

THE BUS ARBITER

*ECE298 Final Design Project
in collaboration with ECE241
ECE, University of Toronto*

Team Members:

Madhureema Dutta-
993737248;
Akshay Ahooja-
993829452

Date Submitted:

December 2, 2005

Tutorial:

#15, Ms. Jessica Gardiner

EXECUTIVE SUMMARY

A request for proposal was presented to the design team of Akshay Ahooja and Madhureema Dutta to design and implement a digital system. This report presents the analysis of the digital system developed by the University of Toronto design team comprising of the aforementioned members for the ECE298 and ECE241 courses. The proposed system is called the “Bus Arbiter”, designed using Quartus software which implements Arbitration Protocol between CPU and DMA, and Parallel and Serial memory modules. The design resembles the Arbitration Protocol of a computer at a very minor level, but includes all the complexities of the Protocol. This design not only had a 95% success rate but it also had some added features to make the design original and user friendly. The Arbiter acts as a “traffic controller” which permits the high priority device (CPU) to interrupt the progression of the low priority device (DMA). The display of arbiter signal on an LED, display of the contents of the two memory modules on the two seven-segment displays on the UltraGizmo board and the demonstration of the transfer of data from devices and memory modules on the VGA screen make the design user friendly.

The report is divided into six main sections that include system overview, user interaction, component description, design description, testing, engineering decisions, and project management. The system overview gives a basic idea of the working of the design. The user interaction details the step-by-step procedure the user needs to follow to work the design. The component description section described the components and their working and then the design description is the working of the integrated system. Testing and engineering decisions details the failed and successful decisions and also enumerates the added features of the design. Lastly, the project management section consists of the project planning and proposed schedules that were established for the design team to follow.

This report discusses all the factors affecting the design of this digital system and the process through which the design was created.

TABLE OF CONTENTS

	Page #
Introduction.....	1
System Overview.....	2
User Interaction	2
Component Description	3
Finite State Machines	4
Other Components.....	6
Design Description	7
Sub System Design	7
Full System Design	9
Testing.....	10
Engineering Decisions	12
Project Management	12
Conclusion.....	13
Appendix A – Important Terms	14
Appendix B – Flow Diagrams	15
Appendix C – State Diagrams.....	17
Appendix D –MUX Configuration.....	19
Appendix E – Waveforms and Schematics	20
Appendix F – Gantt Chart.....	22
Appendix G – Source Code**.....	23

Only included in **241/298 Technical TA version

INTRODUCTION

The present report is an in-depth design analysis of the digital system built by Akshay Ahooja and Madhureema Dutta as a requisite for the ECE241 and ECE298 courses. A request for proposal was presented to the team to design and implement an advanced multi-staged digital system in a creative way with the possibility of using a VGA display via finite state machines. The main purpose of the project was to gain experience dealing with the design of a larger digital system and to deal with the issues in going from a large fuzzy idea to a concrete multi tiered digital system.

To meet this request the team built a hardware based design called the “Bus Arbiter” using Altera Quartus software which implements Arbitration Protocol between different devices and memory modules. The design was kept within scope by selecting the two primary devices, CPU and the Direct Memory Access (DMA), and two memory modules. The design provides an insight on the reading and writing of user requested data to user specified memory modules and copying and swapping of data to user specified memory modules. Indeed the design meets all the expectations of the project and was graded 95% successful by the ECE241 TA.

In addition to the basic model of an arbitration protocol, the design also has some added features which make it unique. The Arbiter which is a major component of the design, acts as a “traffic control” which permits the high priority device (CPU) to interrupt the progression of the low priority device (DMA). The arbiter signal is also displayed on an LED which makes the design user friendly. Another user friendly feature of the design the display of the contents of the two memory modules on the two seven-segment displays on the UltraGizmo board. The transfer of data from devices and memory modules is also demonstrated on the VGA screen. The design also has a “swap” feature which allows the user to swap data between any two memory addresses. Hence the design not only meets the required specs of the project but its added features make it unique and user friendly.

SYSTEM OVERVIEW:

The Bus Arbiter is a multi-stage system that slightly mimics the process of several prioritized devices sending information across a bus to their respective memory modules. At the user end, there are four choices available based on the input. One can read or write to the memory modules by sending in an address and desired data. Furthermore, the user can copy and swap data between memory modules by sending in two addresses. An additional feature in this design is the ability for the top priority device to interrupt the lower priority device. If the lower priority device is processing information, the higher priority device can interrupt, complete its process and go back to the lower level device to finish its previous procedure. It must be noted that the lower priority device cannot be activated while the higher priority device is busy.

This digital system was designed entirely on Altera Quartus using its Schematic and Verilog HDL editors. It was then burned onto a Flex10K board (25Mhz Clock) and used in combination with the Ultragizmo Board.

USER INTERACTION:

In order for the user to use any of the above features, the following steps must be followed:

Please refer to the User Interaction Chart (FAST Diagram) in **Appendix B**.

1. Turn on System: Place on_off switch to high
2. Select Read, Write, Copy, or Swap
 - a. For Read/Write, place *rw_dma* switch to high
 - Read: place *rw_cs* switch to low
 - Write: place *rw_cs* switch to high
 - b. For Copy/Swap, place *rw_dma* switch to low
 - Copy: place *rw_cs* switch to low
 - Swap: place *rw_cs* switch to high
3. Start: Place Go switch to high
4. For Read:

- a. Enter Address: Set up address in the *DIP Switches* and set *ready* switch to high
- b. Data in address entered will show on its respective 7-Segment display

For Write:

- a. Enter Address: Set up address in the *DIP Switches* and set *ready* switch to high
- b. Set *ready* switch back to low
- c. Enter Data: Set up data in *DIP Switches* and set *ready* switch to high
- d. Set *ready* switch back to low

For Copy / Swap:

- e. Enter First Address: Set up address in the *DIP Switches* and set *ready* switch to high
- f. Set *ready* switch back to low
- g. Enter Second Address: Set up address in *DIP Switches* and set *ready* switch to high
- h. Set *ready* switch back to low

5. Stop: Place *Go* switch to low

NOTE: interruption of ready or write can only occur while copying or swapping. If at any point in Step 4, an interruption can be instantiated by toggling the *Go* switch from high, to low, back to high. The interruption will go through Step's 4-5, and once completed, will return back to pre-interrupt state.

COMPONENT DESCRIPTION

To implement the above features, the design was broken up into four FSM's: Central Processing Unit (CPU), Direct Memory Access (DMA), Read-Write-Copy-Swap Controller (RWCS), and Video Graphics Array (VGA). There were three other major system components also: The Arbiter, Parallel Input/Output Memory Module, and Serial Input/Output Memory Module.

FINITE STATE MACHINES

CENTRAL PROCESSING UNIT (CPU):

The CPU is a top priority device responsible for the reading and writing features. It is the most important FSM, as it holds the power over the entire design. All the initial inputs of the user (excluding memory storage values) are directly connected and controlled by the CPU. Once the CPU examines initial user inputs, after the machine is turned on using the *on_off* switch and started using the *Go* switch, it communicates with the other FSM's accordingly. The CPU and DMA collaborate to make the Central Control Station.

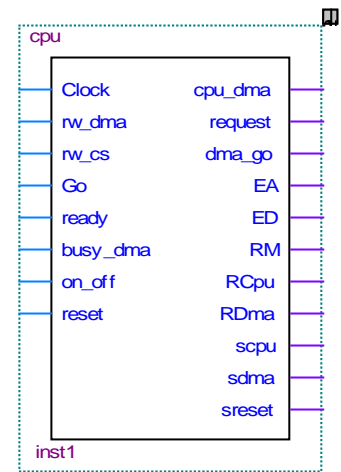


Figure 1: CPU Symbol

The CPU controls the master reset signal (*on_off* switch), as well as the request line, when it is ready for more data. The CPU practices “handshaking”, a familiar protocol for communication between two devices, in order to get data from the user. This FSM stays in a steady state as long as the user has not toggled the *Go* switch. Once it has been toggled, the CPU reads in what feature the user would like to use (explained in the User Interaction section). If the feature is a CPU feature (reading or writing), it requests for the first address and enables the “A” register to store the value once it is obtained. The value is obtained by toggling the *ready* switch once the data is set up on the *DIP Switches*. If the feature is set to “write” it will then again perform handshaking to obtain the data, and then enable the “D” register in order to store that value.

DIRECT MEMORY ACCESS (DMA):

The DMA is the other major FSM that is responsible for the copying and swapping features. It is also responsible to know how much of its task it has completed at each stage, so that if an interruption is to occur, it can start off where it left off, once it is allowed to continue. It does this by storing a “0” in the “P_CS” register if it is copying, and a “1” if it is swapping. Furthermore, once the first address is

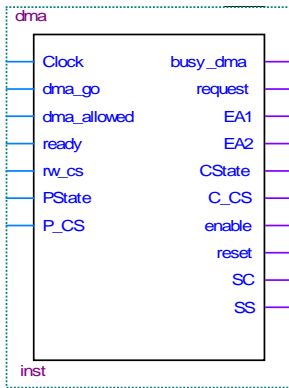


Figure 2: DMA Symbol

obtained, this FSM enables the “PState” register to store a “1”, signaling it has crossed the first step to completing its task. At every instance the DMA is activated, it refers back to these registers to know where to begin procedure from. For both copying and swapping features, the DMA performs handshaking with the user to obtain the first and second address. It then enables registers “A1” and “A2”, respectively, to store the inputs. The CPU and DMA collaborate to make the Central Control Station.

READ-WRITE COPY-SWAP CONTROLLER (RWCS):

The RWCS FSM is responsible for interacting with the memory modules and carrying out the read, write, copy and swap tasks. It has the ability to enable two registers, “D1” and “D2”. “D1” is connected directly to the memory modules for writing purposes, while “D2” is a register to hold a temporary value while implementing a swap. The following is a basic outline of the four features:

Read: on the input of an address, A1, searches in respective memory module and displays its data

Write: on the input of an address, A1, and data, D1; the write feature stores D1 in A1

Copy: on the input of two addresses, A1 and A2; data from A1 is copied into memory location A2.

→*Procedure:* Data of A1 stored in D1, and written into address of A2.

Swap: on the input of two addresses, A1 and A2, data from A1 is stored in A2, and the data in A2 is stored in A1 simultaneously.

→*Procedure:* Data of A1 stored in D1, and data of A2 stored in D2. Data in D2 stored in address in A1, and data in D1 stored in address in A2.

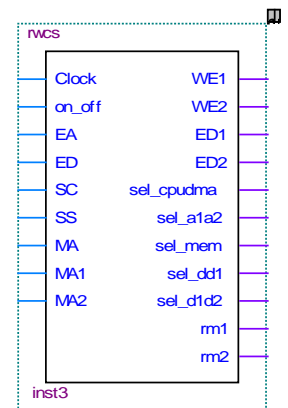


Figure 3: RWCS Symbol

VIDEO GRAPHICS ARRAY (VGA):

The VGA FSM works in conjunction with VGACon¹ in order to display the implementation of each feature on the VGA Display. It waits for a start signal from one of the devices or memory modules, and then writes one pixel on the device which is working, and one pixel on the memory module it is interacting with. **Figure 4** shows an

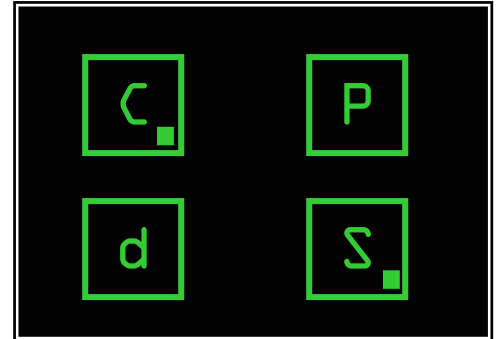


Figure 4: VGA display of the CPU writing data into Serial I/O memory module

example of the CPU writing data into the Serial Input/Output memory module (memory modules are discussed on page 7).

Using the *write_request* input, and the *write_allowed* input (from VGACon) VGA follows the following basic procedure, similar to handshaking, in order to communicate with the VGA screen via VGACon:

1. Set *write_request* to “1”, and if *write_allowed* is not “1”, wait for the high signal
2. Once *write_allowed* is “1”, keep *write_request* at “1” for one clock cycle.
3. Set *write_request* to “0” and wait for *write_allowed* to go back to “0”

When sending in the request signal, the colour, row, and column are also specified on the *colour[2..0]*, *row[5..0]*, and *column[5..0]* outputs respectively. Once a new procedure is begun by the user, the green pixels written during the previous procedure are rewritten by black pixels following the same procedure explained above. The initial screen is provided to the VGA by a MIF file.

OTHER COMPONENTS:

THE ARBITER:

The Arbiter is the traffic controller of the Central Control Station. Its output is an input into the DMA called *dma_allowed*, which is only at “1” (the DMA is allowed to be active) when the CPU is not

¹ VGACon is a VHDL module written by Deshanand Singh and edited by Aaron Egier as a communicator between any self written module and the VGA screen. Nov 26, 2004

busy. The DMA is only allowed to start or continue with its process while *dma_allowed* is at “1”. The Arbiter consists of an XOR gate with *cpu_dma* (at “1” when CPU is busy), and *on_off* switches as the inputs. The Arbiter acts as a passageway towards the memory modules.

MEMORY MODULES:

There are two memory modules in this digital design; Parallel Input/Output memory module and Serial Input/Output memory module. The memory locations entered in by the user are between the hexadecimal values 0_{16} and F_{16} ($0000_2 - 1111_2$). The Parallel I/O memory module consists of the addresses between 0_{16} and 7_{16} ($0000_2 - 0111_2$), while the Serial I/O memory module consists of addresses between 8_{16} and 15_{16} ($1000_2 - 1111_2$). Both memory modules are synchronous with the clock, and can hold 4-bit words. The results of the reading memory modules are displayed on two 7-Segment displays, one for each module. The storage registers between the incoming inputs and the memory modules are known as the **Input Data Accumulator**.

DESIGN DESCRIPTION

Before discussing the full system design, it is necessary to look into the inner workings of some important subsystems. How the interruptions work between the CPU and DMA, and the way the memory is processed into each module are two important subsystems to look at.

SUB SYSTEM DESIGN:

MEMORY ALLOCATION PROCESSING: (*refer to Appendix D for further information*)

In order for data to be transferred from the Input Data Accumulator to its respective memory modules, a Memory Allocation Processing system was designed. This was necessary since the memory modules only have one input for data and one input for address, while there are outputs of data and addresses coming from multiple different registers in the Input Data Accumulator. This system is a collection of multiplexers all controlled by the RWCS FSM in order to control the flow of data.

The first multiplexer, *mux_CpuDma*, controls flow based on what FSM is controlling the activated procedure. For example, if the user is writing, *mux_CpuDma* will only allow data flow from registers “A” and “D” (CPU controlled address and data registers). Similarly, if the user is swapping, a DMA implemented feature, *mux_CpuDma* will only allow data from registers “A1” and “A2” (DMA controlled address’s registers).

The second multiplexer, *mux_AA1_DA2*, controls whether it wants information from the first user input or the second user input, depending on what process it is computing. The third multiplexer, *mux_mem*, controls storage of data being read from the Parallel I/O or Serial I/O memory module. There is also a multiplexer, *mux_DD1*, connected to the “D1” register (register connecting to data lines of both memory modules). During writing, the “D” register (controlled by CPU) has data flow, and while copying or swapping, the contents out of *mux_mem* have the data flow. The final multiplexer in Memory Allocation Processing, *mux_D1D2*, is used for the purposes of copying and swapping. As stated above, the “D1” register has connection to the data lines of the memory modules, but this is so through this multiplexer. The other choice is register “D2”, which stores a temporary value necessary during swapping procedures.

HIGH PRIORITY INTERRUPTION: (*refer to Appendix E for further information*)

The high priority interruption gives certain devices priority over the others. In this design, this is provided inside the Central Control Station. The CPU has top priority, while the DMA has second priority. The interruption can occur while the DMA is busy. While getting either the first address or the second address, the user can toggle the Go switch with “*cpu_dma*” at “0”, in order to activate a CPU feature. Once the CPU has completed its task, control will be given back to the DMA to complete its task where it left off.

First is a reminder of how the DMA knows where it left off its last task. It successfully knows this by storing a “0” in the “P_CS” register if it is copying, and a “1” if it is swapping. Furthermore, once the first address is obtained, this FSM enables the “PState” register to store a “1”, signaling, it has crossed

the first step to completing its task. Also, while the DMA is busy, it constantly sends a *busy_dma* signal to the CPU.

When the machine is turned on, the *dma_allowed* signal is set to high. But if there was an interruption, it is suddenly set to low, and that sends DMA into its initial stages, with *busy_dma* still on high. Once the CPU completes its task, it checks for this *busy_dma* signal, and as in this case it would be on high, it will reactivate DMA. Once DMA begins in its initial stages it checks the “P_CS” and “PState” registers, and goes to the appropriate stage accordingly. At the end of DMA’s process, the two registers are reset.

Simulation waveforms of this priority interruption can be found in Appendix E.

FULL SYSTEM DESIGN

With knowledge of each component and subsystem, the full system design can now be discussed. Once the design is loaded onto the board, all the FSM’s are on their initial stages, and all registers are reset. Only the CPU moves on to its next stage when the system is turned on using the *on_off* switch.

On the basis of the user inputs (refer to User Interaction on page 2), the CPU either performs the reading/writing task or sends control over to the DMA to perform the copying/swapping task. The DMA is activated using a *dma_go* signal which is pulsed by the CPU. The DMA stays in its initial stages before the pulse. Once the data has been collected from the user via CPU or DMA, and stored in their appropriate registers inside the Input Data Accumulator, there is a pulse going to RWCS and VGA for read, write, copy and swap, named *EA*, *ED*, *SC* and *SS*, respectively. This activates RWCS and VGA to go to its read, write, copy or swap stage to start the appropriate procedure. Selecting the proper multiplexers in the Memory Allocation Processing and enabling the *write_enabled* signals in either memory module, the proper feature is implemented. There are secondary registers “Mn” for each address register “A”, “A1” and “A2” that hold the most significant bit of the address. This is used as an

indication for which memory module to store the data in. If the value in “Mn” is “0”, then it is stored in the Parallel Input/Output memory module, while if it is “1” it is stored in the Serial Input/Output memory module. Also, using the procedure outlined in the Video Graphics Array section on page 4 the proper pixels are printed onto the screen. Also, the locations of memory being read will be displayed on the Seven-Segment displays. There are two displays showing one memory module each. LED’s also display the *request*, *Arbiter* and *on_off* signals.

Once the tasks are completed, all FSM’s go back to their initial stages, while the CPU waits for the next *Go* signal. If *on_off* is to go to low, all registers in the Input Data Accumulator will be reset, as well as the CPU will go to its initial stage.

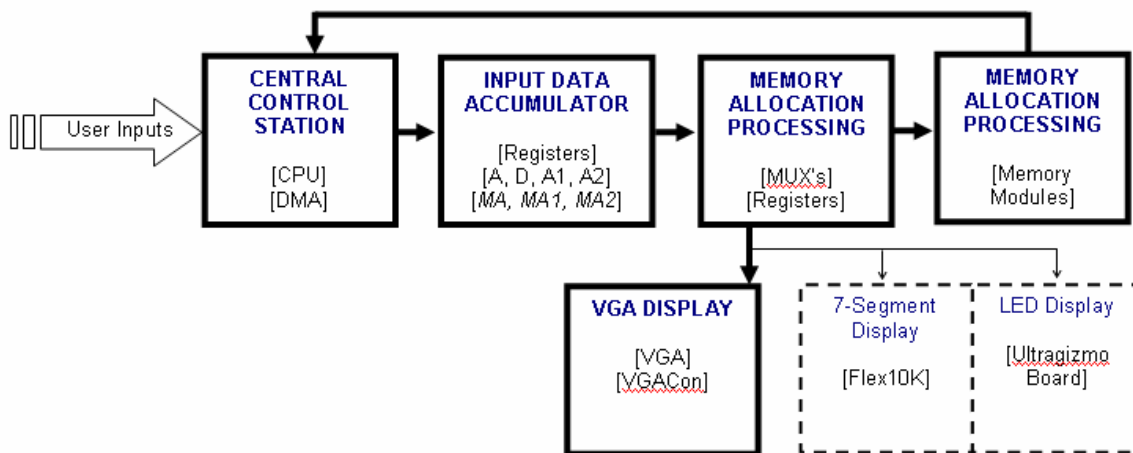


Figure 5: Integrated System Flow

TESTING

The testing of the design was conducted in two stages. The first stage consisted of module wise testing and the second stage was the testing of the integrated system.

MODULE-WISE TESTING

The initial testing was performed on each component separately. Each FSM was separately tested by its corresponding simulations in Altera Quartus. Extensive individual simulations were performed to exhaust all possible test cases. Quite a few errors were discovered during this initial

testing process. One of them was that the *cpu_dma* signal continued to stay high even after the CPU had finished processing. Hence it created major problems because the *dma_allowed* signal never went to high and the DMA was never allowed to process. The error was found out by tracing the CPU FSM Verilog code which showed that in some stages of the FSM the *cpu_dma* was not changed back to low.

Another important crisis was to fix the “interrupt” feature. To keep the stage of the DMA conserved during interruption, two temporary storage registers were used- one to indicate if both addresses had been input and the other to indicate if DMA was copying or swapping. The addition of these two registers made the “interrupt” feature 100% flawless.

Initially four different FSMs were designed to perform the read, write, copy and swap features. Testing showed that memory module consisted of a single enable signal. Hence to accommodate four different FSMs a four option selection multiplexer was required which created simulation errors. Hence the four different FSMs were integrated as one RWCS FSM which performed all the four functions based on user specified selection and the idea for the multiplexer was dropped.

After detailed testing and debugging of each individual module the design was integrated and testing was performed on the integrated system.

INTEGRATED TESTING

Integrated testing was done by simulations and testing the completed design on the Altera board. Incorrect results were displayed on the seven segment displays which were caused by inaccurate connections in the final schematic. The read, write and copy features worked correctly but problems were encountered with the swap feature. The Quartus simulation showed correct results but it failed to work on the Altera board. Due to time constraints the error could not be fixed.

Another problem encountered was with the VGA display. Quartus failed to recognize the VGA FSM as a state machine due to inexplicable reasons. Two ECE241 TAs looked into the matter and

came to a conclusion that it may be due to a bug in Quartus. Due to these reasons integrated testing couldn't be completed and hence the design couldn't be made perfect.

ENGINEERING DECISIONS

The “Bus Arbiter” design implemented by the design team has added features which make it original and user friendly. The greatest feature of the design is the priority queue in which the CPU is given priority over the DMA. This feature was added to resemble the actual design of a computer. For effective working of the priority queue, the interrupt feature was added to the design which allowed the high priority device (CPU) to interrupt the functioning of the low priority device (DMA) and after the processing of the CPU the DMA resumes from the point it left off. This feature added complexity to the design and made it original.

Another important engineering decision was that except the *on_off* signal all other signals were made toggle ready and not steady state signals. So if the user by chance flips any switch back while the design is in operation, it doesn't obstruct the processing.

The arbiter signal is displayed on an LED for easy comprehension of the user and the contents of the two memory modules were displayed on the seven segment displays of the Ultrazimo board to reflect the accuracy of the design. Also, the transfer of data between the devices and the memory modules is displayed on the VGA screen. All these added features and the engineering decisions taken make the design original and user friendly.

PROJECT MANAGEMENT

The design team decided on milestones for each lab period and prelab time and distributed the work accordingly. Smaller components of the design were written individually but most of the FSMs were written and the testing was done as a team to minimize integration problems. Most of the allotted

time was consumed in debugging of FSMs and hence due to time constraints the design turned out to be 95% successful.

Refer to the Gantt chart provided in **Appendix F** for details regarding milestones set and tasks completed.

CONCLUSION

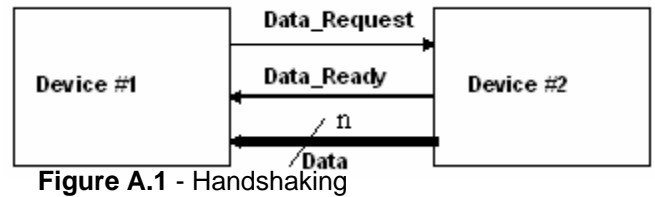
The “Bus Arbiter” design proved to be a fairly successful digital project and provided a lot of practical experience to the design team. The swap feature which worked perfectly in simulation failed to work when loaded on the Altera Flex 10K board. All other features performed exactly according to the theoretical design. Correct data was read and written into correct memory modules and also rightly copied between memory addresses. The seven segment display proved the accuracy of the design. The VGA display couldn't be made to operate due to inexplicable behavior of Altera Quartus software. Though the “Bus Arbiter” resembles the Arbitration Protocol of a computer at a very minor level, it includes all the complexities of the Protocol and so the complexities of the Hex keypad were avoided by using the *DIP Switches* for input. Considering the complexity of the project and the time constraints the project was judged 95% successful by the ECE241 TA. Hence the “Bus Arbiter” can be judged a successful completion and a learning experience.

APPENDIX A

IMPORTANT TERMS

Handshaking: (Excerpt from ECE241F Lab 6 – J. Rose, University of Toronto, ECE)

To transmit data between two devices, it is often necessary to provide what are called “handshaking” signals that ensure that the data is received correctly, particularly when two devices are running at very different speeds. Consider the situation illustrated in **Figure A.1**, in which n bits of data are to be transmitted from Device #2 to Device #1. When Device #1 requires new data, it raises the Data_Request line high (to “1”). Once Device #2 sees this and has placed the correct data on the n Data lines, it raises the Data_Ready line high. When #1 has taken the data (typically by storing it in a D- register) it lowers the Data_Request line after which #2 lowers the Data_Ready line. Device #1 can only raise a new request *after* the Data_Ready line is lowered. This procedure is called a “full handshake” and ensures that the data is transferred correctly, even when the two devices are running at vastly different speeds.



High: setting a desired input/output to “1” – V_{cc} .

Low: setting a desired input/output to “0” – ground.

Pulse: To turn a signal to high for one clock cycle, instead of keeping it on for an indefinite period of time.

Toggle: to change the signal of a switch. In the case of this system, to change it from $0 \rightarrow 1 \rightarrow 0$ or from $1 \rightarrow 0 \rightarrow 1$, depending on what its previous state is.

MIF File: a file that is hard coded into VGACon, as an initial file to display once the VGA is displayed.

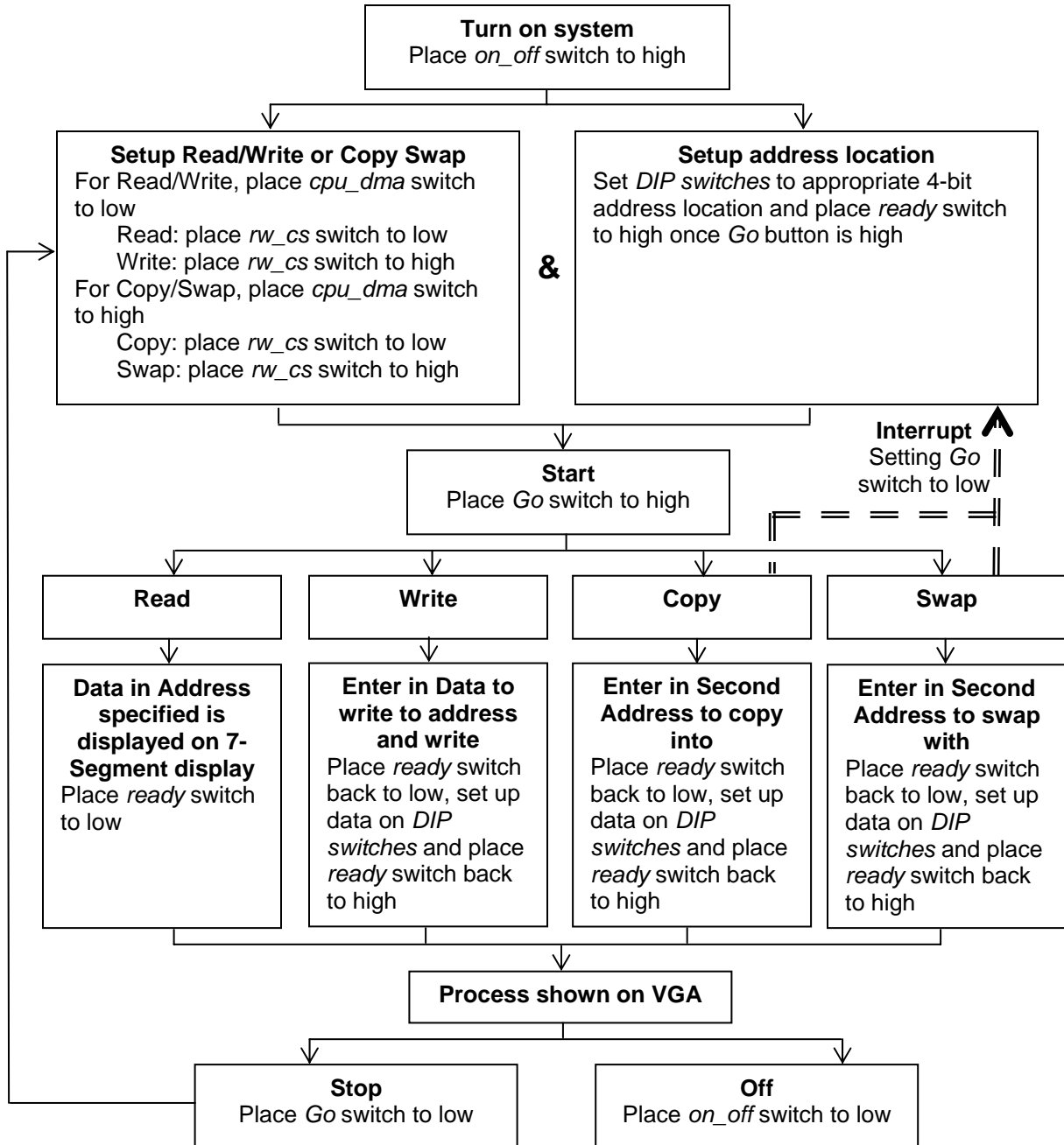
XOR Gate: follows the following truth table

X_0	X_1	F
0	0	0
0	1	1
1	0	1
1	1	0

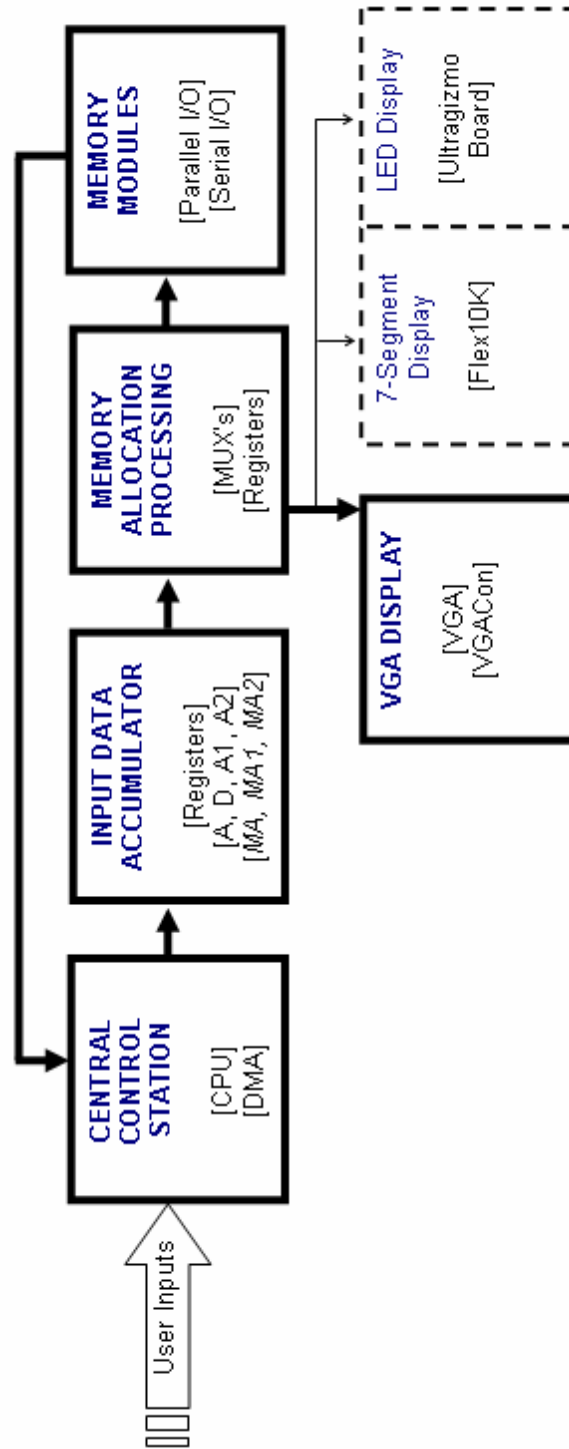
APPENDIX B

FLOW DIAGRAMS

FAST Diagram



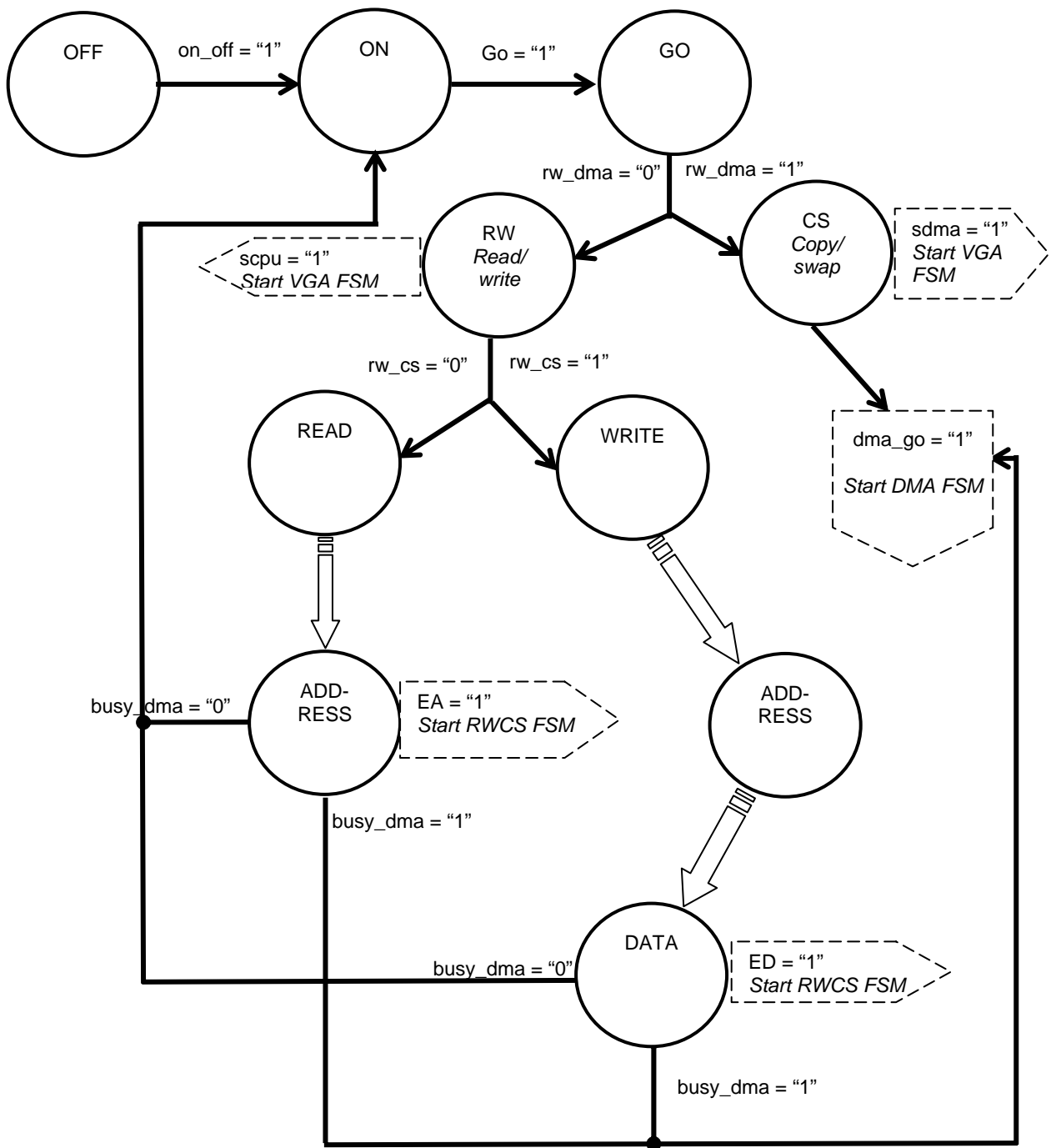
Signal Flow Diagram



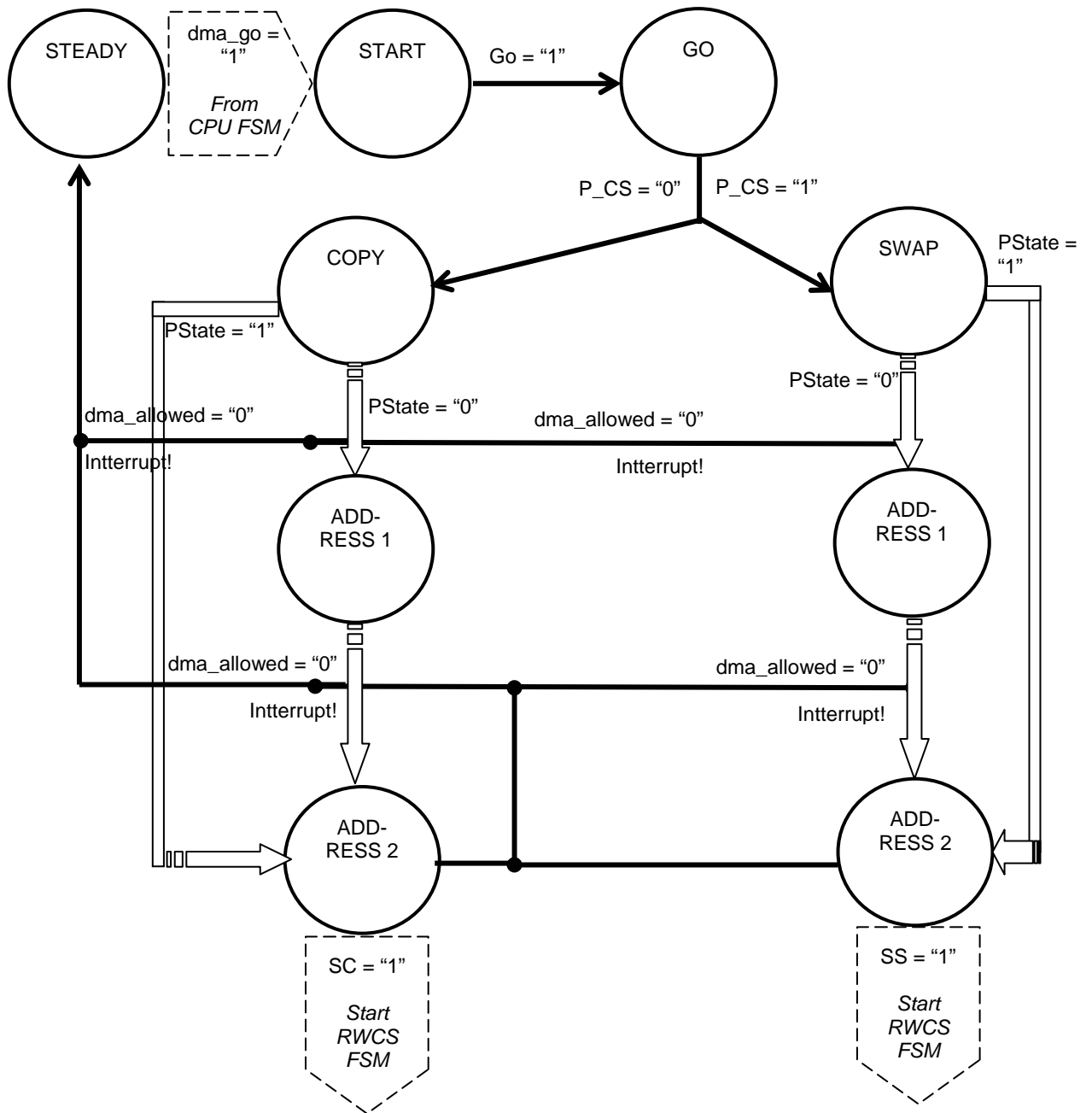
APPENDIX C STATE DIAGRAMS

CPU FSM

 = handshaking (refer to Appendix A for more information)

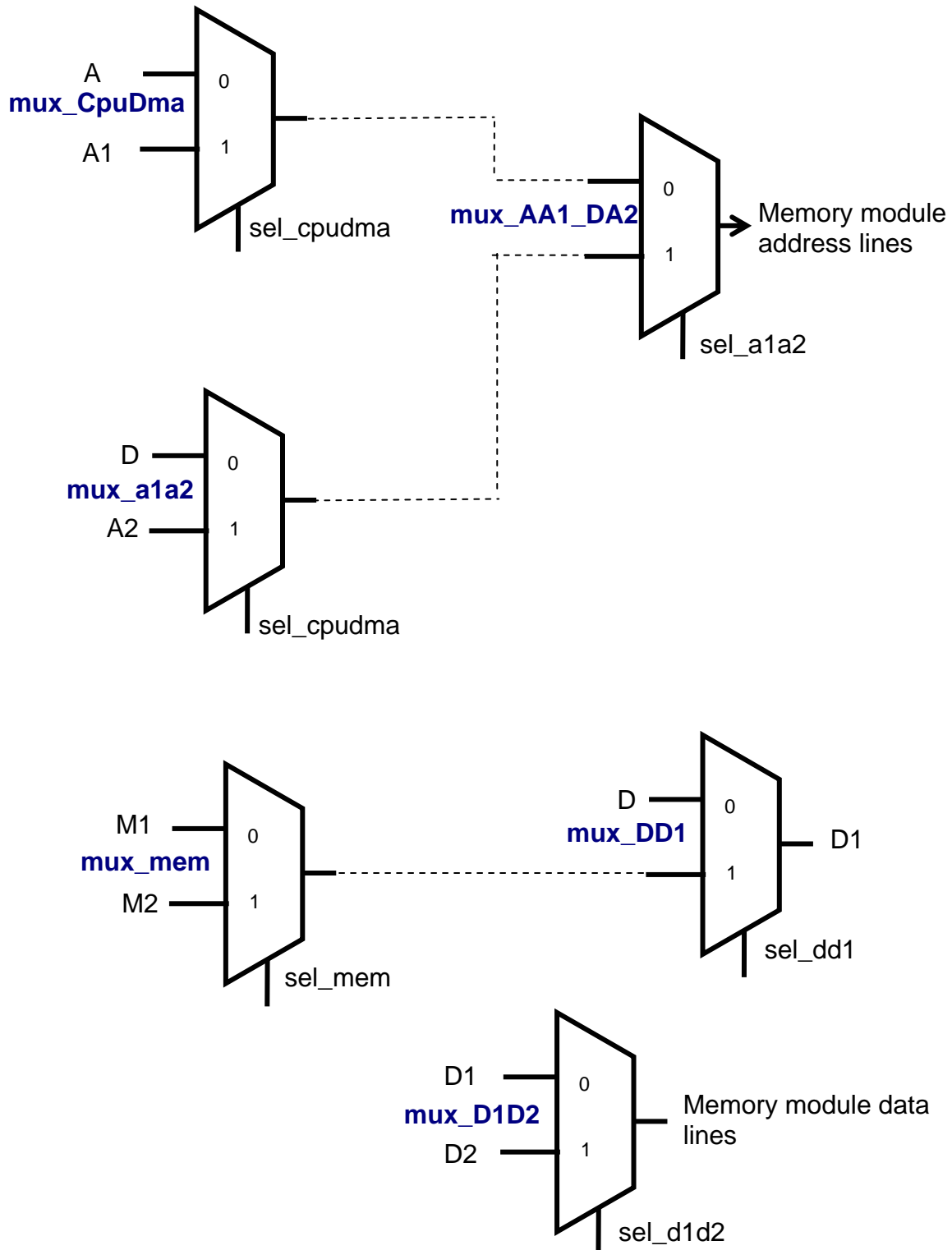


DMA FSM



⇔ = handshaking (refer to Appendix A for more information)

APPENDIX D
MEMORY ALLOCATION MULTIPLEXER CONFIGURATION



APPENDIX E

WAVEFORMS AND SCHEMATICS

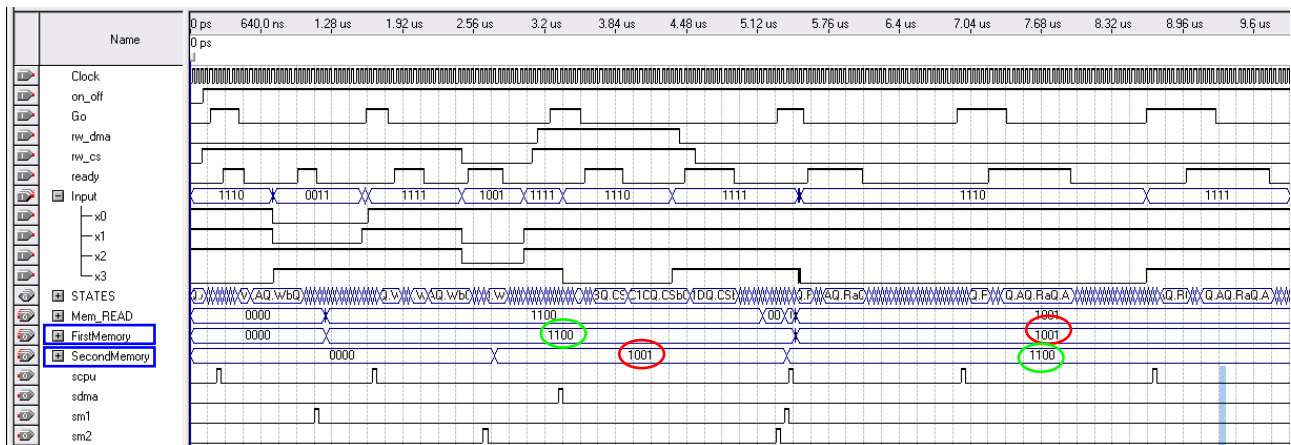


Figure E.1: Integrated Design - Swap simulation

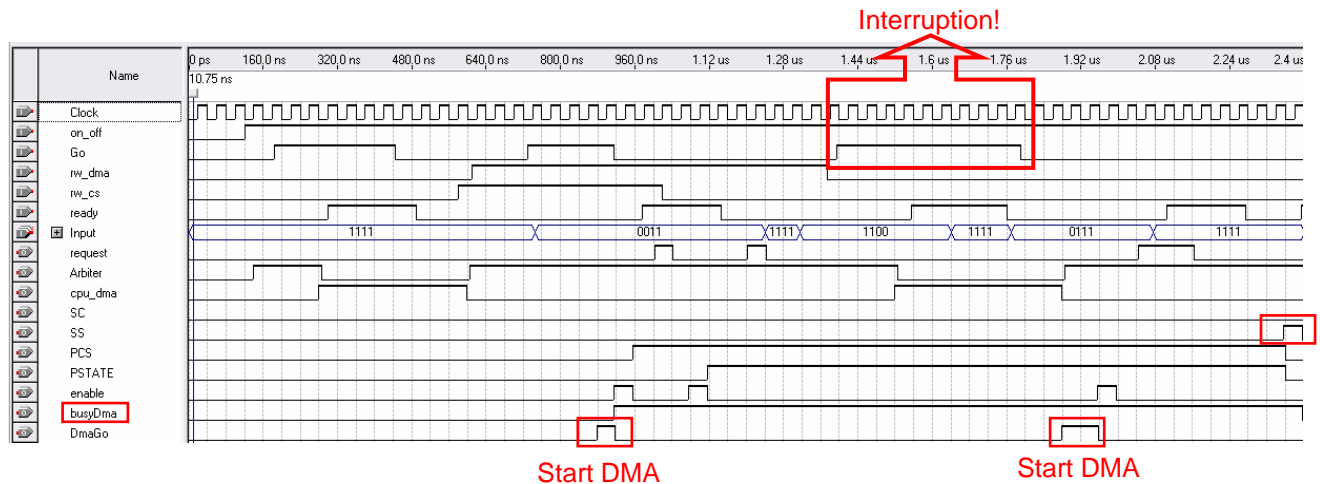


Figure E.2: Integrated design - interruption simulation

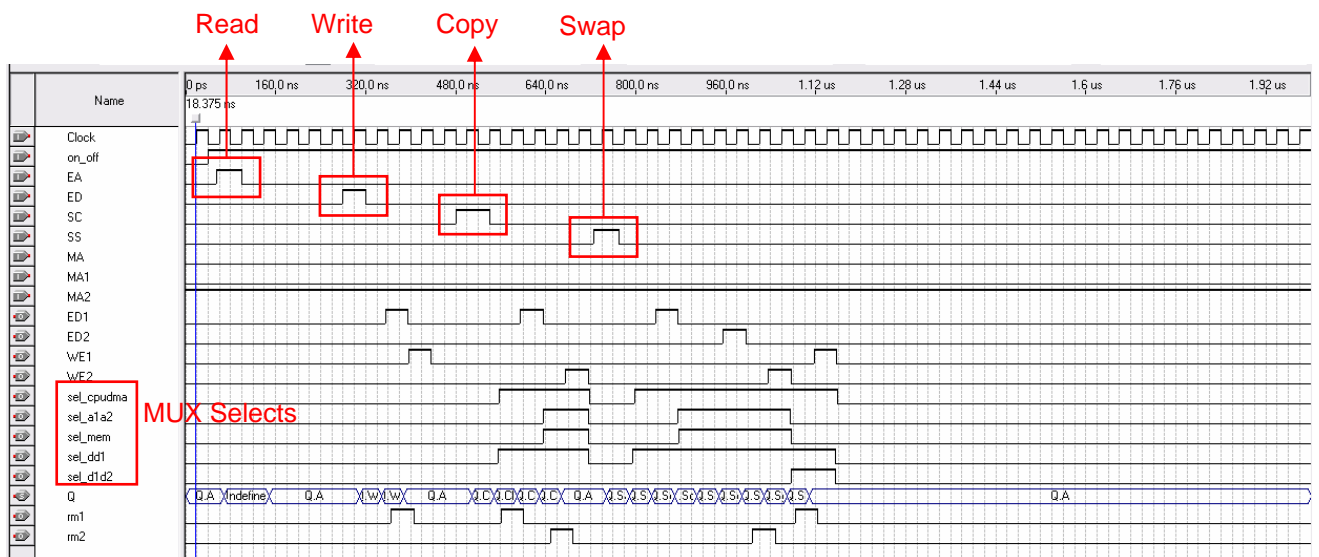
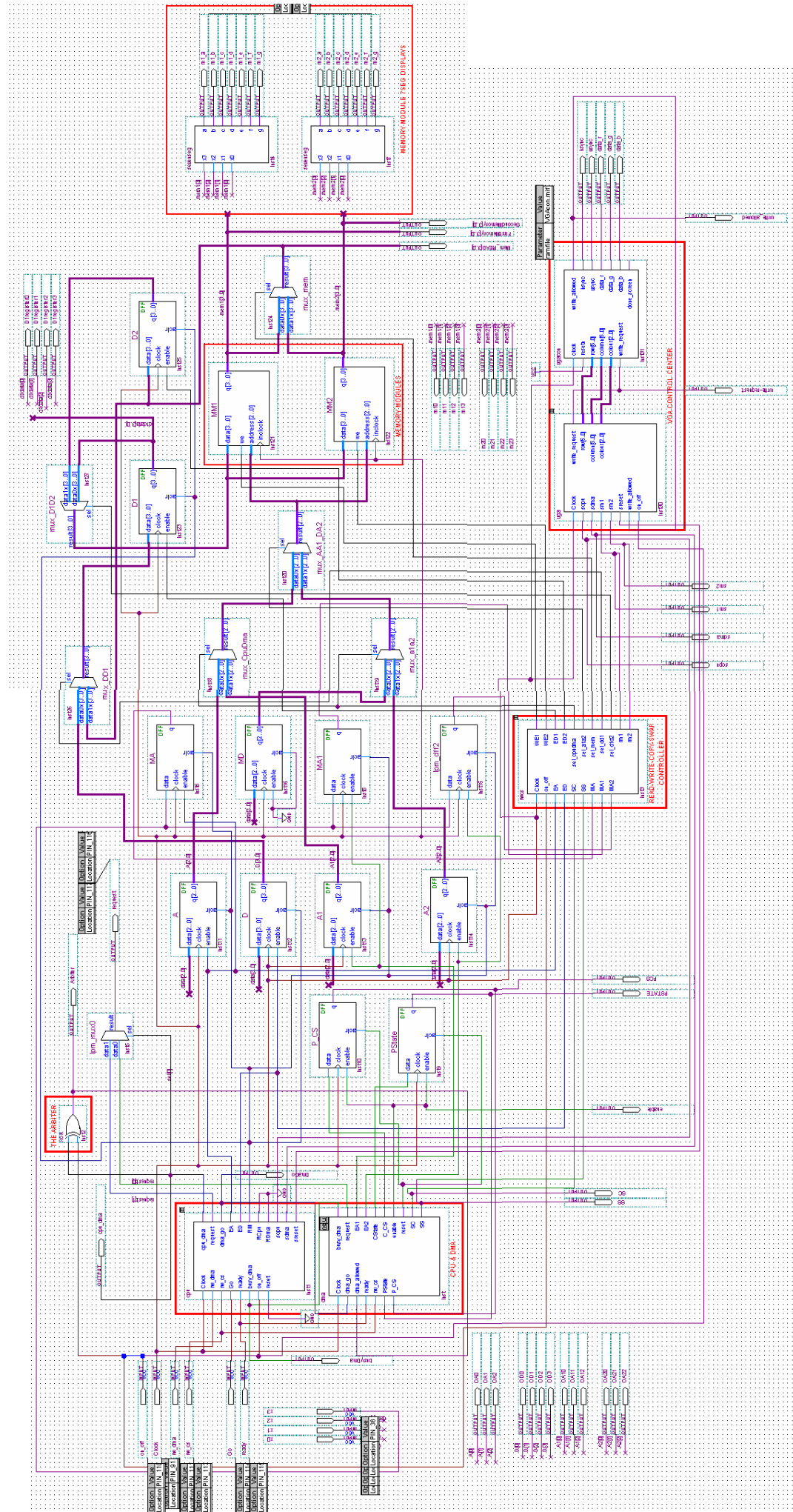


Figure E.3: RWCS – read, write, copy and swap simulations

INTEGRATED DESIGN: SCHEMATIC



APPENDIX F

GANTT CHART

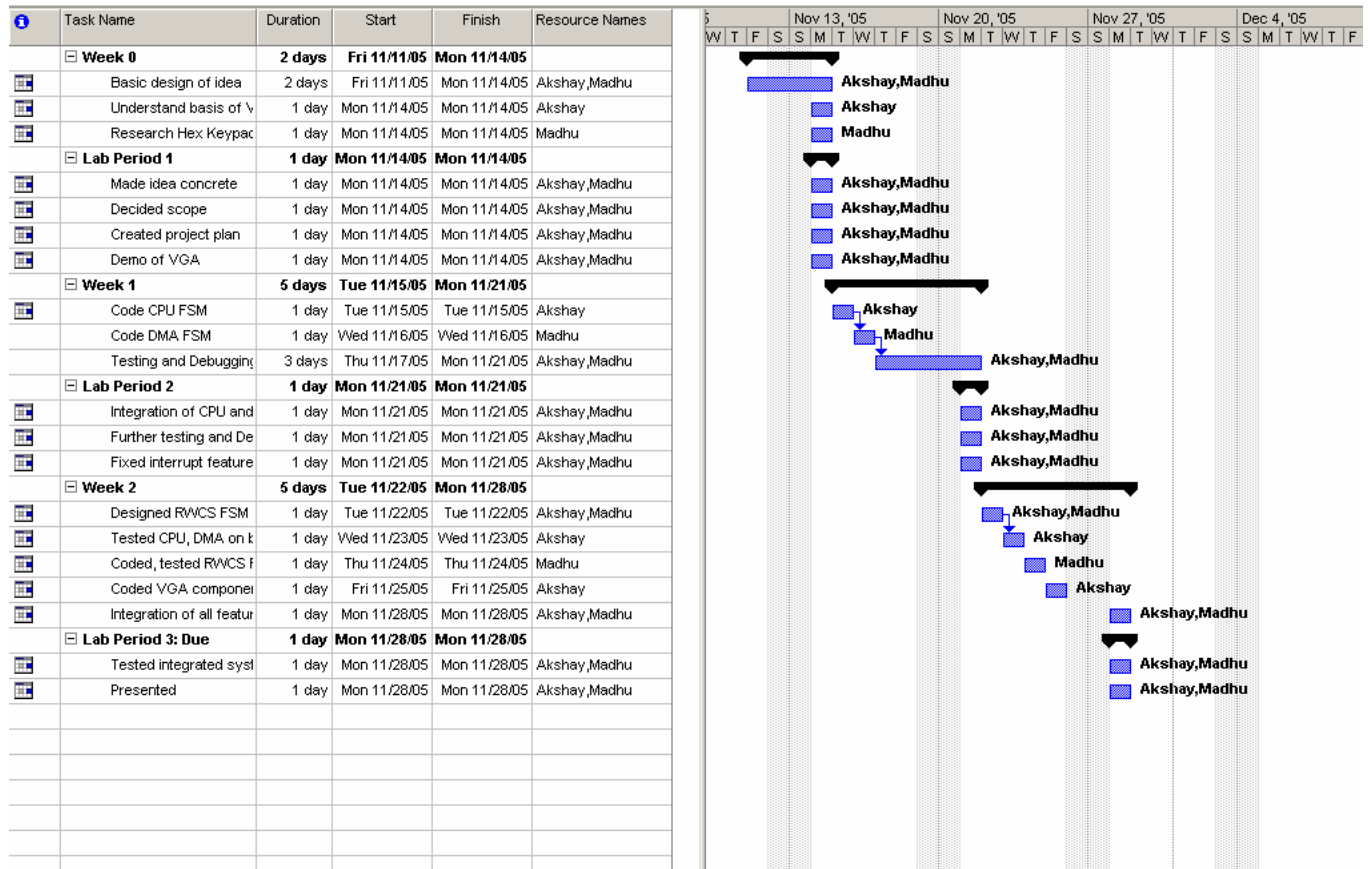


Figure F.1: Gantt Chart of Project 2

APPENDIX G

SOURCE CODE
