Hardware Design

The hardware implementation should include the following features and elements:

- displaying system messages (“DONE”, “SENT”, “RECD”, “END”) as well as received data ranging from 0 to 255 in decimal (0x00 to 0xFF) via a bank of four-digit 7-segment displays.
- setting up external interrupt pins (P3.2 and P3.3) on the SiliconLab C8051F120DK MCU to initiate data transmission when the board is configured as a Master and to display the received data when the board is configured as a Slave

<u>As a Transmitter (Master):</u>

- Invoke an external interrupt service routine to write 0x*00* to 0x*FF* to a RAM block of 256 bytes.
- Display “DONE” on seven-segment displays after data written to RAM.
- Invoke another interrupt service routine to specify the address of the designated receiver(s) within the network. *Note that the 9th transmit bit should be set in SCON (TB8 = 1) in order to distinguish the address byte from a data byte.*
- Start transmission through the SBUF SFR.
- Display “SENT” on seven-segment displays after data have been transmitted.

<u>As a Receiver (Slave):</u>

- Enable multiprocessor communications by setting the SM2 bit in SCON SFR such that the serial interrupt is only invoked if the received 9th bit (RB8 in SCON) is set.
- Determine if the slave is among the target receiver list. If so, the slave will start receiving data and save them to a RAM block.
- SM2 bit must be cleared during data reception.
- Display “RECD” on seven-segment displays after data reception.
- Invoke an interrupt service routine to display the received data one byte a time with an interval of one second.
- Display “END” on seven-segment displays after received data displayed.

<u>Typical setup requirements for multiprocessor UART communication</u>:

- Configure the Serial Port Control Register (SCON) to operate in multiprocessor variable mode (Mode 3)
- *Serial Interrupt (ES0), Timer_0 Overflow Interrupt (TF0)* and *External Interrupt_0* (IE0) should be activated
- Configure serial transmission at the desired baud rate by setting Timer_1 into 8-bit auto-reload mode. *Note that the reload value in TH1 SFR must be carefully calculated based on the target crystal frequency.*
- The SMOD bit (bit 7 in PCON) can be set to double the existing baud rate.
- IE, IP, TCON, TMOD, and SCON SFRs should be properly configured upon initialization

Receiver (slave) Configuration Code that allows multiprocessor communications via universal asynchronous receiver and transmitter (UART)

```
// AUTH: Nikolay A. Atanasov
// DATE: 08 NOV 2006
//
// Engineering Department
// Trinity College
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f120.h>                  // SFR declarations

//-----------------------------------------------------------------------
// 16-bit SFR Definitions
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------
#define ADDRESS   0x05                                  // Slave Address
#define DATA_SIZE 256                                   // The size of the transmitted data

// Letter codes for a 7 Segment display
#define R 0x08
#define E 0x30
#define C 0x72
```

```
#define D 0x42
#define N 0x6A

// Number codes for an active-low 7 Segment display
#define ZERO  0x01
#define ONE   0x4F
#define TWO   0x12
#define THREE 0x06
#define FOUR  0x4C
#define FIVE  0x24
#define SIX   0x60
#define SEVEN 0x0F
#define EIGHT 0x00
#define NINE  0x0C
#define OFF   0xFF

sbit LED = P1^6;                              // green LED: '1' = ON; '0' = OFF
sbit MSEL1 = P1^5;                            // Multiplexer Select bits
sbit MSEL0 = P1^4;

//-----------------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------------

unsigned char xdata ram_block[DATA_SIZE];   // Predefined RAM block to store DATA
short count = 0;                              // Counts the number of bytes received
short interrupt_count = 0;                    // Counts the number of Timer 0 overflows
short refresher = 0;                          // Remembers which digit should be refreshed next
bit RECD_flag = 0;                                                    // Received flag
bit END_flag = 0;                                                     // End flag

unsigned char digit_one = OFF;                // Holds the binary code representation for digit one
unsigned char digit_two = OFF;                // Holds the binary code representation for digit two
unsigned char digit_three = OFF;              // Holds the binary code representation for digit three


//-----------------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------------

// Support Subroutines
unsigned char BCD_7SEG(unsigned char digit);
void Receive_Toggle(void);              // Toggle Receive Mode ON or OFF
void Toggle_T0(void);                   // Display Data stored in RAM_BLOCK sequentially
void clear_display(void);               // Clears the display
void Display(void);

// Interrupt Service Routines
void EX0_ISR(void);
void Timer0_ISR (void);                 // Timer 0 Overflow Interrupt Service Routine
void ES_ISR (void);                     // Serial Interrupt Service Routine
```

```
// Initialization Subroutines
void Init_Device(void);
void Timer_Init(void);
void UART_Init(void);                    // Set up Serial Port Control Register
void Port_IO_Init(void);
void Oscillator_Init(void);
void Interrupts_Init(void);              // Set up Serial Interrupt
void Address_Init(void);
void LED_Init(void);


//-----------------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------------
void main (void)
{
        // Receiver:
        // ES_ISR() is called only when RI = 1
        // when a byte is received RI = 1 iff RB8(9th) = 1 AND SM2 = 1
        // address byte: xxxxxxxx1 (9th) -> If being addressed set SM2 = 0 to receive data!
        // data byte: xxxxxxxx0 (9th) -> Does not interrupt any slave when SM2 = 1

        // disable watchdog timer
        WDTCN = 0xde;
        WDTCN = 0xad;

        SFRPAGE = CONFIG_PAGE;          // Switch to configuration page

        // Configure UART to allow multiprocessor communication at 9600 baud rate
        // 1. Configure SCON to operate in Mode 3 (multiprocessor variable mode)
        // 2. Configure Baud Rate by setting up Timer 1
        Init_Device();

        EA = 1;                                         // Enable global interrupts

        SFRPAGE = LEGACY_PAGE;                  // Page to sit in for now

        while(1);                                                       // Waiting loop
}

//----------------------------------------------------------------------------
// Support Subroutines
//----------------------------------------------------------------------------

//-----------------------------------------------------------------------------------
// BCD_7SEG
//-----------------------------------------------------------------------------------
//
// @param:      argument: a decimal number digit from 0 to 9
//                      return:    a binary code representation of digit
//                                                      for an active low 7 Segment Display
//
unsigned char BCD_7SEG(unsigned char digit)
{
        unsigned char result;
        switch(digit)
        {
                case 0:
                                result = ZERO;
                                break;
                case 1:
                                result = ONE;
                                break;
                case 2:
                                result = TWO;
                                break;
                case 3:
                                result = THREE;
                                break;
                case 4:
                                result = FOUR;
                                break;
                case 5:
                                result = FIVE;
                                break;
                case 6:
                                result = SIX;
                                break;
                case 7:
                                result = SEVEN;
                                break;
                case 8:
                                result = EIGHT;
                                break;
                case 9:
                                result = NINE;
                                break;
                default:
                                result = OFF;
        }
        return result;
}// BCD_7SEG()


//------------------------------------------------------------------------------------
// Receive_Toggle
//------------------------------------------------------------------------------------
//
// Turn Slave Receiving On or Off by toggling the SM2 bit in the SCON register
//
void Receive_Toggle()
```

```
{
        char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

        SFRPAGE = UART0_PAGE;
        SM20 ^= 1;                                                  // toggle the state of SM20

        SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}//Receive_Init()

//---------------------------------------------------------------------------------
// Toggle_T0
//---------------------------------------------------------------------------------
//
// Toggles Timer 0 between Running and Off
//
void Toggle_T0(void)
{
        char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

        SFRPAGE = TIMER01_PAGE;

        // Configure TCON register
        TR0 ^= 1;                   // Toggle Timer 0 run control

        // Configure IE register
        ET0 ^= 1;                                                   // Toggle Timer 0 overflow interrupt

        SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}// Toggle_T0()

//---------------------------------------------------------------------------------
// clear_display
//---------------------------------------------------------------------------------
//
// Resets the seven segment display
//
void clear_display(void)
{
        char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

        SFRPAGE = TIMER01_PAGE;

        // Configure TCON register
        TR0 &= 0;                   // Turn off Timer 0 run control

        // Configure IE register
        ET0 &= 0;                                           // Turn off Timer 0 overflow interrupt

        SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page

        RECD_flag = 0;
```

```
    END_flag = 0;

    P2 = OFF;
}//clear_display()

//----------------------------------------------------------------------------------
// Display
//----------------------------------------------------------------------------------
//
// Refreshes the display with the appropriate value for the current mode
//
void Display(void)
{
    switch(refresher)
    {
        case 0:                                         // MSB
                MSEL1 = 0;
                MSEL0 = 0;
                if(RECD_flag)
                    P2 = R;
                else
                    P2 = OFF;

                refresher++;
                break;
        case 1:
                MSEL1 = 0;
                MSEL0 = 1;
                if(RECD_flag || END_flag)
                    P2 = E;
                else
                    P2 = digit_three;
                refresher++;
                break;
        case 2:
                MSEL1 = 1;
                MSEL0 = 0;
                if(RECD_flag)
                    P2 = C;
                else if (END_flag)
                    P2 = N;
                else
                    P2 = digit_two;

                refresher++;
                break;
        case 3:
                MSEL1 = 1;
                MSEL0 = 1;
                if(RECD_flag || END_flag)
                    P2 = D;
```

```
                                        else
                                                P2 = digit_one;

                                        refresher = 0;
                                        break;
                        default: break;
        }// switch()
}// Display()

//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// EX0_ISR
//-----------------------------------------------------------------------------
//
// Starts the displaying of the data that is stored in ram_block
// by setting the appropriate flags
//
void EX0_ISR (void) interrupt 0
{
        RECD_flag = 0;                                          //Start displaying data
        END_flag = 0;
}// EX0_ISR()

//-----------------------------------------------------------------------------
// Timer0_ISR
//-----------------------------------------------------------------------------
//
// Refreshes the display every 5ms
// Every one seconds sends the next piece of data stored into ram_block
//
void Timer0_ISR (void) interrupt 1
{
        short letter;

        char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

        // Reload Timer 0 to start counting over
        // CRITICAL REGION

   EA = 0;                              // disables the interrupts to protect the critical region

        SFRPAGE = TIMER01_PAGE;
   TH0 =  0xDC;                                 // Load the initial value to start counting again
   TL0 =  0x00;

        EA = 1;                                 // enables the interrupt after the critical region is handled

        SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
        interrupt_count++;                          // increase the interrupt occurrence count

        // When interrupt_count occurs 200 times
        // 1 second has passed
        if(interrupt_count != 200)                          // 1 sec has not passed yet
        {
                //refresh display
                Display();
                return;
        }
        else                                                        // 1 sec has passed
        {
                if(RECD_flag || END_flag)
                {
                        interrupt_count = 0;
                        Display();                                  // Display RECD or END
                        return;
                }
                if(count == DATA_SIZE)                              // data has been displayed
                {
                        END_flag = 1;                               // Display END
                        count = 0;
                        interrupt_count = 0;
                        return;
                }

                letter = ram_block[count];                  // display the next character
                digit_one = BCD_7SEG(letter%10);      //LSB
                digit_two = BCD_7SEG((letter%100)/10);
                digit_three = BCD_7SEG(letter/100);   //MSB

                count++;

                interrupt_count = 0;                        // Start counting over

                Display();
        }
}// Timer0_ISR()

//------------------------------------------------------------------------------------
// ES_ISR
//------------------------------------------------------------------------------------
//
// Serial Interrupt Service Routine
// called only when the received byte is an address byte (9th bit RB8 = 1)
// and the address matches the address of the slave stored in the
// SADDR0 Register
//
void ES_ISR (void) interrupt 4
{
        char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

        RI0 = 0;                                                    // Clear the RI flag
        LED = 1;

        //***************RECEIVER************************
        // Determine if itself is among the receiver list
        // If YES -> SM2 = 0 to receive data
        // Save Data to a predetermined RAM block
        // Display RECD
        //
        // Invoke ISR to display the Data sequentially on display with an interval of 1 sec
        // Display END
        //**********************************************

        if(SM20 == 1 && SBUF0 != ADDRESS)       // Waiting mode, Still not addressed
                return;
        if(SM20 && SBUF0 == ADDRESS)    // Waiting mode, being addressed by the incoming byte
        {
                Receive_Toggle();                                   // Turn on Receiving Mode
                clear_display();                            // Clears the display
                count = 0;                                  // resets the ram_block location
        }
        else                                                                // Receiving mode
        {
                ram_block[count] = SBUF0;       // Store data in the predetermined RAM block

                count++;                        // Indicate that the next byte has been received

                if(count == DATA_SIZE)                              // All data has been received
                {
                        Receive_Toggle();                           // Turn off Receiving mode
                        count = 0;                                          // Reset count
                        RECD_flag = 1;                                      // Display RECD
                        Toggle_T0();                        // Start Timer 0 for TDM approach
                }

        }

        SFRPAGE = SFRPAGE_SAVE;         // Restore SFR page

        //***************TRANSMITTER*********************
        // Invoke ISR to begin writing #00 to #FF to a 256 bytes RAM block
        // Display DONE
        //
        // Invoke ISR to specify receiver and start transmission
        // Display SENT
        //**********************************************
}// ES_ISR()
```

```
//-----------------------------------------------------------------------------
// Initialization Subroutines
//-----------------------------------------------------------------------------
// Peripheral specific initialization functions,
// Called from the Init_Device() function
//

//-----------------------------------------------------------------------------
// Init_Device
//-----------------------------------------------------------------------------
//
// Initialization function for device
//
void Init_Device(void)
{
        Timer_Init();
        UART_Init();
        Port_IO_Init();
        Oscillator_Init();
        Interrupts_Init();
        Address_Init();
        LED_Init();
}// Init_Device()

//-----------------------------------------------------------------------------
// Timer_Init
//-----------------------------------------------------------------------------
//
// Initializes the proper mode of operation for Timers 0 and 1
// Loads the proper initial values into the timers
// Timer 0 - counts for approximately 5 ms and overflows
// Timer 1 - configures a baud rate of 9600 bit/sec
//
void Timer_Init(void)
{
   SFRPAGE   = TIMER01_PAGE;

   TMOD      = 0x21;                          // T1M1 = 1; Timer 1 in 8bit Auto-reload Mode
                                              // T0M0 = 1; Timer 0 in 16bit Counter Mode

        // Configure load value for T0 to count for 1 sec
        //--------------------------------------------------
        // CLK = SYSCLK/12 = 1.8432 MHz
        // It takes 0.54253472222 microsec per 1 count
        //
        // Therefore Timer 0 should count exactly 1843200 times
        // before 1 sec is passed
        //
        // The Max Count of Timer 0 is 65536
        //
        // Timer 0 can be set to count up to 9216 and after it overflows
        // exactly 200 times 1 second has passed. This way Timer 0 can be used
        // for a TDM approach to refresh the displays
        //
        // Load_value = (65535 - 9216 + 1) = 56320
        //
        // As shown above the intial value of Timer 0
   // should be set to 56320 = 0xDC00
        //--------------------------------------------------
        TH0 = 0xDC;                                         // Load initial value into Timer 0
        TL0 = 0x00;

        // Configure load value for T1 to get a 9600 baud rate
        //--------------------------------------------------
   //
        // TH1 = 256 - ((f_crystal / 384) / Baud)
        //
        //-------(f_crystal = 22.1184 MHz)---------
        // TH1 = 256 - ((22118400 / 384) / 9600)
   // TH1 = 250
        //--------------------------------------------------
   TH1 = 0xFA;                                         // Load initial value into TH1

   TCONx = 0x41;                                       // TR1 = 1; enable Timer 1 Run Control
                                                 // IT0 = 1; /INT0 is edge triggered, falling-edge
}

//-----------------------------------------------------------------------------
// UART_Init
//-----------------------------------------------------------------------------
//
// Initializes Serial Communication into Multiprocessor Variable Mode
//
void UART_Init(void)
{
   SFRPAGE   = UART0_PAGE;
   SCON0     = 0xF0;

        /*
        * SM00 = 1
        * SM01 = 1 - Selects Mode 3 - Multiprocessor Variable Mode
        *
        * SM20 = 1 - Multiprocessor Communications Enable
        * 0: Logic level of ninth bit is ignored.
        * 1: RI0 is set and an interrupt is generated only
        * when the 9th bit (RB8) is set
        *
        * REN0 = 1 - UART0 Reception Enabled
        */
}

//-----------------------------------------------------------------------------
```

```
// Port_IO_Init
//-----------------------------------------------------------------------------
//
// Routes the Serial Interrupt and the External Interrupt Input pins to the crossbar
// Configure the Crossbar and GPIO ports
//
void Port_IO_Init()
{
   // P0.0  -  TX0 (UART0), Open-Drain, Digital
   // P0.1  -  RX0 (UART0), Open-Drain, Digital
   // P0.2  -  INT0 (Tmr0), Open-Drain, Digital
   // P0.3  -  Unassigned,  Open-Drain, Digital

   SFRPAGE   = CONFIG_PAGE;
   XBR0      = 0x04;                // UART0EN = 1: UART0 I/O Enable Bit.
                                                      // TX routed to P0.0, and RX routed to P0.1
   XBR1      = 0x04;                //INT0E = 1: /INT0 Input Enable Bit.
   XBR2      = 0x40;                //40 - Enabled weak pull-ups
                                                      //C0 - Disabled weak pull-ups

}

//-----------------------------------------------------------------------------
// Oscillator_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the external oscillator
// at 22.1184 MHz
//
void Oscillator_Init()
{
         // Configure The External Oscillator to use a 22.1184 MHz frequency
   int i = 0;
   SFRPAGE   = CONFIG_PAGE;

         // Step 1. Enable the external oscillator.
   OSCXCN    = 0x67;

         // Step 2. Wait 1ms for initialization
   for (i = 0; i < 3000; i++);

         // Step 3. Poll for XTLVLD => '1'
   while ((OSCXCN & 0x80) == 0);

         // Step 4. Switch the system clock to the external oscillator.
   CLKSEL    = 0x01;
   OSCICN    = 0x00;
}

//-----------------------------------------------------------------------------
// Interrupts_Init
```

```
//-----------------------------------------------------------------------------
//
// Enables the required Interrupts by configuring the
// - Interrupt Enable Register
// - Interrupt Priority Register
//
void Interrupts_Init()
{
   IE       = 0x11;          // ES0 = 1: Enable UART0 Interrupt
                                                   // EX0 = 1: Enable External Interrupt 0
   IP       = 0x10;          // PS0 = 1: UART0 Interrupt Priority Control
}

//-----------------------------------------------------------------------------
// Address_Init
//-----------------------------------------------------------------------------
//
// Enables the required Interrupts by configuring the
// - Interrupt Enable Register
// - Interrupt Priority Register
//
void Address_Init(void)
{
        SADDR0 = ADDRESS;           // The masked address is set
        SADEN0 = 0xFF;                          // All bits of the address are checked
}

//------------------------------------------------------------------------------------
// LED_Init
//------------------------------------------------------------------------------------
//
// Enable P1.6 (LED) as push-pull output
//
void LED_Init(void)
{
        P1MDOUT |= 0x40;
        LED = 0;
}
```

A.3 Transmitter (master) Configuration Code that allows multiprocessor communications via universal asynchronous receiver and transmitter (UART)

```
#include <c8051f120.h>
//------------------------------------------------------------------------------------
// Global CONSTANTS
//------------------------------------------------------------------------------------
#define ADDRESS 0x05
#define S 0x24                                  // Letter codes
#define E 0x30
#define N 0x6A
```

```
#define T 0x70
#define D 0x42
#define O 0x62

sbit LED = P1^6;

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void UART_Init();
void Interrupts_Init();
void Timer_Init();
void Port_IO_Init();
void Oscillator_Init();
void LED_Init(void);

//-----------------------------------------------------------------------------
// Variable Declaration
//-----------------------------------------------------------------------------
int xdata fill[256];

int data dummy = 0;
int n = 0;
short refresher = 0;

sbit MSEL1 = P1^5;                                  // Multiplexer Select bits
sbit MSEL0 = P1^4;

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void) {

  // disable watchdog timer
  WDTCN = 0xde;
  WDTCN = 0xad;


   Timer_Init();
   UART_Init();
   Interrupts_Init();
   Port_IO_Init();
   Oscillator_Init();

        LED_Init();
  SFRPAGE = LEGACY_PAGE;            // Page to sit in for now


  while (1) {                  // spin forever

  }
```

```
}

void LED_Init(void)
{
            P1MDOUT |= 0x40;
            LED = 0;
}
//-----------------------------------------------------------------------------
// Initialization Subroutines
//-----------------------------------------------------------------------------
void Timer_Init()
{
   SFRPAGE   = TIMER01_PAGE;

   TMOD      = 0x21;
   //CKCON     = 0x02;                      // Timer 0 uses a pre-scaled SYSCLK/48 = 0.4608 MHz
                                            // Timer 1 uses a pre-scaled SYSCLK/12 = 1.8429 MHz

   TH0 = 0xDC; //4C                                        // Load initial value into Timer 0
   TL0 = 0x00;


   TH1 = 0xFA;                                             // Load initial value into TH1

            TCON      = 0x41;                              // TR1 = 1; enable Timer 1 Run Control
                                                 // IT0 = 1; /INT0 is edge triggered, falling-edge
}

void UART_Init()
{
   SFRPAGE   = UART0_PAGE;
   SCON0     = 0xC0;
}
void Interrupts_Init()
{
   IE        = 0x91;
   IP        = 0x10;          // PS0 = 1: UART0 Interrupt Priority Control
}

void Oscillator_Init()
{
            // Configure The External Oscillator to use a 22.1184 MHz frequency
   int i = 0;
   SFRPAGE   = CONFIG_PAGE;

            // Step 1. Enable the external oscillator.
   OSCXCN    = 0x67;

            // Step 2. Wait 1ms for initialization
   for (i = 0; i < 3000; i++);
```

```
        // Step 3. Poll for XTLVLD => ‘1’
   while ((OSCXCN & 0x80) == 0);

        // Step 4. Switch the system clock to the external oscillator.
   CLKSEL    = 0x01;
   OSCICN    = 0x00;
}


void Port_IO_Init()
{
   // P0.0  -  TX0 (UART0), Push-Pull,  Digital


   SFRPAGE   = CONFIG_PAGE;

   P0MDOUT   = 0x01;                            //configuring Port 1 output mode by
                                                //configuring the P0mDOUT SFR
                                                //it is enabling the last bit
                                                //which when set to 1, sets P0.0 to Push-Pull



   XBR0      = 0x04;                            //Enabling the third LSB in XBR0 SFR
                                                //this routes UART0 TX to P0.0
                                                //allowing transmission to take place
                                                //through this pin



  XBR1      = 0x04;    //Enabling the third LSB in XBR1 SFR
                                                //this routes external interrupt 0 to P0.2



  XBR2      = 0x40;                             //Enabling the crossbar by setting
                                                //the fourth MSB in the XBR2 SFR to 1
                                                //also the universal weak-pullup bit
                                                //is kept at 0 to enable weak-pullup
}
//-----------------------------------------------------------------------------
// Support Subroutines
//-----------------------------------------------------------------------------
//the code below writes 00 to FF to RAM

void fillup()
{
        char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

        int i;

        for(i=0;i<256;i++)
        {
                fill[i]=i;  //write from 00 to FF to array called "fill"
        }

        dummy = 1;
        SFRPAGE = TIMER01_PAGE;

        // Configure TCON register
        TR0 |= 1;                         // Turn on Timer 0 run control

        // Configure IE register
        ET0 |= 1;                                                   // Turn on Timer 0 overflow interrupt

        SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page
}

//--------------------------------------------------------------------------------
//Interrupt Service Routine
//--------------------------------------------------------------------------------

void EX0_ISR (void) interrupt 0
 {

 if (dummy == 0)
 fillup();
 else
 {
        TB80 = 1;                               //make all slaves listen
        SBUF0=ADDRESS;                          //send out address to specify listener

 }
}
void Timer0_ISR (void) interrupt 1
{
        char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page


        // Reload Timer 0 to start counting over
        // CRITICAL REGION

   EA = 0;                                      // disables the interrupts to protect the critical region

        SFRPAGE = TIMER01_PAGE;
   TH0 =  0xDC;          //4C                   // Load the initial value to start counting again
   TL0 =  0x00;

        EA = 1;                                 // enables the interrupt after the critical region is handled

        SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page


        if(dummy != 0)
        {
```

```
        //TDM approach to display RECD
        switch(refresher)
        {
            case 0:
                    MSEL1 = 0;
                    MSEL0 = 0;
                    if(dummy == 1)
                        P2 = D;
                    else
                        P2 = S;

                    refresher++;
                    break;
            case 1:
                    MSEL1 = 0;
                    MSEL0 = 1;
                    if(dummy == 1)
                        P2 = O;
                    else
                        P2 = E;
                    refresher++;
                    break;
            case 2:
                    MSEL1 = 1;
                    MSEL0 = 0;
                    P2 = N;
                    refresher++;
                    break;
            case 3:
                    MSEL1 = 1;
                    MSEL0 = 1;
                    if(dummy == 1)
                        P2 = E;
                    else
                        P2 = T;
                    refresher = 0;
                    break;
            default: break;
        }
    }

}// Timer0_ISR()
//--------------------------------------------------------------------------------
//SBUF INTERRUPT
//--------------------------------------------------------------------------------

void ES_ISR (void) interrupt 4
{
    TI0 = 0;
    TB80 = 0;     //make only the chosen listener to listen
      //TB8 = 0 is only relevant when ISR occurs after sending address byte
```

```
if(n < 256)
        {

                SBUF0 = fill[n];
                n++;
        }
        else
        {
                dummy = 2;
                LED = 1;
        }
}
```