Ben Linam
Dylan Hall
EEL4712L
14 Sept. 2018
Lab 2

EEL4712L
Lab 2 – DE1-SoC Tutorial
Report

Preliminary Setup

Overview

*For this lab, we will be orienting ourselves with the new **IDE** used for **VHDL** in this course, **Quartus Prime.** Also, we will be learning some coding conventions and syntax for **VHDL**. We will be learning how to write logic designs and form them into one project which will enable us to test the logic on an **Altera DE1-SoC FPGA.***

Materials

- An Altera DE1-SoC kit
- A computer with Quartus Prime

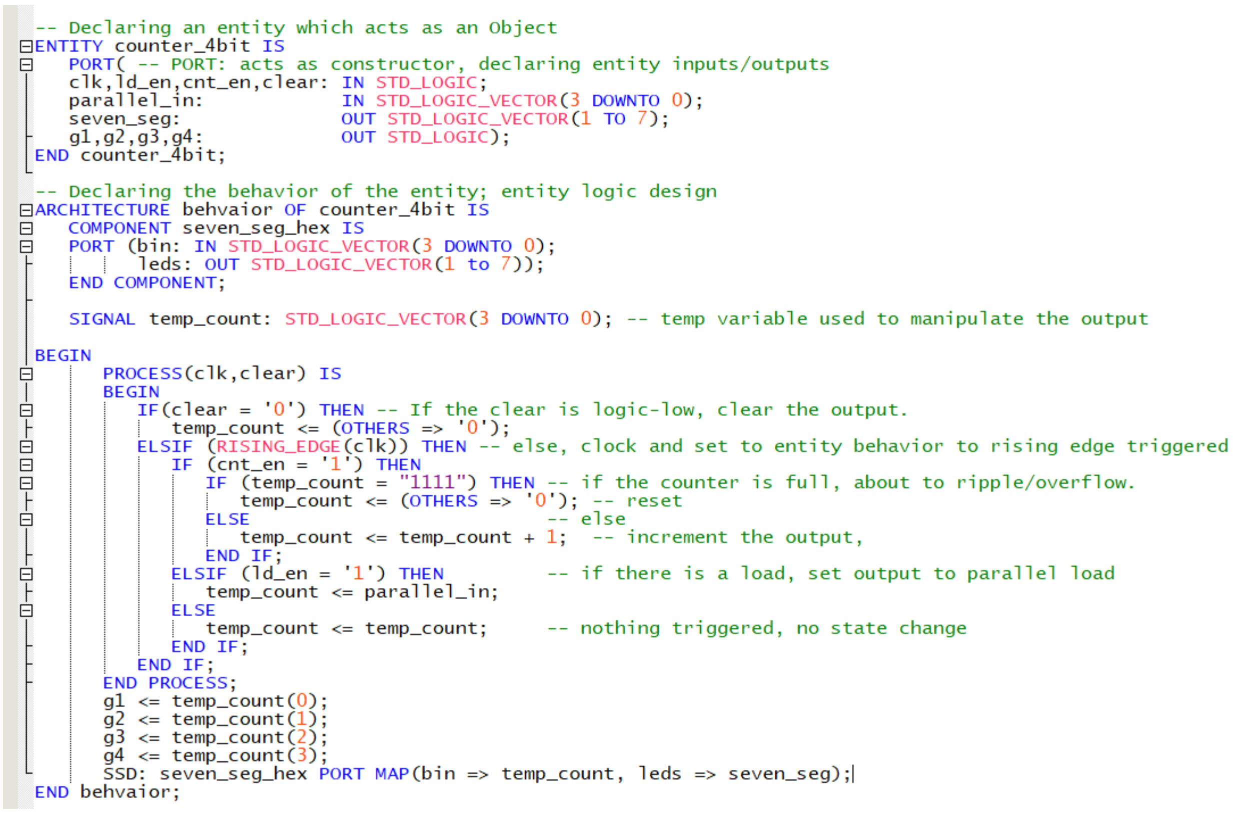
Procedure

*We will be using **ENTITY** designs in this laboratory. We will be designing a **4-bit-up-counter** as well as a **seven-segment display.** Both of which will be added to the same project in order for the **4-bit-counter**'s output to be pinned to the **seven segments** input. Once designed, we will be using the **Pin Assignment** utility provided by **Quartus Prime** to determine the **Software → Hardware** implementations.*

*Such that, we will be setting the logic inputs/outputs from our **VHDL** designs to physical pins on our **Altera DE1-SoC** boards.*

- Realization/Simulation –

*Below you will see a screen capture of the **Quartus Prime logic design** of the **4-bit up-counter**.*



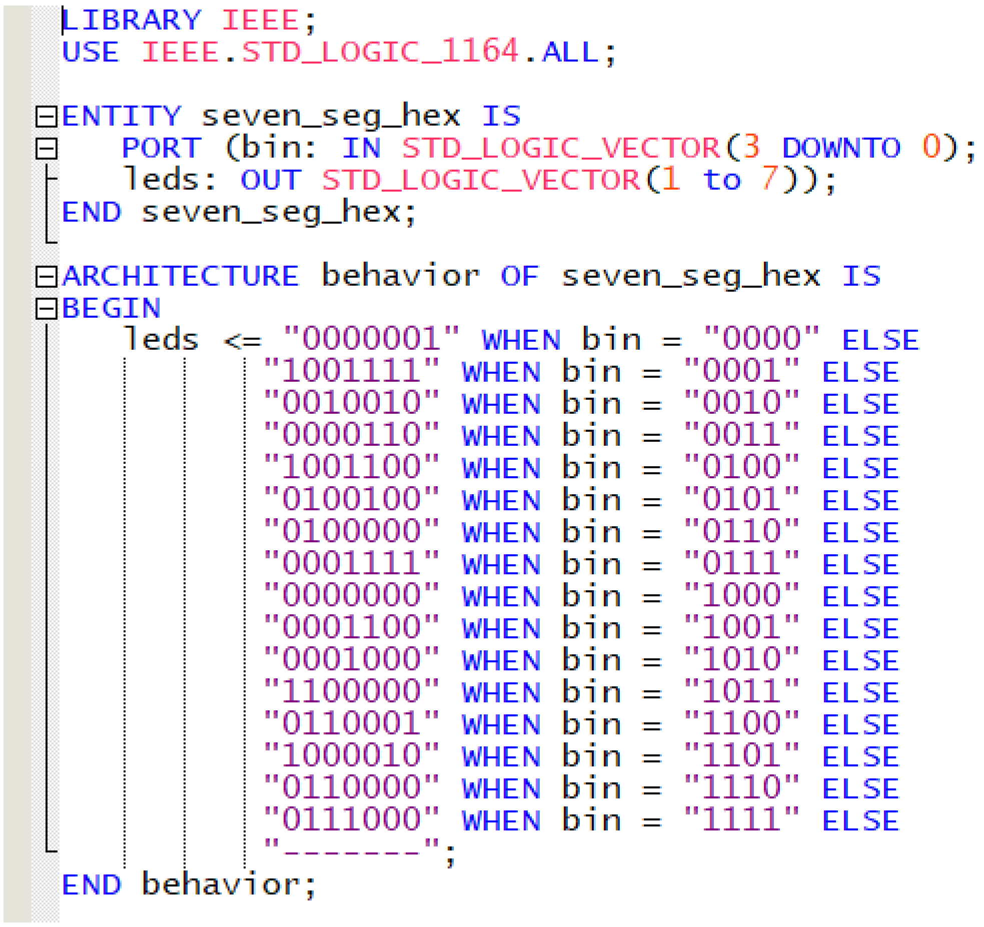
```vhdl
-- Declaring an entity which acts as an Object
ENTITY counter_4bit IS
   PORT( -- PORT: acts as constructor, declaring entity inputs/outputs
   clk,ld_en,cnt_en,clear: IN STD_LOGIC;
   parallel_in:            IN STD_LOGIC_VECTOR(3 DOWNTO 0);
   seven_seg:              OUT STD_LOGIC_VECTOR(1 TO 7);
   g1,g2,g3,g4:            OUT STD_LOGIC);
END counter_4bit;

-- Declaring the behavior of the entity; entity logic design
ARCHITECTURE behvaior OF counter_4bit IS
   COMPONENT seven_seg_hex IS
   PORT (bin: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         leds: OUT STD_LOGIC_VECTOR(1 to 7));
   END COMPONENT;

   SIGNAL temp_count: STD_LOGIC_VECTOR(3 DOWNTO 0); -- temp variable used to manipulate the output

BEGIN
      PROCESS(clk,clear) IS
      BEGIN
         IF(clear = '0') THEN -- If the clear is logic-low, clear the output.
            temp_count <= (OTHERS => '0');
         ELSIF (RISING_EDGE(clk)) THEN -- else, clock and set to entity behavior to rising edge triggered
            IF (cnt_en = '1') THEN
               IF (temp_count = "1111") THEN -- if the counter is full, about to ripple/overflow.
                  temp_count <= (OTHERS => '0'); -- reset
               ELSE                      -- else
                  temp_count <= temp_count + 1;  -- increment the output,
               END IF;
            ELSIF (ld_en = '1') THEN         -- if there is a load, set output to parallel load
               temp_count <= parallel_in;
            ELSE
               temp_count <= temp_count;     -- nothing triggered, no state change
            END IF;
         END IF;
      END PROCESS;
      g1 <= temp_count(0);
      g2 <= temp_count(1);
      g3 <= temp_count(2);
      g4 <= temp_count(3);
      SSD: seven_seg_hex PORT MAP(bin => temp_count, leds => seven_seg);
END behvaior;
```

Figure 1. Screen capture displaying the **Quartus Prime logic design** for the **4-bit-counter.**

*For this chip to functionally "count," both the inputs **cnt_en** and **clk** must be **logic-1** for the chip to change state. This entity design also handles possible parallel load inputs. Parallel load is an input into the chip which acts as the starting place for the initial state. Such that, if a parallel load of **"1000,"** is provided, the counter will start at **"1000"**, and continue counting, **"1001, 1010, ..., 1111."***

- Realization/Simulation Cont.–

*Below you will see a screen capture of the **Quartus Prime logic design** of the **seven-segment hex display**.*



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY seven_seg_hex IS
    PORT (bin: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    leds: OUT STD_LOGIC_VECTOR(1 to 7));
END seven_seg_hex;

ARCHITECTURE behavior OF seven_seg_hex IS
BEGIN
    leds <= "0000001" WHEN bin = "0000" ELSE
            "1001111" WHEN bin = "0001" ELSE
            "0010010" WHEN bin = "0010" ELSE
            "0000110" WHEN bin = "0011" ELSE
            "1001100" WHEN bin = "0100" ELSE
            "0100100" WHEN bin = "0101" ELSE
            "0100000" WHEN bin = "0110" ELSE
            "0001111" WHEN bin = "0111" ELSE
            "0000000" WHEN bin = "1000" ELSE
            "0001100" WHEN bin = "1001" ELSE
            "0001000" WHEN bin = "1010" ELSE
            "1100000" WHEN bin = "1011" ELSE
            "0110001" WHEN bin = "1100" ELSE
            "1000010" WHEN bin = "1101" ELSE
            "0110000" WHEN bin = "1110" ELSE
            "0111000" WHEN bin = "1111" ELSE
            "-------";
END behavior;
```

-

Figure 2. Screen capture displaying the **Quartus Prime logic design** for the **seven-segment display.**

- Code Logic -

*The entity in this **VHDL** code acts like a class like that of **C++**. This creates the black box we will be designing the logic for. Then, we sett up the entities **PORTS**, which are just the input and output pins on the black box. These allow us to provide inputs to run the black box based on a logic behavior we define later.*

*In the **PROCESS** of the design, when the input **clear** is logic-0, then the design will clear its outputs, restarting either at **"0000,"** or whatever parallel load is provided. Otherwise, it will toggle states based on the rising edge of the **clock** input.*

*If the **cnt_en** is enabled (count enable) and **temp_count** (temporary buffer to manipulate, in place of **cnt_out**) is at **"1111,"** then set **temp_count** to **"0000."** This segment of logic acts as the final state of the **4-bit-counter,** which explicitly only counts to a maximum of four bits.*

*Otherwise, if **temp_count** does not equal **"1111,"** and **cnt_en** is enabled, **temp_count** in incremented.*

*If **cnt_en** is not enabled, but **ld_in** is, this will tell the design that a parallel load is intended, setting **temp_count** to any provided **parallel_in**.*

*If none of the above cases are provided, then **temp_count** remains the same.*

*Finally, we set the four output pins on the black box, **g1, g2, g3** and **g4,** to the different bits loaded into **temp_count.***

*Quartus prime uses **Little Endian** style of bit format in which the **LSB (least significant bit)** is stored in the $\mathbf{0}^{th}$index of any **bit vector**.*

The final call, seen here:

SSD: seven_seg_hex PORT MAP(bin => temp_count, leds => seven_seg)

*Acts as a function call which enables the seven segment display drivers on the **Altera DE1-SoC Board**. It also maps whatever input/output pins you want to one another. Seen here, we are mapping **bin** to **temp_count,** which tells the software/hardware that the output of the **4-bit-counter** will be what drives **bin (binary input)** of the hex display. Also that the variable **led** seen in **Figure 2.** will be mapped to the bit vector **seven_seg.***

- Logic/Truth Tables –

Below you will see the logic table for the logic implementation seen in ***Figure 1.***

Term	G4	G3	G2	G1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Table 1. Truth table for the **4-bit counter.**