LUSHAY LABS

Stepper Motor Controller

Syeda Rafia Fizza Naveed
Department of Electrical (Computer Systems)
Usman Institute of Technology University, Karachi
syedarafiafizzanaveed@gmail.com

Abstract

This project revolves around the development of a smart stepper motor controller utilizing the capabilities of the Tang Nano 9k FPGA. The primary objective is to establish a highly precise motor control system, facilitating accurate positioning and motion management. This project report presents the design and implementation of a Stepper Motor Controller, a critical component in precision motion control systems. Stepper motors are employed in applications where accurate positioning, controlled movement, and reliability are essential. The project aims to develop a controller that enables users to command the stepper motor's direction, speed, and start/stop actions. The report outlines the design process, simulation, testing, results, and potential applications.

Lushay Labs has played a vital part in this project, using their expertise with the Tang Nano 9k FPGA. They've provided crucial resources and guidance throughout the project, including advanced tools, advice, and access to open-source projects. With their help, I'm developing a precise stepper motor controller.

Their commitment to sharing and innovation is enriching the project and making its development faster. Their collaboration has not only made the project possible but also helps in learning and promotes growth in FPGA-based engineering. At the end of the day, there'll be a cool motor controller ready to go, useful in many situations where precise movement is a must.

Keywords: *Stepper motor controller, Tang Nano 9k FPGA, precision control, motion management, Lushay Labs, RISC-V, collaboration, innovation, FPGA-based engineering, open-source projects, technological growth*

Introduction

The stepper motor is an electromechanical device; it converts electrical power into mechanical power. Stepper motors are widely used in industries such as manufacturing, robotics, and medical devices due to their ability to move precisely in discrete steps. However, effective control of these motors is crucial to harness their potential. This project addresses the need for a versatile and accurate Stepper Motor Controller.

This project aims to develop a comprehensive Stepper Motor Controller using Verilog as the hardware description language. The controller is designed to work with the Tang Nano 9k FPGA and interface with the ULN2003 motor driver and 28BYJ-48 stepper motor. It provides precise control over various parameters such as direction, speed, and motion initiation.

By combining Verilog programming, the Tang Nano 9k FPGA, the ULN2003 motor driver, and the 28BYJ-48 stepper motor, this integrated system enables accurate and controlled motion. This project contributes to the advancement of automation by introducing a holistic approach to precise movement control. The integration of advanced technologies in this field has the potential to revolutionize automation frameworks, improving accuracy and operational efficiency.

Hardware and Software Tool Requirements

- **Hardware: Tang Nano 9k FPGA**
  Using the Tang Nano 9k FPGA as our hardware tool. With its strong capabilities, it's the perfect platform to effectively achieve our project goals.

- **Software: VS Code**
  Developing the software aspect of the project using Visual Studio Code (VS Code), a widely used coding platform known for its versatility.

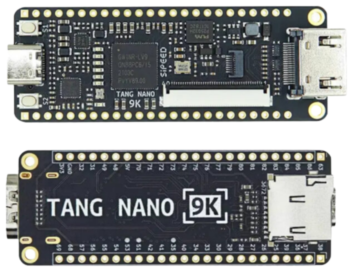
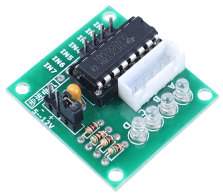
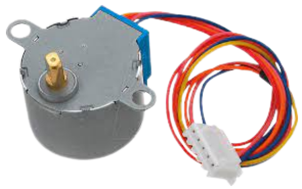
- **Extension: Lushay Code**
  Enhancing our toolkit, using the lushay lab extension of VS Code named “Lushay Code” with a complete open-source FPGA toolchain within VS Code. This extension improves the coding experience and offers extra features to streamline development.

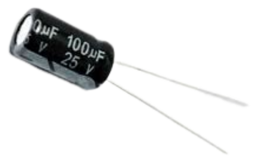
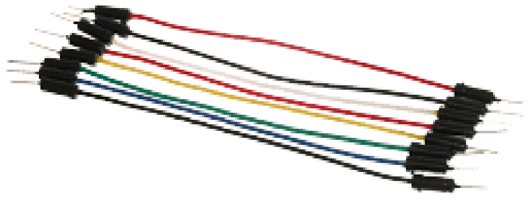
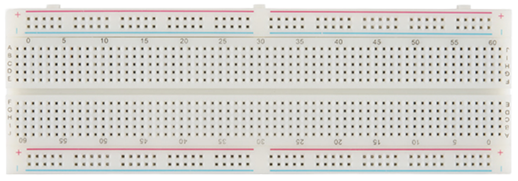
- **OpenSource Toolchain: OSS-CAD-Suite**
  Our project is tapping into the comprehensive OpenSource Toolchain, particularly the OSS-CAD-Suite. OSS-CAD-Suite is a project maintained by the yosys team which provides pre-built binaries for MacOSX, Windows and Linux.

- **Synthesis Tool: Yosys**
  For tasks like synthesis and optimization, using the dependable Yosys tool. Well-known in the FPGA community, Yosys is valued for its ability to transform code into optimized hardware structures.

- **Port Bitstream onto the FPGA: OpenFPGA Loader**
  Utilizing the openFPGALoader tool to program the final bitstream onto the FPGA.

Required Components

a.	Tang Nano 9k FPGA	
b.	ULN2003 Stepper Driver	
c.	Stepper Motor	
d.	Push Buttons	

e.	Capacitor (100uF)	
f.	Jumper Wires	
g.	Bread Board	

Methodology

The methodology applied in implementing the stepper motor controller encompasses a structured hardware description language-based strategy. The controller's behavior is defined using Verilog, outlining its responses to user inputs such as start/stop, direction, and speed adjustments. A clock-driven mechanism is adopted for synchronization, with logic governing step pulse generation and LED status. A comprehensive testbench is constructed for simulation, enabling rigorous testing of diverse scenarios. This methodology ensures a systematic and organized approach, facilitating the creation of a dependable and effective stepper motor controller for seamless integration with the Tang Nano 9k FPGA platform.

Features of Stepper Motor Controllers

Button 1: Start and Stop Functionality

Stepper motor controllers enable users to initiate and halt motor movement precisely. This feature is crucial for applications where controlled and synchronized movement is essential.

Bit 0: When this bit is set to 0, the motor starts moving.

Bit 1: When it's set to 1, the motor stops.

Button 2: Direction Control

Controllers allow users to dictate the direction in which the motor turns. By toggling between clockwise and counterclockwise movements, users can achieve the desired outcome.

Bit 0: When bit is set to 0, the motor rotates in a clockwise direction.

Bit 1: When it's set to 1, the motor rotates in a counterclockwise direction.

Button 3: Speed Control

The speed at which a stepper motor moves is controlled by the rate of pulse delivery. The controller adjusts this rate, allowing for both fast and slow movements, catering to various applications' needs.

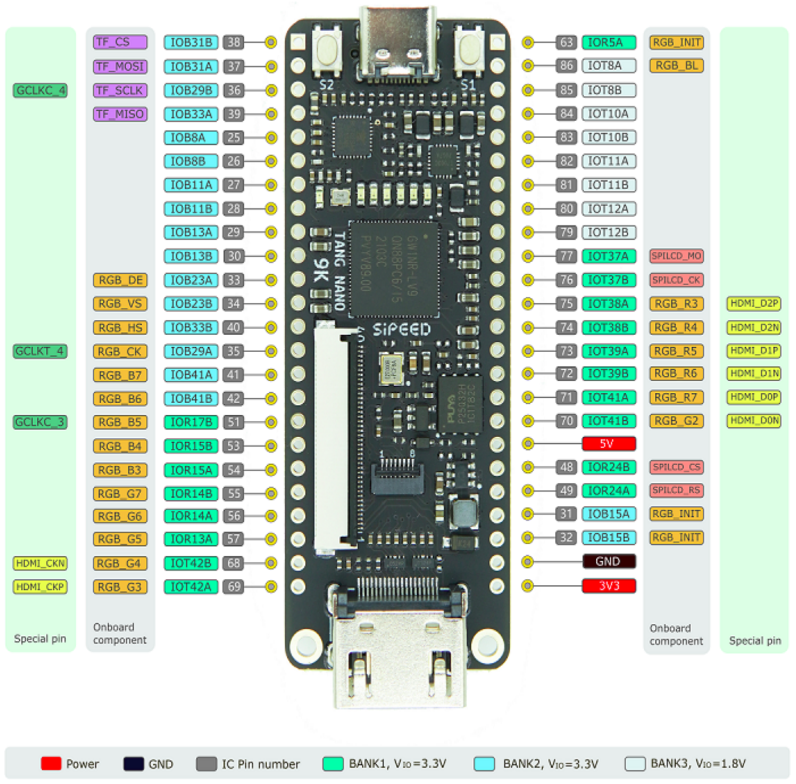
Bit 0: When bit is set to 0, the motor speed is high.

Bit 1: When it's set to 1, the motor speed is low.

Hardware Description

A. Tang Nano 9K FPGA

The Tang Nano 9K FPGA is a compact and versatile field-programmable gate array (FPGA) designed for various hardware implementation projects. It offers a powerful platform for digital system development, enabling users to create custom hardware solutions.

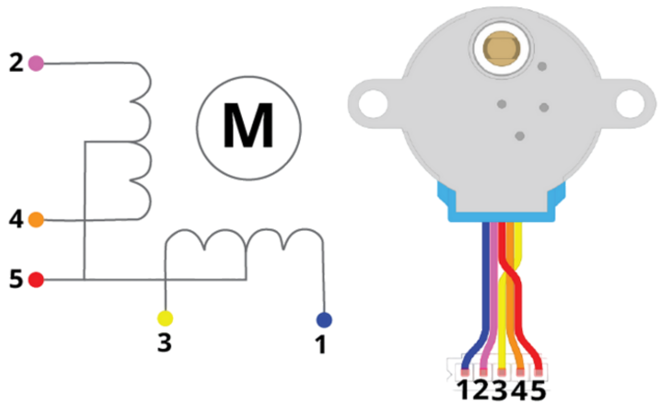


Specifications:

- Abundant logic cells for complex digital designs
- Multiple input and output ports for external connectivity
- Resources for precise clock signal control and synchronization
- Field-programmable for customization and reconfiguration
- Small form factor suitable for embedded and space-constrained applications
- Compatibility with various development environments and programming languages
- Robust processing capabilities for diverse applications
- Efficient operation with minimal power consumption
- Supported by a rich ecosystem of development tools and resources
- Offers a balance of features, performance, and affordability

B. 28BYJ-48 Stepper Motor

The 28BYJ-48 Stepper Motor is a low-cost, 5-volt DC motor known for its ability to move in precise steps. It has four coils of wire that are activated in a specific sequence to create rotational motion. This motor is commonly used in applications like controlling blinds and air conditioning units. It achieves good torque for its size due to a gear reduction ratio of around 64:1, allowing it to rotate at approximately 15 RPM. The label "28BYJ-48" encompasses various models with different voltage ratings, winding resistances, and gearing options. According to the datasheet, the recommended driving method is the half-step technique, involving eight distinct signals.



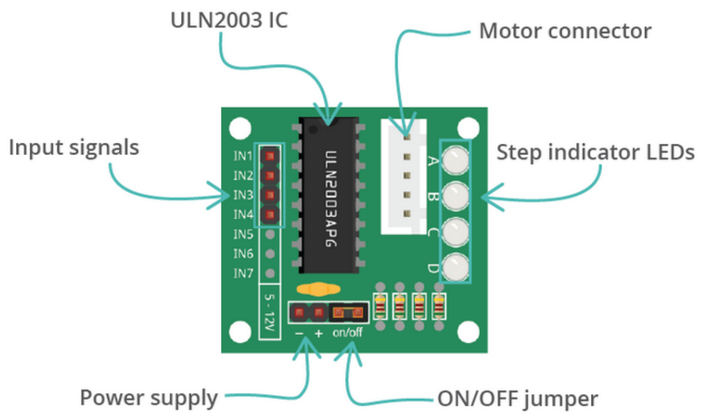
Specifications:

- Operating Voltage: 5 volts DC
- Number of Phases: 4
- Reduction Ratio: 1/64
- Step Angle: 5.625 degrees per step
- Operating Frequency: 100 Hertz

- DC Resistance: 50 ohms with a tolerance of ±7% at 25 degrees Celsius
- Idle Pull-In Frequency: Greater than 600 Hertz
- Idle Pull-Out Frequency: Greater than 1000 Hertz
- Pull-In Torque: Greater than 34.3 milliNewton meters at 120 Hertz
- Self-Positioning Torque: Greater than 34.3 milliNewton meters
- Friction Torque: Ranges from 600 to 1200 gram-force centimeters
- Pull-In Torque: 300 gram-force centimeters
- Insulation Resistance: Greater than 10 megaohms at 500 volts
- Insulation Withstand Voltage: 600 volts AC, 1 milliampere, 1 second
- Temperature Rise: Less than 40 Kelvin at 120 Hertz
- Noise Level: Less than 35 decibels at 120 Hertz, no load, at a distance of 10 centimeters

ULN2003 Motor Driver

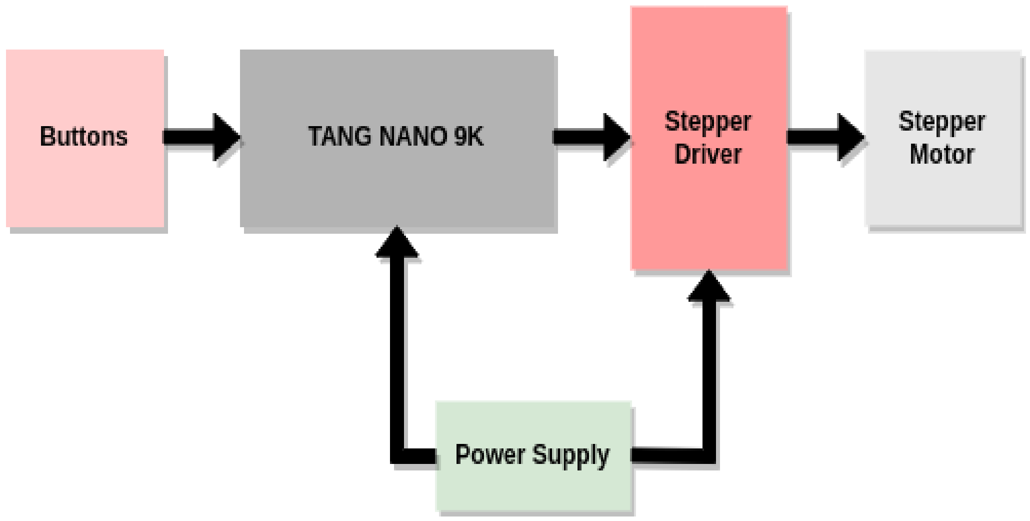
The ULN2003 is a reliable array of Darlington transistors with high voltage and current. It consists of seven sets of NPN Darlington pairs, each of which can successfully handle inductive loads and is supplied with a common-cathode clamp diode. The current rating of a single Darlington pair is 500mA, and these pairs can be connected in parallel to increase current capacity even further. Driving relays, hammers, lamps, screens (including LED gas displays), line drivers, and logic buffers are a few examples of typical uses. For each Darlington pair, the ULN2003 comes with a 2.7k series base resistor to make it easier to work with TTL or 5V CMOS devices.



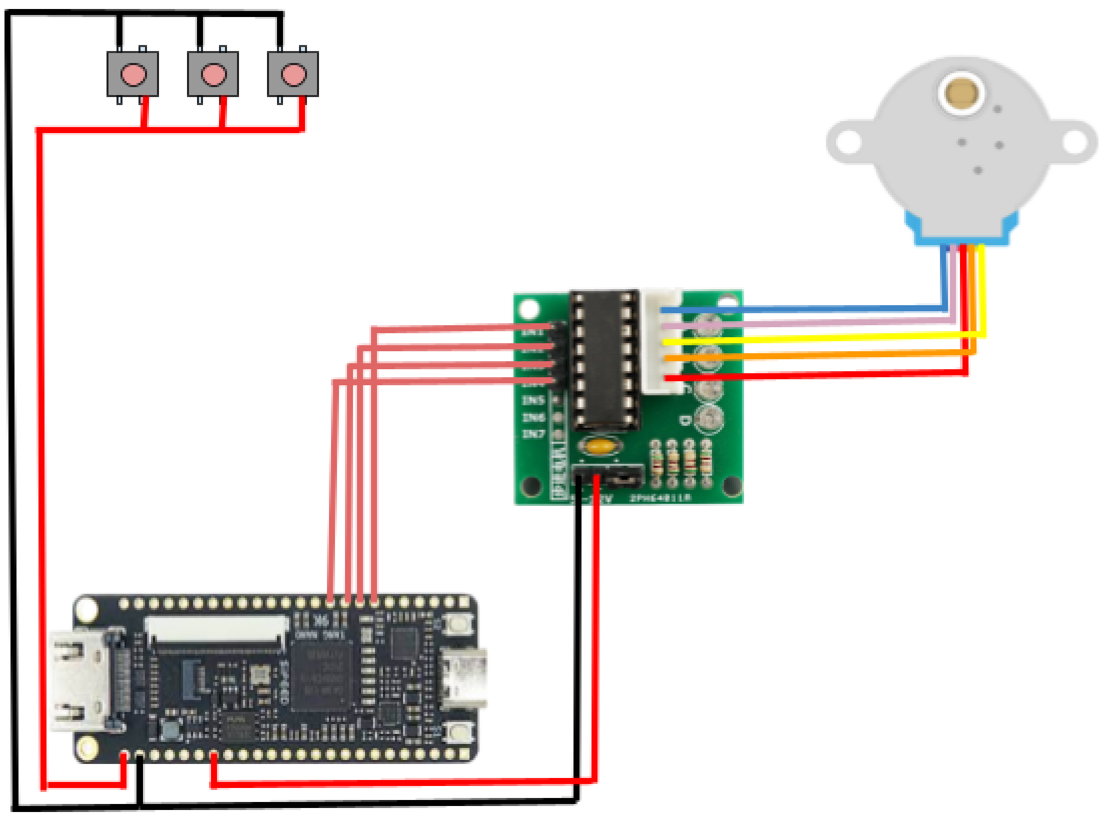
Specifications:

- Capable of handling a collector current of 500mA for each individual output.
- Can provide high-voltage outputs of up to 50 volts.
- Compatible with different types of logic inputs.
- Suitable for applications involving the control of relays.

Block Diagram



Circuit Diagram



Code

Design

```
module top(
   input wire clk,                     // Clock signal
   input wire btn_start_stop,          // Button 1: Start/Stop motor
   input wire btn_direction_control, // Button 2: Direction control (0 -
Clockwise, 1 - Anti-clockwise)
   input wire btn_speed,               // Button 3: Speed control (0 - Speed
[high], 1 - Speed [low])

   // step signals to generate pulses
   output wire in1,
   output wire in2,
   output wire in3,
   output wire in4,
   // leds to mapped with buttons
   output wire led,
   output wire led1,
   output wire led2,
   output wire led3
   );
```

This is the module definition named "***top,***" which encapsulates the functionality of the stepper motor controller. It has input ports for clock signal **(clk),** buttons for start/stop motor control **(btn_start_stop)**, direction control **(btn_direction_control)**, and speed control **(btn_speed)**. Additionally, it has output ports for generating step pulses **(in1, in2, in3, in4)**, as well as LEDs mapped to the buttons **(led, led1, led2, led3)**.

```
reg [14:0] counter_speed1 = 0;     // 15 bits counter for high speed
reg [19:0] counter_speed2 = 0;     // 20 bits counter for low

reg led_state = 0;                // led_state variable for counter
reg  [3:0]motor_out  =  0;               //  motor_out  variable  for  clockwise  and
anti-clockwise direction
reg [2:0]step = 0;                // step variable for direction signal
```

Here, various registers are declared to store and manage different aspects of the stepper motor control. **counter_speed1** and **counter_speed2** are counters used for controlling the motor speed. **led_state** is a variable used to toggle the LED state. **motor_out** is a variable used to control the motor output pattern, and **step** is used to keep track of the current step in the motor sequence.

```
always @(posedge clk) begin
   if(btn_speed) begin
       if (counter_speed1 == 1000) begin
           counter_speed1 <= 0;
           led_state <= ~led_state; // Toggle the LED state
       end
       else begin
           counter_speed1 <= counter_speed1 + 1;
       end
   end
   else begin
       if (counter_speed2 == 67500) begin
           counter_speed2 <= 0;
           led_state <= ~led_state; // Toggle the LED state
       end
       else begin
           counter_speed2 <= counter_speed2 + 1;
       end
   end
end
```

This **always** block triggers on the positive edge of the clock signal. Depending on the value of the **btn_speed button,** either **counter_speed1** or **counter_speed2** is incremented. When the counter reaches a certain value (1000 for **counter_speed1** and 67500 for **counter_speed2**), it resets and toggles the **led_state**, which is then used to control an LED.

```
  assign led = (counter_speed1 < 1000) ? led_state : 1'b0;
```

The value of the **led** output is determined by the c**ounter_speed1** value. If **counter_speed1** is less than 1000, **led** is set to the **led_state** value (toggled LED), otherwise, it is set to logic low (**1'b0**).

```
   always @(posedge led_state)
   begin
       if(btn_start_stop == 1)begin
           if(step < 3'd5)begin
               if(btn_direction_control)begin
                   case(step)
```

```
                        0 : motor_out <= 4'b0001;     // Motor output for step 0
(Clockwise direction)
                        1 : motor_out <= 4'b0010;     // Motor output for step 1
(Clockwise direction)
                        2 : motor_out <= 4'b0100;     // Motor output for step 2
(Clockwise direction)
                        3 : motor_out <= 4'b1000;     // Motor output for step 3
(Clockwise direction)
                   endcase
               end
               else begin
                   case(step)
                        0 : motor_out <= 4'b1000;     // Motor output for step 0
(Anticlockwise direction)
                        1 : motor_out <= 4'b0100;     // Motor output for step 1
(Anticlockwise direction)
                        2 : motor_out <= 4'b0010;     // Motor output for step 2
(Anticlockwise direction)
                        3 : motor_out <= 4'b0001;     // Motor output for step 3
(Anticlockwise direction)
                    endcase
                end
               step <= step + 1;          // Increment step counter
           end
            else begin
                step <= 'b0;              // Reset step counter
            end
       end
        else begin
            motor_out <= 'b0;              // Turn off motor output
        end
   end
```

This **always** block triggers on the positive edge of the **led_state**. It contains the main motor control logic. Depending on the **btn_start_stop button**, the motor control logic generates the appropriate motor output **(motor_out)** based on the current **step** value. The step counter is incremented to move through the motor sequence. When **step** reaches a certain point, it resets to 0.

```
   assign in1 = motor_out[0];              // Assign motor output 4
   assign in2 = motor_out[1];              // Assign motor output 2
   assign in3 = motor_out[2];                // Assign motor output 3
   assign in4 = motor_out[3];                // Assign motor output 4
    assign led1 = btn_start_stop;          // Assign start/stop button input to
LED 1 output
    assign led2 = btn_direction_control;     // Assign direction control button
input to LED 2 output
    assign led3 = btn_speed;                 // Assign speed button input to LED
3 output
endmodule
```

These **assign** statements connect the appropriate signals to their corresponding outputs. Motor outputs **in1**, **in2**, **in3**, and **in4** are assigned based on the **motor_out** variable. LEDs **led1**, **led2**, and **led3** are directly assigned to the corresponding button inputs.

Testbench

```
module test;
   reg clk;
   reg btn_start_stop;
   reg btn_direction_control;
   reg btn_speed;
   wire in1;
   wire in2;
   wire in3;
   wire in4;
   wire led;
   wire led1;
   wire led2;
   wire led3;

   //Instantiate the Top Module
   top top_i (
       .clk(clk),
       .btn_start_stop(btn_start_stop),
       .btn_direction_control(btn_direction_control),
       .btn_speed(btn_speed),
       .in1(in1),
       .in2(in2),
       .in3(in3),
       .in4(in4),
       .led(led),
       .led1(led1),
       .led2(led2),
       .led3(led3)
   );
```

In this module named **"test,"** the ports required for simulation are declared. These ports include clock signal **(clk),** buttons for start/stop **(btn_start_stop)**, direction control (btn_direction_control), and speed control **(btn_speed)**, as well as the various motor control signals, LED outputs, and step pulse outputs.
An instance of the top module is instantiated within this **test** module, connecting the declared ports with the ports of the main **top** module.

```
    // Initialize signals
    initial begin
        clk = 0;
        btn_start_stop = 0;
        btn_direction_control = 0;
        btn_speed = 0;
        #10;

        // Set start/stop button to 1
        btn_start_stop = 1;
        btn_direction_control = 0;
        btn_speed = 0;
        #3000000;

        // Set direction control button to 1
        btn_start_stop = 1;
        btn_direction_control = 1;
        btn_speed = 0;
        #3000000;

        // Set speed button to 1
        btn_start_stop = 1;
        btn_direction_control = 1;
        btn_speed = 1;
        #1000000;

        $finish;
```

```
   end
```

The **initial** block initializes the simulation signals. The clock **clk** is initially set to 0. After a delay of 10 time units **(#10)**, the sequence of button settings is applied to simulate user interactions:

1. The **btn_start_stop** button is set to 1, simulating the motor starting.
2. After a delay of 3000000 time units **(#3000000)**, the **btn_direction_control** button is set to 1, simulating a change in motor direction.
3. After another delay of 3000000 time units, the **btn_speed button** is set to 1, simulating a change in motor speed.

Finally, after a delay of 1000000 time units **(#1000000)**, the simulation is finished using **$finish**.

```
   // Toggle clock every 2 time units
   always #2 clk = ~clk;
```

This **always** block generates a clock signal with a period of 4 time units, effectively toggling the **clk** signal every 2 time units.

```
   // Dump waveform to VCD file
   initial begin
   $dumpfile("smc.vcd");
   $dumpvars(0, test);
 end
endmodule
```

This section sets up the dumping of simulation waveforms to a VCD (Value Change Dump) file named "smc.vcd." The **$dumpfile** command specifies the name of the VCD file, and **$dumpvars** command specifies the scope of variables to be included in the VCD. In this case, it includes all variables within the **test** module.

.cst File

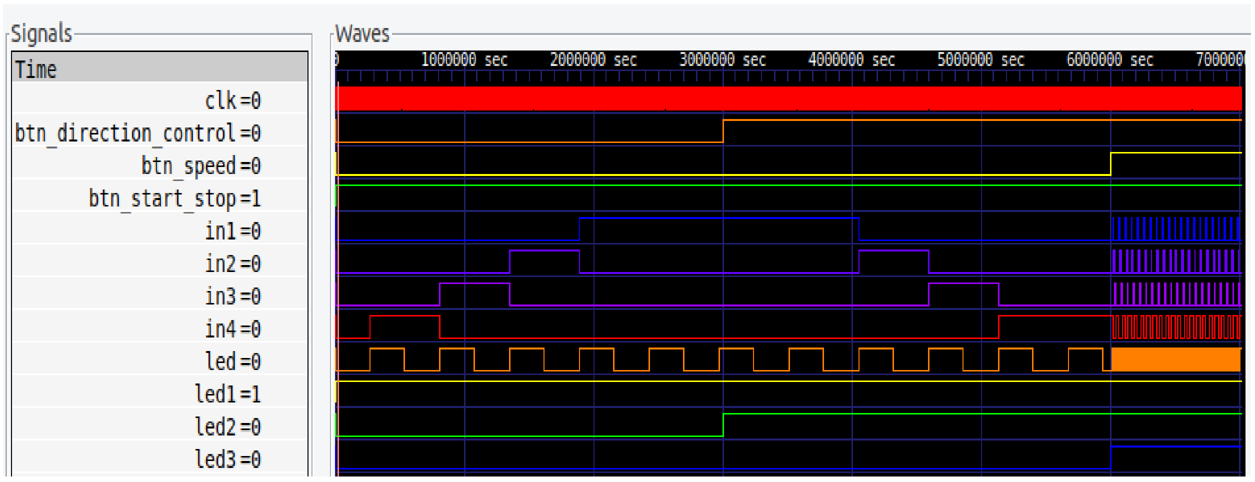
```
IO_LOC  "in4" 30;
IO_PORT "in4" IO_TYPE=LVCMOS33;

IO_LOC  "in3" 29;
IO_PORT "in3" IO_TYPE=LVCMOS33;

IO_LOC  "in2" 28;
IO_PORT "in2" IO_TYPE=LVCMOS33;

IO_LOC  "in1" 27;
IO_PORT "in1" IO_TYPE=LVCMOS33;
```

```
IO_LOC  "btn_speed" 40;
IO_PORT "btn_speed" IO_TYPE=LVCMOS33;

IO_LOC  "btn_direction_control" 34;
IO_PORT "btn_direction_control" IO_TYPE=LVCMOS33;

IO_LOC  "btn_start_stop" 33;
IO_PORT "btn_start_stop" IO_TYPE=LVCMOS33;

IO_LOC  "clk" 52;
IO_PORT "clk" IO_TYPE=LVCMOS33 PULL_MODE=UP;

IO_LOC  "led" 10;
IO_PORT "led" DRIVE=8 IO_TYPE=LVCMOS18;

IO_LOC  "led1" 11;
IO_PORT "led1" DRIVE=8 IO_TYPE=LVCMOS18;

IO_LOC  "led2" 13;
IO_PORT "led2" DRIVE=8 IO_TYPE=LVCMOS18;

IO_LOC  "led3" 14;
IO_PORT "led3" DRIVE=8 IO_TYPE=LVCMOS18;
```

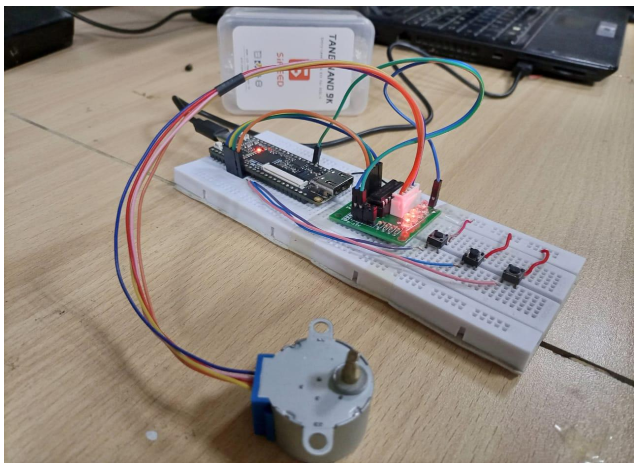
Simulation and Synthesis

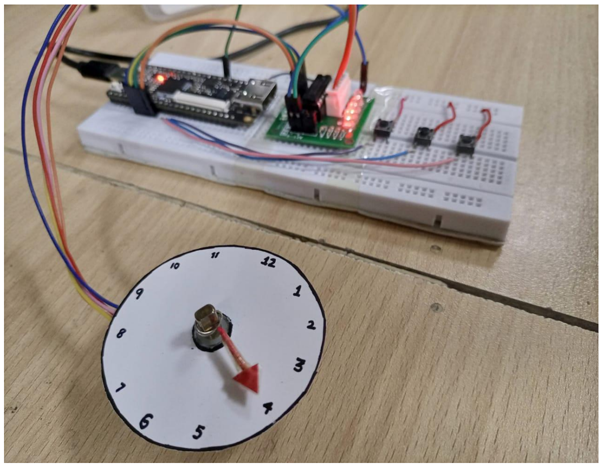
Test Environment

Firstly, we compiled the RTL code and the test bench together and found no error. After the successful compilation the RTL code is simulated using a test bench. The compilation and simulation is done successfully using iVerilog. The RTL code is then synthesized using Tang Nano 9k FPGA and the desired output is obtained successfully without any error.



Result





Conclusion

The final output of the test bench shows that it was highly successful to develop the Stepper Motor Controller using Verilog on the Tang Nano 9k FPGA along with the ULN2003 motor driver and the 28BYJ-48 stepper motor. LED indicators validated the synchronization between user inputs and motor action, and simulation results showed the correct translation of control signals into motor movement. The project's design objectives of directional control, variable speed, and accurate motion start and stop were all met. This project shows how integrated systems can be used to provide precise motion control, with real-world applications in automation and robotics.

References

[1] Lushay Labs. (2023, August 19). Lushay Labs. https://learn.lushaylabs.com/

[2] Tang Nano 9K: Getting Setup. (2023, August 19). Tang Nano 9K: Getting Setup. https://learn.lushaylabs.com/getting-setup-with-the-tang-nano-9k/

[3] Tang Nano Examples - Sipeed Wiki. (2023, August 19). Tang Nano Examples - Sipeed Wiki. https://wiki.sipeed.com/hardware/en/tang/Tang-Nano-Doc/examples.html

[4] Tang Nano 9K Examples. (2023, August 19). Tang Nano 9K Examples. https://github.com/lushaylabs/tangnano9k-series-examples

[5] Y. Weihua, C. Lan, X. Mogen and H. Yusheng, "Design of a Step-Motor Control System Based on FPGA," 2007 8th International Conference on Electronic Measurement and Instruments, Xi'an, China, 2007, pp. 4-874-4-877, doi: 10.1109/ICEMI.2007.4351282.

BIOGRAPHIES OF AUTHOR

Syeda Rafia Fizza Naveed, a graduate in Computer Systems Engineering from Usman Institute of Technology in the year 2023. Currently working as a research intern at MicroElectronics Research Lab (MERL). I have enhanced my skills using tools like Vivado, Logisim, Verilator, AWS-FPGA, and more. Furthermore, I have been actively involved in RISC-V based projects, showcasing my practical skills. In the span of the past two years, I have undertaken impactful research projects, including the NOVA1 SoC and a Multicore SoC for SMP Linux, which have afforded me hands-on experience in implementing innovative RISC-V technology solutions. I'm motivated to make significant contributions in the domain of computer systems engineering by my constant dedication to ongoing learning, research and development.