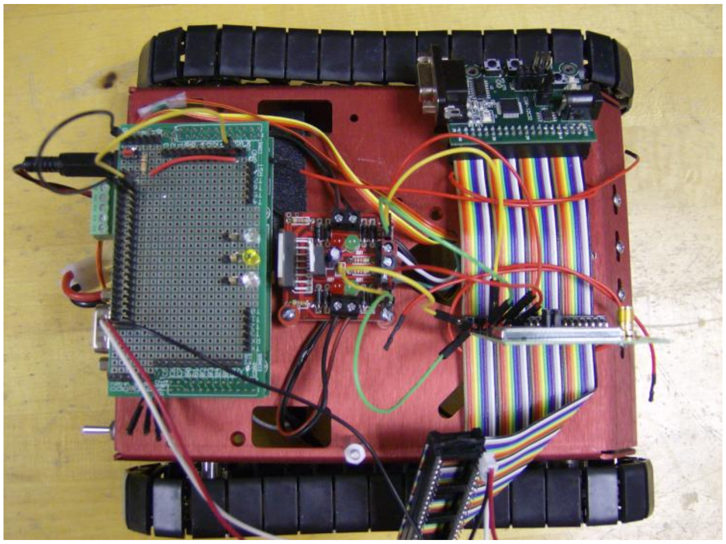
Zigbot



Scott Ruskamp and David Schamber
ECE331-01 Final Project
May 2007

Introduction

For our final project in Embedded System Design we chose to demonstrate the functionality of the ZigBee MC13192U module. To do this, we designed and implemented a remote control robot system. The controller is mounted on a Freescale Project Board with two potentiometers for forwards/backwards and rotate left/right control and the receiver is mounted on a Trackster robot named Cindy.

Objectives and Implementation

The main objective was to interface the ZigBee modules with the HC12S microcontroller, while also taking advantage of the knowledge and previous written code from the Embedded System Design class.

To achieve these goals, we decided to build a remote control robot system. The robot is controlled using the PWM module on the HC12S, which is powered by the robot's battery. The duty cycle of the PWM will be specified by digital values obtained using interrupts and the ATD module hooked up to potentiometers on the controller board, which send an 8-bit value representing the range from full reverse or left (0) to full forward or right (255) over the ZigBee protocol.

ZigBee demo code[1] was used for this project and modified for the robot needs. The modified code is highlighted and shown in Appendix A.

Hardware Design

[1] From CD that came with ZigBee hardware

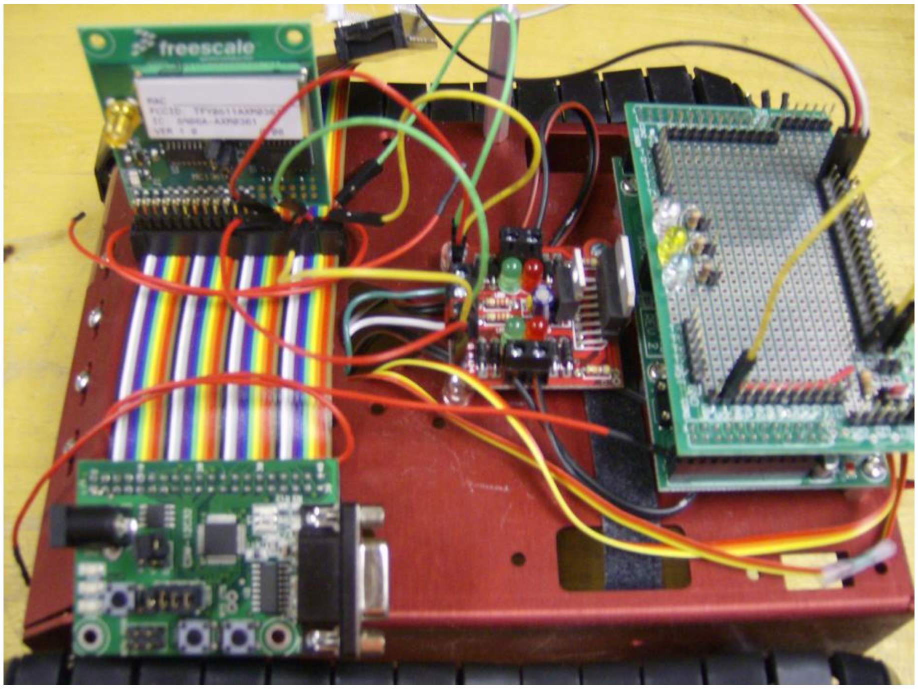


Figure 1: Zigbot Hardware Design

Figure 1 shows the layout on the Trackster robot. Table 1 below specifies which pins are connected from the robot to the microcontroller. Pins 1 to 24 are also used to connect the ZigBee module to the microcontroller.

Microcontroller Pin	Trackster Pin
1-24	MC13192U
1	VCC
3	GND
30	E1-2
32	E3-4
34	I4
36	I3
38	I2
40	I1

Table 1: Trackster Pin Layout

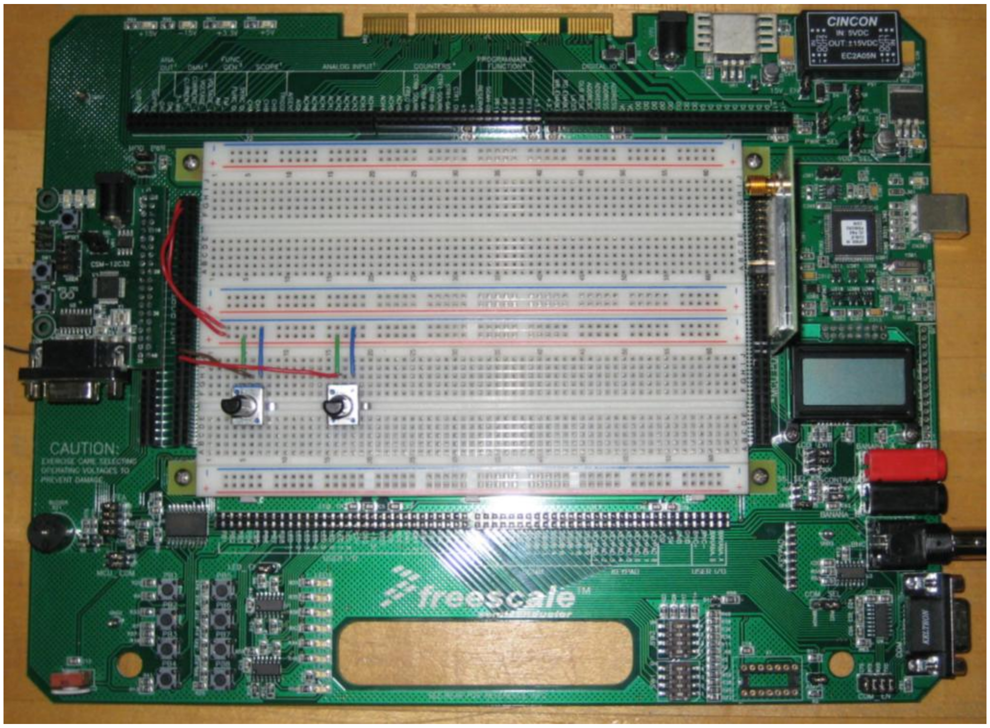


Figure 2: Remote control hardware design

Figure 2 above shows the layout of the remote control that was created on the Freescale Project Board. Table 2 below gives the specific pin assignments used to connect the components to the microcontroller.

Microcontroller Pin	Component
1-24	MC13192U
37	Potentiometer 1
39	Potentiometer 2

Table 2: Remote Control Pin Layout

Parts List

All parts were either provided free of charge or were already owned. A list of parts is shown below in Table 2.

Part	Quantity	Source
MC9S12C32 Microcontroller board	2	Lab kits
MC13192U ZigBee board	2	Dr. Song
Freescale Project Board	2	Dr. Song
Trackster Robot	1	Dr. Berry
Potentiometers	2	Lab kits

Ribbon cable	1	Lab kit, modified by technician in the parts room

Table 3: List of Parts Used

Conclusion

In conclusion, we were able to demonstrate wireless communication using the ZigBee modules by creating a remote control robot system with potentiometers as controls. The robot not only uses new technology but also takes advantage of what we learned in class, like analog to digital conversion, interrupts, and pulse width modulation. And the best part is that the robot is fun to play with!

A future project could have improved robot controlling or even some automation on the robot, such as obstacle avoidance or wall following.

Appendix A: Source Code

Controller

```
/**********************************************************************
******
Wireless_UART.c

Application Note: This simple application forms a point to point
wireless uart with ACKs.
Power either the 13192SARD or GB60 EVB with a MC13192 2.2 IC and
connect the UART1 on the EVB and
Uart1 for 13192SARD.  If you have legacy ARD boards this by default
must use UART2.

UNZIP this in the apps directory with other app demos.

Download the same code to 2 boards.

Set the Hyperterms for 38400 No flow control.
Type a character on one of the hyperterms.  It will send the character
via MC13192 to
the other board and display it on its respective Hyperterm.  It will be
displayed and the rx
board will send back "ACK".  The TX board will be listening for the
ACK.  On every timeout
the TX will retransmit the character.  The number of successive tries
is reflected
on the LEDs.  1 light is one retry.  2 is two retries.  3 is three
retries etc.

If the ack is received the LEDs are reset.
If the ack is not received by the 5th retry the character is dropped
and the
LEDs are reset.

The timeout period is controlled by TIMEOUT_PERIOD
The number of retries is kept in RETRY_COUNT.  Note retries past 5 will
reset

Also, on the GB60EVBs COM_SW 4 must be on to allow the RX on the UART2
to work.

$Author: a20639 $
$Date: 2005/08/11 20:46:23 $
$Name:  $
**********************************************************************
*****/
#include <hidef.h> /* for EnableInterrupts macro */
#include "pub_def.h"
#include "simple_mac.h"
#include "SCI.h"
#include "ATD.h"
#include "mc13192_hw_config.h"
//#include "bootloader user api.h"
#include "wireless_uart.h"
```

```
//JCB
#include "drivers.h"
#include "simple_phy.h"
#include "MC13192_regs.h"
#include "mcu_hw_config.h"
#include "mcu_spi_config.h"


/////////////////////////////
/* note: Buad Rate = 38400 */
/////////////////////////////

/*Defines*/

/*Global Variables*/
UINT8 gu8RTxMode;


extern UINT8 gu8SCIDataFlag;
extern UINT8 gu8SCIData[2];

extern UINT8 gu8ATDReadyFlag;       // ATD globals
extern UINT8 gu8ATDChannel6Data;
extern UINT8 gu8ATDChannel7Data;

UINT8 SCI_TestTx = 0;
INT8 gi8AppStatus = 0;

#ifdef BOOTLOADER_ENABLED
    #define TIMEOUT_PERIOD 0x2000 /*Changed to 0x2000 due to lower UART
baud rate. 28.09.04 MVC*/
#else
    #define TIMEOUT_PERIOD 0x1000
#endif BOOTLOADER_ENABLED

#define RETRY_COUNT 4

#if defined (_HCS12C)
  #pragma LINK_INFO DERIVATIVE "mc9s12c32"
#endif

#if defined (_HCS12DT)
  #pragma LINK_INFO DERIVATIVE "mc9s12dt256"
#endif

#if defined (_HCS12XDT)
  #pragma LINK_INFO DERIVATIVE "mc9s12xdt512"
#endif

#pragma CODE_SEG DEFAULT

void main(void)
{
    tRxPacket gsRxPacket;
    tTxPacket gsTxPacket;
    UINT8 gau8RxDataBuffer[8];
    UINT8 gau8TxDataBuffer[8];
    UINT8 u8RetryNo = 0;
    UINT16 u16Count = 0;
            UINT8 u8tmp;
  /* Initialize the packet.*/
    gsTxPacket.u8DataLength = 0;
    gsTxPacket.pu8Data = &gau8TxDataBuffer[0];
    gsRxPacket.u8DataLength = 0;
    gsRxPacket.pu8Data = &gau8RxDataBuffer[0];
    gsRxPacket.u8MaxDataLength = 8;
    gsRxPacket.u8Status = 0;

    MCUInit();
    MC13192Init();
            SCIInit();
            InitATD();

    /*******************************************************************
     *   To adjust output power call the
MLME_MC13192_PA_output_adjust() with:
     *
     *   MAX_POWER     (+3 to +5dBm)
     *   NOMINAL_POWER (0 dBm)
     *   MIN_POWER     ~(-16dBm)
     *
     *   or somewhere custom ? (0-15, 11 (NOMINAL_POWER) being Default
power)
       *
     ******************************************************************
     */

    MLMEMC13192PAOutputAdjust(MAX_POWER);    //Set MAX power setting
 /* MLMEMC13192PAOutputAdjust(MIN_POWER);    //Set MIN power setting */
 /* MLMEMC13192PAOutputAdjust(NOMINAL_POWER);    //Set Nominal power
setting */

    /* Init LED's */
    LED1 = 1; /* Default is off */
    LED2 = 1;
    LED3 = 1;
    LED4 = 1;

    LED1DIR = 1; /*Output*/
    LED2DIR = 1;
    LED3DIR = 1;
    LED4DIR = 1;

    PB1PU = 1;
    PB1DIR = 0;

    MLMESetMC13192ClockRate(0);  /* Set initial Clk speed */

    #if defined (_HCS12C)
      UseExternalClock();    /* switch clock sources */         //JCB
    #endif
    LED1 = 1;
    LED2 = 1;
    LED3 = 1;
    LED4 = 1;
    LED1DIR = 1;
    LED2DIR = 1;
    LED3DIR = 1;
    LED4DIR = 1;



    SCITransmitStr("\r\rWireless Typematic Demo\n\r");

    EnableInterrupts;

    gi8AppStatus = INITIAL_STATE;    /* Initial Mode */
    if (MLMESetChannelRequest(15) == SUCCESS)
    {
      gi8AppStatus= RECEIVER_ALWAYS_ON;
    }

    for(;;)
     {

        switch (gi8AppStatus)
         {

            case IDLE_STATE:
                /* Switch to RECEIVER_ALWAYS_ON */
                gi8AppStatus= RECEIVER_ALWAYS_ON;
                break;

            case RECEIVER_ALWAYS_ON:
                    u8RetryNo = 0;
                    MLMERXEnableRequest(&gsRxPacket, 0);
                    gi8AppStatus = WAITING_FOR_ACK;
                    LOW_POWER_WHILE();
                break;

            case WAITING_FOR_ACK:
                /* Do nothing.  Go to sleep waiting for TO or RX_IRQ */
                break;

            case TRANSMIT_DATA:
                gi8AppStatus= IDLE_STATE;

                if (MLMERXDisableRequest() != SUCCESS) { /* Turn off
the RX forever mode. */
                    gi8AppStatus= TRANSMIT_DATA;
                    break;
                }

                gau8TxDataBuffer[0] = gu8ATDChannel6Data;    /* Load
channel 6 (left wheel) into TX packet */
                gau8TxDataBuffer[1] = gu8ATDChannel7Data;    /* Load
channel 7 (right wheel) into TX packet */
                gsTxPacket.u8DataLength = 2;
                if ((MCPSDataRequest(&gsTxPacket) == SUCCESS))  /* 
transmit data */
                {
                    gi8AppStatus = WAITING_FOR_ACK;
                    MLMERXEnableRequest(&gsRxPacket,TIMEOUT_PERIOD);
                    u16Count = 0;
                }

                gu8SCIDataFlag = 0;           /* Clear data ready flag 
*/
                gu8ATDReadyFlag = 0;
                ResetATD();
                break;

            case TRANSMIT_ACK:
                gi8AppStatus= RECEIVER_ALWAYS_ON;
                gau8TxDataBuffer[0] = 'A';
                gau8TxDataBuffer[1] = 'C';
                gau8TxDataBuffer[2] = 'K';
                gsTxPacket.u8DataLength = 3;
                MCPSDataRequest(&gsTxPacket); /* transmit data */
                break;

            case TIMEOUT_STATE:
                if (u8RetryNo < RETRY_COUNT)
                {
                    gi8AppStatus= TRANSMIT_DATA;    /* Retransmit. */
                    switch (u8RetryNo % 4)
                    {
                        case 0x00:
                            LED1 = 0;
                            LED2 = 1;
                            LED3 = 1;
                            LED4 = 1;
                            u8RetryNo++;
                            break;
                        case 0x01:
                            LED1 = 1;
                            LED2 = 0;
                            LED3 = 1;
                            LED4 = 1;
                            u8RetryNo++;
                            break;
                        case 0x02:
                            LED1 = 1;
                            LED2 = 1;
                            LED3 = 0;
                            LED4 = 1;
                            u8RetryNo++;
                            break;
                        case 0x03:
                            LED1 = 1;
                            LED2 = 1;
                            LED3 = 1;
                            LED4 = 0;
                            u8RetryNo++;
                            break;
                    }
                } else
                {
                    /* Give up on packet. */

                    LED1 = 1;
                    LED2 = 1;
                    LED3 = 1;
                    LED4 = 1;
                    gi8AppStatus= RECEIVER_ALWAYS_ON;
                    u8RetryNo = 0;
                }
        }



        if (gu8SCIDataFlag == 1 || gu8ATDReadyFlag)
        {
            gi8AppStatus = TRANSMIT_DATA;
        }
    }
}

void MCPSDataIndication(tRxPacket *gsRxPacket)
{
    /*
     * Place your code here to handle a mac layer data indication.
     * RX packet is in the global structure
     * gsRxPacket.dataLength and gsRxPacket.data
       */
    if (gsRxPacket->u8Status == SUCCESS)
    {
        /* Packet received */
        if (gsRxPacket->pu8Data[0] == 'A' && gsRxPacket->pu8Data[1] ==
'C' && gsRxPacket->pu8Data[2] == 'K')
        {
            if (gi8AppStatus== WAITING_FOR_ACK)
            {
                LED1 = 1;
                LED2 = 1;
                LED3 = 1;
                LED4 = 1;
                gi8AppStatus= RECEIVER_ALWAYS_ON; /* go back to
rx_mode. */
            }
        } else    /* Not an ACK */
        {
            SCITransmitStr(&gsRxPacket->pu8Data[0]);
            gi8AppStatus = TRANSMIT_ACK;
        }

    }
    if (gsRxPacket->u8Status == TIMEOUT)
    {
        /* Received TIMEOUT */
        gi8AppStatus = TIMEOUT_STATE;
    }
}

void MLMEMC13192ResetIndication(void)
{
    //Notifies you that the MC13192 has been reset.
    //Application must handle this here.

}
```

```
// ATD code
// Created by David Schamber
#include "ATD.h"

#define POWERUP_DELAY 200

UINT8 gu8ATDReadyFlag;
UINT8 gu8ATDChannel6Data;
UINT8 gu8ATDChannel7Data;

// Initializes ATD channel 6 & 7
void InitATD(void) {
  int i = 0;

  TSCR1_TEN = 1;          // Enable timer module

  ATDCTL2_ADPU = 1;       // Turn on A/D unit

  // Wait for module to power up
  for(i = 0; i < POWERUP_DELAY; i++) {;}

  ATDCTL2_AFFC = 0;       // Use normal clear mode
  ATDCTL2_AWAI = 0;       // Do not power down in wait mode
  ATDCTL2_ASCIE = 1;      // Enable sequence complete interrupt
  ATDCTL2_ETRIGE = 0;     // Disable external trigger

  ATDCTL3_S8C = 0;        // Use 2 conversions per sequence
  ATDCTL3_S4C = 0;
  ATDCTL3_S2C = 1;
  ATDCTL3_S1C = 0;

  ATDCTL4_SRES8 = 1;      // Use 8-bit resolution
  ATDCTL4_SMP0 = 0;       // Select 2 conversions clock periods
  ATDCTL4_SMP1 = 0;
  ATDCTL4_PRS0 = 1;       // Divide bus clock by 12
  ATDCTL4_PRS1 = 0;
  ATDCTL4_PRS2 = 1;
  ATDCTL4_PRS3 = 0;
  ATDCTL4_PRS4 = 0;

  ATDCTL5_DJM = 1;        // Right justified data
  ATDCTL5_DSGN = 0;       // Unsigned data
  ATDCTL5_MULT = 1;       // Use multi-channel mode
  ATDCTL5_SCAN = 0;       // Use single conversion sequence mode
  ATDCTL5_CC = 1;         // Set channel 6 as the first channel in the
sequence
  ATDCTL5_CB = 1;
  ATDCTL5_CA = 0;

  ATDDIEN_IEN6 = 0;       // Disable channel 6 digital input
  ATDDIEN_IEN7 = 0;       // Disable channel 7 digital input

  DDRAD_DDRAD6 = 0;       // Set data direction registers
  DDRAD_DDRAD7 = 0;

  return;
}
```

```
// Resets ATD so it can perform another conversion
void ResetATD(void) {
  ATDCTL5_DJM = 1;        // Right justified data
  ATDCTL5_DSGN = 0;       // Unsigned data
  ATDCTL5_MULT = 1;       // Use multi-channel mode
  ATDCTL5_SCAN = 0;       // Use single conversion sequence mode
  ATDCTL5_CC = 1;         // Set channel 6 as the first channel in the 
sequence
  ATDCTL5_CB = 1;
  ATDCTL5_CA = 0;
}

// Handles ATD conversion sequence complete interrupt
interrupt void ATD_ISR(void){
  if(ATDSTAT1_CCF0){
    gu8ATDChannel6Data = ATDDR0L;       // Save channel 6 result
    ATDSTAT1_CCF0 = 1;                  // Clear channel 6 complete 
flag
  }
  if(ATDSTAT1_CCF1){
    gu8ATDChannel7Data = ATDDR1L;       // Save channel 7 result
    ATDSTAT1_CCF1 = 1;                  // Clear channel 7 complete 
flag
  }

  gu8ATDReadyFlag = 1;                  // Signal that ATD data is 
ready to be sent

  ATDSTAT0_SCF = 1;                     // Clear conversion complete 
flag

  return;
}
```

Receiver

```
/**********************************************************************
******
Wireless_UART.c

Application Note: This simple application forms a point to point
wireless uart with ACKs.
Power either the 13192SARD or GB60 EVB with a MC13192 2.2 IC and
connect the UART1 on the EVB and
Uart1 for 13192SARD.  If you have legacy ARD boards this by default
must use UART2.

UNZIP this in the apps directory with other app demos.

Download the same code to 2 boards.

Set the Hyperterms for 38400 No flow control.
Type a character on one of the hyperterms.  It will send the character
via MC13192 to
the other board and display it on its respective Hyperterm.  It will be
displayed and the rx
board will send back "ACK".  The TX board will be listening for the
ACK.  On every timeout
the TX will retransmit the character.  The number of successive tries
is reflected
on the LEDs.  1 light is one retry.  2 is two retries.  3 is three
retries etc.

If the ack is received the LEDs are reset.
If the ack is not received by the 5th retry the character is dropped
and the
LEDs are reset.

The timeout period is controlled by TIMEOUT_PERIOD
The number of retries is kept in RETRY_COUNT.  Note retries past 5 will
reset

Also, on the GB60EVBs COM_SW 4 must be on to allow the RX on the UART2
to work.

$Author: a20639 $
$Date: 2005/08/11 20:46:23 $
$Name:  $
**********************************************************************
*****/
#include <hidef.h> /* for EnableInterrupts macro */
#include "pub_def.h"
#include "simple_mac.h"
#include "SCI.h"
#include "mc13192_hw_config.h"
//#include "bootloader user api.h"
#include "wireless_uart.h"


//JCB
```

```
#include "drivers.h"
#include "simple_phy.h"
#include "MC13192_regs.h"
#include "mcu_hw_config.h"
#include "mcu_spi_config.h"


/////////////////////////////
/* note: Buad Rate = 38400 */
/////////////////////////////

/*Defines*/

/*Global Variables*/
UINT8 gu8RTxMode;

UINT8 echo;
UINT8 echoData[2];
int right_wheel, left_wheel;

extern UINT8 gu8SCIDataFlag;
extern UINT8 gu8SCIData[2];
UINT8 SCI_TestTx = 0;
INT8 gi8AppStatus = 0;

#ifdef BOOTLOADER_ENABLED
    #define TIMEOUT_PERIOD 0x2000 /*Changed to 0x2000 due to lower UART
baud rate. 28.09.04 MVC*/
#else
    #define TIMEOUT_PERIOD 0x1000
#endif BOOTLOADER_ENABLED

#define RETRY_COUNT 4

#if defined (_HCS12C)
  #pragma LINK_INFO DERIVATIVE "mc9s12c32"
#endif

#if defined (_HCS12DT)
  #pragma LINK_INFO DERIVATIVE "mc9s12dt256"
#endif

#if defined (_HCS12XDT)
  #pragma LINK_INFO DERIVATIVE "mc9s12xdt512"
#endif

#pragma CODE_SEG DEFAULT

void main(void)
{
    tRxPacket gsRxPacket;
    tTxPacket gsTxPacket;
    UINT8 gau8RxDataBuffer[8];
    UINT8 gau8TxDataBuffer[8];
    UINT8 u8RetryNo = 0;
    UINT16 u16Count = 0;
            UINT8 u8tmp;
  /* Initialize the packet.*/
    gsTxPacket.u8DataLength = 0;
    gsTxPacket.pu8Data = &gau8TxDataBuffer[0];
    gsRxPacket.u8DataLength = 0;
    gsRxPacket.pu8Data = &gau8RxDataBuffer[0];
    gsRxPacket.u8MaxDataLength = 8;
    gsRxPacket.u8Status = 0;

    MCUInit();
    MC13192Init();
            SCIInit();
    /*******************************************************************
     *   To adjust output power call the
MLME_MC13192_PA_output_adjust() with:
     *
     *   MAX_POWER     (+3 to +5dBm)
     *   NOMINAL_POWER (0 dBm)
     *   MIN_POWER     ~(-16dBm)
     *
     *   or somewhere custom ? (0-15, 11 (NOMINAL_POWER) being Default
power)
       *
     *****************************************************************
     */

    MLMEMC13192PAOutputAdjust(MAX_POWER);    //Set MAX power setting
 /* MLMEMC13192PAOutputAdjust(MIN_POWER);    //Set MIN power setting */
 /* MLMEMC13192PAOutputAdjust(NOMINAL_POWER);    //Set Nominal power
setting */

    /* Init LED's */
    LED1 = 1; /* Default is off */
    LED2 = 1;
    LED3 = 1;
    LED4 = 1;

    LED1DIR = 1; /*Output*/
    LED2DIR = 1;
    LED3DIR = 1;
    LED4DIR = 1;

    PB1PU = 1;
    PB1DIR = 0;

    MLMESetMC13192ClockRate(0);  /* Set initial Clk speed */

    #if defined (_HCS12C)
      UseExternalClock();    /* switch clock sources */         //JCB
    #endif

    LED1 = 1;
    LED2 = 1;
    LED3 = 1;
    LED4 = 1;
    LED1DIR = 1;
    LED2DIR = 1;
    LED3DIR = 1;
    LED4DIR = 1;

    SCITransmitStr("\r\rWireless Typematic Demo\n\r");

    EnableInterrupts;

    gi8AppStatus = INITIAL_STATE;    /* Initial Mode */
    if (MLMESetChannelRequest(15) == SUCCESS)
    {
      gi8AppStatus= RECEIVER_ALWAYS_ON;
    }

    for(;;)
     {

        switch (gi8AppStatus)
         {

            case IDLE_STATE:
                /* Switch to RECEIVER_ALWAYS_ON */
                gi8AppStatus= RECEIVER_ALWAYS_ON;
                break;

            case RECEIVER_ALWAYS_ON:
                    u8RetryNo = 0;
                    MLMERXEnableRequest(&gsRxPacket, 0);
                    gi8AppStatus = WAITING_FOR_ACK;
                    LOW_POWER_WHILE();
                break;

            case WAITING_FOR_ACK:
                /* Do nothing.  Go to sleep waiting for TO or RX_IRQ */
                break;

            case TRANSMIT_DATA:
                gi8AppStatus= IDLE_STATE;

                if (MLMERXDisableRequest() != SUCCESS) { /* Turn off
the RX forever mode. */
                    gi8AppStatus= TRANSMIT_DATA;
                    break;
                }

                if(echo == 1) {
                  gau8TxDataBuffer[0] = echoData[0];
                  gau8TxDataBuffer[1] = echoData[1];
                  gau8TxDataBuffer[2] = '\0';
                  gsTxPacket.u8DataLength = 3;
                  echo = 0;
                } else{
                  gau8TxDataBuffer[0] = gu8SCIData[0];    /* Load the
SCI data into gsTxPacket */
                  gau8TxDataBuffer[1] = '\0';    /* Sending String */
                  gsTxPacket.u8DataLength = 2;
                }
                if ((MCPSDataRequest(&gsTxPacket) == SUCCESS))  /*
transmit data */
                {
                    gi8AppStatus = WAITING_FOR_ACK;
                    MLMERXEnableRequest(&gsRxPacket,TIMEOUT_PERIOD);
                    u16Count = 0;
                }
                gu8SCIDataFlag = 0;
                break;

            case TRANSMIT_ACK:
                gi8AppStatus= RECEIVER_ALWAYS_ON;
                gau8TxDataBuffer[0] = 'A';
                gau8TxDataBuffer[1] = 'C';
                gau8TxDataBuffer[2] = 'K';
                gsTxPacket.u8DataLength = 3;
                MCPSDataRequest(&gsTxPacket); /* transmit data */
                break;

            case TIMEOUT_STATE:
                if (u8RetryNo < RETRY_COUNT)
                {
                    gi8AppStatus= TRANSMIT_DATA;    /* Retransmit. */
                    switch (u8RetryNo % 4)
                    {
                        case 0x00:
                            LED1 = 0;
                            LED2 = 1;
                            LED3 = 1;
                            LED4 = 1;
                            u8RetryNo++;
                            break;
                        case 0x01:
                            LED1 = 1;
                            LED2 = 0;
                            LED3 = 1;
                            LED4 = 1;
                            u8RetryNo++;
                            break;
                        case 0x02:
                            LED1 = 1;
                            LED2 = 1;
                            LED3 = 0;
                            LED4 = 1;
                            u8RetryNo++;
                            break;
                        case 0x03:
                            LED1 = 1;
                            LED2 = 1;
                            LED3 = 1;
                            LED4 = 0;
                            u8RetryNo++;
                            break;
                    }
                } else
                {
                    /* Give up on packet. */
                    LED1 = 1;
                    LED2 = 1;
                    LED3 = 1;
                    LED4 = 1;
                    gi8AppStatus= RECEIVER_ALWAYS_ON;
                    u8RetryNo = 0;
                }
        }



        if (gu8SCIDataFlag == 1)
        {
            gi8AppStatus= TRANSMIT_DATA;
        }
    }
}

void MCPSDataIndication(tRxPacket *gsRxPacket)
{
    /*
     * Place your code here to handle a mac layer data indication.
     * RX packet is in the global structure
     * gsRxPacket.dataLength and gsRxPacket.data
       */
    if (gsRxPacket->u8Status == SUCCESS)
    {
        /* Packet received */
        if (gsRxPacket->pu8Data[0] == 'A' && gsRxPacket->pu8Data[1] ==
'C' && gsRxPacket->pu8Data[2] == 'K')
        {
            if (gi8AppStatus== WAITING_FOR_ACK)
            {
                LED1 = 1;
                LED2 = 1;
                LED3 = 1;
                LED4 = 1;
                gi8AppStatus= RECEIVER_ALWAYS_ON; /* go back to
rx_mode. */
            }
        } else    /* Not an ACK */
        {
            SCITransmitStr(&gsRxPacket->pu8Data[0]);

            right_wheel  = gsRxPacket->pu8Data[0] - 128;
            left_wheel  = gsRxPacket->pu8Data[0] - 128;

            right_wheel -= gsRxPacket->pu8Data[1] - 128;
            left_wheel += gsRxPacket->pu8Data[1] - 128;

            if(right_wheel > 0) {
              RIGHT_FORWARD = 0;
              RIGHT_REVERSE = 1;
            } else{
              RIGHT_FORWARD = 1;
              RIGHT_REVERSE = 0;
              right_wheel *= -1;
            }

            right_wheel += RIGHT_OFFSET;

            if(right_wheel > 255)
              right_wheel = 255;

            PWM_RIGHT_DTY = right_wheel;

            if(left_wheel > 0) {
              LEFT_FORWARD = 0;
              LEFT_REVERSE = 1;
            } else{
              LEFT_FORWARD = 1;
              LEFT_REVERSE = 0;
              left_wheel *= -1;
            }

            left_wheel += LEFT_OFFSET;

            if(left_wheel > 255)
              left_wheel = 255;

            PWM_LEFT_DTY = left_wheel;

            gu8SCIDataFlag = 1;
            echo = 1;
            echoData[0] = gsRxPacket->pu8Data[0];
            echoData[1] = gsRxPacket->pu8Data[1];

            gi8AppStatus = TRANSMIT_ACK;
        }

    }
    if (gsRxPacket->u8Status == TIMEOUT)
    {
        /* Received TIMEOUT */
        gi8AppStatus = TIMEOUT_STATE;
    }
}

void MLMEMC13192ResetIndication(void)
{
    //Notifies you that the MC13192 has been reset.
    //Application must handle this here.

}
```

```
// port_config_C32.h
#include "mcu_hw_config.h"
#define MC13192_CE                PTM_PTM3
#define MC13192_CE_PORT           DDRM_DDRM3
#define MC13192_ATTN              PTT_PTT1
#define MC13192_ATTN_PORT         DDRT_DDRT1
#define MC13192_RTXEN             PTAD_PTAD1
#define MC13192_RTXEN_PORT        DDRAD_DDRAD1
#define MC13192_RESET             PTT_PTT0
#define MC13192_RESET_PORT        DDRT_DDRT0
#define MC13192_RESET_PULLUP      PERM_PERM0
#define MC13192_ANT_CTRL          PORTB_BIT4
#define MC13192_ANT_CTRL_PORT     DDRB_BIT4
#define MC13192_ANT_CTRL2         PORTB_BIT4
#define MC13192_ANT_CTRL2_PORT    DDRB_BIT4
#define ANT_CTRL_OFF              0
#define ANT_CTRL_ON               1

#define PWM_RIGHT_DDRT            DDRT_DDRT2
#define PWM_RIGHT_MODRR           MODRR_MODRR2
#define PWM_RIGHT_PWME            PWME_PWME2
#define PWM_RIGHT_PWMPOL          PWMPOL_PPOL2
#define PWM_RIGHT_PWMCAE          PWMCAE_CAE2
#define PWM_RIGHT_PERIOD          PWMPER2
#define PWM_RIGHT_DTY             PWMDTY2

#define PWM_LEFT_DDRT             DDRT_DDRT3
#define PWM_LEFT_MODRR            MODRR_MODRR3
#define PWM_LEFT_PWME             PWME_PWME3
#define PWM_LEFT_PWMPOL           PWMPOL_PPOL3
#define PWM_LEFT_PWMCAE           PWMCAE_CAE3
#define PWM_LEFT_PERIOD           PWMPER3
#define PWM_LEFT_DTY              PWMDTY3

#define LEFT_FORWARD              PTT_PTT4
#define LEFT_FORWARD_DDRT         DDRT_DDRT4
#define LEFT_REVERSE              PTT_PTT5
#define LEFT_REVERSE_DDRT         DDRT_DDRT5
#define RIGHT_FORWARD             PTT_PTT6
#define RIGHT_FORWARD_DDRT        DDRT_DDRT6
#define RIGHT_REVERSE             PTT_PTT7
#define RIGHT_REVERSE_DDRT        DDRT_DDRT7

#define LEFT_OFFSET               127
#define RIGHT_OFFSET              117

#define MC13192_IRQ_Disable()     INTCR_IRQEN = 0
#define MC13192_IRQ_Enable()      INTCR_IRQEN = 1

#if defined (_HCS12C)

#define IRQInit()                 INTCR = 0x40; // Enables Interrupt
PIN. IRQSC = 0x14
#define CLEAR_IRQ_FLAG()          asm nop;
#define IRQPinEnable()            asm nop;
#define IRQReadPin()                                PORTE_BIT1
#define AssertCE()                MC13192_CE = 0;
```

```
#define DeAssertCE()                  MC13192_CE = 1;
#define RTXENDeAssert()                                     MC13192_RTXEN =
0;
#define RTXENAssert()                                         MC13192_RTXEN =
1;

#endif
```

```
/**
 * Copyright (c) 2004, Freescale Semiconductor
 * Freescale Confidential Proprietary
 *
 * File name : mcu_hw_config.c
 * Project name: SMAC (Simple Media Access Controller)
 *
 * Department : Freescale Radio Products Division
 *
 * Description : MCU Hardware configuration routines.
 *
 * $Author: a20639 $
 * $Date: 2005/08/11 20:46:25 $
 * $Name:  $
 */

//#include "device_header.h" /* include peripheral declarations */
#include "pub_def.h"
#include "MC13192_regs.h"
#include "MC13192_hw_config.h"
#include "mcu_hw_config.h"
#include "drivers.h"
#include "hidef.h"


/* Global Variables */
extern UINT8 gu8RTxMode;
extern UINT8 gu8IRQF;


/*
 * UseExternalClock: Switch the MCU from internal to MC13192 supplied
clock.
 * The MCU FLL is not engaged.
 *
 * Parameters: None
 *
 * Return : None
 */
#if defined (_HCS12)
void UseExternalClock() //use_external_clock()
{
  CLKSEL_PLLSEL = 0;
  PLLCTL_PLLON  = 0;  /* PLL OFF, Takes BUSCLK from external oscillator
*/
}

/*
 * UseMcuClock: Switch the MCU from external to internal clock.
 *
 * Parameters: None
 *
 * Return : None
 */
void UseMcuClock() //use_mcu_clock()
{
  CLKSEL_PLLSEL = 0;
  PLLCTL_PLLON  = 0;    /* PLL OFF, Takes BUSCLK from external
oscillator */
  SYNR  = 0x00;                       /* 2 x OSCCLK x [SYNR + 1] */
  REFDV = 0x01;         /* (2 x OSCCLK x [SYNR + 1])/REFDV */
  PLLCTL |= 112;        /* PLLCTL: PLLON=1,AUTO=1,ACQ=1 */
                   /* */
  while(!CRGFLG_LOCK);  /* Wait */
  CLKSEL_PLLSEL = 1;    /* Select clock source from PLL */
}
#endif


/*
 * MC13192Restart: Restart the MC13192.
 *
 * Parameters: None
 *
 * Return : None
 */
void MC13192Restart()
{
    gu8RTxMode = SYSTEM_RESET_MODE;
    IRQInit();                  /* Turn on the IRQ pin. */
    MC13192_RESET = 1;          /* Take MC13192 out of reset */
    while (IRQReadPin() == 1);  /* Poll waiting for MC13192 to assert
the irq */
                                /* Empty Body */      /* (i.e. ATTN).
*/
    (void)SPIDrvRead(0x24);           /* Clear MC13192 interrupts */
    IRQACK();                   /* ACK the pending IRQ interrupt */
    IRQPinEnable();             /* Pin Enable, IE, IRQ CLR, negative
edge. */
}


/*
 * MC13192ContReset: Reset (continuous) the MC13192.
 *
 * Parameters: None
 *
 * Return : None
 */

void MC13192ContReset()
{
    gu8RTxMode = SYSTEM_RESET_MODE;
    IRQInit();                      /* Set for negative edge. */
    MC13192_RESET = 0;              /* Place the MC13192 into reset */
}


/*
 * GPIOInit: Initialize the MCU-to-MC13192 GPIO direction and data.
 *
 * Parameters: None
 *
 * Return : None
 */
void GPIOInit()
{
   PWM_LEFT_DDRT = 1;
   PWM_LEFT_MODRR = 1;
   PWM_LEFT_PWME = 1;
   PWM_LEFT_PWMPOL = 1;
   PWM_LEFT_PWMCAE = 1;
   PWM_LEFT_PERIOD = 255;
   PWM_LEFT_DTY = LEFT_OFFSET;    // not moving

   LEFT_FORWARD = 1;
   LEFT_FORWARD_DDRT = 1;
   LEFT_REVERSE = 0;
   LEFT_REVERSE_DDRT = 1;

   PWM_RIGHT_DDRT = 1;
   PWM_RIGHT_MODRR = 1;
   PWM_RIGHT_PWME = 1;
   PWM_RIGHT_PWMPOL = 1;
   PWM_RIGHT_PWMCAE = 1;
   PWM_RIGHT_PERIOD = 255;
   PWM_RIGHT_DTY = RIGHT_OFFSET;    // not moving

   RIGHT_FORWARD = 0;
   RIGHT_FORWARD_DDRT = 1;
   RIGHT_REVERSE = 1;
   RIGHT_REVERSE_DDRT = 1;

   MC13192_RESET_PULLUP = 0;
   MC13192_CE = 1;
   MC13192_ATTN = 1;
   MC13192_RTXEN = 0;
   MC13192_RESET = 0;                      /* initially reset MC13192 */
   MC13192_RESET_PORT = 1;
   MC13192_CE_PORT = 1;
   MC13192_ATTN_PORT = 1;
   MC13192_RTXEN_PORT = 1;
//   MC13192_RESET = 1;                    /* Now let go of reset
MC13192 */       //JCB

    #if defined (ANTENNA_SWITCH)
        MC13192_ANT_CTRL2_PORT = 1;         /* Output for antenna port
RX */
        MC13192_ANT_CTRL_PORT = 1;         /* Output for antenna port TX
*/
        MC13192_ANT_CTRL2 = 1;           /* Signal to turn on RX antenna
*/
        MC13192_ANT_CTRL = 1;            /* Signal to turn on TX antenna
*/
    #endif

    #if defined (LNA)
        MC13192_LNA_CTRL = LNA_OFF;      /* Turn off the LNA out of
reset */
        MC13192_LNA_CTRL_PORT  = 1;      /* Enable the port for OUTPUT
*/
```

```
    #endif

    #if defined (PA)
        MC13192_PA_CTRL = PA_OFF;       /* Turn off the PA out of Reset
*/
        MC13192_PA_CTRL_PORT = 1;       /* Enable the port for OUTPUT
*/
    #endif

}


/*
 * MCUInit: Initialize the MCU COP, GPIO, SPI and IRQ.
 * Set the desired MC13192 clock frequency here.
 *
 * Parameters: None
 *
 * Return : None
 */
void MCUInit(void)
{
    UINT16 u16IrqReg =0;
    UINT8 u8AttnIrq = FALSE;
    UINT8 u8Timer;

    #if defined (_HCS12C)         //JCB
        _DISABLE_COP();             /* Turn off the watchdog. */
      #endif

      #if defined (_HCS12DT) || defined (_HCS12XDT)       //JCB
        COPCTL = 0;

      CLKSEL = 0;
      PLLCTL = 0x40;          //PLL OFF
      SYNR = 1;               // 2 x OSCCLK x [SYNR + 1]
      REFDV = 0;              // (2 x OSCCLK x [SYNR + 1])/REFDV
      PLLCTL = 0x70;          // PLLCTL: PLLON=1,AUTO=1,ACQ=1
      while(!CRGFLG_LOCK);    // Wait
      CLKSEL_PLLSEL = 1;      // Select clock source from PLL

      #endif

    gu8RTxMode = RESET_DELAY;

    /* Add a delay to debouce the reset switch on development boards
~200ms */
    TSCR1 = 0x80;                                          /* Timer
Enable */
                                 /*
                                  *  0b10000000
                                  *    ||||||||__ Allows the timer to
function normally.TIMER Enable
                                  *     |||||||___ Allows the timer
module to continue running during wait.
                                  *      ||||||____ Allows the timer
counter to continue running while in freeze mode.
```

```
                                  *       |||||_____ Active-high SPI
clock.
                                  *        ||||______ Allows the timer
flag clearing to function normally.
                                  *         |||_______ Unimplemented
                                  *          ||________ Unimplemented
                                  *           |_________ Unimplemented
                                  */
    TSCR2 = 0x05;                                          /* Set the
Timer module to use BUSCLK as
                                 * reference with Prescaler at / 32
                                 */
    do
    {
        u8Timer = TCNTLo;         /* Get value of timer register (LOW
byte) */
    } while (u8Timer <= 0x80);    /*
                                   * Poll for TIMER LO to be greater
than
                                   * 0x80 at 4MHz/32
                                   */
    TSCR2 = 0x00;                 /* Return to reset values */

    gu8RTxMode = SYSTEM_RESET_MODE;
    GPIOInit();
    SPIInit();
    IRQInit();                   /* Turn on the IRQ pin. */
    gu8RTxMode = MC13192_RESET_MODE;
    MC13192_RESET = 1;          /* Take MC13192 out of reset */

    while (u8AttnIrq == FALSE)
     {
       if (IRQReadPin() == 0)  /* Check to see if IRQ is asserted */
        {
            u16IrqReg = SPIDrvRead(0x24);   /*
                                             * Clear MC13192 interrupts
and
                                             * check for ATTN IRQ from
13192
                                             */
            u16IrqReg &= 0x400;
            if (u16IrqReg == 0)
            {
                u8AttnIrq = FALSE;
            }
            else
            {
                u8AttnIrq = TRUE;
            }
        }
     }

    IRQPinEnable();        /* Pin Enable, IE, IRQ CLR, negative edge.
*/
    gu8RTxMode = MC13192_CONFIG_MODE;

  return;
```

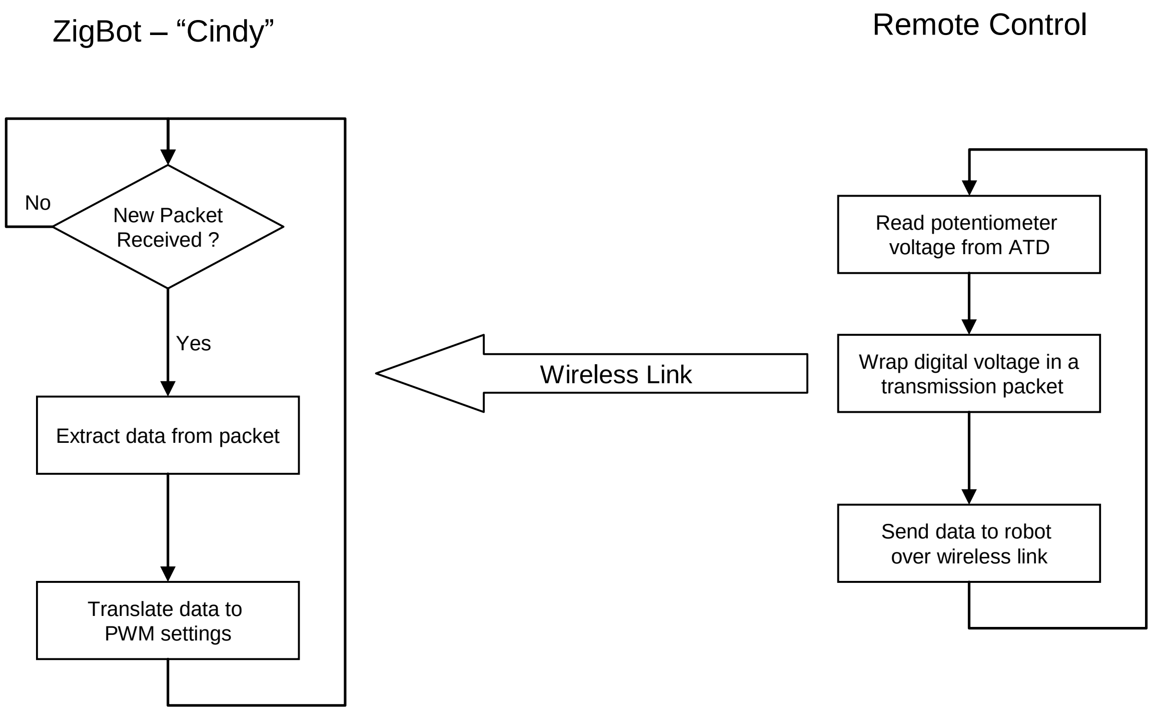
```
}

/*
 * IRQPinLow: Checks IRQ Pin to see if is low.
 *
 * Parameters: None
 *
 * Return : 1 if IRQ is Low.
 */

UINT8 IRQPinLow(void)
{
   if(IRQReadPin()==0)
   {
       return 0;
   }

    return 1;
}
```

Appendix B: Flowchart



ZigBot – “Cindy”
Remote Control
No
New Packet Received ?
Yes
Extract data from packet
Translate data to PWM settings
Wireless Link
Read potentiometer voltage from ATD
Wrap digital voltage in a transmission packet
Send data to robot over wireless link