ECE 255 Lab 3: VHDL, FPGA, and Multiplexers

Author: Austin Strobel

Lab 3

ECE 255 Section 001

Date of completion: 11/02/2022

Introduction:

The goal of this lab was to learn to use Vivado and our FPGA boards. Students were then tasked with writing VHDL to simulate a 2-to-1 multiplexer, and a 4-to-1 multiplexer (MUX). For Task 1, we had to upload an existing bit file to the FPGA. It was not required to alter any VHDL code for this step. We simply had to confirm that the bit file uploaded to the FPGA simulated a MUX by flipping all possible combinations of switches that are defined in the constraints file – pin V17 for S, pin V16 for A, pin W16 for B, and pin E19 for g. For Task 2, the simple.vhd file from the tutorial had to be modified to implement MUX logic instead of XOR logic. The const.xdc file was used as the constraints file. A, B, and S are the inputs, while g is used as the output. Combinations of flipping switches was the way that we confirmed our output. For Task 3, we were tasked with instantiating a 4-to-1 MUX using three 2-to-1 MUX's. The 2-to-1 MUX from Task 2 becomes a component of the 4-to-1 MUX for Task 3. The inputs for the 4-to-1 MUX are A, B, C, D, S0, and S1, while the output is Y. The constraints file was modified to account for these inputs and output: pin V17 for S0, pin V16 for S1, pin W16 for A, pin W17 for B, pin W15 for C, pin V15 for D, and pin E19 for Y. Of course, generating a bitstream and programming our FPGA (then testing the output by flipping various combinations of switches) was how we confirmed that our 4-to-1 MUX file was, indeed, operating as intended.

The motive of this lab was to get exposed to Vivado and learn how to code a bit of VHDL to simulate simple circuits that we had dealt with earlier in the course: 2-to-1 and 4-to-1 MUX's. 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.

Methods:

I followed the directions in the Vivado tutorial and in the Lab 3 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). I verified that the output for the schematics in these three tasks was correct based on the behavior of a 2-to1 MUX and a 4-to1 MUX on our FPGA. This output can also be displayed on a truth table using the same inputs and outputs laid out in the "Introduction" section for the 2-to-1 MUX and the 4-to-1 MUX.

Design Description:

The 2-to-1 MUX for Task 2 includes two AND gates and an OR gate, as a MUX should. The inputs are clearly shown as S, A, and B, while the output is g. The 2-to-1 MUX for Task 3 doesn't show all the logic gates, but it displays the same inputs and output. The 4-to-1 MUX for Task 3 shows the instantiation of the three 2-to-1 MUX's and contains the input pins A, B, C, D, S0, and S1, as well as the output pin Y.

(Design Description continued)

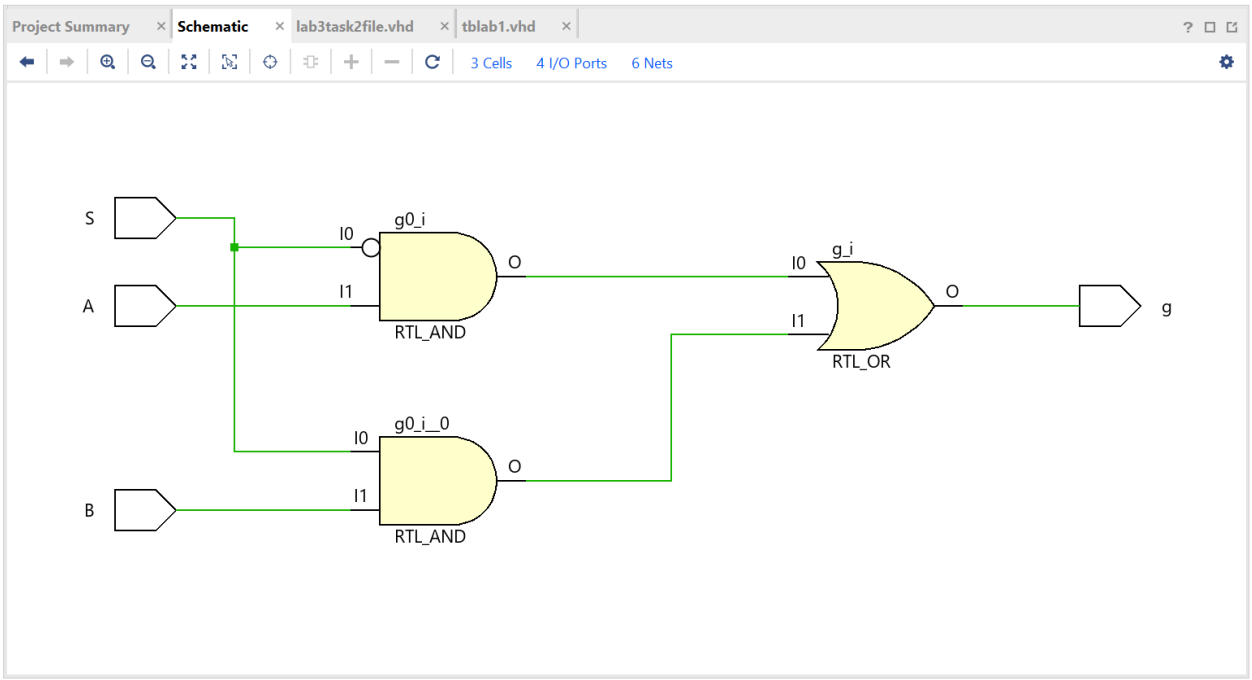
2-to-1 MUX:

g = S'A + SB

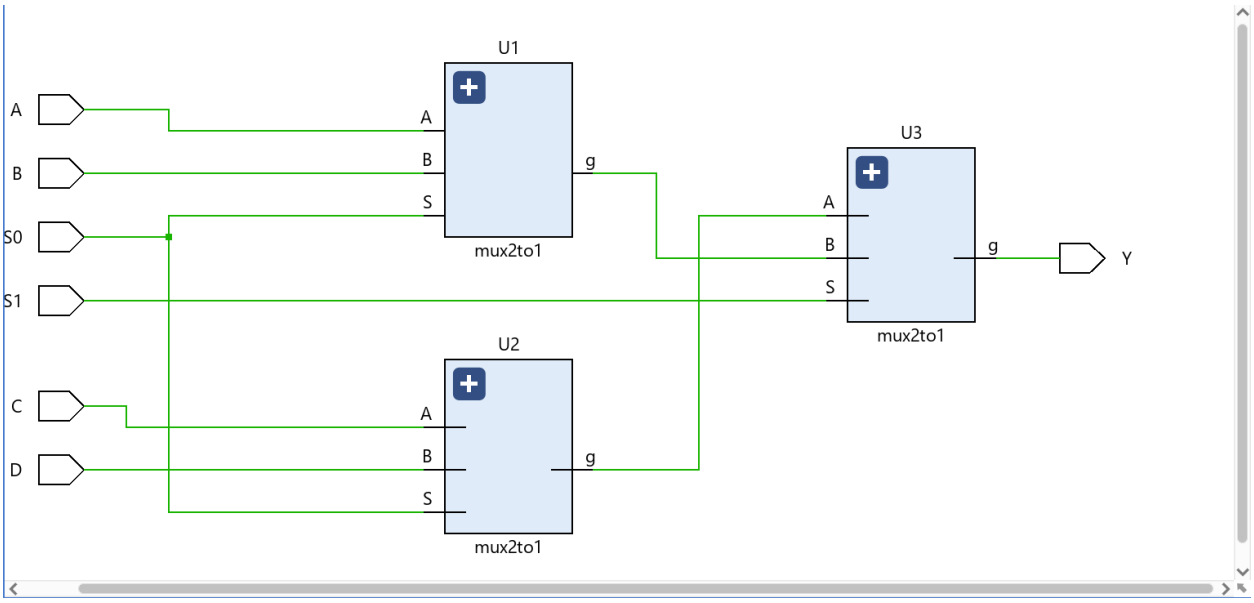
4-to-1 MUX:

Y = S0'S1'A + S0'S1B + S0S1'C+ S0S1D

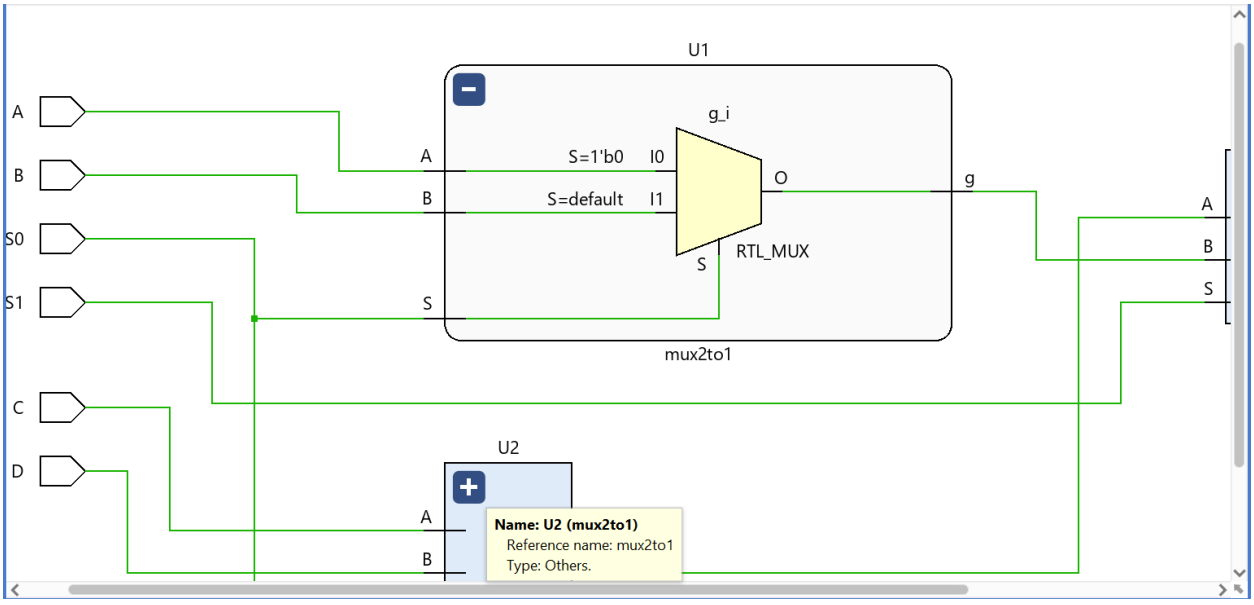
Task 2 2-to-1 MUX:



Task 3 4-to-1 MUX:



Task 3 2-to-1 MUX



Results:

Task 1 contains two pictures of FPGA showing those two different input combinations and the appropriate output. Task 2 contains two pictures of FPGA showing those two different input combinations and the appropriate output. Task 3 contains four pictures of FPGA showing those two different input combinations and the appropriate output. These output combinations, for all tasks, match those of a truth table for a 2-to-1 MUX and a 4-to-1 MUX.

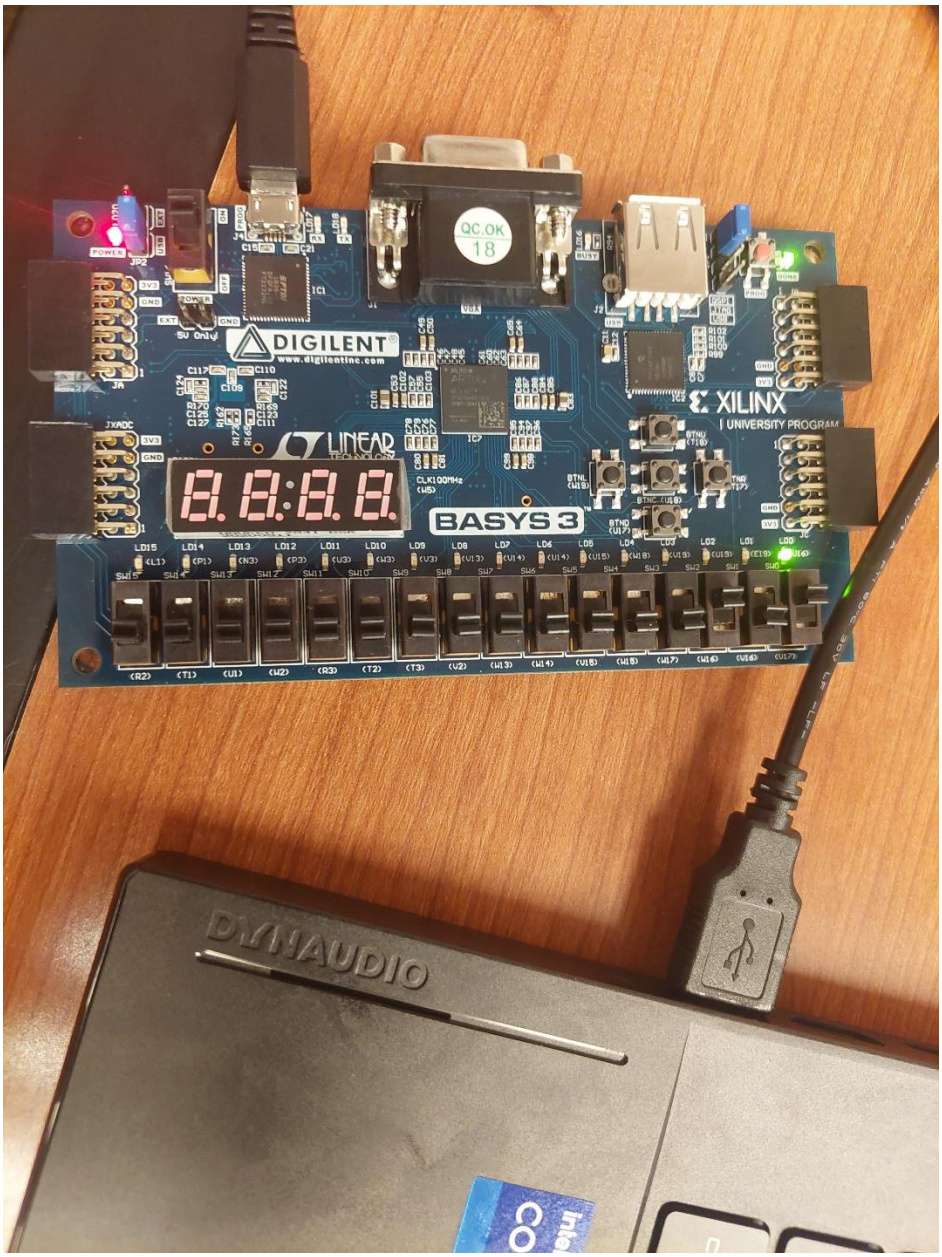
2-to-1 MUX Truth Table (part of one)

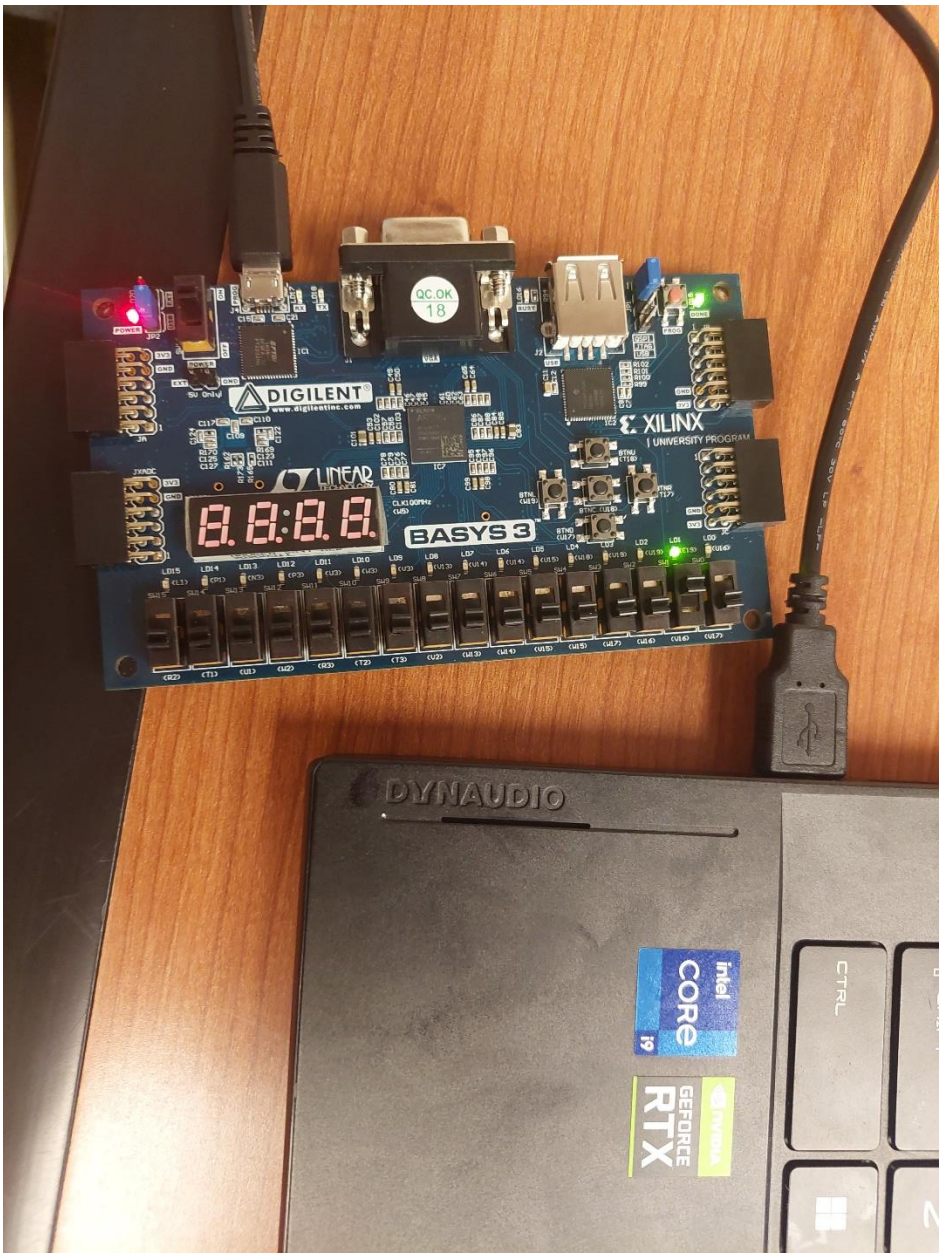
A	B	S	g
0	1	1	1
1	0	0	1

4-to-1 MUX Truth Table (part of one)

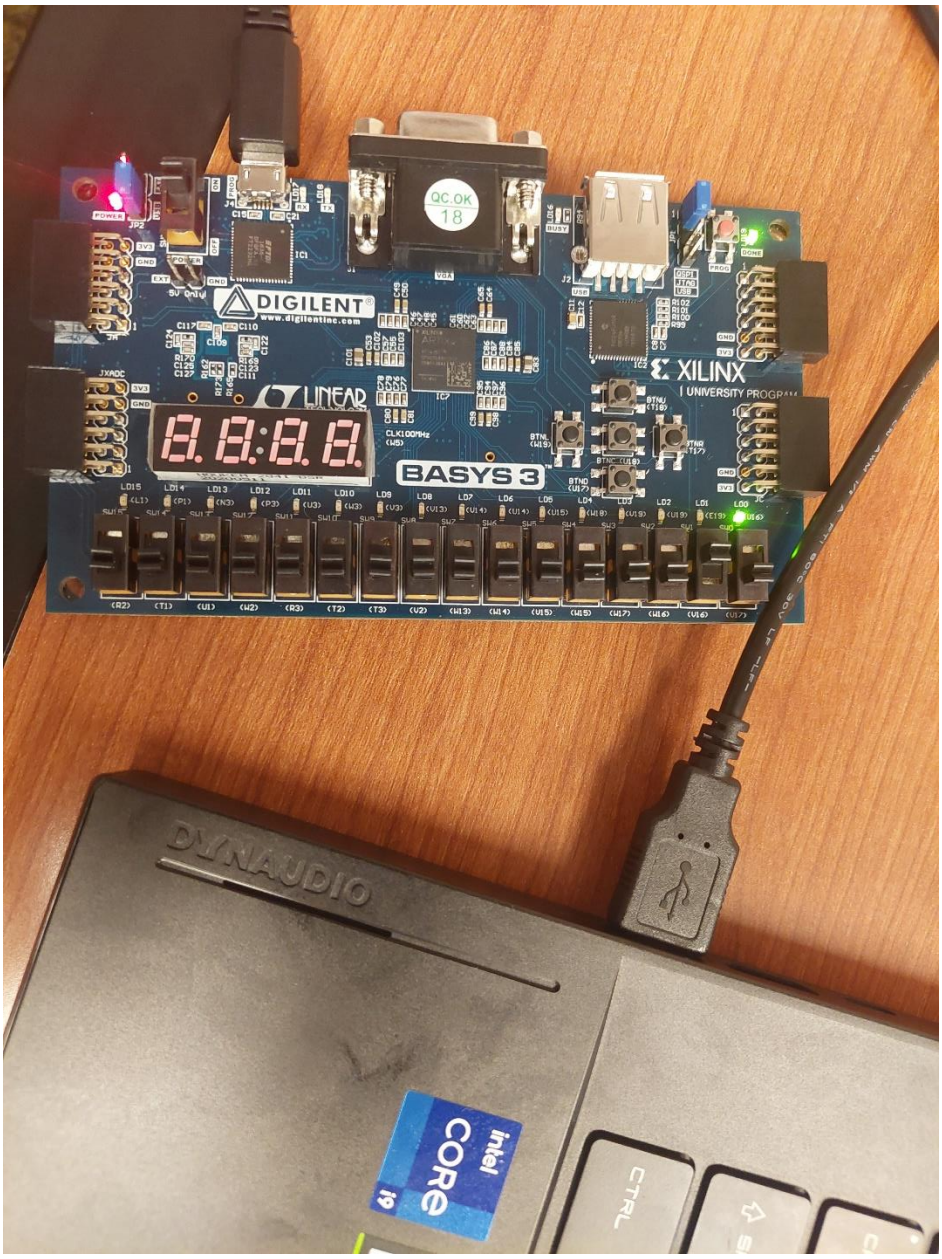
A	B	C	D	S0	S1	Y
0	1	1	0	1	1	1
1	1	0	0	0	1	1
1	0	0	0	0	1	1
1	0	1	0	0	0	1

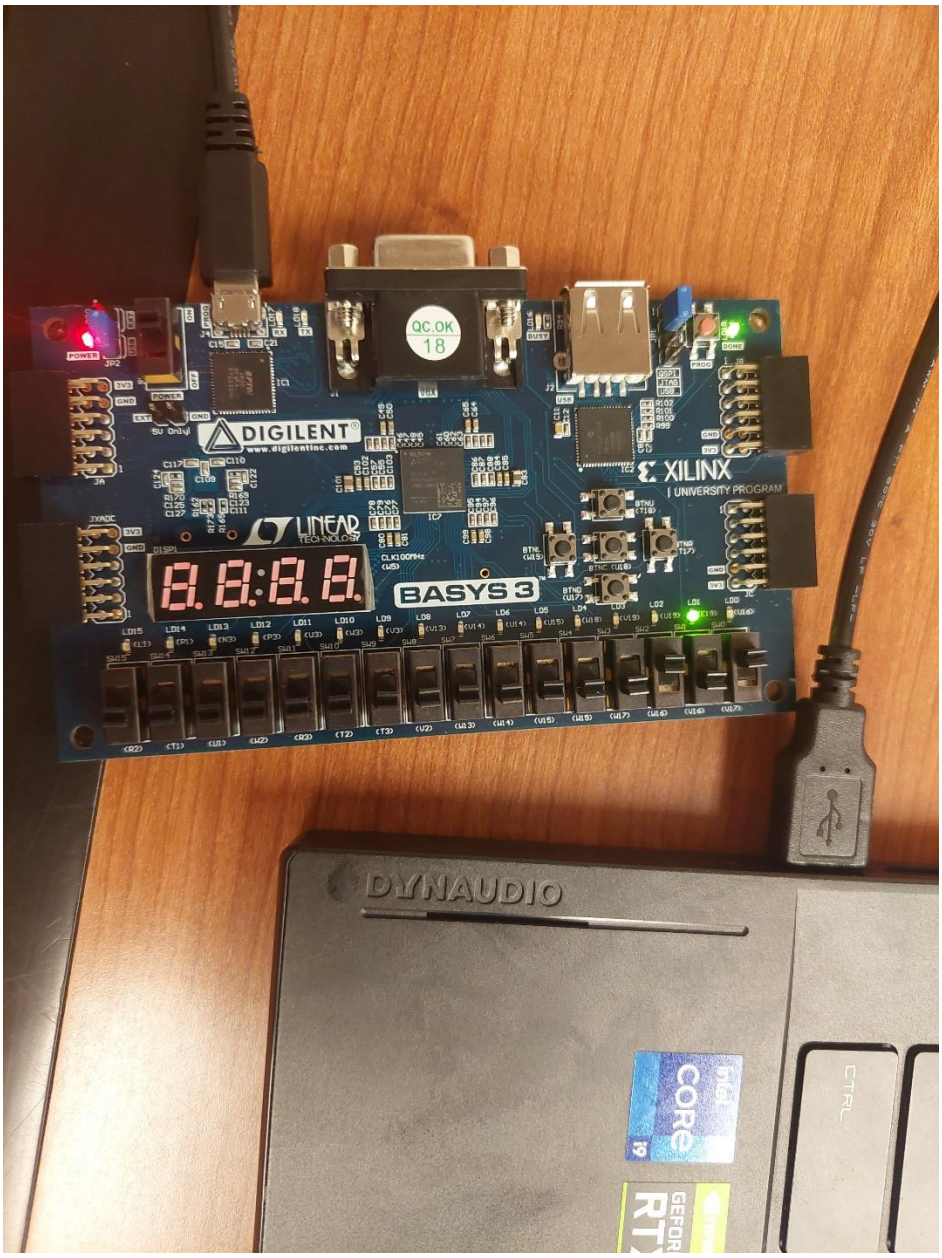
Task 1 FPGA pictures:


QC.OK
18
DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3
DYNAUDIO

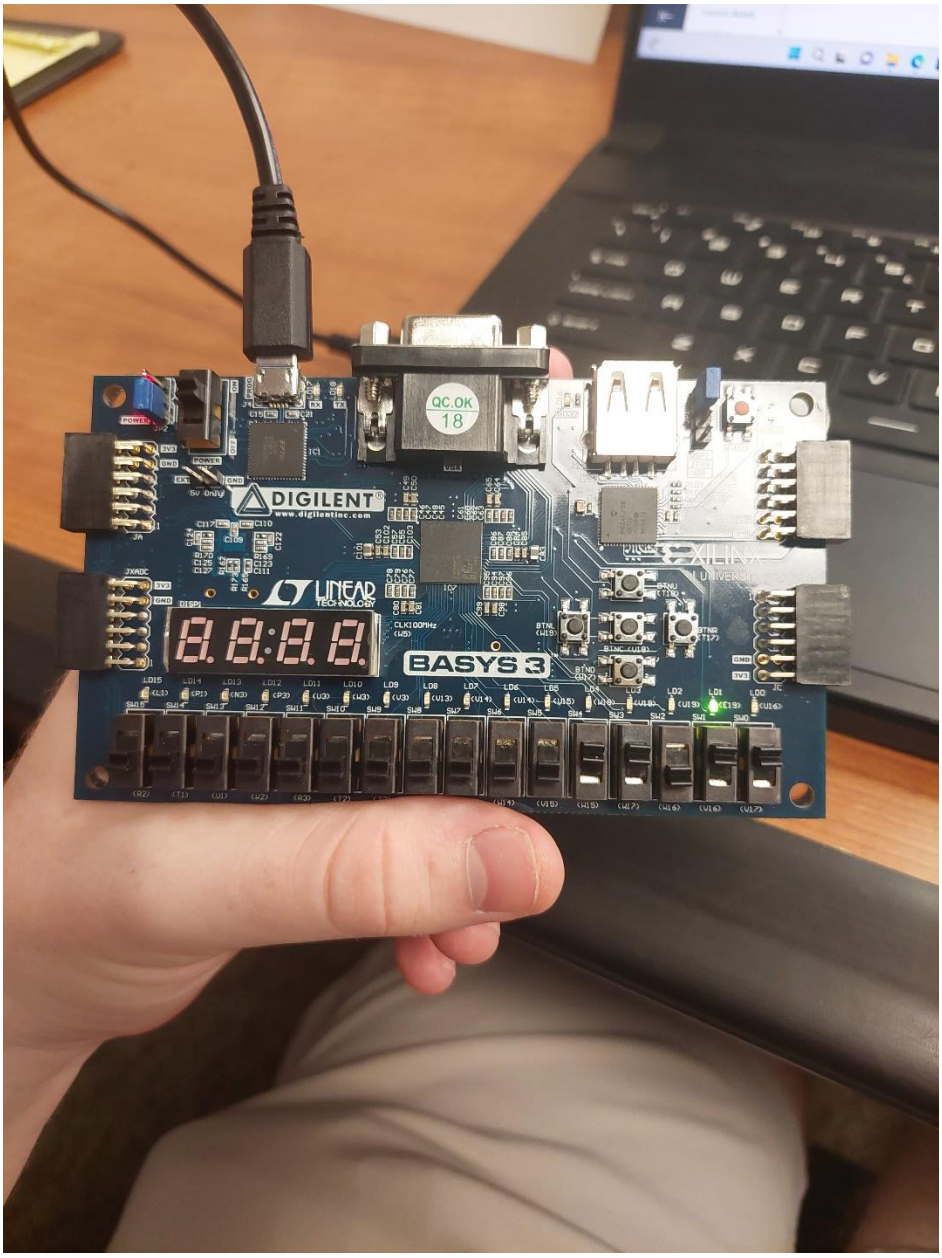

DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR
BASYS 3
DYNAUDIO
intel
CORE
i9
GEFORCE
RTX
CTRL

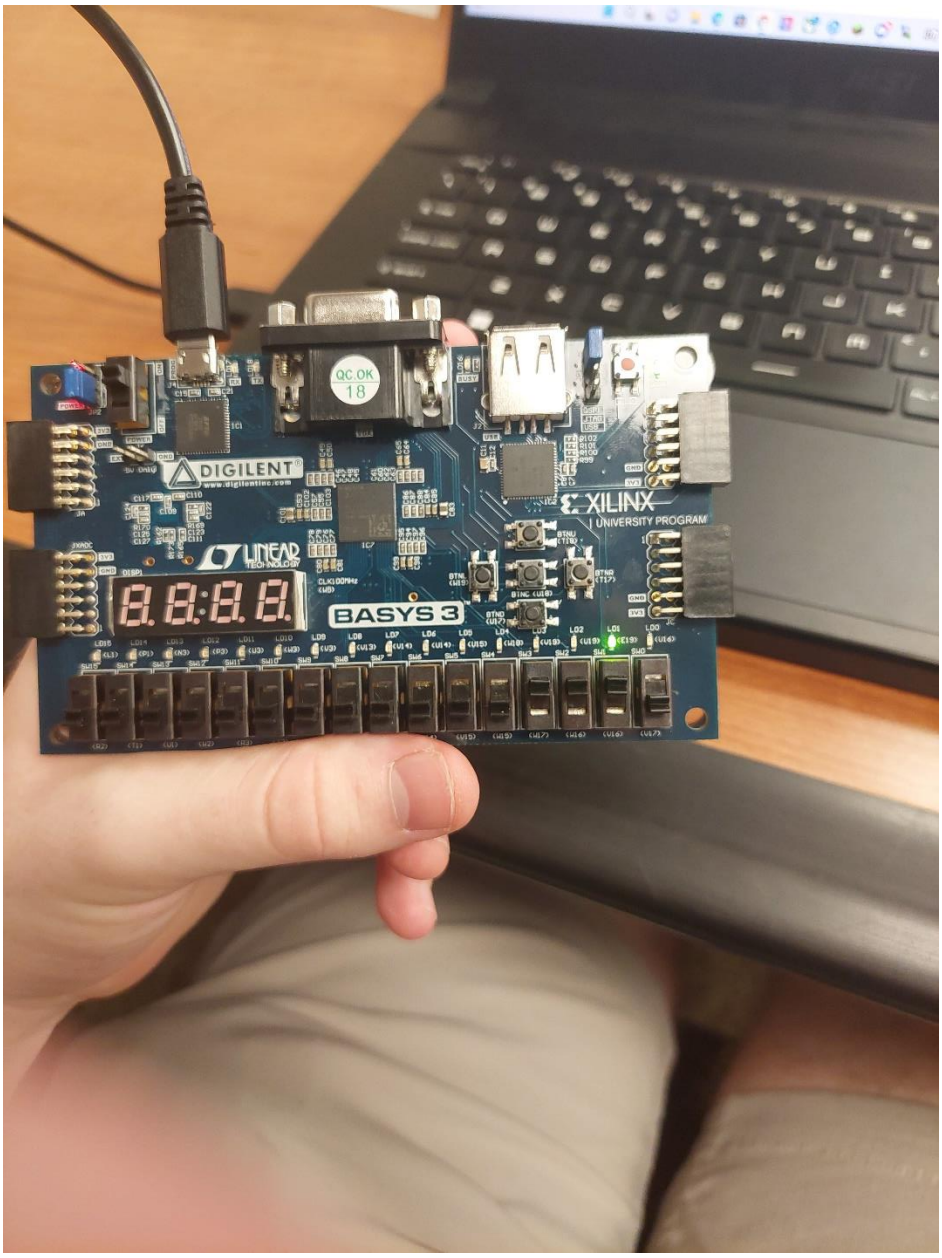
Task 2 FPGA pictures:

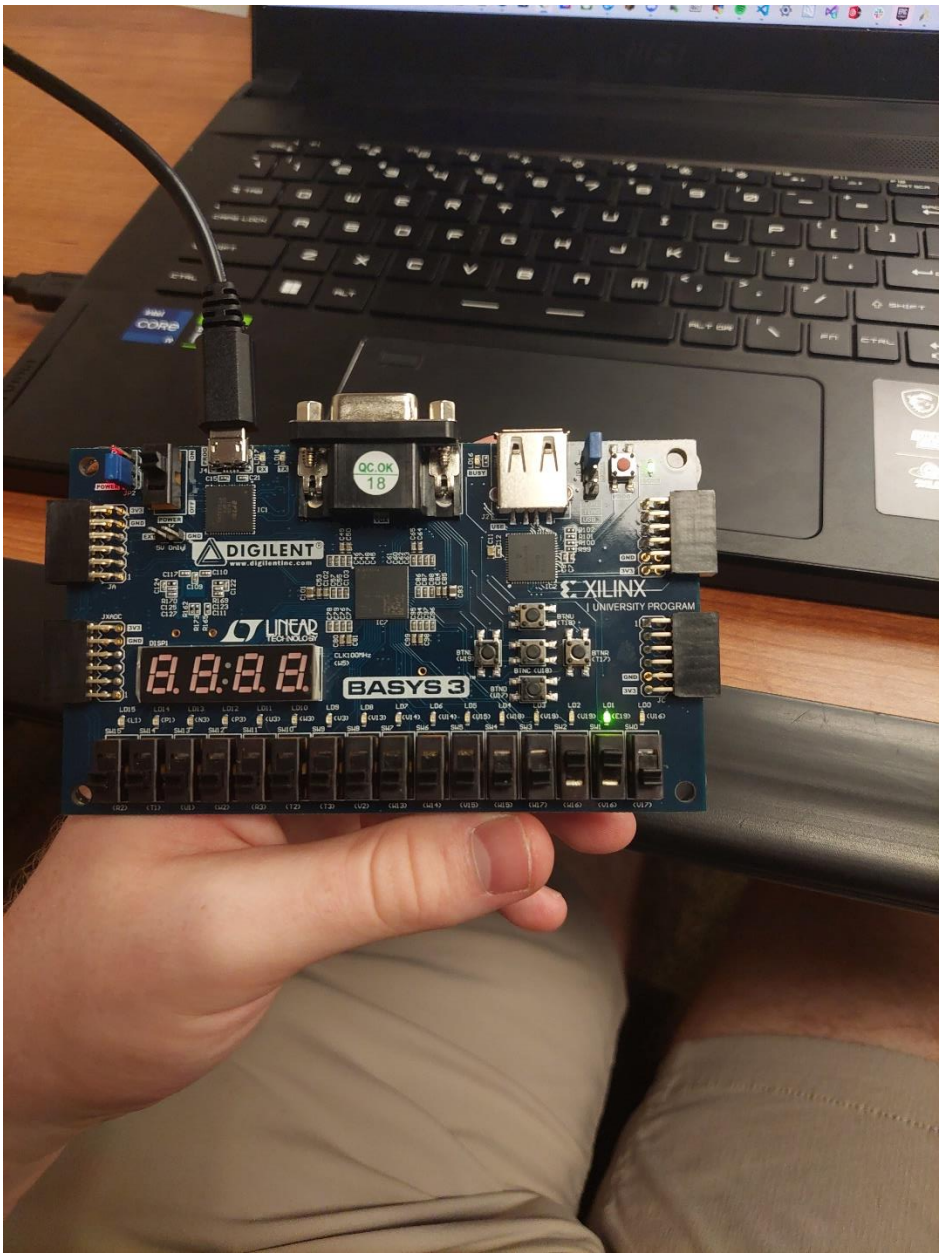


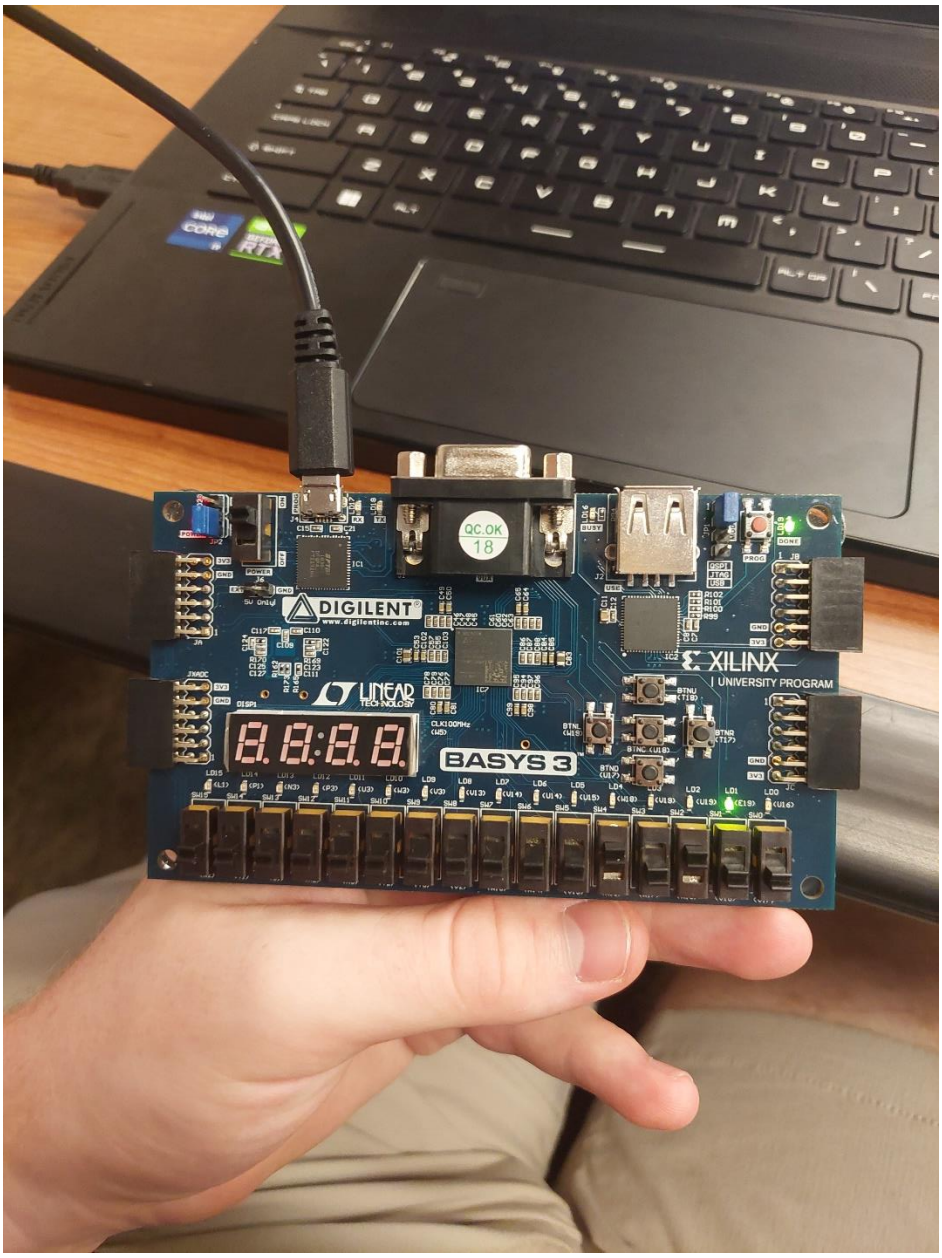

QC.OK
18
DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3
DYNAUDIO
CTRL

Task 3 FPGA pictures:




QC.OK
18
DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR
TECHNOLOGY
BASYS 3


QC.OK
18
DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3


DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3
QC.OK
18

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. In this case, Y (3-bit) was added in Task 2. I learned that the simulation takes 60ns before running a new simulation.

Appendix

Task 2 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 lab3task2file is
   Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          S : in STD_LOGIC;
          g : out STD_LOGIC);
end lab3task2file;

architecture Behavioral of lab3task2file is
   signal  temp : STD_LOGIC; --unused

begin
   g <= ((not S) and A) or (S and B);
end Behavioral;
```

Task 3 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 mux4to1 is
   Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          C : in STD_LOGIC;
          D : in STD_LOGIC;
          S0 : in STD_LOGIC;
          S1 : in STD_LOGIC;
          Y : out STD_LOGIC);
end mux4to1;

architecture Behavioral of mux4to1 is

component mux2to1 is
   Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          S : in STD_LOGIC;
          g : out STD_LOGIC);
end component;

signal t1, t2 : STD_LOGIC;

begin

U1: mux2to1 port map (A, B, S0, t2);
U2: mux2to1 port map (C, D, S0, t1);
U3: mux2to1 port map (t1, t2, S1, Y);

end Behavioral;
```

Extra credit:

Using the provided test bench file in the downloaded zip file, I was able to run a simulation of the signals for the 2-to-1 MUX in Task 2. The entity name in the test bench has to align with the name of the file that we're currently in. These waveforms represent the input (pins A, B, and S) and the output pin, g. If A is 0, B is 1, and S is 1, then the output, g, is 1. If A is 1, B is 0, and S is 0, then the output, g, is 1. This Boolean logic matches up with the waveforms shown below.