A Laboratory Manual for

Programming Laboratory III

(310254) Semester – VI

(Computer Engineering)

Bachelor Degree in Engineering

UNIVERSITY OF PUNE, GANESHKHIND

ESTD - 1928

SNJB'S LATE SAU. KBJ COLLEGE OF ENGINEERING, CHANDWAD

DEPARTMENT OF COMPUTER ENGINEERING

LABORATORY MANUAL DEVELOPMENT PROJECT

Designations	Team for design
Project Institution	Shri Neminath Jain Bramhacharyashram's KBJ College of Engineering, Neminagar, Chandwad -
Project Commencement	June 2014
Head Of Institution	Dr.J J Chopde, SNJB's KBJ College of Engineering, Neminagar, Chandwad -423101.
Chief Project Coordinator	Prof. M .R. Sanghavi Head, Department Of Computer Engineering SNJB's, KBJ College of Engineering, Neminagar, Chandwad -423101.
Project Coordinator	Prof. Ms. K. S. Kotecha Associate Professor, Department Of Computer Engineering, SNJB's KBJ College of Engineering, Neminagar, Chandwad -423101.
Subject Expert	Prof. S.B. Ambhore and Prof. A.L. Maind Assistant Professor, Department of Computer Engineering, SNJB's KBJ College of Engineering, Neminagar, Chandwad -423101

SNJB'S LATE SAU. KBJ COLLEGE OF ENGINEERING, CHANDWAD

ESTD - 1928

DEPARTMENT OF COMPUTER ENGINEERING

Certificate

This is Certify that Mr./Ms._____Roll No _____ of Seventh Semester of Bachelor Engineering in Computer has completed the term work satisfactorily in Programming Laboratory III in the Academic Year 20_ to 20__ as prescribed in the curriculum.

Place:

Date: Exam Seat No._____

Subject Teacher Head of Department Principal

Stamp of Institution

List of Experiment and Record of Progressive Assessment

Serial No.	Name of Experiment	Page No	Date of Performan	Date of Submission	Assess ment	Signature of Faculty
Group A						
1	Develop an application using Beeglebone Black/ ARM Cortex A5 development board to simulate the operations of LIFT					
2	Develop an application using Beeglebone Black/ ARM Cortex A5 development board to simulate the working of signal lights.					
3	Implement an calculator (64 bit Binary Multiplication) application using concurrent lisp					
4	Apply the Following Software Engineering to all assignments(No 1,2,3 of Group A and B). Mathematical Modeling must result into UML Requirements. Apply Assignment No 4a to 4d for all Group A and Group B assignments of Embedded Operating system and Concurrent and Distributed Programming. Use tools Open source tools like ArgoUML, UMLLet, StarUML or equivalent tools for UML models) Or Use Agile or Scrum-Agile ethodologies and Tools.Use of Possitive and Negative Testing.					
4a	Design mathematical model of the Application/system using set theory, algebraic system, relations and functions, Deterministic and Non-Deterministic entities.					
4b	Analyze requirements from the Problem statement, mathematical model, Domain requirements and identify Functional, Non functional, Actors, Usecases for the application/system. Create usecase diagram, activity diagram/swimlane diagram for each usecase.					
4c	Design the architecture for the system /application using package diagram , deployment diagram. Design classes using class diagram.					
4d	Design the behavior of the system /application using state machine diagram and sequence diagram.					

5	Create Project plan, SRS, Design document and Test Plan for one group-C assignment from embedded operating system or Concurrent and Distributed Programming					
6	Write an application to parse input text file concurrently and compare the result of concurrent parsing with serial parsing (Use concurrent YACC parser)					

Group B						
1	Write an application to and demonstrate the change in BeagleBoard/ ARM Cortex A5 Microprocessor /CPU frequency or					
3	Vedic Mathematics method to find square of 2-digit number is used in a distributed programming. Use shared memory and distributed (multi-CPU) programming to complete the task.					
4	Implement a Parallel ODD-Even Sort algorithm using GPU or ARM equivalent.					
7	Implement *nxn* matrix parallel multiplication using CUDA/OpenCL GPU, use shared memory.					
8	Develop a network based application by setting IP address on BeagleBoard/ ARM Cortex A5.					
9	Implement a Multi-threading application for echo server using socket programming in JAVA					
10	Implement Reader-Writer problem using OPENMP					
Group C						
1	Develop Robotics(stepper motor) Application using Beagle Board.					
				Total:		

Assignment: 1

Regularity (2)	Performance(5)	Oral(3)	Total (10)	Dated Sign

Title of Assignment: Simulation of LIFT operations

Problem Definition: Develop an application using Beaglebone Black/ ARM Cortex A5 development board to simulate the operations of LIFT.

1.1 Perquisite: Beaglebone Black, minicom

1.2 Learning Objective: To simulate the operations of LIFT on BBB.

1.3 Relevant Theory / Literature Survey:

1.3 Introduction

The element14 BeagleBone Black is identical in technical design and functionality as the specified BeagleBoard.org product (BeagleBone Black) and runs on the version of the software provided by BeagleBoard.org to element14. General support for this board is available from the BeagleBoard.org community.

1.4 Setup:

1. Connect beaglebone black to PC with +5v supply and USB cable.
2. Open terminal in ubuntu.
3. Type sudo su ---enter
   Type password--enter
4. Then type minicom –s---enter
5. Goto serial port setup-
   - Press A
   - Change /dev/ttyxx to **/dev/ttyACM0**—enter
   - Press G
   - Press Enter
6. Save setup as dfl – enter
7. Goto EXIT – enter
8. Now system will boot and then type username and password
   and you booted into beaglebone black.
9. Type su –enter.

1.5 CONNECTOR DETAILS:

Connector Number	PIN Number	PIN Description	PIN connects to board	Function
P9	1,2	GND	GND(25pin)	GND
P9	3,4	VCC(3.6V)	VCC(26pin)	VCC
P8	7	GPIO2[2]	FRC-6pin	In
P8	8	GPIO2[3]	FRC-5pin	In
P8	9	GPIO2[5]	FRC-4pin	In
P8	10	GPIO2[4]	FRC-3pin	In
P8	11	GPIO1[13]	FRC-13pin	Out
P8	12	GPIO1[12]	FRC-14pin	Out
P8	13	GPIO0[23]	FRC-11pin	Out
P8	14	GPIO0[26]	FRC-12pin	Out
P8	15	GPIO1[15]	FRC-9pin	Out
P8	16	GPIO1[14]	FRC-10pin	Out
P8	17	GPIO0[27]	FRC-7pin	Out
P8	18	GPIO2[1]	FRC-8pin	Out
P8	19	GPIO0[22]	FRC-23pin	Out
P8	21	GPIO1[30]	FRC-24pin	Out
P9	11	GPIO0[30]	FRC-21pin	Out
P9	12	GPIO1[28]	FRC-22pin	Out
P9	13	GPIO0[31]	FRC-19pin	Out
P9	14	GPIO1[18]	FRC-20pin	Out
P9	15	GPIO1[16]	FRC-17pin	Out

P9	16	GPIO1[19]	FRC-18pin	Out
P9	23	GPIO1[17]	FRC-16pin	Out
P9	24	GPIO0[15]	FRC-15pin	Out

TABLE 1

1.6 How to get GPIO pin number:

- Once you have decided the pin number which you would like to use as a GPIO, you need to find out its corresponding reference number.
- For example, if you would like to use pin 12 on P8 expansion header, Then find out its default function. Note down the entire signal name. In this case, pin 12 is GPIO1_12.So any GPIO you come across would be referenced as GPIOX_Y. Identify X,Y.
- Use the formula below to find the corresponding reference number:

Reference number =((X*32)+ Y)

Hence, pin 12 would be referenced as gpio 44 in the kernel.

1.7 To use GPIO pin as GPIO in programs follow the steps :

- To make GPIO pin xx as output type command in terminal(xx—pin number)
  echo xx > /sys/class/gpio/export -- press enter
  echo out > /sys/class/gpio/gpioxx/direction --press enter
  for example:
  echo 44 > /sys/class/gpio/export
  echo out > /sys/class/gpio/gpio44/direction

- To make GPIO pin xx as input type command in terminal(xx—pin number)
  echo xx > /sys/class/gpio/export -- press enter
  echo in > /sys/class/gpio/gpioxx/direction --press enter
  for example:
  echo 44 > /sys/class/gpio/export
  echo in > /sys/class/gpio/gpio44/direction
- After making GIPO pin as output, To change the initial value of output pin type command in
  terminal(x—1/0,xx—pin number)
  echo x > /sys/class/gpio/gpioxx/value –press enter
  for example:
  echo 0 > /sys/class/gpio/gpio44/value
  OR
  echo 1 > /sys/class/gpio/gpio44/value

1.8 PROCEDURE:

1. Make a connection as shown in above Table1.
2. Make pin as input or output as told in function block of Table1.

Export pins :

echo 45 > /sys/class/gpio/export

3. Make New folder with name Lift elevator and go to that folder using command

mkdir lift_elevator—press enter

cd lift_elevator—press enter

4. Create and open a file with command

vim lift_elevator.cpp

5. Write a program and save it.
6. Now compile and run the program using command

g++ lift_elevator.cpp -o lift_elevator –press enter

./lift_elevator –press enter

1.9 Oral Questions:

1) What are the steps to enter BBB?
2) What is difference between OS and EOS?

Assignment: Group A2

Regularity (2)	Performance(5)	Oral(3)	Total (10)	Dated Sign

Title of Assignment: Simulation of signal lights operations

Problem Definition: Develop an application using Beeglebone Black/ ARM Cortex A5 development board to simulate the working of signal lights.

2.1 Perquisite: Beaglebone Black, minicom

2.2 Learning Objective: To simulate the operations of signal lights operations on BBB.

2.3 Relevant Theory / Literature Survey:

2.3.1 Introduction:

The element14 BeagleBone Black is identical in technical design and functionality as the specified BeagleBoard.org product (BeagleBone Black) and runs on the version of the software provided by BeagleBoard.org to element14. General support for this board is available from the BeagleBoard.org community.

2.3.2 Setup:

2. Connect beaglebone black to PC with +5v supply and USB cable.
3. Open terminal in ubuntu.
3. Type sudo su ---enterType password.
4. Then type minicom –s---enter
5. Goto serial port setup-
   - Press A
   - Change /dev/ttyUSBxx to /dev/ttyACM0—enter
   - Press G
   - Press Enter
6. Save setup as dfl – enter
7. Goto EXIT – enter
8. Now system will boot and then type username and password and you booted into beaglebone black.
9. Type su –enter.

2.4 CONNECTOR DETAILS:

Connector Number	PIN Number	PIN Description	PIN connects to accessory board	Function
P9	1,2	GND	GND(25pin)	GND
P9	5	VCC(5V)	VCC(26pin)	VCC
P9	11	GPIO0[30]	FRC21pin	Out
P9	12	GPIO1[28]	FRC-22pin	Out
P9	13	GPIO0[31]	FRC-19pin	Out
P9	14	GPIO1[18]	FRC-20pin	Out
P9	15	GPIO1[16]	FRC-17pin	Out
P9	16	GPIO1[19]	FRC-18pin	Out
P9	24	GPIO0[15]	FRC-15pin	Out
P9	23	GPIO1[17]	FRC-16pin	Out
P8	11	GPIO1[13]	FRC-13pin	Out
P8	12	GPIO1[12]	FRC-14pin	Out
P8	13	GPIO0[23]	FRC-11pin	Out
P8	14	GPIO0[26]	FRC-12pin	Out
P8	15	GPIO1[15]	FRC-9pin	Out
P8	16	GPIO1[14]	FRC-10pin	Out
P8	17	GPIO0[27]	FRC-7pin	Out
P8	18	GPIO2[1]	FRC-8pin	Out

TABLE 1

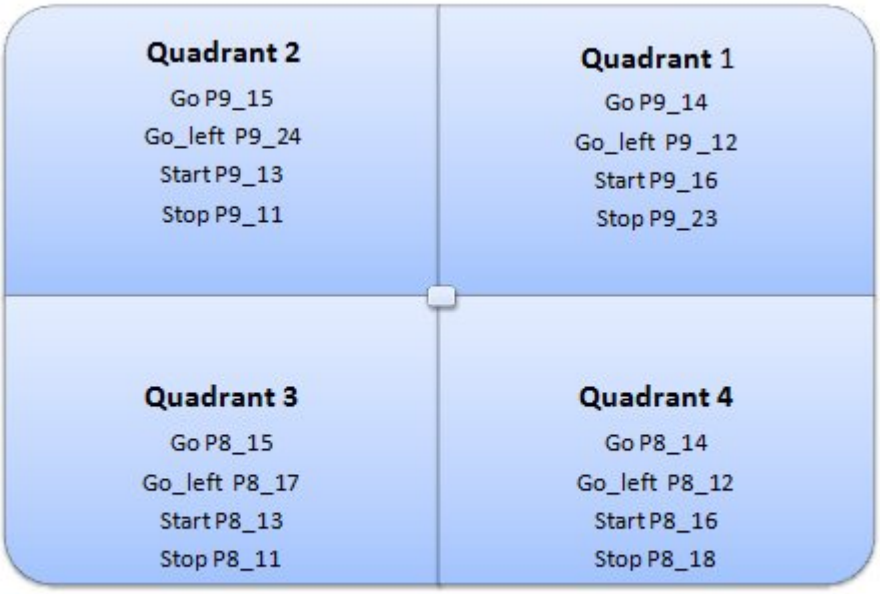


Fig. Traffic Light

2.5 How to get GPIO pin number:

- Once you have decided the pin number which you would like to use as a GPIO, you need to find out its corresponding reference number.
- For example, if you would like to use pin 12 on P8 expansion header, Then find out its default function. Note down the entire signal name. In this case, pin 12 is GPIO1_12.So any GPIO you come across would be referenced as GPIOX_Y. Identify X,Y.
- Use the formula below to find the corresponding reference number:
  Reference number =((X*32)+ Y)
  Hence, pin 12 would be referenced as gpio 44 in the kernel.

2.6 To use GPIO pin as GPIO in programs follow the steps:

- To make GPIO pin xx as output type command in terminal(xx—pin number)
  echo xx > /sys/class/gpio/export -- press enter
  echo out > /sys/class/gpio/gpioxx/direction --press enter
  for example:
  echo 44 > /sys/class/gpio/export
  echo out > /sys/class/gpio/gpio44/direction

- To make GPIO pin xx as input type command in terminal(xx—pin number)
  echo xx > /sys/class/gpio/export -- press enter
  echo in > /sys/class/gpio/gpioxx/direction --press enter
  for example:
  echo 44 > /sys/class/gpio/export
  echo in > /sys/class/gpio/gpio44/direction
- After making GIPO pin as output, To change the initial value of

output pin type command in terminal(x—1/0,xx—pin number)
echo x > /sys/class/gpio/gpioxx/value –press enter
for example:
echo 0 > /sys/class/gpio/gpio44/value
OR
echo 1 > /sys/class/gpio/gpio44/value

2.7 PROCEDURE:

1. Make a connection as shown in above Table1.
2. Make pin as input or output as told in function block of Table1.
   Export gpio pins:
   echo 45 > /sys/class/gpio/export
   Set direction for pins:
   echo out > /sys/class/gpio/gpio45/direction
3. Make pin as input or output as told in function block of Table1.
4. Make New folder with name Lift elevator and go to that folder using command
   mkdir traffic_light—press enter
   cd lift_traffic_light—press enter
5. create and open a file with command
   vim traffic_light.c
6. write a program and save it.
   Now compile and run the program using command
   gcc traffic_light.c -o traffic_light –press enter
   ./traffic_light –press enter
   OR
   g++ stepper.cpp –o stepper –press enter
   ./stepper -- enter

2.8 Oral Questions:

3) How to activate any pin as Input or output? Using command and through program?
4) What is GPIO?
5) How many pins are there of GPIO ?
6) What is minicom?

Assignment: Group A 6

Regularity (2)	Performance(5)	Oral(3)	Total (10)	Dated Sign

Title of Assignment: Lex and Yacc

Problem Definition: Write an application to parse input text file concurrently and compare the result of concurrent parsing with serial parsing (Use concurrent YACC parser)

6.1 Perquisite: lex and yacc, openmp

6.2 Learning Objective: To achieve concurrent parsing and compare result with serial .

6.3 Relevant Theory / Literature Survey:

6.3.1 Introduction
Compilation steps: following diagram shows the compilation steps.

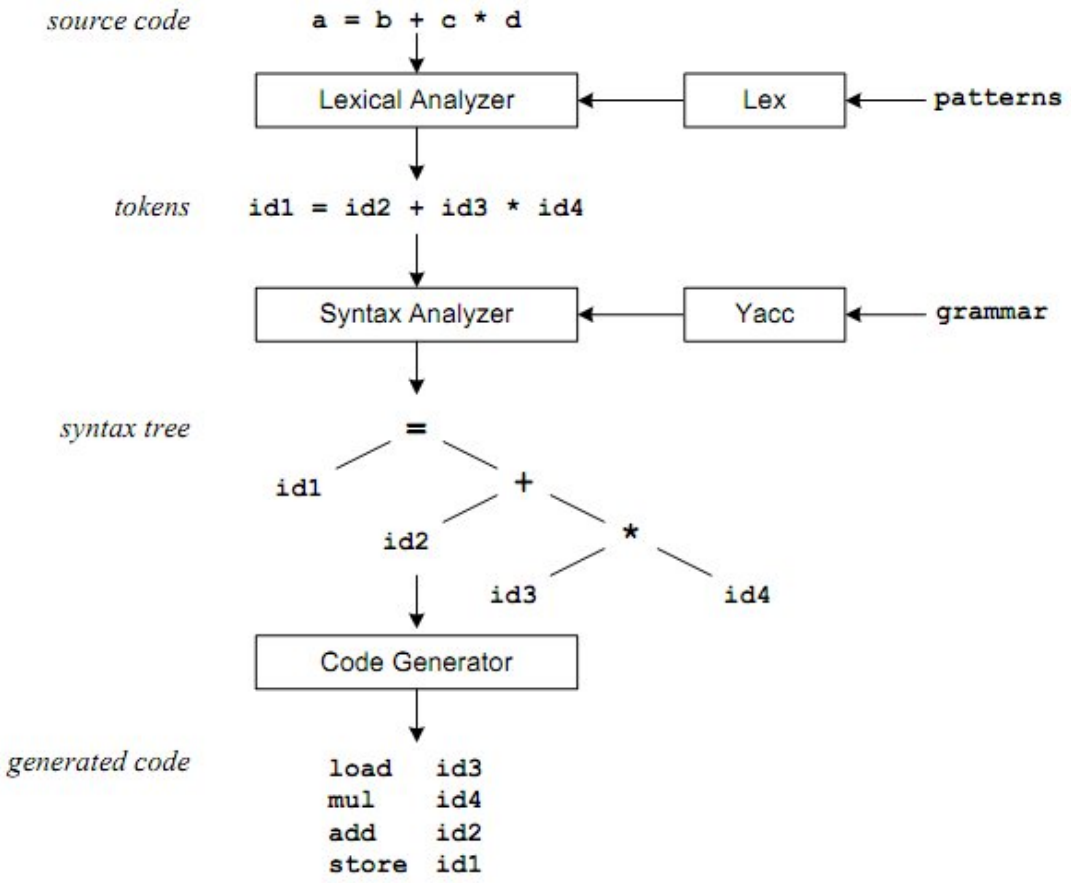


Figure 1: Compilation Sequence

The **patterns** in the above diagram is a file you create with a text editor. Lex will read your patterns and generate C code for a lexical analyzer or scanner. The lexical analyzer matches strings in the input, based on your patterns, and converts the strings to tokens. Tokens are numerical representations of strings, and simplify processing. When the lexical analyzer finds identifiers in the input stream it enters them in a symbol table. The symbol table may also contain other information such as data type (integer or real) and location of each variable in memory. All subsequent references to identifiers refer to the appropriate symbol table index.

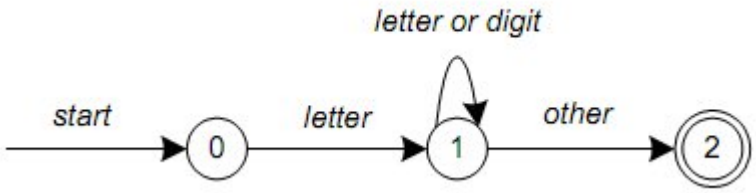
The **grammar** in the above diagram is a text file you create with a text edtior. Yacc will read your grammar and generate C code for a syntax analyzer or parser. The syntax analyzer uses grammar rules that allow it to analyze tokens from the lexical analyzer and create a syntax tree. The syntax tree imposes a hierarchical structure the tokens. For example, operator precedence and associatively are apparent in the syntax tree. The next step, code generation, does a depth-first walk of the syntax tree to generate code. Some compilers produce machine code, while others, as shown above, output assembly language.

6.4 Lex Theory

During the first phase the compiler reads the input and converts strings in the source to tokens. With regular expressions we can specify patterns to lex so it can generate code that will allow it to scan and match strings in the input. Each pattern specified in the input to lex has an associated action. Typically an action returns a token that represents the matched string for subsequent use by the parser. Initially we will simply print the matched string rather than return a token value. The following represents a simple pattern, composed of a regular expression,

that scans for identifiers. Lex will read this pattern and produce C code for a lexical analyzer that scans for identifiers.

letter(letter|digit)*



Metacharacter	Matches
.	any character except newline
\n	newline
*	zero or more copies of the preceding expression
+	one or more copies of the preceding expression
?	zero or one copy of the preceding expression
^	beginning of line
$	end of line
a|b	a or b
(ab)+	one or more copies of ab (grouping)
"a+b"	literal "a+b" (C escapes still work)
[]	character class

Table 1. Pattern matching

Expression	Matches
abc	abc
abc*	ab abc abcc abccc ...
abc+	abc abcc abccc ...
a(bc)+	abc abcbc abcbcbc ...
a(bc)?	a abc
[abc]	one of: a, b, c
[a-z]	any letter, a-z
[a\-z]	one of: a, -, z
[-az]	one of: -, a, z
[A-Za-z0-9]+	one or more alphanumeric characters
[\t\n]+	whitespace
[^ab]	anything except: a, b
[a^b]	one of: a, ^, b
[a|b]	one of: a, |, b
a|b	one of: a, b

Table 2.Pattern matching examples

6.5 Yacc Theory

Grammars for yacc are described using a variant of Backus Naur Form (BNF). This technique, pioneered by John Backus and Peter Naur, was used to describe ALGOL60. A BNF grammar can be used to express *context-free* languages. Most constructs in modern programming

languages can be represented in BNF. For example, the grammar for an expression that multiplies and adds numbers is

E -> E + E E -> E * E E -> id

Three productions have been specified. Terms that appear on the left-hand side (lhs) of a production, such as **E** (expression) are nonterminals. Terms such as **id** (identifier) are terminals (tokens returned by lex) and only appear on the right-hand side (rhs) of a production. This grammar specifies that an expression may be the sum of two expressions, the product of two expressions, or an identifier. We can use this grammar to generate expressions:

E -> E * E (r2) -> E * z (r3) -> E + E * z (r1) -> E + y * z (r3) -> x + y * z (r3)

At each step we expanded a term and replace the lhs of a production with the corresponding rhs. The numbers on the right indicate which rule applied. To parse an expression we need to do the reverse operation. Instead of starting with a single nonterminal (start symbol) and generating an expression from a grammar we need to reduce an expression to a single nonterminal. This is known as *bottom-up* or *shift-reduce* parsing and uses a stack for storing terms.

6.6 Procedure Run Program for serial parsing:

1. **$lex abc.l**
2. **$yacc –d abc.y**
3. **$gcc y.tab.c lex.yy.c**
4. **$./a.out file1.txt**

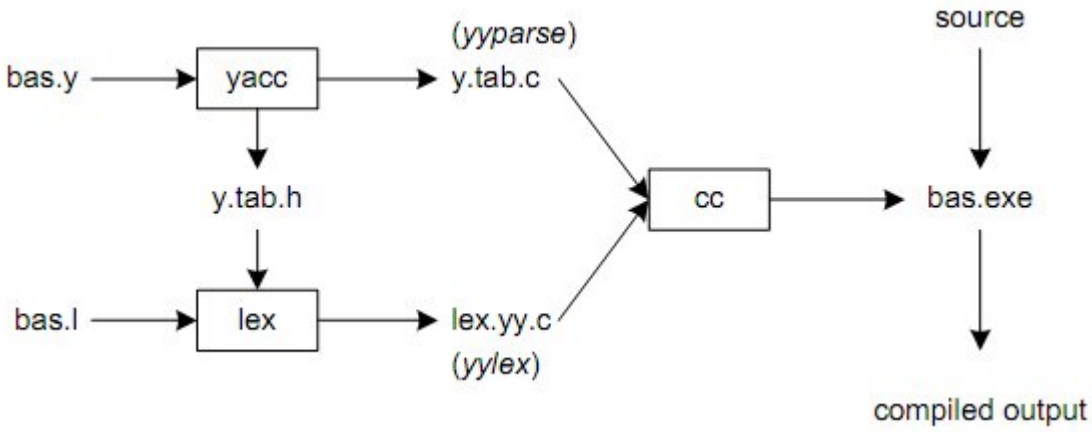


Figure 2. Building a Compiler with Lex/Yacc

1.8 Procedure Run Program for concurrent parsing:

1. **$lex abc.l**
2. **$yacc –d abc.y**
3. **$gcc y.tab.c lex.yy.c –ll**
4. **$gcc openmp.c –fopenmp –o p**
5. **./p file1.txt file2.txt file3.txt**

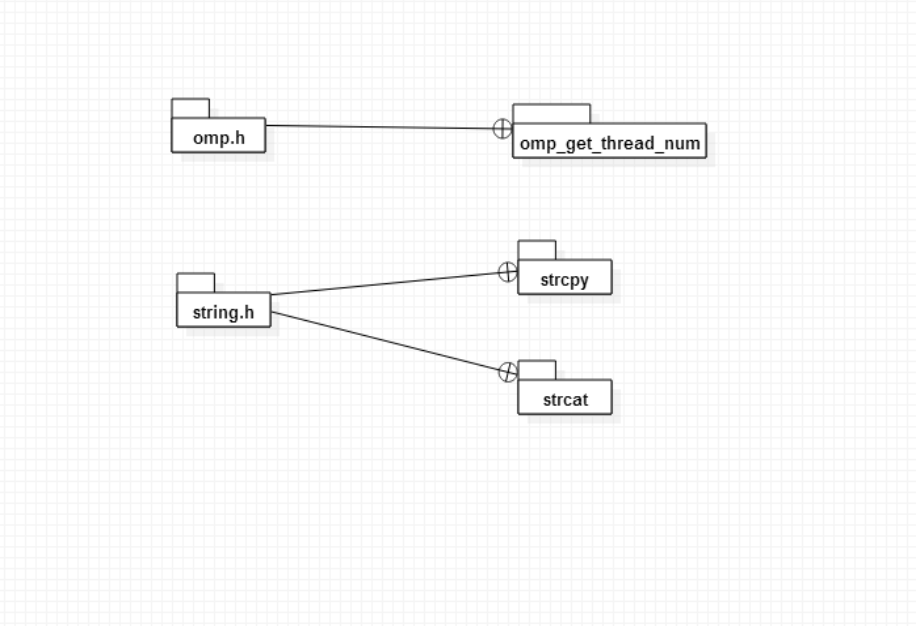


Figure 1:Package diagram

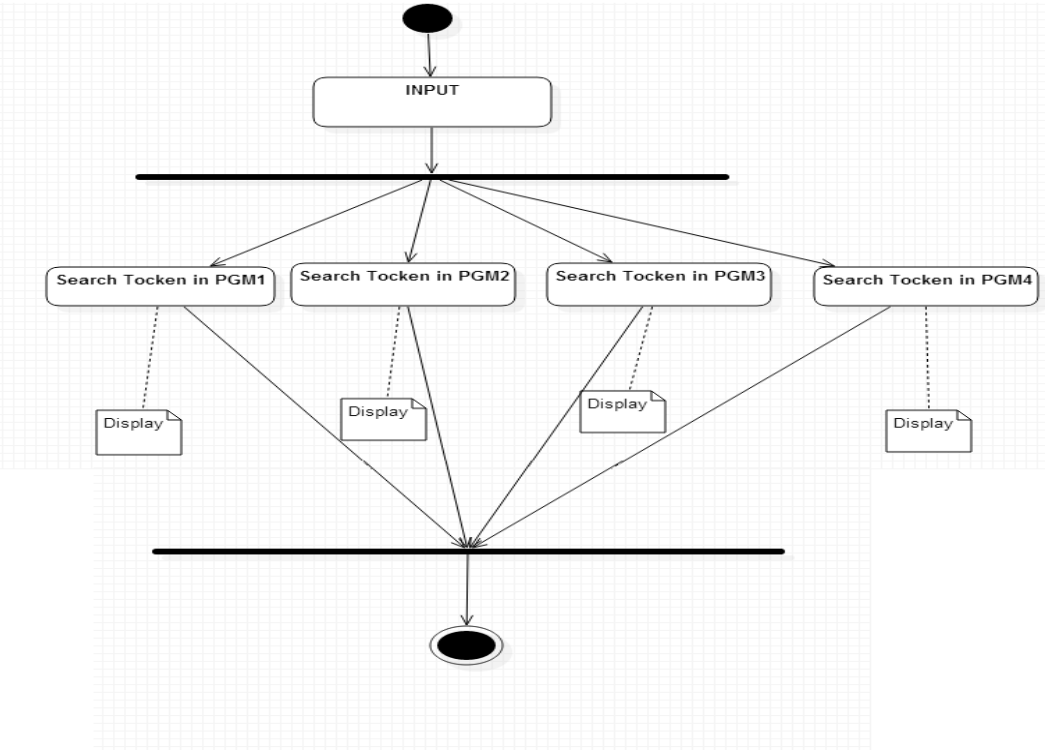


figure 2:Activity diagram with fork

figure 3:Activity diagram

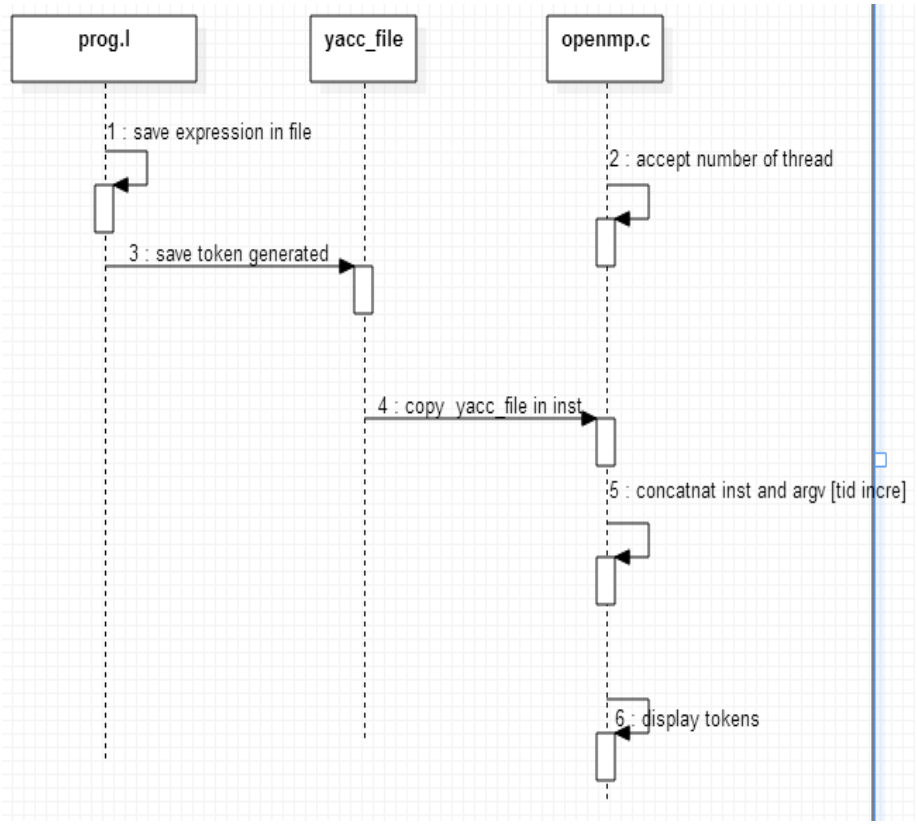


figure 4:Sequence diagram

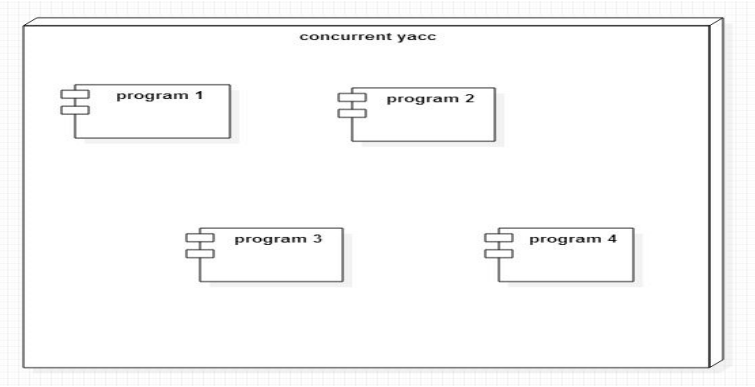


figure 5:Deployment diagram

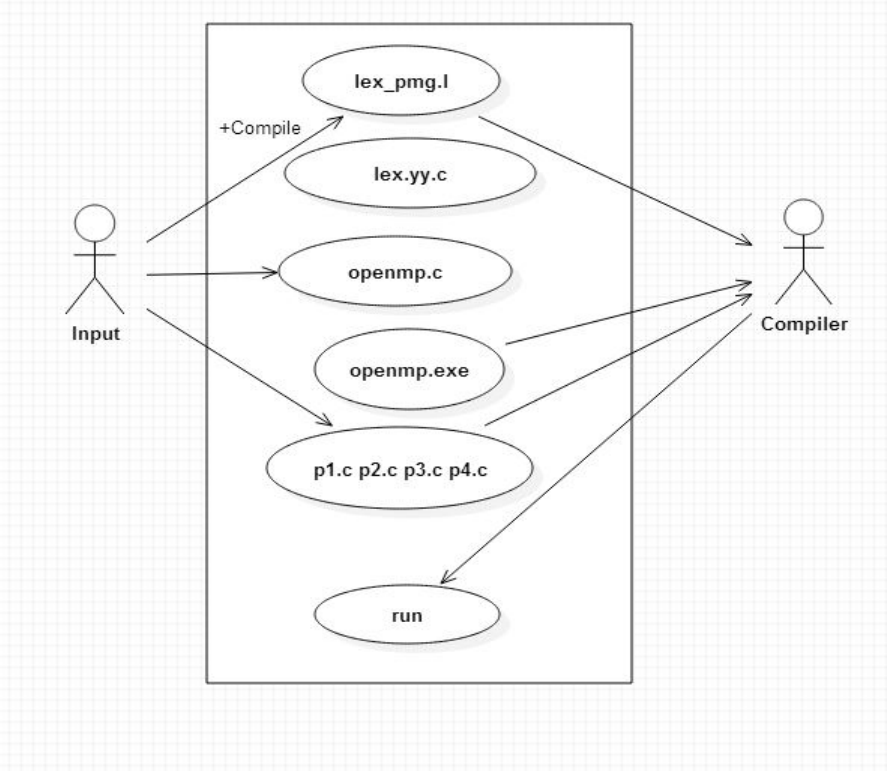


figure 6: Use Case diagram

1.9 Oral Questions:

7) What is lex?
8) What is yacc?
9) What is difference between serial and concurrent execution and how you have achieved in your program?

Assignment: B1

Regularity (2)	Performance(5)	Oral(3)	Total (10)	Dated Sign

Title of Assignment: PWM

Problem Definition: Write an application to and demonstrate the change in BeagleBoard/ ARM Cortex
A5 /Microprocessor /CPU frequency or square wave of programmable frequency.

1.1 Perquisite: Beaglebone Black, minicom

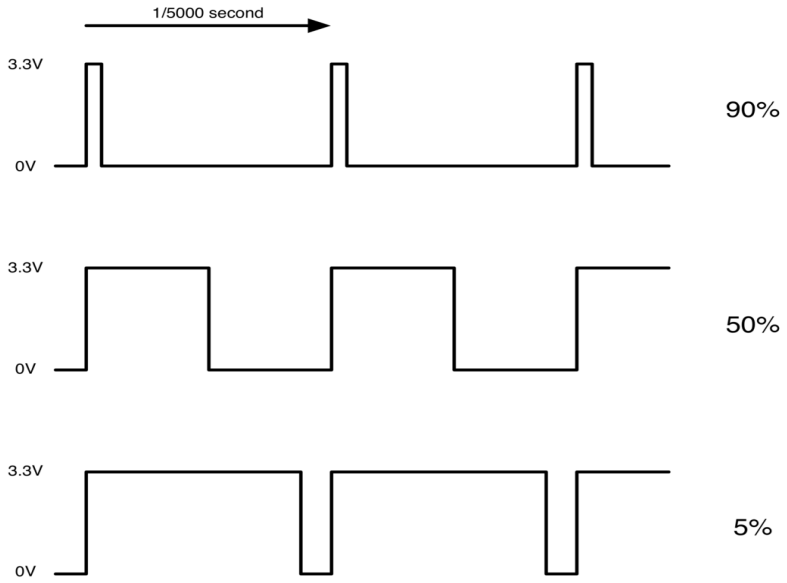
1.2 Learning Objective: To simulate the operations of LIFT on BBB.

1.3 Relevant Theory / Literature Survey:

1.3 Introduction

The element14 BeagleBone Black is identical in technical design and functionality as the specified BeagleBoard.org product (BeagleBone Black) and runs on the version of the software provided by BeagleBoard.org to element14. General support for this board is available from the BeagleBoard.org community.

Pulse Width Modulation (or PWM) is a technique for controlling power. We also use it here to control the brightness of each of the LEDs.
The diagram below shows the signal from one of the GPIO pins on the BBB.



Every 1/5000 of a second, the PWM output will produce a pulse from 3.3V down to 0V. The length of this pulse is controlled by the 'set_duty_cycle' function.
If the output is high for 90% of the time then the load will get 90% of the power delivered to it. We cannot see the LEDs turning on and off at that speed, so to us, it just looks like the brightness is different.

1.4 Configuring Pin 13 on P8 Header for PWM

The BBB was specifically designed to be a dynamic piece of hardware, enabling third-party developers to create their own custom configurations and extensions known as c*apes*. The board is so flexible, it can change its hardware configuration at runtime using an in-kernel mechanism known as the C*ape Manager* in conjunction with *Device Tree Overlays*.The *Device Tree* consists of a set of human readable text files known as DTS files that end in the ".dts" extension. DTS files can be edited using a simple text editor to set the configuration for a particular pin. These source files then get compiled into DTB files, a binary format ending in the ".dtbo" extension. This process creates what are known as *device tree fragments* or o*verlays.* The kernels *Cape Manager* can then dynamically load and unload the DTB files post-boot as well as at runtime to set the hardware configuration.

```
1 root@beaglebone:/lib/firmware# ls *pwm*
2 am33xx_pwm-00A0.dtbo
3 am33xx_pwm-00A0.dts
4 bone_pwm_P8_13-00A0.dtbo
5 bone_pwm_P8_13-00A0.dts
6 ...
```

All we need to do is load them using the *Cape Manager*, but first, lets get acquainted with a very useful command that will help us determine if our overlays were properly loaded:

```
1 root@beaglebone:~# more /sys/devices/bone_capemgr.8/slots
2 0: 54:PF---
3 1: 55:PF---
4 2: 56:PF---
5 3: 57:PF---
6 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
7 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
```

Out of the box, the BBB shows the above slots. Lets add two more slots to configure pin 13 on header P8 for PWM by executing the following commands:

```
1 root@beaglebone:~# echo am33xx_pwm > /sys/devices/bone_capemgr.8/slots
2 root@beaglebone:~# echo bone_pwm_P8_13 > /sys/devices/bone_capemgr.8/slots
```

To confirm the overlays loaded properly we run the slots command again:

```
1 root@beaglebone:~#  more /sys/devices/bone_capemgr.8/slots
2 0: 54:PF---
3 1: 55:PF---
4 2: 56:PF---
5 3: 57:PF---
6 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
7 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
8 7: ff:P-O-L Override Board Name,00A0,Override Manuf,am33xx_pwm
9 8: ff:P-O-L Override Board Name,00A0,Override Manuf,bone_pwm_P8_13
```

Our board is now configured for PWM on pin13 of the P8 header! Before we move on, however, it's worth noting these changes are not permanent. If you power off the board, the PWM slots we just added will disappear. Thankfully, we don't have to repeat the above steps each time we power up the board. The Cape Manager supports a method to load the overlays at boot time by adding the following argument to the "uEnv.txt" file:

```
1 capemgr.enable_partno=am33xx_pwm,bone_pwm_P8_13
```

Make sure to append the argument in a single line like this:

```
1 root@beaglebone:~# more /media/BEAGLEBONE/uEnv.txt
2 optargs=quiet drm.debug=7 capemgr.enable_partno=am33xx_pwm,bone_pwm_P8_13
```

1.5 Oral Questions:

1) What are the PWM?
2) What is use of PWM and what are the pins numbers used in BBB?

3) What is duty cycle?

Assignment: Group C1

Regularity (2)	Performance(5)	Oral(3)	Total (10)	Dated Sign

Title of Assignment: Stepper Motor

Problem Definition: Develop Robotics(stepper motor) Application using Beagle Board.

1.1 Perquisite: Beaglebone Black, minicom

1.2 Learning Objective: To simulate the operations of LIFT on BBB.

1.3 Relevant Theory / Literature Survey:

1.3 Introduction

The element14 BeagleBone Black is identical in technical design and functionality as the specified BeagleBoard.org product (BeagleBone Black) and runs on the version of the software provided by BeagleBoard.org to element14. General support for this board is available from the BeagleBoard.org community.

1.4 CONNECTOR DETAILS:

Connector Number	PIN Number	PIN Description	PIN connects to board	Function
P9	1,2	GND	GND(25pin)	GND
P9	5	VCC(5V)	VCC(26pin)	VCC
P9	11	GPIO0[30]	FRC-21pin	Out
P9	12	GPIO1[28]	FRC-22pin	Out
P9	13	GPIO0[31]	FRC-19pin	Out
P9	14	GPIO1[18]	FRC-20pin	Out

Table 1: Pins for stepper motor

1.5 **How Stepper Motors Work**

Stepper motors consist of a permanent magnetic rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator. Figure 1 illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.

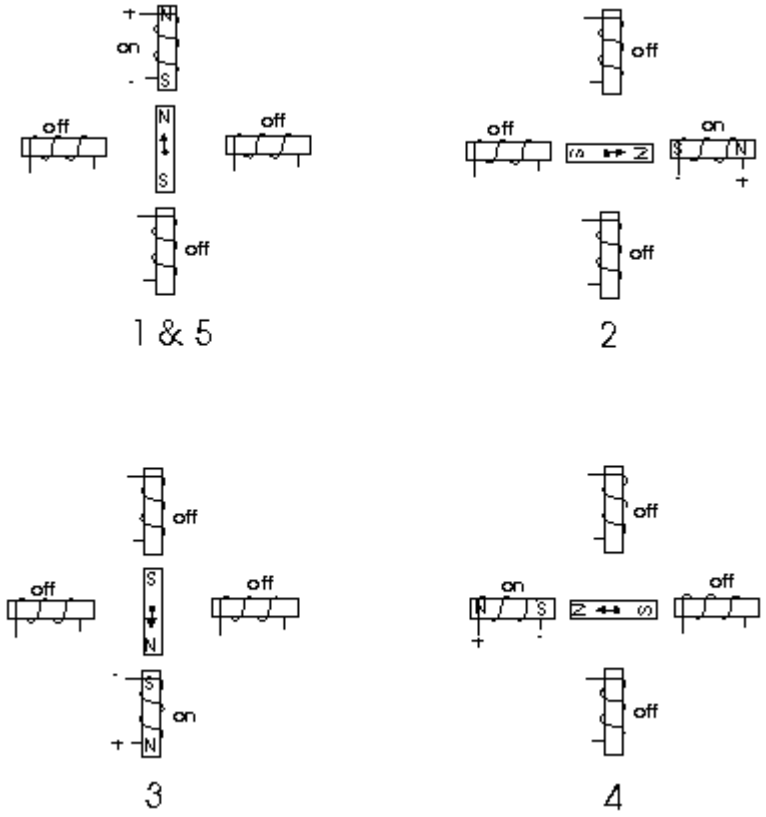


Figure 1

In the above example, we used a motor with a resolution of 90 degrees or demonstration purposes. In reality, this would not be a very practical motor for most applications. The average stepper motor's resolution -- the amount of degrees rotated per pulse -- is much higher than this. For example, a motor with a resolution of 5 degrees would move its rotor 5 degrees per step, thereby requiring 72 pulses (steps) to complete a full 360 degree rotation.

You may double the resolution of some motors by a process known as "half-stepping". Instead of switching the next electromagnet in the rotation on one at a time, with half stepping you turn on both electromagnets, causing an equal attraction between, thereby doubling the resolution. As you can see in Figure 2, in the first position only the upper electromagnet is active, and the rotor is drawn completely to it. In position 2, both the top and right electromagnets are active, causing the rotor to position itself between the two active poles. Finally, in position 3, the top

magnet is deactivated and the rotor is drawn all the way right. This process can then be repeated for the entire rotation.

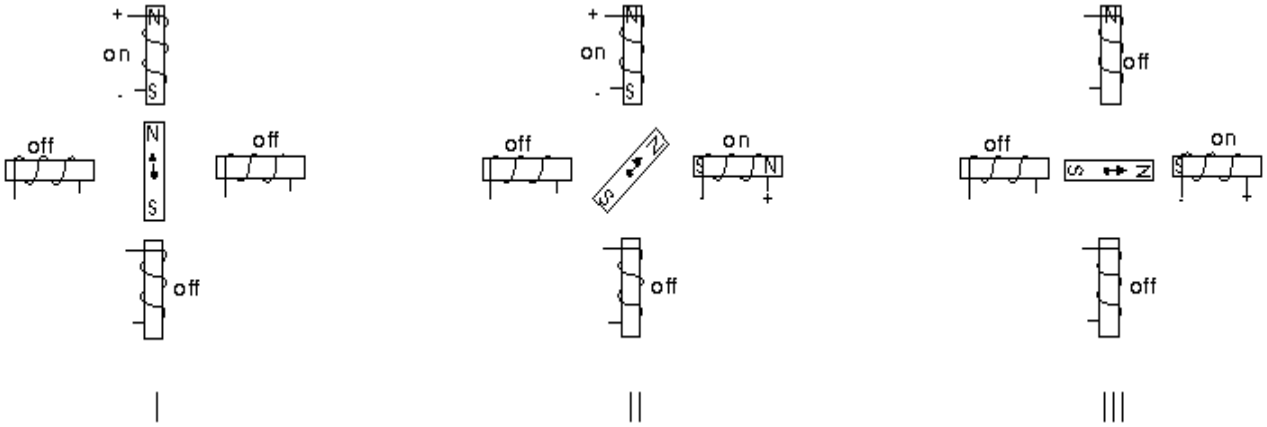


Figure 2

There are several types of stepper motors. 4-wire stepper motors contain only two electromagnets, however the operation is more complicated than those with three or four magnets, because the driving circuit must be able to reverse the current after each step. For our purposes, we will be using a 6-wire motor.

The specific stepper motor we are using for our experiments (ST-02: 5VDC, 5 degrees per step) has 6 wires coming out of the casing. If we follow Figure 3, the electrical equivalent of the stepper motor, we can see that 3 wires go to each half of the coils, and that the coil windings are connected in pairs. This is true for all four-phase stepper motors.

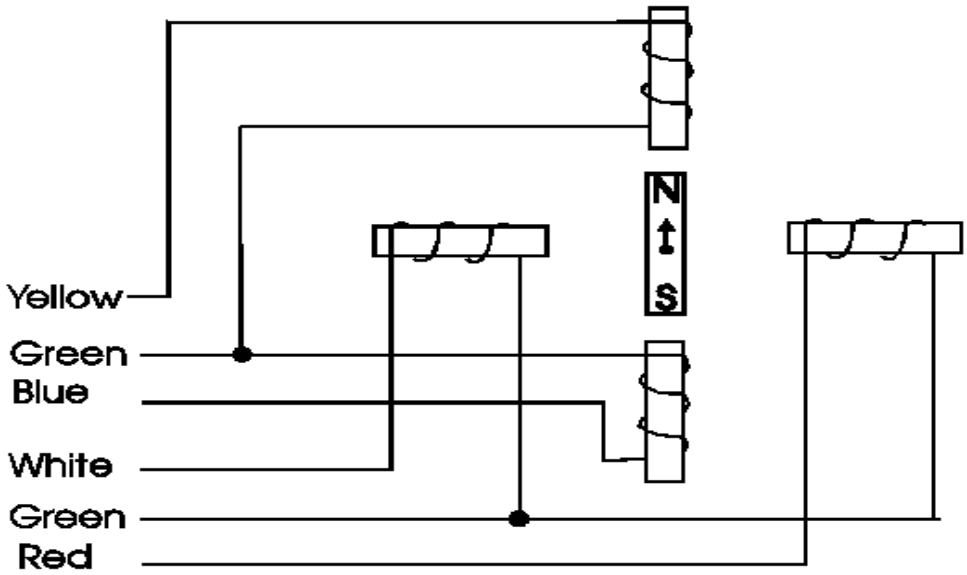


Figure 3

However, if you do not have an equivalent diagram for the motor you want to use, you can make a resistance chart to decipher the mystery connections. There is a 13 ohm resistance between the center-tap wire and each end lead, and 26 ohms between the two end leads. Wires originating from separate coils are not connected, and therefore would not read on the ohm meter.

1.6 Advantages and disadvantages of stepper motors

The reason for using a stepper motor is to achieve precise control: you can make it move through a defined angle.

Stepper motors can sometimes be quite jerky, because they start and stop each step with a sudden impulse, which isn't always what you want if you're trying to build a precision machine.

1.7 Oral Questions:

1) How stepper motor works?
2) What are the application of stepper motor?