“If you have an apple and I have an apple and we exchange these apples then you and I will still each have one apple. But if you have an idea and I have an idea an we exchange these ideas then each of us will have two ideas”

George Bernard Shaw

EE318 Electronic Design Lab Project Report, EE Dept, IIT Bombay, April 2009

SMART GLOVE CONTROLLED TOY CAR

Group B10

06007006 Mehul Tikekar <mehultikekar@iitb.ac.in>
06007011 Akshay Sehgal <akshaysehgal@iitb.ac.in>
06007016 Pradyot Porwal <pradyot_porwal@iitb.ac.in>

Guide –Dipankar

ABSTRACT

The project aimed to design a “smart” glove controlled toy car. The motion of the user’s hand was to be detected using the smart glove and the generated signals were to control the movement of the toy car. In order to detect the motion of the hand, the glove incorporates an accelerometer as the motion sensor. The data from the accelerometer is digitized and sent over an RF channel to the receiver module situated on the car. The module on the car converts the data received into PWM signals to drive the motors. The primary objectives of the project were attained in a period of 7-8 weeks following which some advanced objectives concerning semantics based complex commands were outlined and discussed with our guide. Work on the secondary objectives was satisfactorily completed in the next 4 weeks.

Smart Glove Controlled Toy Car

CONTENTS

INTRODUCTION

In our project we have created a toy car that is controlled by the motion of the user's hand. The motion is detected by an accelerometer embedded in the eponymous "Smart Glove" worn by the user.

Following are the photographs of the glove and the toy car:

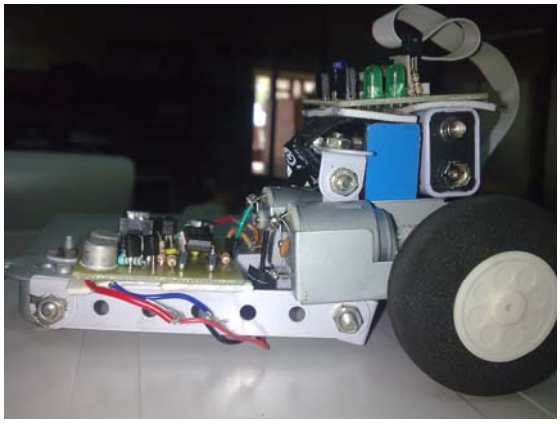


Figure 1 (a) The Smart Glove (b) Toy Car

MOTIVATION/RATIONALE BEHIND THE PROJECT

To be honest we thought that controlling a vehicle just by waving your hands would look totally cool and amazing, like something shown in a science fiction movie!!! However on probing the idea further we found that not only was the final product intrinsically appealing the process of making the project had a lot of new things to offer as topics to learn about. For instance as a result of the project we got familiar with:

a. Working with accelerometers
b. Working with Lex and Yacc
c. Not to mention working with the Atmega8 microcontroller and RF modules

Besides, the final product also has certain potential applications which will be explained later in this text.

HIGH LEVEL DESCRIPTION

The final product can be divided into two main parts:

A. The Transmitter Module:

The transmitter module (size 3 x 3.5 cm^2) is embedded in a glove (leather fingerless) that can be worn by the user. It has a 3 axis accelerometer that detects the motion of the hand. Data from the accelerometer is sent to the microcontroller which sends it to an RF transmitter. The data is then transmitted in packets to the receiver module. The whole module is powered by a 4.2 V cell phone battery.

Additional Features

1. When the battery voltage falls below 3.5 V, the direction LEDs placed on the glove begin to blink indicating low battery power.
2. A push button switch has been provided on the index finger of the glove so that the user can rest his/her hand while operating the car. On pressing the switch the car continues on its course based on the last data received (before switching).

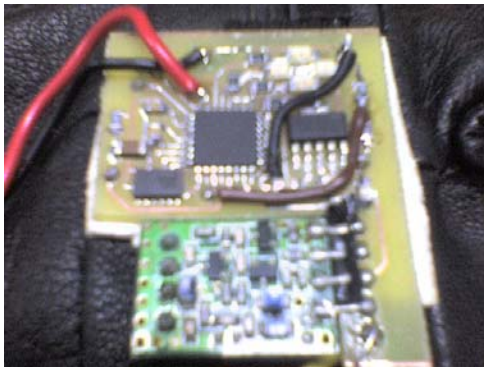


Figure 2 (a) Transmitter Module

B. The Receiver Module:

The receiver module consists of a receiver PCB mounted on the toy car along with one 9 volt Duracell battery for supplying the power to the car motors and the circuit. The circuit receives the RF signal from the transmitter and sends it to the on board microcontroller which converts the data into PWM signals for controlling the car motors. The 70 rpm dc motors are driven by an H-Bridge based motor driver.

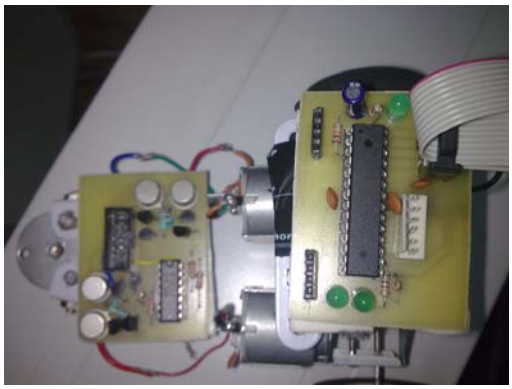
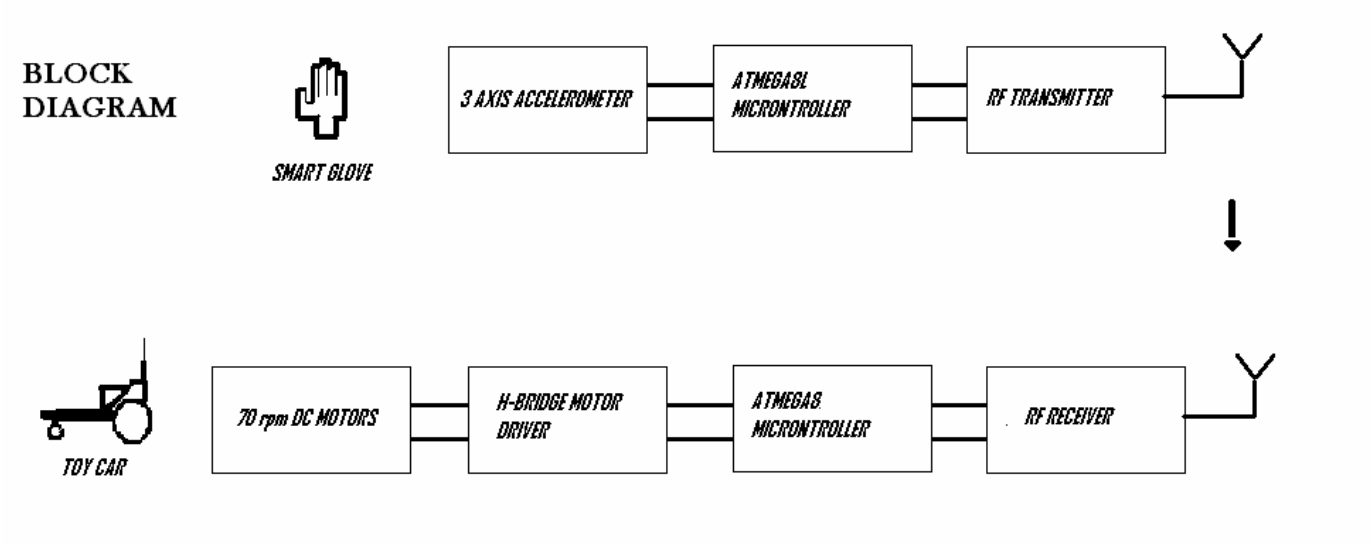


Figure 2(b) Receiver Module on Toy Car



Deliverables of the Project:

The following points had been envisaged as "deliverables" for the present project:

1. Accelerometer based "smart" glove to wirelessly control the toy car
2. Toy car with RF receiver unit and its own portable power supply
3. The user should be able to manipulate the car using the glove over a definite path say figure of 8.

All the three of the above objectives were achieved by the time of the Midsem Evaluation hence the following points were added in the list of deliverables:

1. Use of LEX/YACC for development of semantics based complex commands for toy car control.
2. Battery power low indicator on the glove.
3. Placement of LEDs on the glove to denote the direction of tilt.

HARDWARE DESCRIPTION

We can describe the hardware components under the following sub headings:

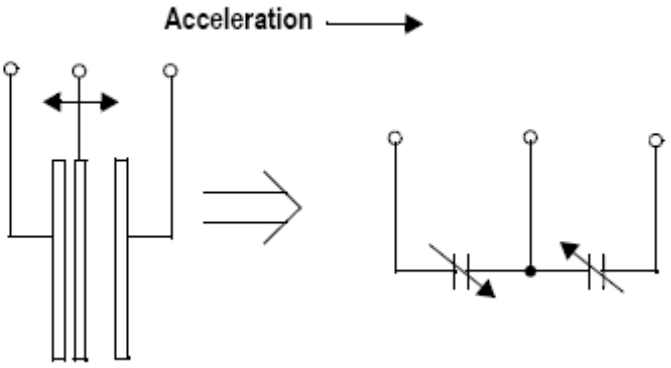
'The Motion Sensor' – The 3-axis low g accelerometer MMA7361L

The MMA7361L is a low power, low profile capacitive micromachined accelerometer that has the following features [1]:

1. 3 mm x 5 mm x 0.1 mm LGA14 package
2. Low Voltage Operation 2.2V-3.6V
3. Sleep Mode
4. Self Test Facility
5. Low Current Consumption - 400µA
6. Selectable Sensitivity 1.5g, 6g
7. Fast Turn-On Time (0.5 ms Enable Response Time)

Principle of Operation

The accelerometer has micromachined capacitive sensing cells or (g-cells) for acceleration detection. The g-cell consists of moving beams fixed at one end placed in between fixed beams with which they form back to back capacitors as shown in the figure:



Simplified Transducer Physical Model

On tilting the acceleration causes the distance between the fixed and the movable beams to change, which results in the increase in the capacitance of one and the decrease in the capacitance of the other back to back capacitor. The difference between the capacitances is used to calculate the acceleration.

When we started out with the project we used the MMA6270 low g 2-axis accelerometer for testing as it was easily available to us however the final product incorporates the MMA7361L. It may be noted that the project requires the data from only two axes, the third axis maybe used for the generation of complex commands using the lexical analyzer (LEX)

'The Brains'- Atmega8 and Atmega8L Microcontrollers

The Atmega8 is a high performance low power AVR 8 bit microcontroller with the following features [2]:

1. Advanced RISC architecture with 130 powerful instructions and 32 x 8 general purpose working registers.
2. 8K bytes of in system self programmable flash memory, 512 bytes EEPROM, 1KB internal SRAM.
3. Programmable serial USART (used for data transfer to the RF module).
4. Power on reset and the presence of 5 different sleep modes.
5. Low operating voltages: 2.7-5.5V (Atmega8L) and 4.5-5.5V(Atmega8)
6. Available in 28 pin PDIP package as well as 32 pin TQFP package

We used the Atmega8 28 pin PDIP package for the receiver module and the Atmega8L 32 pin TQFP package for the transmitter module. The TQFP package being substantially smaller than the PDIP one helped in the size optimization of the transmitter PCB.

The accelerometer runs on a voltage range of 2.7 to 3.6 V, hence to avoid stepping down the voltage within the circuit we used the low voltage Atmega8L so that both components can be directly supplied by a single voltage source.

The internal clock of 1MHz was used for operation and the inbuilt 8 bit ADC was used to convert the analog data from the accelerometer.

'The Radio Set' – ASK based RF Module

We used a crystal tuned PLL based ASK (Amplitude Shift Keying) RF module with the following features [3]:

1. Two operating frequencies i.e. 315/434 MHz. We utilized the 434 MHz frequency.

2. Receiver specifications: 4.75 – 5.25V operating voltage, 1k-10k bps data rate, 2.4 – 3mA current consumption.

3. Transmitter specifications: 3V operating voltage, 1k-10k bps data rate, 9 – 19mA current consumption, 1-6mW power consumption.

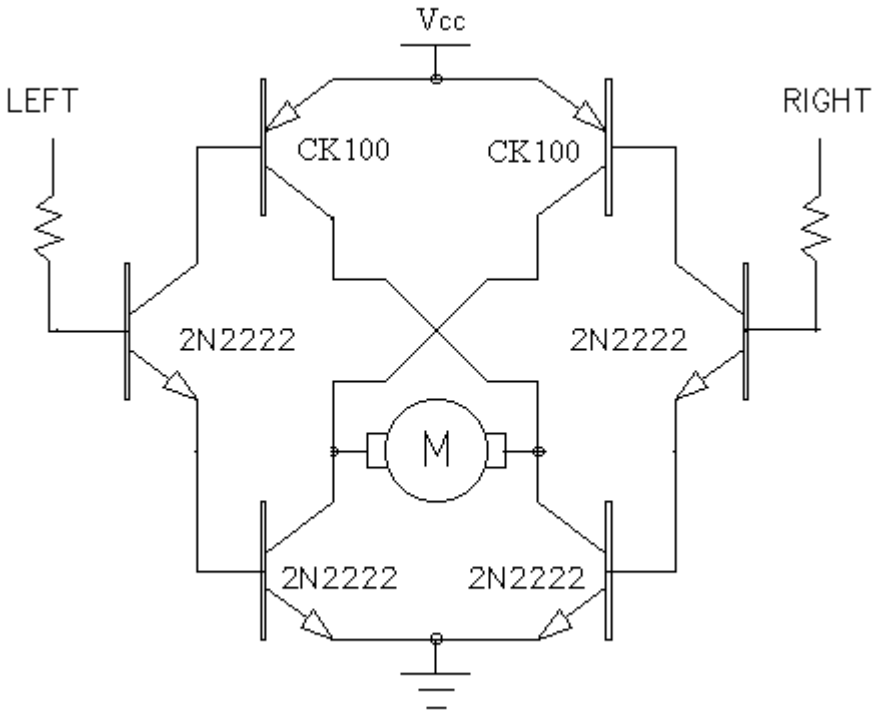
We used a baud rate of 1.2kbps for data transfer. The datasheet recommended the use of an antenna of length 17.2cm for 434MHz frequency of operation, however without the use of the antenna we tested our device and found that data could be successfully transmitted and received over a distance of 15m (WEL3 table 4 to WEL5 table10).

However later in the project the range dropped off drastically and on using a 20cm antenna on the transmitter (and none on the receiver) we achieved a range of 8m.

'The Motor Driver' – Transistor based H-Bridge Circuit

For driving the 70 rpm dc motors we initially thought of using the L293D motor driver chip, however the internal drop in the transistors amounted to as much as 2.6 V which reduced the voltage drop across the motor. To counter this 2.6V drop we could either jack up the voltage supply or go for a different motor driver assembly all together.

So we decided to do the latter and went on to construct a transistor based H Bridge motor driver circuit. The basic schematic of a double input "Tilden" 6 transistor H Bridge is shown below which can help outline its working principle [4]:



TILDEN H-BRIDGE

Working

Here the M is the dc motor, the npn transistors are 2N2222s and the pnp transistors are CK100s. When input1 is high the transistors on the upper right and lower left switch on and drive the motor in one direction say positive and when input2 is high then the transistor set on the upper left and the one on the lower right switches on driving the motor in the opposite direction.

Whether the PWM applied by the microcontroller is to drive the input1 or input2 is decided by connecting an AND gate before the H Bridge which receives as its input the PWM signal and the bit to decide whether reversal of direction is required or not.

The Toy Car

The chassis of the toy car was built using components from a “Mechano” Set.
The car is a three-wheeler with a castor wheel upfront and rear wheels driven by 70 rpm motors. The Receiver module and the powering battery are mounted on the car chassis.
The car without any of electronic circuitry and power supply is shown in the following figure:

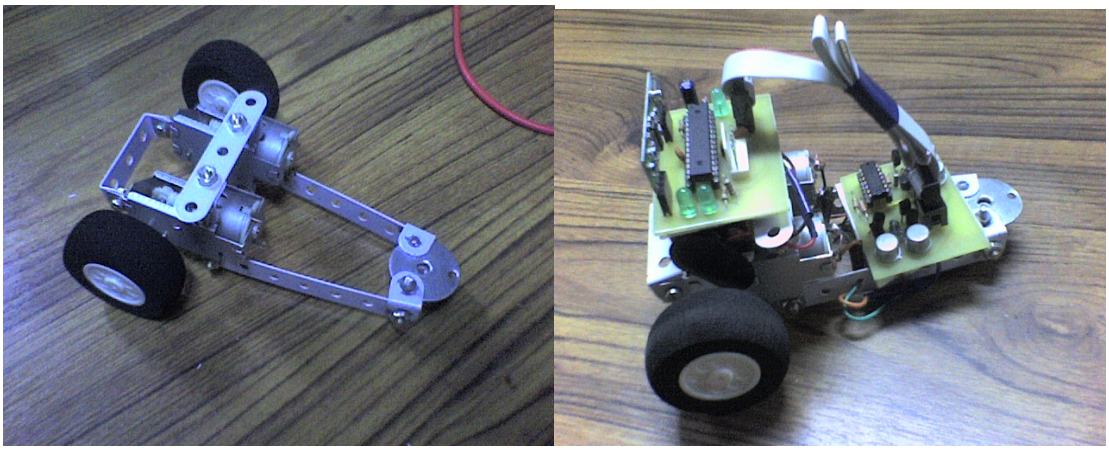


Figure 3 (a) Before… (b) and After…. ☺

And we thus created…

The components listed above combined with a number of other smaller circuit elements form the following circuits for the transmitter and receiver: (Schematics)

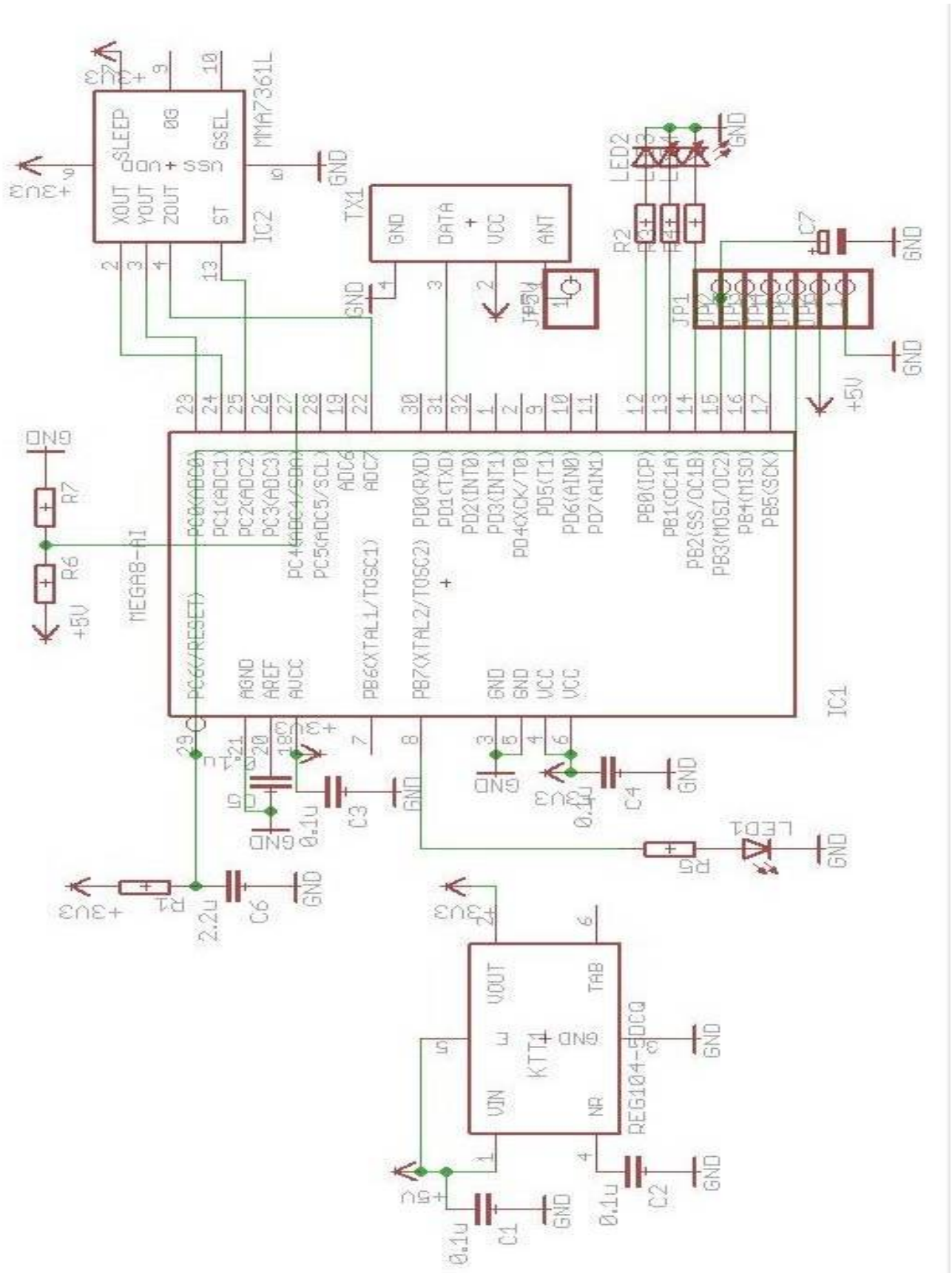


Figure 4 (a) Transmitter Circuit Schematic

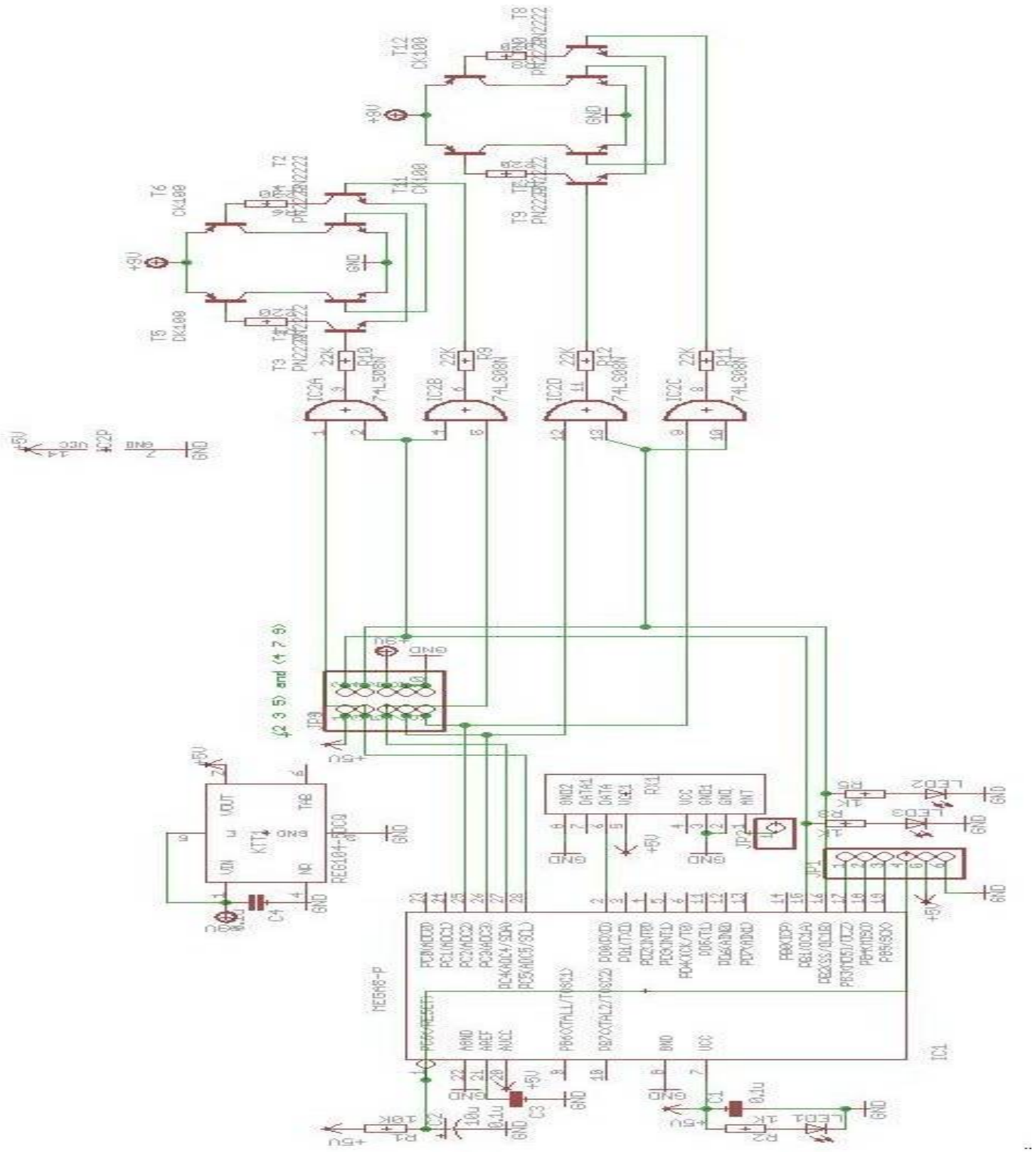


Figure 4 (b) Receiver Circuit Schematic

It may be noted that in the transmitter module it is required to step down the voltage to 3.3 V which is accomplished by the voltage regulator REG104GA5. The regulator was used in its DCQ package which allowed us to have a small transmitter PCB.

SOFTWARE DESCRIPTION

The software description may be covered under two headings i.e.
1. Transmitter and Receiver codes for the glove and car respectively.
2. Transmitter and Receiver codes for the Lex/Yacc operation.

We will cover the description of part 2 in the subsequent section which covers the entire Lex/Yacc part of the project.

Transmitter and Receiver codes for the glove and the car

Overview

The data from the accelerometer is converted to the digital form with the help of the microcontroller ADC. The data is then sent in packets of 4 bytes each consisting of 2 Start Bytes, X o/p of accelerometer, Y o/p of accelerometer at a baud rate of 1.2kbps using the USART. When the 4 byte packets are not being sent at that time the transmitter sends a "Dummy" byte continuously. The Dummy byte suppresses arbitrary noise signals that may be picked up by the receiver and hence improves noise rejection. The provision of two start bytes is also aimed at noise rejection and during testing it was found to be substantially more effective than a single start byte.

Transmitter Code

The code relies on two different interrupts for its operation, one dependent on the USART UDRE vector and the other on a TIMER vector.

Whenever a byte is sent through the USART an internal flag becomes 0 (Data Register Empty) triggering an interrupt which calls an ISR that fills in a new byte into the data register.

The second interrupt is triggered (by the TIMER vector) in regular intervals of time and it calls an ISR that takes in new values from the ADC and calls a function to fill in these new values into the "data byte" variables.

The bytes being sent may be classified as:
1. *Start Byte1 (Fixed)*
2. *Start Byte2 (Fixed)*
3. *Data Byte1 (X o/p) (**Updated at regular intervals**)*
4. *Data Byte2 (Y o/p) (**Updated at regular intervals**)*
5. *Dummy (Fixed)*

(1), (2), (3), (4) form a data packet

The transmitter code also lights up 4 different LEDs (forward, reverse, right and left) dependent on the value of X and Y received from the ADC. The LEDs are indicative of the direction in which the glove is being maneuvered.

The LEDs also flicker when the battery voltage for the transmitter module falls below a certain limit. (This fall is detected by converting the battery voltage value using the ADC and comparing with a reference)

Receiver Code

The receiver again takes in data via the USART in the sequence sent by the transmitter. The data is filled into variables “x” and “y” by triggering an ISR by the USART RXC vector which calls an interrupt every time a byte is received.

On receiving the dummy byte it leaves the value of the “y” and “x” data variables unchanged. (Note that in the code “y” and “x” are related to the forward/reverse and right/left movements respectively)

If a byte other than a start byte or a dummy is received then the “x” and “y” variables are assigned values such that the PWM generated requires the car to stop. This is done to ensure that when the transmitter is off then random noise does not trigger the motion of the car. (Note that the probability of receiving a random signal, when the transmitter is on, is very small owing to the continuous transmission of Dummy Bytes in between data packets)

The data is translated into PWM via a ***look up table*** which results in only discrete values of the PWM being allowed,

Value of “y”	Value of “x”	PWM lmotor	PWM rmotor	Forward/Reverse (F) (R)
y>(midval + dev30)	x>(midval + dev10)	255	210	LM – F RM – F
y>(midval + dev30)	x<(midval – dev10)	210	255	LM – F RM – F
y>(midval + dev30)	-dev10<x-midval <dev10	245	245	LM – F RM – F
y<(midval – dev10)	x > (midval + dev10)	255	210	LM – R RM – R
y<(midval – dev10)	x < (midval - dev10)	210	255	LM – R RM – R
y<(midval – dev10)	-dev10<x-midval <dev10	245	245	LM – R RM – R
-dev10<y-midval <dev30	x > (midval + dev30)	220	220	LM – R RM – F
-dev10<y-midval <dev30	x < (midval - dev30)	220	220	LM – F RM – R
-dev10<y-midval <dev30	-dev30<x-midval <dev30	0	0	LM – Stop! RM – Stop!

Table 1 – Look-Up Table for the Receiver

In the table given above, “midval” stands for the value of “x” or “y” when the glove is in the “no tilt” position. The values “dev10”, “dev30” are measures of deviation from the “midval”.
Some salient features of the look-up scheme are:

1. When the car is moving forward or backward a smaller deviation/tilt is required to turn the car (dev10 as compared to dev30 in the stationary turn case)

2. The tilt required to reverse the car (i.e. dev10) is smaller than when we have to move the car forward (i.e. dev30).

Both the above measures are aimed at enhancing the ease of operating the toy car.

Note that 255 is the maximum value that can be filled into the PWM register and on testing it was found that values lower than 200 are ineffectual in running the motors at all.

The last column of the look-up table indicates that the receiver code generates forward and reverse signals for both the motors. These appear as bit values on Port C of the microcontroller and along with the PWM are fed into an AND gate that controls the H-Bridge motor driver circuits.

EXPERIMENTS WITH LEX/YACC

On completing the basic objectives of the project we proceeded to more advanced operations that dealt with recognition of complex hand movements and their interpretation as commands to our toy car.

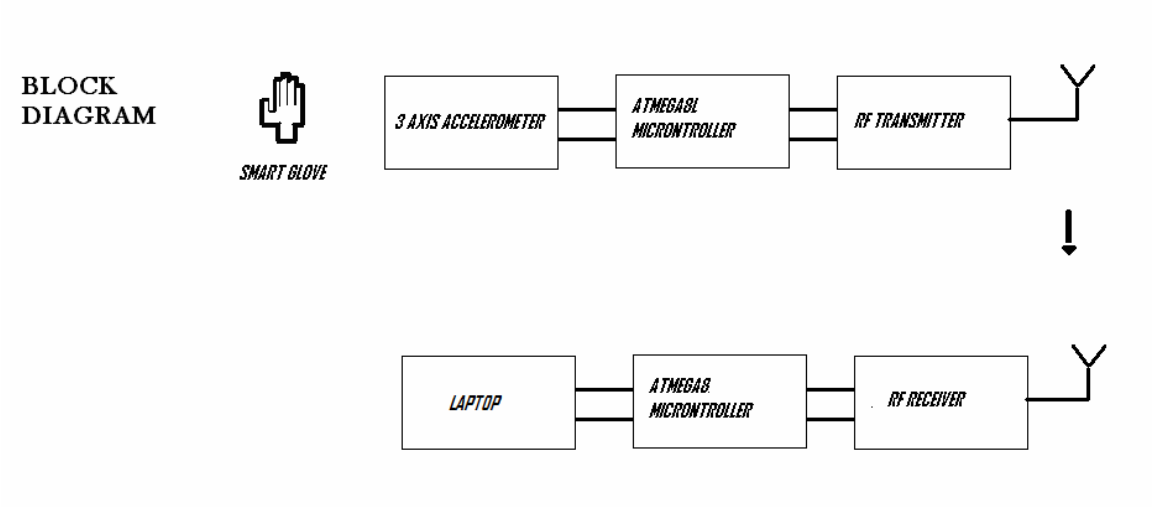
The accomplishment of this goal dramatically expands the scope and utility of the project as it allows the final product to be more than just a vehicle whose motion imitates the motion of the users hand but also allows the performance of various complex maneuvers.

Analysis of hand movements was carried out using LEX(Lexical Analyzer) and YACC(Yet Another Compiler Compiler) whose mode of operation is briefly explained as follows[5].

Working

LEX defines various "tokens" from the data that can be received as input. These tokens form the **syntax** that would help build complex instructions.
YACC defines the "grammar" i.e. how tokens interact with each other and also the "new" instructions based on the stream of tokens generated from an input. The LEX file (.l) and the YACC file (.y) on compiling together generate a C code file and a Finite State Machine that ultimately form the basis of operation.

LEX and YACC in Our Project



a. Hardware

A simple circuit comprising of a RF Receiver Module (ASK, 434MHz), an Atmega8 microcontroller and Serial to USB Connector (connected to Mehul's laptop) was used as the receiving unit for data transmitted from the glove.

It may be noted that the use of MAX232 IC was not required since the Serial to USB Connector being used operated at 5V and stepped the signals to 13V on its own.

b. Software

Tera Term is used to display the data being sent by the microcontroller on the laptop. The data sent is interpreted as the ascii code and the corresponding letters are displayed on the laptop. The code for the transmitter on the glove remains unchanged. The code for the microcontroller on the receiver circuit incorporates both the transmitter and receiver parts.

The receiver part of the code uses an interrupt triggered by the USART RXC vector. On receiving the 4 byte data packet the ISR calls the "Send Packet" function and sends the ASCII values obtained by converting the "x" and "y" data bytes i.e.

X' = ('a' + x/16)
Y' = ('a' + y/16)

The LEX file converts the data received in the form of alphabets ('a' – 'p') into tokens L1 ('a' – 'f'), L2 ('g'- 'h') and L3 ('i' – 'p') (for the X' data) and similarly R1, R2 and R3 for the Y' data.

The YACC file defines "Forward", "Reverse", "Right" and "Left" based on the tokens received. The precise definitions of these "commands" were obtained by experimentation and the relevant YACC file is attached at the end of the project report.

The LEX/YACC program operates on an input file which is continuously modified in real time as the data is received from the microcontroller.

APPROACH TOWARDS THE PROJECT/WORKING STRATEGY

1. As a group we started the project by making a tentative timeline for the completion of the various project tasks and setting up deadlines so as to finish the project in roughly 2/3rd of the total time allotted to us in weeks. As a result of this strategy we were able to complete the primary objectives of the project by the time of the Midsem Evaluation.
2. All the circuits were first made on breadboard and were translated to PCBs in the latter part of the project.

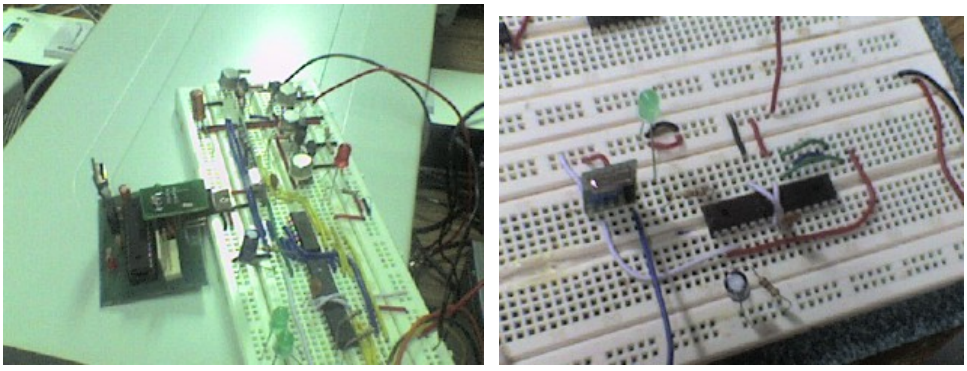


Figure 5 (a) Receiver Breadboard (b) Transmitter breadboard (without accel.)

3. After every lab session the data recorded in experiments and the day's progress was mailed to all the team members, our TA and our guide.
4. The secondary objectives were also achieved in an orderly fashion by following the same strategy as outlined above in a rough period of 4 weeks.
5. At the date of report submission the project stands successfully completed and fulfills all the requirements demanded of it.

FUTURE WORK AND APPLICATIONS

Future Work

The following features may be added to improve the project:

1. The commands generated by LEX/YACC may be transmitted to the toy car and be interpreted and executed as special instructions.
2. A self-modifying code may also be conceived for the toy car that allows the adjustment of PWM levels with falling battery voltage
3. A motor driver IC with an internal voltage drop less than 2.6V may be used instead of the H Bridge Circuits that are bulky and occupy space.
4. An RF module with greater range may be used/ a better antenna can be used for the glove to improve range.
5. A sensor may be built into the receiver to detect jerks in the car motion and be transmitted back to the glove with the help of another RF module. The signals can then be used to stimulate a vibrator in the glove. The whole operation would give the user a realistic feel along with a better idea of the terrain in which the car is being operated.

Applications

1. In its current form our project can be packaged and marketed as a high tech toy car for all age groups.
2. The car can be modified into a vehicle that performs specific data collection operations in inaccessible or hostile environs. A transmitter connected to the toy car can be used to send back the collected data to base. The glove based remote would provide more intuitive control for such a device.

It may be noted that the accelerometer based technology is already in use a number of recently released simulation games and other high tech devices such as Nintendo Wii and ipods.

APPENDIX A

COST OF PROJECT

Items	Quantity	Cost
70 rpm DC Motors	2	250 (INR)
Wheels	2	90 (INR)
Caster	1	20 (INR)
Atmega8 PDIP	2	120 (INR)
Atmega8 TQFP	1	75 (INR)
3 axis Accelerometer MMA7361	1	$ 2.20 (~ 106 INR)
Voltage Regulator REG104GA5	1	$ 2.60 (~ 127 INR)
Voltage Regulator REG104 - EDCQ	1	$ 2.60 (~ 127 INR)
RF Transceiver ASK Module	1	330 (INR)
Duracell 9V Battery	1	140 (INR)
Spare Battery	1	20 (INR)
Double Sided Tape	2	50 (INR)
Lab Tools	--	110 (INR)
Total		1565 (INR)

APPENDIX B

REFERENCES

[1] MMA7361L datasheet, Freescale Semiconductor, Rev 0, 04/2008

[2] Atmega8 Atmega8L datasheet, Atmel®

[3] Instruction Manual, RF module 315/434MHz (ASK) transceiver pair, vendor VegaKit

[4] David Cook, http://www.robotroom.com/HBridge.html, last accessed on 21st April 2009

[5] Ashish Bansal, Sapient Corporation, http://www.ibm.com/developerworks/library/l-lex.html, last accessed on 25th March 2009

APPENDIX C

Source code for transmitter on glove:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/sleep.h>

#define is_high(x,y) (x&_BV(y)) == _BV(y)
#define is_low(x,y)  (x&_BV(y)) == 0

#ifndef F_CPU
//define cpu clock speed if not defined
#define F_CPU 1000000
#endif
//set desired baud rate
#define BAUDRATE 1200
//calculate UBRR value
#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)
//define receive parameters
#define DUMMY '*'
#define START1 '\r'
#define START2 '\n'
#define LEDON 0x11//switch led on command
#define LEDOFF 0x22//switch led off command

#define midval 120
#define dev30  33
#define dev20  22
#define dev10  11

#define PACKETSIZE 4
volatile unsigned char flag, lmotor, rmotor, enable;
volatile unsigned char packet[PACKETSIZE+1];
void USART_Init(void)
{
    UBRRL=(uint8_t)UBRRVAL;
    UBRRH=(UBRRVAL>>8);
    UCSRB=_BV(TXEN)|_BV(UDRIE);

    flag = PACKETSIZE;
    enable = 1;
    packet[0] = START1;
    packet[1] = START2;
    packet[PACKETSIZE] = DUMMY;
}

void Send_Packet(unsigned char data1, unsigned char data2)
{
```

```
    unsigned char temp;
    temp = PORTB;
    PORTB = 0x87;

    while(flag != PACKETSIZE){};
    packet[2] = data1;
    packet[3] = data2;
    flag = 0;

    PORTB = temp;
}

ISR(USART_UDRE_vect)
{
    UDR = packet[flag];
    flag++;
    if(flag>PACKETSIZE) flag = PACKETSIZE;
}

ISR(TIMER1_COMPA_vect)
{
    unsigned char p;
  ADMUX  = _BV(ADLAR)|_BV(REFS0)|_BV(MUX2);
  ADCSRA |= _BV(ADSC);
  while(ADCSRA & _BV(ADSC)) {};
  p = ADCH;
   while (p < 135)
   {
            PORTB = 0x87;
            _delay_ms(100);
            PORTB = 0x00;
            _delay_ms(400);
            ADCSRA |= _BV(ADSC);
            while(ADCSRA & _BV(ADSC)) {};
            p = ADCH;
   }

    if(enable == 1)
    {
            unsigned char x, y;
            ADMUX  = _BV(ADLAR)|_BV(REFS0);
            ADCSRA |= _BV(ADSC);
            while(ADCSRA & _BV(ADSC)) {};
            y = ADCH;

            ADMUX  = _BV(ADLAR)|_BV(REFS0)|_BV(MUX0);
            ADCSRA |= _BV(ADSC);
            while(ADCSRA & _BV(ADSC)) {};
            x = ADCH;
```

```
            if(x < midval - dev10)
            {
                    PORTB |= _BV(PB7);
                    PORTB &= ~(_BV(PB1));
            }
            else if(x > midval + dev30)
            {
                    PORTB |= _BV(PB1);
                    PORTB &= ~(_BV(PB7));
            }
            else PORTB &= ~(_BV(PB1)|_BV(PB7));

            if(y < midval - dev30)
            {
                    PORTB |= _BV(PB2);
                    PORTB &= ~(_BV(PB0));
            }
            else if(y > midval + dev30)
            {
                    PORTB |= _BV(PB0);
                    PORTB &= ~(_BV(PB2));
            }
            else PORTB &= ~(_BV(PB0)|_BV(PB2));

                    Send_Packet(x,y);

    }

}

void Main_Init(void)
{
    DDRB = 0x87;
    PORTB |= _BV(PB3);
    DDRC |= _BV(PC2);
    PORTC &= ~(_BV(PC2));

    ADCSRA = _BV(ADEN) | _BV(ADPS1) | _BV(ADPS0);// Enable ADC with
    prescaler = 8
    ADMUX  = _BV(ADLAR)|_BV(REFS0); //left adjust ADC,voltage reference =
    2.56V

    TCCR1A = 0;
    TCCR1B = _BV(WGM12)|_BV(CS12);          //CTC mode, time delay set to
    100ms
    OCR1A  = 390;
    TIMSK = _BV(OCIE1A);
    flag = 0;
    enable = 1;
    sei();
```

```
}

int main(void)
{
    USART_Init();
    Main_Init();

    while(1)
    {
            while(is_high(PINB,PB3)) {};
            _delay_ms(5);                   // ensure that transients dont affect
            if(is_low(PINB,PB3))                                    // check again if low,
    then toggle enable
            {
                    enable ^= 0x01;
                    _delay_ms(1000);                        // debounce
            }
    }
    return 0;
}
```

Source Code for Receiver Module (Toy Car) :

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#ifndef F_CPU
//define cpu clock speed if not defined
#define F_CPU 1000000
#endif
//set desired baud rate
#define BAUDRATE 1200
//calculate UBRR value
#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)
//define receive parameters
#define DUMMY '*'
#define START1 '\r'
#define START2 '\n'

#define LEDON 0x11//LED on command
#define LEDOFF 0x22//LED off command

#define midval 120
#define dev30  33
#define dev20  22
#define dev10  11
```

```
#define l_ocr OCR1BL
#define r_ocr OCR1AL

volatile unsigned char flag,x,y;
void USART_Init(void)
{
	UBRRL=(uint8_t)UBRRVAL;			//low byte
	UBRRH=(UBRRVAL>>8);		//high byte
	//Default data frame format: asynchronous mode,no parity, 1 stop bit, 8 bit size
	//Enable Transmitter and Receiver and Interrupt on receive complete
	UCSRB=(1<<RXEN)|(1<<RXCIE);
}

ISR(USART_RXC_vect)
{
	unsigned char data;
	data = UDR;
	if(flag == 3)			//receiving second byte
	{
		x = data;
		flag = 0;
	}
	else if(flag == 2)		//receiving first byte
	{
		y = data;
		flag =3;
	}
	else if(flag == 1)
	{
		if(data == START2) flag = 2;
		else {x = midval; y = midval;flag = 0;}
	}
	else
	{
		if(data == START1) flag = 1;	//receiving start byte
		else if(data == DUMMY) flag = 0;
		else {x= midval; y = midval;}
	}
}

void Main_Init(void)
{
	DDRC  = _BV(PC2)|_BV(PC3)|_BV(PC4)|_BV(PC5);

	TCCR1A = _BV(COM1A1)|_BV(COM1B1)|_BV(WGM10);
	TCCR1B = _BV(WGM12)|_BV(CS10);		//8-bit Fast PWM, Prescaler = 1,
Non-inverting mode
	DDRB   = _BV(PB1)|_BV(PB2);
```

```
        sei();
        flag = 0;
}

#define Lf PC5
#define Lr PC4
#define Rf PC3
#define Rr PC2
#define LPWM PB1
#define RPWM PB2

void lmotor_forward(void)
{
        PORTC &= ~(_BV(Lr));
        PORTC |= _BV(Lf);
        }
void lmotor_reverse(void)
{
        PORTC &= ~(_BV(Lf));
        PORTC |= _BV(Lr);
}
void lmotor_stop(void)
{
   PORTC &= ~(_BV(Lr)|_BV(Lf));
}

void rmotor_forward(void)
{
        PORTC &= ~(_BV(Rr));
        PORTC |= _BV(Rf);
}
void rmotor_reverse(void)
{
        PORTC &= ~(_BV(Rf));
        PORTC |= _BV(Rr);
}
void rmotor_stop(void)
{
   PORTC &= ~(_BV(Rf)|_BV(Rr));
}

signed char lspeed, rspeed;

int main(void)
{
        Main_Init();
        USART_Init();

        //unsigned int duty_left, duty_right;
        while(1)
```

```
    {
        if(y > (midval + dev30))
        {
            lmotor_forward();
            rmotor_forward();
          if(x > (midval + dev10))
            {
                l_ocr = 255;
                r_ocr = 210;
            }
            else if(x < (midval - dev10))
            {
                l_ocr = 210;
                r_ocr = 255;
            }
            else
            {
                l_ocr = 245;
                r_ocr = 245;
            }
        }
        else if(y < (midval - dev10))
        {
            lmotor_reverse();
            rmotor_reverse();
          if(x > (midval + dev10))
            {
                l_ocr = 255;
                r_ocr = 210;
            }
            else if(x < (midval - dev10))
            {
                l_ocr = 210;
                r_ocr = 255;
            }
            else
            {
                l_ocr = 245;
                r_ocr = 245;
            }
        }
else
        {
            if(x > (midval + dev30))
            {
                l_ocr = 220;
                r_ocr = 220;
                rmotor_forward();
                lmotor_reverse();
            }
```

```
                        else if(x < (midval - dev30))
                        {
                                l_ocr = 220;
                                r_ocr = 220;
                                rmotor_reverse();
                                lmotor_forward();
                        }
                        else
                        {
                                lmotor_stop();
                                rmotor_stop();
                                l_ocr = 0;
                                r_ocr = 0;
                        }
                }

        }
        return 0;
}
```

Source Code for Receiver Module (LEX/YACC expmt):

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/sleep.h>

#define is_high(x,y) (x&_BV(y)) == _BV(y)
#define is_low(x,y)  (x&_BV(y)) == 0

#define BAUDRATE 1200
#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)
#define PACKETSIZE 5
#define DUMMY '*'
#define START1 '\r'
#define START2 '\n'

#define midval 130

volatile unsigned char flag, enable,x,y;
volatile unsigned char packet[PACKETSIZE];

void USART_Init(void)
{
        UBRRL = (uint8_t)UBRRVAL;
        UBRRH = (UBRRVAL>>8);
        UCSRB = _BV(TXEN)|_BV(RXEN)|_BV(RXCIE);
        flag = 0;
```

```
	enable = 1;
	packet[0] = START1;
	packet[1] = START2;
	packet[4] = DUMMY;
}

void Send_Packet(unsigned char data1, unsigned char data2)
{
	PORTC ^= _BV(PC5);
	packet[2] = data1;
	packet[3] = data2;
	unsigned char i;
	for(i=0;i<PACKETSIZE;i++)
	{
		while ( !( UCSRA & (1<<UDRE)) );
		UDR = packet[i];
	}
}

ISR(USART_RXC_vect)
{
	unsigned char data;
	data = UDR;
	if(flag == 3)			//receiving second byte
	{
		x = data;
		flag = 0;
		Send_Packet('a'+(x/16),'a'+(y/16));
	}
	else if(flag == 2)		//receiving first byte
	{
		y = data;
		flag =3;
	}
	else if(flag == 1)
	{
		if(data == START2) flag = 2;
		else {x = midval; y = midval; flag = 0;}
	}
	else
	{
		if(data == START1) flag = 1;	//receiving start byte
		else if(data == DUMMY) flag = 0;
		else {x = midval; y = midval;}
	}
}

void Main_Init(void)
{
	DDRC   = _BV(PC5);
```

```
        sei();
}

int main(void)
{
        USART_Init();
        Main_Init();

        while(1)
        {
        }
        return 0;
}
```

Source Code for LEX file:

```
%{
   #include <stdlib.h>
   void yyerror(char *);
        #include"main.h"
%}
%s DUMMY
%s START
%s DATA

%%

"*" {BEGIN DUMMY;}
<DUMMY>\n  {BEGIN START;}
<START>[a-f] {BEGIN DATA; return(LT1);}
<START>[g-j] {BEGIN DATA; return(LT2);}
<START>[k-p] {BEGIN DATA; return(LT3);}


<DATA>[a-f] {BEGIN DUMMY; return(RT1);}
<DATA>[g-j] {BEGIN DUMMY; return(RT2);}
<DATA>[k-p] {BEGIN DUMMY; return(RT3);}
\n ;
.   ;
<<EOF>> ;
%%
```

Source Code for YACC file

```
%{
   #include <stdio.h>
   int yylex(void);
   void yyerror(char *);
%}

%token LT1 LT2 LT3 RT1 RT2 RT3

%%

rhyme : string1 {printf ("Left\n"); YYABORT;}
          |       string2 {printf ("Right\n"); YYABORT;}
          | string3 {printf ("Forward\n"); YYABORT;}
          | string4 {printf ("Reverse\n"); YYABORT;}
          | error {;}
          | error string1 {printf ("Left\n"); YYABORT;}
          | error string2 {printf ("Right\n"); YYABORT;}
          | error string3 {printf ("Forward\n"); YYABORT;}
          | error string4 {printf ("Reverse\n"); YYABORT;}
          ;

string1 :  LT1 rt LT2 rt LT3
                  | LT1 rt LT3
                  | LT1 rt LT2 rt LT2 rt LT3
                  ;

string2 :  LT3 rt LT2 rt LT1
                  | LT3 rt LT1
                  | LT3 rt LT2 rt LT2 rt LT1
                  ;

string3 :  RT1 lt RT2 lt RT3
                  | RT1 lt RT3
                  | RT1 lt RT2 lt RT2 lt RT3
                  ;

string4 :  RT3 lt RT2 lt RT1
                  | RT3 lt RT1
                  | RT3 lt RT2 lt RT2 lt RT1
                  ;

lt     :  LT1
                  | LT2
                  | LT3
                  ;

rt     :  RT1
                  | RT2
                  | RT3
                  ;
```

```
%%
extern FILE * yyin;
void yyerror() {}

int yywrap(void) {
   return 1;
}

int main(void) {
        //yyin = fopen("input.txt", "r");
        while(1){ yyparse();}
        //fclose(yyin);
        return 0;
}
```
