LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

Department of Electrical and Information Technology

Digital Project EDI021

Robot

Andreas Lord
Zongyi Jiang
Baydai Abdulameer

2010-05-15

Abstract

There are several steps in the process of building a robot prototype. Designing and choosing components. Assembling the hardware. Writing software.

In a dream world all of this will work straight away, but it never does in the real world. While building this project, there were a lot of problems and solution were found to most of these problems, but not all of them.

This project is based around ATmega 16 micro controller and the JTAG developing interface. The were a lot of problems with these components together with the electrical engine. The project consumed several processors during its developing stages, and the reason for way these processors faulted is not really clear.

Context

Introduction

The purpose with this course is to illustrate how developing a new prototype can be done in the industry. The group is suppose to, from their own idea, design and build a prototype based around some logics or a micro processor.

This project built a robot that can manoeuvre by it self without hitting any obstacles in its surroundings.

Requirements

The requirements were set up at the beginning of the project as follows:

- The robot should drive around in a room with out hitting anything
- The robot's movement is controlled with a start/stop button
- The robot should stop completely if it can not avoid a collision

There were also some extra requirements in case of time in the project:

- The robot should display the way it walked
- The robot should be able to back out of a corner
- The robot should remember were it's been and never walk the same way twice

Hardware

The project is based on several important hardware components. The robot is controlled via a AVR ATmega 16 micro controller. To be able to detect obstacles the robot has two Sharp 2YOA02 analog distance sensors.
The robot is powered with a 6 volt electrical battery, but the components use 5 volt so a LP3855 voltage regulator is used. The mechanical part is based on two engines directly connected to one wheel each, via a small gearbox.
To control the engines with a digital signal a L298 full bridge driver is used.
To show the way the robot walked, the prototype have a GDM12864HLCM matrix display. Also two buttons and two led-lights are used, for controlling and showing state of the robot.

ATmega16

The ATmega16[1] is a 8-bit micro controller with 16 Kb memory. It has four 8 bits I/O ports, which can be used either as standard digital I/O or as some special functions. Some of the special functions that were used are AD converters and external interruption triggers. To program the microprocessor a J-tag interface was used, this connects the processor with the developing environment in a PC via USB.

Sharp 2YOA02

The distance sensor that was used in this project is an analog distance sensor. It has a range of 20 -150[2] cm with a angel of 8-60[3] degrees. it generates a analog voltage depending on the distance to an object, the closer the object is the stronger signal. Distance sensor has three connections, one for power supply, one to ground and one for signal.

L298

To control the engine with a digital signal a dual full bridge driver[4] were used. One for each engine. The full bridge takes two power supplies, one for the electronically logics and one stronger that is for the out put pins. The out put is

controlled via three digital signals per bridge, In1, In2 and enable.
The bridge allows using two wheels and this can control both wheels individually, either to left, right or let it roll freely.

LP3855

The LP3855[5] is a voltage regulator, that takes a input voltage up to seven volt, and generates a 5 volt output with an accuracy of ±1.5%. These ultra low dropout linear regulators respond very quickly to step changes in load, which makes them suitable for low voltage microprocessor applications.

GDM12864HLCM

The display that is used is a matrix display[6] with a resolution of 64 * 128 pixels. It is divided up into two equal halves of 64*64 pixels that are switched via chip select.
You can send two kinds of commands to, instructional-command and data-command. Instruction commands were for to choose where in the screen we wanted to write and data-commands were to write a byte (8 pixels in a row) at the selected location. The instructions were to select the x and y coordinates.

Execution

When the requirements were set up and approved, the schematics were drawn up. After a little consulting with the project supervisor, and the schematics were approved the construction began. There were some miner problems with the construction, but it was solved easily. The major time spent on the construction was the placement of the parts. The schematics did not specified physical layout of the components. All components where assembled at the same time.

The prototype is based on one rectangular circuit board with all the components, except the engine with wheels and gearbox, on the up-side. On the same side all power cables are laid out and soldered to corresponding pins. On the other side all logical cables are wired. The engine with wheels and front wheel is of the same hight so it naturally was attached to the down-side of the board, giving the board a horizontal level. A second circuit board holding the screen is placed over the first circuit board on four screws.

When the robot was build the programming started, first a small test program was written to ensure all parts worked as they should. Here we found out that one of the distance sensors did not work as intended, so it got replaced.
Next step in the programing was to learn all the special functions that the micro controller had. For example how the AD-converter[1] worked or external interrupts or the internal timers.

The program is based on a few different parts, all written in the same main file.

The first part is the main function that only first calls the set up function setReg() and then have three if statements that determent if the engines should be turned of or on due to obstacles detected by the distance sensors.

1 AD-converter = analog to digital conversion. The distance sensors gives a analog signal that needed to be converted to digital so that the micro controller can use the information.

The second part is the interrupt routines. There are one routine for every type of expected interrupt and one general that handles all other interrupts. The general interrupt routine is not doing anything in this prototype.

The interrupt routine for the external interrupts is turning on or off the engined depending on if the engines are on or off before the routine is called.

We are using a timer to activate the DC-conversion, when the timer is started it will run for ever. The timer is basically a counter that counts up, and every time the counter hit its maximum value it will trigger a interrupt and start over from zero.

The timer interrupt routine start a new AD-conversion. When the a AD-conversion is done, a AD interrupt is called. The AD interrupt routine saves the value of the conversion to a variable and then change the settings for the AD to the other sensor.

The last part is the set up function that sets up all the ports to be either out or in put according to the schematics. It also set up the special functions like which pin is just for external interrupt, and which pins is used for the AD-conversion, and it enables interrupts via the sei() function.

One of the extra requirements were to display the way the robot walked on a display. A solution for this was started, but not finished. All hardware bits were added, but a working software solution was never reached.

Result and Discussion

In the beginning of the project there were not so many problems, the schematics were rather pain free constructed, with some miner hints and tips from the supervisor. This part of the project demanded a lot of reading in component manuals to find out if this is the component that can be used in this project.

The construction part started good, with most time spent on actually planing where to put the different parts on the circuit board. We think we made a rather optimal design for both testing purpose and using the prototype.

After writing some test program we found out that one of the sensor was not working correctly, but due to the design it was easily changed.

In the next phase of the project the real big problems started. We had a small program where the external interrupt should give a enable the h-bridge so that the engine would run. But there were some major problems flashing the micro-controller. We ended up blocking 5 different micro controllers some how. While trying to run our program in debugging mode the micro controllers lowest registers called fuses got changed, causing the micro controller to get blocked for the interface used to flash the micro controller. In the end we still have no real solution to this problem, we made some minor changes in the code and then it worked.

We added on the functionality of the AD-conversion with its interrupt, and got this almost to work. The conversion was continuously starting over when one conversion was done. For testing purpose we first just turned on one of the LED-lights when a detections value was greater then a trigger value. This worked all good, but when we changed to turning on/off the engines instead of the LED-lights nothing worked, and the micro controller got blocked again. To solve this we added a timer that trigger an interrupt, and on that interrupt we start a new conversion. By doing this we released some working load of the processor and this made the conversions to work fine with the engines.

The last part added on to the project was the writing to the screen. When designing the hardware we already thought of this, so adding the hardware was easily done. We never reached this requirement though, due to several problems. we had a jumping reset signal, and never got to communicate with the screen. The debugging environment behaved really peculiar way, it jumped from instruction to instruction in a random way. The reason for this we never found out, but most probably it was due to the hardware.

One general reason for some of our problems can be because of the engines. When they are turned of, the still run in a free wheel mode for a short period of time. During this the engines actually works as a generator, which generates a currant that our prototype does not take care of.

Refrences

[1] Atmel Corporation, "8-bit AVR Microcontroller with 16K byte In-system programmable flash", 2003

[2] Sharp corporation, "GP2Y0A02YK0F", *Reference manual*, *2006*

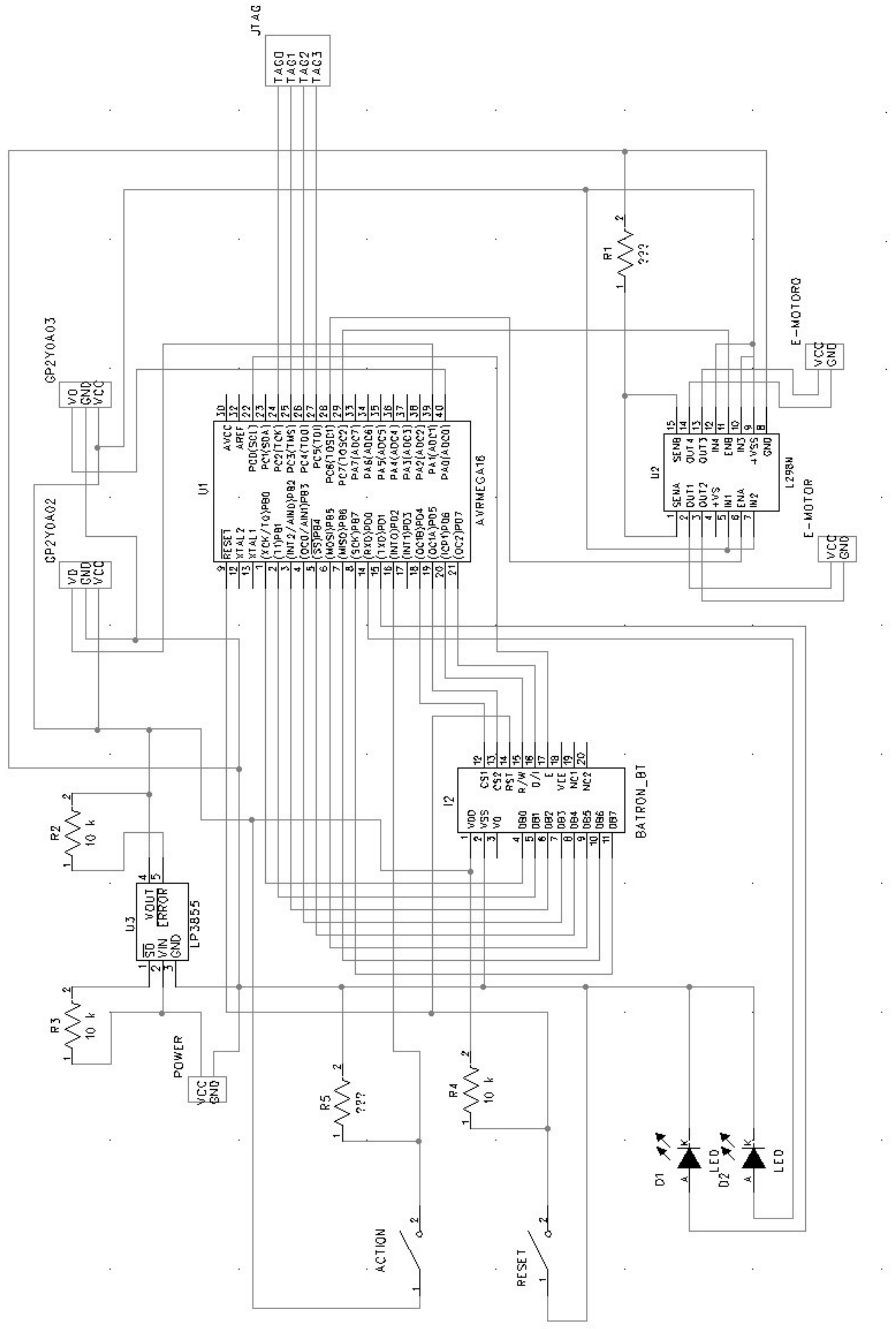
[3] Acroname Inc.
"http://www.acroname.com/robotics/parts/R144GP2Y0A02YK.html",
Web page, 2010

[4] STMicroelectronics, "L298 DUAL FULL-BRIDGE DRIVER",
Reference manual, 1998

[5] National Semiconductor Corporation, "LP3852/LP3855
1.5A Fast Response Ultra Low Dropout Linear Regulators",
Reference manual, 2005

[6] Unknown, "GDM-12864C", *Reference manual*,

Appandix A - Schematics



Apendix B – Source code

```
#define TRLV 60                              // Pre defined triger level
                                                for AD-converter to turn
                                                on
#define STLV 100                             // Pre defined triger level
                                                for AD-converter to stop
                                                on

#define LEFT 0                               //
#define RIGHT 1                              // Macros
#define OFF 0                                //
#define ON 1                                 //

#include <avr/io.h>                          // IO libary
#include <avr/interrupt.h>                   // Interupt libary



void setReg();                               // Initial the setup function
volatile int on_off;                         // Boolean variable that
                                                defines if the engine is
                                                running or not
volatile int det_right;                      // Varible to store value
                                                from sensor 1
volatile int det_left;                       // Varible to store value
                                                from sensor 2
volatile int left_right;

int main(void)                               // Main method
{

    setReg();                                // Call for setup function

    while(1){                                // Eternal loop
        if(on_off == ON){
            if (det_right >= TRLV){
                PORTC &= ~_BV(PC7);
            }else{
                PORTC |= _BV(PC7);
            }
            if (det_left >= TRLV){
                PORTC &= ~_BV(PC6);
            }else{
                PORTC |= _BV(PC6);
            }

        }
    }
    return 0;
}

ISR(INT0_vect)                               // Interuption rutine for
                                                external interuption
                                                through button
{
    if(on_off == OFF) {
        on_off = ON;
        PORTC |= _BV(PC6)|_BV(PC7);

    }else{
        on_off = OFF;
        PORTC &= ~_BV(PC6)&~_BV(PC7);
    }
}

ISR(ADC_vect)                       // Interuption rutine for AD-
                                       convetion
{
    if(left_right == RIGHT){
        det_right = ADCH;
        left_right = LEFT;
        ADMUX |= _BV(MUX0);
    }else{
        det_left = ADCH;
        left_right = RIGHT;
        ADMUX &= ~_BV(MUX0);
    }

}

ISR(TIMER0_OVF_vect)                    // Interuption rutine for
                                           overflow on timer
{
    ADCSRA |= _BV(ADSC);
    PORTD ^= _BV(PD0);
}

ISR(BADISR_vect)                        // Default interuption rutine
{

}

void setReg()                       // Setup function
{
    on_off = OFF;                   // Set bollean
    det_right = 0;                  // Give varible start value
    det_left = 0;                   // Give varible start value

    DDRD |= _BV(PD0)|_BV(PD1);      // Set pin PD0 and PD1 to output
    PORTD &= ~_BV(PD0)&~_BV(PD1);   // Give output value 0 to pin PD0
                                       and PD1

    DDRC |= _BV(PC6)|_BV(PC7);      // Set pin PC6 and PC7 to output
    PORTC &= ~_BV(PC6)&~_BV(PC7);   // Give output value 0 to pin PC6
                                       and PC7

    TIMSK |= _BV(TOIE0);            // Enable timmer interupt

    GICR |= _BV(INT0);              // Enable external interupt 0
    MCUCR |= 0x03;                  //  Setting external interupt to
                                       trigger on rinsing flank
    sei();                              // Call for function that
                                           activats interuption

    left_right = LEFT;                  // Setting varible
    ADMUX &= 0x00;                      // Clearing register
    ADMUX |= _BV(REFS0)|_BV(ADLAR);     // Selcting reference to ADC
    ADCSRA |= _BV(ADEN)|_BV(ADIE);      // Enable AD converter and
                                           interupt for ADC


    TCCR0 |= _BV(CS02);//               // Starting timmer with 256
                                           prescaling
}
```

Apendix C – Photos