Lab 3 Report

Michael Cooke

October 2021

Contents

1 Part A

1.1 Create BCD Counter

1.1.1 Summary

A BCD counter is a bit more complicated as there has to be an interrupt when the hexadecimal equals A. This took a little bit of logic and a couple flip-flops to recognise when the 9^{th} state was reached and then reset the counter and output a overflow signal. From here scaling to four digits was relatively straightforward.

1.1.2 Difficulties

When implementing the highest local layer I was initially unsure what to do with the overflow of the most significant digit and left it hanging without an end. This cause the program to fail compilation and cost some time and as I initially wrote the counter without busses it took a bit to convert it.

I made the decision to initially write it without busses to allow me to better visualise the logic and check everything. My BCD implementation was done without reading the lab slides, which did increase the difficulty of the task, but I enjoyed the challenge and had enough time due to stuVac. One notable error I made was when I connected the counter wrong during bus implementation and got this waveform: SupportingVideo/CountingWrong

1.1.3 Proof

Here you can seethe first digit count from 0-9 and then overflow correctly: SupportingVideo/BCDWaveform
QuartusFiles/BCDCounter

1.2 Create Ripple Counter

1.2.1 Summary

In order to create a ripple counter it is a simple process of scaling a sequencer to the correct size. This counter can then be used to create the hexadecimal numbers fed into the HexToSSD converter to create a system that increments a hexadecimal number each clock cycle.

1.2.2 Difficulties

This was incredibly straightforward as $16 = 2^4$ and hence the implementation was very straightforward. Additionally, bus implementation saved some time.

1.2.3 Proof

Here you can seethe first digit count from 0-F and then overflow correctly:
QuartusFiles/RippleWaveform
QuartusFiles/RippleClockedCounterBus

2 Part B

2.1 Top Level BCD

2.1.1 Summary

For this task we had to join all the previous components with the BCD counter and make it run on a clock such that the numbers incremented in an appropriate time period with all four displays appearing to be active. To achieve this I fist has to change some of the existing parts to bus implementation. In the end I decided to just wrap the HexToSSD block in a busses and leave the low level implementation as is in order to save time.

2.1.2 Difficulties

The lab 3b notes are far to explicit for this task to be complicated.

2.1.3 Proof

QuartusFiles/FPGABCD

2.2 Program Board

2.2.1 Summary

Before I loaded this part of the lab onto the board I was aware of the problem that would evidently arise. Firstly, as the clock was tied such that the counter would increment at the same rate as the MUX switched the numbers that would be seen to increment 0000,0004,0008,0002,0012,0016,0010,0020...etc. On the other hand, as amusing as this would have been to watch, the clock had to exceed the human eye to make the MUX work and hence both the counting speed and switching would create a pattern of all LED segments active on all digits.

2.2.2 Difficulties

Yay! I was spot on. Also as this was part of the lab is it considered a difficulty?

2.2.3 Proof

SupportingVideo/PreBusFastCount

2.3 Clock Divider

2.3.1 Summary

To slow the clock such that counter was incrementing at a rate visible to the human eye. I used LPM counters as they allow for quick and easy control

2.3.2 Difficulties

The problem wasn't that simple. Due to the difference in gate delay between the implementation of the sequencer and the multiplexer as well as the incredible speed of the clock without slowing the clock I got:

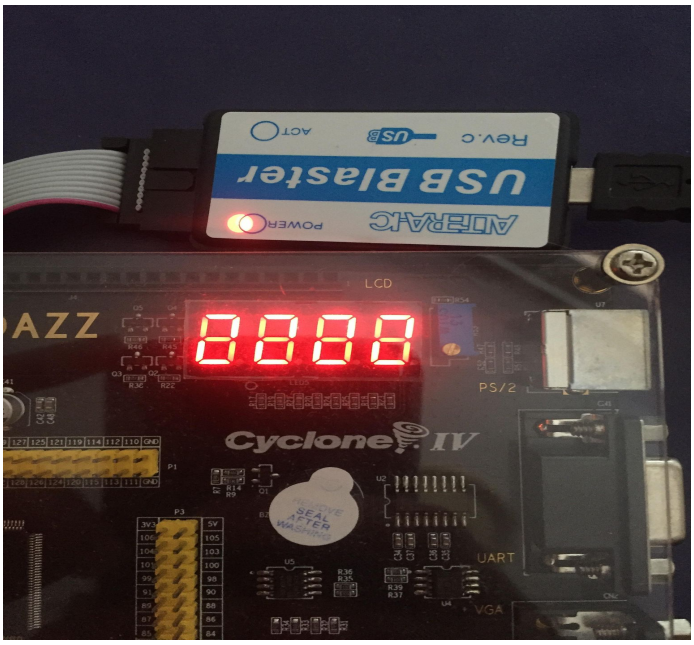

Figure 1: This was hard to get my head around

Eventually, I checked the lab notes and implemented the divider on the multiplexer, as I stepped down the speed I got the shadow seen in SupportingVideo/ResetWorkingShadow. Luckily, on my way down I hit the perfect number to get SupportingVideo/ShadowS which cemented my understanding of the problem. After that I was able to get a clean result.

2.3.3 Proof

SupportingVideo/BCDWorking

2.4 Top Level Ripple

2.4.1 Summary

This task was a simple matter of switching out the BCD counter for the hexadecimal.

2.4.2 Difficulties

None

2.4.3 Proof

QuartusFiles/FPGARipple.bdf

2.5 Program Board

2.5.1 Summary

To load the hexadecimal version onto the board and check that the reset was still working. For good measure I also left this running whilst I was at work in order to check that all digits worked as intended.

2.5.2 Difficulties

Due to the work done with the BCD version of this architecture everything when smoothly.

2.5.3 Proof

SupportingVideo/Base16Working

3 System I/O

Parameter	Min	Typical	Max	Units	Description
VDD	4.5	5	5.5	V	Power
In1	-	0	-	V	Reset Button

Table 1: System Inputs and Outputs

4 Use Case

This system will increment a counter displayed on the four digit LED display in either base 10 or 16 at a given time interval. The counter will reset whenever the left most button is pressed or when it reaches 9999 or FFFF dependant on

the base. All options are hard coded prior to initialisation. The system needs to be powered by 5V via the USB connection or a battery.