KOCAELİ UNIVERSITY ENGINEERING FACULTY

STEPPER MOTOR
PMOD REFERENCE
DOCUMENT

FURKAN YARDIMCI
2021-2022

DEPARTMENT: Electronic and Communacition Engineering
TEACHER: Associate Profesor ANIL ÇELEBİ

KOCAELİ, 2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PURPOSE:

The purpose of this document is to provide a document containing the necessary features about the design and use of stepper motor pmod, prepared by Kocaeli University Students FURKAN YARDIMCI and ASLINUR SERİN.

<u>İNTRO/OVERWİEV:</u>

The pmod stepper motor module provides the required number of headers to drive the 4-pin bipolar and 6-pin unipolar stepper motors with the Basys-3 board. Stepper motors are structures that contain magnets and coils that polarize this magnet to provide movement. When electric current passes through the coils in stepper motors, the motor creates torque due to the magnetic field created by this current. Each movement of this motor is called a step. The inputs of the Basys-3 card cannot provide this current required for the step motor to work. With the current/power amplifier circuit consisting of LM293D and mosfets on the pmod stepper module, it increases the low current it receives from the Basys-3 card to the levels that required for driving the stepper motor. In this way, the stepper motor is driven.

<u>FEATURES:</u>

- Stepper motor driver for 4 and 6-pin motors
- Can drive both motors simultaneously
- Multiple LEDs to indicate signal propagation
- Jumper for optional external power
- 2×6-pin pmod connector with GPIO interface

<u>CONNECTİONS/INTERFACİNG:</u>

The Pmod STEP communicates with the host board via the GPIO (General-purpose input/output) protocol. Microcontrollers usually include GPIOs. Depending on the application, a microcontroller's GPIOs may comprise its primary interface to external circuitry or they may be just one type of I/O used among several, such as analog signal I/O, counter/timer, and serial communication.

This Pmod offers headers for both 4-pin and 6-pin stepper motors. 4-pin stepper motors only work in the bipolar configuration, requiring that the two inputs on each electromagnetic coil are brought to the correct logic level voltages to induce current flow in the correct direction. The 6-pin stepper motor header on this Pmod can be oriented for either bipolar or unipolar configuration. The two extra pins on this header provide two positive power pins as a source of current for when an input on one end of a coil is driven to a logic low voltage level.

SM PMOD Reference Designator	Function
P2	This pin 2x6, right-angle header provides VCC, GND and (8) digital 1/0 signals from the Basy3's microcontroller to the SM PMOD through any of the Pmod connector ports on the Basys-3.

P3	4-wire stepper motor interface connector.
P4	6-wire stepper motor interface connector.
P9	This header provides access to pins 1-4 of the SM PMOD header J1 (labeled SIG1-SIG4 on the SM PMOD schematic).
P10	This header provides access to pins 5-8 of the SM PMOD header J1 (labeled SIG5-SIG8 on the SM PMOD schematic).
P7	GND
P8	GND
P6	External power screw terminal block. Provide power to the SM PMOD from an external source rather than from the Basys-3 board. NOTE: Pins 1 and 3 of P5 must be shorted (pins 2 and 3 of P5 open) to power the SM PMOD from an external source.
J1	External power measurement point. Use P7 or P8 (GND) on SM PMOD as the return to measure the voltage of the external power applied to the board.
P5	Power selection jumper. Jumper (short) pins 1 and 2 of P5 to supply power from the external power screw terminal block P6 or jumper (short) pins 2 and 3 of P5 to supply power from the Basys-3 board.
P1	Digital I/O signal state indication LED power jumper. Install the jumper (short pins 1 and 2) on P1 to provide power to bias the digital 1/0 signal state indication LEDs. Remove the jumper to reduce power consumption of board for battery powered applications.

TABLE 1.

STEPPER MOTOR CONTROL:

Stepper Motor Drive Sequence Code							
	Binary Value				Coil		
Decimal Value	A	B	C	D	AB	CD	Current Direction
8	1	0	0	0	ON	OFF	A -> B
10	1	0	1	0	ON	ON	A -> B, C -> D
2	0	0	1	0	OFF	ON	C -> D
6	0	1	1	0	ON	ON	B -> A, C ->D
4	0	1	0	0	ON	OFF	B -> A
5	0	1	0	1	ON	ON	B -> A, D -> C
1	0	0	0	1	OFF	ON	D -> C
9	1	0	0	1	ON	ON	A -> B, D -> C
8	1	0	0	0	ON	OFF	A -> B
10	1	0	1	0	ON	ON	A -> B, C -> D
2	0	0	1	0	OFF	ON	C -> D
6	0	1	1	0	ON	ON	B -> A, C ->D
4	0	1	0	0	ON	OFF	B -> A
5	0	1	0	1	ON	ON	B -> A, D -> C
1	0	0	0	1	OFF	ON	D -> C
9	1	0	0	1	ON	ON	A -> B, D -> C

TABLE 2

VERİLOG HDL:

We used a state machine to evaluate multiple situations, as it can give various outputs according to the polarization status of the stepper motor coils. The state diagram is as follows.

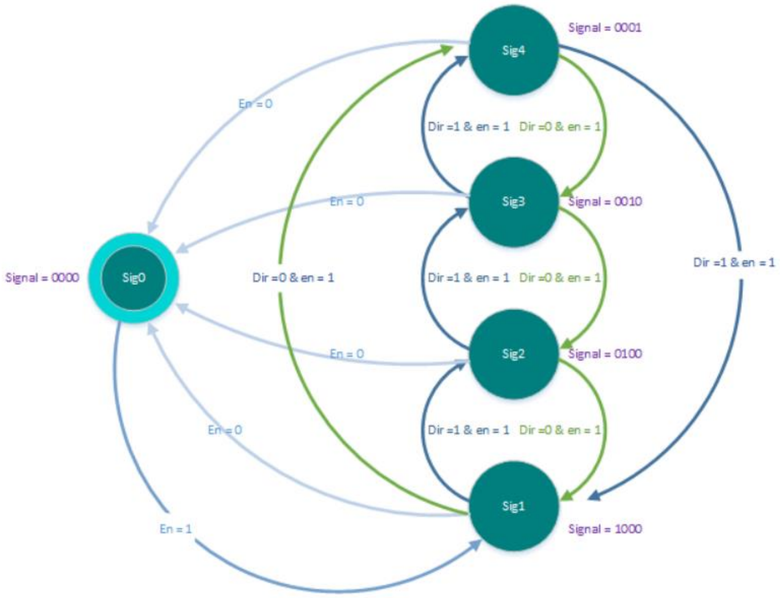

Image 1

<u>CODE:</u>

PMOD_STEP_DRIVER.V:

```
`timescale 1ns / 1ps

module pmod_step_driver(
    input rst,
    input dir,
    input clk,
    input en,
    output reg [3:0] signal
    );

    // local parameters that hold the values of
    // each of the states. This way the states
    // can be referenced by name.
    localparam sig4 = 3'b001;
    localparam sig3 = 3'b011;
    localparam sig2 = 3'b010;
    localparam sig1 = 3'b110;
    localparam sig0 = 3'b000;

    // register values to hold the values
    // of the present and next states.
    reg [2:0] present_state, next_state;

    // run when the present state, direction
    // or enable signals change.
    always @ (present_state, dir, en)
    begin
        // Based on the present state
        // do something.
        case(present_state)
        // If the state is sig4, the state where
        // the fourth signal is held high.
```

```
        sig4:
        begin
            // If direction is 0 and enable is high
            // the next state is sig3. If direction
            // is high and enable is high
            // next state is sig1. If enable is low
            // next state is sig0.
            if (dir == 1'b0 && en == 1'b1)
                next_state = sig3;
            else if (dir == 1'b1 && en == 1'b1)
                next_state = sig1;
            else
                next_state = sig0;
        end
        sig3:
        begin
            // If direction is 0 and enable is high
            // the next state is sig2. If direction
            // is high and enable is high
            // next state is sig4. If enable is low
            // next state is sig0.
            if (dir == 1'b0&& en == 1'b1)
                next_state = sig2;
            else if (dir == 1'b1 && en == 1'b1)
                next_state = sig4;
            else
                next_state = sig0;
        end
        sig2:
        begin
            // If direction is 0 and enable is high
            // the next state is sig1. If direction
            // is high and enable is high
            // next state is sig3. If enable is low
            // next state is sig0.
            if (dir == 1'b0&& en == 1'b1)
                next_state = sig1;
            else if (dir == 1'b1 && en == 1'b1)
                next_state = sig3;
            else
                next_state = sig0;
        end
        sig1:
        begin
            // If direction is 0 and enable is high
            // the next state is sig4. If direction
            // is high and enable is high
            // next state is sig2. If enable is low
            // next state is sig0.
            if (dir == 1'b0&& en == 1'b1)
                next_state = sig4;
            else if (dir == 1'b1 && en == 1'b1)
                next_state = sig2;
            else
                next_state = sig0;
        end
        sig0:
        begin
            // If enable is high
            // the next state is sig1.
            // If enable is low
```

```
                // next state is sig0.
                if (en == 1'b1)
                    next_state = sig1;
                else
                    next_state = sig0;
            end
            default:
                next_state = sig0;
            endcase
        end

        // State register that passes the next
        // state value to the present state
        // on the positive edge of clock
        // or reset.
        always @ (posedge clk, posedge rst)
        begin
            if (rst == 1'b1)
                present_state = sig0;
            else
                present_state = next_state;
        end

        // Output Logic
        // Depending on the state
        // output signal has a different
        // value.
        always @ (posedge clk)
        begin
            if (present_state == sig4)
                signal = 4'b1000;
            else if (present_state == sig3)
                signal = 4'b0100;
            else if (present_state == sig2)
                signal = 4'b0010;
            else if (present_state == sig1)
                signal = 4'b0001;
            else
                signal = 4'b0000;
        end
endmodule
```

CLOCK_DIV.V:

```
`timescale 1ns / 1ps

module clock_div(
    input clk,
    input rst,
    output reg new_clk
    );

    // The constant that defines the clock speed.
    // Since the system clock is 100MHZ,
    // define_speed = 100MHz/(2*desired_clock_frequency)
    localparam define_speed = 26'd5000000;

    // Count value that counts to define_speed
    reg [25:0] count;

    // Run on the positive edge of the clk and rst signals
    always @ (posedge(clk),posedge(rst))
    begin
        // When rst is high set count and new_clk to 0
        if (rst == 1'b1)
        begin
            count = 26'b0;
            new_clk = 1'b0;
        end
        // When the count has reached the constant
        // reset count and toggle the output clock
        else if (count == define_speed)
        begin
            count = 26'b0;
            new_clk = ~new_clk;
        end
        // increment the clock and keep the output clock
        // the same when the constant hasn't been reached
        else
        begin
            count = count + 1'b1;
            new_clk = new_clk;
        end
    end
endmodule
```

PMOD_STEP_INTERFACE.V:

```
`timescale 1ns / 1ps

module pmod_step_interface(
    input clk,
    input rst,
    input direction,
    input en,
    output [3:0] signal_out
    );

    // Wire to connect the clock signal
    // that controls the speed that the motor
    // steps from the clock divider to the
    // state machine.
    wire new_clk_net;

    // Clock Divider to take the on-board clock
    // to the desired frequency.
    clock_div clock_Div(
        .clk(clk),
        .rst(rst),
        .new_clk(new_clk_net)
        );

    // The state machine that controls which
    // signal on the stepper motor is high.
    pmod_step_driver control(
        .rst(rst),
        .dir(direction),
        .clk(new_clk_net),
        .en(en),
        .signal(signal_out)
        );

endmodule

```

SCHEMATIC:

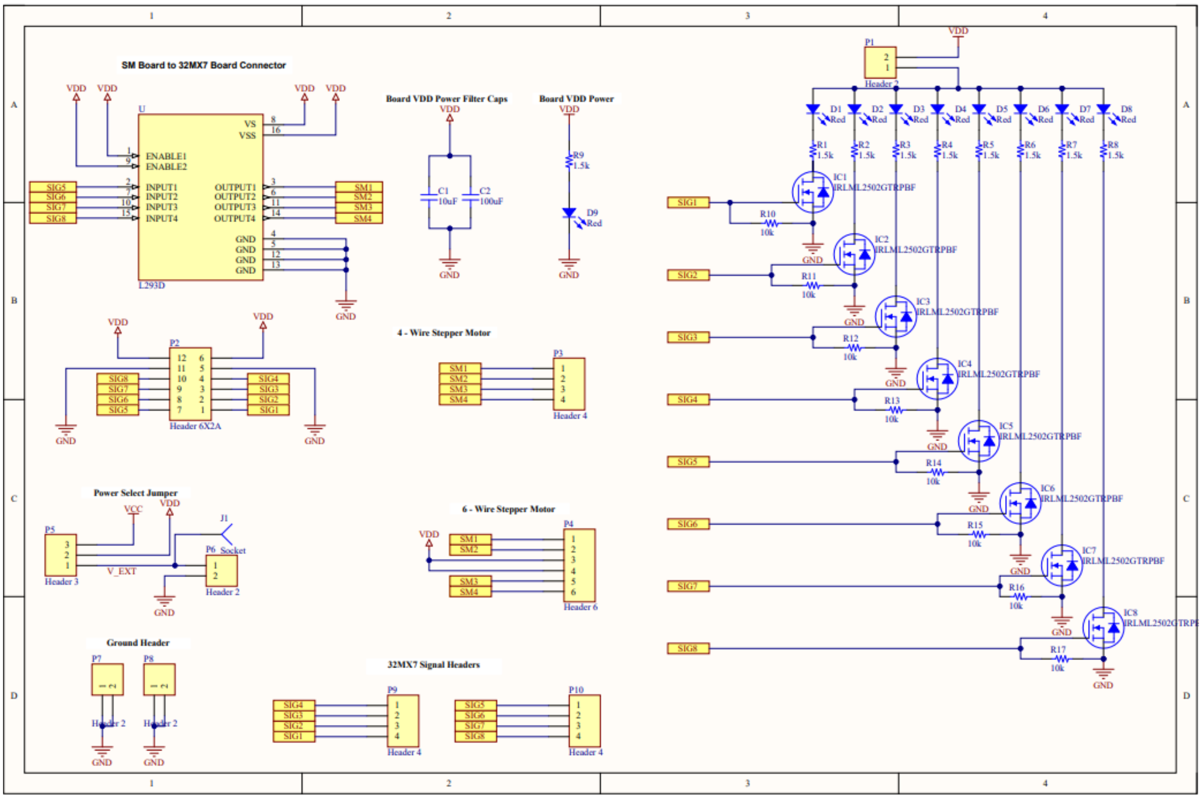

Image 2

BOARD LAYOUT:

TOP:

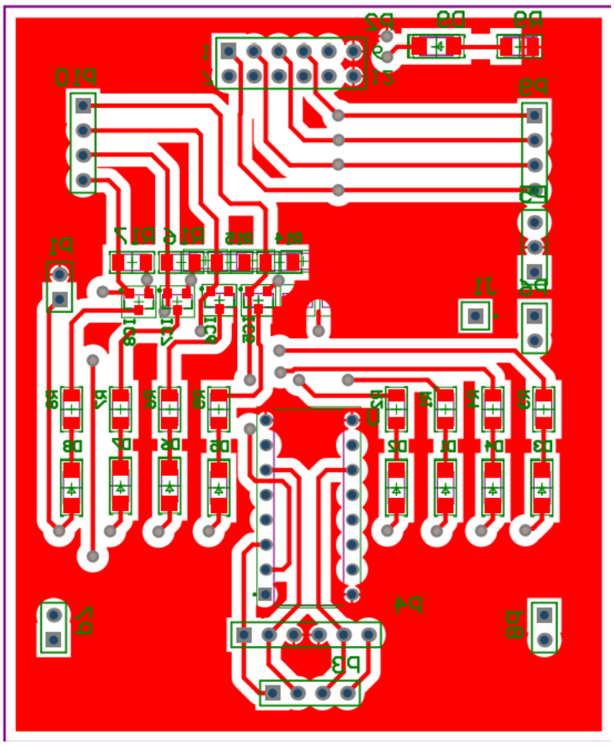

Image 3

BOTTOM:

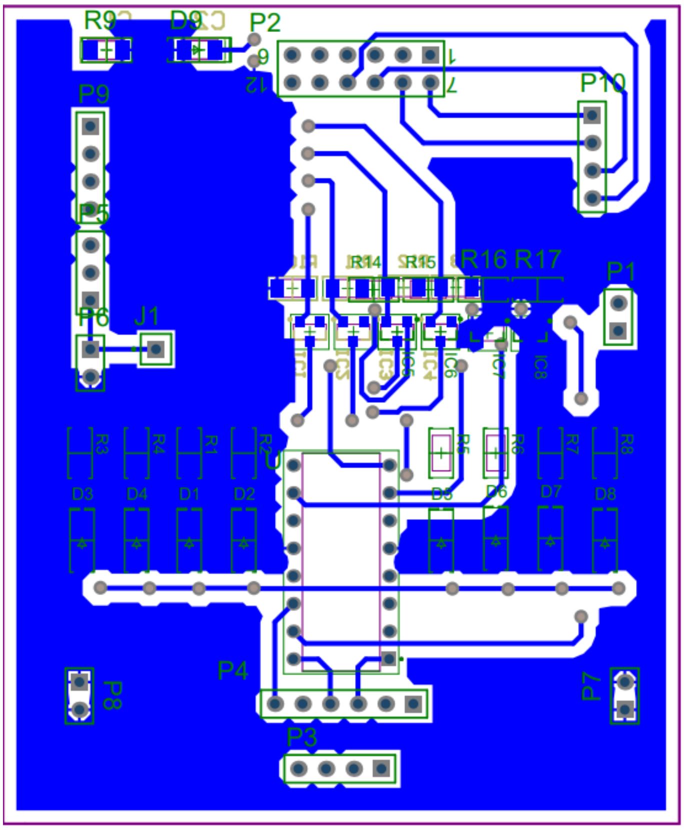

Image 4

TOP/BOTTOM:

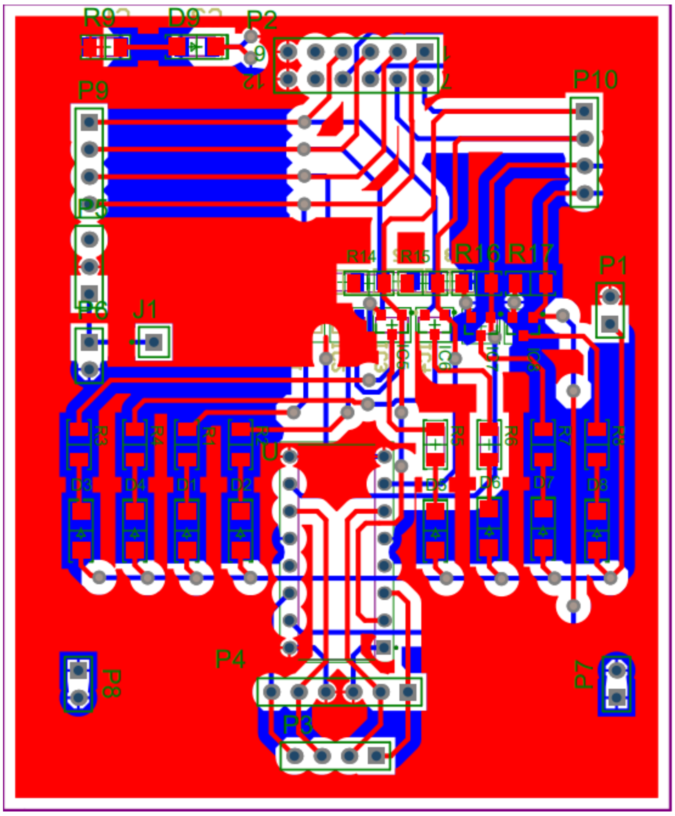

Image 5

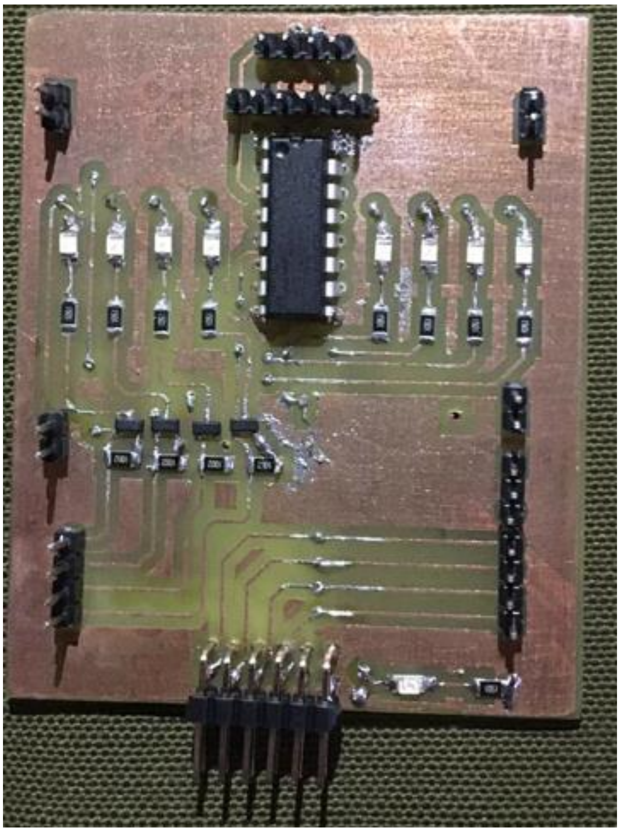

Image 6

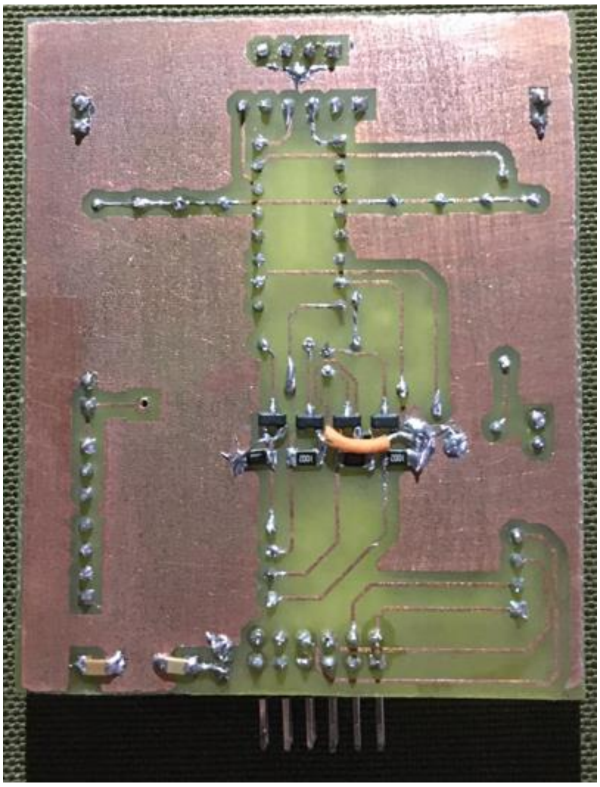

Image 7

MATERIALS LIST:

Comment	Description	Designator	Footprint	LibRef	Quantity
10uF	CAP CER 10UF 16V X5R 1206	C1	CAP 1206/3216	CL31A106KOCLNNC	1
100uF	CAP CER 100UF 6.3V X5R 1206	C2	CAP 1206/3216	CL31A107MQHNNNE	1
Red	LED RED CLEAR 1206 SMD	D1, D2, D3, D4, D5, D6, D7, D8, D9	LED 1206/3216 RED	SM1206NHC-IL	9
IRLML2502GTRPBF	MOSFET N-CH 20V 4.2A SOT-23-3	IC1, IC2, IC3, IC4, IC5, IC6, IC7, IC8	INFINEON SOT-23-3	IRLML2502GTRPBF	8
Socket	Socket	J1	PIN1	Socket	1
Header 2	Header, 2-Pin	P1, P6, P7, P8	HDR1X2	Header 2	4
Header 6X2A	Header, 6-Pin, Dual row	P2	HDR2X6_CEN	Header 6X2A	1
Header 4	Header, 4-Pin	P3, P9, P10	HDR1X4	Header 4	3
Header 6	Header, 6-Pin	P4	HDR1X6	Header 6	1
Header 3	Header, 3-Pin	P5	HDR1X3	Header 3	1
1.5k	RES SMD 1.5K OHM 1% 1/4W 1206	R1, R2, R3, R4, R5, R6, R7, R8, R9	RES 1206/3216	ERJ-8ENF1501V	9
10k	RES SMD 10K OHM 1% 1/4W 1206	R10, R11, R12, R13, R14, R15, R16, R17	RES 1206/3216	ERJ-8ENF1002V	8
L293D	Bipolar Motor Driver Parallel 16-PowerDIP	U	DIP880W50P254L200 0H510Q16	L293D	1

Image 8

REFERENCES:

- https://www.elektrovadi.com/urun/pmod-step-motor-surucu-pmod-stepper
- https://digilent.com/reference/pmod/pmodstep/reference-manual?redirect=1
- https://www.robocombo.com/blog/icerik/step-motor-nedir-ne-ise-yarar-nasil-calisir
- https://digilent.com/reference/learn/fundamentals/communication-protocols/gpio/start
- https://www.instructables.com/How-to-Control-a-Stepper-Motor-With-an-FPGA/
- https://www.youtube.com/watch?v=ePSCZ_DtF7c
- https://howtomechatronics.com/how-it-works/electrical-engineering/stepper-motor/