Programming Timers of 8051/8052 Microcontroller

Lab Exercises on October 23, 2020

Department of Electronics and Computer Engineering
Pulchowk Campus, Lalitpur

Ashlesh Pandey

PUL074BEX007

Contents

List of Figures

List of Tables

Source Codes

1 Introduction

1.1 MCS-51 Family Microcontroller Chips

Based on the Harvard architecture of designing ICs, the MCS-51 microcontroller chips were originally developed by Intel to be used in small embedded systems. The MCS-51 chips now use Complementary Metal-Oxide Semiconductor (CMOS) instead of the original NMOS, and are thus known as 80C51 chips. Texas Instruments, Atmel, Dallas Semiconductors, Silicon Laboratories, ASTX, and many more distributors manufacture and sell the MCS-51 family microcontroller chips.
The different features of the 8051 microcontroller are:

- 8-bit ALU, Accumulator and Registers, making it an 8-bit microcontroller.
- 4KB of ROM for the programs, also called program memory.
- 8-bit data bus meaning that it can access 8 bits of data in one operation.
- 128 Bytes of RAM for the variables, also called data memory.
- 32 I/O lines, i.e. 4 ports with 8 lines each.
- 16-bit address bus meaning that it can access 65536 locations of RAM and ROM.
- 2 16-bit timers/counters.
- 1 full-duplex serial port for serial communication (UART).
- 6 interrupt sources(2 external interrupts, 2 timer interrupts & 2 serial interrupts).

1.2 Timers in 8051

The 8051 microcontroller has two on-chip timers/counters that are used for timing durations for delays or counting external events. Timers allow real-time delays to be implemented in the system. With the frequency at which a microcontroller operates, we can calculate the required delay.
8051 has two timers, namely Timer 0 and Timer 1, which are both 16 bit. Each timer is separated into two bytes namely TH0/TL0 for Timer 0 and TH1/TL1 for Timer 1.

1.2.1 TMOD Register

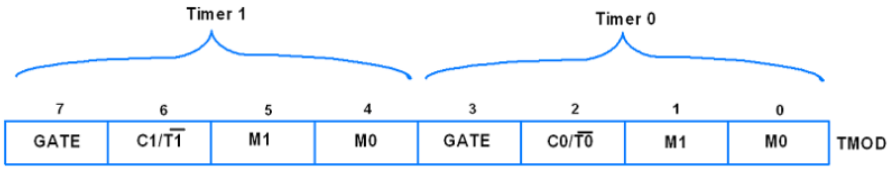


Figure 1: TMOD register bit description

TMOD is an 8-bit register that is used by both the timers in 8051. The lower nibble is used by Timer 0 and upper nibble is used by Timer 1. Short description for each bit is mentioned below:

Gate

Gating control bit is used to select between the timer/counter mode. When set, timer is enable when INTx pin is high and TRx control pin is set. When cleared, timer is enabled TRx control bit is set.

Counter/Timer

Cleared to enable timer. Set to enable counter operation.

M1 and M0

Used as select pins for different modes of timers.

M1	M0	Mode	Description
0	0	0	13-bit timer mode: 8-bit timer/counter 8 bit of THx and least significant 5 bits of TLx are cascaded.
0	1	1	16-bit timer mode: 16-bit timer/counter THx and TLx are cascaded.
1	0	2	8-bit auto reload: THx holds a value which is to be reloaded in TLx each time it overflows.
1	1	3	Split timer mode: Split 16-bit timerx into two 8-bit timer i.e. THx and TLx like two 8-bit timer. Only Timer 0 is used in this mode.

Table 1: Timer modes selection combinations

1.2.2 TCON Register

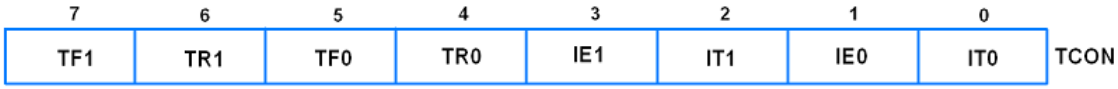


Figure 2: TCON register bit description

TCON is an 8-bit register used by both timers in 8051 as well as interrupts. THe lower nibble is used by interrupts and upper nibble is used by timers. Short description for each bit is mentioned below:

TF1 and TF0

Timer 1 and Timer 0 overflow flags. Set when timer overflows the maximum value.

TR1 and TR0

Timer 1 and Timer 0 run enable. Set to start the respective timer, and reset to stop the timer.

IE1 and IE0

Interrupt 1 and Interrupt 0 edge detect flags. Set when interrupt edge is detected.

IT1 and IT0

Interrupt 1 and Interrupt 0 type control bits. When set, it detects the falling edge interrupt, when cleared it detects zero level interrupt.

1.2.3 Timer Modes

Timer Mode 0 (13-bit timer mode)

This mode is selected by choosing M1 = 0 and M0 = 0 in the TMOD register. TH (8-bit) and TL (lower 5 bits) of timer are cascaded to form a 13-bit timer. Values from 0000 H to 1FFF H can be loaded into timer's register. After loading required values to TH and TL register, Timer 0 is started using command SETB TR0 and Timer 1 is started by using the command SETB TR1. Timer counts from loaded values up to 1FFF H and rolls over to 0000 H setting TF0 flag for Timer 0 and TF1 for Timer 1.

Timer Mode 1 (16-bit timer mode)

This mode is selected by choosing M1 = 0 and M0 = 1 in the TMOD register. TH (8-bit) and TL (8 bits) of timer are cascaded to form a 16-bit timer. Values from 0000 H to FFFF H can be loaded into timer's register. After loading required values to TH and TL register, Timer 0 is started using command SETB TR0 and Timer 1 is started by using the command SETB TR1. Timer counts from loaded values up to FFFF H and rolls over to 0000 H setting TF0 flag for Timer 0 and TF1 for Timer 1.

Timer Mode 2 (8-bit auto reload mode)

This mode is selected by choosing M1 = 1 and M0 = 0 in the TMOD register. Only TL (8 bits) register of timer is used. Values from 00 H to FF H can be loaded into timer's register. TH register automatically loads a copy of the initial value into TL register. After TL receives a copy of the 8-bit initial value, TL0 of Timer 0 is started using command SETB TR0 and TL1 of Timer 1 is started by using the command SETB TR1. Timer counts from loaded values up to FF H and rolls over to 00 H setting TF0 flag for Timer 0 and TF1 for Timer 1.

Timer Mode 3 (Split timer mode)

This mode is selected by choosing M1 = 1 and M0 = 1 in the TMOD register. Timer 0 splits into two 8-bit timers. TL0 and TH0 act as two separate timers. TL0 uses TR0 and TF0 while TH0 uses TR1 and TF1 bit. After TL0 and/or TH0 are initialized with 8-bit initial value, timers are started. TL0 is started using command SETB TR0 and TH0 is started by using the command SETB TR1. Timer counts from loaded values up to FF H and rolls over to 00 H setting TF0 flag if TL0 overflows and setting TF1 flag if TH0 overflows. Only Timer 0 can be used in this mode.

1.2.4 Delay calculation

The required timer mode is selected using the combinations mentioned in Table 1. The timer is clocked by an external oscillator of frequency 11.0592 MHz. The frequency for timer is $(\frac{1}{12})^{th}$ of the frequency of external oscillator.

$$Frequency\ of\ oscillator = 11.0592MHz$$

$$Machine\ cycle\ frequency = \frac{11.0592}{12} = 921.6\ KHz$$

$$Machine\ cycle\ period = \frac{1}{921.6\ KHz} = 1.085\ \mu s$$

Consider that we want a delay of T seconds. We can calculate the required HEX values to be entered in TH and TL of the timer registers in different modes as,

$$(Z)_{10} = \frac{T}{1.085\ \mu s}$$

$$(N)_{10} = (Maximum\ timer\ value + 1) - Z_{10}$$

Converting $(N)_{10}$ into hexadecimal value will give the required hex output that needs to be stored in the THx and/or TLx. It is to be noted that the *maximum timer value* is different for different modes and the final hexadecimal value needs to be cascaded into equivalent form as per the mode.

1.3 7-Segment LED Display

7-segment LED display is a basic electronic device that is made up of 7 led segments each arranged in such a way that specific configurations of these LEDs allow certain alpha-numeric characters to be displayed on the device. Most 7-segment LEDs also contain an additional LED to indicate the decimal point when two or more 7-segment LED displays are used in conjuction. Each of the seven LEDs have a positional reference with pin outs generally being named as a, b, c, d, e, f, g and DP. One end of all the LEDs are commoned out whereas other end is provided with a proper biasing voltage to either turn on or off the segment depending on the terminal polarity. Forward biasing of the appropriate LED segments are used to display the desired character or pattern. Widely used in digital clocks, small-scale calculators, electronic meters and digital display units, the 7-segment LED displays serve a handy purpose in digital electronics.

1.3.1 Digital Drive Pattern

For a common cathode 7-segment LED display, the segments a through g and DP must be provided HIGH logic in order to be turned on. Likewise, for a common anode 7-segment LED display, the segments a through g and DP must be provided LOW logic in order to be turned on. The different combinations that are to be used to display certain characters are called digital drive pattern. The table that stores these patterns is called a lookup table.

Character	DP	g	f	e	d	c	b	a	HEX value
0	0	0	1	1	1	1	1	1	3F H
1	0	0	0	0	0	1	1	0	06 H
2	0	1	0	1	1	0	1	1	5B H
3	0	1	0	0	1	1	1	1	4F H
4	0	1	1	0	0	1	1	0	66 H
5	0	1	1	0	1	1	0	1	6D H
6	0	1	1	1	1	1	0	1	7D H
7	0	0	0	0	0	1	1	1	07 H
8	0	1	1	1	1	1	1	1	7F H
9	0	1	1	0	1	1	1	1	6F H

Note: For displaying any character along with a decimal point, the HEX value should be ORed with 80 H.

Table 2: Lookup table for a Common-Cathode 7-segment display

2 Objectives

The primary objective of this lab experiment is to understand various programming concepts for timers in 8051/8052 microcontroller. The programming concepts for timers will enable us to write assembly and Embedded C codes for the 8051/8052 microcontroller capable of:

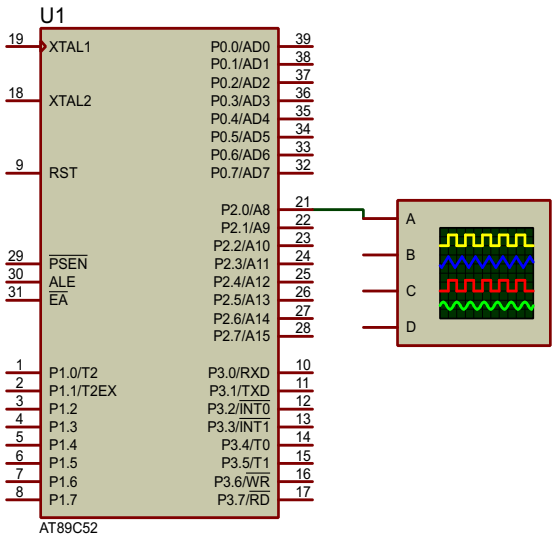
- Applying timers in different timing modes.
- Implementing accurate delays using timers.

3 Lab Experiment Environment

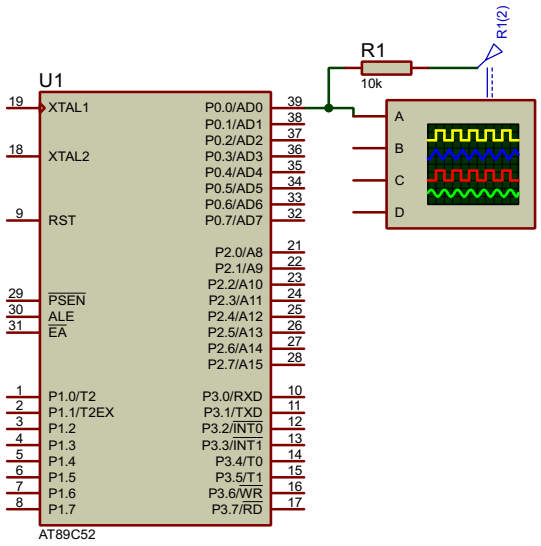
The lab experiments will be performed virtually via various simulation softwares. The basic usages of these tools will allow us to write and visualize different programs involving timers in 8051/8052 microcontroller. Due to a simulated environment, the observations are carefully selected to represent the problems with maximum efficiency.

3.1 Circuit Simulation

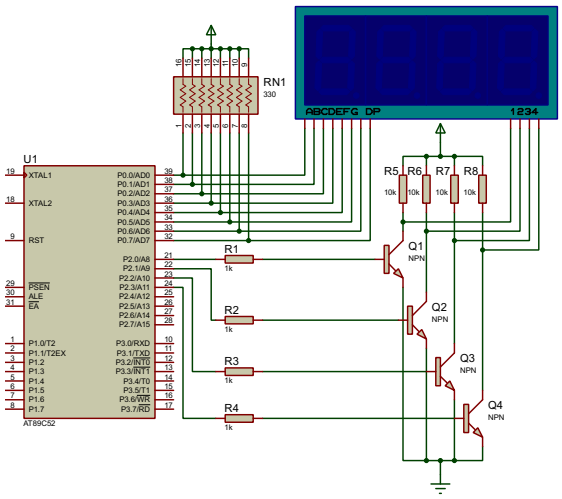
A profession PCB layout, circuit design and simulation tool, Proteus Design Suite was used to simulate the interface between a 8052 microcontroller and an oscilloscope for Problem 1 and Problem 2, and a common cathode 7-segment LED display for Problem 3. Additional electronic components such as resistors, resistor bus, transistors were also used to attain the circuits shown in Figure 3.



(3.1) Schematic for Problem 1



(3.2) Schematic for Problem 2



(3.3) Schematic for Problem 3

Figure 3: Circuit diagrams for Proteus simulation

3.2 Code Editor and Compiler

KEIL μVision, which is a product of the ARM Ltd. was used as the code editor for the assembly and embeddded C codes for the different lab problems. KEIL products include C/C++ compilers, debuggers, integrated development and simulation environments, RTOS and middleware libraries, and evaluation boards for ARM, Cortex-M, Cortex-R4, 8051, C166, and 251 processor families. The compiler built-in with KEIL converts the codes into respective HEX codes that are understandable by the microcontroller.

4 Lab Problems

Problem 1
Generate a periodic square wave having a period of 15 ms and a duty cycle of 20%. The waveform should be produced at pin zero of port two (P2.0). The XTAL frequency is 11.0592 MHz. Observe the waveform on an oscilloscope and measure the ON and OFF timers.

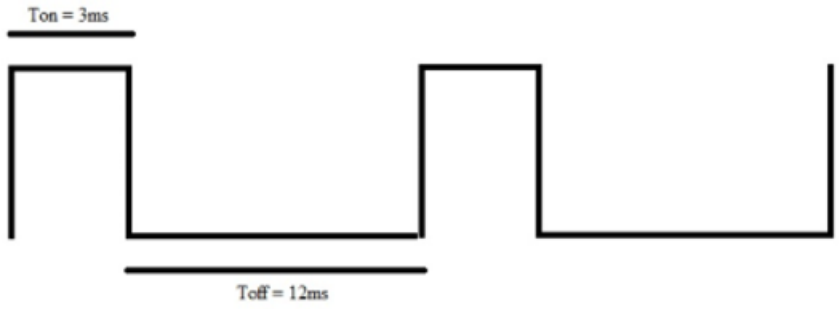


Figure 4: Waveform to be generated for Problem 1

a. Using Timer 1 in mode 0 (13-bit timer mode)

Assembly Code

```
ORG 00H
MOV TMOD,#00
HERE: MOV TL1,#14H
MOV TH1,#0A9H
SETB P2.0
ACALL DELAY
MOV R2,#04H
AGN: MOV TL1,#14H
MOV TH1,#0A9H
CLR P2.0
ACALL DELAY
DJNZ R2,AGN
SJMP HERE
DELAY: SETB TR1
AGAIN: JNB TF1, AGAIN
CLR TR1
CLR TF1
RET
END
```

Code 1: Problem 1 - Assembly

Embedded C Code

```
#include <reg51.h>
sbit select_bit = P2 ^ 0;
void delay(void)
{
    TMOD = 0x00;
    TL1 = 0x14;
    TH1 = 0xA9;
    TR1 = 1;
    while (!TF1);
    TR1 = 0;
    TF1 = 0;
}
```

```
void main(void)
{
    int i;
    while (1)
    {
        select_bit = 1;
        delay();
        for (i = 0; i < 4; i++)
        {
            select_bit = 0;
            delay();
        }
    };
}
```

Code 2: Problem 1 - Embedded C

b. Using Timer 0 is mode 1 (16-bit timer mode)

Assembly Code

```
ORG 00H
MOV TMOD,#01
HERE: MOV TL0,#34H
MOV TH0,#0F5H
SETB P2.0
ACALL DELAY
MOV R2,#04H
AGN: MOV TL0,#34H
MOV TH0,#0F5H
CLR P2.0
ACALL DELAY
DJNZ R2,AGN
SJMP HERE
DELAY: SETB TR0
AGAIN: JNB TF0, AGAIN
CLR TR0 ;1
CLR TF0 ;1
RET ;2
END
```

Code 3: Problem 1 - Assembly

Embedded C Code

```
#include <reg51.h>
sbit select_bit = P2 ^ 0;
void delay(void)
{
    TMOD = 0x01;
    TL0 = 0x34;
    TH0 = 0xF5;
    TR0 = 1;
    while (!TF0);
    TR0 = 0;
    TF0 = 0;
}
void main(void)
{
    int i;
    while (1)
    {
        select_bit = 1;
        delay();
        for (i = 0; i < 4; i++)
        {
            select_bit = 0;
            delay();
        }
    };
}
```

Code 4: Problem 1 - Embedded C

c. Using Timer 1 in mode 2 (8-bit auto-reload timer mode)

Assembly Code

```
ORG 00H
MOV TMOD,#20H
HERE: MOV R2,#0FH
AGN: MOV TH1,#48H
SETB P2.0
ACALL DELAY
DJNZ R2,AGN
MOV R2,#3CH
AGN1: MOV TH1,#48H
CLR P2.0
ACALL DELAY
DJNZ R2,AGN1
SJMP HERE
DELAY: SETB TR1
AGAIN: JNB TF1, AGAIN
CLR TR1
```

```
CLR TF1
RET
END
```

Code 5: Problem 1 - Assembly

Embedded C Code

```
#include <reg51.h>
sbit select_bit = P2 ^ 0;
void delay(void)
{
    int i;
    TMOD = 0x20;
    for (i = 0; i < 15; i++)
    {
        TH1 = 0x48;
        TR1 = 1;
        while (!TF1);
        TR1 = 0;
        TF1 = 0;
    }
}
void main(void)
{
    int i;
    while (1)
    {
        select_bit = 1;
        delay();
        for (i = 0; i < 4; i++)
        {
            select_bit = 0;
            delay();
        }
    };
}
```

Code 6: Problem 1 - Embedded C

d. Using Timer 0 (TL0) in mode 3 (8-bit split timer mode)

Assembly Code

```
ORG 00H
MOV TMOD,#03H
HERE: MOV R2,#0FH
AGN: MOV TL0,#48H
SETB P2.0
ACALL DELAY
DJNZ R2,AGN
MOV R2,#3CH
AGN1: MOV TL0,#48H
CLR P2.0
ACALL DELAY
DJNZ R2,AGN1
SJMP HERE
DELAY: SETB TR0
AGAIN: JNB TF0, AGAIN
CLR TR0
CLR TF0
RET
END
```

Code 7: Problem 1 - Assembly

Embedded C Code

```
#include <reg51.h>
sbit select_bit = P2 ^ 0;
void delay(void)
{
    int i;
    TMOD = 0x03;
    for (i = 0; i < 15; i++)
    {
        TL0 = 0x48;
        TR0 = 1;
        while (!TF0);
        TR0 = 0;
        TF0 = 0;
    }
}
void main(void)
{
    int i;
    while (1)
    {
        select_bit = 1;
        delay();
        for (i = 0; i < 4; i++)
        {
            select_bit = 0;
            delay();
        }
    };
}
```

Code 8: Problem 1 - Embedded C

Problem 2
Generate the periodic waveform as shown in figure 11. The waveform should be produced at pin zero of port zero (P0.0). The XTAL frequency is 11.0592 MHz. Observe the waveform on an oscilloscope and measure the ON and OFF times.

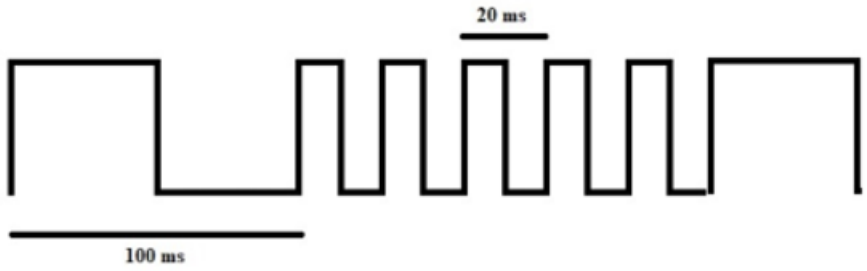


Figure 5: Waveform to be generated for Problem 2

a. Using Timer 0 in mode 0 (13-bit timer mode)

Assembly Code

```
ORG 00H
MOV TMOD,#00
REPEAT: MOV R2,#02H
LOOP1: MOV R1,#0AH
HERE1: MOV TL0,#00H
MOV TH0,#70H
ACALL DELAY
DJNZ R1,HERE1
CPL P0.0
DJNZ R2,LOOP1
MOV R2,#0AH
LOOP2: MOV R1,#02H
HERE2: MOV TL0,#00H
MOV TH0,#70H
ACALL DELAY
DJNZ R1,HERE2
CPL P0.0
DJNZ R2,LOOP2
SJMP REPEAT
DELAY: SETB TR0
AGAIN: JNB TF0, AGAIN
CLR TR0
CLR TF0
RET
END
```

Code 9: Problem 2 - Assembly

Embedded C Code

```
#include <reg51.h>
sbit select_bit = P0 ^ 0;
void delay(int factor)
{
    int i;
    TMOD = 0x00;
    for (i = 0; i < factor; i++)
    {
        TL0 = 0x00;
        TH0 = 0x70;
        TR0 = 1;
        while (!TF0);
        TR0 = 0;
        TF0 = 0;
    }
}
void main(void)
{
    int i;
    while (1)
    {
        select_bit = 1;
        delay(10);
        select_bit = 0;
        delay(10);
        for (i = 0; i < 5; i++)
```

```
        {
            select_bit = 1;
            delay(2);
            select_bit = 0;
            delay(2);
        }
    };
}
```

Code 10: Problem 2 - Embedded C

b. Using Timer 1 is mode 1 (16-bit timer mode)

Assembly Code

```
ORG 00H
MOV TMOD,#10H
REPEAT: MOV R2,#02H
HERE1: MOV TL1,#0FEH
MOV TH1,#4BH
ACALL DELAY
CPL P0.0
DJNZ R2,HERE1
MOV R2,#0AH
HERE2: MOV TL1,#00H
MOV TH1,#0DCH
ACALL DELAY
CPL P0.0
DJNZ R2,HERE2
SJMP REPEAT
DELAY: SETB TR1
AGAIN: JNB TF1, AGAIN
CLR TR1
CLR TF1
RET
END
```

Code 11: Problem 2 - Assembly

Embedded C Code

```
#include <reg51.h>
sbit select_bit = P0 ^ 0;
void delay(char TH, char TL)
{
    TH1 = TH;
    TL1 = TL;
    TMOD = 0x10;
    TR1 = 1;
    while (!TF1);
    TR1 = 0;
    TF1 = 0;
}
void main(void)
{
    int i;
    while (1)
    {
        select_bit = 1;
        delay(0x4B, 0xFE);
        select_bit = 0;
        delay(0x4B, 0xFE);
        for (i = 0; i < 5; i++)
        {
            select_bit = 1;
            delay(0xDC, 0x00);
            select_bit = 0;
            delay(0xDC, 0x00);
        }
    };
}
```

Code 12: Problem 2 - Embedded C

c. Using Timer 0 in mode 2 (8-bit auto-reload timer mode)

Assembly Code

```
ORG 00H
MOV TMOD,#02H
LOOP: MOV R2,#02H
HERE1: MOV R1,#0C0H
HER1: MOV TH0,#1AH
ACALL DELAY
DJNZ R1,HER1
CPL P0.0
DJNZ R2,HERE1
MOV R2,#0AH
HERE2: MOV R1,#26H
HER2: MOV TH0,#1AH
ACALL DELAY
DJNZ R1,HER2
CPL P0.0
DJNZ R2,HERE2
```

```
SJMP LOOP
DELAY: SETB TR0
AGAIN: JNB TF0, AGAIN
CLR TR0
CLR TF0
RET
END
```

Code 13: Problem 2 - Assembly

Embedded C Code

```
#include <reg51.h>
sbit select_bit = P0 ^ 0;
void delay(int factor)
{
    int i;
    for (i = 0; i < factor; i++)
    {
        TMOD = 0x02;
        TH0 = 0x1A;
        TR0 = 1;
        while (!TF0);
        TR0 = 0;
        TF0 = 0;
    }
}
void main(void)
{
    int i;
    while (1)
    {
        select_bit = 1;
        delay(192);
        select_bit = 0;
        delay(192);
        for (i = 0; i < 5; i++)
        {
            select_bit = 1;
            delay(38);
            select_bit = 0;
            delay(38);
        }
    };
}
```

Code 14: Problem 2 - Embedded C

d. Using Timer 0 (TH0) in mode 3 (8-bit split timer mode)

Assembly Code

```
ORG 00H
MOV TMOD,#03H
REPEAT: MOV R2,#02H
HERE1: MOV R1,#0C0H
HER1: MOV TH0,#1AH
ACALL DELAY
DJNZ R1,HER1
CPL P0.0
DJNZ R2,HERE1
MOV R2,#0AH
HERE2: MOV R1,#26H
HER2: MOV TH0,#1AH
ACALL DELAY
DJNZ R1,HER2
CPL P0.0
DJNZ R2,HERE2
SJMP REPEAT
DELAY: SETB TR1
AGAIN: JNB TF1, AGAIN
CLR TR1
CLR TF1
RET
END
```

Code 15: Problem 2 - Assembly

Embedded C Code

```
#include <reg51.h>
sbit select_bit = P0 ^ 0;
void delay(int factor)
{
    int i;
    for (i = 0; i < factor; i++)
    {
        TMOD = 0x03;
        TH0 = 0x1A;
        TR1 = 1;
        while (!TF1);
        TR1 = 0;
        TF1 = 0;
    }
}
void main(void)
{
    int i;
```

```
    while (1)
    {
        select_bit = 1;
        delay(192);
        select_bit = 0;
        delay(192);
        for (i = 0; i < 5; i++)
        {
            select_bit = 1;
            delay(38);
            select_bit = 0;
            delay(38);
        }
    };
}
```

Code 16: Problem 2 - Embedded C

Problem 3

Design a digital minutes and seconds in double digit format. The clock should count from 00:00 to 59:59 and repeat. Time should be displayed in decimal format using four 7-segment LED units. A decimal point should separate minutes from seconds. Use an appropriate timer and timer mode. Use port 0 (P0) to send data to 7-segment LED units. Use transistors as switches to activate or deactivate the 7-segment LED units using pins 0, 1, 2 and 3 of port 2 (P2.0, P2.1, P2.2, P2.3).

Assembly Code

```
ORG 00H
MOV TMOD,#10H
MOV P2,#00H
MOV DPTR,#LABEL1
START: MOV R0,#00
MOV R1,#00
LOOP1: MOV R7,#27H
MAIN: MOV A,R0
ACALL HTOD
MOV B,A
ANL A,#0FH
ACALL DISPLAY
SETB P2.3
MOV P0,A
ACALL DELAY_T
CLR P2.3
MOV A,B
ANL A,#0F0H
SWAP A
ACALL DISPLAY
SETB P2.2
MOV P0,A
ACALL DELAY_T
CLR P2.2
MOV A,R1
ACALL HTOD
MOV B,A
ANL A,#0FH
ACALL DISPLAY
ORL A,#80H
SETB P2.1
MOV P0,A
ACALL DELAY_T
CLR P2.1
MOV A,B
ANL A,#0F0H
SWAP A
ACALL DISPLAY
SETB P2.0
MOV P0,A
ACALL DELAY_T
CLR P2.0
DJNZ R7,MAIN
CJNE R0,#3BH,LESS
INC R1
MOV R0,#0FFH
LESS: INC R0
CJNE R1,#3CH,LOOP1
AJMP START
HTOD: MOV B,#0AH
DIV AB
SWAP A
ADD A,B
RET
DELAY_T: MOV TL1,#3FH
MOV TH1,#0E8H
SETB TR1
AGAIN: JNB TF1, AGAIN
CLR TR1
CLR TF1
RET
DISPLAY: MOVC A,@A+DPTR
RET
LABEL1: DB 3FH
DB 06H
DB 5BH
DB 4FH
DB 66H
DB 6DH
DB 7DH
DB 07H
DB 7FH
DB 6FH
END
```

Code 17: Problem 3 - Assembly

Embedded C Code

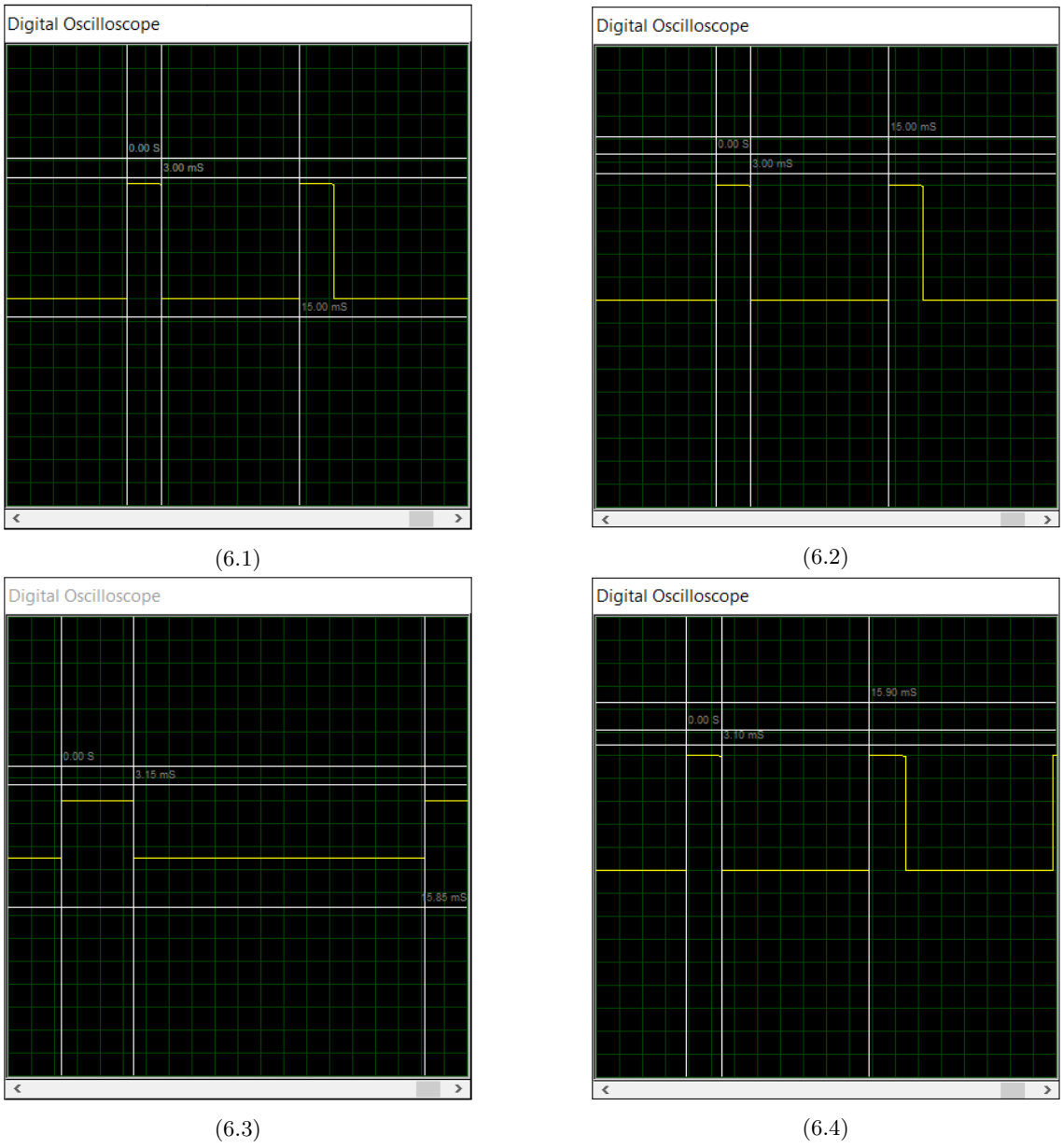
```
#include <reg51.h>
unsigned char led_pattern[10] = {
    0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d,
    0x7d, 0x07, 0x7f, 0x6f};
void delay(void)
{
    TMOD = 0x10;
    TH1 = 0xE9;
    TL1 = 0x3F;
    TR1 = 1;
    while (!TF1);
    TR1 = 0;
    TF1 = 0;
}
void display(int min, int sec)
{
    int i, r, led[4];
    led[0] = min / 10;
    led[1] = min % 10;
    led[2] = sec / 10;
    led[3] = sec % 10;
    for (r = 0; r < 39; r++)
    {
        P2 = 0x01;
        for (i = 0; i < 4; i++)
        {
            if (i == 1)
                P0 = led_pattern[led[i]] | 0
    x80;
            else
                P0 = led_pattern[led[i]];
            delay();
            P2 <<= 1;
        }
    }
}
void main(void)
{
    int i, j;
    while (1)
        for (i = 0; i < 60; i++)
            for (j = 0; j < 60; j++)
                display(i, j);
}
```

Code 18: Problem 3 - Embedded C

5 Observations

The observations for the oscilloscope observations are taken from Proteus in VSM debugging mode. The different waveforms obtained are better observed with cursors labeling the different time variations at key points. The waveforms as required by the questions are observed in the oscilloscope with minimum variations in the delays. Minor variations in the timing are results of neglected decimal values while calculating delays. For the digital clock as required by Problem 3, the 7-segment display was used. Few snippets of the observed display have been included. It is to be noted that the timings as shown in the observations vary by about 5 ms for 30 minutes run. This is most likely due to the neglected decimal points in delay calculation.

Problem 1



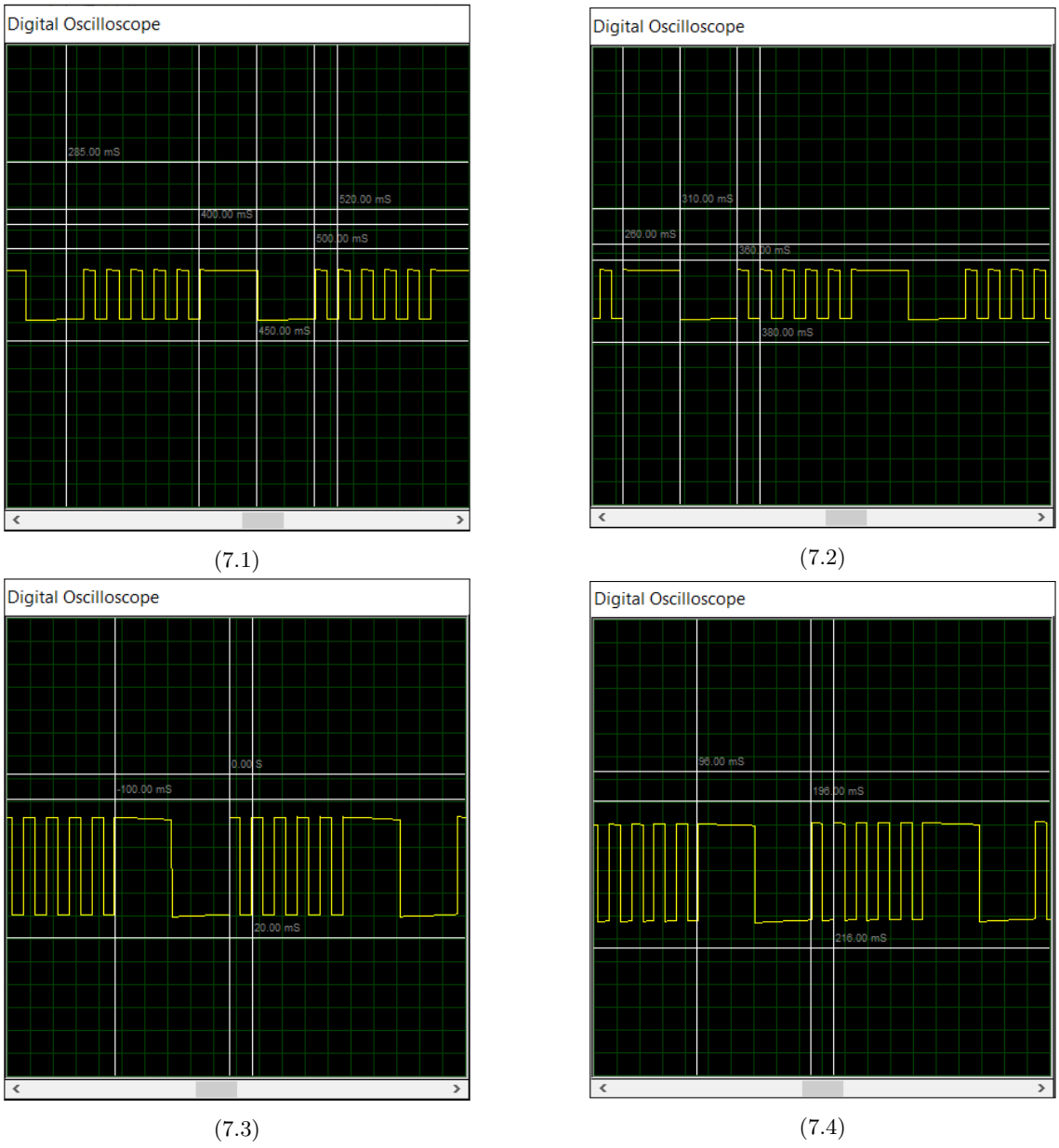
(6.1) (6.2)

(6.3) (6.4)

Figure 6: Observation for Problem 1

Figure 6 shows the observations for Problem 1. The observations for mode 0 and mode 1 are undeviated from actual requirement but that for mode 2 and mode 3 are slightly off due to error in delay calculation. This can be solved by a detailed analysis of the exact machine cycles needed for the delay to be exact.

Problem 2



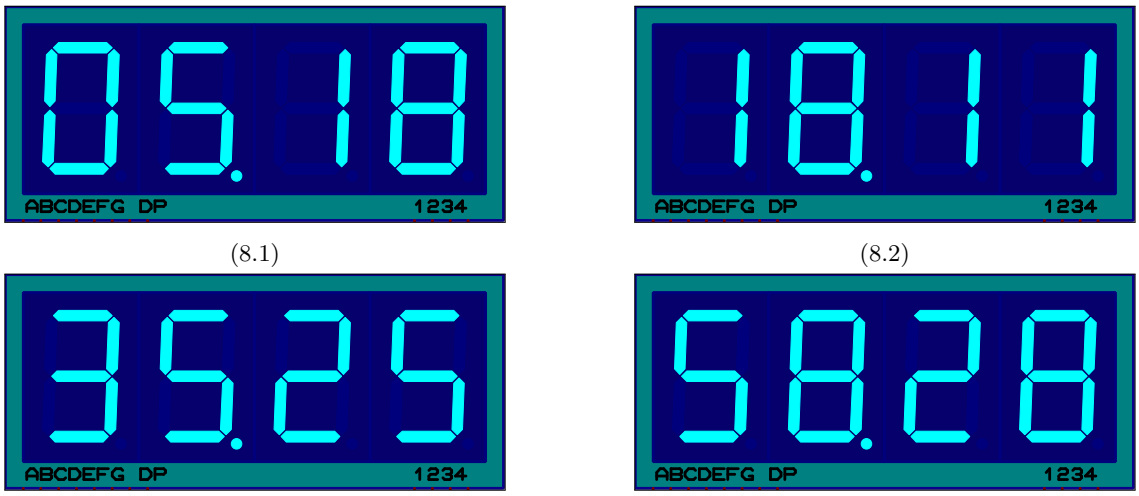
(7.1) (7.2)

(7.3) (7.4)

Figure 7: Observation for Problem 2

Figure 7 shows the observations for Problem 2. The observations for mode 0, mode 1 and mode 2 are undeviated from actual requirement but that mode 3 is slightly off due to error in delay calculation. This can be solved by a detailed analysis of the exact machine cycles needed for the delay to be exact.

Problem 3



(8.1) (8.2)

(8.3) (8.4)

Figure 8: Observation for Problem 3

Figure 8 shows the observations for Problem 3. The digital clock was achieved by introducing accurate delays for a clock to function. 7-segment LED displays were fed with accurate delays between the multiplexed lines to avoid flickering. Port 0 is used to feed the bit configurations for the LED units. Timer mode 1 is used for programming purpose since it is the easiest way to program a delay. The values for minute and second are compared with 60 such that any values crossing it would change either the minute hand or reset the clock as a whole. The minute and second denominations are separated by decimal point by ORing the minute's ones place value. The simulation results for four different points of time are shown above and the results were cross checked with the Proteus VSM animating clock which is considered to be real-time.

6 Discussion

In this lab experiment, programming timers in 8051/8052 was dealt with various problems at hand. Various waveforms with varying delays were produced using ports of an 8051/8052 microcrontroller by using delays from timers. Different modes of timers were practiced, and it helped us attain much needed programming concepts for timers in 8051 using both assembly and Embedded C langugaes.

Additional References

[1] D. Sharma. *8051 Timers*. en. [Online]. Available: URL: https://www.ee.iitb.ac.in/course/~dghosh/EE712/8051/Timers.pdf.

[2] B. Vuksanovic. *Programming Timers on 8051*. en. [Online]. Available: URL: https://www.8051projects.net/files/public/1252055169_5507_FT25871_l2_timers_and_counters.pdf.