. When the FPGA has been configured, and is running, then both conf_done and nstatus (/init) should be driven high by the FPGA. Thus, the configuration master may detect successful configuration by monitoring these bits. The configuration clock is used by the ORCAs during configuration mode. The configuration sync signal (csync) is used to tell the bus controller when a new configuration byte may be written to the FPGA(s). (By the ORCA specs a new configuration byte must be written to the FPGA every eight configuration clock cycles.)

3.3.4.1 General Purpose (GP) Register

This register is used mainly to facilitate communications between the i960 and the pc603. Both the 603 and the P2's i960 may read and write the general purpose register. The GP register provides a means to pass byte sized values between the two processors.

3.3.4.2 LPI interrupt

This bit allows the State Machine to interrupt the i960. This has been designed to allow the 603 to notify the resource manager when a stream computation is complete.

3.3.4.3 LP SMI interrupt

This bit allows the i960 to interrupt the 603 to request a service. Supported services are to run a 603 application in a specific application code slot, and to configure the Boswell and/or Johnson ORCA.

3.3.4.4 Bus Priority Register

The bus priority register is used by the bus controller during bus arbitration. Each of the priority bits determine which device may own the bus when the bus is requested simul-

taneously by more than one device. In addition to priority bits, the board mode bits are also used by the bus controller.

Register Bit	Meaning
Q1	Boswell Priority bit (Boswell ORCA/603)
Q2	Johnson Priority bit (Johnson ORCA/603
Q3	PC603 bus priority bit (603/i960) (Direct to i960 data bus)
Q4	board config mode bit
Q5	board start mode bit
Q6	i960 accessible /HRESET to 603 (Direct to i960 data bus)
Q7	Unused
Q8	Unused

Table 3.3: Bus Priority Register

3.3.4.1 Helper/Configuration Register

The Configuration Register holds bits which help control configuration of the FPGAs. In addition, the Boswell and Johnson LUT chip selects are on this register, but their outputs are enabled whenever the i960 is driving the 603 data bus. (That is bits Q4 and Q5 are driven whenever the page register is driving.)

The ORCA configuration clock enable bit starts the configuration clock. This should only be activated during configuration of the ORCAs, after the /INIT bits have gone high. The Boswell or Johnson Configuration start bits should be strobed low to start configuration or reconfiguration of the FPGAs. Alternatively, if these bits are kept low, then the FPGAs are held in reset. The Boswell or Johnson look up table chip select bits may be considered an extension of the page register. They are used as chip selects when writing

the LUTs. The Configuration Master bit (603/LP) specifies which microprocessor may configure the ORCAs.

Register Bit	Meaning
Q1	ORCA configuration clock enable bit
Q2	Boswell ORCA configuration start bit
Q3	Johnson ORCA configuration start bit
Q4	Boswell Look Up Table Chip Select
Q5	Johnson Look Up Table Chip Select
Q6	Configuration Master bit
Q7	Unused
Q8	Unused

Table 3.4: Helper/Configuration Register

3.3.4.1 FPGA interrupt Store

This address stores the values of the Boswell and Johnson interrupts to the 603. When the 603 receives an external interrupt, this register is polled to discover which ORCA sent the interrupt. The meaning of the values are as follows:

0x01 - Boswell ORCA interrupt is active

0x02 - Johnson ORCA interrupt is active

0x03 - Both Boswell and Johnson ORCA interrupts are active

3.3.4.2 Page Register

Since only 16 bits of the i960's address bus is brought into the register interface and the PC603data bus is 23 bits wide, this register must drive the upper order address bits,

providing a means of paging. In addition, this register allows i960 accesses to assert chip selects to certain devices. The Boswell and Johnson ORCA select are used as chip selects when reading or writing registers on the ORCA FPGAs. This may be for such purposes as flood parameter passing, or stream parameter passing.The Page 0 SRAM select bit is used

Register Bit	Meaning
Q1	Unused
Q2	Boswell ORCA Select
Q3	Johnson ORCA Select
Q4	Page 0 SRAM Select
Q5	PC603 Address Bus Bit 16
Q6	PC603 Address Bus Bit 17
Q7	PC603 Address Bus Bit 18
Q8	PC603 Address Bus Bit 19

Table 3.5: Page Register

when reading or writing the Page 0 SRAM. The last four bits of the Page Register are the upper order address bits of the 603 address bus.

3.3.5 Bus Controller

If the 603 is the brains of the State Machine, then the bus controller is the heart of the board. Its functions can be divided into two main groups, namely arbitration, and transfer timing control.

3.3.5.1 Arbitration

The bus controller is responsible for arbitrating between bus masters for each of the three busses on the State Machine board. Each of the busses have two possible bus masters, and it is the responsibility of the bus controller to control ownership of each bus. For the 603 bus, either the 603 or the i960 via the register interface may own the address and

data busses. However, the current state of the board may dictate that only certain transactions should be allowed. To help simplify control in the Bus controller of the State

Mode Encoding <startbit, config bit>	Mode Name
0,0	Idle
0,1	Write
1,0	Config
1,1	Normal

Table 3.6: Board Modes

Machine, there are four board modes for the State Machine: Idle, Write, Config, and Normal modes.

During Idle mode, the State Machine initiates no activity. The only transfers supported are P2 i960 accesses to the register interface, to change the board mode. This is the default mode of the State Machine after power-up, and after a system reset. It ensures that the board initiates no activity until the P2 board brings it out of Idle Mode

During Write mode, the i960 on the P2 may write to registers on Register Interface, write to the Page 0 SRAM to download 603 code, and optionally ORCA configuration data, and program both the Boswell and Johnson SRAM Look-Up Tables (LUTs). This is the only mode in which the i960 has full access to the State Machine (e.g. the Boswell and Johnson busses). That is, the i960 may set values in the register interface, write to the PAGE 0 swam, write values to registers in the ORCAs, and program the LUTs. Here, the 603 is either in /HRESET, or all of its transfers are stalled in this mode to ensure that the i960 has full access.

When the State Machine is in Configuration mode, the 603 is brought out of Hard Reset (See special note on 603 /HRESET in Appendix B.1.) This is the mode in which either the 603 or the i960 configures the FPGAs. If the i960 is the Configuration Master (as determined by a bit in Register interface), then it has the same access privileges as in WRITE MODE. If it is not the Configuration Master, then the i960 only has access to the Register Interface, and the PAGE0 SRAM. If the 603 is the Configuration Master, then it has access to the register interface, the Page 0 SRAM, and the boswell and johnson busses. Otherwise, all of its transactions are stalled until the State Machine is brought out of configuration mode.

Finally, during Normal mode, the State Machine is in full operation. The 603 has full access to the 603 and Boswell/Johnson busses. The i960 only has access to the register interface so that is may change the board mode, or check board status.

The above board modes help determine which transactions may be made in different states of the board. These are used in the arbitration mechanism All of the bus masters of the State Machine follow a general arbitration sequence to obtain the bus. This sequence will be explained in the context of arbitration for the PC603 address bus. The simplified equation that the bus controller uses to give a bus grant is as follows and is explained below:

$$/BG = !(!/BR \ \& \ !BB \ \& \ (P1 \ \# \ !\langle i960 \ requesting \rangle)) \qquad (3.1)$$

In the above equation, /BG is the bus grant signal. The /BR term refers to the bus request signal asserted by a potential bus master. The /BB signal specifies when the bus is busy and is asserted by a bus master whenever it is driving the bus. The P1 signal is a priority bit as described under the register interface section. The term '<i960 requesting>' signifies that the i960 is requesting the bus. (The actual terms are more than a simple sig-

nal and are explained below.) The 603 initiates arbitration for possession of the address bus by asserting a bus request bit (/BR) to the bus controller. If the bus busy (BB) signal is not asserted and either the 603 has bus priority or does not have priority but the i960 is not simultaneously requesting the bus, then the 603 receives the bus grant. Arbitration for the Boswell and Johnson busses follow the same general scheme as above.

For the full version of the equation above, extra logic terms are added for board mode restrictions. (The equation is expanded as per DeMorgan's rule for reasons of illustration.)

$$/BG = !(!/BR \ \& \ !BB \ \& \ (P1 \ \# \ !\langle i960\ requesting\rangle) \ \& \ ((strt \ \& \ confst) \ \# \ (strt \ \& \ !confst \ \& \ Conf\_master))) \qquad (3.2)$$

The first term of the added section specifies that the /BG may be asserted while in NORMAL mode. The second term specifies that the /BG may only be asserted during CONFIG mode is the 603 is the configuration master.

Note that for the PC603 bus, the above arbitration scheme is separated for the address and data busses. This is to accommodate the 603's ability to initiate split-phase bus transactions. Thus, the address and data tenures need to coincide for the PC603 bus. Although the above example explains the PC603 address bus arbitration, the PC603 data bus uses a similar mechanism. The use of split-bus transactions allows other bus activity to occur between the address and data tenures.

As previously mentioned, the i960 requests the 603 bus in a slightly different way. While the 603 microprocessor is designed for multi-processor systems, the i960 embedded controller assumes that it has full possession of its busses. However, the State Machine realizes an arbitration scheme by taking advantage of the i960's /READY signal which can prolong an access when deasserted. When the i960 begins a transaction to the State Machine, it asserts the /ADS timing signal and either the /CS0 or the /CS1 which the bus controller interprets as a bus request from the i960 for the PC603 bus. Because there are

address and data latches between the PC603 bus and the i960's own bus, the State Machine can keep the i960 from driving the PC603 bus by not asserting the drive signals to these buffers. The bus controller prolongs the i960 transaction by deasserting the / READY signal until the i960 is granted the bus. Once the bus is granted, the bus controller waits a short period of time to allow the State machine board to service the transaction, then asserts /READY active low to allow the i960 to terminate the transaction.

3.3.5.1 Transfer Timing Control

In addition to bus arbitration, the bus controller is also responsible for the transfer timing. Each type of transfer has a control finite state machine in the bus controller to control the assertion of signals such as load and drive signals. For single-beat transfers, the timing control is fairly simple. The bus controller ensures that the destination latches on to the address and data bus values only when they are valid. Transfers are acknowledged once the values are latched in.

More complex timing control is required for 603 burst transfers. The advantage of burst transfers is that the arbitration overhead for multiple memory accesses are reduced by grouping the accesses into a single transaction that accesses sequential addresses. Thus, arbitration occurs only once, and the consecutive addresses must be generated externally to the 603. For instruction fetches and other accesses to the Page 0 SRAM, the burst mode counting is performed by the Page 0 Memory Address Register (MAR). This MAR consists of address registers, as well an EPLD which performs the counting. The MAR loads in the address generated by the 603, and the counter updates the low order address bits during the burst transfer. Similarly, the Boswell and Johnson busses are capable of burst mode transfers. For these busses, the counting is performed by PALCE22v10 programmable array logic device, called the counters pal.

In addition to burst transfers, FPGA configuration requires complex timing control. As explained in Section 3.2.1.2 above, a new configuration byte must be written to the FPGA every eight configuration clock cycles. To help meet this timing requirement, the register interface not only generates the configuration clock, but also a configuration synchronization clock (CSYNC) which is one-eighth the frequency of the configuration clock. Thus, the control FSM in the bus controller ensures that only one configuration byte is written during a single CSYNC clock period. The exception is when both FPGAs are being configured simultaneously. In this case, one Boswell configuration data byte is clocked in when CSYNC is high, and one Johnson configuration date byte is clocked in when CSYNC is low.

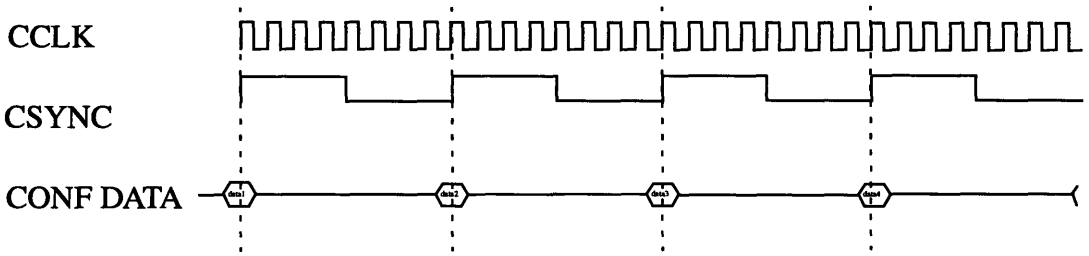


Figure 3.4: Configuration Timing Diagram

3.3.6 Control Protocol between the 603 and the i960

The Register Interface and the Page0 SRAM described above give the i960 and the 603 the means to pass data and have shared memory with each other. However, this is not an efficient means of a semaphore. To realize a handshake protocol between the i960 and the 603, each processor has a dedicated interrupt to each other.

3.3.6.1 i960 to 603 signal (SMI interrupt)

The i960 may interrupt the 603 to request a service. This is accomplished by enabling the SMI interrupt bit in the Register Interface, causing the 603 to vector to the /SMI inter-

rupt handler. This handler code checks the value in the General Purpose to find the type of service requested.

The 603 may acknowledge this interrupt by clearing the enable bit itself on the register interface.

3.3.6.2 603 to i960 signal (LPI interrupt)

The 603 may interrupt the i960 to signal when it has completed a flood service. When the 603 enables this bit, an interrupt signal (which is bussed with the other stream processor cards' signals) is generated to the i960. The i960 in turn polls each State Machine installed to identify which board sent the interrupt. The i960 may then acknowledge the interrupt by clearing the enable bit on the register interface.

The i960 also has direct access to HRESET and the PC603 bus priority bits. i.e. Writing to these bits does not require ownership of the PC603 bus.

3.3.7 Control Protocol between the 603 and the Boswell/ Johnson ORCAs

3.3.7.1 ORCA to 603 interrupt

The ORCAs need to be able to signal the 603 when it has completed a flood transfer. Both the Boswell and the Johnson ORCA have interrupt signals which are wire ORed to feed the EXTERNAL INTERRUPT pin on the 603. The 603 can then poll the register interface FPGA interrupt store address to identify which ORCA interrupted it.

For applications in which the 603 performs computations on the stream data, this signal indicates to the 603 that it may begin. For applications in which only the FPGAs are affecting the stream data, this signal indicates that the stream process has completed. At his point, the 603 uses the LPI interrupt described above to signal completion of the computation.

3.3.7.2 603 to ORCA interrupt acknowledge

The 603 may acknowledge the above interrupt by asserting the BINT_ACK or JINT_ACK signals which are wired to high order address bits on the 603. For applications involving the 603, this signals indicate to the ORCAs that the 603 has completed its computations.

3.4 Flood Transfer Control Hardware

The above board control mechanisms provide the i960 with the means to read and affect board status via the register interface. The flood interface complements that by providing the means to stream video data into and out of the State machine via the stream data interface. Physically, it is realized in the FPGA logic.

The P2 processor board supplies a set of control signals to each of the stream processor to facilitate flood transfers. These signals describe stream data location and format as well as provide timing information to the stream processor and are described below.

The stream interface receives three signals from the hub which are important during the flood transfer. The first signal is called EHubclk. During flood transfers stream data changes values relative to EHubclk. The next signal is Floodclk which is half the frequency of EHubclk. This clock signal exists to accommodate slower P2 logic devices. Finally, the three FloodOK signals are used to signify the start of a transfer. These signals change relative to FloodClk.

As mentioned above, there are three FloodOK signals to allow simultaneous flood transfers across the hub interface of the P2. Prior to any flood transfer, the State Machine must be told which FloodOK channel to use along with other parameters to describe the transfer size and addressing. The first such parameter is called the FloodOK Index and specifies which of the three FloodOK timing signals to use. The next parameter is Flood

Delay. Due to latency of the different device involved in the flood transfer (*e.g.* other stream processors, VRAM banks, the crosspoint switch) valid stream data would start streaming into the State Machine a fixed number of EHubclk cycles after the FloodOK signal goes active.

The next set of parameters help characterize the stream data as a two-dimensional transfer such as rectangular video frame of data. Data is stored in the P2's VRAM in a two-dimensional representation to reflect the rectangular frame, even though the VRAM itself is inherently one-dimensional. To perform a stream process on a subset of that data, the State Machine requires parameters to compute the physical address of the smaller frame. The xsize parameter specifies the width of the frame, while the ysize parameter specifies the length of the frame.The stride parameter specifies the width of the full frame. Finally, the start parameter specifies the offset address of the sub-frame relative to the start of full-frame.

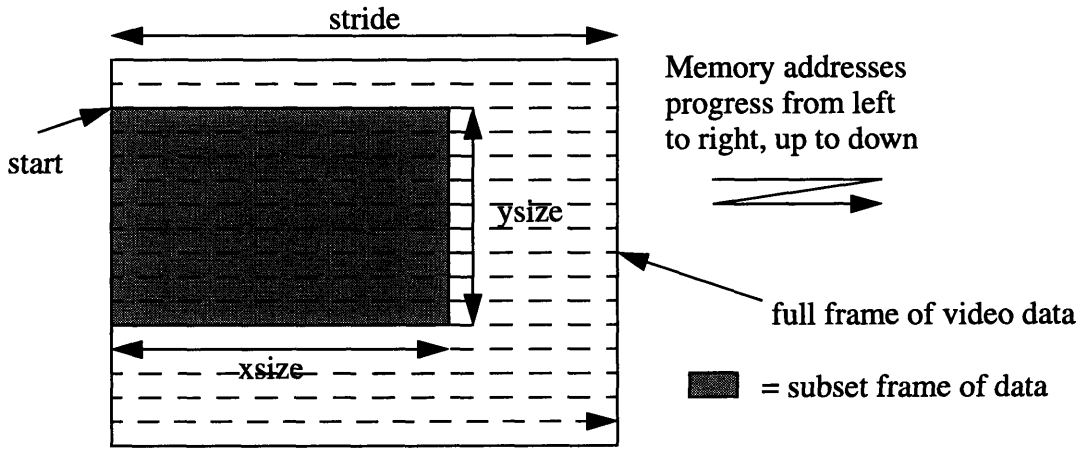


Figure 3.5: Flood transfer parameters

Using these parameters, the ORCAs can receive and send stream data to the hub side. When combined the handshake signals described in Section 3.3.6 and Section 3.3.7, the above signals allow the ORCAs to perform single-phase computations. To perform dual-phase stream computations, the ORCA must use the flood interface along with the arbitration scheme from Section 3.3.5 to stream video data into the Boswell or Johnson SRAM. The flood interface follows the general arbitration mechanism previously discussed with one addition. Each of the ORCAs may assert a buslock signal to retain possession of the its bus once it has been granted. This allows the ORCA to arbitrate once for the entire flood transfer, thus reducing control overhead. With both the flood transfer interface and the bus arbitration mechanism, the ORCAs may handle both single and dual-phase stream operations.

Chapter 4
Software Environment

The previous chapter explained how the P2 can configure the board and how stream data can move into and out of the State Machine. Further it describes the means by which the State Machine computational elements may access and move data around the board. However, the applications are what determine how the stream data is processed in the State Machine. These applications are discussed in the next chapter, but first we must describe the 603 software support for the applications.

The software environment of the State Machine is what allows applications to take advantage of the computational abilities on-board. The first part of the software environment is the memory mapping which is used by both the operating system code and the application code to write to and read from devices on board. The other part of the software environment is the operating system. Another important aspect of the software environment is how 603 code, and in particular the operating system is compiled.

4.1 603 Memory Address Space

The following is a discussion of the 603 memory address space. An overview of the entire memory space is first given, then specific parts of it are discussed.

4.1.1 603 Address Space

The 603 address space describes the way in which external devices appear to the 603 as addresses. In hardware, ten of the twelve high order address bits are wired to individual logic controlling the chip enable of the each device. Thus, to select each device for read or

write the 603 would assert the appropriate address line active high. The comprehensive memory map is shown in Figure 4.1 . In particular, notice that address 0xFFF0_0100 is

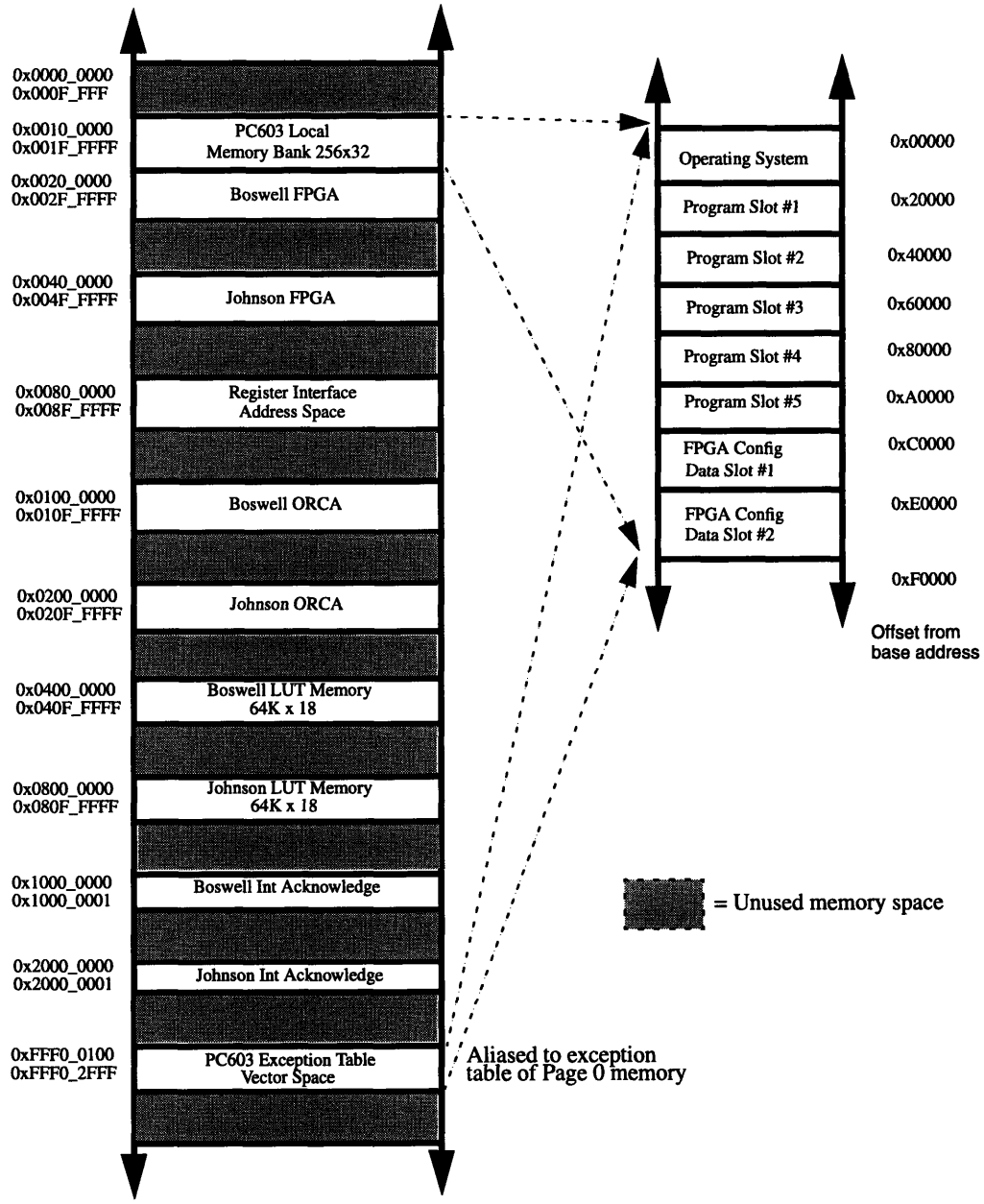


Figure 4.1: PC603 Memory Space

aliased to the Page 0 memory, because the PowerPC603 vectors to this physical address to find the exception table. This aliasing is performed in firmware.

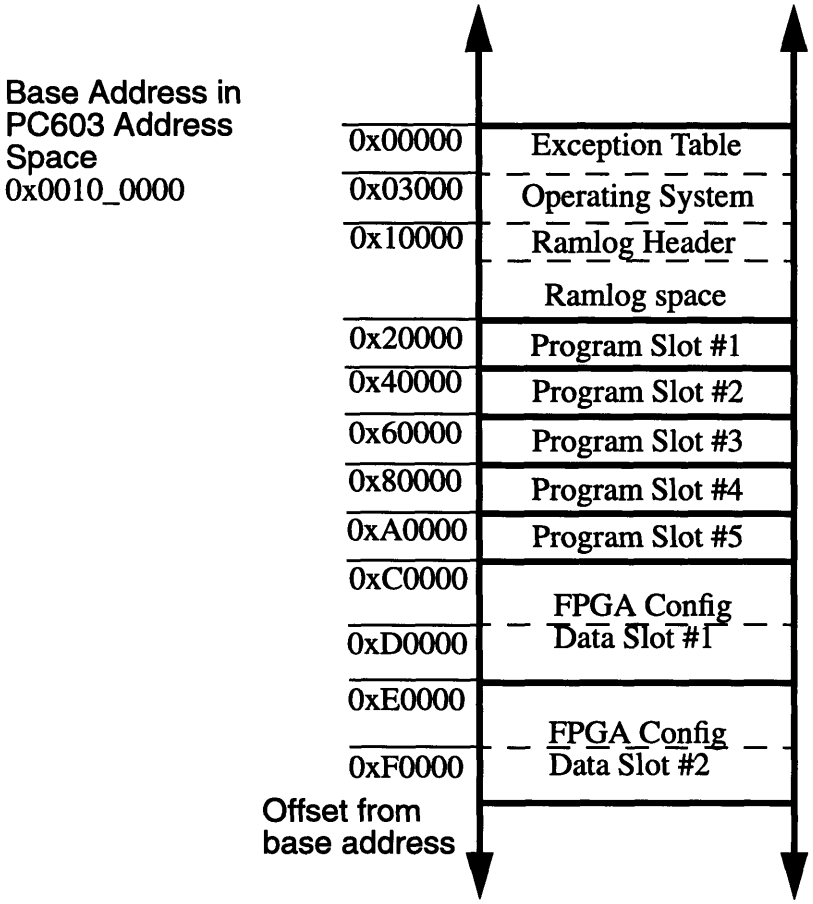


Figure 4.2: Page 0 SRAM memory space

4.1.2 Page 0 SRAM

As described the previous chapter, the Page 0 SRAM is divided into 128Kbyte slots so that the 603 Block Address Translation facilities may be used to provide not only memory translation but also protection. The figure is repeated in Figure 4.2 for a better framework in which to explain the operating system below.

4.1.3 Application code

Figure 4.3 memory map shows how each application code slot is organized. Each code slot is 128 Kbyte large. As can be seen, the application code is compiled to effective address zero, and grows down into the slot. Each application slot has its own stack space

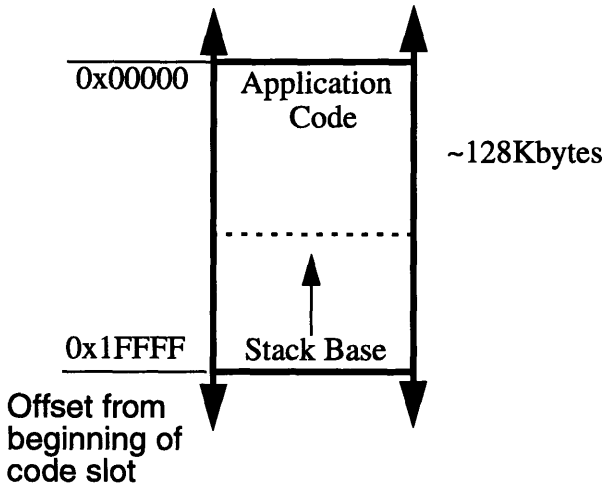


Figure 4.3: Application Memory Space

which starts at the bottom of the code slot, and grows up. Application code is further discussed in the following chapter.

4.1.4 Configuration Code Slot

The configuration code slot holds bit-streams which determine the behavior of the ORCAs. Since the configuration data for this particular model of the ORCA is 59,305 bytes long, so each configuration slot has enough room to hold two bit-stream files. Each slot will contain data for both the Boswell and Johnson ORCAs. By convention, the Boswell configuration data will exist in the lower 64Kbytes of a configuration slot, while the Johnson version of the data will exist in the upper 64Kbyte. Typically the two configu-

ration files in a slot will implement the same algorithm. However, the State Machine is versatile enough to allow different algorithms in the same code slot.

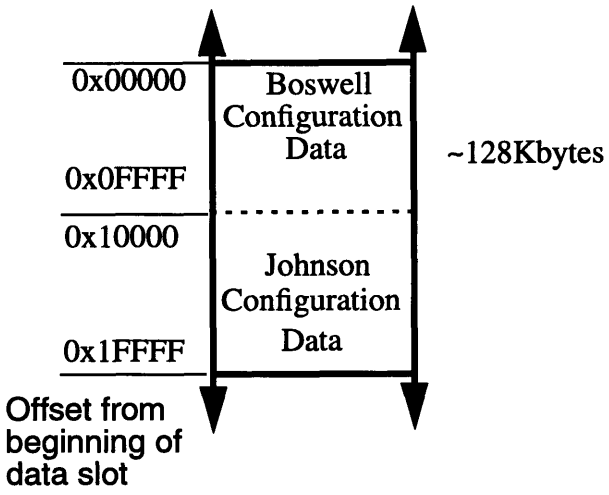


Figure 4.4: Configuration Data Memory Space

4.2 Operating System

4.2.5 Design Objectives

The operating system (OS) for the State Machine has three main design requirements. First, the operating system must be compact so that it may fit into the limited memory space on board the State Machine. Second, the (OS) must be kept simple so that no extra supporting hardware is required. Finally, it must be able to switch to application code quickly and efficiently. With these design constraints in mind, the OS was design was kept to the bare minimal functionality. The main tasks that the operating system may perform may be divided into three categories: bootstrap code, interrupt handler code, and context switching. The description of the operating system is followed by a discussion of the compiling and loading process.

4.2.6 Bootstrap Code

The Bootstrap Code is the first set of instructions that the PC603 executes upon coming out of system reset or after power-up. This code writes to 603 internal configuration registers to set up the proper running conditions, such as little-endian operation, power-saving mode, enabling of caching and of address translation. Also, this code may run power-on diagnostic tests, and initialize structures (e.g. ramlogging, address translation)

The 603 is native big-endian, and therefore boots up in big endian mode. However, the i960 and therefore the stream data is native little-endian. Therefore, the first task that the bootstrap code must perform is to change the 603 to little endian operation. This may be done by modifying a 603 special purpose register, called the Machine State Register (MSR). However, because the 603 supports pipelined and out-of-order execution, a synchronizing instruction must be used to ensure that all big-endian code is executed before switching to little-endian mode. We take advantage of the PowerPC instruction 'return from interrupt' (RFI) to perform the update of the MSR itself, synchronization, as well as branching to a new address. (For more details on the RFI instruction, the reader is referred to reference [16].) Note that the instructions of the bootstrap code up to and including the RFI must be compiled in big-endian, while the rest of the code should be in little endian. Details of compilation are discussed later in Section 4.3. Once the 603 has changed endianness, it can continue the bootstrap code.

The following is a software description of power-on sequences for the 603. The 603 code changes to little endian mode, and branch to the actual bootstrap code. Here, dynamic power management is enabled to allow power-saving. Next, the 603 initializes the BAT registers and enables address and data memory translation, The 603 enables

address and data caching, and zeroes the time base. Finally, the 603 vectors to the main code where it sits in a loop until a service request is made.

4.2.7 Exception handling

The PowerPC exception handling facilities allow the 603 to automatically change to.

Exception	Physical Address of Handler
Reserved	0xFFF0_0000
System Reset	0xFFF0_0100
Machine check	0xFFF0_0200
Data access	0xFFF0_0300
Instruction Access	0xFFF0_0400
External Interrupt	0xFFF0_0500
Alignment	0xFFF0_0600
Program	0xFFF0_0700
Floating-point unavailable	0xFFF0_0800
Decrementer	0xFFF0_0900
Reserved	0xFFF0_0A00 - 0xFFF0_0B00
System call	0xFFF0_0C00
Trace	0xFFF0_0D00
Reserved	0xFFF0_0E00
Reserved	0xFFF0_0E10 - 0xFFF0_0FFF
Instruction Translation miss	0xFFF0_1000
Data load translation miss	0xFFF0_1100
Data store translation miss	0xFFF0_1200
Instruction address breakpoint	0xFFF0_1300
System Management Interrupt	0xFFF0_1400
Reserved	0xFFF0_1500 - 0xFFF0_2FFF

Table 4.1: Exceptions

supervisor mode to handle exceptions. When the 603 encounters an external interrupt, internal error or other deviation from normal operation, it vectors to a specific hardware address based on the causing influence. Although different conditions may cause the 603 to vector to the same interrupt handler address, the PowerPC architecture has special purpose registers that the handler may access to identify the cause of the exception. In addition, the critical processor state bits are saved to Status Save/Restore registers (SRR0 and SRR1) and are restored when returning from the interrupt handler. The above table lists all of the possible exceptions that the 603 may receive as well as the physical address of the interrupt handlers. Handlers which are important to the operation of the State Machine are described below.

4.2.7.1 System Reset/Power-Up Handler

The system reset handler is the code that the 603 vectors to after power-up as well as after a system reset. After either of those causing conditions, the 603 reverts to its native big-endian operation, regardless of the exception endianness setting. The code in the handler is itself trivial, as it simply changes the board to little endian mode then vectors to the boot strap code. The 603 then proceeds as described above in Section 4.2.6.

4.2.7.2 System Management (i960 to 603) Interrupt handler

The system management interrupt (/SMI) interrupt is used to implement a service request from the i960 processor. The i960 writes a command byte to the GP register in the Register IF, then interrupts the 603 via the /SMI signal to signal that it is requesting a service from the 603. When the 603 receives the interrupt, then it will vector to the SMI interrupt handler in the table (address 0xFFF0_1400). Depending on the command byte, the

603 will vector to a different service. Services that are currently supported are listed in Table 4.2 below.

Command Byte Macro	Command Meaning
SLOT1	Run Code in Application Slot #1
SLOT2	Run Code in Application Slot #2
SLOT3	Run Code in Application Slot #3
SLOT4	Run Code in Application Slot #4
SLOT5	Run Code in Application Slot #5
SLOT6	Run Code in Application Slot #6

Table 4.2: i960 Service Request Command Bytes

For each of the above services, the i960 needs to have written a code segment in the appropriate code slot. The interrupt handler then performs a context switch which consists of the following steps.

The operating system must retrieve the parameter-passing structure (described below) from the application slot stack, and place the parameters into the general purpose registers, where the application code will retrieve them. (The cross-compiler used for applications determines that the application code will retrieve its arguments from the 603 general purpose register number three.) Once the parameters are ready, the 603 programs the IBAT registers such that they map the effective address (EA) of the application code (whose range starts with address 0) to the correct physical address (*i.e.* remapping the EA to the correct process slot).

Finally, the interrupt handler issues an RFI command to jump to the correct process slot and change back to user-level operation.

4.2.7.1 External (ORCA) Interrupt Handler

The external interrupt provides the ORCAs a direct signal to the 603. Either the Boswell or the Johnson FPGA may assert the external interrupt. The handler for interrupt from ORCA FPGAs must poll interrupt store address on register interface. Depending on that value, the interrupt handler will update a global variable in software to signify that the interrupt has been received. Typically, the 603 will have been running code in one of the applications slots, and this global value will allow the program to proceed. For more details, on application code, see Chaper 5. After the interrupt handler has updated this global variable, it will restore the previously running code.

4.2.8 RAMLOG

Ramlogging is a Cheops system-wide means of debugging by 'printing' a tokenized log into RAM. The State Machine will utilize ramlogging slightly differently. Here, ramlogging provides a of means message-passing from the State Machine to the P2 board. A ramlog consists of a circular queue in memory with head and tail pointers in the Ramlog header. When a value is added to the queue by the 603 on the State Machine, the head pointer is advanced. The i960 of the P2 periodically checks the head and tail pointers of the ramlog. If the two pointers do not match, then the i960 pops values off of the circular queue, and then advances the tail pointer. If the 603 fills the entire circular queue and runs out of space, it overwrites the least recent value in the queue.

4.2.9 Context Switching

Context switching is performed as a result of a service request from the i960. During this context switch, parameters for the application code must be passed from the i960 to the 603. The following two sections describe the parameter passing structure and mechanism. The third section then explains the context switching itself.

4.2.9.1 Parameter Passing Structure

The parameter passing structure was designed with the three following requirements in mind. First, in order to allow translation from i960 data types to the native representation of the stream processor, the parameters should be tagged with the type. Second, the

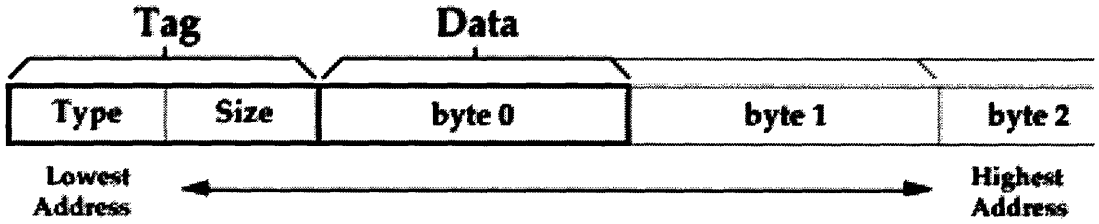


Figure 4.5: .Parameter Word

parameter structure should allow large and different size parameters to be passed. Finally, the parameter structure should be compact to reduce the parameter download overhead time. These requirements strongly suggest a tagged variable length parameter-structure to describe each parameter. This structure is shown above in Figure 4.5.As can .

Tag ID	Parameter Type	Description
0	unsigned Integer	integer values one to sixteen bytes long. also used for bit fields
1	signed integer	two's complement integer one to sixteen bytes long
2	floating point	IFFF floating point standard, the size parameter specifies the precision
3	boolean false	again the parameter type encodes the data
4	boolean true	again the parameter type encodes the data
5	shared memory ID	id for accessing large of persistent data objects
6	tagged shared memory ID	id for accessing large or persistent data objects which are aggregates of dissimilar data objects.

Table 4.3: Parameter Types

be seen above, the first nibble specifies the parameter tag and the second nibble specifies the size of each parameter in number of bytes. The list of possible parameter types are explained above in Table 4.3. To allow multiple parameters to be passed, a parameter list will be used. This list structure, as shown below in Figure 4.6, consists of a size byte which specifies the number parameters, followed by a linked list of the parameters themselves.

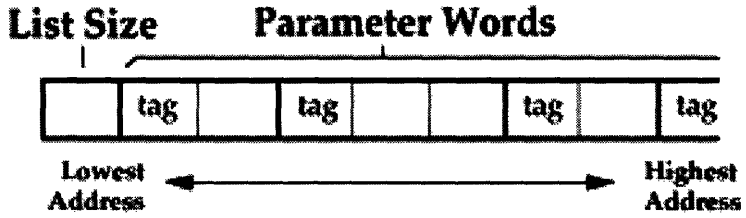


Figure 4.6: Parameter Structure

4.2.9.1 Parameter Passing Mechanism

The above sections described the parameter passing describe how the parameters are stored in a generic structure. This section describes how this structure is used to pass values to the State Machine for running application code. This process can be divided into two parts: the i960 side, and the 603 side.

The i960 takes the parameters to be fed to the State Machine application and packages it into the parameter structure described above. It then writes this parameter structure into the stack structure of the appropriate application code slot.

On the 603 side, the context switching code extracts the parameter structure from the stack structure. From here the 603 extracts the parameters out of parameter structure and places them into the 603 GP registers. (The gcc compiler implemented for State Machine applications expects application parameters to be found in the 603 internal General Pur-

pose Registers, starting with GPreg3.) For larger structures which will not fit entirely in a GP register, a pointer is to the full parameter in memory is placed into the 603 GP register instead.

4.2.9.2 Context Switching

The previous two sections described the parameter passing structure and mechanism. Here we explain how they are used in the context of context switching. Recall that context switching is used by the operating system to have the 603 vector to user application code. Because each application executable is compiled to effective address zero, the BAT registers must be reprogrammed to translate application code effective addresses to the correct code slot in the page 0 SRAM. In addition, the permission of the processor should change to user-level execution. Finally, the 603 should issue the RFI command to synchronize instructions, allow the address translation and permission changes to take affect, and branch to the application code.

4.3 Compiling the Operating System

A preliminary cross-compiler has been constructed to run on the RS6000's Since the 603 is native big-endian, the first instructions that it sees upon power-up/hard reset must be in big endian. (The State machine itself is little endian since the rest of Cheops is little-endian.) This means that the first command in the exception table offset 0x00100 should write to the MSR register to set bit 31 to 1. (See [16].pg. 2-23 for MSR bit settings.) These commands which change the endianness must therefore be in big-endian themselves.

The PowerPC 603's version of Little Endian bears further explanation. Upon power-up, the 603 is in big-endian mode. Changing the endianness (See [16], pg.2-43 for the code segment to do this.) means changing the byte order as if the 603 were running in 64-bit mode. Since we are running in 32-bit mode, this leads to a peculiar addressing scheme:

The 603 recognizes that single beat fetches of instructions occur in word lengths. Thus, the bytes of the instruction are not reordered. However, every other instruction should be swapped. That is, given the following instruction code:

0x00100 ABCD
0x00104 EFGH
0x00108 IJKL
0x0010c MNOP

where the letters A,B,...,P are used to specify bytes, the instruction fetch unit would access in the order:

0x00104 EFGH
0x00100 ABCD
0x0010c MNOP
0x00108 IJKL

This is equivalent to exclusive-ORing the address with 0x07 and zeroing out the lower order bits because of the access size. That is, word accesses will have the third lowest order address bit XORed and the two lowest order address bits zeroed out. Short accesses XOR the second and third lowest address bits and zero out the first. Finally, byte sized accesses simply XOR all three lowest order address bits.

Since the bulk of the code will be downloaded in little-endian, this addressing scheme plus the fact that the 603 will boot in big-endian mode make it necessary to reorder the instruction words in the big endian section.

Chapter 5
State Machine Applications

Now that the hardware implementation and the software environment have been described, the discussion can finally turn towards State Machine applications. As the board has three computational devices on-board, namely the 603 and the two ORCAs, there is the opportunity to use these resources in different ways. This gives rise to the application types, which will be discussed first. After we have identified the kinds of applications, we cover the application programs themselves.

5.1 State Machine Application Types

State Machine applications consist of a software executable as well as a hardware description for the ORCA(s). Although all applications consist both software and hardware descriptions, they can be classified into three groups which differ in the role that these descriptions play.

The first application class is an FPGA-only function is probably the simplest. The 603 does not contribute to computation in this type of application, but does help communicate the progress of computation to the i960. Computations such as bit-swapping or sample-filtering can be performed by one of the FPGAs and the pre-programmed SRAM LUTs.

The second application is where the two FPGAs are working in tandem. A 52-bit bus directly connects the two FPGAs and may be used for passing not only data address values, but also handshake control signals. In addition, each FPGA may drive the address bus of the other FPGAs SRAM.

The third type of application is where the 603 is used to perform stream computations. In the simplest example of this type of application, the 603 performs all of the computa-

tions on the stream data, and the FPGA functions as a flood-controller to stream video data into and out of the SRAM.

5.2 Application Descriptions

5.2.0.1 603 Application programs

As mentioned above, a State Machine application consists of both a software program and a hardware description for the ORCA(s). The software component is written in very simple C code. Since the 603 is serving as an embedded controller, it has no direct access to a file system, nor to standard input/output.Thus, commands such as fopen, fread, printf, and scanf are not supported for the state machine. Instead the 603 utilizes the Ramlog message passing scheme (See Section 4.2.8) to communicate with the external environment. To invoke reading and writing to the 603 memory space described in Section 4.1.1, the application code will use C pointer syntax.

Example 603 application code pseudocode

The following is a pseudocode description of how a 603 application would progress:

```
{
while (!GLOBAL_FLAG);/* Wait until the Global Variable is set by the
                        External Interrupt Handler from Section 3.3.7 */
do computations on stream data;
assert interrupt acknowledge back to the ORCA
assert LPI interrupt        /* i960 interrupt from Section 3.3.6 */
}
```

This pseudocode illustrates the use of the handshake protocols between the 603 and the ORCA and between the 603 and the i960.

5.2.0.2 ORCA Hardware Description

The hardware description for the ORCAs are generated with a design flow which uses industry standard design tools. A design is entered in VHDL, a text description of the logic behavior, using the Synopsys design and synthesis tools. Synopsys not only has VHDL libraries, but also a compiler which generates an EDIF netlist. This netlist is input

into the AT&T Foundry place and route tools which take in the design description and determine the optimal was to realize it in the ORCAs hardware. This generates a file containing the bit-stream as well as a file description header. This header is stripped from the file to produce a configuration bit-stream ready for download into the ORCAs.

Chapter 6
Results and Conclusion

The State Machine as described above has been designed using the Cadence design tools. Board layout, construction, and assembly were performed by an outside contractor. Completed boards were returned within approximately six weeks, during which the operating system and test routines were designed. The assembled boards were tested and debugged. There were several key difficulties in debugging the board which extended the timetable of testing beyond the scope of this thesis. First of all, the procurement and installation of the Cadence design tools delayed design entry for several months. Secondly, the 603 manual was vague in regards to operation in 32-bit mode in little-endian addressing. Further, the IBM application note pertaining to device wiring was inaccurate. Finally, the AT&T documentation about the ORCA configuration process was inaccurate as well. However, despite these minor setbacks, the following functionalities were successfully tested and debugged:

- power supply
- i960 accesses to both register interface and Page 0 SRAM
- i960 writes to Boswell and Johnson ORCA registers
- i960 to 603 interrupt - hardware and interrupt handler
- configuration from i960
- 603 instruction format (compilation, loading)
- all supported 603 transaction types to Page 0 SRAM
- all supported603 transaction types to register interface
- all supported 603 transaction types to Boswell and Johnson SRAMs
- 603 code for block address translation, data and instruction caching
- C code compiler and loader for applications
- ORCA to Boswell/Johnson writes and reads

The following functionalities are yet to be tested:

- programming and use of LUTs

- 603 to Boswell/Johnson ORCA interrupt
- 603 to i960 interrupt
- FPGA configuration from the 603

The tested functions enable the State Machine to perform both FPGA-only applications and applications where FPGAs work in tandem. Further work with the State Machine should obviously continue testing of the functionalities. In addition, applications and benchmark tests may be performed to measure Cheops performances with and without a State Machine. Another interesting benchmark would be to compare the State Machine's performance versus the other stream processor (whose functionalities are static).

Another study may be to test the resources of the ORCAs themselves. The Boswell and Johnson ORCA pin assignments were designed to be symmetric to each other to allow the intra-FPGA bus to be shortened. The specifications in the ORCA databook claim that due to the symmetry of both the routing and computational resources, that such a difference in pin assignments should not produce a significant difference. It would be worthwhile to run benchmarks to test this claim, and for not only logic performance, but also for logic routability.

Although the Page 0 memory space has been explicitly mapped out in Section 4.1.2, a study may be conducted to find the optimum ratio of number of 603 code slots versus number of FPGA configuration data slots for video stream processing.

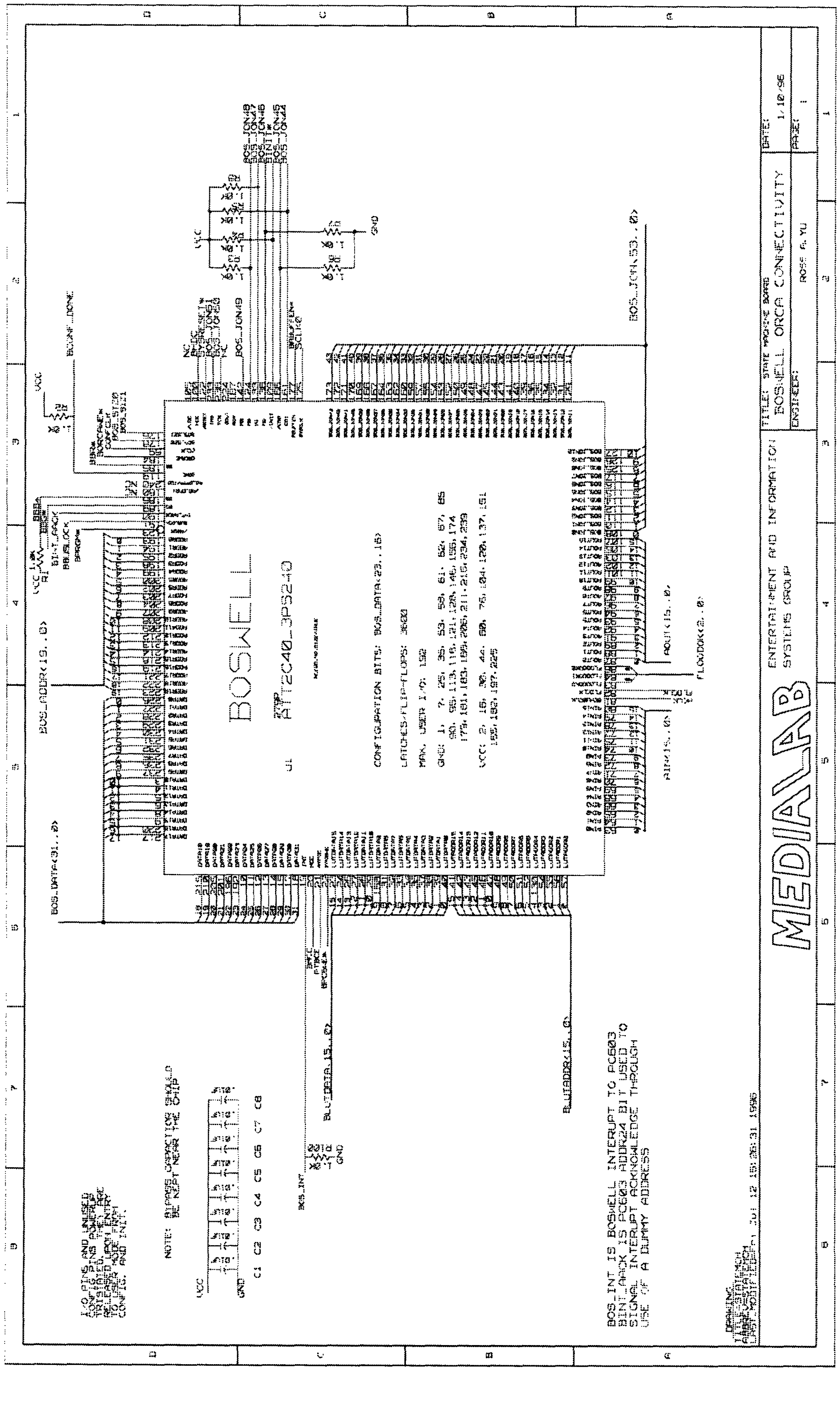
As can be seen the State Machine holds many opportunities to experiment with reconfigurable hardware. Different stream processor designs may be implemented as State Machine applications with a shortened design cycle. The caching of State Machine functionalities may be studied to optimize Cheops performance. Finally, the because of versa-

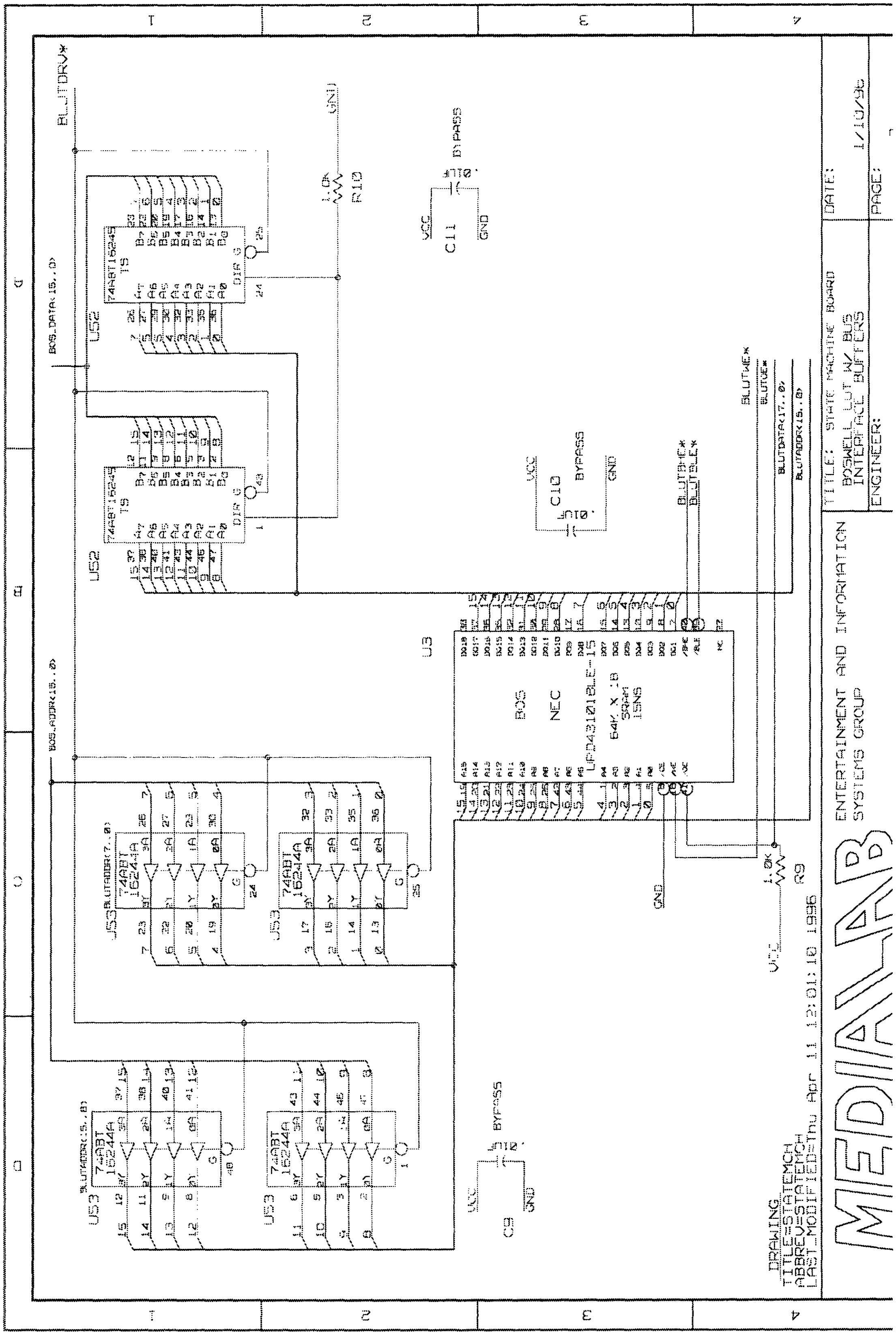
tility of its data and control paths, the State Machine maybe able to implement algorithms not easily realized.

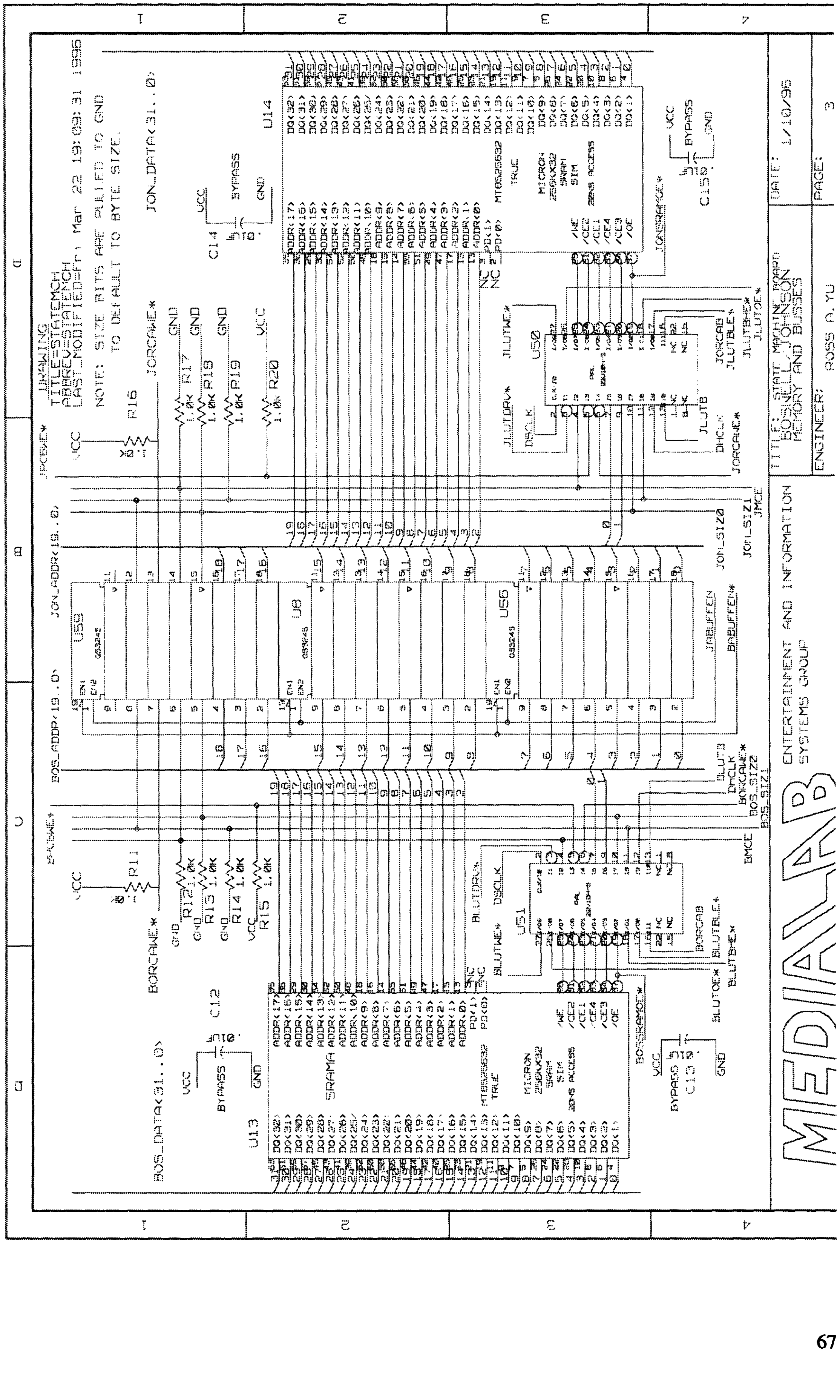
Appendix A
Schematics

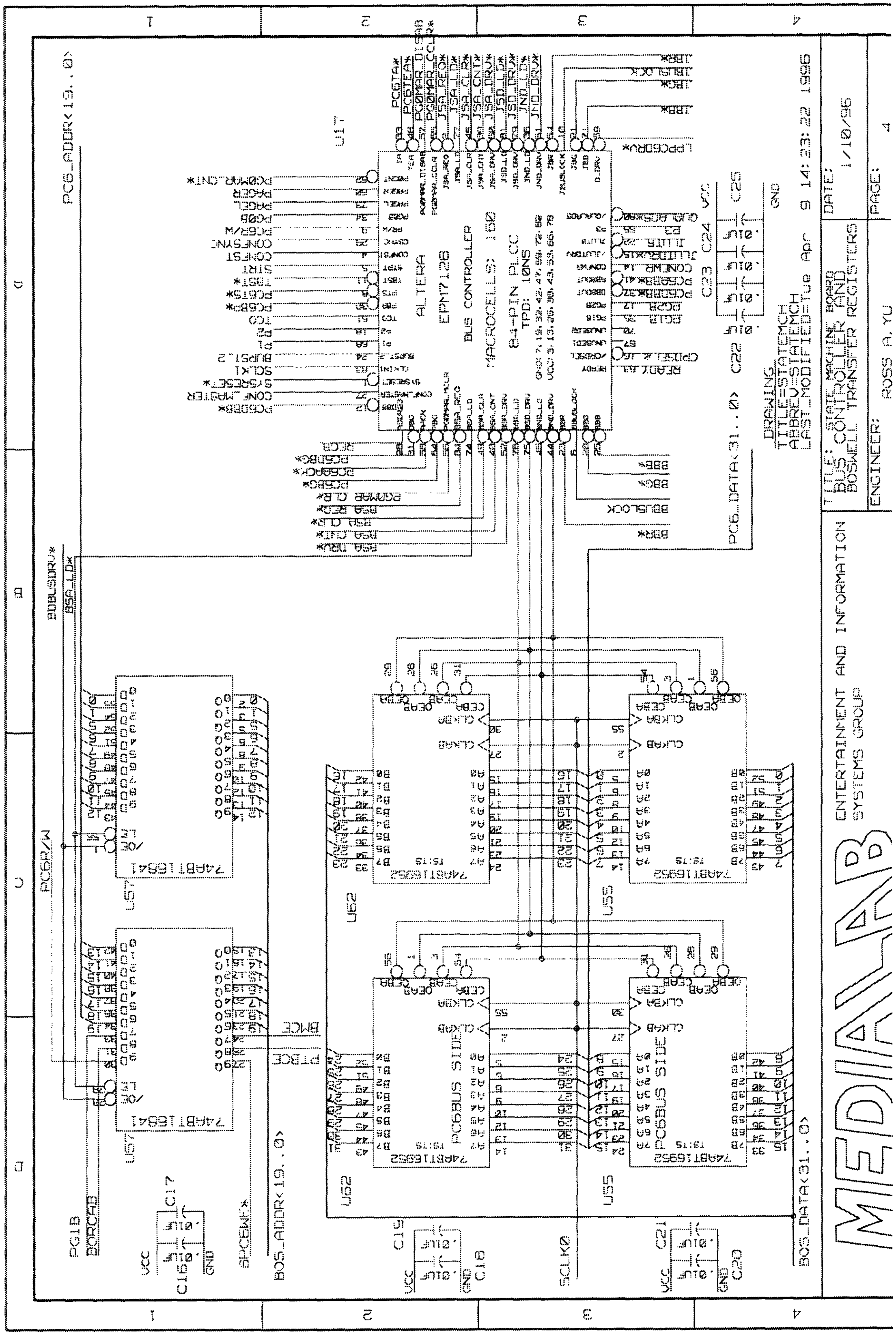
The following pages show the schematics of the State Machine Board. These schematics

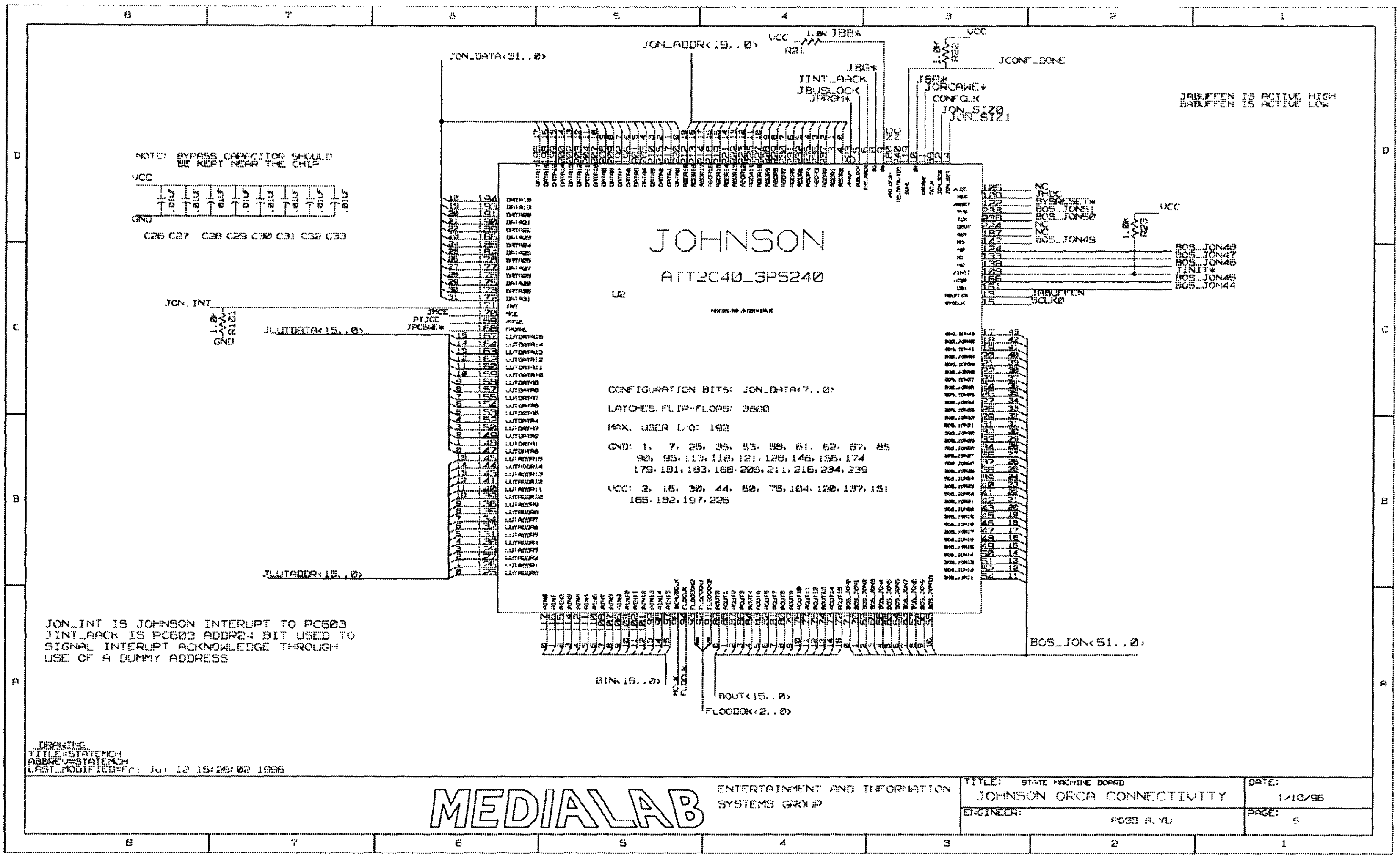
are generated by the Cadence Design Tools using the Concept entry tools. .

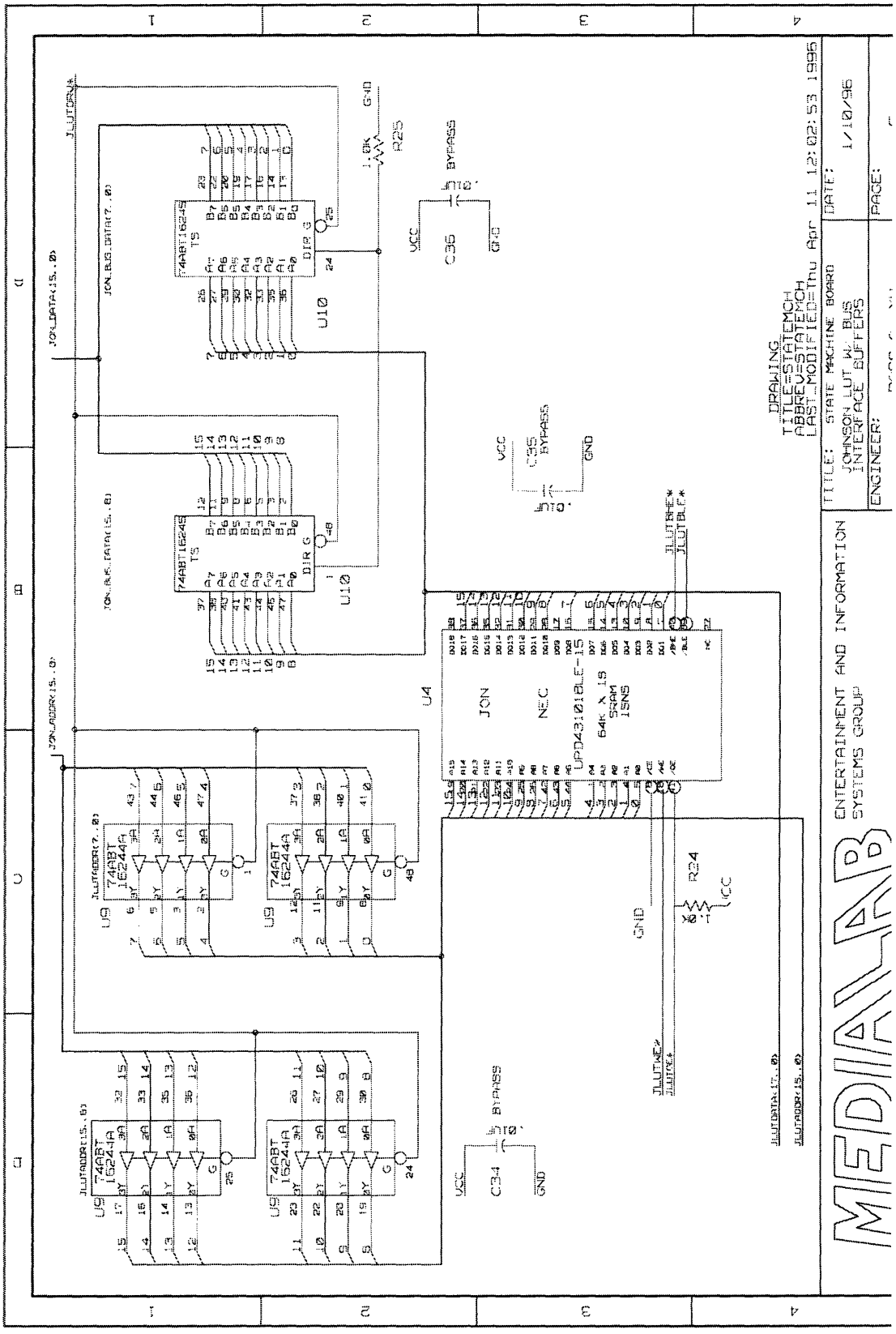

BOSWELL
ATT2C40_3PS240
J1
CONFIGURATION BITS: BOS_DATA(23..16)
LATCHES/FLIP-FLOPS: 3600
MAX. USER I/O: 192
BOS_ADDR(19..0)
BOS_DATA(31..0)
BOS_JCN(53..0)
BCONF_DONE
BLUDATA(15..0)
BLUADDR(15..0)
AOUT(15..0)
AIN(15..0)
FLOODOK(2..0)
BOS_INT
NOTE: BYPASS CAPACITOR SHOULD BE KEPT NEAR THE CHIP
C1 C2 C3 C4 C5 C6 C7 C8
I/O PINS AND UNUSED CONFIG PINS POWER UP TRISTATED. THEY ARE RELEASED UPON ENTRY TO USER MODE FROM CONFIG. AND INIT.
BOS_INT IS BOSWELL INTERUPT TO PC603
BINT_AACK IS PC603 ADDR24 BIT USED TO SIGNAL INTERUPT ACKNOWLEDGE THROUGH USE OF A DUMMY ADDRESS
MEDIALAB
ENTERTAINMENT AND INFORMATION SYSTEMS GROUP
TITLE: STATE MACHINE BOARD
BOSWELL ORCA CONNECTIVITY
ENGINEER: ROSE A. YU
DATE: 1/10/96
PAGE: 1

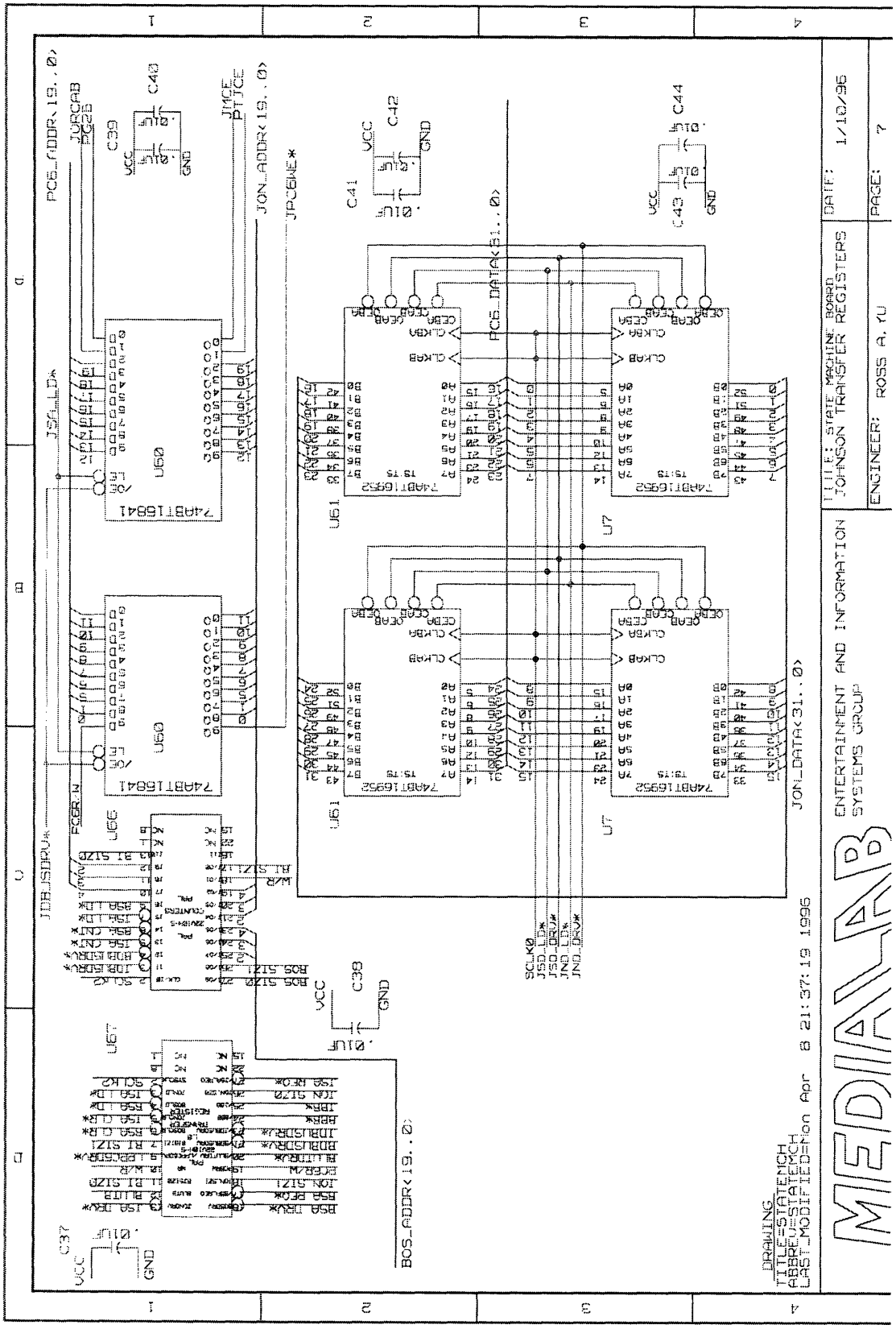

TITLE: STATE MACHINE BOARD
BOSWELL LUT W/ BUS INTERFACE BUFFERS
ENGINEER:
DATE:
1/10/96
PAGE:
ENTERTAINMENT AND INFORMATION SYSTEMS GROUP
DRAWING
TITLE=STATEMCH
ABBREV=STATEMCH
LAST_MODIFIED=Thu Apr 11 12:01:10 1996
U3
BOS
NEC
UPD431010LE-15
64K X 16
SRAM
15NS
U52
74ABT16245
TS
U53
74ABT
16244A
C9
C10
C11
.01uF
BYPASS
R9
R10
1.0K
VCC
GND
BLUTDRV*
BLUTWE*
BLUTOE*
BLUTDATA<17..0>
BLUTADDR<15..0>
BLUTBHE*
BLUTBLE*
BOS_DATA<15..0>
BOS_ADDR<15..0>
BLUTADDR<7..0>
BLUTADDR<15..8>

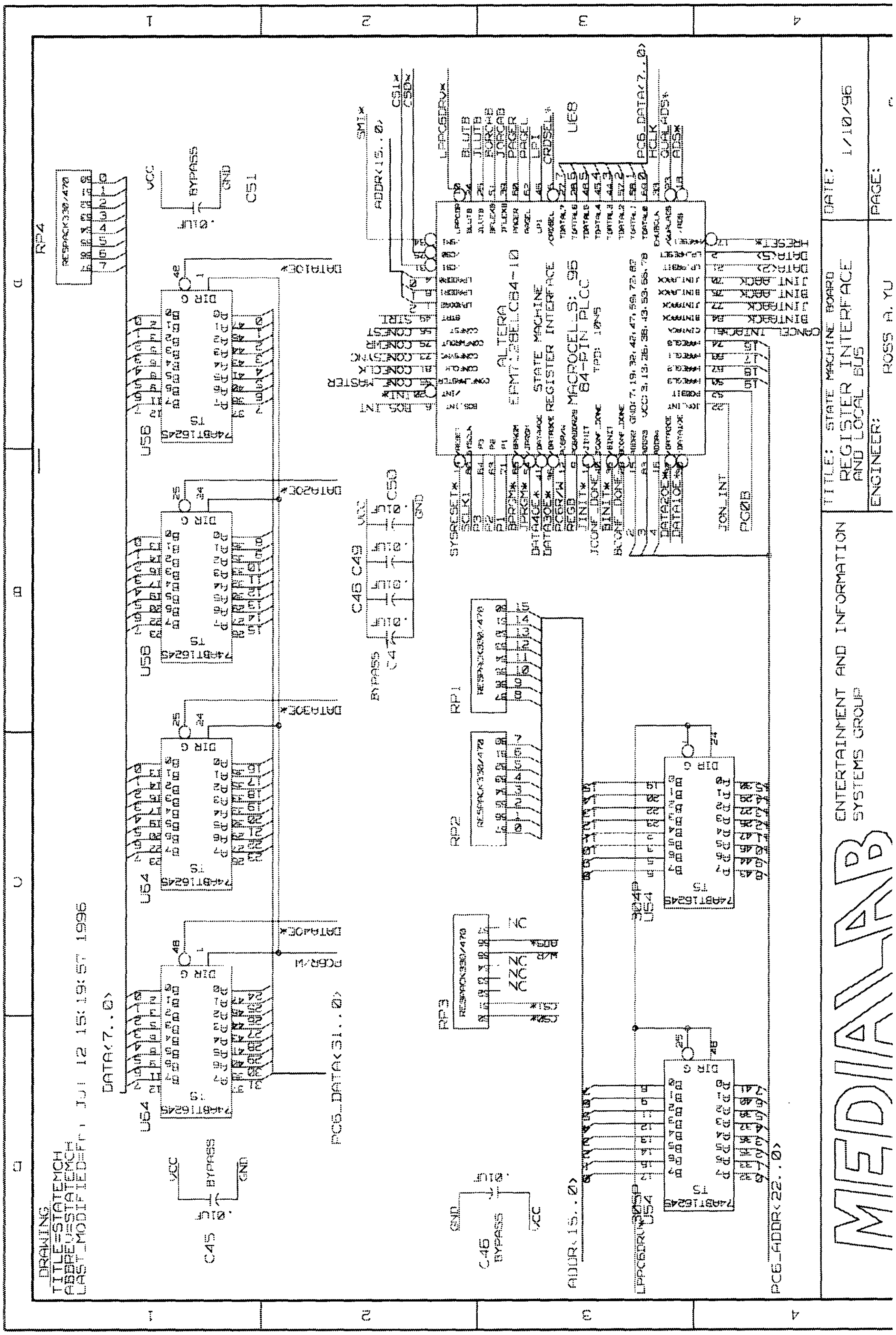

DRAWING
TITLE=STATEMCH
ABBREV=STATEMCH
LAST_MODIFIED=Fri Mar 22 19:09:31 1996
NOTE: SIZE BITS ARE PULLED TO GND
TO DEFAULT TO BYTE SIZE.
TITLE: STATE MACHINE BOARD
BOSNELL JOHNSON
MEMORY AND BUSSES
DATE: 1/10/95
PAGE: 3
ENGINEER: ROSS A. YU
MEDIALAB
ENTERTAINMENT AND INFORMATION
SYSTEMS GROUP


ALTERA
EPM7128
BUS CONTROLLER
MACROCELLS: 160
84-PIN PLCC
TPD: 10NS
74ABT16952
74ABT16841
PC6BUS SIDE
PC6_ADDR<19..0>
PC6_DATA<31..0>
BOS_ADDR<19..0>
BOS_DATA<31..0>
DRAWING
TITLE=STATEMCH
ABBREV=STATEMCH
LAST_MODIFIED=Tue Apr 9 14:23:22 1996
TITLE: STATE MACHINE BOARD
BUS CONTROLLER AND
BOSWELL TRANSFER REGISTERS
DATE: 1/10/96
PAGE: 4
ENGINEER: ROSS A. YU
ENTERTAINMENT AND INFORMATION
SYSTEMS GROUP
MEDIALAB

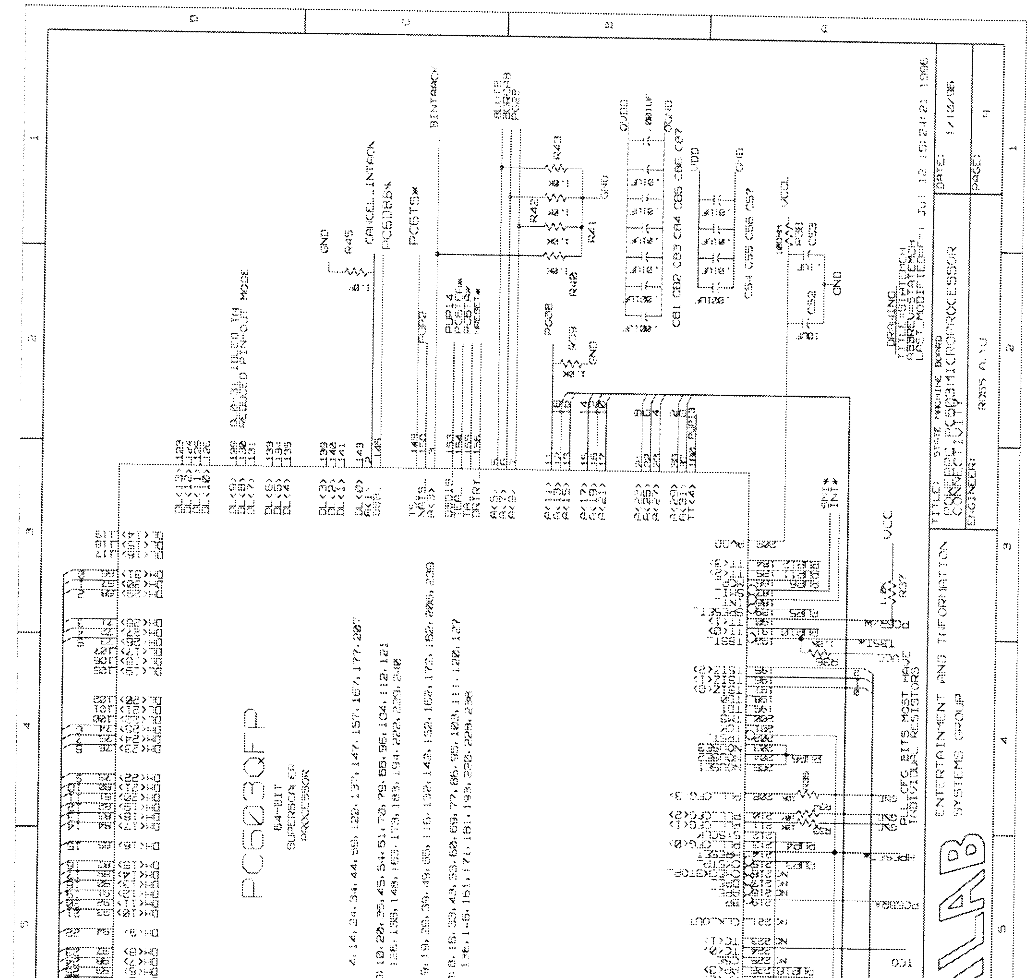

JOHNSON
ATT2C40_3PS240
U2
CONFIGURATION BITS: JON_DATA<7..0>
LATCHES,FLIP-FLOPS: 3600
MAX. USER I/O: 192
GND: 1, 7, 25, 35, 53, 58, 61, 62, 67, 85
90, 95,113,118,121,128,146,156,174
179,181,183,188,205,211,216,234,239
VCC: 3, 16, 30, 44, 50, 76,104,120,137,151
165,192,197,225
NOTE: BYPASS CAPACITOR SHOULD
BE KEPT NEAR THE CHIP
C26 C27 C28 C29 C30 C31 C32 C33
JON_DATA<31..0>
JON_ADDR<19..0>
JON_INT
JLUTDATA<15..0>
JLUTADDR<15..0>
BIN<15..0>
BOUT<15..0>
FLOGDOK<2..0>
BOS_JON<51..0>
JCONF_DONE
JABUFFEN IS ACTIVE HIGH
BABUFFEN IS ACTIVE LOW
JON_INT IS JOHNSON INTERUPT TO PC603
JINT_AACK IS PC603 ADDR24 BIT USED TO
SIGNAL INTERUPT ACKNOWLEDGE THROUGH
USE OF A DUMMY ADDRESS
DRAWING
TITLE=STATEMCH
ABBREV=STATEMCH
LAST_MODIFIED=Fri Jul 12 15:20:02 1996
MEDIALAB
ENTERTAINMENT AND INFORMATION
SYSTEMS GROUP
TITLE: STATE MACHINE BOARD
JOHNSON ORCA CONNECTIVITY
ENGINEER: ROSS A. YU
DATE: 1/10/96
PAGE: 5

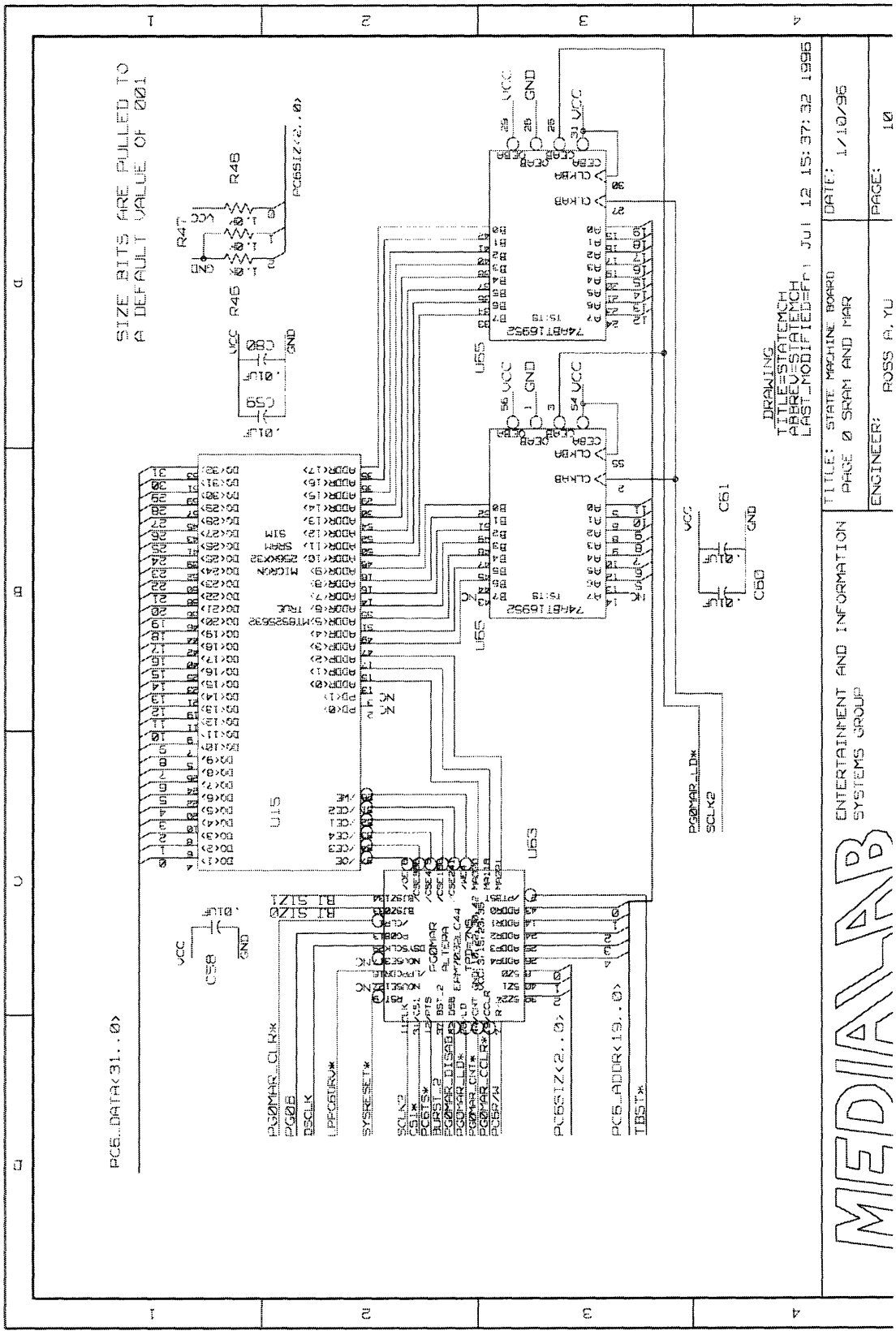

MEDIALAB
ENTERTAINMENT AND INFORMATION
SYSTEMS GROUP
DRAWING
TITLE=STATEMACH
ABBREV=STATEMCH
LAST_MODIFIED=Thu Apr 11 12:02:53 1996
TITLE: STATE MACHINE BOARD
JOHNSON LUT W/ BUS
INTERFACE BUFFERS
ENGINEER:
DATE: 1/10/96
PAGE:
U4
JON
NEC
UPD431018LE-15
64K X 18
SRAM
ISNS
U10
74ABT16245
TS
DIR G
U9
74ABT
16244A
R25
1.8K
R24
C34
C35
C36
C55
BYPASS
VCC
GND

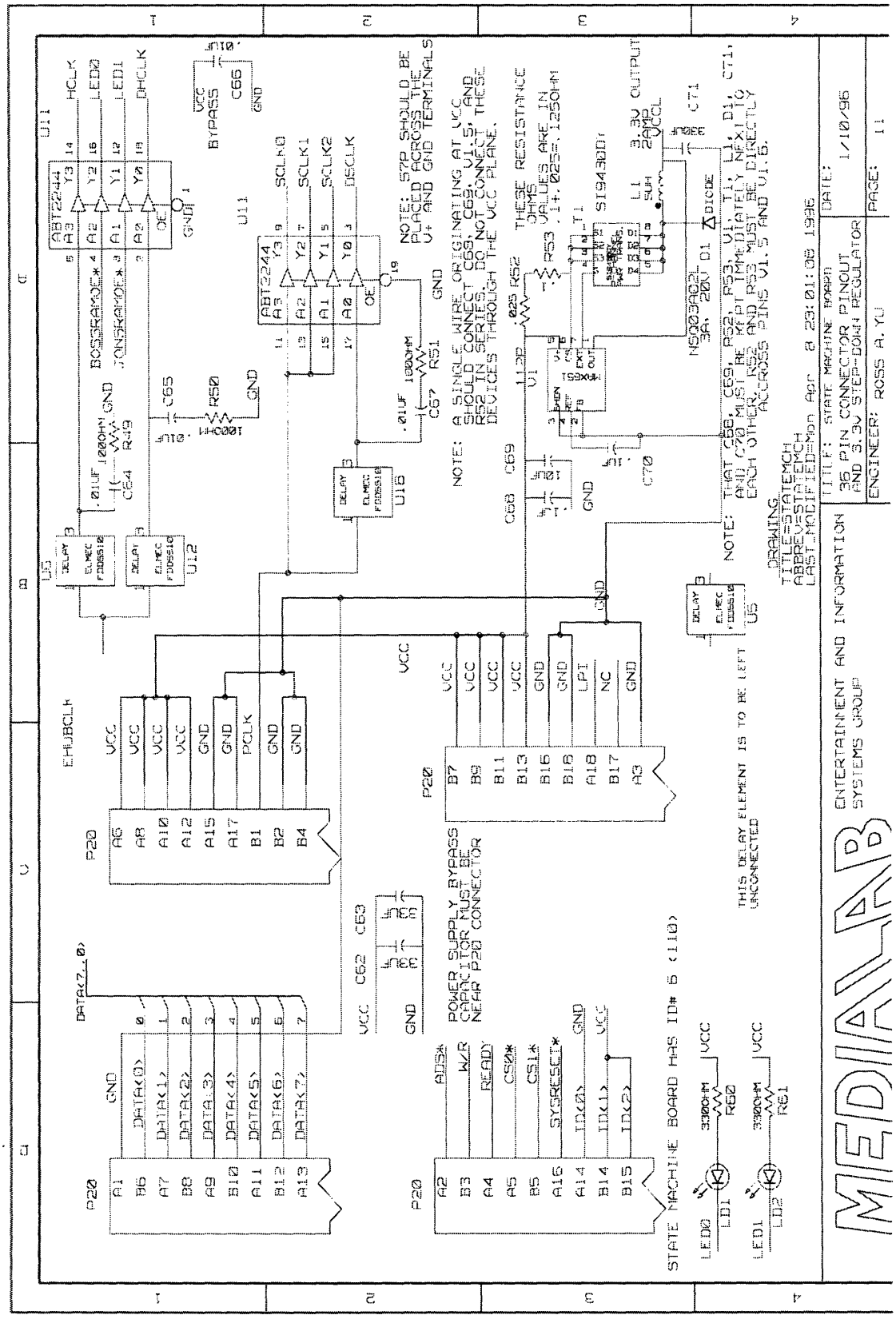

MEDIA LAB
ENTERTAINMENT AND INFORMATION SYSTEMS GROUP
TITLE: STATE MACHINE BOARD
JOHNSON TRANSFER REGISTERS
DATE: 1/10/95
ENGINEER: ROSS A. YU
PAGE: 7
DRAWING
TITLE=STATEMENCH
ABBREV=STATEMCH
LAST_MODIFIED=Mon Apr 8 21:37:19 1996
PCB_ADDR<19..0>
JUPCAB
PCBB
JMCE
PTCE
JPCBNE*
JON_ADDR<19..0>
PCB_DATA<31..0>
JON_DATA<31..0>
BOS_ADDR<19..0>
JSB_LDA*
JSB_JSRDV*
PCBR_W
SCLK0
JSD_LDA*
JSD_DRV*
JND_LDA*
JND_DRV*
U60
U61
U66
U67
U7
74ABT16841
74ABT16952
C37
C38
C39
C40
C41
C42
C43
C44
VCC
GND
.01UF
10UF

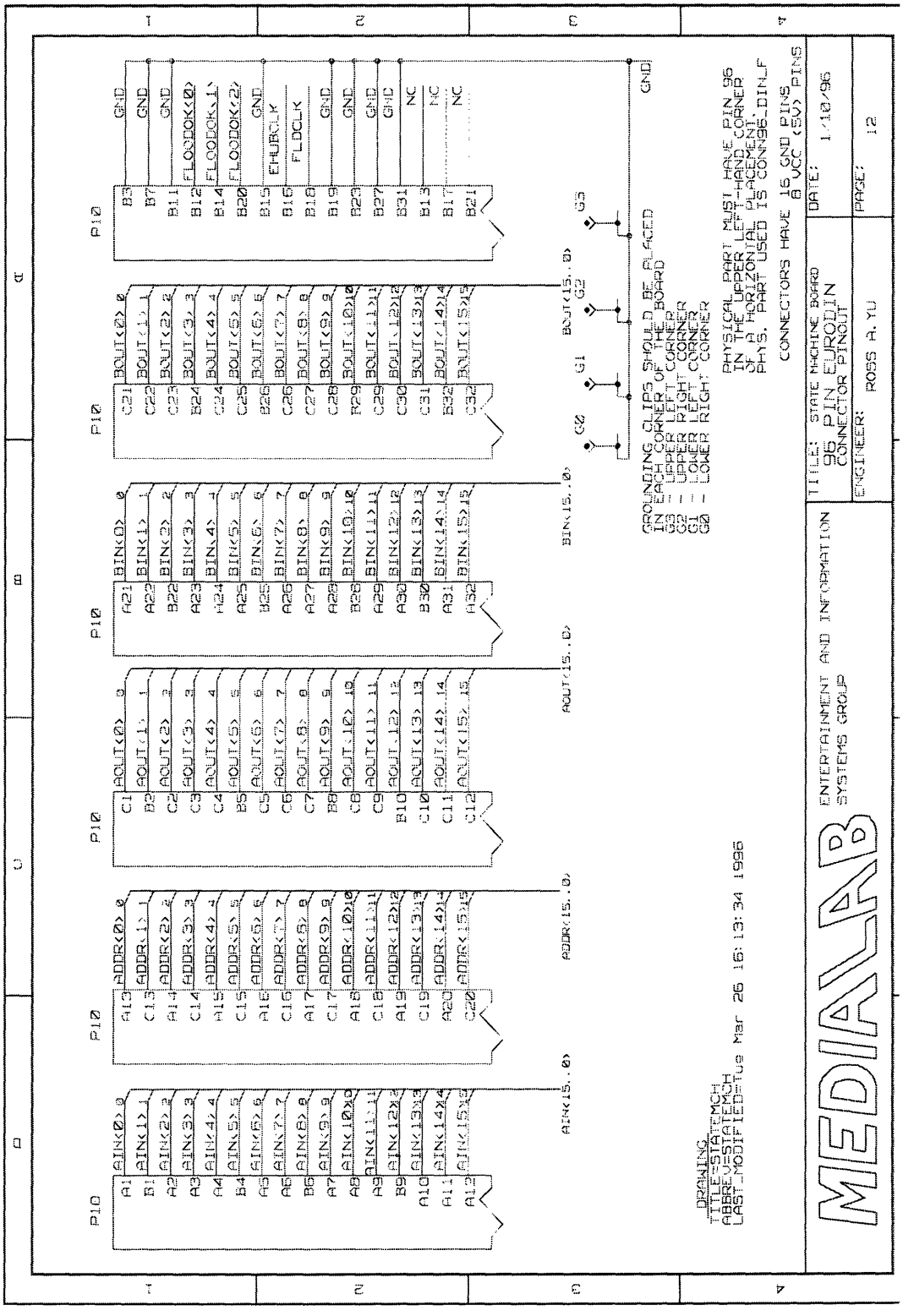


TITLE: STATE MACHINE BOARD
REGISTER INTERFACE
AND LOCAL BUS
ENGINEER: ROSS A. YU
DATE: 1/10/96
PAGE:
ENTERTAINMENT AND INFORMATION
SYSTEMS GROUP
MEDIALAB
ALTERA
EPM7128ELC84-10
STATE MACHINE
REGISTER INTERFACE
MACROCELLS: 96
84-PIN PLCC


603 OPERATING IN
REDUCED-PINOUT MODE
PC603QFP
64-BIT
SUPERSCALER
PROCESSOR
MEDIALAB
ENTERTAINMENT AND INFORMATION
SYSTEMS GROUP


SIZE BITS ARE PULLED TO
A DEFAULT VALUE OF 001
PCS_DATA<31..0>
PCSSIZ<2..0>
R48
R47
R46
R45
U55
U56
U65
U15
U63
74ABT16952
C58
C59
C60
C61
C68
VCC
GND
DRAWING
TITLE=STATEMACH
ABBREV=STATEMCH
LAST_MODIFIED=Fri Jul 12 15:37:32 1996
TITLE: STATE MACHINE BOARD
PAGE 0 SRAM AND MAR
ENGINEER: ROSS A. YU
DATE: 1/10/96
PAGE: 10
MEDIALAB
ENTERTAINMENT AND INFORMATION
SYSTEMS GROUP


MEDIALAB
ENTERTAINMENT AND INFORMATION SYSTEMS GROUP
TITLE: STATE MACHINE BOARD
36 PIN CONNECTOR PINOUT AND 3.3V STEP-DOWN REGULATOR
ENGINEER: ROSS A. YU
DATE: 1/10/96
PAGE: 11
NOTE: A SINGLE WIRE ORIGINATING AT VCC SHOULD CONNECT C69, U1.5, AND R52 IN SERIES. DO NOT CONNECT THESE DEVICES THROUGH THE VCC PLANE.
NOTE: 57P SHOULD BE PLACED ACROSS THE V+ AND GND TERMINALS
THESE RESISTANCE OHMS VALUES ARE IN .1+.025=.125OHM
3.3V OUTPUT 2AMP
STATE MACHINE BOARD HAS ID# 6 (110)
POWER SUPPLY BYPASS CAPACITOR MUST BE NEAR P20 CONNECTOR
THIS DELAY ELEMENT IS TO BE LEFT UNCONNECTED
DRAWING
TITLE=STATEMCH
ABBREV=STATEMCH
LAST_MODIFIED=Mon Apr 8 23:01:36 1996


MEDIALAB
ENTERTAINMENT AND INFORMATION SYSTEMS GROUP
TITLE: STATE MACHINE BOARD
96 PIN EURODIN CONNECTOR PINOUT
DATE: 1/10/96
ENGINEER: ROSS A. YU
PAGE: 12
CONNECTORS HAVE 15 GND PINS 8 VCC (5V) PINS
PHYSICAL PART MUST HAVE PIN 96 IN THE UPPER LEFT-HAND CORNER OF A HORIZONTAL PLACEMENT. THIS PART USED IS CONN96_DIN_F
GROUNDING CLIPS SHOULD BE PLACED IN EACH CORNER OF THE BOARD
G3 = UPPER LEFT CORNER
G2 = UPPER RIGHT CORNER
G1 = LOWER LEFT CORNER
G0 = LOWER RIGHT CORNER
DRAWING TITLE=STATEMACH ABBREVSTATEMACH LAST_MODIFIED=Tue Mar 26 16:13:34 1996

Appendix B
Special Notes on Device Resets

B.1 Special notes on 603 Hard Reset (/HRESET)

The 603's reset signal is dependent upon the mode, but is not completely determined by it. During IDLE MODE, the 603 is held in hard reset, and stays there until the board mode is switched to either CONFIG MODE or NORMAL MODE (whichever is done first). Optionally, the i960 may order the 603 into reset by writing to a bit in the register interface. In this case, the 603 remains in HRESET until this bit is deasserted, and the board mode is CONFIG or NORMAL MODE. (At this point, the 603 starts with the boot strap code.) This feature was added to allow the i960 software to reset the 603,without placing the board into IDLE Mode. It also allows the State Machine board to be used without the 603. (i.e. the FPGAs may work without the 603 initiating any bus transactions.)

B.2 Special note on ORCA Reset pin

The ORCA FPGAs may be reset by not only the Cheops hard reset button, but also by writing to the configuration control bit in the register interface. Configuration starts when this bit is asserted low then released to high. By asserting and keeping this bit low, the FPGA may be held in reset. (Note that this automatically occurs when the Cheops sysreset button is hit.) This gives both the i960 and the 603 the ability to reset the FPGAs, and possibly hold them in reset.

References

[1] Edward K. Acosta, V. Michael Bove, Jr., John A. Watlington, and Ross A. Yu. "Reconfigurable Processor for a Data-Flow Video Processing System." *SPIE Conference on Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing*. Philadelphia, Pennsylvania, October 25-26, 1995.

[2] E. K. Acosta. *A Programmable Processor for the Cheops Image Processing System.* Master's thesis, Massachusetts Institute of Technology, 1995.

[3] A.K. Agerwala and T.G. Rauscher. *Foundations of Microprogramming Architecture, Software, and Applications,* Chapter 1. Academic Press, 1976.

[4] NEC Corporation. *Memory Products Data Book Volume 2 of 2: SRAMs, ASMs, EEPROMS*, 1993.

[5] AT&T Microelectronics. *AT&T Field-Programmable Gate Arrays Data Book*, April 1995.

[6] Jeffrey M. Arnold. The SPLASH 2 software environment. In *IEEE Workshop on FPGAs for Custom Computing Machines,* pages 88-93, April 1993.

[7] Jonathan William Babb. Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulation. Master's thesis, Massachusetts Institute of Technology, February 1994.

[8] S. Casselman. Virtual computing and the virtual computer. In *IEEE Workshop on FPGAs for Custom Computing Machines,* pages 43-49, April 1993.

[9] Andre DeHon. DPGA-coupled microprocessors: Commodity ICs for the early 21st century. Transit note #100, MIT Artificial Intelligence Laboratory, January 1994.

[10] David E. Van den Bout. The Anyboard: Programming and enhancements. In *IEEE Workshop on FPGAs for Custom Computing Machines, pages 68-76,*April 1993.

[11] David Van den Bout, Joe Morris, Douglas Thomas, Scot Labrossi, Scott Wingo, and Dean Hallman. Anyboard: An FPGA-based, reconfigurable system. In *IEEE Design & Test of Computers*, pages 21-30, September 1992.

[12] Patrick W. Foulk. Data-folding in SRAM configurable FPGAS. In *IEEE Workshop on FPGAs for Custom Computing Machines,* pages 163-171, Napa California, April 1993.

[13] K. Ghose. On the VLSI realization of complex instruction sets using RISC-like components. In *Proceedings of VLSI and Computers. First International Conference on Computer Technology, Systems and Applications*, Hamburg, West Germany, May 1987.

[14] Regina L. Haviland, Greg J. Gent, Scott R. Smith. An FPGA-based custom coprocessor for automatic image segmentation applications. In *IEEE Workshop on FPGAs for Custom Computing Machines,* pages 172-179, Napa, California, April 1994.

[15] John L, Hennesy and David A. Patterson. *Computer Architecture A Quantitative Approach.* Morgan Kauffmann, San Mateo, California, 1990.

[16] IBM Microelectronics, *PowerPC 603 RISC Microprocessor User's Manual*, 1994.

[17] Christian Iseli and Eduardo Sanchez. Beyond superscalar using FPGAs. In *1993 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pages 486-490, October 1993.

[18] V. Michael Bove Jr. and John A. Watlington. Cheops: A data-flow system for real-time video processing. Technical report, MIT Media Laboratory, June 1993.

[19] V. Michael Bove Jr. and Andrew B. Lippman. Scalable open-architecture television. *SMPTE Journal*, January 1992.

[20] X.-P. Ling and H. Amano. WASMII: A data driven computer on a virtual hardware. In *IEEE Workshop on FPGAs for Custom Computing Machines,* pages 33-42, April 1993.

[21] Karin Schmidt Reiner W. Hartenstein Alexander G. Hirschbiel Michael Riedmuller and Michael Weber. A novel ASIC design approach based on a new machine paradigm. *IEEE Journal of Solid-State Circuits*, 26(7), July 1991.

[22] David M. Lewis, Marcus H. van Ierssel, and Daniel H. Wong. A field-programmable accelerator for compiled code applications. In *1993 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pages 491-496, October 1993

[23] J. Watlington. *Cheops Hardware Reference*. MIT Media Laboratory, February 1993.

[24] J. Watlington. *Stream Processor Interface*. MIT Media Laboratory, February 1994.

[25] Ian Page Wayne Luk, Vincent Lok. Hardware acceleration of divide-and-conquer paradigms: a case study. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 192-201, April 1993.

[26] Lalit Agarwal, Mike Wazlowski, and Sumit Ghosh. An asynchronous approach to efficient execution of parallel programs in adaptive architectures utilizing FPGAs. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 101-110, April 1994.

[27] M. Wazlowski and L. Agarwal. PRISM-II compiler and architecture. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 9-16, April 1993.

[28] A. Lynn Abbott, Peter M. Athanas, and Adit Tarmaster. Accelerating Image Filters Using a Custom Computing Machine. In *SPIE Conference on Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing.* Philadelphia, Pennsylvania, October 25-26, 1995.

[29] John Villasenor, Chris Jones, and Brian Schoner. Video Communications Using Rapidly Reconfigurable Hardware. In *IEEE Transactions on Circuits and Systems for Video Technology.* Vol. 5, No. 6, pages 565-567, December 1995.