Fachhochschule Esslingen, Hochschule für Technik,

Standort Göppingen, Fachbereich Mechatronik, PDV

Université Toulon-Var, institut universitaire de technologie, gestion électronique et informatique industrielle

Development of software, electronics and conceptions for a mobile robot

SS 2002: 11. Februar bis 19. Juli

Maximilian Heise, Mat. Nr. 721160

Betreuer FHTE:

Prof. Dr.-Ing. Klaus Harig

Betreuer Université Toulon-Var:

Prof. Philippe Arlotto, GEII

Prof. Didier Colas, Chef du Pôle Cultur

1 Table of Contents:

2 Introduction

2.1 Principal goal

The principal goal of this project is to develop strategies and first solutions for an autonomous mobile robot which will be participating in a competition between IUTs of different French universities.

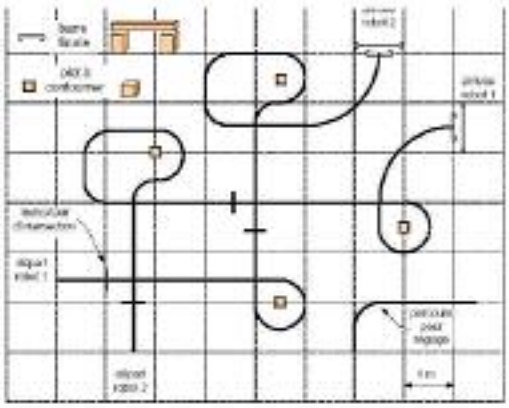
2.2 The mobile robots hardware

The hardware, that is the chassis, the motors, the wheels and the battery are standardized to make the competition not a mechanical one but a competition of software and electronics.

For more information see [1, rules.pdf]

2.3 Rules of the competition

To robots must complete a course looking similar to the one in Picture 2-1.



Picture 2-1, the course for the competition

For more information see [2, Base_roulante.pdf].

2.4 Université Toulon-Var, IUT, GEII

The university of Toulon is situated east of Toulon in a suburb called La Garde which lies between Mont Coudon to the Nord, the sea and another smaller community called Le Pradet to the south. The next town to the west is Hyères, a beautiful town which is in general friendlier than Toulon, which is not typical for the French Côte d'Azur because of its large military port and arsenal.

What is very practical is that everything a student living on campus needs is within 10min walk.

Picture 2-2, Université Toulon-Var, east side

Picture 2-3, Toulon

Picture 2-4, La Garde

2.5 Working Environment

The project documentation in general is written in English as the most common denominator, to be accessible to the staff and students in Toulon as well as back in Germany. Most of the supplied documents describing the robot are in French, therefore some effort was made to extract some essential information to be explained here. External documentation of different ICs, like the Microchip's PIC midrange family, is in English.

Most of the manuals are in Adobe PDF format.

For designing electronics (schematics and layout) the French version of Proteus 5.2. was used. For developing the software Microchip's MpLab in combination with Hi-Tech's PIC c compiler PICC together XEmacs as editor were used.

3 First steps with a PIC16F84 on a prototype board

At first, some practice was needed to get accustomed with Microchip’s free IDE Mplab, the eepromer and the pic family in general.

3.1 Programming asm on RISC architectures

Professor Arlotto first had to explain the RISC (Reduced Instruction Code) way to program in assembler. The RISC philosophy is that you do not have a lot of different special assembler commands, e.g. Intel x86 "JNZ", "jump if zero flag not set" or "JNC", "jump if carry flag not set", but a small number of multi-purpose commands. In addition to that, instructions are optimised so that they all take the same amount of time to execute (1µs for normal and 2µs for branches on the PIC16F84 and PIC16F877 at 4MHz). RISC makes it easier for the programmer to program in assembler.

So "JNZ" becomes btfss STATUS, 2 on the PIC midrange family.

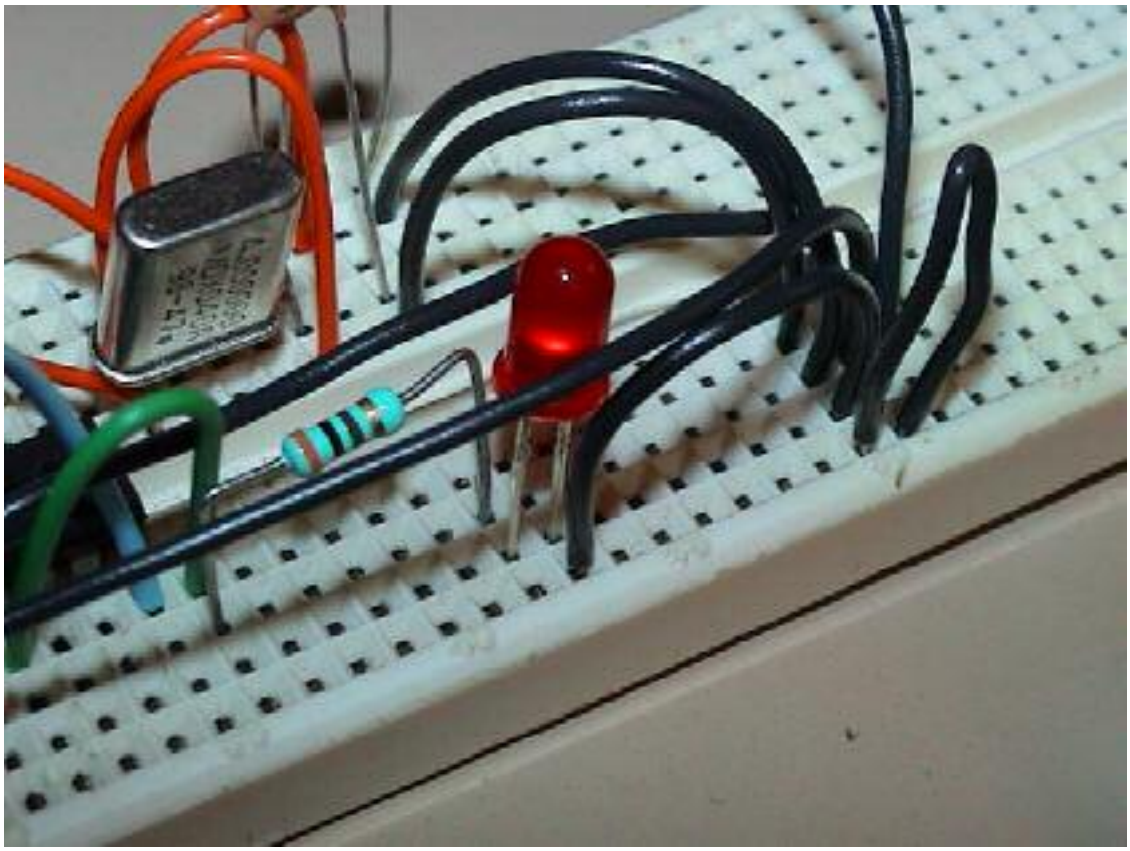
btfss STATUS, 2 (or btfss 0x03, 2 as STATUS is only a symbolic name for address 0x03) stands for bit test file skip set. btfss skips the next instruction if the specified bit is set. btfsc, bit test file skip clear is available, too. For more informations see [3, PIC16F84a.pdf manual, p. 57]

As a result there are only 35 single instructions to learn to program the PIC16F84 or PIC16F877, respectively.

The PIC16F8X manual is available at Microchip's web site at [4, http://www.microchip.com].

3.2 The first program

This first program is flashing a LED with 0.5ms off and 1ms on if a press button connected to and input is pressed.



Picture 3-1, 16F84 on a prototype board

This little more than cut 'n paste from a project of Professor Arlotto. Understanding and verifying it was a good way to get to know the PIC's assembler better ([6, ex001.asm]). Because the press button acts as a pull-down a 0V at the input signifies that the button is pressed.

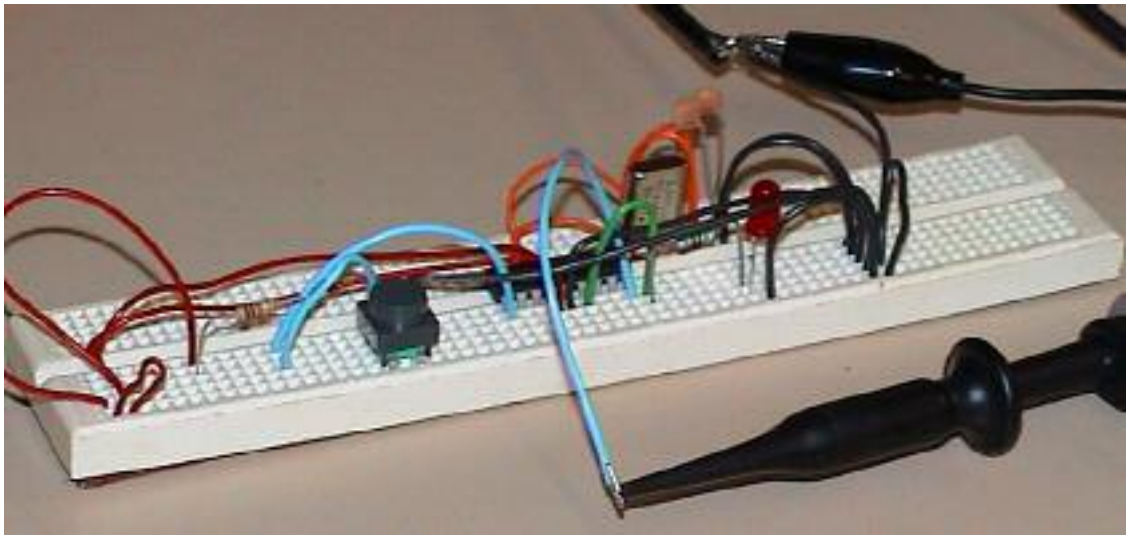
The second program was still simple ([7, ex002.asm]) , and does nothing more then switching a led on if you press a button and switching it back off when the button is not pressed.

4 PIC16F628 on a prototype board

Once more familiar with the small PIC, the next circuit with a PIC16F628 was built to begin generating PWM signals. At the beginning, two fixed relations between the PWM period and the duty cycle were used, which were chosen by pressing or releasing a button. Later on a poti was used to read an analog value, which was scaled to the 8 bits the PWM module is offering in the MSB register. This was then used to generate a variable PWM duty cycle.

For a PWM explanation see 16.2 PWM signal generation

See the Picture 4-1 to get an idea.



Picture 4-1, showing a prototype with a 16F628

See the source, [8, pwm001.asm] to see the functionality of this prototype.

For more information on the PIC16F62X see [4, PIC16F62X manual]

5 Generating two PWM signals on a PIC16F877

Once the PWM generation was working for one channel, the project was transferred to the PIC16F877, which is offering two PWM modules.
For a detailed list of the feature of the 16F87X family see [9, PIC16F87X manual]

5.1 FSR and INDF registers

When programming in assembler the indirect addressing feature of the PIC µC family is very practical. By writing an address to register FSR (file select register) reading and writing register INDF addresses the register pointed to by FSR. For more information see [9, p. 26].

Indirect addressing example

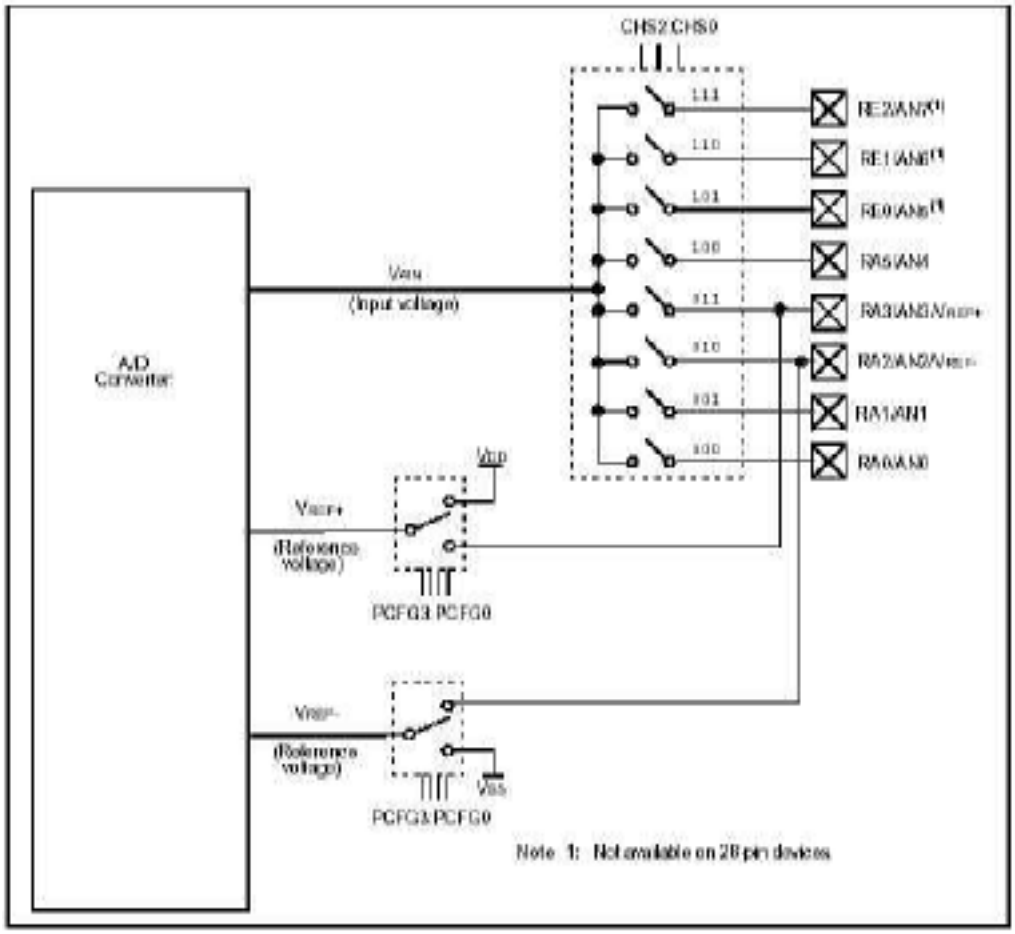
```
movlw 0x20      ;initialize pointer
movwf FSR       ;to RAM
NEXT
clrf INDF       ;clear INDF register
incf FSR        ;inc pointer
btfss FSR,4     ;all done?
goto NEXT       ;no clear next
                ;yes continue
```

I used the FSR feature here to toggle between the two ADC and PWM channels I used in this project. See the asm function togglePWMChannel for details.

The source is again available at [9, pwm_f877.asm

5.2 ADC conversion on the PIC16F877

The 16F877 has 8 multi-channel 10 bit ADCs (Analog to Digital Converters) with sample and hold and configurable high and low voltage reference.



Picture 5-1, AD block diagram on the PIC16F87X

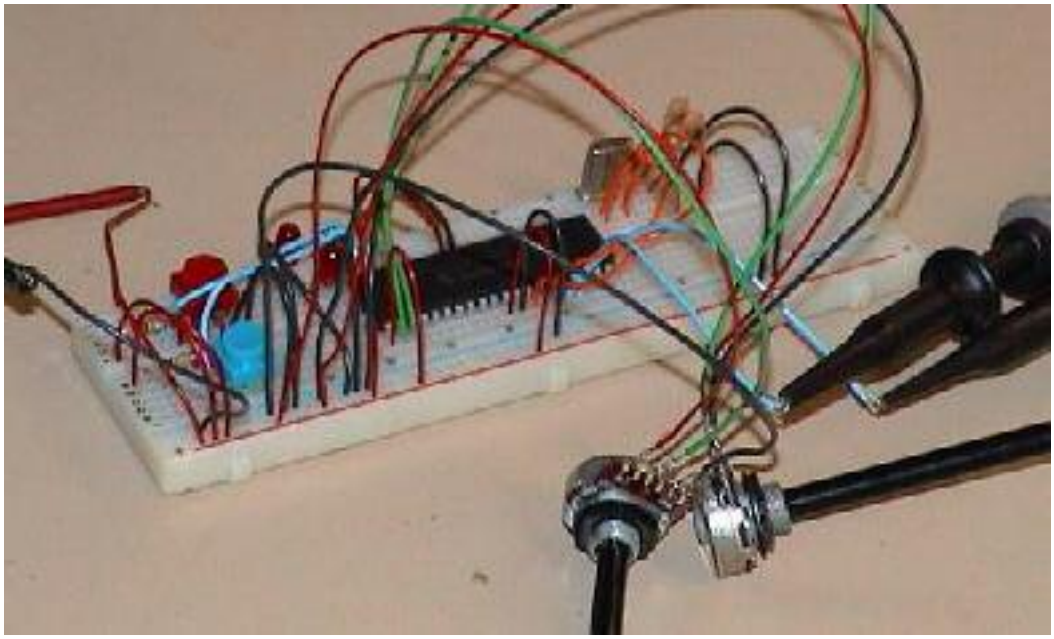
As it can be seen in Picture 5-1, only one AD conversion can be done at one time, it is not possible to do several AD conversions at one time.

If there are no more spare pins left on the µC, some pins can economized by using all but one AD pins as GPIO and connecting the AD sources to a address decoder which is in turn connected to the last pin configured for AD.

In such a way, with this 8 pins on the AD module, it is possible to have $2^{5=}32$ separate AD inputs with high and low voltage reference, or have 2^3=8 AD inputs with high and low voltage reference and 2 more pins free.

Note that the RA4/T0CKI pin, which lies between the other pins of port A, which is shared with the ADC module, is an open drain output and Schmitt trigger input.

Note again that the port A pins are configured as AD input, therefore a more special initialization sequence is needed, see [9, PIC16F87X manual, p. 29] for more information.

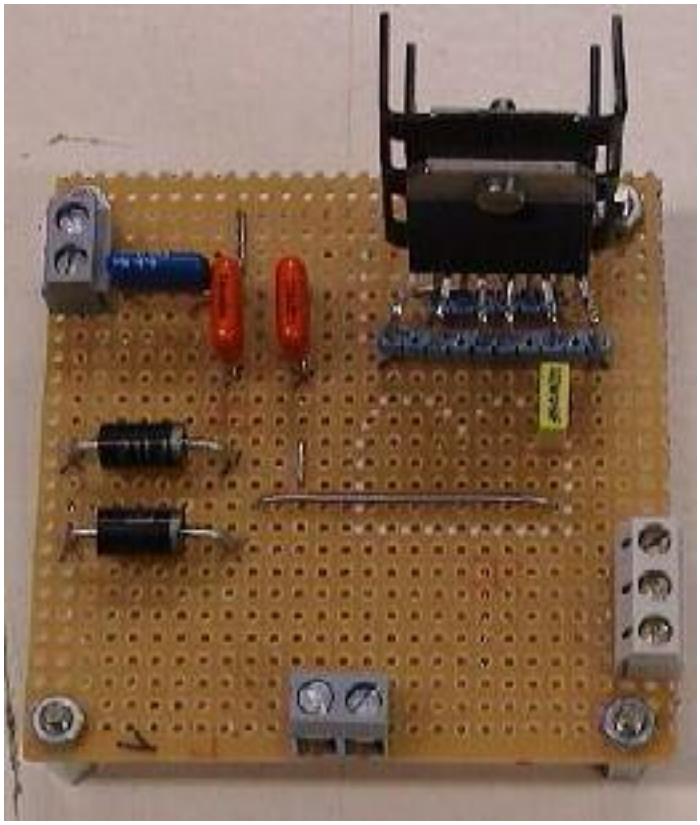


Picture 5-2, PIC16F877 generating first dual pwm signals

Examples of how to use and configure the AD module can be found in ad.c and ad.h or [10, pwm_f877.asm].

6 Motor electronics, H-bridge and PWM operation

The two motor electronic boards have inputs for ground and +12V (2 pin connector on the upper left corner), inputs for signals from the microcontroller, ENABLE, IN1 and IN2 (3 pin connector on the lower right side corner) and outputs to the motor (2 pin connector in the middle of lower side).

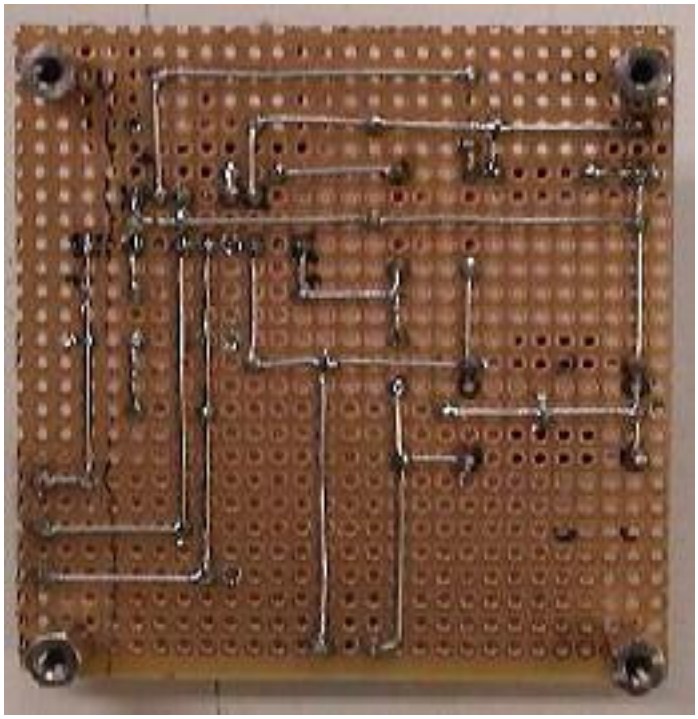


Picture 6-1, motor electronics, top view

6.1 H-Bridge Operation

Because the motor can not be driven directly by the current supplied by the µC, a special circuit is needed to get the energy for the motor's operation from a separate power source.

While this circuit can be build by different parts like relays, bipolar transistors, power MOSFETs or specials ICs the schema of this circuit always stays the same.



Picture 6-2, motor electronics, bottom view

A H-Bridge always consists of four switches which form the sides of the letter H and the connectors for the motor which form the middle of the H.

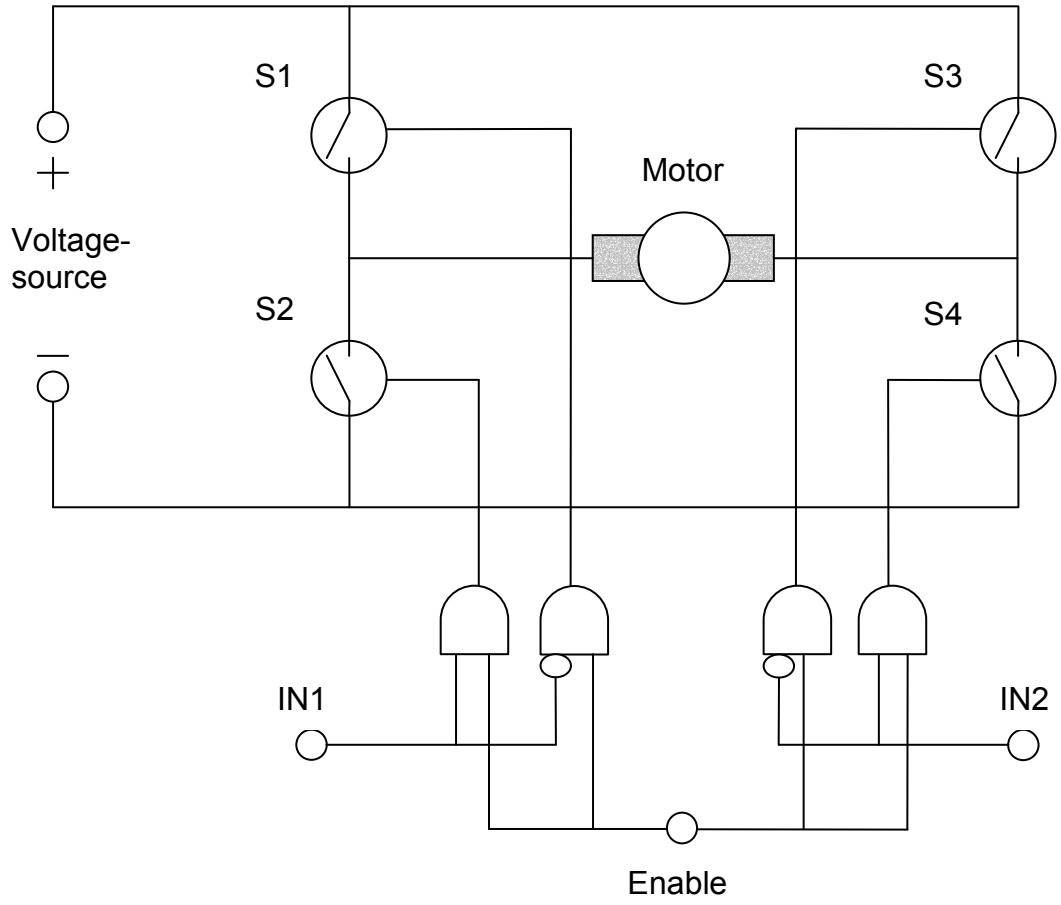
The switches are opened and closed in such a way so that a voltage in a certain direction drives the motor left or right.

If the switches S1 and S4 are open while S2 and S3 are closed the motor turns to one direction and if S2 and S3 are open while S1 and S4 are closed the motor turns to the other direction.

The motor will turn without any resistance if all switches are closed, and will brake if S1 and S3 are closed and S2 and S4 are opened or vice versa.

Enable, IN1 and IN2 are signals from the µC needed to drive the switches S1, S2, S3 and S4.

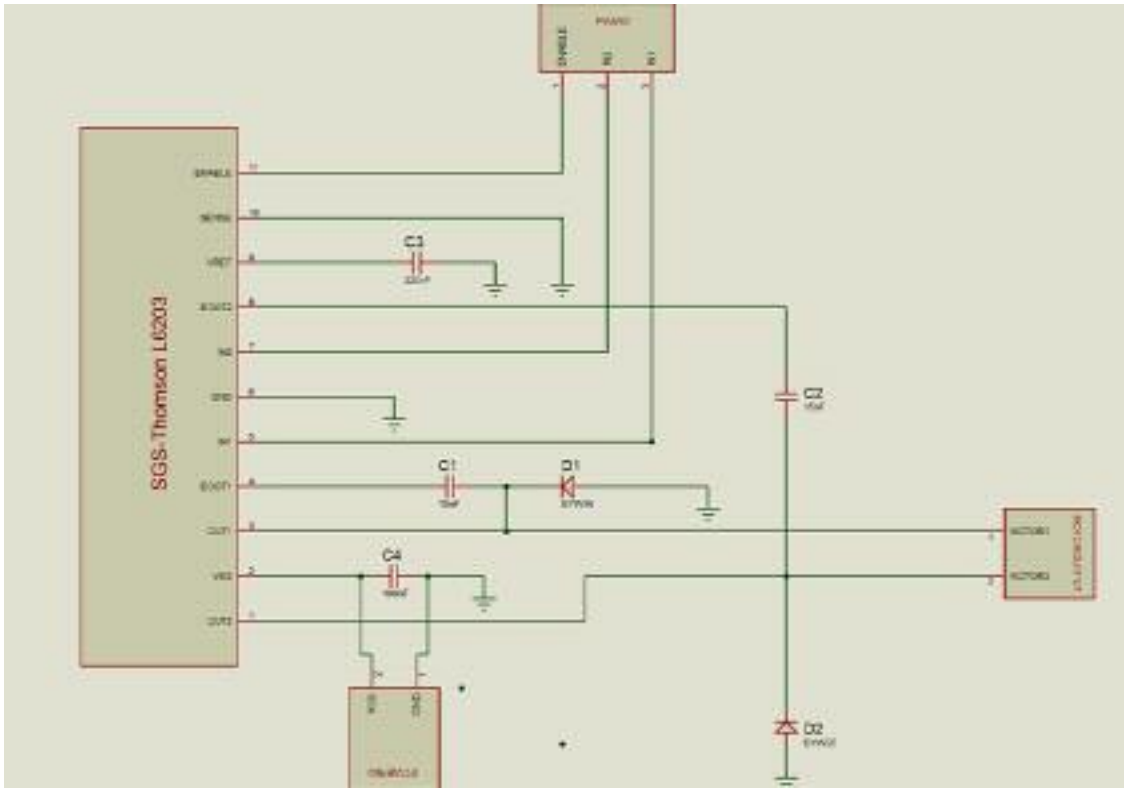
The table 4-4 demonstrates the different possibilities of the signal and the resulting behaviour of the motor.



Picture 6-3, schematic of the h-bridge and the µC inputs necessary

Enable	IN1	IN2	S1	S2	S3	S4	Motor
1	0	1	1	0	0	1	Turns right
1	1	0	0	1	1	0	Turns left
0	X	X	0	0	0	0	No resistance
1	0	0	1	0	1	0	Brakes
1	1	1	0	1	0	1	Brakes

Table 6-1, all possibilities of signals for the h-bridge, X -> don’t care



Picture 6-4, schematic of the motor electronics board

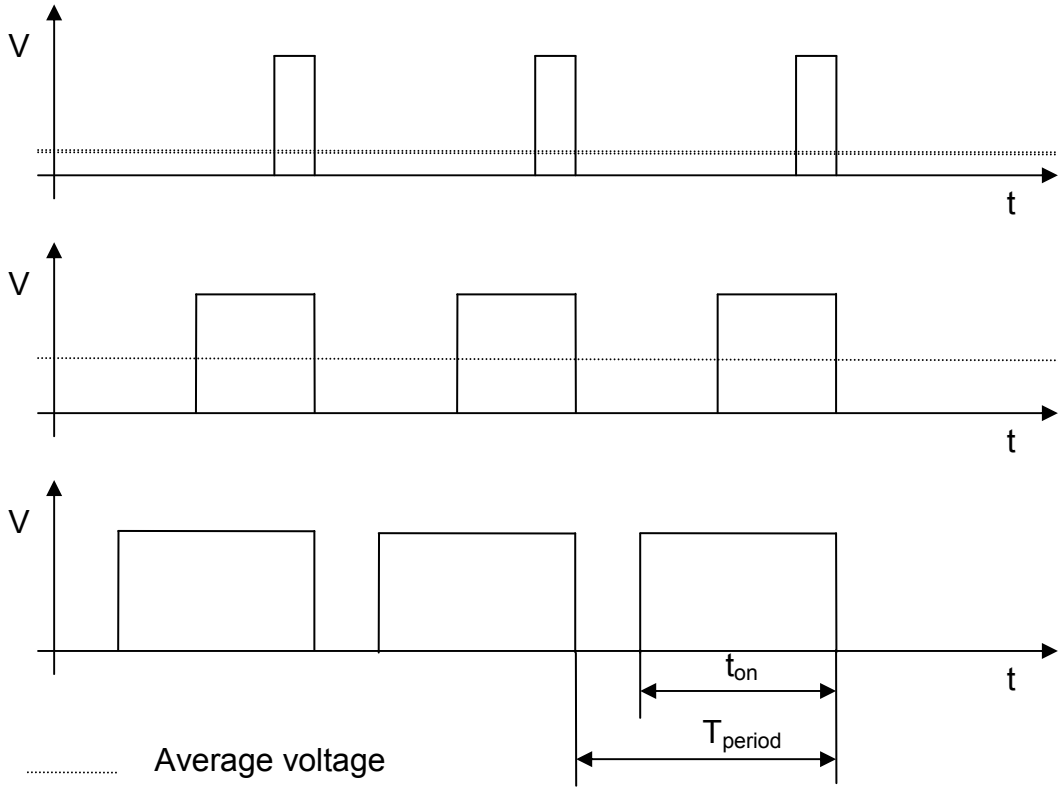
6.2 PWM signal generation

A PWM signal is generated by using switches IN1 and IN2 (or S1 to S4) in such a way, so that different voltage averages between 0V and V_{max} are generated.

In practice, this is achieved by connecting Enable and one IN signal to GPIO (general purpose i/o) pins and connecting the other IN signal to a PWM unit (or CAPCOM capture compare unit which can be used to generate a PWM signal) on the µC.

Care has to be taken that the PWM signal's frequency does not exceed the maximal input frequency of the parts or IC used or the signal will just be full speed.

One should note that frequencies between 20 and 15 to 20kHz may result in vibrations that can hearted by the human ear.



Picture 6-5, pwm signal explanation, voltage at the motor electronics inputs

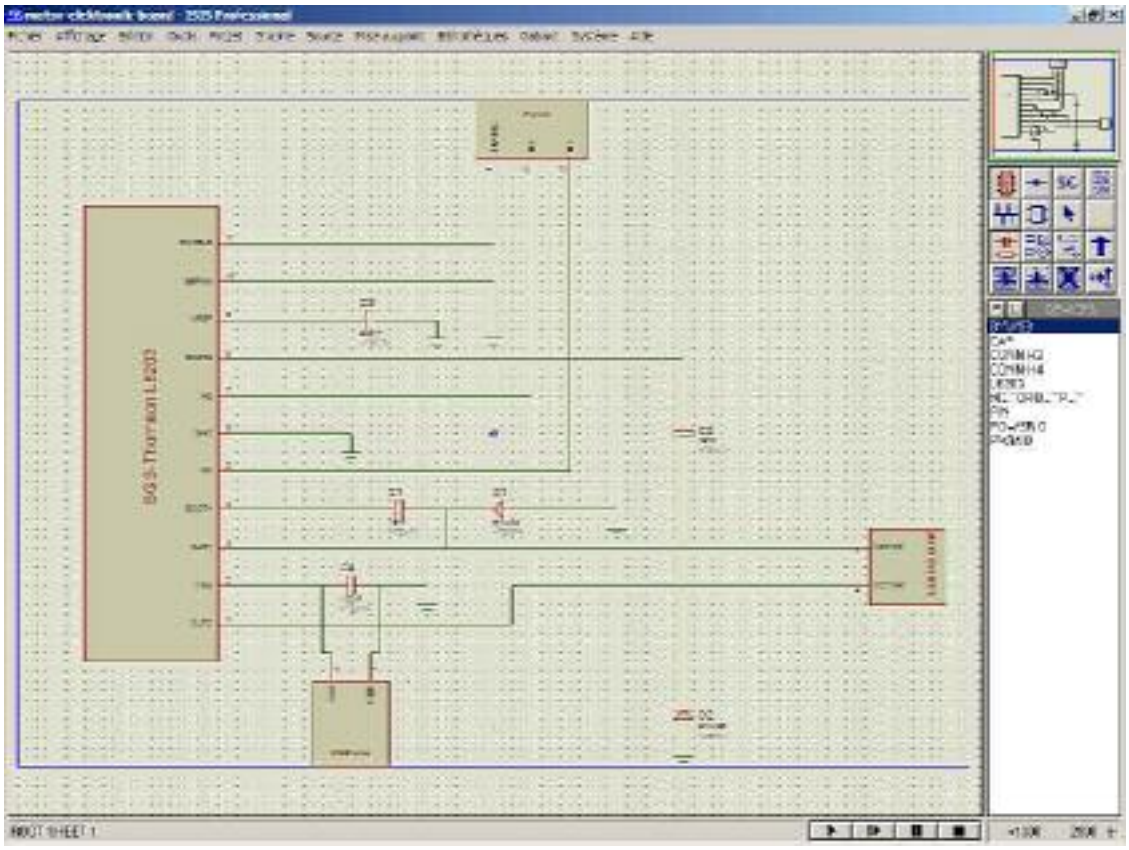
PWM signal relation = $\frac{t_{on}}{t_{period}}$

A special IC was used in this project, a SGS-Thomson L6203. For further information please consult this IC's documentation [11, SGS-Thomson L6203 manual].

7 Working with Proteus, Isis and Ares

Proteus, which it's two parts ISIS (used for electronic schematics) and ARES (Advanced Routage and Editing Software) is used for the development of electronic circuits.

7.1 Isis



Picture 7-1, ISIS screenshot

Usage of Isis is not easy to learn at first, especially if you have never before worked with a electronic CAD. The French language version with its own electronic terms did not make things easier.

The settings are not always were the user expects them to find, the program is not really user friendly.

Nevertheless, after some work done the logic of this program becomes better and when used together with its companion Ares one gets accustomed to it.

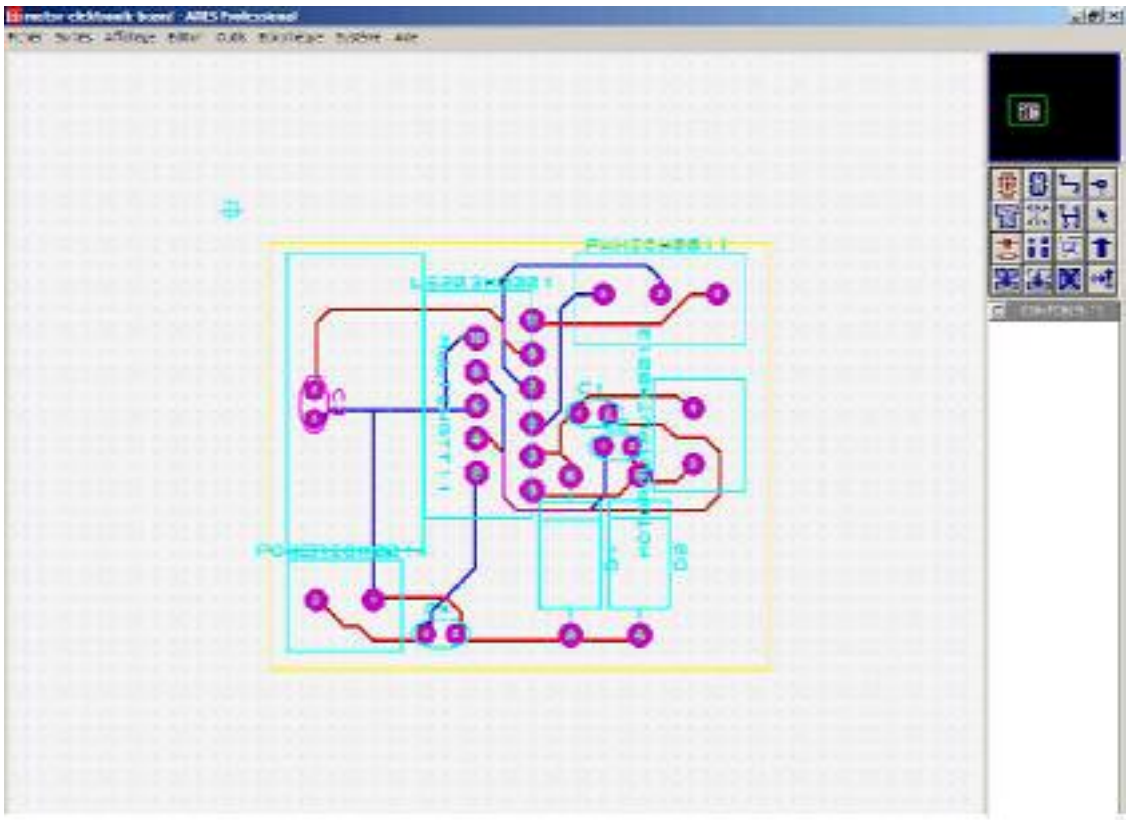
Once the circuit is developed in Isis, it can be processed in Ares for the routage and layout.

Isis helps the user in creating the schematic by supplying a large library of electronic parts. Even µCs are included.

Once the schematic is ready the user can simulate the circuit and test for errors if pins were correctly declared during the design phase.

Pin can be assigned to Input, Output, I/0, Power etc. to that connections can be tested if they are possible or if they will make some parts disappear in a cloud of smoke.

7.2 Ares



Picture 7-2, Ares screenshot

Ares usually has a black background, but for printing purposes this was changed here

As stated before, the similarities between ARES and ISIS are obvious. Like ISIS, ARES comes with part libraries

8 Microchip IDE Mplab, PIC C compiler from Hi-Tech

8.1 Problems with Mplab and the ICD

Mplab is the IDE supplied for free by Microchip, the manufacturer of the PIC family of µPs and µCs.

While this development tool is very well suited for programming assembler and writing the object code to a µC with a eeprom writer, the integration of the HiTech PIC C compiler C (which is available for the small 16F84 for free for non-commercial development and evaluation), the debugging tool ICD (In Circuit Debugger) and the serial programming by the ICD is sometimes tricky. Maybe newer versions of Mplab will improve this situation.

There are a few things (gotchas) to consider when using Mplab:

The update to V5.61 from V5.5 of Mplab removed the bug that the IDE always lost all information about the ICD when ever the node configuration was opened, which was really annoying. Today Mplab V5.70 is available, upgrading is highly recommended.

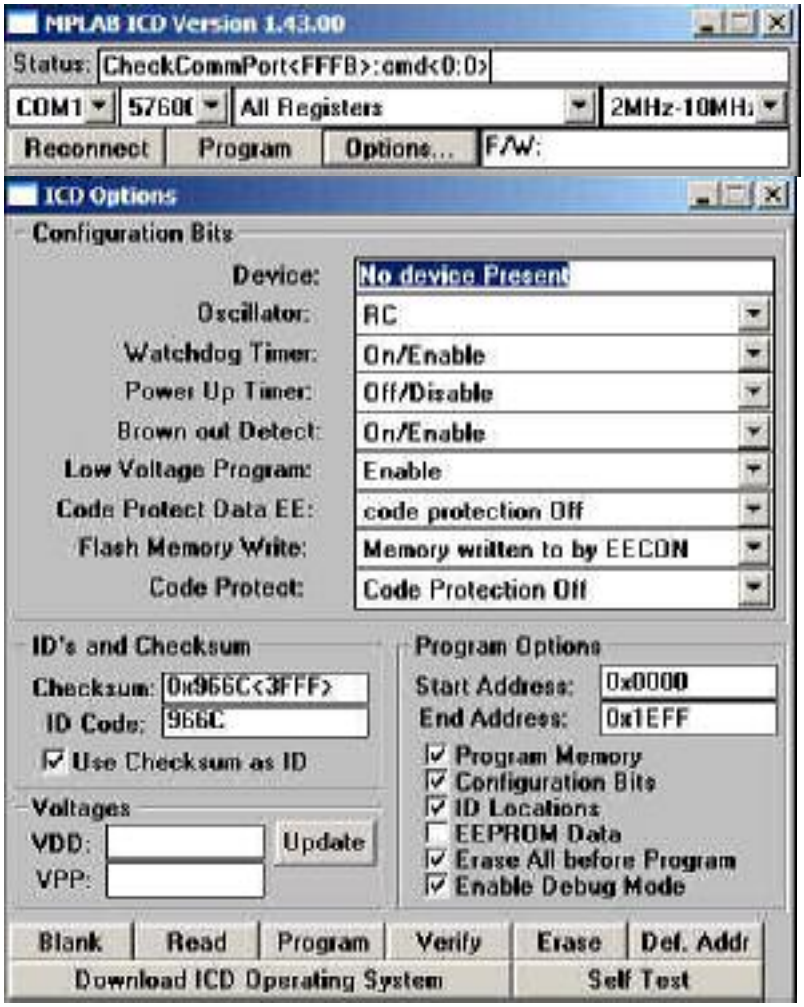
Other annoying things are that file names can only be 8 characters long because Mplab is still a 16bit Windows application. But that will change when V6.0 of Mplab will be released.

There is no syntax highlighting available, so a stray comment in a source file can silently comment out the rest of your source.

It may happen that the IDE will not ask you to save your edited source file on exiting the program. So saving your files before exiting is highly recommened, especially if you do not want a whole day’s work to vanish in an instant.

The ICD window is always visible if you use the ICD in your project, it can not be closed which takes precious space on your screen that is needed if several watch, trace, option and edit windows are open during development.

Reseting the µC via the Mplab IDE and the ICD is only working if the chip has been programmed in "Enable Debug Mode".



Picture 8-1, ICD's main and option window

The following options were set in the node options dialog

Informational messages: Verbose

Warning level: On 3

Generate debug info: On

Assembler Optimizations: On

Global Optimizations: On 3

Include Search path: On CD

Error file: On

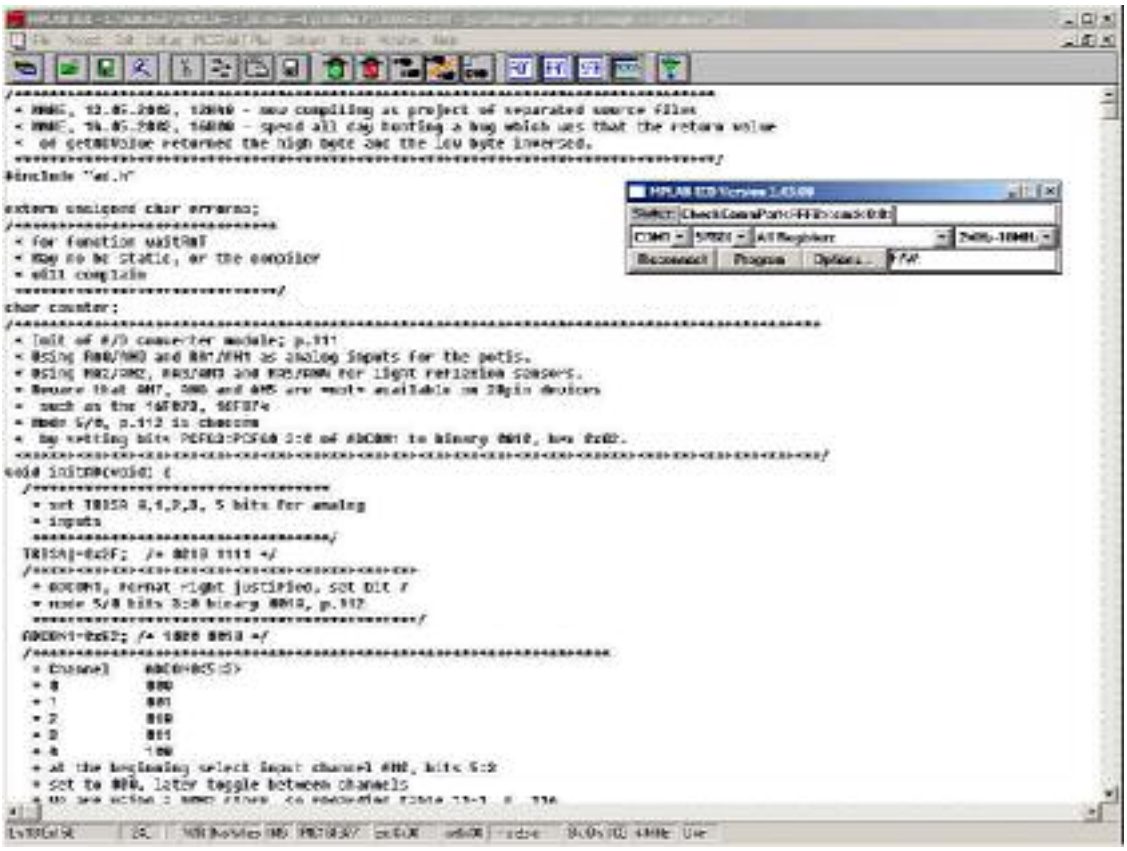
Produce assembler list file: On <-- This is needed for debugging of c code in Mplab

Compile for MPLAB-ICD: On

Strip local symbols: On

Description					Data
Informational messag	Quiet	Verbose			
Generate debug info	On				
Floating point for doul	24-bit	32-bit			
Fast math library for	On				
Printf library	Long	Float			
Create map file	On				
Display complete mes	On				
Output file format	Intel	Motorola	UBROF	Binary	
Error file	On				
Append Errors to file	On				
Compile for MPLAB-IC	On				
Specify offset for ROM	On				
Specify external ROM	On				

Picture 8-2, node options dialog



Picture 8-3, Mplab's main window

8.2 Problems with HiTech's PIC C compiler PICC

There is no feature to detect code that is not reached (dead code), but one can use lint to check for such errors.

The integration of the HiTech PIC C compiler can be improved if the following things are done, these tips are taken from HiTech's web site [4, http://www.microchip.com], but parts of this information can also be found the the PICC manual [12, PICC-manual.pdf].

Error messages can not be used to jump to the error's location

Q: When I compile a PIC program using MPLAB, the error messages from PICC appear the in MPLAB build results window, but if I double click on the message, MPLAB doesn't jump to the error location. How do I fix this?

A: The default error format from PICC is not what MPLAB wants. You can alter this by setting some environment variables. These are:

HTC_ERR_FORMAT=Error[000] %f %l : %s

HTC_WARN_FORMAT=Warning[000] %f %l : %s

How to get MPLAB to display compile errors

Q: When I compile an MPLAB project, I get the message: "MPLAB is unable to find output file "XXXX.OBJ". This may be due to a compile, assemble, or link process failure. Build failed.", but no errors are displayed. How do I know what is producing the error?

A: MPLAB displays any errors after it attempts to compile. These messages are read from error files that the compiler must produce. You need to turn on the "Error file" option for each source node in the project and the HEX link node. In the data field for this option enter the name of the source file with the extension ".err". So if you have a source file called "main.c", then in the node properties for the node, you should specify an error file of main.err. If the HEX node has the same name as one of the source modules, then turn on the "Append Errors to file" option and enter the error file as indicated above.

Local variables in MPLAB

Q: How do I view local variables in MPLAB?

A: To be able to view local varibles in MPLAB, be sure to compile your project with the option "Generate Debug Info" for each node. Also add to the HEX file node the additional command line option –FAKELOCAL. Local variables will then be seen in the format: function_name.local_var So for example, if you had a local variable called "number" and it was within a function called "testing", then it would appear in the symbol list as "testing.number"

Note that from experience, if setting –FAKELOCAL will not help, then a restart of MPLAB may help in these situations.

Incremental Compiles with Hi-Tech C

Q: Under Hi-Tech C and MPLab, every time I recompile, it recompiles everything and then links it. How do I do incremental compiles?

A. Add the line 'c:\ht-pic\include' under 'include path' in the 'edit project' dialogue to enable incremental compiles. You must already have the 'language tool' under the root node set to 'PIC C Linker', not 'PIC C compiler'. Add each source file to the list to be compiled and then linked.

8.3 Using the PICC

The PICC is a full featured ANSI C compiler without support for recursive functions. But these are not really needed with only 8 stack levels (only 7 when the ICD debugs the µC) and limited memory.

Once the PIC Compiler, Linker and assembler are installed using edit->install language tool, the compiler and linker can be used in the node options.

Picture 8-4, language installation tool

9 ICD and ICD interface

The ICD (In Circuit Debugger) is a tool which can be used for limited debugging of the PIC 16F877. In normal use, and when used with common sense, its capabilities are more then enough for most uses.

Picture 9-1, the DB9 connector is the ICD interface

Its features and tradeoffs are:

- In circuit debugging which one breakpoint, one trace and watch windows for variables and a complete memory map if needed.
- Source level (even works with C code) in asm or object code debugging.
- In circuit serial programming, no need to remove the µC from its socket for programming any more.
- RS-232 interface, but it sometimes does not work with USB serial converters !
- You lose 3 pins on port B (bad because 2 of them have interrupt on change options). These are RB3, RB6 and RB7
- You lose one stack level
- You lose memory from 0x1F00 to 0x1FFF as well as 6 general purpose file registers on the µC.
- Sometimes, if you built your own (longer) cable to your own circuit,

transmit errors will occur, therefore always verify the code downloaded to the µC before trying to debug and go hunting for bugs. As shielded cable is therefore the best solution if the cable length exceeds 200mm or if you use speeds > 19,2kBaud.

-

Alternatively there is also the option to use the ICD2, with more features, which is will support the 16F877 in the near future with Mplab V6.

Connection to the ICD module is done by a self-fabricated cable.:

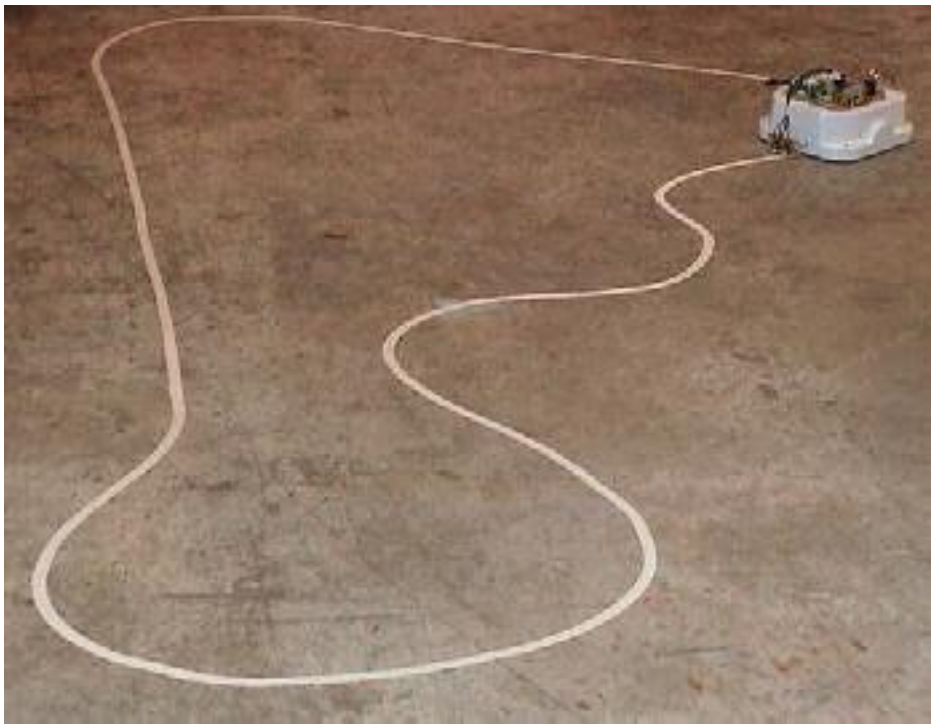
DB9 on own target board			RJ11 ICD module		
Color	PIN	Signal	Color	PIN	Signal
white	1	VPP	white	1	VPP
green	2	RB7	black	2	VDD +12V
yellow	3	RB6	red	3	GND
blue	4	RB3	green	4	RB7
NC	5	NC	yellow	5	RB6
red	6	GND	blue	6	RB3
black	7	VDD +12V	NC	NC	NC
NC	8	NC	NC	NC	NC
NC	9	NC	NC	NC	NC

Table 9-1, pinout of the ICD connection cable

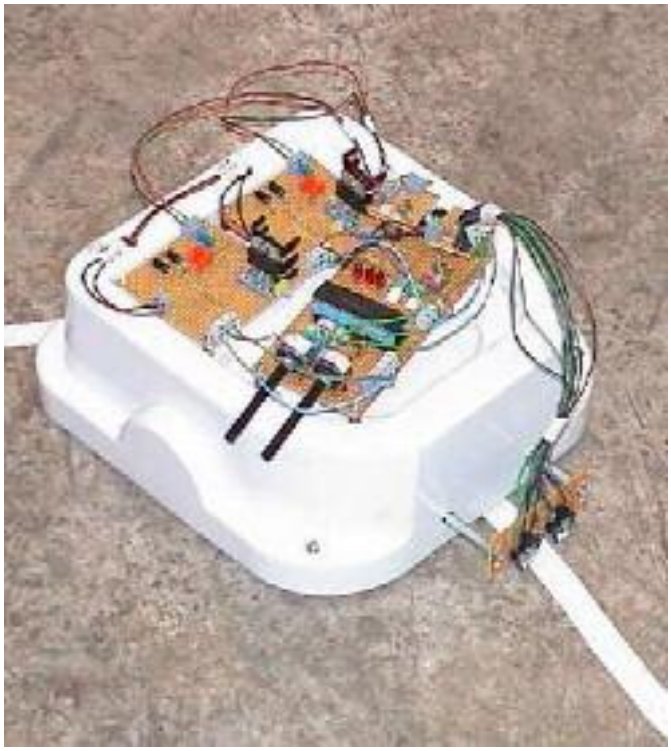
For more information consult the ICD documentation [13, ICD.pdf] and [14, ICD-setup-poster.pdf]

10 The robot crusing around

10.1 Pictures of the robot following the scotch tape



Picture 10-1, the robot on its course

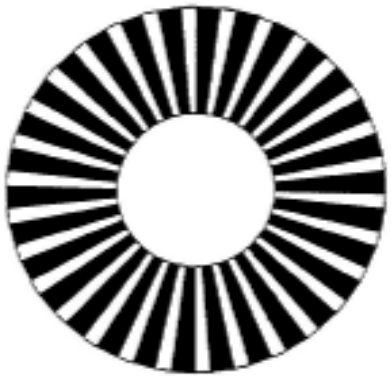


Picture 10-2, close view of the robot cruising

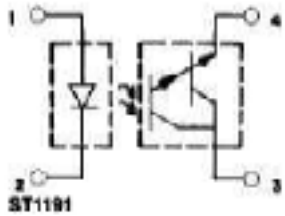
10.2 Quadrature Encoder Input

With one flexible disc containing white and black areas attached to each motor. And with two sensors consisting of of a photodiode and a phototransistor a quadrature encoder can be build for each motor which makes it possible to detect and measure speed and direction.

Picture 10-3, motor with encoder discs



Picture 10-4, the encoder disc



Picture 10-5, schematic of H21B

Due to the limited number of pins on the 16F877, a software solution was tried here by connecting signal A of each encoder to to GPIO pin with an interrupt on change feature thus making it possible to process the change of speed and the sense of direction in the interrupt service routine. However, this is suboptimal as the following calculation shows.

Speed V_{max} of the robot: V_{max}= 2 m/s

Encoder disc has 30 black and white areas: $n_{encoder}$ = 30

Circumference of one wheel: u_{wheel} = 1/6 m

Two wheels n_{wheel} = 2

Worst case for the time $t_{nextEncoderInterrupt}$ after which the next encoder interrupt must be processed.

$$t_{nextEncoderInterrupt} = \frac{u_{wheel}}{n_{encoder} * n_{wheel} * v_{\max}} = 1{,}38ms$$

Formula 10-1

This calculation implies that the encoder signals have a exact phase shift of 90°! If this is not the case, e.g. if the phase shift is only 60°, the time $t_{nextEncoderInterrupt}$ is only 1,38ms*2/3=0,92ms !

With the µC at f = 4MHz one instruction cycle takes

$$tperiod = 0{,}25 * 10e-6s * 4 = 1\mu s$$

Formula 10-2

Because one instruction cycle takes four Q cycles [15, PIC midrange MCU family, midrange.pdf]

So the number of instructions n_{isr} in the ISR (interrupt service routine) processing the encoder interrupt may not exceed:

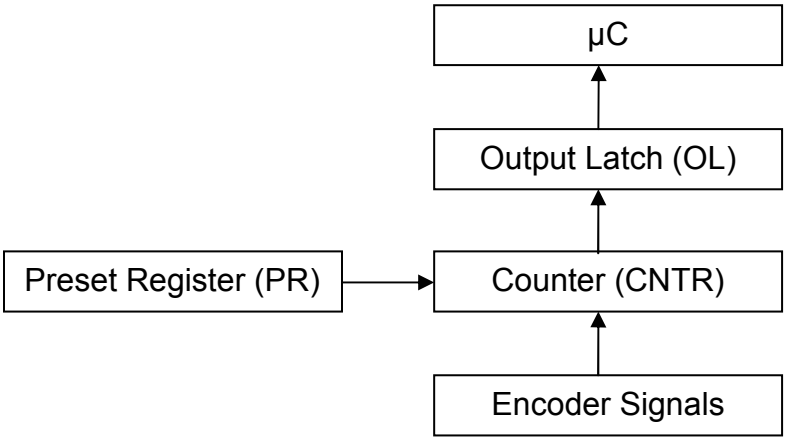
$$n_{isr} = \frac{t_{nextEncoderInterrupt}}{t_{period}} = 1380$$

Formula 10-3

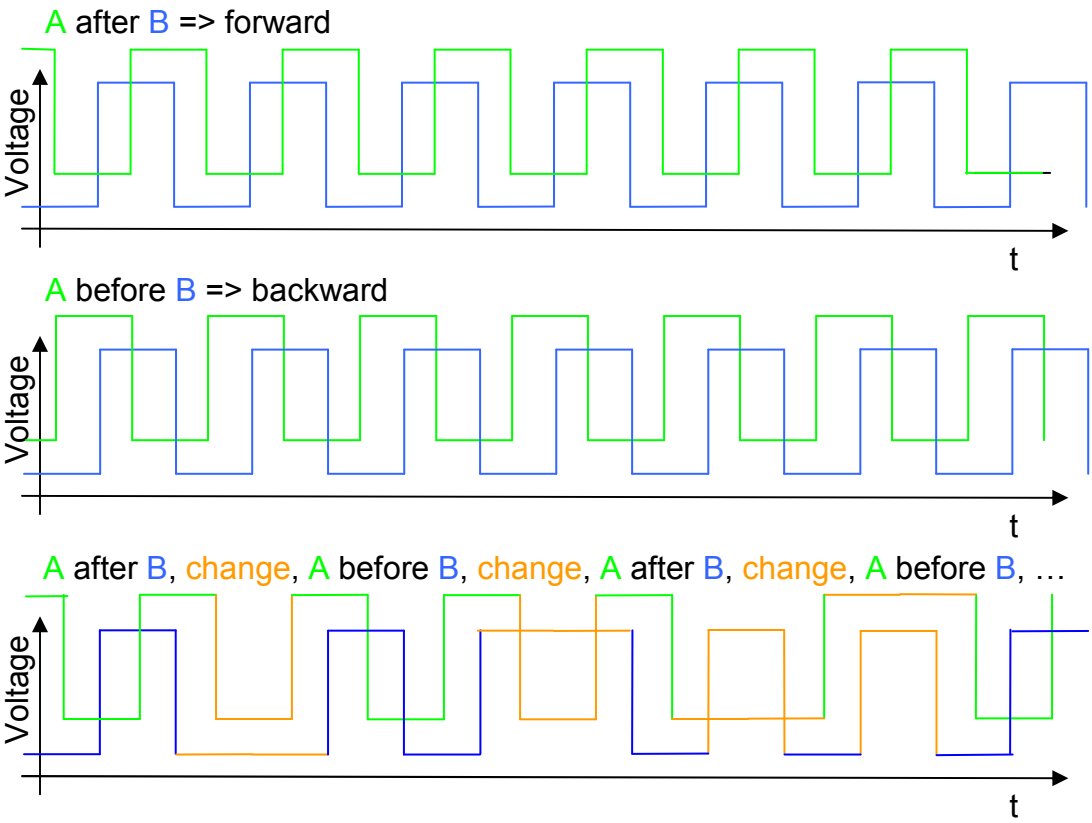
While this seems a lot a first, one must consider that the µC must still process routines for PWM generation, timer event, adc operation and other sensors. If one interrupt it lost because the µC is processing another interrupt and the GIE is disabled, an encoder signal will be silently discarded and the change in position or speed will no be noted.

The correct solution would be to use a dedicated µC or IC for encoder data processing. This solution is, however, not feasible on the 16F87X because this µC does not have an external bus to address external memory. Maybe the I2C bus could be used. Commercial products use special hardware that usually looks similar to Picture 10-6, for more information on encoders see [16, servotogo, servo i/o card hardware manual].

If the 20MHz version of the PIC16F877 would be used, this calculation might look 5 times better, so the software approach might be possible then.



Picture 10-6, commercial encoders



Picture 10-7, signal examples

10.3 Speed and position control

Using the encoder as outlined in 10.2 Quadrature Encoder Input gives the speed and direction of the robot with twice the resolution one encoder can

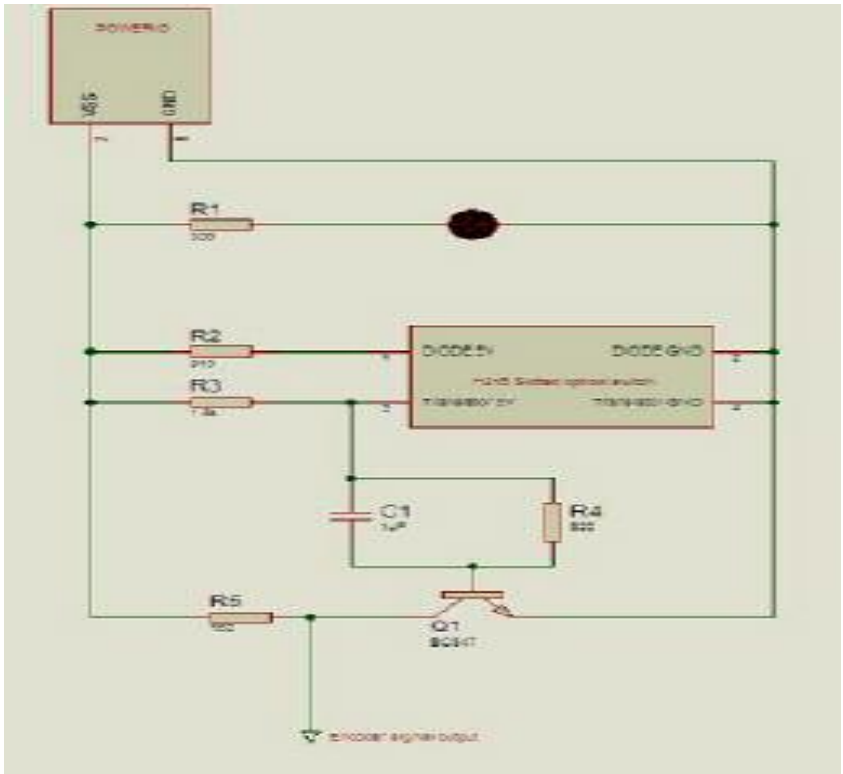
provide. The resolution is only twice and not four times higher as normal because of the limited number of pins with interrupt on change feature, only one signal of a encoder, lets say A, can be connected to such a pin. The other must be connected to a normal input.

Position control as such is not possible using this configuration if no integrator is used to integrate the speed information thus gaining position. However this approach might be too costly in terms of cycles on a 8 bit µC. And to have an appropriate distance before the counter overflows, two registers must be cascaded.

With n_{wheel} and u_{wheel} as well as the information that resolution is doubled the resolution 1/360m. With an 8 bit counter the register will overflow after roughly 710mm. With two cascaded 8 bit counters the register will overflow after 182m.

10.4 Encoder signal circuit

With the supplied optical switch H21B [Picture 10-5, schematic of H21B] an encoder circuit can be build, note that 4 of these circuits are needed (2 wheels with to sensors each).



Picture 10-8, encoder circuit schematics

The LED on top of the circuit can be removed if not indicator is needed.

C1, R4 and Q1 amplify the signal from the H21B, the capacity's charge is sufficient to maintain logic TTL levels up to very low speeds. When the capacity is empty the TTL level will drop to ~4V which should still be sufficient to hold high level.

Signal edge	Signal state	Direction
A high->low	B low	forward
A low->high	B high	forward
B high->low	A high	forward
B low->high	A low	forward
A high->low	B high	backward
A low->high	B low	backward
B high->low	A low	backward
B low->high	A high	backward

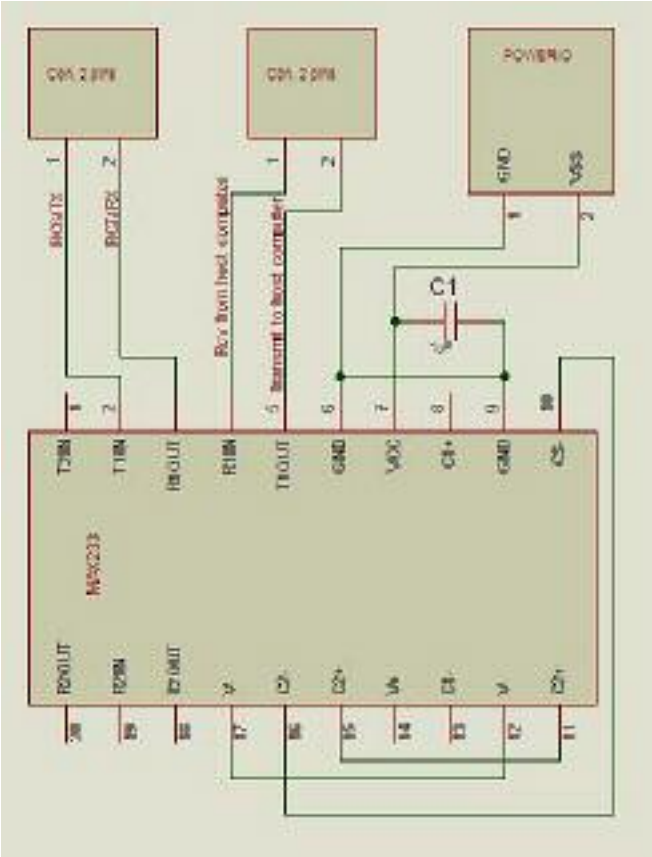
Table 10-1, signal states

11 Serial port on the PIC 16F877

11.1 The serial interface circuit

Because the serial port on the PIC16F877 can not supply the necessary voltage level of 12V for RS232 operation, a special IC is required.

A IC of the industrial standard family of MAX, the MAX233 is used here because of its ease of use. The MAX233 only needs one capacity on the periphery. The MAX233 features two serial ports, but only one is used here.



Picture 11-1, schematic of serial port circuit

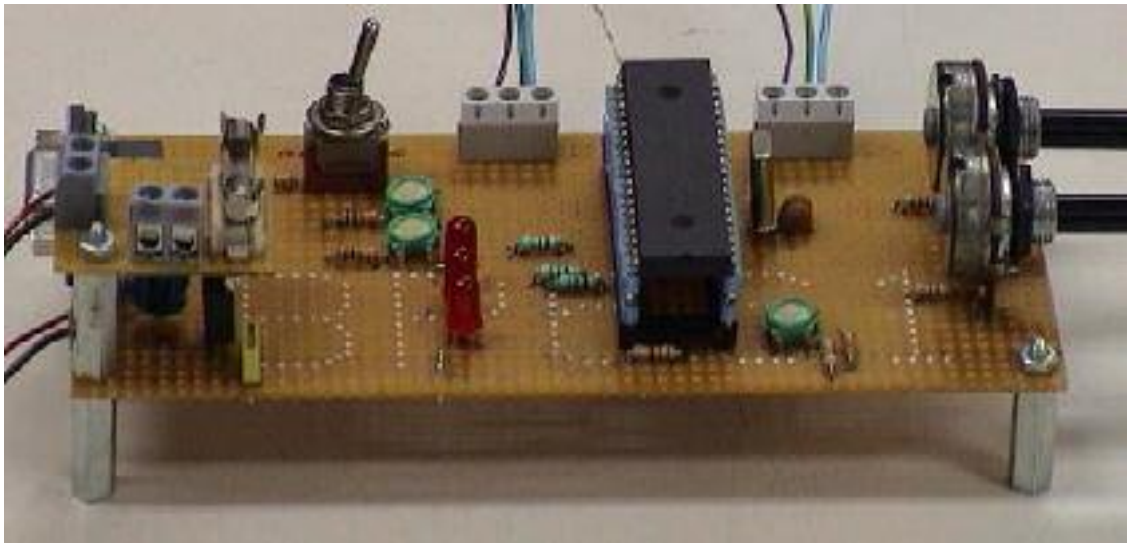
The first connector on the upper left is used to connect the pins RC6/TX and RC7/RX from the µC. The second connector on the upper middle connects to the host PC. The third connector on the upper right connects to +5V and GND.

For more information see the MAX233 documentation at [17, MAX220-MAX249.pdf]

12 Schematic of the main board, PIC16F877, 2x PWM signals

12.1 The main board, photo and description

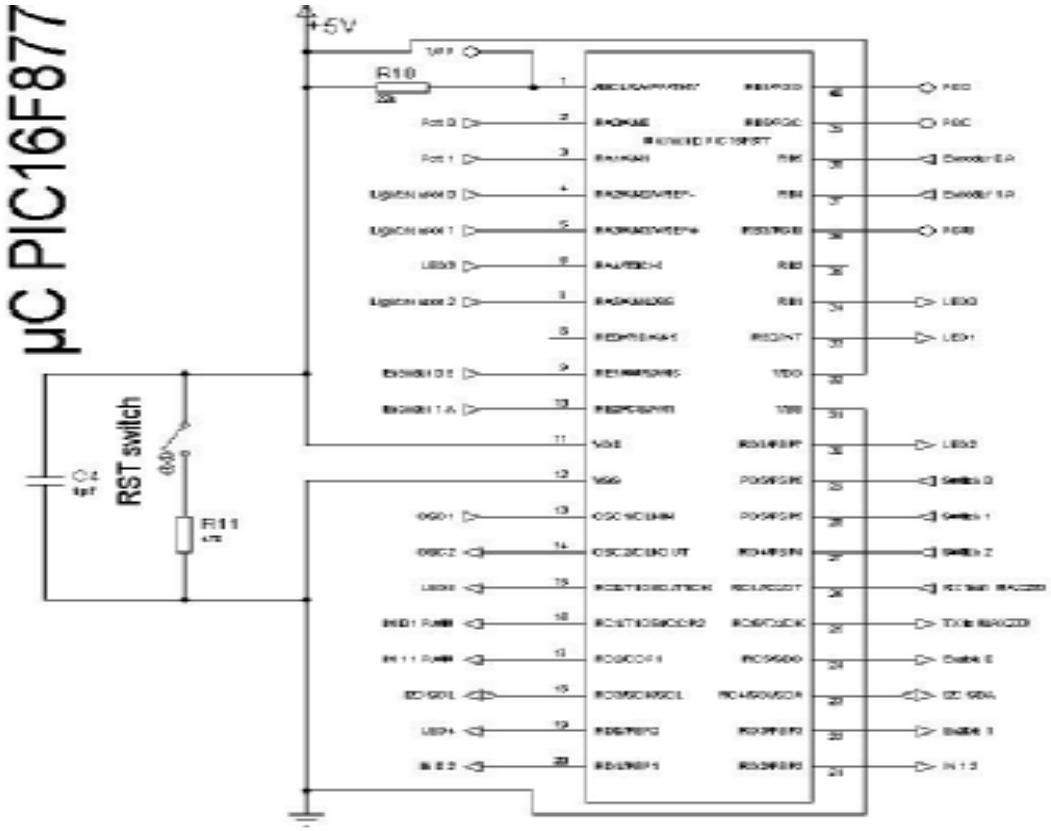
The main board consists of the µC and some of its periphery.



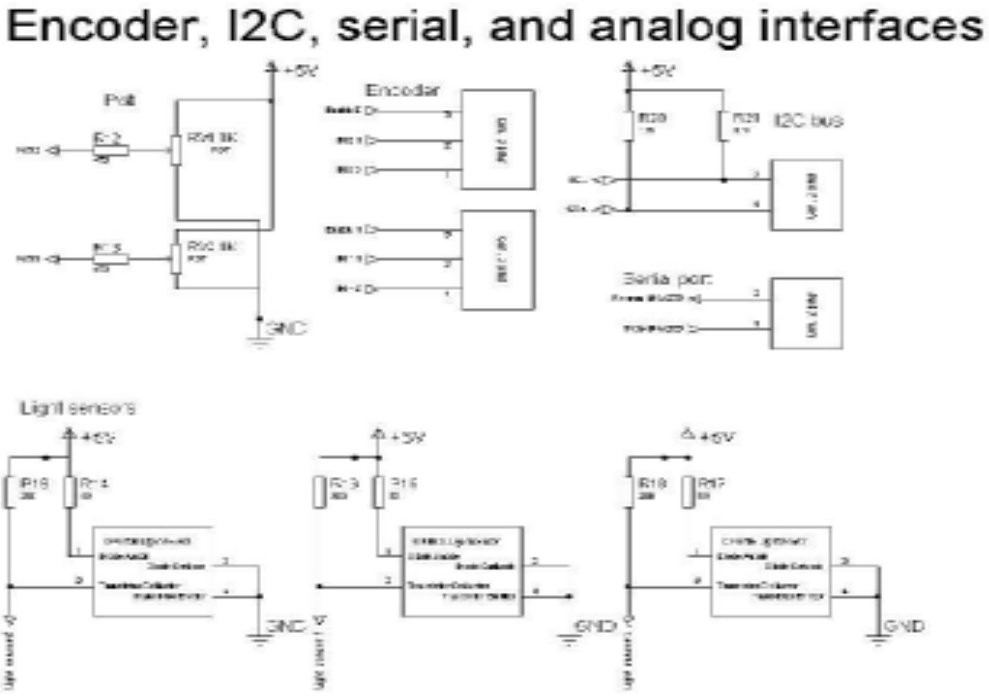
Picture 12-1, the main board

The power supply can be seen on the daughterboard on the lower left corner. Some switches and leds are left to the µC. The µC itself is easily spotted in the middle. The potis are to the right.

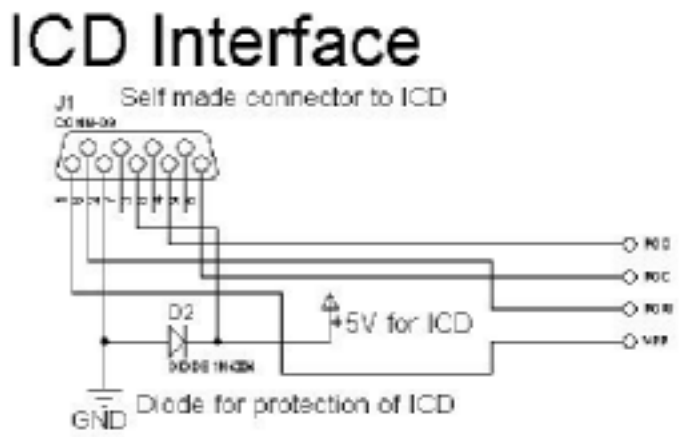
12.2 Schematics of the circuit of the main board



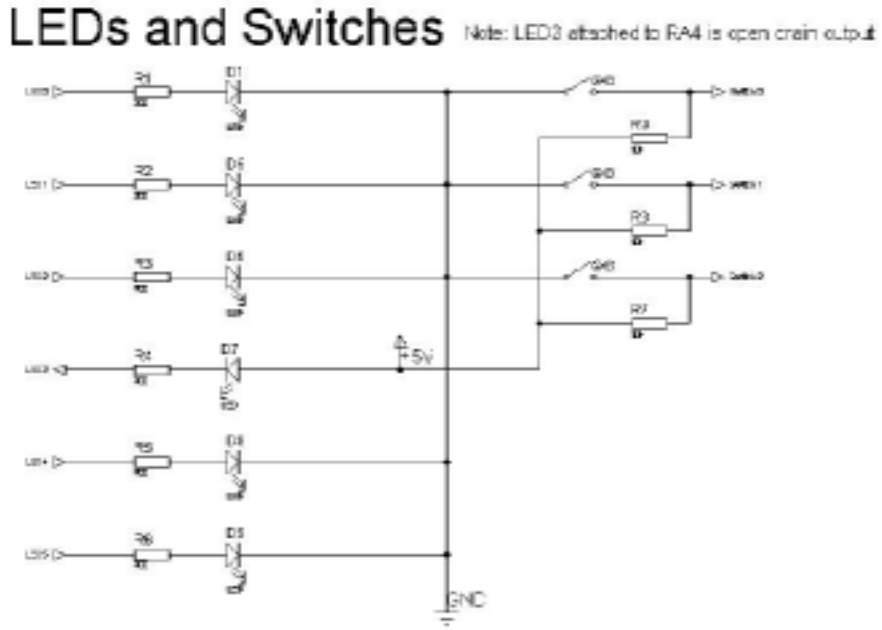
Picture 12-2, the µC



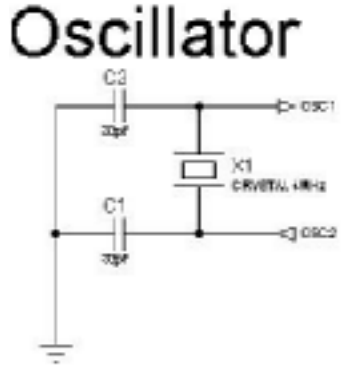
Picture 12-3, encoder, i2c, serial and analog interfaces



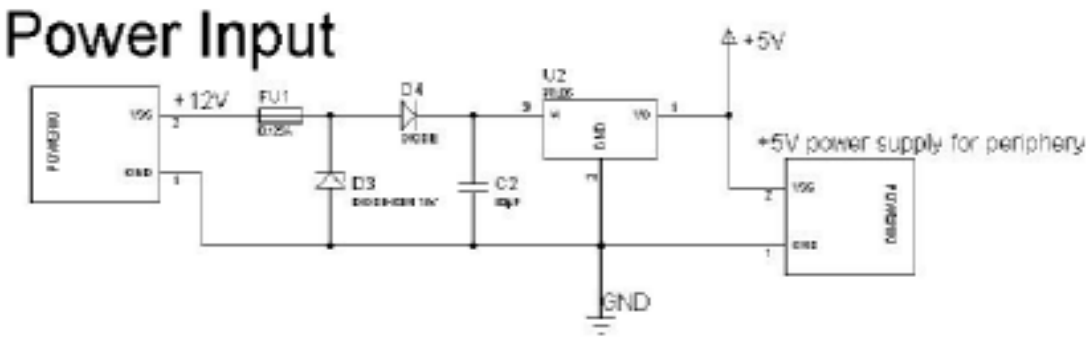
Picture 12-4, the ICD interface



Picture 12-5, leds and switches



Picture 12-6, oscillator



Picture 12-7, power supply

12.3 OPB704 light sensors

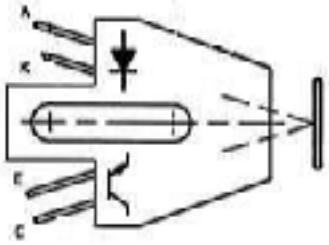
The OPB light sensors are used to detect the reflection of the light emitted by a photodiode by a phototransistor to follow the white scotch tape.

Detection is not easy because the underground may vary. A carpet can be quite easily detected; it almost reflects a light at all. The ground made of stone is almost identical to the reflection of the white scotch tape.

Therefore a capability to teach in ground and scotch was added to the robot's µC to be able to drive on different surfaces without reprogramming. The detection level, that means the level between, the analog values for ground and scotch, can be set in software or stored in the µC's eeprom. Currently the detecting level is set at 1/3 of the different between scotch and ground. This value has been gained from experience and usually works best.

When mounting the OPB704 sensors are oscilloscope is highly recommend to adjust the high of the sensor and the angle, which should be 90° to the surface. This adjustment can really make the difference when cruising around. When adjusting, simply try to get the biggest difference in the input voltages of scotch and ground possible. The software does the rest.

Picture 12-8, OPB704



Picture 12-9, OPB704

12.4 The battery

The battery is a 12V, 1.2Ah lead-acid one. It can be charged with a normal power supply in approx. 14h why connecting it to 13V, 120mA. The current will drop to 0A when finished. It is not allowed to change the battery type for the competition. However, it is allowed to have as many of them in reserve as wanted.

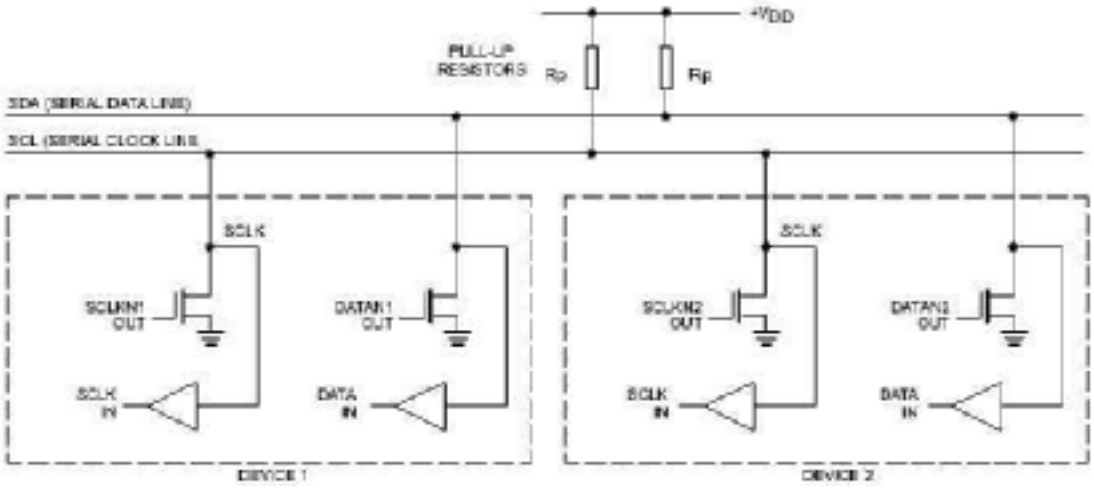
Picture 12-10, the battery

13 I2C bus of the pic 16F877

13.1 The I2C bus

The I2C bus (Inter-IC bus or I^2C) is a bus used on circuits. It is not a fieldbus like CAN or Interbus. Many IC, like LCDs, serial eeproms, remote i/0 ports or ram have I2C interfaces. Philips alone manufactures more than 150 different types of ICs with I2C connector.

It only needs to wires with a 1,7kΩ pull-up. The lines are SCL, Serial Clock Line and SDA, Serial DAta line.



Picture 13-1, i2c bus example

For more information consult the I2C specification [18, I2C_BUS_SPECIFICATION.pdf]

13.2 The compass and ultrasound distance sensor

IUT acquired a compass and an ultrasound distance sensor, both with I2C interfaces, for better navigation of the robot. Perhaps care must be taken with ultrasound detector during the competition because modern photos and camcorders use ultrasound distance sensors for the autofocus.

14 Conclusion

The principal goal to develop software, electronics and conceptions for a mobile robot was achieved. The robot is reliably following the white scotch tape on almost any ground, including the floor seen on Picture 10-1.

In the competition, even tough ambient light level will be much higher; the robot will have no problems, because the carpet reflects almost no light. But the light detecting sensors should then be mounted inside the chassis and there height should be regulated by a mechanical device, perhaps a spring.

Following students working on this projects will have to design at pcb motherboard with connectors for sensors and µC boards so that development can be split among several people (4 students will form a team for the competition). This might then perhaps look like a today's pc motherboard.

Perhaps not only one, but several µC can be used to perform different tasks. One could be responsible for the two encoder channels, one for the light sensors, one for coordination and communications, one to avoid collusions with the second robot with the ultrasound sensors and some bumpers mounted around the robot, another µC could be used for pwm signal generation. They would then be able to share information and send commands by their on chip I2C bus. This also makes cooperation between different students easier if well defined interfaces are established.

15 References

[1] rules.pdf

[2] Base_roulante.pdf

[3, 16f84a.pdf]

[4, PIC16F62X manual]

[5, http://www.microchip.com]

[6, ex001.asm]

[7, ex002.asm]

[8, pwm001.asm]

[9, PIC16F87X manual]

[10, pwm_f877.asm]

[11, SGS-Thomson L6203 manual]

[12, PICC-manual.pdf]

[13, ICD.pdf]

[14, ICD-setup-poster.pdf]

[15, PIC midrange MCU family, midrange.pdf]

[16, servotogo, servo i/o card hardware manual]

[17, MAX220-MAX249.pdf]

[18, I2C_BUS_SPECIFICATION.pdf]

16 Index of tables and pictures

17 Annex

17.1 Resistors

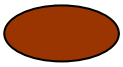
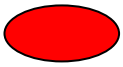
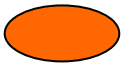
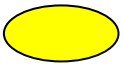
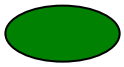
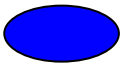
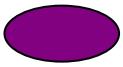
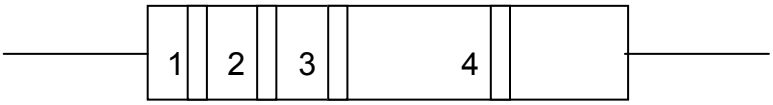
0	1	2	3	4
				
Noir	Brun	Rouge	Orange	Jeune
Ne	**mangez**	**rien**	**ou**	**jeunez**
voila	**bien**	**votre**	**grosse**	**bêtise**
Vert	Bleu	Violet	Gris	Blanc
				
5	6	7	8	9

Table 17-1, French mnemonic for resistors



Colour	Value	Value *10	Multiplicator $*10^{Value}$	Tolerance
Black	0	0	0	
Brown	1	1	1	1%
Red	2	2	2	2%
Orange	3	3	3	
Yellow	4	4	4	
Green	5	5	5	0,5%
Blue	6	6	6	0,25%
Gray	8	8	8	
White	9	9	9	
Gold			-1	5%
Silver			-2	10%
Nothing				20%

Table 17-2, table of resistor colour codes