Data types, operators, combinational logic

Calvin Reese
cjreese@fortlewis.edu

2/10/22

1 Introduction

This HW, we worked on refreshing our binary logic and uploading/testing some example codes to an FPGA.

2 Materials and Methods

The tutorial for making these examples are in `http://www.yilectronics.com/Courses/CE433/s2022/lectures/week2_dataTypes/week2_dataTypes.html`

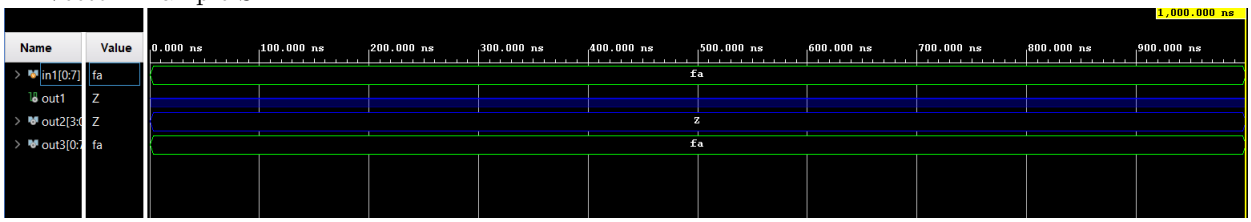
3 Results

3.1 Task 3

Vector Example Code

```
1 `timescale 1ns / 1ps
2 //////////////////////////////////////////////////
3
4 module vector_defn(num1,res1,res2,res3);
5 input[7:0] num1;
6 output res1;
7 output [3:0] res2;
8 output [0:7] res3;
9
10 assign res1=num1[2];
11 assign res2=num1[7:4];
12 assign res3=num1;
13 endmodule
14
15 module vecor_defn_tb;
16 reg[0:7] in1;
17
18 wire out1;
19 wire[3:0] out2;
20 wire[0:7] out3;
21
22 vector_defn UUT(.num1(in1),.res1(out2),.res3(out3));
23
24 initial begin
25         in1 = 8'hFA;
26         #100;
27 end
28 endmodule
29
30
```

Vector Example Sim



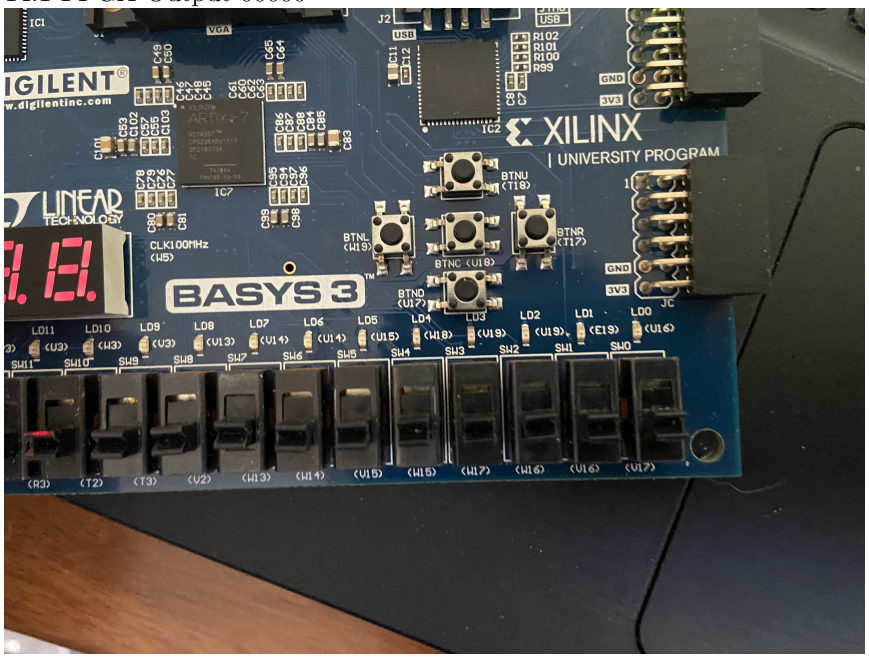
3.2 Task 4

14.1 Code

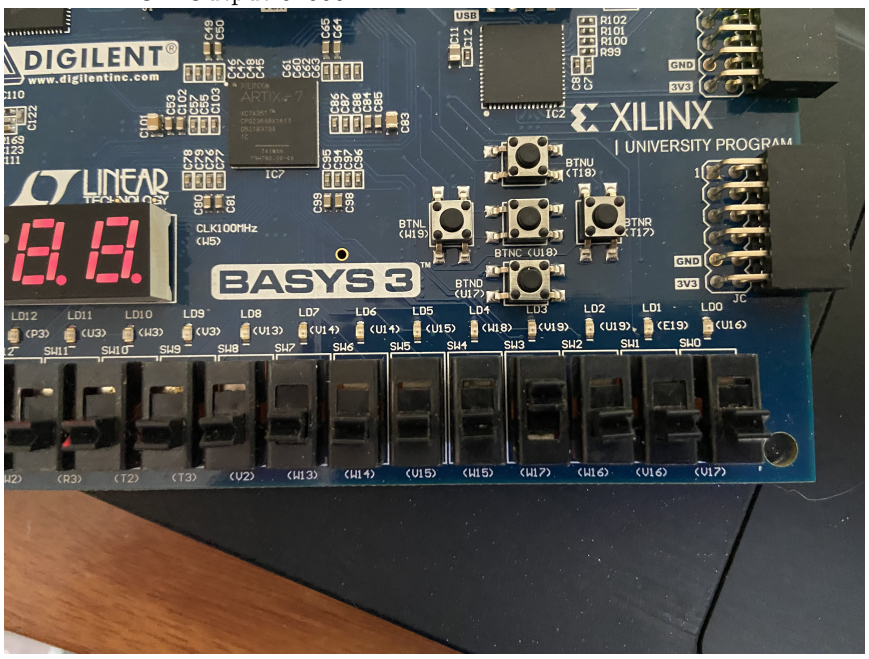
```
timescale 1ns / 1ps
//////////////////////////////////////////////////
module homeAlarm_top(sw,led);
input [4:0] sw;
output [0:0] led;
homeAlarm UUT(.a(led),.s(sw[3:0]),.m(sw[4]));
endmodule
```

```
timescale 1ns / 1ps
//////////////////////////////////////////////////
module homeAlarm(a,s,m);
input m;
input[3:0]s;
output a;
assign a=(s[0]|s[1]|s[2]|s[3])&m;
endmodule
```

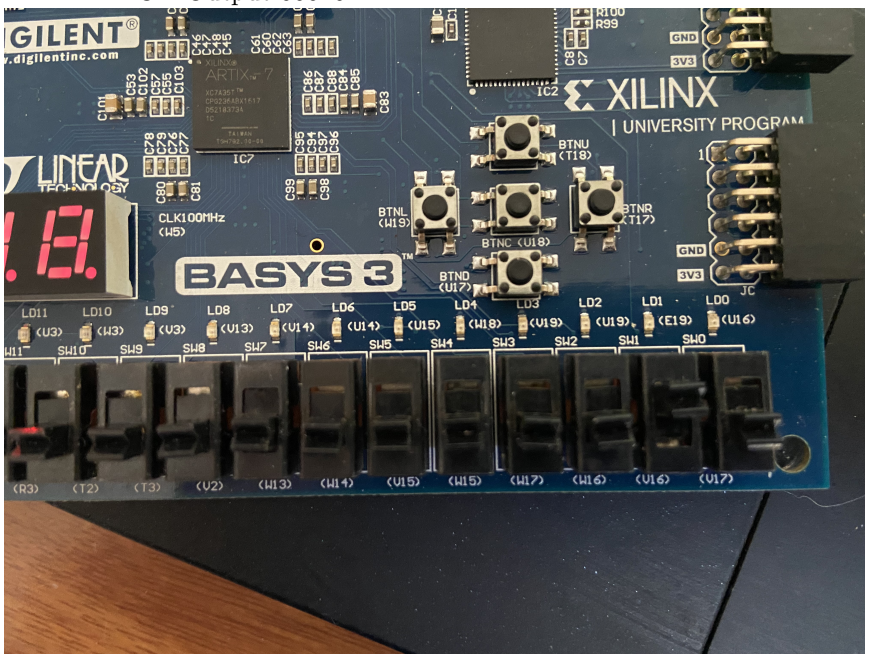
14.1 FPGA Output 00000



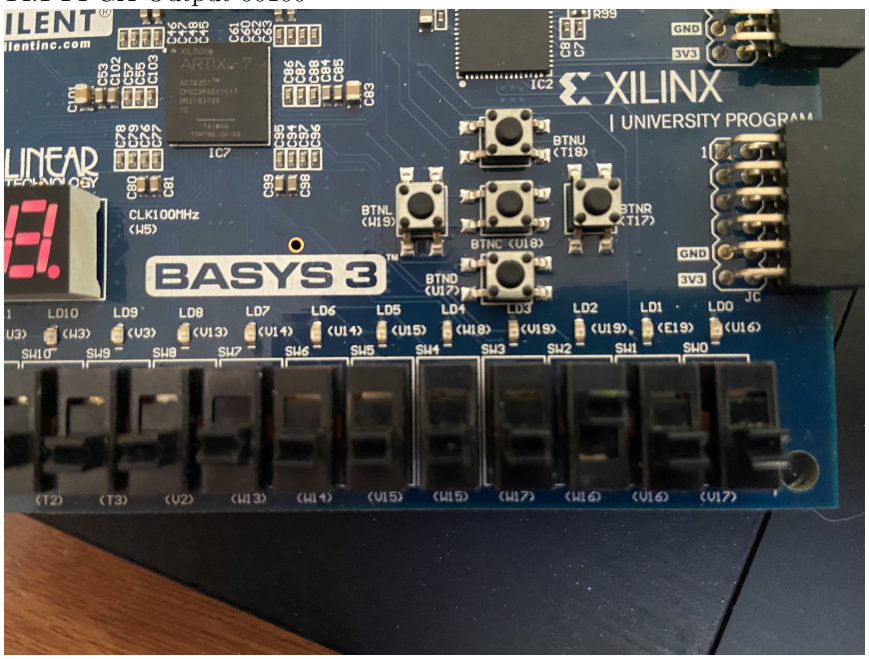
14.1 FPGA Output 01000



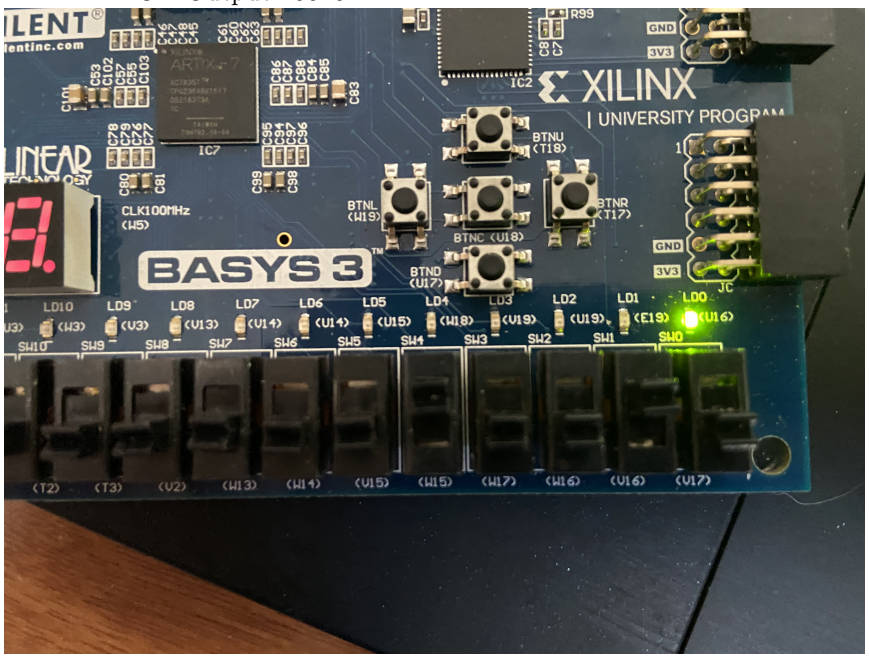
14.1 FPGA Output 00010



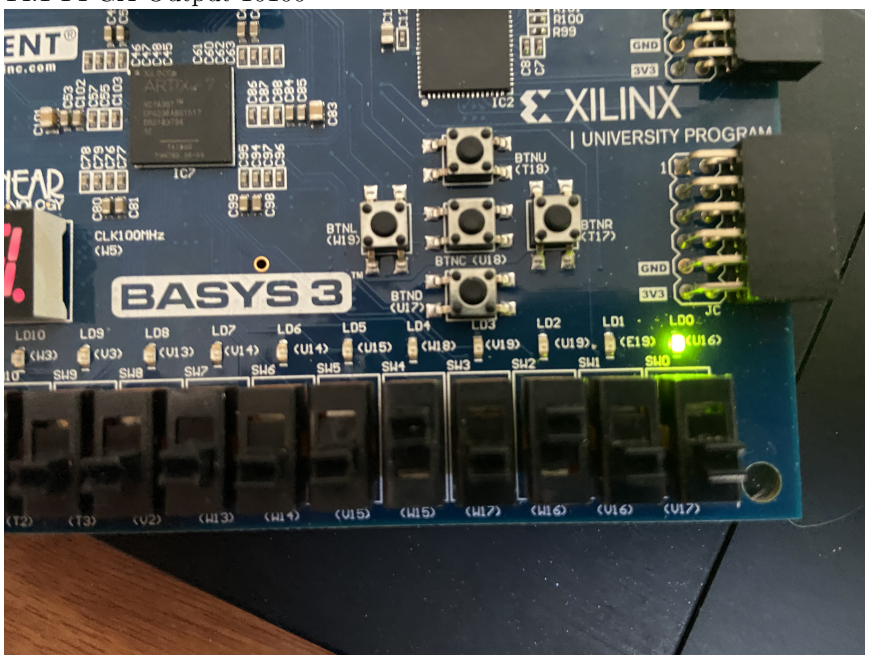
14.1 FPGA Output 00100



14.1 FPGA Output 10010



14.1 FPGA Output 10100

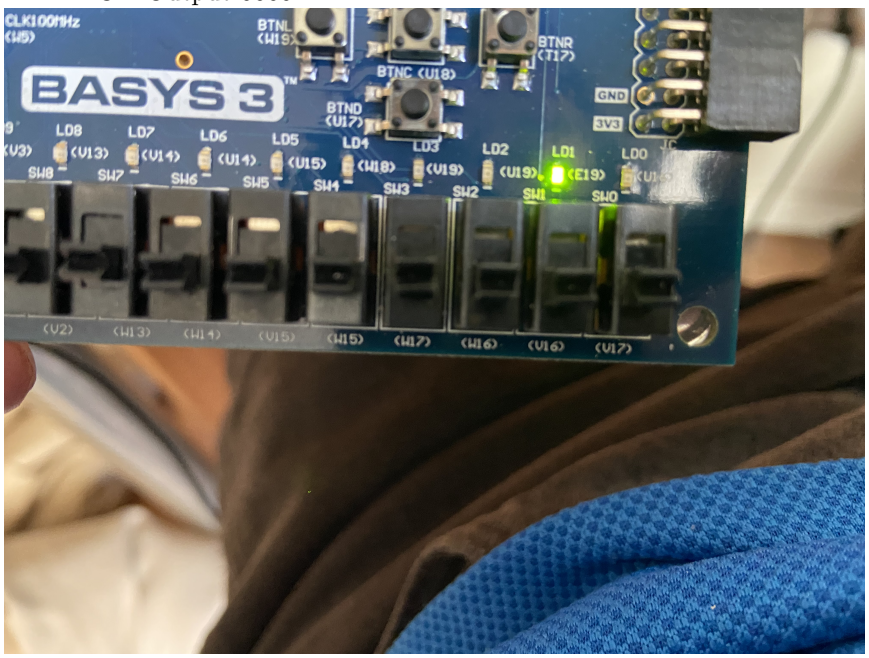


14.2 Code

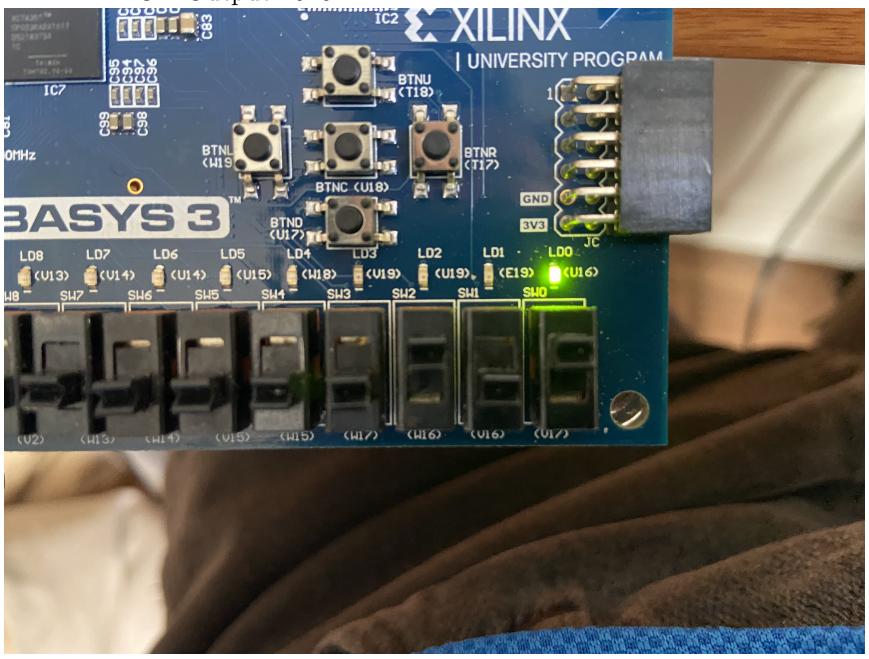
```
1 `timescale 1ns/1ps
2
3 module digitalSafe_top(sw,led);
4 input [3:0] sw;
5 output [1:0] led;
6 digitalSafe UUT(.l(led),.s(sw));
7 endmodule
~
~
```

```
1 `timescale 1ns/1ps
2
3 module digitalSafe(l,s);
4 input[3:0] s;
5 output [1:0] l;
6 parameter p=4'b0101;
7 assign l[0] = ~(s[0]^p[0]) & ~(s[1]^p[1]) & ~(s[2]^p[2]) & ~(s[3]^p[3]);
8 assign l[1] = ~l[0];
9 endmodule
```

14.2 FPGA Output 0000



14.2 FPGA Output 1010

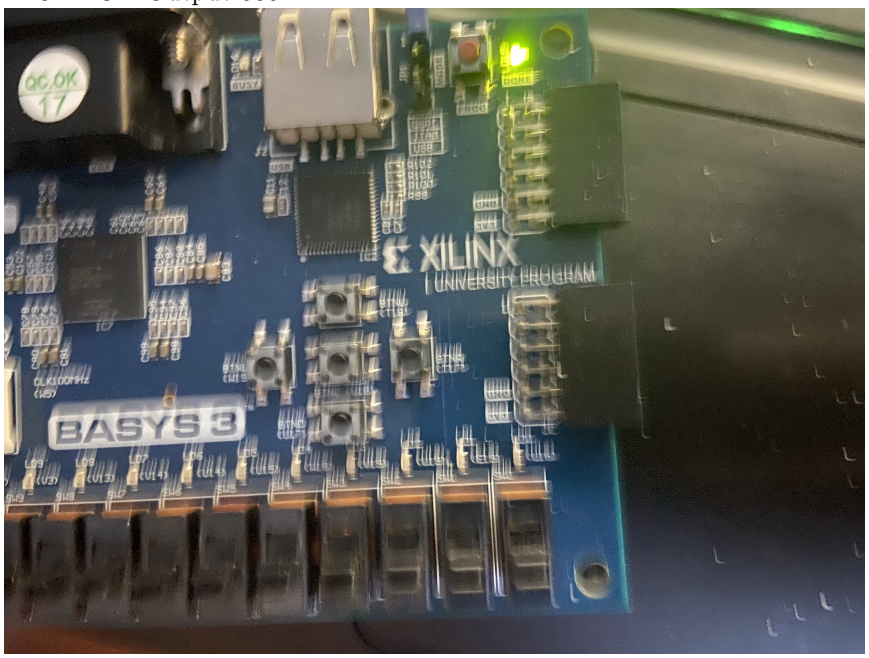


14.3 Code

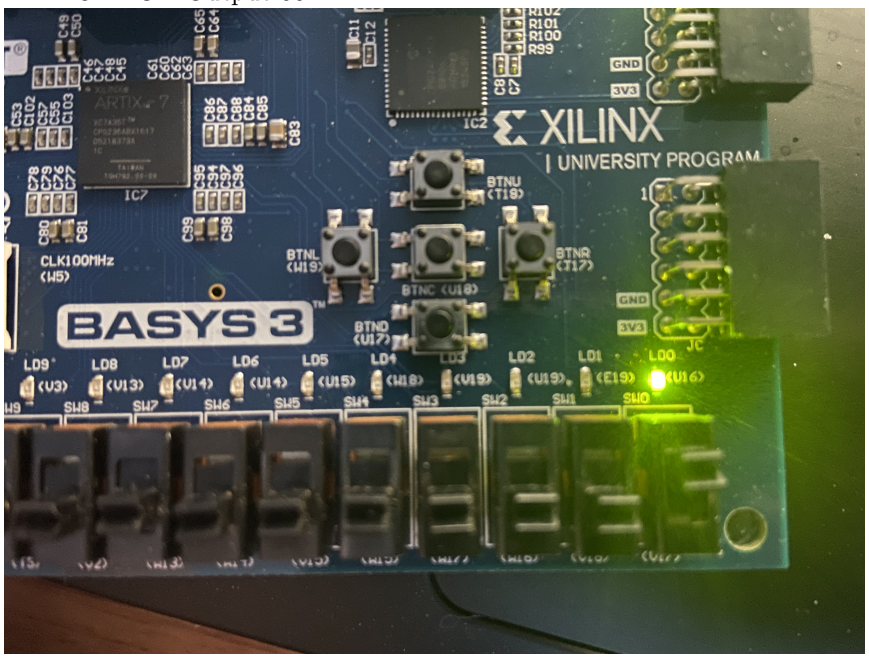
```
timescale 1ns/1ps
////////////////////////////////
module parkingCount_top(sw,led);
input[2:0] sw;
output [1:0] led;
parkingCount UUT(.c(led),.s(sw));
endmodule
```

```
timescale 1ns/1ps
////////////////////////////////
module parkingCount(c,s);
input [2:0]s;
output[1:0]c;
assign c[0]=(~s[0]&~s[1]&s[2])+(~s[0]&s[1]&~s[2])+(s[0]&~s[1]&~s[2])+(s[0]&s[
assign c[1] = (~s[0]&s[1]&s[2])+(s[0]&~s[1]&s[2])+(s[0]&s[1]&~s[2])+(s[0]&s[1
endmodule
```

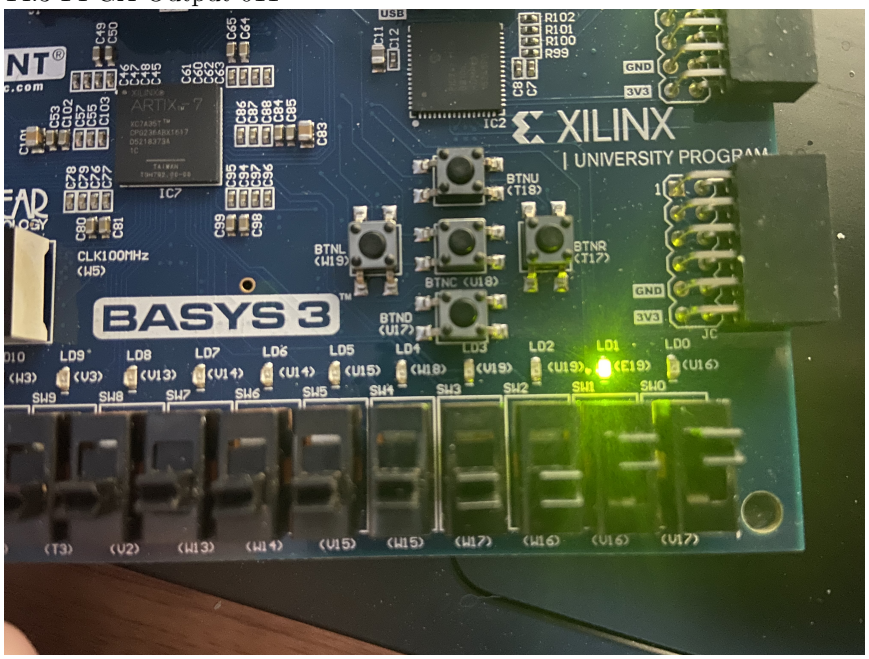
14.3 FPGA Output 000



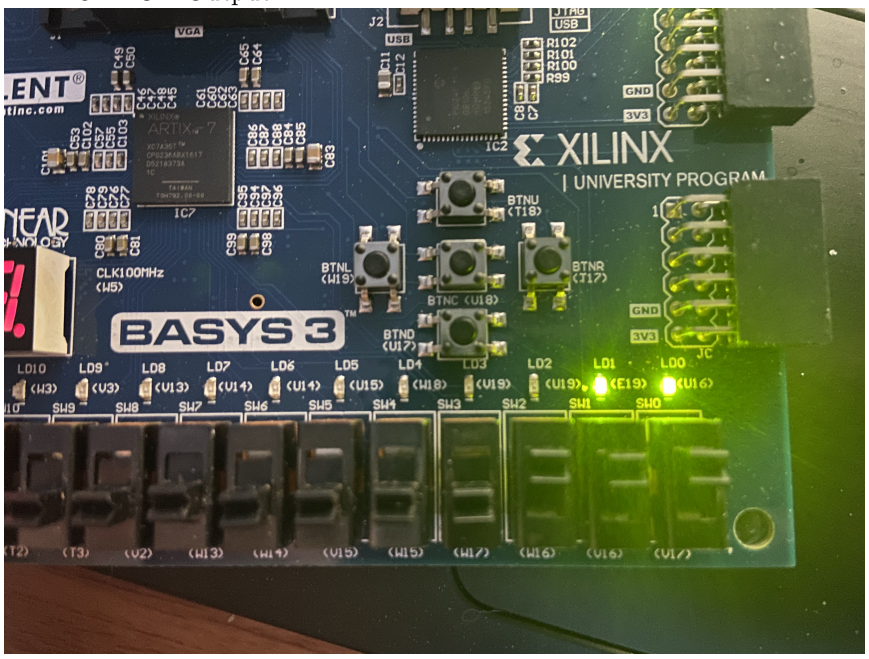
14.3 FPGA Output 001



14.3 FPGA Output 011



14.3 FPGA Output 111



4 Discussion

All of this was fairly simple just to follow your tutorial. I now understand the difference between [0:7] and [7:0] and I am feeling confident in my ability. 41.1 would light up if bit 5 and any other bit was high. 14.2 would have led 2 lit if it was anything besides 1010. 14.3 could give the bit sequence of how many of the 3 switches were high regardless of the sequence.

Task 1

a) 0000 0000 0001 0100, 0100 0000 0000 0000

0000 0000 1000 0000. 1000 0000 0000 0000

0000 0000 0000 0000 . 0010 0000 0000 0000

1000 0000 0010 0110 . 0010 0000 0000 0000

1000 0000 0011 0010 . 1010 0000 0000 0000

b)

0 01100 0010000011

0 10000 110 1000000

1 1001 0 1110100000

Task 2

0000 1111,0100 + 0000 0100,0010 = 0001 0011,0110

19,375

0000 1111,0100 + 1111 1011,1110 = 1 0000 1011,0010

11,125