Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

School of Electrical and Computer Engineering

Third Year B. Tech. (EE)

Microcontroller and Applications
Course Code: ECE2003B

Laboratory Manual

2023-2024

Preface

Microcontroller and Applications

Microcontrollers are single chip computers, integrating processor, memory and other peripheral modules into a single System-on-Chip (SoC). The single chip solution makes the footprint of the computational element small in the overall system package, eliminating the necessity of additional chips on board. Microcontrollers like 8051, PIC belong to this category. This course includes Silicon Labs' 8-bit Microcontroller C8051F340 which implements the standard 8051 organization. It introduces undergraduate students to the field of microcontrollers – what they are, how they work, how they interface with their I/O components, and what considerations the programmer has to observe in hardware-based and embedded programming. Understanding the architecture is the basis followed by hardware intricacies of these processors and their programming will be covered. Different system design examples built around these processors will also be elaborated. The lab for this course covers experiments based on interfacing of peripherals with C8051F340. Programming is in assembly and embedded C. IDE is Keil µvision and Simplicity Studio. After completion of this lab, students should able to develop a microcontroller-based interfaces for real life applications.

Microcontroller and Applications
I N D E X

CERTIFICATE

Certified that Mr./Ms._____ of Class **T. Y. B. Tech.** Division_____ Roll No._____ has completed the laboratory work in the subject **Microcontroller and Applications** during the **Semester V** in the School of Electrical and Computer Engineering during the Academic Year 2023-2024.

Signature of the Faculty **Head of the School**

Laboratory Instructions

Microcontroller and Applications

1. Students should have a valid ID card before entering the laboratory.
2. Playing of games on computer in the lab is strictly prohibited.
3. Before leaving the lab, users must close all programs/windows and turn-off the computer.
4. Do not modify or delete any important file and install any software or settings in the computer
5. Internet facility is to be used only for educational/ study purpose.
6. If any problem arises, please bring the same to the notice of Lab In-Charge.
7. Each student must take mobile phones in “Switched Off” or “Vibration” mode while entering and or working in laboratory.
8. In case of theft / destruction of the computers or peripherals, double the cost of that item will be charged from the student/user.
9. Tampering with the hardware or software settings will not be tolerated.
10. Lab Assistants are available to assist with basic computer and software problems.
11. Copying and/or installing of pirated software not permitted.
12. Personal files are not to be stored on the local drive. Students are responsible for their own means of digital storage. All lab computers are configured to remove any data stored or any programs installed by users.
13. Do not leave your personal belongings at the computer desk. The School is not responsible for items left behind.
14. Users are strictly prohibited from downloading, viewing or distributing any offensive materials.
15. Internet surfing or chatting for personal reasons is not allowed.

Guidelines to use µVISION IDE

Keil MicroVision is an Integrated Development Environment (IDE), which includes a text editor to write programs, a compiler to convert the source code to hex files and a debugger to test, verify, and optimize the application code.

Keil µVision can be used for:

- Writing programs in C/C++ or Assembly language
- Compiling and Assembling Programs
- Debugging program
- Creating Hex and Axf file
- Testing the program without real Hardware (Simulator Mode)

Step 1: After opening Keil uV, Go to **Project** tab and

Create new µVision project

Now Select new folder and give name to Project.

Step 2: After Creating project now **Select your device model** eg. Silicon Lab's C8051F340

Step 3: So now your project is created and **Message** window will appear to add start-up file of your device. Click on **Yes** so it will be added to your project folder you are programming in C and click on no if you are programming in assembly.

Step 4: Now go to File and create new file and save it with **.asm** extension in the folder where you created the project if you want to write program in assembly language

Step 5: Now write your program and save it again.

Step 6: After that on left you see project window [if it's not there….go to View tab and click on restore window to default].

Now come on Project window.

Right click on target and click on **options for target**

Here you can change your device also.

Step 7: Now Expand target and you will see source group

Right click on group and click on **Add files to source group**

Now add your program file which you have written in C/assembly.

You can see program file added under source group.

Step 8: Now Click on **Build target**. You can find it under Project tab or in toolbar. It can also be done by pressing **F7** key.

Step 9: you can see Status of your program in **Build output** window

[If it's not there go to view and click on Build output window]

Now you are done with your program.

Keil is a tool which can be used as debugger and simulator.

NOTE: *Use this tool for Experiment No. 1 and 2.*

T. Y. B. Tech (Electrical and Computer Engineering)

Trimester: V **Subject: Microcontroller and Applications**
Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 01 and 02

Name of the Experiment:
1. Simple Assembly language programming.
2. Complex Assembly language programming.

Performed on: 22/08/2023

Submitted on: 03/10/2023

Marks	Teacher's Signature with date

Aim:

1. Simple Assembly language programming.
   a. Write assembly language program for addition of two 8-bit numbers.
   b. Write assembly language program for addition of N 8-bit numbers. Take the input numbers from memory and store result in memory.

2. Complex Assembly language programming.
   a. Find square of a number using DPTR.

Theory:

Programming in the sense of Microcontrollers (or any computer) means writing a sequence of instructions that are executed by the processor in a particular order to perform a predefined task. The three levels of Programming Languages are as shown in figure 1.1.

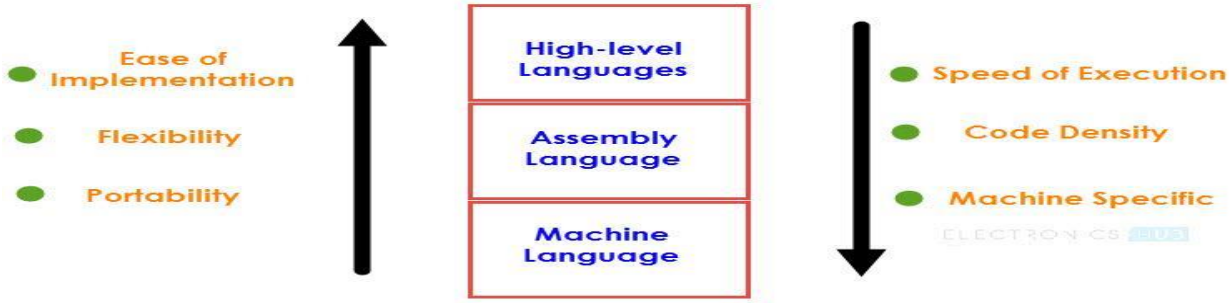


Figure1.1: Programming Languages

Assembly Language is a pseudo-English representation of the Machine Language. The 8051 Microcontroller Assembly Language is a combination of English like words called Mnemonics and Hexadecimal codes with the following advantages:

- The Programs written in Assembly gets executed faster and they occupy less memory.
- With the help of Assembly Language, you can directly exploit all the features of a Microcontroller.
- Using Assembly Language, you can have direct and accurate control of all the Microcontroller's resources like I/O Ports, RAM, SFRs, etc.
- Compared to High-level Languages, Assembly Language has less rules and restrictions.

Syntax of CIP51 Instruction:

[Label:] Instructions [;Comments]

CIP51 has about 109 instructions. These can be grouped into the following categories

1. Arithmetic Instructions
2. Logical Instructions
3. Data Transfer instructions
4. Boolean Variable Instructions
5. Program Branching Instructions

The ADD instruction is used for the addition of two operands. The destination operand is always in register A, while the source operand can be register, immediate data or memory. The AF, CY and P bits of the flag register are affected by the ADD instruction depending on the operands. Program Status Word (PSW) is the flag register as shown in figure 1.2.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CY	AC	FO	RS1	RS0	OV	UD	P

Figure1.2: Bit fields in Program Status Word (PSW) Register

Table 1.1: Functions of flags

Symbol	**Function**
CY	Carry flag
AC	Auxiliary Carry flag (For BCD Operations)
F0	Flag 0 (Available to the user for General Purpose)

RS1, **RS0**	Register bank select: RS1 RS0 Working Register Bank and Address 0 0 Bank0 0 1 Bank1 1 0 Bank2 1 1 Bank3
OV	Overflow flag
UD	User definable flag
P	Parity flag; P=1 for odd no. of 1's; P=0 for Even no. of 1's

Algorithm:

Sample Example:

Program:

Note: *Attach the printout of the code.*

Conclusion:

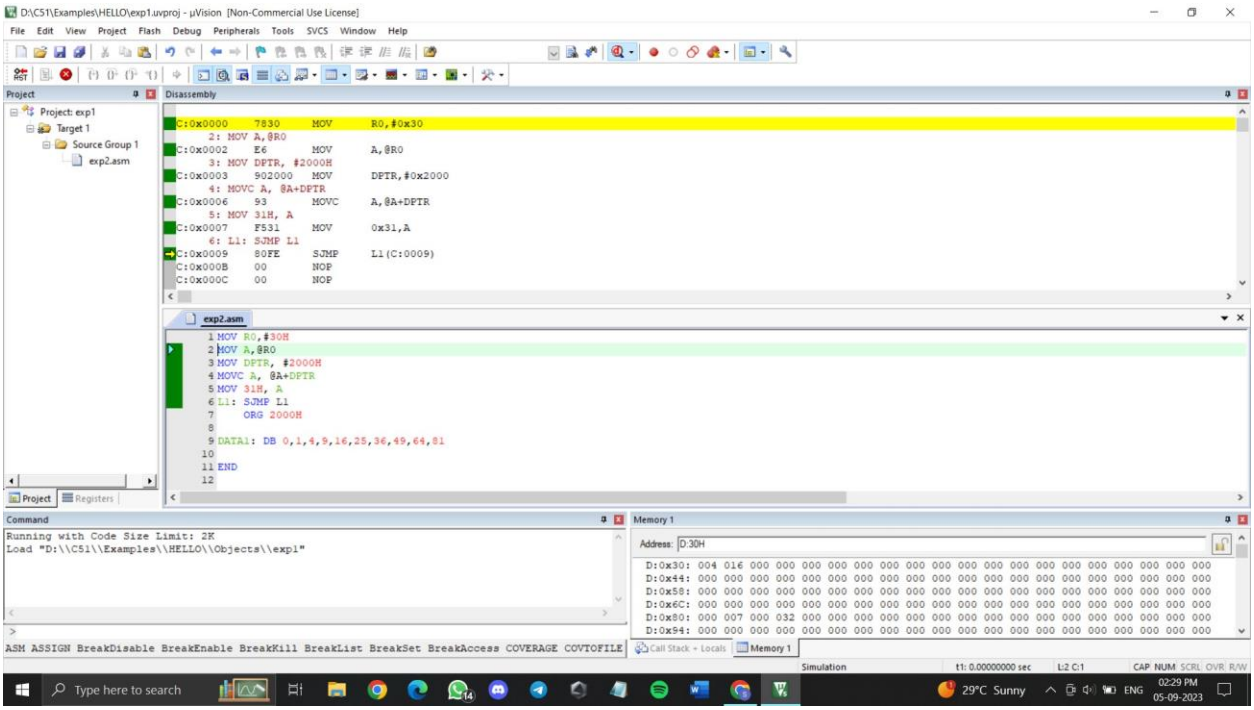
Study Question:

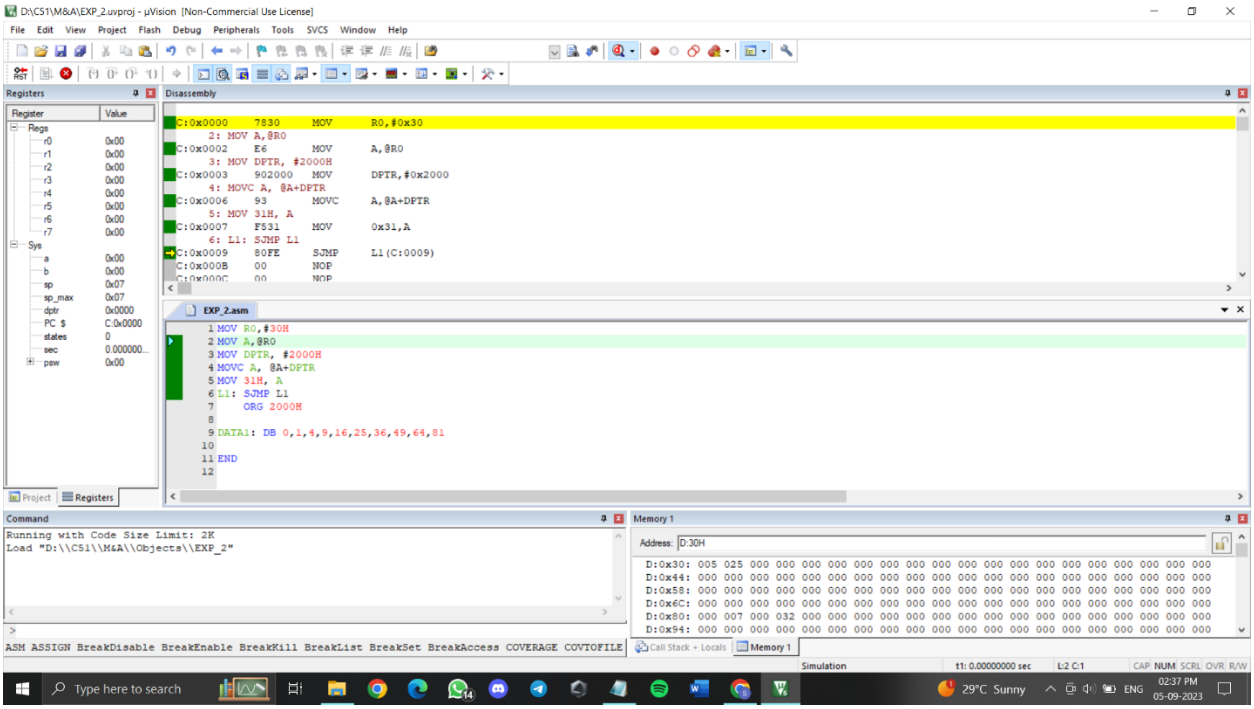
1. Explain the instructions: DJNZ, INC, SUBB.
2. What is an IDE?

Additional links:

1. *https://nptel.ac.in/courses/108105102*
2. *https://www.electronicshub.org/8051-microcontroller-assembly-language-programming/*

Practical-1


D:\C51\Examples\HELLO\exp1.uvproj - µVision [Non-Commercial Use License]
C:0x0000 7830 MOV R0,#0x30
2: MOV A,@R0
C:0x0002 E6 MOV A,@R0
3: MOV DPTR, #2000H
C:0x0003 902000 MOV DPTR,#0x2000
4: MOVC A, @A+DPTR
C:0x0006 93 MOVC A,@A+DPTR
5: MOV 31H, A
C:0x0007 F531 MOV 0x31,A
6: L1: SJMP L1
C:0x0009 80FE SJMP L1 (C:0009)
C:0x000B 00 NOP
C:0x000C 00 NOP
exp2.asm
1 MOV R0,#30H
2 MOV A,@R0
3 MOV DPTR, #2000H
4 MOVC A, @A+DPTR
5 MOV 31H, A
6 L1: SJMP L1
7 ORG 2000H
8
9 DATA1: DB 0,1,4,9,16,25,36,49,64,81
10
11 END
12
Command
Running with Code Size Limit: 2K
Load "D:\\C51\\Examples\\HELLO\\Objects\\exp1"
Memory 1
Address: D:30H
D:0x30: 004 016 000 000 000 000 000 000 000 000 000 000 000 000 000 000
D:0x44: 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
D:0x58: 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
D:0x6C: 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
D:0x80: 000 007 000 032 000 000 000 000 000 000 000 000 000 000 000 000
D:0x94: 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
Simulation
t1: 0.00000000 sec
L:2 C:1
29°C Sunny
ENG
02:29 PM
05-09-2023


D:\C51\M&A\EXP_2.uvproj - µVision [Non-Commercial Use License]
C:0x0000 7830 MOV R0,#0x30
2: MOV A,@R0
C:0x0002 E6 MOV A,@R0
3: MOV DPTR, #2000H
C:0x0003 902000 MOV DPTR,#0x2000
4: MOVC A, @A+DPTR
C:0x0006 93 MOVC A,@A+DPTR
5: MOV 31H, A
C:0x0007 F531 MOV 0x31,A
6: L1: SJMP L1
C:0x0009 80FE SJMP L1 (C:0009)
C:0x000B 00 NOP
EXP_2.asm
1 MOV R0,#30H
2 MOV A,@R0
3 MOV DPTR, #2000H
4 MOVC A, @A+DPTR
5 MOV 31H, A
6 L1: SJMP L1
7 ORG 2000H
8
9 DATA1: DB 0,1,4,9,16,25,36,49,64,81
10
11 END
12
Command
Running with Code Size Limit: 2K
Load "D:\\C51\\M&A\\Objects\\EXP_2"
Memory 1
Address: D:30H
D:0x30: 005 025 000 000 000 000 000 000 000 000 000 000 000 000 000 000
D:0x44: 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
D:0x58: 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
D:0x6C: 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
D:0x80: 000 007 000 032 000 000 000 000 000 000 000 000 000 000 000 000
D:0x94: 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
Simulation
t1: 0.00000000 sec
L:2 C:1
29°C Sunny
ENG
02:37 PM
05-09-2023

22/8/23

Expt

PAGE No.
DATE 22/8/23

(3) Write assembly language program for addition of n numbers of 8 bit

Note- MOV D, S JC/JNC → Jump if NO carry.
Destination, Source, Jump if carry

```
MOV R0, #50H ; initialize pointer to i/p  R0 = 50H
MOV R1, #05H ; Initialize Counter, R1 = 5H
MOV A, #00H
MOV 55H, #00H ; Initialize Carry=00
L2: ADD A, @R0 ; A = A + @R0 ⇔ Sum = Sum + a[i]
```

A = 00 + 29H = 29H
A = 29H + 35H = 64H
A = 64H + 64H = ...

(General Purpose Register R0 → R7)

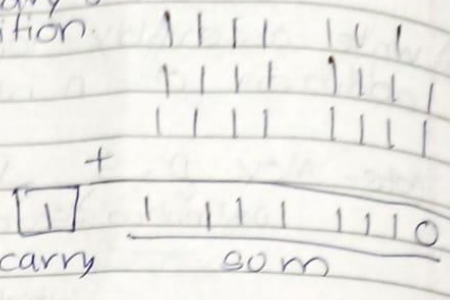
```
carry→1  JNC L1
         INC 55H
L1: INC R0
    DJNZ R1, L2
    MOV 56H, A
L3: SJMP L3
    END
```

8-bit Adder

Data Memory (8-bit)

	00	
50H	29H	+1
51H	35H	+1
52H	64H	+1
53H	56H	+1
54H	99H	+1
55H	Carry	+1
56H	Sum	

Binary 8-bit addition



Equivalent C code

```
#include <stdio.h>
int main(){
    int sum = 0;
    int arr[5] = {10, 20, 30, 40, 50};
    for(int i=0; i<5; i++)
    {
        sum = sum + arr[i];
    }
    printf(sum);
}
```

assembler

Embedded C → ALP → MLP

~~Assembler~~ ~~compiler~~ compiler assembler

- C8051F340

IDE
- Editor

assembly language → machine language

embedded C → assembly → machine

H

PAGE No.
DATE / /

Keil, microvision5

Project > new uvision project

Search your microcontroller > C8051f340

File > new > save as 'source group1'
Add exsting files to group i sourcegrp

Select 'add.asm' > open add.asm

write the program. a.sm
save
Build target.
Debug the program
↳ start
↳ click memory 1 (Bottom right)
to access memory → d:40H → enter.
right click on memory.
Modify memory

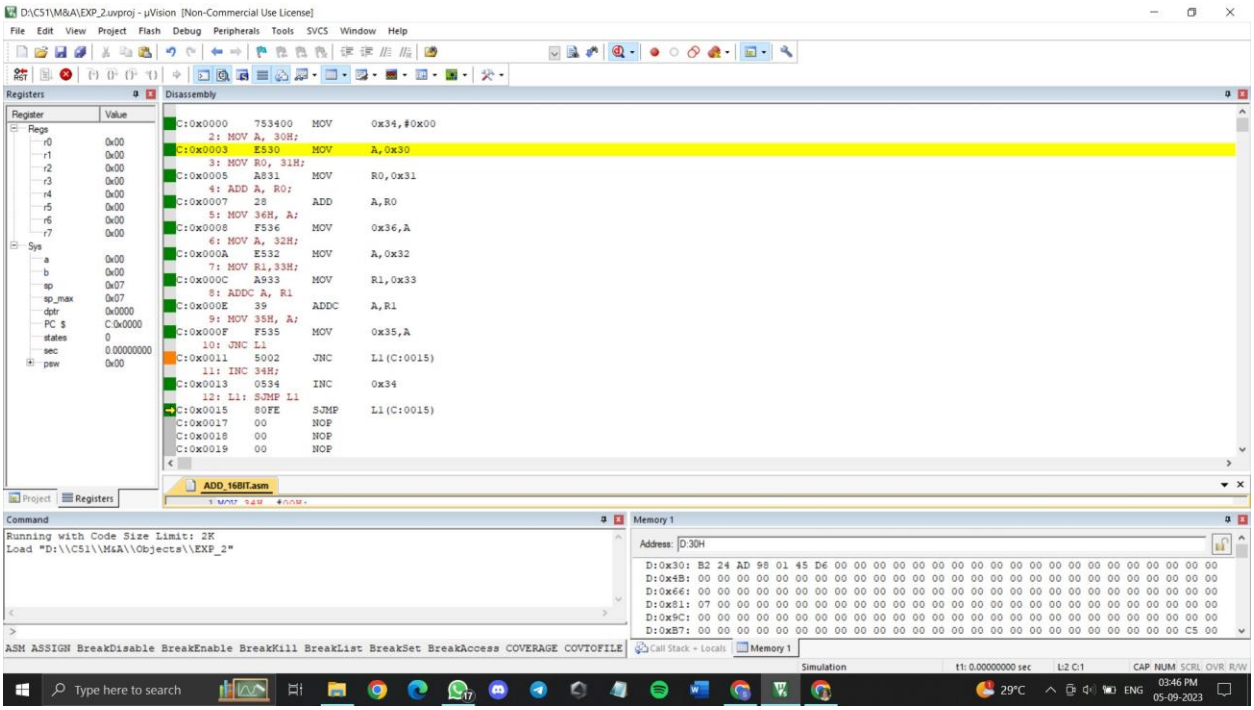
Eg –

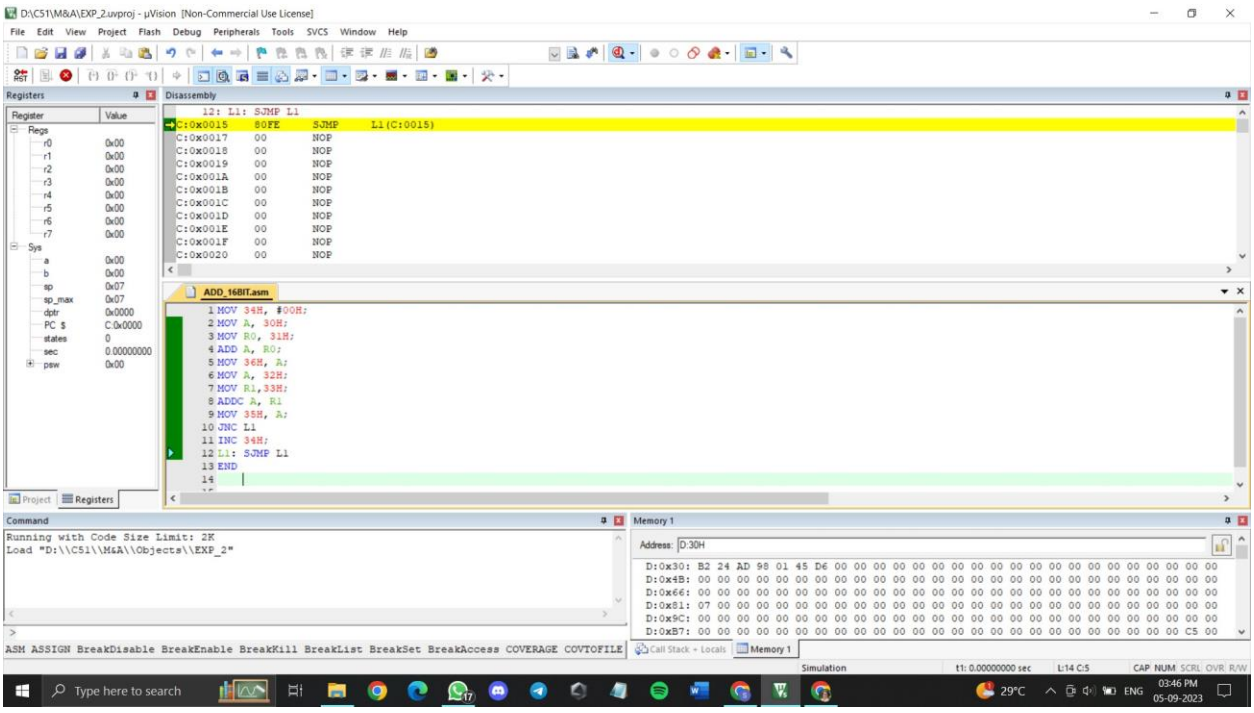
```
mov r0, #40h
mov a, #05h
Add a, @R0
MOV r1, A
mov 42h, a
end.
```

FFH, 18H, 56H, 7BH, DEH.

NDZ
22/8/23

Practical-2


D:\C51\M&A\EXP_2.uvproj - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
C:0x0000 753400 MOV 0x34,#0x00
2: MOV A, 30H;
C:0x0003 E530 MOV A,0x30
3: MOV R0, 31H;
C:0x0005 A831 MOV R0,0x31
4: ADD A, R0;
C:0x0007 28 ADD A,R0
5: MOV 36H, A;
C:0x0008 F536 MOV 0x36,A
6: MOV A, 32H;
C:0x000A E532 MOV A,0x32
7: MOV R1,33H;
C:0x000C A933 MOV R1,0x33
8: ADDC A, R1
C:0x000E 39 ADDC A,R1
9: MOV 35H, A;
C:0x000F F535 MOV 0x35,A
10: JNC L1
C:0x0011 5002 JNC L1(C:0015)
11: INC 34H;
C:0x0013 0534 INC 0x34
12: L1: SJMP L1
C:0x0015 80FE SJMP L1(C:0015)
C:0x0017 00 NOP
C:0x0018 00 NOP
C:0x0019 00 NOP
ADD_16BIT.asm
Command
Running with Code Size Limit: 2K
Load "D:\\C51\\M&A\\Objects\\EXP_2"
Memory 1
Address: D:30H
D:0x30: B2 24 AD 98 01 45 D6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x4B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x66: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x81: 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x9C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0xB7: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C5 00
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
Simulation
Type here to search
29°C
ENG
03:46 PM
05-09-2023

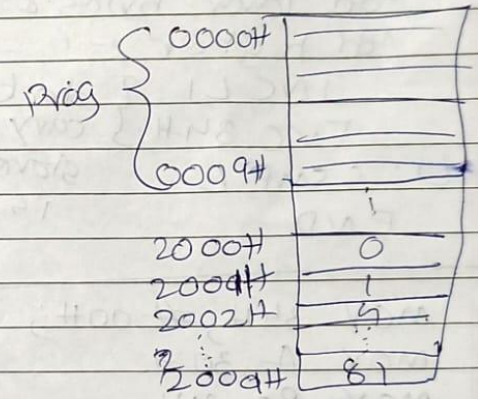

D:\C51\M&A\EXP_2.uvproj - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
12: L1: SJMP L1
C:0x0015 80FE SJMP L1(C:0015)
C:0x0017 00 NOP
C:0x0018 00 NOP
C:0x0019 00 NOP
C:0x001A 00 NOP
C:0x001B 00 NOP
C:0x001C 00 NOP
C:0x001D 00 NOP
C:0x001E 00 NOP
C:0x001F 00 NOP
C:0x0020 00 NOP
ADD_16BIT.asm
1 MOV 34H, #00H;
2 MOV A, 30H;
3 MOV R0, 31H;
4 ADD A, R0;
5 MOV 36H, A;
6 MOV A, 32H;
7 MOV R1,33H;
8 ADDC A, R1
9 MOV 35H, A;
10 JNC L1
11 INC 34H;
12 L1: SJMP L1
13 END
Command
Running with Code Size Limit: 2K
Load "D:\\C51\\M&A\\Objects\\EXP_2"
Memory 1
Address: D:30H
D:0x30: B2 24 AD 98 01 45 D6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x4B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x66: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x81: 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x9C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0xB7: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C5 00
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
Simulation
Type here to search
29°C
ENG
03:46 PM
05-09-2023

PAGE No.
DATE 5/9/23

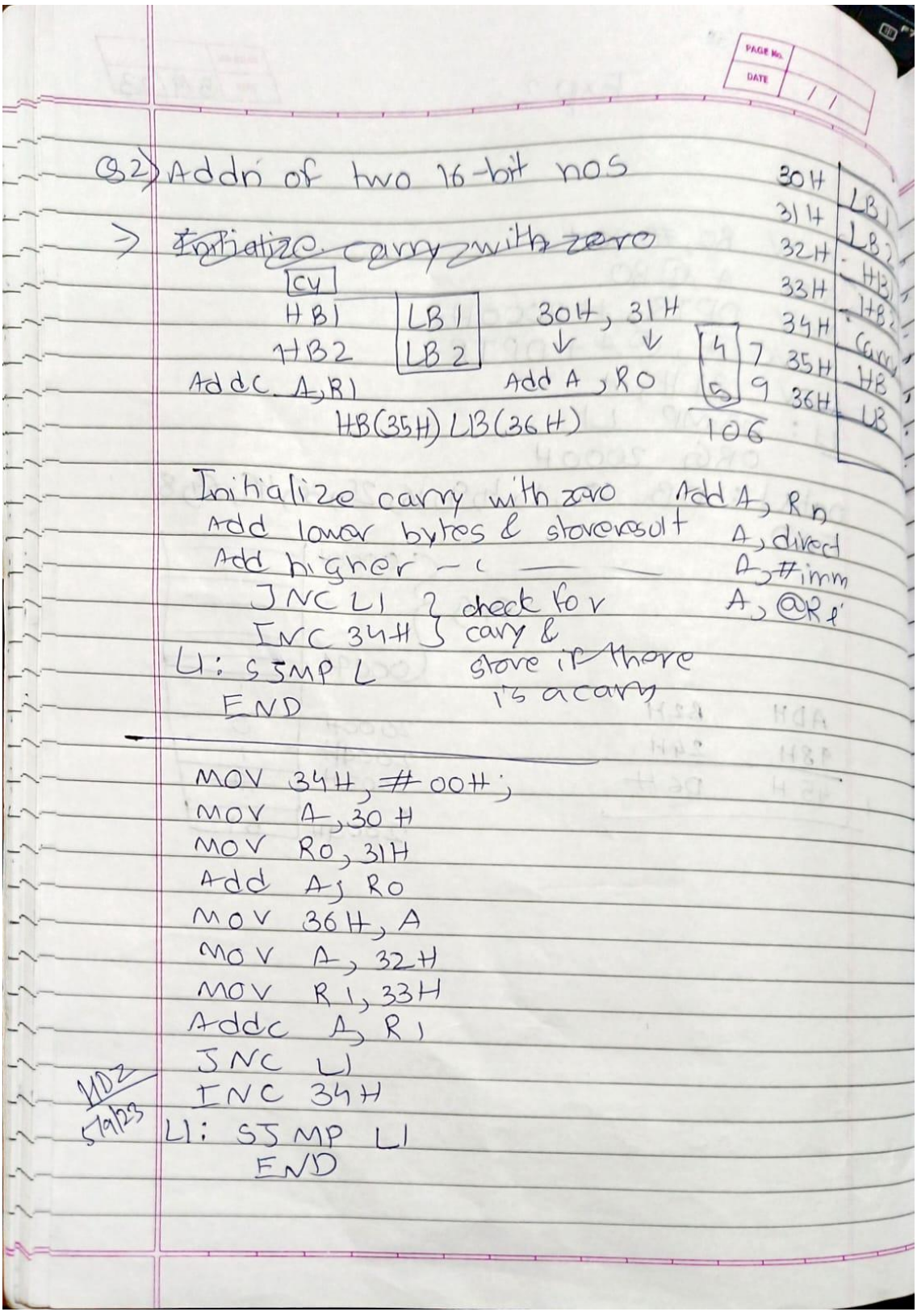
Exp 2

Q1)

```
MOV  R0, #30H
MOV  A, @R0
MOV  DPTR, #2000H
MOVC A, @A+DPTR
MOV  31H, A
L1: SJMP L1
    ORG 2000H
data1: DB 0, 1, 4, 9, 16, 25, 36, 49, 64, 81
END
```



ADH	B2H
98H	24H
45H	D6H



Q2) Addn of two 16-bit nos

> ~~Initialize carry with zero~~

CY		
HB1	LB1	
HB2	LB2	

30H, 31H

Add A, R0

Addc A, R1

HB(35H) LB(36H)

30H	LB1
31H	LB2
32H	HB1
33H	HB2
34H	Carry
35H	HB
36H	LB

4 7
5 9
106

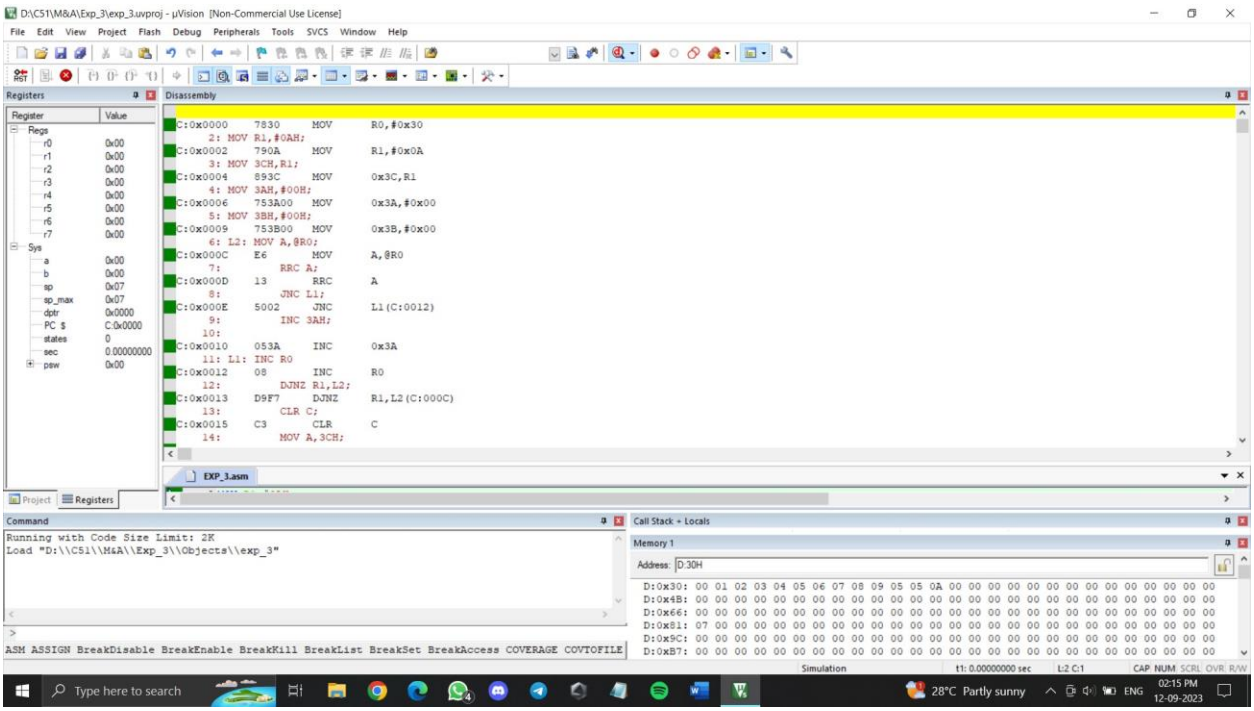
Initialize carry with zero
Add lower bytes & store result
Add higher - (
JNC L1 } check for
INC 34H } carry &
L1: SJMP L store if there
END is a carry

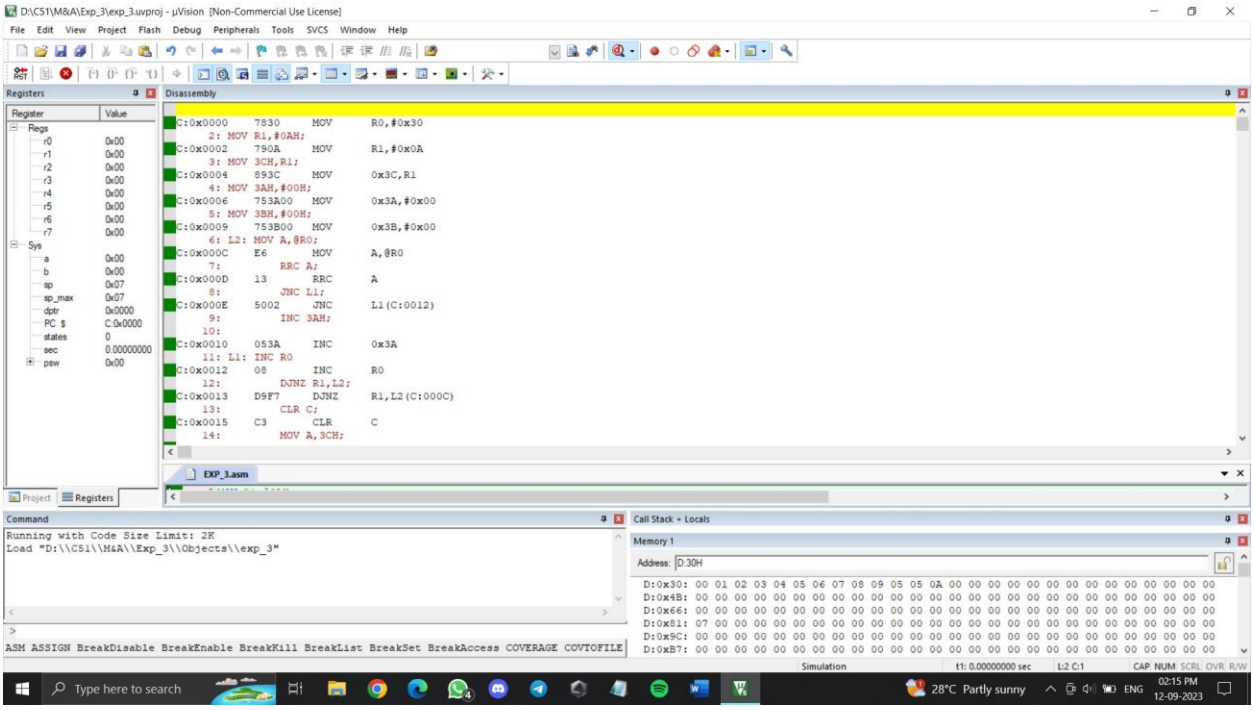
Add A, Rn
A, direct
A, #imm
A, @Ri

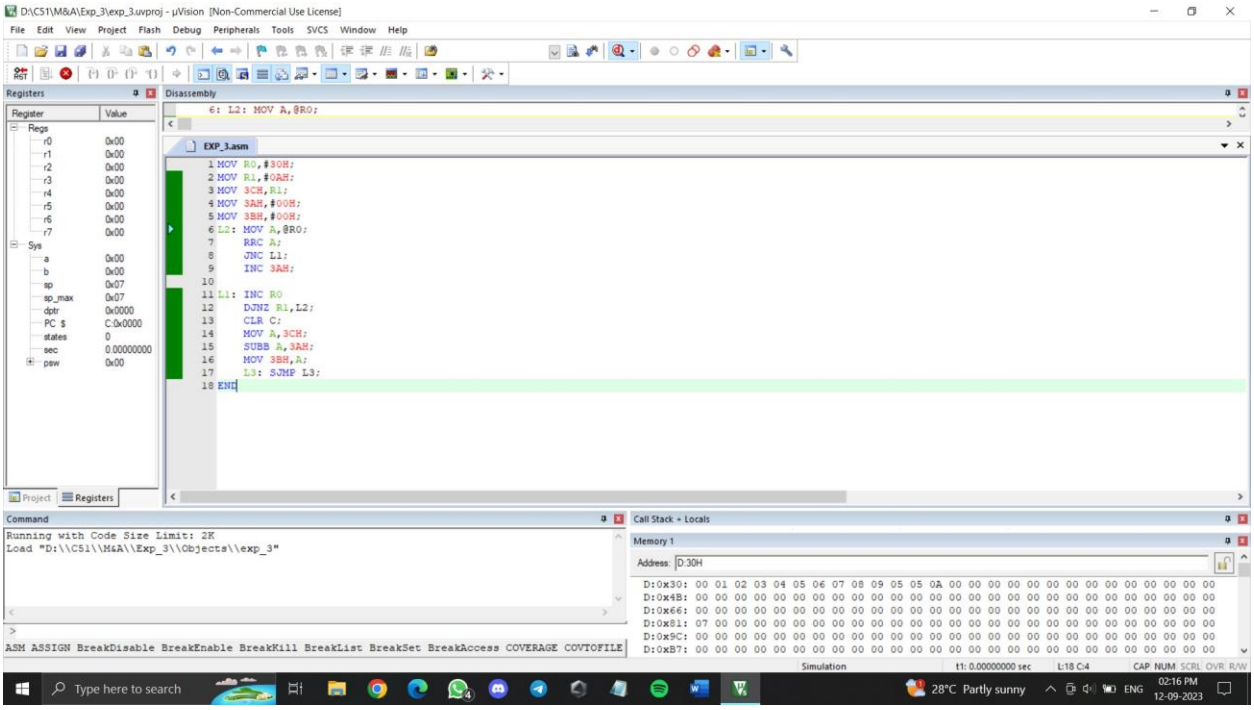
```
MOV 34H, #00H;
MOV A, 30H
MOV R0, 31H
Add A, R0
MOV 36H, A
MOV A, 32H
MOV R1, 33H
Addc A, R1
JNC L1
INC 34H
L1: SJMP L1
    END
```

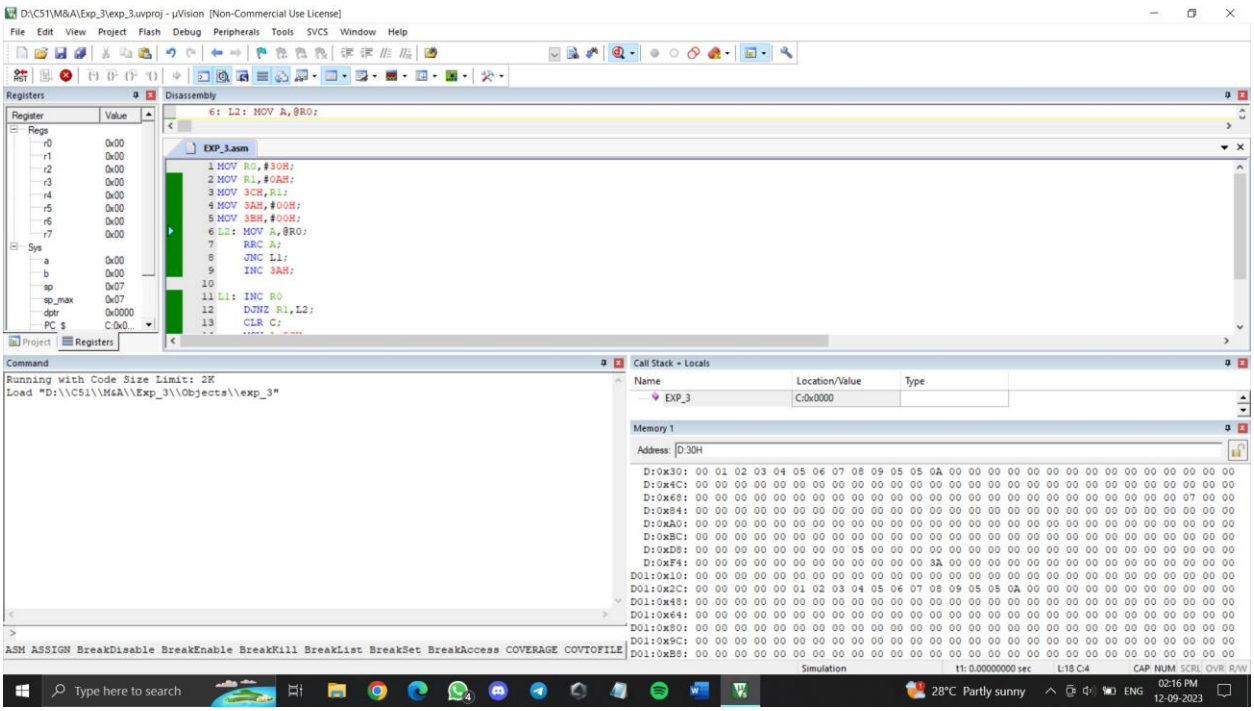
HD2
5/9/23

Practical-3


D:\C51\M&A\Exp_3\exp_3.uvproj - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Disassembly
C:0x0000 7830 MOV R0,#0x30
2: MOV R1,#0AH;
C:0x0002 790A MOV R1,#0x0A
3: MOV 3CH,R1;
C:0x0004 893C MOV 0x3C,R1
4: MOV 3AH,#00H;
C:0x0006 753A00 MOV 0x3A,#0x00
5: MOV 3BH,#00H;
C:0x0009 753B00 MOV 0x3B,#0x00
6: L2: MOV A,@R0;
C:0x000C E6 MOV A,@R0
7: RRC A;
C:0x000D 13 RRC A
8: JNC L1;
C:0x000E 5002 JNC L1(C:0012)
9: INC 3AH;
10:
C:0x0010 053A INC 0x3A
11: L1: INC R0
C:0x0012 08 INC R0
12: DJNZ R1,L2;
C:0x0013 D9F7 DJNZ R1,L2(C:000C)
13: CLR C;
C:0x0015 C3 CLR C
14: MOV A,3CH;
EXP_3.asm
Command
Running with Code Size Limit: 2K
Load "D:\\C51\\M&A\\Exp_3\\Objects\\exp_3"
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
Call Stack + Locals
Memory 1
Address: D:30H
Type here to search
28°C Partly sunny
ENG
02:15 PM
12-09-2023


D:\C51\M&A\Exp_3\exp_3.uvproj - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Disassembly
C:0x0000 7830 MOV R0,#0x30
2: MOV R1,#0AH;
C:0x0002 790A MOV R1,#0x0A
3: MOV 3CH,R1;
C:0x0004 893C MOV 0x3C,R1
4: MOV 3AH,#00H;
C:0x0006 753A00 MOV 0x3A,#0x00
5: MOV 3BH,#00H;
C:0x0009 753B00 MOV 0x3B,#0x00
6: L2: MOV A,@R0;
C:0x000C E6 MOV A,@R0
7: RRC A;
C:0x000D 13 RRC A
8: JNC L1;
C:0x000E 5002 JNC L1(C:0012)
9: INC 3AH;
10:
C:0x0010 053A INC 0x3A
11: L1: INC R0
C:0x0012 08 INC R0
12: DJNZ R1,L2;
C:0x0013 D9F7 DJNZ R1,L2(C:000C)
13: CLR C;
C:0x0015 C3 CLR C
14: MOV A,3CH;
EXP_3.asm
Command
Running with Code Size Limit: 2K
Load "D:\\C51\\M&A\\Exp_3\\Objects\\exp_3"
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
Call Stack + Locals
Memory 1
Address: D:30H
Type here to search
28°C Partly sunny
ENG
02:15 PM
12-09-2023


D:\C51\M&A\Exp_3\exp_3.uvproj - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
6: L2: MOV A,@R0;
EXP_3.asm
1 MOV R0,#30H;
2 MOV R1,#0AH;
3 MOV 3CH,R1;
4 MOV 3AH,#00H;
5 MOV 3BH,#00H;
6 L2: MOV A,@R0;
7 RRC A;
8 JNC L1;
9 INC 3AH;
11 L1: INC R0
12 DJNZ R1,L2;
13 CLR C;
14 MOV A,3CH;
15 SUBB A,3AH;
16 MOV 3BH,A;
17 L3: SJMP L3;
18 END
Command
Running with Code Size Limit: 2K
Load "D:\\C51\\M&A\\Exp_3\\Objects\\exp_3"
Call Stack + Locals
Memory 1
Address: D:30H
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
Type here to search
28°C Partly sunny
02:16 PM
12-09-2023


D:\C51\M&A\Exp_3\exp_3.uvproj - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
6: L2: MOV A,@R0;
EXP_3.asm
1 MOV R0,#30H;
2 MOV R1,#0AH;
3 MOV 3CH,R1;
4 MOV 3AH,#00H;
5 MOV 3BH,#00H;
6 L2: MOV A,@R0;
7 RRC A;
8 JNC L1;
9 INC 3AH;
11 L1: INC R0
12 DJNZ R1,L2;
13 CLR C;
Command
Running with Code Size Limit: 2K
Load "D:\\C51\\M&A\\Exp_3\\Objects\\exp_3"
Call Stack + Locals
Name
Location/Value
Type
EXP_3
C:0x0000
Memory 1
Address: D:30H
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
Type here to search
28°C Partly sunny
02:16 PM
12-09-2023

12/9/23

Exp 3

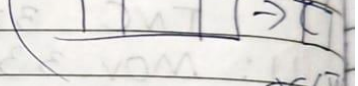
PAGE No.
DATE / /

Q) Program to count the number of odd and even numbers from set of given 10 numbers.

```
MOV R0, #30H
MOV R1, #0AH
MOV R2, R1
MOV 3AH, #00H
MOV 3BH, #00H
L2: MOV A, @R0
RRC A
JNC L1
INC 3AH
L1: INC R0
DJNZ R1, L2
CLR C
MOV A, R2
SUBB A, 3AH
MOV 3BH, A
L3: SJMP L3
```

30H
31H
39H
3AH odd
3BH even

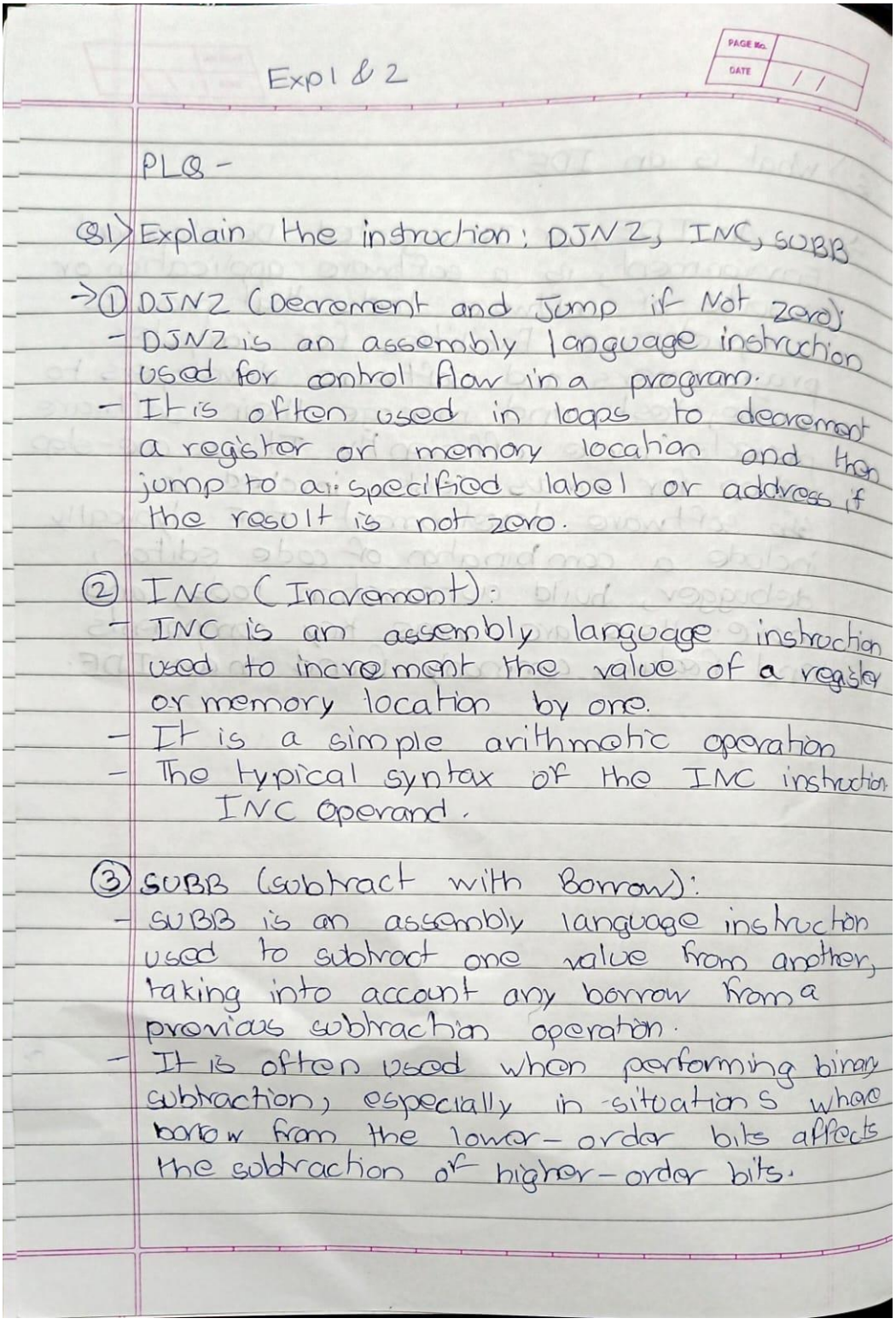
ACCC RRC



HOD 12/9/23

Q) Sort nos in ascending (1) & descending (2) order

Post Lab Questions:



Exp 1 & 2

PAGE No.
DATE / /

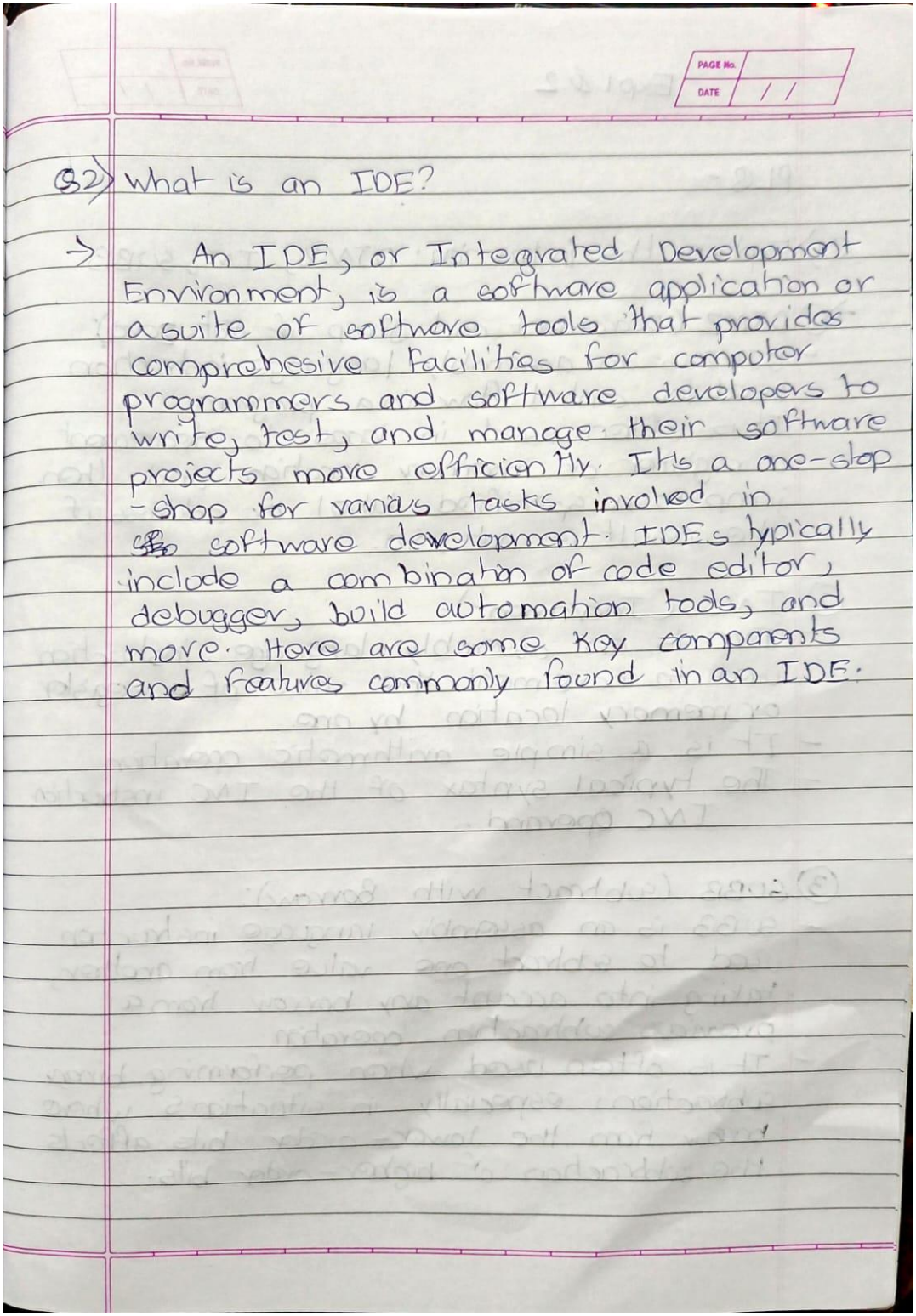
PLQ -

Q1) Explain the instruction: DJNZ, INC, SUBB

→ (1) DJNZ (Decrement and Jump if Not zero):
- DJNZ is an assembly language instruction used for control flow in a program.
- It is often used in loops to decrement a register or memory location and then jump to a specified label or address if the result is not zero.

(2) INC (Increment):
- INC is an assembly language instruction used to increment the value of a register or memory location by one.
- It is a simple arithmetic operation
- The typical syntax of the INC instruction
  INC Operand.

(3) SUBB (subtract with Borrow):
- SUBB is an assembly language instruction used to subtract one value from another, taking into account any borrow from a previous subtraction operation.
- It is often used when performing binary subtraction, especially in situations where borrow from the lower-order bits affects the subtraction of higher-order bits.



PAGE No.
DATE / /

Q2) What is an IDE?

→ An IDE, or Integrated Development Environment, is a software application or a suite of software tools that provides comprehensive facilities for computer programmers and software developers to write, test, and manage their software projects more efficiently. It's a one-stop-shop for various tasks involved in software development. IDEs typically include a combination of code editor, debugger, build automation tools, and more. Here are some key components and features commonly found in an IDE.

Guidelines to use Silicon Lab's Simplicity Studio

Simplicity Studio simplifies the Embedded development process with one-click access to everything developers need to complete their projects using an integrated development environment (IDE) based on Eclipse 4.5.

Steps to create project and program compilation:

Steps:

- Open Simplicity Studio.
- Click File > **Import…**, to import the project.
- Click General > Existing Projects into Workspace and click Next.
- First Select Root Directory of your Eclipse Template for EPB_F340. Then select "Copy projects into workspace" check box and Click Finish.
- Every time you import a project make sure to rename it. So **Right Click Project > Rename** or press **F2** while selecting project to rename it.
- To write source files, right click on Sources, go to New > Source File and you will see New Source File Wizard. Enter Source File name (For example **main.c**) then Click Finish.
- To write header files, right click on Includes, go to New > Header File and you will see New Header File Wizard. Enter Header File name (For example **main.h**) then Click Finish.
- Write your code and then save your Files.

OR

- Copy necessary .c and .h files to your local Sources and Includes project folders respectively. They will be added to your project in Simplicity Studio IDE.
- Click on Build Project in **Project>Build Project** to compile and build the project.
- After successful building the project the .hex file will be generated in the project folder in the workspace.

Steps to use hardware:

- Connect 9V DC Power supply to the educational practice board for F340
- Connect USB cable between PL4 connector of EPBF340 board and PC.
- Using the RUN/PROGRAM mode selection switch set the board in the program mode by pressing switch SW2 and press reset.
- Using download tool (USB Bootloader) download the .HEX file to the target board.
- Wait for the "Successfully loaded image" message in display.
- Connect flat cable between ASK25 and EPBF340 boards according to each experiment's requirement.
- Using the RUN/PROGRAM mode selection switch, set the board in the run mode by releasing Switch SW2. Apply reset to execute the program.
- Observe the expected output of the experiment.

NOTE: *Use this tool for Experiment No. 3 to 10. Draw interfacing diagram on separate sheet and attach the printout of the tested code.*

F340 Hardware

The EPB-F340Mini is a stand-alone card--allowing developers to evaluate the C8051F340 USB Flash MCU Family to determine if it meets their application requirements. Furthermore, the module is an excellent platform to develop and run software for the 8051 processor. The EPB-F340Mini is shipped with a C8051F340. The EPB-F340Mini allows full speed verification of 8051 code. In addition, an onboard C2 connector provides interface to emulators, with assembly language and 'C' high level language debug.

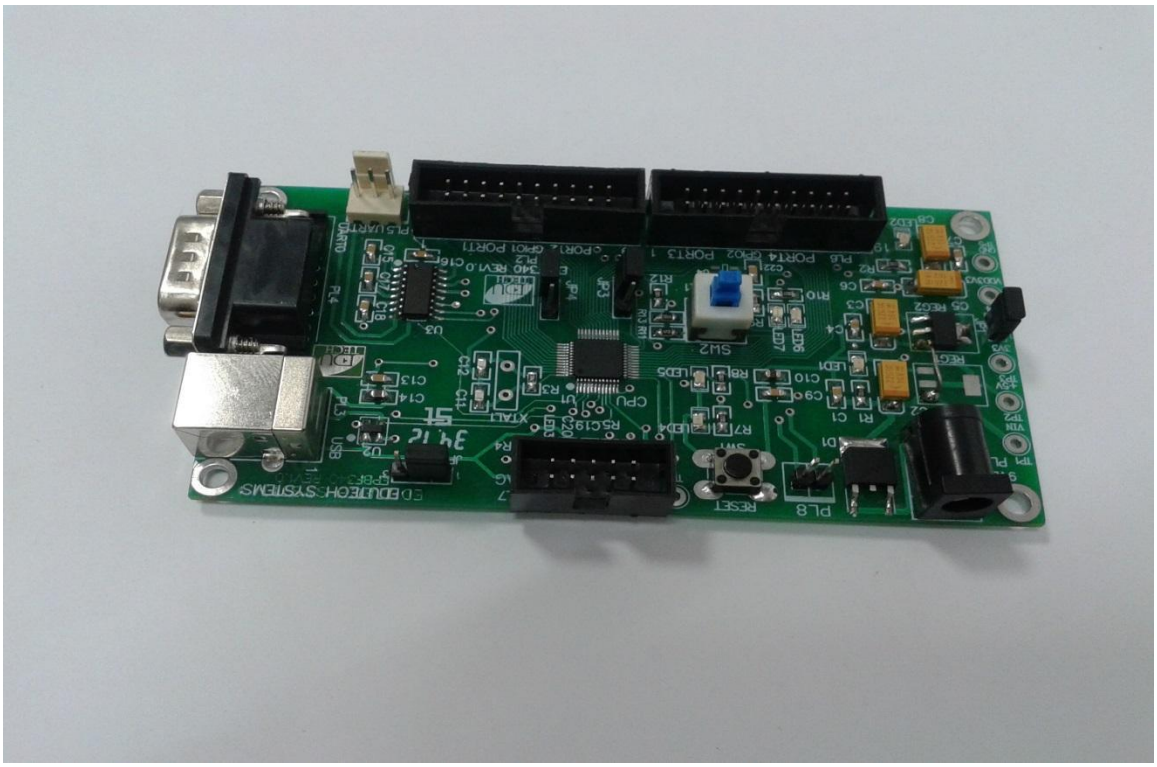


Figure 3.1: The EPB-F340Mini development board

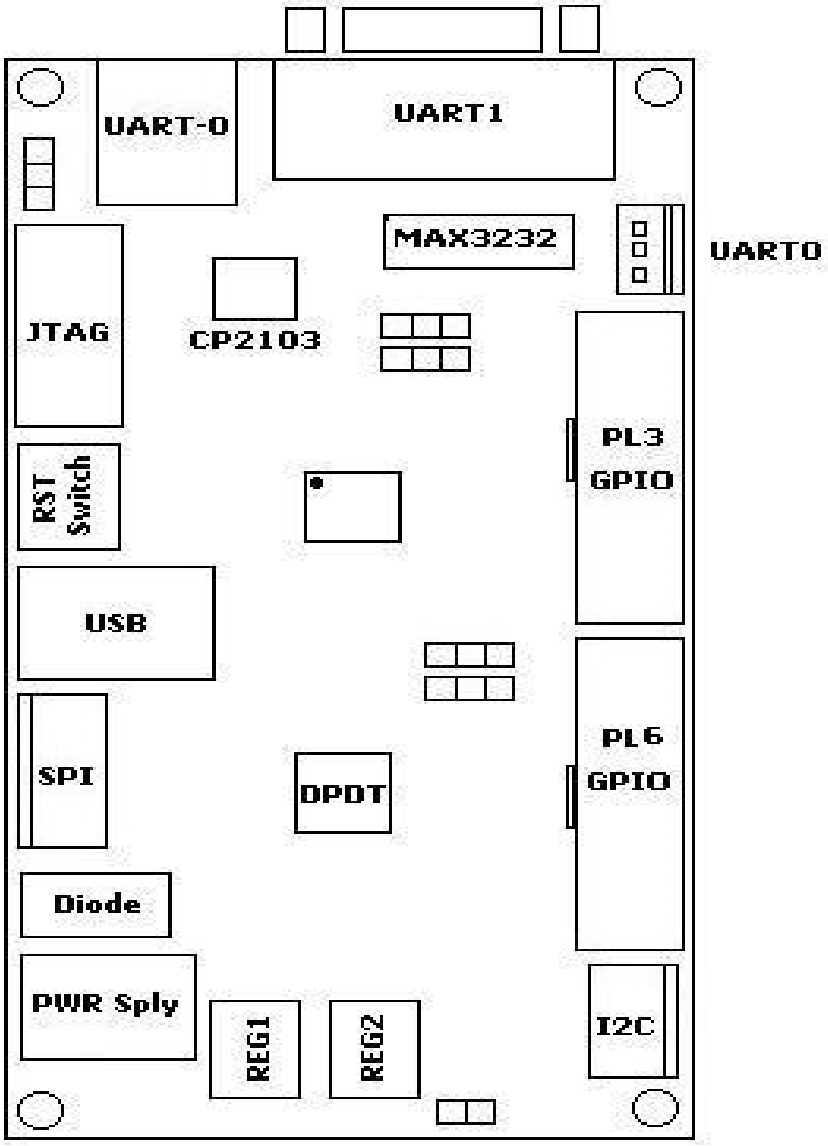


Figure 3.2 The EPB-F340Mini development board layout

The EPB-F340Mini has 8 connectors. The function of each connector is shown in the table below:

Table 3.1 EPB-F340Mini connector description

Unit	Reference	Description
Power jack connector	PL1	Power jack connector
I2C	PL2	4 Pins
I/O Port	PL3	PORT1 and 2 GPIO
USB	PL4	USB Connector
JTAG	PL5	JTAG Interface
I/O Port	PL6	PORT3 and 4 GPIO
SPI	PL7	6 Pins
UART0/Debug	PL8	USB Connector
UART0	PL9	3 Pins Connector
UART1	PL10	DB9 -M Connector

General Purpose I/O Interfacing Kit – ASK 25A

The general purpose I/O interfacing Board is a generic board which focuses on the interfacing of different input and out devices to microcontroller. The board is populated with variety of devices like LED, Key, Relay, LCD, Signal conditioning circuit for ADC, DC motor interface, Stepper Motor interface, I2C EEROM, SPI EEPROM, 2x2 Matrix Key board etc.

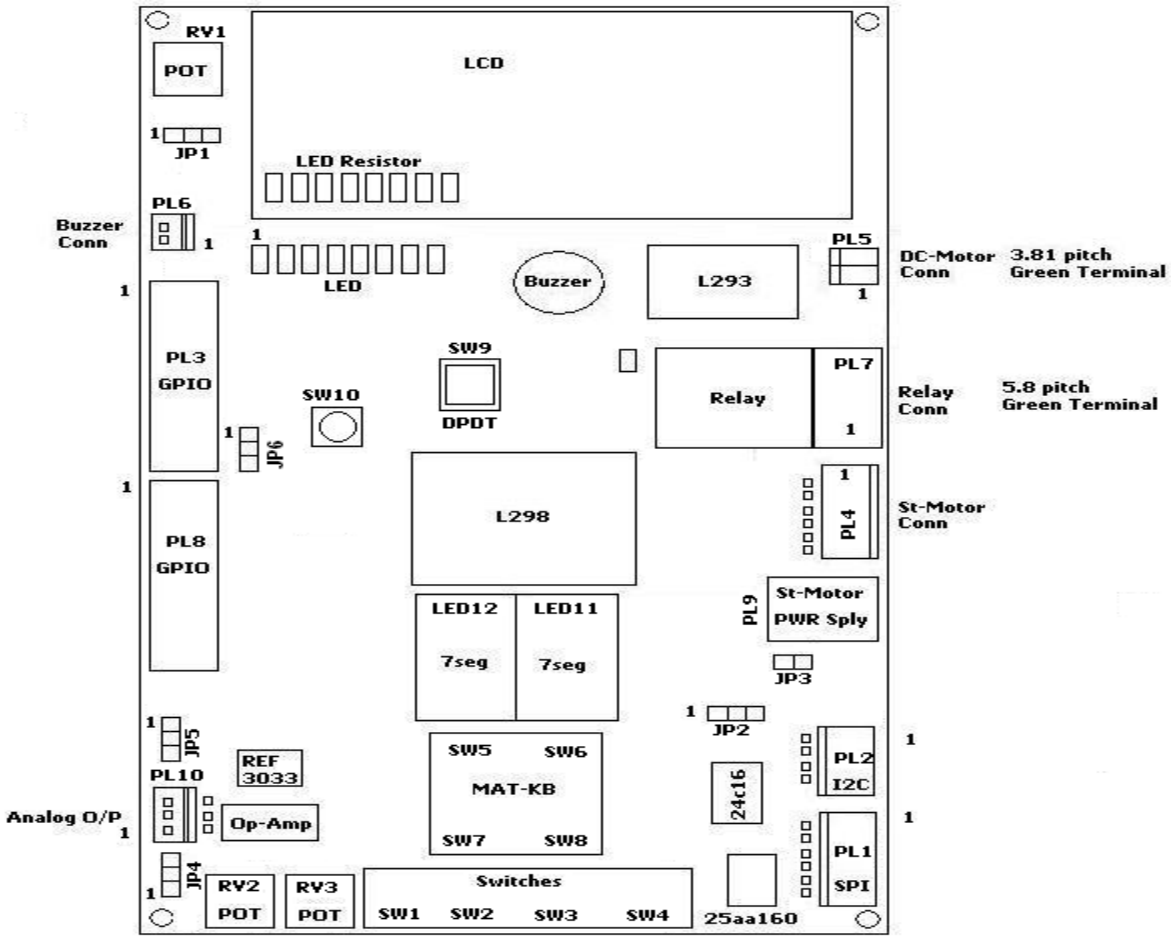


Figure 3.3 The ASK-25 board layout

The ASK-25A has 10 connectors. The function of each connector is shown in the table below:

Table 3.2 ASK-25A connector description

Unit	Reference	Description
SPI	PL1	6 Pins
I2C	PL2	4 Pins
I/O Port	PL3	GPIO
Stepper Motor Connector	PL4	6 Pins Connector
DC Motor Connector	PL5	2 Pin Green Terminal
Buzzer Connector	PL6	2 Pin Connector
Relay Connector	PL7	3 Pin Green Terminal
I/O Ports	PL8	GPIO

Stepper Motor Power Supply	PL9	Power Socket
Analog Output	PL10	3 Pin Connector

T. Y. B. Tech (Electrical and Computer Engineering)

Trimester: V **Subject: Microcontroller and Applications**
Name: Shreerang Mhatre **Class: TY**
Roll No: 52 **Batch: A3**

Experiment No: 03

Name of the Experiment: Interfacing of LED, Buzzer, Relay and Switch with C8051F340

Performed on: 03/10/2023

Submitted on: 07/10/2023

Marks	Teacher's Signature with date

Aim: Write C program for interfacing of LED, Buzzer, Relay and Switch with C8051F340 to turn it ON when key is pressed.

Apparatus: EPBF340 Board, ASK25 board, Connectors

Theory:

Ports in C8051F340:

Digital and analog resources are available through 40 I/O pins. These pins are available on ports. C8051F340 has 5 ports, each port has eight pins. Each of the Port pins can be defined as general-purpose I/O (GPIO) or analog input. This resource assignment flexibility is achieved through the use of a Priority Crossbar Decoder. Registers XBR0, XBR1, and XBR2 are used to assign the digital I/O resources to the physical I/O Port pins.

Port I/O Initialization

Port I/O initialization consists of the following steps:

Step 1. Select the input mode (analog or digital) for all Port pins, using the Port Input Mode register (PnMDIN).

Step 2. Select the output mode (open-drain or push-pull) for all Port pins, using the Port Output Mode register (PnMDOUT).

Step 3. Select any pins to be skipped by the I/O Crossbar using the Port Skip registers (PnSKIP).

Step 4. Assign Port pins to desired peripherals (XBR0, XBR1) as shown in figure 2.3.

Step 5. Enable the Crossbar (XBARE = '1').

SFR definitions:

1. **PnMDIN: Portn Input Mode**

0: Corresponding P0.n pin is configured as an analog input.

1: Corresponding P0.n pin is not configured as an analog input.

2. **PnMDOUT: Portn Output Mode**

0: Corresponding P0.n Output is open-drain.

1: Corresponding P0.n Output is push-pull.

3. **PnSKIP: Portn Skip**

0: Corresponding P0.n pin is not skipped by the Crossbar.

1: Corresponding P0.n pin is skipped by the Crossbar.

4. **XBR0, XBR1, and XBR2 are used to assign the digital I/O resources to the physical I/O Port pins**

SFR Definition 15.1. XBR0: Port I/O Crossbar Register 0

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
CP1AE	CP1E	CP0AE	CP0E	SYSCKE	SMB0E	SPI0E	URT0E	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xE1

SFR Definition 15.2. XBR1: Port I/O Crossbar Register 1

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
WEAKPUD	XBARE	T1E	T0E	ECIE	PCA0ME			00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xE2

SFR Definition 15.3. XBR2: Port I/O Crossbar Register 2

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
							URT1E	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xE3

LED: LED is an output device. 8 LED's are connected to 8 port pins. The LED's are connected in common anode configuration. Thus, to turn ON the LED logic '0' must be given and to turn OFF the led logic '1' must be given.

Buzzer: Buzzer is an output device. The buzzer is turned ON when logic '1' is applied to the port pin and turned OFF when logic '0' is applied to port pin.

Relay: Relay is an output device. The relay is turned ON when logic '0' is applied to port pin and turned OFF when logic '1' is applied to the port pin.

Switch/button: Switch is an input device. Push button switches are used. When the switch is released the port pin has logic '1' and when the switch is pushed the port pin has logic '0'.

Algorithm:

Interfacing Diagram:

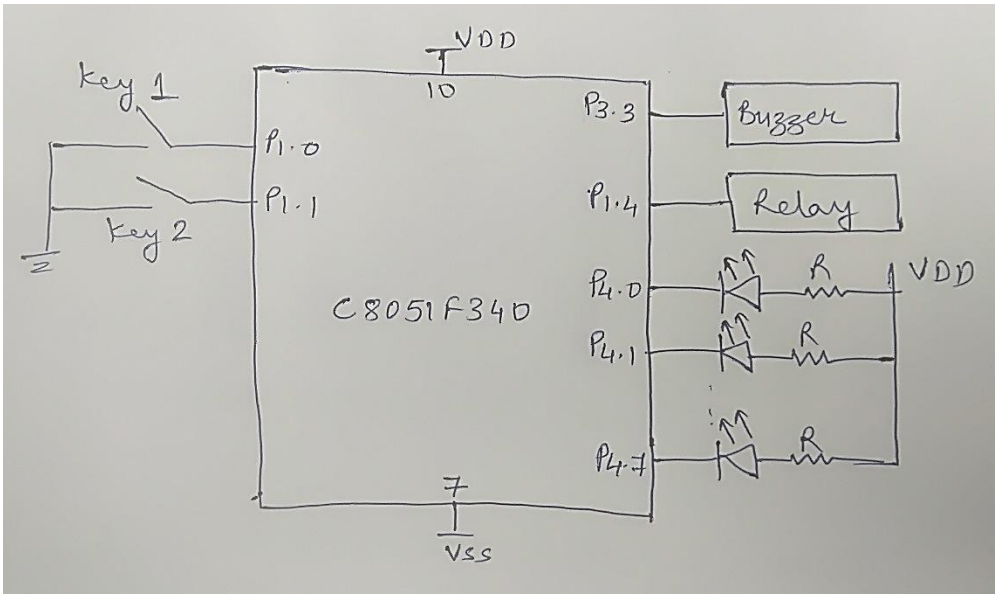


Figure 3.4 LED, Buzzer, Relay, and Switch Interfacing Diagram

Hardware Connections:

Connect flat cable between PL8 connector of ASK25 and PL6 connector of EPBF340 board. Connect other flat cable between PL3 connector of ASK25 and PL3 connector of EPBF340 board.

Table 3.3 Hardware connections between EPBF340 and ASK25 board

Pin Connection	PL8 Connector of ASK25	PL6 Connector of EPBF340	PL3 Connector of ASK25	PL3 Connector of EPBF340
1			SW1	P1.0
2			SW2	P1.1
3			SW3	P1.2
4	BUZZER	P3.3	SW4	P1.3
5			RELAY	P1.4
10	LED1	P4.0		
11	LED2	P4.1		
12	LED3	P4.2		
13	LED4	P4.3		
14	LED5	P4.4		
15	LED6	P4.5		
16	LED7	P4.6		
17	LED8	P4.7		
18		3.3 V		3.3 V
19	5V	5.0 V	5V	5.0 V
20	GROUND	GND	GROUND	GND

Program: Attach printout of the tested code.

Expected Result:

LEDs, Buzzer and Relay should turn on when respective key is pressed.

Conclusion:

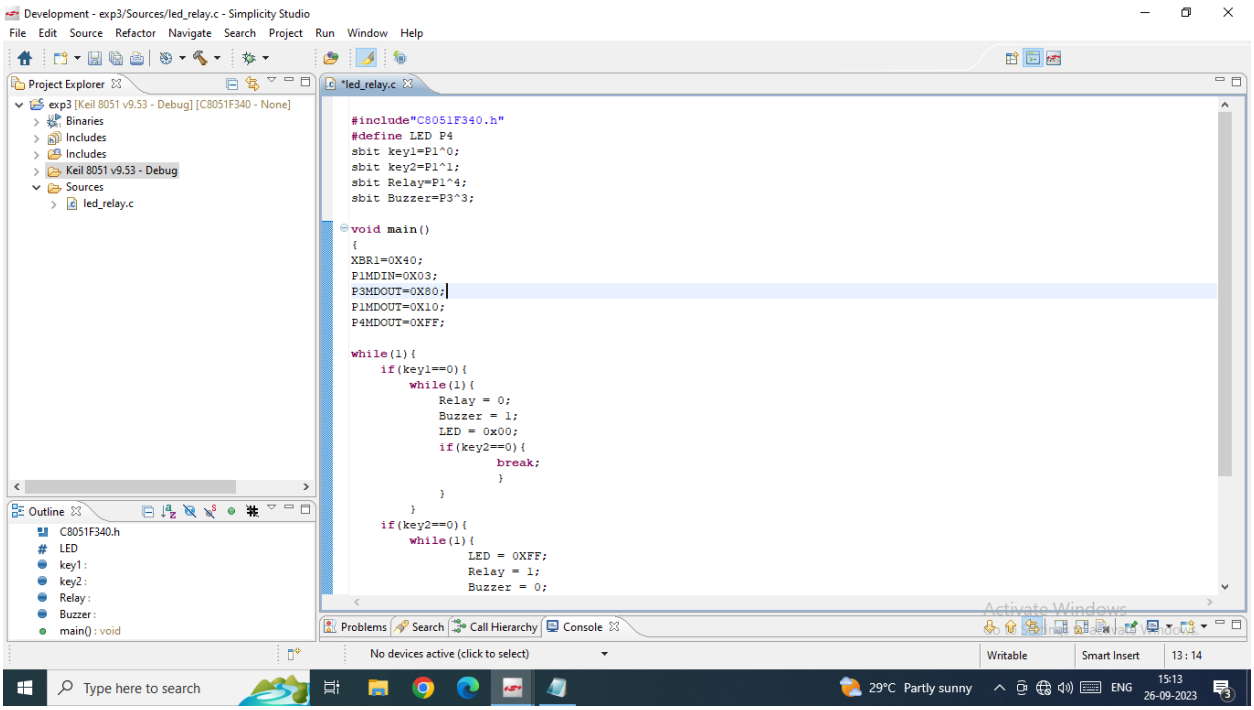
Study Question:

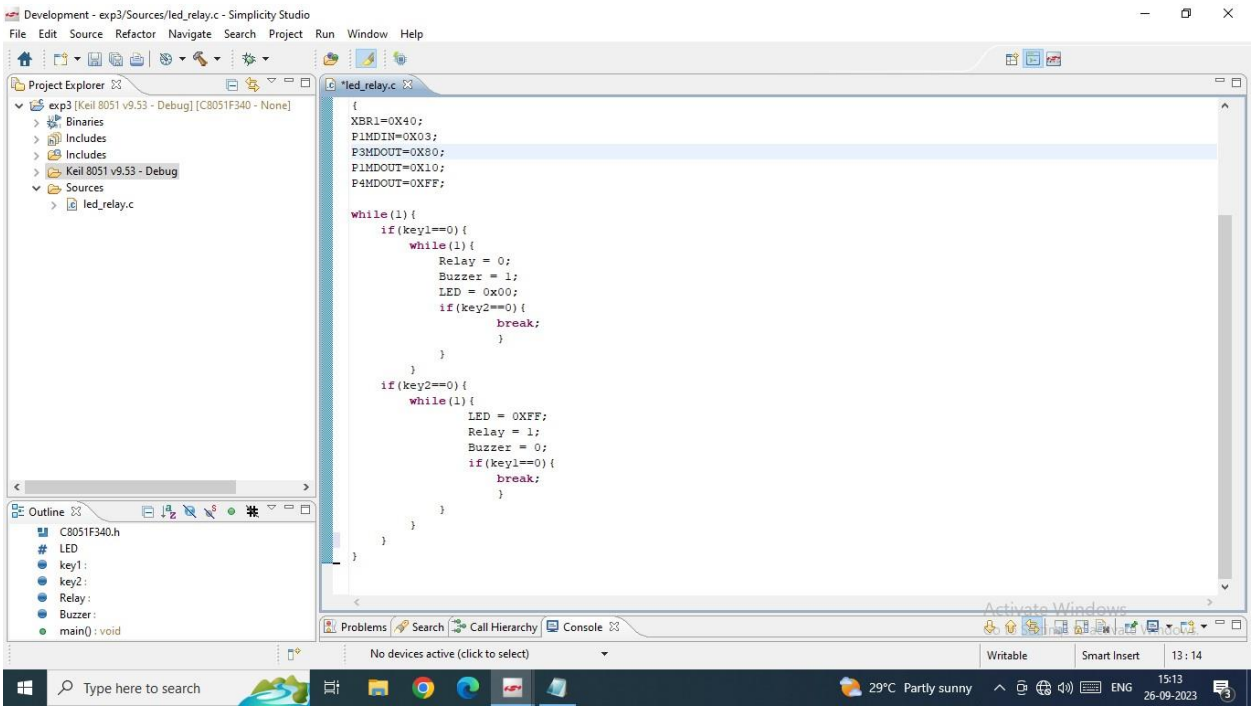
1. Explain the common anode and common cathode configuration of LED.
2. How many ports are available in C8051F340?
3. How to configure port as an input/output?
4. Explain Priority Crossbar Decoder.

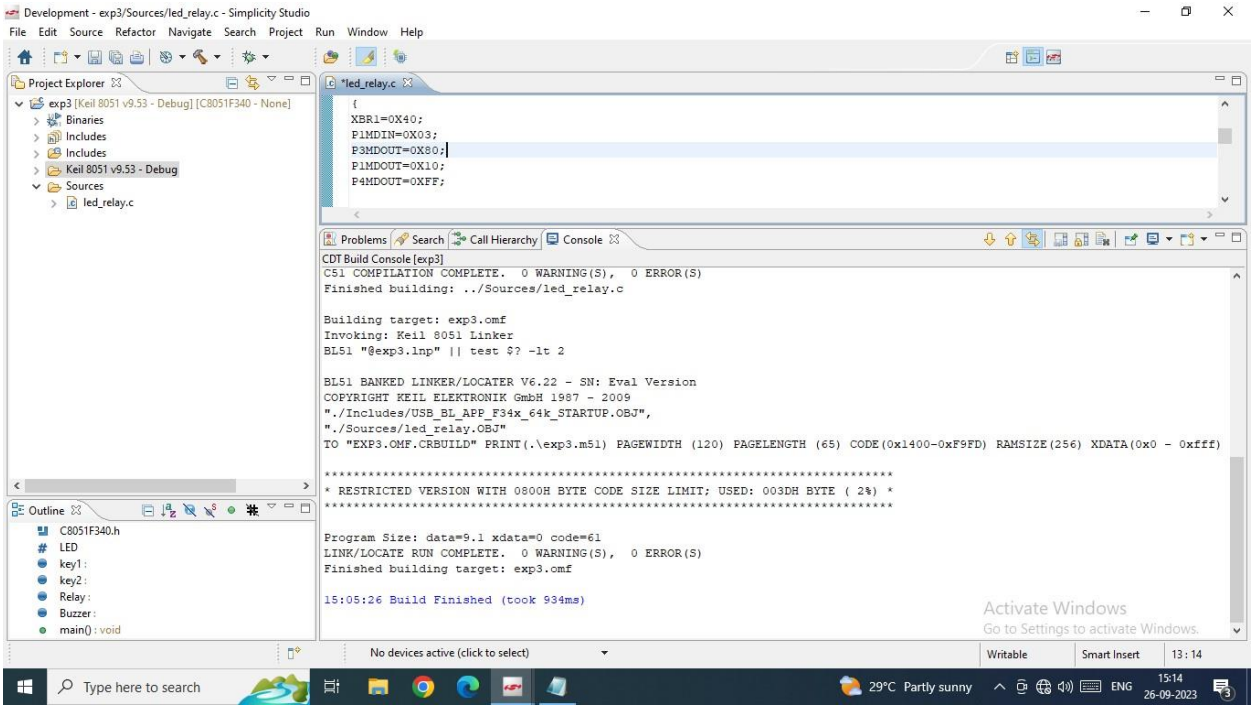
Additional links:

1. *https://www.silabs.com/documents/public/data-sheets/C8051F34x.pdf*
2. *https://aticleworld.com/interfacing-of-switch-and-led-using-the-8051/*

CODE:


Development - exp3/Sources/led_relay.c - Simplicity Studio
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
exp3 [Keil 8051 v9.53 - Debug] [C8051F340 - None]
Binaries
Includes
Includes
Keil 8051 v9.53 - Debug
Sources
led_relay.c
Outline
C8051F340.h
LED
key1 :
key2 :
Relay :
Buzzer :
main() : void
*led_relay.c
#include"C8051F340.h"
#define LED P4
sbit key1=P1^0;
sbit key2=P1^1;
sbit Relay=P1^4;
sbit Buzzer=P3^3;
void main()
{
XBR1=0X40;
P1MDIN=0X03;
P3MDOUT=0X80;
P1MDOUT=0X10;
P4MDOUT=0XFF;
while(1){
if(key1==0){
while(1){
Relay = 0;
Buzzer = 1;
LED = 0x00;
if(key2==0){
break;
}
}
}
if(key2==0){
while(1){
LED = 0XFF;
Relay = 1;
Buzzer = 0;
Problems Search Call Hierarchy Console
No devices active (click to select)
Writable Smart Insert 13 : 14
Type here to search
29°C Partly sunny
ENG 15:13 26-09-2023

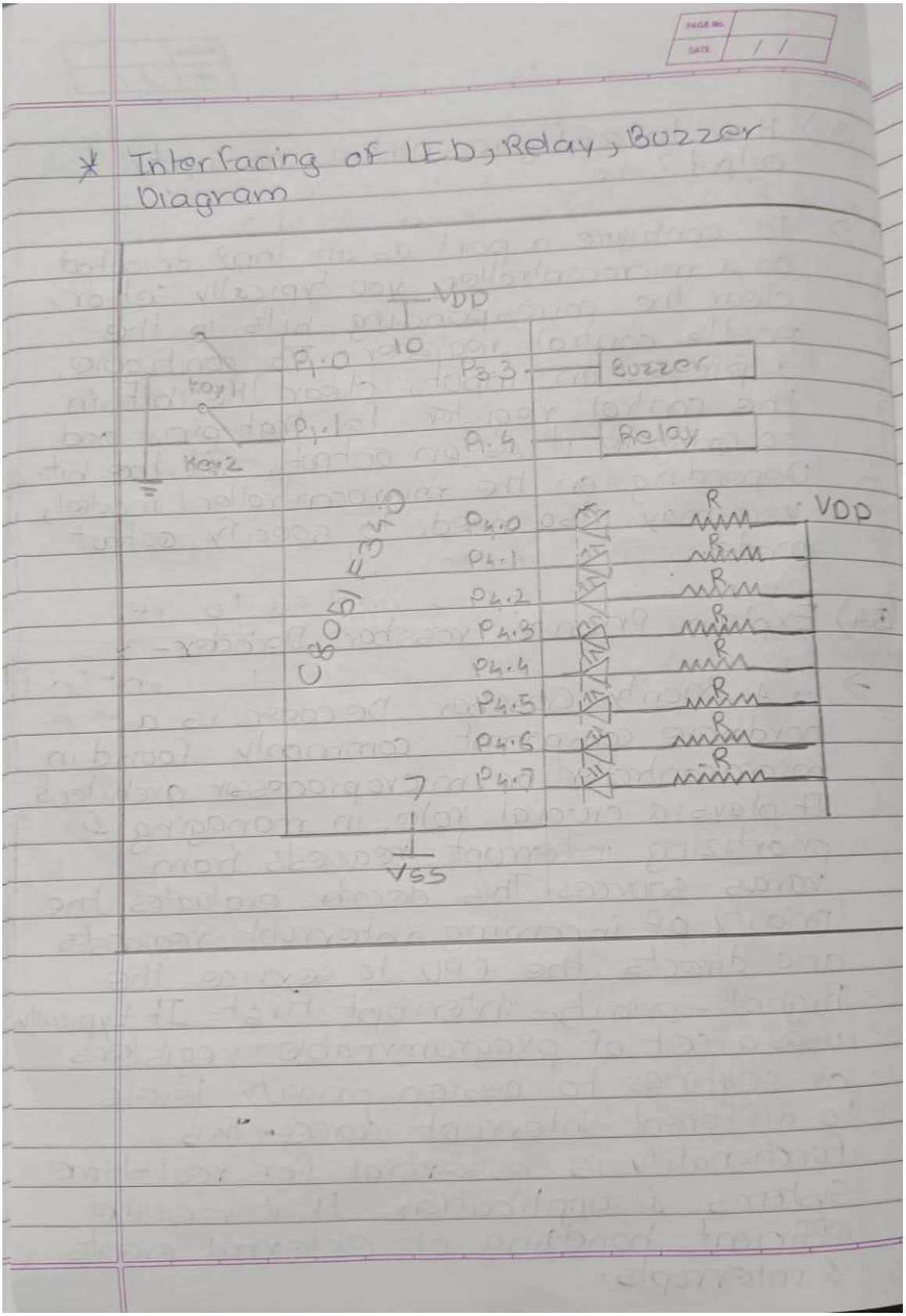

Development - exp3/Sources/led_relay.c - Simplicity Studio
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
exp3 [Keil 8051 v9.53 - Debug] [C8051F340 - None]
Binaries
Includes
Includes
Keil 8051 v9.53 - Debug
Sources
led_relay.c
Outline
C8051F340.h
LED
key1 :
key2 :
Relay :
Buzzer :
main() : void
*led_relay.c
{
XBR1=0X40;
P1MDIN=0X03;
P3MDOUT=0X80;
P1MDOUT=0X10;
P4MDOUT=0XFF;
while(1){
if(key1==0){
while(1){
Relay = 0;
Buzzer = 1;
LED = 0x00;
if(key2==0){
break;
}
}
}
if(key2==0){
while(1){
LED = 0XFF;
Relay = 1;
Buzzer = 0;
if(key1==0){
break;
}
}
}
}
}
Problems Search Call Hierarchy Console
No devices active (click to select)
Writable Smart Insert 13 : 14
Type here to search
29°C Partly sunny
ENG 15:13 26-09-2023


Development - exp3/Sources/led_relay.c - Simplicity Studio
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
exp3 [Keil 8051 v9.53 - Debug] [C8051F340 - None]
Binaries
Includes
Includes
Keil 8051 v9.53 - Debug
Sources
led_relay.c
*led_relay.c
{
XBR1=0X40;
P1MDIN=0X03;
P3MDOUT=0X80;
P1MDOUT=0X10;
P4MDOUT=0XFF;
Problems Search Call Hierarchy Console
CDT Build Console [exp3]
C51 COMPILATION COMPLETE. 0 WARNING(S), 0 ERROR(S)
Finished building: ../Sources/led_relay.c
Building target: exp3.omf
Invoking: Keil 8051 Linker
BL51 "@exp3.lnp" || test $? -lt 2
BL51 BANKED LINKER/LOCATER V6.22 - SN: Eval Version
COPYRIGHT KEIL ELEKTRONIK GmbH 1987 - 2009
"./Includes/USB_BL_APP_F34x_64k_STARTUP.OBJ",
"./Sources/led_relay.OBJ"
TO "EXP3.OMF.CRBUILD" PRINT(.\exp3.m51) PAGEWIDTH (120) PAGELENGTH (65) CODE(0x1400-0xF9FD) RAMSIZE(256) XDATA(0x0 - 0xfff)
* RESTRICTED VERSION WITH 0800H BYTE CODE SIZE LIMIT; USED: 003DH BYTE (2%) *
Program Size: data=9.1 xdata=0 code=61
LINK/LOCATE RUN COMPLETE. 0 WARNING(S), 0 ERROR(S)
Finished building target: exp3.omf
15:05:26 Build Finished (took 934ms)
Outline
C8051F340.h
LED
key1 :
key2 :
Relay :
Buzzer :
main() : void
Activate Windows
Go to Settings to activate Windows.
No devices active (click to select)
Writable
Smart Insert
13 : 14
Type here to search
29°C Partly sunny
ENG
15:14
26-09-2023


PL3 Connector
PL8 Connector
8.8.


PL3 Connector
Pin
LCD
MAT-KB
Switch
Stepper Motor
Relay
PL8 Connector
Pin
LED
7seg
DC Motor
Buzzer
No devices active (click to select)

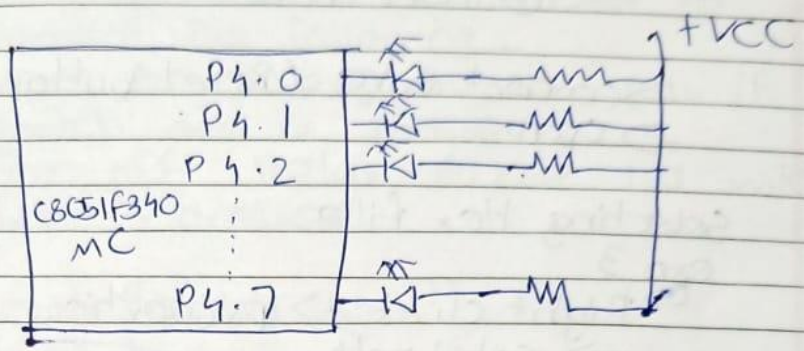
* Interfacing of LED, Relay, Buzzer Diagram



Exp 3

PAGE No.
DATE 12/9/23

Interfacing diagram

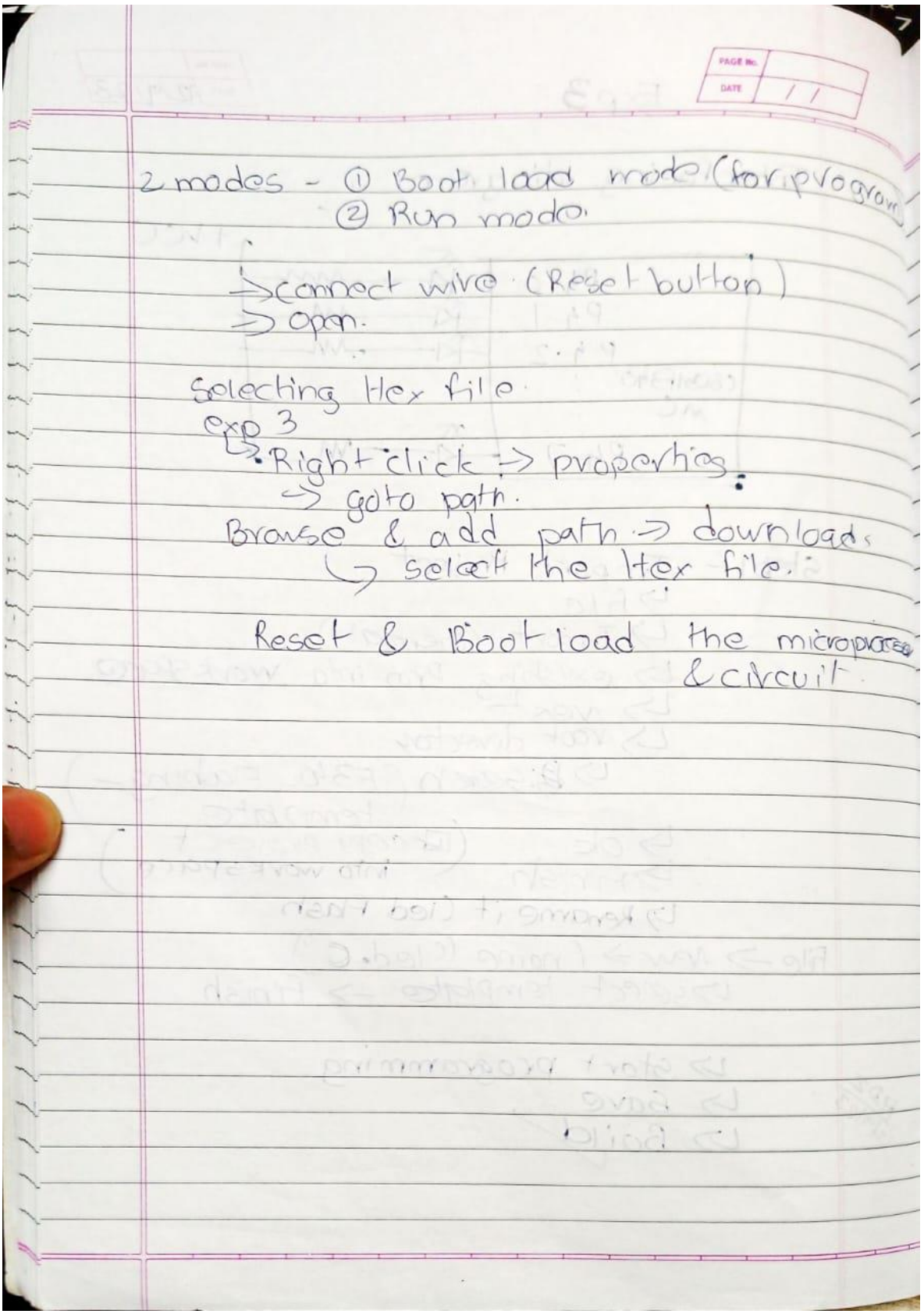


Step1 - Import Project
↳ file
↳ Import (General)
↳ existing Pros into workspace
↳ next
↳ root directory
↳ Browse (F340_Flashing_template)
↳ ok (copy project into workspace)
↳ Finish.
↳ Rename it (led Flash)

File → New → (name ("led.c"))
↳ select template → Finish

↳ Start programming
↳ Save
↳ Build

HDV 12/9/23



2 modes - ① Boot load mode (for programming)
② Run mode.

→ connect wire (Reset button)
→ open.

Selecting Hex file.
exp 3
→ Right click → properties.
→ goto path.
Browse & add path → downloads
↳ select the Hex file.

Reset & Boot load the microprocessor & circuit.

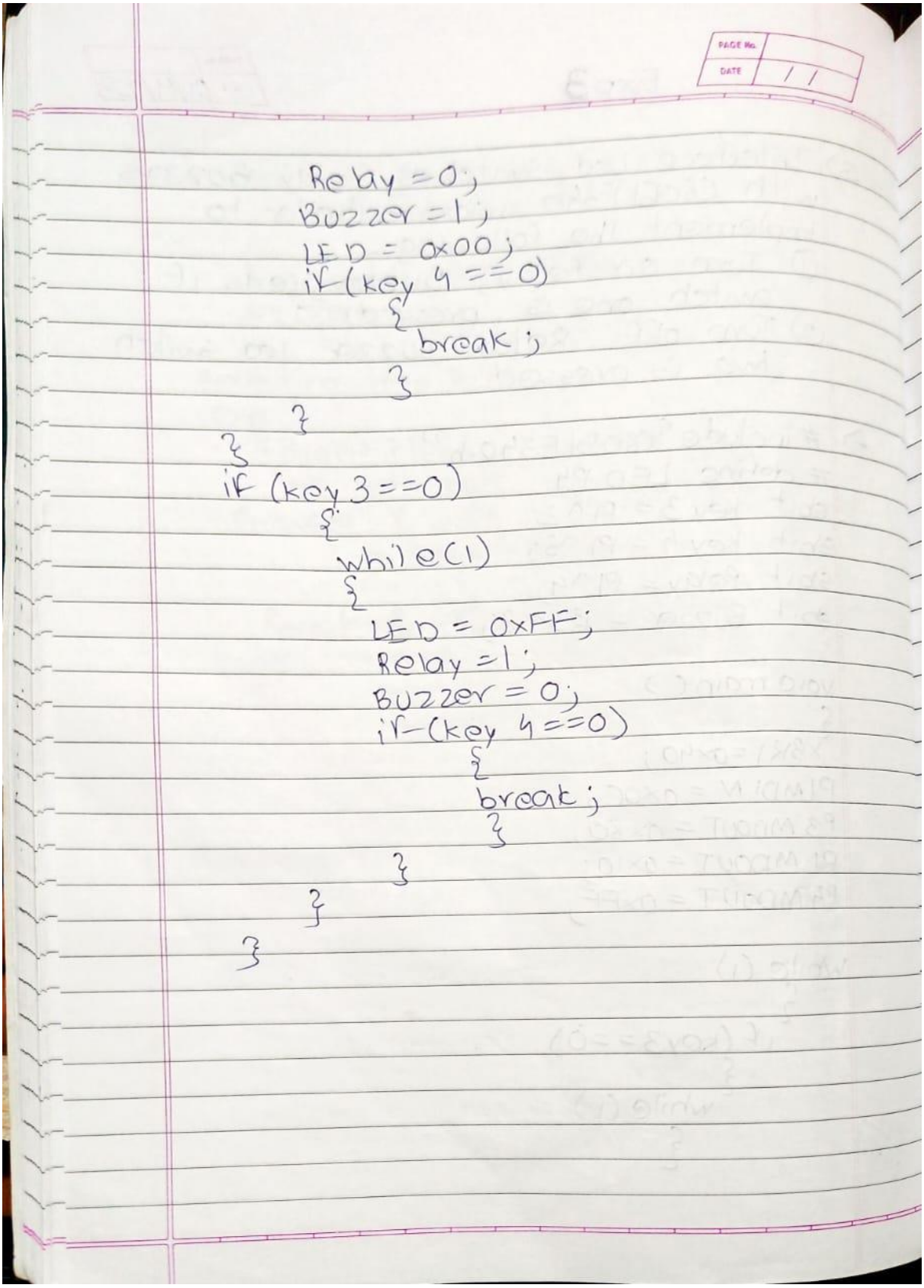
Exp 3

PAGE No.
DATE 26/9/23

3) Interface Led, switches, relay Buzzers with C8051F340 microcontroller to implement the following -
① Turn ON Relay, Buzzer, Leds if switch one is pressed.
② Turn off Relay Buzzer led switch two is pressed.

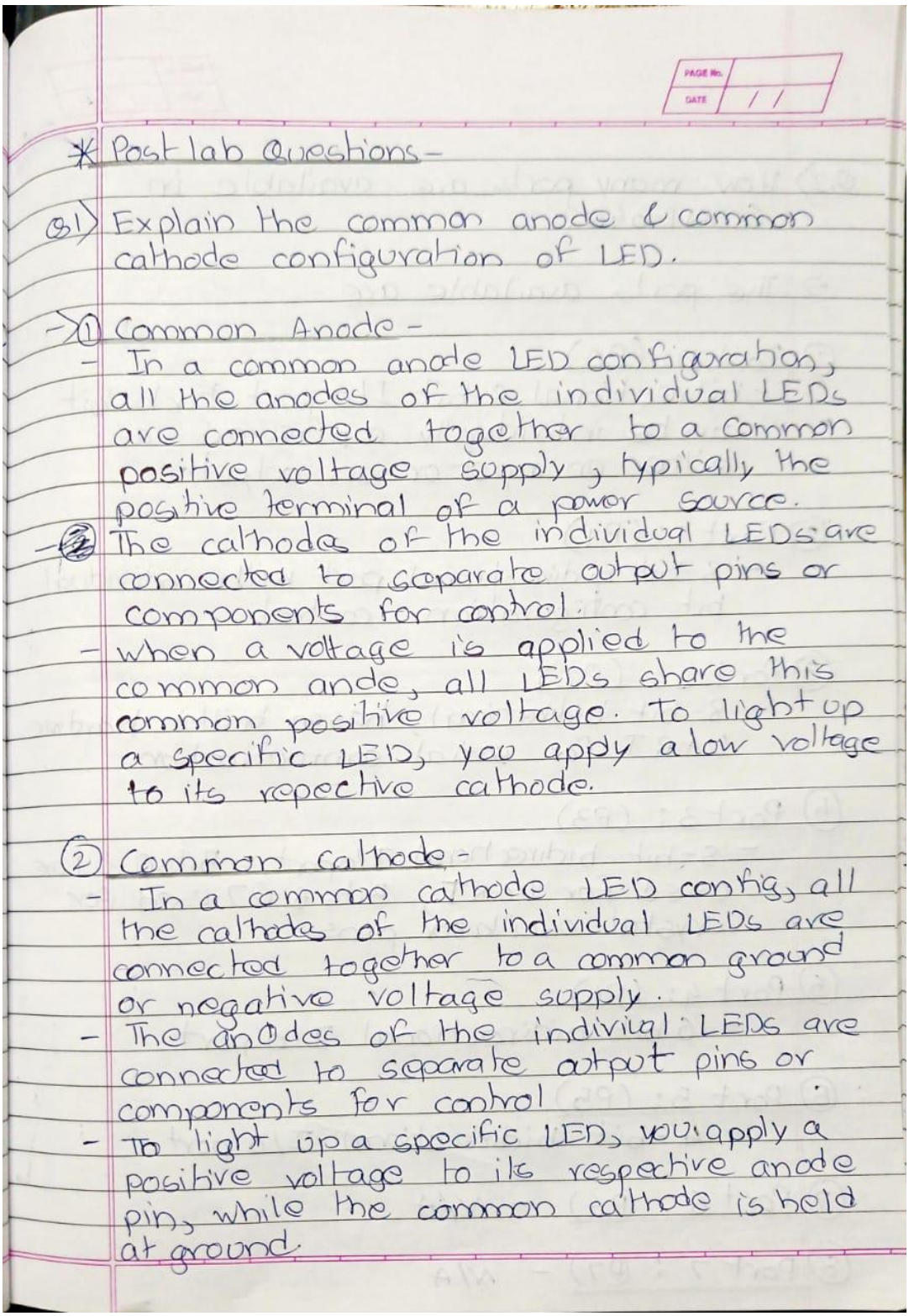
->
```
#include "C8051F340.h"
#define LED P4
sbit key3 = P1^2;
sbit key4 = P1^3;
sbit Relay = P1^4;
sbit Buzzer = P3^3;

void main()
{
 XBR1 = 0x40;
P1MDIN = 0x0C;
P3MDOUT = 0x80;
P1MDOUT = 0x10;
P4MDOUT = 0xFF;

while(1)
   {
     if(key3 == '0')
       {
         while(1)
           {
```

KDZ
26/9/23



```
Relay = 0;
Buzzer = 1;
LED = 0x00;
if (key 4 == 0)
    {
      break;
    }
  }
}
if (key 3 == 0)
  {
    while(1)
    {
      LED = 0xFF;
      Relay = 1;
      Buzzer = 0;
      if (key 4 == 0)
          {
            break;
          }
    }
  }
}
```

POST LAB QUESTIONS:



PAGE No.
DATE / /

* Post lab Questions -

Q1) Explain the common anode & common cathode configuration of LED.

-> (1) Common Anode -
- In a common anode LED configuration, all the anodes of the individual LEDs are connected together to a common positive voltage supply, typically the positive terminal of a power source.
- (2) The cathodes of the individual LEDs are connected to separate output pins or components for control.
- when a voltage is applied to the common ande, all LEDs share this common positive voltage. To light up a specific LED, you apply a low voltage to its repective cathode.

(2) Common cathode -
- In a common cathode LED config, all the cathodes of the individual LEDs are connected together to a common ground or negative voltage supply.
- The anodes of the indivilal LEDs are connected to separate output pins or components for control.
- To light up a specific LED, you apply a positive voltage to its respective anode pin, while the common cathode is held at ground.

PAGE No.
DATE / /

Q2) How many ports are available in C8051F340?

→ The ports available are -

① Port 0: (P0)
- bidirectional 8-bit I/O port. Each bit can be individually configured as either an input or an output.

② Port 1: (P1)
- 8-bit bidirectional port with individual bit configuration control.

③ Port 2: (P2)
- 8-bit bidirectional, has an built in hardware UART for serial communication

④ Port 3: (P3)
- 8-bit bidirectional I/O port. Bits 0 & 1 are used for UART, Bits 6 & 7 used for crystal oscillator pins

⑤ Port 4: (P4)
- 6 bit bidirectional I/O port

⑥ Port 5: (P5)
- 4-bit bidirectional I/O port

⑦ Port 6: (P6) - N/A

⑧ Port 7: (P7) - N/A

PAGE No.
DATE / /

Q3) How to configure port as an input / output?

→ To configure a port as an input or output on a microcontroller, you typically set or clear the corresponding bits in the port's control register. To configure a pin as an input, clear the bit in the control register for that pin, and to configure it as an output, set the bit. Depending on the microcontroller model, you may also need to specify output mode.

Q4) Explain Priority Crossbar Decoder-

→ A Priority Crossbar Decoder is a hardware component commonly found in microcontroller or microprocessor architects It plays a crucial role in managing & priortizing interrupt requests from various sources. This decoder evaluates the priority of incoming interrupt requests and directs the CPU to service the highest-priority interrupt first. It typically uses a set of programmable registers or settings to assign priority levels to different interrupt sources. This functionality is essential for real-time systems & applications that require efficient handling of external events & interrupts.

T. Y. B. Tech (Electrical and Computer Engineering)

Trimester: V **Subject: Microcontroller and Applications**
Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 04

Name of the Experiment: Interfacing of LCD

Performed on: 03/10/2023

Submitted on: 04/11/2023

Marks	Teacher's Signature with date

Aim: Write C program for interfacing of 16x2 LCD with C8051F340 in 8-bit mode.

Apparatus: EPBF340 Board, ASK25 board, Connectors

Theory:

LCD has the ability to display letters, numbers and characters. A 16x2 LCD can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix.

LCD pin descriptions:

Vcc, Vss and Vee:

While Vcc and Vss provide +5V and ground, respectively, Vee is used for controlling LCD contrast.

Register Select (RS):

There are two very important registers inside the LCD. The RS pin is used for their selection as follows.

a. RS = 0: the instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home.
b. RS = 1: the data register is selected, allowing the user to send the data to be displayed on the LCD.

Read/write (R/W):

R/W input allows the user to write information to the LCD or read information from it. R/W = 1 when reading, R/W = 0 when writing.

Enable (EN):

The enable pin is used by the LCD to latch information presented to its data pins. When data is supplied to data pins, a high to low pulse must be applied to the pin in order for the LCD to latch in the data present at the data pins. This pulse must be a minimum of 450ns wide.

Data bus (D0 – D7):

The 8-bit data pins, D0-D7 are used to send the information to the LCD or read the contents of the LCD's internal registers. To display the numbers and letters, we send ASCII codes to these pins while making RS=1.

There are also instruction command codes that can be sent to the LCD to clear the display or blink the cursor.

We also use RS = 0 to check the busy flag bit to see if the LCD is ready to receive information. The busy flag is D& and can be read when R/W = 1 and RS=0. When D7 =1, the LCD is busy taking care of internal operations and will not accept any new information. When D7 = 0, the LCD is ready to receive new information.

Table 3.1 Pin Assignment of 16x2 LCD

Pin number	Symbol	Level	I/O	Function
1	Vss	-	-	Power supply (GND)
2	Vcc	-	-	Power supply (+5V)
3	Vee	-	-	Contrast adjust
4	RS	0/1	I	0 = Instruction input 1 = Data input
5	R/W	0/1	I	0 = Write to LCD module 1 = Read from LCD module
6	E	1, 1->0	I	Enable signal
7	DB0	0/1	I/O	Data bus line 0 (LSB)
8	DB1	0/1	I/O	Data bus line 1
9	DB2	0/1	I/O	Data bus line 2
10	DB3	0/1	I/O	Data bus line 3
11	DB4	0/1	I/O	Data bus line 4

Pin number	Symbol	Level	I/O	Function
12	DB5	0/1	I/O	Data bus line 5
13	DB6	0/1	I/O	Data bus line 6
14	DB7	0/1	I/O	Data bus line 7 (MSB)
15	VB+	1	-	Backlight Supply
16	VB-	0	-	

In 8-bit mode eight data pins are used. 8-bit ASCII value of a character is sent at a single time period and displayed on the LCD.

Interfacing Diagram:

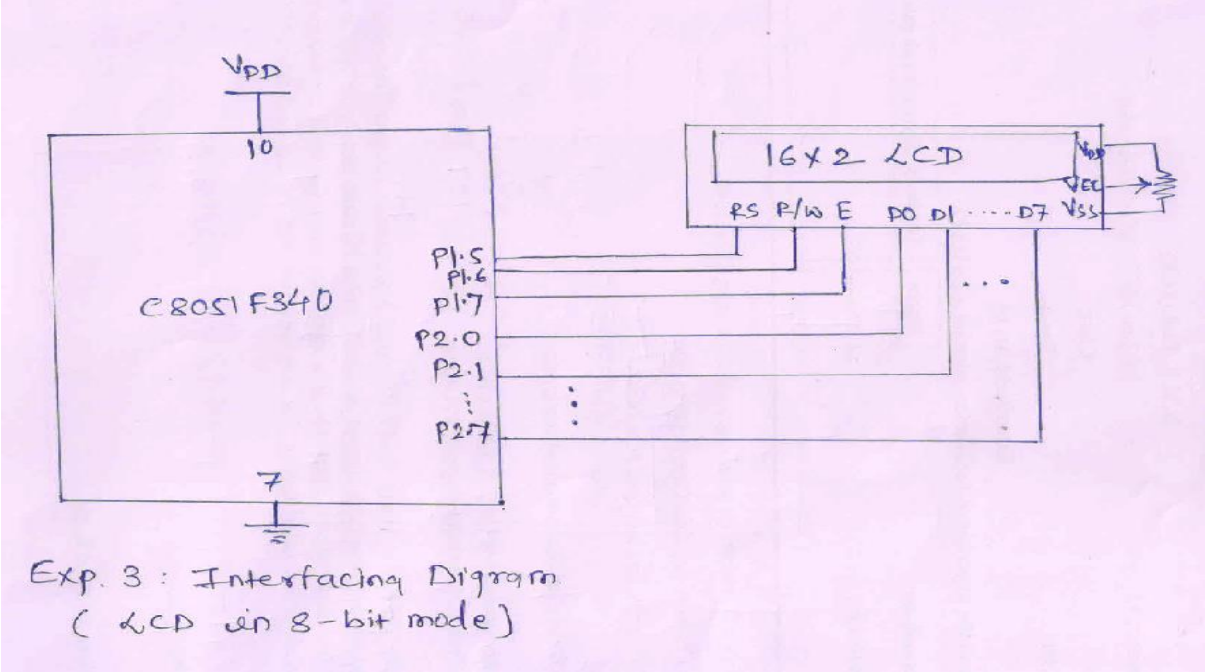


Figure 3.1 Interfacing Diagram of 16x2 LCD with C8051F340

Hardware Connections:

Connect flat cable between PL3 connector of ASK25 and PL3 connector of EPBF340 board.

Table 3.1 Hardware connections between EPBF340 and ASK25 board for LCD Interfacing

Pin Connection	PL3 Connector of ASK25	PL3 Connector of EPBF340

6	RS	P1.5
7	R/W	P1.6
8	EN	P1.7
9		
10	D0	P2.0
11	D1	P2.1
12	D2	P2.2
13	D3	P2.3
14	D4	P2.4
15	D5	P2.5
16	D6	P2.6
17	D7	P2.7
19	5V	5.0 V
20	GROUND	GND

Program: Attach printout of the tested code.

Result:
String should be displayed on the LCD.

Conclusion:

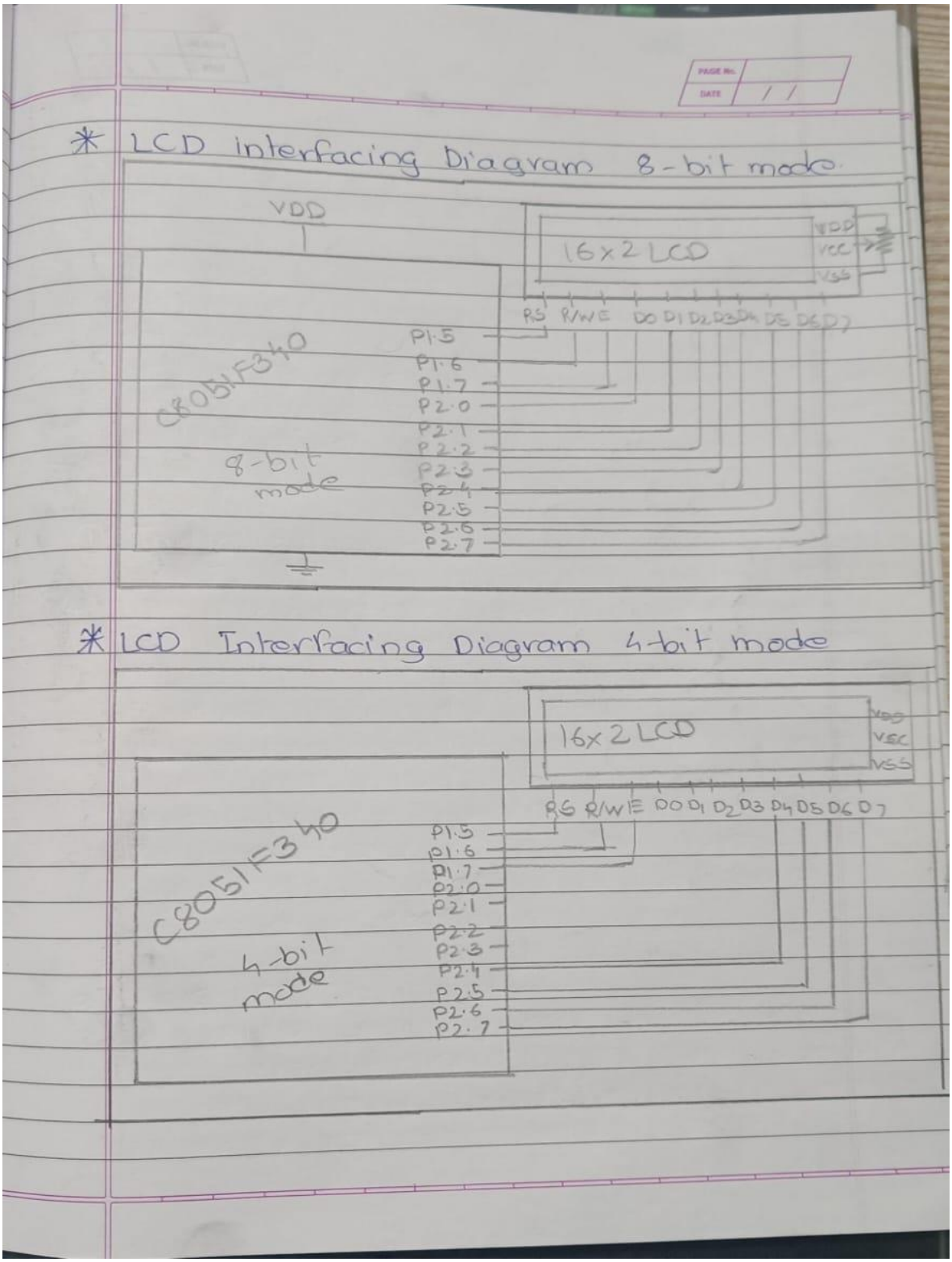
Study Question:

1. Explain the 4-bit mode of LCD.
2. Explain the significance of RS pin and list commands of LCD.
3. Explain buzy flag.

Additional link:

1. *https://www.electronicshub.org/interfacing-16x2-lcd-8051/*

LCD Interfacing Diagrams:

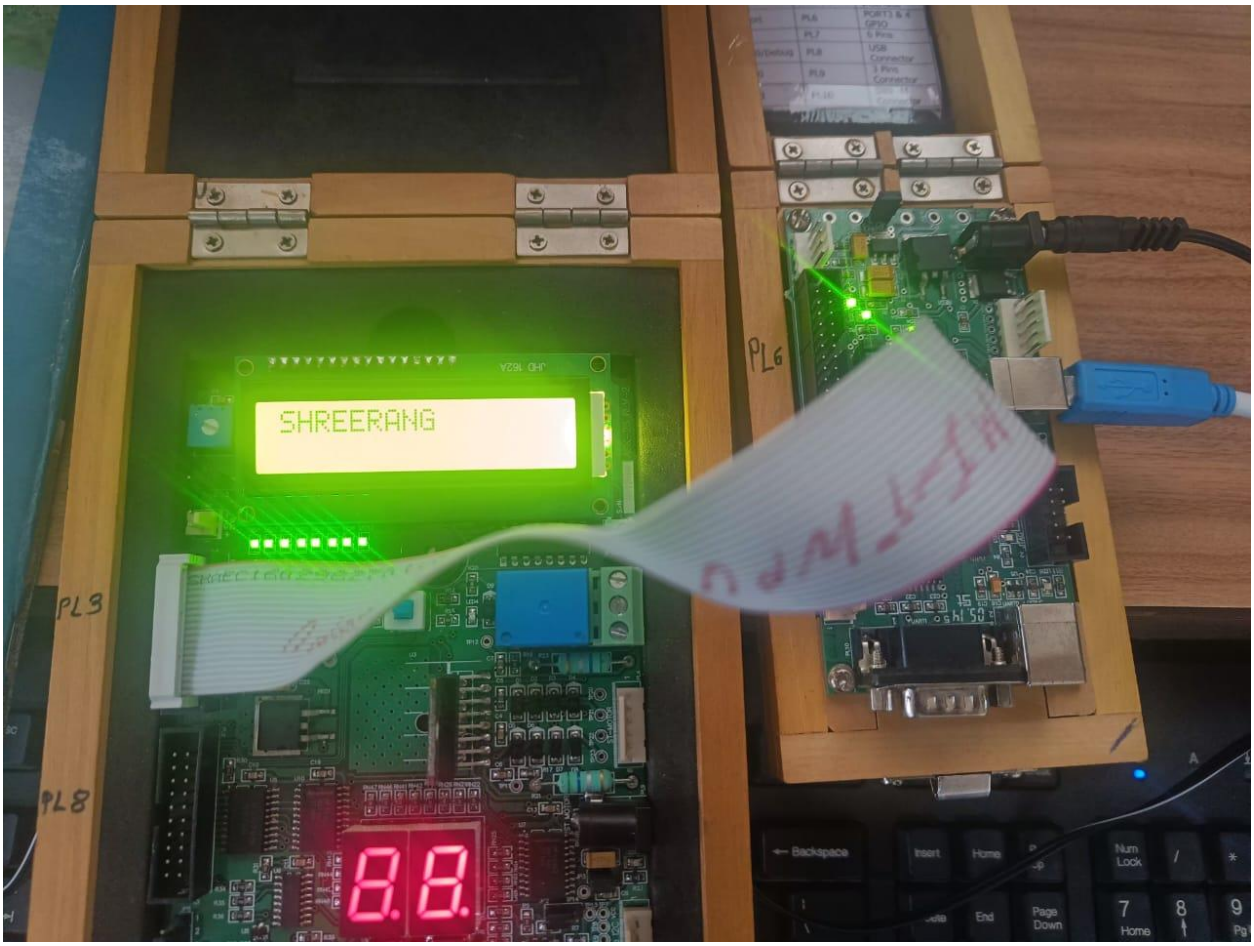


Code for normal LCD Interfacing with C8051F340

```
// Exp 4 Basic LCD Interfacing

/*
Name: Shreerang Mhatre
Rollno: 52
Class: TY
*/

#include"c8051f340.h"
void DelayMs(unsigned int Ms);
void Write_command_LCD(unsigned char character);
void Write_Data_LCD(unsigned char name);
sbit LCD_RS=P1^5;
sbit LCD_RW=P1^6;
sbit LCD_EN=P1^7;

void main()
{
    XBR1=0x40;
    P2MDOUT=0xFF;
    P1MDOUT=0xE0;

    Write_command_LCD(0x38);
    DelayMs(50);
    Write_command_LCD(0x01);
    DelayMs(50);
    Write_command_LCD(0x0C);
    DelayMs(50);
    Write_command_LCD(0x80);
    DelayMs(50);
    Write_Data_LCD('W');
    DelayMs(50);
    Write_Data_LCD('P');
    DelayMs(50);
    Write_Data_LCD('U');
    DelayMs(50);
    while(1);
}
```

```
void DelayMs(unsigned int Ms)
{
    unsigned int n;
    unsigned int i;
    for(n=0;n<Ms;n++)
    {
        for(i=0;i<65;i++);
    }
}

void Write_Command_Lcd(unsigned char command)
{
    LCD_RS=0;
    LCD_RW=0;
    P2=command;
    LCD_EN=1;
    DelayMs(15);
    LCD_EN=0;
}

void Write_Data_LCD(unsigned char character)
{
    LCD_RS=1;
    LCD_RW=0;
    P2=character;
    LCD_EN=1;
    DelayMs(15);
    LCD_EN=0;
}
```

LCD Interfacing with C8051F340 in 8-bit Mode:



CODE: LCD displaying Name In 8-bit mode

```
// Exp 4 LCD displaying Name In 8-bit mode

/*
Name: Shreerang Mhatre
Rollno: 52
Class: TY
*/

#include"c8051f340.h"
void DelayMs(unsigned int Ms);
void Write_command_LCD(unsigned char character);
void Write_Data_LCD(unsigned char name);
sbit LCD_RS=P1^5;
sbit LCD_RW=P1^6;
sbit LCD_EN=P1^7;

void main()
{
    unsigned char name[]={"SHREERANG"};
    int i;
    XBR1=0x40;
    P2MDOUT=0xFF;
    P1MDOUT=0xE0;

    Write_command_LCD(0x38);
    DelayMs(50);
    Write_command_LCD(0x01);
    DelayMs(50);
    Write_command_LCD(0x0C);
    DelayMs(50);
    Write_command_LCD(0x80);
    DelayMs(50);

    for(i=0;name[i]!='\0'; i++)
    {
        Write_Data_LCD(name[i]);
        DelayMs(50);
    }
    while(1);
```

```
}
void DelayMs(unsigned int Ms)
{
    unsigned int n;
    unsigned int i;
    for(n=0;n<Ms;n++)
    {
        for(i=0;i<65;i++);
    }
}

void Write_Command_Lcd(unsigned char command)
{
    LCD_RS=0;
    LCD_RW=0;
    P2=command;
    LCD_EN=1;
    DelayMs(15);
    LCD_EN=0;
}

void Write_Data_LCD(unsigned char character)
{
    LCD_RS=1;
    LCD_RW=0;
    P2=character;
    LCD_EN=1;
    DelayMs(15);
    LCD_EN=0;
}
```

LCD Interfacing with C8051F340 in 4-bit Mode:

CODE: LCD displaying Name In 4-bit mode

```
// Exp 4 LCD displaying Name In 4-bit mode
/*
Name: Shreerang Mhatre
Rollno: 52
Class: TY
*/

#include"c8051f340.h"
void DelayMs(unsigned int Ms);
void Write_command_LCD(unsigned char character);
void Write_Data_LCD(unsigned char name);
sbit LCD_RS=P1^5;
sbit LCD_RW=P1^6;
sbit LCD_EN=P1^7;

void main()
{
    unsigned char name[]={"SHREERANG"};
    int i;
    XBR1=0x40;
    P2MDOUT=0xFF;
    P1MDOUT=0xE0;

    Write_command_LCD(0x28);
    DelayMs(50);
    Write_command_LCD(0x01);
    DelayMs(50);
    Write_command_LCD(0x0C);
    DelayMs(50);
    Write_command_LCD(0x80);
    DelayMs(50);

    for(i=0;name[i]!='\0'; i++)
    {
        Write_Data_LCD(name[i]);
        DelayMs(50);
    }
    while(1);
}
void DelayMs(unsigned int Ms)
```

```
{
    unsigned int n;
    unsigned int i;
    for(n=0;n<Ms;n++){
        for(i=0;i<65;i++);
    }
}

void Write_Command_Lcd(unsigned char command)
{
    P2=(command & 0xF0);
    LCD_RS=0;
    LCD_RW=0;
    LCD_EN=1;
    DelayMs(15);
    LCD_EN=0;

    P2=(command & 0x0F)<<4;
    LCD_RS=0;
    LCD_RW=0;
    LCD_EN=1;
    DelayMs(15);
    LCD_EN=0;
}

void Write_Data_LCD(unsigned char character)
{
    P2=(character & 0xF0);
    LCD_RS=1;
    LCD_RW=0;
    LCD_EN=1;
    DelayMs(15);
    LCD_EN=0;

    P2=(character & 0x0F)<<4;
    LCD_RS=1;
    LCD_RW=0;
    LCD_EN=1;
    DelayMs(15);
    LCD_EN=0;
}
```

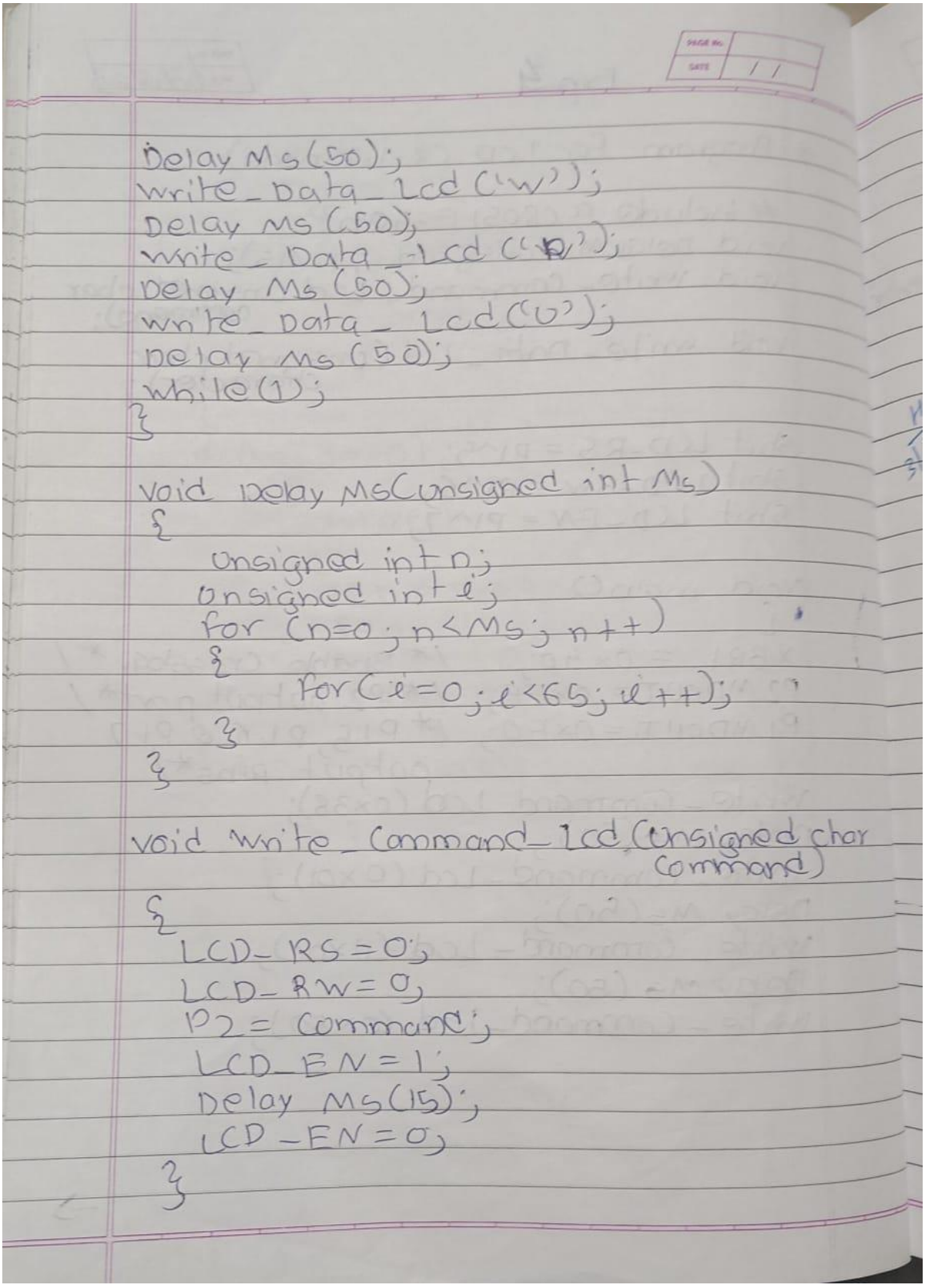
Exp 4

PAGE No.
DATE 02/10/23

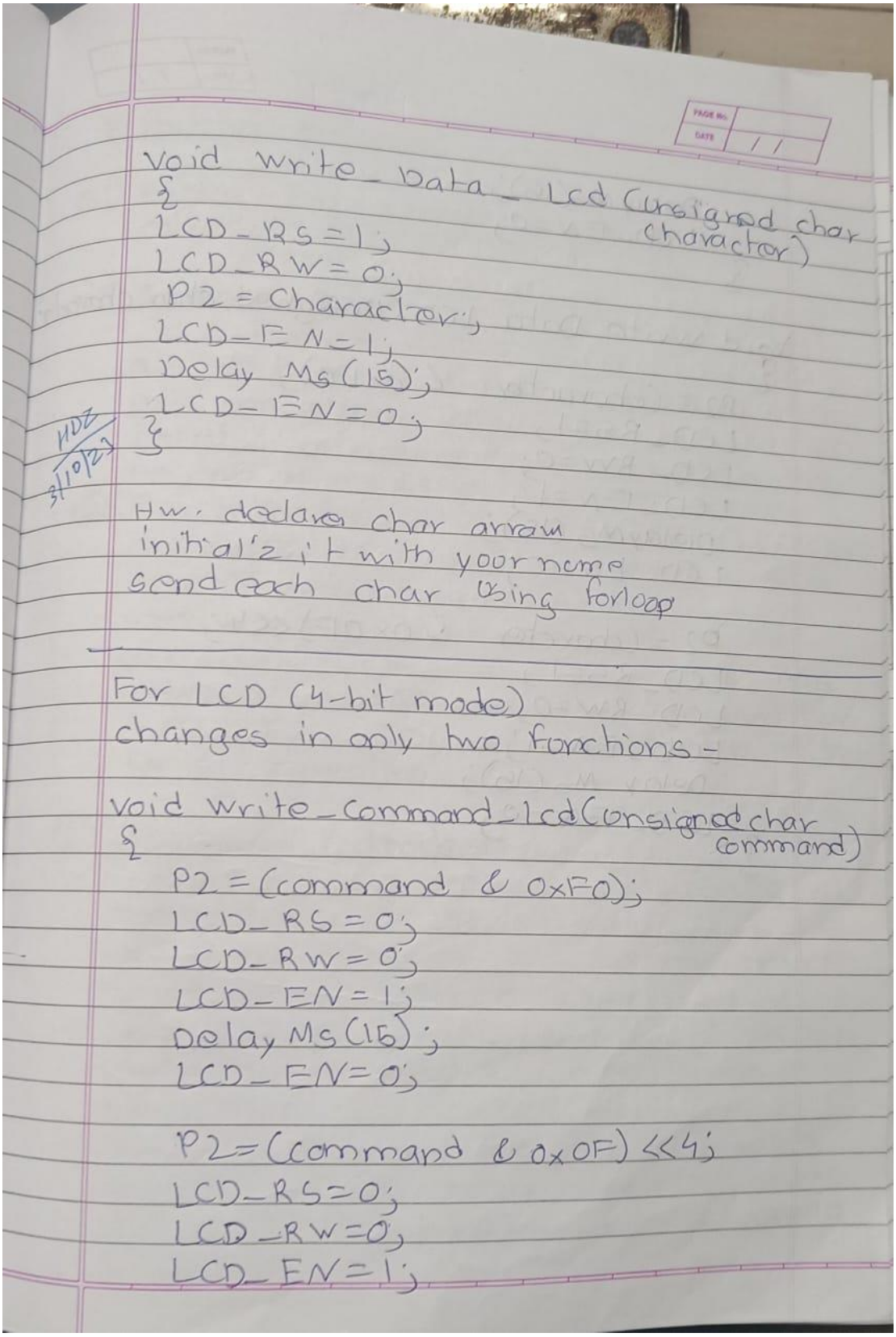
* Program for LCD (8-bit mode)

```
#include "C8051F340.h"
void DelayMs(unsigned int MS);
void write_Command_Lcd(unsigned char command);
void write_Data_Lcd(unsigned char character);

sbit LCD_RS = P1^5;
sbit LCD_RW = P1^6;
sbit LCD_EN = P1^7;

void main()
 {
XBR1 = 0x40;      /* Enable Crossbar */
P2MDOUT = 0xFF;   /* P2 output port*/
P1MDOUT = 0xE0;   /* P1.5, P1.6 & P1.7
                     output pins*/
write_Command_Lcd(0x38);
DelayMs(50);
write_Command_Lcd(0x01);
DelayMs(50);
write_Command_Lcd(0x0C);
DelayMs(50);
write_Command_Lcd(0x80);
```

→



```
Delay Ms(50);
write_Data_Lcd('w');
Delay Ms(50);
write_Data_Lcd('p');
Delay Ms(50);
write_Data_Lcd('u');
Delay Ms(50);
while(1);
}

void Delay Ms(unsigned int Ms)
{
    unsigned int n;
    unsigned int i;
    for(n=0; n<Ms; n++)
    {
        for(i=0; i<65; i++);
    }
}

void write_Command_Lcd(unsigned char
                       Command)
{
  LCD_RS=0;
  LCD_RW=0;
  P2=Command;
  LCD_EN=1;
  Delay Ms(15);
  LCD_EN=0;
}
```



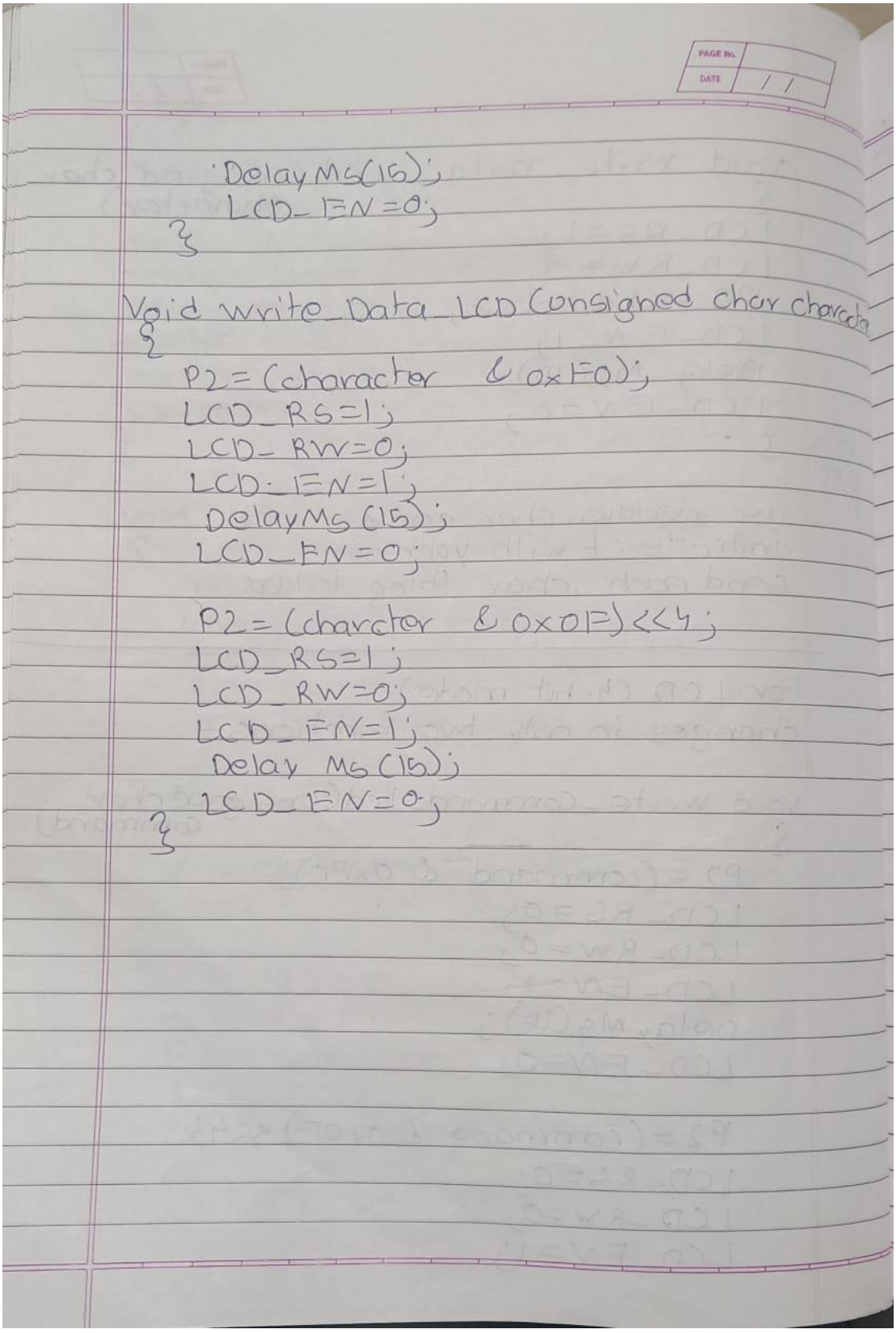
```
void write_data_lcd(unsigned char character)
{
LCD_RS=1;
LCD_RW=0;
P2=character;
LCD_EN=1;
Delay Ms(15);
LCD_EN=0;
}
```

HDZ 31/01/23

Hw. declare char arrau
initialize it with your name
send each char using forloop

For LCD (4-bit mode)
changes in only two fonctions -

```
void write_command_lcd(unsigned char command)
{
  P2=(command & 0xF0);
  LCD_RS=0;
  LCD_RW=0;
  LCD_EN=1;
  Delay Ms(15);
  LCD_EN=0;

  P2=(command & 0x0F)<<4;
  LCD_RS=0;
  LCD_RW=0;
  LCD_EN=1;
```



```
    DelayMs(15);
    LCD_EN=0;
}

Void write_Data_LCD (unsigned char character)
{
  P2=(character & 0xF0);
  LCD_RS=1;
  LCD_RW=0;
  LCD_EN=1;
   DelayMs(15);
  LCD_EN=0;

  P2=(character & 0x0F)<<4;
  LCD_RS=1;
  LCD_RW=0;
  LCD_EN=1;
   Delay Ms(15);
  LCD_EN=0;
}
```

Exp 4

PAGE NO.
DATE 3/10/23

* Post lab Questions

Q1) Explain the 4-bit mode of LCD

→ The 4-bit mode of an LCD is a communication protocol that allows a microcontroller or other controlling device to send and commands to the LCD using only 4 data lines, as opposed to the more common 8-bit mode that uses 8 data lines.

① Initialization -
To start communication with the LCD, the microcontroller sends a series of initialization commands in 8-bit mode

② Sending commands & data -
- The microcontroller sends the 4 most significant bits (MSBs) of a command or data byte to the LCD.
- The microntroller then sends the 4 least significant bits (LSBs) of the same command or data byte.

③ Store (Enable) signal -
After sending each 4-bit nibble, the microcontroller toggles the Enable (E) signal to signal the LCD that the data is ready for processing.

PAGE No.
DATE / /

Q2) Explain the significance of RS pin and list commands of LCD.

→ ① The RS (Register Select) pin on an LCD is a crucial control input that differentiates between sending commands (RS=0) for configuring the LCD, such as clearing the display or setting the cursor position, & sending character data (RS=1) to be displayed on the screen.

② Common LCD commands include clearing the display (0x01), returning the cursor to the home position (0x02), setting the entry mode (0x04), controlling display state (0x08), etc and specifying the DRAM address to position the cursor (0x80 to 0xFF).

③ These commands, combined with the RS pin's state, allow for control and interaction with the LCD to display text & graphics as desired.

Q3) Explain buzy flag.

→ The "buzy flag" is a status flag within an LCD controller that indicates whether the LCD is currently in the process of executing a command or is ready → to accept new commands or data. It serves as an status indicator.

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

T. Y. B. Tech (Electrical and Computer Engineering)

Trimester: V **Subject: Microcontroller and Applications**
Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 05

Name of the Experiment: Interfacing of 8-bit DAC with C8051F340

Performed on: 28/11/2023

Submitted on: 07/11/2023

Marks	Teacher's Signature with date

Aim: Write C program for interfacing of 8 bit DAC with C8051F340 to generate
i) Square wave
ii) Triangular wave

Apparatus: EPBF340 board, DSO, DSO probes, DAC Board

Theory:

The digital to analog converter involves translating digital information to equivalent analog information. DAC 0808 is R-2R ladder DAC giving output analog current so need to convert in voltage. I to V converter is used using LF351.DAC and LF351 require dual power supply of +15V &-15V.

DAC 0808 features:

1. 8 bit digital to analog converter
2. Low power consumption 33mW with ± 5V.
3. Power supply voltage range ± 4.5V to ± 18V.
4. Non-inverting digital inputs are TTL and CMOS compatible.
5. 16 pin DIP.
6. High speed multiplying input slew rate: 8mA/⌠s.
7. Relative accuracy - ± 0.19 % error maximum.
8. Fast settling time: 150 ns typical.
9. Full-scale current match: ± 1 LSB typical.

Applications:

1. Programmable power supply.
2. DC motor speed control.
3. Speed synthesis

Interfacing Diagram:

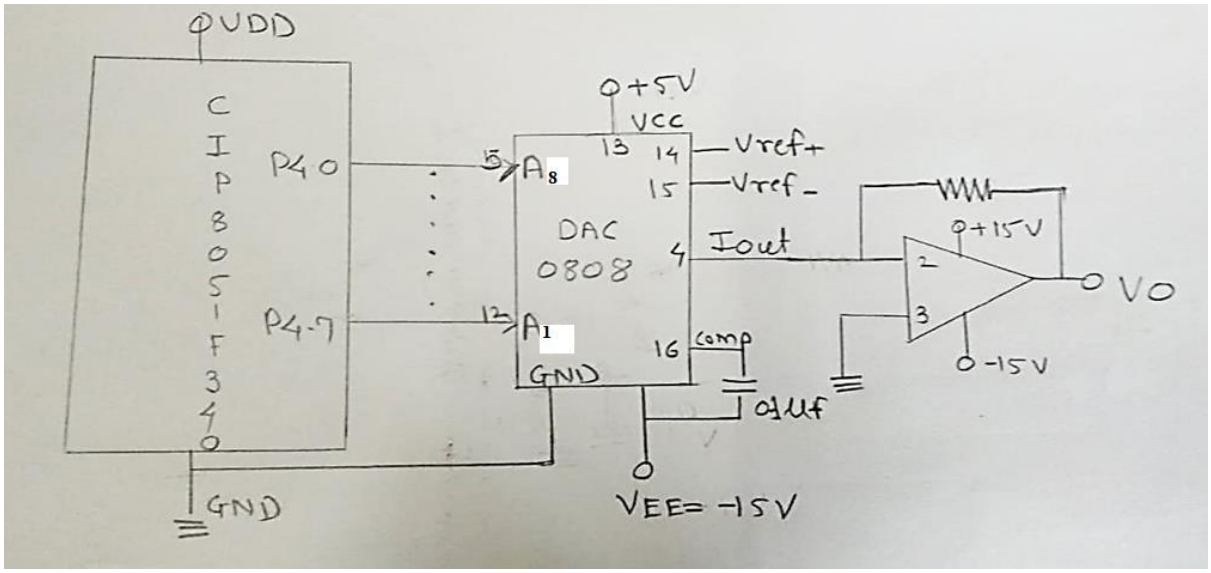


Figure 6.1 Interfacing Diagram of 8-bit DAC with C8051F340

Hardware Connections:

Connect dual power supply of 15V to DAC board. Digital data is available on P4 so Connect flat cable between PL6 connector of EPBF340 board to DAC data lines on DAC board.

Pin Connection	PL6 Connector of EPBF340	Pin Connection	DAC board data lines socket
1			
2			
3			
4	P3.3		
5			
10	P4.0	3	D0
11	P4.1	4	D1
12	P4.2	5	D2
13	P4.3	6	D3
14	P4.4	7	D4

15	P4.5	8	D5
16	P4.6	9	D6
17	P4.7	10	D7
18	3.3 V		NC
19	5.0 V	1	5.0 V
20	GND	**2**	GND

Program: Attach printout of the tested code.

Result:

Observe square and triangular wave on DSO.

Conclusion:

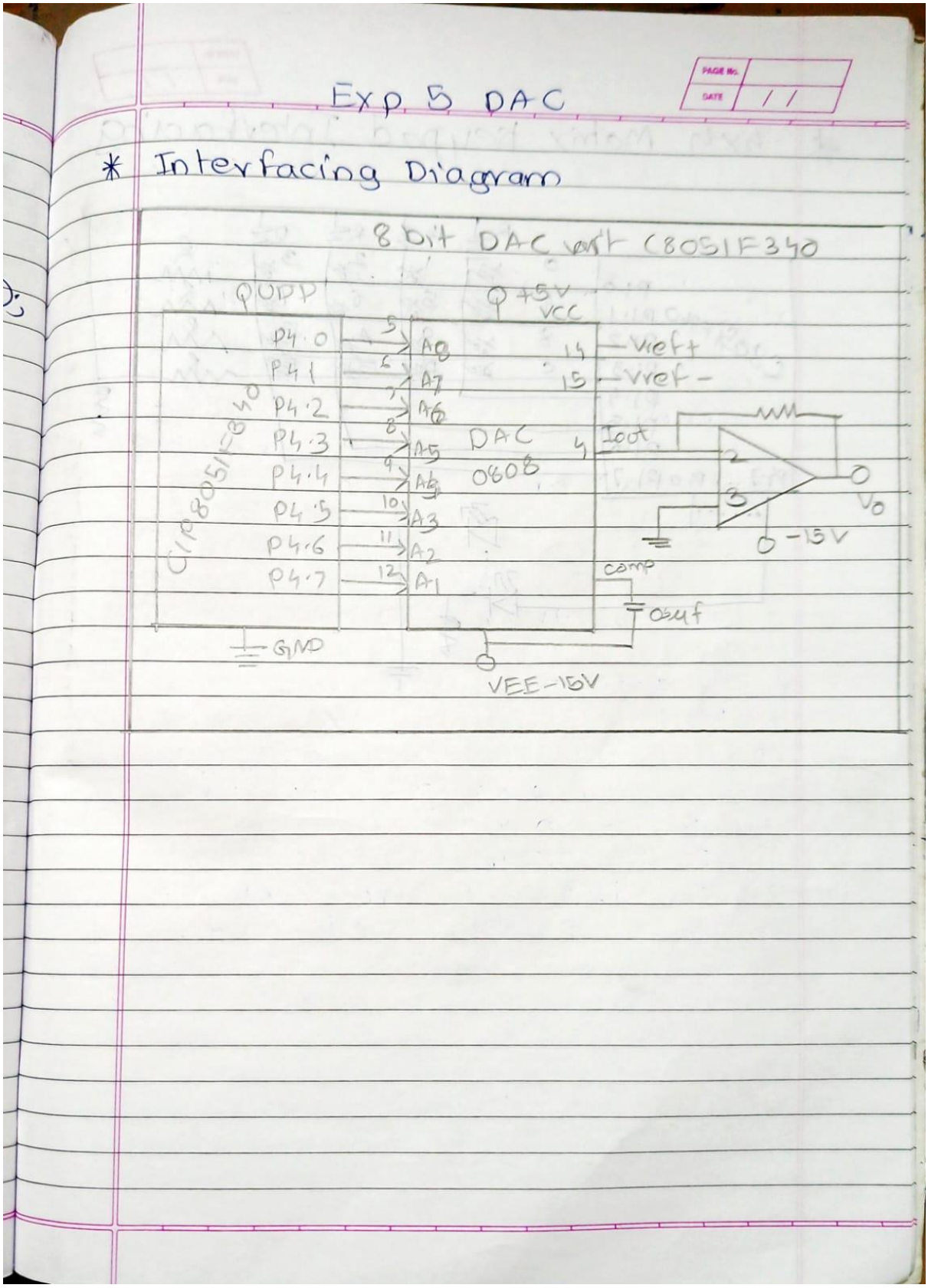
Study Questions:

1. Write a program to generate trapezoidal wave using DAC
2. Explain different types of DAC

Additional Links:

https://nptel.ac.in/courses/112103174/module2/lec8/1.html

Interfacing Diagram of 8-bit DAC with C8051F340



DAC Interfacing with C8051F340 for Square Waveform:

```
// Exp - 5 DAC Interfacing with C8051F340
/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY

DAC_Square Waveform

*/

#include"c8051f340.h"
void delay(unsigned int Ms);
void main(){
    P4MDOUT=0xff;
    while(1){
        P4=~P4;
        delay(50);
    }
}
void delay(unsigned int Ms){
    unsigned int n;
    unsigned int i;
    for(n=0;n<Ms;n++){
        for(i=0;i<65;i++);
    }
}
```

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

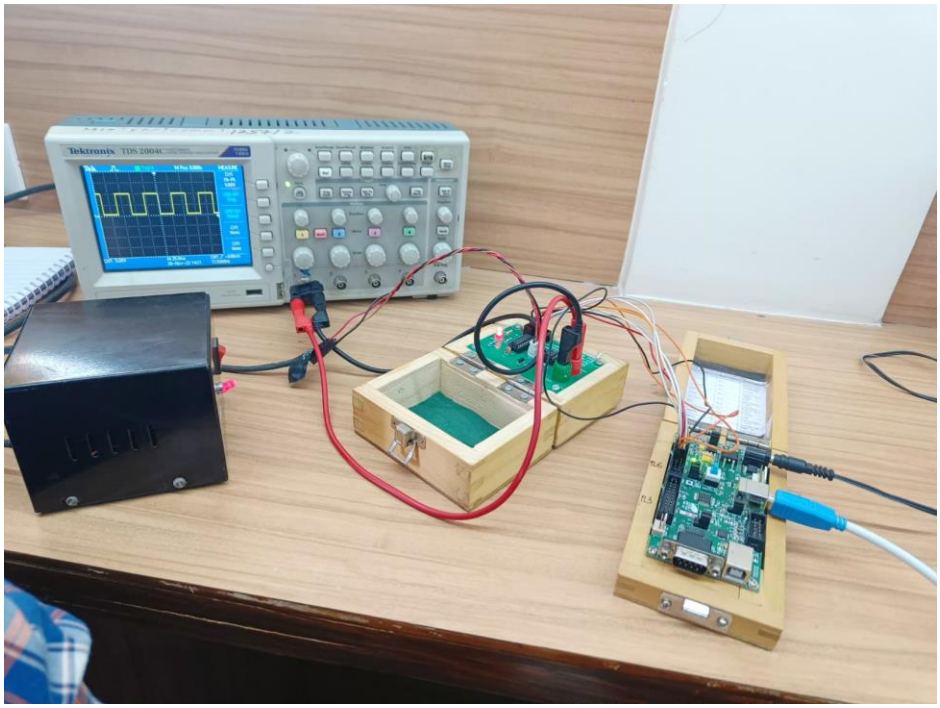
DAC Interfacing with C8051F340 for Triangular Waveform:

```
// Exp - 5 DAC Interfacing with C8051F340
/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY

DAC_Triangular Waveform

*/
#include"c8051f340.h"
void main(){
    int i;
    P4MDOUT=0xff;
    while(1){
        for(i=0; i<=254;i++){
            P4=i;
        }
        for(i=255; i>=1; i--){
            P4=i;
        }
    }
}
```

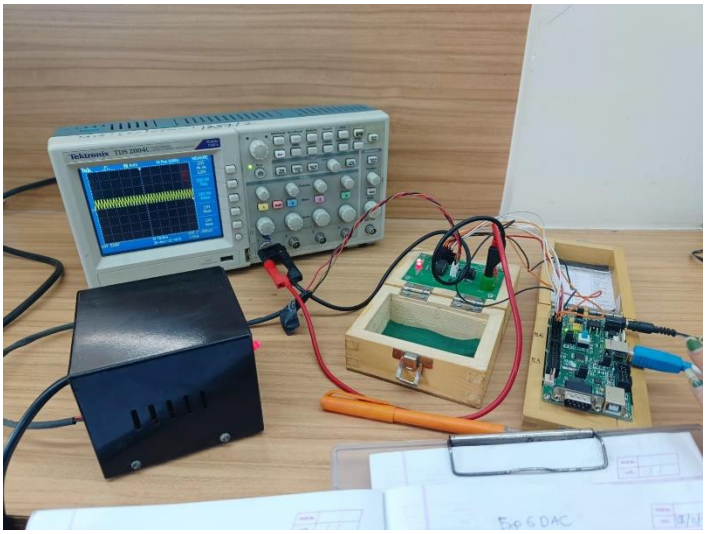
DAC Interfacing with C8051F340 for Sawtooth Waveform:

```
// Exp - 5 DAC Interfacing with C8051F340
/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY

DAC_Sawtooth Waveform

*/

#include"c8051f340.h"
void main(){
    int i;
    P4MDOUT=0xff;
    while(1){
        for(i=0; i<=254;i++){
            P4=i;
        }

    }
}
```

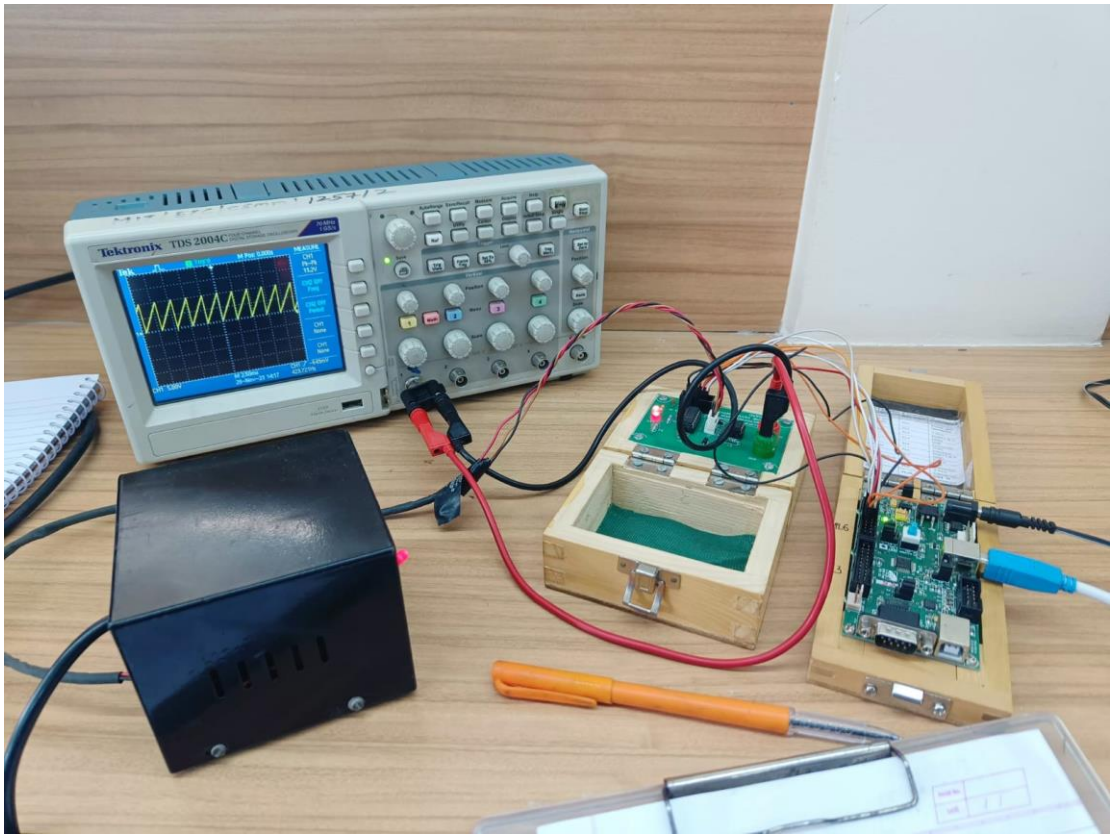
Output for Square Waveform:



Output for Triangular Waveform:



Output for Sawtooth Waveform:



PAGE No.
DATE / /

Exp5 DAC

	Amplitude	frequency	Time period
① square	9.60 V	17.50 Hz	25.0ms
② Triangular (256)	10.2 V	17.8.6 Hz	5.60ms
③ Triangular (128)	5.20 V	110 Hz	10.00ms
④ sawtooth	11.2 V	429.72 Hz	2.50ms

TIDE
28/11/23

PAGE No.
DATE / /

* Post lab Questions

Q1) Write a program to generate trapezoidal wave using DAC

→

```
# include "C8051F340.h"
Void main () {
  int i;
  P4MDOUT = 0xFF;
  while (1) {
    for (int i=0; i<=254; i++) {
        P4 = i;
    }
    delay (80);
    for (int i=255; i>=1; i--) {
        P4 = i;
    }
    delay(80);
  }
}
Void delay (unsigned int ms) {
  unsigned int n;
  unsigned int i;
  for (n=0; n<ms; n++) {
      for (j=0; j<65; j++);
  }
}
```

PAGE No.
DATE / /

(Q2) Explain different types of DAC.

→ (1) Binary weighted resistor DAC - utilizes a ladder network of resistors
(2) Each bit in digital input contributes to the o/p voltage through a specific resistor.
(3) Precision & linearity depend on resistor matching
(4) R-2R ladder DAC employs a ladder network of resistors in a specific arrangement.

Resistors are in either 2R or R configuration
(1) Segmented DAC (Digital Potentiometer) consists of multiple resistors segments that can be individually switched in or out.
(2) PWM DAC (Sigma delta DAC. converts digital input to analog by averaging the duty cycle of a high frequency pulse train.
(5) Achievers high resolution & good linearity.

→

T. Y. B. Tech (Electrical and Computer Engineering)

Trimester: V **Subject: Microcontroller and Applications**
Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: 3**

Experiment No: 06

Name of the Experiment: Programming of on chip ADC

Performed on: 21/11/2023

Submitted on: 24/11/2023

Marks	Teacher's Signature with date

Aim: Write C program for programming of on chip ADC of C8051F340.

Apparatus: EPBF340 Board, ASK25 board, Connectors

Theory:

Analog to digital converter is among the most widely used device for data acquisition. It is used to convert the analog signals to digital numbers so that microcontroller can read and process them.

On-chip ADC Features:

- 10-Bit ADC
- Up to 200 ksps
- Built-in analog multiplexer with single-ended and differential mode
- V_{REF} from external pin, internal reference, or VDD

Interfacing Diagram:

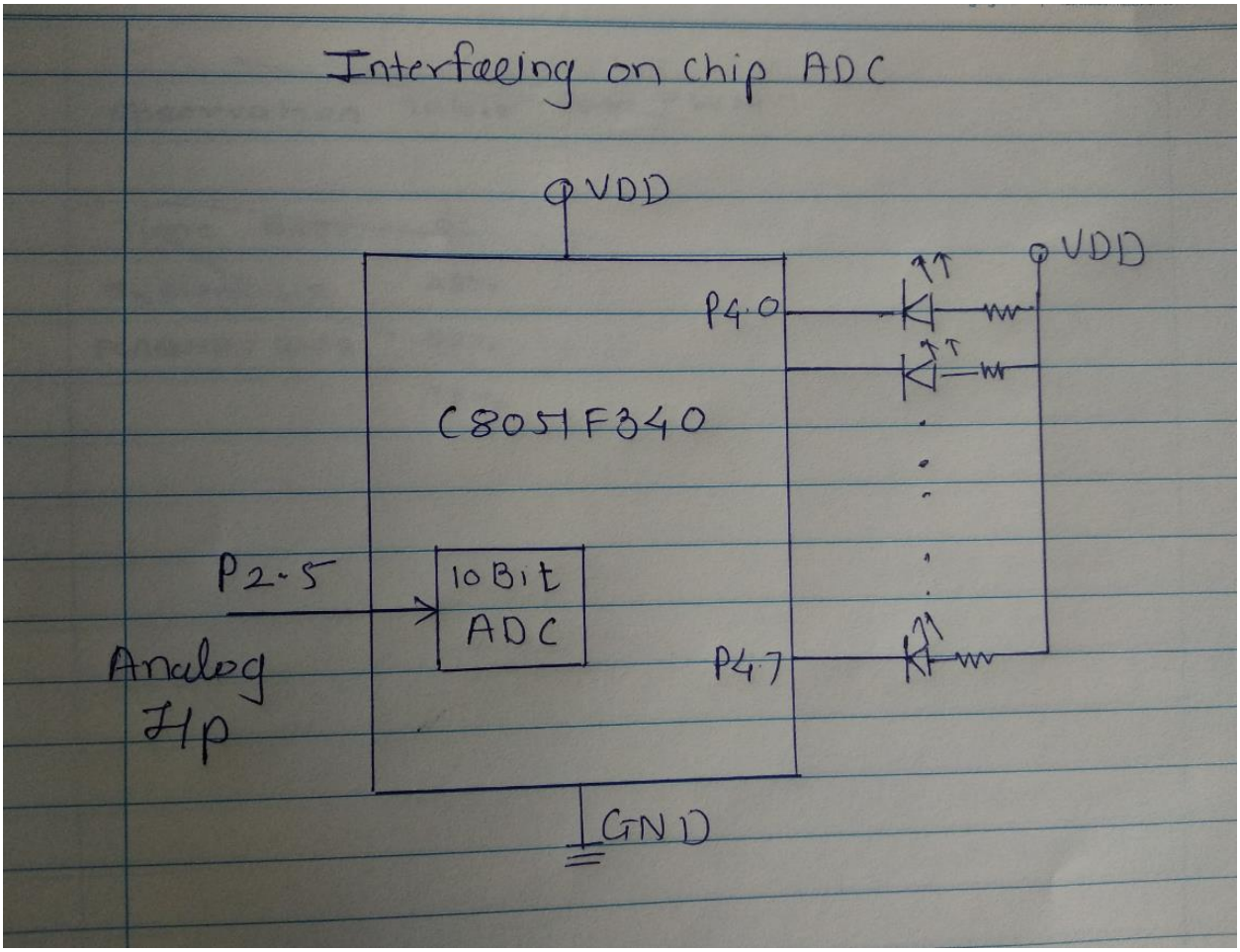


Figure 5.1 Interfacing Diagram for onchip ADC programming

Algorithm:

Hardware Connections: Connect single lead wire between P2.5 (Pin15 of PL3 connector) of EPBF340 board and Pin1 of PL10 connector of ASK25. To provide the ground also connect 20pin flat cable between PL6 connector of EBF340 board and PL8 connector of ASK25.

Connect USB cable between PL8 connector of EPBF340 board and PC.

F340 Reference	Device ASK25
P2.5 (Pin15 of PL3 connector)	Pin 1 of PL10 connector (Pot RV2)
PL6	PL8

Program: Attach the tested code.

Calculations:

Dout= (Vin/Vref)*1023

Table 1

Vin(Given)	Dout (Decimal)	Dout (Hex)	Dout (Binary)
3.3 V	**1023**	**3FF**	**1111111111**
1V			
2.3V			
2V			
1.3 V			

Table 2

Dout (Binary)	Dout (Hex)	Dout (Decimal)	Vin (Calculated)
0011101101			
1110001100			
0101100110			
1100011100			
1010011011			
1111111111			

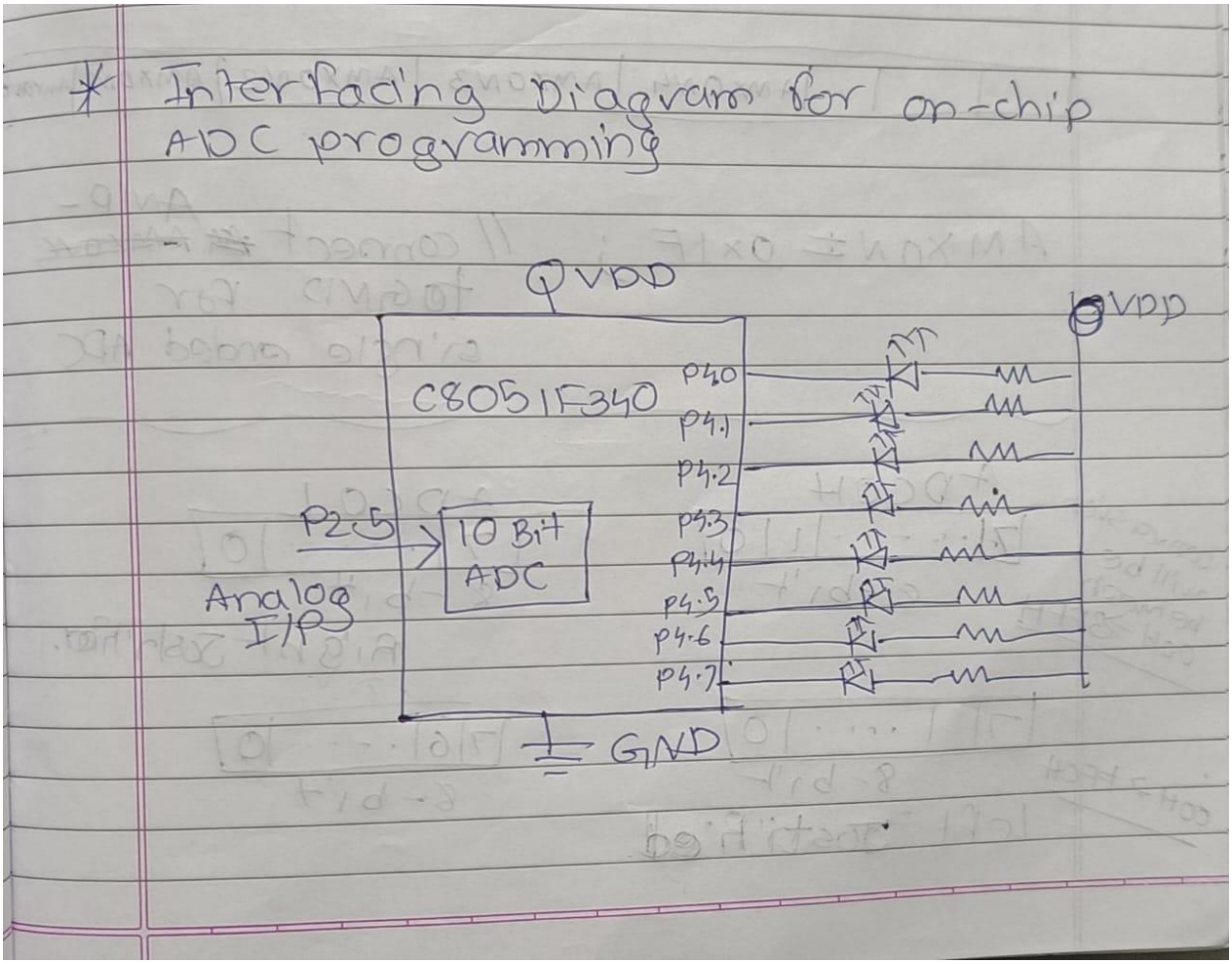
Conclusion:

Study Question:

1. Give the main factor affecting the step size of ADC in C8051F340.

2. Give the formats for control registers associated with C8051F340 ADC.

Interfacing Diagram:

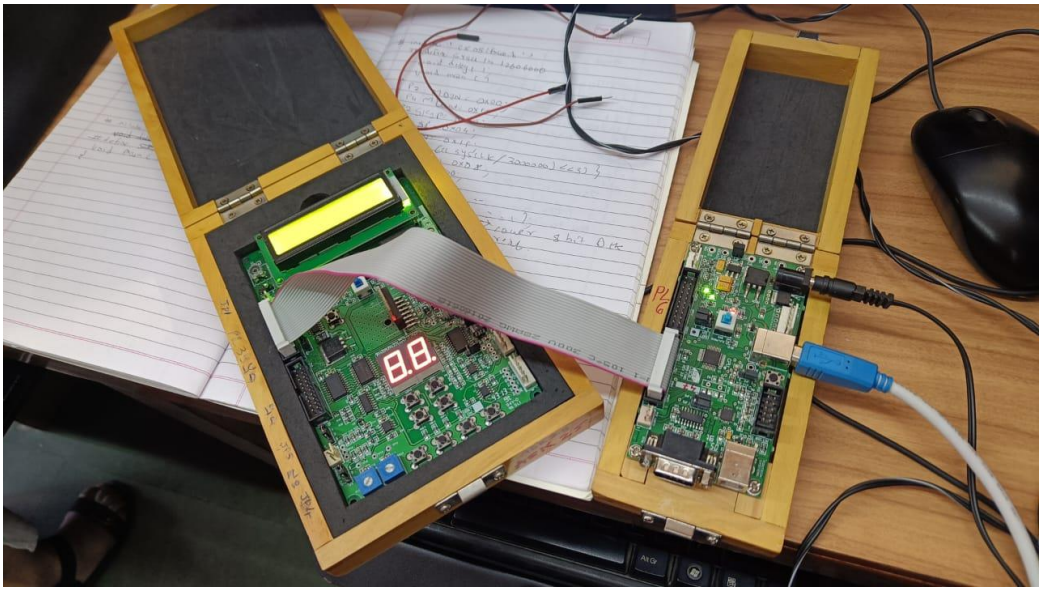


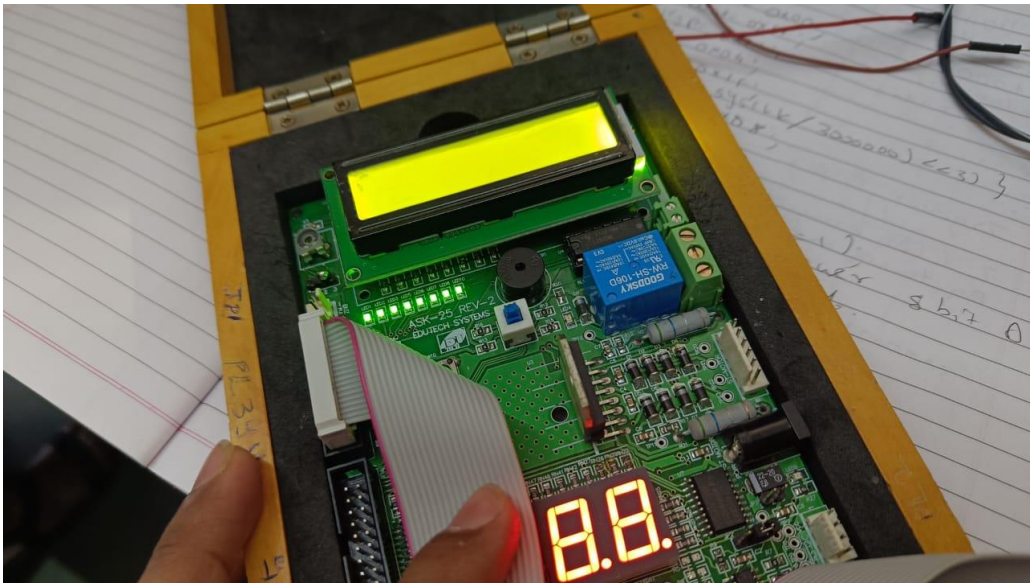
Code For ADC:

```
// Exp - 5 ADC Interfacing with C8051F340
/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY
*/

#include "C8051F340.h"
#define SYSClk 12000000
sbit Buzzer= P3^3;
void delay(unsigned int Ms);
void main()
{
    XBR1= 0X40;
    P4MDOUT= 0XFF;
    www.mitwpu.edu.in
    Buzzer= 0;
    P2SKIP= 0X20;
    P2MDIN= 0XD0;
    AMX0P= 0X04;
    AMX0N= 0x1F;
    ADC0CF= (((SYSClk/3000000)-1)<<3);
    REF0CN= 0x08;
    ADC0CN= 0x80;
    AD0EN= 1;
    {
        ADC0CN =0x90;
        while (AD0BUSY == 1);
        delay(50);
        P4= ~ ADC0L;
        delay(50);
        P4= ~ ADC0H;
        delay(50);
    }
while(1);
}
void delay(unsigned int Ms)
{
```

```
    unsigned int n;
    unsigned int i;
    for (n-0; n<Ms; n++)
    {
        for (i=0; i<65; i++);
        www.mitwpu.edu.in
    }
}
```

Demonstration Of ADC:





Exp 5 ADC

PAGE No.
DATE / /

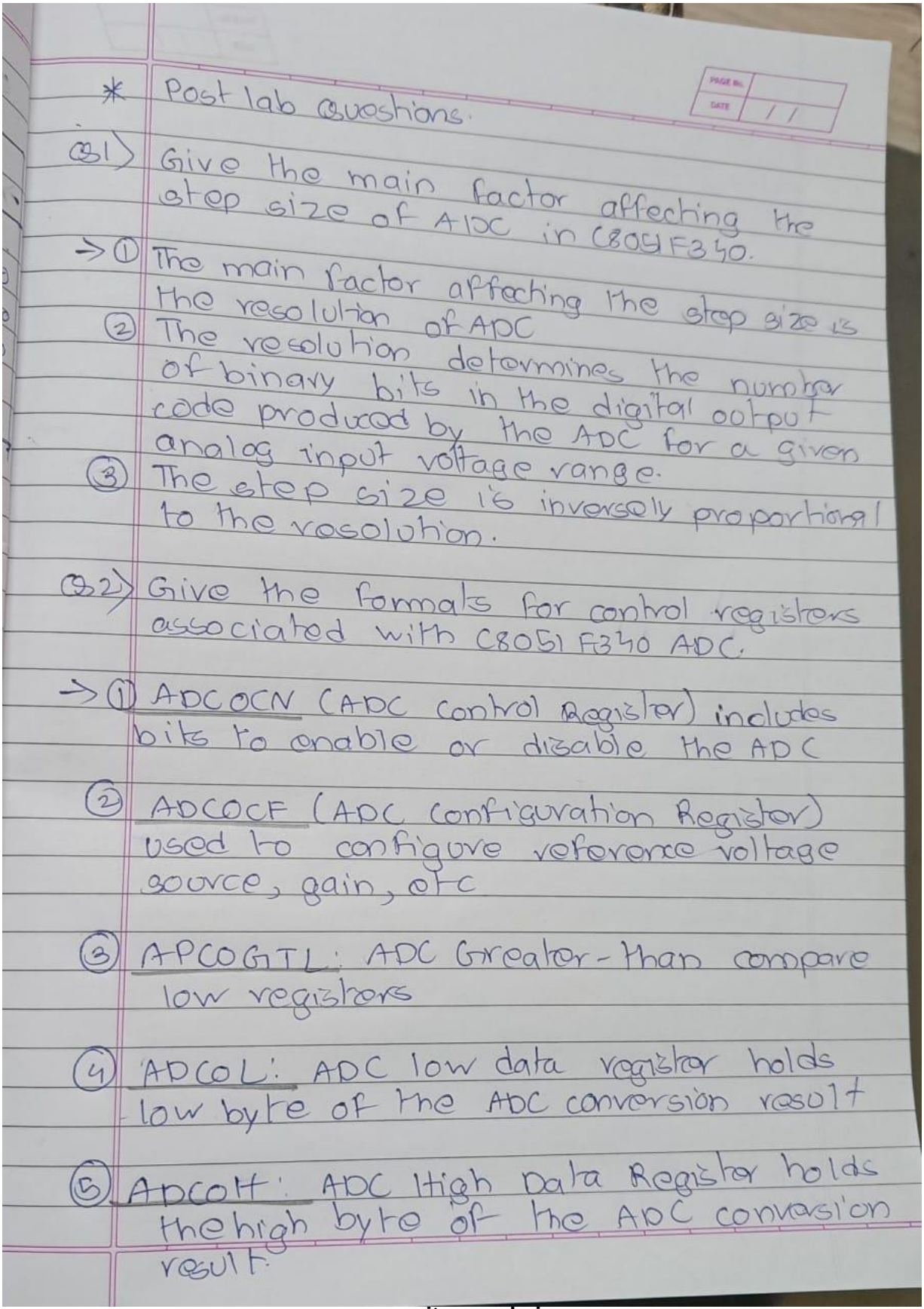
* observation table.

Dout Binary	Decimal	Vin (app)	Vin (calc)	% error
0110010110	406	1.24	1.30	4.83%
1000111010	570	1.75	1.83	4.57%
1010010100	660	2.08	2.12	1.92%
1100100110	806	2.53	2.6	2.76%
1110111111	959	3.03	3.09	1.98%

$$Dout = \frac{Vin}{Vref} \times 1023$$

$$Vin(calc) = \frac{Dout \times Vref}{1023}$$

$$= \frac{406 \times 3.3}{1023}$$



* Post lab questions.

Q1) Give the main factor affecting the step size of ADC in C8051F340.

→ ① The main factor affecting the step size is the resolution of ADC
② The resolution determines the number of binary bits in the digital output code produced by the ADC for a given analog input voltage range.
③ The step size is inversely proportional to the resolution.

Q2) Give the formats for control registers associated with C8051 F340 ADC.

→ ① ADC0CN (ADC control Register) includes bits to enable or disable the ADC

② ADC0CF (ADC configuration Register) used to configure reference voltage source, gain, etc

③ ADC0GTL: ADC Greater-than compare low registers

④ ADC0L: ADC low data register holds low byte of the ADC conversion result

⑤ ADC0H: ADC High Data Register holds the high byte of the ADC conversion result.

T. Y. B. Tech (Electrical and Computer Engineering)

Trimester: V **Subject: Microcontroller and Applications**
Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 07

Name of the Experiment: Generation of PWM using C8051F340 to control speed of DC motor
Performed on: 28/11/2023

Submitted on: 07/12/2023

Marks	Teacher's Signature with date

Aim: Write C program to generation PWM using C8051F340 to control speed of DC motor

Apparatus: EPBF340 board, DSO, DSO probes, DC motor

Theory:

DC Motors: A direct current (DC) motor is widely used device that translate electrical pulses into mechanical movement. In the DC motor we have only + and _ leads. Connecting them to a DC voltage source moves the motor in one direction . By reversing the polarity, the DC motor will move in the opposite direction.

Unidirectional control:

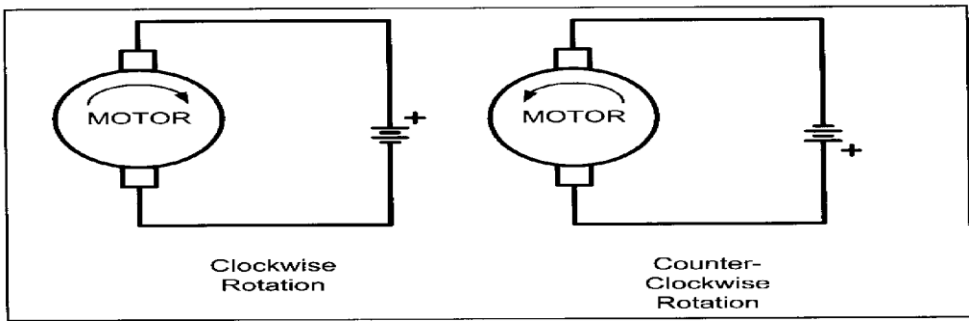


Figure 7.1: Unidirectional control of DC motor

Pulse Width Modulation (PWM):

The speed of motor depends on the three factors: i) load, ii) voltage, and iii) current. For a given fixed load we can maintain the steady speed by using a method called pulse width modulation (PWM). By changing (modulating) the width of the pulse applied to DC motor we can increase

or decrease the amount of power provided to the motor , thereby increasing or decreasing the motor speed. Notice that although the voltage has a fixed amplitude, it has a variable duty cycle.

PWM generation in C8051F340:

The Programmable Counter Array (PCA0) provides enhanced timer functionality. The PCA consists of a dedicated 16-bit counter/timer and five 16-bit capture/compare modules. Each module can be used independently to generate a pulse width modulated (PWM) output on its associated CEXn pin. The frequency of the output is dependent on the timebase for the PCA counter/timer. The duty cycle of the PWM output signal is varied using the module's PCA0CPLn capture/compare register.

Interfacing Diagram:

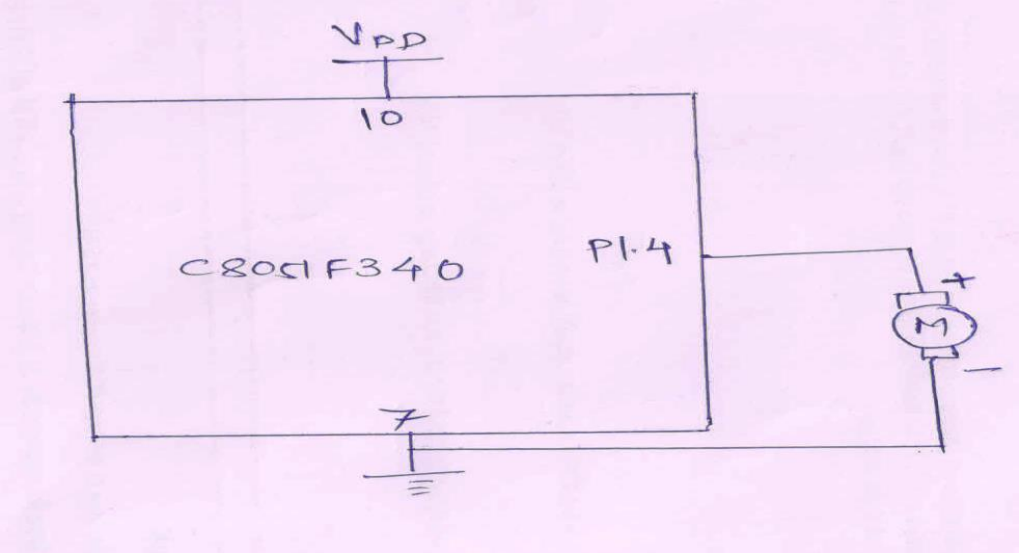


Figure 7.2 Interfacing Diagram of DC motor with C8051F340

Calculations of duty cycle: *DutyCycle = (*256-*PCA0CPHn)/256*

Hardware Connections: Output is available on Port pin P1.4. Observe waveform at pin no. 5 of PL3 connector of EPBF340 board with respect to ground on DSO/CRO. After this, connect DC motor between the same pin.

Program: Attach printout of the tested code.

Calculations:

Find the value to be loaded in PCA0L for generating the PWM waveform of following frequencies and duty cycle: Consider System clock = 12 MHz

Desired Frequency and Duty cycle	PCA0L	PCA0CPH0
60KHz - 50%		
100KHz - 25%		
140KHz - 75%		

Result:

The duty cycle of the PWM waveform should be observed on DSO/CRO.

OR

DC Motor should run with speed varying w.r.t the change in value of PWM.

Conclusion:

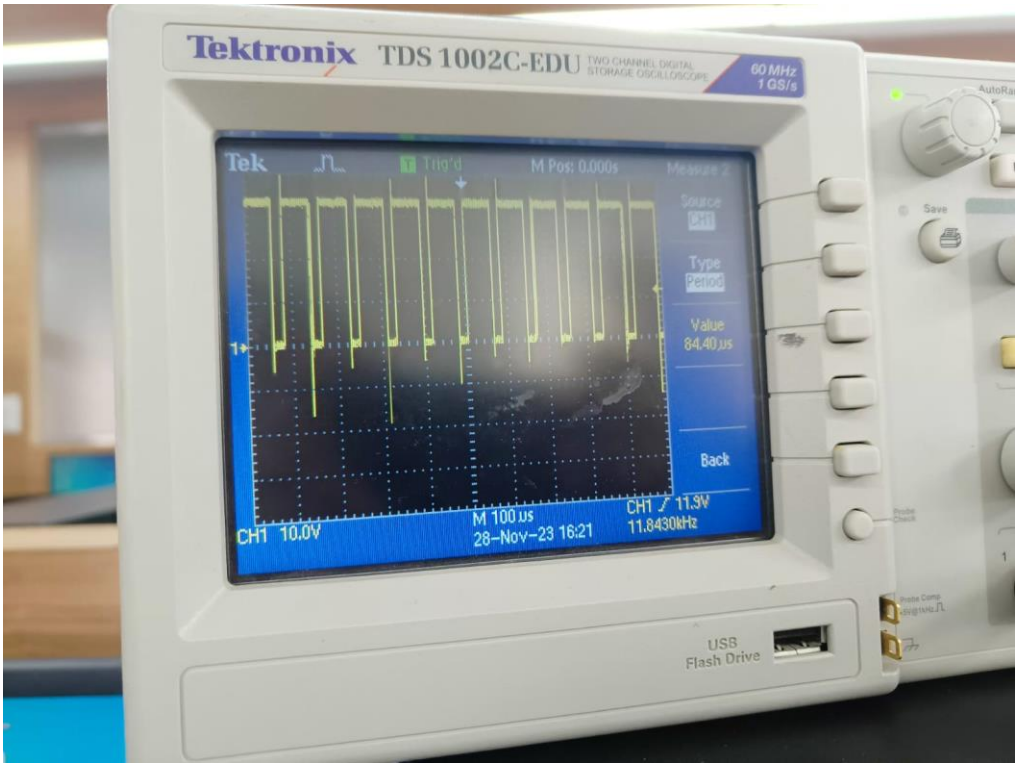
Study Question:

1. Define duty cycle.
2. Write the steps to program PCA to generate PWM
3. Write down the equations for the frequency and duty cycle of PWM in C8051F340

Code for PWM using C8051F340:

```
// Exp - 7 Generation of PWM using C8051F340 to control speed of DC motor
/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY

*/

#include "c8051f340.h"
#define SYSCLK 3000000
void main(){
    PCA0L=0x10;
    OSCICN=0x83;
    CLKSEL=0x00;
    XBR1=0x41;
    P2MDOUT=0x08;
    P0SKIP=0xff;
    P1SKIP=0xff;
    P2SKIP=0x07;
    while(1){
        PCA0MD=0x02;
        PCA0CPM0=0x42;
        PCA0CPH0=(256-(256*0.75));
        CR=1;
    }
}
```

Output:


Tektronix TDS 1002C-EDU
60 MHz 1 GS/s
Tek
M Pos: 0.000s
Source CH1
Type Period
Value 84.40 µs
Back
CH1 10.0V
M 100 µs
28-Nov-23 16:21
CH1 11.9V
11.8430kHz
USB Flash Drive

Exp 7 PWM

PAGE No.
DATE 24/11/23

```
#include<C8051F340.h>
#define SYSCLK 3000000
void () {
    PCAOL= 0x10;
    OSCICN = 0x83;
    CLKSEL = 0x00;
    XBR1 = 0x41;
    P2MDOUT = 0x08;
    P0SKIP = 0xff;
    P1SKIP = 0xFF;
    P2SKIP = 0x07;
    while (1){
        PCA0MD = 0x02;
        PCA0CPM0 =0x42;
        PCA0CPH0 = (256 - (256 * 0.75));
        CR =1;
    }
}
```

PAGE No.
DATE / /

	Time Base	Duty Cycle (DC)	Time Period Measured on DSO	Time period calculated PCA0L=0x10	PCA0CPH0 = 256 − (256 * DC)
①	SysClk	25%	21.12μs	20 μSec	192
	PCA0MD	50%	21.10μs	20 μSec	128
	=0x08	75%	21.11μs	20 μSec	64
②	SYSCLK/12	25%	253.2μs	240 μSec	192
	PCA0MD	50%	253.2μs	240 μSec	128
	=0x00	75%	253.2μs	240 μSec	64
③	SYSCLK/4	25%	84.37μs	84.37μs	192
	PCA0MD	50%	84.40μs	84.32μs	128
	=0x02	75%	84.40μs	84.37μs	64

Time Period = (256 − PCA0L) * Time for clk cycle

(256 − PCA0L) × 1μSec

240 × 1 μSec

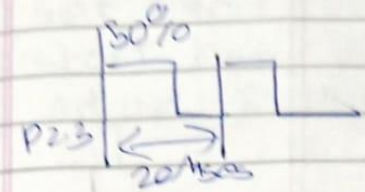
240 μSec

240 × 0.33 μSec

HDL
28/11/23

PAGE No.
DATE / /

3) Write an embedded C program for generation of PWM wave form with a frequency of 20 ms & duty cycle of 50% on pin 2.3



Sys Clk = 12MHz

1Clk cyc = $\frac{1}{12MHz}$

= 83.3nsec

count = $\frac{70\,msec}{83.3\,nsec}$

= 240

value = 256 − 240

P(A0)L = 16 = $(10)_H$

① count = $\frac{60kHz}{83.3nHz} = \frac{0.016 \times 10^{-3}}{83.3 \times 10^{-9}}$

value = 256 − 192 = ~~64 d~~ $(57)_d = (39)_H$

= ~~220 0480~~

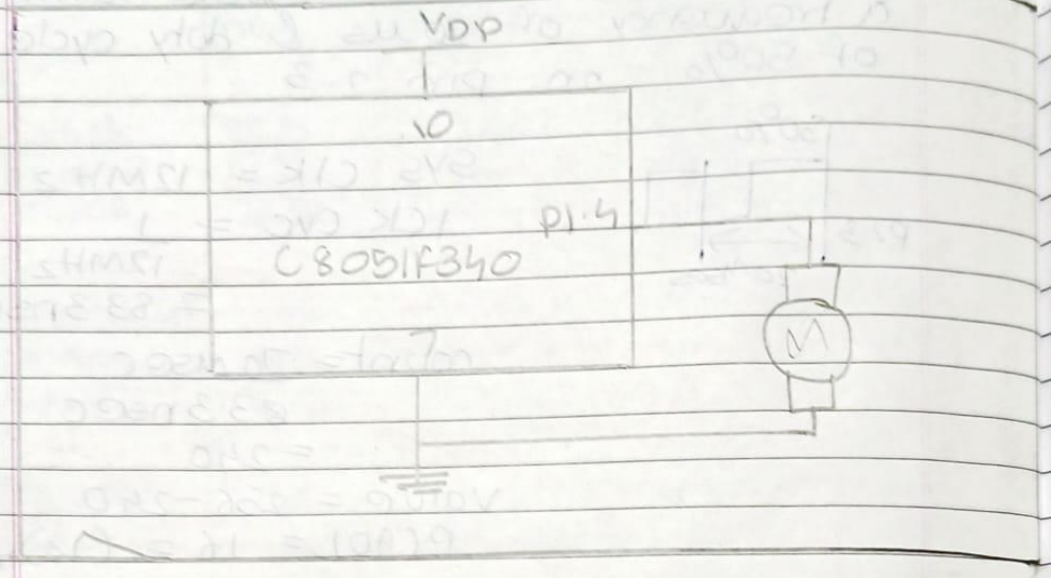
② Time period = $\frac{1}{100 \times 10^3} = 0.01\,ms$

count = $\frac{0.01\,ms}{83.3\,ns} = 120$

value = 256 − 120 = $(136)_d = (88)_H$

PAGE No.
DATE / /

* Interfacing Diagram



* Post lab Questions

Q1) Define Duty cycle -

-> Duty cycle is a measure used in electronics and engineering to describe the ratio of time of a system, device or component is active (on) compared to the total time of its operation. It is often expressed as a percentage and represents the portion of the total time that a system spends in an active state.

->

Q2) write the steps to program PCA to generate PWM.

→ ① Initialize PCA module.
② Set PWM Period and Duty Cycle
③ Start the PCA Module.
④ Adjust PWM Parameters as Needed

Q3) write down the equations for the freqency and duty cycle of PWM in C8051f340.

→

Freqency:

$$F_{PWM} = \frac{SYSCLK}{PCA\ Counter \times (2^{16} - PCA\ Module\ 0\ High\ Byte \times 256 + PCA\ Module\ 0\ Low\ Byte)}$$

Duty Cycle:

$$Duty\_Cycle = \frac{PCA\ Module\ 0\ High\ Byte \times 256 + PCA\ Module\ 0\ Low\ Byte}{2^{16}}$$

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

T. Y. B. Tech (Electrical and Computer Engineering)

Trimester: V **Subject: Microcontroller and Applications**
Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 08

Name of the Experiment: Implement UART with C8051F340

Performed on: 5/12/2023

Submitted on: 7/12/2023

Marks	Teacher's Signature with date

Aim: Write a C program for serial communication using C8051F340 to transfer data from C8051F340 to PC

Apparatus: EPBF340 Board, Connectors

Theory: Serial Communication is of two types Synchronous and Asynchronous. The asynchronous mode is used to connect the C8051F340 to PC serial port for the purpose of full duplex serial data transfer.C8051F340 has inbuilt UART (Universal Asynchronous Receiver Transmitter). Baud rate is a significant factor for serial communication of microcontroller with other devices. For communication with PC the baud rate of 9600 is selected.
baud rate generation:
Timer-1 is used to generate baud rate for mode-1 serial communication by using overflow flag of the timer to determine the baud frequency. Timer-1 is used in timer mode-2 as an auto-reload 8-bit timer. The data rate is generated by timer-1 using the following formula.

$$TH1 = 256-(SYSCLK/Desired\ baud\ rate/2)$$

Interfacing Diagram:

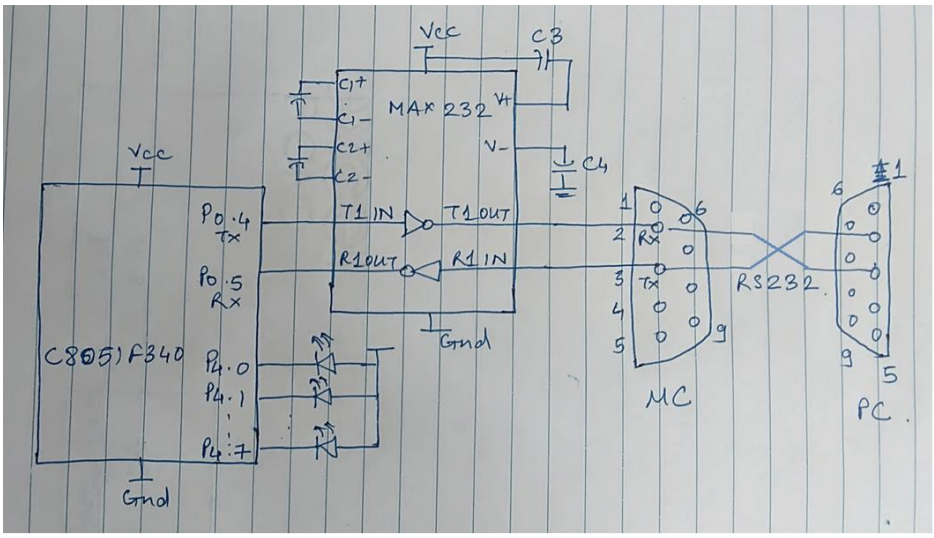


Figure 4.1 Interfacing Diagram for UART

Program:

Expected Result:

The string should be displayed on HyperTerminal.

Conclusion:

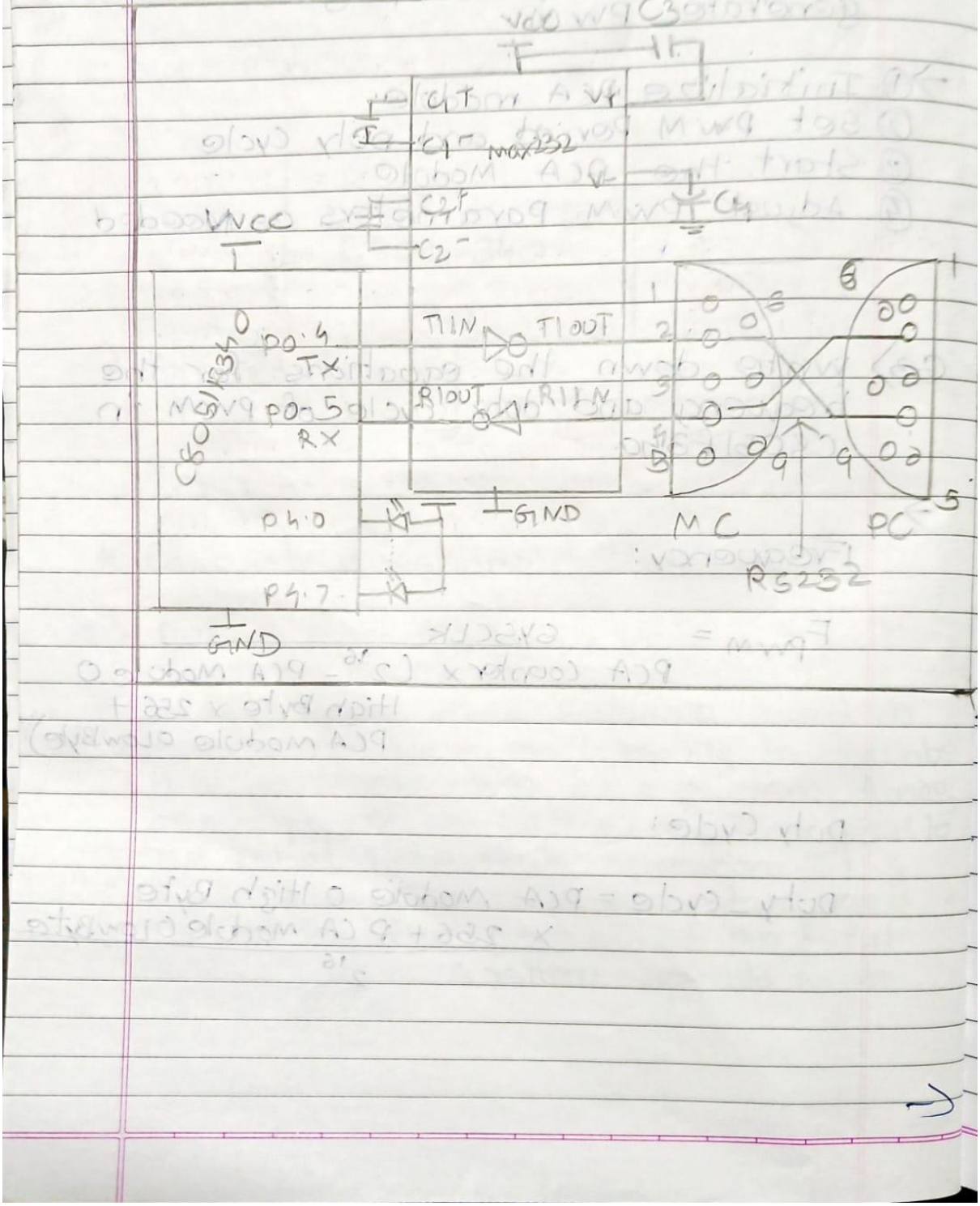
Study Question:

1. Explain the need of MAX232 in serial communication.
2. Write the Port Properties for setting Hyper Terminal connection.
3. Explain UART registers.

PAGE No.
DATE / /

Exp 8 UART

* Interfacing diagram.



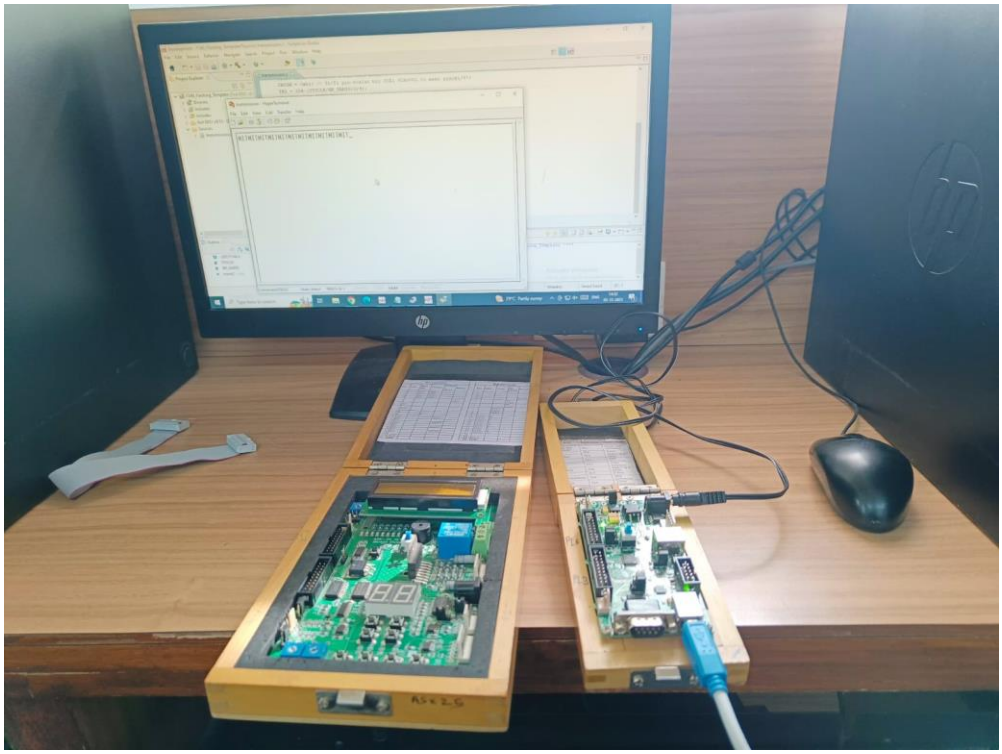
Tranmission program for Uart with C8051F340

```
// Exp - 8 Implement UART with C8051F340
/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY

Tranmission program:
*/

#include"c8051f340.h"
#define sysclk 12000000
#define BR_UART0 9600

void main()
{
    char ch[]=("SHREERANG");
    int i;
    OSCICN = 0X80;
    XBR0=0X01;
    XBR1=0X40;
    P0MDOUT=0X10;
    SCON0=0X00;
    CKCON=0X01;
    TH1=256-(sysclk/BR_UART0/2/4);
    TH1=TL1;
    TMOD=0x20;
    TR1=1;
    while(1)
    {
        for(i=0;ch[i]!='\0';i++)
        {
            SBUF0=ch[i];
            while(TI0==0);
        }
    }
}
```

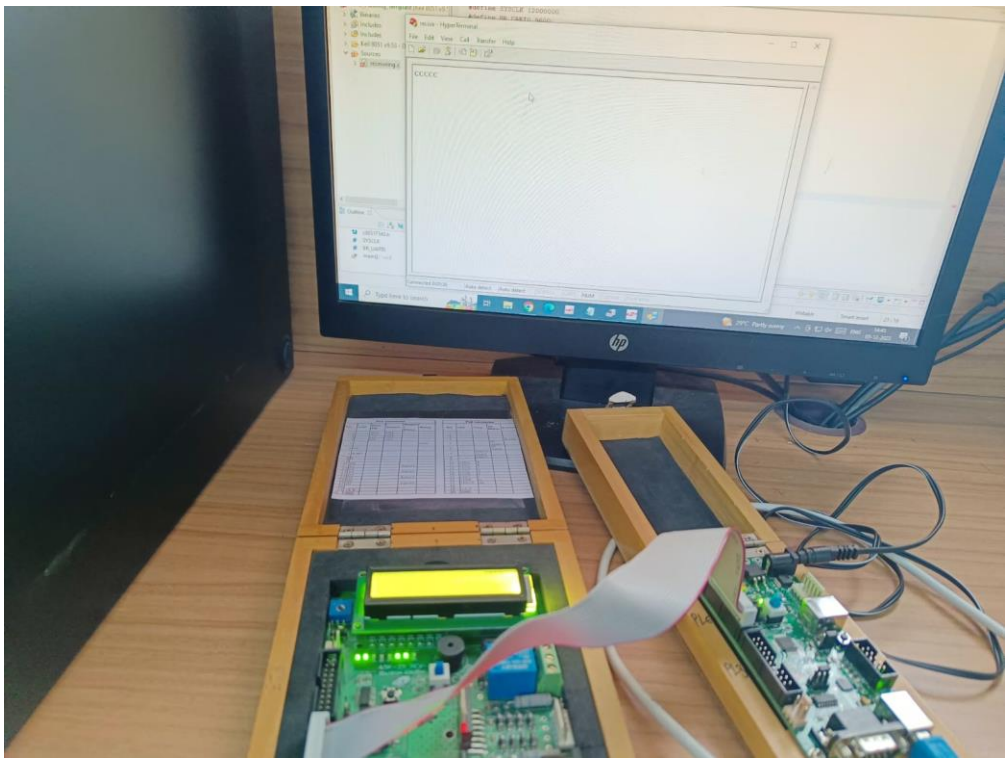
Reciving program for Uart with C8051F340

```
// Exp - 8 Implement UART with C8051F340
/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY

Reciving Program:

*/

#include"c8051f340.h"
#define sysclk 12000000
#define BR_UART0 9600
sbit BUZZER=P3^3;

void main()
{
    OSCICN = 0X83;
    XBR0=0X01;
    XBR1=0X40;
    P3MDOUT=0X08;
    BUZZER=1;
    P0MDIN=0X20;
    P4MDOUT=0xFF;
    SCON0=0X10;
    CKCON=0X01;
    TH1=256-(sysclk/BR_UART0/2/4);
    TH1=TL1;
    TMOD=0x20;
    TR1=1;
    while(RI0==0)
    {
        P4=~SBUF0;
        RI0=0;
    }
}
```

Transmission Output:



Receiving Output:



PAGE No.
DATE / /

Q) Study Question

Q1) Explain the need of MAX232 in serial communication.

→ The MAX232 is a crucial component in serial communication systems, serving as a bridge between devices operating at different voltage levels and signal polarities. In older RS-232 communication, signals are represented by voltage levels ranging from -15V to +15V with inverted logic, while modern microcontrollers typically operate at lower voltage levels and use non-inverted logic. The MAX232 addresses this discrepency by performing voltage level conversion and signal inversion, ensuring seamless communication between devices.

Q2) write the Port properties for setting Hyper Terminal connection.

→ Port Properties for setting Hyper Terminal connection are -

1. Baud Rate
2. Data Bits
3. Parity
4. stop Bits
5. Flow Control.

PAGE No.
DATE / /

Q3) Explain UART registers.

→ UART (Universal Asynchronous Receiver/Transmitter) registers are hardware registers in a microcontroller or communication module responsible for controlling and managing serial communication. These registers include configuration setting such as baud rate, data bits, parity and stop bits. Additionally, there are status registers that provide information about the current state of the UART, including flags for transmit and receive buffers, errors and interrupts.

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

T. Y. B. Tech (Electrical and Computer Engineering)

Trimester: V **Subject: Microcontroller and Applications**
Name: Shreerang Mhatre **Class: TY**
Roll No: 52 **Batch: A3**

Experiment No: 09

Name of the Experiment: Interfacing of Stepper motor with C8051F340.

Performed on: 31/10/2023

Submitted on: 07/11/2023

Marks	Teacher's Signature with date

Aim: Write C program for interfacing of Stepper motor with C8051F340 to rotate in clockwise and anticlockwise direction.

Apparatus: EPBF340 board, Stepper motor Board

Theory:

A stepper motor is known by its important property to convert a train of pulses into a precisely defined increment in the shaft position. A stepper motor is a widely used a device that translates electrical pulses into mechanical movement. In applications such as disk drivers, dot matrix printers, and robotics, the stepper motor is used for position control. Every stepper motor has a permanent magnet rotor (also called as shaft) surrounded by a stator the most common stator motors have four stator windings that are paired with a centre-tapped common as shown in the figure. This type of stator motor is commonly referred to as four phase stepper motor. The centre tap allows a change of current direction in each of two coils, when a winding is grounded, thereby resulting in polarity change of the stator. Notice that while a conventional motor shaft runs freely, the stepper motor shaft moves in a fixed repeatable increment which allows one to move it to a precise position.

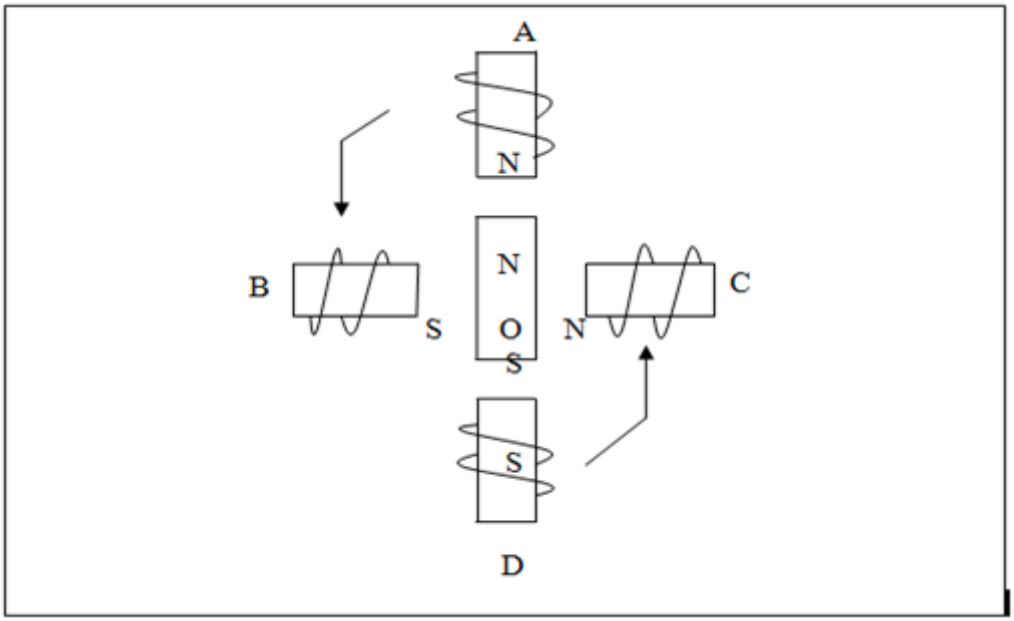


Figure 8.1 Stepper motor windings

This repeatable fixed movement is possible as a result of basic magnetic theory where poles of same polarity repel and opposite attract. The direction of rotation is indicated by the stator poles. The stator poles are determined by the current sent through the wire coils A, B,C and D as shown in Figure 8.1. As the direction of current is changed, the polarity is also changed causing the reverse motion of the rotor. The stepper motor discussed here has a total of six leads: four leads representing the four stator windings and two commons for the centre tapped leads. As the sequence of power is applied to each stator winding, the rotor will rotate. There are several widely used sequences where each has a different degree of precision. The stepping sequence of excitations is as shown in Table 8.1.

Table 8.1 The stepping sequence

Windings	**D**		**C**		**B**		**A**	
Sequence in hex on Port 4	**P4.7**	**P4.6**	**P4.5**	**P4.4**	**P4.3**	**P4.2**	**P4.1**	**P4.0**
0A	0	0	0	0	1	0	1	0
88	1	0	0	0	1	0	0	0
A0	1	0	1	0	0	0	0	0
22	0	0	1	0	0	0	1	0

Types of Stepper Motor:

There are three main types of stepper motors, they are:

1. Permanent magnet stepper
2. Hybrid synchronous stepper
3. Variable reluctance stepper

Permanent Magnet Stepper Motor: Permanent magnet motors use a permanent magnet (PM) in the rotor and operate on the attraction or repulsion between the rotor PM and the stator electromagnets.

Variable Reluctance Stepper Motor: Variable reluctance (VR) motors have a plain iron rotor and operate based on the principle that minimum reluctance occurs with minimum gap, hence the rotor points are attracted toward the stator magnet poles.

Hybrid Synchronous Stepper Motor: Hybrid stepper motors are named because they use a combination of permanent magnet (PM) and variable reluctance (VR) techniques to achieve maximum power in a small package size.

Advantages of Stepper Motor:

1. The rotation angle of the motor is proportional to the input pulse.
2. The motor has full torque at standstill.
3. Precise positioning and repeatability of movement since good stepper motors have an accuracy of 3 – 5% of a step and this error is non-cumulative from one step to the next.
4. Excellent response to starting, stopping and reversing.
5. 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.
6. The motors response to digital input pulses provides open-loop control, making the motor simpler and less costly to control.
7. It is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft.
8. A wide range of rotational speeds can be realized as the speed is proportional to the frequency of the input pulses.

Applications:

1. **Industrial Machines** – Stepper motors are used in automotive gauges and machine tooling automated production equipment.
2. **Security** – new surveillance products for the security industry.
3. **Medical** – Stepper motors are used inside medical scanners, samplers, and also found inside digital dental photography, fluid pumps, respirators and blood analysis machinery.

4. **Consumer Electronics** – Stepper motors in cameras for automatic digital camera focus and zoom functions.

Interfacing Diagram:

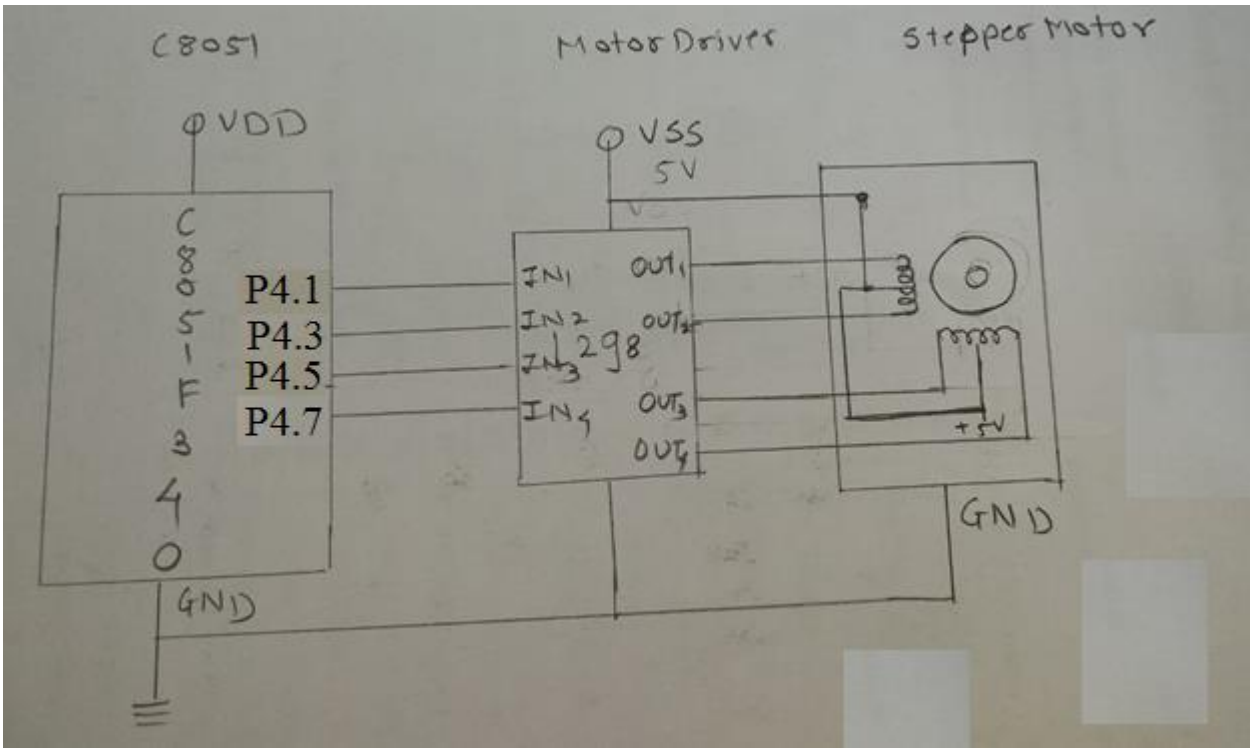


Figure 8.2 Interfacing Diagram of stepper motor with C8051F340

Hardware Connections:

Connect flat cable between PL3 connector of ASK25 and PL6 connector of EPBF340 board. Connect stepper motor at PL4 connector of ASK25. Connect 9V power supply at PL9 connector of ASK25 before connecting the stepper motor. Press DIP switch SW9 on ASK25 to run the motor.

Pin Connection	PL3 Connector of ASK25	PL6 Connector of EPBF340
11	Input 1 (A)	P4.1
13	Input 2 (B)	P4.3
15	Input 3 (C)	P4.5
17	Input 4 (D)	P4.7
19	5V	5.0 V
20	GROUND	GND

Program: Attach printout of the tested code.

Expected Result:

Stepper Motor will rotate in the clockwise and anticlockwise direction

Conclusion:

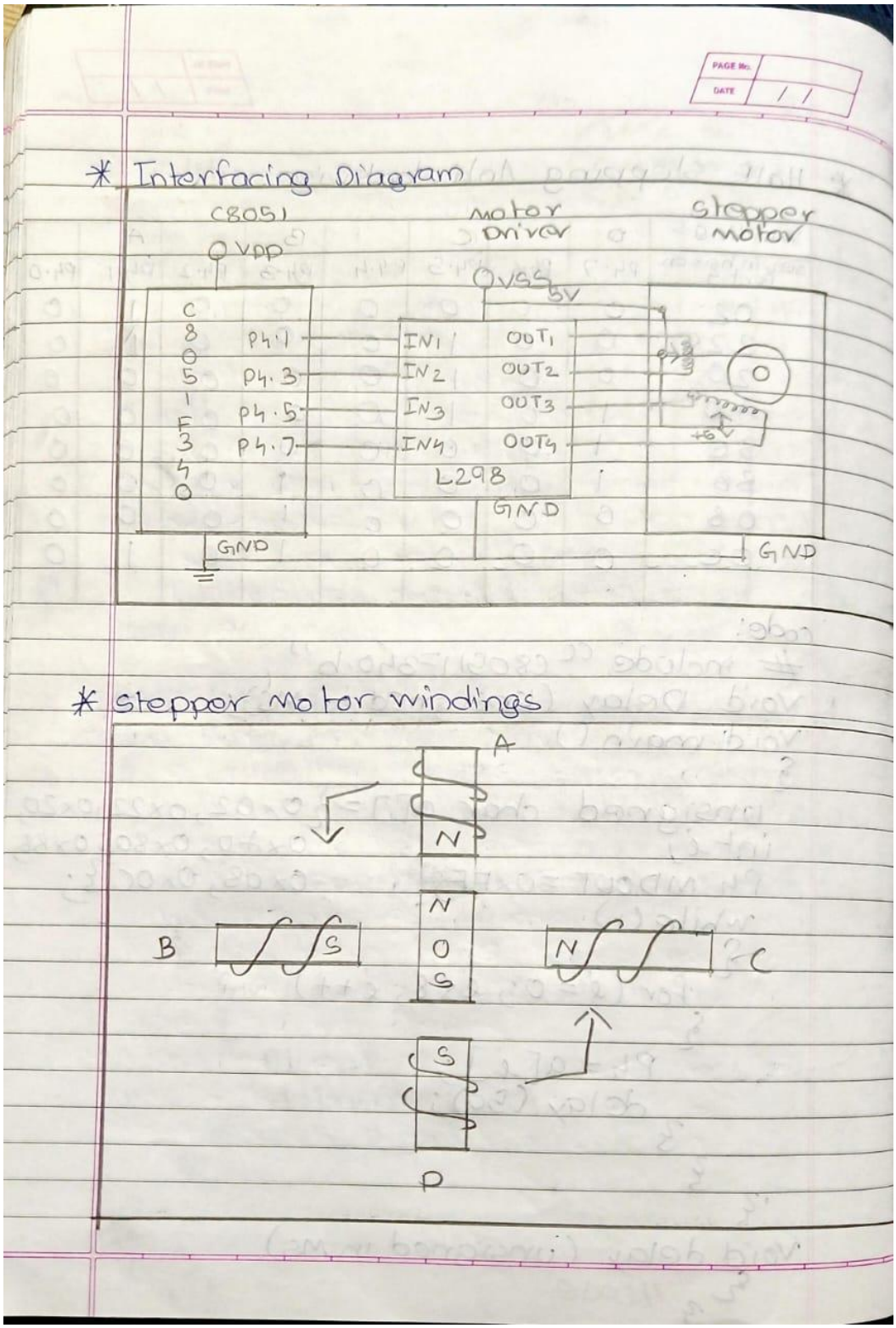
Study Questions:

1. Explain different types of stepper motors and its applications.
2. Explain full stepping & half stepping concept of stepper motor.
3. List specifications of stepper motor and explain need of diver.

Additional Links:

https://nptel.ac.in/courses/112103174/16

Interfacing of Stepper motor with C8051F340.



Code for Full Stepping in Clockwise Direction:

```
// Exp 9 Interfacing of Stepper motor with C8051F340
// Full Stepping in Clockwise Direction

/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY
*/

#include "C8051F340.h"
void delay(unsigned int Ms);
void main()
{
    char a[4]={0x02, 0x20, 0x80, 0x08};
    int i;
    P4MDOUT=0xFF;
    while(1)
    {
        for(i=0; i<4;i++)
        {
            P4=a[i];
            delay(60);
        }
    }
}
void delay(unsigned int Ms)
{
    unsigned int n;
    unsigned int j;
    for(n=0;n<Ms;n++)
    {
        for(j=0;j<65;j++);
    }
}
```

Code for Full Stepping in AntiClockwise Direction:

```
// Exp 9 Interfacing of Stepper motor with C8051F340
// Full Stepping in AntiClockwise Direction

/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY
*/

#include "C8051F340.h"
void delay(unsigned int Ms);
void main()
{
    char a[4]={0x02,0x08, 0x80, 0x20};
    int i;
    P4MDOUT=0xFF;
    while(1)
    {
        for(i=0; i<4;i++)
        {
            P4=a[i];
            delay(60);
        }
    }
}
void delay(unsigned int Ms)
{
    unsigned int n;
    unsigned int j;
    for(n=0;n<Ms;n++)
    {
        for(j=0;j<65;j++);
    }
}
```

Code for Half Stepping in Clockwise Direction:

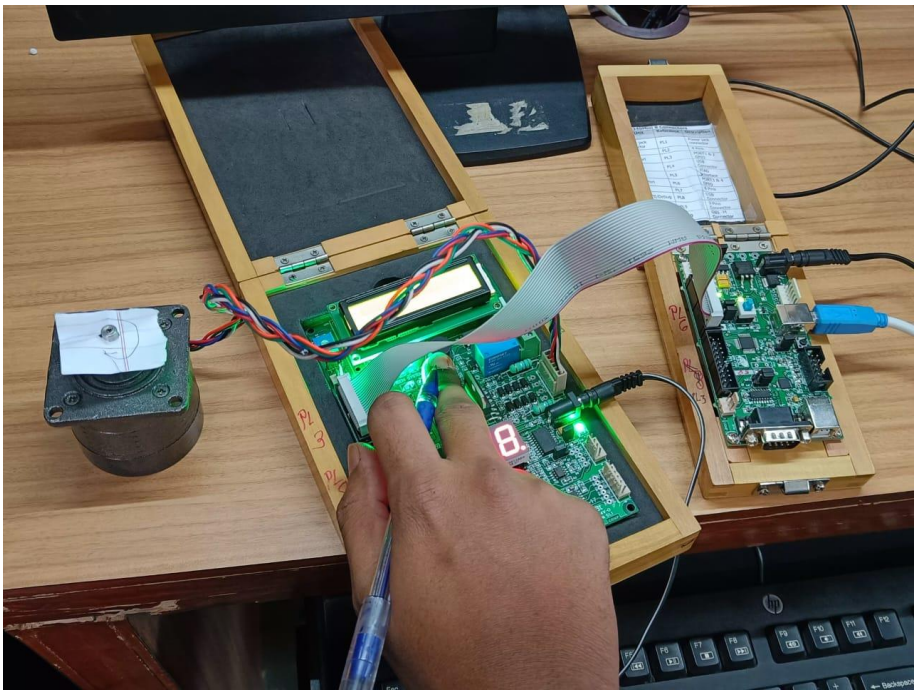
```
// Exp 9 Interfacing of Stepper motor with C8051F340
// Half Stepping in Clockwise Direction

/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY
*/

#include "c8051F340.h"
void delay(unsigned int Ms);
void main()
{
    char a[]={0x02, 0x22, 0x20, 0xA0,0x80,0x80,0x88,0x0C};
    int i;
    P4MDOUT=0xFF;
    while(1)
    {
        for(i=0; i<8;i++)
        {
            P4=a[i];
            delay(60);
        }
    }
}
void delay(unsigned int Ms)
{
    unsigned int n;
    unsigned int j;
    for(n=0;n<Ms;n++)
    {
        for(j=0;j<65;j++);
    }
}
```

Code for Half Stepping in AntiClockwise Direction:

```
// Exp 9 Interfacing of Stepper motor with C8051F340
// Half Stepping in AntiClockwise Direction

/*
Name: Shreerang Mhatre
Rollno: 52
Batch: A3
Class: TY
*/

#include "c8051F340.h"
void delay(unsigned int Ms);
void main()
{
    char a[]={0x02, 0x0A, 0x08, 0x88,0x80,0xA0,0x20,0x22};
    int i;
    P4MDOUT=0xFF;
    while(1)
    {
        for(i=0; i<8;i++)
        {
            P4=a[i];
            delay(60);
        }
    }
}
void delay(unsigned int Ms)
{
    unsigned int n;
    unsigned int j;
    for(n=0;n<Ms;n++)
    {
        for(j=0;j<65;j++);
    }
}
```

Demonstration of Full Stepping and Half Stepping in Clockwise Direction

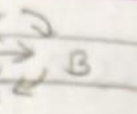
Demonstration of Full Stepping and Half Stepping in AntiClockwise Direction



Exp 9 Stepper Motor

Page No.
Date 31/10/23

* Full stepping Clockwise



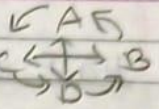
windings sequence in hex on Port 4	D		C		B		A	
	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
02	0	0	0	0	0	0	1	0
08	0	0	0	0	1	0	0	0
80	1	0	0	0	0	0	0	0
20	0	0	1	0	0	0	0	0

Code:

```
#include "C8051F340.h"
void delay (usingned int Ms);
void main ()
{
  unsigned char a[]={0x02,0x08,0x80,0x20};
  int i;
  P4MDOUT = 0xFF;
  while(1)
   {
    for (i=0; i<4; i++)
     {
        P4 = a[i];
        delay (50);
     }
   }
}
void delay (unsigned int Ms)
 {
   //code
 }
```

PAGE No.
DATE / /

* Full stepping Anticlockwise.



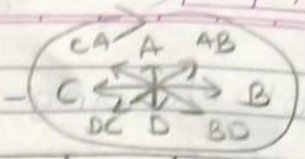
windings	D		C		B		A	
sequence in hex on Port 4	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
02	0	0	0	0	0	0	1	0
20	0	0	1	0	0	0	0	0
80	1	0	0	0	0	0	0	0
08	0	0	0	0	1	0	0	0
	0	0	0	0	0	0	0	0

Code:

```
#include "C8051F340.h"
void delay (unsigned int ms);
void main()
{
  unsigned char a[] = {0x02, 0x20,
                       0x80, 0x08};
  int i;
  P4MDOUT = 0xFF;
  while(1)
  {
   for (i=0; i<4; i++)
    {
      P4 = a[i];
      delay (60);
    }
  }
}
void delay (unsigned int ms)
{
  // code.
}
```

PAGE No.
DATE / /

* Half stepping clock wise –



windings	D		C		B		A	
seq in hex on port 4	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
02	0	0	0	0	0	0	1	0
0A	0	0	0	0	1	0	1	0
08	0	0	10	0	1	0	0	0
88	1	0	0	0	1	0	0	0
80	1	0	0	0	0	0	0	0
A0	1	0	1	0	0	0	0	0
20	0	0	1	0	0	0	0	0
22	0	0	11	0	0	0	1	0

code:

```
#include "C8051F340.h"
void Delay (unsigned in ms);
void main()
{
  unsigned char a[]={0x02, 0x0A, 0x08,
  int i;                0x88, 0x80, 0xA0,
  P4MDOUT=0xFF;         0x20, 0x22 };
   while(1)
   {
     for (i=0; i<8, i++)
     {
        P4=a[i];
        delay(50);
     }
   }
}
void delay(unsigned in ms)
{ }// code
```

PAGE No.
DATE / /

* Half Stepping Anti clockwise

windings	D		C		B		A	
seq in hex on port 4	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
02	0	0	0	0	0	0	1	0
22	0	0	1	0	0	0	1	0
20	0	0	1	0	0	0	0	0
A0	1	0	1	0	0	0	0	0
80	1	0	0	0	0	0	0	0
88	1	0	0	0	1	0	0	0
08	0	0	0	0	1	0	0	0
0C	0	0	0	0	1	0	1	0

code:

```
# include "C8051F340.h"
void Delay (unsigned in MS);
void main ()
{
  unsigned char a[] = {0x02, 0x22, 0x20,
  int i;                0xA0, 0x80, 0x88,
  P4 MDOUT = 0xFF;      0x08, 0x0C};
  while (1)
  {
    for (i = 0; i < 8, i++)
    {
      P4 = a[i];
      delay (50);
    }
  }
}
void delay (unsigned in MS)
{
  //code
}
```

PAGE No.
DATE / /

* Postlab Questions -

Q1) Explain different types of stepper motors and its applications.

-> ① Permanent Magnet Stepper Motor (PM) -
These motors are relatively simple and cost-effective but may have lower torque & precision compared to other types.
Used in applications like printers, plotters & low-precision positioning sys.

② Variable Reluctance Stepper Motor (VR) -
They offer a moderate level of torque & precision. However, they can be noisy and are less common than other types.
Used in applications like textile machinery, disk drivers & some CNC machines.

③ Hybrid Stepper Motor
Provide a balance between cost, precision, and torque. They are the most common type of stepper motor and offer good all-around performance.
Used in various applications, including 3D printers, CNC machines, robotics & camera lens focusing systems

Q2) Explain full stepping & half stepping concept of stepper motor.

→ ① Full stepping -
- In full stepping, each coil or phase of the stepper motor is energized to its full extent one at a time.
- Each step in a full-stepping sequence results in a 90-degree rotation of the motor shaft in a four-phase motor. In a two-phase motor, each step results in a 180-degree rotation.
- Full stepping provides more torque but a coarser resolution.

② Half stepping -
- In half stepping provides a finer resolu and smoother as the rotor can be positioned at an intermediate angle between the full steps.
- In half stepping, a combination of two adjacent phases is energized simultanea This results in more steps per rotation & smaller step angles, typically half the size of a full step
- a half-stepping sequence would involve the following activations AB, A, BC, C, CD, D, DA, B & so on.

PAGE No.
DATE / /

Q3) List specifications of stepper motor & explain need of driver.

→ Specifications of stepper motor are -

1. Step Angle
2. Holding Torque
3. Rated Current
4. Resistance and Inductance
5. Rotor Inertia
6. Physical size
7. operating voltage
8. shaft & configuration.

A stepper motor driver is essential for controlling the precise motion and current regulation of a stepper motor. It provides the necessary electrical signals to the motor, enabling accurate positioning, speed control, and microstepping for applications that require high precision and reliability. Additionally, stepper motor drivers offer protection features and interface compatibility, making them crucial components for integrating stepper motors.

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

T. Y. B. Tech (Electrical and Computer Engineering)

Trimester: V **Subject: Microcontroller and Applications**
Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 10

Name of the Experiment: Interfacing EEPROM using SPI with C8051F340

Performed on: 05/12/2023

Submitted on: 07/12/2023

Marks	Teacher's Signature with date

Aim: Write a program to write and read data on SPI based EEPROM.

Apparatus: EPBF340 board, ASK25 board, Connectors

Theory:

The Enhanced Serial Peripheral Interface (SPI0) of C8051F340 provides access to a flexible, full-duplex synchronous serial bus. SPI0 can operate as a master or slave device in both 3-wire or 4-wire modes, and supports multiple masters and slaves on a single SPI bus.

SPI0 four wire Master Mode Operation is used to read and write data on EEPROM. The four signals used by SPI0 are MOSI, MISO, SCK, NSS.

SPI0CFG, SPI0DAT, and SPI0CN SFRs of SPI are configured to select 4 wire master mode.

SPI clock frequency is configured using following formula.

SPI0CKR = (SYSCLK/(2*SPI_CLOCK))-1

25AA160 16 Kbit Serial Electrically Erasable PROMs is used. The memory is accessed via a simple Serial Peripheral Interface compatible serial bus. The bus signals required are a clock input (SCK) plus separate data in (SI) and data out (SO) lines. Access to the device is controlled through a Chip Select (CS) input.

Data written data on EEPROM and read from EEPROM is displayed on LCD.

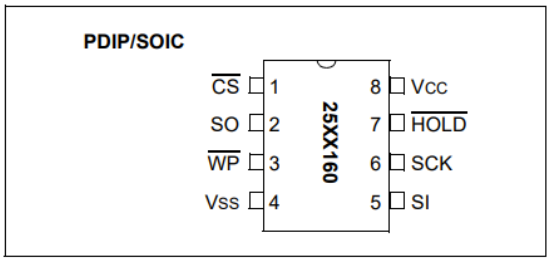


Figure 9.1 EEPROM 25AA160

Interfacing Diagram:

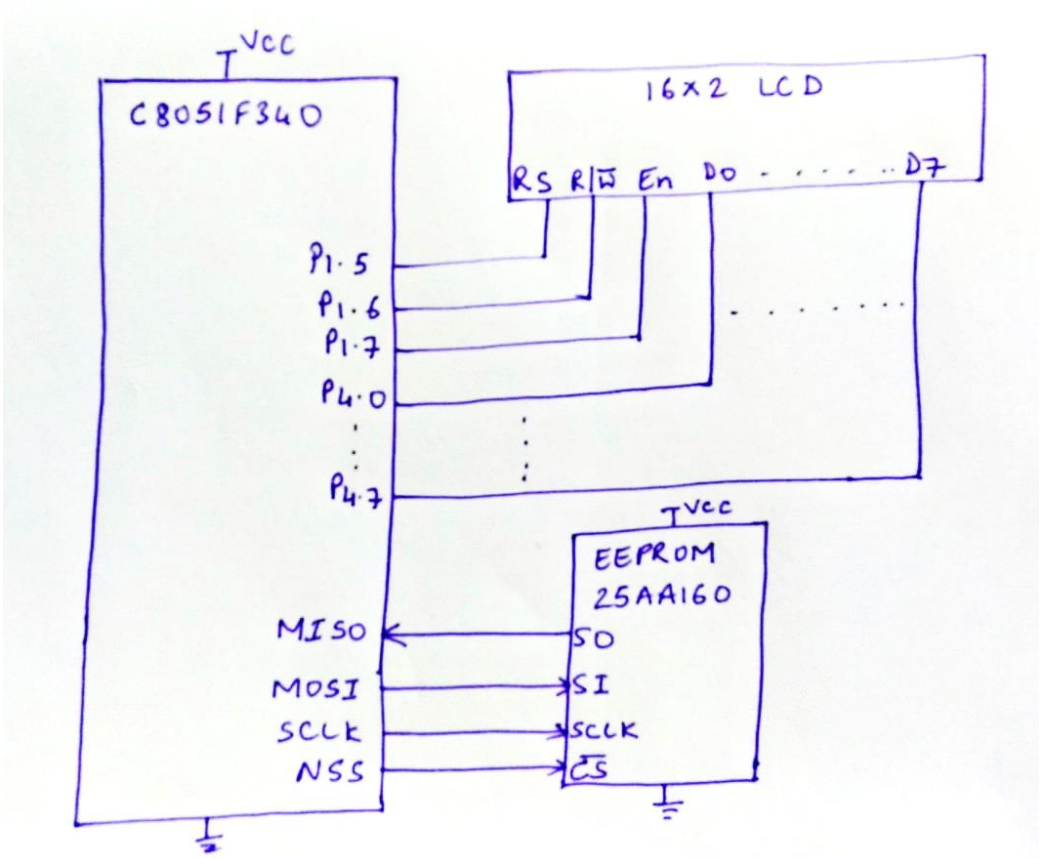


Figure 9.2 Interfacing Diagram EEPROM using SPI

Hardware Connections: Connect FRC cable between PL7 connector (SPI) of EPBF340 board and PL1 of ASK 25.
Connect flat cable between PL3 connector of ASK25 and PL3 connector of EPBF340 board.

Program: Attach printout of the tested code.

Expected Result:

EEPROM data should be displayed on LCD.

Conclusion:

Study Questions:

1. Explain SFRs of SPI0 in detail.
2. Compare SPI with I2C

Additional Links:

https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html

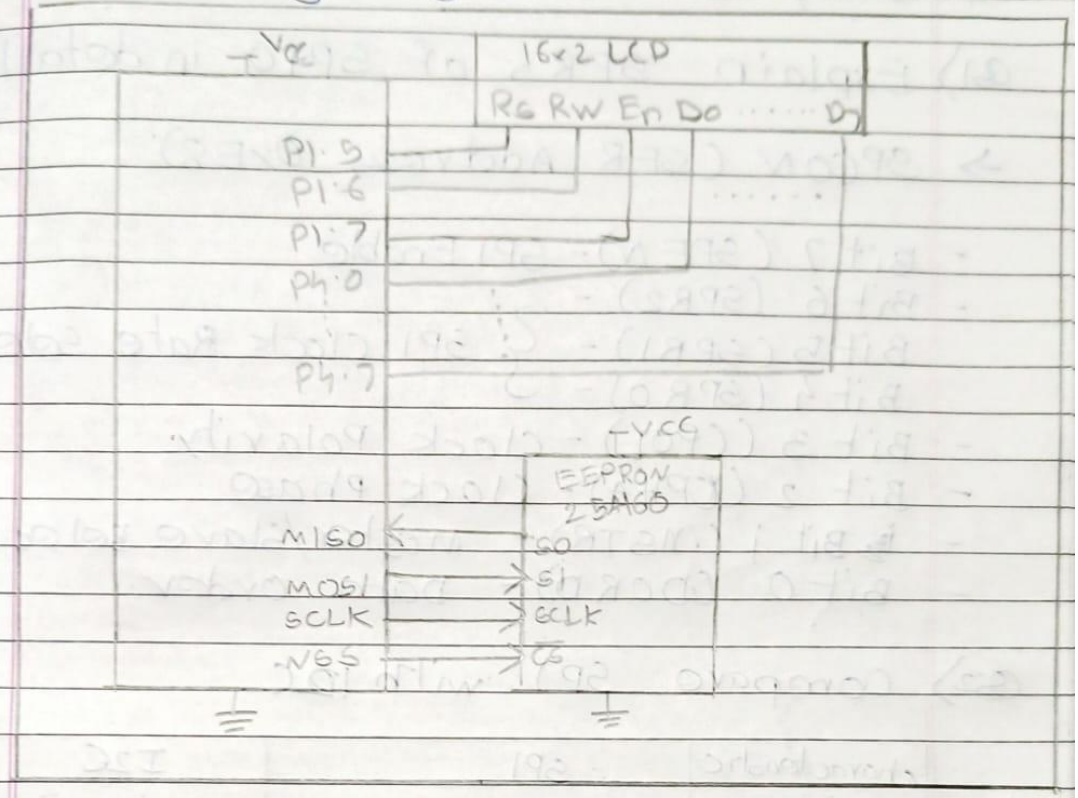
https://microcontrollerslab.com/introduction-to-spi-communication-protocol/

http://ww1.microchip.com/downloads/en/devicedoc/21231d.pdf

Exp 10 EEPROM

PAGE No.
DATE / /

* Interfacing Diagram:



PAGE No.
DATE / /

* study Q

Q1) Explain SFRs of SPI0 in detail.

→ SPCON (SFR Address 0xE8)

- Bit 7 (SPEN) - SPI Enable
- Bit 6 (SPR2) -
  Bit 5 (SPR1) - } SPI clock Rate select
  Bit 4 (SPR0) -
- Bit 3 (CPOL) - clock Polarity.
- Bit 2 (CPHA) - Clock Phase
- Bit 1 (MSTR) - Master/Slave select
- Bit 0 (DORD) - Data order.

Q2) Compare SPI with I2C.

characteristic	SPI	I2C
Topology	Master - Slave	Master-Slave
No. of wires	Requires separate lines for data (MOSI/MISO), clock (SCK) & chip select	Requires only two wires: data (SDA) & clock (SCL)
Data Transfer	Full - duplex	Half - duplex.
Data Rate	Generally supports higher data rates	Typically operates at lower data rates
Addressing	chip select lines	Unique 7 or 10 bit