ECEN30 Lab 1 - Introduction to VHDL and FPGAs

Joshua Benfell - 300433229

July 26, 2020

1 Objectives

The primary objective of this lab is to learn the basics of VHDL and the process taken to put this code onto an FPGA. For this, the Vivado development environment will be used to develop the circuit descriptions and simulate them before uploading them to the chip. In addition to this, the location and use of example code is covered to make the overall process faster for future tasks. It is important that we meet the objective and become familiar with VHDL and the tool chain as it will be used in future lab experiments.

2 Methodology

To start off this lab, a counter project was created in Vivado and the ports were defined as a clock input, a direction input, and a count output which was a bus. It as set as a bus as this meant it would take up multiple bits and allow counting to numbers larger than 1.

This project was initially filled with template code for a counter, which Vivado has a wide selection, so it will be useful to refer back to this for other projects as a starting point. There was no copy to clipboard button anywhere and it didn't offer the option to add this file to your project, so the way to use this code was to highlight it all, copy and paste. From here, the IEEE.NUMERIC_STD.ALL library was included which allowed us to use the unsigned data type for keeping track of the count as a variable. For outputting this variable to the LEDs that are meant to be turned on, it needs to be typecast as a std_logic_vector as this is what the bus is wanting as it's type.

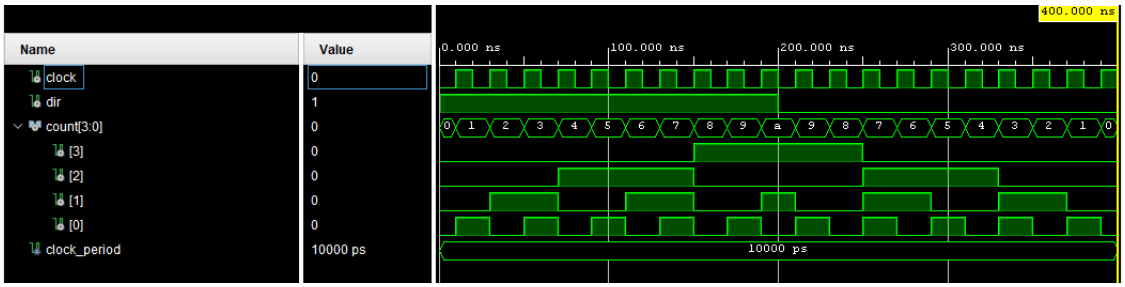

Figure 1: Timing graph from the counter simulation

Before this code can be put onto some hardware it should be tested to ensure it functions as intended. For this a test bench was written that implement the counter entity, maps the signals in that entity to the signals in the test bench and then runs the counters methods by toggling the clock every predefined period and after 200ns, it switches the direction bit

to allow testing of both directions. Figure 1 shows the results of this testing and it can be seen on the line with hexadecimal, that the counter is counting in the desired order.

Now that it's been tested, it had to be implemented in software. This involved selecting which pins were to be used. While this information was provided in the lab script. It won't always be, so how this selection would be done in future projects is by looking at the silk screen on the circuit board to determine which pins we want to use. In the Vivado software package, it is possible to select the IO standard used on the pins. This initially displays "default (LVCMOS18)", which could be taken as default is selected and this is what it is. However, this is not the case and it is actually, this is the default you should select, which is very confusing for this package and something to remember for the future.

The code could finally be uploaded to the FPGA. As this was running at the base clock speed of the chip, it as counting too fast for it to be perceived. All that was visible was 4 always on LEDs at varying brightnesses as the on off time acted as a PWM signal. To fix this, the number of bits that had to be counted was increased to 28 and the 4 MSBs were selected as the output. This was more visible. This didn't get simulated as the time it would have to simulate over would've taken far too long.

The circuit boards have more display options on them than just the LEDs. One of the options is seven segment displays. To use this, a decoder entity for the displays was made. This took all the expected inputs and mapped them to an output combination this is done through a lookup table instead of traditional electronics which will speed up the conversion. The entity is them registered in the counter so that it can be used, and the outputs mapped to the required ports in this component. Once mapping the pins was done, the code could be uploaded, which turned all the displays on. Something to investigate next time is how to do a single display.

Up until this point, the program has been uploaded to volatile memory, which means that when the board was reset, the counter program didn't load in. To store the program on the board so that it can be reused after a power loss, it was uploaded to the non-volatile flash memory.

3 Code

3.1 Counter

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity counter is
6      Port (
7              clk : in STD_LOGIC;
8              direction : in STD_LOGIC;
9              count_out : out STD_LOGIC_VECTOR (3 downto 0);
10             Seg_Out : out STD_LOGIC_VECTOR (6 downto 0)
11         );
12 end counter;
13
14 architecture Behavioral of counter is
15
16 -- PART 1
17 --signal count_int : unsigned(3 downto 0) := (others => '0');
18 -- END PART 1
19 -- PART 2
```

```
signal count_int : unsigned(27 downto 0) := (others => '0'); -- make it 28 bits long
-- END PART 2

--PART 3
Component ssd_decoder is  -- This registers the entity ssd_decoder in this project.
    Port (
           Bin : in STD_LOGIC_VECTOR (3 downto 0);
           Seg_Out : out STD_LOGIC_VECTOR (6 downto 0)
          );
end Component;
-- END PART 3


begin

process (clk)  -- Forces Sequential Operation
begin
   if clk='1' and clk'event then
      if direction='1' then  -- Comparison with char?
         count_int <= count_int + 1; -- Push count_int + 1 into count_int
      else
         count_int <= count_int - 1;
      end if;
   end if;
end process;

-- PART 1
-- count_out <= std_logic_vector(count_int);  -- Type cast from unsigned to std_logic_vector
-- END PART 1
-- PART 2
count_out <= std_logic_vector(count_int(27 downto 24)); -- Select 4MSBs of the 28 bit long thing
-- END PART 2

-- PART 3
ssd: ssd_decoder PORT MAP (
    Bin => std_logic_vector(count_int(27 downto 24)),
    Seg_Out => Seg_Out
);
-- END PART 3

end Behavioral;
```

code/counter.vhd

3.2 Counter Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity counter_tb is
--  Port ( );
end counter_tb;

architecture Behavioral of counter_tb is

Component Counter
```

```
port(
    clk : in STD_LOGIC;
    direction : in std_logic;
    count_out : out std_logic_vector(3 downto 0)
);
end Component;

signal clock : std_logic := '0';
signal dir : std_logic := '0';
signal count : std_logic_vector(3 downto 0) := "0000";

constant clock_period : time := 10ns;

begin

uut: counter PORT MAP (
    clk => clock,
    direction => dir,
    count_out => count
);

process
begin
    dir <= '1';
    wait for 200ns;
    dir <= '0';
    wait for 200ns;
end process;


clock <= not clock after clock_period;

end Behavioral;
```

code/counter_tb.vhd

3.3 Seven Segment Display Decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ssd_decoder is
    Port ( Bin : in STD_LOGIC_VECTOR (3 downto 0);
           Seg_Out : out STD_LOGIC_VECTOR (6 downto 0));
end ssd_decoder;

architecture Behavioral of ssd_decoder is

begin

with Bin select
    Seg_Out <= "1111001" when "0001",   -- 1
               "0100100" when "0010",   -- 2
               "0110000" when "0011",   -- 3
               "0011001" when "0100",   -- 4
               "0010010" when "0101",   -- 5
               "0000010" when "0110",   -- 6
               "1111000" when "0111",   -- 7
               "0000000" when "1000",   -- 8
               "0010000" when "1001",   -- 9
```

```
                "0001000" when "1010",  -- 10 (A)
                "0000011" when "1011",  -- 11 (B)
                "1000110" when "1100",  -- 12 (C)
                "0100001" when "1101",  -- 13 (D)
                "0000110" when "1110",  -- 14 (E)
                "0001110" when "1111",  -- 15 (F)
                "1000000" when others;  -- 0


end Behavioral;
```

code/ssd_decoder.vhd

4 Questions

4.1 Q1

The name of the Xilinx chip we are using is the Artix 7 (Xilinx Part Number XC7A100T-1CSG324C) on the Nexys4DDR board by Digilent. This chip has 324 pins.

4.2 Q2

This chip operates above 450MHz internally.

4.3 Q3

If we assume that the our clock is running at 450MHz, as stated by the data sheet on the wiki, we know that the bit only flips on the rising edge, making the 24th bit, the equivalent of a 25th bit if the clock is considered the first bit as bit 0 is triggering at half the clock frequency. Therefore, the frequency of bit 24 can be found using eq. (1).

$$\begin{aligned} F &= \frac{450 \times 10^6}{2^{25}} \\ &= 1.49\text{Hz} \end{aligned} \tag{1}$$

4.4 Q4

Using a clock frequency of 450MHz again, getting a frequency of ~3KHz can be done by reversing eq. (1). This is shown in eq. (2)

$$\begin{aligned} 3000 &= \frac{450 \times 10^6}{2^n} \\ 2^n &= \frac{450 \times 10^6}{3000} \\ &= 150000 \\ n &= \log_2 150000 \\ &= 17.19 \\ &\approx 17 \end{aligned} \tag{2}$$

The result of eq. (2) gives a frequency of 3.4KHz.

4.5 Q5

It assigns the signal count_out to the variable count_int which is configured as an output to some on board LEDs turning on the ones dictated by what's stored in count_int and now count_out.

4.6 Q6

A test bench is used to provide a simulated environment for testing your code before it goes onto the FPGA. This is useful as it allows us to simulate timings, whether it is possible to load the code onto the FPGA physically. This is all easier in a test bench as you don't need special equipment to see what your hardware is doing, the built in simulations provide an accurate enough representation of the chip and are able to reproduce what you can expect on the hardware. Additionally, you don't need to have an FPGA on hand to run your code.