WIRELESS GESTURE CONTROLLED TANK TOY

Report for ECE 4760 project for school of electrical and computer engineering

By,

Rick Wong (rw363)

Professor: Bruce Land

Date:

2011-05-10

TABLE OF CONTENTS

LIST OF FIGURES

1 ABSTRACT

The objective of this project is to build a tank car that can be controlled by gesture wirelessly. User is able to control motions of the tank by wearing the controller glove and performing predefined gestures. This tank can detect block objects and stop automatically; in addition, feedback messages are sent to the controller and warn the user by a vibration motor. This project provides a basic platform for many potential applications such as wireless controlled car racing games, gesture human-machine interfacing, and etc.

For this project, ATMega168 microcontroller and gyro scope are employed for the controller; ATMega168, H-bridge, and ultra-sonic sensor are employed for the controlled tank. A pair of wireless UART module, Wi.232, is used to communicate between the controller and tank. However, the hardware is also ready for ZigBee wireless protocol.

2 INTRODUCTION

2.1 Project Overview

Most of controllers of existing remote toys, as shown in Figure 1, require users to interface with joysticks and push buttons. Comparing to these conventional controllers, I built a wireless gesture controller which enables toys to mock hand motions in all three dimensions as shown in Figure 2. To demonstrate this wireless gesture controller, a remote tank is also implemented, as shown in Figure 3.

Figure 1 Conventional Wireless Controller

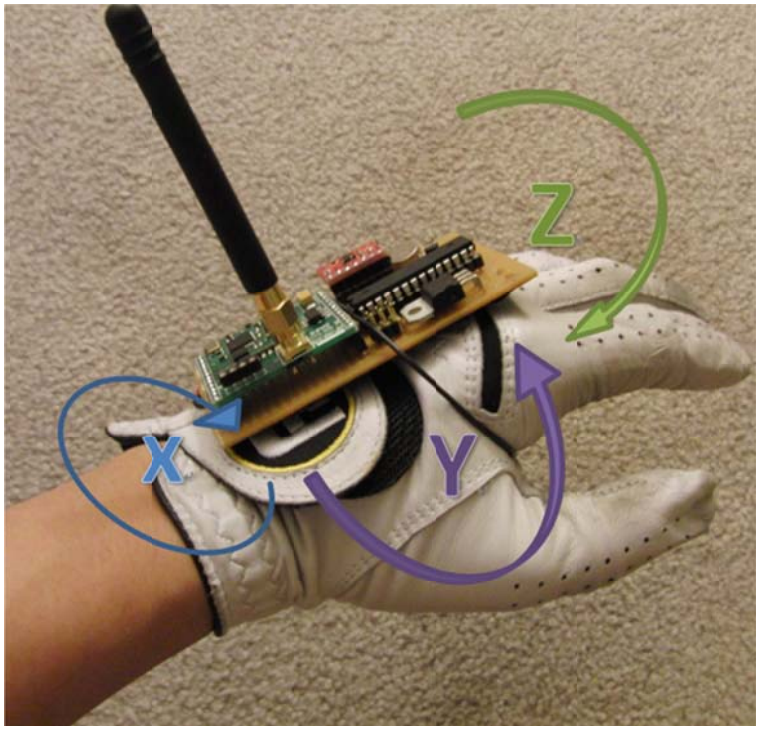


Figure 2 Gesture Wireless Controller

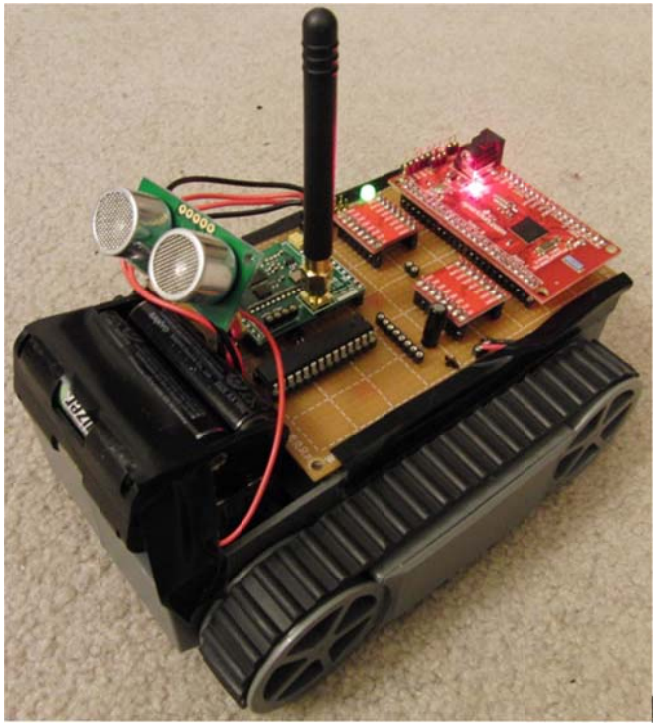


Figure 3 Remote Controlled Tank

2.2 System Block Diagram

The below overall system block diagram illustrates the structure of the system, the modules and the communication protocols between them.

The whole is divided into four main parts: Remote Tank and Gesture Controller as described below. A pair of wireless-serial module communicates between these two parts.

As shown in Figure 4, the microcontroller, MCU collects angular acceleration data from the gyro scope and translates these motion data into corresponded commands which control the motors on the remote tank before sending these commands to the wireless-serial module via UART protocol.

The remote tank reads the commands sent by the gesture controller via UART protocol from the wireless-serial module and performs the required motor controls.

On the other hand, feedbacks from the ultra-sonic senor and encoder are sent from the remote tank back to the gesture controller wirelessly as the way the gesture controller sends commands.

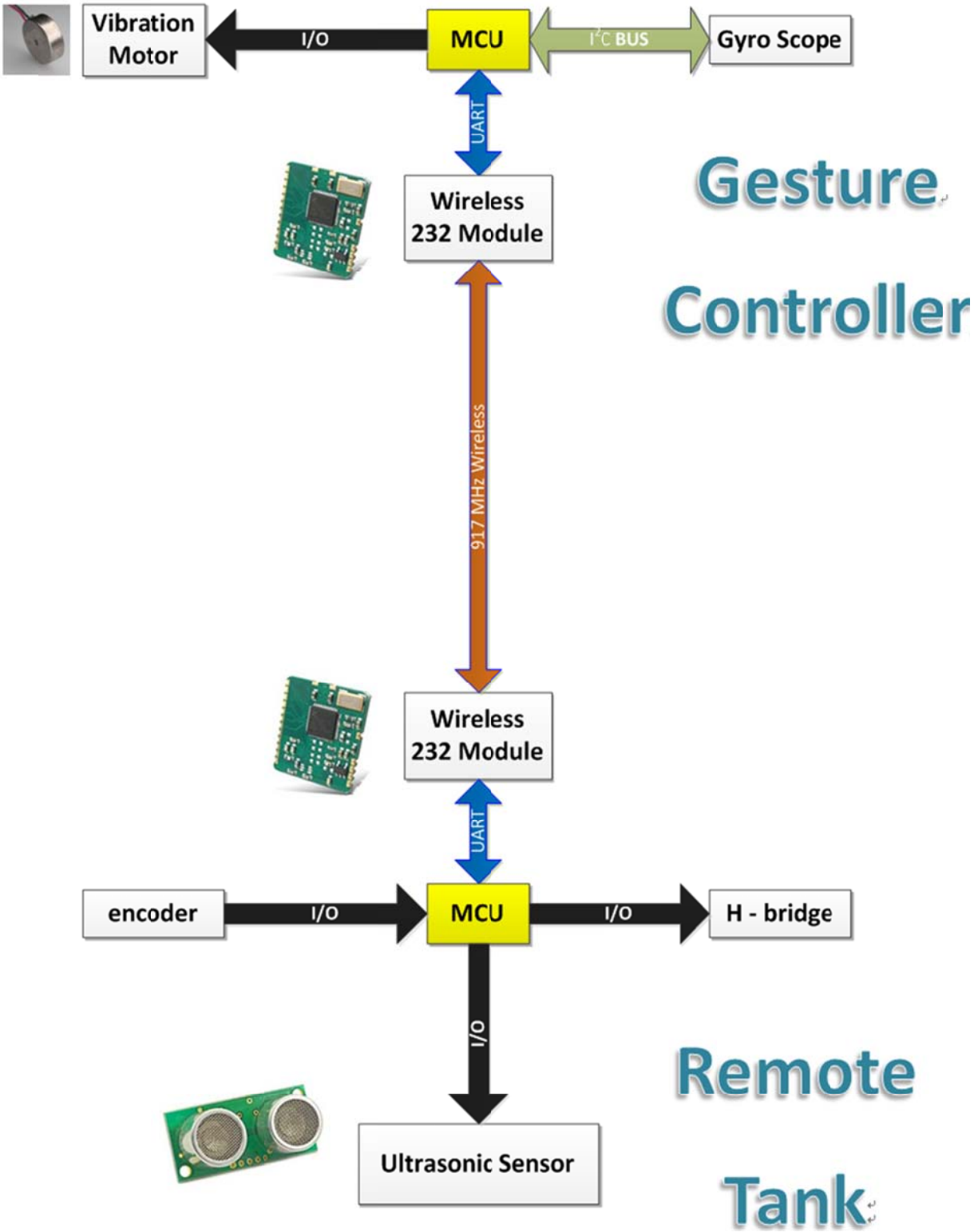


Figure 4 System Block Diagram

3 MAJOR COMPONENTS

In this chapter, the major components are introduced.

3.1 ATMega168

Similar to other AVR microcontrollers, including the ATMega644 used in ECE 4760, ATMega168 is a member of the AVR MCU family from ATMEL Inc. It is one of the ideal MCU for simple and inexpensive embedded applications. The main reason I chose this chip is that I have a couple of them denoted for free and clearly it has the enough performance to do the expected jobs. This MCU is briefly introduced and unnecessary details are skipped due to the similarities shared with the ATMeg644.

3.2 Features

Figure 5 is the system block diagram of the ATMega168 MCU used in this project.

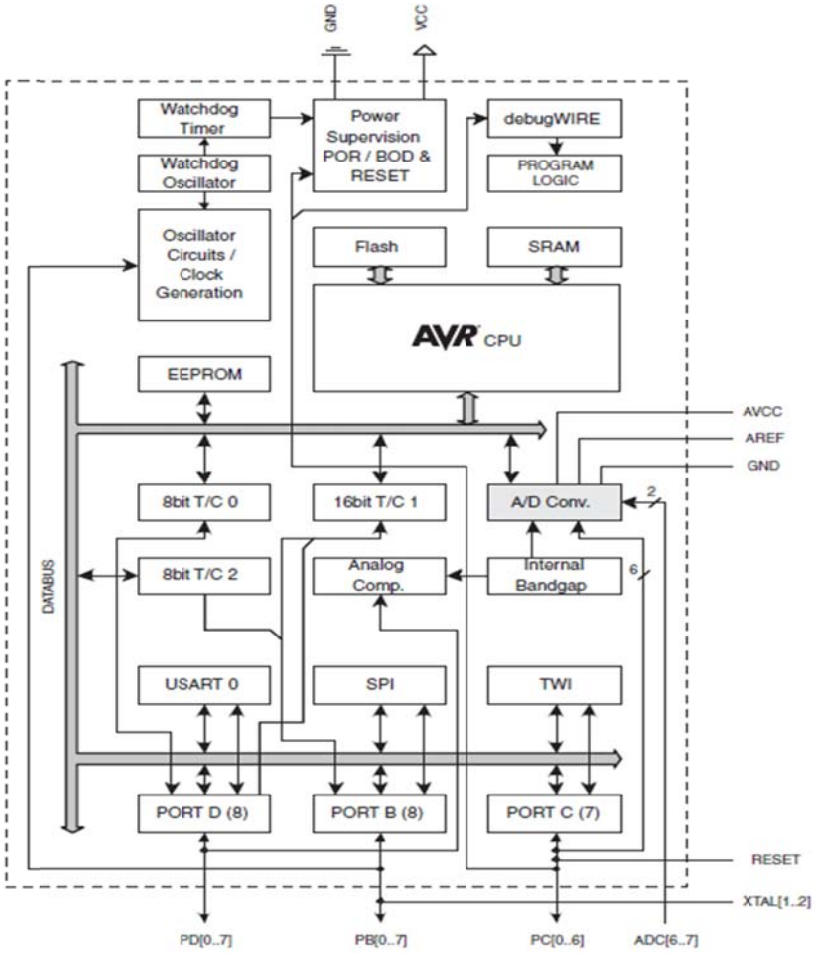


Figure 5 ATMega168 system block diagram

The features of the MCU that relate to this project are listed below.

- High Performance, Low Power AVR® 8-Bit Microcontroller
- 16K Bytes of In-System Self-programmable Flash program memory
- 1K Bytes Internal SRAM
- In-System Programming by On-chip Boot Program
- Two 8-bit Timer/Counters with separate prescaler and compare mode
- 16-bit Timer/Counter with separate prescaler, compare mode, and capture mode
- Six PWM Channels
- Programmable Serial USART
- Byte-oriented 2-wire Serial Interface (Philips I2C compatible)
- Programmable Watchdog Timer with Separate On-chip Oscillator
- Interrupt and Wake-up on Pin Change
- Power-on Reset and Programmable Brown-out Detection

3.3 Gyro Scope

The gyro scope used in this project is the ITG-3200 from InvenSense Inc. ITG-3200 was the world's first single-chip, digital-output, 3 axis MEMS gyro scope that has built-in temperature sensor for user calibrations. It converts the low-pass filtered angular acceleration data using the three built-in 16-bit analog-to-digital converter and outputs the digital data via I^2C protocol.

The features of this gyro scope that relate to this project are listed below (abstracted from the ITG-2300 datasheet from InvenSense Inc.)

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyros) on one integrated circuit with a sensitivity of 14.375 LSBs per °/sec and a full-scale range of ±2000°/sec
- Three integrated 16-bit ADCs provide simultaneous sampling of gyros while requiring no external multiplexer
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Low frequency noise lower than previous generation devices, simplifying application development and making for more-responsive motion processing
- Low 6.5mA operating current consumption for long battery life

3.4 Wi.232DTS Wireless-Serial Module

The Wi.232DTS wireless-serial module from Radiotronix Inc. is a simple and inexpensive wireless communication solution that supports CRC error checking, programmable UART rate, and multiple transmission channels. In short, it basically acts like a standard UART

cable. The only tricky part of using this module is that the UART RXD0 pin from the MCU shall be connected to the TXD0 pin from this module, and the UART TXD0 pin from the MCU shall be connected to the RXD0 pin from this module. The pin-out of this module is shown in Figure 6 and Table 1. Please note that this figure and table are from the ITG-3200 datasheet from Radiotronix Inc.

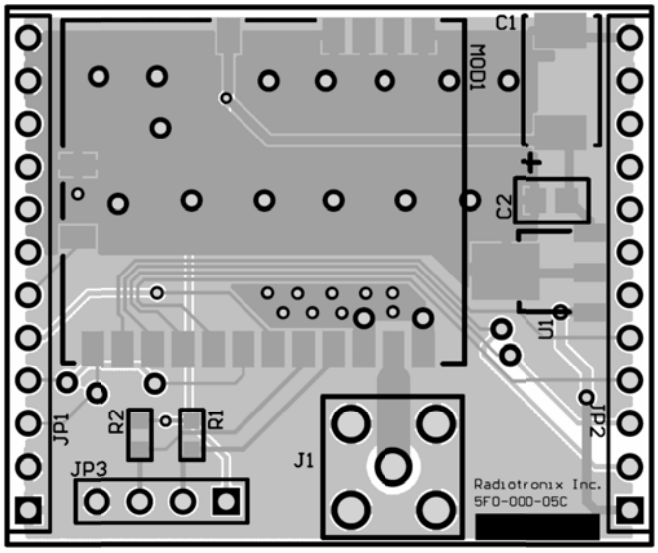


Figure 6 Wi.232 Wireless-Serial Module

Pin Name	Pin Description
JP2 Pin 1	4V to 12V Vdd
JP2 Pin 2	RxD0, Wi.232DTS RxD pin
JP2 Pin 3	TXD0, WI.232DTS TXD PIN
JP2 Pin 4	CTS0, Wi.232DTS CTS pin
JP2 Pin 5	CMD (Active Low), Wi.232DTS CMD pin
JP2 Pin 6	RxD1 – Reserved for Future – No Connect
JP2 Pin 7	TxD1 – Reserved for Future – No Connect
JP2 Pin 8	RTS1 – Reserved for Future – No Connect
JP2 Pin 9	DTR1 – Reserved for Future – No Connect
JP2 Pin 10	DTR1* - Reserved for Future – No Connect
JP2 Pin 11	GND
JP2 Pin 12	GND
JP1 Pin 1	GND
JP1 Pin 2	GND
JP1 Pin 3	AD1 – Reserved for Future – No Connect
JP1 Pin 4	AD0 – Reserved for Future – No Connect
JP1 Pin 5	RSSI – Reserved for Future – No Connect
JP1 Pin 6	PD5 – Reserved for Future – No Connect
JP1 Pin 7	PD4 – Reserved for Future – No Connect
JP1 Pin 8	PD3 – Reserved for Future – No Connect
JP1 Pin 9	PD2 – Reserved for Future – No Connect
JP1 Pin 10	PD1 – Reserved for Future – No Connect
JP1 Pin 11	GND
JP1 PIN 12	GND

Table 1 Wi.232 Wireless-Serial Module Pin-out

3.5 SRF05 Ultra-Sonic Distance Sensor

The ultra-sonic distance sensor, SRF05, used in this project has a detection range from 3 cm up to 4 meters. It supports dual-pin and single-pin modes and the single-pin configuration as shown in Figure 7 is used in this project to simplify the hardware connection.

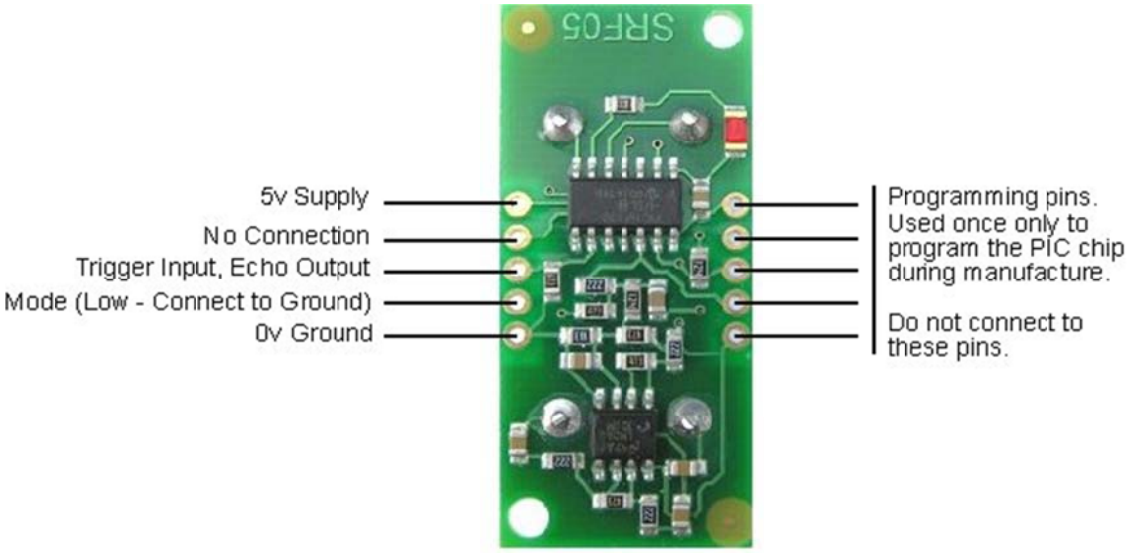


Figure 7 SRF05 Ultra-Sonic Senor

Using this sensor is fairly easy. The Echo output of this senor is connected to the input capture channel of the ATMega168 and timer1 of the MCU is used to measure the timer period echo pulse which is used to calculate the distance using Equation 1. The control trigger pulse and resulting echo pulse timing diagraming is shown in Figure 8.

$$\text{Distance} = \text{echo pulse time} * \frac{0.13736}{\text{CPU speed}}$$

Equation 1. Calculate distance from pulse time

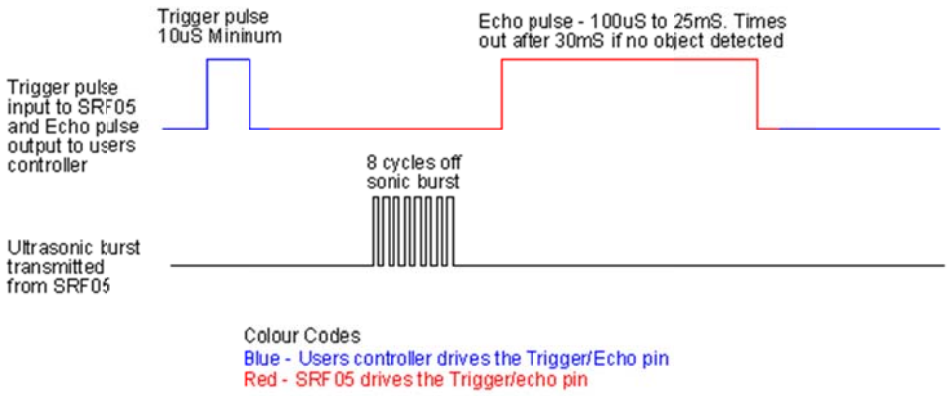


Figure 8 SRF05 Timing Diagram

4 SOFTWARE HIGHLIGHTS

In this project, more than 2,000 lines have been coded; discussions of all the code are obviously unpractical and meaningless. Therefore, in this chapter, highlights of the software development are presented.

4.1 Software Reset using Watch-Dog

For reliability and safety concerns, the remote tank shall be able to be reset wireless in case of system hang or other extreme situations. To archive that, I employed a watch-dog in the ATMega168 to hot reboot the MCU whenever the system hangs or received reset command from the gesture controller. The reboot function is shown below. Please note that after calling reboot function, the MCU will fall into a reboot loop unless the watch-dog is disabled within the pre-set time after MCU reboot.

```
/*---------------------------------------------------------------------
  * Function details:        reboot the chip
  * Name:                    reboot()
  * Usage:                   call this function to reset the chip
  * Input:                   none
  * Return:                  none
  * Attention:               TO BE UPDATED
  * Notes:                   none
---------------------------------------------------------------------*/
void reboot(void)
{
   wdt_enable(WDTO_15MS);
   while (1) {;}
}

//watch dog define
#define DOG_SLEEP
{MCUSR &= (~(SET<<WDRF));WDTCSR|=((SET<<WDCE)|(SET<<WDE));WDTCSR=CLEAR;}

Chip_Init()
{
//disable watch dog
DOG_SLEEP;
……
}
```

4.2 Smooth Motor Control with Safety Features

Safety concerns are always carried out through the development of this project. Considering the possible break-down of the wireless communication, the remote tank only acts when there is input command sent from the gesture controller. Whenever there is break-down on the wireless transmission and the remote tank receive no further command signal, the remote tank stops until communication is reestablished. In addition, a connection password check is performed upon receiving the wireless package before any command is accepted by the remote tank.

Additionally, the remote tank stops immediately whenever a block object is detected at 30 cm from the front of the tank, sends warning message to the gesture controller, and notify the user by turning on the vibration motor.

An issue created by these safety checks is that the motor does not rotate continuously since it waits a new command and does all the safety check routines. To solve this issue, I let the MCU exam the safety conditions and analyze the received command while the motor rotates for a short period of time, as shown in below code.

```
 for (motor_count = CLEAR; motor_count < PWM_OUT_Count; motor_count++)
{
    if ((distance) && (distance<MIN_DISTANCE) && (ADIR_MASK & 
U_RArray[SUB_COMMAND]) && (BDIR_MASK & U_RArray[SUB_COMMAND]))
        {
            Clean_Up();
            ACM_Status=STATE_ACM_FREE;
        break;
     }
  if (USART_REVD)
  {
   ACM_Status = STATE_COMMAND_REVD;
   break;
  }
  _delay_ms(MOTOR_RUN_TIME);
}
```

5 Circuit

5.1 Original Plan - PCB

Initially, I intended to use ATMega128RFA1 as the MCU as it has built-in wireless communication ability via IEEE 802.15.4 protocol. I designed the Printed-Circuit-Board as shown in below Figure 9 to Figure 11 since using PCBs is much more reliable and occupies smaller space.

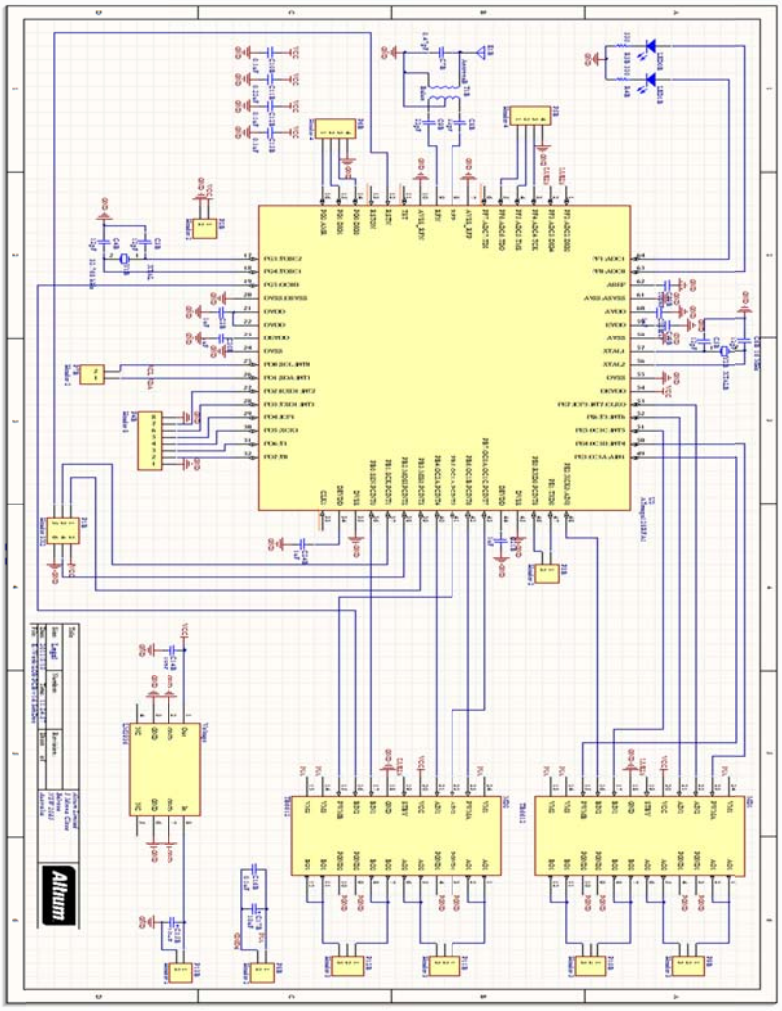


Figure 9 Schematic of the Remote Tank

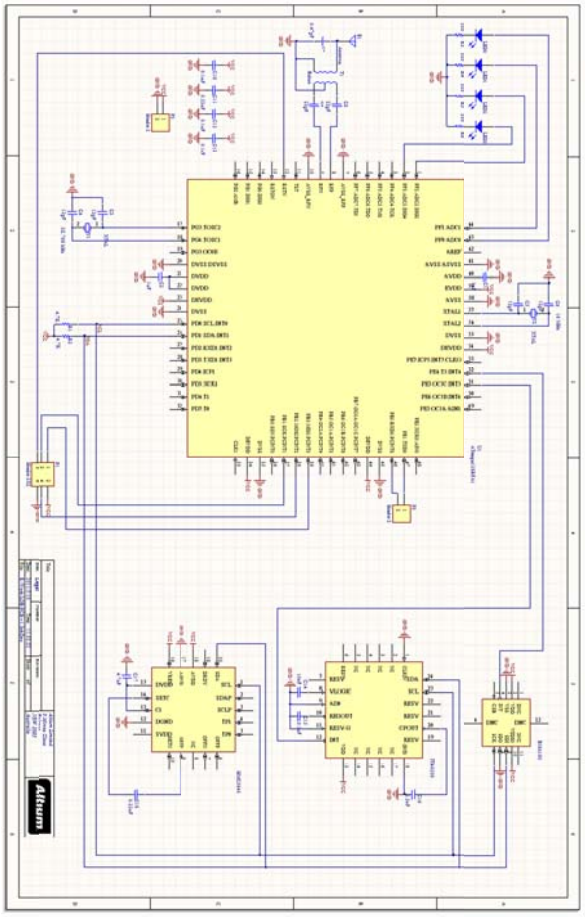


Figure 10 Schematic of partial of the Gesture Controller

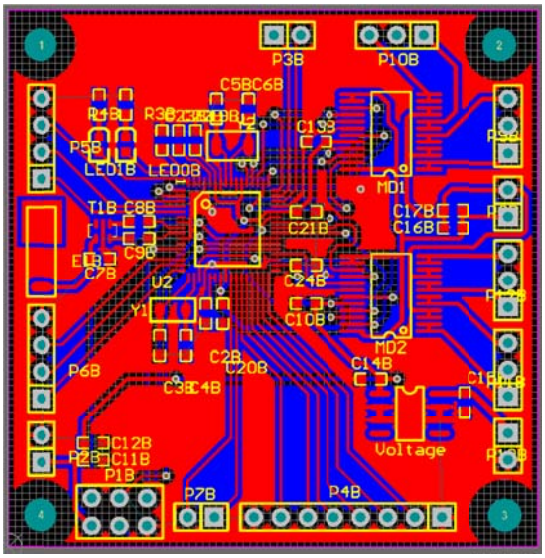


Figure 11 PCB Drawing of the Remote Tank

However, the PCB manufactory made a mistake when fabricating my PCBs and the re-made boards could not be delivered on time. For this reason, I had to switch to my back up plan and used solder board.

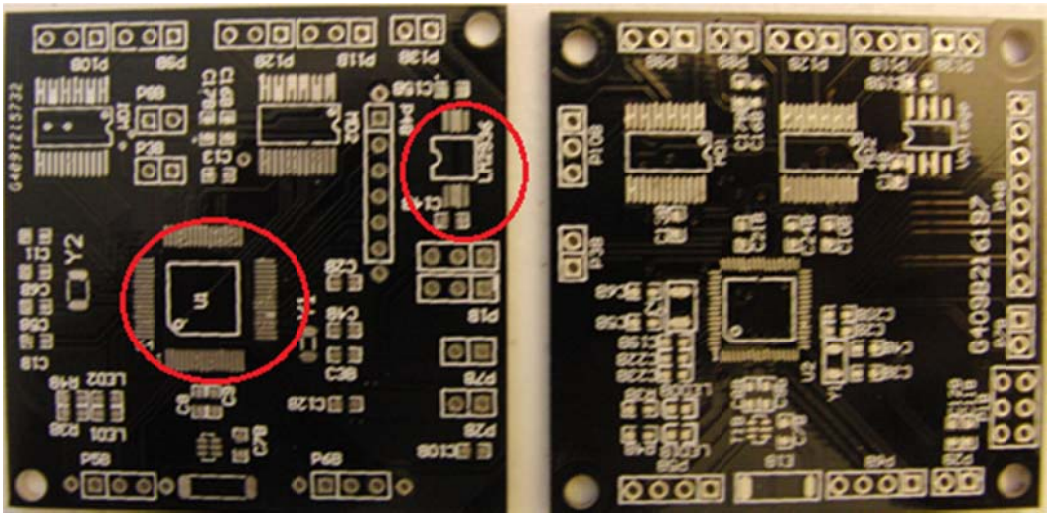


Figure 12 Defected PCB Boards on the left and Re-made PCB on the right

5.2 Backup Plan – Solder Board

As mentioned previously, I had to switch to my backup plan and soldered the below boards, as shown in Figure 13 to Figure 16.

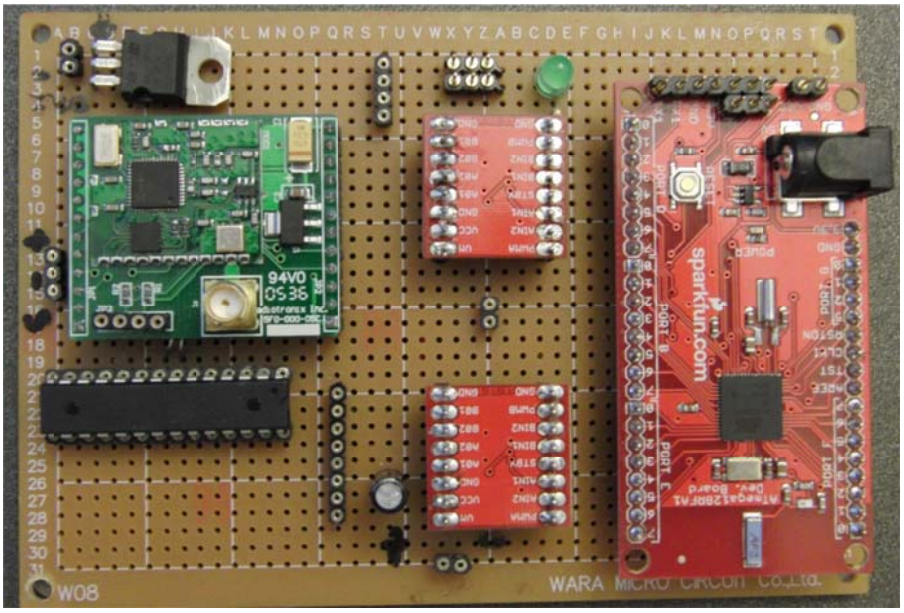


Figure 13 Top Side of the Solder Board on the Remote Tank

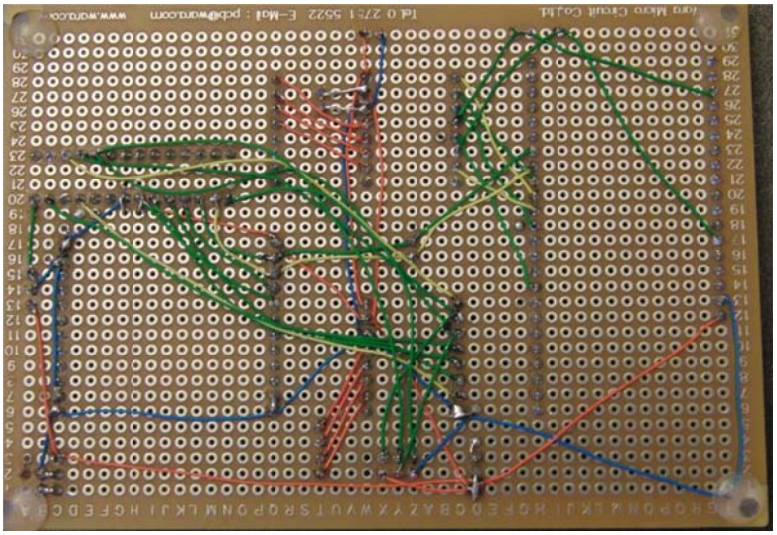


Figure 14 Bottom Side of the Solder Board on the Remote Tank

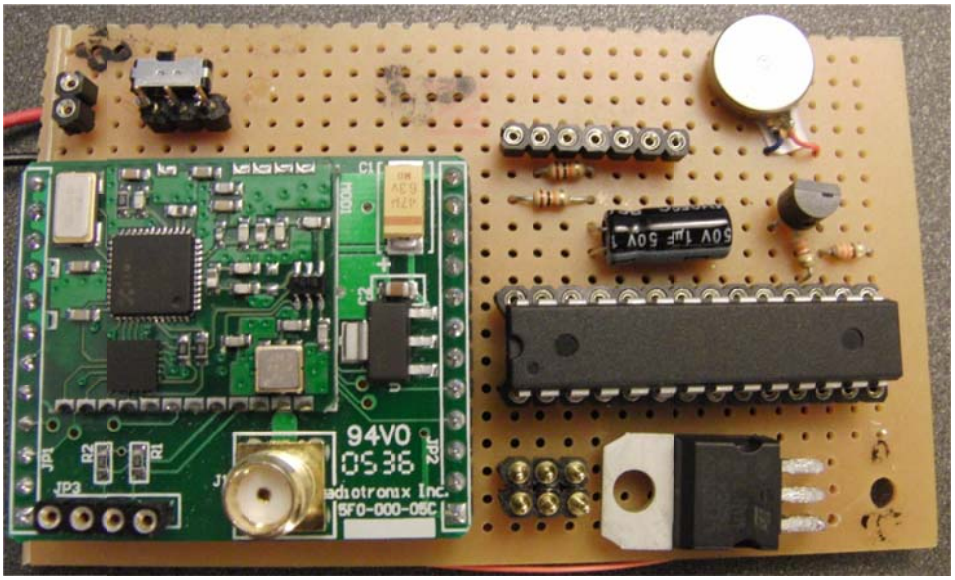


Figure 15 Top Side of the Solder Board on the Remote Tank

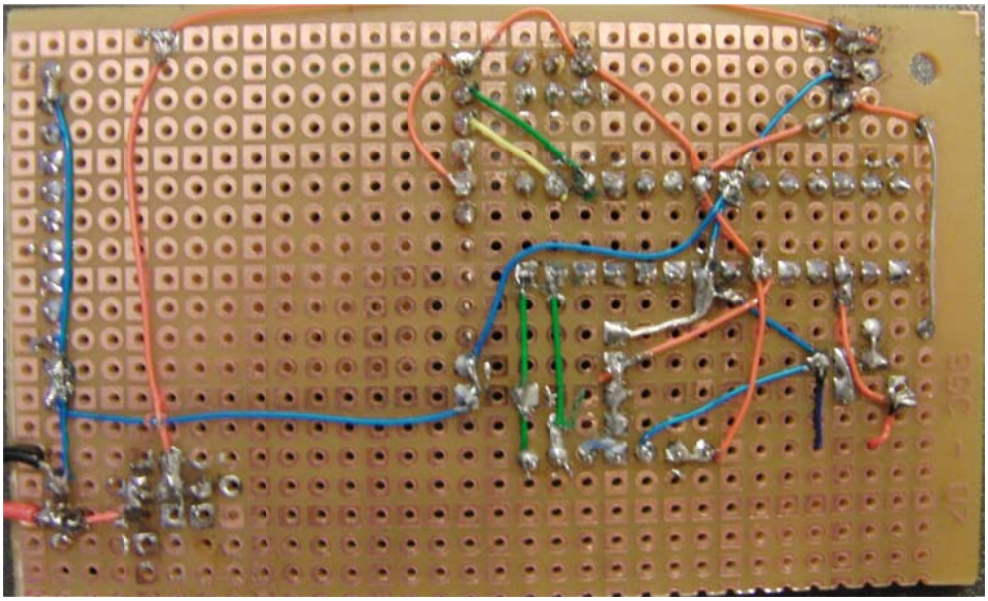


Figure 16 Bottom Side of the Solder Board on the Remote Tank

6 CONCLUSION

6.1 Summary

This development of this project is challenging yet quite enjoyable. The designed gesture controller and the remote tank work as expected with the control and feedback functionalities.

6.2 Lessons I learned

I borrowed some of the code such as the I^2C driver I coded back in 2008 and ported to this project. The lessons I learned here is that a well commented code can be easily re-learned and recycled. Therefore, commenting the code is completely worth doing.

The second lesson I learned is that always have a backup plan. For instance, I never expected a professional PCB manufactory would make that mistake and delayed my project dramatically. I should have started my backup plan earlier and it would almost ensure a better quality project. However, in some senses, “better late than nothing”. Surviving with a “Plan B” once again ensures that backup plans will be generated through my future project developments.

6.3 Intellectual Property Considerations

Some of the ideas used in this project relate to one of my MEng projects at Cornell University with Prof. Garica. And some of the code I recycled from my robotic project back in 2008. In terms of the hardware, I have used existing module intensively to shorter the development phases and the design of these modules belong to their companies. In terms of the software, most of the code in this project are coded either by the current me or myself from 3 years ago, expect the built-in drivers from the AVR Stuido 4, for example, delay functions. Another exception is that I have borrowed the ultra-sonic sensor driver from my 2008 robotic project partner, Jessica Sun Ye (Thanks).

6.4 Ethical Considerations

Since objective this project is to build a gesture remote controlled tank toy, there is no serious ethical considerations shall be involved expect some users may concerns the unencrypted wireless packages although it only contains gesture and feedback information.

The main concern through the development of this project is safety. As discussed

previously, several of safety enforcement algorithms have been employed to ensure the safety of the unit and corner cases are covered to the ability of the designers.

During the development of this project, IEEE code of ethics is carefully followed. I avoided offence of other's patent and claimed by code by commenting them to the extent of my ability.

6.5 Legal Considerations

The wireless-serial module I obtained from Prof. Bruce Land uses 917 MHz RF signal which is certified by the Federal Communications Commission. In addition, the antenna used in this module is also certified by FCC.

7 APPENDIX

7.1 Budget

The overall cost of this project is controlled within the $75 budget as shown in

Item	Quantity	Unit Cost	Total Cost	Note
Wi.232DTS	2	$0.00	$0.00	Donated by Prof. Bruce Land
ITG3200 Gyro Scope	1	$0.00	$0.00	Donated by Prof Garcia Ephrahim
TB6612FNG Motor Driver	1	$8.89	$8.89	Sparkfun
ATMega168	1	$8.89	$8.89	Donated by Prof Patrick Leung
Ultra-Sonic Sensor	1	$12.99	$12.99	Futlect
Vibration Motor	1	$1.00	$1.00	Sparkfun
Solder Board	2	$1.79	$3.58	Futlect
Battery	6	$1.50	$9.00	Amazon
Tank Base	1	$20.00	$20.00	Amazon
Battery Holder	2	$1.00	$2.00	Donated by Prof. Bruce Land
Header	80	$0.05	$4.00	Donated by Prof. Bruce Land
Switch	1	$0.20	$0.20	Donated by Prof. Bruce Land
Resistors, Capacitor, Led, Tansistor	0	$0.00	$0.00	Donated by Prof. Bruce Land
Total			$70.55	

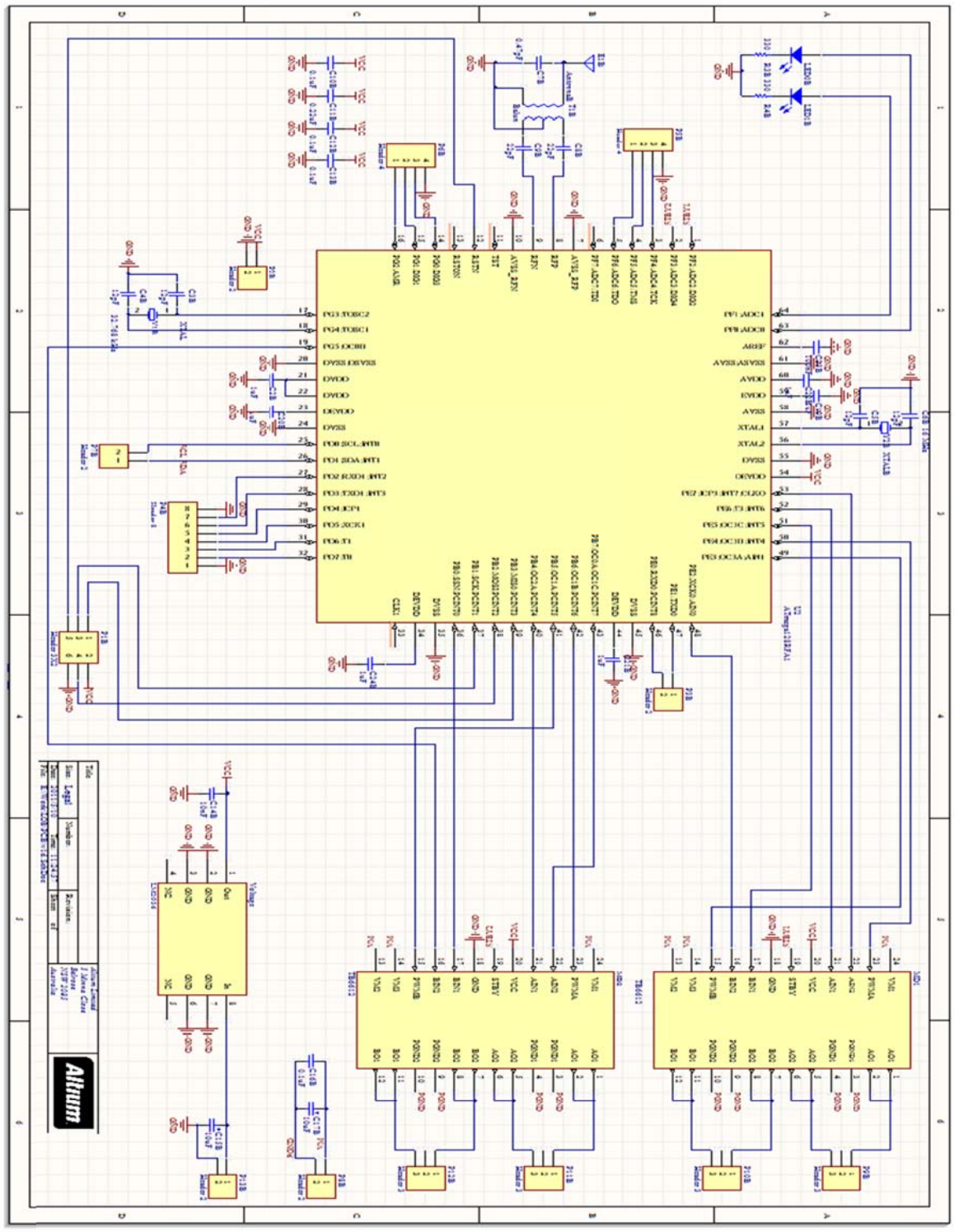
Table 2 Cost of the Project

7.2 Demonstration Video

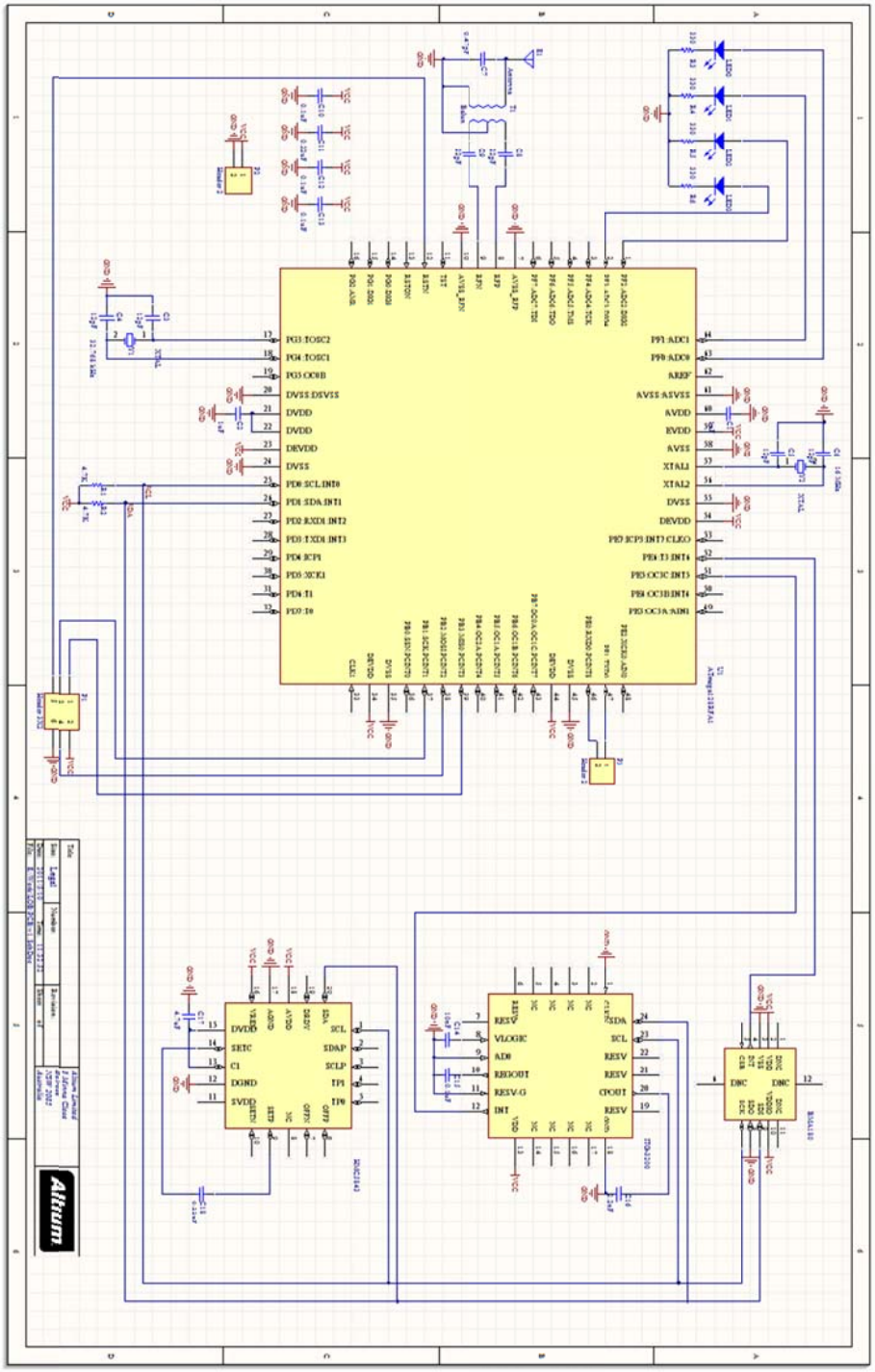
http://pralpha.net/Project/Gesture_Controller/Video
http://www.youtube.com/watch?v=P7jezbWjMsE

7.3 Schematics

Schematic of the Remote Tank



Schematic of partial of the Gesture Controller



7.4 Acknowledgement

- Professor Bruce Land: timely help and kindly support through the course and the final project. A big donator =)
- Professor Garica Ephrahim: provides support upon request and donated parts

7.5 Code Files

http://pralpha.net/Project/Gesture_Controller/Code

8 REFERENCE

8.1 Datasheets

- ATMega168
  http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf
- ITG-3200
  http://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf
- TB6612FNG
  http://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf

8.2 Vendors

- Digkey
- Sparkfun
- Futlect
- Amazon

8.3 Code Borrowed from Others

- Ultra-Sonic Driver borrowed from Jessica Sun Ye

```
/*-------------------------------------------------------------------------
 * Name:                    config.h
 * Version:                 V 1.3 by Rick on 2011-05-08
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
-------------------------------------------------------------------------*/
//min distance before warning msg
#define SAFE_DISTANCE       30

#define ITG3200_AVG_SIZE    3

//define usart driver speed
#define USART BAUD          9600
#define U_RECV_BUFFER_SIZE  64
#define U_SEND_BUFFER_SIZE  64
#define U_HEADER            0

//TWI input / output array buffer size
#define Module TWI Address  GYR_TWI_Address
#define TWI BUFFER SIZE     64
#define TWI_MIN_WAIT        20
#define TWI MAX TRY         100
#define TWI WAIT TIME       23
#define TWI_SPEED           400000

/*-------------------------------------------------------------------------
Section 2:

        Please DO NOT change the below defines
-------------------------------------------------------------------------*/
//U RArray , TWI RArray and TWI_WArray defines
#define U RECV LENGTH   2
#define SUB PASSWORD    0
#define SUB ADDRESS     0
#define SUB COMMAND     1
#define SUB_DATA        2

//define gobal tasks
#define DTAC_TASK           0xFF
#define DBM ANN REQ         0xFE
#define DBM ANN RPY         0xFD
#define DBM_MAX_FAIL        3

//define TWI ADDRESS
#define GEN TWI Address     0x00
#define CMM_TWI_Address     0x01
#define SCM TWI Address     0x02
#define ACM TWI Address     0x03
#define UDM TWI Address     0x04
#define DBM TWI Address     0x05
#define LOB TWI Address     0x06
#define GYR TWI Address     0x07
#define BAS_TWI_Address     0x08
#define BMP_TWI_Address     0x77

//default values
#define TRUE                1
#define FALSE               0
#define ENABLE              1
#define DISABLE             0
#define SET                 1
#define CLEAR               0x00
#define ALLSET              0xFF
```

```
/*-------------------------------------------------------------------------
 * Name:                    connection.h
 * Usage:                   Hardware connections such as pins and ports
 * Target MCU:              ATMega168
 * Version:                 V 0.1 by Rick on 2011-05-08
-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
-------------------------------------------------------------------------*/
//vibration motoe
#define VIB_MOT_DDR     DDRB
#define VIB_MOT_PORT    PORTB
#define VIB_MOT_PIN     3

//USART PORT
#define USART_DDR       DDRD
#define USART_PORT      PORTD
#define USART_RX_PIN    0
#define USART_TX_PIN    1


//TWI PORT
#ifndef TWI_DDR
#define TWI_DDR         DDRC
#endif
#ifndef TWI_PORT
#define TWI_PORT        PORTC
#endif
#ifndef TWI_CLK
#define TWI_CLK         5
#endif
#ifndef TWI_DATA
#define TWI_DATA        4
#endif
```

```
/*---------------------------------------------------------------------------
 * Name:                    control.c
 * Usage:                   main file for control side
 * Target MCU:              ATMega168
 * Version:                 V 0.2 by Rick on 2011-05-08
---------------------------------------------------------------------------*/
#include "control.h"

//#define DEBUG

/*---------------------------------------------------------------------
 * Function details:        main function
 * Name:                    main()
 * Usage:                   none
 * Input:                   none
 * Return:                  none
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
int main(void)
{
    Chip init();
    while(1)
    {

        if (USART_REVD == 1)
        {
            if (CONNECTION_PASSWORD==U_RArray[SUB_PASSWORD])
            {
                if ((U_RArray[SUB_COMMAND]<SAFE_DISTANCE) && (U_RArray[SUB_COMMAND]
))
                {
                    VIB_MOT_PORT |= (SET<<VIB_MOT_PIN);
                }
                else
                {
                    VIB_MOT_PORT &= (~(SET<<VIB_MOT_PIN));
                }
            }
            Clean_Up();
        }
        getITG3200(2,0);
        _delay_ms(50);
        if (X_flag)
        {
            X_flag=CLEAR;
            if (1==angle_x)
            {
                Current_Command=0b11110111;
                #ifdef DEBUG
                printf("f ");
                #endif
            }
            else if (-1==angle_x)
            {
                Current_Command=0b10110011;
                #ifdef DEBUG
                printf("b ");
                #endif
            }
        }
        if (Z_flag)
        {
            Z_flag=CLEAR;
            if (1==angle_z)
            {
                Current_Command=0b10100110;
                #ifdef DEBUG
                printf("l ");
```

```
                    #endif
                }
                else if (-1==angle_z)
                {
                    Current Command=0b11100010;
                    #ifdef DEBUG
                    printf("r ");
                    #endif
                }
            }
            if (Y_flag)
            {
                Y flag=CLEAR;
                if (1==angle_y)
                {
                    Current Command=0b11010111;
                    #ifdef DEBUG
                    printf("lc ");
                    #endif
                }
                else if (-1==angle_y)
                {
                    Current Command=0b11110101;
                    #ifdef DEBUG
                    printf("rc ");
                    #endif
                }
            }
            USART Transmit('a');
            USART_Transmit(Current_Command);
        }
    }


    /*---------------------------------------------------------------------
     * Function details:        chip initialization
     * Name:                    Chip init()
     * Usage:                   to init the chip
     * Input:                   none
     * Return:                  NO ERROR if success
                                ERR CHIP_INIT if fail
     * Attention:               none
     * Notes:                   none
    ---------------------------------------------------------------------*/
    unsigned char Chip_init(void)
    {
        unsigned char fail;             //fail flag
        //disable gobal interrupts
        cli();
        //init gobal variables
        ERR = CLEAR;
        TWI REVD = CLEAR;
        TWI RNum = CLEAR;
        Current Command = CLEAR;
        angle x=CLEAR;
        angle y=CLEAR;
        angle_z=CLEAR;

        //init watch dog
        DOG SLEEP;

        MCUCR = CLEAR;              //init status and control registers
        PRR   = CLEAR;              //power controller

        //init timers
        TIMSK0 = CLEAR;             //timer 0 interrupt sources
        TIMSK1 = CLEAR;             //timer 1 interrupt sources
        TIMSK2 = CLEAR;             //timer 2 interrupt sources
```

```
        //init extern interrupts
        EICRA  = CLEAR;             //extended ext ints
        EIMSK  = CLEAR;             //extended ext int marks
        PCMSK0 = CLEAR;             //pin change mask 0
        PCMSK1 = CLEAR;             //pin change mask 1
        PCMSK2 = CLEAR;             //pin change mask 2
        PCICR  = CLEAR;             //pin change enable

        //init ports
        DDRB   = CLEAR;             //default
        DDRC   = CLEAR;             //default
        DDRD   = CLEAR;             //default

        VIB_MOT_DDR |= (SET<<VIB_MOT_PIN);

        //init TWI
        fail = TWI_INIT(Module_TWI_Address,FALSE);

        //init UARTA
        fail |= USART_Init();

        //init ITG3200 Gyro Scope
        fail |= ITG3200_INIT();

        #ifdef DEBUG
        printf("\r\nInit Finished\r\n");
        #endif

        //enable interrupts
        sei();

        if (fail)
        {
         return ERR_CHIP_INIT;       //error occurs when init the chip
        }
        else
        {
         return NO_ERROR;
        }
    }

    /*---------------------------------------------------------------------
     * Function details:        handler ERR if exists
     * Name:                    ERR Handler()
     * Usage:                   call this function to handler ERR message
     * Input:                   none
     * Return:                  none
     * Attention:               TO BE UPDATED
     * Notes:                   none
    ---------------------------------------------------------------------*/
    void ERR_Handler(void)
    {
    ;
    }

    /*---------------------------------------------------------------------
     * Function details:        reboot the chip
     * Name:                    reboot()
     * Usage:                   call this function to reset the chip
     * Input:                   none
     * Return:                  none
     * Attention:               TO BE UPDATED
     * Notes:                   none
    ---------------------------------------------------------------------*/
    void reboot(void)
    {
      wdt_enable(WDTO_15MS);
      while (1) {;}
    }
```

```
/*-------------------------------------------------------------------
 * Function details:        USART RX ISR handler
 * Name:                    ISR(USART RX vect)
 * Usage:                   called when data received via USART
 * Input:                   none
 * Return:                  none
 * Attention:               x will reset the module
 * Notes:                   none
-------------------------------------------------------------------*/
ISR(USART_RX_vect)
{
    U RArray[U Recv Count] = UDRn;
    if (RESET CMD==U_RArray[U_Recv_Count]) reboot();
    U Recv Count++;
    if (U_RECV_LENGTH == U_Recv_Count)
    {
        U Recv Count = 0;
        USART_REVD = 1;
    }
}

/*-------------------------------------------------------------------
 * Function details:        Clean up USART status
 * Name:                    Clean Up
 * Usage:                   called to clean up USART data
 * Input:                   none
 * Return:                  none
 * Attention:               none
 * Notes:                   none
-------------------------------------------------------------------*/
void Clean_Up(void)
{
 //handle error if any
 if (!ERR)
 {
    ERR_Handler();
 }
 //clean up
 memset(U RArray, CLEAR, U RECV LENGTH);    //clear Usart RArray array
 U Recv Count = CLEAR;                      //clear Usart counter
 USART_REVD = CLEAR;                        //clear USART_REVD flag
}
```

```
/*-------------------------------------------------------------------------
 * Name:                    control.h
 * Usage:                   main header file for control side
 * Target MCU:              ATMega168 (default @ 8MHz)
 * Version:                 V 0.2 by Rick on 2011-05-08
-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
-------------------------------------------------------------------------*/
#define RESET CMD           'x'
#define CONNECTION PASSWORD 'b'
signed int angle x;
signed int angle y;
signed int angle_z;

//#define sbi(var, mask)   ((var) |= (unsigned int8_t)(1 << mask))
//#define cbi(var, mask)   ((var) &= (unsigned int8_t)~(1 << mask))

/*-------------------------------------------------------------------------
Section 2:

        Please DO NOT change the below defines
-------------------------------------------------------------------------*/
//include files
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>
#include <avr/wdt.h>
#include "config.h"
#include "connection.h"
#include "error.h"
#include "TWI.c"
#include "usart.c"
#include "itg3200.c"

//watch dog define
#define DOG_SLEEP      {MCUSR &= (~(SET<<WDRF));WDTCSR|=((SET<<WDCE)|(SET<<WDE));
WDTCSR=CLEAR;}

unsigned char ERR;                  //error flag
unsigned char Current_Command;      //Motor moving done flag
unsigned char FLAG;
/*-------------------------------------------------------------------------
Section 3:

        Declaration of functions
-------------------------------------------------------------------------*/
unsigned char Chip init(void);
void ERR Handler(void);
void reboot(void);
void Clean_Up(void);
```

```c
/*---------------------------------------------------------------------------
 * Name:                    itg3200.c
 * Usage:                   ITG3200 Gyro Scope driver
 * Target MCU:              ATMega168
 * Version:                 V 0.1 by Rick on 2011-05-08
---------------------------------------------------------------------------*/
#include "itg3200.h"

//Global variables
signed char count=CLEAR,last data pointer=(1-ITG3200 sample size);
struct ITG3200 data GYRO data[ITG3200_sample_size],GYRO_sum;
unsigned char X_flag,Y_flag,Z_flag;

/*---------------------------------------------------------------------
 * Function details:        ITG3200 Initialize
 * Name:                    ITG3200 INIT()
 * Usage:                   to init ITG3200 Gyro Scope
 * Input:                   none
 * Return:                  Success: return 0, Fail: return 1
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
unsigned char ITG3200_INIT(void)
{
    ITG3200_Clean();
    ITG3200Write(PWR M, (SET<<H RESET));    // Reset to defaults
    ITG3200Write(SMPL, CLEAR);  // SMLPRT DIV = 0
    ITG3200Write(DLPF, ((SET<<FSL_SEL1)|(SET<<FSL_SEL1)));  // DLPF_CFG = 0, FS_SEL
= 3
    ITG3200Write(INT C, ((SET<<ITG RDY EN)|(SET<<RAW_RDY_EN))); // Generate
interrupt when device is ready or raw data ready
    ITG3200Write(PWR_M, CLEAR);
    return NO_ERROR;
}

/*---------------------------------------------------------------------
 * Function details:        read all the registers from ITG3200
 * Name:                    ITG3200ViewRegisters()
 * Usage:                   call to check the reg values from ITG3200
 * Input:                   none
 * Return:                  none
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
void ITG3200ViewRegisters(void)
{
    printf("\nWHO AM_I (0x00): 0x%x\n", ITG3200Read(WHO));
     delay ms(1000);
    printf("SMPLRT DIV (0x15): 0x%x\n", ITG3200Read(SMPL));
    printf("DLPF_FS (0x16): 0x%x\n", ITG3200Read(DLPF));
    printf("INT CFG (0x17): 0x%x\n", ITG3200Read(INT C));
    printf("INT STATUS (0x1A): 0x%x\n", ITG3200Read(INT S));
    printf("TEMP OUT H (0x1B): 0x%x\n", ITG3200Read(TMP H));
    printf("TEMP OUT L (0x1C): 0x%x\n", ITG3200Read(TMP L));
    printf("GYRO XOUT H (0x1D): 0x%x\n", ITG3200Read(GX H));
    printf("GYRO XOUT L (0x1E): 0x%x\n", ITG3200Read(GX L));
    printf("GYRO_YOUT_H (0x1F): 0x%x\n", ITG3200Read(GY_H));
    printf("GYRO YOUT L (0x20): 0x%x\n", ITG3200Read(GY L));
    printf("GYRO ZOUT H (0x21): 0x%x\n", ITG3200Read(GZ H));
    printf("GYRO ZOUT L (0x22): 0x%x\n", ITG3200Read(GZ L));
    printf("PWR_MGM (0x3E): 0x%x\n", ITG3200Read(PWR_M));
}

/*---------------------------------------------------------------------
 * Function details:        read xyz gyro and temperature data
 * Name:                    getITG3200()
 * Usage:                   call to read data from ITG3200
 * Input:                   avergae:    times that data to be averaged
 *                          print:      whether to print the data
```

```
 * Return:                  none
 * Attention:               none
 * Notes:                   none
-----------------------------------------------------------------------*/
void getITG3200(int average,char print)
{
    signed int gx, gy, gz,t;
    signed int gyrox = 0;
    signed int gyroy = 0;
    signed int gyroz = 0;
    signed long temperature = 0;
    unsigned int i;

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

        ITG3200 CHECK;
        gx = (ITG3200Read(GX_H))<<8;
        gx |= ITG3200Read(GX_L);

        gy = (ITG3200Read(GY H))<<8;
        gy |= ITG3200Read(GY_L);

        gz = (ITG3200Read(GZ H))<<8;
        gz |= ITG3200Read(GZ_L);

        t = (ITG3200Read(TMP H))<<8;
        t |= ITG3200Read(TMP_L);

        gyrox += gx;
        gyroy += gy;
        gyroz += gz;
        temperature += t;
    }

    GYRO data[count].gx = (int)((gyrox/average+ITG3200 x offset)/14);
    GYRO data[count].gy = (int)((gyroy/average+ITG3200 y offset)/14);
    GYRO data[count].gz = (int)((gyroz/average+ITG3200 z offset)/14);
    GYRO data[count].temper = 35+(((int)(temperature/average))+
ITG3200_temper_offset)/280;

    if (last_data_pointer>=0)
    {
        GYRO sum.gx+=GYRO data[count].gx-GYRO data[last data pointer].gx;
        GYRO sum.gy+=GYRO data[count].gy-GYRO data[last data pointer].gy;
        GYRO sum.gz+=GYRO data[count].gz-GYRO data[last data pointer].gz;
        GYRO_sum.temper+=GYRO_data[count].temper-GYRO_data[last_data_pointer].
temper;
    }
    else
        GYRO_sum.temper=GYRO_data[count].temper;

    if (GYRO_data[count].gx>300)
    {
        if (1==angle_x)
        angle x=1;
        else if (0==angle_x)
        {angle x=1;X flag=SET;}
        else if (-1==angle_x)
        angle_x=0;
    }
    else if (GYRO_data[count].gx<-300)
    {
        if (1==angle_x)
        angle x=0;
        else if (0==angle x)
        {angle x=-1;X flag=SET;}
        else if (-1==angle_x)
        angle_x=-1;
```

```
        }

        if (GYRO_data[count].gy>300)
        {
            if (1==angle_y)
            angle y=1;
            else if (0==angle_y)
            {angle y=1;Y flag=SET;}
            else if (-1==angle_y)
            angle_y=0;
        }
        else if (GYRO_data[count].gy<-300)
        {
            if (1==angle_y)
            angle y=0;
            else if (0==angle y)
            {angle y=-1;Y flag=SET;}
            else if (-1==angle_y)
            angle_y=-1;
        }

        if (GYRO_data[count].gz>300)
        {
            if (1==angle_z)
            angle_z=1;
            else if (0==angle z)
            {angle z=1;Z flag=SET;}
            else if (-1==angle_z)
            angle_z=0;
        }
        else if (GYRO_data[count].gz<-300)
        {
            if (1==angle_z)
            angle z=0;
            else if (0==angle z)
            {angle z=-1;Z flag=SET;}
            else if (-1==angle_z)
            angle_z=-1;
        }

        if (print)
        {
            //printf("%d    %d  %d  %d\n", GYRO_sum.gx,GYRO_sum.gy,GYRO_sum.gz,GYRO_sum
    .temper);
            //printf("%d    %d  %d  %d\n", GYRO_data[count].gx,GYRO_data[count].gy,
    GYRO data[count].gz,GYRO sum.temper);
            //printf("%d    %d  %d\n", angle x,angle y,angle z);
            printf("\r\n%d  %d  %d\r\n",angle_x,angle_y,angle_z);
        }

        count++;
        last data pointer++;
        if (count>ITG3200 sample size) count=CLEAR;
        if (last_data_pointer>ITG3200_sample_size) last_data_pointer=CLEAR;
    }


    /*---------------------------------------------------------------------
     * Function details:        read register from ITG3200
     * Name:                    ITG3200Read()
     * Usage:                   call to read reg in ITG3200 and return char
     * Input:                   Address: the address of the reg to be read
     * Return:                  the value of the reg in char format
     * Attention:               none
     * Notes:                   none
    ---------------------------------------------------------------------*/
    char ITG3200Read(unsigned char address)
    {
        TWI_MR_Sensor(ITG3200_TWI_Address, address, 1);
```

```
    return TWI_RArray[0];
}

/*---------------------------------------------------------------------
 * Function details:        write register to ITG3200
 * Name:                    ITG3200Write()
 * Usage:                   call to write a data to reg in ITG3200
 * Input:                   Address: the address of the reg to be read
 *                          data: the data to be written
 * Return:                  none
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
void ITG3200Write(unsigned char address, unsigned char data)
{
    TWI WArray[0]=address;
    TWI WArray[1]=data;
    TWI_MT(ITG3200_TWI_Address,2);
}

/*---------------------------------------------------------------------
 * Function details:        ITG3200 clean up
 * Name:                    TWI INIT()
 * Usage:                   to clean up ITG3200 data
 * Input:                   none
 * Return:                  none
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
void ITG3200_Clean(void)
{
    count = CLEAR;
    last data pointer=(1-ITG3200_sample_size);
    GYRO sum.gx=CLEAR;
    GYRO sum.gy=CLEAR;
    GYRO sum.gz=CLEAR;
    GYRO_sum.temper=CLEAR;
}
```

```
/*-------------------------------------------------------------------------
 * Name:                    itg3200.h
 * Target MCU:              ATMega168
 * Version:                 V 0.1 by Rick on 2011-05-08
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
-----------------
Section 1:
    The blow defines can be changed if necessary
-------------------------------------------------------------------------
---------------*/
#define ITG3200 x offset        150
#define ITG3200 y offset        5
#define ITG3200 z offset        -65
#define ITG3200 temper offset   13200
#define ITG3200_sample_size     11


/*-------------------------------------------------------------------------
-----------------
Section 2:
    Please DO NOT change the blow defines unless you are sure
-------------------------------------------------------------------------
---------------*/
struct ITG3200_data
{
    signed int gx;
    signed int gy;
    signed int gz;
    signed int temper;

};
#define ITG3200 R           0xD3    // ADD pin is pulled high
#define ITG3200_W           0xD2    // So address is 0x69

#define ITG3200 TWI Address 0x69
#define WHO                 0x00
#define SMPL                0x15
#define DLPF                0x16
#define INT C               0x17
#define INT S               0x1A
#define TMP_H               0x1B
#define TMP L               0x1C
#define GX H                0x1D
#define GX L                0x1E
#define GY H                0x1F
#define GY L                0x20
#define GZ H                0x21
#define GZ_L                0x22
#define PWR_M               0x3E

#define H RESET             7
#define CLK SEL0            0
#define CLK SEL1            1
#define CLK SEL2            2
#define INTANYCLEAR         4
#define ITG RDY EN          2
#define RAW RDY EN          0
#define FSL SEL1            4
#define FSL_SEL0            3

#define ITG3200 CHECK       {char wait_index=0;while ((!(ITG3200Read(INT_S) & (SET<
<RAW_RDY_EN))) & (wait_index<=250));}

/*-------------------------------------------------------------------------
-----------------
Section 3:
    Declaration of functions
```

```
        ------------------------------------------------------------------------------
        ---------------*/
        unsigned char ITG3200 INIT (void);
        void ITG3200ViewRegisters(void);
        void getITG3200(int average,char print);
        char ITG3200Read(unsigned char address);
        void ITG3200Write(unsigned char address, unsigned char data);
70      void ITG3200_Clean(void);
```

```c
/*-------------------------------------------------------------------------
 * Name:                    TWI.c
 * Usage:                   TWI driver
 * Target MCU:              ATMega168
 * Version:                 V 0.8 by Rick on 2011-05-08
-------------------------------------------------------------------------*/
#include "TWI.h"

//Global variables
unsigned char TWI WArray[TWI BUFFER SIZE];  //input output array
unsigned char TWI RArray[TWI BUFFER SIZE];  //output array point
unsigned char *TWI R P = NULL;              //temp input array point used in ISR
unsigned char TWI N S;
unsigned char TWI REVD;                     //TWI reveived flag
unsigned char TWI_RNum;                     //length of TWI package

/*-------------------------------------------------------------------------
 * Function details:        TWI Initialize
 * Name:                    TWI INIT()
 * Usage:                   to init TWI of this MCU
 * Input:                   Address: 7 bit TWI address of this MCU
                            GCall:   Accept General Call 1 / Otherwire 0
 * Return:                  Success: return 0, Fail: return 1
 * Attention:               none
 * Notes:                   none
-------------------------------------------------------------------------*/
unsigned char TWI_INIT(unsigned char Address, unsigned char GCall)
{
  TWI Clean();
  TWI DDR &= (~(1<<TWI CLK)) & (~(1<<TWI DATA));
  TWI PORT |= (1<<TWI CLK) | (1<<TWI DATA);
  TWBR=TWI_BR;                          //TWI BUS Speed
  TWAR=(Address<<1)|GCall;              //TWI address of this MCU
  TWDR=0x00;                            //clear data register
  TWSR=0x00;                            //clear status register
  TWI Slave();                          //default slave mode
  return 0;
}

/*-------------------------------------------------------------------------
 * Function details:        TWI Master Transmit with Resend ability
 * Name:                    TWI MT RESEND()
 * Usage:                   Keep resending msg via TWI for MAX_TRY
 * Input:                   Address
 * Return:                  Success: return 0, Fail: return 1
 * Attention:               default values of TWI MAX TRY and
                            TWI WAIT TIME are defined in TWI.h
                            and the values in the header file of MCU
                            will overwrite them
 * Notes:                   none
-------------------------------------------------------------------------*/
unsigned char TWI_MT_RESEND(unsigned char Address, unsigned int Length)
{
  unsigned char fail = CLEAR;               //fail flag
  unsigned char try = CLEAR;                //try # counter

  //try to resend for MAX_TRY times before give up
  for (try = CLEAR;try < TWI_MAX_TRY; try ++)
  {
    fail = TWI MT(Address, Length); //try to send data via TWI
    if (!fail)                      //flag is 0 if send success
       return NO ERROR;             //task done with no error
    _delay_ms(TWI_MIN_WAIT+TWI_WAIT_TIME);  //elsewise, delay and retry
  }
  return fail;
}

/*-------------------------------------------------------------------------
 * Function details:        TWI Master Transmit
```

```
70       * Name:                    TWI MT()
         * Usage:                   to send multi data to Address via TWI
         * Input:                   Address
         * Return:                  Success: return 0, Fail: return 1
         * Attention:               none
         * Notes:                   none
        ----------------------------------------------------------------------*/
        unsigned char TWI_MT(unsigned char Address, unsigned int Length)
        {
              unsigned int counter = CLEAR;             //count # of byte sent
80            unsigned char *TWI W P=NULL;
              TWI_W_P=TWI_WArray;               //Save the current pointer

              if (TWI N S)                      //if MCU is slaving existing master
                return ERR TWI SEND;            //return and wait to resend
                                                //this prevent hang due BUS START error

              TWI_Start();                      //sent Start
              TWI_Wait();                       //wait for reply

90            if(TWI TestAck()!=START)          //check if Start is sent through
                return ERR_TWI_SEND;

              Address=(Address<<1)&WD;          //combine the 7 bit Address and W
              TWI_Write8Bit(Address);           //send Address and W
              TWI_Wait();                       //wait for reply

              switch (TWI_TestAck())
              {
               case MT SLA ACK:                 //SLA W sent successfully
100                 TWI Write8Bit(*TWI_W_P++);  //send Wdata
                    counter ++;
                    break;
               case MT SLA NOACK:               //Not Ack received
                    TWI Stop();                 //Stop TWI and quit
                     delay ms(5);               //Delay needed between Stop and Slave
                    TWI Slave();                //Slave mode
                    return ERR TWI SEND;
               case M ARB:                      //Arb lost in slave address/data
               case S ARB R:                    //Arb lost, slave R mode
110            case S ARB G:                    //Arb lost, slave General call R mode
               case S ARB_T:                    //Arb lost, slave General call T mode
               default:
                    TWI Slave();                //Slave mode
                    return ERR_TWI_SEND;        //Arb Lost, quit
              }

           while(TRUE)
           {
              TWI_Wait();
120
              switch (TWI_TestAck())
              {
               case MT DATA ACK:                //Data sent though
                    if(counter < Length)        //if more data in output array
                    {
                    TWI_Write8Bit(*TWI_W_P++);  //send Wdata
                    counter ++;
                    break;
                    }
130                 TWI Stop();                 //send task done if output array empty
                     delay ms(5);               //Delay needed between Stop and Slave
                    TWI Slave();                //Slave mode
                    return NO_ERROR;            //Successful, return

               case MT DATA NOACK:              //Not Ack received
                    TWI Stop();                 //Stop TWI and quit
                    _delay_ms(5);               //Delay needed between Stop and Slave
                    TWI_Slave();                //Slave mode
```

```
                return ERR_TWI_SEND;          //Error, return
140          case M_ARB:                      //Arb lost in slave address/data
             default:
                TWI_Slave();                  //Slave mode
                return ERR_TWI_SEND;          //Arb Lost, quit
             }
          }
          return ERR_TWI_SEND;
       }

       /*---------------------------------------------------------------------
150     * Function details:        TWI Master Receive
        * Name:                    TWI_MR()
        * Usage:                   to receive 1 byte data to Address via TWI
        * Input:                   Address
        * Return:                  0 if success or 1 if fail
        * Attention:               none
        * Notes:                   none
       ---------------------------------------------------------------------*/
       unsigned char TWI_MR(unsigned char Address)
       {
160          TWI_Start();                     //sent Start
             TWI_Wait();                      //wait for reply
             if(TWI_TestAck()!=START)
               return 1;                      //check if Start is sent through

             Address=(Address<<1)|RD;         //combine Address and R
             TWI_Write8Bit(Address);          //send Address and R
             TWI_Wait();                      //wait for reply
             switch (TWI_TestAck())
             {
170           case MR_SLA_ACK:                //SLA_R sent successfully
                TWI_Receive();                //start to receive data
                break;
              case MR_SLA_NOACK:              //Not Ack received
                TWI_Stop();                   //Stop TWI and quit
                 delay_ms(5);                 //Delay needed between Stop and Slave
                TWI_Slave();                  //Slave mode
                return 1;
              case M_ARB:                     //Arb lost in slave address/data
              case S_ARB_R:                   //Arb lost, slave R mode
180           case S_ARB_G:                   //Arb lost, slave General call R mode
              case S_ARB_T:                   //Arb lost, slave General call T mode
              default:
                TWI_Slave();                  //Slave mode
                return 1;                     //Arb lost, quit
             }

             TWI_Wait();
             switch (TWI_TestAck())
             {
190           case MR_DATA_NOACK:             //Data is received
                *TWI_RArray=TWDR;             //Save Received data
                TWI_Stop();                   //TWI receiving successful
                 delay_ms(5);                 //Delay needed between Stop and Slave
                TWI_Slave();                  //Slave mode
                return 0;
              default:
                TWI_Slave();                  //Slave mode
                return 1;
             }
200    }

       /*---------------------------------------------------------------------
        * Function details:        TWI interrupt service routine
        * Name:                    TWI_S()
        * Usage:                   TWI Slave mode interrupt subroutine
        * Input:                   none
        * Return:                  none
```

```
 * Attention:                none
 * Notes:                    none
-------------------------------------------------------------------*/
ISR (TWI_vect)
{
 //TWI N S is the slave mode flag indicates that  MCU is servering othter master
 //this MCU will hang at TWI_Wait after TWI_Start if entering master mode while
 //it servers existing master on BUS. Therefore MCU can only init "Start" when it
 //is not on the "half way" of slave mode
 TWI N S = 1;
 switch (TWI_TestAck())
 {
  // Slave Receive Mode
  case SR SLA ACK:                          //Start is received
       TWI R P=TWI RArray;                  //set temp input array point
       TWI S Ack();                         //ack the master
       break;
  case SR_DATA_ACK:                         //Data is received
  case SR DATA NOACK:                       //Last Byte of Data is received
  case SR GDATA ACK:                        //General called, data received
  case SR GDATA NOACK:                      //General called, last data received
       *TWI R P=TWDR;                       //Copy received data to input array
       TWI R P++;                           //increase the pointer
       TWI S Ack();                         //Send Ack
       break;
  case SR STOP:                             //STOP received
       TWI N S = 0;                         //done slave mode
                                            //clear slave mode flag, MCU can be master
now

       TWI RNum = (TWI_R_P - TWI_RArray);
       TWI_REVD = 1;

       TWI S Ack();                         //Send Ack
       break;
  case S ARB R:                             //Arb lost, slave R mode
  case SR GSLA ACK:                         //General call address received
  case S ARB G:                             //Arb lost, General call received
       TWI S Ack();                         //Send Ack
       break;

  // Slave Transmit Mode
  case ST_SLA_ACK:                          //Start is received
  case ST DATA ACK:                         //Ready to send data
       TWDR=*TWI_WArray;                    //Send data in TWI_WData
       break;
  case ST DATA NOACK:                       //Data cannot be send
  case ST LAST DATA:                        //Last Data is send
       TWI S NoAck();                       //Send Not Ack
       break;
  case S ARB T:                             //Arb lost, slave T mode
       TWDR=*TWI WArray;                    //Send data in TWI_WData
       TWI S Ack();                         //Send Ack
       break;
  case ST NO INFO:                          //No infomation
       TWI_N_S = 0;                         //clear slave mode flag, MCU can be master
now
       break;
  case ST BUS ERROR:                        //Bus error caused by Start/Stop
       TWI_N_S = 0;                         //clear slave mode flag, MCU can be master
now
       TWI Stop();                          //Stop transmission
        delay ms(1);
       TWI Slave();
       break;

  default:
       TWI_S_Ack();                         //Send
       TWI_N_S = 0;                         //clear slave mode flag, MCU can be master
```

```
now
       break;
 }
}

/*---------------------------------------------------------------------
 * Function details:        TWI Master Receive from Sensor
 * Name:                    TWI MR Sensor()
 * Usage:                   to receive 1 byte data from Address via TWI
 * Input:                   Address
 * Return:                  0 if success or 1 if fail
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
unsigned char TWI_MR_Sensor(unsigned char Module, unsigned char Address, unsigned
char Length)
{
      unsigned char *TWI_R_P=NULL;

      //TWI Disenable();
      //TWI_Enable();


      TWI_R_P=TWI_RArray;               //Save the current pointer

      TWI Start();                      //sent Start
      TWI_Wait();                       //wait for reply

      Module=(Module<<1)&WD;            //combine Address and R
      TWI Write8Bit(Module);            //send Address and R
      TWI_Wait();

      TWI Write8Bit(Address);           //send Address and R
      TWI_Wait()

      TWI_Start();

       delay ms(1);
      Module=Module|RD;
      TWI Write8Bit(Module); // read from this I2C address, R/*W Set
      TWI_Wait();

      while (--Length)
      {
        TWI Receive Ack();          //start to receive data with ACK
        TWI Wait();
        *TWI_R_P++ = TWDR; //Read the LSB data
      }

      TWI_Receive();            //start to receive data with ACK
      TWI Wait();
      *TWI_R_P = TWDR; //Read the MSB data

      TWI_Stop();

      //TWI Disenable();
      //TWI_Enable();

      return 0;
}

/*---------------------------------------------------------------------
 * Function details:        TWI clean up
 * Name:                    TWI Clean()
 * Usage:                   to clean up TWI data
 * Input:                   none
 * Return:                  none
 * Attention:               none
 * Notes:                   none
```

```
-----------------------------------------------------------------------*/
void TWI_Clean(void)
{
  memset(TWI_WArray,CLEAR,TWI_BUFFER_SIZE);
  memset(TWI_RArray,CLEAR,TWI_BUFFER_SIZE);
  TWI_N_S=CLEAR;
  TWI_REVD=CLEAR;                       //TWI reveived flag
  TWI_RNum=CLEAR;                       //length of TWI package

}
```

```
/*-------------------------------------------------------------------------
 * Name:                    TWI.h
 * Usage:                   TWI parameters
 * Target MCU:              ATMega168
 * Version:                 V 0.8 by Rick on 2011-05-08
-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------
Section 2:

        Please DO NOT change the below defines
-------------------------------------------------------------------------*/
//TWI BR determines the TWI speed
//FOSC and TWI SPEED are defined in config.h
#define TWI_BR (F_CPU/TWI_SPEED-16)/2

//TWI Status
//General
#define TWCR_MASK       0x0F
#define START           0x08
#define RE START        0x10
//Master Transmit
#define MT SLA ACK      0x18
#define MT SLA NOACK    0x20
#define MT DATA ACK     0x28
#define MT_DATA_NOACK   0x30
//Master Receive
#define MR SLA ACK      0x40
#define MR SLA NOACK    0x48
#define MR DATA ACK     0x50
#define MR DATA NOACK   0x58
//Slave Transmit
#define ST SLA ACK      0xA8
#define ST DATA ACK     0xB8
#define ST DATA NOACK   0xC0
#define ST LAST_DATA    0xC8
//Slave Receive
#define SR SLA ACK      0x60
#define SR DATA ACK     0x80
#define SR DATA NOACK   0x88
#define SR GSLA ACK     0x70
#define SR GDATA ACK    0x90
#define SR GDATA NOACK  0x98
#define SR_STOP         0xA0
//Arbitration lost
#define M ARB           0X38
#define S ARB R         0x68
#define S ARB G         0x78
#define S ARB T         0xB0
//Other status
#define ST_NO_INFO      0xF8
#define ST_BUS_ERROR    0x00

//TWI operation
#define RD 0x01
#define WD 0xFE
#define TWI Slave()         (TWCR=(1<<TWEN)|(1<<TWEA)|(1<<TWIE))
//Slave mode
#define TWI_Start()         (TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN))
//Start I2C
#define TWI_Stop()          (TWCR=(1<<TWINT)|(1<<TWSTO)|(1<<TWEN))
//Stop I2C
#define TWI_Wait()          {char i=0; while(!(TWCR&(1<<TWINT))&& (i < 255)) i++;}
```

```
//Wait until interrupt
#define TWI TestAck()       (TWSR&0xf8)
//check Status Code
#define TWI Receive()       (TWCR=TWCR&(TWCR_MASK|(1<<TWINT)|(1<<TWEN)))
//Receive from TWI
#define TWI Receive_Ack()   (TWCR=TWCR&(TWCR_MASK|(1<<TWINT)|(1<<TWEA)|(1<<TWEN)))
//Receive from TWI
#define TWI Write8Bit(x)    {TWDR=(x);TWCR=TWCR&(TWCR_MASK|(1<<TWINT)|(1<<TWEN));}
//Write to TWI
#define TWI_S_Ack()         (TWCR=TWCR&(TWCR_MASK|(1<<TWEA)|(1<<TWINT)))
//Send ACK
#define TWI S NoAck()       (TWCR=TWCR&(TWCR_MASK|(1<<TWINT)))
//Send NoACK
#define TWI Connect()       (TWCR=TWCR | (1<<TWEA))
#define TWI Disconnect()    (TWCR=TWCR & ~ (1<<TWEA))
#define TWI Disenable()     TWCR &= (~(SET<<TWEN));
#define TWI_Enable()        TWCR |= (SET<<TWEN);

/*-------------------------------------------------------------------------
Section 3:

        Declaration of functions
-------------------------------------------------------------------------*/
unsigned char TWI INIT(unsigned char Address, unsigned char GCall);
unsigned char TWI_MT_RESEND(unsigned char Address, unsigned int Length);
unsigned char TWI MT(unsigned char Address, unsigned int Length);
unsigned char TWI MR(unsigned char Address);
unsigned char TWI_MR_Sensor(unsigned char Module, unsigned char Address, unsigned
char Length);
void TWI Clean(void);
//extern unsigned char TWI REVD;
//extern unsigned char TWI_RNum;
```

```c
/*---------------------------------------------------------------------
 * Name:                            USART.c
 * Usage:                           USART driver
 * Target MCU:              ATMega168
 * Version:                     V 0.2 by Rick on 2011-05-08
---------------------------------------------------------------------*/
#include "usart.h"

unsigned char U Recv Count;
unsigned char USART REVD;           //TWI reveived flag
unsigned char U RArray[U RECV BUFFER SIZE];
unsigned char U WArray[U SEND BUFFER SIZE];
static FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW
);

/*---------------------------------------------------------------------
 * Function details:            USART Initialize
 * Name:                                USART_Init()
 * Usage:                               to init USART of this MCU
 * Input:                               none
 * Return:                          Success: return 0, Fail: return 1
 * Attention:                       none
 * Notes:                               none
---------------------------------------------------------------------*/
unsigned char USART_Init(void)
{
    USART DDR|=((SET<<USART_TX_PIN)&(~(SET<<USART_RX_PIN)));
    U Recv Count = 0;
    /* Set band rate */
    UBRRnH=(F CPU/16UL/USART BAUD-1)/256;
    UBRRnL=(F CPU/16UL/USART BAUD-1)%256;
    /* Enable receiver and transmitter and receive complete interrupt*/
    UCSRnB= (SET<<RXENn)|(SET<<TXENn);
    /* Set frame format: 8 data, 2 stop bits */
    UCSRnC = ((SET<<UCSZn1)|(SET<<UCSZn0))&(~(SET<<UMSELn0));
    USART R S();                        //enable receive complet interrupt
    stdout = stdin = stderr = &uart_str;
    return 0;
}

/*---------------------------------------------------------------------
 * Function details:            USART Transmition
 * Name:                                USART_Transmit()
 * Usage:                               scall to end out a char
 * Input:                               char to be sent
 * Return:                          none
 * Attention:                       none
 * Notes:                               none
---------------------------------------------------------------------*/
void USART_Transmit (unsigned char data) //output char
{
    /* Wait for empty transmit buffer */
    while (!(UCSRnA & (1<<UDREn)));
    /* Put data into buffer, sends the data */
    UDRn = data;
}

/*---------------------------------------------------------------------
 * Function details:            USART Receiver
 * Name:                                USART Receive()
 * Usage:                               call to get the received char
 * Input:                               none
 * Return:                          char that received
 * Attention:                       none
 * Notes:                               none
---------------------------------------------------------------------*/
unsigned char USART_Receive (void) //receive char
{
    while (!(UCSRnA & (1<<RXCn)));
```

```
    return UDRn;
}

/*---------------------------------------------------------------------
 * Function details:            output a string via USART
 * Name:                            U Out s()
 * Usage:                           call to output a sting
 * Input:                           string to be sent
 * Return:                      none
 * Attention:                   none
 * Notes:                           none
---------------------------------------------------------------------*/
void U_Out_s (signed char *s)           // Output string with change line
{
    while (*s)
    {
        USART Transmit(*s++);
        _delay_ms(5);
    }
}

/*---------------------------------------------------------------------
 * Function details:            output a int via USART
 * Name:                            U Out i()
 * Usage:                           call to output a int
 * Input:                           int to be sent
 * Return:                      none
 * Attention:                   none
 * Notes:                           none
---------------------------------------------------------------------*/
void U_Out_i(signed int val )
{
  char buffer[sizeof(int)*8+1];
  U_Out_s( (signed char *) itoa(val, buffer, 10) );
}

/*---------------------------------------------------------------------
 * Function details:            output a package via USART
 * Name:                            USART Send()
 * Usage:                           call to output the package pre-stored
in
 *                                      U WArray
 * Input:                           length of the package to be output
 * Return:                      Success: return 0, Fail: return 1
 * Attention:                   none
 * Notes:                           none
---------------------------------------------------------------------*/
unsigned char USART_Send(unsigned char length)
{
 unsigned char i = CLEAR;
 unsigned char *U_Send_P;

 U Send P = U WArray;
 for (i=CLEAR; i < length; i++)
 {
  USART Transmit(*U_Send_P++);
  _delay_ms(5);
 }

 return NO_ERROR;
}

/*---------------------------------------------------------------------
 * Function details:            USART receive interrupt service routing
 * Name:                            U RX S()
 * Usage:                           the received data are stored in
U RArray
 * Input:                           none
 * Return:                      none
```

```
 * Attention:                              none
 * Notes:                                  none
-------------------------------------------------------------------*/
void U_RX_S(void)
{
    U RArray[U Recv_Count] = UDRn;
    U_Recv_Count++;
    if (U_RECV_LENGTH == U_Recv_Count)
    {
        U Recv Count = 0;
        USART_REVD = 1;
    }
}


/* ---------------------------------------------------------------------------
 * Below are code obtained from Bruce Land, 2011-03
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <joerg@FreeBSD.ORG> wrote this file.  As long as you retain this notice you
 * can do whatever you want with this stuff. If we meet some day, and you think
 * this stuff is worth it, you can buy me a beer in return.        Joerg Wunsch
 * ---------------------------------------------------------------------------
 *
 * Stdio demo, UART implementation
 *
 * $Id: usart.c,v 1.1 2011/05/11 16:24:32 r Exp $
 *
 * Mod for mega644 BRL Jan2009
 */

/*
 * Send character c down the UART Tx, wait until tx holding register
 * is empty.
 */
int uart_putchar(char c, FILE *stream)
{

  if (c == '\a')
    {
      fputs("*ring*\n", stderr);
      return 0;
    }

  if (c == '\n')
    uart putchar('\r', stream);
  loop until bit is set(UCSRnA, UDRE0);
  UDRn = c;

  return 0;
}

/*
 * Receive a character from the UART Rx.
 *
 * This features a simple line-editor that allows to delete and
 * re-edit the characters entered, until either CR or NL is entered.
 * Printable characters entered will be echoed using uart_putchar().
 *
 * Editing characters:
 *
 * . \b (BS) or \177 (DEL) delete the previous character
 * . ^u kills the entire input buffer
 * . ^w deletes the previous word
 * . ^r sends a CR, and then reprints the buffer
 * . \t will be replaced by a single space
 *
 * All other control characters will be ignored.
 *
 * The internal line buffer is RX_BUFSIZE (80) characters long, which
```

```
 * includes the terminating \n (but no terminating \0).  If the buffer
 * is full (i. e., at RX BUFSIZE-1 characters in order to keep space for
 * the trailing \n), any further input attempts will send a \a to
 * uart putchar() (BEL character), although line editing is still
 * allowed.
 *
 * Input errors while talking to the UART will cause an immediate
 * return of -1 (error indication).  Notably, this will be caused by a
 * framing error (e. g. serial line "break" condition), by an input
 * overrun, and by a parity error (if parity was enabled and automatic
 * parity recognition is supported by hardware).
 *
 * Successive calls to uart getchar() will be satisfied from the
 * internal buffer until that buffer is emptied again.
 */
int uart_getchar(FILE *stream)
{
  unsigned char c;
  char *cp, *cp2;
  static char b[U RECV_BUFFER_SIZE];
  static char *rxp;

  if (rxp == 0)
    for (cp = b;;)
      {
      loop until bit is set(UCSRnA, RXC0);
      if (UCSRnA &  BV(FE0))
        return  FDEV EOF;
      if (UCSRnA &  BV(DOR0))
        return _FDEV_ERR;
      c = UDRn;
      /* behaviour similar to Unix stty ICRNL */
      if (c == '\r')
        c = '\n';
      if (c == '\n')
        {
          *cp = c;
          uart putchar(c, stream);
          rxp = b;
          break;
        }
      else if (c == '\t')
        c = ' ';

      if ((c >= (unsigned char)' ' && c <= (unsigned char)'\x7e') ||
          c >= (unsigned char)'\xa0')
        {
          if (cp == b + U RECV BUFFER SIZE - 1)
            uart_putchar('\a', stream);
          else
            {
          *cp++ = c;
          uart_putchar(c, stream);
            }
          continue;
        }

      switch (c)
        {
        case 'c' & 0x1f:
          return -1;

        case '\b':
        case '\x7f':
          if (cp > b)
            {
          uart putchar('\b', stream);
          uart_putchar(' ', stream);
          uart_putchar('\b', stream);
```

```
            cp--;
              }
            break;

          case 'r' & 0x1f:
            uart_putchar('\r', stream);
280         for (cp2 = b; cp2 < cp; cp2++)
              uart_putchar(*cp2, stream);
            break;

          case 'u' & 0x1f:
            while (cp > b)
              {
            uart_putchar('\b', stream);
            uart_putchar(' ', stream);
            uart_putchar('\b', stream);
290         cp--;
              }
            break;

          case 'w' & 0x1f:
            while (cp > b && cp[-1] != ' ')
              {
            uart_putchar('\b', stream);
            uart_putchar(' ', stream);
            uart_putchar('\b', stream);
300         cp--;
              }
            break;
          }
          }

        c = *rxp++;
        if (c == '\n')
          rxp = 0;

310     return c;
      }
```

```
/*-------------------------------------------------------------------------
 * Name:                    TWI.h
 * Target MCU:              ATMega168
 * Version:                 V 0.8 by Rick on 2011-05-08
-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
-------------------------------------------------------------------------*/
#define UBRRnL  UBRR0L
#define UBRRnH  UBRR0H
#define UCSRnA  UCSR0A
#define UCSRnB  UCSR0B
#define UCSRnC  UCSR0C
#define RXENn   RXEN0
#define TXENn   TXEN0
#define UMSELn0 UMSEL00
#define UCSZn1  UCSZ01
#define UCSZn0  UCSZ00
#define UDREn   UDRE0
#define RXCn    RXC0
#define UDRn    UDR0
/*-------------------------------------------------------------------------
Section 2:

        Please DO NOT change the below defines
-------------------------------------------------------------------------*/
#define USART_R_S()         UCSR0B|=(1<<RXCIE0)

/*-------------------------------------------------------------------------
Section 3:

        Declaration of functions
-------------------------------------------------------------------------*/
unsigned char USART Init(void);
void USART Transmit(unsigned char data);
unsigned char USART Receive(void);
void U Out s(signed char *s);
void U Out i(signed int val );
unsigned char USART_Send (unsigned char);
void RXC S(void);
extern unsigned char USART_REVD;
int uart putchar(char c, FILE *stream);
int uart_getchar(FILE *stream);
```

```
/*-------------------------------------------------------------------------
 * Name:                    error.h
 * Version:                 V 1.1 by Rick on 2008-04-15
-------------------------------------------------------------------------*/

//ERR array length
#define ERR_BUFFER_SIZE 10

//general ERR defines
#define NO_ERROR        0

//driver ERR defines
#define ERR CHIP INIT   1
#define ERR USART INIT  2
#define ERR USART SEND  3
#define ERR TWI INIT    4
#define ERR_TWI_SEND    5

//CMM ERR defines
#define ERR MAIN        10
#define ERR PC TASK     11
#define ERR SUB TASK    12
#define ERR CMM TASK    13
#define ERR TASK SWITCH 14
#define ERR_RST         15

//UDM ERR defines
#define ERR IR INIT             20
#define ERR EMPTY TWI COMMAND   21
#define ERR INVAILD TWI COMMAND 22
#define ERR WRONG SENDER        23
#define ERR_EMPTY_IR_COMMAND    24

//SCM ERR defines
#define ERR SCM INIT          30
#define ERR EMPTY SCM RCV     31
#define ERR CHECK FD          32
#define ERR CHECK TP          33
#define ERR_SCM_SEND          34

//ACM ERR defines
#define ERR MOTOR INIT            40
#define ERR_EMPTY_MOTOR_COMMAND   41
#define ERR_INVAILD_MOTOR_COMMAND 42

//DBM ERR defines
#define ERR NO SENDER         50
#define ERR_WRONG_COMMAND     51

//DBM exist defines
#define DBM_MAX_FAIL          3
```

```
/*-------------------------------------------------------------------------
 * Name:                    config.h
 * Version:                 V 1.1 by Rick on 2008-04-15
-------------------------------------------------------------------------*/
//type define
typedef unsigned char   uchar;
typedef unsigned int    uint;
typedef unsigned long   ulong;
typedef signed char     schar;
typedef signed int      sint;
typedef signed long     slong;

//define MCU speed
#define FOSC G      8000000
#define FOSCM_G     8

//define usart driver speed
#define USART_BAUD          9600
#define U RECV BUFFER SIZE  64
#define U SEND BUFFER SIZE  64
#define U RECV LENGTH       2
#define U_HEADER            0

//TWI input / output array buffer size
#define Module_TWI_Address  ACM_TWI_Address
#define TWI BUFFER SIZE     64
#define TWI MIN WAIT        20
#define TWI MAX TRY         100
#define TWI WAIT TIME       23
#define TWI_SPEED           400000

//U_RArray , TWI_RArray and TWI_WArray defines
#define U RECV LENGTH   2
#define SUB PASSWORD    0
#define SUB ADDRESS     0
#define SUB COMMAND     1
#define SUB_DATA        2

//define gobal tasks
#define DTAC TASK       0xFF
#define DBM ANN REQ     0xFE
#define DBM_ANN_RPY     0xFD

#define DBM_MAX_FAIL    3

//define TWI ADDRESS
#define GEN TWI Address 0x00
#define CMM TWI Address 0x01
#define SCM TWI Address 0x02
#define ACM_TWI_Address 0x03
#define UDM TWI Address 0x04
#define DBM_TWI_Address 0x05


#define PULSEM1 ON      PULSE2 ON
#define PULSEM1 OFF     PULSE2 OFF
#define PULSEM2_ON      PULSE0_ON
#define PULSEM2 OFF     PULSE0 OFF
#define M1 DIS          OC2A DISCON
#define M1 CON          OC2A CON
#define M2 DIS          OC0B DISCON
#define M2 CON          OC0B CON
#define M1 VALUE        OCR2A
#define M1 CNT          TCNT2
#define M2 VALUE        OCR0B
#define M2_CNT          TCNT0

//motor run time in ms
#define MOTOR_RUN_TIME 50
```

```
70      #define PWM_OUT_Count  10

        //PWM duty cycles
        #define MIN DUTY A     DUTY 20
        #define SLOW DUTY A    DUTY 50
        #define MID DUTY A     DUTY 70
        #define FAST_DUTY_A    DUTY_80

        #define MIN DUTY B     DUTY 20
        #define SLOW DUTY B    DUTY 50
80      #define MID DUTY B     DUTY 70
        #define FAST_DUTY_B    DUTY_80
```

```
/*-------------------------------------------------------------------------
 * Name:                    connection.h
 * Usage:                   Hardware connections such as pins and ports
 * Target MCU:              ATMega168
 * Version:                 V 0.1 by Rick on 2011-05-08
-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
-------------------------------------------------------------------------*/
//USART PORT
#define USART_DDR       DDRD
#define USART_PORT      PORTD
#define USART_RX_PIN    0
#define USART_TX_PIN    1


//TWI PORT
#ifndef TWI_DDR
#define TWI_DDR         DDRC
#endif
#ifndef TWI_PORT
#define TWI_PORT        PORTC
#endif
#ifndef TWI_CLK
#define TWI_CLK         5
#endif
#ifndef TWI_DATA
#define TWI_DATA        4
#endif


#define IN1A_DDR        DDRB
#define IN1A_PORT       PORTB
#define IN2A_DDR        DDRB
#define IN2A_PORT       PORTB
#define PWMA_DDR        DDRB
#define PWMA_PORT       PORTB
#define SBA_DDR         DDRD
#define SBA_PORT        PORTD

#define IN1B_DDR        DDRD
#define IN1B_PORT       PORTD
#define IN2B_DDR        DDRD
#define IN2B_PORT       PORTD
#define PWMB_DDR        DDRD
#define PWMB_PORT       PORTD
#define SBB_DDR         DDRD
#define SBB_PORT        PORTD


//output pin
#define IN1A_PIN        4
#define IN2A_PIN        5
#define PWMA_PIN        3
#define SBA_PIN         7

#define IN1B_PIN        4
#define IN2B_PIN        3
#define PWMB_PIN        5
#define SBB_PIN         7
```

```
/*-------------------------------------------------------------------------
 * Name:                    base.c
 * Usage:                   To control tank base.
 * Input:                   8-bit motor command data from Usart BUS.
 * Return:                  ERR if exists
 * Target MCU:              ATMega168
 * Version:                 V 0.2 by Rick on 2011-05-11
-------------------------------------------------------------------------*/
#include "base.h"
//#define DEBUG

/*-------------------------------------------------------------------
 * Function details:        main function
 * Name:                    main()
 * Usage:                   none
 * Input:                   none
 * Return:                  never if success
                            ERR_MAIN if error
 * Attention:               none
 * Notes:                   none
-------------------------------------------------------------------*/
int main (void)
{
  //init watch dog
  DOG_SLEEP;
  unsigned char ACM Status = CLEAR;      //UDM state machine
  unsigned char motor count = CLEAR;     //motor PWM counter
  ERR = Chip init();             //init this MCU
  stdout = stdin = stderr = &uart_str;

  //handler ERR if failed to init this MCU
  if (ERR)
  {
  ERR_Handler();
  }
/*  while(1)
    {
        if (USART_REVD)
        {
        USART Transmit(U RArray[SUB PASSWORD]);
        USART Transmit(U_RArray[SUB_COMMAND]);
        U Out s(" ");
        USART_REVD=0;
        }
        //else
        //U Out s("n");
        _delay_ms(1);
    }
*/
  while(TRUE)
  {
     Check Distance();
     if ((distance) && (distance<MIN DISTANCE) && (ADIR_MASK & U_RArray[SUB_COMMAND
]) && (BDIR_MASK & U_RArray[SUB_COMMAND]))
      {
         #ifdef DEBUG
         printf("disntance=%d ",distance);
         #endif
         Clean Up();
         ACM_Status=STATE_ACM_FREE;
      }
   USART Transmit('b');
   USART Transmit(distance);
   //state machine
   switch (ACM_Status)
    {
    case STATE COMMAND REVD:     //Command received from Usart, processing
          U_Recv_Count = CLEAR;                      //clear Usart counter
         if(CONNECT_PWD != U_RArray[SUB_PASSWORD]) //not sending by CMM
```

```
                {
                 #ifdef DEBUG
                 USART Transmit(U RArray[SUB PASSWORD]);
                 fprintf(stdout,"\r\nWrong Password\r\n");
                 #endif
                 ERR = ERR WRONG_SENDER;
                 Clean_Up();
                 ACM Status = STATE_ACM_FREE;
                 break;
                }

                #ifdef DEBUG
                fprintf(stdout,"\r\nCorrect Password\r\n");
                fprintf(stdout,"%d\n\r",U_RArray[SUB_COMMAND]);
                CMD Translate();
                #endif

                if (!(MOTOR_MASK & U_RArray[SUB_COMMAND]))
                {
                 ERR = ERR INVAILD_MOTOR_COMMAND;
                 Clean Up();
                 ACM Status = STATE_ACM_FREE;
                 break;
                }

                if (!(COMMAND_MASK & U_RArray[SUB_COMMAND]))
                {
                 ERR = ERR EMPTY_TWI_COMMAND;
                 Clean Up();
                 ACM Status = STATE_ACM_FREE;
                 break;
                }

                //done basic check, go to translate command
                ACM Status = STATE INIT_COMMAND;
           case STATE INIT COMMAND:
                ERR = Setup PWM();
                //U RArray[SUB_COMMAND]=CLEAR;
                Clean Up();
                if (ERR)                //Commands that may be dangerous
                {
                 printf("Error\r\n");
                 ACM_Status = STATE_ACM_FREE;
                 break;
                }
                Load Bullet();
                //Bullet is ready and fired, entering sending state
                ACM Status = STATE_SEND_COMMAND;
                break;
           case STATE_SEND_COMMAND:    //Sending PWM

                MOTOR RUN = SET;
                Fire Bullet();
                ACM_Status = STATE_ACM_FREE;

          for (motor_count = CLEAR; motor_count < PWM_OUT_Count; motor_count++)
          {
             if ((distance) && (distance<MIN DISTANCE) && (ADIR_MASK & U_RArray[SUB_COMMAND]
        ) && (BDIR_MASK & U_RArray[SUB_COMMAND]))
                 {
                     Clean Up();
                     ACM_Status=STATE_ACM_FREE;
                break;
              }
           if (USART_REVD)
           {
            ACM Status = STATE_COMMAND_REVD;
            break;
           }
```

```
        _delay_ms(MOTOR_RUN_TIME);
      }

140           break;
         case STATE_ACM_FREE:
           if (MOTOR_RUN)
           {
            //stop outputing PWM
            Cool_Motor();
           }
           if (USART_REVD)
           {
               Check Distance();
150            if ((distance) && (distance<MIN_DISTANCE) && (BDIR_MASK & ADIR_MASK &
    U RArray[SUB COMMAND]))
               {
                  Clean Up();
                  ACM_Status=STATE_ACM_FREE;
               }
                  ACM_Status = STATE_COMMAND_REVD;
           }
           break;
         default:
             break;
160      }
       }

       return ERR_MAIN;
     }


     /*---------------------------------------------------------------------
      * Function details:        chip initialization
      * Name:                    Chip init()
170   * Usage:                   to init the chip
      * Input:                   none
      * Return:                  NO ERROR if success
                                 ERR CHIP_INIT if fail
      * Attention:               none
      * Notes:                   none
     ---------------------------------------------------------------------*/
     unsigned char Chip_init(void)
     {
         unsigned char fail=CLEAR;               //fail flag
180      //disable gobal interrupts
         cli();
         //init gobal variables
         ERR = CLEAR;
         USART REVD = CLEAR;
         U_Recv_Count = CLEAR;

         MCUCR = CLEAR;              //init status and control registers
         PRR   = CLEAR;              //power controller

190      //init timers
         TIMSK0 = CLEAR;             //timer 0 interrupt sources
         TIMSK1 = CLEAR;             //timer 1 interrupt sources
         TIMSK2 = CLEAR;             //timer 2 interrupt sources

         //init extern interrupts
         EICRA  = CLEAR;             //extended ext ints
         EIMSK  = CLEAR;             //extended ext int marks
         PCMSK0 = CLEAR;             //pin change mask 0
         PCMSK1 = CLEAR;             //pin change mask 1
200      PCMSK2 = CLEAR;             //pin change mask 2
         PCICR  = CLEAR;             //pin change enable

         //init ports
         DDRB   = CLEAR;             //default
```

```
    DDRC   = CLEAR;             //default
    DDRD   = CLEAR;             //default

    //Chip Clear Reg();
    //Set PORT Output_Low();
    Motor Init();
    Distance_Init();


    //init UART
    USART Init();
    printf("Init Finished\r\n");

    //enable interrupts
    sei();

    if (fail)
    {
    return ERR_CHIP_INIT;
    }
    return NO_ERROR;

}

/*---------------------------------------------------------------------
 * Function details:        convert input commands
 * Name:                    CMD Translate()
 * Usage:                   call this function convert the input CMD
 * Input:                   none
 * Return:                  none
 * Attention:               TO BE UPDATED
 * Notes:                   none
---------------------------------------------------------------------*/
void CMD_Translate(void)
{
          char CMD=U RArray[SUB COMMAND];
          U RArray[SUB COMMAND]=CLEAR;
          //U RArray[SUB_COMMAND]=(1<<MOTOR_MASK);
          if ('x'==CMD)
             {Clean Up();
             reboot();}
          else if ('f'==CMD)
             U_RArray[SUB_COMMAND]= 0b11000100;
          else if ('b'==CMD)
             U RArray[SUB COMMAND]= 0b10000000;
          else if ('l'==CMD)
             U RArray[SUB COMMAND]= 0b10000100;
          else if ('r'==CMD)
             U RArray[SUB COMMAND]= 0b11000000;
          else if ('s'==CMD)
             U_RArray[SUB_COMMAND]= 0b10001000;

          if ('f'==U RArray[SUB DATA])
             U RArray[SUB COMMAND]|=0b10110011;
          else if ('m'==U RArray[SUB DATA])
             U RArray[SUB COMMAND]|=0b10100010;
          else if ('l'==U_RArray[SUB_DATA])
             U RArray[SUB_COMMAND]|=0b10010001;
         CMD=CLEAR;

}


void Clean_Up(void)
{
 //handle error if any
 if (!ERR)
 {
    ERR_Handler();
```

```
    }
    //clean up
    memset(U_RArray, CLEAR, U_RECV_LENGTH);     //clear Usart RArray array
    U_Recv_Count = CLEAR;                       //clear Usart counter
    USART_REVD = CLEAR;                         //clear USART_REVD flag
}
/*---------------------------------------------------------------------
 * Function details:        translate and setup motor command
 * Name:                    Setup_PWM
 * Usage:                   translate commands received via USART
 * Input:                   none
 * Return:                  NO_ERROR if success
                            ERR_INVAILD_MOTOR_COMMAND if fail
 * Attention:               TO BE UPDATED
 * Notes:                   none
---------------------------------------------------------------------*/
unsigned char Setup_PWM (void)
{
 unsigned char fail = CLEAR;

 if (MOTOR_RUN && (Current_Command != U_RArray[SUB_COMMAND]))
 {
  STOP_MOTOR;
   delay_ms(MOTOR_RUN_TIME);
  //standy
  SBA_LOW;
  SBB_LOW;
 }

 Current_Command = U_RArray[SUB_COMMAND];

 //first check if we shall stop the motor immediatly
 if (STOP_MASK & Current_Command)
 {
  STOP_MOTOR;
  //clear timer setting
  M1_VALUE = CLEAR;
  M2_VALUE = CLEAR;

  return NO_ERROR;
 }

 //set up motorA
 //direction of motorA, CW if set, CCW is clear
 if (ADIR_MASK & Current_Command)
 {
  IN1A_HIGH;
  IN2A_LOW;
  //U_Out_s("Af");
 }
 else
 {
  IN1A_LOW;
  IN2A_HIGH;
 }

 //speed of motorA
 switch ((ASPD_MASK & Current_Command)>>4)
 {
  case SLOW:
       M1_VALUE = SLOW_DUTY_A;
       break;
  case MIDDLE:
       M1_VALUE = MID_DUTY_A;
       break;
  case FAST:
       M1_VALUE = FAST_DUTY_A;
       break;
  case NOSPEED:
```

```
            M1_VALUE = CLEAR;
         default:
           break;
        }

        //set up motorB
        //direction of motorB, CCW if set, CW is clear
350     if (BDIR_MASK & Current_Command)
        {
         IN1B_HIGH;
         IN2B_LOW;
         //U_Out_s("Bf");
        }
        else
        {
         IN1B_LOW;
         IN2B_HIGH;
360     }
        //speed of motorB
        switch (BSPD_MASK & Current_Command)
        {
         case SLOW:
              M2_VALUE = SLOW_DUTY_B;
              break;
         case MIDDLE:
              M2_VALUE = MID_DUTY_B;
              break;
370      case FAST:
              M2_VALUE = FAST_DUTY_B;
              break;
         case NOSPEED:
              M2_VALUE = CLEAR;
         default:
           break;
        }
        //fprintf(stdout,"M1 %d, M2 %d",M1_VALUE,M2_VALUE);
        if (fail)
380     return ERR_INVAILD_MOTOR_COMMAND;
        return NO_ERROR;
       }

       /*---------------------------------------------------------------------
        * Function details:        load the values into timers
        * Name:                    Load_Bullet()
        * Usage:                   none
        * Input:                   none
        * Return:                  none
390     * Attention:               none
        * Notes:                   none
       ---------------------------------------------------------------------*/
       void Load_Bullet(void)
       {
        //PWM pins
        M1_CON;              //connect to OCnA
        M2_CON;              //connect to OCnB

        //standby pins
400     SBA_HIGH;            //make sure SBA is high when start
        SBB_HIGH;            //make sure SBB is high when start
       }

       /*---------------------------------------------------------------------
        * Function details:        Start timers and output PWM
        * Name:                    Fire_Bullet()
        * Usage:                   none
        * Input:                   none
        * Return:                  none
410     * Attention:               none
        * Notes:                   none
```

```
------------------------------------------------------------------------*/
void Fire_Bullet(void)
{
 PULSEM1 ON;          //start output PWM pulse 1
 PULSEM2 ON;          //start output PWM pulse 2
 //U_Out_s("FIRE\r\n");
}
/*------------------------------------------------------------------------
 * Function details:        load the values into timers
 * Name:                    Cool_Motor()
 * Usage:                   none
 * Input:                   none
 * Return:                  none
 * Attention:               none
 * Notes:                   none
------------------------------------------------------------------------*/
void Cool_Motor(void)
{
 PULSEM1 OFF;        //stop output PWM pulse
 PULSEM2_OFF;        //stop output PWM pulse

 //disconnnect PWM pins
 M1 DIS;         //disconnect from OCnA
 M2_DIS;         //disconnect from OCnB

 //set motor pins to low
 LOW_PINS;

 //clear timer
 M1 CNT = CLEAR;
 M2_CNT = CLEAR;

 MOTOR_RUN = CLEAR;
}

/*------------------------------------------------------------------------
 * Function details:        handler ERR if exists
 * Name:                    ERR Handler()
 * Usage:                   call this function to handler ERR message
 * Input:                   none
 * Return:                  none
 * Attention:               TO BE UPDATED
 * Notes:                   none
------------------------------------------------------------------------*/
void ERR_Handler(void)
{
;
}

ISR(USART_RX_vect)
{
//test=UDR0;
//USART_REVD = 1;
//if ('x'==UDR0)
    //reboot();
U RArray[U Recv_Count] = UDR0;
U_Recv_Count++;
if (U_RECV_LENGTH == U_Recv_Count)
{
U Recv Count = 0;
USART_REVD = 1;
}
}

/*------------------------------------------------------------------------
 * Function details:        reboot the chip
 * Name:                    reboot()
 * Usage:                   call this function to reset the chip
 * Input:                   none
```

```
 * Return:                  none
 * Attention:               TO BE UPDATED
 * Notes:                   none
-------------------------------------------------------------------------*/
void reboot(void)
{
  //wdt_disable();
  wdt_enable(WDTO_15MS);
  while (1) {}
}
```

```
/*-------------------------------------------------------------------------
 * Name:                    base.h
 * Version:                 V 1.6 by Rick on 2011-05-11
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
-------------------------------------------------------------------------*/
//connection password
#define CONNECT_PWD     97
#define MIN_DISTANCE    30

//MCU speed
//Timer divider defines must also be changed with this MCU speed
#define FOSC            8000000
#define FOSCM           8

//only enable in debug mode
#define DEBUG

/*-------------------------------------------------------------------------
Section 2:

        Please DO NOT change the below defines
-------------------------------------------------------------------------*/

//default values
#define TRUE        1
#define FALSE       0
#define ENABLE      1
#define DISABLE     0
#define SET         1
#define CLEAR       0x00
#define ALLSET      0xFF

//include files
#include <avr/io.h>
//define interrupt service rountings
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "error.h"
//genral config and settings
#include "config.h"
#include "connection.h"
//drivers
#include "usart.c"
#include "motor.c"
#include "UltraICP.c"

#define BASE

//general variables
unsigned char ERR;                  //error flag
unsigned char MOTOR_RUN;            //Motor moving done flag
unsigned char Current_Command;      //Motor moving done flag

//speed defines
#define NOSPEED         0
#define SLOW            1
#define MIDDLE          2
#define FAST            3
```

```
70      //puls defines
        #define DUTY 0          0x00
        #define DUTY 10         0x1A
        #define DUTY 20         0x33
        #define DUTY 22         0x38
        #define DUTY 23         0x3A
        #define DUTY_25         0x40
        #define DUTY 30         0x4D
        #define DUTY 32         0x52
        #define DUTY 33         0x54
80      #define DUTY 35         0x5A
        #define DUTY 40         0x66
        #define DUTY 50         0x80
        #define DUTY 52         0x85
        #define DUTY 53         0x87
        #define DUTY 55         0x8C
        #define DUTY 60         0x99
        #define DUTY_70         0xB3
        #define DUTY 80         0xCC
        #define DUTY 90         0xE5
90      #define DUTY_100        0xFF

        //watch dog define
        #define DOG SLEEP       {MCUSR &= (~(SET<<WDRF));WDTCSR|=((SET<<WDCE)|(SET<<WDE));
        WDTCSR=CLEAR;}

        //TASK DEFINES
        //PWM off define
        //#define PULSE3 ON    {TCNT3=0x00; TCCR3B |= (1<<CS30);}
        //#define PULSE3 OFF       {TCCR3B = 0x00;}
        #define PULSE2 ON      {TCNT2=0x00; TCCR2B |= (1<<CS20);}
100     #define PULSE2_OFF     {TCCR2B = 0x00;}
        #define PULSE1 ON      {TCNT1=0x00; TCCR1B |= (1<<CS10);}
        #define PULSE1 OFF     {TCCR1B = 0x00;}
        #define PULSE0 ON      {TCNT0=0x00; TCCR0B |= (1<<CS00);}
        #define PULSE0_OFF     {TCCR0B = 0x00;}

        //OC3A connection
        //#define OC3A CON     {TCCR3A |= (1<<COM3A1);}
        //#define OC3A DISCON      {TCCR3A &= ~ (1<<COM3A1);}
        //OC3B connection
110     //#define OC3B CON     {TCCR3A |= (1<<COM3B1);}
        //#define OC3B_DISCON      {TCCR3A &= ~ (1<<COM3B1);}
        //OC2A connection
        #define OC2A CON       {TCCR2A |= (1<<COM2A1);}
        #define OC2A DISCON    {TCCR2A &= ~ (1<<COM2A1);}
        //OC1A connection
        #define OC1A CON       {TCCR1A |= (1<<COM1A1);}
        #define OC1A DISCON    {TCCR1A &= ~ (1<<COM1A1);}
        //OC1B connection
        #define OC1B CON       {TCCR1A |= (1<<COM1B1);}
120     #define OC1B DISCON    {TCCR1A &= ~ (1<<COM1B1);}
        //OC0B connection
        #define OC0B CON       {TCCR0A |= (1<<COM0B1);}
        #define OC0B_DISCON    {TCCR0A &= ~ (1<<COM0B1);}

        //stop motor
        #define STOP MOTOR     {IN1A PORT &= (~(SET<<IN1A PIN)); \
                                IN2A PORT &= (~(SET<<IN2A PIN)); \
                                SBA PORT |= (SET<<SBA PIN); \
                                IN1B PORT &= (~(SET<<IN1B PIN)); \
130                             IN2B PORT &= (~(SET<<IN2B PIN)); \
                                SBB_PORT |= (SET<<SBB_PIN);}
        //low pins
        #define LOW PINS       {IN1A PORT &= (~(SET<<IN1A PIN)); \
                                IN2A PORT &= (~(SET<<IN2A PIN)); \
                                PWMA PORT &= (~(SET<<PWMA PIN)); \
                                SBA_PORT &= (~(SET<<SBA_PIN)); \
                                IN1B_PORT &= (~(SET<<IN1B_PIN)); \
```

```
                            IN2B PORT &= (~(SET<<IN2B PIN)); \
                            PWMB PORT &= (~(SET<<PWMB PIN)); \
140                         SBB_PORT &= (~(SET<<SBB_PIN));}

    //PWM pins
    #define PWMA LOW       {PWMA PORT &= ~(SET<<PWMA PIN);}
    #define PWMB_LOW       {PWMB_PORT &= ~(SET<<PWMB_PIN);}

    //IN1/2 pins
    #define IN1A HIGH      {IN1A PORT |= (SET<<IN1A PIN);}
    #define IN2A HIGH      {IN2A PORT |= (SET<<IN2A PIN);}
    #define IN1B HIGH      {IN1B PORT |= (SET<<IN1B PIN);}
150 #define IN2B_HIGH      {IN2B_PORT |= (SET<<IN2B_PIN);}

    #define IN1A LOW       {IN1A PORT &= ~(SET<<IN1A PIN);}
    #define IN2A LOW       {IN2A PORT &= ~(SET<<IN2A PIN);}
    #define IN1B LOW       {IN1B PORT &= ~(SET<<IN1B PIN);}
    #define IN2B_LOW       {IN2B_PORT &= ~(SET<<IN2B_PIN);}

    //standby pins
    #define SBA HIGH       {SBA PORT |= (SET<<SBA PIN);}
    #define SBB_HIGH       {SBB_PORT |= (SET<<SBB_PIN);}
160
    #define SBA LOW        {SBA PORT &= ~(SET<<SBA PIN);}
    #define SBB_LOW        {SBB_PORT &= ~(SET<<SBB_PIN);}

    //State machine defines
    #define STATE ACM FREE        0
    #define STATE COMMAND REVD    1
    #define STATE INIT COMMAND    2
    #define STATE_SEND_COMMAND    3

170 //MASK define
    #define MOTOR MASK     0x80
    #define COMMAND MASK   0x7F
    #define STOP MASK      0x08
    #define ADIR MASK      0x40
    #define ASPD MASK      0x30
    #define BDIR MASK      0x04
    #define BSPD_MASK      0x03

    /*-------------------------------------------------------------------------
180 Section 3:

            Declaration of functions
    -------------------------------------------------------------------------*/

    void ERR Handler(void);
    uchar Chip init(void);
    void CMD_Translate(void);
    void Clean Up(void);
    uchar Setup PWM (void);
190 void Load Bullet(void);
    void Fire Bullet(void);
    void Cool Motor(void);
    void reboot(void);
```

```
/*-------------------------------------------------------------------------
 * Name:                    motor.c
 * Driver Details:          PWM drives DC motor
 * Usage:                   Call Motor_Init to init
 * Input:                   None
 * Return:                  ERR if exists
 * Attention:               Module settings will overwrite the
                            default settings in motor.h
 * Note:                    To be used on ATMega168.
                            Only "Section 1" in motor.h shall be changed when
                            necessary.
 * Version:                 V 0.2 by Rick on 2011-05-11
-------------------------------------------------------------------------*/
#include "motor.h"
/*-------------------------------------------------------------------
 * Function details:        init Timer3 to PWM pulse train
 * Name:                    Init_Timer3()
 * Usage:                   use PULSE3A ON to start ouptut PWM pulse,
                            use PULSE3A_OFF to stop output PWM pulse.
 * Input:                   none
 * Return:                  NO ERROR if success
 * Attention:               none
 * Notes:                   none
-------------------------------------------------------------------*/
/*
uchar Init_Timer3(void)
{
 //init PWM
 //invert pin, PWM phase correct mode
 TCCR3A |= (1<<COM3A1) | (1<<COM3B1) | (1<<WGM30);
 TCCR3B = CLEAR;
 TCNT3  = CLEAR;
  //speed per invert
 OCR3A  = CLEAR;
 TIMSK3 = CLEAR;
 return NO_ERROR;
}*/

/*-------------------------------------------------------------------
 * Function details:        init Timer2 to PWM pulse train
 * Name:                    Init_Timer2()
 * Usage:                   use PULSE2A_ON to start ouptut PWM pulse,
                            use PULSE2A_OFF to stop output PWM pulse.
 * Input:                   none
 * Return:                  NO ERROR if success
 * Attention:               none
 * Notes:                   none
-------------------------------------------------------------------*/
uchar Init_Timer2(void)
{
 //init PWM
 //invert pin, PWM phase correct mode
 TCCR2A |= (1<<COM2A1) | (1<<WGM20);
 TCCR2B = CLEAR;
 TCNT2  = CLEAR;
  //speed per invert
 OCR2A  = CLEAR;
 TIMSK2 = CLEAR;
 return NO_ERROR;
}
```

```
/*---------------------------------------------------------------------
 * Function details:        init Timer1 to PWM pulse train
 * Name:                    Init Timer1()
 * Usage:                   use PULSE1A ON to start ouptut PWM pulse,
                            use PULSE1A_OFF to stop output PWM pulse.
 * Input:                   none
 * Return:                  NO_ERROR if success
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
uchar Init_Timer1(void)
{
 //init PWM
 //invert pin, PWM phase correct mode
 TCCR1A |= (1<<COM1B1) | (1<<WGM10);
 TCCR1B = CLEAR;
 TCNT1  = CLEAR;
  //speed per invert
 OCR1B  = CLEAR;
 TCNT1=0x00;
 return NO_ERROR;
}

/*---------------------------------------------------------------------
 * Function details:        init Timer0 to PWM pulse train
 * Name:                    Init Timer0()
 * Usage:                   use PULSE0A ON to start ouptut PWM pulse,
                            use PULSE0A_OFF to stop output PWM pulse.
 * Input:                   none
 * Return:                  NO ERROR if success
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
uchar Init_Timer0(void)
{
 //init PWM
 //invert pin, PWM phase correct mode
 TCCR0A |= (1<<COM0B1) | (1<<WGM00);
 TCCR0B = CLEAR;
 TCNT0  = CLEAR;
  //speed per invert
 OCR0B  = CLEAR;
 TIMSK0 = CLEAR;
 return NO_ERROR;
}

/*---------------------------------------------------------------------
 * Function details:        Motor initialization
 * Name:                    Motor Init()
 * Usage:                   to init the PWM and control signals
 * Input:                   none
 * Return:                  NO ERROR if success
                            ERR IR_INIT if fail
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
uchar Motor_Init (void)
{
 uchar fail = CLEAR;

 //motor A
 //MOTORA_DDR = MOTORA_DDR | (1<<IN1A_PIN) | (1<<IN2A_PIN) | (1<<PWMA_PIN) | (1<<
SBA PIN);
 //MOTORA PORT = MOTORA_PORT & ~(1<<IN1A_PIN) & ~(1<<IN2A_PIN) & ~(1<<PWMA_PIN) & ~
(1<<SBA PIN);
 IN1A DDR |= (SET<<IN1A PIN);
 IN2A DDR |= (SET<<IN2A PIN);
 PWMA_DDR |= (SET<<PWMA_PIN);
 SBA_DDR |= (SET<<SBA_PIN);
```

```
        IN1A_PORT &= (~(SET<<IN1A_PIN));
        IN2A_PORT &= (~(SET<<IN2A_PIN));
140     PWMA_PORT &= (~(SET<<PWMA_PIN));
        SBA_PORT &= (~(SET<<SBA_PIN));


        //motor B
        //MOTORB_DDR = MOTORB_DDR | (1<<IN1B_PIN) | (1<<IN2B_PIN) | (1<<PWMB_PIN) | (1<<
       SBB_PIN);
        //MOTORB_PORT = MOTORB_PORT & ~(1<<IN1B_PIN) & ~(1<<IN2B_PIN) & ~(1<<PWMB_PIN) & ~
       (1<<SBB_PIN);
        IN1B_DDR |= (SET<<IN1B_PIN);
        IN2B_DDR |= (SET<<IN2B_PIN);
        PWMB_DDR |= (SET<<PWMB_PIN);
150     SBB_DDR |= (SET<<SBB_PIN);

        IN1B_PORT &= (~(SET<<IN1B_PIN));
        IN2B_PORT &= (~(SET<<IN2B_PIN));
        PWMB_PORT &= (~(SET<<PWMB_PIN));
        SBB_PORT &= (~(SET<<SBB_PIN));

        fail |= Init_Timer2();
        fail |= Init_Timer0();

160     if (!fail)
        {
         return NO_ERROR;
        }
        return ERR_IR_INIT;
       }
```

```
/*-------------------------------------------------------------------------
 * Name:                    motor.h
 * Version:                 V 1.3 by Rick on 2011-05-11
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
        Default settings are written for Atmega168
-------------------------------------------------------------------------*/

//output port
/*
#ifndef MOTORA_DDR
#define MOTORA_DDR      DDRB
#endif
#ifndef MOTORA_PORT
#define MOTORA_PORT     PORTB
#endif
#ifndef MOTORB_DDR
#define MOTORB_DDR      DDRD
#endif
#ifndef MOTORB_PORT
#define MOTORB_PORT     PORTD
#endif

//output pin
#ifndef IN1A_PIN
#define IN1A_PIN        5
#endif
#ifndef IN2A_PIN
#define IN2A_PIN        4
#endif
#ifndef PWMA_PIN
#define PWMA_PIN        3
#endif
#ifndef SBA_PIN
#define SBA_PIN         0
#endif
#ifndef IN1B_PIN
#define IN1B_PIN        7
#endif
#ifndef IN2B_PIN
#define IN2B_PIN        6
#endif
#ifndef PWMB_PIN
#define PWMB_PIN        5
#endif
#ifndef SBB_PIN
#define SBB_PIN         0
#endif
*/
/*-------------------------------------------------------------------------
Section 2:

        Please DO NOT change the below defines
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
Section 3:

        Declaration of functions
-------------------------------------------------------------------------*/
//uchar Init_Timer3(void);
uchar Init_Timer2(void);
uchar Init_Timer1(void);
uchar Init_Timer0(void);
```

```
70      uchar Motor_Init (void);
```

```c
/*****************************************/
//File Name: UltraICP.c
//Function: Input Capture Ultrasonic Signal
//Writer: Jessica
//Date: Mar, 2008
//Target MCU: ATmega168
//Ocillator: define in config.h
/*****************************************/

#ifndef UltraICP C
#define UltraICP_C
#endif

#include "UltraICP.h"

unsigned char startrecord;
unsigned char EdgeUp;
unsigned int distance;
unsigned int time;
unsigned int temp;

//initial chip for distance dectection
unsigned char Distance_Init (void)
{
  //inital ports
  //DDRB |= (1<<0);  //set B port output mode
  //PORTB &= ~(1<<0);  //set low gain

  //initial timer/counter 1
  TCCR1A = 0x00;
  TCCR1B = 0x00;  //stop T/C1 first
  TCNT1 = 0x000;   //clear timer
  ICR1 = 0x000;   //clear input capture register
  TIMSK1 = 0x00;  //clear T/C1 interrupt mask register

  return 0;
}

//interrupt service routing for #pragma interrupt_handler ICP_fun:11
ISR(TIMER1_CAPT_vect)
{
  if (1 == EdgeUp)
  {
      time = ICR1;
      TC1 Stop();  //stop timer
      ICP Disable();  //disable input capture
      TCNT1 = 0x000;   //clear timer
      ICR1 = 0x000;

      EdgeUp = 0;
      startrecord = 0;
  }
  else
  {
      //TC1 Clk 8();         //trigger timer for clk/8
      TCCR1B |= (1<<CS11);
      TCCR1B &= ~(1<<CS12);
      TCCR1B &= ~(1<<CS10);

      TC1 ICPSet1();  //set falling edge trigger
      TC1 Overflow();  //nable T/C1 overflow
      EdgeUp ++;
  }
}

//interrupt service routing for #pragma interrupt_handler TC1_overflow:14
ISR(TIMER1_OVF_vect)
{
    //printf("OVF\n");
```

```
70        TC1_Stop();  //stop timer
          ICP_Disable();  //disable input capture
          TCNT1 = 0x000;   //clear timer

          distance = 0xFF;   //distance out of range
          EdgeUp = 0;
          startrecord = 0;
        }

        //send trigger pulse to ultrasonic sensor
80      unsigned char start(void)
        {
          startrecord = 1;
          DDRB |= (1<<0);   //set B port output mode
          PORTB |= (1<<0);  //PB0(ICP1) output a 5us pulse
           delay_us(10);
          PORTB &= ~(1<<0);  //stop pulse
          DDRB &= ~(1<<0);  //set B port input mode

          return 0;
90      }

        /*---------------------------------------------------------------------
         * Function details:  check distane from ultrasonic sensor PING)))
         * Name:              Check_Distance(
         * Usage:             none
         * Input:             none
         * Return:            Success: distance (8 bit, 3cm to 254cm, 0x11 to 0xFE)
                              Fail: 0x01
                              Out of Range: 0xFF
100      * Attention:         none
         * Notes:             none
        ---------------------------------------------------------------------*/
        unsigned int Check_Distance(void)
        {
          unsigned char fail = 0;
          unsigned char i = 0;
          cli();
          EdgeUp = 0;
          distance = 0x00;
110       temp = 0x000;
          startrecord = 0;

          fail = start();  //send trigger pulse to ultrasonic sensor

          ICP_FlagClear();  //clear ICP flag
          TC1_ICPSet2();  //set rising edge trigger
          ICP_Enable();  //enable input capture

          sei();
120
          while (startrecord == 1)
          {
             delay_us(50);
            i++;

            if (i == MAX_TRY_DISTANCE)
            {
              fail |= 1;
              break;
130         }
          }

              temp =  0.13736 * time / 8;  //cm.

              if ((temp <= 3) | (temp >= 255))
                distance = 0xFF;
              else
                distance = temp;
```

```
140       /*if ((fail) || (0==distance))
            {distance=255;}*/
            return distance;
        }

        /********************************** N O T E ***********************************
        ****************/

        /*
        Timer1 overflow issue
        Ultrasonic sensor measure distance range is from 2cm to 3m
150     since time(us)*0.01717=distance(cm), time is from about 116us to 17472us
        A 16 bits timer can express data from 0 to 65535
        1 jumper of timer = CPU/scalar = 1/FREQ*Clk (Clk >= 1)
        which should be smaller than 17472us/65535 = 0.2666E-6
        So FREQ/Clk should be smaller than 3.75E6
        possible choices: FREQ=8MHz, Clk=8/64/...; or FREQ=20MHz, Clk=8/64/...; or FREQ=
        4MHz, Clk=1/8/64/...
        so do not need to worry about overflow
        */

        /*****************************************************************************
        ***************/
```

```
#ifndef UltraICP H
#define UltraICP_H
#endif

//ICP operation

#define TC1_Stop()       (TCCR1B &= ~(1<<CS11))  //stop T/C1
#define TC1 ICPSet1()    (TCCR1B &= ~(1<<ICES1))  //set falling edge trigger
#define TC1 ICPSet2()    (TCCR1B |= (1<<ICES1))  //set rising edge trigger
#define TC1 Clk 8()      (TCCR1B |= (1<<CS11))  //trigger timer for clk/8
#define TC1_Overflow()   (TIMSK1 |= (1<<TOIE1))  //enable T/C1 overflow

#define ICP Disable()    (TIMSK1 &= ~(1<<ICIE1))  //disable input capture
#define ICP Enable()     (TIMSK1 |= (1<<ICIE1))  //enable input capture
#define ICP_FlagClear() (TIFR1 |= (1<<ICF1))  //clear ICP flag

#define MAX_TRY_DISTANCE         250  //waiting interrupt time
```

```c
/*-------------------------------------------------------------------------
 * Name:                    TWI.c
 * Usage:                   TWI driver
 * Target MCU:              ATMega168
 * Version:                 V 0.8 by Rick on 2011-05-08
-------------------------------------------------------------------------*/
#include "TWI.h"

//Global variables
unsigned char TWI WArray[TWI BUFFER SIZE];  //input output array
unsigned char TWI RArray[TWI BUFFER SIZE];  //output array point
unsigned char *TWI R P = NULL;              //temp input array point used in ISR
unsigned char TWI N S;
unsigned char TWI REVD;                     //TWI reveived flag
unsigned char TWI_RNum;                     //length of TWI package

/*-------------------------------------------------------------------------
 * Function details:        TWI Initialize
 * Name:                    TWI INIT()
 * Usage:                   to init TWI of this MCU
 * Input:                   Address: 7 bit TWI address of this MCU
                            GCall:   Accept General Call 1 / Otherwire 0
 * Return:                  Success: return 0, Fail: return 1
 * Attention:               none
 * Notes:                   none
-------------------------------------------------------------------------*/
unsigned char TWI_INIT(unsigned char Address, unsigned char GCall)
{
  TWI Clean();
  TWI DDR &= (~(1<<TWI CLK)) & (~(1<<TWI DATA));
  TWI PORT |= (1<<TWI CLK) | (1<<TWI DATA);
  TWBR=TWI_BR;                          //TWI BUS Speed
  TWAR=(Address<<1)|GCall;              //TWI address of this MCU
  TWDR=0x00;                            //clear data register
  TWSR=0x00;                            //clear status register
  TWI Slave();                          //default slave mode
  return 0;
}

/*-------------------------------------------------------------------------
 * Function details:        TWI Master Transmit with Resend ability
 * Name:                    TWI MT RESEND()
 * Usage:                   Keep resending msg via TWI for MAX_TRY
 * Input:                   Address
 * Return:                  Success: return 0, Fail: return 1
 * Attention:               default values of TWI MAX TRY and
                            TWI WAIT TIME are defined in TWI.h
                            and the values in the header file of MCU
                            will overwrite them
 * Notes:                   none
-------------------------------------------------------------------------*/
unsigned char TWI_MT_RESEND(unsigned char Address, unsigned int Length)
{
  unsigned char fail = CLEAR;               //fail flag
  unsigned char try = CLEAR;                //try # counter

  //try to resend for MAX_TRY times before give up
  for (try = CLEAR;try < TWI_MAX_TRY; try ++)
  {
    fail = TWI MT(Address, Length); //try to send data via TWI
    if (!fail)                      //flag is 0 if send success
       return NO ERROR;             //task done with no error
    _delay_ms(TWI_MIN_WAIT+TWI_WAIT_TIME);  //elsewise, delay and retry
  }
  return fail;
}

/*-------------------------------------------------------------------------
 * Function details:        TWI Master Transmit
```

```
70       * Name:                    TWI MT()
         * Usage:                   to send multi data to Address via TWI
         * Input:                   Address
         * Return:                  Success: return 0, Fail: return 1
         * Attention:               none
         * Notes:                   none
        ----------------------------------------------------------------------*/
        unsigned char TWI_MT(unsigned char Address, unsigned int Length)
        {
              unsigned int counter = CLEAR;             //count # of byte sent
80            unsigned char *TWI W P=NULL;
              TWI_W_P=TWI_WArray;               //Save the current pointer

              if (TWI N S)                      //if MCU is slaving existing master
                return ERR TWI SEND;            //return and wait to resend
                                                //this prevent hang due BUS START error

              TWI_Start();                      //sent Start
              TWI_Wait();                       //wait for reply

90            if(TWI TestAck()!=START)          //check if Start is sent through
                return ERR_TWI_SEND;

              Address=(Address<<1)&WD;          //combine the 7 bit Address and W
              TWI_Write8Bit(Address);           //send Address and W
              TWI_Wait();                       //wait for reply

              switch (TWI_TestAck())
              {
               case MT SLA ACK:                 //SLA W sent successfully
100                 TWI Write8Bit(*TWI_W_P++);  //send Wdata
                    counter ++;
                    break;
               case MT SLA NOACK:               //Not Ack received
                    TWI Stop();                 //Stop TWI and quit
                     delay ms(5);               //Delay needed between Stop and Slave
                    TWI Slave();                //Slave mode
                    return ERR TWI SEND;
               case M ARB:                      //Arb lost in slave address/data
               case S ARB R:                    //Arb lost, slave R mode
110            case S ARB G:                    //Arb lost, slave General call R mode
               case S ARB_T:                    //Arb lost, slave General call T mode
               default:
                    TWI Slave();                //Slave mode
                    return ERR_TWI_SEND;        //Arb Lost, quit
              }

           while(TRUE)
           {
              TWI_Wait();
120
              switch (TWI_TestAck())
              {
               case MT DATA ACK:                //Data sent though
                    if(counter < Length)        //if more data in output array
                    {
                    TWI_Write8Bit(*TWI_W_P++);  //send Wdata
                    counter ++;
                    break;
                    }
130                 TWI Stop();                 //send task done if output array empty
                     delay ms(5);               //Delay needed between Stop and Slave
                    TWI Slave();                //Slave mode
                    return NO_ERROR;            //Successful, return

               case MT DATA NOACK:              //Not Ack received
                    TWI Stop();                 //Stop TWI and quit
                    _delay_ms(5);               //Delay needed between Stop and Slave
                    TWI_Slave();                //Slave mode
```

```
                return ERR_TWI_SEND;          //Error, return
140          case M_ARB:                      //Arb lost in slave address/data
             default:
                TWI_Slave();                  //Slave mode
                return ERR_TWI_SEND;          //Arb Lost, quit
             }
          }
          return ERR_TWI_SEND;
       }

       /*---------------------------------------------------------------------
150     * Function details:        TWI Master Receive
        * Name:                    TWI_MR()
        * Usage:                   to receive 1 byte data to Address via TWI
        * Input:                   Address
        * Return:                  0 if success or 1 if fail
        * Attention:               none
        * Notes:                   none
       ---------------------------------------------------------------------*/
       unsigned char TWI_MR(unsigned char Address)
       {
160          TWI_Start();                     //sent Start
             TWI_Wait();                      //wait for reply
             if(TWI_TestAck()!=START)
               return 1;                      //check if Start is sent through

             Address=(Address<<1)|RD;         //combine Address and R
             TWI_Write8Bit(Address);          //send Address and R
             TWI_Wait();                      //wait for reply
             switch (TWI_TestAck())
             {
170           case MR_SLA_ACK:                //SLA_R sent successfully
                TWI_Receive();                //start to receive data
                break;
              case MR_SLA_NOACK:              //Not Ack received
                TWI_Stop();                   //Stop TWI and quit
                 delay_ms(5);                 //Delay needed between Stop and Slave
                TWI_Slave();                  //Slave mode
                return 1;
              case M_ARB:                     //Arb lost in slave address/data
              case S_ARB_R:                   //Arb lost, slave R mode
180           case S_ARB_G:                   //Arb lost, slave General call R mode
              case S_ARB_T:                   //Arb lost, slave General call T mode
              default:
                TWI_Slave();                  //Slave mode
                return 1;                     //Arb lost, quit
             }

             TWI_Wait();
             switch (TWI_TestAck())
             {
190           case MR_DATA_NOACK:             //Data is received
                *TWI_RArray=TWDR;             //Save Received data
                TWI_Stop();                   //TWI receiving successful
                 delay_ms(5);                 //Delay needed between Stop and Slave
                TWI_Slave();                  //Slave mode
                return 0;
              default:
                TWI_Slave();                  //Slave mode
                return 1;
             }
200    }

       /*---------------------------------------------------------------------
        * Function details:        TWI interrupt service routine
        * Name:                    TWI_S()
        * Usage:                   TWI Slave mode interrupt subroutine
        * Input:                   none
        * Return:                  none
```

```
 * Attention:                none
 * Notes:                    none
-------------------------------------------------------------------*/
ISR (TWI_vect)
{
 //TWI N S is the slave mode flag indicates that  MCU is servering othter master
 //this MCU will hang at TWI_Wait after TWI_Start if entering master mode while
 //it servers existing master on BUS. Therefore MCU can only init "Start" when it
 //is not on the "half way" of slave mode
 TWI N S = 1;
 switch (TWI_TestAck())
 {
  // Slave Receive Mode
  case SR SLA ACK:                          //Start is received
       TWI R P=TWI RArray;                  //set temp input array point
       TWI S Ack();                         //ack the master
       break;
  case SR_DATA_ACK:                         //Data is received
  case SR DATA NOACK:                       //Last Byte of Data is received
  case SR GDATA ACK:                        //General called, data received
  case SR GDATA NOACK:                      //General called, last data received
       *TWI R P=TWDR;                       //Copy received data to input array
       TWI R P++;                           //increase the pointer
       TWI S Ack();                         //Send Ack
       break;
  case SR STOP:                             //STOP received
       TWI N S = 0;                         //done slave mode
                                            //clear slave mode flag, MCU can be master
now

       TWI RNum = (TWI_R_P - TWI_RArray);
       TWI_REVD = 1;

       TWI S Ack();                         //Send Ack
       break;
  case S ARB R:                             //Arb lost, slave R mode
  case SR GSLA ACK:                         //General call address received
  case S ARB G:                             //Arb lost, General call received
       TWI S Ack();                         //Send Ack
       break;

  // Slave Transmit Mode
  case ST_SLA_ACK:                          //Start is received
  case ST DATA ACK:                         //Ready to send data
       TWDR=*TWI_WArray;                    //Send data in TWI_WData
       break;
  case ST DATA NOACK:                       //Data cannot be send
  case ST LAST DATA:                        //Last Data is send
       TWI S NoAck();                       //Send Not Ack
       break;
  case S ARB T:                             //Arb lost, slave T mode
       TWDR=*TWI WArray;                    //Send data in TWI_WData
       TWI S Ack();                         //Send Ack
       break;
  case ST NO INFO:                          //No infomation
       TWI_N_S = 0;                         //clear slave mode flag, MCU can be master
now
       break;
  case ST BUS ERROR:                        //Bus error caused by Start/Stop
       TWI_N_S = 0;                         //clear slave mode flag, MCU can be master
now
       TWI Stop();                          //Stop transmission
        delay ms(1);
       TWI Slave();
       break;

  default:
       TWI_S_Ack();                         //Send
       TWI_N_S = 0;                         //clear slave mode flag, MCU can be master
```

```
now
       break;
 }
}

/*---------------------------------------------------------------------
 * Function details:        TWI Master Receive from Sensor
 * Name:                    TWI MR Sensor()
 * Usage:                   to receive 1 byte data from Address via TWI
 * Input:                   Address
 * Return:                  0 if success or 1 if fail
 * Attention:               none
 * Notes:                   none
---------------------------------------------------------------------*/
unsigned char TWI_MR_Sensor(unsigned char Module, unsigned char Address, unsigned
char Length)
{
      unsigned char *TWI_R_P=NULL;

      //TWI Disenable();
      //TWI_Enable();


      TWI_R_P=TWI_RArray;               //Save the current pointer

      TWI Start();                      //sent Start
      TWI_Wait();                       //wait for reply

      Module=(Module<<1)&WD;            //combine Address and R
      TWI Write8Bit(Module);            //send Address and R
      TWI_Wait();

      TWI Write8Bit(Address);           //send Address and R
      TWI_Wait()

      TWI_Start();

       delay ms(1);
      Module=Module|RD;
      TWI Write8Bit(Module); // read from this I2C address, R/*W Set
      TWI_Wait();

      while (--Length)
      {
        TWI Receive Ack();          //start to receive data with ACK
        TWI Wait();
        *TWI_R_P++ = TWDR; //Read the LSB data
      }

      TWI_Receive();            //start to receive data with ACK
      TWI Wait();
      *TWI_R_P = TWDR; //Read the MSB data

      TWI_Stop();

      //TWI Disenable();
      //TWI_Enable();

      return 0;
}

/*---------------------------------------------------------------------
 * Function details:        TWI clean up
 * Name:                    TWI Clean()
 * Usage:                   to clean up TWI data
 * Input:                   none
 * Return:                  none
 * Attention:               none
 * Notes:                   none
```

```
-----------------------------------------------------------------------*/
void TWI_Clean(void)
{
  memset(TWI_WArray,CLEAR,TWI_BUFFER_SIZE);
  memset(TWI_RArray,CLEAR,TWI_BUFFER_SIZE);
  TWI_N_S=CLEAR;
  TWI_REVD=CLEAR;                       //TWI reveived flag
  TWI_RNum=CLEAR;                       //length of TWI package

}
```

```
/*-------------------------------------------------------------------------
 * Name:                    TWI.h
 * Usage:                   TWI parameters
 * Target MCU:              ATMega168
 * Version:                 V 0.8 by Rick on 2011-05-08
-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------
Section 2:

        Please DO NOT change the below defines
-------------------------------------------------------------------------*/
//TWI BR determines the TWI speed
//FOSC and TWI SPEED are defined in config.h
#define TWI_BR (F_CPU/TWI_SPEED-16)/2

//TWI Status
//General
#define TWCR_MASK       0x0F
#define START           0x08
#define RE START        0x10
//Master Transmit
#define MT SLA ACK      0x18
#define MT SLA NOACK    0x20
#define MT DATA ACK     0x28
#define MT_DATA_NOACK   0x30
//Master Receive
#define MR SLA ACK      0x40
#define MR SLA NOACK    0x48
#define MR DATA ACK     0x50
#define MR DATA NOACK   0x58
//Slave Transmit
#define ST SLA ACK      0xA8
#define ST DATA ACK     0xB8
#define ST DATA NOACK   0xC0
#define ST LAST_DATA    0xC8
//Slave Receive
#define SR SLA ACK      0x60
#define SR DATA ACK     0x80
#define SR DATA NOACK   0x88
#define SR GSLA ACK     0x70
#define SR GDATA ACK    0x90
#define SR GDATA NOACK  0x98
#define SR_STOP         0xA0
//Arbitration lost
#define M ARB           0X38
#define S ARB R         0x68
#define S ARB G         0x78
#define S ARB T         0xB0
//Other status
#define ST_NO_INFO      0xF8
#define ST_BUS_ERROR    0x00

//TWI operation
#define RD 0x01
#define WD 0xFE
#define TWI Slave()         (TWCR=(1<<TWEN)|(1<<TWEA)|(1<<TWIE))
//Slave mode
#define TWI_Start()         (TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN))
//Start I2C
#define TWI_Stop()          (TWCR=(1<<TWINT)|(1<<TWSTO)|(1<<TWEN))
//Stop I2C
#define TWI_Wait()          {char i=0; while(!(TWCR&(1<<TWINT))&& (i < 255)) i++;}
```

```
//Wait until interrupt
#define TWI TestAck()       (TWSR&0xf8)
//check Status Code
#define TWI Receive()       (TWCR=TWCR&(TWCR_MASK|(1<<TWINT)|(1<<TWEN)))
//Receive from TWI
#define TWI Receive_Ack()   (TWCR=TWCR&(TWCR_MASK|(1<<TWINT)|(1<<TWEA)|(1<<TWEN)))
//Receive from TWI
#define TWI Write8Bit(x)    {TWDR=(x);TWCR=TWCR&(TWCR_MASK|(1<<TWINT)|(1<<TWEN));}
//Write to TWI
#define TWI_S_Ack()         (TWCR=TWCR&(TWCR_MASK|(1<<TWEA)|(1<<TWINT)))
//Send ACK
#define TWI S NoAck()       (TWCR=TWCR&(TWCR_MASK|(1<<TWINT)))
//Send NoACK
#define TWI Connect()       (TWCR=TWCR | (1<<TWEA))
#define TWI Disconnect()    (TWCR=TWCR & ~ (1<<TWEA))
#define TWI Disenable()     TWCR &= (~(SET<<TWEN));
#define TWI_Enable()        TWCR |= (SET<<TWEN);

/*-------------------------------------------------------------------------
Section 3:

        Declaration of functions
-------------------------------------------------------------------------*/
unsigned char TWI INIT(unsigned char Address, unsigned char GCall);
unsigned char TWI_MT_RESEND(unsigned char Address, unsigned int Length);
unsigned char TWI MT(unsigned char Address, unsigned int Length);
unsigned char TWI MR(unsigned char Address);
unsigned char TWI_MR_Sensor(unsigned char Module, unsigned char Address, unsigned
char Length);
void TWI Clean(void);
//extern unsigned char TWI REVD;
//extern unsigned char TWI_RNum;
```

```c
/*---------------------------------------------------------------------
 * Name:                            USART.c
 * Usage:                           USART driver
 * Target MCU:              ATMega168
 * Version:                     V 0.2 by Rick on 2011-05-08
---------------------------------------------------------------------*/
#include "usart.h"

unsigned char U Recv Count;
unsigned char USART REVD;           //TWI reveived flag
unsigned char U RArray[U RECV BUFFER SIZE];
unsigned char U WArray[U SEND BUFFER SIZE];
static FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW
);

/*---------------------------------------------------------------------
 * Function details:            USART Initialize
 * Name:                                USART_Init()
 * Usage:                               to init USART of this MCU
 * Input:                               none
 * Return:                          Success: return 0, Fail: return 1
 * Attention:                       none
 * Notes:                               none
---------------------------------------------------------------------*/
unsigned char USART_Init(void)
{
    USART DDR|=((SET<<USART_TX_PIN)&(~(SET<<USART_RX_PIN)));
    U Recv Count = 0;
    /* Set band rate */
    UBRRnH=(F CPU/16UL/USART BAUD-1)/256;
    UBRRnL=(F CPU/16UL/USART BAUD-1)%256;
    /* Enable receiver and transmitter and receive complete interrupt*/
    UCSRnB= (SET<<RXENn)|(SET<<TXENn);
    /* Set frame format: 8 data, 2 stop bits */
    UCSRnC = ((SET<<UCSZn1)|(SET<<UCSZn0))&(~(SET<<UMSELn0));
    USART R S();                        //enable receive complet interrupt
    stdout = stdin = stderr = &uart_str;
    return 0;
}

/*---------------------------------------------------------------------
 * Function details:            USART Transmition
 * Name:                                USART_Transmit()
 * Usage:                               scall to end out a char
 * Input:                               char to be sent
 * Return:                          none
 * Attention:                       none
 * Notes:                               none
---------------------------------------------------------------------*/
void USART_Transmit (unsigned char data) //output char
{
    /* Wait for empty transmit buffer */
    while (!(UCSRnA & (1<<UDREn)));
    /* Put data into buffer, sends the data */
    UDRn = data;
}

/*---------------------------------------------------------------------
 * Function details:            USART Receiver
 * Name:                                USART Receive()
 * Usage:                               call to get the received char
 * Input:                               none
 * Return:                          char that received
 * Attention:                       none
 * Notes:                               none
---------------------------------------------------------------------*/
unsigned char USART_Receive (void) //receive char
{
    while (!(UCSRnA & (1<<RXCn)));
```

```
    return UDRn;
}

/*---------------------------------------------------------------------
 * Function details:            output a string via USART
 * Name:                            U Out s()
 * Usage:                           call to output a sting
 * Input:                           string to be sent
 * Return:                      none
 * Attention:                   none
 * Notes:                           none
---------------------------------------------------------------------*/
void U_Out_s (signed char *s)           // Output string with change line
{
    while (*s)
    {
        USART Transmit(*s++);
        _delay_ms(5);
    }
}

/*---------------------------------------------------------------------
 * Function details:            output a int via USART
 * Name:                            U Out i()
 * Usage:                           call to output a int
 * Input:                           int to be sent
 * Return:                      none
 * Attention:                   none
 * Notes:                           none
---------------------------------------------------------------------*/
void U_Out_i(signed int val )
{
  char buffer[sizeof(int)*8+1];
  U_Out_s( (signed char *) itoa(val, buffer, 10) );
}

/*---------------------------------------------------------------------
 * Function details:            output a package via USART
 * Name:                            USART Send()
 * Usage:                           call to output the package pre-stored
in
 *                                      U WArray
 * Input:                           length of the package to be output
 * Return:                      Success: return 0, Fail: return 1
 * Attention:                   none
 * Notes:                           none
---------------------------------------------------------------------*/
unsigned char USART_Send(unsigned char length)
{
 unsigned char i = CLEAR;
 unsigned char *U_Send_P;

 U Send P = U WArray;
 for (i=CLEAR; i < length; i++)
 {
  USART Transmit(*U_Send_P++);
  _delay_ms(5);
 }

 return NO_ERROR;
}

/*---------------------------------------------------------------------
 * Function details:            USART receive interrupt service routing
 * Name:                            U RX S()
 * Usage:                           the received data are stored in
U RArray
 * Input:                           none
 * Return:                      none
```

```
 * Attention:                              none
 * Notes:                                  none
-------------------------------------------------------------------*/
void U_RX_S(void)
{
    U RArray[U Recv_Count] = UDRn;
    U_Recv_Count++;
    if (U_RECV_LENGTH == U_Recv_Count)
    {
        U Recv Count = 0;
        USART_REVD = 1;
    }
}


/* ---------------------------------------------------------------------------
 * Below are code obtained from Bruce Land, 2011-03
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <joerg@FreeBSD.ORG> wrote this file.  As long as you retain this notice you
 * can do whatever you want with this stuff. If we meet some day, and you think
 * this stuff is worth it, you can buy me a beer in return.        Joerg Wunsch
 * ---------------------------------------------------------------------------
 *
 * Stdio demo, UART implementation
 *
 * $Id: usart.c,v 1.1 2011/05/11 16:24:32 r Exp $
 *
 * Mod for mega644 BRL Jan2009
 */

/*
 * Send character c down the UART Tx, wait until tx holding register
 * is empty.
 */
int uart_putchar(char c, FILE *stream)
{

  if (c == '\a')
    {
      fputs("*ring*\n", stderr);
      return 0;
    }

  if (c == '\n')
    uart putchar('\r', stream);
  loop until bit is set(UCSRnA, UDRE0);
  UDRn = c;

  return 0;
}

/*
 * Receive a character from the UART Rx.
 *
 * This features a simple line-editor that allows to delete and
 * re-edit the characters entered, until either CR or NL is entered.
 * Printable characters entered will be echoed using uart_putchar().
 *
 * Editing characters:
 *
 * . \b (BS) or \177 (DEL) delete the previous character
 * . ^u kills the entire input buffer
 * . ^w deletes the previous word
 * . ^r sends a CR, and then reprints the buffer
 * . \t will be replaced by a single space
 *
 * All other control characters will be ignored.
 *
 * The internal line buffer is RX_BUFSIZE (80) characters long, which
```

```
 * includes the terminating \n (but no terminating \0).  If the buffer
 * is full (i. e., at RX BUFSIZE-1 characters in order to keep space for
 * the trailing \n), any further input attempts will send a \a to
 * uart putchar() (BEL character), although line editing is still
 * allowed.
 *
 * Input errors while talking to the UART will cause an immediate
 * return of -1 (error indication).  Notably, this will be caused by a
 * framing error (e. g. serial line "break" condition), by an input
 * overrun, and by a parity error (if parity was enabled and automatic
 * parity recognition is supported by hardware).
 *
 * Successive calls to uart getchar() will be satisfied from the
 * internal buffer until that buffer is emptied again.
 */
int uart_getchar(FILE *stream)
{
  unsigned char c;
  char *cp, *cp2;
  static char b[U RECV_BUFFER_SIZE];
  static char *rxp;

  if (rxp == 0)
    for (cp = b;;)
      {
      loop until bit is set(UCSRnA, RXC0);
      if (UCSRnA &  BV(FE0))
        return  FDEV EOF;
      if (UCSRnA &  BV(DOR0))
        return _FDEV_ERR;
      c = UDRn;
      /* behaviour similar to Unix stty ICRNL */
      if (c == '\r')
        c = '\n';
      if (c == '\n')
        {
          *cp = c;
          uart putchar(c, stream);
          rxp = b;
          break;
        }
      else if (c == '\t')
        c = ' ';

      if ((c >= (unsigned char)' ' && c <= (unsigned char)'\x7e') ||
          c >= (unsigned char)'\xa0')
        {
          if (cp == b + U RECV BUFFER SIZE - 1)
            uart_putchar('\a', stream);
          else
            {
          *cp++ = c;
          uart_putchar(c, stream);
            }
          continue;
        }

      switch (c)
        {
        case 'c' & 0x1f:
          return -1;

        case '\b':
        case '\x7f':
          if (cp > b)
            {
          uart putchar('\b', stream);
          uart_putchar(' ', stream);
          uart_putchar('\b', stream);
```

```
            cp--;
              }
            break;

          case 'r' & 0x1f:
            uart_putchar('\r', stream);
280         for (cp2 = b; cp2 < cp; cp2++)
              uart_putchar(*cp2, stream);
            break;

          case 'u' & 0x1f:
            while (cp > b)
              {
            uart_putchar('\b', stream);
            uart_putchar(' ', stream);
            uart_putchar('\b', stream);
290         cp--;
              }
            break;

          case 'w' & 0x1f:
            while (cp > b && cp[-1] != ' ')
              {
            uart_putchar('\b', stream);
            uart_putchar(' ', stream);
            uart_putchar('\b', stream);
300         cp--;
              }
            break;
          }
          }

        c = *rxp++;
        if (c == '\n')
          rxp = 0;

310     return c;
      }
```

```
/*-------------------------------------------------------------------------
 * Name:                    TWI.h
 * Target MCU:              ATMega168
 * Version:                 V 0.8 by Rick on 2011-05-08
-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
Section 1:

        The below defines can be changed if necessary
-------------------------------------------------------------------------*/
#define UBRRnL  UBRR0L
#define UBRRnH  UBRR0H
#define UCSRnA  UCSR0A
#define UCSRnB  UCSR0B
#define UCSRnC  UCSR0C
#define RXENn   RXEN0
#define TXENn   TXEN0
#define UMSELn0 UMSEL00
#define UCSZn1  UCSZ01
#define UCSZn0  UCSZ00
#define UDREn   UDRE0
#define RXCn    RXC0
#define UDRn    UDR0
/*-------------------------------------------------------------------------
Section 2:

        Please DO NOT change the below defines
-------------------------------------------------------------------------*/
#define USART_R_S()         UCSR0B|=(1<<RXCIE0)

/*-------------------------------------------------------------------------
Section 3:

        Declaration of functions
-------------------------------------------------------------------------*/
unsigned char USART Init(void);
void USART Transmit(unsigned char data);
unsigned char USART Receive(void);
void U Out s(signed char *s);
void U Out i(signed int val );
unsigned char USART_Send (unsigned char);
void RXC S(void);
extern unsigned char USART_REVD;
int uart putchar(char c, FILE *stream);
int uart_getchar(FILE *stream);
```

```
/*-------------------------------------------------------------------------
 * Name:                    error.h
 * Version:                 V 1.1 by Rick on 2008-04-15
-------------------------------------------------------------------------*/

//ERR array length
#define ERR_BUFFER_SIZE 10

//general ERR defines
#define NO_ERROR        0

//driver ERR defines
#define ERR CHIP INIT   1
#define ERR USART INIT  2
#define ERR USART SEND  3
#define ERR TWI INIT    4
#define ERR_TWI_SEND    5

//CMM ERR defines
#define ERR MAIN        10
#define ERR PC TASK     11
#define ERR SUB TASK    12
#define ERR CMM TASK    13
#define ERR TASK SWITCH 14
#define ERR_RST         15

//UDM ERR defines
#define ERR IR INIT             20
#define ERR EMPTY TWI COMMAND   21
#define ERR INVAILD TWI COMMAND 22
#define ERR WRONG SENDER        23
#define ERR_EMPTY_IR_COMMAND    24

//SCM ERR defines
#define ERR SCM INIT          30
#define ERR EMPTY SCM RCV     31
#define ERR CHECK FD          32
#define ERR CHECK TP          33
#define ERR_SCM_SEND          34

//ACM ERR defines
#define ERR MOTOR INIT            40
#define ERR_EMPTY_MOTOR_COMMAND   41
#define ERR_INVAILD_MOTOR_COMMAND 42

//DBM ERR defines
#define ERR NO SENDER         50
#define ERR_WRONG_COMMAND     51

//DBM exist defines
#define DBM_MAX_FAIL          3
```