Assignment No. C-1

Roll No:

Aim :

Develop Robotics(stepper motor) Application using Beagle Board.

Software Required :

• Linux Operating System

• GCC Compiler.

Hardware Required :

• Beaglebone Black/ ARM Cortex Processor

• Robotic Arm ( Stepper Motor )

• Interfacing cables

Theory :

Stepper Motor

A step motor can be viewed as a synchronous AC motor with the number of poles (on both rotor and stator) increased, taking care that they have no common denominator. Additionally, soft magnetic material with many teeth on the rotor and stator cheaply multiplies the number of poles (reluctance motor). Modern steppers are of hybrid design, having both permanent mag- nets and soft iron cores. A stepper motor is made up of a rotor, which is normally a permanent magnet and it is, as the name suggests the rotating component of the motor. A stator is another part which is in the form of winding. In the diagram below, the center is the rotor which is surrounded by the stator winding. This is called as four phase winding.

Unipolar vs Bipolar

nipolar motors use 6 wires and require a unipolar driver. In addition to the A and B phases, there are two extra wires called the common wires, Current

1

Figure 1: Stepping Sequence of Stepper Motor

always flows in one direction: from the phases, through the common wires. In addition, only one portion of the motor is energized at a time. Bipolar motors use 4 wires and require a bipolar drive. Common wires are not used. Current can flow in two directions. In addition, two phases can be energized at one time.

Working of Stepper Motor

The centre tap on the stator winding allows the current in the coil to change direction when the winding are grounded. The magnetic property of the stator changes and it will selectively attract and repel the rotor, thereby resulting in a stepping motion for the motor.

Stepping Sequence

In order to get correct motion of the motor, a stepping sequence has to be followed. This stepping sequence gives the voltage that must be applied to the stator phase. Normally a 4 step sequence is followed. When the sequence is followed from step 1 to 4, we get a clock wise rotation and when it is followed from step 4 to 1, we get a counter clockwise rotation.

2

Interfacing Diagram

Algorithm

1. Start the Kit with four electromagnets A , B , C , D .

2. Set P9-11 , P9-12 , P9-13 ,P9-14 as output pins.

3. Anticlockwise Rotation

• First iteration

• Set P9-11 HIGH

• Set P9-12 LOW

• Set P9-13 LOW

• Set P9-14 LOW

• Second iteration

• Set P9-11 LOW

• Set P9-12 HIGH

• Set P9-13 LOW

• Set P9-14 LOW

• Third iteration

• Set P9-11 LOW

• Set P9-12 LOW

• Set P9-13 HIGH

3

• Set P9-14 LOW

• Fourth iteration

• Set P9-11 LOW

• Set P9-12 LOW

• Set P9-13 LOW

• Set P9-14 HIGH

Mathematical Model

Let S be a set such that

S={s, e, i, o, f, DD, NDD, success, failure}

s= initial state

e = end state

i= input of the system.

o= output of the system.

f= functions

DD-Deterministic Data it helps identifying the load store functions or as- signment functions.

NDD- It is Non deterministic data of the system S to be solved.

Success-Stepper motor rotates in desired direction.

Failure-Desired outcome not generated or forced exit due to system error.

States: { S0 ,S1 , S2 , S3 , S4 , S5 }

S0: initial State (Power supply)

S1: Monitor Display editor (Write and Design application using gedit)

S2: BeagleBone Black

4

**Initial State**

S0

**Initializing beaglebone**

S4

**Writting code**

**Anti-Clockwise Rotation**

**Initializing S1 beaglebone**

S2

**Start Stepper Motor**

S3

**Writting code**

**Clockwise**

**S4 Rotation**

S3: stepper Motor

S4: Final State (clockwise Rotation)

S5: Final State (Anti-clockwise Rotation)

5

UML Diagrams :

Activity Diagram

6

Use-case Diagram

Class Diagram

CONCLUSION :

Hence, we have successfully developed and demonstrated Robotic Arm application using stepper motor with Beagleblack bone board..

7

Course Outcomes :

Output (Screenshots)

Minicom Terminal

8

Stepper Motor :

9

0.1 Code :

import Adafruit.BBIO\_GPIO as gpio import time

gpio.setup("P9\_11",gpio.OUT) gpio.setup("P9\_12",gpio.OUT) gpio.setup("P9\_13",gpio.OUT) gpio.setup("P9\_14",gpio.OUT)

while 1: gpio.output("P9\_11",gpio.HIGH) gpio.output("P9\_12",gpio.LOW) gpio.output("P9\_13",gpio.LOW) gpio.output("P9\_14",gpio.LOW) time.sleep(2) gpio.output("P9\_11",gpio.LOW) gpio.output("P9\_12",gpio.HIGH) gpio.output("P9\_13",gpio.LOW) gpio.output("P9\_14",gpio.LOW) time.sleep(2) gpio.output("P9\_11",gpio.LOW) gpio.output("P9\_12",gpio.LOW) gpio.output("P9\_13",gpio.HIGH) gpio.output("P9\_14",gpio.LOW) time.sleep(2) gpio.output("P9\_11",gpio.LOW) gpio.output("P9\_12",gpio.LOW) gpio.output("P9\_13",gpio.LOW) gpio.output("P9\_14",gpio.HIGH) time.sleep(2) ;clockwise over gpio.output("P9\_14",gpio.HIGH) gpio.output("P9\_13",gpio.LOW) gpio.output("P9\_12",gpio.LOW) gpio.output("P9\_11",gpio.LOW) time.sleep(2) gpio.output("P9\_14",gpio.LOW) gpio.output("P9\_13",gpio.HIGH) gpio.output("P9\_12",gpio.LOW) gpio.output("P9\_11",gpio.LOW) time.sleep(2)

10

gpio.output("P9\_14",gpio.LOW) gpio.output("P9\_13",gpio.LOW) gpio.output("P9\_12",gpio.HIGH) gpio.output("P9\_11",gpio.LOW) time.sleep(2) gpio.output("P9\_14",gpio.LOW) gpio.output("P9\_13",gpio.LOW) gpio.output("P9\_12",gpio.LOW) gpio.output("P9\_11",gpio.HIGH) time.sleep(2) ; anticlockwise over

11