ECE 527 SoC Design Machine Problem 1

Andrew Smith(atsmith3) and Thomas Furlong(tfurlon2)

1. INTRODUCTION

This MP was designed to become familiar with the Xilinx tools. We set up the development environment on a lightweight ubuntu distribution and interfaced with the ZedBoard. We walked through the compilation process as well as practiced incorporating the Zynq hard IP core into the design by generating a Board Support Package and writing software in the Xilinx SDK.

2. Part A

2.1. Description

For the first part of the MP we had to write a small verilog module for the programmable logic fabric portion of the Xilinx Zinq 7000 chip. This module needs to read the positions of the 8 on board switches and display the switch position on the 8 user LEDs. The switch status was displayed on the LEDs after 3 clock cycles and the center button was used as a reset.

2.2. Assumptions

We did not have to make any assumptions for this part of the MP as the directions were very straight forward.

2.3. System Configuration

The part A of the machine problem was very simple so we only needed one module in the programmable logic fabric. This module took in inputs from the switches, a single input from the reset button, and a clock. The module output a vector to the LEDs containing information on the switch state. The module contained three buffering registers to ensure that the switch state appeared on the LEDs after exactly 3 clock cycles. The module is shown in figure 1.

2.4. Entities

Entity	Description
basic_i_o	Hardware top level, contains pipeline

2.5. Design

When designing we only considered one solution. Using a pipeline to transfer switch state information to the LEDs. This would ensure that the LEDs were updated with switch information after exactly 3 cycles for every change in the switches. Had we used a counter or other option it would have more complex logic and been harder to guarantee the LEDs were updated after 3 cycles.

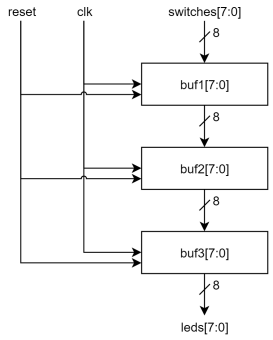

Figure 1. Block Diagram for Part A

2.6. Performance

This was a very small design and it took up very little resources on the Zinq 7000. The usage is shown in the table below. Figure 2 shows how the design was implemented on the device.

Table 1. Resource Usage Part A

Resource	Utilization	Available	Utilization %
LUT	1	53200	0.01
FF	24	106400	0.02
IO	18	200	9.00
BUFG	1	32	3.13

Because this design used minimal logic most of the power consumed by the device was static power. As the transistors were mostly siting idle across the programmable logic fabric.

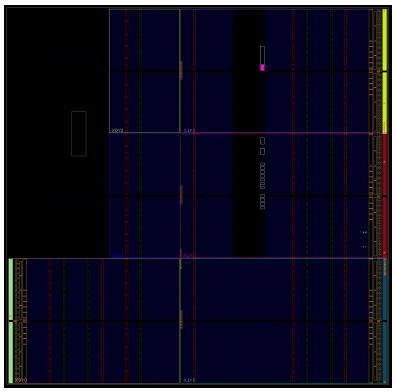

Figure 2. Device Mapping for Part A

Table 2. Resource Usage Part A

Type	Power
Static	0.122 W
Dynamic	0.007 W
Total	**0.129 W**

Due to the design small size the router did not have a problem satisfying the timing constraints of the clock. Shown below is some of the timing information from the timing report.

Table 3. Timing Report Part A

Type	Time
Worst Hold Slack	0.104 ns
Worst Negative Slack	8.442 ns

2.7. Difficulties/Bugs

The only difficulty encountered in this section was just being unfamiliar with the Xilinx design suite.

3. Part B

3.1. Description

For this part of the MP, we added four more push buttons to the design. These push buttons effect the output to the LED's based on simple logic operations. We also had to create a finite state machine that used the push buttons to control what was output to the LEDs.

3.2. Assumptions

In Part B, we assume that when a mode button is pressed, the system stays in that mode until another button is pressed. This is justified because the user should not have to hold down the mode button to see the desired LED output. Also, we assumed when the reset button is pressed, the LEDs will all turn off and when the reset button is released, the system returns to Mode 0. This is justified because the user should just have to press the reset button once to reset the system.

3.3. System Configuration

Part B of this MP was not too complicated. Our implementation required a new module for the finite state machine as well as a new top level module. The new top level module connected the input switches and buttons to the finite state machine and control logic.

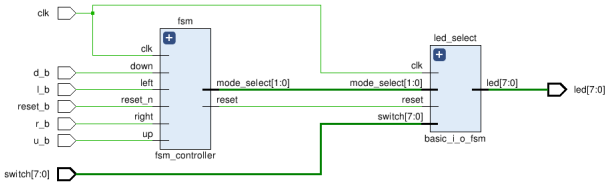

Figure 3. Block Diagram for Part B

3.4. Entities

Entity	Description
top	This is the hardware top level that connects the finite state machine to basic_i_o.
fsm_controller	This is the finite state machine that keeps track of the current state, holds next state logic, and sends control signals to the basic_i_o module.
basic_i_o	This module takes in the control logic from the state machine and selects the correct altered switch data to send to the LEDs.

3.5. Design

When designing this part of the MP, we first considered a design where the mode would only be selected when the push button was held down. We realized this was not the best approach. We then switched to a design were a mode was locked in when we pressed the push button. This lead to a more extensive state machine but the state control logic was still the same.

3.6. Performance

This was a very small design and it took up very little resources on the Zinq 7000. The usage is shown in the table below. Figure 4 shows how the design was implemented on the device.

Table 5. Resource Usage Part B

Type	Power
Static	0.122 W
Dynamic	0.009 W
Total	**0.131 W**

Table 6. Timing Report Part B

Type	Time
Worst Hold Slack	0.253ns
Worst Negative Slack	0.7.47ns
Worst Pulse Width Slack	4.5 ns

Table 4. Resource Usage Part B

Resource	Utilization	Available	Utilization %
LUT	22	53200	0.04
FF	11	106400	0.01
IO	22	200	11.00
BUFG	1	32	3.13

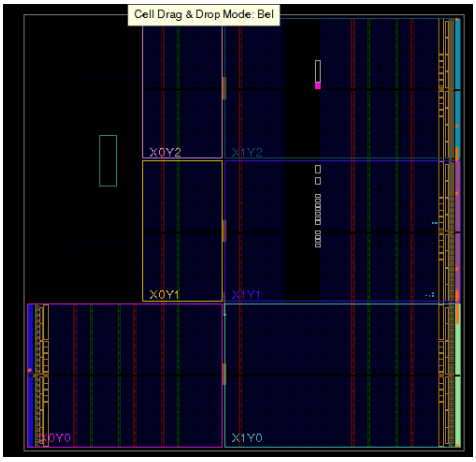

Figure 4. Device Mapping for Part B

Because this design was also small, most of the power consumed by the device was static power.

3.7. Difficulites/Bugs

For this part of the MP, the biggest difficulty was using the Verilog syntax of a finite state machine. After finding some resources online, the rest of this part was straightforward.

4. Part C

4.1. Description

For the third part of the MP we had to interface the embedded ARM core with the switches and LEDs through the AXI bus and imitating the behavior of part B in software. The objective of this part was to familiarize ourselves with the process creating custom IP modules and writing software to interface with them.

4.2. Assumptions

The one assumption we made for this part of the MP is that during the modes a change in the switches will be reflected in the LEDs.

4.3. System Configuration

The Zynq 7000's integrated ARM cores interface with the switches and LEDs through a GPIO AXI port. The AXI interface block has two 8 bit channels and are directly connected to the switches and the LEDs. The block diagram of the configuration is shown in figure 3. When the GPIO module is created the Xilinx toolkit will automatically connect the modules for you and instantiate some supporting hardware. The toolkit automatically instantiated a system reset block so that the cores can be reset, as well as an AXI interconnect to allow for multiple AXI modules to be connected.

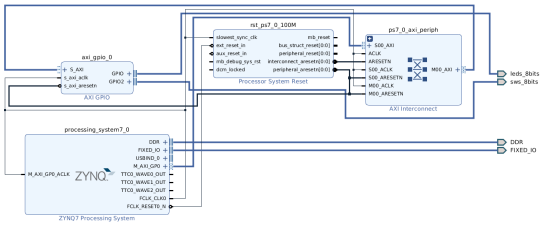

Figure 5. Block Diagram for Part C

A board support package, or BSP, is then generated from our hardware configuration. This allows the software development kit to understand what hardware it has access to and what memory locations the modules are located at. The software then interfaces through memory mapped I/O.

4.4. Entities

Entity	Description
Zinq7	Main ARM core with interconnect and cache
AXI GPIO	GPIO AXI module to interface with switches
System Reset	Resets processor and connected devices
AXI Interconnect	Interconnect fabric for multiple AXI modules

4.5. Design

The other than instantiating a GPIO AXI module, the entire MP was done in software. The program looped through four distinct states spending about 1 second in each state. Within a state, the program sampled the state of the switches through DiscreteRead of the switch channel (channel 0) and then applied the operation for that state to the switch data. The data was then displayed on the LEDs though a DiscreteWrite to the LED channel (channel 1). The modes performed the same functions as Part B and the software flow chart is shown in figure 6.

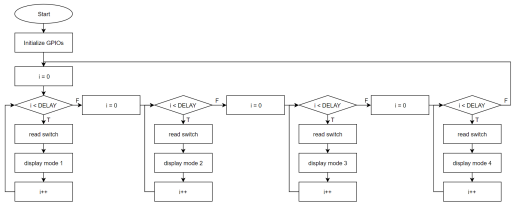

Figure 6. Program Flow for Part C

We settled on the delay iterations number just by trial and error.

4.6. Performance

This a comparably larger design than the other two parts. The AXI GPIO module resided on the Programmable Logic fabric and that contained a lot of logic and registers. The AXI interface is a complex module so it occupied a lot of space after place and route (Figure 7).

Table 7. Resource Usage Part C

Resource	Utilization	Available	Utilization %
LUT	423	53200	0.80
LUTRAM	63	17400	0.36
FF	648	106400	0.61
IO	16	200	8.00
BUFG	1	32	3.13

This design incorporates the PS7 core which will constantly be consuming dynamic power as it loops through the program which is why the dynamic power is dramatically larger than the static power consumed. This desing is also larger than the others using more on chip resources which is also a reason the power has dramatically increased.

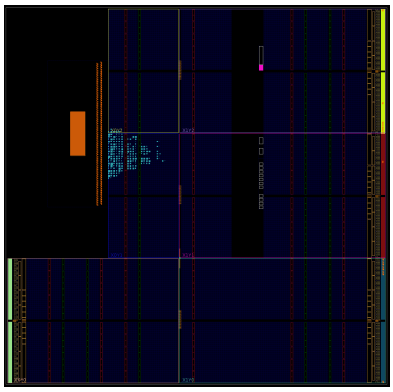

Figure 7. Device Mapping for Part C

Table 8. Resource Usage Part C

Type	Power
Static	0.154 W
Dynamic	1.535 W
Total	**1.689 W**

Again the design was very small so the place and route tool had no problem meeting the timing constraints. There are very reasonable worst setup and hold times for our design.

Table 9. Timing Report Part C

Type	Time
Worst Hold Slack	0.045 ns
Worst Negative Slack	4.292 ns
Worst Pulse Width Slack	4.020 ns

4.7. Difficulties/Bugs

The only difficulty encountered in this section was generating board support packages and making sure the package was up to date with the hardware. We had ran into issues where the BSP did not match the hardware that was loaded on the FPGA.

5. What We Learned

At the start of this machine problem, we were introduced to the Xilinix program Vivado. After developing a simple design which dealt with basic input and output devices on the board, we moved onto a slightly larger design that used a finite state machine. Finally, we implemented a simple SoC using the Zynq system. This system taught us about block design automatons, AXI interfaces, and how to work with embedded software.

6. Environment

We used Lubuntu 17.04 and 18.04 in a virtualbox virtual machine to do all of the work for this MP.