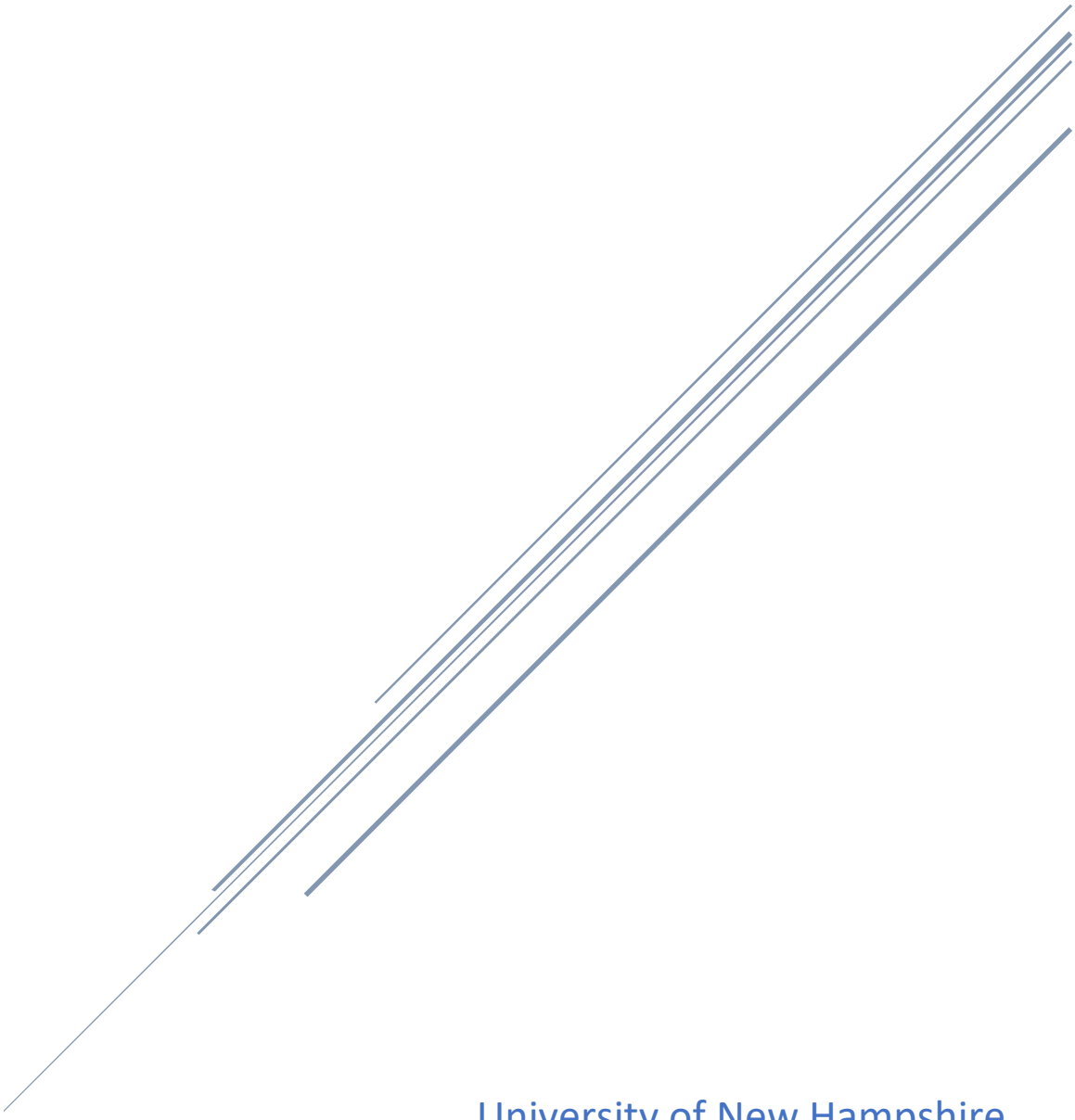
FINAL PROJECT: ROBOTIC ARM

Ryan Baer, Nicholas Snyder



University of New Hampshire

ECE 583

Overview:

The goal of this project was to program a robotic arm using servo motors. To accomplish this, the team first converted the one-hundred-megahertz clock signal available on the board to a pulse-width modulated (PWM) signal of fifty hertz, with a duty cycle range of one to two milliseconds. Servos require a PWM signal as input to control their position, with a duty cycle of one millisecond indicating a position of negative ninety degrees from center and a duty cycle of two milliseconds indicating a position of positive ninety degrees from center. The team then used the I/O capabilities of the Basys 3 board to hard code set positions into the arm, with the switches on the bottom of the board used to control each servo. Each servo was driven by two switches, allowing for up to four possible positions, but the team only used three positions to control the arm's movements: full left, full right, and center. The output signal was then sent through the Pmod headers on the side of the board. However, since servos typically require around five volts for proper operation, the team had to power the servos externally, as the board was only able to supply a maximum of 3.3 volts.

Why we chose this:

We selected this for our final project because we have experience in robotics, and we already had access to multiple servo motors and a 3D printer. To make the most of our respective skills, Ryan focused on designing and modeling the arm pieces, while Nick worked on the codebase. Once the individual tasks were completed, we assembled the arm and worked together to debug the code.

Challenges:

Throughout the development of our final design, we faced several challenges. Initially, we struggled to synthesize our design with delays like we had used in our simulations. After overcoming this hurdle, we faced the issue of our design's actual performance not matching our simulations. We noticed jittery arm movement when the pulse frequency was either too large or too small. It took us several trials with different clock divisions and pulse widths to obtain satisfactory results. However, we still fell short of our initial goal of achieving 180 degrees of movement. Finally, we had hardware-related challenges as the shoulder servo lacked the torque to hold the rest of the arm in place. To overcome this, we switched to a higher quality digital servo. Despite these challenges, we were able to work together to overcome them and deliver a functional robotic arm.

Design sources:

Our design is split between three modules, top.v, PWM_generator.v, and clock_divider.v so all design sources, simulation sources, and simulation results are shown below.

clock_divider.v

```
`timescale 1ns / 1ps
module clk_divider (
    input wire clk_in,
    output reg clk_out
    );

    reg [22:0] counter = 0;
    parameter div = 30;

    initial clk_out = 1;

    always @(posedge clk_in)
```

```
    begin
        if (counter == div)
        begin
            counter <= 0;
            clk_out <= ~clk_out;
        end
        else
        begin
            counter <= counter + 1;
        end
    end
endmodule
```

PWM_generator.v

```
`timescale 1ns / 1ps
module PWM_generator(
    input [1:0] position,
    input clk,
    output reg modulated_signal
    );

    clk_divider clk_div(clk, clk_out);

    reg [15:0] width;
    reg [15:0] time_v;

    // set signal high at t=0
    initial begin
        modulated_signal = 1;
        time_v = 0;
    end

    always @(posedge clk_out)
    begin

        // determine width from switch position
        case (position)
            2'b00: width <= 1500;//3
            2'b01: width <= 2200; //4
            2'b10: width <= 800; //2
            2'b11: width <= 1500; //3
        endcase

        if (time_v < width) modulated_signal <= 1;
        else modulated_signal <= 0;

        if (time_v > 20000) time_v <= 0;
    end
endmodule
```

top.v

```
`timescale 1ns / 1ps
module top (
    input clk,              // board clock
    input [5:0] sw,         // 2 switches per servo, 3 positions
    output [8:0] light,
    output [2:0] JB          // 3 servos in total, Pmod Jb pins 1-3
    );

    // update position of servo 0
    PWM_generator pwm_0(sw[1:0], clk, JB[0]);

    // update position of servo 1
    PWM_generator pwm_1(sw[3:2], clk, JB[1]);

    // update position of servo 2
    PWM_generator pwm_2(sw[5:4], clk, JB[2]);

    assign light[0] = sw[0];
```

```
    assign light[1] = sw[1];
    assign light[2] = sw[2];
    assign light[3] = sw[3];
    assign light[4] = sw[4];
    assign light[5] = sw[5];

    assign light[6] = JB[0];
    assign light[7] = JB[1];
    assign light[8] = JB[2];

endmodule
```

Simulation Sources:

testbench_clock_divider.v

```
`timescale 1ns / 1ps
module testbench_clk_divider;

    reg clk_in = 0;
    wire clk_out;

    clk_divider cd (
        .clk_in(clk_in),
        .clk_out(clk_out)
    );

    always #10 clk_in = ~clk_in;
    initial
    begin
        #10000000;
        $finish;
    end

endmodule
```

testbench_PWM_generator.v

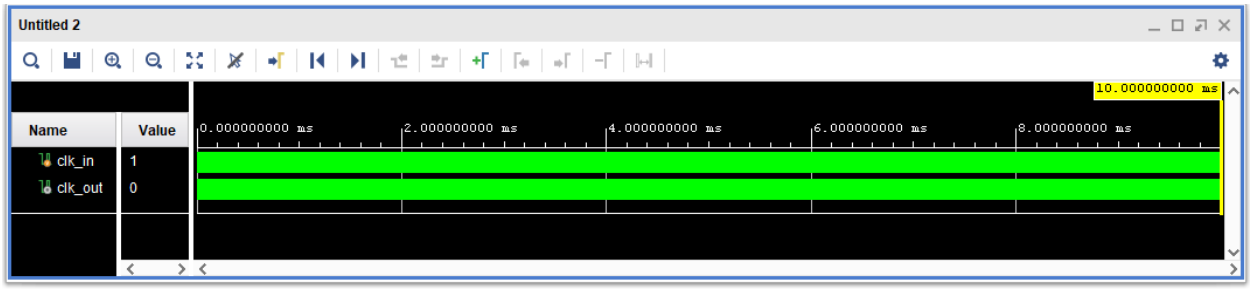
```
`timescale 1ns / 1ps
module testbench_PWM_generator;

    reg [1:0] position = 2'b00;
    reg clk = 0;

    PWM_generator dut1(
        .position(position),
        .clk(clk),
        .modulated_signal(modulated_signal)
    );

    always #10 clk = ~clk;

    initial
    begin
        position = 2'b00;
        #100000000;
        position = 2'b01;
        #100000000;
        position = 2'b10;
        #100000000;
        position = 2'b11;
        #100000000;
        $finish;
    end
endmodule
```

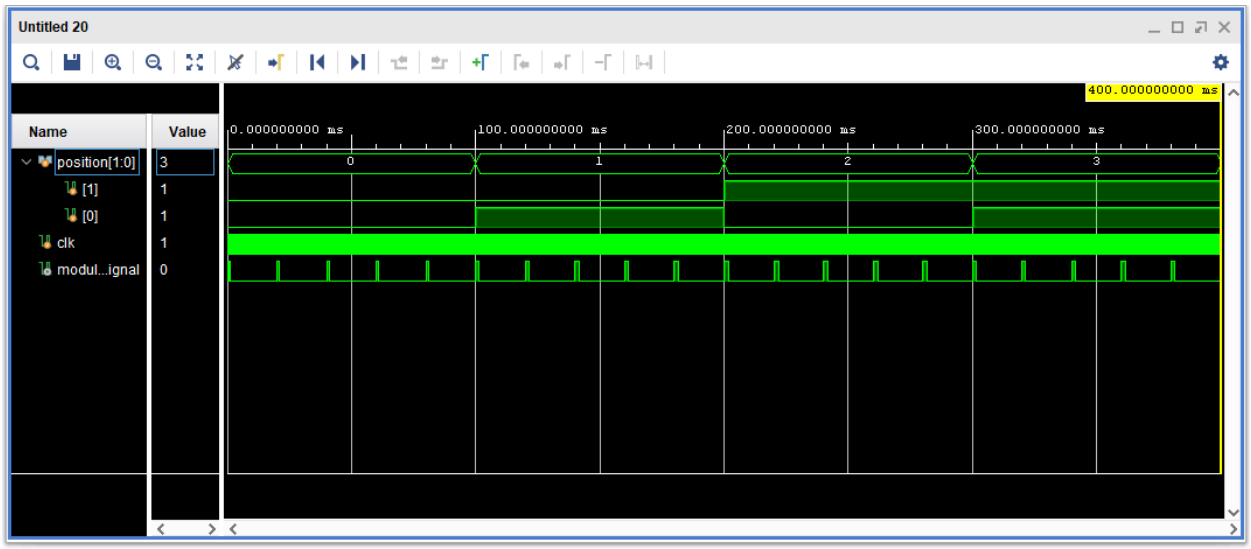
tb_top.v

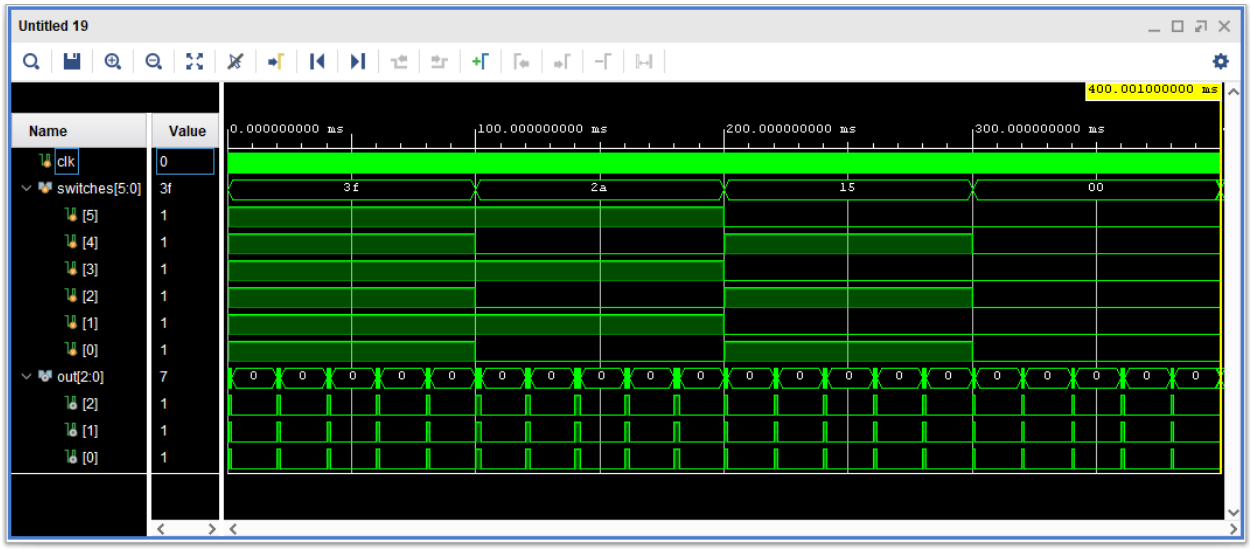
```
module tb_top;
    // Declare inputs and outputs for the testbench
```

```
    reg clk;
    reg [5:0] switches;
    wire [2:0] out;

    // Instantiate the top module
    top dut (
        .clk(clk),
        .sw(switches),
        .JB(out)
    );

    // Clock generation
    initial clk = 0;
    always #10 clk = ~clk; // Generate a 100MHz clock

    // Stimulus generation
    initial begin

        // Move servos to 0 degrees
        switches = 8'b111111; // set all servos to 1.5ms duty cycle
        #100000000;

        // Move servos to 90 degrees
        switches = 8'b101010; // set all servos to 2ms duty cycle
        #100000000;

         // Move servos to 0 degrees
        switches = 8'b010101; // set all servos to 1.5ms duty cycle
        #100000000;

        // Move servos to -90 degrees
        switches = 8'b000000; // set all servos to 1ms duty cycle
        #100000000;

        // Move servos to 0 degrees
        switches = 8'b111111; // set all servos to 1.5ms duty cycle
        #100000000;

        // End simulation
        $finish;
    end
endmodule
```

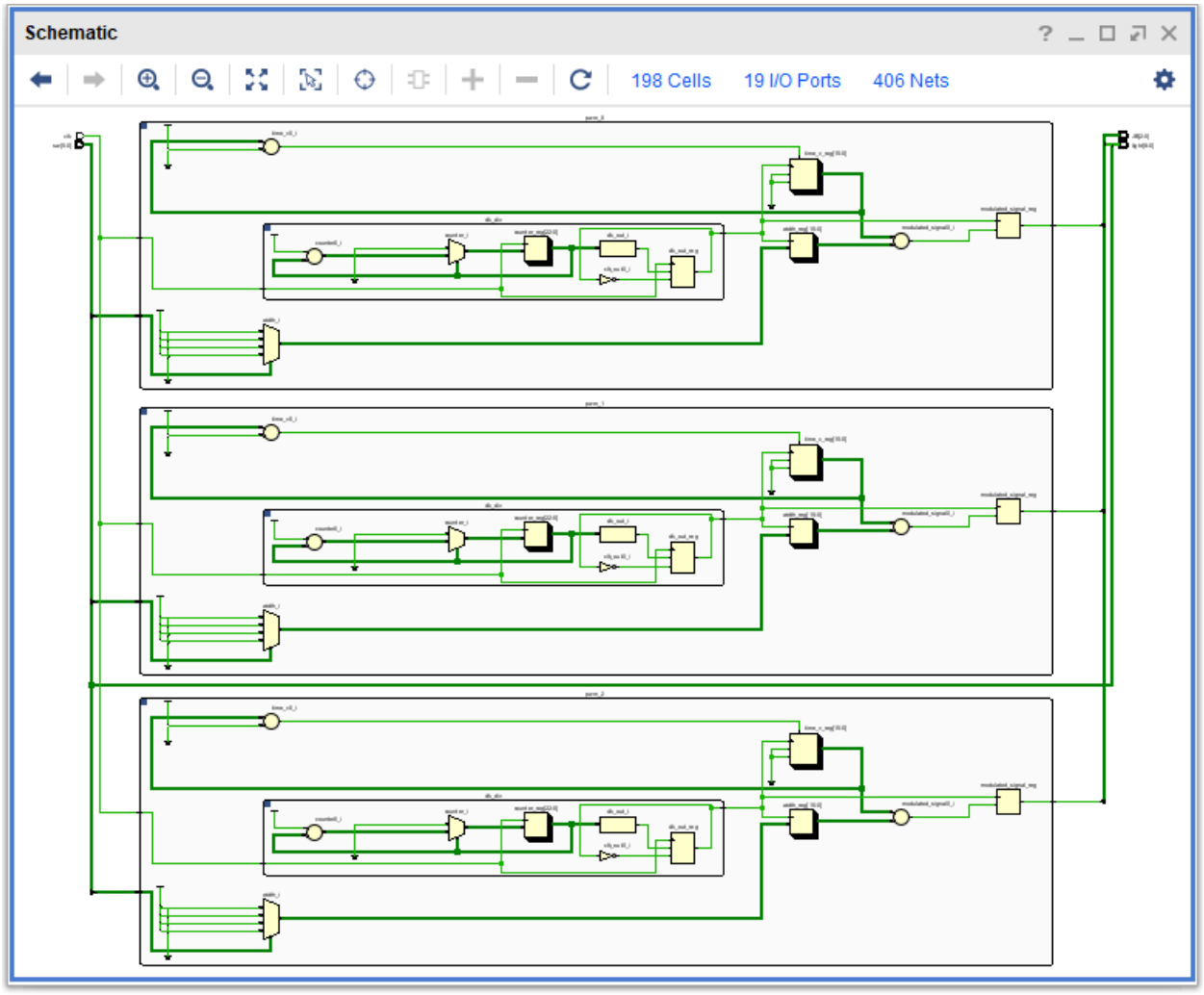
Simulation Results:




Untitled 20
Name
Value
position[1:0]
3
[1]
1
[0]
1
clk
1
modul...ignal
0
0.000000000 ms
100.000000000 ms
200.000000000 ms
300.000000000 ms
400.000000000 ms
0
1
2
3


Untitled 19
Name
Value
clk
0
switches[5:0]
3f
[5]
1
[4]
1
[3]
1
[2]
1
[1]
1
[0]
1
out[2:0]
7
[2]
1
[1]
1
[0]
1
0.000000000 ms
100.000000000 ms
200.000000000 ms
300.000000000 ms
400.001000000 ms
3f
2a
15
00

RTL Schematic:



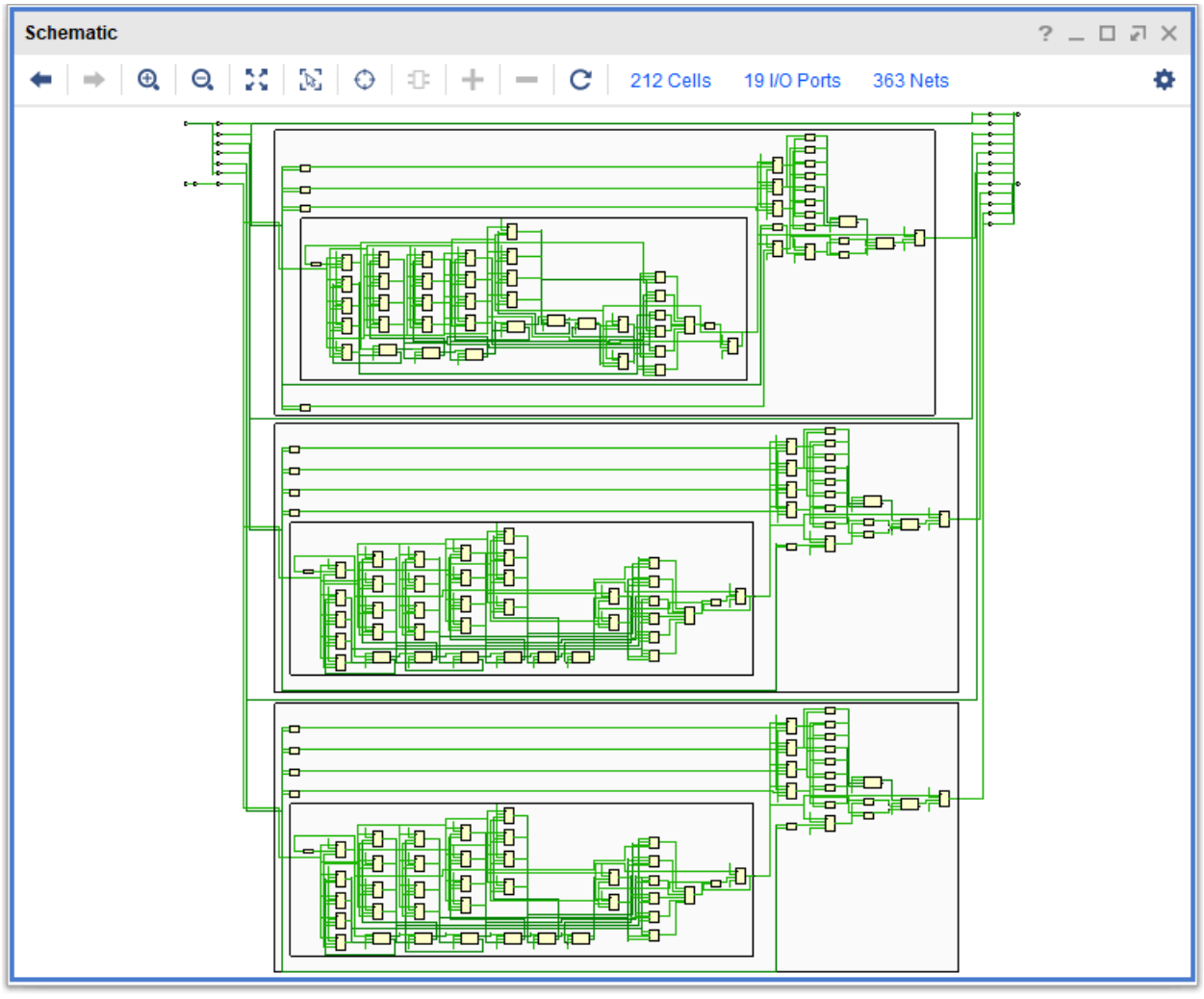
Constraints:

RoboticArm.xdc

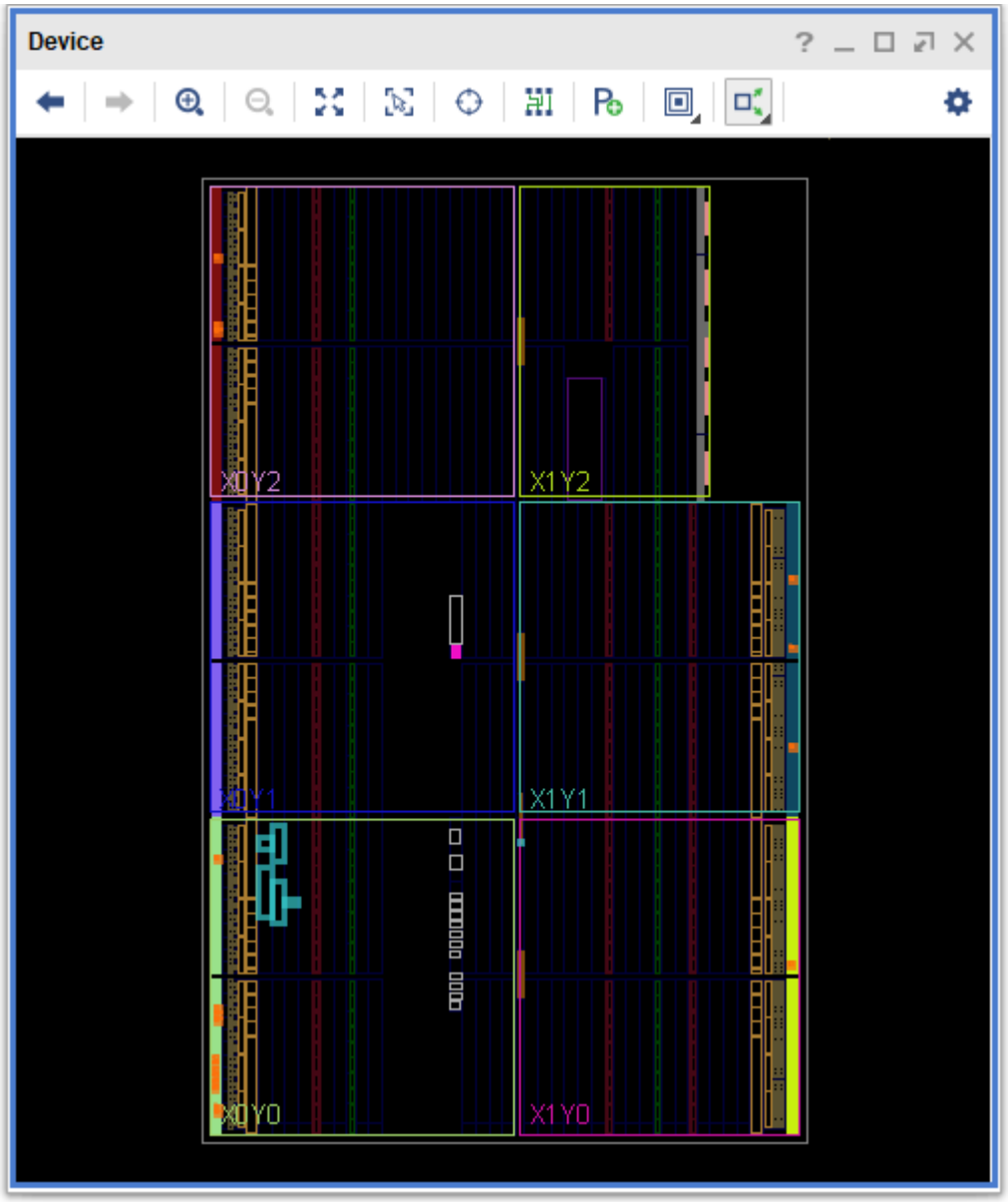
```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]

## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {sw[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {sw[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {sw[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {sw[3]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {sw[4]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {sw[5]}]

## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {light[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {light[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {light[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {light[3]}]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {light[4]}]
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports {light[5]}]
set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports {light[6]}]
set_property -dict { PACKAGE_PIN P1    IOSTANDARD LVCMOS33 } [get_ports {light[7]}]
```

```
set_property -dict { PACKAGE_PIN L1    IOSTANDARD LVCMOS33 } [get_ports {light[8]}]

## Pmod Header JB
set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports {JB[0]}];#Sch name = JB1
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports {JB[1]}];#Sch name = JB2
set_property -dict { PACKAGE_PIN B15   IOSTANDARD LVCMOS33 } [get_ports {JB[2]}];#Sch name = JB3

## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```

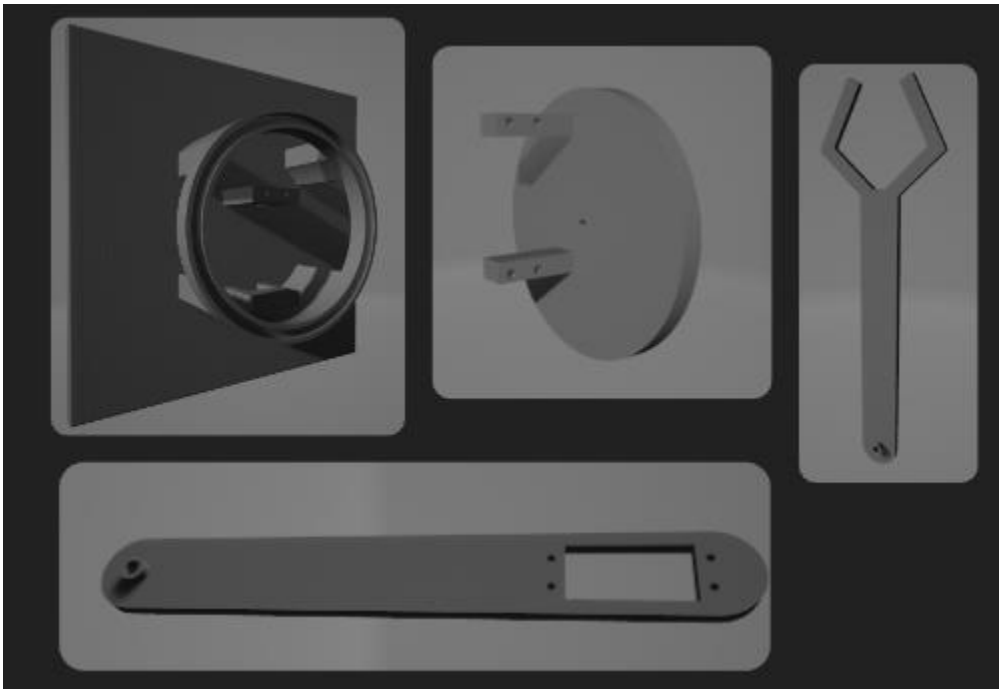
Synthesized Design:



Implemented Design:



3D Models:



Final Design: