DSD Group07 LAB 2

Simon Foucher 260.223.197

Paul Doumet 260.226.189

PART 1: LOG2 DECODER

**1) Goal**Our goal here is to design a log function circuit using a look up table. The output of this circuit should be a 16 bit binary word representing the log base 2 of a 12 bit binary word.
The 12 bit binary word represents a number between 1 and 2; therefore the output should represent a number between 0 and 1.

\*\*Please note that due to a HDD failure, we lost all the data for the first part of the lab. We recovered whatever we could from emails and rebuilt the components last minute while writing this report, but were not able to get the whole thing working a second time on such a short notice. You may check with our TAs who initialed out sheets during lab time to make sure that everything worked. We also got a USB key to back up our data from now on.

 **2) Description**Our circuit will have 2^12 inputs therefore we used MATLAB to help us out with the computation of the log base 2 of our input as well as the 16 bit output it should produce (the 16 bit output represents the log base 2 of our 12 bit input).
In our code “j” represents the number between 1 and 2, “l” represents the log base 2 of “j” and “k” represents the output. All the results for our 3 variables are computed in hexadecimals.


From this code we created our .mif file. Our memory depth is 2048 which represents 2^11 because our most significant bit is always going to be 1 (Numbers are all between 1 and 2) and our width is 16 as our output is a 16 bit binary word.



This symbol which represents a LUT was built using the VHDL code included inside the folder. The mif file is inserted in the VHDL code and will be used as the look up table for our LUT. We have 2 inputs: one that represents a 12 bit binary word and the other one is just a clock. Our Look up table has 2^11= 2048 entries due to the fact that our most significant bit is ignored ( it is always equal to 1 for our input). Our only output is a 16 bit binary word that represents the log2 base of our input.

We simulated the circuit and evaluated a few input values to make sure that the output was coherent with them, which was the case.

Afterwards, we were askes to test out: 0000 0000 0000, 1111 1111 1111 1111, 0101 0101 0101 and 1010 1010 1010.

When choosing our input values, since we are representing numbers from 1 to 2, the smallest value we can input is 1, whuch is coded 1000 0000 0000, therefore, we were not able to input all 0s and 0101 0101 0101.

As for the 2 other cases, we got the required output. We also observed different setteling times from one output to the other. (Worst case TSU 5.629ns, Worst case TCO 13.003ns and Worst case TH -3.598ns)

DSD LAB 2 PART 2: LED Segment decoder

**1)Goal**The goal here is to design a circuit that will take a 7 bit binary input representing the digits from 0 to 99 and output the symbol representing this specific input using only one LED. This has to be done while making the outputs active low.

**2) Description**

Main Function: Displays a 7 bit binary number on the right most digit LED of the Cyclone II board, and the ripple blank out on the first Red LED light.

Files created:

Project name: ‘g07\_Lab2.qpf’

Circuit name: ‘g07\_segment\_decoder\_circuit.bdf’

Main component: ‘g07\_segment\_decoder.bsf’

VHDL source code: ‘g07\_segment\_decoder.vhd’

Wave File: ‘g07\_segment\_decoder.vwf’

Inputs:

* Code[6..0] : A 7 bit vector, used to input numbers (in binary) to be displayed. The number can range from 0 to 127
* RippleBlank\_In : Detects if there is a ripple blank signal coming from another digit.

Outputs:

* Segment[6..0] : A 7 bit vector used to code which digit of the LED will be illuminated (We used the bit map provided in the User manual and standard port naming). Each bit represents a segment on the LED display with TRUE = LED[OFF] and FALSE = LED[ON]
* RippleBlank\_Out : Passes a Ripple Blank signal to the next LED display

PIN Assignments:

Implementation:

The number to be displayed (0-127) is imputed using the 7 rightmost switches (binary representation) on the Quartus II board. The leftmost switch is used as Ripple Blank in signal.

Numbers 0-99 will appear on the right most LED digit display, using the ‘alien alphabet’ provided in the lab statement. Numbers 100-127 will cause the LED to go blank.

If ripple blank in is set and the number to be displayed is ‘0’, no digits will be displayed and Ripple blank out will be set (the first red light on the board will be turned on).

The VHDL source code can be seen on the file: ‘g07\_segment\_decoder.vhd’

Testing:

First, we simulated the circuit using a wave file. The signal stabilized in less than 50ns, so since switches will be turned on and off by a human, there shouldn’t be instability issues. We also looked at a few I/O combinations and compared then with the code, to make sure that they matched:

Which came from the VHDL code:

"0000001" when code = "0000000" else--0

"1001111" when code = "0000001" else--1

"0010010" when code = "0000010" else--2

"0000110" when code = "0000011" else--3

"1001100" when code = "0000100" else--4

"0100100" when code = "0000101" else—5

Afterwards, we uploaded the .sof file onto the board to test it. We manually tested every single number from 0 to 99 and compared the displayed digit to the ‘alien alphabet’ provided with the course description. Every time we saw a discrepancy, we adjusted the code. We also tested the Ripple function, and slightly tweaked the VHDL code to ensure that the LED display got turned off and the Ripple Blank Out signal was on (maped to the first red LED light on the board) whenever Ripple Blank In was on and a 0 was to be displayed.

Based on these tests, we know for a fact that the circuit behaves exactly as suppose, and that there are no mistakes in the decoding.