A New Lab for 6.111: A Coin operated Vending Machine Controller

by

Calvin J. Lin

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 19, 1999
[June 1999]

Author ______________________
Department of Electrical Engineering an Computer Science
May 21, 1999

Certified by [illegible]
Donald E. Troxel
Thesis Supervisor

Accepted by [illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

An New Lab for 6.111:
A Coin Operated Vending Machine Controller

by

Calvin J. Lin

Submitted to the Department of Electrical Engineering and Computer Science

May 21, 1999

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This new lab was designed for the purpose of updating the curriculum of MIT's digital design course, 6.111 Introductory Digital Systems Lab. The subject of the laboratory assignment is the construction of a coin operated vending machine controller. It requires students to design, build and debug this controller using current technology available in the 6.111 lab. Because this is the first design exercise for most students, the assignment is structured in a manner appropriate for teaching the design process Students gain practical experience in designing a real digital system using industry based tools and standards. It is expected that completion of this lab will prepare the students for subsequent labs and a final project.

Thesis Advisor: Donald E. Troxel
Title: Professor

Acknowledgments

This thesis would not have been completed if I did not receive the help and support from several people. First are the students who patiently undertook the first iteration of this assignment. I know I gave them a hard lab to complete and hope they were able to gain the knowledge I wanted to impart.

I would also like to thank the staff of 6.111 whom I had the good fortune to come to know during my brief time with the class. Not only did they give me great suggestions on how to improve this lab, but kept me company when I was helping students into the early hours of the morning.

I would like to thank my family for giving me the opportunity and instilling in me the drive to complete my education at MIT. Don't worry Mom and Dad, your money hasn't gone to waste.

Finally, I would like to thank my friends for making the time I've spent here very enjoyable. They not only supported me in my classwork, but they also made sure I had fun while I was here.

Table of Contents

List of Figures

List of Tables

Chapter 1

Overview

The new lab project, which is intended to replace the second lab in 6.111, is the design of a coin operated vending machine controller. The vending machine controller is an interesting and familiar subject for students which also provides practical experience in the design of a digital system. Central to this design process is the use of a high level description language, VHDL, to implement and debug certain hardware components of the lab. By teaching VHDL, this lab gives students a very valuable and powerful tool which is used in industry today.

This assignment requires students to implement a Finite State Machine (FSM) to determine the behavior of the vending machine controller.The system takes as inputs various control signals and switches as well as coin inputs. Based on these signals, the controller will step through the different states of the FSM and provide outputs as described by FSM's state transition table. A vending machine controller was chosen because it is a very common piece of equipment all students are familiar with. By eliminating the confusion of describing a complex user interface, this project allows students to focus purely on the design.

This document describes the design and implementation of the lab. It begins with a discussion concerning why a new lab is needed as well as various issues involved in designing this lab. It will then describe the actual design process through which the lab assignment was produced and various problems and changes that resulted from implementing the lab. The lab handout given to the students, a sample solution, as well as checkoff guidelines for the lab are included in the next sections. The conclusion sums up the work that has been done and evaluates the effectiveness of this project.

Chapter 2

Background

This chapter describes the background behind the design of a new Lab Two for 6.111. Understanding the background is important because it gives a sense of purpose for the project. It is also important to know how this project fits with the rest of the work that has been done. Finally, clearly defined goals for the new lab will help focus the design and implementation of this project.

2.1 Updating 6.111

College courses have an inherent problem of not staying up to speed with current technology. 6.111 is no exception. Digital technology has become so complex that industry designers need to use a high level language to describe these circuits. Until a couple of years ago, 6.111 did not have that capability. A very strong effort has been made to correct this problem by adopting VHDL as part of the curriculum. This thesis is another step toward making 6.111 a more useful class for students.

2.1.1 Previous Work

Marc Tanner '98, as a graduate student at MIT under the supervision of Prof. Troxel first investigated the use of VHDL in 6.111 [1]. He established the infrastructure for using VHDL and researched what hardware and software should be used in order to make programming Complex Programmable Logic Devices (CPLDs) with VHDL a viable option for the students. To demonstrate these new tools Marc implemented a new design for the third lab of 6.111.

2.1.2 Work Since

Since Marc's update, some changes to the other two labs have been made. In the fall term of 1998, the students were given the option to implement the second lab using VHDL [2]. They had the choice of designing the project with the old tools, using a CPLD chip to implement the FSM only, or implementing the entire lab using one or two CPLDs. Lab One has also been updated to introduce students to VHDL programming [3].

2.2 Problem Statement

As a continuation of the work that has been done, identifying the specific problems that remain is an important step. There are three issues that serve as the impetus for designing a new lab. The first one is to add an increased emphasis on testing and verification. The second is to add more flexibility to the class. Third is to bridge the gap between the updated Lab Three with the rest of the curriculum.

2.2.1 Testing and Verification

Testing has become a vital part of the digital design industry. Because microelectronic devices have become increasingly complex, testing has become an immense problem. In 6.111, verifying the functionality of a completed system is all that has been done in the past. This, however, is not enough. Testing systems beyond base functionality is essential in developing a robust, cost effective product. Designers seek to investigate and solve as many unanticipated problems as possible before the product reaches the market. Because testing is very important in industry, it is worthwhile to investigate adding this aspect to the class.

2.2.2 Flexibility

For many years, the lab exercises for 6.111 have remained relatively unchanged. Although the specifics of the projects had been altered to prevent students from copying previous work, the actual project topics have not changed. The design methodology that is taught in lab still remains valid, but the students would benefit from the design of a new lab. The new up-to-date lab will not only help to increase student interest, but also add flexibility to the 6.111 curriculum by providing the instructors with more choices of lab projects.

2.2.3 Bridge for Lab Three

This lab serves as a bridge between Lab One and Lab Three. After Marc Tanner's successful update of the third lab, students have been learning how to use VHDL effectively. However, lab three has traditionally been a very large and time consuming project.

Having to learn how to use a completely new language to implement their projects is a lot to ask from the students. The work to upgrade the old Lab Two was an effort to make a smoother transition. Because the lab was not designed to teach VHDL, it was only partially successful. A new lab focusing on teaching the basics of VHDL programming, is necessary to make the curriculum more rewarding for the students.

2.3 Design Goals

The lab assignment must satisfy certain design criteria before it can effectively address the problems stated above. Traditionally, the second lab is the first exposure 6.111 students have to the design process. Therefore, the assignment must emphasize that process. This means the project must not be too complex. Otherwise, it will distract from the educational goals of the lab.

2.3.1 Educational Goals

The new lab must augment the old educational goals with the new tools that have been made available to the students. FSM control, memory usage, and timing circuitry are all concepts that have been taught in the past. For this lab, the students are asked to implement some of these components using VHDL.

The assignment must present the students with the necessary steps to properly design and build a digital system. This presentation, however, cannot be too restrictive, limiting the students' creativity in coming up with their own design. Also, testing and verification must be added to these steps because these concepts are vital components of proper design.

2.3.2 Scope of Project

As a continuation of prior work, this lab must utilize the new tools and hardware that has been incorporated into 6.111. VHDL and CPLDs should be used in the design of the project. The project must also be a practical application of digital design students can relate with. To make the lab a rewarding educational experience, the project itself must be challenging and interesting for the students.

The new lab must be completed within the time allotted. With an increase in complexity and the addition of a testing and verification strategy, the scope of the project has the potential to become too large. Therefore, when designing the lab it is important to set reasonable goals the students can accomplish in time.

Chapter 3

Design of the Laboratory Assignment

A Coin Operated Vending Machine Controller offers an ideal choice for the second lab in 6.111. The controller is simple enough to be implemented using a simple FSM, while being complex enough to make an interesting design project. It offers a wide range of possibilities for the actual representation of the vending machine as well as the structure needed to teach concepts of digital design. All of the students should be familiar with examples of this product making it a fun project for them to build and demonstrate.

This chapter describes how a vending machine controller design was adapted to fit within the constraints of a 6.111 lab. The first section discusses how the control of a vending machine is possible using an FSM. The second section addressed organization of the lab assignment. The last section discusses how this assignment satisfies the design criteria.

3.1 Vending Machine Operation

Although a vending machine is a very complex piece of machinery, the electronic controls are rather simple. FSM control of the vending machine is possible because the physical actions of the machine can be broken down to a small number of distinct states. These states are enumerated within the FSM and control is achieved by encoding the appropriate state transition table [4].

For this assignment, the model of the vending machine can be broken down into four modes: program, display, vend, and test. The action the vending machine will take in each mode is determined by a sequence of steps, each represented by a state. There are a few other states which govern the flow of the table as well as some initial start-up states. It is a conglomeration of all these states that the FSM is built.

3.2 Organizing the Assignment

The laboratory assignment which is handed out to students at the beginning of the lab is included in chapter 4. The purpose of the handout is to give students information about the project they are about to undertake. It outlines the specifications of the vending machine, presents a suggested solution, and states the various deadlines for the lab. Also, it clearly outlines the steps the students need to follow in order to design their project. The project itself is divided into three sections: design review, implementation, and report.

3.2.1 Design Review

Because this is the first design problem many of the students will encounter, it is necessary to help them throughout this process. The design review offers the students an opportunity to discuss their design with the instructors. It also gives the instructors an opportunity to check and evaluate the students' progress.

Before the review, students are required to design and draw block and circuit diagrams for most of their components. They are encouraged to take a look at various timing issues for each of the components and understand how that would affect their design. Also, they should have a state transition diagram describing the behavior of the vending machine controller.

During the design review, the students will go over their design with a TA. The TA will correct any errors in the design, offer suggestions to make the design better, and answer any questions the students might have about the design. Only after talking with a TA will they be able to continue with their lab.

3.2.2 Building the Project

After the design review, students have two weeks to build and demonstrate their controller. During this time, they need to wire up the hardware on their lab kits, write the VHDL code that will be programmed into the CPLD, and debug their project. During the demonstration, the TA's will have a set of guidelines which will help facilitate this check-off process. These guidelines are included in chapter 7.

3.2.3 Laboratory Report

The final task the students are required to complete is a written report of the lab. The content of this report should include a section on the specifications of the vending machine controller and a description of the controller they implemented. The students are to include circuit, block, and timing diagrams for the various components of the project. Also, any VHDL code they programmed should be attached to this document.

The required report is intended to be a clear and concise description of their work. The students have the option of submitting this document to MIT's writing department to fulfill their phase two writing requirement. If they choose to do so, there are more guidelines for both content and format the students will need to follow.

3.3 Satisfying Design Criteria

This project satisfies the design requirements as stated in chapter 2.

3.3.1 Educational Goals

The educational goals for this lab are to teach various digital design theories using the up-to-date tools available to the students. This lab requires the students to use an FSM, memory unit, and clock generator in the implementation of their design. The students are also required to use VHDL in designing and realizing their FSM component. The experience the students gain is not just an understanding of the usage of these components and the structure of the VHDL programming language; they also gain practical experience using these design tools.

This lab also presents the design process clearly and succinctly. The handout gives students specific steps in the design and implementation of their project. For the third lab, they design another, more complex project. By the end of the two labs, the students should be confident enough with their design skills to be able to conceive, design and implement their final project.

3.3.2 Scope of Project

Because this lab is centered around a VHDL implementation of the FSM controller, students will gain experience using these tools. Also, the vending machine controller is a very practical electronic component. Digital designers in industry could conceivably work on a similar problem.

This familiar subject of the project is both challenging and interesting. It is also small enough for the students to build in lab. Because the controller is relatively simple, the time involved in building it should fall within the constraints of the class.

Chapter 4

A Discussion on Implementation

This chapter discusses how well this lab assignment was implemented. There were some initial problems that caused difficulties for the students. Based on feedback received from the students, changes were made to improve the lab. There is, however, more work that can be done to make the lab even better.

4.1 Initial problems

Despite the fact that the lab assignment was designed to make the learning experience challenging but fun, there were some problems with the lab that made it a frustrating experience for the students. These problems detracted from the educational value of the exercise.

4.1.1 Fitting

The problem that plagued the students the most was fitting their VHDL code in the CPLD provided. Although there is enough of space on the chip, the students had difficulty making their project fit. The main cause of this problem was that students did not have enough experience with VHDL. Their code was inefficient because they did not understand how it synthesizes into hardware. Also, they did not know how to solve the fitting problems themselves.

4.1.2 Testing

Students traditionally approached testing as an afterthought. They like to design things and tend to ignore testing beyond basic functionality. Unfortunately for this lab, testing became more of a chore rather than an integral component of the design. Because students left testing toward the end, some never completed it. Also, the testing routines took up a large amount of space in the CPLD, adding to the fitting problem. Many students resented having to implement testing without understanding the necessity of proper testing in industry.

4.2 Student Feedback

Students were given the opportunity to comment on the content and implementation of this lab. Approximately 15 out of 88 students responded to an informal e-mail survey that was sent the class mailing list. Other students talked with TAs and LAs in lab, some included a feedback section in their report. The students predominantly felt that this lab was very frustrating and difficult. They cited several reason that contributed to this opinion.

4.2.1 Software vs. Hardware

The heart of this lab is the code that is programmed into the CPLD. Students only needed to wire three chips. Feedback from the students indicated that they felt most of their time was spent coding rather than designing hardware. They felt that they were so occupied with debugging their code that they did not learn digital design skills.

4.2.2 Learning VHDL

Although most of the students' time was spent coding, the feedback indicated that the students did not feel they understood VHDL. They felt that the fitting issue was such a distraction that they did not have time to learn proper VHDL programming.

4.2.3 Difficulty Compared to Lab Three

One of the initial goals of this lab was to facilitate the learning of VHDL in order to make the transition to lab three smoother. Conversations with the students after they finished the third lab indicated that the experience with VHDL did make the that lab much easier to understand. In fact, they felt the third lab was easier than the second. However, the transition between Lab One and this lab became disjoint.

4.3 Modifications to the Lab

Using observations made in lab and feedback from both students and instructors, several changes have been made to the lab assignment. These changes will hopefully address some of the problems the students had.

4.3.1 More Structured Assignment

Teaching proper coding of VHDL will address better understanding of digital systems as well as the fitting problem. The crux of the problem is that this lab is the first time students have to apply the knowledge imparted to them in lecture. The original assignment only outlined the system level diagram without detailing how the VHDL code should be structured. This left the students with too much to figure out independently.

This is the first time students are asked to design digital hardware; it is also the first time they are asked to code in VHDL. To facilitate learning these two concepts, a more structured assignment is necessary. The key to understanding digital design and how VHDL fits into the picture is to realize that VHDL is a hardware description language. Digital engineers use VHDL to describe the hardware they design.

Conceptualizing the CPLD as discrete elements will help students learn proper digital design and VHDL programming. Having the students think about implementing the CPLD as separate hardware components will reinforce basic concepts of digital design. Asking them to code each component separately will help their understanding of VHDL code organization.

4.3.2 Testing Eliminated

Although testing is a very important part of any design process, it is too important of a topic to be added as an afterthought. Testing and verification is such a large task that many companies require a separate group of engineers to complete it. There is enough material concerning testing to create an entire course at MIT. If in fact testing cannot be presented properly, it is better to eliminate it rather than distract the students from learning more central concepts.

4.4 Future work

Although the changes that have been made should make this lab more rewarding for the students, many things that can be done to improve it.

4.4.1 Testing and Verification

As stated above, testing should not be attempted unless it is taught properly. Evaluating the practicality of testing in a lab is necessary before it can be implemented. If implemented, it requires more specific instructions in the laboratory handout. The instructors and lab assistants must fully support and encourage students in this endeavor.

4.4.2 More Modifications

If this lab is used in the future, small changes are necessary to prevent students from copying previous projects. Some easy modifications can change the user interface or the model of the vending machine. Changing the number of coin inputs or products is relatively simple.

A more involved modification is to utilize the timer circuit students build in lab one. The timer could control a timeout signal. If the user does not select a product before the time expires, the collected money would be returned. This signal can even be used to make the outputs "cleaner." The machine will display the price, change and product vended for a certain amount of time, after which the displays would be cleared.

4.4.3 An Interactive Demo

The current demo project consists only of a lab kit. To present this to the students, a camera would be used to project the kit onto a screen. One of the original goals of this project was to develop a more interactive and realistic demo. Unfortunately due to time constraints, this goal was not accomplished. Therefore, building some kind of mini-vending machine, or programming a GUI for a workstation would make the project more appealing to the students. This demo could be made available to the students to enhance their demonstrations.

4.4.4 Designing a New Lab

This lab is closely based on the old Massachusetts Stoplight Controller Lab [2]. Because of this, the structure and implementation of the labs are very similar. It would be worthwhile to investigate implementing a completely new type of lab. The new lab would be able to take advantage of the many new programmable chips and boards that are currently being integrated into the class.

Chapter 5

The Laboratory Handout

The following section contains the laboratory assignment that was handed to the students. It was adapted from the Massachusetts Stoplight Controller lab handout which was used in previous semesters of 6.111 [2].

Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science
6.111 - Introductory Digital Systems Laboratory

Laboratory 2 - Finite State Machines

Handout Date: February 17, 1999
Design Due: February 24, 1999
Lab Checkoff Date: March 8, 1999
Report Due: March 10, 1999

Introduction

This laboratory exercise concerns the design and implementation of a coin operated vending machine controller. The heart of your implementation will be a synchronous finite state machine (FSM).

This lab is designed to give you a methodology for designing and building a digital system and creating procedures to test it for completeness. This lab is also designed to give you practical experience using VHDL to implement your design.

This laboratory is an exercise that reflects current issues digital designers face in industry. Design and implementation have always been part of this industry, but as microchips have become more complicated and the overhead for fixing mistakes become more expensive, verification and testing has become an integral part of digital design. This lab will introduce some testing techniques and procedures designers use today.

Your vending machine controller FSM is also given the task of loading static RAM locations with the cost of the vended products. The FSM will also display these values by reading the RAM locations.

Vending Machine Controller

The vending machine to be controlled accepts only coins (nickels, dimes, and quarters) and vends four products. There are two input switches which serve as the function and product selectors. There are also two reset switches.

Along with the three coin denominations, there will be another input, the execute or "GO" button. Essentially, this button will be used as the action button. In any operating mode, this is the button that will initiate any action in the FSM controller.

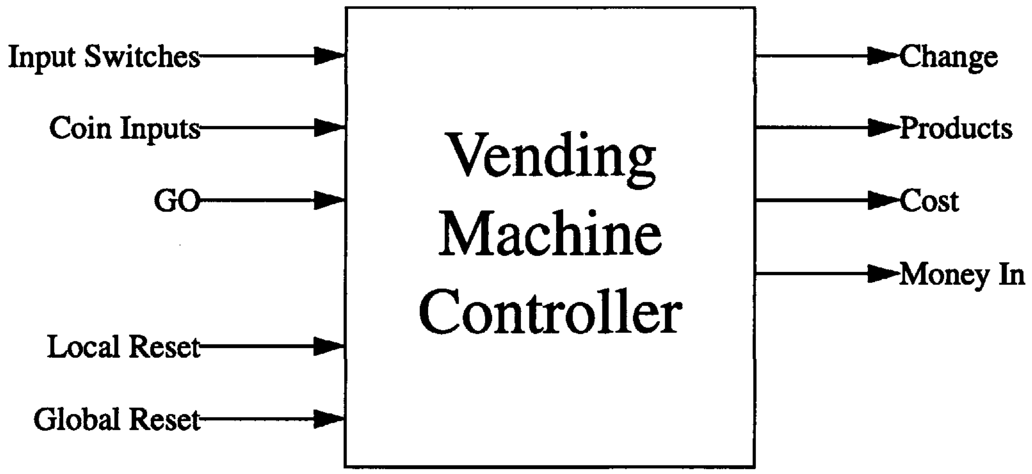

Figure 5.1: Vending Machine Top Level Controller Block Diagram

As outputs, the vending machine controller should indicate to the user whether or not the product selected has been vended as well as the change returned. The machine should also display how much money was put in as well as how much a product costs.

Upon start-up, the user will use the global reset switch to initialize the FSM. This switch will set off a simple routine which will test and clear the memory. If this routine fails, it will halt the FSM. Leds will be used to indicate whether or not the start-up routine has completed successfully or not. The user should be able to use this switch at any time during the operation of the machine.

After running through the initial start-up routine, the operator can switch the controller between four modes: normal operation, programming, display, and free. Using the selection switches, the operator of the vending machine can switch between these modes by selecting the appropriate function and pressing the "GO" button.

In normal mode, the vending machine operates just like any other vending machine. The controller will keep a running total when coins are deposited. To vend a product, the user will select the product and push the "GO" button. If there is enough money in the machine, it will vend the specified product and return any change. The controller will ignore any vend requests if not enough money had been deposited.

In programming mode, the prices for the products are stored in memory. Using the same buttons used to simulate coin input, the operator of the vending machine has the ability to program different prices for the various products.

In display mode, the operator can display prices that were programmed into the vending machine controller by selecting the product and pressing the "GO" button. This will allow the operator of the vending machine to manually check the prices that were programmed.

In free mode, the vending machine will ignore any of the programmed prices and always vend a product when it is selected.

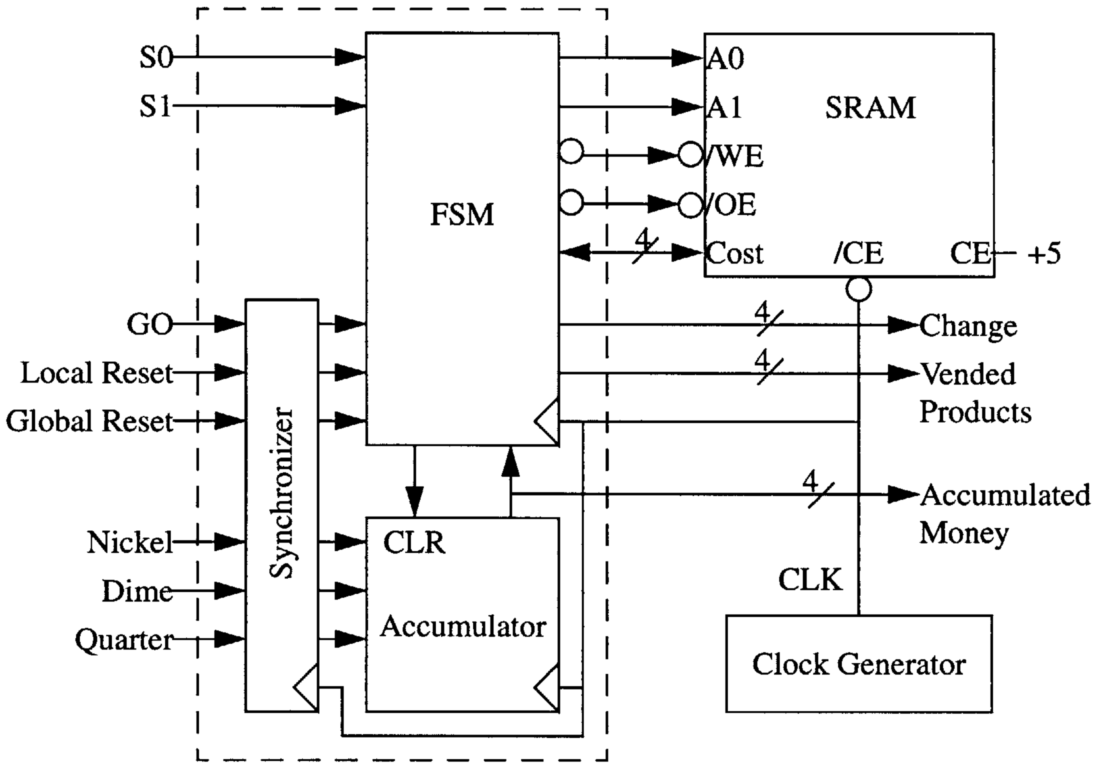

Figure 5.2: System Level Block Diagram

The local reset signal is used to switch out of a mode. The operator of the vending machine can terminate any mode by using this switch. When this switch is activated, the FSM will return to the function selection state. From there, the operator can select the next function to be executed. This switch should be synchronized to the system clock.

Specifications

A system level block diagram of the vending machine controller is shown in figure 5.2. There are three main components to this system, the SRAM, Clock Generator and the FSM Controller (represented by the dashed box).

The SRAM is used to store the prices for the four vended products. Remember to wire the unused RAM address lines to GND. However, leave the unused I/O pins unconnected. The RAM is implemented using the 6264 SRAM. Timing specification and data sheets are included at the end of this handout.

The Clock Generator is the component that produces the FSM clock, CLK. This is the signal that the entire system will be driven and synchronized to. You are welcome to use the timer you constructed in Lab One.

Global RESET	(from Synchronizer)
Local RESET	(from Synchronizer)
GO	(from Synchronizer)
S1 and S0	Determine Function and Product (from switches)
Accumulated Money	Amount of money in the machine (from Accumulator)

Table 5.1: FSM Input Signals

A1 and A0	Specifies an address in the SRAM
WE	Drives values into SRAM
OE	Drives values from SRAM
Cost	Used to read and store the cost of products to SRAM
Vended Products	Vended product indicators (to Led's)
Change	Change indicator (to HEX led's)

Table 5.2: FSM Output Signals

F0	F1	
0	0	Examine memory locations specified by address switches (display)
0	1	Store new value in memory location of address switches (programming)
1	0	Run vending machine (normal)
1	1	Vend products for free (free)

Table 5.3: Table of FSM Modes

The FSM Controller is the heart of the system. It should be implemented using VHDL and programmed into a CPLD chip. It consists of three separate components, the Synchronizer, Accumulator, and FSM.

The Synchronizer is used to sync the GO, coin, and reset inputs to the system clock. While it is possible to effect this synchronization by being clever and absorbing the synchronizing function within your FSM, it is strongly suggested that you explicitly synchronize the input signals with D

flip-flops. It is up to you to design the exact timing of each of these signals. Remember that you only want to assert any one of these signals only once per clock cycle. Also, you should consider what would happen if multiple buttons were pushed at once.

The Accumulator is used to store the amount of money that has been put into the machine. It is up to you to devise an appropriate data representation for the money. Don't forget to include a clear signal which will set the value in the accumulator to 0. Also remember to output the accumulated money to the FSM and HEX led's.

The FSM itself should be the physical embodiment of the state transition table for the FSM you created. Although the Warp synthesizer takes care of creating this table, it is important to remember FSM timing considerations when writing the VHDL code for it.

The input and output signals for the FSM are listed and described in table 5.1 and 5.2. You may use any polarity you like, e.g., N_WE or WE as you choose. table 5.3 describes the four functions specified by the two function switches.

A partially completed VHDL source file is located in the 6.111 locker, /mit/6.111/vhdl/lab2/lab2.vhd. Copy it to your locker by executing:

```
cd
mkdir lab2; cd lab2
cp /mit/6.111/vhdl/lab2/lab2.vhd lab2.vhd
chmod 600 lab2.vhd
```

The VHDL source file provided is not complete enough to create a CPLD file yet. For example, it does not include the complete FSM specification.

Procedure

For the sake of simplicity, use the switches and lights available to simulate the machine. The four push-buttons should be used for the three coin inputs and the GO button. Use the switches to implement the two reset and input switches. You should use the hex leds to display the amount of money in the machine, the cost of the products and the change returned, and one led per product to indicate if it has been vended.

Use the following steps to facilitate your design, implementation, test, and report of the lab.

1. Before proceeding with the details of the FSM design, you should design the circuitry needed to synchronize the GO, Nickel, Dime and Quarter signals. You will want these synchronized signals to be asserted only once per clock cycle.

2. Draw up a complete logic diagram of the system and a state transition diagram of your FSM.

3. Sketch out timing diagrams which completely demonstrate the operation of each function of your FSM.

4. You should discuss the design with a member of the teaching staff before programming your CPLD. Bring the sketches and drawings of your FSM to this Design Checkoff.

5. Before wiring the entire system together, it is recommended that you wire and verify each individual component separately. Test the functionality of the SRAM first. Use

the switches to manually store values into the SRAM and verify them by loading them to the led's.

6. It is recommended that you also program and verify the FSM controller in parts. For each separate component, use a combination of the Nova simulator, logic analyzer and manual inputs to verify that they work. Build each component on top of each other, starting with the synchronizer, then accumulator, and finally the FSM itself.

7. Once the FSM controller has been verified, put the entire system together. Now that you understand how the signals work for each component, it will be easier for you to debug the interactions between the components within FSM controller as well as the SRAM. Again, use the logic analyzer to debug the various control signal outputs of the FSM.

8. Once you have debugged the system for functionality, you should demonstrate the system to one of the instructors. Have all your timing diagrams, state diagrams, VHDL file, and logic diagrams available for this demonstration.

9. If you have time, you should test the system for erroneous inputs: inputs that the project was not designed to handle. What happens when you switch into program mode during the vend cycle? What happens when you push two coin buttons at once? Investigate how robust your design is in regard to these inputs.

Laboratory Report

You should write up a report for this lab which meets the requirements specified in the "Report Guide" handout. Your report should include the following: data paths, an FSM, VHDL source file and the corresponding state diagram, at least one logic diagram, and all timing diagrams. You should also discuss your design, method of implementing it, and how you tested for functionality and bugs. The report should flow, be well organized, and most importantly, be complete. Verbosity is not a requirement.

Design Tips

There are two major problems you will encounter when designing and implementing systems in VHDL: understanding how the code synthesizes into hardware and fitting the entire system onto one chip. An understanding of the first will help to solve the second. These are some tips to help you solve these problems with regard to this lab as well as general tips for designing digital systems.

It is very important to remember the proper approach to programming in VHDL. VHDL is a (H)ardware (D)escription (L)anguage. Instead of laying out circuits and components using a CAD program or wiring various chips together, you will be describing actual physical components using programming constructs.

Therefore, you should approach programming in VHDL like you would if you were designing this lab using the IC chips you have on hand. The FSM controller is separated into three separate parts. Think about and implement each component separately. Once you do, then put them together either by instantiating them using component declarations or by declaring separate processes for each. If needed, break each of the separate components down even further. Remember that the hierarchical nature of VHDL is a very powerful tool.

You have been introduced in lectures and problem sets to various ways if implementing FSM's in VHDL. Because each implementation synthesizes differently, it is very important to understand how they are synthesized and the hardware that results. When designing your FSM think about how you would generate the equations for the outputs of your state transition diagram.

You will be spending a large part of your time trying to make your code fit in the CPLD. The secret to understanding how to make your VHDL code efficient is to understand how it synthesizes into hardware. As mentioned above, each different implementation of an FSM will synthesize differently. Some will take up more space than others.

Remember, you want to minimize the amount of product terms your code will generate. Think about where these product terms come from. How does changing the number of inputs and outputs affect this? How does the number of case statements affect the number of product terms? It is very bad to have conditionals which use comparators or arithmetic operators. Think about how checking for one of these conditions affects the conditionals that don't use these operators.

It is important to realize how important PROPER verification and testing is in the design process. It can make the difference between success and failure. The underlying trait that is most useful is patience. It is very easy to become so eager to program and wire up the entire system and hope that it works the first time you turn it on. Unfortunately, 99 percent of the time, it doesn't. By then, it become very hard to isolate bugs in such a complex system.

Therefore, you should have the patience to test each part of the system individually. Not only verify that it works, but understand the timing involved in making it work. If you do this, you will be able to eliminate the parts individually from consideration when you encounter bugs while integrating the system. You can concentrate on the interactions of the parts rather than the parts themselves.

The last and simplest thing you can do to make your life easier is to have the patience to wire NEATLY. Instead of connecting wires using big arches, try using right angles to wire around the chips and protoboard. Making your chips accessible is very important. You should minimize the possibility of unplugging a wire when you need to replace chips. Also, use different colored wires to differentiate between different lines. It is much easier to follow a single wire if its different than the wires next to it. Remember, the neater the wiring, the easier it is for you, the TA's and LA's to debug your kit. The easier it is for you to debug, the more time you will save.

6264 SRAM Chip

Because you will be using a separate SRAM chip, data sheets for the 6264 SRAM are attached. PLEASE read the data sheet carefully as this chip is easily damaged by incorrect use (wiring). ASK QUESTIONS IF YOU ARE NOT SURE!

The 6264 has a tristate Input/Output (I/O) bus. Reread the handout "Gates, Symbols, and Busses" which pertains to bussing. The I/O bus of the 6264 MUST be driven by a tristate buffer; use the 74LS244 included in your kit, or you can use outputs of the CPLD.

Tristate bus contention occurs when two (or more) drivers are active at the same time. The 6264 tristate output is enabled when the /OE input is asserted low, the /CS is asserted low, and the /WE line is high. While it is true that many logic designers allow tristate bus contention to occur for short times (due to chip delays), it is not a good idea. For this laboratory exercise you are to ensure that NO tristate bus contention can occur.

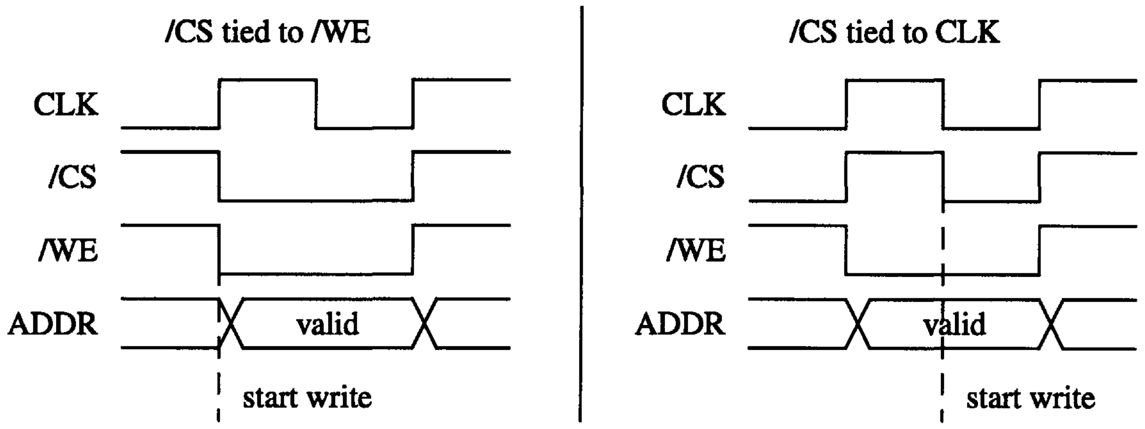

Figure 5.3: SRAM Address Timing

The actual write pulse is the AND of both the /CS and the /WE asserted low. It is essential that the address lines to the SRAM not change when the write pulse is active. Otherwise you may write to multiple locations!

While the 6264 is advertised as a static RAM, a memory cycle is actually initiated whenever ANY address line changes. Thus, the address lines may NOT be tristated whenever the /CS is asserted, as the internal timing circuitry is actuated by noise on the HI-Z address lines.

One way to solve both the tristate bus contention and address holding problems is to connect /CS to the CLK signal. This fix prevents bus contention because the SRAM does not drive the output when /WE is asserted low before /CS is asserted. Figure 3 outlines the reason this fix solves the address holding problem. The left diagram has /CS ties to /WE. We see that the address isn't valid when the write occurs. The right diagram shows /CS tied to CLK. It is obvious to see that the address have enough time to settle before the write happens.

More specific timing diagrams for both the read and write cycle for the SRAM are included with the data sheets.

Chapter 6

A Sample Solution

This chapter describes a sample solution to the Vending Machine Controller assignment. Built to satisfy the specifications set in the Laboratory Handout, this solution can be used as a demonstration for the students. It is only one of many possible solutions.

Following the design recommendations in the Lab Handout, the Coin Operated Vending Machine Controller can be split up into three hardware components: the CPLD, memory storage unit, and clock generator. There are three components programmed into the CPLD: the synchronizer, accumulator, and FSM. Figure 4.2 on page 28 shows the system level diagram, and the VHDL code for this project is included in Appendix B.

6.1 Clock Generator

The clock signal is generated using a 1.8432MHz square wave crystal oscillator. Although the frequency of the oscillator is slow enough to handle the propagation delays of the system, the clock signal is used to drive a divider circuit because the oscillator has a low fanout. All that is required is to feed the clock signal through a buffer. However, the

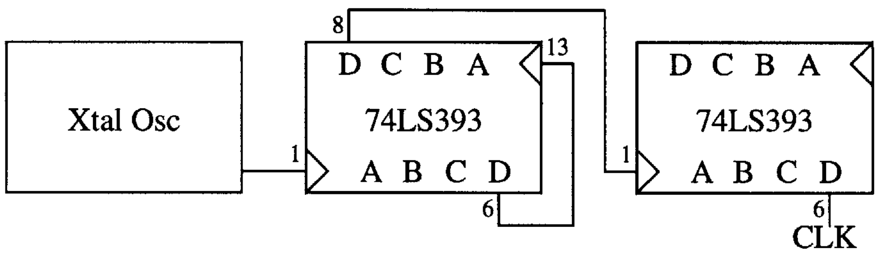

Figure 6.1: Clock Generator Schematic

students should have already built a clock timer circuit in the first lab and have the option of using it. This example does use a timer circuit. The divider is built out of three four bit counters (74LS393) [5]. By taking the 11th bit of the counter chain, a 450Hz clock signal is generated.

6.2 Memory Storage Unit

This project uses a 6116 SRAM to implement the memory storage unit. Although the handout given to the students recommends using a 6264 SRAM, the 6116 SRAM behaves almost identically to the 6264 SRAM. The only perceivable difference is that the 6116 SRAM doesn't have the extra chip select pin. Because this pin is tied to Vdd when the 6264 SRAM is used, its absence in the 6116 SRAM is inconsequential. The pin assignments for the SRAM are shown in Figure 6.2.

6.3 CPLD

A 372i CPLD is used to store the VHDL implementation for the three programmed components [6]. The CPLD also has the capability of storing up to 320 product terms, divided up into 64 macrocells. 32 macrocells are used for I/O, the remaining 32 are buried within the chip. The CPLD has a total of 37 possible external pin connections. These

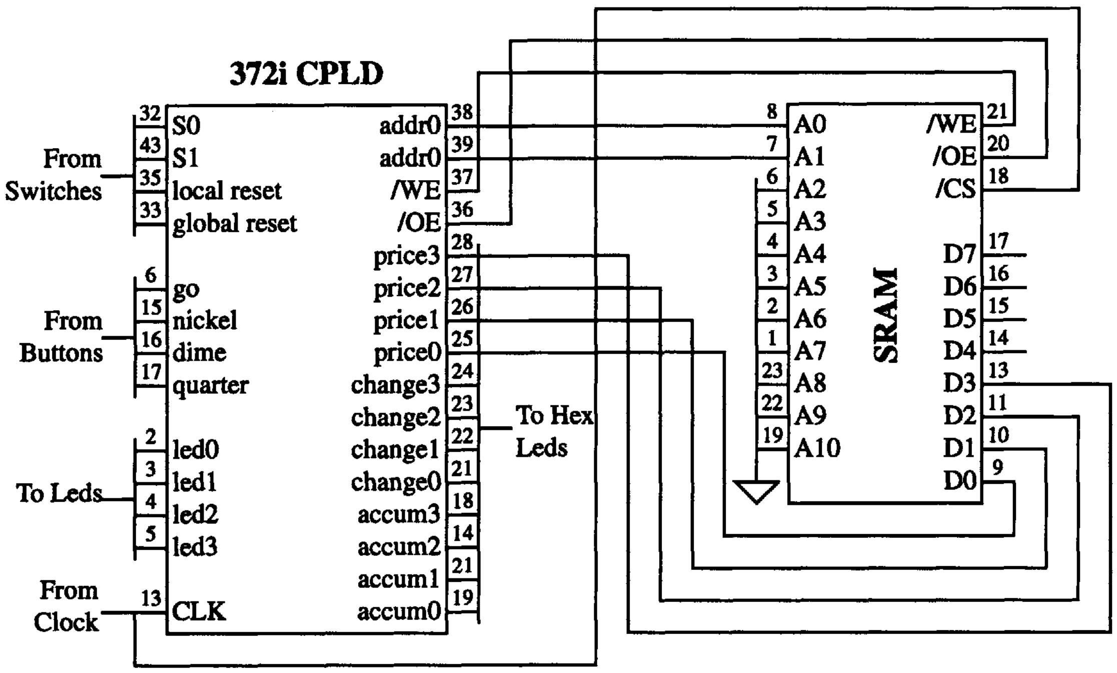

Figure 6.2: CPDL and SRAM Schematic

specifications are sufficient to implement the three components programmed in the CPLD. This example only uses 34 pins, 50 macrocells, and 259 product terms.

The CPLD takes as inputs the clock, coin inputs, go, local and global reset, and address switches. It outputs to the SRAM the address, write enable, and output enable signals. Price, change, and accum go to the hex leds and the product selection indicators go to the normal leds. Figure 6.2 shows the pinout assignments of the CPLD.

As seen in figure 5.2, the functionality of the CPLD can be split into three distinct components: the synchronizer, accumulator, and the FSM. This project separates these components into three different files, with synchronizer and accumulator instantiated within the top-level FSM file. The reasons these components are separated are to take advantage of VHDL's hierarchical organizational capability, to allow for flexibility in the system's design, and to facilitate testing and debugging [7].

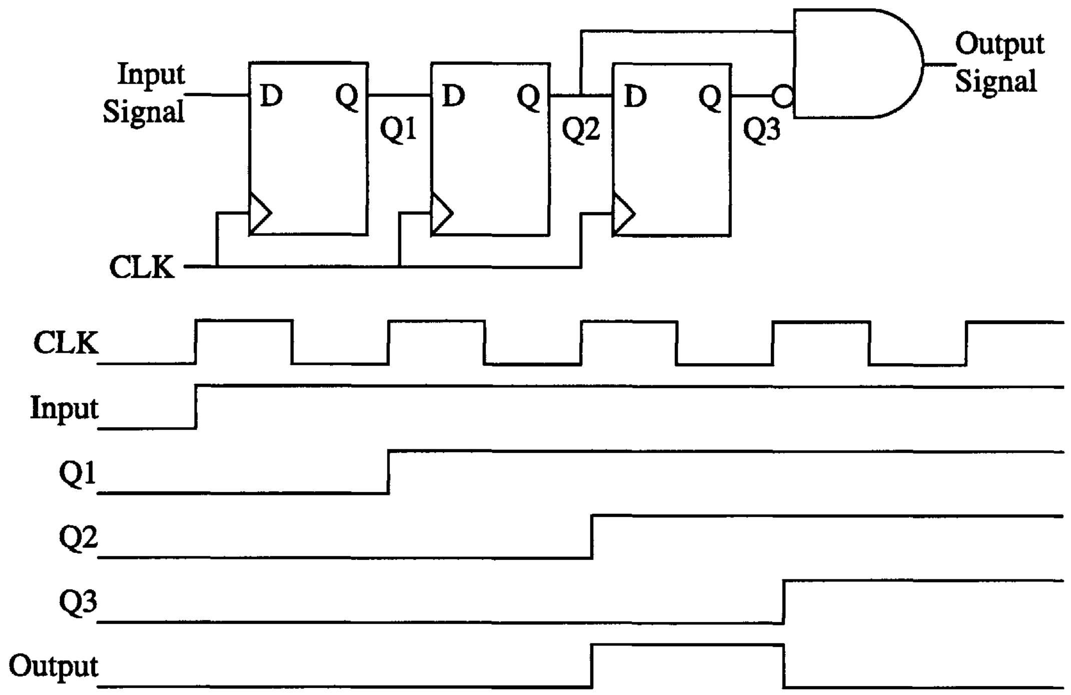

Figure 6.3: Synchronizer Block Diagram and Timing

6.3.1 Synchronizer

A single synchronizer consists of three edge triggered flip flops, an AND gate, and an inverter, shown in figure 6.3. The first flip-flop is used to eliminate any metastable states that may be generated. The second and third flip-flops are used to generate the single clock period pulse that is needed for these signals. The timing diagram in figure 6.3 shows how the output is generated. Because there are six signals that need to be synchronized, this circuit is instantiated six times.

6.3.2 Accumulator

The accumulator is used to store the amount of money that has been inputted into the machine. It also raises a flag if there is enough money in the machine when a product is selected. It takes as inputs the three coin denominations, the price of a product, and the

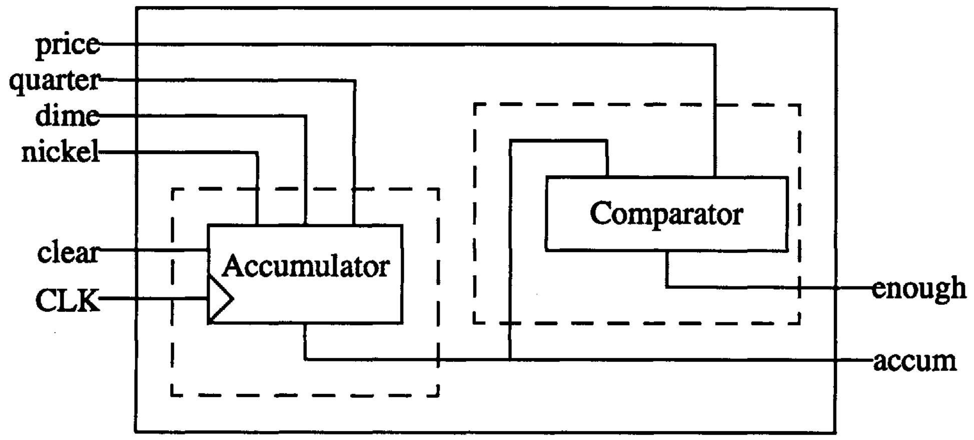

Figure 6.4: Accumulator Block Diagram

accumulator clear signal. It outputs the accumulated money and a status signal which indicates that there is enough money in the machine.

Money is stored as the denomination of the coin divided by 5 (nickel = 1, dime = 2, quarter = 5). When a coin is deposited, the accumulator will add that amount of money to the running total. Also, it always checks the running total against the price of the product. If it is greater than the price, then the "enough" flag is raised. Finally, when the accumulator clear signal is set to 1, the accumulator will clear the accumulated money.

The VHDL code for the accumulator is separated into two processes indicated by the dashed boxed in figure 6.4. The first is a clocked process which contains the code that accumulates the money. The second process contains the comparator which determines whether or not to raise the enough flag. It is implemented as combinational logic.

6.3.3 FSM

The FSM is implemented according to the specifications described in the lab handout. This design uses a 11 state mealy machine to implement its functionality. The FSM

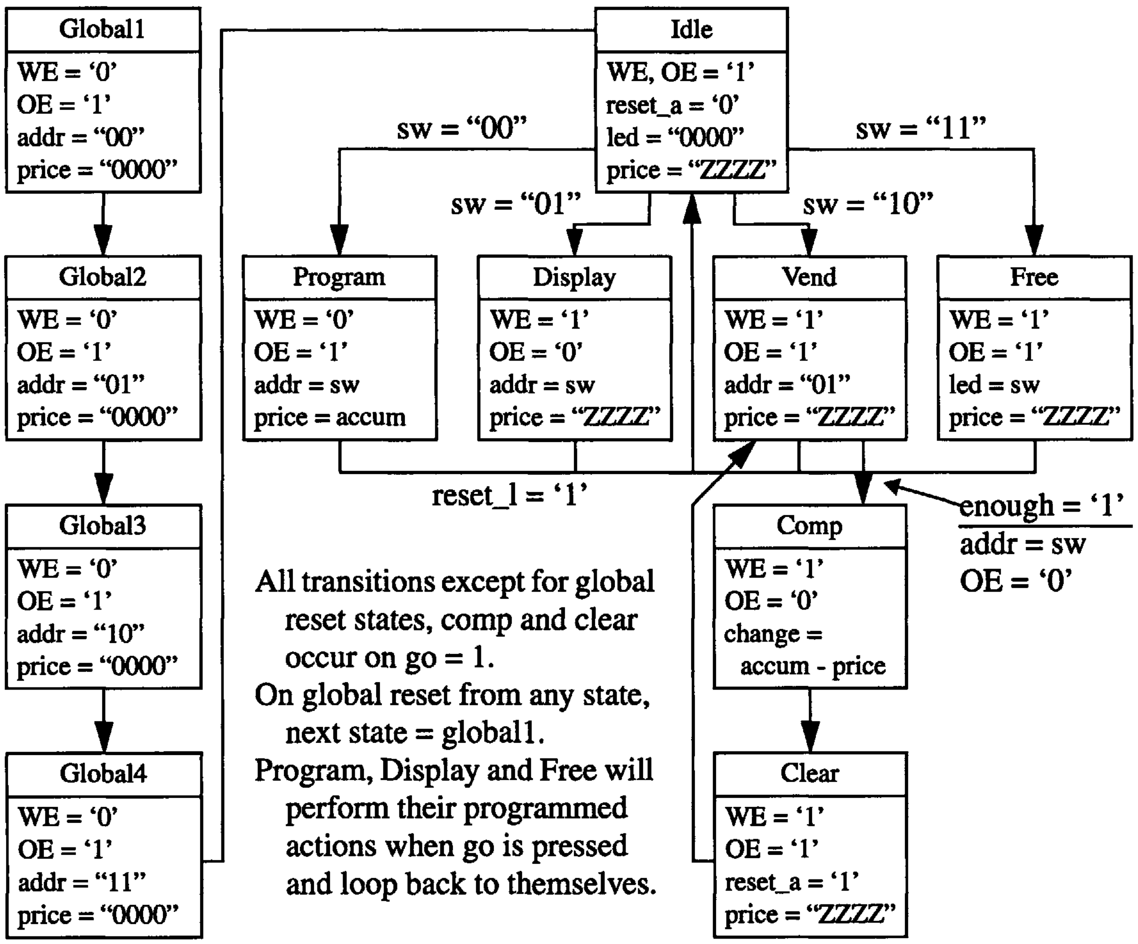

Figure 6.5: FSM State Transition Diagram

will determine the outputs and next state based on the current state and inputs. Figure 6.5 shows the state transition diagram of the FSM.

The VHDL code for the FSM is split into two different processes. One process takes the current state and the inputs and determines the next state and output. Essentially, this is the process which encodes the state transition table for the FSM. The other process, clocks the next state into the current state. This clocked process regulates the states.

Upon start-up and global reset, the controller will start the global reset routine. This sequence of four states will clear the contents of the SRAM. The controller will then transition to the idle state. When the go button is pushed in the idle state, the controller will select one of the four states to go to based on switch inputs.

If go is pressed in the program mode, the controller will set the address to the switch inputs, the SRAM input to the accumulated price, and the SRAM to write. The price that was set will be programmed to the appropriate address in memory.

If go is pressed in the display mode, the controller will set the address to the switch inputs and the SRAM to output the data. Because the output is already connected to the hex leds, the cost of the product will be displayed.

In vend mode, the controller will transition to the comp state only if the cost of the product is less than or equal to the accumulated money. Otherwise it will remain in the vend state. If there is enough money, the controller will output the calculated change and appropriate led indicator. It will then transition to the clear state where the accumulator is cleared.

In free mode, the controller will ignore all accumulated money and cost of products. When go is pressed, it will light up the appropriate product leds based on the input switches.

In all four modes, if local reset is selected, the modes will transition to the idle state. Also, if the global reset is selected in any of the states, the machine will immediately transition to the first global reset state.

Chapter 7

Checkoff Guidelines

This chapter contains the guidelines TAs can use to help checkoff students' lab projects. It is important to remember that these are only guidelines. It is up to the TA's judgment whether or not the students have satisfied the lab requirements. If they can justify their design choices, then their project is satisfactory.

Lab 2: Vending Machine Controller Checkoff Guidelines

Basic Functionality

The hardware components are the first thing to check. Check the wiring for the clock generator, SRAM, and CPLD. Pay attention especially to how the SRAM is wired up. Ask the students how any why they wired up the SRAM.

When checking off the CPLD, ask the students to produce their VHDL code. Make sure he/she understands how the code relates to the operation of the machine. Ask them how their code would be synthesized into hardware components. If convenient, have the students show a simulation of the components. If they did not use the simulator, encourage them to do so for the next lab.

Next, ask them how they implemented the synchronizer and accumulator. Make sure they understand why the synchronizer needs three flip flops instead of two. Also ask about how the delayed synchronizer output affects the operation of the FSM. For the accumulator check to see that the code does indeed works. Point out places where code might be inefficient.

To verify the base functionality of the FSM, have the students demonstrate that each of the four functions of the vending machine controller works. Have a copy of their state transition diagram handy. Ask the students to step through the diagram while demonstrating their project.

The programming and display modes can be verified in conjunction. Have the students program distinct prices in each of the four products. Verify that the correct price was programmed using the display mode.

To verify the vend mode, use the prices programmed when verifying the programming mode. Try different combinations of coin inputs to verify that the accumulator is working. Vending products with accumulated money that is greater than, equal to, and less than the price of a product. Make sure that the correct price, change, and product light are displayed when the product is vended. The product led should not light up if not enough money was inputted.

The local reset signal should be functional, otherwise switching between modes would not be possible. One thing to make sure, on local reset, the accumulator should be cleared. For the global reset signal, verify that all the memory locations are cleared using the display mode.

Questions to Ask Students

These are interesting issues that the students are not required to have an answer for. If they have not addressed them, then bringing it up is sufficient. If they have addressed these issues in their design, they should be rewarded appropriately.

1. What happens when the accumulator rolls over? What is a good solution to this problem?
2. What will happen if multiple inputs are pressed at once? Will all the inputs be processed? If not, which input has precedence?
3. What state does the machine start when it is first powered up? What would happen if somehow the machine got stuck in an unanticipated state?
4. How is their VHDL code organized and why did they choose to organize it in that manner? Why does separating the components make the design process easier?
5. What part of the VHDL code takes up the most space in the CPLD? Why?

Chapter 8

Conclusion

This new lab for 6.111, The Coin Operated Vending Machine Controller, succeeds as a continuation of the work that has been done to update the 6.111 curriculum. It effectively utilizes the new tools that were recently developed while leaving room for future technological improvements.

It offers a solution to the problems stated in Chapter 2 while staying within the confines of the design goals. Designed with VHDL in mind, this lab takes advantage of the power inherent in designing digital systems using a high level description language. It accomplishes this without adding more complexity to the problem. Students are given a clear and concise learning experience.

Some problems did occur with the initial implementation of the lab. However, these problems were minor, only distracting minimally from the educational purpose of the lab. With the understanding and patience of the students and the support of the teaching staff, an improved lab is the result of this endeavor.

Although there is still more work that can be done to make this lab even better, this lab can stand in its current form. It does serve a very useful and educational purpose. The success of the students even through the initial problems is a testament to that fact. The reactions from students were positive and the staff does intend to use the new lab for future classes. Except for the failure to integrate testing into the curriculum, the implementation of this new lab can be deemed as a success.

Appendix A

Listing of Code Provided to Students

This appendix contains the files which were provided to the students. The code only gives a framework in which the students were to work.

A.1 Example FSM: lab2.vhd

```
-- This lines are the library and use clauses
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.std_arith.all;

-- This is the FSM's entity declaration.
-- Note that it is not complete.
ENTITY fsm IS
  PORT (clk, go: IN std_logic;
        led: OUT std_logic_vector(3 downto 0));
END fsm;

-- This is the FSM's archticture declaration.
ARCHITECTURE archfsm OF fsm IS

-- External component declarations go here.
-- The actual entity and architecture declarations can be --
-- written in a separate file.
  COMPONENT sync
    PORT (clk, x: IN std_logic;
          y: OUT std_logic);
    END COMPONENT;

-- These lines set up the state encoding as well as any
-- signals.
  TYPE states IS (state1, state2);
    SIGNAL p_state, n_state: states;

BEGIN

-- This is how you instantiate a component.
  go_s: sync port map(clk => clk, x => go, y => sync_go);

-- This process encodes the FSM's state transition table
  PROCESS (sync_go)
  BEGIN
    CASE p_state IS
      WHEN state1 => led <= "1100";
                     IF sync_go = '1' THEN
                       n_state <= state2;
                     ELSE
                       n_state <= state1;
                     END IF;
      WHEN state2 => led <= "0011";
                     IF sync_go = '1' THEN
                       n_state <= state1;
                     ELSE
                       n_state <= state2;
                     END IF;
    END CASE;
  END PROCESS;

-- This process creates the register the states are clocked
through.
  PROCESS (clk)
  BEGIN
    IF rising_edge(clk) THEN
      p_state <= n_state;
    END IF;
  END PROCESS;
END archfsm;
```

Appendix B

Listing of Code Used in the Sample Solution

This appendix contains listings of code used in the sample solution to the laboratory assignment. This code is not handed to the students. The framework for the sample code handed to the students is generated from the overall structure of this code.

B.1 Synchronizer Code: sync.vhd

This file contains the code for the synchronizer. Signal x and clk are the inputs, y is the synchronized output. The three flip-flops are instantiated by the clocked process. Because the AND gate is combinational, it is instantiated outside of a process.

```
library ieee;
USE ieee.std_logic_1164.ALL;

ENTITY sync is
  PORT (clk, x: IN std_logic;
        y: OUT std_logic);
```

```
END sync;

ARCHITECTURE pulse OF sync IS
  SIGNAL a,b,c: std_logic;
BEGIN

y <= b AND (NOT c);

PROCESS (clk, x, a, b, c)
  BEGIN
    IF rising_edge(clk) THEN
      c <= b;
      b <= a;
      a <= x;
    END IF;
  END PROCESS;
END pulse;
```

B.2 Accumulator Code: accum.vhd

This file contains the code for the accumulator. It contains the two process described in chapter 6.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.std_arith.all;

ENTITY count IS
  PORT (clk, reset: IN std_logic;
        quarter, nickel, dime: IN std_logic;
        price: IN std_logic_vector(3 DOWNTO 0);
        enough: OUT std_logic;
        accum: BUFFER std_logic_vector(3 DOWNTO 0));
END count;

ARCHITECTURE archcount OF count IS
BEGIN
PROCESS (price, accum)
  BEGIN
    IF accum >= price THEN
      enough <= '1';
    ELSE
      enough <= '0';
    END IF;
  END PROCESS;

 PROCESS (clk, reset, quarter, nickel, dime, accum)
  BEGIN
    IF rising_edge(clk) THEN
      IF quarter = '1' THEN
        accum <= accum + 5;
      ELSIF nickel = '1' THEN
        accum <= accum + 1;
      ELSIF dime = '1' THEN
        accum <= accum + 2;
      ELSIF reset = '1' THEN
        accum <= "0000";
      END IF;
    END IF;
  END PROCESS;
END archcount;
```

B.3 FSM Code: vendcont.vhd

This file contains the code for the FSM. It is split up into two processes. One encodes the state transition table. The other clocks the next state into the current state. The synchronizer and accumulator are both instantiated in this file.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.std_arith.all;

ENTITY fsm IS
  PORT (clk, reset_g, reset_l: IN std_logic;
        go: IN std_logic;
        sw: IN std_logic_vector(1 downto 0);
        quarter, nickel, dime: IN std_logic;
        price: INOUT std_logic_vector(3 downto 0);
        we, oe: OUT std_logic;
        addr: OUT std_logic_vector(1 downto 0);
        led: OUT std_logic_vector(3 downto 0);
        change: BUFFER std_logic_vector(3 downto 0);
        accum: BUFFER std_logic_vector(3 downto 0));
   ATTRIBUTE pin_numbers OF fsm: ENTITY IS
     "price(0):31 price(1):30 price(2):29 price(3):28" &
```

```
      " led(0):2 led(1):3 led(2):4 led(3):5" &
     " change(0):27 change(1):26 change(2):25 change(3):24" &
      " go:6 nickel:15 dime:16 quarter:17" &
      " addr(0):39 addr(1):38 we:37 oe:36 sw(0):32 sw(1):43";
END fsm;

ARCHITECTURE archfsm OF fsm IS
  COMPONENT count
    PORT (clk, reset: IN std_logic;
          quarter, nickel, dime: IN std_logic;
          price: IN std_logic_vector(3 DOWNTO 0);
          enough: OUT std_logic;
          accum: BUFFER std_logic_vector(3 DOWNTO 0));
  END COMPONENT;

  COMPONENT sync
    PORT (clk, x: IN std_logic;
          y: OUT std_logic);
    END COMPONENT;

  TYPE states IS (global1, global2, global3, global4, idle,
                  program, display, vend, comp, clear, free);
  SIGNAL p_state, n_state: states;
  SIGNAL reset_a: std_logic;
  SIGNAL sync_reset_l, sync_reset_g, sync_go: std_logic;
  SIGNAL sync_nickel, sync_dime, sync_quarter: std_logic;
  SIGNAL enough, sub_go: std_logic;
BEGIN

  go_s: sync port map(clk => clk, x => go, y => sync_go);
  reset_l_s: sync port map(clk => clk, x => reset_l,
                           y => sync_reset_l);
  reset_g_s: sync port map(clk => clk, x => reset_g,
                           y => sync_reset_g);
  nickel_s: sync port map(clk => clk, x => nickel,
                          y => sync_nickel);
  dime_s: sync port map(clk => clk, x => dime,
                        y => sync_dime);
  quarter_s: sync port map(clk => clk, x => quarter,
                           y => sync_quarter);

  accumulator: count port map(clk => clk, reset => reset_a,
      quarter => sync_quarter, nickel => sync_nickel,
      dime => sync_dime, price => price,
      enough => enough, accum => accum);
```

```
PROCESS (p_state, sync_reset_g, sync_reset_l, sync_go, sw,
         enough, accum, change, price)
BEGIN
  IF sync_reset_g = '1' THEN
    n_state <= global1;
  ELSE
    CASE p_state IS
      WHEN global1 => price <= "0000";
                      we <= '0';
                      oe <= '1';
                      addr <= "00";
                      n_state <= global2;
      WHEN global2 => price <= "0000";
                      we <= '0';
                      oe <= '1';
                      addr <= "01";
                      n_state <= global3;
      WHEN global3 => price <= "0000";
                      we <= '0';
                      oe <= '1';
                      addr <= "10";
                      n_state <= global4;
      WHEN global4 => price <= "0000";
                      we <= '0';
                      oe <= '1';
                      addr <= "11";
                      n_state <= idle;
      WHEN idle => price <= "ZZZZ";
                   led <= "0000";
                   change <= "0000";
                   we <= '1';
                   oe <= '1';
                   reset_a <= '0';
                   IF sync_go = '1' AND sw = "00" THEN
                     n_state <= program;
                   ELSIF sync_go = '1' AND sw = "01" THEN
                     n_state <= display;
                   ELSIF sync_go = '1' AND sw = "10" THEN
                     n_state <= vend;
                   ELSIF sync_go = '1' AND sw = "11" THEN
                     n_state <= free;
                   ELSE
                     n_state <= idle;
                   END IF;
      WHEN program => oe <= '1';
                      IF sync_reset_l = '1' THEN
```

```
                  we <= '1';
                  price <= "ZZZZ";
                  reset_a <= '1';
                  n_state <= idle;
                ELSIF sync_go = '1' THEN
                  addr <= sw;
                  we <= '0';
                  price <= accum;
                  reset_a <= '1';
                  n_state <= program;
               ELSE
                  price <= "ZZZZ";
                  we <= '1';
                  n_state <= program;
               END IF;
WHEN display => we <= '1';
                price <= "ZZZZ";
                IF sync_reset_l = '1' THEN
                  oe <= '1';
                  n_state <= idle;
                ELSIF sync_go = '1' THEN
                  addr <= sw;
                  oe <= '0';
                  n_state <= display;
                ELSE
                  oe <= '0';
                  n_state <= display;
                END IF;
WHEN vend => we <= '1';
             price <= "ZZZZ";
             IF sync_reset_l = '1' THEN
               oe <= '1';
               reset_a <= '1';
               n_state <= idle;
            ELSIF sync_go = '1' AND enough = '1' THEN
               addr <= sw;
               oe <= '0';
               n_state <= comp;
             ELSE
               oe <= '1';
               n_state <= vend;
             END IF;
WHEN comp => we <= '1';
             oe <= '0';
             price <= "ZZZZ";
             change <= accum - price;
```

```
                    n_state <= clear;
                    IF sw = "00" THEN
                      led <= "0001";
                    ELSIF sw = "01" THEN
                      led <= "0010";
                    ELSIF sw = "10" THEN
                      led <= "0100";
                    ELSE
                      led <= "1000";
                    END IF;
      WHEN clear => we <= '1';
                     oe <= '1';
                     reset_a <= '1';
                     price <= "ZZZZ";
                     n_state <= vend;
      WHEN free => we <= '1';
                   oe <= '1';
                   price <= "ZZZZ";
                   IF sync_reset_l = '1' THEN
                     n_state <= idle;
                   ELSIF sync_go = '1' AND sw = "00" THEN
                     led <= "0001";
                     n_state <= free;
                   ELSIF sync_go = '1' AND sw = "01" THEN
                     led <= "0010";
                     n_state <= free;
                   ELSIF sync_go = '1' AND sw = "10" THEN
                     led <= "0100";
                     n_state <= free;
                   ELSIF sync_go = '1' AND sw = "11" THEN
                     led <= "1000";
                     n_state <= free;
                   ELSE
                     n_state <= free;
                   END IF;
      WHEN others => we <= '1';
                     oe <= '1';
                     price <= "ZZZZ";
                     n_state <= global1;
    END CASE;
  END IF;
END PROCESS;

PROCESS (clk)
BEGIN
  IF rising_edge(clk) THEN
```

```
      p_state <= n_state;
    END IF;
  END PROCESS;
END archfsm;
```

Biblography

[1] Marc D. Tanner. *Helium Breath: An Updated 6.111 Curriculum.* Master's Thesis, Massachusetts institute of Technology, May 1998.

[2] Donald E. Troxel. *6.111 Laboratory 2: Massachusetts Stoplight Controller.* Massachusetts Institute of Technology. Spring 1999 edition.

[3] Donald E. Troxel. *6.111 Laboratory 1. Logic Analizers, Digital Oscilloscopes, PALs, and CPLDs.* Massachusetts Institute of Technology. Spring 1999 edition.

[4] Prof Troxel and James Kirtley. 6.111 Lecture Notes. Massachusetts Institute of Technology.

[5] Texan Instruments. *The TTL Data Book,* 1988 edition, 1994.

[6] Cypress Semiconductor, *Programmable Logic Data Book,* 1996 edition, 1995.

[7] Kevin Skahill. *VHDL for Programmable Logic.* Addison-Wesley, Menlo Park, 1996.