EEL 5666

Spring 2005

Instructed by Dr. Arroyo and Dr. Schwartz

With Assistance from William Dubel and Steven Pickels

Initial-B

Ian Gerstel

TABLE OF CONTENTS

ABSTRACT

Initial-B was designed to be a high-speed line-following robot, although an occassionally moving robot would be more appropriate. It uses six infrared transmitters and receivers to follow a line and adjusts itself to remain centered on the line through gradual curves and sharp turns, provided it has mobility. It also has bump sensors and an ultrasonic sensor for obstacle detection and avoidance. Optical interrupters are used to measure the velocity of Initial-B and to check for traction on the wheels. Forward velocity is generated from a high-RPM motor in the rear of the robot, connected to a worm gear on the rear axle. Steering is handled by a servo connected to the front two wheels.

EXECUTIVE SUMMARY

Initial-B is an autonomous robot with a drive-train designed to simulate a rear-wheel drive car. The programming is done in C and and written to an Atmel Atmega128 microcontroller. The purpose of this robot is to follow a high-contrast line at high speeds while avoiding obstacles in its path.

Driving power comes from the 595 RPM 12V (currently 357 RPM at 7.2V) Hsiang Neng motor, and steering is controller by a servo connected to the two front wheels. Six Sharp GP2D120 infrared transmitter/receivers are mounted underneath Initial B in a U-formation. The data obtained from these sensors detects the line and uses a lookup table to determine which of 8 possible steering actions will be used to keep the line centered underneath the robot for line-following.

The Devantech SRF04 ultrasonic range finder is used in conjunction with the

optical interrupter sensors and bump sensors for obstacle avoidance. The optical interrupters on the front and rear axles use external interrupts to accurately calculate the velocity of Initial B, as well as to check for loss of traction (when the front and rear axle RPMs are not within 20% of each other). From that data the stopping distance of the robot is calculated so the robot can stop at an appropriate distance if something is in its path. If, however, something moves abruptly in front of it, either by activating a bump sensor on the front or by getting so close that the robot cannot stop in time, it will turn quickly off the line and then turn back an instant later, trying to find the line again.

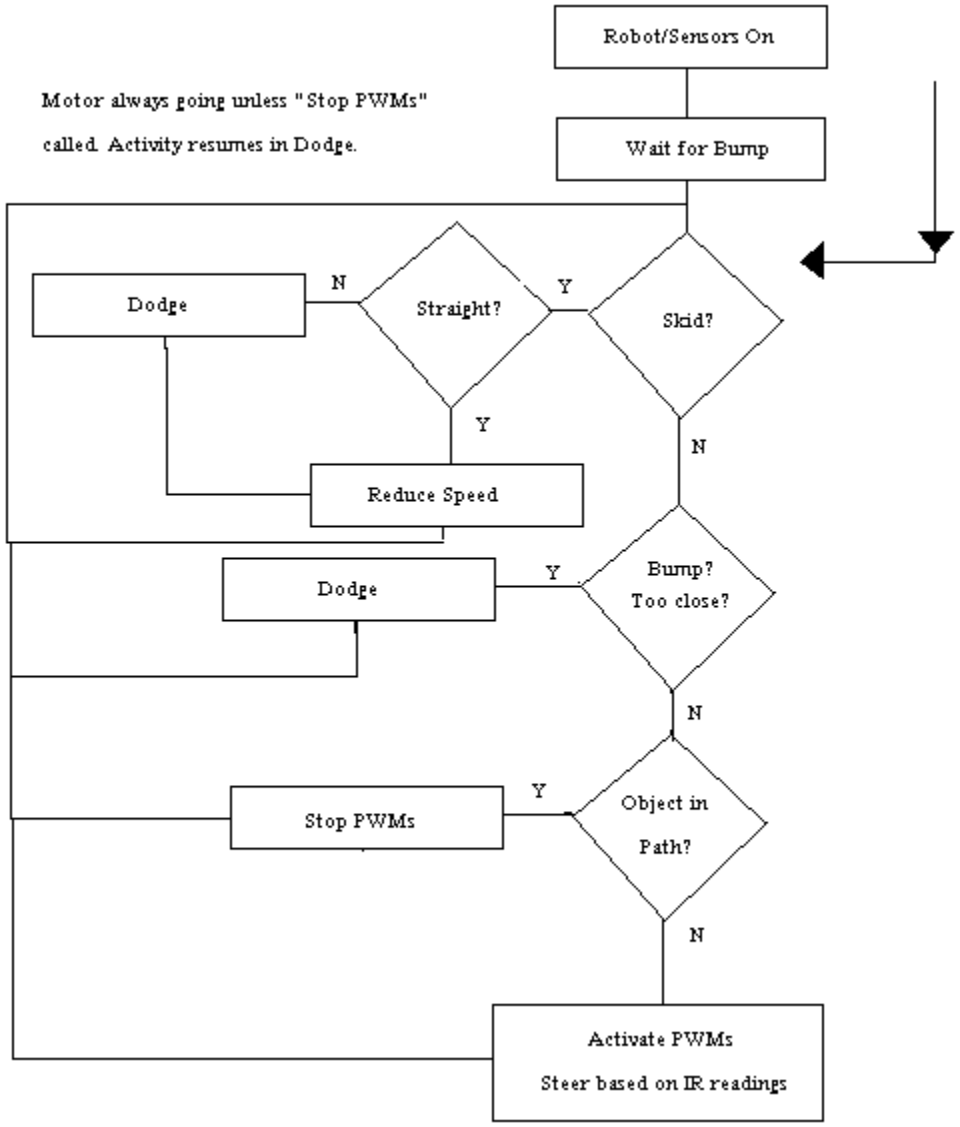
INTRODUCTION

This project's goal was to refine high-speed line-tracking on a rear-wheel drive vehicle. Initial B attempts to stay on a high-contrast line while driving as fast as it can and avoiding obstacles in its path. It uses 4 types of sensors with a total of 11 sensors on it. Because it uses infrared sensors for line detection each IR output is connected to a potentiometer so that line data can be calibrated due to changes in lighting conditions.

INTEGRATED SYSTEM, MOBILE PLATFORM, ACTUATION

The system is designed to emulate a rear-wheel drive car to handle well while moving quickly. It contains a motor and a servo. The motor acts as the engine to drive the rear axle, and the servo controls the steering of the front wheels. Six infrared transmitter/detector pairs (Sharp GP2D120 model), when operational, track the movement of the high contrast line in relation to the robot and a table, reduced to 8 possible directions changes, is read. The servo responds accordingly. The infrared receivers are set up in a U shape, and are referred to as FL and FR (front left and right),

ML and MR (mediums), and BL and BR (back left and right, the bottom of the U-shape).



Initial B's logic is as follows:

The IR logic is as follows:

if (FL and RL) or (ML + RL): Perpendicular, check memory and decide where to turn

if (FL and !ML): Medium turn left

if (FR and !MR): Medium turn right

if (ML and !MR): Sharp turn left

if (MR and !ML): Sharp turn right

if (BL and BR): Forward

if (BL or (ML and FL)): Tweak to the left

if (BR or (MR + FR)): Tweak to the left

SENSORS

Six Sharp GP2D120 infrared sensors are used to determine the location and curvature of the line so that the robot may align itself to it. A Devantech SRF04 ultrasonic sensor checks for obstacles in front of the robot and gives a distance measurement to the obstacles, accurate to within 2". Bump sensors on the front left and right sides of the robot detect collisions. Optical interrupters on the front and rear axles measure the velocity of Initial-B and check for loss of traction in the wheels. They also work with the sonar to determine a safe distance to stop moving without colliding with any obstacles.

OPTICAL INTERRUPTERS

The Fairchild H21A1 optical interrupter switch consists of an infrared emitter and a phototransistor housed in plastic and separated by a 1/8" gap. When an opaque material passes through the gap it blocks the signal between the emitter and phototransistor, switching the output between on and off. The external interrupts are set on Port A, pins 6 and 7 (front and back axles). Timer counter 3 is used, and an interrupt is set to occur at any logic change. The opaque material—rubber in this case—is attached to the axle so that it blocks the phototransistor, and therefore causes a logic change, once per

revolution. A prescaler of 256 is used so that each count of the timer TCNT3 corresponds to .032 millisecond. Whenever a new interrupt is generated, the value of the last interrupt on that axle is compared to check for timer overflow. This way the speed of both axles can be accurately measured as long as there is more than one revolution per 16-bit timer cycle, about every 2.1 seconds.

CONCLUSION

I learned a lot about robots. What to do, but more importantly, what not to do. I learned a lot of things not to do. The software and sensors are tricky enough to get working, having a complicated design is asking for trouble. Things will blow and go bad, so it's best to always keep 2 spares. It's important to get a platform soon so that everything can be attached to it immediately; the longer connections are loose and get unplugged and reconnected, the greater the chances of connecting something backward. Doing a few hours of research and measurements, checking for voltage and continuity everywhere, and double-checking pinouts will save a lot of time and headaches in the long-run. Of course, if you don't get everything organized and put on your board/platform, you're going to be doing that several times and it gets inefficient. Don't make boards unless you need them, because that ends up being more space used and more cables and that allows for more errors and problems. Dremels, machine screws, and taps are great in a pinch but they aren't a replacement for careful planning. In the end, Initial-B was an expensive project, but can you really put a price on learning?

DOCUMENTATION

Atmel Atmega128 - http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf
Sharp GP2D120 IR module - http://info.hobbyengineering.com/specs/gp2d120.pdf

LCD display - http://www.mil.ufl.edu/courses/eel4744/docs/LCD_Notes_4-bit.pdf
Futaba Servos - http://www.servocity.com/html/s148_standard_precision.html (not hacked)
LM339 - http://www.onsemi.com/pub/Collateral/LM339-D.PDF
Optical Interrupter: http://www.ex.ac.uk/~sritchie/hydro/datasheets/H21A1.pdf

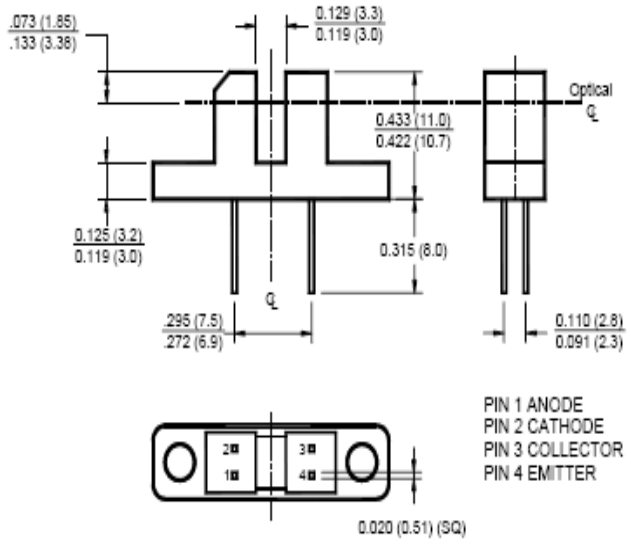
APPENDICES

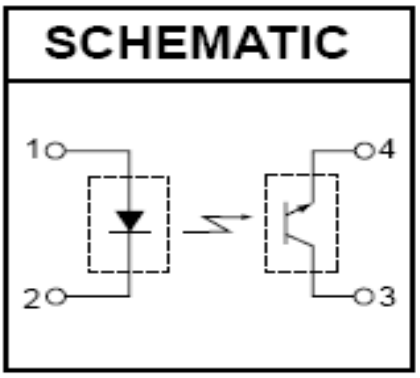
Appendix A: Price list

Manufacturer	Part number	Part description (Qty)	Price
Mekatronix	MEKAVR128	Atmega128 development board (1)	$90.00
Mekatronix	RS-232 Programmer	uC Programmer (1)	$ 25.00
Sharp	GP2D120	IR range finder (6)	$ 82.50
Hitachi	HD44780	24x2 LCD (1)	$ 10.00
n/a	NDL202 0402	Inverter (1)	$12.50
Radio Shack	n/a	RSX RC Car	$80.00
Hi-Tech	HS-422	Standard servo (1)	$ 10.50
Devantech	SRF04	Ultrasonic Range Finder (1)	$ 40.00
National	LM339	Quad Comparator (2)	Samples
Fairchild	H21A1	Optical Interrupter	From Lab

Total cost (neglecting things destroyed or not used): $350.50

Appendix B: Part information





Pictures obtained from:

http://www.ex.ac.uk/~sritchie/hydro/datasheets/H21A1.pdf

Program:

```
// Ian Gerstel
// 4/28/05
// Initial B program
// High speed line following with 6 IRs, 2 bump sensors, sonar, lcd, and a servo and
motor
// Code involving sonar modified from Sara Keene
// Code involving LCD, motor, and servo made with help from William Dubel and Steven
Pickles

// char = 8 (255)                long = 32
// int = 16       (65535)                long long = 64

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
```

```
#include <avr/signal.h>
#include <inttypes.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

#define opt PORTA
#define drive PORTB
#define bumpson PORTC
#define ir PORTD
#define lcd PORTE
#define BUMPSON PINC
#define IR PIND

#define OCR_1MS 125

int OCR_50MS=6250;
int del1, del2, del3;
int temp, init, dist, oldcount, oldcount2;
int speed, maxspeed;
int EN = 0x04;
int RS = 0x01;
int DB = 0xff;
int clear[] = {0x08, 0x18};
int l,r, tl,tr,ml,mr,bl,br, intf,intr;
int turn, bumpl, bumpr;
```

```
double velocity;

int def_putc(char);
void lcd_top(void);
void dels(int); // may be twice as long as it should
void delm(int);          // may need 4000 instead of 8000
void delu(int);
void lcd_init(void);
void lcd_clr(void);
void lcd_line2(void);

void init_timer1(void);
void forward(int);
void brake(void);
void decel(void);

void perpend(void);
void center(void);
void medright(void);
void medleft(void);
void farright(void);
void farleft(void);
void smright(void);
void smleft(void);

void dodge(void);
```

```
void init_timer2(void);
int sonar_read(void);
void init_timer3(void);
void rpms_f(void);
void rpms_r(void);

int main (void) {
        DDRA = 0x00;        // opt-ints on 6 (f), 7 (r), input (00xx xxxx)
        DDRB = 0x78;        // motor on pins 5,4,3 (enable, +, -), servo on 6 (x111 1xxx)
        DDRC = 0x02;        // bumps on pins 2,3, sonar on 0(in),1(out) (xxxx 0010)
        DDRD = 0x00;        // IRs on pins D2-7, all input (0000 0000)
        DDRE = 0xFF;        // LCD pins 7,6,5,3,2,1,0 output (111x 1111)

        fdevopen(def_putc, NULL, 0);

        lcd_init();
        lcd_clr();

        bumpl = bumpr = 0;
        maxspeed = 5000;

        init_timer1();
        init_timer3();
        sei();

        while ((!bumpl)&&(!bumpr)) {        // sit still until bump hit
                delm(500);
```

```
        tl = (IR&0x04);
        tr = (IR&0x08);
        ml = (IR&0x10);
        mr = (IR&0x20);
        bl = (IR&0x40);
        br = (IR&0x80);
        if (tl!=0)
                printf("tl ");
        if (tr!=0)
                printf("tr ");
        if (ml!=0)
                printf("ml ");
        if (mr!=0)
                printf("mr ");
        if (bl!=0)
                printf("bl ");
        if (br!=0)
                printf("br ");
}

intf = intr = TCNT3;  // initialize interrupter counter

speed = maxspeed*.2;          // get up to speed
forward(speed);
init_timer2();
delm(50);
speed = maxspeed*.4;
```

```
        forward(speed);
        delm(50);
        speed = maxspeed*.6;
        forward(speed);
        delm(50);
        speed = maxspeed*.8;
        forward(speed);
        delm(50);
        speed = maxspeed;
        forward(speed);

        while(1) {
                lcd_top();
                delm(100);
                tl = (IR&0x04);         // br bl mr ml tr tl intf intr
                tr = (IR&0x08);
                ml = (IR&0x10);
                mr = (IR&0x20);
                bl = (IR&0x40);
                br = (IR&0x80);
                bumpl = (BUMPSON&0x04);
                bumpr = (BUMPSON&0x08);

                velocity = 7.85*1000/(TCNT3*.032);          // velocity is inches/sec (.032
ms per count, wheel dia = 2.5")

                // Priority: skid, bump, sonar, line
```

```
                if ((intr<(intf*.8) || (intr>(intf*1.2))) && (OCR1B==750)) {
                        maxspeed = maxspeed*.9;
                        speed = maxspeed;
                        forward(speed);
                }
                else if ((intr<(intf*.8)) || (intr>(intf*1.2))) {
                        dodge();
                        maxspeed = maxspeed*.9;
                        speed = maxspeed;
                        forward(speed);
                }
                else if (bumpl!=0) {
                        dodge();
                }
                else if (bumpr!=0) {
                        dodge();
                }
                else if (velocity >= 2*dist) {          // velocity is in inches/sec, assume 2
secs stop
                        dodge();
                }
                else if (dist<15)       // 15 is arbitrary
                        decel();
                else {
                        forward(speed);
                        if ((tl&&bl) || (ml&&bl))
                                perpend();
```

```
                else if (tr&&!ml)
                        medleft();
                else if (tr&&!mr)
                        medright();
                else if (ml&&!mr)
                        farleft();
                else if (mr&&!ml)
                        farright();
                else if (bl || (ml&&tl))
                        smleft();
                else if (br || (mr&&tr))
                        smright();
                else center();
            }
        }
        return(0);
}

//
void init_timer1(void) {
        TCCR1A = 0xA8;      // 10101000, ABC clear on match, PWM phase+freq correct
        TCCR1B = 0x12;      // 00010010, prescale/8, PWM phase+freq correct

        ICR1 = 5000;  // 5000/(8000000/8) = 5ms, 10ms wave
        // OCR1A,B,C = PORTB pins 7,6,5 (a7, B6, C5)
}
```

```
//
void init_timer2(void) {
        TCCR2 = 0;
        TIFR  |= BV(OCIE2)|BV(TOIE2);
        TIMSK |= BV(TOIE2)|BV(OCIE2);        // enable output compare interrupt
        TCCR2  = BV(WGM01)|BV(CS02)|BV(CS00); // CTC, prescale = 128
        TCNT2  = 0;
        OCR2   = OCR_50MS;
}

SIGNAL(SIG_OUTPUT_COMPARE2) {
        sonar_read();
}

//
void init_timer3(void) {
        TCCR3A = 0x00;              // no output compare, normal counting
        TCCR3B = 0x44;              // 0100 0100, prescale/256, normal counting
(.032ms/count)
        TCCR3C = 0x00;              // don't force output compares
        EICRB = 0xA0;               // generate interrupts on pins 7,6 on any edge
        EIMSK = 0xC0;               // enable ext ints on pins A6, A7
}

SIGNAL(SIG_INTERRUPT0) {
        rpms_f();
}
```

```
SIGNAL(SIG_INTERRUPT1) {
	rpms_r();
}

//
void rpms_f(void) {
	if (TCNT3 < oldcount)
		intf -= (65535 + TCNT3);
	else intf -= TCNT3;
	oldcount = TCNT3;
}

//
void rpms_r(void) {
	if (TCNT3 < oldcount2)
		intr -= (65535 + TCNT3);
	else intr -= TCNT3;
	oldcount2 = TCNT3;
}

//
int sonar_read(void) {
	//printf("sonar read");
	dist = 0;
	int timeout = 50;
```

```
	bumpson = (bumpson|(!0x02));  //start with uC output (pin 1) pulse low (sonar input)

	bumpson = (bumpson|0x02); // PULSE Trigger sonar input, pin 1
	delu(20);	//pinging
	bumpson=(bumpson|(!0x02)); // end ping
	delu(120); // 100us delay before receiver circuitry is enabled

	while((BUMPSON&0x01)==0x00 && timeout) { // pin 0 is uC input (out from sonar)
		timeout--;
		delu(100);                                                    // timeout makes sure it doesn't freeze if no response
	}                                                                     // while ECHO is low, wait for ECHO receive

	while((BUMPSON&0x01) == 0x01) {                 // while ECHO is high, it is determining distance
		dist++;
		delu(40);
	}
	lcd_clr();
	printf("dist=%u", dist);
	return(0);
}

void dodge() {
```

```
        if (turn==1)
                farright();
        else farleft();
        delm(100);
        center();
        brake();
        delm(50);
        speed = maxspeed*.5;
        forward(speed);
        delm(50);
        if (turn==1)
                farleft();
        else farright();
        speed = maxspeed*.75;
        forward(speed);
        delm(50);
        speed = maxspeed;
        forward(speed);
}

//
void forward(int spd) {
        OCR1C = spd;          // set pins 5+4 high (xxx1 0xxx) -- was xx11 0xxx
        drive = (drive|0x10);
        drive = (drive&!0x08);
        turn = 0;
}
```

```
//
void brake(void) {
        drive = (drive|0x18);  // set pins 4+3 high
}

//
void decel(void) {
        OCR1C = 0x00;         // no PWM
}

//
void perpend(void) {
        brake();
        if (turn<0)       // if perp to line while moving left
                farright();                  // make a sharp right
        else
                farleft();
        delm(50);
        speed = maxspeed*.5;
        forward(speed);
        delm(50);
        speed = maxspeed*.75;
        forward(speed);
        delm(50);
        speed = maxspeed;
        forward(speed);
```

```
}

//
void center(void) {
        OCR1B = 750;          // 1.5ms = 15% duty, 750
}

//
void farright(void) {
        OCR1B = 1000;         // 2ms = 20% duty, 1000
        turn = 1;
}

//
void farleft(void) {
        OCR1B = 500; // 1 ms = 10% duty, 500
        turn = -1;
}

//
void medright(void) {
        OCR1B = 900;          // 2ms = 20% duty, 1000
        turn = 1;
}

//
void medleft(void) {
```

```
        OCR1B = 600; // 1 ms = 10% duty, 500
        turn = -1;
}

//
void smright(void) {
        OCR1B = 800;          // 2ms = 20% duty, 1000
        turn = 1;
}

//
void smleft(void) {
        OCR1B = 700; // 1 ms = 10% duty, 500
        turn = -1;
}

//
int def_putc(char ch) {
        // send high nib
        lcd = ((ch&0xf0)|RS)|EN;     // latch when enable high
        delu(200);
        lcd = ((ch&0xf0)|RS);                  // send when enable low
        delu(200);

        // send low nib
        lcd = ((ch<<4)|RS)|EN;        // latch when enable high
        delu(200);
```

```
        lcd = (ch<<4)|RS;               // send when enable low
        delm(2);

        return ch;
}

//
void lcd_top(void) {
        init = 0x02;    // display on; cursor on; blink on

                // send high nib
        lcd = (init&0xf0)|EN; // latch when enable high
        delu(200);
        lcd = (init&0xf0);              // send when enable low
        delu(200);

                // send low nib
        lcd = (init<<4)|EN;    // latch when enable high
        delu(200);
        lcd = (init<<4);                // send when enable low

        delm(2);
}

//
void dels(int slen) {
// 1/8 Mhz = .000000125 s
```

```
// 1ms = .001 s
// .001/.000000125 = 8000

        for (del1=0; del1<8000; del1++) {
                for (del2=0; del2<100; del2++) {
                        for (del3=0; del3<slen; del3++) {
                        }
                }
        }
}

//
void delm(int mlen) {
// 1/8 Mhz = .000000125 s
// 1ms = .001 s
// .001/.000000125 = 8000

        for (del1=0; del1<8000; del1++) {
                for (del3=0; del3<mlen; del3++) {
                }
        }
}

//
void delu(int ulen) {
// .000001/.000000125 = 8
        for (del1=0; del1<8; del1++) {
```

```
                for (del3=0; del3<ulen; del3++) {
                }
        }
}

void lcd_clr() {
        init = 0x01;     // display on; cursor on; blink on

                // send high nib
        lcd = (init&0xf0)|EN; // latch when enable high
        delu(200);
        lcd = (init&0xf0);              // send when enable low
        delu(200);

                // send low nib
        lcd = (init<<4)|EN;     // latch when enable high
        delu(200);
        lcd = (init<<4);                // send when enable low

        delm(2);
}

void lcd_line2(void) {
        init = 0xc0;     // display on; cursor on; blink on

                // send high nib
        lcd = (init&0xf0)|EN; // latch when enable high
```

```
        delu(200);
        lcd = (init&0xf0);              // send when enable low
        delu(200);

            // send low nib
        lcd = (init<<4)|EN;     // latch when enable high
        delu(200);
        lcd = (init<<4);                // send when enable low

        delm(2);
}

//
void lcd_init(void) {
// Initialize the LCD
        lcd = 0x00;

        delm(16);
        init = 0x30;

        lcd = init|EN;  // latch when enable high
        delu(200);
        lcd = init;             // send when enable low

        delm(5);
        init = 0x30;
```

```
lcd = init|EN;  // latch when enable high
delu(200);
lcd = init;                  // send when enable low

delu(105);
init = 0x30;

lcd = init|EN;  // latch when enable high
delu(200);
lcd = init;                  // send when enable low

delm(5);
init = 0x20;

lcd = init|EN;  // latch when enable high
delu(200);
lcd = init;                  // send when enable low

delu(200);
init = 0x28;     // 2-line mode

        // send high nib
lcd = (init&0xf0)|EN; // latch when enable high
delu(200);
lcd = (init&0xf0);               // send when enable low
delu(200);
```

```
        // send low nib
lcd = (init<<4)|EN;     // latch when enable high
delu(200);
lcd = (init<<4);                    // send when enable low

delu(200);
init = 0x0f;

        // send high nib
lcd = (init&0xf0)|EN; // latch when enable high
delu(200);
lcd = (init&0xf0);                  // send when enable low
delu(200);

        // send low nib
lcd = (init<<4)|EN;     // latch when enable high
delu(200);
lcd = (init<<4);                    // send when enable low

delu(200);
init = 0x01;     // display on; cursor on; blink on

        // send high nib
lcd = (init&0xf0)|EN; // latch when enable high
delu(200);
lcd = (init&0xf0);                  // send when enable low
delu(200);
```

```
            // send low nib
        lcd = (init<<4)|EN;     // latch when enable high
        delu(200);
        lcd = (init<<4);                    // send when enable low

        delm(2);
}
```