```
/*******************************************************************************
 Module
   MasterSM.c

 Revision
   2.0.1

 Description
   This is a template for the top level Hierarchical state machine

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 02/08/12 01:39 jec      converted from MW_MasterMachine.c
 02/06/12 22:02 jec      converted to Gen 2 Events and Services Framework
 02/13/10 11:54 jec      converted During functions to return Event_t
                         so that they match the template
 02/21/07 17:04 jec      converted to pass Event_t to Start...()
 02/20/07 21:37 jec      converted to use enumerated type for events
 02/21/05 15:03 jec      Began Coding
*******************************************************************************/
/*----------------------------- 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 "MasterSM.h"
#include "ES_ServiceHeaders.h"

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

#define HeaderByte      0b01111110
#define BroadcastMSB    0xFF
#define BroadcastLSB    0xFF
#define TXOptions       0b00000000

#define StatusIndex     9
#define BatteryIndex    10
#define ChargeIndex     11
#define LEDIndex        12
#define CameraIndex    13

#define AccelNeutral    338
#define OneG            164
#define gLimit          .2

#define ROAMER1         1
#define ROAMER2         2
#define ROAMER3         3

/*---------------------------- Module Functions ---------------------------*/
static void processKeyboard(uint16_t);
```

```
static void InitComm(void );
static void InitTimers(void );
static boolean sendByte(char );
static void readMsg(void );
static void sendSM(void );
static void receiveSM(void );
static void RoamerLEDOff(void );



/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, though if the top level state machine
// is just a single state container for orthogonal regions, you could get
// away without it
static MasterState_t CurrentState;
static TXState_t CurrentTXState = TX_START;
// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority;
static char transmit[108];
static char receive[108];
static int currentRoamer = 11;
static int DestinationMSB = 0x00;
static int DestinationLSB = 0x00;
static boolean receiving, sending, connecting, disconnecting, broadcast, connected;
static int errorcount = 0;
static boolean camera = FALSE;
static boolean lastCamera = TRUE;




/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitMasterSM

 Parameters
     uint8_t : the priorty of this service

 Returns
     boolean, False if error in initialization, True otherwise

 Description
     Saves away the priority,  and starts
     the top level state machine
 Notes

 Author
     J. Edward Carryer, 02/06/12, 22:06
****************************************************************************/
boolean InitMasterSM ( uint8_t Priority )
{
  ES_Event ThisEvent;
```

```
  MyPriority = Priority;  // save our priority
  CurrentState = Waiting;

  ThisEvent.EventType = ES_ENTRY;
  // Start the Master State machine

  InitComm();
  InitTimers();

  //Port outputs/inputs
  DDRP |= (BIT1HI | BIT3HI | BIT4HI | BIT5HI);

  DDRS &= BIT2LO; //S2 to input for SCI1 receive
  DDRS |= BIT3HI; //S3 to output for SCI2 transmit
  DDRU &= 0;
  DDRT |= BIT4HI;

  ADS12_Init("OOOOAAAA");

  Servo_SetAngle(4,90);
  PTAD = 0x0;
  //Initial state actions

  StartMasterSM( ThisEvent );
  receiving = FALSE;
  sending = FALSE;
  connecting = FALSE;
  broadcast = FALSE;
  connected = FALSE;
  disconnecting = FALSE;



  return TRUE;
}

/*****************************************************************************
 Function
     PostMasterSM

 Parameters
     ES_Event ThisEvent , the event to post to the queue

 Returns
     boolean False if the post operation failed, True otherwise

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

 Author
     J. Edward Carryer, 10/23/11, 19:25
*****************************************************************************/
boolean PostMasterSM( ES_Event ThisEvent )
{
```

```
  return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunMasterSM

 Parameters
   ES_Event: the event to process

 Returns
   ES_Event: an event to return

 Description
   the run function for the top level state machine
 Notes
   uses nested switch/case to implement the machine.
 Author
   J. Edward Carryer, 02/06/12, 22:09
****************************************************************************/
// make recursive call warning into info
#pragma MESSAGE INFORMATION C1855
ES_Event RunMasterSM( ES_Event CurrentEvent )
{
   ES_Event currEvent;
   unsigned char MakeTransition = False;/* are we making a state transition? */
   MasterState_t NextState = CurrentState;
   ES_Event EntryEventKind = { ES_ENTRY, 0 };
   ES_Event ReturnEvent = CurrentEvent;
   static int i = 0;
   //printf("%d\r\n", CurrentEvent.EventParam);

   if (CurrentEvent.EventType == ES_TIMEOUT && CurrentEvent.EventParam == 0)
   {

      sending = TRUE;
      SCI1CR2 |= _S12_TIE;
      if(connecting || disconnecting || connected)
      {
         ES_Timer_SetTimer(0,200);
         ES_Timer_StartTimer(0);
      }

   }

   if (CurrentEvent.EventType == ES_TIMEOUT && CurrentEvent.EventParam == 1)
   {

      PostButtonSM(CurrentEvent);

   }

   if (CurrentEvent.EventType == RoamerSwitch)
   {
      //currentRoamer = CurrentEvent.EventParam;
```

```
   int lastRoamer = currentRoamer;
   if (CurrentEvent.EventParam == 1) currentRoamer = ROAMER1;
    if (CurrentEvent.EventParam == 2) currentRoamer = ROAMER2;
    if (CurrentEvent.EventParam == 3) currentRoamer = ROAMER3;
   if (connected)
   {
      if (currentRoamer != lastRoamer)
      {
         connecting = TRUE;
        printf("Connecting to roamer!\r\n");
        ES_Timer_SetTimer(0,200);
         ES_Timer_StartTimer(0);
      }

   } else
   {
      connecting = TRUE;
        printf("Connecting to roamer!\r\n");
        ES_Timer_SetTimer(0,200);
         ES_Timer_StartTimer(0);
   }


    if (connected) disconnecting = TRUE;
}

if (CurrentEvent.EventType == ES_NEW_KEY)
{
   processKeyboard(CurrentEvent.EventParam);
}

if (CurrentEvent.EventType == ButtonDown  && CurrentEvent.EventParam == 0)
{
   camera = camera ? FALSE : TRUE;
}

switch(CurrentState)
{
   case Waiting:

   break;

   case Transmitting:

   break;

   case Receiving:

   break;

}



if ( CurrentEvent.EventType == EV_CLEAR)
```

```
  {
       // First pass exit messages to the lower level machines
       CurrentEvent.EventType = ES_EXIT;
       RunMasterSM(CurrentEvent);
       // Now pass entry messages, since this is a self transition
       CurrentEvent.EventType = ES_ENTRY;
       RunMasterSM(CurrentEvent);
  }

  if (MakeTransition == True)
  {
      //   Execute exit function for current state
      CurrentEvent.EventType = ES_EXIT;
      RunMasterSM(CurrentEvent);

      CurrentState = NextState; //Modify state variable

      //   Execute entry function for new state
      // this defaults to ES_ENTRY
      RunMasterSM(EntryEventKind);
      printf("Changed States\r\n");
  }

  // in the absence of an error the top level state machine should
  // always return ES_NO_EVENT
  CurrentEvent.EventType = ES_NO_EVENT;
  return(CurrentEvent);
}
/*****************************************************************************
 Function
     StartMasterSM

 Parameters
     ES_Event CurrentEvent

 Returns
     nothing

 Description
     Does any required initialization for this state machine
 Notes

 Author
     J. Edward Carryer, 02/06/12, 22:15
*****************************************************************************/
void StartMasterSM ( ES_Event CurrentEvent )
{
  // local variable to get debugger to display the value of CurrentEvent
  volatile ES_Event LocalEvent = CurrentEvent;
  // if there is more than 1 state to the top level machine you will need
  // to initialize the state variable
  //ES_Timer_SetTimer(1, 2000);
  //ES_Timer_StartTimer(1);
  CurrentState = Waiting;
```

```
  // now we need to let the Run function init the lower level state machines
  // use LocalEvent to keep the compiler from complaining about unused var
  RunMasterSM(LocalEvent);
  return;
}

MasterState_t QueryMasterSM(void )
{
   return CurrentState;
}

/***************************************************************************
 private functions
 ***************************************************************************/

static ES_Event DuringMaster( ES_Event Event )
{

    ES_Event currEvent;

    return(Event);
}




static void processKeyboard(uint16_t input)
{
   ES_Event currEvent;
   char key = (char)input;


   currEvent.EventType = ES_ENTRY;
   if (key == 'a')
   {
     currentRoamer = 11;
      connecting = TRUE;
      if (connected) disconnecting = TRUE;
      errorcount = 0;
      ES_Timer_SetTimer(0,200);
      ES_Timer_StartTimer(0);
      printf("Connecting to Roamer 11\r\n");
   }
   if (key == 'b')
   {
      broadcast = TRUE;
      printf("Broadcasting a message!\r\n");
   }
   if (key == 'c')
   {
      disconnecting = TRUE;
      printf("Disconnecting!\r\n");
      ES_Timer_SetTimer(0,200);
      ES_Timer_StartTimer(0);
   }
```

```
if (key == 'd')
{
  currentRoamer = 90;
   connecting = TRUE;
   if (connected) disconnecting = TRUE;
   ES_Timer_SetTimer(0,200);
   ES_Timer_StartTimer(0);
  printf("Connecting to Roamer 90\r\n");
}
if (key == 'e')
{
   currentRoamer = 11;
    connecting = TRUE;
   printf("Connecting without disconnect!\r\n");
}
if (key == '1')
{
  currentRoamer = 17;
   connecting = TRUE;
   if (connected) disconnecting = TRUE;
   errorcount = 0;
   ES_Timer_SetTimer(0,200);
   ES_Timer_StartTimer(0);
   printf("Connecting to Roamer 17\r\n");
}
if (key == '2')
{
  currentRoamer = 6;
   connecting = TRUE;
   if (connected) disconnecting = TRUE;
   errorcount = 0;
   ES_Timer_SetTimer(0,200);
   ES_Timer_StartTimer(0);
   printf("Connecting to Roamer 11\r\n");
}
if (key == '3')
{
  currentRoamer = 11;
   connecting = TRUE;
   if (connected) disconnecting = TRUE;
   errorcount = 0;
   ES_Timer_SetTimer(0,200);
   ES_Timer_StartTimer(0);
   printf("Connecting to Roamer 11\r\n");
}
if (key == 'p')
{
 Servo_SetAngle(4,165);
}

if (key == 'o')
{
 Servo_SetAngle(4,0);
}
```

```
   if (key == '0')
   {
      PTAD |= BIT5HI;
   }

   if (key == '9')
   {
      PTAD |= BIT4HI;
   }

   if (key == '8')
   {
      PTP |= BIT1HI;

   }
   if (key == 'm')
   {
      lastCamera = lastCamera ? FALSE : TRUE;
   }
   if (key == 'n')
   {
        static double accx, accy, gx, gy;
        static int left, right;
      accx = ADS12_ReadADPin(0);
          accy = ADS12_ReadADPin(1);
          gx = (accx-AccelNeutral)/OneG;//Due to mount
          gy = (accy-AccelNeutral)/-OneG; //Due to mount
          if (!lastCamera) gx *= -1;
          if (gx < gLimit && gx > -gLimit) gx = 0;
          if (gy < gLimit && gy > -gLimit) gy = 0;
          left = (int)(gx/.7 * 128 + gy/.7 * 96) + 128;
          right = (int)(gx/.7 * 128 - gy/.7 * 96) + 128;
          if (left < 0) left = 0;
          if (left > 0xFF) left = 0xFF;
          if (right < 0) right = 0;
          if (right > 0xFF) right = 0xFF;
          printf("right = %d left = %d \r\n", right, left);
   }

}

static void InitComm(void )
{
   //Registers for SCI
   SCI1CR2 |= (_S12_RE | _S12_RIE | _S12_TE);

   SCI1BDH = 0;
   SCI1BDL = 156; //24MHz / (16*SCIBDL)    = 9615


}

static void InitTimers(void )
{
   ES_Timer_SetTimer(0, 200);
```

```
   ES_Timer_StartTimer(0);

   ES_Timer_SetTimer(1, 200);
   ES_Timer_StartTimer(1);


   //Set TIM1 to Servo control
   TIM1_TSCR1 |= _S12_TEN;
                  // enable timer system
   TIM1_TSCR2 = _S12_PR1 | _S12_PR0; // 011 = 24MHz / 8 = 3MHz

   //Select channels on TIM1 for output compare
   TIM1_TIOS |= (_S12_IOS4 | _S12_IOS5 | _S12_IOS6 | _S12_IOS7);
   //Program a rise on compare
   TIM1_TCTL1 |= (_S12_OL4 | _S12_OM4);
   TIM1_TCTL1 |= (_S12_OL5 | _S12_OM5);
   TIM1_TCTL1 |= (_S12_OL6 | _S12_OM6);
   TIM1_TCTL1 |= (_S12_OL7 | _S12_OM7);
   //Set bit to toggle overflow
   TIM1_TTOV |= (_S12_TOV4 | _S12_TOV5 | _S12_TOV6 | _S12_TOV7);
   //Output pulse width for inital position
   TIM1_TC4 = 0xFFFF - 4500;
   TIM1_TC5 = 0xFFFF - 4500;
   TIM1_TC6 = 0xFFFF - 4500;
   TIM1_TC7 = 0xFFFF - 4500;
   //For any pins to remove from timer
}

static boolean sendByte(char msg )
{
   if ((SCI1SR1 & _S12_TDRE) != 0)
   {
      SCI1DRL = msg;
      return TRUE;
   } else
   {
      return FALSE;
   }

}



static void sendSM(void )
{
    static int checksum;
    static int FrameID = 0x01;
    static char byteToSend = 0x00;
    static double accx, accy, gx, gy;
    static int lastGripperState;
    static int left, right;
    switch(CurrentTXState)
    {
       case TX_START:
            if(sendByte(0x7E)) CurrentTXState = TX_MSBLength;
```

```
        checksum = 0;
        break;
    case TX_MSBLength:
        if(sendByte(0x00)) CurrentTXState = TX_LSBLength;
        break;
    case TX_LSBLength:
        if(sendByte(0x0B)) CurrentTXState = TX_API; // Change this to actual length of packet
        break;
    case TX_API:
        if(sendByte(0x01))
        {
           CurrentTXState = TX_FRAME;
           checksum += 0x01;
        }
        break;
    case TX_FRAME:
        if(sendByte(FrameID))
        {
           CurrentTXState = TX_MSBDestination;
           checksum += FrameID++;
           if (FrameID > 50) FrameID = 0x01;
        }
        break;
    case TX_MSBDestination:
        if (disconnecting) {
            byteToSend = DestinationMSB;
        } else if (connecting || broadcast) {
            byteToSend = BroadcastMSB;
        } else {
            byteToSend = DestinationMSB;
        }
        //byteToSend = BroadcastMSB;
        //byteToSend = 0x21;
        if(sendByte(byteToSend))
        {
           CurrentTXState = TX_LSBDestination;
           checksum += byteToSend;
        }
        break;
    case TX_LSBDestination:
        if (disconnecting) {
            byteToSend = DestinationLSB;
        } else if (connecting || broadcast) {
            byteToSend = BroadcastLSB;
           printf("Broadcasting\r\n");
        } else {
            byteToSend = DestinationLSB;
        }
        //byteToSend = BroadcastLSB;
        if(connected) printf("Connected\r\n");
        //byteToSend = 0x88;
        if(sendByte(byteToSend))
        {
           CurrentTXState = TX_OPTIONS;
           checksum += byteToSend;
```

```
      }
      broadcast = FALSE;
      break;
  case TX_OPTIONS:
      if(sendByte(TXOptions))
      {
         CurrentTXState = TX_FIRST;
         checksum += TXOptions;
      }
      break;
  case TX_FIRST:
      byteToSend = 0x02;
      if(disconnecting){
          byteToSend = 0x01;
      } else if (connecting){
          byteToSend = 0x00;
      }
      if(sendByte(byteToSend))
      {
         CurrentTXState = TX_LEFT;
         checksum += byteToSend;
      }

          //printf("byte to send %d", byteToSend);
      break;
case TX_LEFT:
   accx = ADS12_ReadADPin(0);
   accy = ADS12_ReadADPin(1);
   gx = (accx-AccelNeutral)/OneG;//Due to mount
   gy = (accy-AccelNeutral)/-OneG; //Due to mount
   //if (!lastCamera) gx *= -1;
   if (gx < gLimit && gx > -gLimit) gx = 0;
   if (gy < gLimit && gy > -(gLimit)) gy = 0;
   left = (int)(gx/.7 * 128 + gy/.7 * 96) + 128;
   right = (int)(gx/.7 * 128 - gy/.7 * 96) + 128;
   if (left < 0) left = 0;
   if (left > 0xFF) left = 0xFF;
   if (right < 0) right = 0;
   if (right > 0xFF) right = 0xFF;

   byteToSend = left;
   if(disconnecting){
       byteToSend = 0x00;
   } else if (connecting){
       byteToSend = currentRoamer;
   }
   if(sendByte(byteToSend))
   {
      CurrentTXState = TX_RIGHT;
      checksum += byteToSend;

   }
   break;
case TX_RIGHT:
   byteToSend = right;
```

```
   if(connecting || disconnecting) byteToSend = 0x00;
   if(sendByte(byteToSend))
   {
      CurrentTXState = TX_GRIPPER;
      checksum += byteToSend;
   }
   break;
case TX_GRIPPER:
//printf("Sensor one = %d Sensor two = %d\r\n", ADS12_ReadADPin(2), ADS12_ReadADPin(3));
   if (ADS12_ReadADPin(2) > 500 && ADS12_ReadADPin(3) > 500)
   {
      byteToSend = 0x80;
      printf("Gripper closed!\r\n");
   } else if (ADS12_ReadADPin(2) <= 500 && ADS12_ReadADPin(3) <= 500)
   {
      byteToSend = 0x00;
   } else
   {
      byteToSend = lastGripperState;
   }
   lastGripperState = byteToSend;

   if(connecting || disconnecting) byteToSend = 0x00;
   if(sendByte(byteToSend))
   {
      CurrentTXState = TX_CAMERA;
      checksum += byteToSend;
   }
   break;
case TX_CAMERA:
   if (camera)
   {
      byteToSend = 0x80;
   } else
   {
      byteToSend = 0x00;
   }

   if(connecting || disconnecting) byteToSend = 0x00;
   if(sendByte(byteToSend))
   {
      CurrentTXState = TX_DIGITAL;
      checksum += byteToSend;
   }
   break;
case TX_DIGITAL:
   byteToSend = (PTU & 0x70) >> 4; //Takes bits 4, 5, and 6 with the and, and shifts
   right 4 to send

   if(connecting || disconnecting) byteToSend = 0x00;
   //printf("Sent Packet! %d \r\n", byteToSend);
   if(sendByte(byteToSend))
   {
      CurrentTXState = TX_Checksum;
      checksum += byteToSend;
```

```
        }
        break;
      case TX_Checksum:
          SCI1CR2 &= ~_S12_TIE;
          sending = FALSE;
          if(sendByte(0xFF - (checksum & 0xFF)))
          {
              CurrentTXState = TX_START;

             if(connecting || disconnecting || connected) errorcount++;

             printf("Error count = %d\r\n", errorcount);
             if (errorcount > 5)
             {
                connected = FALSE;
                connecting = FALSE;
                disconnecting = FALSE;
                RoamerLEDOff();
                DestinationMSB = 0x00;
                 DestinationLSB = 0x00;
                 errorcount = 0;
             }

          //  printf("Sent Packet! accx = %d, accy = %d, right = %d, left = %d\r\n",
          (int)accx, (int)accy, right, left);
          }

          break;
      default:

          break;
    }
}

static void receiveSM(void )
{
   char input;
   static boolean active;
   static int i;
   static int checksumindex, checksum;
   static int MSBLength, LSBLength, Length, APIIdent, SourceMSB, SourceLSB, Source, RSSI,
   Options;
   static int LEDBits, Status, Battery, Charge, Camera;
   int test;
   if ((SCI1SR1 & _S12_RDRF) != 0)
   {
      input = SCI1DRL;
      input &= 0xFF;
   }


   if (((input & 0xFF) == HeaderByte ) && active != 1)
   {
       active = 1;
       i = 2;
```

```
    receiving = TRUE;
    receive[1] = HeaderByte;
    //printf("Got header\r\n");
} else if (active == 1)
{
   receive[i] = input; // i is the byte number
   //printf("i = %d, input = %x\r\n", i, input);
    switch(i)
    {
       case 2:
            MSBLength = receive[i];
         break;
     case 3:
         LSBLength = receive[i];
         Length = (int)(LSBLength + (MSBLength << 8));

         //printf("Length = %x\r\n", Length);
         if (Length < 1 || Length > 100)
         {
            Length = 3;
         }
         break;
     case 4:
         APIIdent = receive[i] & 0xFF;
         printf("API = %x\r\n", APIIdent);
         break;
     case 5:
        if(APIIdent == 0x81)
        {
           SourceMSB = receive[i] & 0xFF;
        } else if (APIIdent == 0x89)
        {

        }
         break;
     case 6:
         if(APIIdent == 0x81)
        {
           SourceLSB = receive[i] & 0xFF;
            Source = (int)(SourceLSB + (SourceMSB << 8));
               if (connected && (SourceLSB != DestinationLSB || SourceMSB != DestinationMSB))
               {
                  active = 0;
               }
            //printf("SourceMSB = %x\r\n", SourceMSB);
            //printf("SourceLSB = %x\r\n", SourceLSB);
            printf("Source = %x\r\n", Source);
        } else if (APIIdent == 0x89)
        {
           if(receive[i] == 0) printf("Hooray! Got Success\r\n");
           if(receive[i] == 1) printf("No Ack Received\r\n");
        }
         break;
     case 7:
         if(APIIdent == 0x81)
```

```
    {
       RSSI = receive[i];
    }
     break;
case 8:
    if(APIIdent == 0x81)
    {
        Options = receive[i];
    }
     break;
case StatusIndex:
     if (APIIdent == 0x81)
     {
         Status = receive[i] & 0xFF;
         if (disconnecting && Status == 0x05 && SourceLSB == DestinationLSB && SourceMSB
         == DestinationMSB){
                disconnecting = FALSE;
                connected = FALSE;
                DestinationMSB = 0x00;
                DestinationLSB = 0x00;
                RoamerLEDOff();
                printf("Disconnected!\r\n");
           } else if (connecting && Status == 0x04){
                printf("Connected to Source  = %x\r\n", Source);
               connected = TRUE;
               RoamerLEDOff();
               if (currentRoamer == ROAMER1)
               {
                 PTP |= BIT1HI;
                 printf("Roamer 11 connected\r\n");
               }
               else if (currentRoamer == ROAMER2)
               {
                 PTAD |= BIT4HI;
               }
               else if (currentRoamer == ROAMER3)
               {
                 PTAD |= BIT5HI;
               }
               connecting = FALSE;
                DestinationMSB = SourceMSB;
                DestinationLSB = SourceLSB;
           } //Takes last 3 bits
     }
     break;
case BatteryIndex:
     if (APIIdent == 0x81 && SourceLSB == DestinationLSB && SourceMSB == DestinationMSB)
     {
        Battery = receive[i] & 0xFF;
        Servo_SetAngle(4, (Battery*11)/17);
     }
     break;
case ChargeIndex:
     if (APIIdent == 0x81)
     {
```

```
            Charge = receive[i];
        }
        break;
    case LEDIndex:
        if (APIIdent == 0x81 && SourceLSB == DestinationLSB && SourceMSB == DestinationMSB)
        {
            LEDBits = receive[i];
            //PTU = LEDBits << 3;
        }
        break;
    case CameraIndex:
        if (APIIdent == 0x81 && SourceLSB == DestinationLSB && SourceMSB == DestinationMSB)
        {
            Camera = receive[i] & 0xFF;
            if ((Camera & BIT7HI) != 0)
            {
               lastCamera = TRUE;//Camera forwards/default
            } else
            {
               lastCamera = FALSE;//Camera other way
            }
            //test = 244;
            //printf("Camera is %d, %x, %u\r\n", Camera, Camera, Camera);
        //  printf("test is %d, %x, %u\r\n", test, test, test);
        }
        //Set Servo angle here
        break;
    default:
        break;
}
        //Process data if needed here

if (i > Length + 3){ // Checksum
    checksumindex = 4;
    checksum = 0;
    for (checksumindex; checksumindex < i; checksumindex++){
        checksum += receive[checksumindex];

    }
    if((checksum & 0xFF) == 0xFF)
        {
           printf("Good checksum\r\n");
        }
    if (APIIdent == 0x81 && connected)
    {
      errorcount = 0;
       printf("XBee received Source = %x, Length = %d, Status = %d, Battery = %d, Charge =
       %d, LEDBits = %d, Camera = %d\r\n", Source, Length, Status, Battery, Charge, LEDBits,
        Camera);
    }

    active = 0;

}
i++;
```

```
   }
}

void interrupt _Vec_sci1 readMsg(void )
{

   static char input;

   if ((SCI1SR1 & _S12_RDRF) != 0)
   {
      receiveSM();
   } else if ((SCI1SR1 & _S12_TDRE) != 0 && sending == TRUE)
   {
      sendSM();
   }


}

static void RoamerLEDOff(void )
{

   PTP &= BIT1LO;
   PTAD &= (BIT4LO & BIT5LO);
}
```