HAYDAY KAMIL MOUSA

HMA@FOSS.DK

COMPANY TEIS

INGENJORSJOBB (VER_3)

A bipolar Stepper Motor Control with Processor NIOS II and Cyclone IV/V FPGA

C_Engineering Work_b

Haydar Kamil Mousa

2015-04-23

A stepper motor is an electrical device which converts electrical pulses into discrete mechanical movements. The shaft or the spindle of stepper motor rotates in discrete step increments when the electrical command pulses are applied to it in the proper sequence. In this work, the movement of stepper motor is analyzed based on Field Programmable Gate Implementation FPGA. In this work, we will connect a bipolar stepper motor 8HY2042-01N, four phases, four wires with an external driver circuit (this circuit will be designed), and then the driver circuit will be connected with FPGA device in ALTERA DE2-115 Board. The slide switches in DE2-115 will be used to control direction of rotation and speed of stepper motor. Central Processing Unit NIOS II Process and CASE_2 hardware architecture will be used during this work.

Contents

1 Introduction

A stepper motor is an electrically powered motor that creates rotation from electrical current driven into the motor. Physically, stepper motors can be large but are often small enough to be driven by current on the motor shaft. This is unlike a DC motor that exhibits continuous rotation. Although it is possible to drive a stepper motor in a manner where it has near continuous rotation, doing so requires more finesse of the input waveform that drives the stepper motor. Figure_1 illustrate some basic differences in stepper and DC motor rotation.

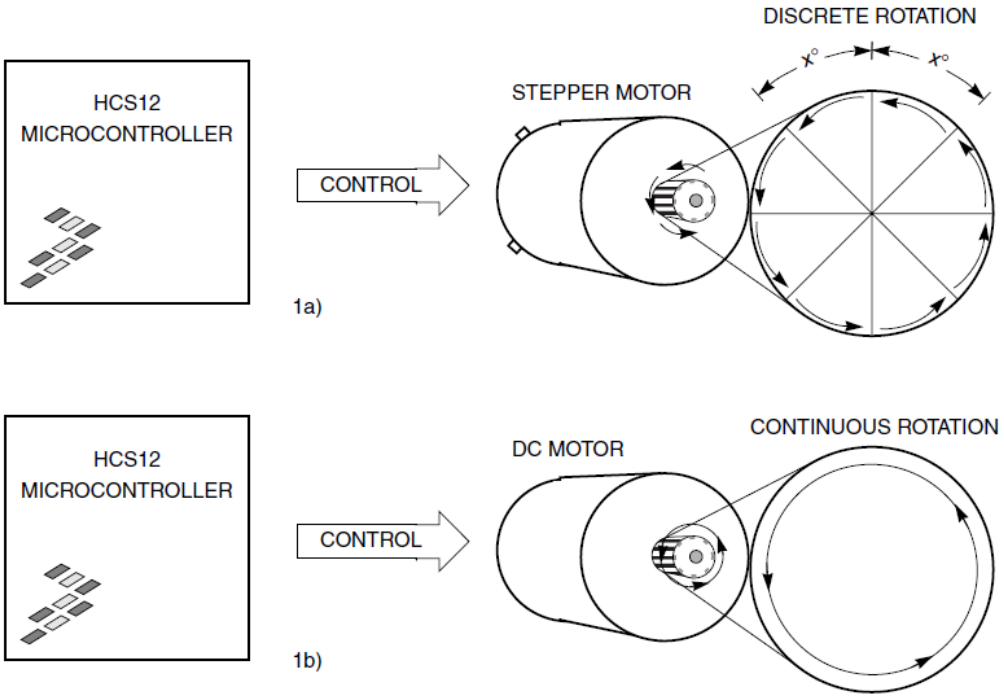


Figure_1 stepper motor vs. DC motor rotation

The most important criteria for stepper motor are the speed and position control depending upon the certain application. Since the frequency of the digital input pulses control the speed of stepper motor, the motor can be rotated at a rated speed matching the requirement of the user. Fugure_2 shows the basic diagram for stepper motor control. It consists of the controller, driver and stepper motor.

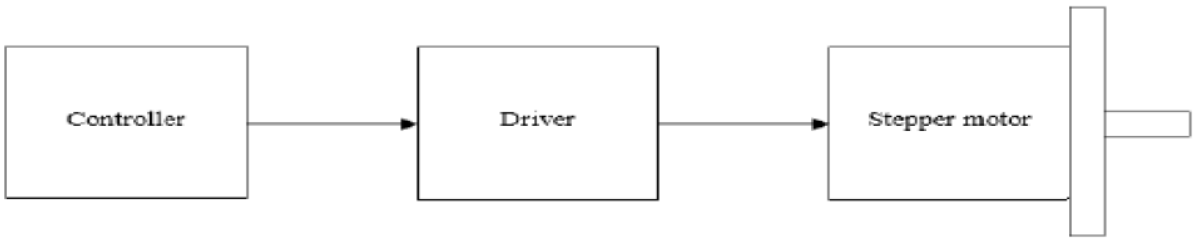


Figure_2 Block diagram of Stepper Motor Control

1.1 Background

The stepper motor has the following features:

- The rotation angle of the motor is proportional to the input pulse.
- The motor has full torque at standstill(if the windings are energized)
- Precise positioning and repeatability of movement since good stepper motors have an accuracy of – 5% of a step and this error is non cumulative from one step to the next.
- Excellent response to starting/stopping/reversing.
- Very reliable since there are no contact brushes in the motor. Therefore the life of the motor is simply dependant on the life of the bearing.
- The motors response to digital input pulses provides open-loop control, making the motor simpler and less costly to control.
- It is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft.
- A wide range of rotational speeds can be realized as the speed is proportional to the frequency of the input pulses.

In this work, we will connect a bipolar stepper motor 8HY2042-01N to motor driver circuit and then connect this motor driver to the DE2-115 Board. This stepper motor is a great way to get things moving, especially when positioning and repeatability is a concern.

2 Requirements specification

2.1 The system requirements

The required system is to design a hardware construction for simple control for bipolar stepper motor. This will be done by connecting a bipolar stepper motor model 8HY2042-01N to a driver circuit, which will be designed for this purpose. The driver circuit will be connected to FPGA device in ALTERA DE2-115 Board via GPIO connector JP5. Figure_3 and Figure_4 show the schematic diagram how to interface a bipolar stepper motor to four controller pins of motor driver circuit.

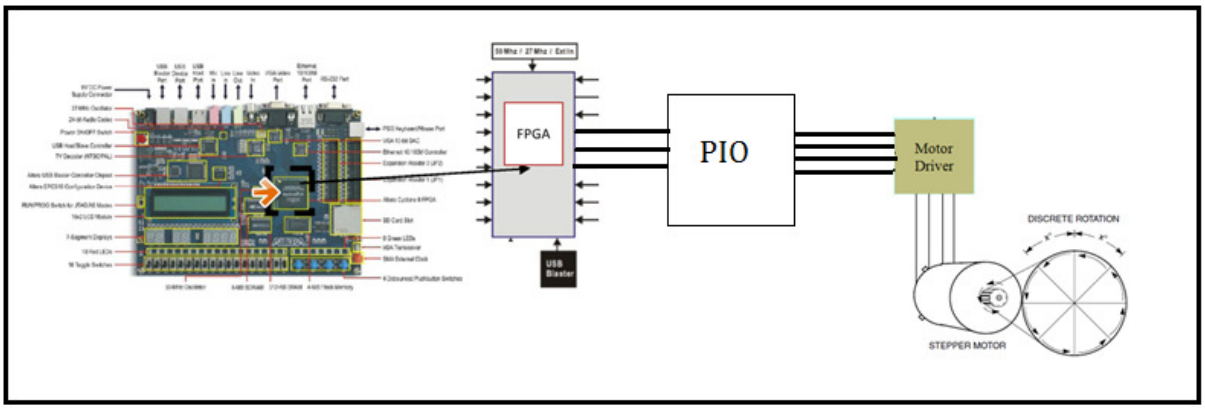


Figure_3 the schematic diagram for stepper motor, driver, PIO, and FPGA device

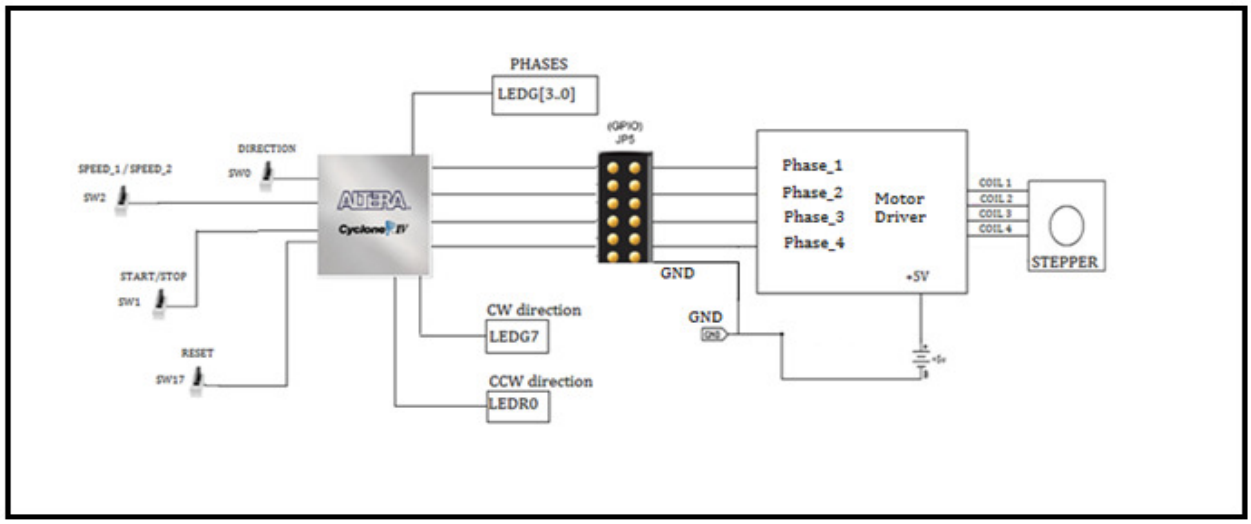


Figure_4 the connection diagram for bipolar stepper motor, driver, and FPGA device.

Motor Driver: This is a dual full bridge driver. It is a high voltage, high current designed to accept standard TTL logic levels. The whole design is shown Figure_5.

DIRECTION (DIR): Controls the direction of running of stepper motor.

CLOCK: Controls the speed of stepper motor.

STOP/START: Starts and stops the running of stepper motor.

GND: Both designed motor driver and DE2-115 should have the same ground.

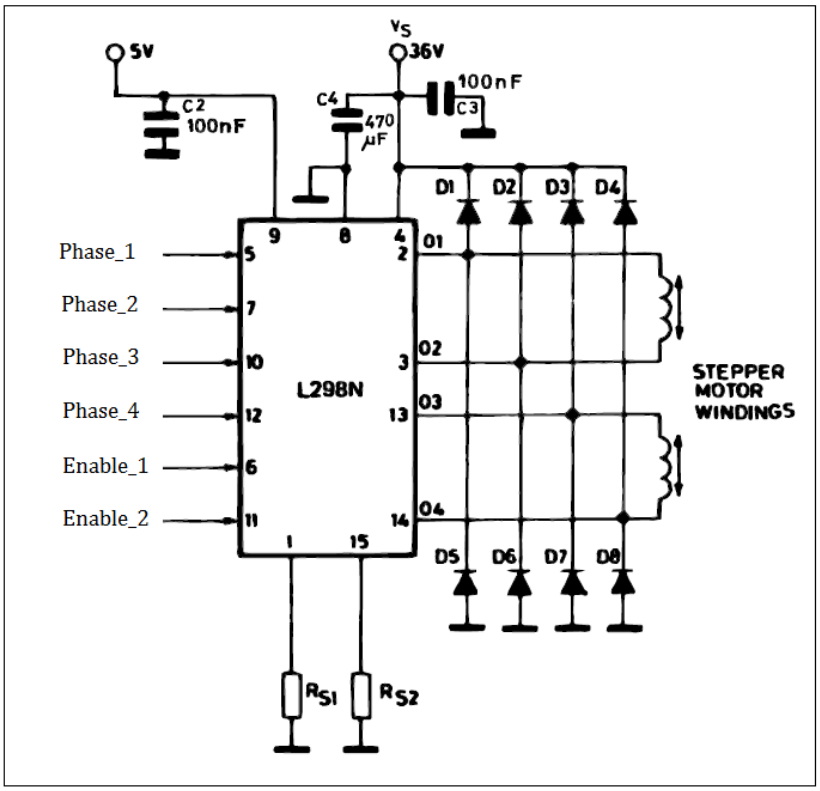


Figure_5 schematic diagram for motor driver circuit

The simplest way of interfacing a bipolar stepper motor to DE2-115 Board is to design a driver circuit with L298N. The L298 is an integrated monolithic circuit in a 15 lead Multiwatt and PowerSO20 packages. The driver circuit contains internal clamp diodes to dissipated voltage spikes when driving inductive loads. To control the stepper motor, apply voltage to each of the coils in a specific sequence, se Figure_5.

2.2 Constructions code requirements

Task_1

- ”head file” in all C-files.
- Name of the variables should be cleared.
- The C-codes should be well commented.
- All the drive routines should be existed in BSD.
- The C-codes should be constructed, not only the application file.

Task_2

- The specification, test protocol, SW architecture, hierarchy, memory map, all the I/O ports and functions should be cleared. Test protocol will test at least one case for each task.

Task_3

- The time analysis of the C-code. Answer the questions about the available time. The frequency for NIOS is 50MHz.

Task_4

- All the drive routines should be existed in components under HAL/src and inc. ”tcl” files should define all the drive routines for their respective component, in order to have them in BSP map (not under the applications maps).

Task_5

- Not applicable, because this work is implemented for stepper motor control not for game.

Task_6

- Describe how the code is optimized to get better execution time and memory size (by changing the setting of tools and code). Describe the memory architecture and explain why code/ data are linked to different memories.

Task_7

- A short film should be published in YouTube. This film explains how to control the bipolar stepper motor. This film starts with a short description of the work, and then a short summery of the SW construction (max 3 minutes for whole presentation).

2.3 Delevery requirements

Task_8

The delivery will be consisted of three maps:

a. Construction and description

- Constructions report in standard form (Task_3, Task_6).
- A short SW manual for drive routines (Task_5).
- SW files in on map (Task_1).
- All IP components and timer with their drive routines (Task_4).

b. User manual

- A simple user manual (describe how to run and control the stepper motor), (Task_2).

c. Diverse

- Start report.

Task_9

- The delivery will be done via the platform **Itslearning**. The name of the file is "firstname_lastname_C_ingenjorsjobb_b.zip".

3 Time plan

The activities and the time plan for the project:

- Design a hardware construction for simple control of stepper motor.
- Write the suitable C-Code to communicate with processor NIOS II, and thus controls the direction and speed of stepper motor.
- Use NIOS II Processor in ALTERA DE2-115 Board to control the motor driver.

Figure_8 shows a block diagram of ALTERA DE2-115 with Cyclone FPGA and Nios II processor.

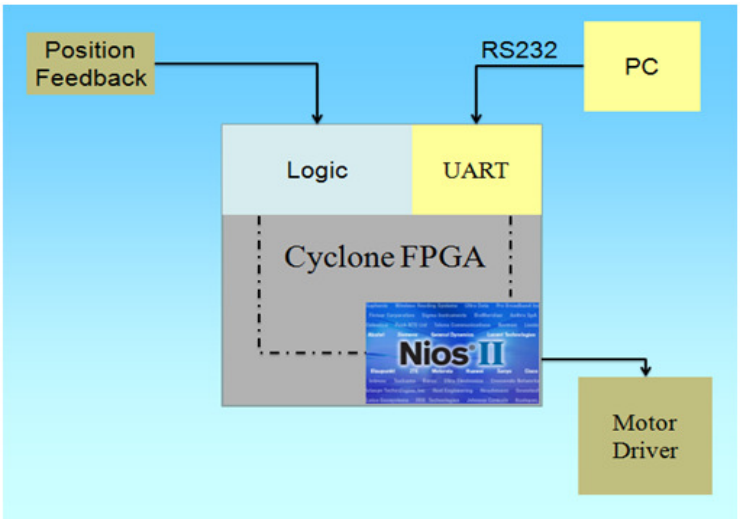


Figure_8 NIOS II Processor is used to control the motor driver

- Connect the bipolar stepper motor to the motor driver circuit.
- Select the required Pin assignment for the whole design.
- Two LED will be used to indicate the direction of running. LEDG_7 is used for clockwise rotation and LEDR_0 for counter clockwise.
- LEDG[3..0] are used to indicate the changing of sequences(phases).
- Use external power supply to provide the required DC voltage to the stepper motor.

- Connect the logic circuit of motor driver to 5VDC in DE2-115 Board.
- Be sure that both motor driver and DE2-115 Board have the same GND.
- SW0 in DE2-115 is used to control the direction of running of stepper motor. SW17 is used for reset. SW1 is to start/stop the stepper motor, while SW2 to change the speed of stepper motor.
- The sequence would go like in Table_1:

Table_1 shows the sequence of phases

Lead Wire Color	---> CW Direction (1-2 Phase)							
	1	2	3	4	5	6	7	8
4 ORG	-	-						-
3 YEL		-	-	-				
2 PIK				-	-	-		
1 BLU						-	-	-

- Design motor driver circuit, as shown in Figure_9.

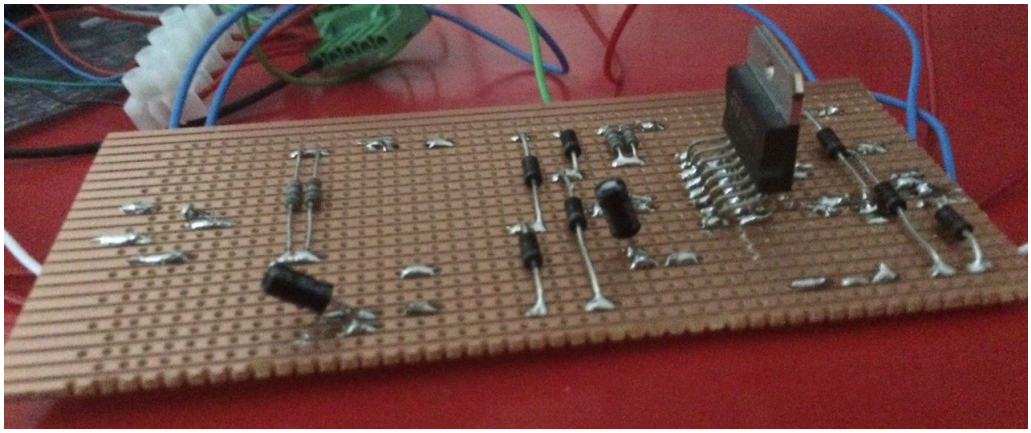


Figure_9 PCB for motor driver circuit

The following components are used:

1. L298 N * 1 Pcs.
2. High speed diodes 1N4448 * 8 Pcs.
3. Ceramic resistor 1Ohm * 4 Pcs , 2 Watt .
4. Capacitor 0.01μF * 2 Pcs , 50V .

4 Design specification

4.1 System architecture

The architecture hardware can consist of one or many components and sub-components. The architecture describes the content of these components and the sub-components inside the structure. In this work, the hardware architecture consists of the following components (as shown in Figure_10 and Figure_11).

- Central Processing Unit CPU: This is NIOS II.
- Avalon Bus to communicate between the components.
- Internal FPGA RAM to hold data.
- NIOS II JTAG UART, to communicate with external terminal.
- FPGA device can be programmed / configured to nearly any hardware architecture.
- The software program is the program (source code) that is compiled and assembled to the object code and the linked to form an executable program.

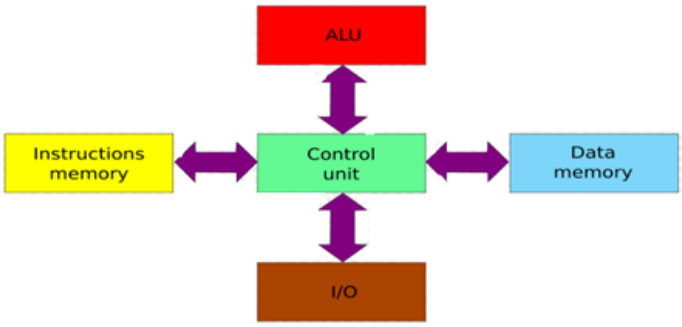


Figure_10 Computer Architecture

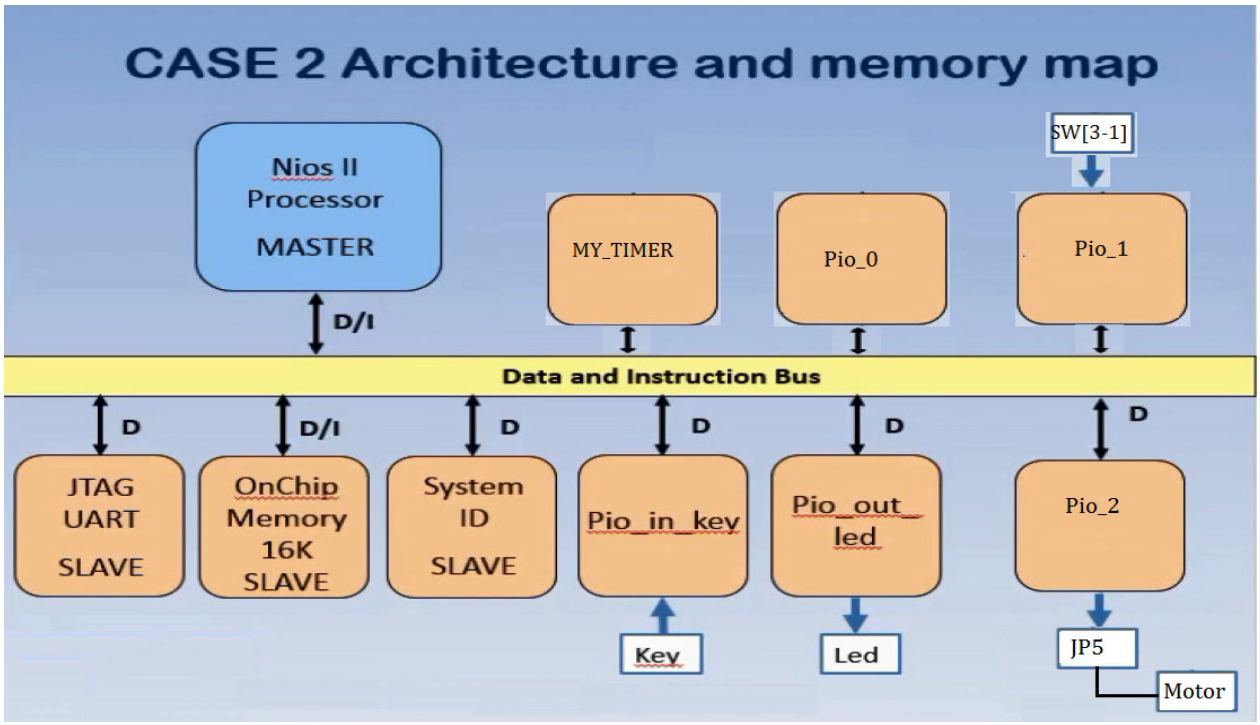


Figure_11 shows CASE_2 hardware architecture used in this work

Memory map is a layout of the address space to different parts as I/O resistors, data and instructions. It can be said that the memory map is a table with addresses and information of what part that is connected to which address. Table_2 below shows the memory map for this case.

Table_2 Memory Map

Slave Descriptor	Address Range	Size	Attributes
TERASIC_SRAM_0	0x200000 – 0x2FFFFF	1048576	Memory
Onchip_ram	0xC0000 -	196608	Memory
Jtag_uart	0x101078 -	8	Printable
Sysid_qsys_0	0x00009020 – 0x00009027	8	
pio_in_key	0x101060 -	16	
pio_out_led	0x101050	16	
pio_0	0x101030 -	16	
pio_1	0x101010 -	16	
Pio_2	0x101000 -	16	
Pio_IR_8_bit	0x101020 -	16	
Altera_avalon_sysid_qsys	0x101070-	8	
MY_TIMER	0x101040	16	
IP_VGA	0x0	524288	

4.2 I/O- pins

The whole PIN Planner of the system is shown in Table_3

Table_3 PIN Planner

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved
clk_50	Input	PIN_Y2	2	B2_N0	PIN_Y2	2.5 V (default)	
clk_out_a	Output	PIN_Y17	4	B4_N0	PIN_Y17	2.5 V (default)	
DIR	Input	PIN_AB28	5	B5_N1	PIN_AB28	2.5 V (default)	
DIR_OUT	Output	PIN_AC15	4	B4_N2	PIN_AC15	2.5 V (default)	
LEDG	Output	PIN_G21	7	B7_N1	PIN_G21	2.5 V (default)	
LEDG_phases[3]	Output	PIN_E24	7	B7_N1	PIN_E24	2.5 V (default)	
LEDG_phases[2]	Output	PIN_E25	7	B7_N1	PIN_E25	2.5 V (default)	
LEDG_phases[1]	Output	PIN_E22	7	B7_N0	PIN_E22	2.5 V (default)	
LEDG_phases[0]	Output	PIN_E21	7	B7_N0	PIN_E21	2.5 V (default)	
LEDR	Output	PIN_G19	7	B7_N2	PIN_G19	2.5 V (default)	
PHASE[3]	Output	PIN_AD21	4	B4_N0	PIN_AD21	2.5 V (default)	
PHASE[2]	Output	PIN_AC21	4	B4_N0	PIN_AC21	2.5 V (default)	
PHASE[1]	Output	PIN_AB21	4	B4_N0	PIN_AB21	2.5 V (default)	
PHASE[0]	Output	PIN_AB22	4	B4_N0	PIN_AB22	2.5 V (default)	
reset_n	Input	PIN_Y23	5	B5_N2	PIN_Y23	2.5 V (default)	
SPEED	Input	PIN_AC27	5	B5_N2	PIN_AC27	2.5 V (default)	
START	Input	PIN_AC28	5	B5_N2	PIN_AC28	2.5 V (default)	
STOP	Output	PIN_Y16	4	B4_N0	PIN_Y16	2.5 V (default)	

4.3 Tools – ”settings och assignments”

Figure_12 shows the Linker Script in settings.bsp of Nios II BSP Editor.

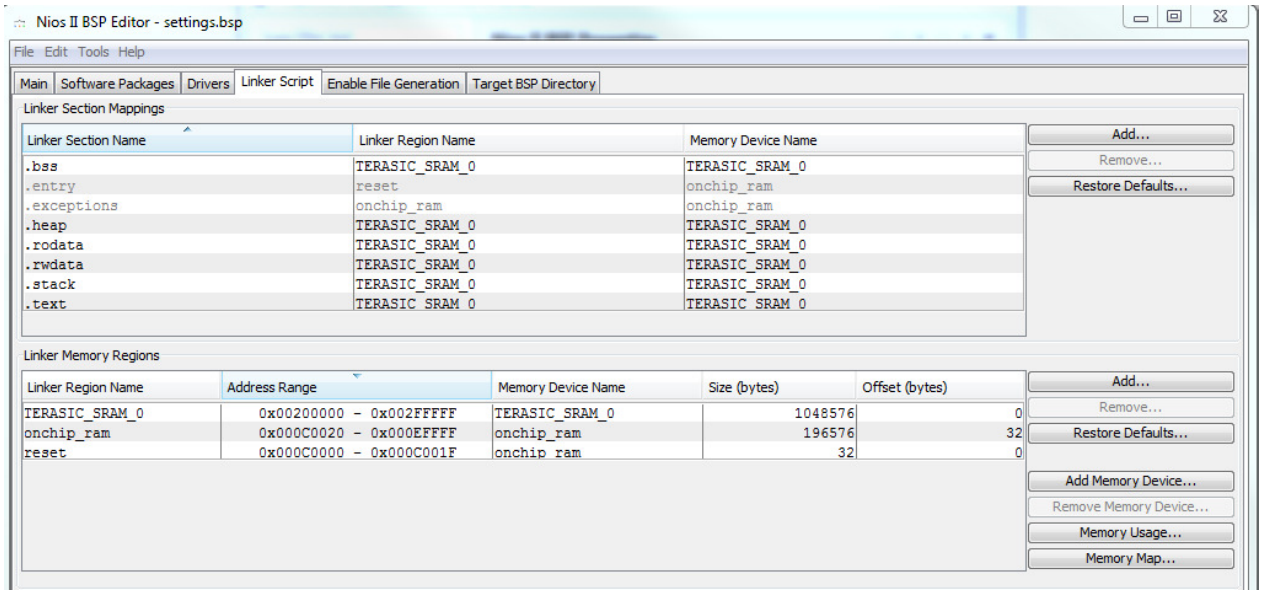


Figure_12 Linker Script in NIOS II BSP Editor

Figure_13 shows that all the drive ruotins and header files are existed in components under HAL/src and inc.

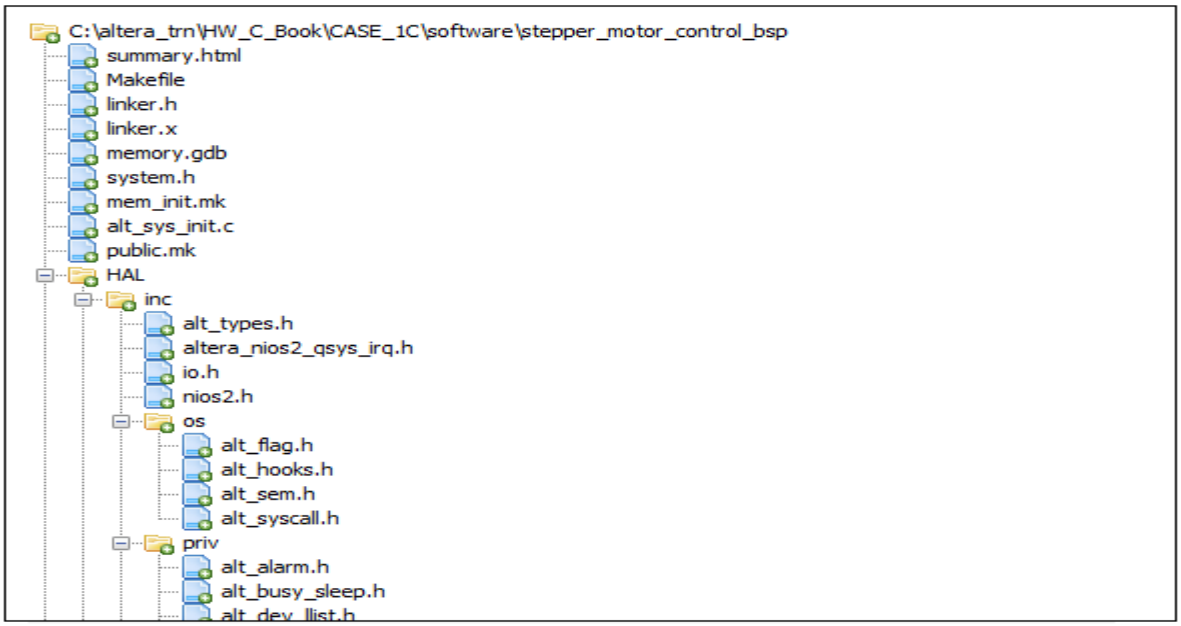


Figure_13 Target BSP Directory

Figure_14 shows the module name and drivers names of Altera that are used in this case.

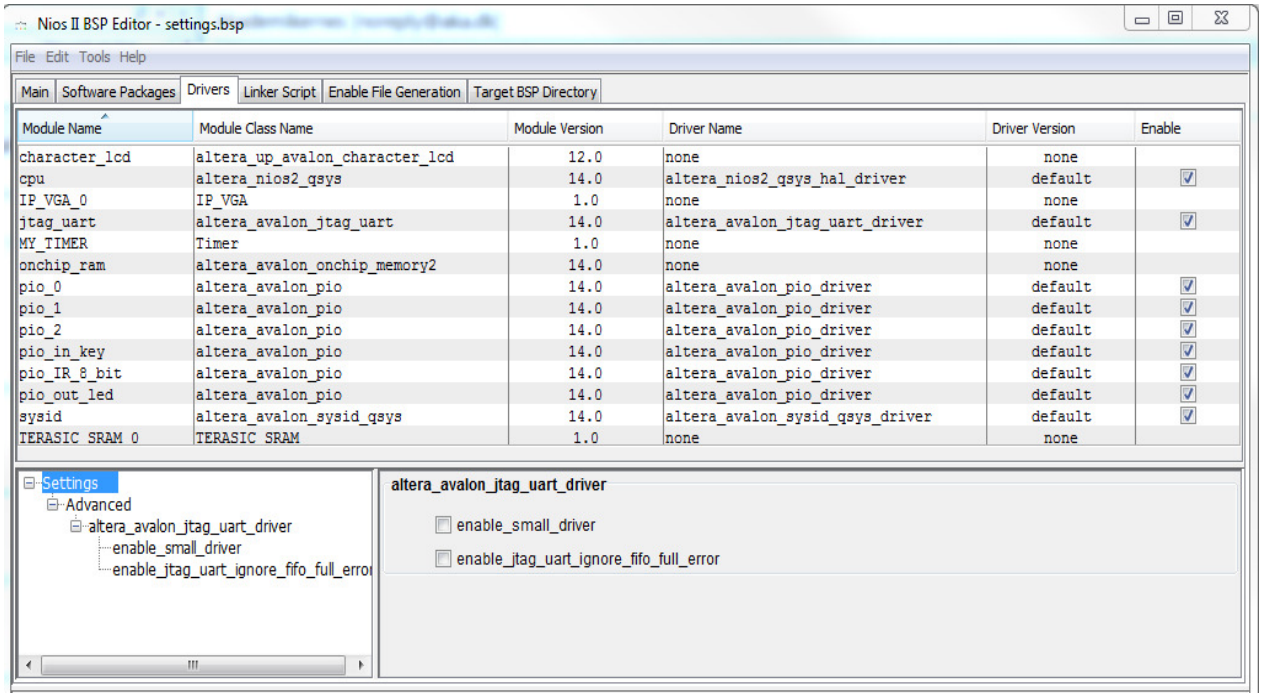


Figure_14 Drivers names of Altera

4.4 Test protocol

Test protocol or test plan as sometimes called, are formal documents that typically outline requirements, activities, resources, documentation and schedules to be completed. Some form of test plan should be developed prior to any test. Test protocol is shown in Table_4.

For our stepper motor, the test protocol will include the following cases:

- **Case_1:** The direction of running for stepper motor will be checked when the slide switch SW1 is set to '1'. The direction should be clockwise and LEDG[0] is lighted. The phases or sequences are 0101, 0110, 1010, 1001, 0101,0110,1010,1001.

- **Case_2:** The direction of running for stepper motor will be checked when the slide switch SW1 is set to '0'. The direction should be clockwise and LEDR[0] is lighted. The phases or sequences are 1001, 1010, 0110, 0101, 1001,1010,0110,0101.

- **Case_3:** Both the direction of running and speed for stepper motor will be checked when the slide switches SW1 and SW3 are set to '1' and '0' respectively. The direction of running will be clockwise and the stepper motor will run faster. The sequences are 0101, 0110,1010,1001,0101,0110,1010,1001.

- **Case_4:** The direction of running for stepper motor will be checked when only the slide switch SW1 is switched from '1' to '0'. The direction of running should be changed from clockwise to counter clockwise. The sequence will be changed from 0101, 0110,1010,1001,0101 to be 1001,1010,0110,0101. The stepper motor will reverse its motion and run slowly.

- **Case_5:** The speed of stepper motor will be checked when only the slide switch SW3 is switched from '1' to '0'. No change should be noticed in the sequence 0101, 0110,1010,1001,0101,0110,1010,1001, but the stepper motor will run faster.

The specification requirement and the test protocol are explained in Table_4.

Table_4 illustrate the requirement and test protocol

CASE	Description	LEDG , LEDR & phases	Verification	Validation
CASE_1	DIR = '1', START = '1' SPEED = '1'	LEDG = '1' LEDR ='0' LEDG_phases : 0101,0110,1010,1001,0101,0110,1010,1001 Stepper motor is running clockwise(slower)	OK	OK
CASE_2	DIR = '0' , START = '1' SPEED = '1'	LEDG = '0' LEDR = '1' LEDG_phases: 1001,1010,0110,0101,1001,1010,0110,0101 Stepper motor is running counter clockwise(slower)	OK	OK
CASE_3	DIR = '1' START = '1' SPEED = '0'	LEDG = '1' LEDR = '0' LEDG_phases: 0101,0110,1010,1001,0101,0110,1010,1001 Stepper motor is running clockwise(faster)	OK	OK
CASE_4	DIR changed from '1' to '0' START='1' SPEED='1'	LEDG changed from '1' to '0' LEDR changed from '0' to '1' LEDG_phases: 0101,0110,1010,1001,0101 changed to 1001,1010,0110,1010 Stepper motor is running clockwise(slower) and then reversing its motion(after 2s)	OK	OK
CASE_5	SPEED changed from '1' to '0' DIR ='1' START = '1'	LEDG = '1' LEDR='0' LEDG_phases : 0101,0110,1010,1001,0101 continued to 0110,1010,1001,0101 Stepper motor is running clockwise(slower) and then running faster(after 2s)	OK	OK

5 Verification of test protocol

5.1 Results from verification

The following diagrams are getting directly from oscilloscope AGILENT DSO-X-3024A. This oscilloscope is connected directly to the following pins of JP5 (GPIO):

Channel_1 is connected to PIN AB_22 on JP5 (with orange color)

Channel_2 is connected to PIN AB_21 on JP5 (with green color)

Channel_3 is connected to PIN AC_21 on JP5 (with blue color)

Channel_4 is connected to PIN AD_21 on JP5 (with red color)

Figure_15 illustrates the motor_phases (sequences) when dir = 1 and speed = 1. In this case the stepper motor is running clockwise.

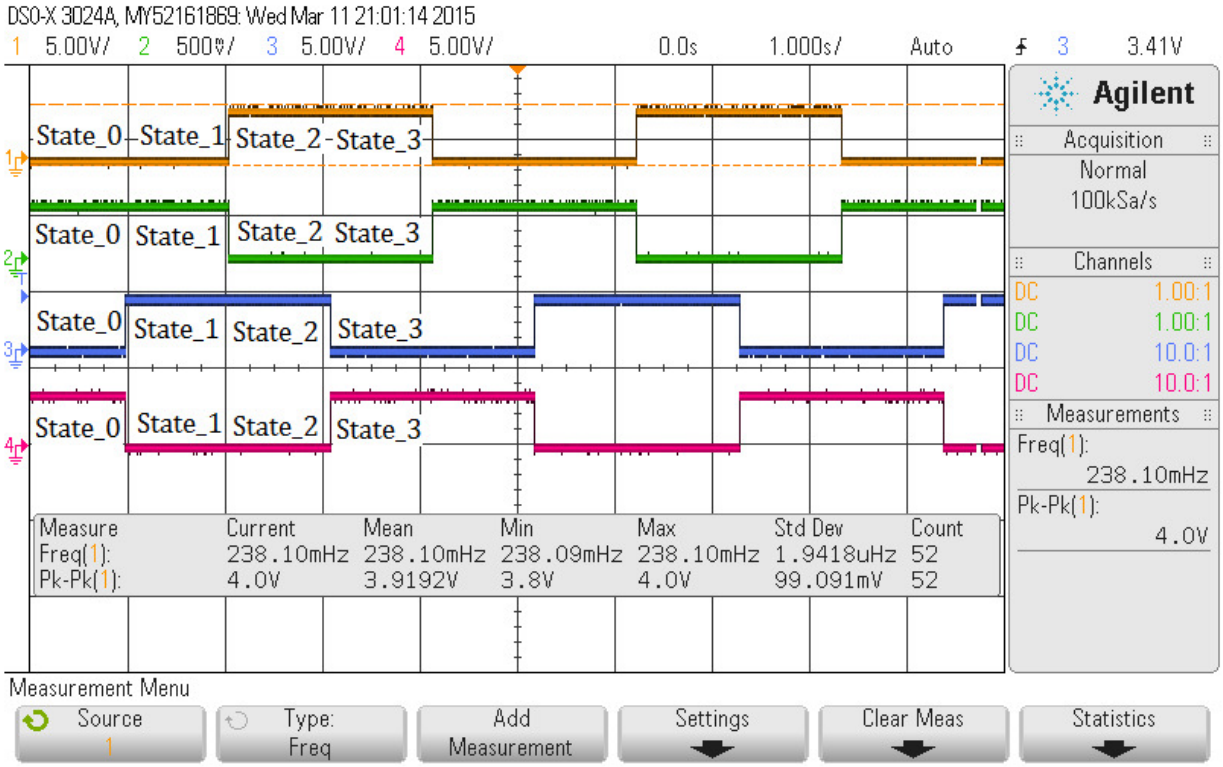


Figure_15 Sequences during ech state machine when dir = 1 and speed = 1

Figure_15 illustrates also motor_phases (sequences) during each state machine of the four state machines.

State_0 , the motor_phases(sequences) are 0101.

State_1 , the motor_phases (sequences) are 0110.

State_2 , the motor_phases (sequences) are 1010.

State_3 , the motor_phases (sequences) are 1001.

Figure_16 shows the measurements of frequency and the current during the four state machines. The measured frequency represents the speed of stepper motor. The frequency is measured to 238.1 mHz when speed (SW3) = 1. The time delay for all four states is measured to 1/238.1 mHz = 4.19 sec. The measured current or voltage represents the motor_phases or sequences on pins of JP5 are measured to 4V.

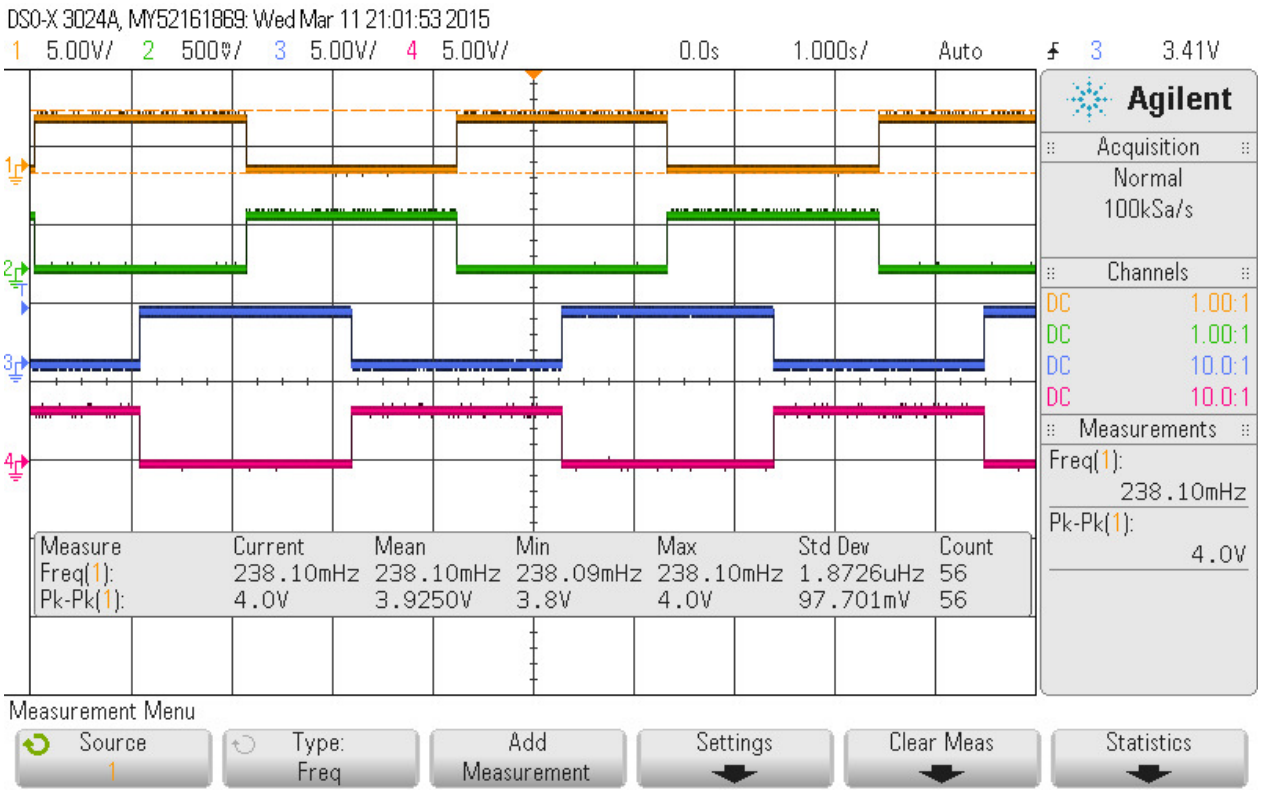


Figure_16 measurments of frequency and current

Figure_17 illustrates the motor_phases (sequences) when dir = 0 and speed = 0. In this case the stepper motor is running counter clockwise. This figure illustrates also motor_phases (sequences) during each state machine of the four state machines.

State_0 , the motor_phases(sequences) are 1001.

State_1 , the motor_phases (sequences) are 1010.

State_2 , the motor_phases (sequences) are 0110.

State_3 , the motor_phases (sequences) are 0101.

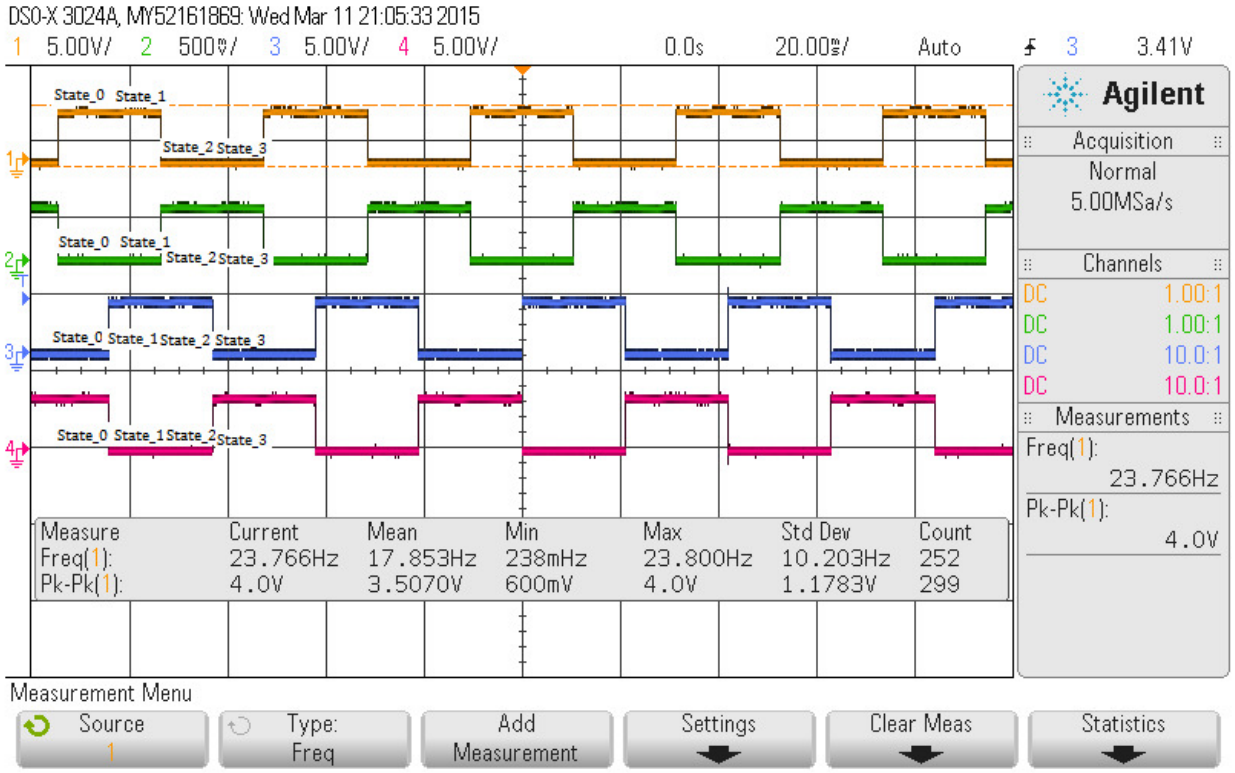


Figure_17 Sequences during each state machine when dir = 0 and speed = 0

Figure_13 shows the measurements of frequency and the current during the four state machines. The measured frequency represents the speed of stepper motor. The frequency is measured to 23.766Hz when speed (SW3) = 0. The time delay for all four states is measured to 1/23.766 Hz = 0.042 sec. This means the stepper motor is running faster when speed = 0. The measured current or voltage represents the motor_phases or sequences on pins of JP5 are measured to 4V.

Figure_18 illustrates the motor_phases (sequences) when dir (SW1) is changed from 0 to 1 and speed = 1. In this case the stepper motor is running slower and reverses its motion from counter clockwise to clockwise. This figure illustrates also motor_phases (sequences) during each state machine of the four state machines.

State_0 , the motor_phases(sequences) are 0110.

State_1 , the motor_phases (sequences) are 1010.

State_2 , the motor_phases (sequences) are 1001.

State_3 , the motor_phases (sequences) are 0101.

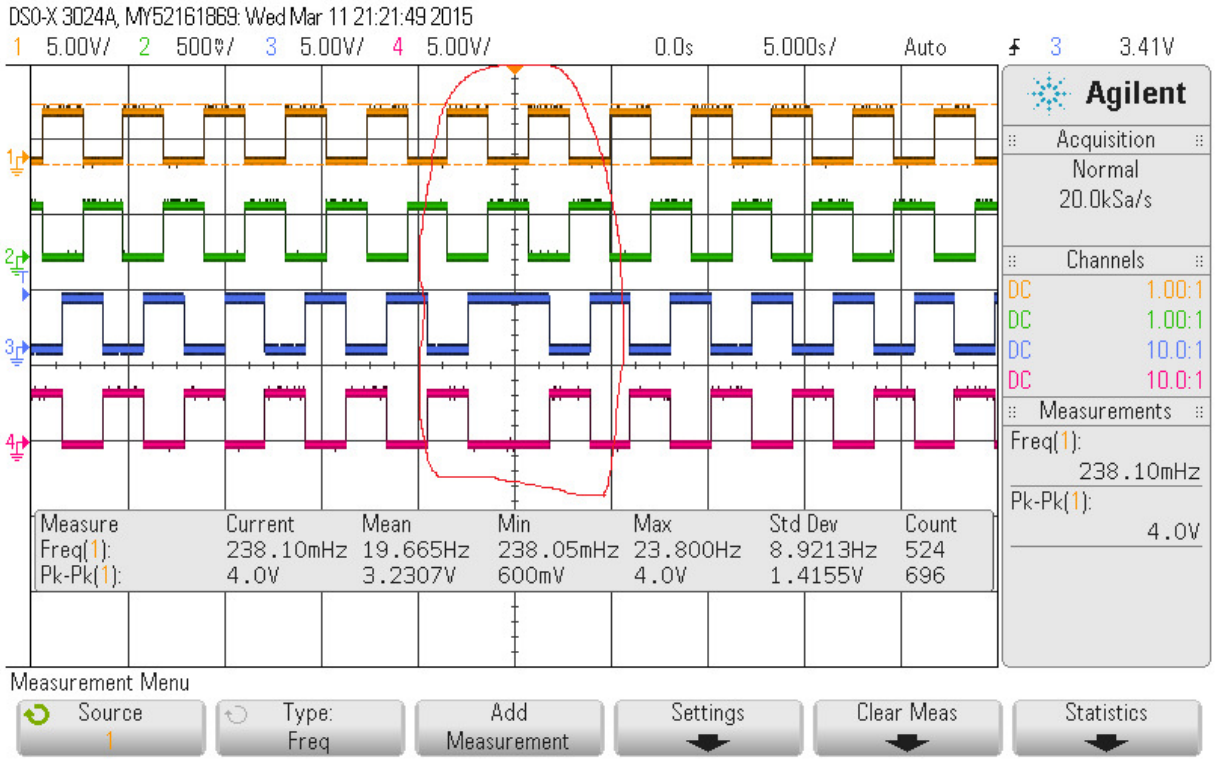


Figure_18 Sequences when dir changed from 0 to 1 and speed = 1

Figure_19 illustrates the motor_phases (sequences) when speed (SW3) is changed from 0 to 1 and dir = 1. In this case, the stepper motor will run slower and clockwise. This figure illustrates also motor_phases (sequences) during each state machine of the four state machines.

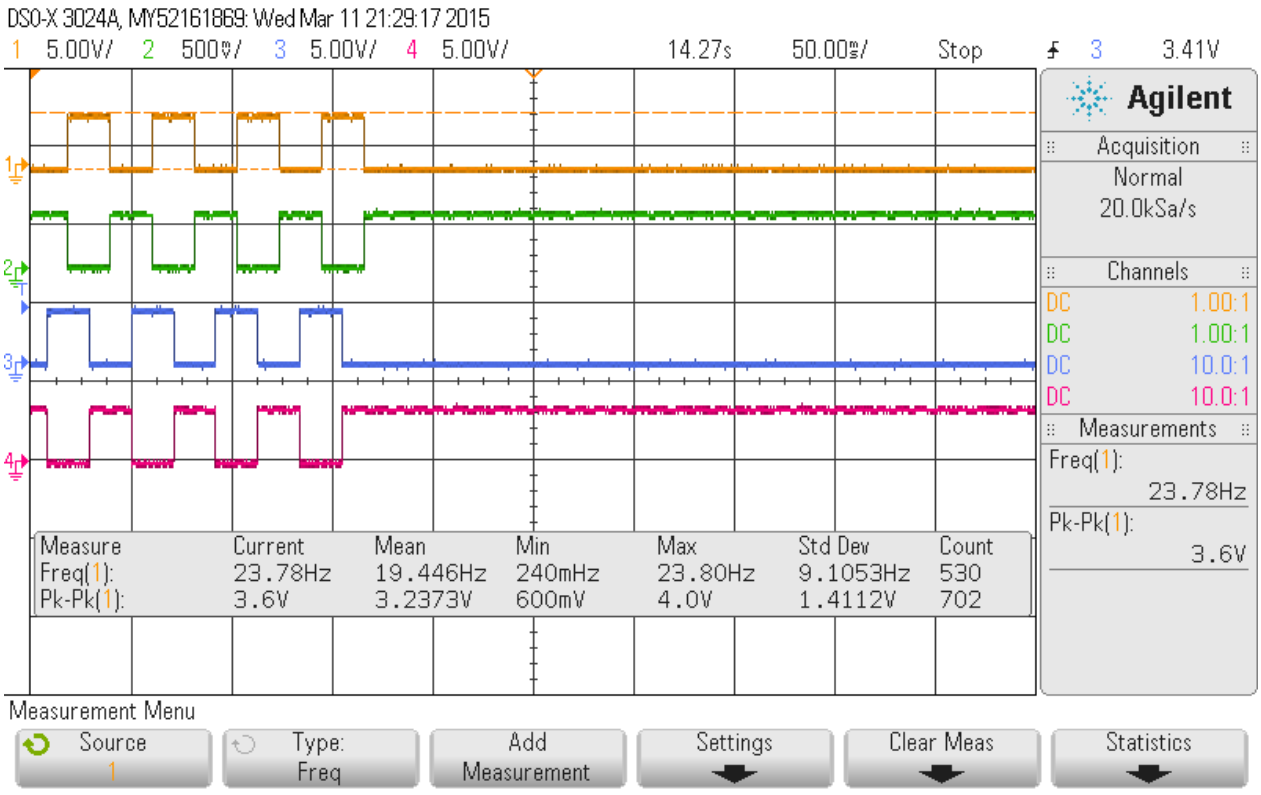


Figure_19 Sequences when speed is changed from 0 to 1 and dir = 1

Figure_20 illustrates the motor_phases (sequences) when speed (SW3) is changed from 1 to 0 and dir = 1. In this case, the stepper motor will run faster and clockwise. This figure illustrates also motor_phases (sequences) during each state machine of the four state machines.

Figure_20 Sequences when speed is changed from 1 to 0 and dir = 1

Figure_21 illustrates the motor_phases (sequences) when start/stop (SW2) is changed from 1 to 0 and dir = 1. In this case, the stepper motor will stop running. This figure illustrates also motor_phases (sequences) during each state machine of the four state machines. No motor_phases will be delivered on pins of JP5.

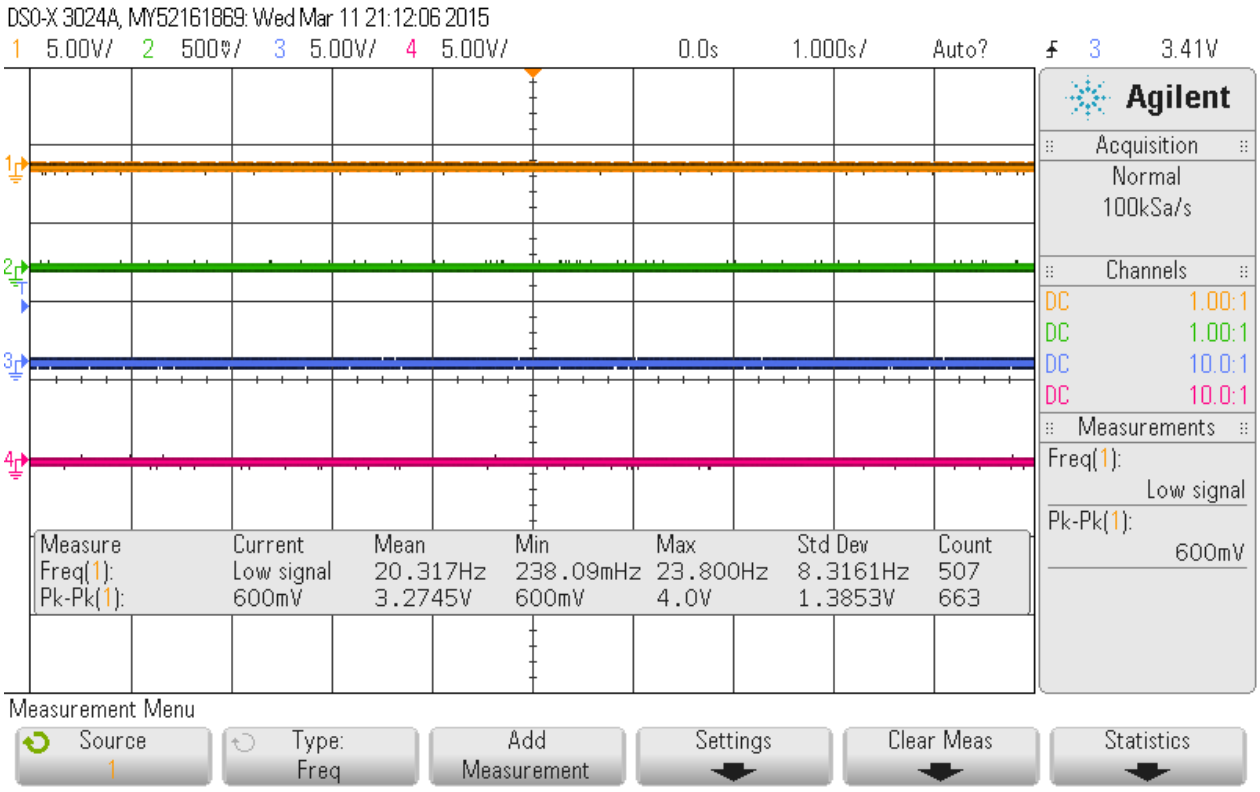


Figure_21 No sequences on the pins of JP5 when start/stop is changed from 1 to 0

5.2 Analysis

5.2.1 Analysis of warning or error signal from tools with comments

(No warning or error signal from tools)

5.2.2 Code optimization

Embedded software often runs on processors with limited computation power, thus optimizing the code becomes a necessity. In this work, the followings optimization techniques for C code are implemented.

- By usning alt_printf() in application in stead of printf(). The command alt_printf() uses 350 bytes , while printf() uses 2240 bytes.
- By deactivate some drive routines and hardware. Figure_22 shows the selecting of **enable_small_c_library** and **enable_reduced_device_drivers** to reduce memory size.

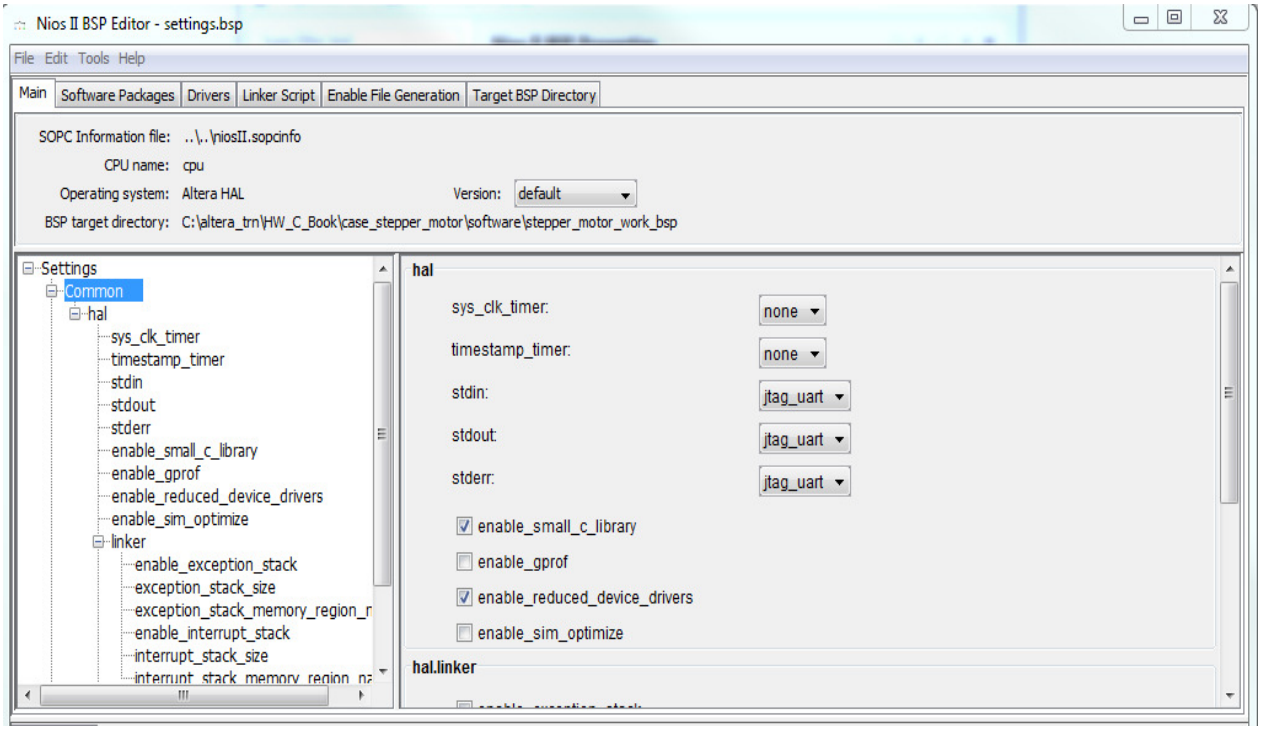


Figure_22 enable_small_c_library and enable_reduced_device_drivers

- By minimizing local variables. If the number of local variables in a function is less, the compiler will be able to fit them into registers. Hence, it will be avoiding frame pointer operations on local variables that are kept on stack. This can result in considerable improvement due to two reasons:
  - All local variables are in registers so this improves performance over accessing them from memory.
  - If no local variables need to be saved on the stack, the compiler will not incur the overhead of setting up and restoring the frame pointer.

5.2.3 Comments of "Design Assistant"

Design Assistant is not applicable in this work.

6 Validation of test protocol

6.1 Result from validation

The following results obtained from validation:

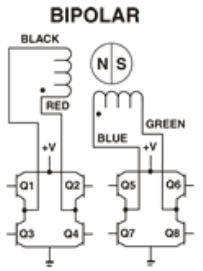
When SW1 is set to '1' (this means DIR='1') , the stepper motor starts running clockwise. The LEDG0 lights and LEDR0 is off. LEDG_phases indicates the sequences (phases).

When SW1 is set to '0' (this means DIR='0'), the stepper motor starts running counter clockwise. The LEDR0 lights and LEDG0 is off.

When SW3 is set to '1' (this means SPEED='1'), the stepper motor starts running slowly. But when SW3 is set to '0', the stepper motor starts running faster. SW3 controls the speed of stepper motor. When SW3 switched from '1' to '0', this will change the frequency generated from 238.1 mHz to be 23.766 Hz.

7 Conclusion

The two phase stepping sequence described utilizes a “bipolar coil winding.” Each phase consists of a single winding. By reversing the current in the windings, electromagnetic polarity is reversed. The output stage of a typical two phase bipolar drive is further illustrated in the electrical schematic diagram and stepping sequence in Figure_23.



CW Rotation ⟶

BipolarStep	Q2-Q3	Q1-Q4	Q6-Q7	Q5-Q8
1	ON	OFF	ON	OFF
2	OFF	ON	ON	OFF
3	OFF	ON	OFF	ON
4	ON	OFF	OFF	ON
1	ON	OFF	ON	OFF

⟵ CCW Rotation

Figure_23 shows the electrical circuit which reverses the current in the windings

In this work, four state machines are implemented in software application to generate and deliver the sequences mentioned in Figure_23. The figure below Figure_24 shows the sequences on pins of JP5. These sequences will be delivered to motor deriver circuit to run the motor.

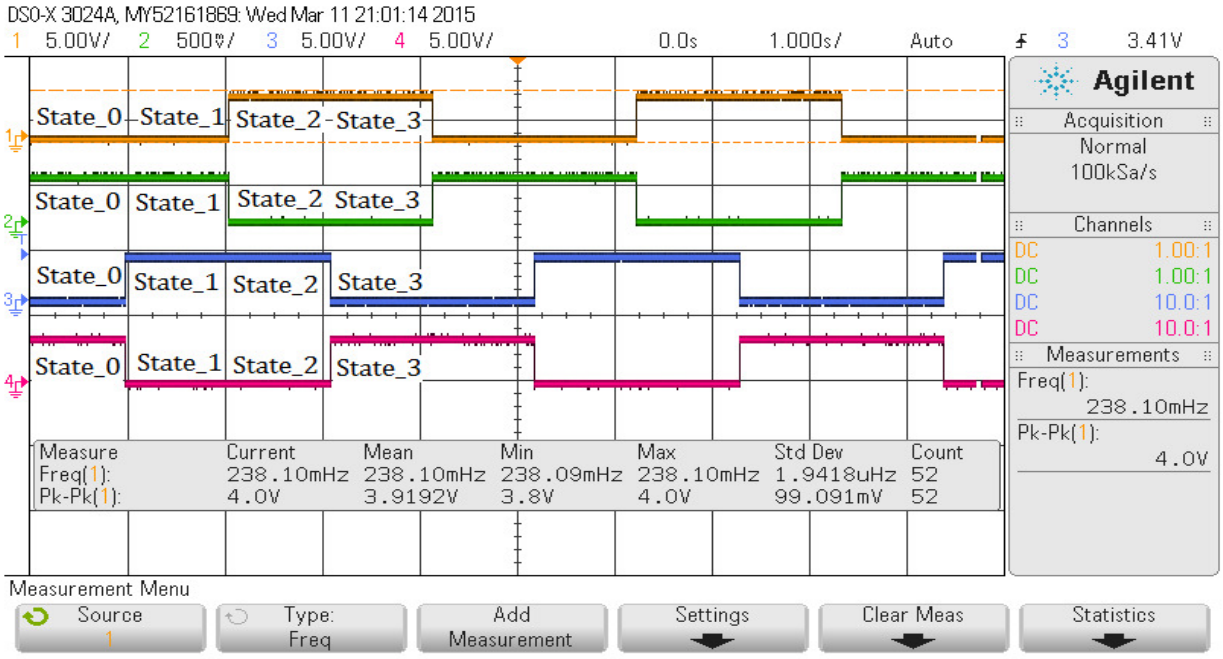


Figure_24 shows the sequences Q2-Q3, Q1-Q4, Q6-Q7, Q5-Q8 during four state machines

8 References

- http://www.haydonkerk.com/Resources/StepperMotorTheory/tabid/192/Default.aspx , 2015-02-19
- http://www.servomotor24.de/schrittmotor/schrittmotoren-hybrid/schrittmotor-8HY2042-02N.html , 2015-02-26
- http://all-about-embedded.blogspot.dk/2012/07/stepper-motor-interface-to.html , 2015-02-26
- http://www.geeetech.com/wiki/index.php/Stepper_Motor_5V_4-Phase_5-Wire_%26_ULN2003_Driver_Board_for_Arduino , 2015-02-19
- https://www.eewiki.net/display/LOGIC/Stepping+Motor+Control+(with+VHDL), 2015-02-19
- http://en.wikipedia.org/wiki/ULN2003A , 2015-02-19
- http://www.tech.dmu.ac.uk/~mgongora/Resources/L298N.pdf , 2015-02-26
- http://www.nmbtc.com/step-motors/engineering/full-half-and-microstepping/ , 2015-02-26
- http://www.haydonkerk.com/Resources/StepperMotorTheory/tabid/192/Default.aspx , 2015-02-26

9 Appendix

9.1 Appendix A: Bipolar stepper motor 8HY2042-01N

Product Description

Step Angle: 1.8°
Flange Size: NEMA 8, (20 x 20mm)
Key features: high accuracy; low inertia; small size

Model Numbering System
Multiple Shaft Options
Wiring Connections
Lead Wires & Cables
Lead Wires Configuration and Stepping Sequence

Model No.	Length (mm)	Holding Torque (mNm)		Rated Current(A)		Configuration	2D Drawings	3D Models
8HY2041-01N		29.5	13.0	0.4	bi-polar			
8HY2042-01N		29.5	16.0	0.6	bi-polar			
							2D Drawing	3D Model

9.2 Appendix B: Full bridge motor driver L298N

L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)

Multiwatt15

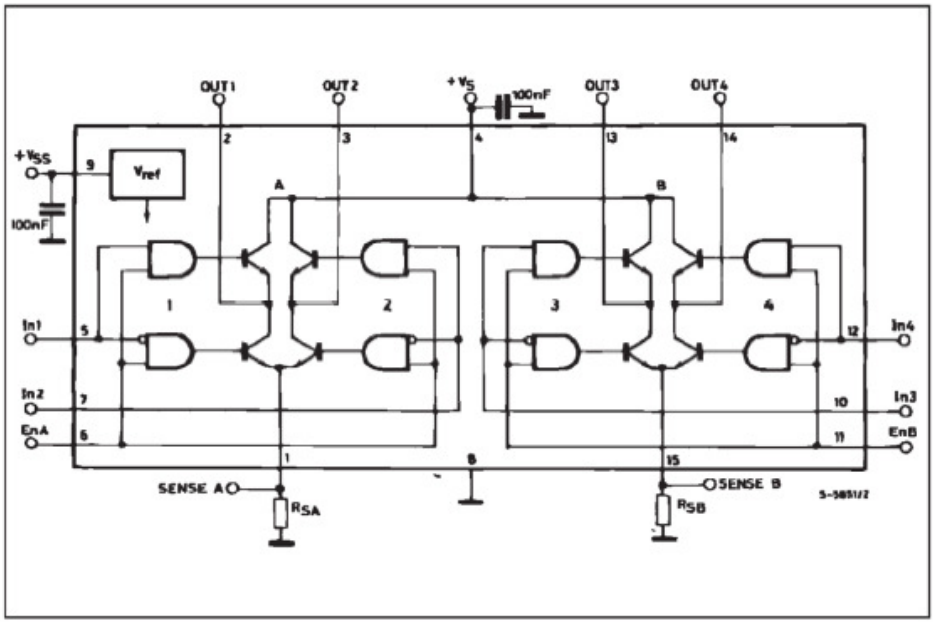
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

BLOCK DIAGRAM



9.3 Appendix C: C code file

```
/*

 -- Company : TEIS AB
 -- Engineer: Haydar Kamil Mousa, hma@foss.dk
 -- Create date : 2015-03-05
 -- Target Device : Altera Cyclone IV EP4CE115F29C7
 -- Tool version  : Quartus V13 and Eclipse
 --
 -- In_signals :
 -- Key_in     : Slide switches SW[3..1] in DE2-115 board. Note SW[0] is not used
 --
 -- Output_signals :
 -- LEDGs(3,2,1,0)  :LEDG control signals


*/

#include <io.h>
#include <alt_types.h>
#include <system.h>
#include <stdio.h>
#include "altera_avalon_pio_regs.h"


#define LEDG_0 0x1 ;  // define LEDG_0
#define LEDG_1 0x2 ;  // define LEDG_1
#define LEDG_2 0x4 ;  // define LEDG_2
#define LEDG_3 0x8 ;  // define LEDG_3

enum state { state_0 , state_1 , state_2 , state_3 } ; // Four state machines are used to generate sequences



void delay(q){    // Delay loop

    int i ;

    for(i = 0 ; i <= q ; i++){ // q has two values according to the SW3 (speed)

        }
}



void motor_stop(){  // phases = "0000" when stop = 1

    alt_u32 motor_phases ;

            IOWR_ALTERA_AVALON_PIO_DATA(PIO_OUT_LED_BASE , 0x0 );  // LEDG = "0000"

            IOWR_ALTERA_AVALON_PIO_DATA(PIO_2_BASE , 0x0 ); // sequences(motor_phases) = "0000"

             //motor_phases or sequences will be connected to GPIO JP5 (PIN_AD21 , PIN_AC21 , PIN_AB21 , PIN_AB22 respectively)
        }
```

```
int main()
{
    enum state current_state ;
    alt_u32  dir , start , speed , q  ; // initial variables
    current_state = state_0 ;  // initial variables

    while(1)
    {
        speed = 0x4 & IORD_ALTERA_AVALON_PIO_DATA(PIO_1_BASE) ; // speed will be connected to slide switch SW[3]
            if (speed == 0x4 )
                q = 500000 ; // This value will give delay to 1.0449 sec.
            else
                q = 5000 ; // This value will give delay to 10.519 msec.
        start = 0x2 & IORD_ALTERA_AVALON_PIO_DATA(PIO_1_BASE) ; // start/stop will be connected to slide switch SW[2]
  dir   = 0x1 & IORD_ALTERA_AVALON_PIO_DATA(PIO_1_BASE) ; // dir will be connected to slide switch SW[1]

if (start == 0x2){     // start = 1 , stop = 0 , to start/stop the running of stepper motor

    switch (current_state){

            case state_0 : if ( dir == 0x1 ){               // dir = 1 , in this case the stepper motor is running clockwise  //
                                    IOWR_ALTERA_AVALON_PIO_DATA(PIO_OUT_LED_BASE , 0x5 ); //  LEDG = "0101"
                                    IOWR_ALTERA_AVALON_PIO_DATA(PIO_2_BASE , 0x5 ); // motor_phases = "0101"
                                    //motor_phases = "0101";
                                    /* phases or sequences will be connected to GPIO JP5 (PIN_AD21 , PIN_AC21 , PIN_AB21 , PIN_AB22 respective
                                    delay(q) ;
                                    current_state = state_1 ;
                                     }
                           else if (dir == 0x0){             // dir = 0 , in this case the stepper motor is running counter clockwise //
```

```
                                IOWR_ALTERA_AVALON_PIO_DATA(PIO_OUT_LED_BASE , 0x9 ); // LEDG = "1001"

                                IOWR_ALTERA_AVALON_PIO_DATA(PIO_2_BASE , 0x9 ); // motor_phases = "1001"

                                //motor_phases = "1001" ;
                                // phases or sequences will be connected to GPIO JP5 (PIN_AD21 , PIN_AC21 , PIN_AB21 , PIN_AB22 respectively)

                                delay(q) ;

                                current_state = state_3 ;

                              }

                else

                                  current_state = state_0 ;


                break ;


 case state_1 : if ( dir == 0x1 ){

                                IOWR_ALTERA_AVALON_PIO_DATA(PIO_OUT_LED_BASE , 0x6 ); // LEDG = "0110"

                                IOWR_ALTERA_AVALON_PIO_DATA(PIO_2_BASE , 0x6 ); // motor_phases = "0110"

                                //motor_phases = "0110" ;
                                // phases or sequences will be connected to GPIO JP5 (PIN_AD21 , PIN_AC21 , PIN_AB21 , PIN_AB22 respectively)


                                delay(q) ;

                                current_state = state_2 ;

                              }

                else if (dir == 0x0){

                                 IOWR_ALTERA_AVALON_PIO_DATA(PIO_OUT_LED_BASE , 0x5 ); // LEDG = "0101"

                                 IOWR_ALTERA_AVALON_PIO_DATA(PIO_2_BASE , 0x5 ); // motor_phases = "0101"

                                 //motor_phases = "0101" ;
                                 // phases or sequences will be connected to GPIO JP5 (PIN_AD21 , PIN_AC21 , PIN_AB21 , PIN_AB22 respectively)

                                 delay(q) ;

                                 current_state = state_0 ;

                                }

                else

                                 current_state = state_1 ;

                break ;


case state_2 : if ( dir == 0x1 ){

                                IOWR_ALTERA_AVALON_PIO_DATA(PIO_OUT_LED_BASE , 0xA );  // LEDG = "1010"

                                IOWR_ALTERA_AVALON_PIO_DATA(PIO_2_BASE , 0xA ); // motor_phases = "1010"

                                //motor_phases = "1010" ;
                                // phases or sequences will be connected to GPIO JP5 (PIN_AD21 , PIN_AC21 , PIN_AB21 , PIN_AB22 respectively)

                                delay(q) ;

                                current_state = state_3 ;
                              }
```

```
                else if (dir == 0x0){

                                    IOWR_ALTERA_AVALON_PIO_DATA(PIO_OUT_LED_BASE , 0x6 );  // LEDG = "0110"

                                    IOWR_ALTERA_AVALON_PIO_DATA(PIO_2_BASE , 0x6 ); // motor_phases = "0110"

                                    //motor_phases = "0110" ;
                                    // phases or sequences will be connected to GPIO JP5 (PIN_AD21 , PIN_AC21 , PIN_AB21 , PIN_AB22 respectively)

                                    delay(q) ;

                                    current_state = state_1 ;

                                  }

                else

                                    current_state = state_2 ;

                break ;


    case state_3 : if ( dir == 0x1 ){

                                  IOWR_ALTERA_AVALON_PIO_DATA(PIO_OUT_LED_BASE , 0x9 );  // LEDG = "1001"

                                  IOWR_ALTERA_AVALON_PIO_DATA(PIO_2_BASE , 0x9 ); // motor_phases = "1001"

                                  //motor_phases = "1001" ;
                                  // phases or sequences will be connected to GPIO JP5 (PIN_AD21 , PIN_AC21 , PIN_AB21 , PIN_AB22 respectively)

                                  delay(q) ;

                                  current_state = state_0;

                                 }
                else if (dir == 0x0){

                                    IOWR_ALTERA_AVALON_PIO_DATA(PIO_OUT_LED_BASE , 0xA );  // LEDG = "1010"

                                    IOWR_ALTERA_AVALON_PIO_DATA(PIO_2_BASE , 0xA ); // motor_phases = "1010"

                                    //motor_phases = "1010" ;
                                    // phases or sequences will be connected to GPIO JP5 (PIN_AD21 , PIN_AC21 , PIN_AB21 , PIN_AB22 respectively)

                                    delay(q) ;

                                    current_state = state_2 ;

                                  }

                                    else

                                                        current_state = state_3 ;

                                    break ;


                }


          } else  motor_stop();

    }

    return 0 ;
}
```