# Rotary Inverted Pendulum

*ECE 526 Project Report*

Balance Amongst the Chaos

*4/28/2016*

# Table of Contents



# Tables of Tables

## Tables of Figures

# 1. Introduction

Picture this—a yogi standing in a park. Holding a pose on a single leg—hands raised above their head remaining absolutely still. They have attained a physical state of balance and equilibrium. If they lose focus, they fall over. If they catch wind of the country's current political climate…chaos ensues.

Essentially, a yogi balancing in a park amongst a wild political climate is what we call a beautiful metaphor for an inverted pendulum! The idea is to find balance amongst chaos—a state of equilibrium in an otherwise unstable system.

# 2. Abstract

We will construct a rotary inverted pendulum and actively drive it toward an unstable equilibrium by writing VHDL to implement a control feedback loop on the Cyclone V FPGA. The movement of the pendulum will send an encoded signal to the control logic on the FPGA. The control logic will process the signal and send a control signal through the motor driver to drive the stepper motor. This will result in the unactuated pendulum arm remaining approximately upright by moving the base of the pendulum arm. Figure 1 is a top level diagram that shows the interconnection between all pieces involved in the rotary inverted pendulum project.

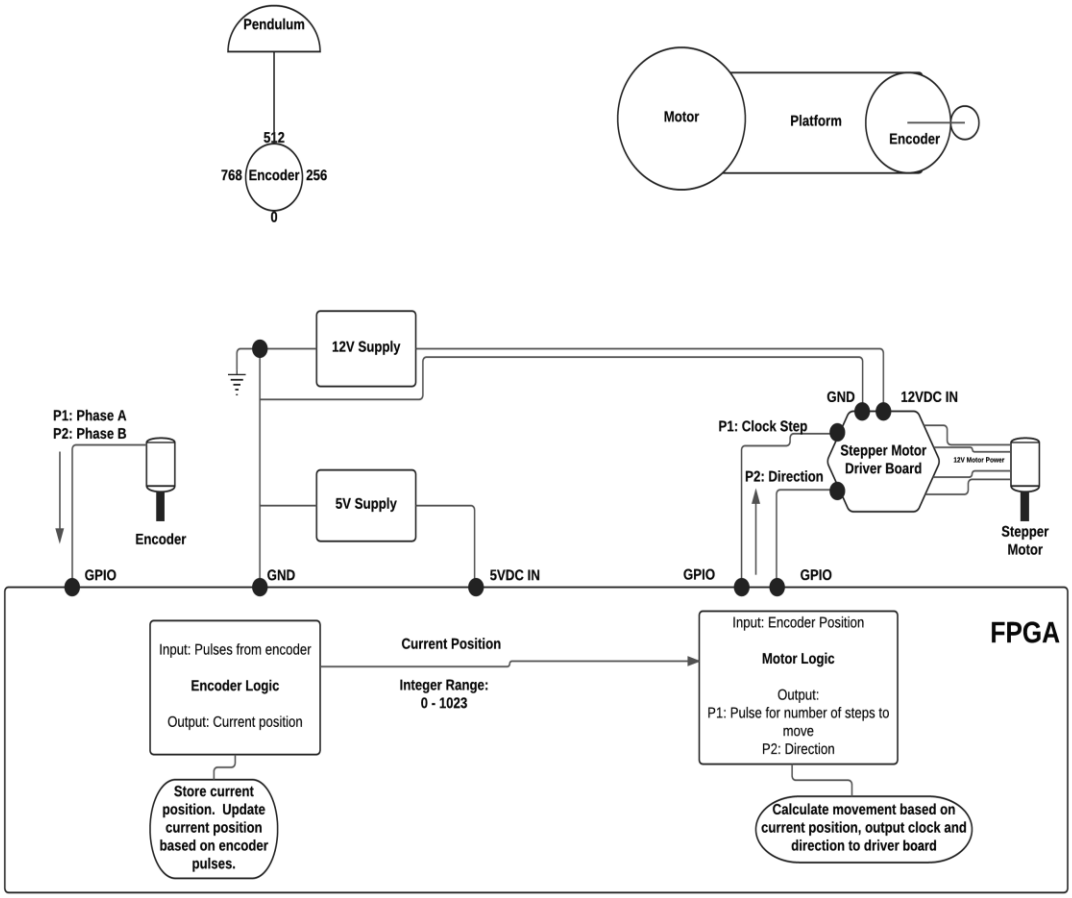


Figure 1 – Inverted Pendulum System Overview

## 3. The Team

| Table 1 - Contact Info | | |
|---|---|---|
| Name | Email | Phone |
| Jesse Chaddock | Jkc0022@uah.edu | 575-642-6234 |
| Brian Irelan | Bti0002@uah.edu | 256-513-9750 |
| Jason Renner | Jgr0007@uah.edu | 865-805-1806 |

## 4. Requirements

i. A physical test bench shall be created to balance and object from an initial state of rest to an inverted balanced state.
ii. Directional position from the object shall be detected by a rotary encoder.
iii. The object shall be moved by a motor.
iv. All control logic shall be implemented on an FPGA.
v. Source files shall be written in VHDL.
vi. Simulation of source files shall be done to verify functionality before integration.

## 5. Hardware

- Stepper Motor - moves the base of the pendulum in order to keep it balanced
- Big Easy Driver – drives stepper motor according to signal received from Cylone V
- Power Supply - supplies 12 volts to Big Easy Driver for operation
- Rotary Encoder – encodes movement of pendulum as signal to be sent to FPGA
- Cyclone V FPGA – houses VHDL implementation of control logic
    - Terasic DE0-Nano SoC – interfaces between encoder, FPGA, and Big Easy Driver
    - Arrow BeMicro CV – second prototyping board to speed up development
- Pendulum – object to be balanaced
- Aluminum Platform – stable base for pendulum to swing on

## 6. Bill of Materials

Table 2 is a representation of costs associated with this project.

Table 2 - Bill of Materials

| Item | Quantity | Cost |
|---|---|---|
| ROB-09238 Stepper Motor with Cable | 2 | $14.95 |
| COM-11102 Rotary Encoder - 1024 P/R (Quadrature) | 1 | $39.95 |
| ROB-12859 Big Easy Driver | 1 | $17.96 |
| BeMicro CV Cyclone V | 1 | $50 |
| Terasic DE0 | 1 | $99 |

## 7. Construction

The rotary inverted pendulum is constructed by first attaching the pendulum to the rotary encoder and connecting them to the stepper motor via the aluminum platform. The stepper motor was clamped to a block of wood to stabilize the device. The encoder is wired to the Terasic or the Arrow development board depending on which board is used for prototyping. The development board is connected to the Big Easy Driver, and the 12 volt power source provides power for the Big Easy Driver. Finally, the Big Easy Driver is wired to the stepper motor.

A bi-polar stepper motor is simply two inductors and some magnets connected to an axle. To operate a stepper motor, higher voltages are required than the FPGA can provide and precise control of these voltages is necessary to move the motor. An external circuit is needed to perform its operation. The Big Easy Driver board simplifies the interface and allows the FPGA to send digital control signals in terms of a clock pulse and a direction bit. Figure 2 is an example of the board and how it simplifies the motor control signals generated by the FPGA.

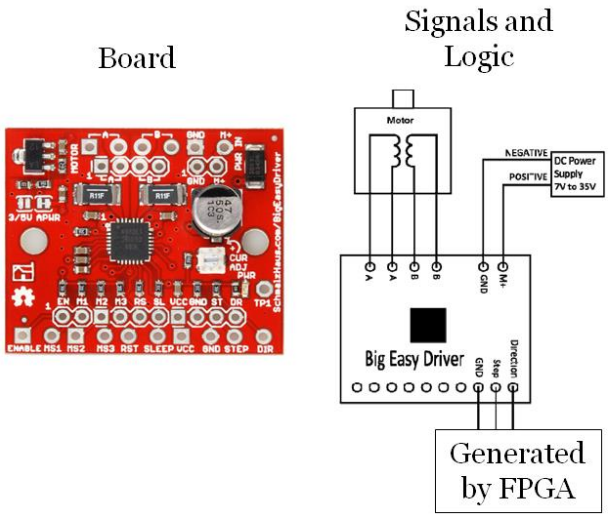


Figure 2 - Big Easy Driver

The test bench construction is represented in Figure 3.

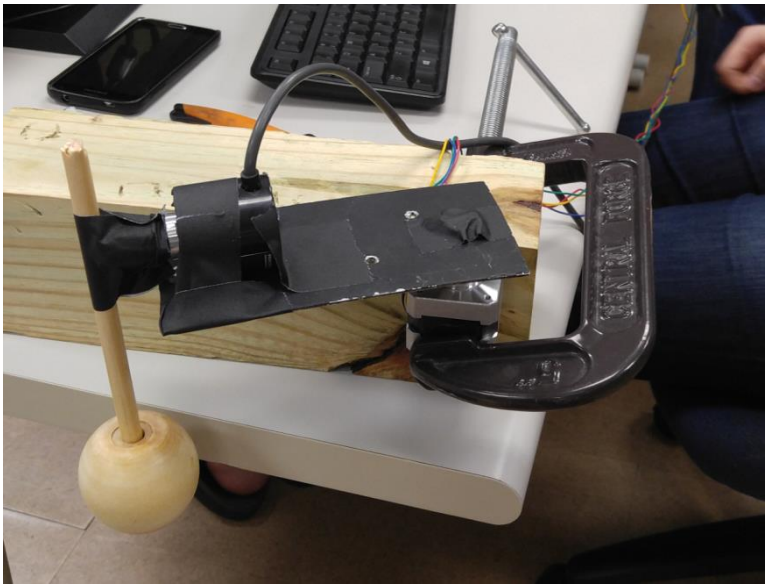

Figure 3 - Test Bench

# 8. VHDL Design and Implementation

## 8.1 Clock Divider

The purpose of the clock divider is to slow down the on-board 50MHz clock to a point that the stepper motor has time to physically react between each pulse. The stepper motor is comprised of two coils. Using the stepper motor specifications from its data sheet, the time required for the stepper motor to make one movement can be calculated using the inductor time constant $\tau = \frac{L}{R}$:

$$I = 0.66\,A \qquad L = 48.4\,mH \qquad V = 12V$$

$$\tau = \frac{L}{R} = \frac{LI}{V} = \frac{0.66 \cdot 0.0484}{12} = 0.00266s$$

$$f = \frac{1}{\tau} \approx 376Hz$$

To calculate the necessary value for the counter, the on-board clock of 50MHz is divided by our needed clock rate, and then divided by 2 (as the process must trigger on both the rising and falling edges).

$$2 \cdot counter = \frac{50MHz}{376Hz} = 132976$$

$$counter = 66488$$

## 8.2 Rotary Encoder

Figure 4 - YUMO Incremental Rotary Encoder

### Overview

A rotary encoder is an electro-mechanical device that converts the motion of an axle into a digital output. Its utilization in the project will allow the FPGA to track the physical motion of the pendulum.

An FPGA is an ideal platform for processing rotary encoder outputs. The maximum rated speed of this particular rotary encoder is 6000 RPM. A clock frequency of 200 kHz or greater would be needed to accurately sample the output of the encoder at all rated speeds. Although this is a fairly low clock rate, the fact that the output does not need to be sampled may contribute to overall power savings.

Two digital outputs of the rotary encoder are utilized to detect motion. Phase A and Phase B output a square wave with a frequency that correlates with the current rotation of the rotary encoder. As the encoder is turned clockwise, Phase A leads Phase B by 90 degrees. As the encoder is turned counter-clockwise, Phase B leads Phase A by 90 degrees.

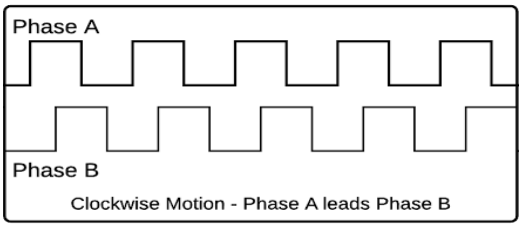


Figure 5 - Phase A Leading

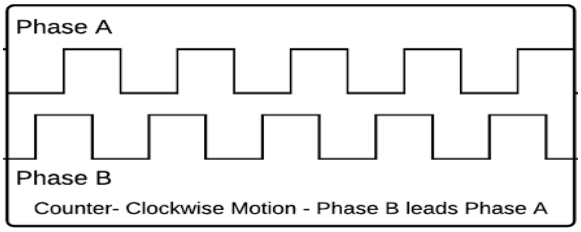


Figure 6 - Phase B Leading

### Processes

The VHDL process will need to accurately recognize both clockwise and counterclockwise motion of the rotary encoder. The FPGA cannot physically monitor the falling edge and rising edge of a signal in the same process. Due to this fact, three separate process are required.

| Table 3 - Rotary Encoder Sensitivity List | | |
|---|---|---|
| Process | Sensitivity List | Result |
| 1 | Rising Edge of Phase A | Increments variable when Phase A leads Phase B (Clockwise Motion) |
| 2 | Falling Edge of Phase A | Increments variable when Phase B leads Phase A (Counter-Clockwise Motion) |
| 3 | Rising Edge of Phase A | Outputs difference between variables in process 1 and 2 |

## 8.3 Stepper Controller

### Overview

The stepper controller is the control logic for the inverted pendulum. It takes the encoder position and direction and outputs a pulse, direction, and step to the Big Easy Driver board to turn the motor. The system itself is a feedback controller and the goal is to reach a state of equilibrium where the output is no longer driving change to the input and vice versa. As the system begins in an unstable state we expect to see heavy oscillation and movement dwindle towards a steady state. Ideally we have a function that looks similar to a dampened sine wave where the amplitude approaches zero over time.

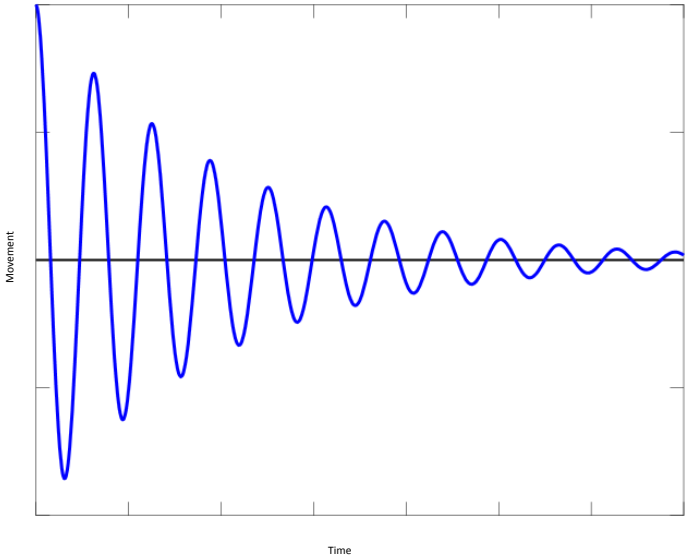


Figure 7 - Ideal System Movement into Equilibrium

## Rotary Encoder Application

The rotary encoder will provide the positional location of the pendulum. Upon start up the positional assignment of the rotary encoder will be zeroed. This will provide an initial known state for the system. Take the assumption that the pendulum will start in a hanging position opposite of the state of equilibrium. The resolution of the YUMO A6B2-CWZ3E-1024 is 1024. By applying a unit circle over the position of the rotary encoder at (0, -π) the rotary encoder position is 0, (π, 0) 256, (0, π) 512, and (-π, 0) 768. This is depicted in Figure 8.

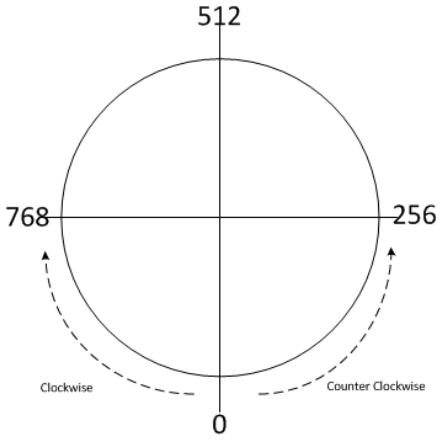


Figure 8 - Rotary Encoder Positional Assignments

From the assumed starting position, clockwise rotation results in decrementing the position counter of the rotary encoder and counterclockwise rotation results in incrementing the position. The

system is constrained by the resolution of the encoder meaning that the amount of precision is limited to the number of ticks the encoder can report to the FPGA.

## Stepper Motor Application

The stepper motor is the source of movement for the pendulum. It provides the stimulus to change the position of the encoder. The motor position can be calculated by knowing the step size, step direction, and number of pulses that have been applied to the motor. Again assume that the starting position is 0. This will be the known state of the motor position. A full step can move the motor 1.8 degrees for a single pulse supplied to the motor. A half step will move the motor .9 degrees for a single pulse supplied to the motor. 180 degrees of motion will take 200 pulses at a half step and 100 pulses at a full step. The positional assignments are illustrated below in Figure 9.

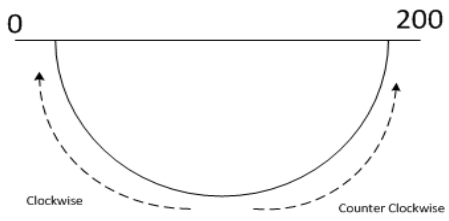


**Figure 9 - Stepper Motor Positional Assignments**

If the starting position is initialized at zero, position can be tracked easily by adding two ticks for full step movement and 1 tick for half step movement when a rising edge of a pulse is detected.

## State Machine

The overall algorithm for the feedback controller is designed as a Moore state machine with four specific states: Initialization, Swing Up, Balancing Act, and Balanced. The state machine is evaluated every clock cycle to determine the necessary action to have the pendulum reach the point of equilibrium. Figure 10 displays the overall process while Table 4 gives a brief overview of what happens in each state.

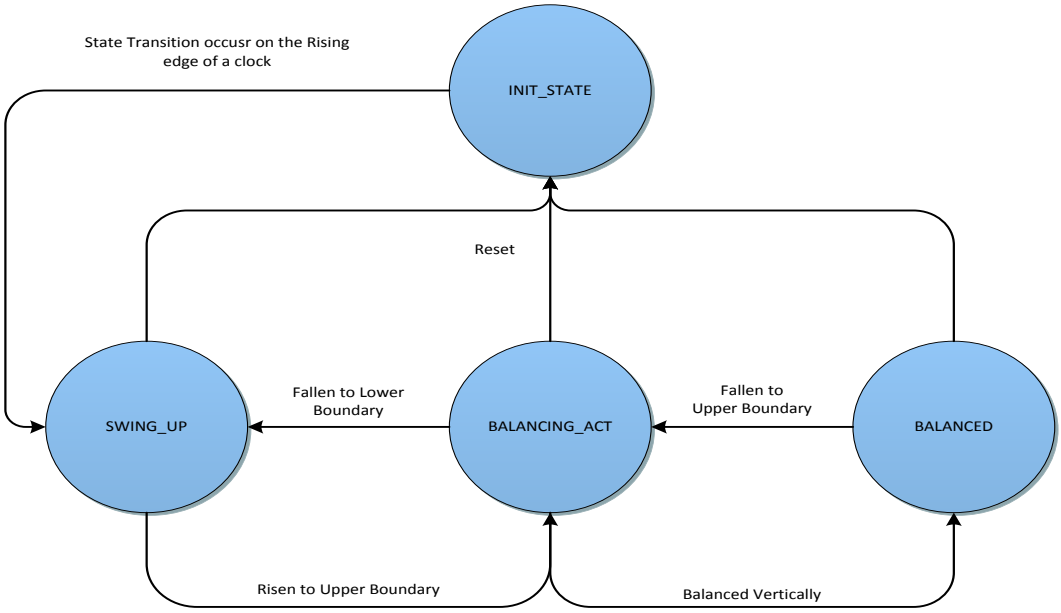


Figure 10 - Stepper Controller State Machine

| Table 4 - State Machine Overview | | | | |
|---|---|---|---|---|
| State | Function | Output | Transitions To | Case |
| INIT_STATE | The initialization state initializes position variables to zero. This is the referenced starting point with motor position at zero and rotary encoder position assumed at zero and hanging vertically. | While in the initialization state, the pulse output is driven to zero, rotation is set to counter clockwise and step size is set to full step. Although these are initialized the motor will not move without a pulse. | SWING_UP | Once a rising edge of a clock is detected and the reset signal is low, the state machine will transition into the SWING_UP state. |
| SWING_UP | The Swing Up state will move the pendulum back and forth at a full step in order to swing the pendulum above the established boundary condition for either clockwise or counter clockwise rotation on the rotary encoder. The directional signal to the motor is toggled upon reaching a certain point; the motor position is limited to 180 degrees of motion to keep the cables from wrapping around the test bench. | A pulse out will be transition high or low every clock cycle depending on the previous pulse output. Step size will be a constant full step out while in this state. Direction is reversed after hitting or passing a specified boundary point. | INIT_STATE | The state machine will transition into the initialization state upon the reset signal going high. |
| | | | BALANCING_ACT | The state machine will transition into the Balancing Act state upon the pendulum arm being above a certain threshold. |
| BALANCING_ACT | The Balancing Act state will toggle the direction of the motor in the opposing direction the pendulum is falling. This will apply a force in the opposing direction the pendulum is falling in hopes that it will be reach a state of equilibrium. | A pulse out will be transition high or low every clock cycle depending on the previous pulse output. Step size will be a constant half step out while in this state. Direction is reversed | INIT_STATE | The state machine will transition into the initialization state upon the reset signal going high. |
| | | | SWING_UP | The state machine will transition into the Swing Up state upon falling below the established threshold boundaries. |
| | | | BALANCED | The state machine will transition into the balanced state upon |

| | | | | |
|---|---|---|---|---|
| | | | | reaching the boundary condition established to be in the required state. |
| BALANCED | The balanced state is the state of equilibrium where the position no longer needs to be adjusted. The pendulum arm is upside down and vertically balanced. | The pulse out will be forced to zero so the motor position will not move. | INIT_STATE | The state machine will transition into the initialization state upon the reset signal going high. |
| | | | BALANCING_ACT | The state machine will transition into the Balancing Act state upon the pendulum arm being falling below a certain threshold. |

## Swing Up State

While in the swing up state the logic is constrained by the position of the rotary encoder as well as the position of the motor. The area shaded in red in Figure 11 and Figure 12 depicts the range of motion that each piece of hardware is restricted to.

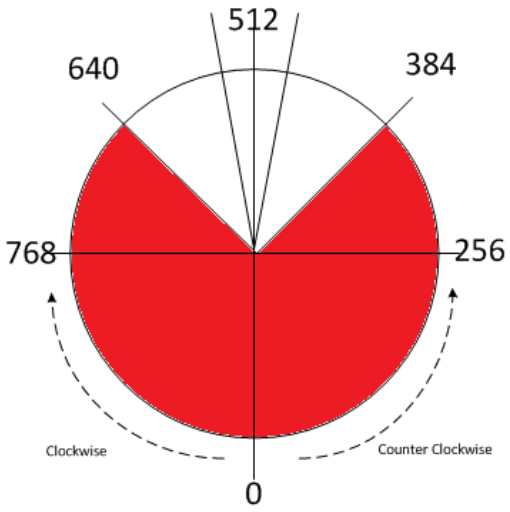


Figure 11 - Swing Up State Pendulum Position

The pendulum arm is given 135 degrees of freedom from the initialized position in both the clockwise and counter clockwise direction. This relates to a count value of 384 counter clockwise or a count value of 640 clockwise. Once above these certain boundary conditions the state machine will transition into the Balancing Act state.

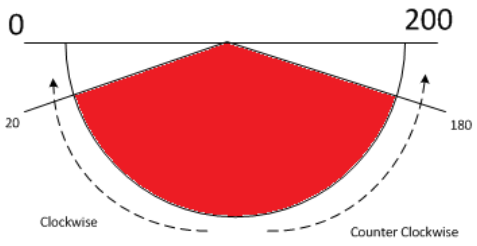


Figure 12 - Swing Up State Motor Position

To avoid unwanted wrapping of our rotary encoder cables and create a consistent swing with the pendulum arm the motor position will be tracked and maintained between 20 and 180 ticks while in

the swing up state. If the motor reaches a position in the Balancing Act state that is greater or less than the 20 to 180 tick range, it will switch direction until the upper or lower boundary is hit to maintain the limited range of movement in this state.

## Balancing Act State

While in the Balancing Act state the logic is constrained by the position of the encoder. The area shaded in red in Figure 13 is a representation of the 70 degrees of motion that this state is limited to.

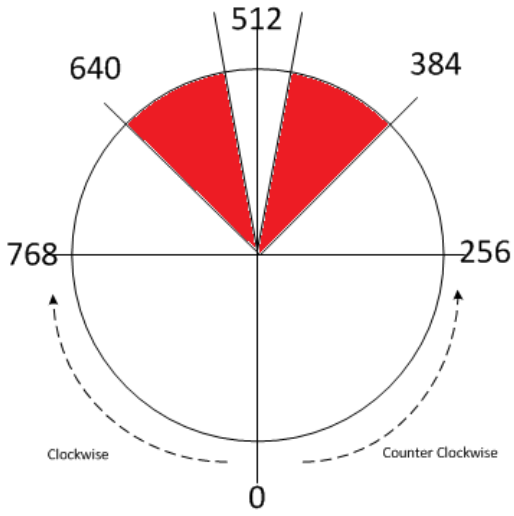


Figure 13 - Balancing Act Rotary Encoder Positions

The direction the pendulum is falling is passed to the stepper controller. While in this state the direction the motor moves is in the opposing direction the pendulum is falling. This will counteract the force and drive the system into a state of equilibrium. Figure 14 is a depiction of the motor movement based on the direction the pendulum is falling.

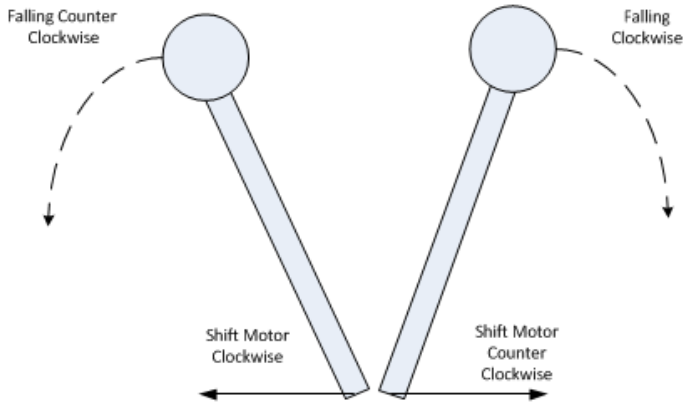


Figure 14 – Balancing Act Motor Movement

Future implementation of this process can be revised based on the rate of change at which the pendulum is falling. The value can be assessed based on the previous recorded position during a read from the clock. Based on the rate of change the step size can be adjusted to offset a greater rate of change in the pendulum.

## Balanced State

Within the Balanced state, the system has assumed the stable point and no longer requires positional adjustment. The state is given a few degrees of freedom and currently covers around 20 degrees of movement, but can be easily adjusted. The area shaded in red in Figure 15 represents the positional assignment of the pendulum while in the Balanced state.

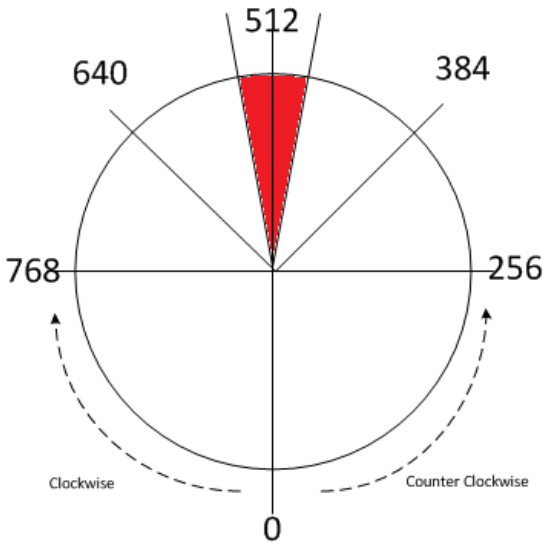


Figure 15 - Balanced Encoder Position

The design was implemented in VHDL with two processes controlling the state logic. Table 5 represents how the state logic described above are seen in the VHDL. For a more detailed view of the VHDL refer to the source code STEPPER_CONTROLLER.VHD.

| Table 5- VHDL Stepper Controller Processes | | |
|---|---|---|
| Process | Sensitivity List | Result |
| 1 | Rising Edge of Clk, Reset | Next state logic to move the state machine along. |
| 2 | Falling Edge of Clk, Reset | Output logic based on the states. |

# 9. Source Control

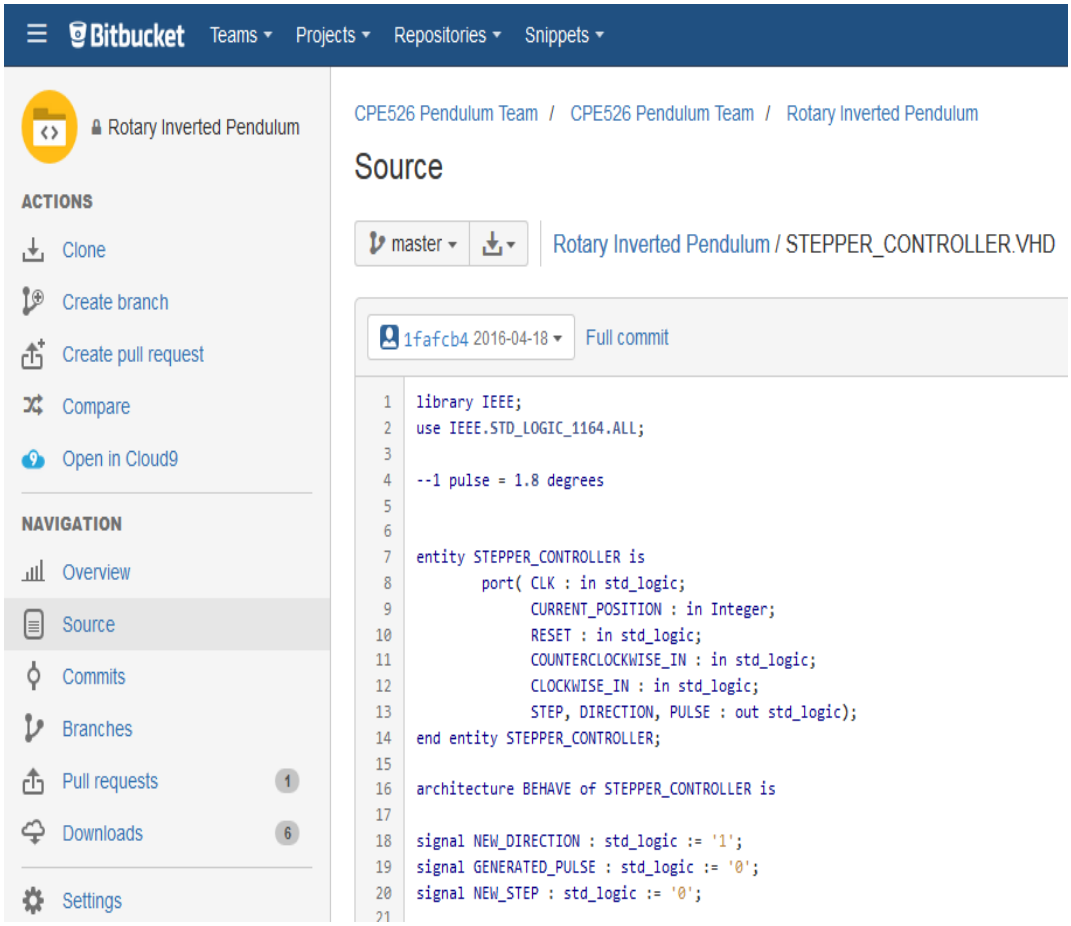


Figure 16 - Bit Bucket

Git is a revision control system that eases management of source code in a team work environment. A version history of all source modifications made during a project can be stored in a Git database.

Our VHDL source code was managed with a Git repository on BitBucket.org. BitBucket offers a web frontend that offers access to many features of Git and a limited text based development environment.

# 10. Simulation

## 10.1 Rotary Encoder

To verify the correct operation of the rotary encoder, a test bench was developed. The first revision of this test bench generated Phase A and B that would simulate only constant clockwise or constant counter clockwise rotation.

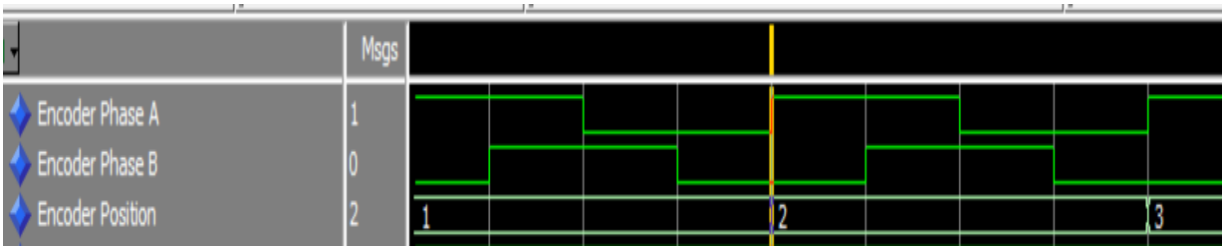


Figure 17- Clockwise Motion

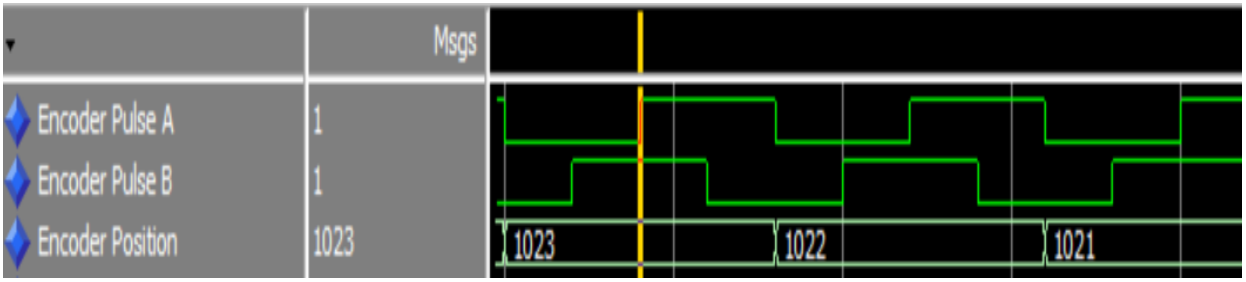


Figure 18 - Counterclockwise Motion

It was then extended to change direction. It was in this process that problems with the initial design were discovered.

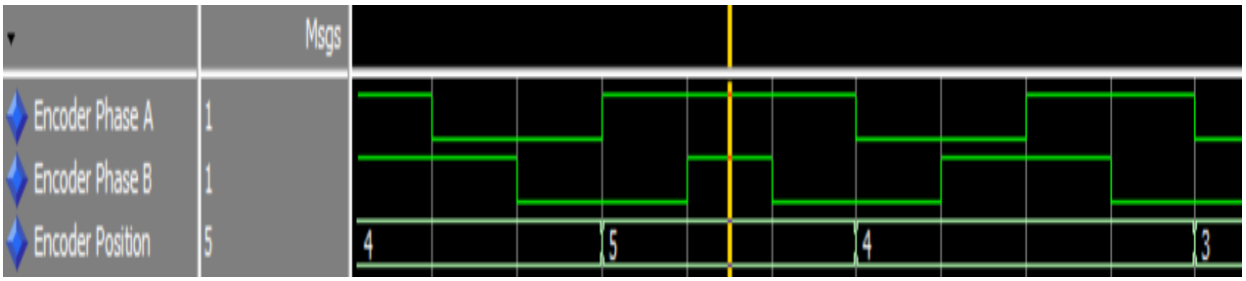


Figure 19 - Alternating Motion

As the simulated encoder changed direction, it was noticed that the position was not updated on the falling edge of the clock in the original design. The result was that the position accumulated error and drifted away from its initial zero point. To correct this issue discovered during the simulation phase, the three process design was written that corrected the issue. This design was able to run the test bench thousands of times and correctly return to its zero position at the end of every cycle.

## 10.2 Stepper Controller

In order to simulate the Stepper Controller a pulse generator was modified to supply a positional counter to simulate the rotary encoder input. On every rising edge of the clock signal the position would increment. This was used as an effective way to check the state transition logic

described by the design in Section 8.3. All state transition occurs on the rising edge of the clock while the state output occurs on the falling edge. Since the output is dependent on the state of the state machine, the Stepper Controller is considered a Moore State Machine. An asynchronous reset is utilized to initialize the state machine into a known state.

Figure 20, below, show the Stepper Controller with a high reset for the first 3 clock cycles. Upon the reset signal going low, the state logic will be evaluated at the first rising edge of the clock. The INIT_STATE becomes the current state and the next state to transition to is the SWING_UP state. The following rising edge of the clock transitions the state machine into the SWING_UP state and upon a few more sequential clock pulses the pulse out begins to toggle and the motor position will increment. Also note that the output pulse occurs on the falling edge of the input clock.

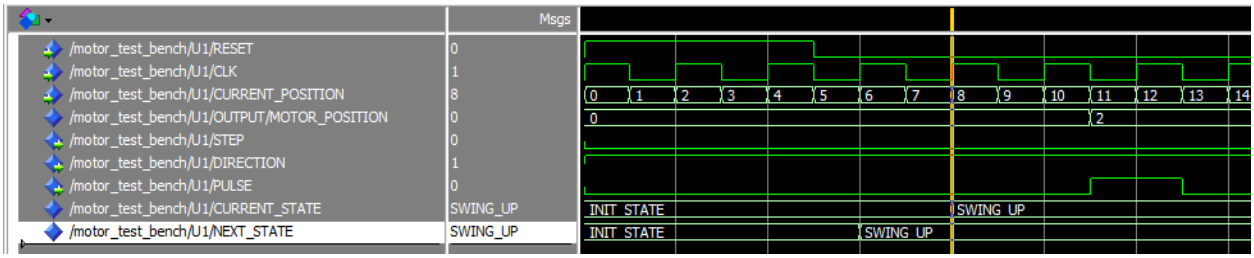


Figure 20 - State Transition to Swing Up

Figure 21 shows that once the defined boundary condition for the motor has been reached, the signal for direction toggles. In this instance a value of 1 is interpreted as counterclockwise by the big easy driver and 0 is clockwise. The position for the motor also decrements on every rising edge of the output pulse.

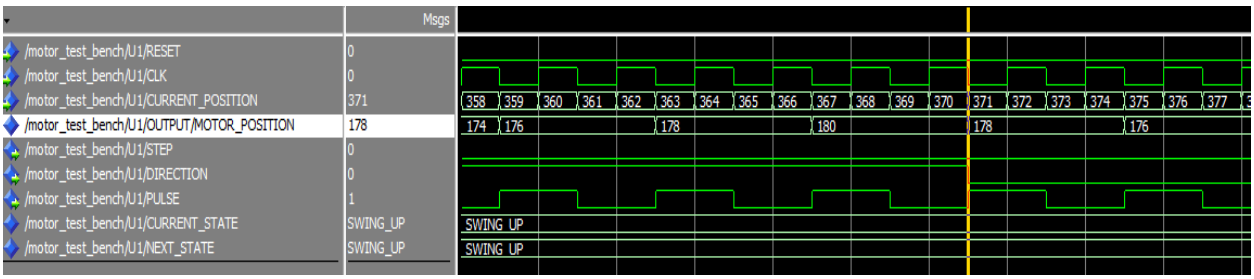


Figure 21 - Change in Motor Direction

Figure 22 shows the transition from the SWING_UP state to the BALANCING_ACT state. Before the transition occurs, the previous clock cycle evaluates the position of the rotary encoder input and sees that it has met the boundary conditions to transition. The transition then occurs on the next rising edge of the clock. While in the BALANCING_ACT state the step position changes from a logic 0 to a logic 1 to signify a half step input to the big easy driver. The motor position also begins to decrement/increment by a single tick.

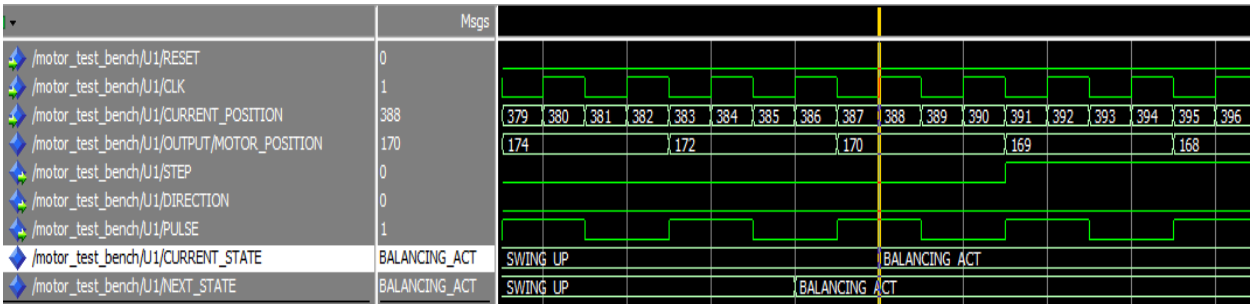


Figure 22 - State Change to Balancing Act

Figure 23 shows the transition into the balanced state. The boundary conditions have been met for the rotary encoder position. After the state machine has transitioned into this state the pulse out is forced to low so the motor will no longer turn. This is where the system has reached the state of equilibrium.

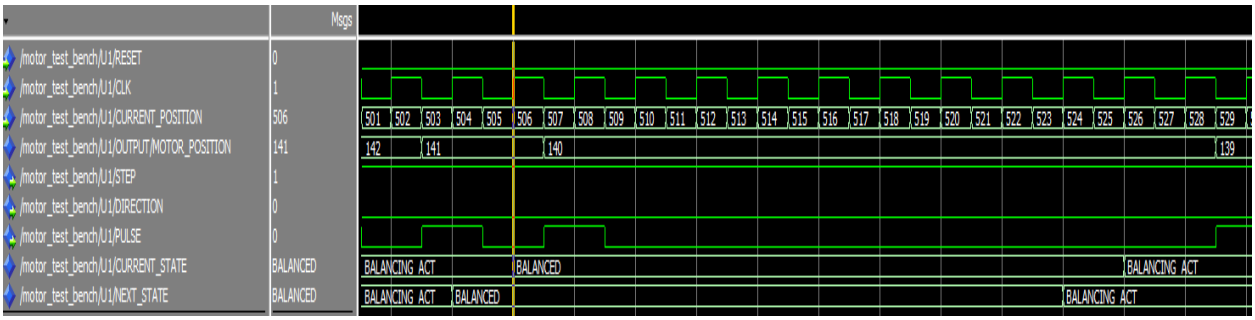


Figure 23 - State Changed to Balanced

# 11. Integration

Once the VHDL models for both the Rotary Encoder and Stepper Controller had been implemented and simulated it was time to integrate. A schematic approached was used for integration. The schematic gives a visual depiction of the VHDL entities as well as their inputs, outputs, and interconnections. Quartus can create a block diagram file for a given piece of VHDL. An instance of this block diagram is in essence an instance of the VHDL that was written. The block diagram can be inserted into a schematic and connected with other instances of VHDL such as the Rotary Encoder and the Stepper Controller. Figure 24 is the top level block diagram for the rotary inverted pendulum.

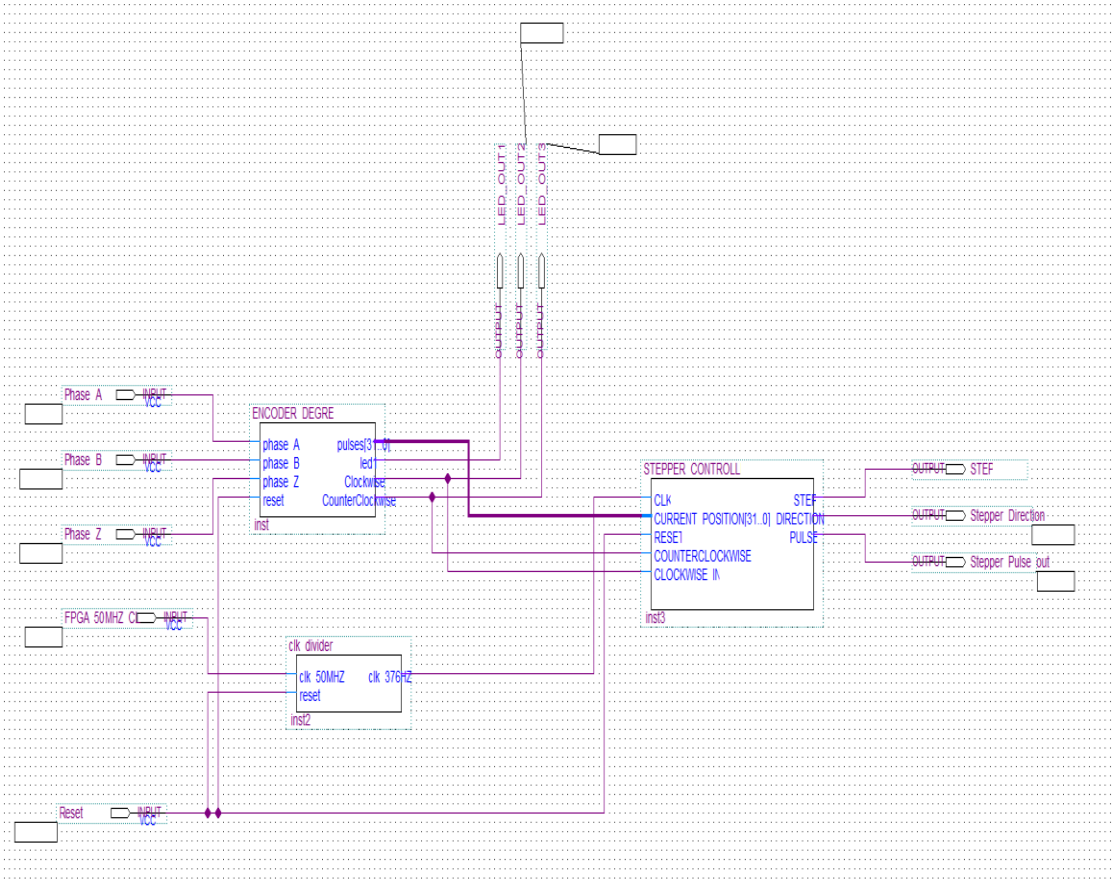


**Figure 24 - Top Level Block Diagram**

Once an input is placed on the schematic, in order for it to be associated with an element on the FPGA, a pin needs to be assigned to it. Pin assignments vary depending on the FPGA in use, but the data sheet will have the pins listed for GPIO, LEDs, Switches, etc. Figure 25 is an example of the Quartus Pin planner tool that was utilized in assigning the input and output pins for the Rotary Encoder and the Steper Controller for the feedback system.

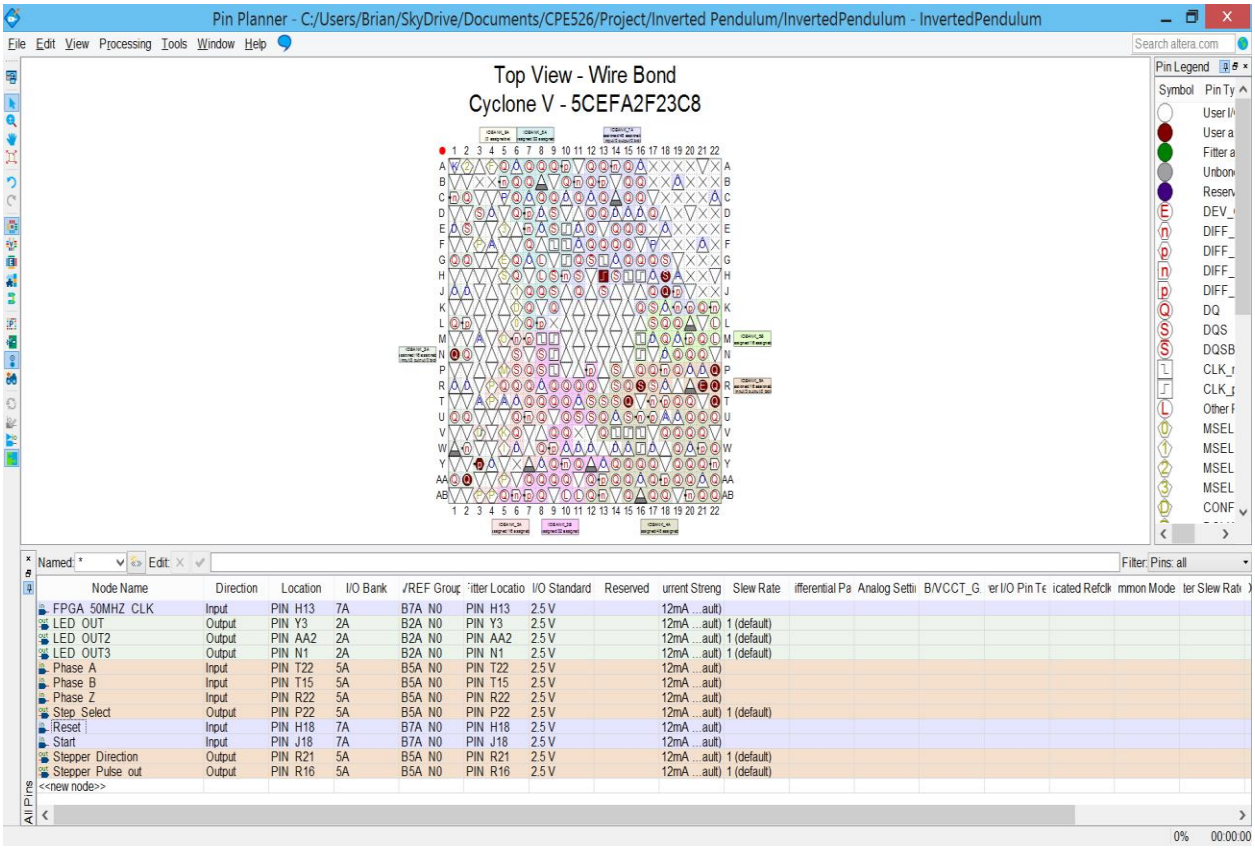


Figure 25 - Pin Assignments

# 12. Conclusion

As the semester ends, we find that our internal chaos is gradually fading into a peaceful state of equilibrium. Much like the inverted pendulum, there was an initial swing up state where we realized that we needed to work on the project, a balancing act state as we tried, failed, and tried again to maintain an understanding of VHDL, and finally a point of blissful equilibrium where we achieved our goals.

Overall, we have met all of the requirements that we set for ourselves and consider the design to be a success—that's not to say we couldn't do with a few improvements.

The fruits of our efforts may be viewed below:

https://www.youtube.com/watch?v=q4N4nYDXhlA

# 13. Data Sheets

ROB-09238 Stepper Motor

http://www.sparkfun.com/datasheets/Robotics/SM-42BYG011-25.pdf

COM-11102 Rotary Encoder - 1024 P/R (Quadrature)

http://cdn.sparkfun.com/datasheets/Robotics/E6B2Encoders.pdf

ROB-12859 Big Easy Driver

https://cdn.sparkfun.com/datasheets/Robotics/A4988-Datasheet.pdf

BeMicro CV Cyclone V

https://www.arrow.com/en/products/bemicrocv/arrow-development-tools

Terasic DE0

http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=364&FID=0c266381d75ef92a8291c5bbdd5b07eb