Visvesvaraya National Institute
of
Technology, Nagpur

Embedded Systems (ECP403)

Lab Report

Submitted By :
Shruti Murarka (BT18ECE099)
Semester 5
Electronics and Communication Engineering Dept.

Submitted To :
Dr. Ankit A Bhurane
Course Instructor

Contents

1 8051 Micro-controller **2**

1 Experiment-1: Running LED, Switches + LED 3

2 Experiment-2: Interface DAC to display a Sine Wave 6

3 Experiment-3: Interfacing 4x3 keypad 9

4 Experiment-4: Interfacing 2x16 LCD 13

5 Experiment-5: Interfacing motor 17

6 Experiment-6: Generating Square wave of 2 KHz & it's verification. . 20

7 Experiment-7: Serial Transmission (UART) 23

8 Experiment-8: I^{2}C Communication 29

9 Experiment-9: SPI Communication 33

Chapter 1

8051 Micro-controller

Experiment-1: Running LED, Switches + LED

Aim : a.To blink LED's using 8051 & EdSim51.
b. Connect switches to LED's using 8051 & EdSim51.

Problem Statement:
a. Develop the logic & code(assembly level & C) to display to & fro pattern on the LED's .
b. Write the assembly level & C code to connect switches to LED's.

Code :

Assembly Code

```
a.
;to & fro pattern on the LED's.

; Logic 0 on a port 1 pin turns on the LED and vice versa.

MOV A, #FEH

HERE:
    MOV P1, A
;MOVE A TO PORT 1  i.e. 0th LED

    RL A
;ROTATE CONTENT OF A
;LEFT BY ONE BIT

    CJNE A, #01111111B, HERE
;check if a == 7f i.e.7th led
;continue here

THERE:
    MOV P1, A
    RR A     ;ROTATE RIGHT
    CJNE A, #1111110B, THERE
;check if a == FE i.e.0th led
;continue There
JMP HERE

b.
;connection of switches to LED
;switches are connected to port 2 and LED's to port 1

```

C Code

```
a.
//to & fro pattern on the LED's
#include<reg51.h>

//Hex values for LED Bank
unsigned char LED[] = ... {0xFE,0xFD,0xFB,0xF7,
    0xEF,0xDF,0xBF,0x7F};

unsigned char k = 0;//variable
void main()
{
    while(1) //infinite loop
    {
        for(k = 0; k<8 ;k++)
        {
//Blink in forward manner
            P1 = LED[k];
        }

        for(k=7 ; k≥0 ;k−−)
        {
//Blink in backward manner
            P1 = LED[k];
        }
    }
}

b.
//Switches+LED's
#include<reg51.h>

```

```
; closed switch represents logic 0 which is then copied to port 1

HERE:
      MOV P1, P2
      JMP HERE
```

```
void main()
{
//switches are connected to port 2 and LED's to port 1
     while(1)
     P1=P2;
}
```

Output: For developing & compiling the assembly output, EdSim simulator is used & MCU 8051 IDE for C code.

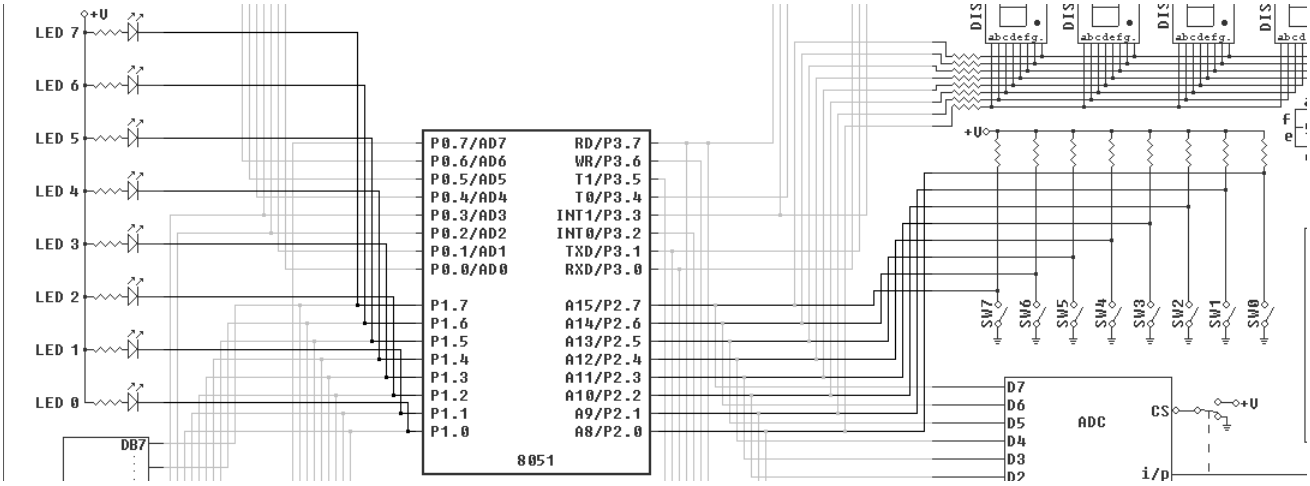
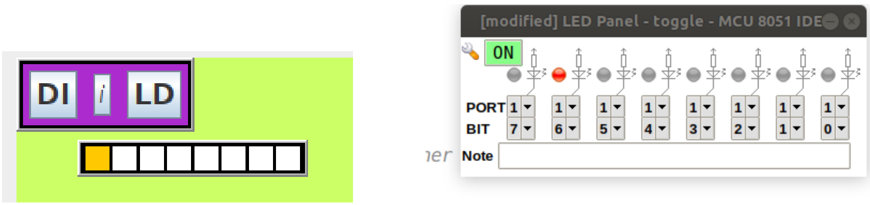


Figure 1.1: Figure showing the **Circuit Diagram of interfacing LED's & switches with 8051.**

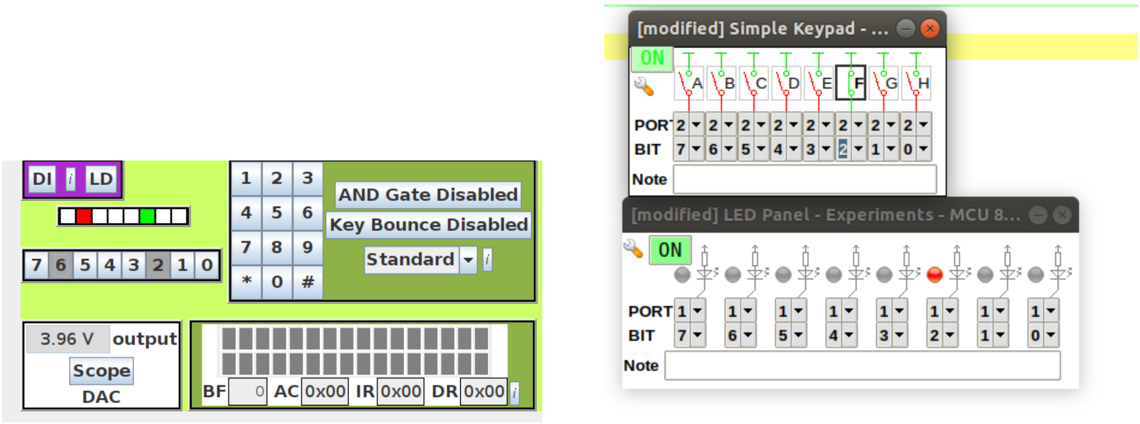


(a)

(b)

Figure 1.2: Figure showing **output of Blinking LED's.**

<u>Discussion & Observations</u> : The LED's are considered active low i.e. high when given logic 0 & off when given logic 1. Hence port connected with LED bank is given compliment of LEDs to be glown. Similarly switches are also active low. Hence the values when a switch is pressed is directly moved to port connected with LED bank.



(a) (b)

Figure 1.3: Figure showing **output of Switches with LED's.**

Conclusion : We developed assembly language code and C code for interfacing LED bank & switches with 8051. The circuit diagram connecting 8085 with LED bank & switches is shown in figure 1.1.
We observed to & fro pattern on the LED's. The output for blinking LED's is obtained as shown in figure 1.2.
Later, switches are connected to LED's. The output is obtained as shown in figure 1.3. 0

Experiment-2: Interface DAC to display a Sine Wave

Aim : Display a Sine wave using Digital to Analog Converter (DAC).

Problem Statement : Write a assembly language code & C code for microprocessor 8051 to display sine wave.

Code :

Assembly Code

```
;SINE WAVE

CLR P0.7     ; ENABLE DAC

CLR A        ;CLEAR ACC
;SET DATA POINTER TO LOOKUP
MOV DPTR, #LOOKUP
BACK: MOV B, #00H

;DATA FROM ACC+DATA POINTER IS STORED IN ACC
;DATA IS SENT TO DAC i.e. PORT 1
;THEN B IS INCREMENTED
;B IS MOVED TO A
; i.e. A IS INCREASED

LOOP: MOVC A, @A+DPTR
      MOV P1,A
      INC B
      MOV A,B
;IF CONTENT OF A!=40 CONTINUE THE LOOP
      CJNE A,#40, LOOP
;IF ACC==40 JUMP TO BACK
;I.E. AGAIN START THE SINE WAVE
       JMP BACK

; DATA SET OF SIN FUNCTION SCALED FROM 0 TO 255

LOOKUP:
DB 255, 253, 249, 241, 231
DB 218, 202, 185, 167, 147
DB 128, 108, 88, 70, 53
DB 37, 24, 14, 6, 2, 0
DB 2, 6, 14, 24, 37, 53
```

C Code

```c
//Sine wave without using array
#include<reg51.h>
#include<math.h>

void main()
{
    while(1)
    {
    int i;
    double r;
//Producing Sine Wave
    for (i=0; i≤360; i++)
    {r = sin(2*3.1415*i/360);}

    }
}
```

```
DB 70, 88, 108, 127
DB 147, 167, 185, 202, 218
DB 231, 241, 249, 253
```

Output: For developing & compiling the assembly output, EdSim simulator is used & Keil logic analyzer for C code.

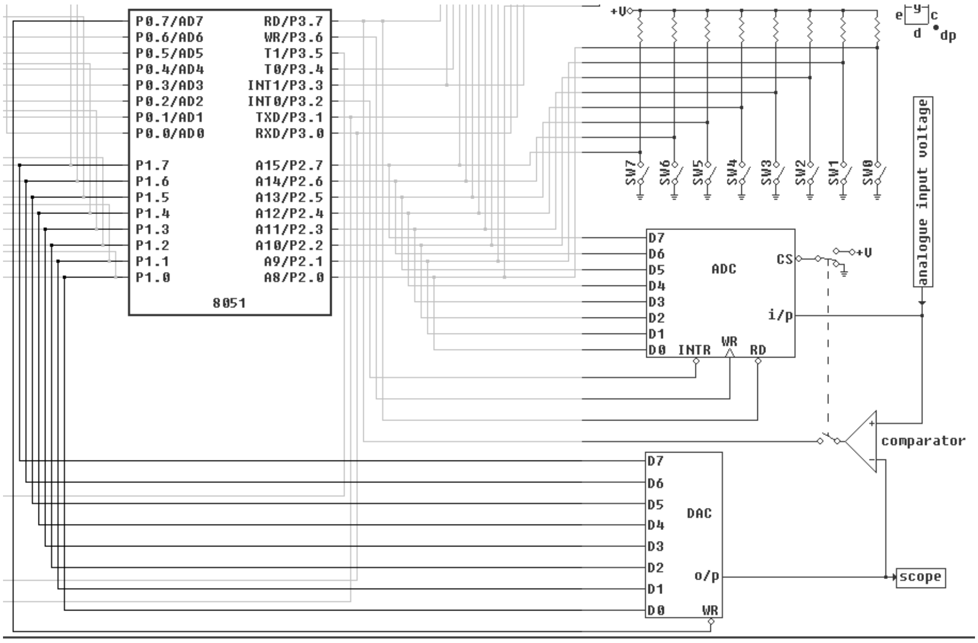
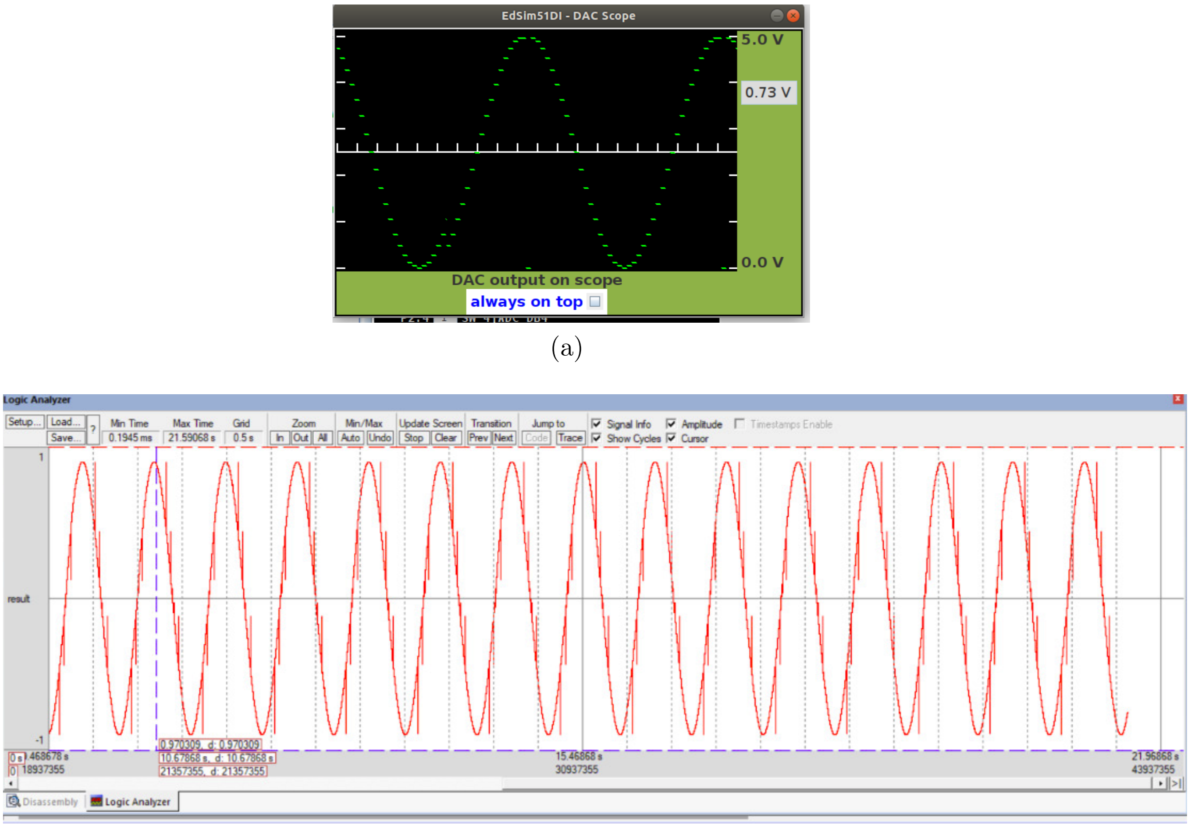


Figure 1.4: Figure showing **circuit diagram interfacing DAC with 8051.**

Discussion & Observations : An array is declared in assembly language containing samples of sine wave to be displayed while in C code samples are generated. Loop is run to access values of array and is moved to port connected with DAC. The program runs infinitely & hence we get continuous sine wave on DAC.
Also, more the samples smoother the curve.

Conclusion : We developed assembly language code & C code to display Sine Wave using DAC. The circuit diagram connecting 8051 with DAC is shown in figure 1.4.
The output for Sine Wave is obtained as shown in figure 1.5.



Min Time 0.1945 ms

Max Time 21.59068 s

Grid 0.5 s

0 s 1.468678 s

0 18937355

0.970309, d: 0.970309

10.67868 s, d: 10.67868 s

21357355, d: 21357355

15.46868 s

30937355

21.96868 s

43937355

(a)

(b)

Figure 1.5: Figure showing **Sine wave generated on DAC.**

Experiment-3: Interfacing 4x3 keypad

Aim : Interfacing 4x3 Keypad with 8051 using EdSim51.

Problem Statement : Write assembly level code & C code to interface keypad with 8051 & display the number on 7-segment display as well as it's binary equivalent on the LED's.

Code :

Assembly Code

```
;KEYPAD

SETB  P3.7   ;INITIALIZE 7-SEG
SETB P3.6
SETB P3.5 ; ENABLE DISPLAY 3

;SET DATA POINTER TO 7-SEGMENT
MOV DPTR, #SEVENSEG

AGAIN:
; CHECKING FOR ROW 0
    ;CLEAR P0.0
    MOV P0, #11111110B
    CLR A ;MOV 0 TO ACC
    ;COLUMN CHECK
    JNB P0.5, DISPLAY

; CHECKING FOR ROW 3
    ;CLEAR P0.3
    MOV P0, #11110111B
    INC A
    ;COLUMN CHECK
    JNB P0.6, DISPLAY
    INC A
    JNB P0.5, DISPLAY
    INC A
    JNB P0.4, DISPLAY

; CHECKING FOR ROW 2
    ;CLEARING P0.2
    MOV P0, #11111011B
    INC A
    ;COLUMN CHECK
    JNB P0.6, DISPLAY
```

C Code

```
#include<reg51.h>
//Array containing values to Display 0-9 in 7-Segment Display
unsigned char Seg[] = ...
   {0xC0,0xF9,0xA4,0xB0,0x99,
    0x92,0x82,0xF8,0x80,0x90};
void display(unsigned char num)
{
//LED binary Equivalent
     P0 = ¬(num);
//7-seg display
    P2 = Seg[num];
    return;
}

void main()
{
 while(1)
 {
 //Initially LED's and 7-segment is OFF.
     P0 = 0xFF;
     P2 = 0xFF;
     P1 = 0xFF;

     P1_0 = 0; //Row 4 check
     //Column check
     if( P1_5 == 0)
         {display(0);//display number
          break;}

     P1_0=P1_1=P1_2=P1_2=1;

    INC A
    JNB P0.5, DISPLAY
    INC A
    JNB P0.4, DISPLAY

;CHECKING FOR ROW 1
    ;CLEARING P0.1
    MOV P0, #11111101B
    INC A
    ;COLUMN CHECK
    JNB P0.6, DISPLAY
    INC A
    JNB P0.5, DISPLAY
    INC A
    JNB P0.4, DISPLAY

;WHEN NO BUTTON IS PRESSED
    MOV P1,#0FFH ;ALL LES's OFF
    MOV P2,#0FFH ;7-SEG OFF
    JMP AGAIN

;DISPLAY BINARY EQUIVALENT ON LED"S
;DISPLAY NO. ON 7-SEG
DISPLAY:
;COMPLIMENT A TO DISPLAY BINARY EQUIVALENT ON LED's
         CPL A
         MOV P1,A
         CPL A
;MOVE DATA FOR 7-SEGMENT IN ACC
         MOVC A,@A+DPTR
         MOV P2,A
         JMP AGAIN

; DATASET OF VALUES TO DISPLAY 0-9 IN 7-SEGMENT DISPLAY

SEVENSEG:
DB 0C0H, 0F9H, 0A4H, 0B0H, 99H
DB 92H, 82H, 0F8H, 80H, 90H
```

```
    P1_3=0; //Row 1 check
    //Column check
    if (P1_6 == 0) //C1 check
        {display(1);
         break;}
    else if(P1_5 == 0) //C2 check
        {display(2);
         break;}
    else if(P1_4 == 0)//C3 check
        {display(3);
         break;}

    P1_0=P1_1=P1_2=P1_2=1;
    P1_2 =0; //Row 2 check
    //Column check
    if (P1_6 == 0)
        {display(4);
         break;}
    else if(P1_5 == 0)
        {display(5);
         break;}
    else if(P1_4 == 0)
        {display(6);
         break;}

    P1_0=P1_1=P1_2=P1_2=1;
    P1_1=0; //Row 3 check
    //Column check
    if (P1_6 == 0)
        {display(7);
         break;}
    else if(P1_5 == 0)
        {display(8);
         break;}
    else if(P1_4 == 0)
        {display(9);
         break;}
  }
}
```

Output : For developing & compiling the assembly output, EdSim simulator is used & MCU 8051 IDE for C code.

Discussion & Observations : The logic to scan numbers of keypad is developed. When no key is pressed the program scans row4, row1, row2, row3 and back to row4

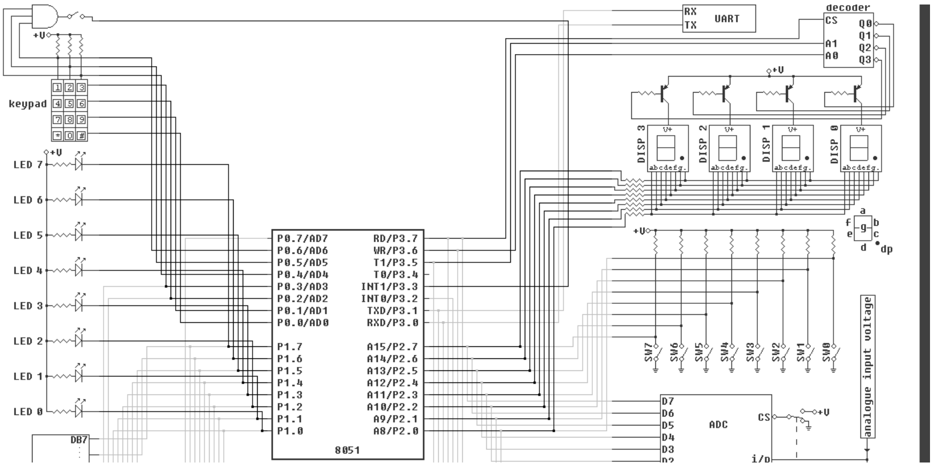
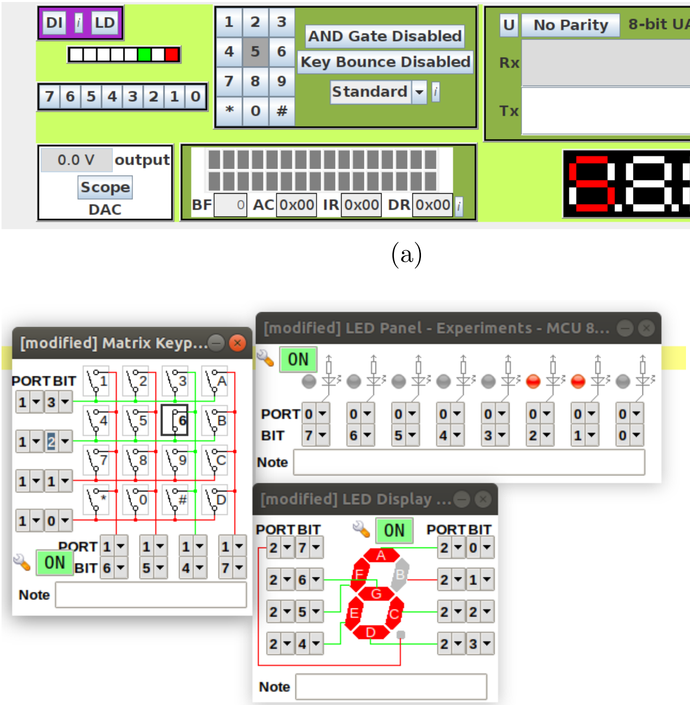


Figure 1.6: Figure showing **circuit diagram interfacing LED bank, 7-segment display & Keypad with 8051.**



AND Gate Disabled

Key Bounce Disabled

No Parity 8-bit UART @ 4800 Bau

BF 0 AC 0x00 IR 0x00 DR 0x00

(a)

(b)

Figure 1.7: Figure showing **output for keypad number by displaying it on 7-segment display & it's binary equivalent on LED bank**

continuously. When any key is pressed on keypad ,LED bank displays its binary equivalent & 7-segment display shows the number pressed. Single row is checked with every column. When any key is pressed the key number is in accumulator & displayed by display function.

Conclusion : We developed assembly language code & C code to interface keypad with 8051 & display the number on 7-segment display & also it's binary equivalent on the LED's. The circuit diagram connecting 8051 with LED bank, 7-segment display & keypad is shown in figure 1.6.
The output for keypad number 5 for asm code & number 6 for C code is shown in figure 1.7.

Experiment-4: Interfacing 2x16 LCD

Aim : Interfacing LCD with 8051 & display name on it using edsim51.

Problem Statement : Write the assembly language code to interface 2x16 LCD with 8051 and display your name on it using 2 line mode.

Code :

Assembly Code

```
;LCD MODULE

;SET DATA POINTER TO NAME STRING
MOV DPTR,#NAME
CLR P0.3;ENABLE LCD COMMAND MODE

;FUNCTION SET-Use 2 lines and 5*7 matrix
MOV P1,#38H
SETB P0.2  ;NEGATIVE EDGE
CALL DELAY
CLR P0.2

;Display ON ,Cursor blinking OFF
MOV P1,#0EH
SETB P0.2 ;NEGATIVE EDGE
CALL DELAY
CLR P0.2

;Increment cursor
MOV P1,#06H
SETB P0.2  ;NEGATIVE EDGE
CALL DELAY
CLR P0.2

;SHRUTI
SETB P0.3  ;DATA MODE

CLR A   ;MOV 0 TO ACC
MOV B, #00H
LOOP:
;LOAD A WITH CHARACTER
        MOVC A, @A+DPTR
        MOV P1, A
```

C Code

```
#include<reg51.h>
#include<string.h>

sbit rs = P3^2;
sbit rw = P3^1;
sbit en = P3^3;

//User defined functions
void LCD_Init();
void LCD_Cmd(unsigned char cmd);
void LCD_string(unsigned char *string);
void LCD_Data(unsigned char data1);
void delay_us(unsigned int t);

void main()
{
//Initialize LCD
   LCD_Init();
//sent name to 1st line of LCD
   LCD_string("SHRUTI ");
// Move the Cursor to Second line First Position
   LCD_Cmd(0xC0);
//sent surname to 2nd line of LCD
   LCD_string("MURARKA ");
//Move the Cursor to First line First position again
   LCD_Cmd(0x80);
   delay_us(2500);
//LCD_Cmd(0x06);//Clear LCD
}
void LCD_Cmd(unsigned char cmd)
```

```
;NEGATIVE EGDE
        SETB P0.2
        CALL DELAY
        CLR P0.2
;INCREMENT B
        INC B
        MOV A, B
;RUN LOOP 6 TIMES
        CJNE A, #6, LOOP

CLR P0.3  ;COMMAND MODE

;Force cursor to the beginning of 2nd line
MOV P1,#0C0H
SETB P0.2   ;NEGATIVE EDGE
CALL DELAY
CLR P0.2

;MURARKA
SETB P0.3  ;DATA MODE
MOV A, #7
MOV B, #7
HERE:
;LOAD A WITH CHARACTERS FROM M
        MOVC A, @A+DPTR
        MOV P1, A
;NEGATIVE EDGE
        SETB P0.2
        CALL DELAY
        CLR P0.2
;INCREMENT B
        INC B
        MOV A, B
;RUN LOOP HERE 7 TIMES
        CJNE A, #14, HERE

JMP LAST

NAME:   ;NAME STRING
DB "SHRUTI MURARKA"

DELAY:  ;DELAY FUNCTION
    MOV R0, #030H
    DJNZ R0, $
    RET
LAST:
    END
```

```
{
    rs = 0; // Command Mode
    rw = 0; //Write mode LCD
    P2 = cmd;// Send the command to LCD
    en= 1;// Negative Edge
    delay_us(1000);
    en= 0;
    return;
}
void LCD_Init()
{
// LCD 2 lines, 5*7 matrix
    LCD_Cmd(0x38);
// Display ON cursor ON
    LCD_Cmd(0x0E);
//Increment Cursor
    LCD_Cmd(0x06);
    return;
}
void LCD_string(unsigned char *string)
{
    int i;
    int j =strlen(string);
//characters sent from string
    for(i=0; i< j; i++)
        {LCD_Data(string[i]);
        delay_us(1000);}
    return;
}
void LCD_Data(unsigned char data1)
{
    rw = 0;//Write mode LCD
    rs = 1;//Data Mode
// Send the data to LCD
    P2 = data1;
    en = 1;// Negative Edge
    delay_us(1000);
    en = 0;
    return;
}
void delay_us(unsigned int t)   //Delay function
{
    while(t!=0)
        t--;
}
```

Output : For developing & compiling the assembly output, EdSim simulator is used & Keil for compiling C code & Proteus for analyzing C Code.

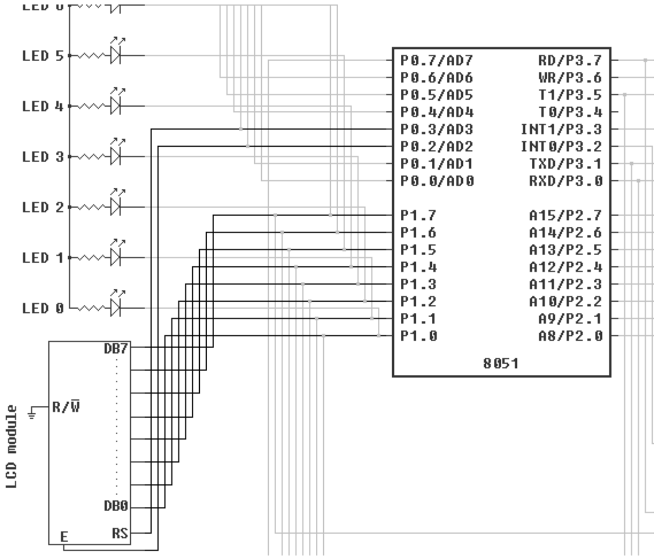
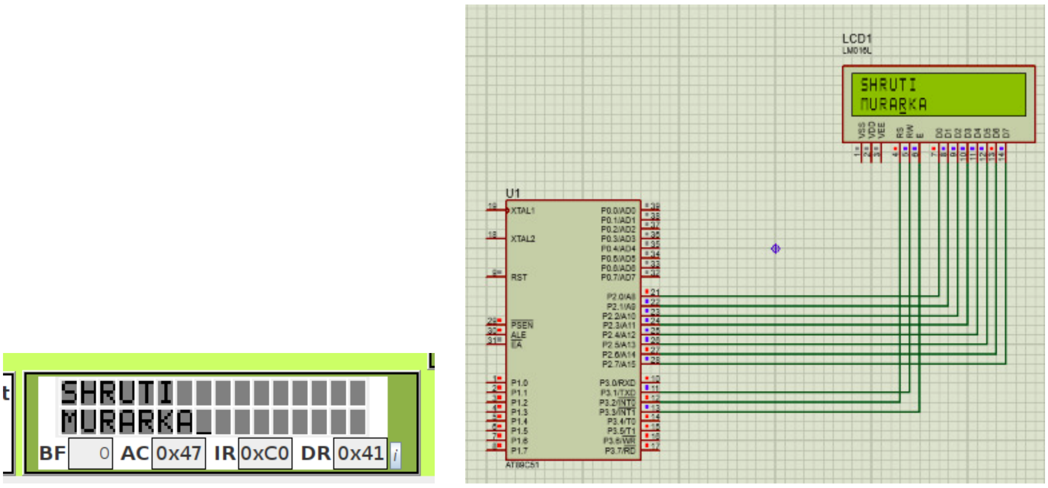


Figure 1.8: Figure showing **circuit diagram interfacing LCD Module with 8051.**

Discussion & Observations : Initially LCD (8-bit Mode) is to be initialized. General pin functions are-
RS : 1-Data register (for write and read)
0-Instruction register(for write), address counter(for read)
R/W: 0- Write
1- Read
E: Starts data read/write.
Afterwards, LCD are sent necessary commands to initialize LCD using LCD data sheet. String to be displayed is stored as string in C & in assembly language, the string to be displayed is stored in database and accessed through DPTR to display. Characters are sent from string to port connected with LCD one after another.

Conclusion : We developed assembly language code & C code to interface LCD Module with 8051 & display name on it using 2-lines. The circuit diagram connecting 8051 with LCD Module is shown in figure 1.8.
The output for LCD Module displaying name is shown in figure 1.9.



SHRUTI
MURARKA_

BF 0 AC 0x47 IR 0xC0 DR 0x41

(a) (b)

Figure 1.9: Figure showing **LCD Module screen**

Experiment-5: Interfacing motor

Aim : To interface the motor with 8051 and count its number of revolutions.

Problem Statement : Write the assembly language code & C code to interface motor with 8051 & display number of revolutions of motor on 7-segment display.

Code :

Assembly Code

```
;MOTOR

;TIMER 1 AS COUNTER IN MODE1
MOV TMOD, #50H
SETB TR1;START TIMER 1

; ENABLE 7-SEGMENT DISPLAY
SETB P3.7
CLR P3.3
CLR P3.4;DISPLAY 0

;SET DATA POINTER TO 7-SEGMENT DISPLAY
MOV DPTR, #SEVENSEG

MAIN:
;FORWARD MOTION MOTOR
      SETB P3.0
      CLR P3.1
;MOVE COUNTER VALUE TO ACC
      MOV A,TL1
      CJNE A, #10, DISPLAY
;IF REVOLTIONS =10
;MOVE TO NEXT INSTRUCTION
; i.e.CLEAR COUNTER
;ELSE DISPLAY NUMBER ON 7-SEG
       ACALL CLEAR

DISPLAY: MOVC A,@A+DPTR
;DISPLAY ON 7-SEGMENT
         MOV P2,A
         JMP MAIN ;BACK TO MAIN

CLEAR: CLR TR1 ;STOP TIMER 1
       MOV TL1,#0 ;CLEAR COUNT
       CLR A ;CLEAR ACC
```

C Code

```
#include<reg51.h>

sbit zero = P3^0;//MOTOR PORTA
sbit one = P3^1; //MOTOR PORTB

void main()
{
    int i;
// Dataset of values to display 0-9 on 7-segment display
unsigned char SevenSeg[] = ...
    {0xC0,0xF9,0xA4,0x99,0xB0,
    0x92,0x82,0xF8,0x80,0x90};
// timer as counter in mode 1
    TMOD = 0x50;
    TR1 = 1; // Start counter

// forward motion of motor
    zero = 0;
    one = 1;

    while(1)
    {
        i = TL1;
        if (i==10)
// reset to zero if count exceeds 9
        {TL1 = 0x00;}
//display on 7-segment
        P1 = SevenSeg[i];
    }
}
```

```
        ;START TIMER AGAIN
        SETB TR1
        RET

; DATASET OF VALUES TO DISPLAY 0-9 ON 7-SEGMENT DISPLAY

SEVENSEG:
DB 0C0H, 0F9H, 0A4H, 0B0H, 99H
DB 92H, 82H, 0F8H, 80H, 90H
```

Output : For developing & compiling the assembly output, EdSim simulator is used & Keil for compiling C code & Proteus for analyzing C Code.

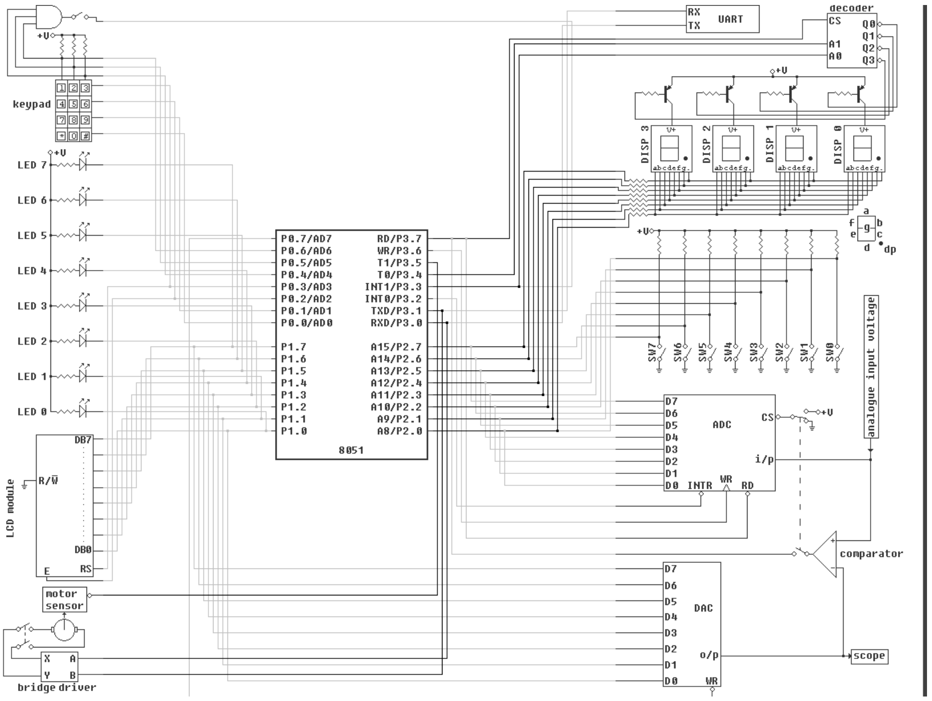
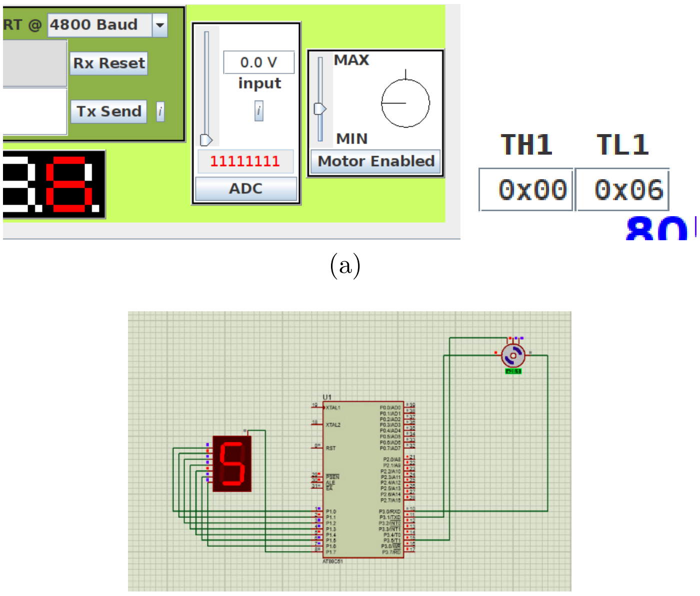


Figure 1.10: Figure showing **circuit diagram interfacing 7-segment display & motor with 8051.**

Discussion & Observations : The motor is rotated in a forward direction i.e. clockwise direction and the number of revolutions is displayed on Display 0 of the 7-segment display. The display shows count from 0 to 9 and then resets. The motor sensor is connected to P3.5, which is the external clock source for timer 1. Therefore, timer 1 is put into counter mode & the timer increments once every motor revolution.



(a)

(b)

Figure 1.11: Figure displaying **No. of revolutions of motor on 7-segment display & counter value .**

According to the timer value, 7-segment is sent data using DPTR/array to display numbers.

Conclusion : We developed assembly language code & C code to interface motor with 8051 & display number of revolutions on 7-segment display The circuit diagram connecting 8051 with motor & 7-segment display is shown in figure 1.10.
The output for 7-segment displaying number of revolutions is shown in figure 1.11.

Experiment-6: Generating Square wave of 2 KHz & it's verification.

Aim : To display square wave of 2kHz frequency and verify it by counting using counters.

Problem Statement : Write the assembly language code & C code to get square wave of 2kHz frequency & verify the same by using timer as counter.

Calculations : Crystal Frequency = 12MHz
Machine frequency = clock frequency/12 = 1 MHz.
1 clock pulse = 1/MHz = 1µsec.

For Frequency = 2 KHz
1 pulse = 1/2KHz = 0.5 msec = 500 µsec
50% duty cycle
250 µsec ON time and 250 µsec OFF line.
Count = 250µsec/1µsec = 250
Counter initial value = 65536 - 250 = 65286 = FF06H

For Timer = 20ms
Count = 20ms/1µsec = 20000
Counter Initial Value = 65536 - 20000 = 45536 = B1E0H

Code :

Assembly Code

```
CLR P0.7  ;ENABLE DAC

;set Timer 1 in mode 1
;set Timer 0 as counter
MOV TMOD,#15H

MOV TH0, #00H
MOV TL0, #00H
SETB TR0 ;START COUNTER

;20ms TIMER for verification
MOV TH1,#0B1H
MOV TL1,#0E0H
SETB TR1        ;START
```

C Code

```
#include<reg51.h>

void delay(int t);
void main()
{
    int t;
    unsigned char A=0x00;
//set Timer 1 in mode 1 and Timer 0 as counter
    TMOD = 0x15;

//Initial counter value 0
    TH0 = 0x00;
    TL0 = 0x00;
    TR0 = 1;   //Start Counter
    TIMER

SQUARE:
;HIGH-LOW-HIGH-LOW
        CPL A
;DAC  Display
        MOV P1,A
;AS P3.5 IS FEEDED TO COUNTER
        MOV P3,A
;124 IS LOADED INTO REGISTER
    3
        MOV R3,#7CH
        ACALL DELAY
;LOOP TILL TIMER 1 OVERFLOW
    FLAG =1
        JNB TF1,SQUARE
JMP LAST  ;END PROGRAM

;DELAY FUNCTION
DELAY:
     ;LOOP TILL R3 == 0
        DJNZ R3,DELAY
        RET

LAST:
END
```

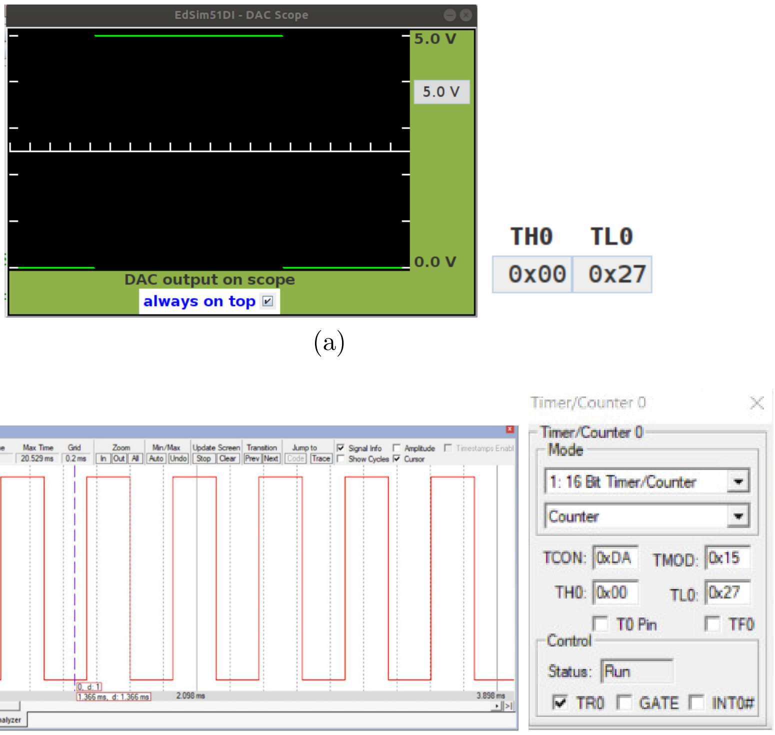
```
    TH1 = 0xB1;
    TL1 = 0xE0; //20ms Timer
    TR1 = 1; //Start Timer

//Run for 20ms only
     while(TF1 != 1)
        {
//High-Low-High-Low
          A = ¬(A);
          P1 = A;
//AS P3.5 IS FEEDED TO COUNTER
          P3 = A;
          t = 29;
          delay(t);
        }
}
void delay(int t)
//Delay Function
{
    while(t != 0)
        {t--;}
    return;
}
```

Output: For developing & compiling the assembly output, EdSim simulator is used & Keil is used to compile and analyze C Code.

Discussion & Observations : For 2KHz square wave by calculation 250us delay is required which is produced by using instruction DJNZ which requires 2 cycles for completion i.e. 2 clock pulse = 2µsec & hence 125 times of instruction DJNZ will give 250µsec delay. But as MOV,CPL instructions also requires time hence to compensate that time we provide 248µsec delay. Square wave is produced by complimenting accumulator after delay.
For verification, we consider program running for 1sec. Hence counter should give ideally 2000 count. P3.5 is the external clock source for timer 1. As in 8051, we have 16 bit timer we can have maximum 65536 counts i.e. 65ms delay max. Hence we analyze output for 20ms & then multiply counter value by 50(1/0.02) for 1sec. Timer is started initially & square wave is generated till timer overflow flag bit is set high i.e. for 20ms. The counter increments everytime P3.5 is set low.

For 20ms, Counter value = 27H = 40D
For 1sec, Counter value = 40 x 50 = 2000.



TH0 TL0
0x00 0x27

Min Time: 0 s
Max Time: 20.529 ms
Grid: 0.2 ms
0 s .498 ms
1.366 ms, d: 1.366 ms

Timer/Counter 0
Mode: 1: 16 Bit Timer/Counter
Counter
TCON: 0xDA TMOD: 0x15
TH0: 0x00 TL0: 0x27
☐ T0 Pin ☐ TF0
Control
Status: Run
☑ TR0 ☐ GATE ☐ INT0#

(a)

(b)

Figure 1.12: Figure showing **Square wave generated on DAC & counter value after 1 sec.**

Conclusion : We developed assembly language code & C code to generate 2KHz square wave & displayed it using DAC & verified it using counters. The circuit diagram connecting 8051 with DAC is shown in figure 1.4.
The output for DAC/logic analyzer displaying 2KHz square wave & counter value is shown in figure 1.12.

Experiment-7: Serial Transmission (UART)

Aim : To Display name using UART Serial Transmission.

Problem Statement : Write assembly & C code to display name using **U**niversal **A**synchronous **R**eceiver **T**ransmitter serial communication using Mode 0 & Mode 1.

Calculations : Crystal Frequency = 11.059MHz
Machine frequency = clock frequency/12 = 921.6 KHz.
1 clock pulse = 1/MHz = 1.08 µsec.

Baud rate for SMOD = '0'
Internal UART frequency FUART= machine frequency/32 = 28.8 KHz
FOR BR = 9600,28800 should be divided by 3.
We put TH1 = 253 = -3 = FDH ,T1 as auto reload mode
Hence 28800 Hz will get divided by 3 as the timer will overflow after every 3 cycles.

Baud Rate for SMOD = '1'
Internal UART frequency FUART= machine frequency/16 = 57.6 KHz
FOR BR = 9600,57600 should be divided by 6.
We put TH1 = 253 = -6 = FAH ,T1 as auto reload mode
Hence 28800 Hz will get divided by 6 as the timer will overflow after every 6 cycles.

Similarly, other standard BAUD RATES and TH1 values

BR for SMOD =0	BR for SMOD =1	TH1(decimal)	TH1(hex)
1200	2400	-28	E8
2400	4800	-12	F4
4800	9600	-6	FA
9600	19200	-3	FD

Also, TH1 & BR can be calculated directly using below formula

$$TH1 = 256 - \frac{F_{oscillator} * 2^{SMOD}}{12 * 32 * BR}$$

$$BR = \frac{F_{oscillator} * 2^{SMOD}}{12 * 32 * (256 - TH1)}$$

Code : Using Mode 0
C Code

```
#include<reg51.h>
#include<string.h>

void String_Data(char *datastr)
{
    int i;
    for(i=0;i<strlen(datastr);i++) // Send each character of ...
        string till the NULL
    {
        SBUF = datastr[i];       //Load data
         while (TI == 0);        //Wait till Transmission completes
         TI=0;                   //Reset transmit interrupt flag
     }
     return;
}

void main()
{
    SCON = 0x00;                          // Mode 0
    String_Data("Shruti Murarka");      // Transmit
    while(1);
}
```

Using Mode 1

Assembly Code

```
;SET DATA POINTER TO NAME
    STRING
MOV DPTR,#NAME

;SET TIMER 1 IN MODE 2
;8-bit AUTO RELOAD MODE
MOV TMOD,#20H
;FOR BR 4800
MOV TH1,#0FAH
;MODE 1 WITH RECEPTION ENABLE
;8-bit UART communication with
     variable baud BR
MOV SCON,#50H

;In case we want to double BR
;after Timer 1 exceeds its
    count FFH
;Possible way to use higher
    crystal oscillator
;rather use PCON(byte
    addressable)
     ;MOV A, PCON
```

C Code

```
#include<reg51.h>
#include<string.h>

void String_Data(char *datastr)
{
    int i;
// Send each character of
    string till the NULL

        for(i=0;i<strlen(datastr);i++)
    {SBUF = datastr[i];
// Wait till Transmission
    completes
     while (TI == 0);
//Reset transmit interrupt flag
     TI=0;}
     return;
}

void main()
{
    ;SETB ACC.7
;SET SMOD = 1 in PCON register
    ;MOV PCON, A

SETB TR1 ;START TIMER 1
CLR A
MOV B,#0

AGAIN:
;LOAD A WITH CHARACTER
    MOVC A,@A+DPTR
;LOAD SBUF REGISTER WITH DATA
    MOV SBUF,A
;WAIT TILL COMPLETE TRANSMISSION
    LOOP:JNB TI,LOOP
;CLEAR TRANSMIT INTERRUPT FLAG
    CLR TI
;INCREMENT B
    INC B
    MOV A,B
;RUN LOOP 14 TIMES
    CJNE A, #14, AGAIN
JMP LAST ;END PROGRAM

NAME:  ;NAME STRING
DB "SHRUTI MURARKA"

LAST:
    END
```

```
// Timer 1 in Mode 2
// 8-bit auto reload mode
    TMOD = 0x20;
// Load value for 9600 baud rate
    TH1 = 0xFD;
// Mode 1, reception enable
//8-bit UART communication with variable BR
    SCON = 0x50;
// Start timer 1
    TR1 = 1;

// Transmit
 String_Data("Shruti Murarka");
    while(1);
}
```

Output: For developing & compiling the assembly output, EdSim simulator is used & Keil is used to compile and analyze C Code.

Discussion & Observations : Serial communication means to transfer data bit by bit serially at a time, preferable for long distance communication as single wire makes it cheaper. UART is asynchronous serial communication protocol. A UART frame consists of the data to be transmitted (8 bits/9 bits) and Start(0) and Stop bits(1). UART chips allow programming of the parity bit for odd, even and no-parity options to check for transmission errors. Baud Rate refers to number of bits transmitted per second. The registers used in serial communication are
1. Serial Control Register(SCON) - The control center for serial communication.
2. Serial Buffer Register(SBUF) - Holds the data to be received or transmitted.
3. Power Control Register(PCON) - Doubles speed of transmission i.e. Baud Rate.
4. Timer Mode Control Register(TMOD) - Controls speed of transmission or Baud

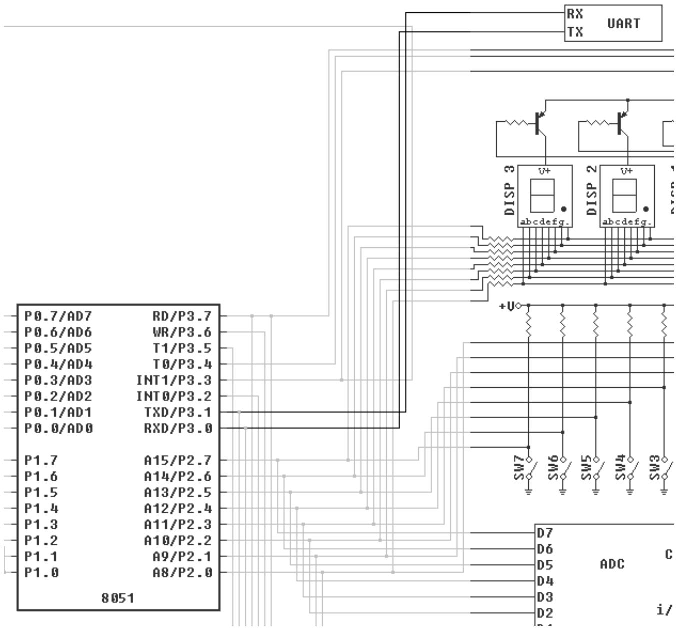
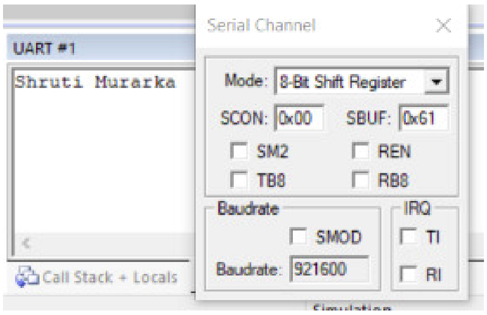


Figure 1.13: Figure showing **circuit diagram for UART.**

Rate.
Above registers are set according to need(Mode 0/1).
In Mode 0 (shift register operation for data transmission) serial port works like shift register. Baud Rate is always Fosci/12. For crystal oscillator frequency 11.059 MHz, BR is found to 1/12*Fosci = 921583.34 == 921500. Quick data transmission occurs synchronously. For assembly code, in edsim51 we have limited standard baud rates hence we cant set baud rate to 921500 for Mode 0 transmission. Due to this we cant simulate assembly code. Keil allows variable baud rate hence C code is complied for Mode 0 Transmission.
The disadvantage of Mode 0 is overcome by Mode 1. Mode 1 allows the baud rate to be variable and is set by Timer 1 of 8051. Assembly Code for BR 4800 is written. We can double the BR i.e. 9600 using PCON register by setting SMOD = 1. For BR = 9600, C Code is written. The calculations for TH1 value is shown in calculations. Codes are very well commented step by step.
For more details refer [Datasheet].

Conclusion : We developed assembly language code & C code to display name using UART Serial Communication using both Mode 0 & Mode 1. The circuit diagram connecting 8051 with UARTis shown in figure 1.13.
The output for UART Serial Transmission using Mode 0 is shown in figure 1.14.
The output for UART Serial Transmission using Mode 1 is shown in figure 1.15.



Shruti Murarka

Mode: 8-Bit Shift Register

SCON: 0x00 SBUF: 0x61

□ SM2 □ REN

□ TB8 □ RB8

Baudrate

□ SMOD

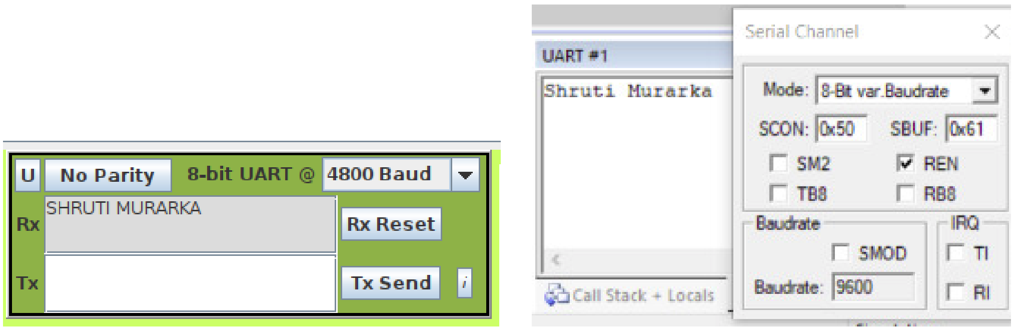
Baudrate: 921600

IRQ

□ TI

□ RI

Figure 1.14: Figure showing **output for UART Serial Transmission using Mode0**



No Parity 8-bit UART @ 4800 Baud

SHRUTI MURARKA

Shruti Murarka

Mode: 8-Bit var.Baudrate

SCON: 0x50 SBUF: 0x61

□ SM2 ☑ REN

□ TB8 □ RB8

Baudrate

□ SMOD

Baudrate: 9600

IRQ

□ TI

□ RI

(a)

(b)

Figure 1.15: Figure showing **output for UART Serial Transmission using Mode1**

<u>Extension</u>: Write assembly code to receive name then transmit it & indicate it with glowing LEDs.

Assembly Code:

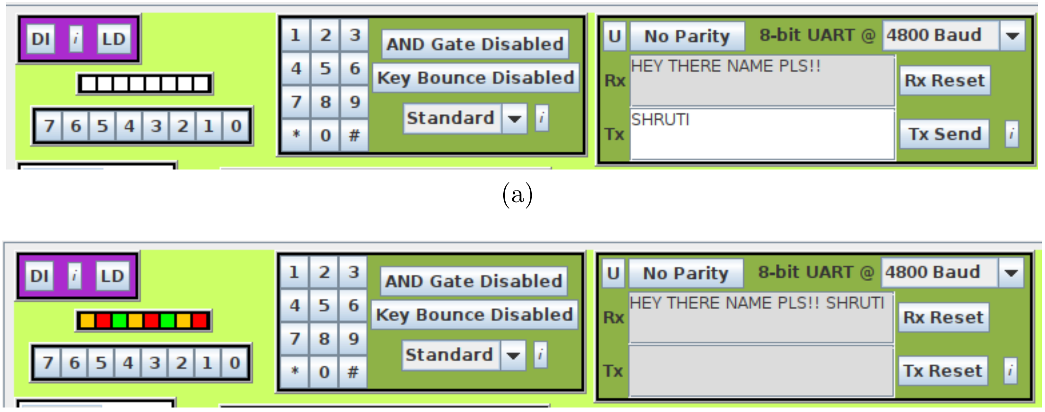
```
MOV DPTR,#NAME       ;SET DATA POINTER TO NAME STRING

MOV TMOD,#20H        ;SET TIMER 1 IN MODE 2,8-bit AUTO RELOAD MODE
MOV TH1,#0FAH        ;FOR BR 4800
MOV SCON,#50H    ;MODE 1(8-bit UART communication with variable ...
    baud BR WITH RECEPTION ENABLE

SETB TR1             ;START TIMER 1
CLR A
MOV B,#0

AGAIN:
    MOVC A,@A+DPTR                  ;LOAD A WITH CHARACTER
    MOV SBUF,A                      ;LOAD SBUF REGISTER WITH DATA
    LOOP:JNB TI,LOOP                ;WAIT TILL COMPLETE TRANSMISSION
    CLR TI                          ;CLEAR TRANSMIT INTERRUPT FLAG
    INC B                           ;INCREMENT B
    MOV A,B
    CJNE A, #21, AGAIN              ;RUN LOOP 21 TIMES
SETB TI
HERE:
    LOP:JNB RI,LOP                  ;WAIT TILL RI FLAG 1
    MOV A,SBUF
    MOV SBUF,A                      ;SEND RECEIVED DATA TO TRANSMIT
    MOV P1,#0                       ;GLOW LEDs
    JNB TI,LOOP                     ;WAIT TILL TRANSMISSION COMPLETES
    SETB TI
    CLR RI
    SJMP HERE                       ;INFINITE LOOP

NAME:  ;NAME STRING
DB "HEY THERE NAME PLS!!  "
```



Rx: HEY THERE NAME PLS!!

Tx: SHRUTI

(a)

Rx: HEY THERE NAME PLS!! SHRUTI

(b)

Figure 1.16: Figure showing **UART Serial Communication (a) before data received (b) after data received**

<u>Conclusion</u>: We also developed assembly language code to receive name using UART Serial Communication, transmit the same using Mode 1 & indicate the same by glowing LEDs.
The output before & after data received is shown in the figure 1.16.

Experiment-8: I^{2}C Communication

Aim : To transfer data using I^{2}C Communication protocol.

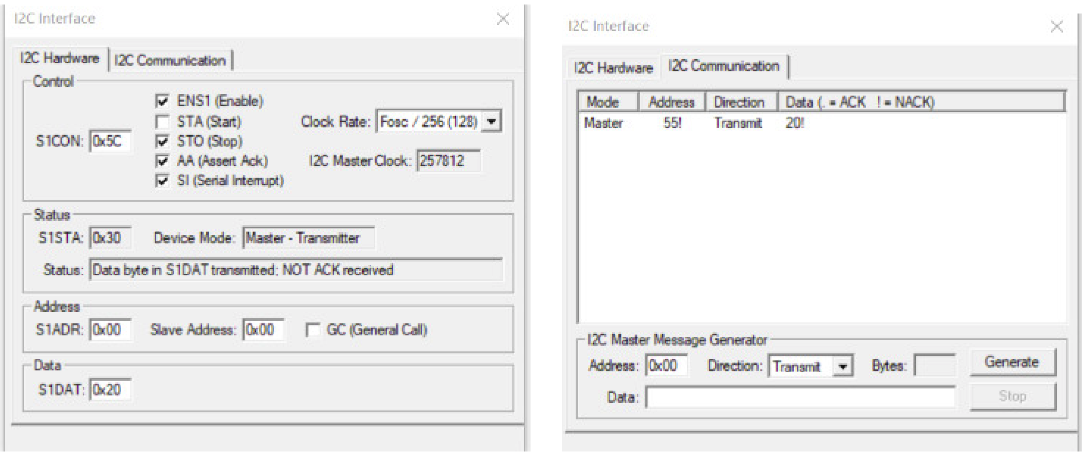
Problem Statement : Write a C code to write 00100000 (20H) to a slave with the address 1010101 with P89C660.

C Code :

```c
#include<reg66x.h> // header file for P89C66X uc
void main()
{   //set Assert Acknowledge Flag and I2C-bus Interface Enable
    S1CON = 0x44;
    STA = 1;         // Set STA to generate START condition
    while (!SI);     // Wait till I2C-bus Interrupt Flag=1 to confirm
    STA = 0;         //  Stop START Condition
    S1DAT = 0xAA;    // slave address 1010101 +0(W) = AAH
    SI = 0;          // clear SI bit
    while (!SI);     // wait till I2C-bus Interrupt Flag=1 to confirm
    S1DAT = 0x20;    // send data 20H
    SI = 0;           // clear SI bit
    while (!SI);      //wait till I2C-bus Interrupt Flag=1 to confirm
    STO = 1;      //set STO to generate STOP condition
    while (1);
}

//Program to send hex values for name string characters
#include<reg66x.h> // header file for P89C66X uc
void main()
{   unsigned char name[25] = "SHRUTI"; //string
    int i;
    //set Assert Acknowledge Flag and I2C-bus Interface Enable
    S1CON = 0x44;
    STA = 1;         // Set STA to generate START condition
    while (!SI);     // Wait till I2C-bus Interrupt Flag=1 to confirm
    STA = 0;         //  Stop START Condition
    S1DAT = 0xAA;    // slave address 1010101 +0(W) = AA
    SI = 0;          // clear SI bit
    for(i=0;i<6;i++)     //loop 6 times
        {while (!SI);// wait till I2C-bus Interrupt Flag=1 to confirm
        S1DAT = name[i];    // send data
        SI = 0;}           // clear SI bit
    while (!SI);      //wait till I2C-bus Interrupt Flag=1 to confirm
    STO = 1;      //set STO to generate STOP condition
    while (1);
}
```

Output : Keil is used to develop and analyse C Code.



I2C Interface

I2C Hardware | I2C Communication

Control

S1CON: 0x5C

☑ ENS1 (Enable)
□ STA (Start)
☑ STO (Stop)
☑ AA (Assert Ack)
☑ SI (Serial Interrupt)

Clock Rate: Fosc / 256 (128)

I2C Master Clock: 257812

Status

S1STA: 0x30 Device Mode: Master - Transmitter

Status: Data byte in S1DAT transmitted; NOT ACK received

Address

S1ADR: 0x00 Slave Address: 0x00 □ GC (General Call)

Data

S1DAT: 0x20

(a)

I2C Interface

I2C Hardware | I2C Communication

Mode	Address	Direction	Data (. = ACK ! = NACK)
Master	55!	Transmit	20!

I2C Master Message Generator

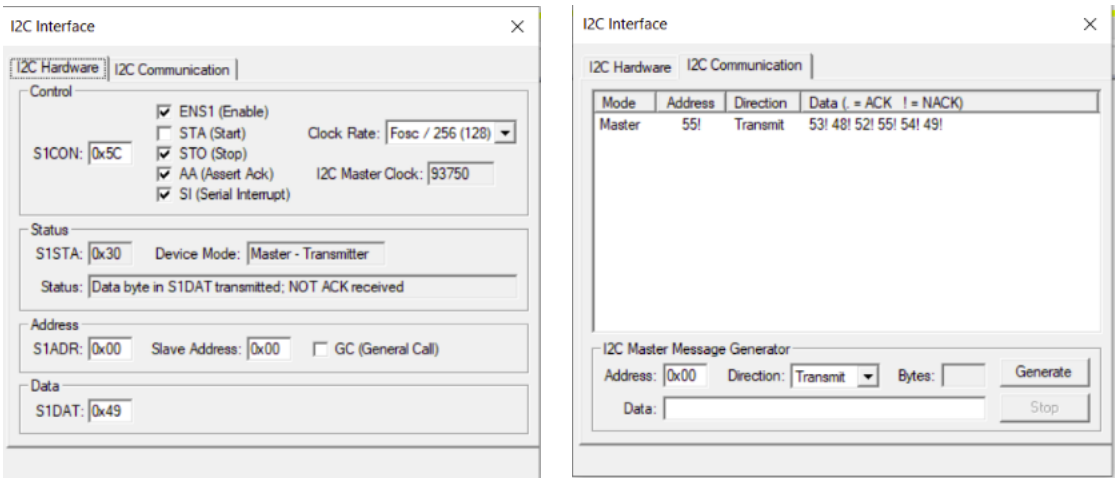
Address: 0x00 Direction: Transmit Bytes:

Data:

(b)

Figure 1.17: Figure showing **I^{2}C Interface**

Discussion & Observations: The Inter-Integrated Circuit (I^{2}C) is a serial synchronous type Communication Protocol that allows multiple slaves to a single master as well as multiple masters controlling single, or multiple slaves.
I^{2}C uses only two wires to transmit data between devices:
1. SDA(Serial Data) – Master and slave sends and receives data.
2. SCL(Serial Clock) – Carries the clock signal.
The bus has two roles :
1. Master node – Generates the clock and initiates communication with slaves.
2. Slave node – Receives the clock and responds when addressed by the master.
Also,
Transmitter – The device which sends the data to the bus.
Receiver – The device which receives the data from the bus.
Accordingly there are 4 types of modes:
1. Master Transmitter – Master node is sending data to a slave.
2. Master Receiver – Master node is receiving data from a slave.
3. Slave Transmitter – Slave node is sending data to the master.
4. Slave Receiver – Slave node is receiving data from the master.
Data is transferred in messages.The message includes start condition, address frame for slave, read/write bits, and one or more data frames that contain the data being



I2C Interface

I2C Hardware | I2C Communication

Control

S1CON: 0x5C

☑ ENS1 (Enable)
☐ STA (Start)
☑ STO (Stop)
☑ AA (Assert Ack)
☑ SI (Serial Interrupt)

Clock Rate: Fosc / 256 (128)

I2C Master Clock: 93750

Status

S1STA: 0x30 Device Mode: Master - Transmitter

Status: Data byte in S1DAT transmitted; NOT ACK received

Address

S1ADR: 0x00 Slave Address: 0x00 ☐ GC (General Call)

Data

S1DAT: 0x49

(a)

I2C Interface

I2C Hardware | I2C Communication

Mode	Address	Direction	Data (. = ACK ! = NACK)
Master	55!	Transmit	53! 48! 52! 55! 54! 49!

I2C Master Message Generator

Address: 0x00 Direction: Transmit Bytes:

Data:

(b)

Figure 1.18: Figure showing **I^{2}C Interface showing Hex values for string characters**

transmitted with ACK/NACK bits between each data frame and stop conditions. Message is shown in figure 1.19.

Start Condition – The master device leaves SCL high and pulls SDA low.

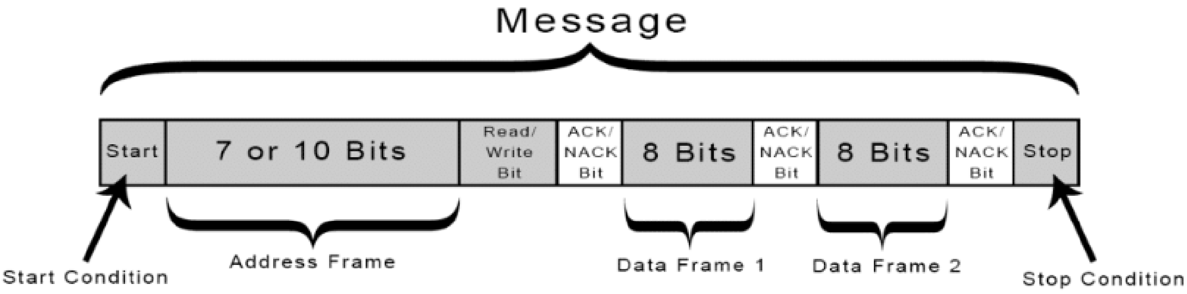


Figure 1.19: Figure showing **Message broken into frames.**

Stop Condition – The SDA line switches from 0 to 1 after the SCL line switches from 0 to 1.

Address Frame: For a 7-bit address, the address is in MSB first, followed by a R/W(1/0) bit indicating read/write operation.

IN P89C660,

The P89C660 CPU interfaces with the I^{2}C bus through four Special Function Reg-

isters (SFRs):
1. S1CON (SIO1 control register): Controls the SIO1(serial i/o) functions: start and restart of a serial transfer, termination of a serial transfer, bit rate, address recognition, and acknowledgment.
2. S1STA (SIO1 status register): Read only register. The five most significant bits contain the status code.
3. S1DAT (SIO1 data register): Contains a byte of serial data to be transmitted or a byte which has just been received.
4. S1ADR (SIO1 slave address register): Contains 7-bit Slave address(MSB) and The LSB of S1ADR is general call bit. When this bit is set, the general call address (00H) is recognized.
For more details refer [Datasheet].

Conclusion : We developed C code to transfer data on given memory address using I^{2}C Communication using P89C660 microcontroller. The output for I^{2}C Communication is shown in figure 1.17.

Experiment-9: SPI Communication

Aim: To transfer data using SPI Communication protocol.

Problem Statement : Write a C code to write a string to a slave with AT89S8252.

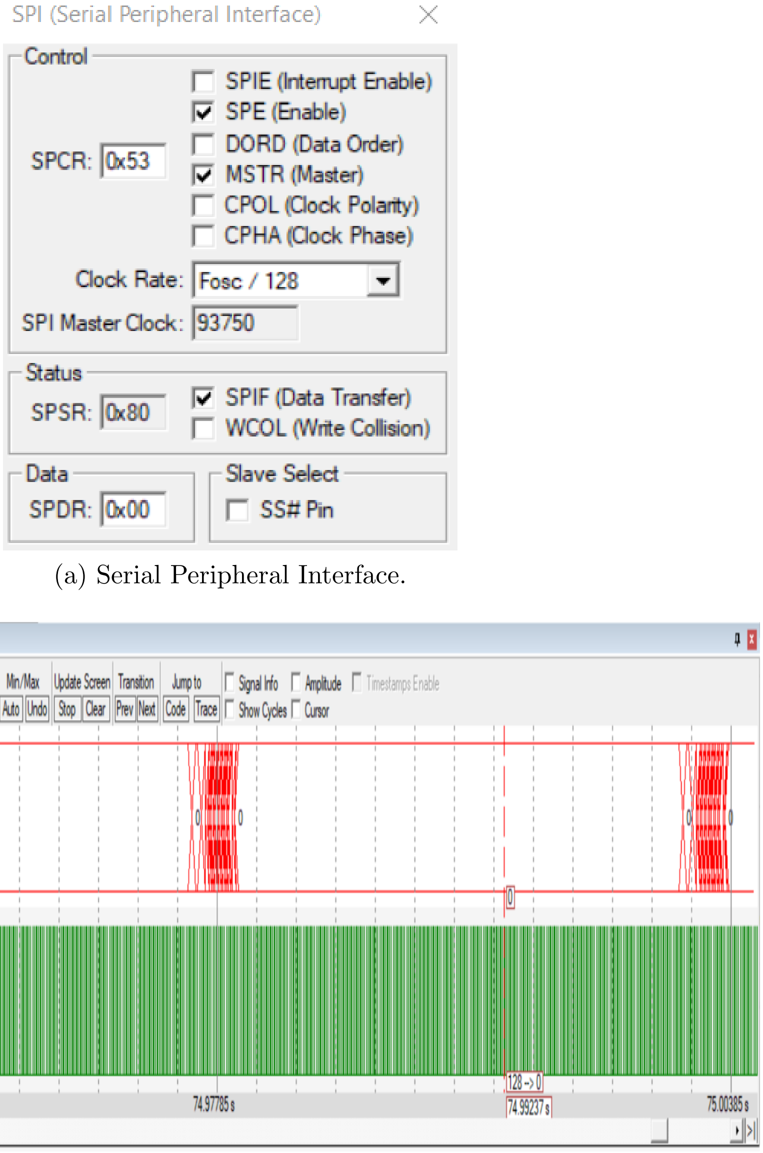
C Code :

```
#include <AT898252.H>
void main ()
{
int i;
char a[]="Shruti here";  //data which needs to be sent
SCON = 0x50;    //Initialize the serial port
TMOD = 0x20;     //self reload mode
TH1 = 221;  //load value
TR1 = 1;    //start timer 1
SPCR = 0x53;    //enable spi in master mode, fosc/128 rate
SS = 0;         //enable slave
while(1)
{
if (a[i]=='\0')     //reset i if end of string is reached
{i=0;}
SPDR = a[i];    //sending the data
while ((SPSR & 0x80) == 0);     //wait for confirmation
{i++;}
SPSR=0x00;
}
}
```

Output : Keil is used to develop and analyse C Code.

Discussion & Observations: One unique benefit of SPI is the fact that data can be transferred without interruption. Any number of bits can be sent or received in a continuous stream. With I2C and UART, data is sent in packets, limited to a specific number of bits. Start and stop conditions define the beginning and end of each packet, so the data is interrupted during transmission.
Confriguration of SPI:

- MOSI (Master Output/Slave Input) – Line for the master to send data to the slave.



SPI (Serial Peripheral Interface)

Control

SPCR: 0x53

☐ SPIE (Interrupt Enable)
☑ SPE (Enable)
☐ DORD (Data Order)
☑ MSTR (Master)
☐ CPOL (Clock Polarity)
☐ CPHA (Clock Phase)

Clock Rate: Fosc / 128

SPI Master Clock: 93750

Status

SPSR: 0x80

☑ SPIF (Data Transfer)
☐ WCOL (Write Collision)

Data

SPDR: 0x00

Slave Select

☐ SS# Pin

(a) Serial Peripheral Interface.

(b) Logic Analzer showing SPDR & SPSR values

Figure 1.20: SPI Communication.

- MISO (Master Input/Slave Output) – Line for the slave to send data to the master.
- SCLK (Clock) – Line for the clock signal.
- SS/CS (Slave Select/Chip Select) – Line for the master to select which slave to send data to.

Datasheet for AT89S8252 :[Datasheet].

We can see the SPSR and SPDR values in the logic analysis window of Keil which shows that the signal has been transmitted.

Conclusion: We developed C code to transfer data using SPI Communication using AT89S8252 microcontroller. The output for SPI Communication is shown in figure 1.20.