ASEN 3300, Fall 2022: Lab 9 Report Submission

Names:	Jared Steffen
	Brady Sivey
	Joshua Geeting
Section:	012

Experiment (25 points)

5.1 Combinatorial Logic

a.

i. Record your response to 5.1.a.i below:

When pin #8 is connected to ground, the light was turned off once the program was done uploading. When pin #8 is float, the LED turns off as well. When pin #8 is 3.3 V, the LED stays on.

ii. Record your response to 5.1.a.ii below:

When pin #9 is float, the LED stays on. When connected to ground, the LED turns off. When connected to 3.3V, the LED stays on.
When pin #10 is float, the LED stays on. When connected to ground, the LED turns off. When connected to 3.3V, the LED stays on.

iii. Record the changes to your code to meet the problem parts. (i.e. *S* = ____;)

1. Invert (logical) Pin 8:

```
// Define input and output pins
const int pinA = 8;       // Astronaut input
const int pinO = 9;       // Oxygen input
const int pinF = 10;      // Fire input
const int pinS = 12;      // Siren ouptut

// the setup function runs once when you press reset or power the board
void setup() {
  pinMode(pinS, OUTPUT);    // Define pinS (siren) as an output pin
  pinMode(pinA, INPUT);     // Define pinA (Astronaut sensor) as an input pin
//  pinMode(pinO, INPUT);     // Define pinO (Oxygen sensor) as an input pin
  pinMode(LED_BUILTIN, OUTPUT); // Define the on board LED as a output pin for testing
  digitalWrite(LED_BUILTIN, LOW); // Turn Off LED to start
}

// the loop function runs over and over again forever
void loop() {
  boolean pinA_State = digitalRead(pinA);  // Read the Astronaut input state
//  boolean pinO_State = digitalRead(pinO);  // Read the Astronaut input state
  boolean pinS_State = !pinA_State; // Siren state set to Astronaut input state

  digitalWrite(LED_BUILTIN, pinS_State);  // Write siren output to Arduino LED
  digitalWrite(pinS, pinS_State);         // Write siren output to pinS on the Arduino
}
```

2. 8 (AND) 9:

```
// Define input and output pins
const int pinA = 8;       // Astronaut input
const int pinO = 9;       // Oxygen input
const int pinF = 10;      // Fire input
const int pinS = 12;      // Siren ouptut

// the setup function runs once when you press reset or power the board
void setup() {
  pinMode(pinS, OUTPUT);    // Define pinS (siren) as an output pin
  pinMode(pinA, INPUT);     // Define pinA (Astronaut sensor) as an input pin
  pinMode(pinO, INPUT);     // Define pinO (Oxygen sensor) as an input pin
  pinMode(LED_BUILTIN, OUTPUT); // Define the on board LED as a output pin for testing
  digitalWrite(LED_BUILTIN, LOW); // Turn Off LED to start
}

// the loop function runs over and over again forever
void loop() {
  boolean pinA_State = digitalRead(pinA);  // Read the Astronaut input state
  boolean pinO_State = digitalRead(pinO);  // Read the Astronaut input state
  boolean pinS_State = pinA_State && pinO_State; // Siren state set to Astronaut input state

  digitalWrite(LED_BUILTIN, pinS_State);  // Write siren output to Arduino LED
  digitalWrite(pinS, pinS_State);         // Write siren output to pinS on the Arduino
}
```

3. 8 (OR) 9:

```
// Define input and output pins
const int pinA = 8;       // Astronaut input
const int pinO = 9;       // Oxygen input
const int pinF = 10;      // Fire input
const int pinS = 12;      // Siren ouptut

// the setup function runs once when you press reset or power the board
void setup() {
  pinMode(pinS, OUTPUT);    // Define pinS (siren) as an output pin
  pinMode(pinA, INPUT);     // Define pinA (Astronaut sensor) as an input pin
  pinMode(pinO, INPUT);     // Define pinO (Oxygen sensor) as an input pin
  pinMode(LED_BUILTIN, OUTPUT); // Define the on board LED as a output pin for testing
  digitalWrite(LED_BUILTIN, LOW); // Turn Off LED to start
}

// the loop function runs over and over again forever
void loop() {
  boolean pinA_State = digitalRead(pinA);  // Read the Astronaut input state
  boolean pinO_State = digitalRead(pinO);  // Read the Astronaut input state
  boolean pinS_State = pinA_State || pinO_State; // Siren state set to Astronaut input state

  digitalWrite(LED_BUILTIN, pinS_State);  // Write siren output to Arduino LED
  digitalWrite(pinS, pinS_State);         // Write siren output to pinS on the Arduino
}
```

4. 8 (XOR) 9:

```
// Define input and output pins
const int pinA = 8;       // Astronaut input
const int pinO = 9;       // Oxygen input
const int pinF = 10;      // Fire input
const int pinS = 12;      // Siren ouptut

// the setup function runs once when you press reset or power the board
void setup() {
  pinMode(pinS, OUTPUT);    // Define pinS (siren) as an output pin
  pinMode(pinA, INPUT);     // Define pinA (Astronaut sensor) as an input pin
  pinMode(pinO, INPUT);     // Define pinO (Oxygen sensor) as an input pin
  pinMode(LED_BUILTIN, OUTPUT); // Define the on board LED as a output pin for testing
  digitalWrite(LED_BUILTIN, LOW); // Turn Off LED to start
}

// the loop function runs over and over again forever
void loop() {
  boolean pinA_State = digitalRead(pinA);  // Read the Astronaut input state
  boolean pinO_State = digitalRead(pinO);  // Read the Astronaut input state
  boolean pinS_State = pinA_State ^= pinO_State; // Siren state set to Astronaut input state

  digitalWrite(LED_BUILTIN, pinS_State);  // Write siren output to Arduino LED
  digitalWrite(pinS, pinS_State);         // Write siren output to pinS on the Arduino
}
```

5. 8 (NAND) 9:

```
// Define input and output pins
const int pinA = 8;       // Astronaut input
const int pinO = 9;       // Oxygen input
const int pinF = 10;      // Fire input
const int pinS = 12;      // Siren ouptut

// the setup function runs once when you press reset or power the board
void setup() {
  pinMode(pinS, OUTPUT);    // Define pinS (siren) as an output pin
  pinMode(pinA, INPUT);     // Define pinA (Astronaut sensor) as an input pin
  pinMode(pinO, INPUT);     // Define pinO (Oxygen sensor) as an input pin
  pinMode(LED_BUILTIN, OUTPUT); // Define the on board LED as a output pin for testing
  digitalWrite(LED_BUILTIN, LOW); // Turn Off LED to start
}

// the loop function runs over and over again forever
void loop() {
  boolean pinA_State = digitalRead(pinA);  // Read the Astronaut input state
  boolean pinO_State = digitalRead(pinO);  // Read the Astronaut input state
  boolean pinS_State = !pinA_State && pinO_State or pinA_State && !pinO_State or !pinA_State && !pinO_State; //

  digitalWrite(LED_BUILTIN, pinS_State);  // Write siren output to Arduino LED
  digitalWrite(pinS, pinS_State);         // Write siren output to pinS on the Arduino
}
```

6. The given logic diagram in lab document:

Simplifies to a XOR

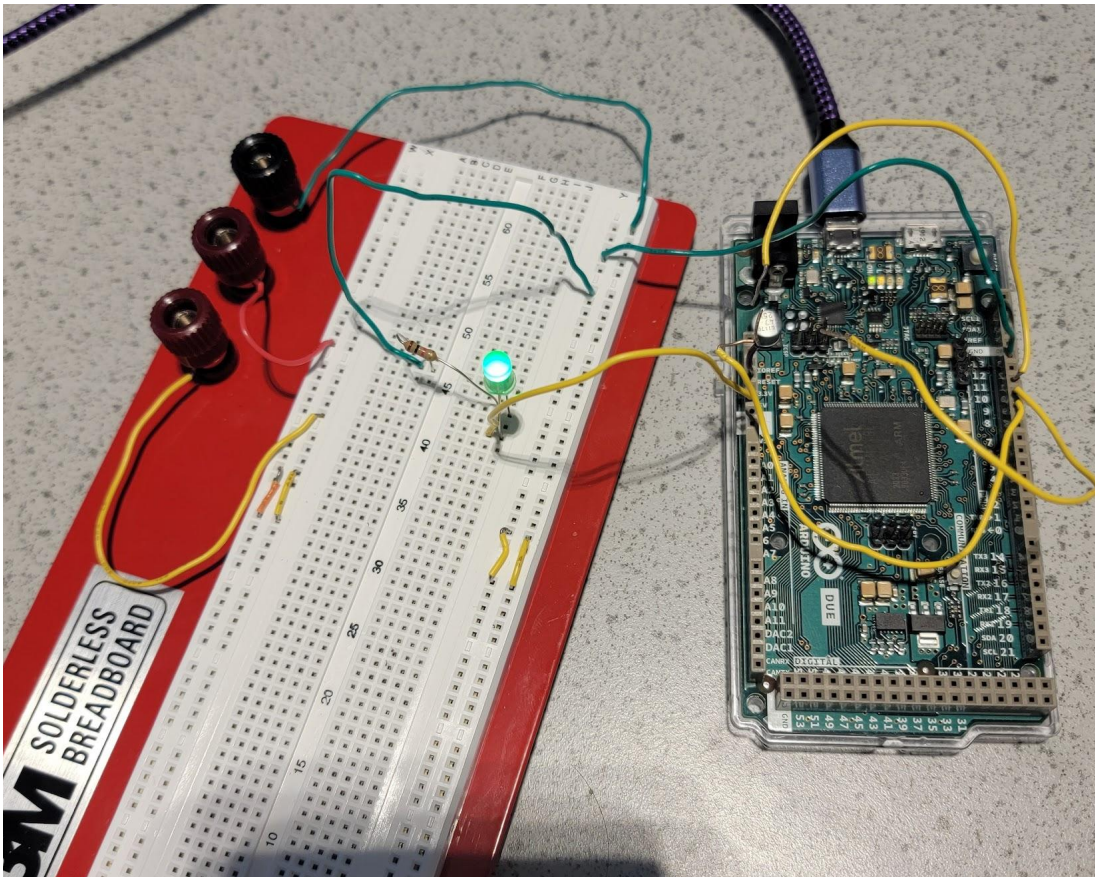
```
// Define input and output pins
const int pinA = 8;       // Astronaut input
const int pinO = 9;       // Oxygen input
const int pinF = 10;      // Fire input
const int pinS = 12;      // Siren ouptut

// the setup function runs once when you press reset or power the board
void setup() {
  pinMode(pinS, OUTPUT);    // Define pinS (siren) as an output pin
  pinMode(pinA, INPUT);     // Define pinA (Astronaut sensor) as an input pin
  pinMode(pinO, INPUT);     // Define pinO (Oxygen sensor) as an input pin
  pinMode(LED_BUILTIN, OUTPUT); // Define the on board LED as a output pin for testing
  digitalWrite(LED_BUILTIN, LOW); // Turn Off LED to start
}

// the loop function runs over and over again forever
void loop() {
  boolean pinA_State = digitalRead(pinA);  // Read the Astronaut input state
  boolean pinO_State = digitalRead(pinO);  // Read the Astronaut input state
  boolean pinS_State = pinA_State ^= pinO_State ; // Siren state set to Astronaut input state

  digitalWrite(LED_BUILTIN, pinS_State);  // Write siren output to Arduino LED
  digitalWrite(pinS, pinS_State);         // Write siren output to pinS on the Arduino
}
```

iv. **Record your response to 5.1.a.iv below:**



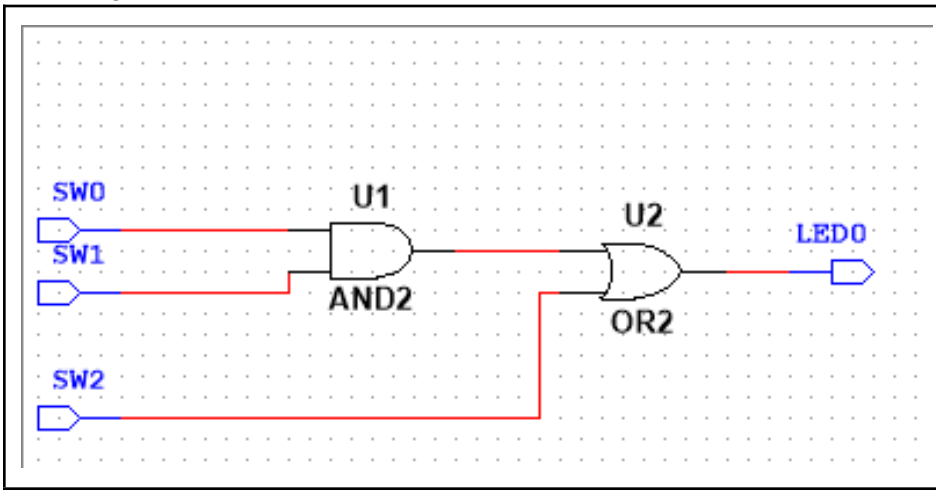
v. **Record your response to 5.1.a.v below:**

A	O	F	S (output 0 or 1)
0	**0**	**0**	0
0	**0**	**1**	1
0	**1**	**0**	0
0	**1**	**1**	1
1	**0**	**0**	0
1	**0**	**1**	1
1	**1**	**0**	1
1	**1**	**1**	1

5.2 Multisim Simulation - Digital Input

d.

viii. Record your response to 5.2.d.viii below:



f.

v. Record your response to 5.2.f.v below:

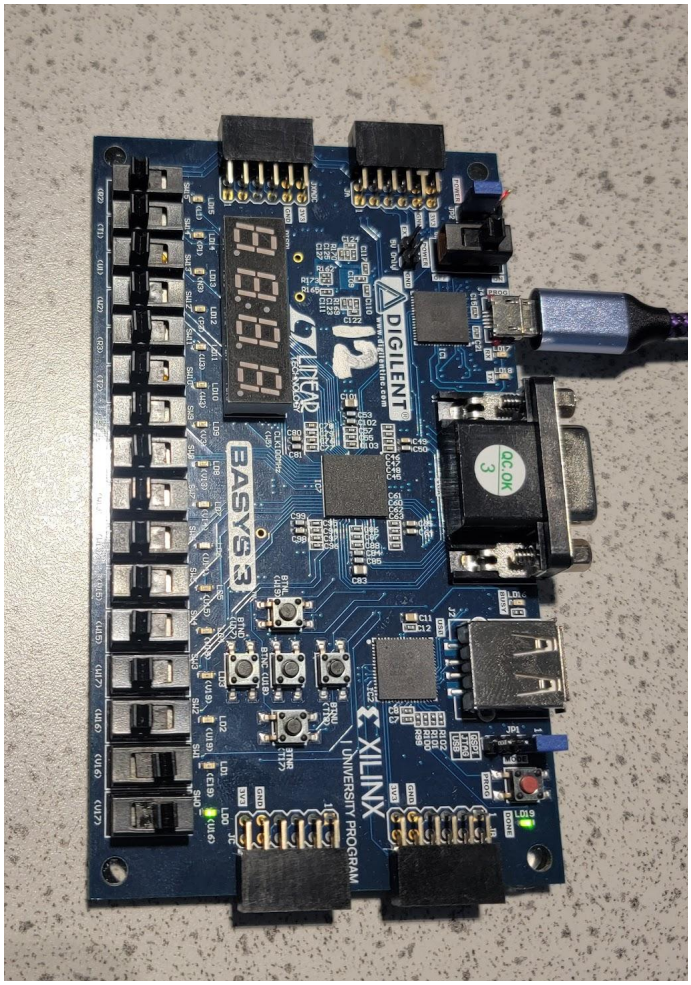
A	O	F	S (output 0 or 1)
0	**0**	**0**	0
0	**0**	**1**	1
0	**1**	**0**	0
0	**1**	**1**	1
1	**0**	**0**	0
1	**0**	**1**	1
1	**1**	**0.**	1
1	**1**	**1**	1

Yes, they do

vi.

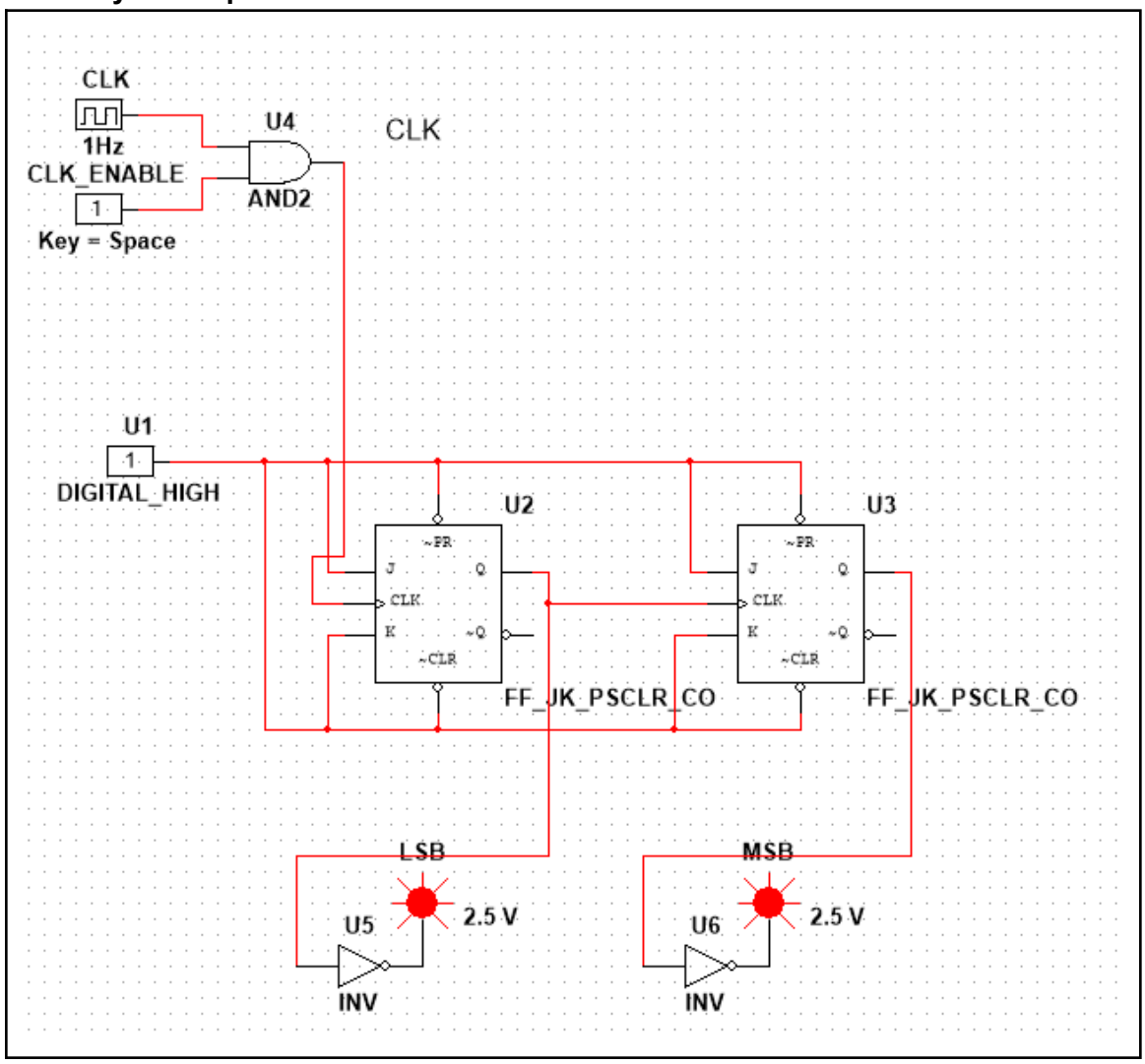
Fire Alarm Input	Linked Switch
A:	SW# 0
O:	SW# 1

F:	SW# 2



5.3 Multisim Simulation - Sequential Logic

a. Record your response to 5.3.a below:



b. 8-bit Counter

i. Record your response to 5.3.b.i below:

We observe the counter increase based on the clock frequency. When read from MSB to LSB, it goes from 00000001 to 00000010 to 00000011 and so on.

ii. Record your response to 5.3.b.ii below:

The 2 bit counter counts to 3 or 11 and then restarts while the 8 bit counter counts to 255 or 11111111. If the clock frequencies are the same for both, the lights turn on at the same frequency. High clock frequency leads to faster counting.

5.4 Arduino Due Frequency Counter

a. Record your response to 5.4.a below:

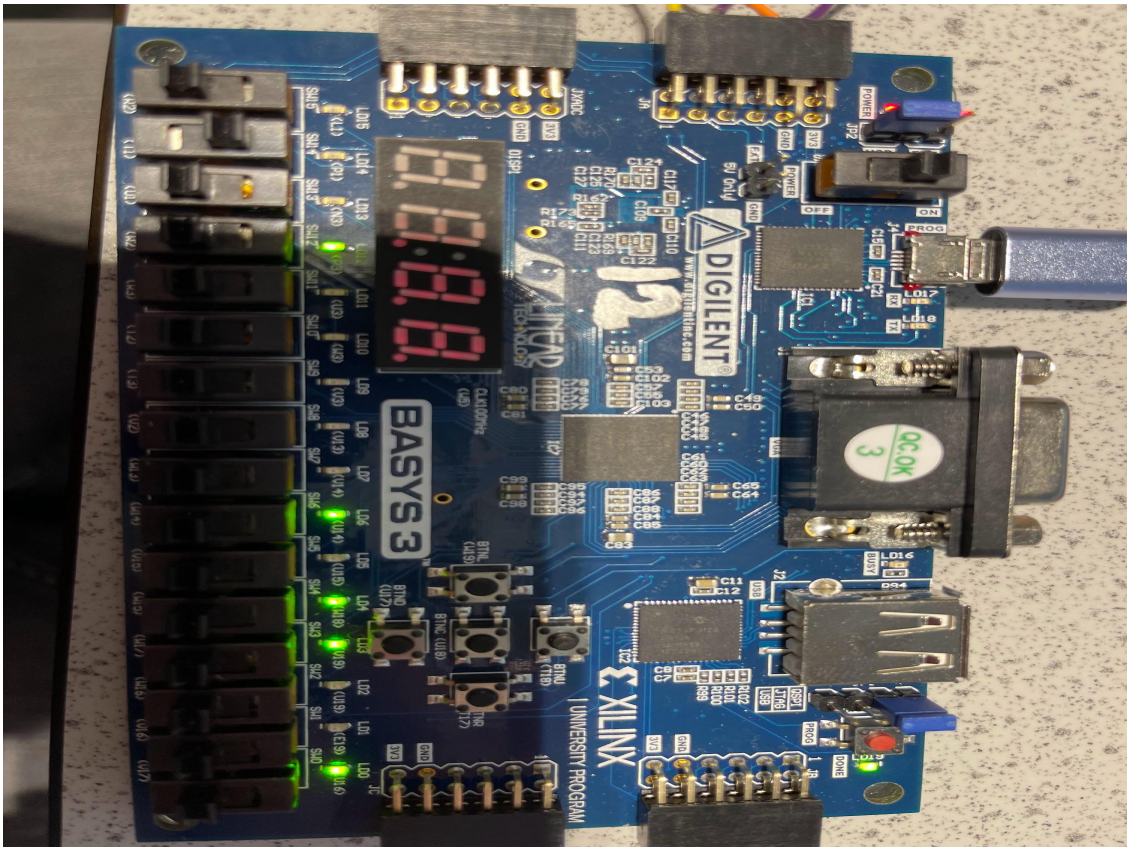
It stops working at >310 kHz

b. Record your response to 5.4.b below:.

When the function generator is set to 100000 Hz, the Arduino Serial Monitor output says 100002.6 Hz

5.5 FPGA Testing 16 Bit Counter

i. Record your response to 5.5.i below:



j. Record your responses to 5.5.j below:

Binary Count	00010000010110010000
Decimal Value of Count:	66960

k. Record your response to 5.5.k below:

Next number to pop up was 00001111011111010000. The value is different this time due to stopping the count at a different time.

l. Record your response to 5.5.l below:

It works up to 10 MHz, which is the maximum of the frequency generator. It is better than the Arduino because the Arduino capped out at 310 kHz. The response time for the Arduino is also much slower.

6. Analysis (25 points)

6.1 Combinatorial Logic

a. **Circuits**

i. **Record your responses to 6.1.a.i below:**

Switch A	Switch B	Lamp
0	0	0
0	1	1
1	0	1
1	1	1

OR

ii. **Record your responses to 6.1.a.ii below:**

Switch A	Switch B	Lamp
0	0	0
0	1	0
1	0	0
1	1	1

AND

b. **Logic circuits**

i. **Record your responses to 6.1.b.i below**

A	B	Output

0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

XOR

ii. **Record your responses to 6.1.b.ii below**

A	B	Output
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

XNOR

iii. **Record your responses to 6.1.b.iii below**

A	B	Output
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

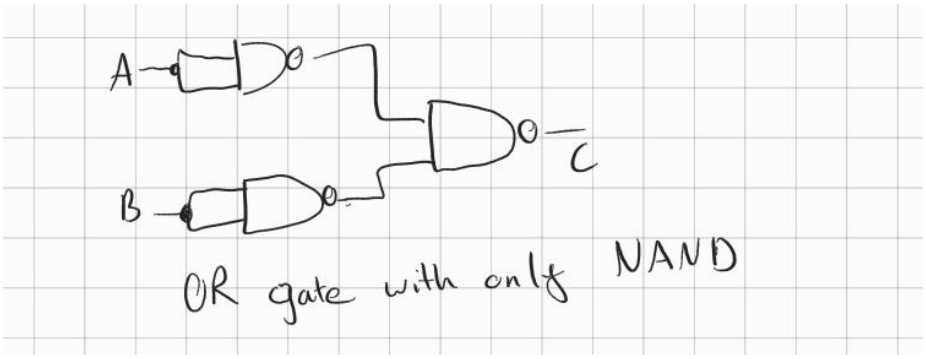
XNOR

iv. **Record your responses to 6.1.b.iv below**

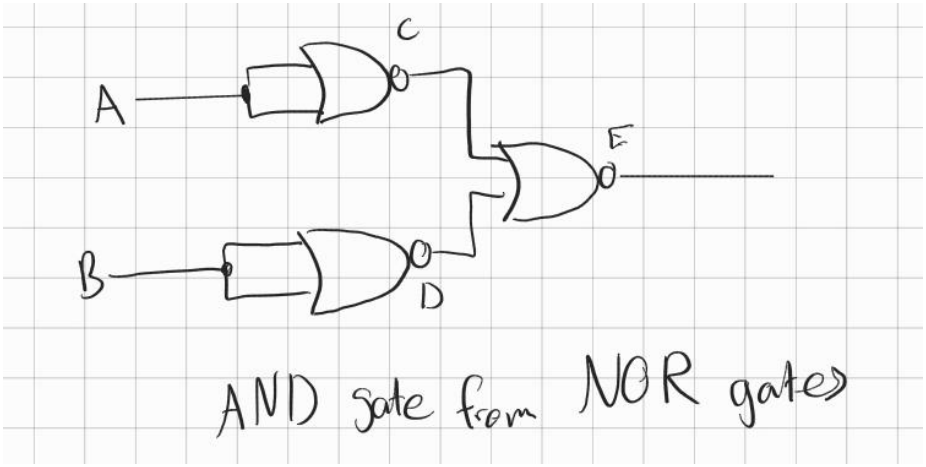
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR

c. **Record your response to 6.1.c below:**



d. Record your response to 6.1.d below:



e. Record your responses to 6.1.e below:

X = (A+B)$\overline{B}$ +$\overline{B}$+ CB

Truth Table			
A	**B**	**C**	**Out**
0	**0**	**0**	**1**
0	**0**	**1**	**1**
0	**1**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**1**
1	**0**	**1**	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

f. Simplify the following Boolean expressions

i. $X = (A + B)\,\underline{B} + \underline{B} + BC$

X = $\underline{B} + \mathrm{C}$

ii. $X = \underline{A}(A + B) + (B + AA)(A + \underline{B})$

X = B+A

6.2 Sequential Logic

a. Record your response to 6.2.a below:

A half-adder adds two bits (A and B) independently, producing a sum (S) and a carry (C). It does not consider carry input from previous stages, resulting in a 4-row truth table.
A full adder is more complex, adding three bits (A, B, and Cin) and considering carry from the previous stage. It outputs a sum (S) and a carry-out (Cout), making it suitable for cascading to add multi-bit numbers. And because of the three variables, it results in a 8-row truth table.

6.3 Frequency Counter

a. Record your response to 6.3.a in the box below:

In a synchronous counter, all flip-flops (or stages) change state simultaneously with each clock pulse. This simultaneous state change is facilitated by a common clock signal that triggers all flip-flops at the same time. The design of synchronous counters aims for better precision and synchronization, albeit at the cost of potentially requiring more complex circuitry. One advantage of synchronous counters is the uniform propagation delay through all stages, ensuring that all flip-flops change state in sync, contributing to precise timing in digital circuits.

On the other hand, an asynchronous counter (Ripple Counter) operates differently. In this type of counter, the clock signal triggers only the first flip-flop. Subsequent flip-flops change their state based on the output of the preceding flip-flop, causing a 'ripple' effect through the stages. While asynchronous counters boast a simpler design, they may encounter issues such as propagation delay and clock skew. These challenges can lead to potential timing problems, particularly because the propagation delay tends to increase as the count progresses. Consequently, asynchronous counters may offer less precise timing compared to a synchronous counter.

b. Record your response to 6.3.b in the box below:

Despite the Xilinx Artiz7 FPGA being a better choice over the Arduino Due, there are still times when the Arduino might be the more appropriate choice. For starters, Arduino is cheaper, making it more openly available. The Arduino is also less complex and easier to use, making it a better choice for simpler projects or less experienced users. The Xilinx Artiz7 FPGA excels at parallel processing and also has the feature of implementing custom logic tailored hardware. They also have a much faster

processing speed/time, making it the obvious choice for more complex designs. The Arduino has a smaller number of inputs and outputs compared to the Xilinx Artiz7 FPGA. While both have similar types of inputs and outputs, the Xilinx Artiz7 FPGA has customizable interfaces through programmable logic. The Arduino has a much slower clock speed than the Xilinx Artiz7 FPGA. The Arduino is, however, much more cost effective and will often do the job well enough for smaller/less complex projects. The Arduino Due costs around $50 and the Xilinx Artiz7 FPGA costs around $300. While the power used generally depends on the task/application, typically it is safe to assume that the Arduino consumes less power than the Xilinx Artiz7 FPGA. This however, comes at the cost of many other features of the Xilinx Artiz7 FPGA being better than the Arduino (capabilities, speed, uses, etc.).

c. Record your response to 6.3.c in the box below:

The given code was a frequency counter that was interrupt driven. Interrupt driven is also more efficient when compared to polling. According to the article Polling vs Interrupts: Exploring their Differences and Applications - Total Phase, polling is less efficient due to the fact that it is constantly polling for events while interrupt can detect when those events are happening and do other processes in the meantime. The maximum frequency measurement possible with the given configuration was ~310 kHz. The Arduino fails when it does due to the Arduino's limitations. At higher frequencies, quicker rise and fall times as well as how often they occur make it harder for the Arduino to detect pulses. At these high frequencies, the Arduino also could overflow and it would fail to detect higher frequencies.

d. Record your response to 6.3.d in the box below:

Up to 310 kHz, the Arduino frequency measurement was fairly accurate. When in the higher range of those frequencies, the Arduino would be off by a little (on the scale of 5-10 Hz). The most plausible source of this error is the runtime of the Arduino script, but it could also be attributed to the Arduino clock not being perfect and as the frequencies get higher, there would be more error seen in the measurements due to the Arduino having a harder time detecting the pulses.

e. Record your response to 6.3.e in the box below:

The BASYS3 is more capable due to its higher clock speed.