```
/*******************************************************************************
 Module
   SCI_Receive.c

 Revision
   1.0.1

 Description
   This is a template file for implementing a simple service under the
   Gen2 Events and Services Framework.

 Notes

 History
 When               Who        What/Why
 -------------- ---        --------
 01/16/12 09:58 jec          began conversion from TemplateFSM.c
*******************************************************************************/
/*---------------------------- Include Files ----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "SCI_Receive.h"
#include "SCI_Send.h"
#include "ES_Timers.h"
#include "Coach.h"
#include "ADS12.h"
#define Deliminer 0x7E
#define WaitTime   500
#define MASK 0x07     //Mask for the LED
//#define ADTime 20

/*---------------------------- Module Defines ----------------------------*/

/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service
*/

/*---------------------------- Module Variables ----------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;
static RXState_t CurrentState;
static unsigned char Counter = 1;          //Counter is to keep track of the first 3 bytes
static unsigned char RCounter = 0;
static unsigned char RunSum = 0;
static unsigned char MSB_Message, LSB_Message, Num, RX_Data, CheckSum, Sum, J, i, Tem, k;
unsigned char ORI_Data[50], Trans_Data[50], LEDArray[8];
// The above array is for tranfmission of the received data, order: MSB, LSB, Data...
static ES_Event PostEvent;

static unsigned char Energy_Level[8];


/*---------------------------- Module Code ----------------------------*/
/*******************************************************************************
 Function
     Init_SCI_Receive
```

```
 Parameters
       uint8_t : the priorty of this service

 Returns
       boolean, False if error in initialization, True otherwise

 Description
       Saves away the priority, and does any
       other required initialization for this service
 Notes

 Author
       J. Edward Carryer, 01/16/12, 10:00
*****************************************************************************/
bool Init_SCI_Receive ( uint8_t Priority )
{
     ES_Event ThisEvent;

     MyPriority = Priority;

     //Initialize SCI Service
     //DDRT = 0x00; //Configure all the pins on Port T as inputs
     //DDRU = 0x00; //Configure U pins to be inputs
     //Set baud rate to ~9600 (0x00 to SCI1BDH, 156 to SCI1BDL)
     SCI1BDH = 0x00;
     SCI1BDL = 157;
     SCI1CR1 &= ~_S12_M;   //Configure for 8-bit transmission (clear M in SCI1CR1)
     SCI1CR1 &= ~_S12_LOOPS;   //Disable looping (clear LOOPS in SCI1CR1)
     SCI1CR1 &= ~_S12_PE; //Disable parity checking (clear PE in SCI1CR1)
     SCI1CR1 &= ~_S12_WAKE; //Configure for idle line wake-up (clear WAKE in SCI1CR1)
     SCI1CR1 &= ~_S12_ILT;
     SCI1CR1 &= ~_S12_RSRC;
     //SCI1CR1 |= _S12_ILT; //Count for idle line after stop bit (set ILT in SCI1CR1)
     //SCI1CR1 |= _S12_RSRC; //Set receiver source to pin (for use in debugging with LOOPS set, set RSRC
in SCI1CR1)
     SCI1CR2 |= (_S12_TE)|(_S12_RE); //Enable transmitter and receiver (set TE and RE in SCI1CR2)


     //ClearLED();   // Clear the LED array

     DisableInterrupts;   //Initialize Timer 0, Channel 4

//Configure clock scaler to divide sytem clock by 128
     TIM1_TSCR2 |= (_S12_PR2)|(_S12_PR1)|(_S12_PR0);

//Configure channel 4 as output compare
     TIM1_TIOS |= _S12_IOS4;

//Configure channel 4 to leave pin alone
     TIM1_TCTL1 &= (~_S12_OM4)&(~_S12_OL4);


//Initialize output compare register
//TIM0_TC4 = TIM0_TCNT + LED_SHIFT_INTERVAL;

//Enable interrupts for channel 4
//TIM0_TIE |= _S12_C4I;
```

```
//Enable Timer
    TIM1_TSCR1 |= _S12_TEN;


//Clear flags
    TIM1_TFLG1 = _S12_C4F;


    EnableInterrupts;

    Tem = 0;
    CurrentState = Wait_7E;


    printf("Finish Receive init~~~~~~~~~~~~~~\n\r");
    //printf("here now\r\n");

    // post the initial transition event
    ThisEvent.EventType = ES_INIT;
    if (ES_PostToService( MyPriority, ThisEvent) == true)
        {
            return true;
        }
    else
        {
            return false;
        }
}

/****************************************************************************
 Function
     Post_SCI_Receive

 Parameters
     EF_Event ThisEvent ,the event to post to the queue

 Returns
     boolean False if the Enqueue operation failed, True otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
bool Post_SCI_Receive( ES_Event ThisEvent )
{
    return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
     Run_SCI_Receive

 Parameters
   ES_Event : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise
```

```
 Description
   add your description here
 Notes

 Author
   J. Edward Carryer, 01/15/12, 15:23
*****************************************************************************/
ES_Event Run_SCI_Receive( ES_Event ThisEvent )
{
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

//printf("comes into receive state");



    switch ( CurrentState )
        {
        case Wait_7E :
            if ( ThisEvent.EventType == Received && ThisEvent.EventParam == Deliminer)//
                {

                    //printf("IN 7E");

                    RunSum = 0;                     // It is important to make the RunSum back to 0 here
                    CurrentState = Wait_MSB;
                    Counter = 2;

                    ES_Timer_SetTimer (Receive_TIMER, WaitTime);       // Use timer for each state
below
                    ES_Timer_StartTimer(Receive_TIMER);

                }
            break;

        case Wait_MSB :
            if ( ThisEvent.EventType == Received && Counter == 2 )//
                {
                    MSB_Message = ThisEvent.EventParam;
                    CurrentState = Wait_LSB;
                    Counter = 3;

                    ES_Timer_SetTimer (Receive_TIMER, WaitTime);
                    ES_Timer_StartTimer(Receive_TIMER);

                    //Since the first 3 bits are 0x7E and two address byte, the counter is up to 3
                }
            else if ((ThisEvent.EventType==ES_TIMEOUT) && (ThisEvent.EventParam==Receive_TIMER))
                {
                    CurrentState = Wait_7E;         // If time out, go back to state Wait_7E
                    printf("RMSB timeout");
                }

            break;

        case Wait_LSB :
            if ( ThisEvent.EventType == Received && Counter == 3 )//
                {
                    LSB_Message = ThisEvent.EventParam;
```

```
            Num = LSB_Message;
            J = Num;   // For the index we use for the Trans_Data

            Counter = 1;
            CurrentState = Suckupbody;

            ES_Timer_SetTimer (Receive_TIMER, WaitTime);
            ES_Timer_StartTimer(Receive_TIMER);
        }


    else if ((ThisEvent.EventType==ES_TIMEOUT) && (ThisEvent.EventParam==Receive_TIMER))
        {
            CurrentState = Wait_7E;         // If time out, go back to state Wait_7E
            printf("RLSB timeout");
        }

    break;


case Suckupbody :

    if ( ThisEvent.EventType == Received && Num > 1)
        {
            //Now we add the "real" data into RunSum
            RX_Data = ThisEvent.EventParam;
            ORI_Data [J - Num] = RX_Data;
            RunSum = RunSum + RX_Data;
            Num = Num - 1;

            ES_Timer_SetTimer (Receive_TIMER, WaitTime);
            ES_Timer_StartTimer(Receive_TIMER);

        }

    else if ( ThisEvent.EventType == Received && Num == 1)
        // For the last data we get, we change the state
        {
            RX_Data = ThisEvent.EventParam;
            ORI_Data [J - Num] = RX_Data;
            RunSum = RunSum + RX_Data;
            CurrentState = Wait_CheckSum;

            ES_Timer_SetTimer (Receive_TIMER, WaitTime);
            ES_Timer_StartTimer(Receive_TIMER);

        }

    else if ((ThisEvent.EventType==ES_TIMEOUT) && (ThisEvent.EventParam==Receive_TIMER))
        {
            CurrentState = Wait_7E;         // If time out, go back to state Wait_7E
            printf("RSuck timeout");
        }


    break;


case Wait_CheckSum:
```

```
if (ThisEvent.EventType == Received)
    {

        //printf("Wait for cal checksum");


        CheckSum = ThisEvent.EventParam;
        Sum = CheckSum + RunSum;

        //printf("runsum:%x\n\r",RunSum);
        //printf("sum:%x\n\r",Sum);

        RunSum = 0;          //Need to reset RunSum here

        // Test whether the checksum is good
        if (Sum == 0xFF)
            {
                PostEvent.EventType = CheckSum_Good;
                Post_SCI_Receive (PostEvent);
            }
        else
            {
                PostEvent.EventType = CheckSum_Bad;
                Post_SCI_Receive (PostEvent);
            }

        ES_Timer_SetTimer (Receive_TIMER, WaitTime);
        ES_Timer_StartTimer(Receive_TIMER);

    }

else if (ThisEvent.EventType == CheckSum_Good)
    {
        //printf("CheckSum is good!!!\n\r");
        if (ORI_Data[0] == 0x81)
            {
                Trans_Data [0] = ORI_Data [1];     // Address MSB
                Trans_Data [1] = ORI_Data [2];     // Address LSB

                for (i = 0; i < (J - 5); i++)      // J is the original Num
                    {
                        Trans_Data [2+i] = ORI_Data [5+i]; //Data
                    }
                AfterGood();
            }


        CurrentState = Wait_7E;

    }

else if (ThisEvent.EventType == CheckSum_Bad)
    {
        //printf("checksum:%x\n\r",CheckSum);
        printf("Bad...\n\r");
        CurrentState = Wait_7E;

    }

else if ((ThisEvent.EventType == ES_TIMEOUT) &&
```

```
(ThisEvent.EventParam==Receive_TIMER))
                    {
                        CurrentState = Wait_7E;         // If time out, go back to state Wait_7E
                        printf("Rwaitcheck timeout");

                    }

                break;

            }



    return ReturnEvent;
}



unsigned char Query_SCI_Receive ( unsigned char n )
{
    return(Trans_Data[n]);
}


/******************************************************************************
 private functions
 ******************************************************************************/
void AfterGood (void)
{

    static unsigned char Oldtem;
    Oldtem = Tem;


    printf("MSB: %x\n\r", Trans_Data[0]);
    printf("LSB: %x\n\r", Trans_Data[1]);
    //printf("Datatype is: %x\n\r", Trans_Data[2]);
    //printf("Datavalue is: %x\n\r", Trans_Data[3]);


    if (Trans_Data[2] == 0x05)     // If HDR is 0x05, it is pair response
        {

            //printf("Get the pair response~~~\n\r");

            PostEvent.EventType = PAIR_RESP;
            Post_Coach (PostEvent);
        }



    if ((Trans_Data[2] == 0x06) && ((Trans_Data[3] | BIT7LO) == BIT7LO))   // STATUS message with
ACK = 1, tagged out
        {

            PostEvent.EventType = TaggedOut;
            Post_Coach (PostEvent);

            //printf("Get STATUS message----Tag out###\n\r");
```

```
        }


    else if (Trans_Data[2] == 0x06)
        {


            Tem = (Trans_Data[3] & MASK);
            //printf("The tem is:%x\n\r", Tem);


            if (Tem != Oldtem)
                {
                    Write_LED(Tem);
                    Oldtem = Tem;
                }



            PostEvent.EventType = GetHeartBeat;
            Post_Coach (PostEvent);

            //printf("Get STATUS message----HeartBeat###\n\r");


        }


}



void Write_LED(unsigned char display_value)
{
    unsigned char i;

    TIM1_TIE &= ~_S12_C4I; //disable interrupt

    for (i = 0; i <=7; i ++)
        {
            if (i < display_value)
                {
                    Energy_Level[i] = 1;
                }
            else
                {
                    Energy_Level[i] = 0;
                }

        }
    //Energy_Level = display_value; //Update energy level to be displayed

    //Initialize output compare register
    TIM1_TC4 = TIM1_TCNT + LED_SHIFT_INTERVAL;

    //Enable interrupts for timer channel 4 until next LED update
    TIM1_TIE |= _S12_C4I;
}

void interrupt   _Vec_tim1ch4 LED_Shift_Timeout(void)
```

```
{

    static unsigned char data_i = 0;
    static unsigned char shift_i = 0;
    static unsigned char reg_i = 0;


    unsigned char i = 0;
    unsigned char Temp_Energy_Level[8];



    //clear flag
    TIM1_TFLG1 = _S12_C4F;

    if (data_i <= 7)
        {

            if (shift_i == 0) //Shift data into shift register
                {
                    if (Energy_Level[data_i] == 1)
                        {
                            PTT |= BIT7HI;
                        }
                    else
                        {
                            PTT&= BIT7LO;
                        }

                    PTT |= BIT6HI; //Pulse shift register clock high

                    //PTT |= BIT5HI;

                    shift_i = 1; //Set shift data clock flag
                }
            else
                {
                    PTT &= BIT6LO; //Pulse shift register clock low

                    //PTT &= BIT5LO;

                    shift_i = 0; //Clear shift register clock flag

                    data_i ++; //increment data bit counter
                }

            TIM1_TC4 += LED_SHIFT_INTERVAL;

        }
    else if (data_i == 8)
        {
            if (reg_i == 0)
                {
                    PTT |= BIT5HI;    //Pulse register shift clock high

                    reg_i = 1; //Set shift register clock flag

                    TIM1_TC4 += LED_DELAY_INTERVAL;
```

```
            }
        else
            {
                PTT &= BIT5LO; //Pulse register shift clock low

                reg_i = 0; //Clear shift regsiter clock flag

                data_i = 0; //Reset data index

                for (i = 0; i <= 7; i ++)              //Rotate LED pattern
                    {
                        if (i == 0)
                            {
                                Temp_Energy_Level[i] = Energy_Level[7];
                            }
                        else
                            {
                                Temp_Energy_Level[i] = Energy_Level[i - 1];
                            }
                    }

                for (i = 0; i <= 7; i ++)
                    {

                        Energy_Level[i] = Temp_Energy_Level[i];
                    }

                TIM1_TC4 += LED_DELAY_INTERVAL;

                //Disable interrupts for timer channel 4 until next LED update
                //TIM1_TIE &= ~_S12_C4I;

            }


    }


  //TIM1_TC4 += LED_SHIFT_INTERVAL; //Increment output compare register for next
  //shift pulse timeout event

}



/*------------------------------ Footnotes -------------------------------*/
/*------------------------------ End of file -------------------------------*/
```