Digital Clock & Stop Watch

Using MC80F0448 FLASH MCU

Overview

This program is the example of MC80F0448 functions like Timer, LED Driving , Buzzer Driving etc.

The Digital Clock & Stop Watch is operated by Timer 4 and Watch Timer. And this has one Buzzer for Alarm and four keys; Hour, Minute, Reset and Mode. The buzzer turns on during 500ms every second for 2 minutes, if clock time equals alarm time.

Time is displayed by four 7-segments LED.

Hardware Description

MCU Operation

1. **Timer 4**
   - 16 bits, fxin = 4Mhz, Interrupt Cycle = 1 sec
2. **Watch Timer**
   - fxin = 4Mhz, Interrupt Cycle = fxin / 8192 = 2ms
3. **Buzzer**
   - Output frequency = 1.059Khz
   - Operation: 500ms On / 500ms Off per 1 second
4. **Circuit Diagram**

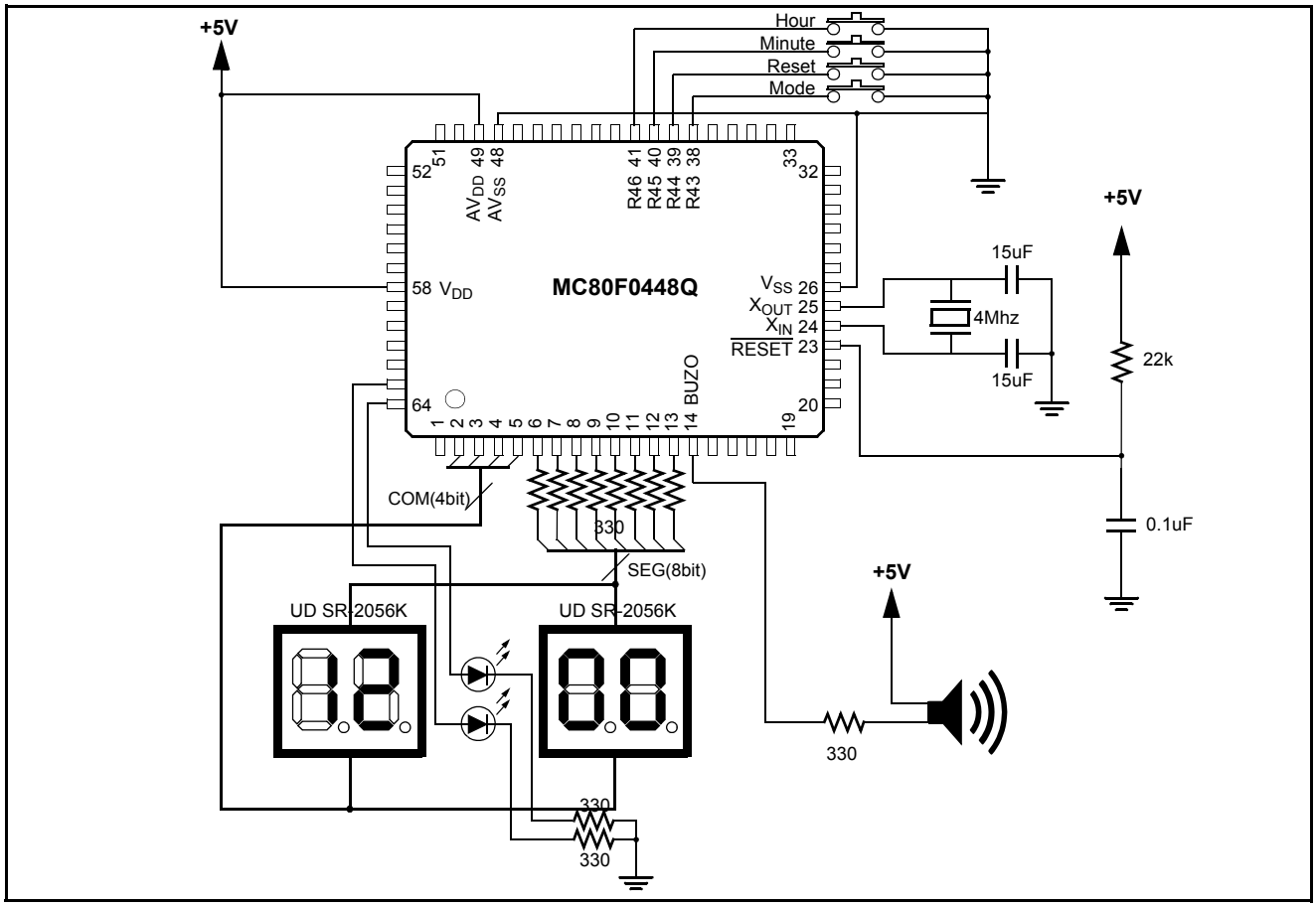


Figure 1. Circuit Diagram

Hardware Operation

Buzzer Operation

A 50% duty pulse can be output to R13/BUZO(Pin No.14) pin to use for piezo-electric buzzer drive. Pin R13/BUZO is assigned for output port of Buzzer driver by setting the bit 2 of PSR1 to "1". For PSR1 register, refer to Manual Section "9. I/O PORTS".

Output pulse through buzzer port is shown in Figure 2.

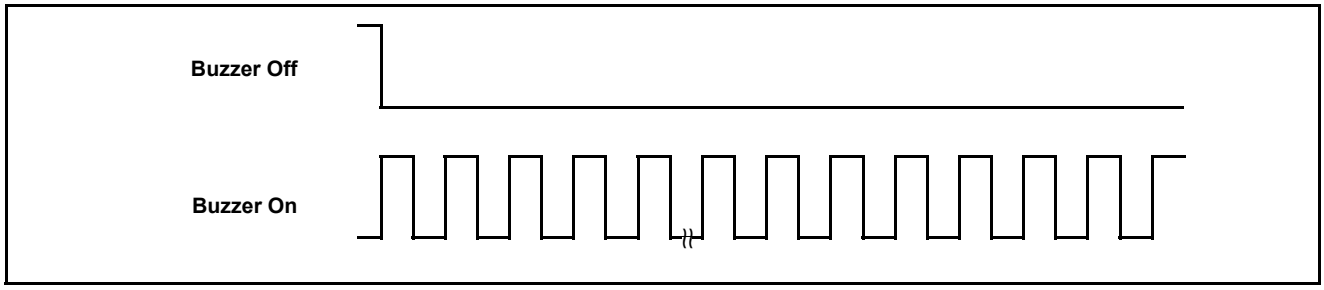


Figure 2. Buzzer Output Pulse

7-Segment Operation

There are 7-Segment two types configuration, common cathode and common anode. So the display must be matched with the driver.

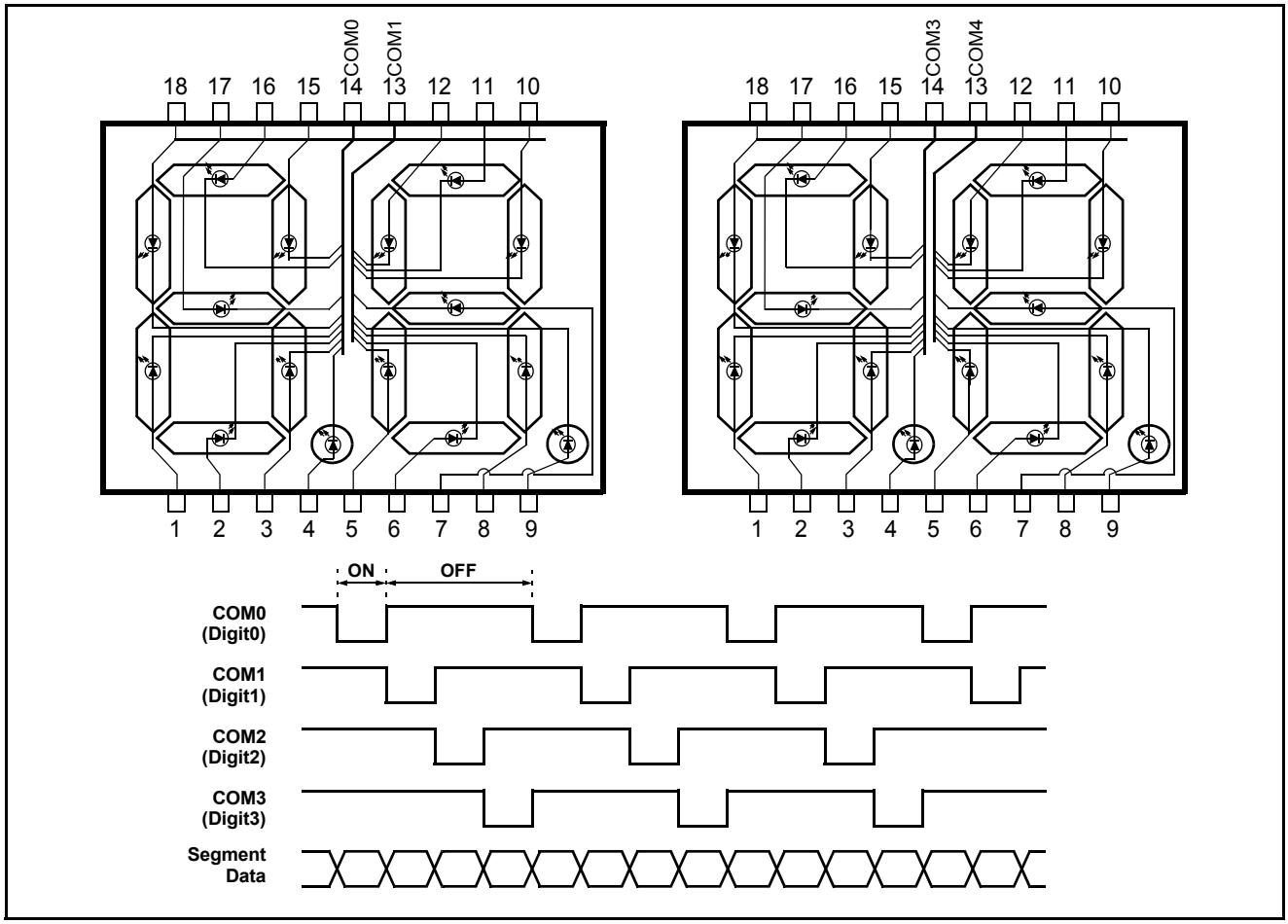


Figure 3. The Pin map and Timing Chart of 7-Segment

Above description type is a common cathode version. That means that the negative leg of each led is connected to a common point. Each LED has a positive leg that is connected to one of the pins of the device. to make it work you need to connect COMx pin to GND. Then to make each segment light up, connect the V_{DD} for that led to ground. A resistor is required to limit the current.

The 7-Segment pin map and timing chart of output signals is shown in Figure 3.

Software Description

Basic Operation

1. Timer 4

Timer 4 interrupt cycle is determined by f_{XIN}(source clock), prescaler divide ratio option and data register value. The equation of Timer Interrupt Cycle calculation is shown below.

$$Int \cdot Cycle = \frac{divide\ ratio \times (data\ register + 1)}{f_{XIN}}$$

For Making 1sec interrupt using by Timer 4 at 4Mhz:

- f_{xin} = 4Mhz
- TM4 = 0001_1111b
- TDR4H = 7, TDR4L = 161

TM4

5	4	3	2	1	0
-	-	CAP4	T4CK2	T4CK1	T4CK0	T4CN	T4ST
0	0	0	1	1	1	1	1

Bit Name	Description
CAP4	**0: Timer/Counter mode** 1: Capture mode selection flag
T4CK2 T4CK1 T4CK0	000: 8-bit Timer, Clock source is $f_{XIN} \div 2$ 001: 8-bit Timer, Clock source is $f_{XIN} \div 4$ 010: 8-bit Timer, Clock source is $f_{XIN} \div 8$ 011: 8-bit Timer, Clock source is $f_{XIN} \div 16$ 100: 8-bit Timer, Clock source is $f_{XIN} \div 64$ 101: 8-bit Timer, Clock source is $f_{XIN} \div 256$ 110: 8-bit Timer, Clock source is $f_{XIN} \div 1024$ **111: 8-bit Timer, Clock source is $f_{XIN} \div 2048$**
T4CN	0: Timer count pause **1: Timer count start**
T4ST	0: When cleared, stop the counting. **1: When set, Timer 0 Count Register is cleared and start again.**

Divide Ratio Selection

The value of Timer Data Register = {(Int.Cycle X f_{XIN}) ÷ Divide Ratio } + 1
= {1 sec X 4 X 10^6 ÷ 2048} + 1
= 01A1h

TDR4H

7	6	5	4	3	2	1	0

TDR4L

7	6	5	4	3	2	1	0

Figure 4. Timer 4 Interrupt Cycle Calculation

2. Watch Timer

In MC80F0448, Watch Timer does not have data register, so Watch Timer interrupt cycle is determined by f_{XIN} and prescaler divide ratio option.

The equation of Watch Timer Interrupt cycle calculation is shown below.

$$Int \cdot Cycle = \frac{divide\ ratio}{f_{XIN}}$$

For Making 1sec interrupt using by Watch Timer at 4Mhz

- f_{xin} = 4Mhz
- WTMR = 1000_1010b

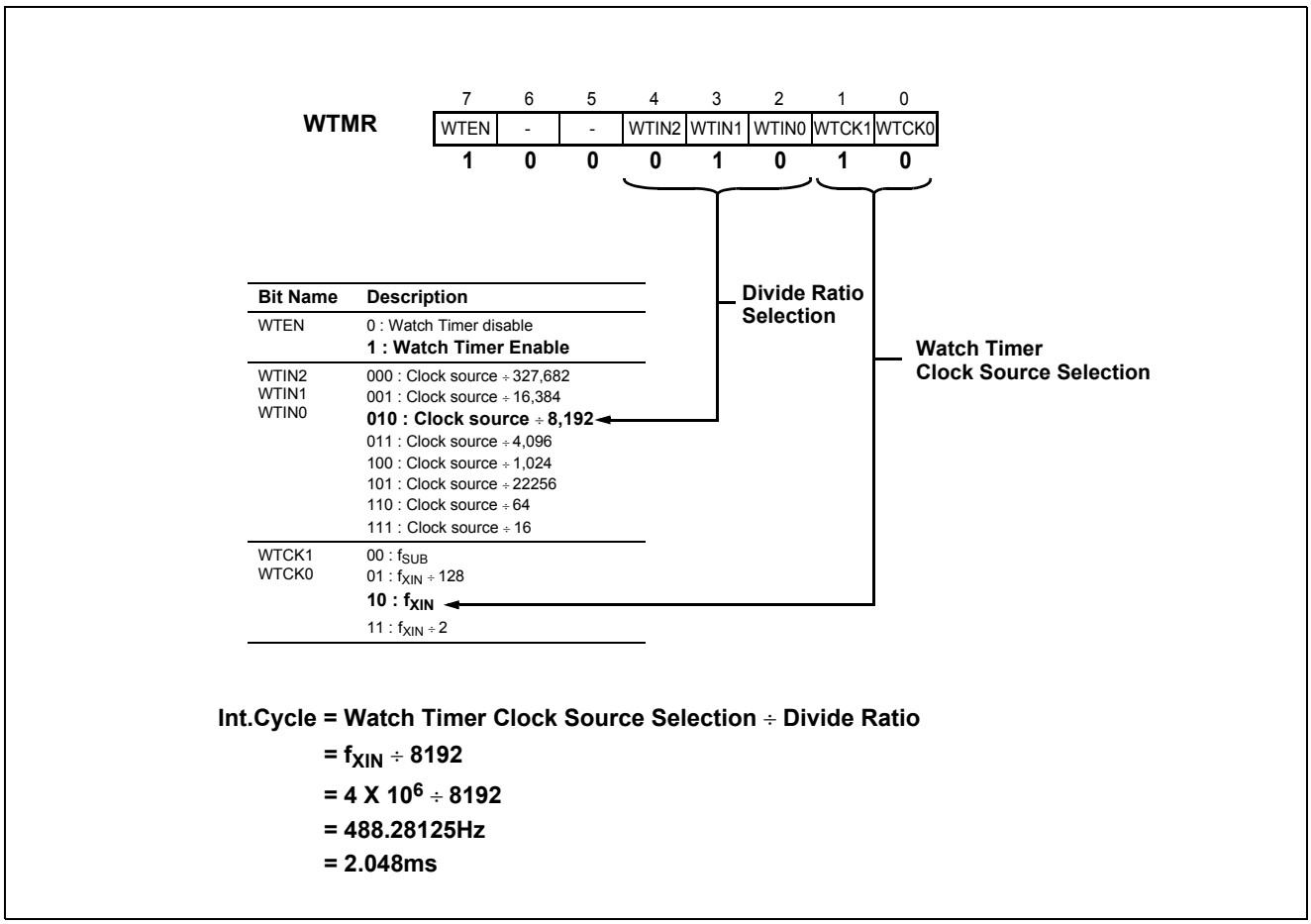


Figure 5. Watch Timer Interrupt Cycle Calculation

3. Key : Low Active

- Hour Key
  - In Clock mode and Alarm mode, increase Clock Hour.
  - If this key is pushed for 5sec continuously, Clock Hour is increased every 200ms.
  - In Stop Watch mode, start/stop the Stop Watch.
- Minute Key
  - In Clock mode and Alarm mode, increase Clock Minute.
  - If this key is pushed for 5sec continuously, Clock Minute is increased every 200ms.
  - In Stop Watch mode, start/stop the Stop Watch.
- Reset Key
  - In Clock mode and Alarm mode, initialize Hour and Minute to “0”.
  - In Stop Watch mode, do nothing.

• Mode Key

- Rotate the mode.

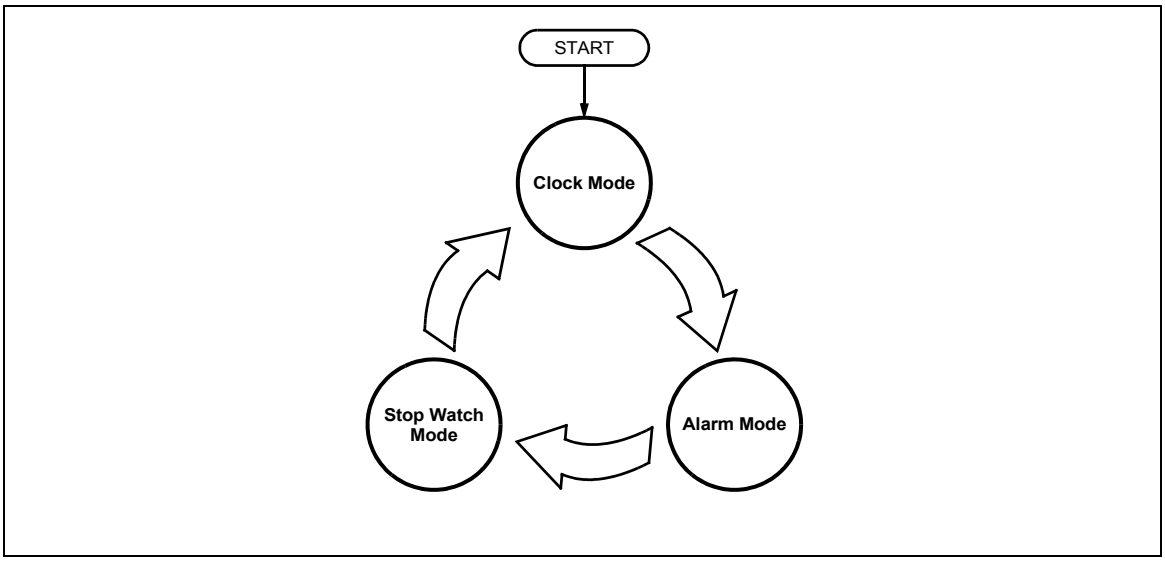


Figure 6. Mode Key Operation

4. Buzzer

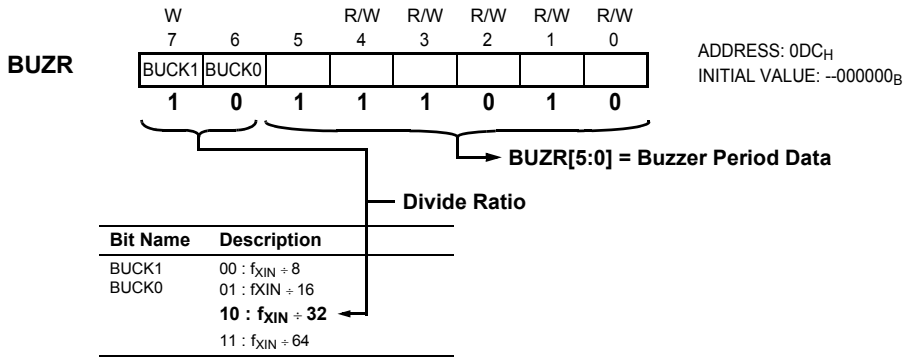
Buzzer frequency is controlled by f_{XIN}(source clock), prescaler divide ratio option and 6-bit period data. The equation of Buzzer frequency calculation is shown below.:

$$f_{BUZ} = \frac{f_{XIN}}{2 \times divide\ \ ratio \times (BUZR[5;0] + 1)}$$

For Making that Buzzer frequency is 1.059Khz at 4Mhz :

- f_{xin} = 4Mhz
- BUZR = 1011_1010b

When main-frequency is 4Mhz, Buzzer frequency is shown as below Table 1.



Buzzer Frequency = f_{XIN} ÷ {2 X Divide Ratio X (BUZR[5:0] +1)}
= 4 X 10^6 ÷ { 2 X 32 x (58(0011_1010b) +1)}
= 4 X 10^6 ÷ 3776
= 1.059KHz

Figure 7. Buzzer frequency Calculation

BUR [5:0]	BUR[7:6]			
	00	01	10	11
00	250.000	125.000	62.500	31.250
01	125.000	62.500	31.250	15.625
02	83.333	41.667	20.833	10.417
03	62.500	31.250	15.625	7.813
04	50.000	25.000	12.500	6.250
05	41.667	20.833	10.417	5.208
06	35.714	17.857	8.929	4.464
07	31.250	15.625	7.813	3.906
08	27.778	13.889	6.944	3.472
09	25.000	12.500	6.250	3.125
0A	22.727	11.364	5.682	2.841
0B	20.833	10.417	5.208	2.604
0C	19.231	9.615	4.808	2.404
0D	17.857	8.929	4.464	2.232
0E	16.667	8.333	4.167	2.083
0F	15.625	7.813	3.906	1.953
10	14.706	7.353	3.676	1.838
11	13.889	6.944	3.472	1.736
12	13.158	6.579	3.289	1.645
13	12.500	6.250	3.125	1.563
14	11.905	5.952	2.976	1.488
15	11.364	5.682	2.841	1.420
16	10.870	5.435	2.717	1.359
17	10.417	5.208	2.604	1.302
18	10.000	5.000	2.500	1.250
19	9.615	4.808	2.404	1.202
1A	9.259	4.630	2.315	1.157
1B	8.929	4.464	2.232	1.116
1C	8.621	4.310	2.155	1.078
1D	8.333	4.167	2.083	1.042
1E	8.065	4.032	2.016	1.008
1F	7.813	3.906	1.953	0.977

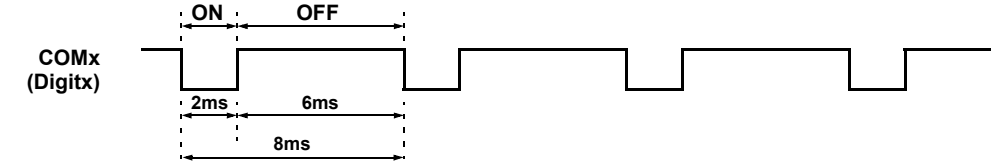
BUR [5:0]	BUR[7:6]			
	00	01	10	11
20	7.576	3.788	1.894	0.947
21	7.353	3.676	1.838	0.919
22	7.143	3.571	1.786	0.893
23	6.944	3.472	1.736	0.868
24	6.757	3.378	1.689	0.845
25	6.579	3.289	1.645	0.822
26	6.410	3.205	1.603	0.801
27	6.250	3.125	1.563	0.781
28	6.098	3.049	1.524	0.762
29	5.952	2.976	1.488	0.744
2A	5.814	2.907	1.453	0.727
2B	5.682	2.841	1.420	0.710
2C	5.556	2.778	1.389	0.694
2D	5.435	2.717	1.359	0.679
2E	5.319	2.660	1.330	0.665
2F	5.208	2.604	1.302	0.651
30	5.102	2.551	1.276	0.638
31	5.000	2.500	1.250	0.625
32	4.902	2.451	1.225	0.613
33	4.808	2.404	1.202	0.601
34	4.717	2.358	1.179	0.590
35	4.630	2.315	1.157	0.579
36	4.545	2.273	1.136	0.568
37	4.464	2.232	1.116	0.558
38	4.386	2.193	1.096	0.548
39	4.310	2.155	1.078	0.539
3A	4.237	2.119	1.059	0.530
3B	4.167	2.083	1.042	0.521
3C	4.098	2.049	1.025	0.512
3D	4.032	2.016	1.008	0.504
3E	3.968	1.984	0.992	0.496
3F	3.907	1.953	0.977	0.488

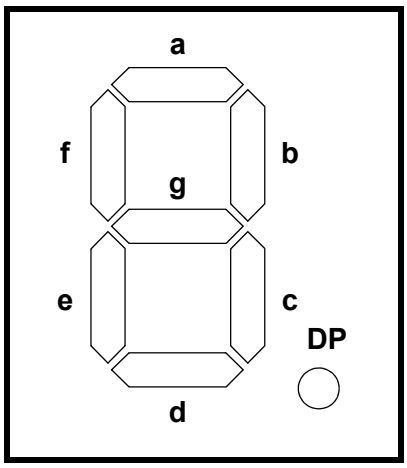
Table 1: Buzzer Frequency(Khz unit)

5. Display : 7-Segment LED

Previous digit display is turned off and next digit is displayed at every Timer interrupt(2ms). In other words, COMx signal is low for 2ms and high for 6ms each other.

The following Figure 8 show ho to form the numbers 0 to 9 and the letters A, b, c, d, E and F.





data Font	a	b	c	d	e	f	g	DP
0	1	1	1	1	1	1	0	0
1	0	1	1	0	0	0	0	0
2	1	1	0	1	1	0	1	0
3	1	1	1	1	0	0	1	0
4	0	1	1	0	0	1	1	0
5	1	0	1	1	0	1	1	0
6	1	0	1	1	1	1	1	0
7	1	1	1	0	0	1	0	0
8	1	1	1	1	1	1	1	0
9	1	1	1	1	0	1	1	0
A	1	1	1	0	1	1	1	0
b	0	0	1	1	1	1	1	0
C	1	0	0	1	1	1	0	0
d	0	1	1	1	1	0	1	0
E	1	0	0	1	1	1	1	0
F	1	0	0	1	1	1	1	0

Table 2: 7-segment Font

*** '0' means that pin is connected to ground.**
'1' means that pin is connected to V_{DD}
*** 7-Segment pin map is described in Fig.3**

Figure 8. 7-Segment Pattern Data

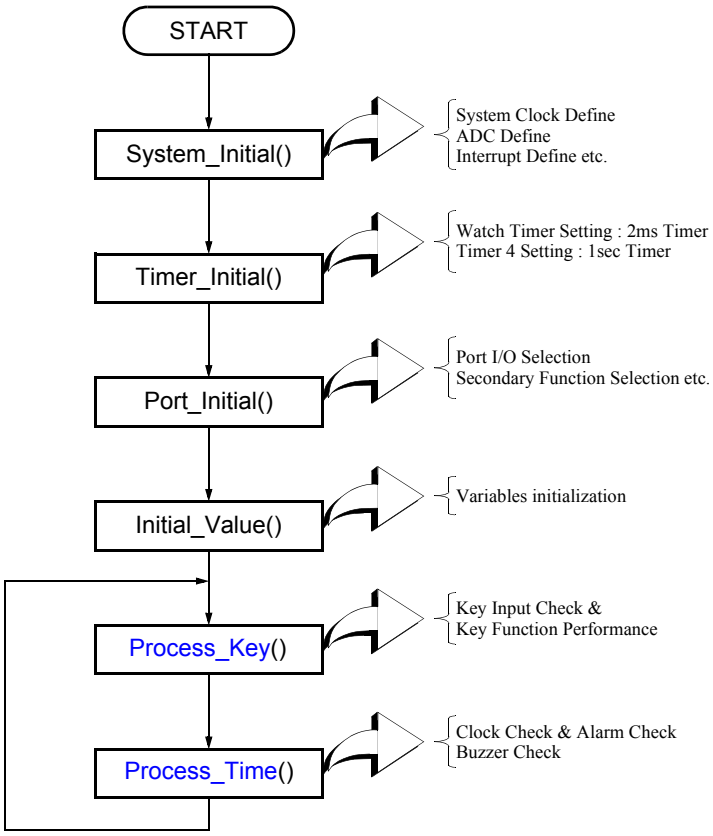
6. RAM Description

Name	Byte	Description
rClock_Min	1	Clock Time Minute Data
rClock_Hour	1	Clock Time Hour Data
rAlarm_Min	1	Alarm Time Minute Data
rAlarm_Hour	1	Alarm Time Hour Data
Hour_10Font	1	Clock Hour High Data
Hour_1Font	1	Clock Hour Low Data
Min_10Font	1	Clock Min High Data
Min_1Font	1	Clock Min Low Data
AHour_10Font	1	Alarm Hour High Data
AHour_1Font	1	Alarm Hour Low Data
AMin_10Font	1	Alarm Min High Data
AMin_1Font	1	Alarm Min Low Data
SMs_100Font	1	Stop Watch Millisecond High Data
SMs_10Font	1	Stop Watch Millisecond Low Data
SSec_10Font	1	Stop Watch Second High Data
SSec_1Font	1	Stop Watch Second Low Data
SMin_10Font	1	Stop Watch Minute High Data
SMin_1Font	1	Stop Watch Minute Low Data
SHour_10Font	1	Stop Watch Hour High Data
SHour_1Font	1	Stop Watch Hour Low Data
COM	1	COM signal mode
rMode	1	System Mode - Clock_ Mode = 1 - Alarm_Mode = 2 - StopWatch-Mode = 3

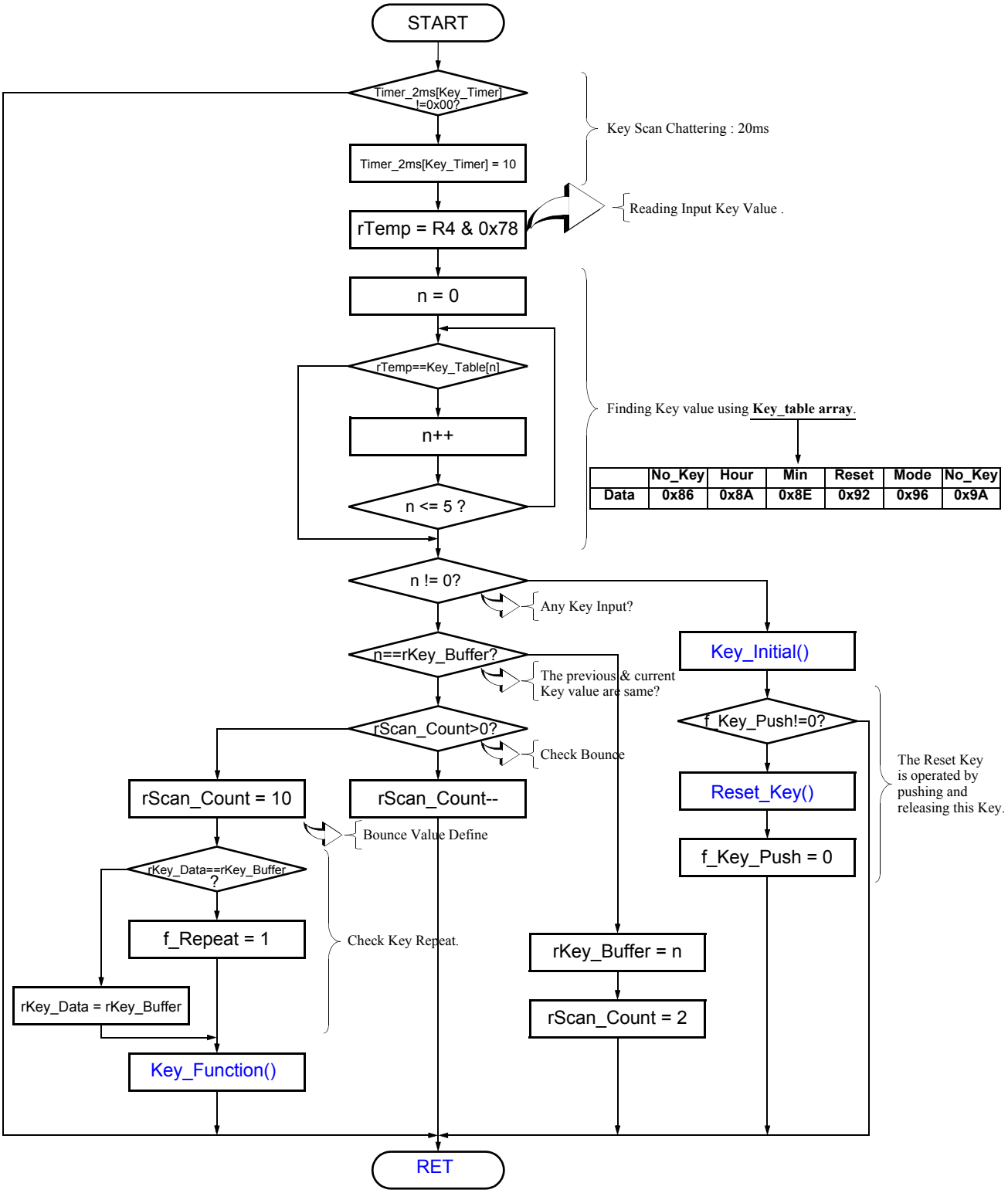
Table 3: Ram Description

Flow Chart

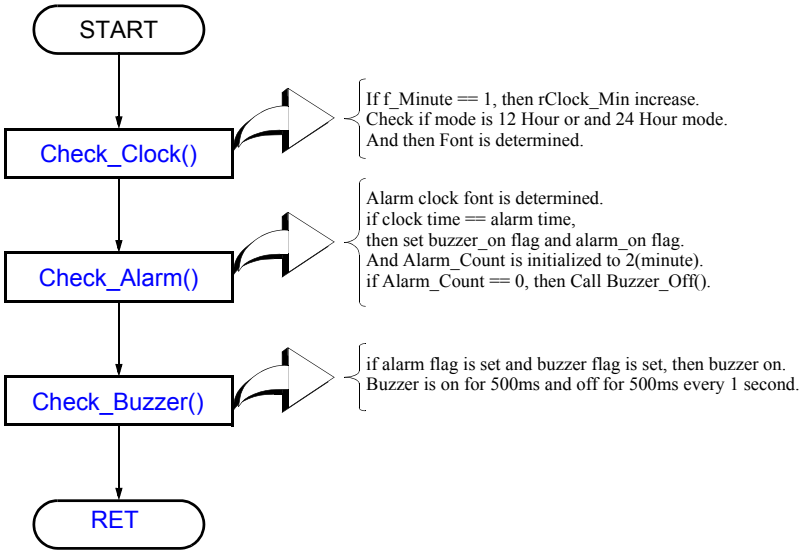
1. main.c



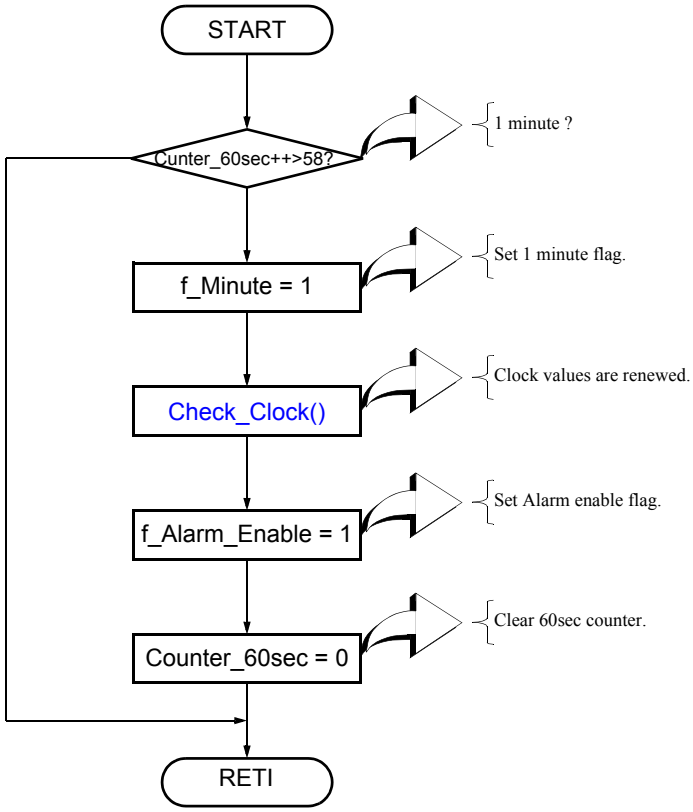
2. void Process_Key()(Key.c)



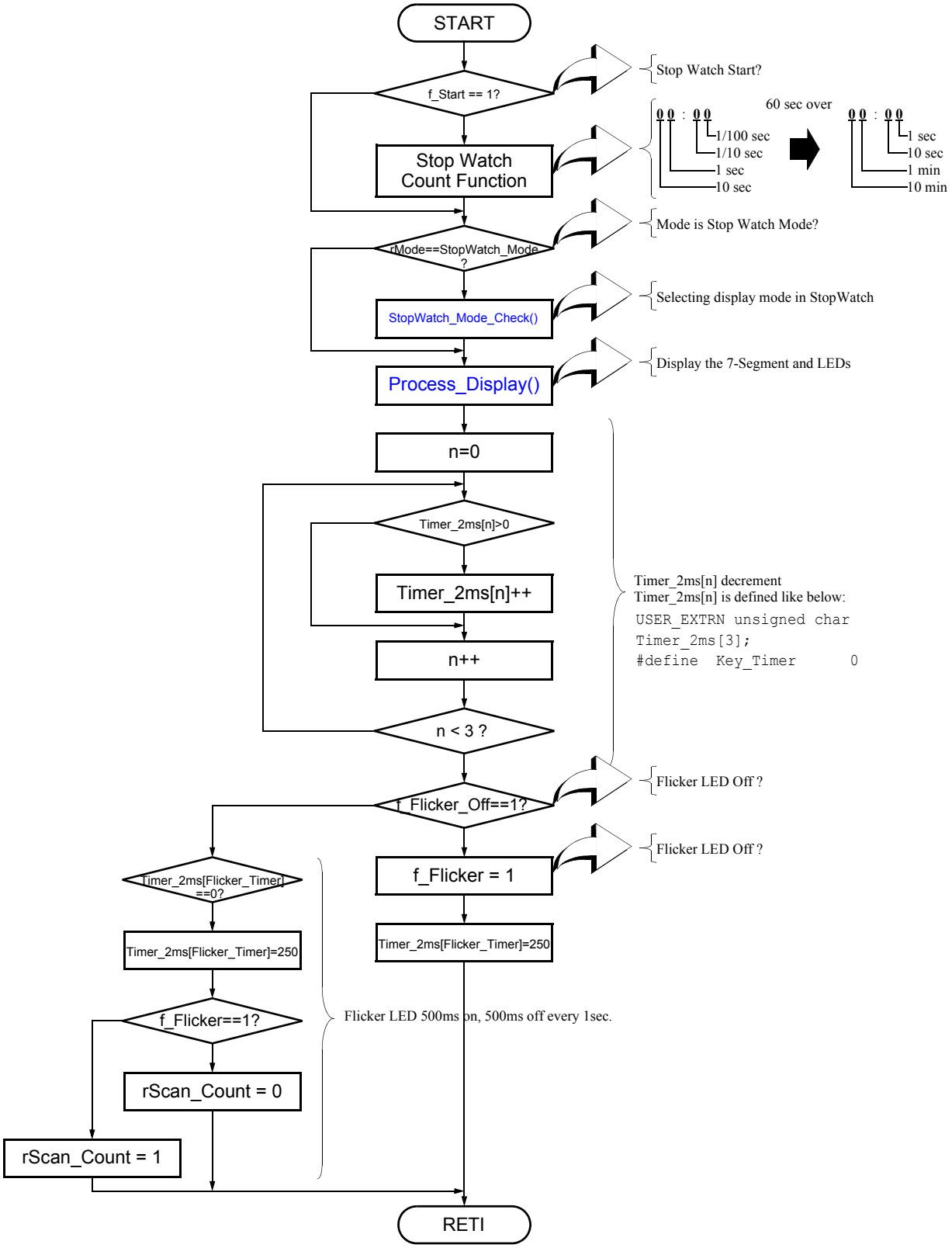
3. void Process_Time(Time.c)



4. void INT3 T4_Int()(Time.c)



5. void INT1 WT_Int()(Time.c)



Source List

main.c

```
#include "hms800.h"
#include "MC80C0448.h"
#define  USER_EXTERN
#include "Variable.h"

void Process_Key();
void Porcess_Time();
void Process_Display();
void Process_Sleep();
void System_Initial();
void Timer_Initial();
void Port_Initial();
void Initial_Value();

//===================================
//    Main Function
//===================================
main()
{
     System_Initial();// System Initialization
     Timer_Initial();// Timer Initialization
     Port_Initial();// Port Initialization

     Initial_Value();// Variables Initialization

     while(1)
     {
          Process_Key();// Key Scan & Key Function
          Process_Time();// Clock & Alarm Check, Buzzer Check
     }
}

//===================================
//    System_Initial Function
//      - Ram page Initialization
//      - ADC / Interrupt Initialization
//===================================
void System_Initial()
{
     DISABLE_INTERRUPT;

     RPR  = 0x00;// Ram Page Initialization
     SSCR = 0x00;// Stop&Sleep Mode Control (0x5Ah --> Stop, 0x0Fh --> Sleep)
     CKCTLR= 0x1E;// Clock Control Register Initialization.
               // BITR clock (=fxin/512) / Watchdog Timer On
     BUZR = 0xBA;// Buzzer Control Register Initialization

     ADCRH= 0x20;// ADC Result Register High Initialization
               // --> 10bit Mode / Clock Source : fxin /8
     ADCM = 0x1A;// ADC Mode Register Initialization
               // --> ADC Enable / Ch 6

     IFR  = 0x00;// Interrupt Flag Register
     IENH = 0x00;// INT3E(7),INT2E(6),INT1E(5),INT0E(4),UART0E(3),UART1E(2),
               // SIOE(1),T0E(0)
     IENL = 0x12;// T1E(7),T2E(6),T3E(5),T4E(4),ADCE(3),WDTE(2),WTE(1),BITE(0)
     IRQH = 0;
     IRQL = 0;
}
```

```
//===================================
//    Timer_Initial Function
//      - Watch Timer : 2ms Timer
//      - Timer 4 : 1sec timer
//===================================
void Timer_Initial()
{
     WDTR = 0xBC;// Watch Dog Timer Register Initialization
     WTMR = 0x8A;// Watch Timer Mode Register Initialization : 2ms Timer
     TM4  = 0x1F;// Timer 4 Control Register Initialization : 1sec Timer
     TDR4L= 0xA1;
     TDR4H= 0x07;
}

//===================================
//    Port_Initial Function
//===================================
void Port_Initial()
{
               // Seg_DP Seg_g_ Seg_b Seg_a __ Seg_f COM0 COM1 COM2
     R0   = 0x00;//    0     0      0     0    __    0    0    0    0
     R0IO = 0xFF;//                          Out
     PU0  = 0x00;//                       No Pull-Up

               // -  -  -  -  __ Buzzer Seg_e Seg_d Seg_c
     R1   = 0x00;// 0 0 0 0 __        0     0     0     0
     R1IO = 0x0F;//     In       __                      Out
     PU1  = 0x00;//       No Pull - Up

               // -  Mode_Key Reset_Key Min_Key  __  Hour_Key  -     -    -
     R4   = 0x00;// 0     0         0        0     __     0      0     0    0
     R4IO = 0x00;//                                 In
     PU4  = 0x78;//No    Yes       Yes       Yes    __    Yes     No   No   No

               // COM3  - LED1 LED0  __  - - - -
     R7   = 0x00;//  0    0  0    0     __  0 0 0 0
     R7IO = 0xB0;// Out  In Out  Out   __    In
     PU7  = 0x00;//             No Pull - Up

               // Secondary Function Selection
     PSR0 = 0x00;//
     PSR1 = 0x04;// R13 port --> Buzzer
     ENABLE_INTERRUPT;
}

//===================================
//    Variables Initialization  Function
//===================================
void Initial_Value()
{
     rMode      = Clock_Mode;
     rClock_Hour= 12;
     rClock_Min = 0;
     rAlarm_Hour= 12;
     rClock_Min = 0;
     f_Alarm_Enable= 0;
     StopWatch_Display_Mode = 5;
     Buzzer_Off();
}
```

Key.c

```
#include "hms800.h"
#include "MC80C0448.h"
#define  USER_EXTERN
#include "Variable.h"

void Key_Initial();
void Key_Function();
void Hour_Key();
void Min_Key();
void Reset_Key();
void Mode_Key();
void Timer_Set();


void Buzzer_On();
void Buzzer_Off();


//===================================
//    Key Value Table
//===================================
unsigned char Key_Table[] __attribute__((section (".text"))) =
{
//     No_Key, Hour_Key, Min_Key, Mode_Key,Reset_Key,No_Key
        0x78,    0x70,     0x68,     0x58,     0x38,   0x78
};

//===================================
//   Process Key Function
//===================================
void Process_Key()
{
     unsigned char n;

     if(Timer_2ms[Key_Timer]!=0x00)       // Key Scanning every 20ms
          return;
     Timer_2ms[Key_Timer] = 10;

     rTemp = R4 & 0x78;                   // Read Key Port

     for(n=0;n<=5;n++)                    // Key Value Check
     {
          if(rTemp == Key_Table[n])
               break;
     }

     if(n)                                // Any Key Input ?
     {
          if(n==rKey_Buffer)              // Key Value is rKey_Buffer ?
          {
               if(rScan_Count)            // Bouncing Counter is not zero?
                    rScan_Count--;
               else                       // Bouncing counter is zero
               {
                    rScan_Count = 10;     // Repeat Counter Setting

                    if(rKey_Data==rKey_Buffer)// Check Repeat
                         f_Repeat = 1;
                    else
                         rKey_Data = rKey_Buffer;
```

```
                    Key_Function();
               }
          }
          else                              // Key Value is not
          {
               rKey_Buffer = n;
               rScan_Count = 2;             // Bouncing Counter Setting
          }
     }
     else                                   // No Key Input
     {
          Key_Initial();
          if(f_Key_Push)                    // Reset_Key is pushed ?
          {
               Reset_Key();                 // Reset_Key Function
               f_Key_Push = 0;

          }
     }
}

//====================================
//  Variables Initialization
//====================================
void Key_Initial()
{
     f_Repeat = 0;
     f_Stop_key = 0;
     rKey_Data = 0;
     rKey_Buffer =  0;

     rSpeed_Mode = 0;
}

//====================================
//  Key Function
//====================================
void Key_Function()
{
     switch(rKey_Data)// Check rKey_Data
     {
          case 1 :
               Hour_Key();
          break;
          case 2 :
               Min_Key();
          break;
          case 3 :
               Reset_Key();
          break;
          case 4 :
               Mode_Key();
               f_Stop_key = 1;
          break;
          default :
          break;
     }
}

//====================================
//  Key Speed Table
//====================================
```

```
uchar Key_Speed_Table[] __attribute__((section (".text"))) =
{
     0,22, 6, 6, 0
};

//==================================
// Repeat Check Function
//==================================
uchar Repeat_Check(uchar SPEED_ID)
{
     if(f_Stop_key)
          return 1;
     else if (rSpeed_Mode != SPEED_ID)    // New Key input ?
     {
          rSpeed_Mode  = SPEED_ID;        // Save the SPEED_ID of new Key
          rSpeed_Count = Key_Speed_Table[rSpeed_Mode];// Set the Counter using Key_Speed_Table
     }
     else if(rSpeed_Count--)
          return 1;
     else
     {
          rSpeed_Count = 1;               // Set the Counter for implementing the Key pushed every 200ms
          if (rSpeed_Mode == SPEED_Reset) // if New Key is Reset_Key?
          {
               if(f_Con)                  // 12:00 Hour Mode ?
                    f_Con = 0;
               else                       // 24:00 Hour Mode
                    f_Con = 1;
               f_Stop_key = 1;

               f_Key_Push = 0;
               return 1;                  // return 1 for stopping Key input
          }
     }

     return 0;
}

//==================================
// Hour Key  Function
//==================================
void Hour_Key()
{
     if(rMode==StopWatch_Mode)            // StopWatch_Mode?
     {
          if(f_Repeat)                    // Repeat is not permitted.
               return;
          if(f_Start)                     // Start flag is set?
          {
               f_Start = 0;               // Clear the flag
               StopWatch_Display_Mode = 3;// Stop the StopWatch
          }
          else                            // flag is clean?
          {
               f_Start = 1;               // set the flag
               Timer_2ms[Flicker_Timer] = 250;
               if(f_Hold)                 // Hold flag is set?
                    StopWatch_Display_Mode = 2;// Start the StopWatch but not refreshed.
               else
                    StopWatch_Display_Mode = 1;// Start the StopWatch and refreshed
          }
     }
```

```
    else                                // rMode is not StopWatch_Mode.
    {
        if(Repeat_Check(SPEED_Hour))    // Repeat Check
            return;
        if(rMode==Clock_Mode)           // Clock Mode?
        {
            rClock_Hour++;              // Increase Hour
            if(rClock_Hour>23)          // Increased Hour is over 23?
                rClock_Hour = 0;        // Initialize Hour

            f_Alarm_On = 0;             // Clear variables with Buzzer.
            Buzzer_Off();
            Alarm_Count = 0;
        }
        else                            // Alarm Mode?
        {
            rAlarm_Hour++;              // Incerese Alarm Hour
            if(rAlarm_Hour>23)          // Increased Alarm Hour is over 23?
                rAlarm_Hour = 0;        // Initialize Alarm Hour
        }
    }
    Check_Clock();                      // Check Clock
}

//==================================
// Minute Key  Function
//==================================
void Min_Key()
{
    if(rMode==StopWatch_Mode)// StopWatch_Mode ?
    {
        if(f_Repeat)                    // Repeat is not permitted.
            return;
        if(f_Start)                     // Start flag is set?
        {
            if(f_Hold)                  // Hold flag is set?
            {
                f_Hold = 0;
                StopWatch_Display_Mode = 1;// Start the StopWatch and refreshed
            }

            else
            {
                f_Hold = 1;
                StopWatch_Display_Mode = 2; // LED display the value when Key was pushed
                                            // and Stopwatch is played continuously.
            }
        }

        else                            // Start flag is clean?
        {

            if(f_Hold)                  // Hold flag is set?
            {
                f_Hold = 0;
                StopWatch_Display_Mode = 4;// LED display the last value of StopWatch.
            }
            else                        // Hold flag is clean?
            {                           // Initialized the StopWatch to zero
                rStopWatch_100ms= 0;
                rStopWatch_10ms= 0;
                rStopWatch_Sec = 0;
```

```
                    rStopWatch_Sec = 0;
                    rStopWatch_Min = 0;
                    rStopWatch_Min = 0;
                    rStopWatch_Hour = 0;
                    rStopWatch_Hour = 0 ;

                    StopWatch_Display_Mode = 5;
               }
          }
     }

     else                                // rMode is not Stopwatch mode?
     {
          if(Repeat_Check(SPEED_Min))
               return;
          if(rMode==Clock_Mode)          // Clock Mode?
          {
               if(f_Alarm_On)            // Alarm On?
               {
                    if(Alarm_Count>0)    // Alarm Counter is not zero.
                         Alarm_Count--;  // Alarm Counter value is decresed.
               }

               rClock_Min++;             // Increase Minute
               if(rClock_Min>59)         // Increased Miute is over 59?
                    rClock_Min = 0;      // Initialized the Minute.
          }
          else                           // Alarm Mode?
          {
               rAlarm_Min++;             // Increase Alarm Minute
               if(rAlarm_Min>59)         // Increased Alrm Miute is over 59?
                    rAlarm_Min = 0;      // Initialized the Alarm Minute.
          }
     }
}

//==================================
// Reset Key  Function
//==================================
void Reset_Key()
{
     if(rMode==StopWatch_Mode)           // Reset Key is not used in Stopwatch_Mode
          return;

     if(Repeat_Check(SPEED_Reset))
          return;
     if(rMode==Clock_Mode)               // Clock Mode?
     {
          if(f_Alarm_On)                 // Alarm is On?
          {
               Buzzer_Off();             // Alarm Off
               Alarm_Count = 0;
               f_Alarm_Enable = 0;
               return;
          }
          else if(f_Key_Push)            // Key Push flag is set?
          {
               if (rClock_Min > 29)
               {
                    rClock_Hour++;
                    if (rClock_Hour > 23)
                         rClock_Hour = 0;
```

```
                }

                rClock_Min = 0;

                Timer_Set();
                Process_Display();
           }
      }

      else                                  // Alarm Mode?
      {
           if (rAlarm_Min > 29)
           {
                rAlarm_Hour++;
                if (rAlarm_Hour > 23)
                     rAlarm_Hour = 0;
           }

           rAlarm_Min = 0;
      }
      f_Key_Push = 1;
}

//==================================
// Mode Key  Function
//==================================
void Mode_Key()
{
      if(f_Stop_key)
           return;

      if(rMode++>2)                         // Mode is over 2?
           rMode = 1;                       // Mode is rotated.
      switch(rMode)
      {
           case 1 :
                f_Flicker_Off= 0;
           break;
           case 2 :
                f_Flicker_Off= 1;
           break;
           case 3 :
                f_Flicker_Off= 1;
           break;
      }
}

//==================================
// Timer Set  Function
//==================================
void Timer_Set()
{
//    TM4   = 0x00;
      TM4   = 0x1F;                         // 1minute Timer
      TDR4L= 0xA1;
      TDR4H= 0x07;

      f_Flicker = 1;
      Timer_2ms[Flicker_Timer] = 250;
}
```

Time.c

```
#include "hms800.h"
#include "MC80C0448.h"
#define  USER_EXTERN
#include "Variable.h"

void Check_Clock();
void Check_Alarm();
void Check_Buzzer();
void Buzzer_On();
void Buzzer_Off();
void StopWatch_Mode_Check();
void SEG_Display();

//===================================
//    Process Time Function
//===================================
void Process_Time()
{
     Check_Clock();
     Check_Alarm();
     Check_Buzzer();
}

//===================================
//    Check Clock  Function
//===================================
void Check_Clock()
{
     if(f_Minute)                         // minute flag set?
     {
          f_Minute = 0;                   // clear flag
          if(f_Alarm_On)                  // Alarm is on?
          {
               if(Alarm_Count>0)          // Alarm Counter is not zero?
                    Alarm_Count--;        // Decrease Alarm Counter
          }
          if(rClock_Min++>59)
          {
               rClock_Min = 0;
               if(rClock_Hour++>23)
                    rClock_Hour = 0;
          }
     }
     if(f_Con)                            // 12:00 Hour Mode?
     {
          if(rClock_Hour==0)              // rClock_Hour is zero?
          {                               // then display 12:00
               Hour_10Font = 1;
               Hour_1Font = 2;
          }
          else if(rClock_Hour>12)         // rClock_Hour > 12
          {
               Hour_10Font = (rClock_Hour-12)/10;
               Hour_1Font = (rClock_Hour-12)%10;
          }
          else
          {
               Hour_10Font = rClock_Hour/10;
               Hour_1Font = rClock_Hour%10;
          }
     }
```

```
    else                                // 24:00 Hour Mode
    {
        Hour_10Font = rClock_Hour/10;
        Hour_1Font = rClock_Hour%10;
    }

    Min_10Font = rClock_Min/10;
    Min_1Font = rClock_Min%10;
}


//==================================
//    Check Alarm  Function
//==================================
void Check_Alarm()
{
    AHour_10Font = rAlarm_Hour/10;
    AHour_1Font = rAlarm_Hour%10;

    AMin_10Font = rAlarm_Min/10;
    AMin_1Font = rAlarm_Min%10;

    if(rMode==Alarm_Mode)               // Alarm Mode?
    {                                   // then initialized variables with Buzzer
        f_Alarm_Enable = 0;
        Buzzer_Off();
        Alarm_Count = 0;
        return;
    }

    if(f_Alarm_On==0)                   // Alarm is not On?
    {
        if(rClock_Hour==rAlarm_Hour)
        {
            if(rClock_Min==rAlarm_Min)
            {
                if(f_Alarm_Enable)      // Alarm is enable?
                {                       // then set variables with Buzzer
                    f_Alarm_On = 1;
                    f_Buzzer_On = 1;
                    Alarm_Count = 2;
                    Timer_2ms[Buzzer_Timer] = 250;
                }
                else                    // Alarm is not enable?
                {
                    f_Alarm_On = 0; // Clear Alarm_On flag
                    Alarm_Count = 0;// Clear Alarm Counter
                }
            }
        }
    }

    else if(Alarm_Count<1)              // Alarm Counter is zero?
    {                                   // then clear variables with Buzzer
        f_Buzzer_On = 0;
        f_Alarm_On = 0;
        Buzzer_Off();                   // f_Buzzor_On = 0;
        Timer_2ms[Buzzer_Timer] = 250;
    }
}
```

```
//==================================
//    Check Buzzer Function
//==================================
void Check_Buzzer()
{
     if(f_Alarm_On)                      // Alarm is On?
     {
          if(Timer_2ms[Buzzer_Timer]==0)
          {
               Timer_2ms[Buzzer_Timer] = 250;
               if(f_Buzzer_On)           // 500ms Off
               {
                    f_Buzzer_On = 0;
                    Buzzer_Off();
               }
               else                      // 500ms On
               {
                    f_Buzzer_On = 1;
                    Buzzer_On();
               }
          }
     }
}

//==================================
//    Buzzer On Function
//==================================
void Buzzer_On()
{
     BUZO = 1;
     R1IO = 0x0F;
}

//==================================
//    Buzzer Off Function
//==================================
void Buzzer_Off()
{
     BUZO  = 0;
     R1IO = 0x07;
}

//==================================
//    StopWatch_Mode Check Function
//==================================
void StopWatch_Mode_Check()
{
     switch(StopWatch_Display_Mode)
     {
          case 1 :                       // Start the StopWatch and refreshed
               f_Flicker_Off= 0;

               SMs_100Font = rStopWatch_100ms%10;
               SMs_10Font = rStopWatch_10ms%10;
               SSec_10Font= rStopWatch_Sec/10 ;
               SSec_1Font = rStopWatch_Sec%10;
               SMin_10Font  = rStopWatch_Min/10;
               SMin_1Font = rStopWatch_Min%10 ;
               SHour_10Font = rStopWatch_Hour/10;
               SHour_1Font =rStopWatch_Hour%10 ;

          break;
```

```
          case 2 :                        // LED display the value when Key was pushed
               f_Flicker_Off= 0;          // and Stopwatch is played continuously.
          break;

          case 3 :                        // Stop the StopWatch
               f_Flicker_Off= 1;
          break;

          case 4 :                        // LED display the last value of StopWatch.
               f_Flicker_Off= 1;

               SMs_100Font = rStopWatch_100ms%10;
               SMs_10Font = rStopWatch_10ms%10;
               SSec_10Font= rStopWatch_Sec/10 ;
               SSec_1Font = rStopWatch_Sec%10;
               SMin_10Font  = rStopWatch_Min/10;
               SMin_1Font = rStopWatch_Min%10 ;
               SHour_10Font = rStopWatch_Hour/10;
               SHour_1Font = rStopWatch_Hour%10 ;

          break;

          case 5 :                        // Initialized the StopWatch to zero
               SMs_100Font = 0;
               SMs_10Font  = 0;
               SSec_10Font = 0;
               SSec_1Font  = 0;
               SMin_10Font = 0;
               SMin_1Font  = 0;
               SHour_10Font = 0;
               SHour_1Font  = 0;

               f_Flicker_Off = 1;
               f_Over_60s = 0;
               f_Over_60m = 0;
          break;
     }

}

//===================================
//    Timer 4 Interrupt Function
//      - Interrupt Period is 1sec
//===================================
void INT3 T4_Int()
{
     if(Counter_60sec++>58)              // Counter is over 59?
     {
          f_Minute = 1;                  // minute flag set
          Check_Clock();
          f_Alarm_Enable = 1;
          Counter_60sec=0;
     }
}

//===================================
//    Watch Timer Intuerrupt Function
//      - Interrupt Period is 2ms
//===================================
void INT1 WT_INT()
{
     unsigned char n;
```

```
    if(f_Start)                         // StopWatch is started?
    {                                   // then counting the time
        if(rStopWatch_2ms++>3)
        {
            rStopWatch_2ms =0;
            if(rStopWatch_10ms++>8)
            {
                rStopWatch_10ms = 0;
                if(rStopWatch_100ms++>8)
                {
                    rStopWatch_100ms = 0;
                    if(rStopWatch_Sec++>58)
                    {
                        rStopWatch_Sec = 0;
                        f_Over_60s = 1;
                        if(rStopWatch_Min++>58)
                        {
                                    f_Over_60s = 0;
                                    f_Over_60m = 1;
                                    rStopWatch_Min = 0;
                                    if(rStopWatch_Hour++>98)
                                    {
                                    f_Over_60m = 0;
                                    rStopWatch_Hour = 0;
                                    }
                        }
                    }
                }
            }
        }
    }

    if(rMode==StopWatch_Mode)
        StopWatch_Mode_Check();


    //SEG_Display();
    Process_Display();// Display Function

    for(n=0;n<3;n++)// Decrease 2ms Timer array
    {
        if(Timer_2ms[n])
            Timer_2ms[n]--;
    }

    if(f_Flicker_Off)
    {
        f_Flicker = 1;
        Timer_2ms[Flicker_Timer] = 250;
    }
    else if(Timer_2ms[Flicker_Timer]==0)
    {
        Timer_2ms[Flicker_Timer] = 250;

        if(f_Flicker)
            f_Flicker = 0;// 500ms flicker Off
        else
            f_Flicker = 1;// 500ms flicker On
    }
}
```

Display.c

```
#include "hms800.h"
#include "MC80C0448.h"
#define  USER_EXTERN
#include "Variable.h"

void Digit_Display(unsigned char n);

//                ____
//        f    / a  / b
//            /____/
//       e  /  g / c
//         /____/
//           d       O dp
//

//===================================
// LED Initialization Function
//===================================
void LED_Clear()
{
     Seg_a= 0;
     Seg_b= 0;
     Seg_c= 0;
     Seg_d= 0;
     Seg_e= 0;
     Seg_f= 0;
     Seg_g= 0;
     Seg_DP= 0;
}

//===================================
// Digit Display  Function
//===================================

void Digit_Display(unsigned char n)
{
     //*******************************
     // Cathode Common
     //*******************************
     LED_Clear();

     switch(n)
     {
          case 0 :
               Seg_a= 1;
               Seg_b= 1;
               Seg_c= 1;
               Seg_d= 1;
               Seg_e= 1;
               Seg_f= 1;
               Seg_g= 0;
               Seg_DP= 0;
          break;

          case 1 :
               Seg_a= 0;
               Seg_b= 1;
               Seg_c= 1;
               Seg_d= 0;
               Seg_e= 0;
               Seg_f= 0;
```

```
        Seg_g= 0;
        Seg_DP= 0;
    break;

    case 2 :
        Seg_a= 1;
        Seg_b= 1;
        Seg_c= 0;
        Seg_d= 1;
        Seg_e= 1;
        Seg_f= 0;
        Seg_g= 1;
        Seg_DP= 0;
    break;

    case 3 :
        Seg_a= 1;
        Seg_b= 1;
        Seg_c= 1;
        Seg_d= 1;
        Seg_e= 0;
        Seg_f= 0;
        Seg_g= 1;
        Seg_DP= 0;
    break;

    case 4 :
        Seg_a= 0;
        Seg_b= 1;
        Seg_c= 1;
        Seg_d= 0;
        Seg_e= 0;
        Seg_f= 1;
        Seg_g= 1;
        Seg_DP= 0;
    break;

    case 5 :
        Seg_a= 1;
        Seg_b= 0;
        Seg_c= 1;
        Seg_d= 1;
        Seg_e= 0;
        Seg_f= 1;
        Seg_g= 1;
        Seg_DP= 0;
    break;

    case 6 :
        Seg_a= 1;
        Seg_b= 0;
        Seg_c= 1;
        Seg_d= 1;
        Seg_e= 1;
        Seg_f= 1;
        Seg_g= 1;
        Seg_DP= 0;
    break;

    case 7 :
        Seg_a= 1;
        Seg_b= 1;
```

```
                Seg_c= 1;
                Seg_d= 0;
                Seg_e= 0;
                Seg_f= 1;
                Seg_g= 0;
                Seg_DP= 0;
            break;

            case 8 :
                Seg_a= 1;
                Seg_b= 1;
                Seg_c= 1;
                Seg_d= 1;
                Seg_e= 1;
                Seg_f= 1;
                Seg_g= 1;
                Seg_DP= 0;
            break;

            case 9 :
                Seg_a= 1;
                Seg_b= 1;
                Seg_c= 1;
                Seg_d= 1;
                Seg_e= 0;
                Seg_f= 1;
                Seg_g= 1;
                Seg_DP= 0;
            break;

        }
/*
        //*******************************
        // Anode Common
        //*******************************
        switch(n)
        {
            case 0 :
                R04 = 0;
                R05 = 0;
                R10 = 0;
                R11 = 0;
                R12 = 0;
                R03 = 0;
                R06 = 1;
                R07 = 1;
            break;

            case 1 :
                R04 = 1;
                R05 = 0;
                R10 = 0;
                R11 = 1;
                R12 = 1;
                R03 = 1;
                R06 = 1;
                R07 = 1;
            break;

            case 2 :
                R04 = 0;
                R05 = 0;
```

```
        R10 = 1;
        R11 = 0;
        R12 = 0;
        R03 = 1;
        R06 = 0;
        R07 = 1;
    break;

    case 3 :
        R04 = 0;
        R05 = 0;
        R10 = 0;
        R11 = 0;
        R12 = 1;
        R03 = 1;
        R06 = 0;
        R07 = 1;
    break;

    case 4 :
        R04 = 1;
        R05 = 0;
        R10 = 0;
        R11 = 1;
        R12 = 1;
        R03 = 0;
        R06 = 0;
        R07 = 1;
    break;

    case 5 :
        R04 = 0;
        R05 = 1;
        R10 = 0;
        R11 = 0;
        R12 = 0;
        R03 = 1;
        R06 = 0;
        R07 = 1;
    break;

    case 6 :
        R04 = 0;
        R05 = 1;
        R10 = 0;
        R11 = 0;
        R12 = 0;
        R03 = 0;
        R06 = 0;
        R07 = 1;
    break;

    case 7 :
        R04 = 0;
        R05 = 0;
        R10 = 0;
        R11 = 1;
        R12 = 1;
        R03 = 0;
        R06 = 1;
        R07 = 1;
    break;
```

```
            case 8 :
                  R04 = 0;
                  R05 = 0;
                  R10 = 0;
                  R11 = 0;
                  R12 = 0;
                  R03 = 0;
                  R06 = 0;
                  R07 = 1;
            break;

            case 9 :
                  R04 = 0;
                  R05 = 0;
                  R10 = 0;
                  R11 = 0;
                  R12 = 1;
                  R03 = 0;
                  R06 = 0;
                  R07 = 1;
            break;

      }
*/
}

//==================================
// Process Display  Function
//==================================
void Process_Display()
{
      //*************************************
      // Cathode Common
      //*************************************
      switch(COM)
      {
            case 0 :
                  COM0 = 0x00;
                  COM1 = 0x01;
                  COM2 = 0x01;
                  COM3 = 0x01;
                  if(rMode==Clock_Mode)
                  {
                        if(Hour_10Font==0)
                              LED_Clear();
                        else
                              Digit_Display(Hour_10Font);
                  }
                  else if(rMode==Alarm_Mode)
                  {
                        if(AHour_10Font==0)
                              LED_Clear();
                        else
                              Digit_Display(AHour_10Font);

                  }
                  else if(rMode==StopWatch_Mode)
                  {
                        if(f_Over_60s)
                              Digit_Display(SMin_10Font);
                        else if(f_Over_60m)
```

```
                Digit_Display(SHour_10Font);
            else
                Digit_Display(SSec_10Font);
        }
    break;

    case 1 :
        COM0 = 0x01;
        COM1 = 0x00;
        COM2 = 0x01;
        COM3 = 0x01;
        if(rMode==Clock_Mode)
            Digit_Display(Hour_1Font);
        else if(rMode==Alarm_Mode)
            Digit_Display(AHour_1Font);
        else if(rMode==StopWatch_Mode)
        {
            if(f_Over_60s)
                Digit_Display(SMin_1Font);
            else if(f_Over_60m)
                Digit_Display(SHour_1Font);
            else
                Digit_Display(SSec_1Font);
        }
    break;

    case 2 :
        COM0 = 0x01;
        COM1 = 0x01;
        COM2 = 0x00;
        COM3 = 0x01;
        if(rMode==Clock_Mode)
            Digit_Display(Min_10Font);
        else if(rMode==Alarm_Mode)
            Digit_Display(AMin_10Font);
        else if(rMode==StopWatch_Mode)
        {
            if(f_Over_60s)
                Digit_Display(SSec_10Font);
            else if(f_Over_60m)
                Digit_Display(SMin_10Font);
            else
                Digit_Display(SMs_100Font);
        }
    break;

    case 3 :
        COM0 = 0x01;
        COM1 = 0x01;
        COM2 = 0x01;
        COM3 = 0x00;
        if(rMode==Clock_Mode)
            Digit_Display(Min_1Font);
        else if(rMode==Alarm_Mode)
            Digit_Display(AMin_1Font);
        else if(rMode==StopWatch_Mode)
        {
            if(f_Over_60s)
                Digit_Display(SSec_1Font);
            else if(f_Over_60m)
                Digit_Display(SMin_1Font);
            else
```

```
                        Digit_Display(SMs_10Font);
                }
            break;

    }
/*
    //**************************************
    // Anode Common
    //**************************************
    switch(COM)
    {
        case 0 :
            COM0 = 0x01;
            COM1 = 0x00;
            COM2 = 0x00;
            COM3 = 0x00;
            if(rMode==Clock_Mode)
            {
                if(Hour_10Font==0)
                    LED_Clear();
                else
                    Digit_Display(Hour_10Font);
            }
            else if(rMode==Alarm_Mode)
            {
                if(AHour_10Font==0)
                    LED_Clear();
                else
                    Digit_Display(AHour_10Font);

            }
            else if(rMode==StopWatch_Mode)
            {
                if(f_Over_60s)
                    Digit_Display(SMin_10Font);
                else if(f_Over_60m)
                    Digit_Display(SHour_10Font);
                else
                    Digit_Display(SSec_10Font);
            }
        break;

        case 1 :
            COM0 = 0x00;
            COM1 = 0x01;
            COM2 = 0x00;
            COM3 = 0x00;
            if(rMode==Clock_Mode)
                Digit_Display(Hour_1Font);
            else if(rMode==Alarm_Mode)
                Digit_Display(AHour_1Font);
            else if(rMode==StopWatch_Mode)
            {
                if(f_Over_60s)
                    Digit_Display(SMin_1Font);
                else if(f_Over_60m)
                    Digit_Display(SHour_1Font);
                else
                    Digit_Display(SSec_1Font);
            }
        break;
```

```
            case 2 :
                  COM0 = 0x00;
                  COM1 = 0x00;
                  COM2 = 0x01;
                  COM3 = 0x00;
                  if(rMode==Clock_Mode)
                        Digit_Display(Min_10Font);
                  else if(rMode==Alarm_Mode)
                        Digit_Display(AMin_10Font);
                  else if(rMode==StopWatch_Mode)
                  {
                        if(f_Over_60s)
                              Digit_Display(SSec_10Font);
                        else if(f_Over_60m)
                              Digit_Display(SMin_10Font);
                        else
                              Digit_Display(SMs_100Font);
                  }
            break;

            case 3 :
                  COM0 = 0x00;
                  COM1 = 0x00;
                  COM2 = 0x00;
                  COM3 = 0x01;
                  if(rMode==Clock_Mode)
                        Digit_Display(Min_1Font);
                  else if(rMode==Alarm_Mode)
                        Digit_Display(AMin_1Font);
                  else if(rMode==StopWatch_Mode)
                  {
                        if(f_Over_60s)
                              Digit_Display(SSec_1Font);
                        else if(f_Over_60m)
                              Digit_Display(SMin_1Font);
                        else
                              Digit_Display(SMs_10Font);
                  }
            break;
      }
*/
      if(COM++>3)
            COM = 0;
}
```