

ECE 255 Lab 4: Basic State Machine on FPGA

Author: Austin Strobel

Lab 4

ECE 255 Section 001

Date of completion: 11/21/2022

## **Introduction:**

The goal of this lab was to learn to use Vivado and our FPGA boards. Students were then tasked with creating a 7-segment display for task 1 and a debounced pushbutton (BTN) pulse, binary counter, and 7-segment display for task 2. For Task 1, we had to create 7 functions to drive the display. This binary-to-hex decoder inputs a positive, 4-bit integer (b3b2b1b0) on switches SW3-SW0 and shows the equivalent hex digit on the FPGA's 7-segment display. We had to use K-maps based off of the given truth table for each function to find the MSOP and turn that into VHDL code. AN1 (an anode) was also used for the display. AN0, AN2, and AN3 were left off. For Task 2, we added a debounced push button and implemented a 3-bit up counter whose output is converted to hex and shown on the FPGA's 7-segment display. We were required to use the 3 given files, d\_ff.vhd, db\_pulse.vhd, and clock\_divide.vhd. In our top VHDL architecture, we included these three files as components. We had to use 3 flip flops (as shown in the schematic below) for our 3-bit up counter. We had to derive Boolean expressions for the next states. The assumptions for our up-counter are that  $Q_{sub}(n + 1)$  is equal to D and since each flip flop can store one bit, we need 3 flip flops for a 3-bit counter. 3-bit input is taken from the 3-bit state output of the up-counter (not from switches). The bit file is then uploaded to the BASYS3 board. We confirm our solution based on observation and trying multiple input combinations.

The motive of this lab was to get exposed to Vivado and learn how to code a bit of VHDL to simulate counters and to further our knowledge of using components from other files. Knowing what the different outputs should have been was our way of confirming that our code and proper alterations of the constraints files were correct. We were also motivated with extra credit to build upon task 2 to grant the user the ability to switch from an up-counter to a down-counter (and vice versa).

## **Methods:**

I followed the directions in the Vivado tutorial and in the Lab 4 pdf. I altered VHDL code, the constraints files, and programmed my FGPA to test the outputs based on the various combinations of inputs (flipping switches and pressing the debounced pushbutton for task 2). I also referenced Jianjun's given code on his Canvas announcement (much appreciated). I verified that the schematics were accurate for tasks 1 and 2 based on the Boolean expressions that I derived.

## **Design Description:**

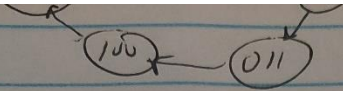
For task 1, each of the 7 Boolean expressions is based on its corresponding K-map (that yields an MSOP). Those K-maps are based off of the given truth table. For task 2, I had 3 K-maps for the outputs' Boolean expressions. I also used port maps for the clock divide, the db pulse, the three flip flops, and the 7-segment display itself. The inputs for task 1 are b0 – b3, and the

outputs are A-G and an0 – an3. The inputs for task 2 are b\_input and clk. The outputs are A-G and an0 – an3. The schematics verify these.

(Design Description continued)

State table:

J bits: 1, 2, 3  
 $\Delta Q_{n+1} = D$



State Table				For 3-bit			Counter		
	Present State			Next State			I/Os of FF ← ?		
	$Q_a$	$Q_b$	$Q_c$	$Q_{a+1}$	$Q_{b+1}$	$Q_{c+1}$	$D_a$	$D_b$	$D_c$
0	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	0
2	0	1	0	0	1	1	0	1	1
3	0	1	1	1	0	0	1	0	0
4	1	0	0	1	0	1	1	0	1
5	1	0	1	1	1	0	1	1	0
6	1	1	0	1	1	1	1	1	1
7	1	1	1	0	0	0	0	0	0

K-maps for task 2:

K-map for  $D_a$

	$\bar{Q}_b \bar{Q}_c$	$\bar{Q}_b Q_c$	$Q_b Q_c$	$Q_b \bar{Q}_c$
$\bar{Q}_a$			1	
$Q_a$	1	1		1

$$D_a = Q_a \bar{Q}_b + Q_a \bar{Q}_c + \bar{Q}_a Q_b Q_c$$

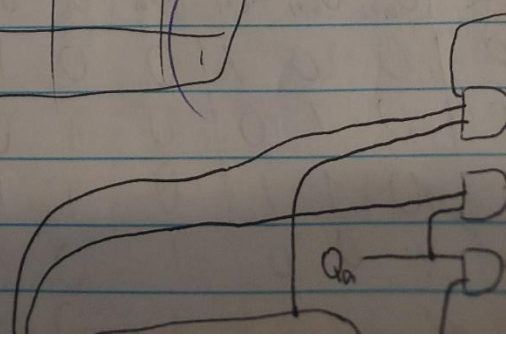
For  $D_b$ :

	$\bar{Q}_b \bar{Q}_c$	$\bar{Q}_b Q_c$	$Q_b Q_c$	$Q_b \bar{Q}_c$
$\bar{Q}_a$		1		1
$Q_a$		1		1

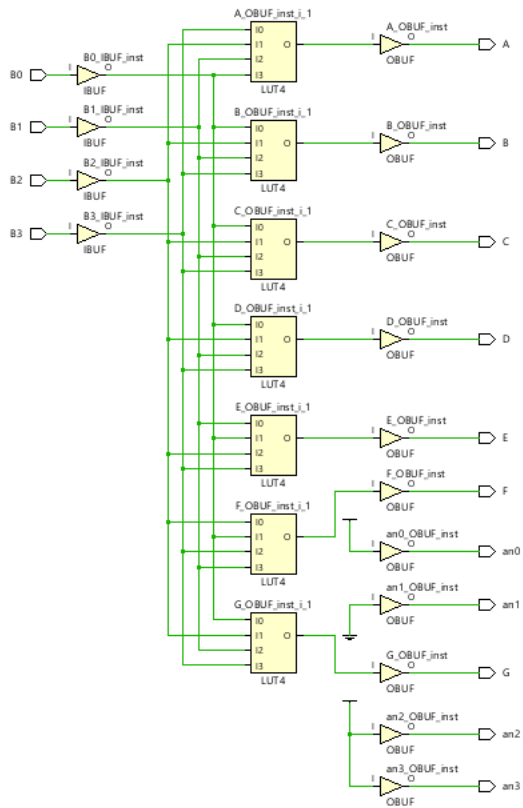
$$D_b = \bar{Q}_b Q_c + Q_b \bar{Q}_c = Q_b \oplus Q_c$$

For  $D_c$ :

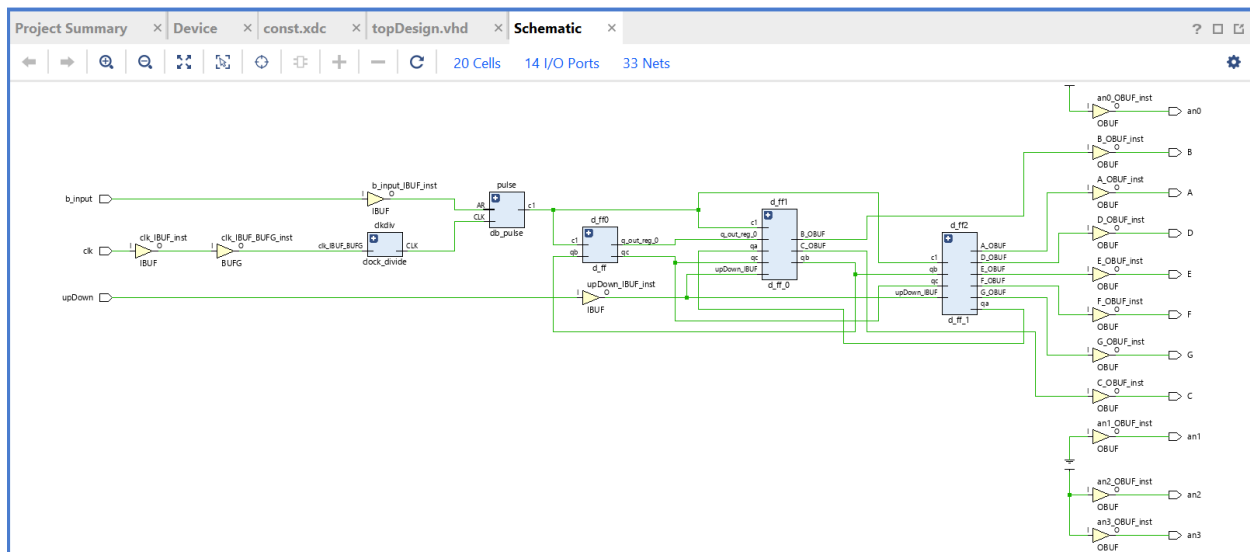
	$\bar{Q}_b \bar{Q}_c$	$\bar{Q}_b Q_c$	$Q_b Q_c$	$Q_b \bar{Q}_c$
$\bar{Q}_a$	1			1
$Q_a$	1			1

$$D_c = \bar{Q}_c$$


## Schematic for task 1:



## Schematic for task 2:



## Results:

Tasks 1 and 2 were verified by Jianjun, so it can be concluded that I have satisfied those requirements. The code, the Boolean logic, and the schematics all lead to the conclusion that I successfully completed those tasks. The derivations for the K-maps based on the truth table for task 1 are indeed correct. The k-maps based on the state table for task 2 are also right.

## Conclusions:

I learned that you can make your own components to use in other circuits in Vivado (and that those components are entities in their own file but are components when used as such in the architecture of another file). I learned that you can upload a bitstream as well as generate one. I also learned that switching between an up counter and a down counter, using an if statement, just requires putting a NOT around the three input statements.

## Appendix

Task 1 Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity task1Lab4Constraints is
  Port ( B0 : in STD_LOGIC;
        B1 : in STD_LOGIC;
        B2 : in STD_LOGIC;
        B3 : in STD_LOGIC;
        A : out STD_LOGIC;
        B : out STD_LOGIC;
        C : out STD_LOGIC;
        D : out STD_LOGIC;
        E : out STD_LOGIC;
        F : out STD_LOGIC;
        G : out STD_LOGIC;
        an1 : out STD_LOGIC;
```

```
an0 : out STD_LOGIC;  
an2 : out STD_LOGIC;  
an3 : out STD_LOGIC);
```

```
end task1Lab4Constraints;
```

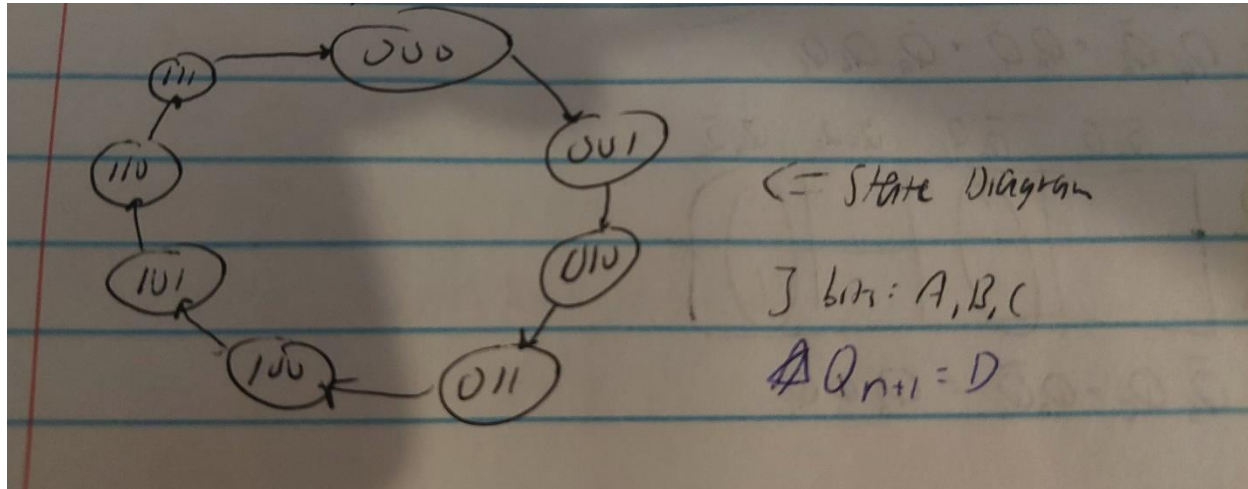
```
architecture Behavioral of task1Lab4Constraints is
```

```
begin
```

```
an1 <= '0';  
an0 <= '1';  
an2 <= '1';  
an3 <= '1';  
A <= (B0 AND NOT B2 AND NOT B1 AND NOT B3) OR (B2 AND NOT B3 AND NOT B1 AND NOT B0) OR (B3  
AND B0 AND B2 AND NOT B1) OR (B0 AND B3 AND NOT B2 AND B1);  
B <= (NOT B1 AND B0 AND B2 AND NOT B3) OR (B3 AND NOT B0 AND B2 AND NOT B1) OR (B0 AND B3  
AND B1) OR (B2 AND NOT B0 AND B1);  
C <= (B1 AND NOT B3 AND NOT B2 AND NOT B0) OR (B3 AND NOT B0 AND B2 AND NOT B1) OR (B2 AND  
B3 AND B1);  
D <= (NOT B1 AND NOT B3 AND NOT B2 AND B0) OR (NOT B3 AND NOT B1 AND B2 AND NOT B0) OR (B0  
AND B2 AND B1) OR (NOT B0 AND NOT B2 AND B1 AND B3);  
E <= (B0 AND NOT B3) OR (B0 AND NOT B1 AND NOT B2) OR (NOT B3 AND B2 AND NOT B1);  
F <= (B1 AND NOT B2 AND NOT B3) OR (B0 AND NOT B2 AND NOT B3) OR (B1 AND NOT B3 AND B0) OR  
(B3 AND B0 AND B2 AND NOT B1);  
G <= (NOT B2 AND NOT B1 AND NOT B3) OR (B1 AND B2 AND NOT B3 AND B0) OR (B3 AND B2 AND NOT  
B0 AND NOT B1);
```

```
end Behavioral;
```

## Task 2 Code and State Diagram:



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

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity topDesign is
```

```
Port (
```

```
  A : out STD_LOGIC;
  B : out STD_LOGIC;
  C : out STD_LOGIC;
  D : out STD_LOGIC;
  E : out STD_LOGIC;
  F : out STD_LOGIC;
  G : out STD_LOGIC;
  an0 : out STD_LOGIC;
  an1 : out STD_LOGIC;
  an2 : out STD_LOGIC;
  an3 : out STD_LOGIC;
  b_input : in STD_LOGIC;
  -- upDown : in STD_LOGIC;
  clk : in STD_LOGIC
);
```

```
end topDesign;
```



architecture Behavioral of topDesign is  
component task1Lab4Constraints is

```
port(
    B0 : in STD_LOGIC;
    B1 : in STD_LOGIC;
    B2 : in STD_LOGIC;
    B3 : in STD_LOGIC;
    A : out STD_LOGIC;
    B : out STD_LOGIC;
    C : out STD_LOGIC;
    D : out STD_LOGIC;
    E : out STD_LOGIC;
    F : out STD_LOGIC;
    G : out STD_LOGIC;
    an1 : out STD_LOGIC;
    an0 : out STD_LOGIC;
    an2 : out STD_LOGIC;
    an3 : out STD_LOGIC);
```

end component;

component clock\_divide is

```
port(
    clk_in : in STD_LOGIC;
    clk_slow : out STD_LOGIC);
```

end component;

component db\_pulse is

```
port(
    Clk1: in STD_LOGIC;
    Key: in std_logic;
    pulse: out std_logic);
```

end component;

component d\_ff is

```
port(
    d_in: in std_logic;
    clock: in std_logic;
    q_out: out std_logic);
```

end component;

```
signal c0: std_logic; -- clock slow down using the clk divide
signal c1: std_logic; -- the clk pulse generated by the bottom press
signal da : STD_LOGIC;
signal db : STD_LOGIC;
signal dc : STD_LOGIC;
signal qc, qb, qa : STD_LOGIC;
```

begin

```

clkdiv: clock_divide port map (clk_in => clk, clk_slow => c0);
pulse: db_pulse port map (Clk1 => c0, Key => b_input, pulse => c1);

da <= (qa AND NOT qb) OR (qa AND NOT qc) OR (NOT qa AND qb AND qc);
db <= qb XOR qc;
dc <= NOT qc;

d_ff0: d_ff port map (d_in => dc, clock => c1, q_out => qc);
d_ff1: d_ff port map (d_in => db, clock => c1, q_out => qb);
d_ff2: d_ff port map (d_in => da, clock => c1, q_out => qa);

sevenSeg: task1Lab4Constraints port map (B0 => qc, B1 => qb, B2 => qa, B3 => '0', A => A, B => B, C => C,
D => D, E => E, F => F, G => G, an0 => an0, an1 => an1, an2 => an2, an3 => an3);

end Behavioral;

```

## Extra credit:

Switching between an up counter and a down counter meant modifying the constraints file, adding a variable to store a 1 or a 0 for the switch position, and adding an if statement for qa, qb, and qc. Adding a NOT to qa, qb, and qc changed the counter from an up counter to a down counter.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity topDesign is
  Port (
    A : out STD_LOGIC;
    B : out STD_LOGIC;
    C : out STD_LOGIC;
    D : out STD_LOGIC;
    E : out STD_LOGIC;
    F : out STD_LOGIC;
    G : out STD_LOGIC;

```

```

        an0 : out STD_LOGIC;
        an1 : out STD_LOGIC;
        an2 : out STD_LOGIC;
        an3 : out STD_LOGIC;
        b_input : in STD_LOGIC;
        upDown : in STD_LOGIC;
        clk : in STD_LOGIC
    );
end topDesign;

```

architecture Behavioral of topDesign is  
 component task1Lab4Constraints is

```

    port(
        B0 : in STD_LOGIC;
        B1 : in STD_LOGIC;
        B2 : in STD_LOGIC;
        B3 : in STD_LOGIC;
        A : out STD_LOGIC;
        B : out STD_LOGIC;
        C : out STD_LOGIC;
        D : out STD_LOGIC;
        E : out STD_LOGIC;
        F : out STD_LOGIC;
        G : out STD_LOGIC;
        an1 : out STD_LOGIC;
        an0 : out STD_LOGIC;
        an2 : out STD_LOGIC;
        an3 : out STD_LOGIC);

```

end component;

component clock\_divide is

```

    port(
        clk_in : in STD_LOGIC;
        clk_slow : out STD_LOGIC);

```

end component;

component db\_pulse is

```

    port(
        Clk1: in STD_LOGIC;
        Key: in std_logic;
        pulse: out std_logic);

```

end component;

component d\_ff is

```

    port(

```

```

    d_in: in std_logic;
    clock: in std_logic;
    q_out: out std_logic);
end component;

signal c0: std_logic; -- clock slow down using the clk divide
signal c1: std_logic; -- the clk pulse generated by the bottom press
signal da : STD_LOGIC;
signal db : STD_LOGIC;
signal dc : STD_LOGIC;
signal qc, qb, qa : STD_LOGIC;
signal B0, B1, B2, B3 : STD_LOGIC;

begin

clkdiv: clock_divide port map (clk_in => clk, clk_slow => c0);
pulse: db_pulse port map (Clk1 => c0, Key => b_input, pulse => c1);

da <= (qa AND NOT qb) OR (qa AND NOT qc) OR (NOT qa AND qb AND qc);
db <= qb XOR qc;
dc <= NOT qc;

d_ff0: d_ff port map (d_in => dc, clock => c1, q_out => qc);
d_ff1: d_ff port map (d_in => db, clock => c1, q_out => qb);
d_ff2: d_ff port map (d_in => da, clock => c1, q_out => qa);

process (upDown)
begin
if (upDown = '1') then
    B0 <= qc;
    B1 <= qb;
    B2 <= qa;
    B3 <= '0';
elsif (upDown = '0') then
    B0 <= NOT qc;
    B1 <= NOT qb;
    B2 <= NOT qa;
    B3 <= '0';
end if;
end process;

sevenSeg: task1Lab4Constraints port map (B0, B1, B2, B3, A => A, B => B, C => C, D => D, E => E,
F => F, G => G, an0 => an0, an1 => an1, an2 => an2, an3 => an3);

```

end Behavioral;