# Remote Speed Control of DC Motors

*by Jan Lichtenbelt Dec 2012*

## Abstract

**In this fantastically thorough article, Jan explains how you can use a PIC microcontroller to control DC motors remotely using Flowcode. With detailed explanations on all aspects of the procedure, this article is a must read for anyone wanting to remotely control DC motors.**

## Requirements

**Software:**

- Flowcode v5 for PIC

## Introduction

Remote control (RC) is used e.g. for model aircrafts, speed cars, ships and battery powered, or life steam trains. The principal is simple. A remote control transmits e.g. at 40 MHz the position of one or more handles to the receiver in the model. Depending on the position of the handles or rotation knobs a pulse of 1 to 2 msec with a repetition of 50 Hz will become available in the model. These pulses can directly be used for servo-motors, but must be 'translated' for DC motors into just *forwards*- (aircraft propeller), or *forwards/reverse*- (model train, ship) speed control.

There are already commercial RC receivers with PWM outputs. But this article shows the possibility to make the PWM from the 1-2 msec pulses.

A microcontroller can translate the RC duty time (5-10%) into a PWM of 0-100%.

There are small differences in requirements for the different type of models. Aircrafts and speed cars need a very fast response and model trains and ships has more natural, slow response. On a receiver failure, the aircraft should fly on, while a speed car should stop as soon as possible. Another failure problem is an overload current of the motor used. What to do in that case, depends on personal wishes.

The choice of the electronic bridge drive for the motor is one with discrete electronic parts and one with a commercial bridge drives BD6222 or the ADP3624. All does have to fulfill the maximum physical dimension available in this kind of hobby models. The L298 dual full bridge drive, used on the Matrix Multimedia EB-022 board has been rejected in the first place due to the fact that the dimensions are to large.

## The microcontroller

The requirements for microcontroller in case of RC speed controlled Motor(s) are:

- RC 1-2 msec pulses must be converted to PWM for motor speed control, see figure 1.
- The accuracy of the pulse length measurements must full the requirement of the accuracy of the PWM percentages.
- Motor control only 'forward' or both 'forward and reverse'
- Self learning maximum forward speed and, if applicable, maximum reverse speed
- Minimum power (PWM) required to start the motor (to overcome start-up friction)
- Minimum dead time in cases of reversing the motor direction
- Choice to leave the PWM unchanged *or* stop the motor during lost RC signals.
- Microcontroller motor current sensing to prevent overload current or bridge temperature, with auto shut-down and facultative auto restart.

Electrical requirements are:

- Bridge drive to control the motor, see figure 2.
- Preventing short circuit in the motor bridge drive (due to different time response of MOSFETs to falling and rising edges)

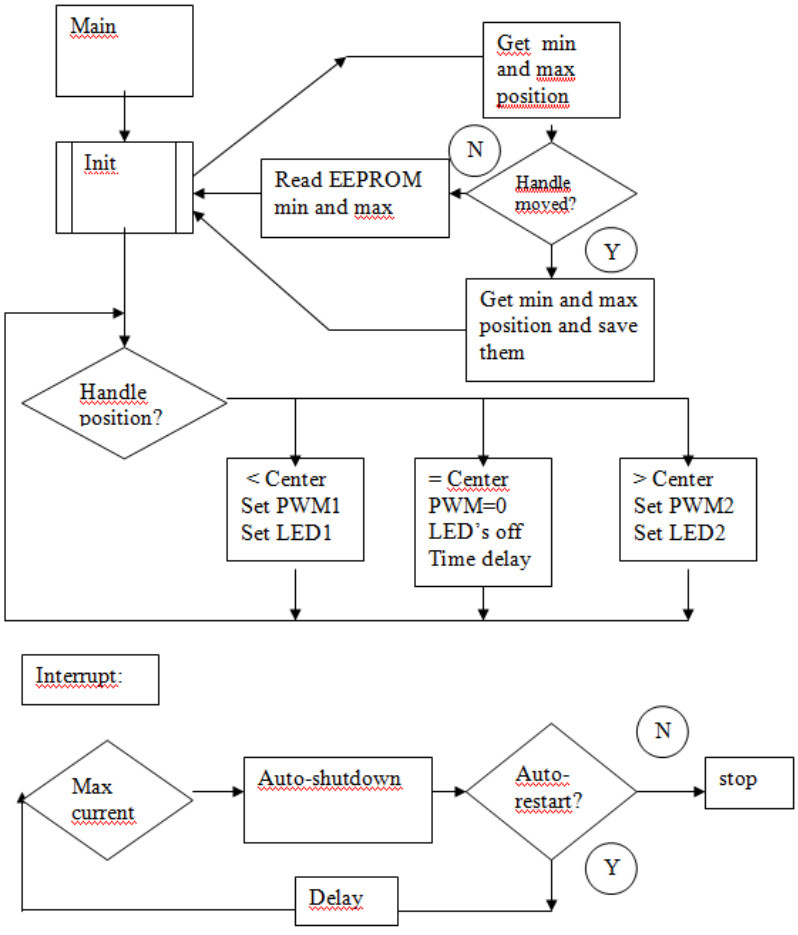



*Figure 1. Schematic Flowcode for RC conversion to PWM signals.*

Other requirements are:
1. The electronic board must be as small as possible. Therefore the 8-pins microcontroller PIC12F1840 will be used.
2. With this 8 pins microcontroller a LCD reading, during testing, is not possible (to less pins). RS232 is used to transmit the debugging data to another microcontroller to show these data.
Additional 2 LED outputs are foreseen, used for forward/reverse lights, showing the start-up procedure and making the auto-shutdown of the microcontroller visible.

Some details are:

1. Pulses are measured by means of Interrupt-On-Change (IOC), both on rising and falling edge of the pulse, on the RC-input pin.
2. No pulses during operation will leave the last PWM unchanged or stops the PWM output.
3. At start-up: If no RC-pulses are available, PWM's will not start.
4. At the start-up procedure within 3 seconds, the minimum and maximum handle positions have to be given. However, if the handle was not touched during this period, the previous values will be used, as stored in the EEPROM. Both forwards/ reverse LED's will be on showing the start-up procedure.
5. After 3 seconds the actual handle position is the centre/neutral value. This will be mostly the middle position for forwards/reverse motor regulations, or the minimum position for forward-only motor regulations. There is no PWM at this neutral position.
6. There are more possibilities to control a motor in a bridge with 2 PWM's, see figure 2. Here we use the most simple method of one PWM continuously low and the other with variable PWM. The reason to choose for this single PWM is the advantage that short circuit due to different rising times of falling and rising edges of PWM's, can not happen in this set-up (see details in datasheet of microcontrollers, chapter Enhanced PWM).
7. Both fast response of PWM on RC-input and slow response by filtering(damping) is possible.
8. Due to frictions, all motors need a minimum of power to start. A minimum PWM can be set, below the motor will not be activated. Pay attention: This value can be different for different type of bridge drivers and for different oscillator frequencies.
9. In the 8 pins configuration, during debugging one LED output is changed into a RS232 (Tx) output.
10. Maximum current or bridge temperature detection will disable power to the motor at overload. Facultative auto-restart is possible or manual restart by means of re-power the microcontroller.

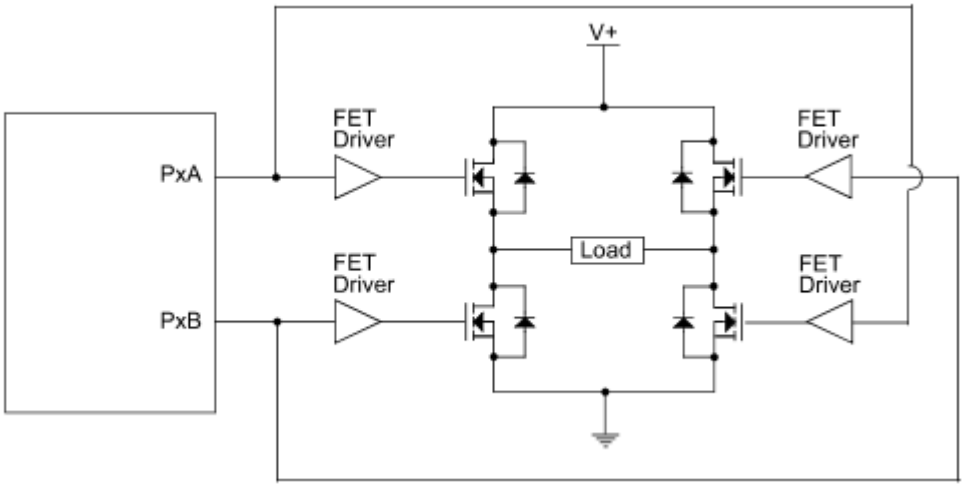



*Figure 2. Full bridge motor control (load=motor). P1A will be low and PWM on P1B, or visa versa, see e.g. datasheet 16F1939 [16F1939].*

## Timers and Accuracy

There are two types of RC control handles:

- Continuous change with mostly a spring which brings the handle always back to a 'neutral' position.
- Change of the handle in small steps, 24 total. In this case the spring has been replaced by a rib.

Continuous change has to be digitalized. The number of steps in 2) is accurate enough for the applications used, that means 24 step full range. Thus in the forward-only mode an accuracy of 24 steps and in the reverse/forward mode 12 steps each. The difference in pulse duty of the RC input is 2 msec – 1 msec = 1 msec. That means 1 msec has to be measured with an accuracy of 1/24 which is about 4%. The timer used, should thus have a period of 1 msec x 4% = 40µsec, or a frequency of about 25 kHz.

The *Timer0* and *Timer1* are to slow, which means we have to use *Timer2*. That creates a problem, because the fast *Timer2* is standard used for PWM. That means no standard Timer is available. But the RC pulse times have to be measured only relative compared to the maximum value. That means any time unit can be used. The quickest timer is a while statement with a counter which increases one count each loop. I call this *Timer9*. The assembler shows that this loop takes 14 steps which is at 32MHz, 14 x 4/32 = 1.75 µsec. This is equal to 570 KHz for *Timer9*. Using this *Timer9*, 2 msec pulses will result in 1150 counts, which is equal to 2 msec/1150 = 1.73 µsec/count. This is about 25 times better then required. Even with 16 MHz oscillator, the accuracy is about 10 better as required.

The calculation of the %PWM from the RC-period time requires 2 divisions, which both can introduce a loss on accuracy, but this will be small for the particular chosen kind of divisions, see Flowcode.
Later the possibility of stopping the motor if the RC signals are lost, made the T*imer9* loop much larger. This is due to an additional if statement to test if both the RC pulse duration and the pulse time do not exceed limits. The while loop becomes now 42 program steps, which makes the *Timer9* a factor 3 slower to 190 kHz (32 MHz oscillator) respectively 95 KHz (16 MHz oscillator). This is still an acceptable frequency.

**Pulse Width Modulation**

Integrating a PWM signal results in a controlled DC signal between 0 Volt and the maximum voltage of the pulses. Motors works as integrators, so the microcontroller just has to produce PWM to control the speed of the motor. Standard voltage/power amplification of the microcontroller PWM is necessary for DC motors.

To control a motor, see figure 3, there are 2 PWM's necessary. The most simple solution is with one PWM continuously on 'low' or 'high' and the other PWM varying from 0-100%. The intention is to have an electronic board as small as possible. Therefore we will use a smallest 8 pins microcontroller PIC12F1840, but this one does have only half bridge possibility. That full bridge can still be used, with our set-up of one PWM fixed and the other PWM with variable duty percentages.

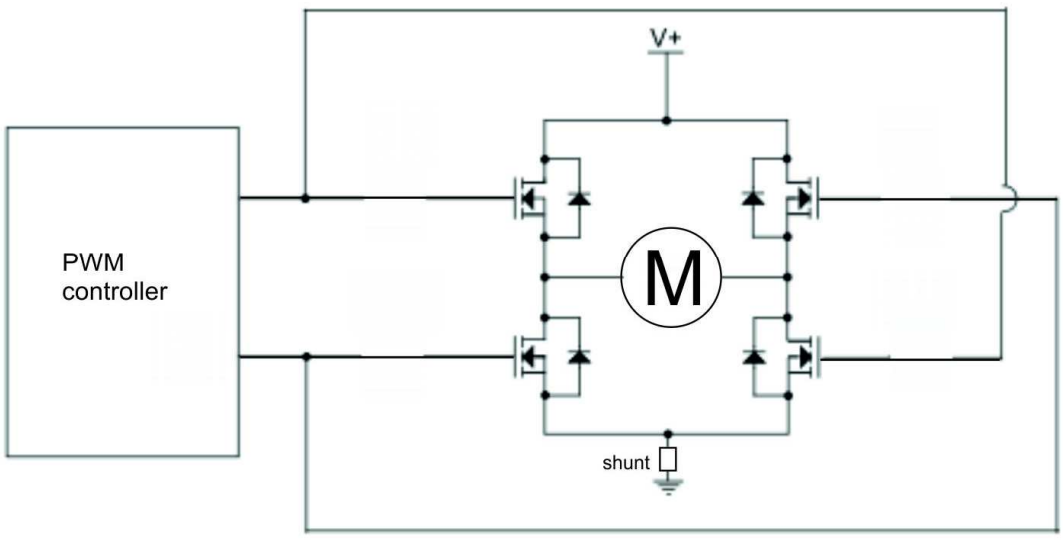



*Figure 3. Full bridge motor control with shunt for current control.*

Standard *Timer2* is used to produce PWM signals. One motor drive used BD6222, see later, requires PWM between 20-100 kHz. At 32 MHz oscillator and 8 bit PWM accuracy this is possible at 31.25 kHz. Using discrete MOSFET's, lower frequencies will work better, but below 20 kHz there is a possibility that system can be heard (for not to old people). For reasons given later on, 16 MHz is used in case of bridge drive with discrete components is used. The PWM frequency will be about 15.5 kHz. The requirement for the ADP3624 bridge driver is that the input pulses are steep as possible. But this is a general requirement, also applicable for all kind of bridge drivers. There ARE no parameters in the microcontroller used to influence this.

# Securities

## *Disabling PWM's at start-up*

This is a simple software security e.g. not to start the motors at the start-up procedure of the microcontroller.

## *Maximum motor-current / bridge-temperature sensing*

By measuring the voltage over the shunt resistance it is possible to control the maximum allowed current through this shunt and thus through the motor, see figure 3. There are 2 possibilities to measure this voltage: 1) with an Analog-Digital-Convertor (ADC) or 2) with the voltage comparator. The requirement is that this should be done at the fastest response as possible, which is the voltage comparator.

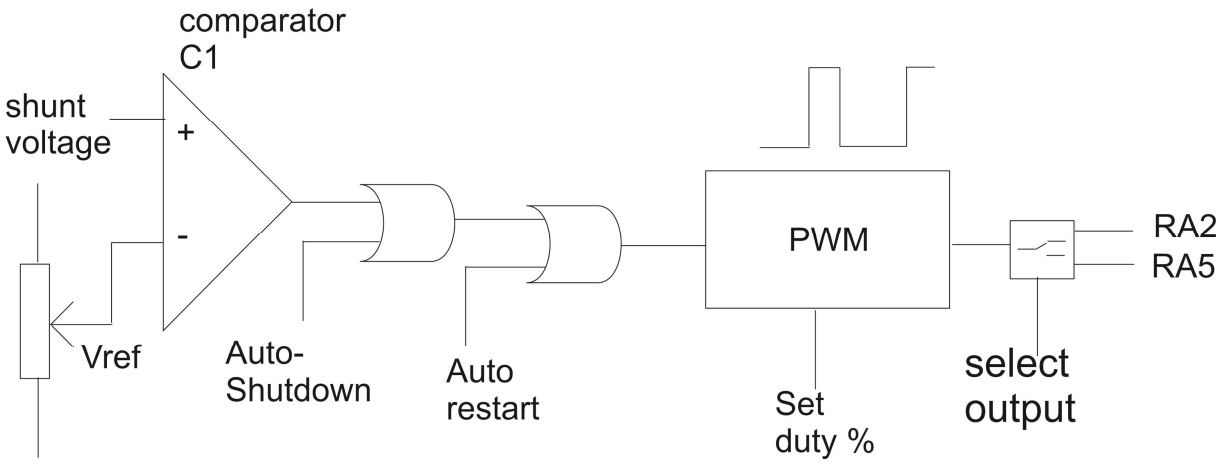



*Figure 4. Control of the PWM with left the auto-shutdown scheme and at the right side the output control between two pins.*

The inputs of the voltage comparator in the microcontroller are the shunt-voltage and an internal reference voltage. This references voltage can be set to the wanted motor current (shunt-voltage). An auto shut-down of both PWM's will prevent the motor for damage. Facultative, an auto restart can be used. Later on it was found that very short peak currents can be present, with an average current far below limits. To prevent auto-shutdown of the PWM in these cases, an external RC circuit can be placed in the sense line. To prevent damage of the input of the comparator by a shunt voltage over the 5 Volt, a zenerdiode of 5.1 Volt must be used parallel to the comparator input.
The bridge driver ADP3624 has an external output for warning for to high IC temperature. The output can be used as sense input. Any reference voltage <5V will do in this case. Remark: A reference voltage = 0V is used in the Flowcode to disable the auto-shutdown procedure.

## *Motor induction and back EMF*

Any speed change of a motor will introduce an EMF and can result in peaks below 0 Volt and peaks above the power supply. Depending on the way of PWM's used, reverse voltages over the MOSFET are possible. To prevent damage of these MOSFET's fast Schottky diodes must be placed parallel to the MOSFET's. Most MOSFET does have these diodes internal, but check the datasheet. To overcome the EMK problems a 10 µF *ceramic* and 100 nF capacitors should be placed over the motor power-supply, placed as close as possible to the motor bridge drive. An additional 100 nF ceramic capacitor has to be placed over threads to the motor, as close as possible to the motor.

## *Noise and Filtering*

The setup as described above, is a response of the PWM on change of the RC time as quick as possible.
However, noise on the RC input, can introduce unwanted changes in PWM's.
Therefore there are 2 filtering possibilities introduced in the Flowcode.

### *a) No Filtering*

In the main program, the RC-time will be measured and the PMW % will be set. Getting the RC-time takes 150 program steps and the forward or reverse loop, to calculate the percentages PWM, takes about 450 program steps, total 600 program steps. That takes 600 x 4 /32 = 75 µsec at 32 MHz or 150 µsec at 16 MHz oscillator. The repeating rate of the RC signal is 50 Hz or 20 msec. That means the program loop time will be fast enough to read each RC pulse in sequence. Of coarse only if no filtering is used.

### *b) No Filtering*

Moving average (MA) can be chosen for the RC-input pulse times. A simple kind of moving average is given on the Matrix Multimedia Forum[*Matrix Multi-media*]. That is at each iteration a particular part of the moving average will be exchanged with the same part of the new value. In formula:

MA_new= MA_old - MA_old/k + NEW-Value/k
with k= any natural number > 0

In Flowcode only the following k-values are used:
k = 2^n = 1,2,4,8,16 with n=0,1,2,3,4
k=1 (n=0) means no filtering.

The reason is that with these k-values a division is a simple roll right action.

The response of the MA depends on the k-value. Large k-values make MA 'slow'. The number of iterations necessary to reach the 'end'-value within the wanted accuracy is given in table 1.

| **# of steps** | | | **Accuracy** | |
|---|---|---|---|---|
| | **k** | **n** | **99%** | **99.9%** |
| **No filtering** | **1** | **0** | 0 | 0 |
| **Small damping** | **2** | **1** | 7 | 10 |
| **Medium damping** | **4** | **2** | 16 | 23 |
| **Strong damping** | **8** | **3** | 35 | 49 |

*Table 1. The number of steps/iterations which are necessary to reach an accuracy of 99% or 99.9%. No filtering with k=0, gives directly an accuracy of 100%.*

E.g. a small damping with k=2 takes 10 RC-signals to reach an accuracy of 99.9% and that takes 10x 20x2 msec= 400 msec, which will be good enough in most applications. This setting is advised to be used as start value. The factor 2 will be explained in noise chapter 4.

### *c) Filtering PWM*

Here also a moving average can be used to make the response on the RC-sender handle more or less smoother. It is advised to use *no* PWM filtering as start value (k=1, n=0). The disadvantage is that the safety of 0% PWM when reversing the motor direction will be lost when using moving averaging here.

### *d) HF Noise*

In the first set up of the program only the RC time on (the pulse time) was measured. But I found that I live in a surrounding with a lot of noise (40 MHz), which lets change the motor speed and direction at random without any RC transmitter. After changing the 40 MHz crystals the noise was reduced but not completely gone. The result is that the motor speed is unpredictable in case the receiver is out of range of the RC transmitter. And that made me to decide that an additional criterion has to be introduced to select RC signals from noise.

The RC frequency is rather constant (about 50 Hz), so the period time between 2 rising pulses should be within a small range. Measuring this period time at the start up and define a range in which all other period times has to fall for a pulse to be recognized as RC-pulse. This improved the working of the motor speed control. The consequences are that not all RC pulses but only the even (or uneven) RC pulses will be measured, thus with 25 Hz maximum.

### *e) Loss of signal*

A special case is that the receiver gets out of the range of the transmitter. With airplanes and model train it is wanted that the motor speed will not change. However, with cars it is wanted that the car stops as quick as possible. Both options are available. Especially in the last case it is strongly advised to set the RC transmitter on *before* powering the receiver and microcontroller.

## Special Registers

Of course the main program is in Flowcode (version 5). But there has to be set some special registers as given in table 2 below and in details explained in the annexes. C -codes are used both in C-code component macro as in interrupt procedures.

| Register Name | Abbr. | Remarks | Used in macro | Page |
|---|---|---|---|---|
| *Annex I Set Internal Oscillator Frequency* | | | | |
| Oscillator Control Register | OSCCON | Settings for 16 and 32 MHz internal oscillator | Main and Init macro | 61 |
| Oscillator tuning register | OSCTUNE | Set frequency to maximum | Init macro | 63 |
| *Annex II Interrupt-On-Change of RC pulses* | | | | |
| Interrupt Control Register | INTCON | Both global interrupt enable and IOC interrupt enable | Interrupt in Get_RC_Time macro | 83 |
| Interrupt-On-Change PortA Positive Edge Register | IOCAP | Rising edge interrupt – on-change of input RA3 | Interrupt in Get_RC_Time macro | 120 |
| Interrupt-On-Change PortA Positive Edge Register | IOCAN | Falling edge interrupt – on-change of input RA3 | Interrupt in Get_RC_Time macro | 120 |
| Interrupt-On-Change PortA Flag Register | IOCAF | Interrupt-on-change flag of input RA3 | Interrupt in Get_RC_Time macro | 120 |
| *Annex III Alternate PWM output pins* | | | | |
| Alternate Pin Function Control Register | APFCON | Change PWM output port between RA2 and RA5 | Init macro | 112 |
| *Annex IV Current limit control* | | | | |
| Voltage Reference Control register 0 | DACCON0 | Reference voltage enabled/disabled and defines the comparator inputs used | Init macro | 144 |
| Voltage Reference Control register 1 | DACCON1 | Set the reference voltage | Init macro | 144 |
| Comparator C1 Control Register 0 | CM1CON0 | Comparator C1 enabled/disabled and a lot of other settings | Init macro | 157 |
| Comparator C1 Control Register 1 | CM1CON1 | Selection IN+ and IN- of the comparator | Init macro | 158 |
| CCP1 Auto-Shutdown Control register | CCP1AS | Auto shutdown enabled/disabled of PWM | Interrupt in Init macro | 208 |
| Enhanced PWM Control register | PWM1CON | Auto restart enabled/ disabled of PWM | Init macro | 209 |

*Table 2. Registers of PIC12F1840 used. Page numbers as used in datasheet DS41441B of Microchip (2011) [Microchips]. Details are given in the annexes.*

## Electronic schemes

There are 3 types of bridge drive used. One only with discrete components and one with a commercial bridge driver BD6222 (maximum 2 A and 18 V) and the ADP3624 (maximum 4A and 18V). The figure 5 shows the microcontroller part and Figures 6 the 3 bridge drivers. The 3 figures 7 show a picture of the 3 PCB respectively.

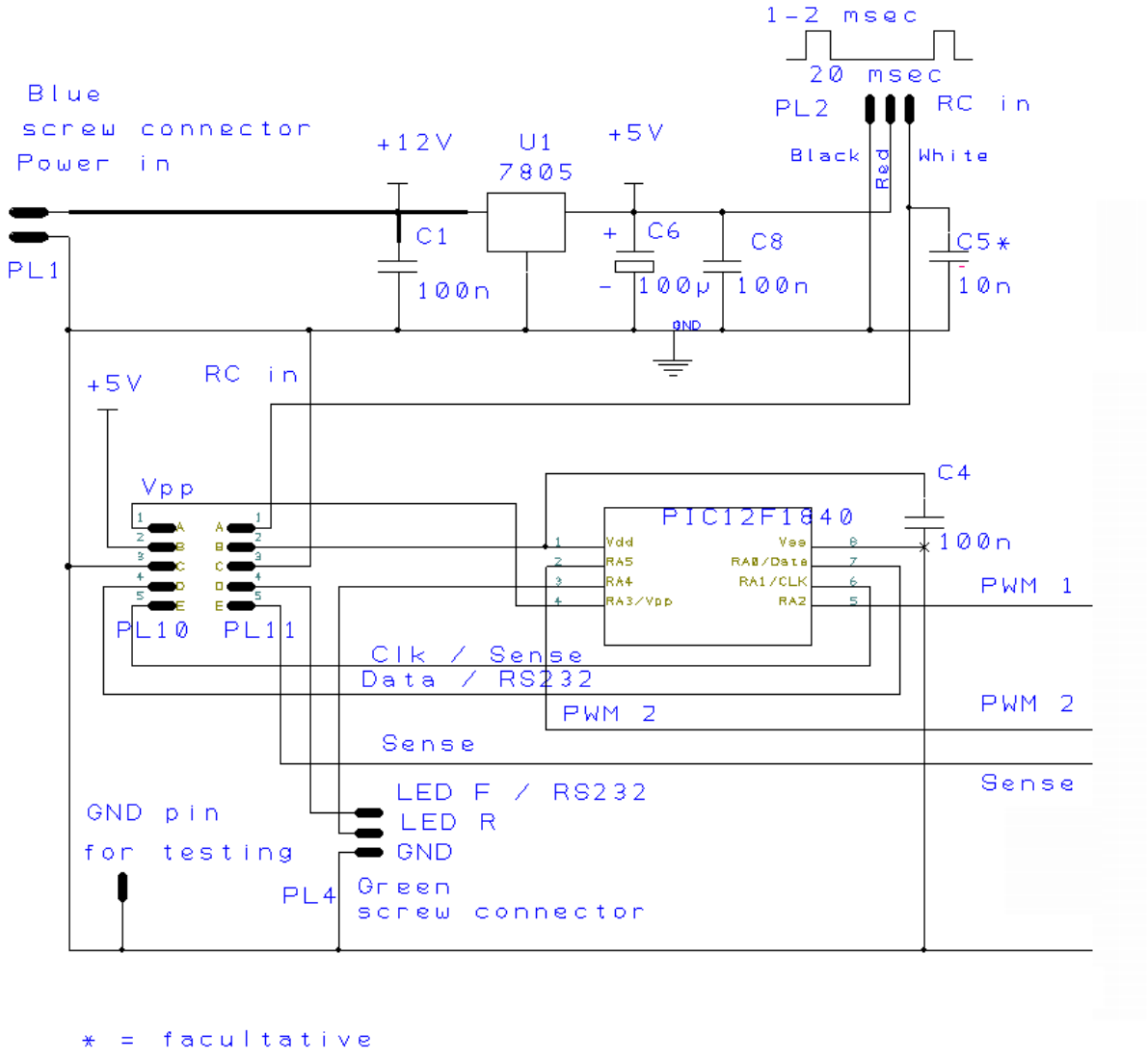



*Figure 5. The PIC12F1840 microcontroller with power supply, RC-input and sense-input(shunt voltage), PWM1-, PWM2-output, 2 LED's output, RS232 output and the In-Circuit Serial Programming (ICSP) connector PL10. On duty PL10 is connected to PL11. C5 is optional. Connections PMW1, PWM2 and Sense continue in figures 6A and 6B.*

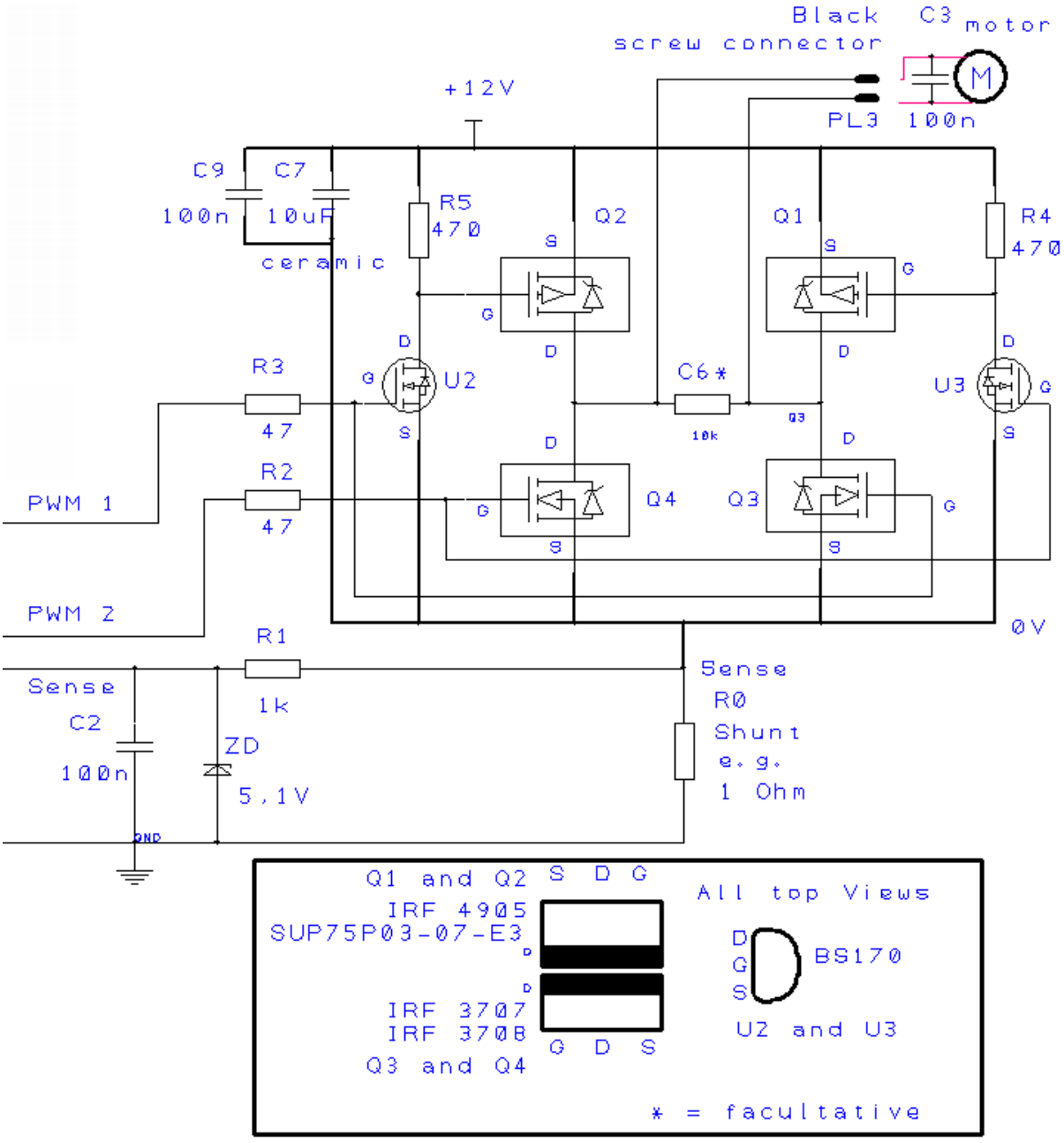



*Figure 6A. The motor bridge driver with 2 power n-MOSFETs (Q3 and Q4) at the lower (0V) side and 2 power p-MOSFETs (Q1 and Q2) at the high voltage side. The gate voltages of the p-MOSFET are produced by the low power BS170 n-MOSFETs. The voltage over the shunt resistance is proportional to the bridge current. Peak voltages can be integrated by the R1C2 circuit and the Zener diode of 5.1V prevents damage of the microcontroller for a shunt voltage over the 5 Volt. Capacitor s C7 (10 µF ceramic) and C3 (100 nF) are essential. R6 is used during testing without a motor. Connections PMW1, PWM2 and Sense continue in figure 5.*

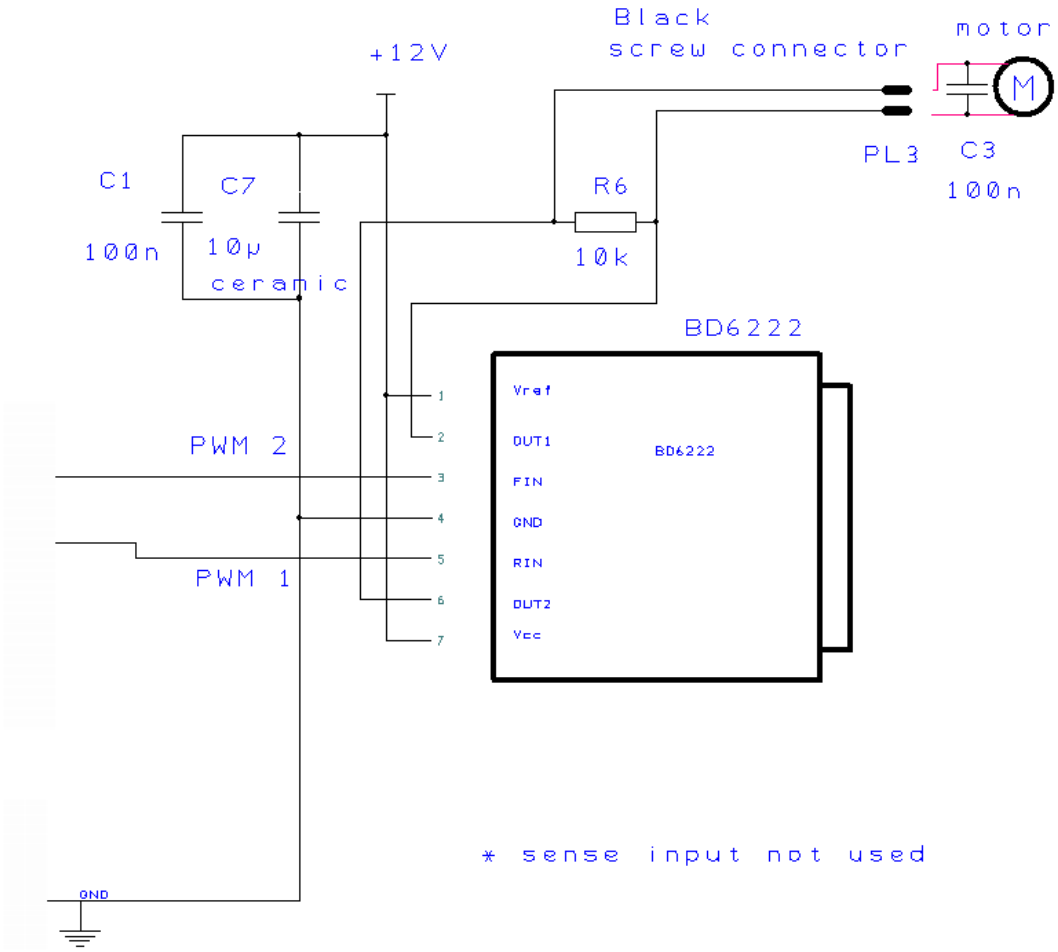



*Figure 6B. With 2A bridge driver BD6222. Due internal safety control no external current control is necessary. Connections PMW1, PWM2 continues in figure. (No sense input)*

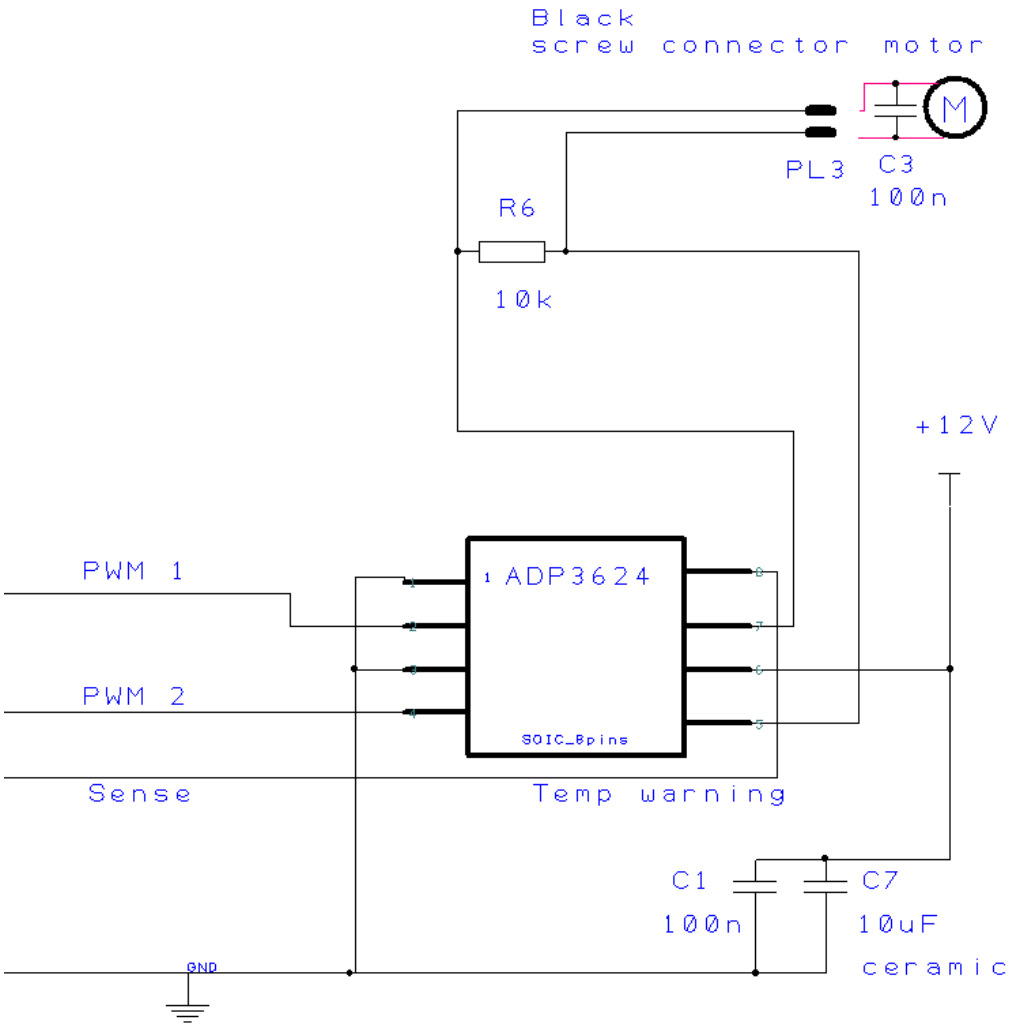



*Figure 6C. . With 4A bridge driver ARP3624. Instead of current control, temperature warning is used to control. Connections PMW1, PWM2 and Sense continue in figure 5.*

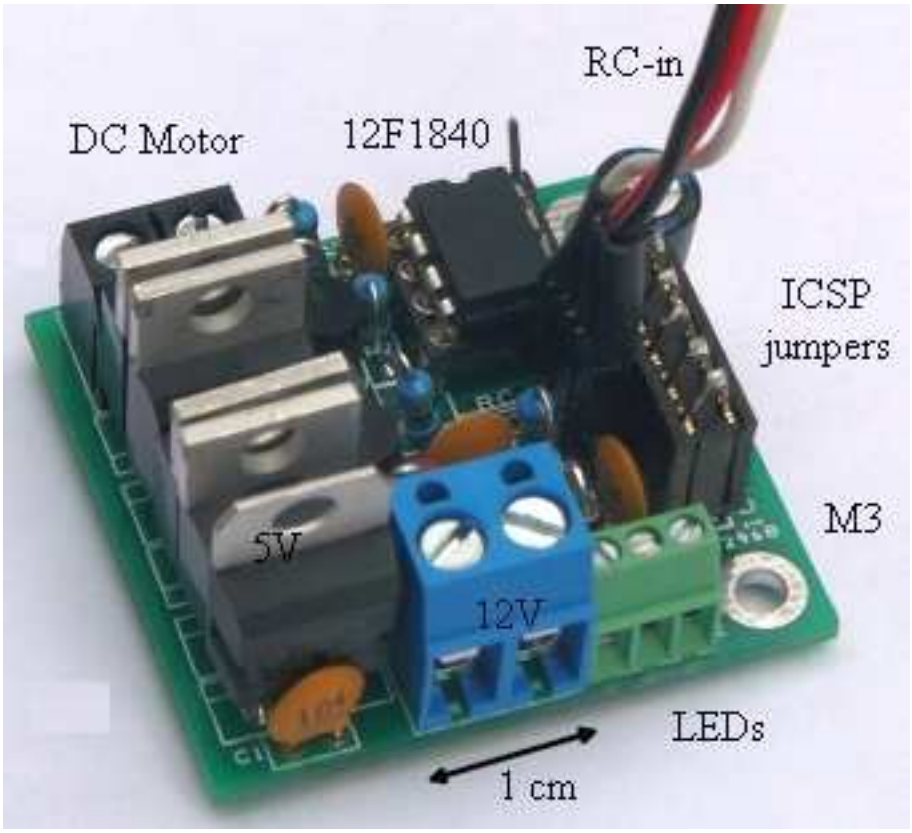



*Figure 7A. Printed Circuit Board (PCB) with discrete components. Board size is 35.5x38 mm*

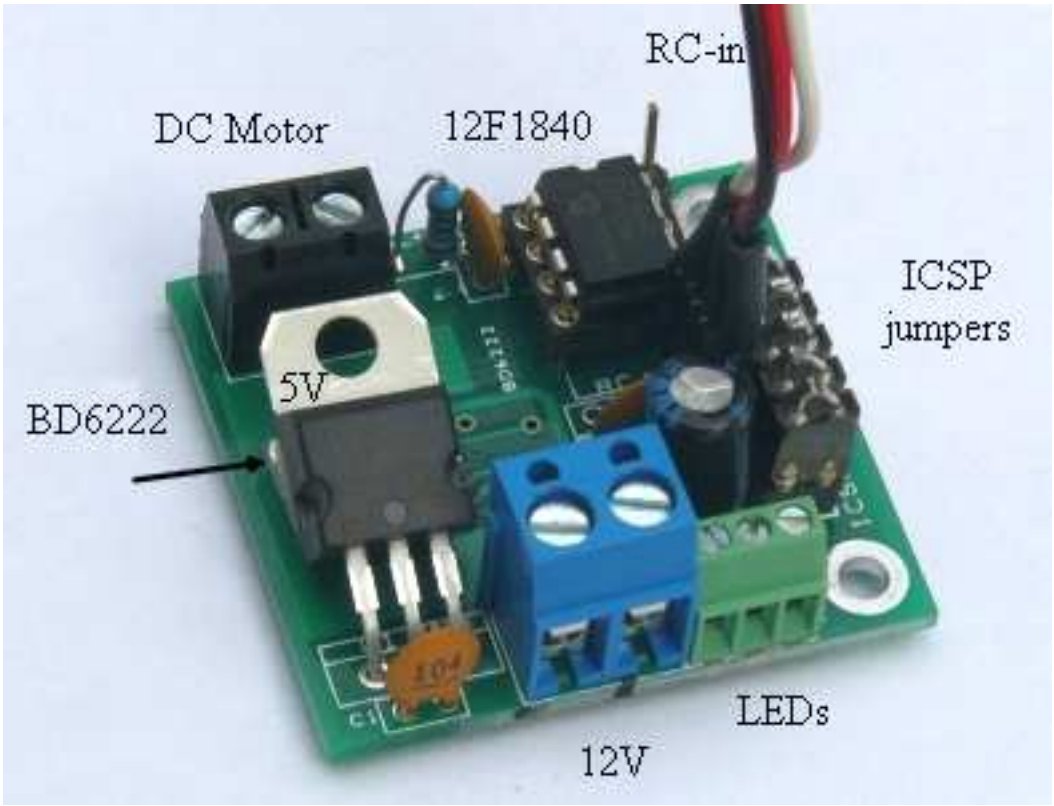



*Figure 7B. PCB with BD6222 bridge driver. Board size is 35.5x38 mm*

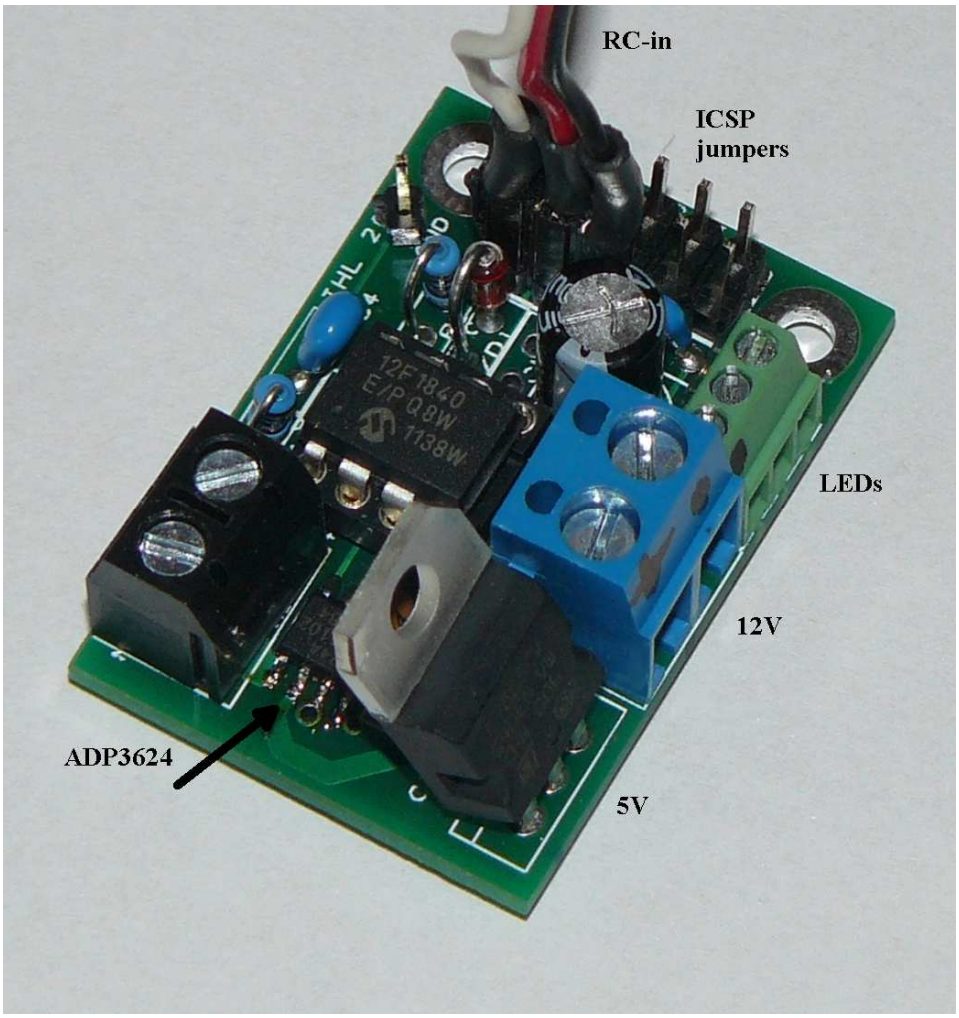



*Figure 7C. PCB with ADP3624 bridge driver. Board size is dimensions are 26.5x 38.2 mm.*

The electronic schemes show the 8-pins microcontroller PIC12F1840 with the motor bridge driver. In Figure 6A, the n-MOSFETs (Q3 and Q4) are selected on minimum internal resistance, maximum current and minimum gate voltage Vgs necessary to open the MOSFET completely. This Vgs should be maximum 4 Volt. IRF3707 and IRF3708 are good candidates. p-MOSFETs (Q1 and Q2) are selected on minimum internal resistance and maximum current. IRF4905 and SUP75P03-07-E3 can be selected. And the n-MOSFETs (U2 and U3) are selected on minimum internal resistance and minimum gate voltage Vgs necessary to open these MOSFET completely. This Vgs should be maximum 4 Volt. The best is the BS107 MOSFET.

Due to reverse EMF of the motor, all MOSFETs should have internal Skottky safety diodes.

The microcontroller will give PMW1 continuously on 0V and PWM2 pulses between 0-100% or visa versa. PWM1 will results in no drain-source currents for U2, Q2 and Q3 and a current through U3, Q1 via the motor and Q4 during PWM2 is high. LED2 will be ‘on’.

The shunt resistance can be chosen yourself. The RC circuit (R1C2) is optional. Do not use this integrating circuit if you expect rarely high currents which need auto-shutdown without auto-restart. Use the RC circuit if you intent to use the auto-restart. The RC time will be at least the time between the shut-down and auto-restart. The zener diode is necessary to protect the microcontroller input. And the resistance R6 parallel to the motor is used during testing of the bridge without a motor. This resistance can be in place with the motor connected. The shunt resistance should be chosen so the voltage over the shunt will be ≤1 Volt. This is necessary to the fact that the n-MOSFET's are fully open with a Vgs of ≥4 Volt. Below 1 Ohm, it is hard to get small resistances. It is possible to make these shunts yourself by means of resistance wire , see annex VIII.1.

High frequency can be present on the RC-input, e.g. if the RC-sender is close to the RC-receiver. In that case it can be necessary to filter this high frequency with e.g. a small 10 nF capacity (C5).

In-Circuit Serial Programming (ICSP) is used by means of the J5 connector of the EB006 board and the PCB board. There is a switch to choice between the programming and operation mode.

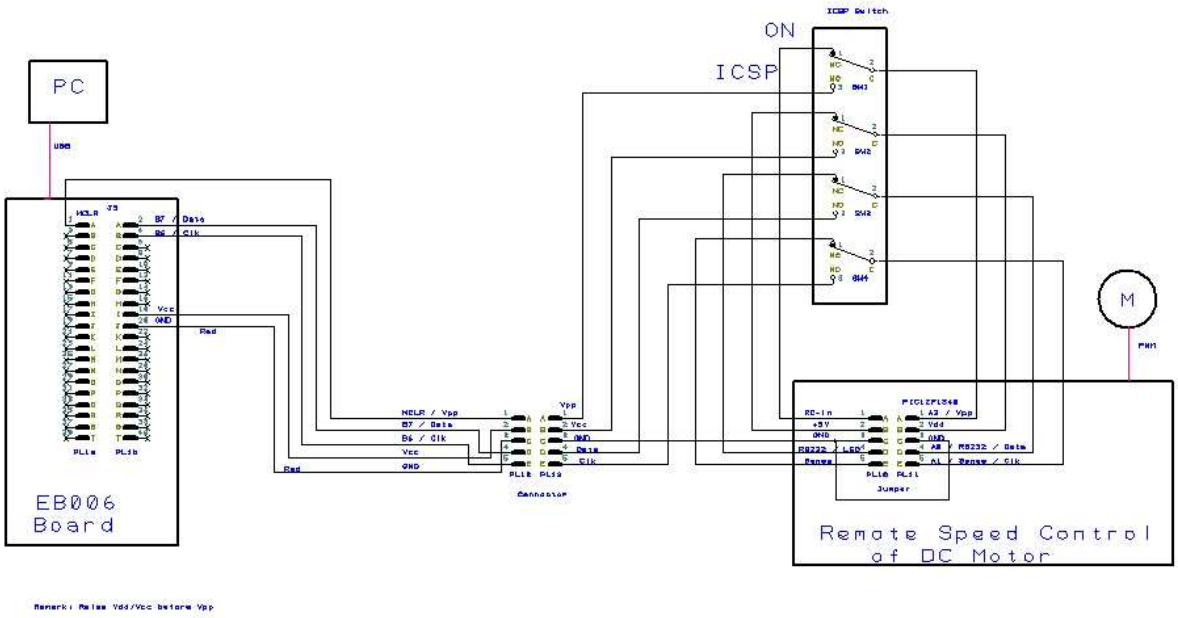



*Figure 8. ICSP with switch between programming and operation mode.*

The microcontroller has only 8 pins. The choice of the pins, like PWM output, RC input etc., is given in an Annex VI. An external master clear MCLR is not possible. MCLR should be realized during powerup. Programming on the board is possible (ICSP). It is advised to set in the PPP options to raise Vdd *before* Vpp (PPP options can be found under the 'options..' knob of the chip configuration window). During operation PL10 should be connected to PL11 with 5 jumpers.

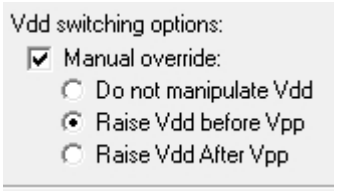



*Figure 9. Recommended PPP-settings in case of ICSP (programming on board)*

The discrete component setup is shown in figure 6A and 7A. The size of the board is as small as possible 35.5 x 38 mm. The advantage of this setup is that it is easy to produce yourself. The disadvantage is the possible presence of high peak currents at the start of each PWM. To prevent that as much as possible, lower PWM frequencies are advised. Remark: These peak currents are mostly damped by the presence of the 10 µF ceramic capacitor.

If the bridge driver BD6222 is used (figure 6B and 7B), nearly all components of figure 6A will be replaced by this bridge driver. I even deleted the current sensing, because the IC itself takes care of that. The advantage of this bridge driver is that the motor control is optimized. The frequency of the PWM should be between 20-100 kHz, which is possible with and internal microcontroller oscillator of 32 MHz giving a 31.25 KHz PWM frequency. A disadvantage can be that the BD6222 is a smd component. A simple way to solder these smd's is described in annex VIII.2.

The 5Volt regulator is oversized for the microcontroller itself. But power can be used by other servo motors also connected to the RC-receiver.

The motor should have a capacitor of 100 nF as close as possible to the motor. Due to reverse EMK of the motor, in both setups a 10 µF and 100 nF, both ceramic capacitors, should be placed over power supply of the bridge driver and as close as possible to this bridge driver. This capacitor is commercial only available as smd but soldering is no problem.

## Working model

The reason to develop this motor bridge driver was that even commercial bridge driver were blow up after a while. That makes me thinking that I can do it myself better. And is it not easy to transform a RC pulse between 1 and 2 msec to a PWM from 0-100%? Developing both software and hardware took much more time then I expected. Time will learn if my setup is better. I learned a lot in the meanwhile, that can help to overcome new problems.

Three PCB boards are described. The third one with the bridge driver ARP3624 is advised. The PCB board is the smallest and the drive has an over-temperature sense output.

The motor is part of the LGB model train (gauge I = 45 mm, scale G about 1:20) powered with 12 NiMH cells and with a remote control (RC) 40 MHz receiver. The motor uses about 200-300 mA. If I stop the motor at full speed about 800 mA is used. Of coarse peak currents can be much higher.

## Summary

In the article the software and hardware development of a Remote speed control of DC motors is given. One of the requirements is that the electronic board is as small as possible. That is the reason that the 8-pins Microchip microcontroller PIC12F1840 has been chosen in combination with a commercial bridge driver ARP3624. The software has been made by means of Flowcode 5 of Matrix Multimedia. The initial values of a lot of parameters can be changed yourself. This software will be available on Matrix Multimadia Forum and/or Elektor Forum. On request C-code or hex file will be available. A movie about this project is available at my website [*author*].

### Acknowledgement

Many people have advised me during the development of these RC motor speed controls. Especially all the contributors of the Matrix Multimedia Forum [*Matrix Multimedia*] and the Elektor Forum [*Elektor Forum*]. Without them the development of respectively the software and hardware was not possible. Theo van Hoof [*Theo van Hoof* ] thanks for your advices on the moment I intended to stop.

## References

**Microchips, MOSFET, MOSFET drivers. ceramic capacitors and resistance wire**

*Microchips:*
PIC12F1840 8 pins 8-bits pins Microchip microcontroller, as used in this project
PIC16F1939 8 pins 8-bits pins Microchip microcontroller e.g. bridge drive figure
*MOSFETs:*
IRF3707 n-MOSFET 30V, 59A, TO-220
IRF3708 n-MOSFET 30V, 62A, TO-220
IRF4905 p-MOSFET -55V, -74A, TO-220
SUP75P03-07-E3 p-MOSFET -30V, -75A, TO220AB
BS170 n-MOSFET 60V 0.5A TO-92
*MOSFET Drivers:*
ADP3624ARDZ 4A MOSFET drivers with non-inverting inputs, SOIC
BD6222 bridge driver 18V 2A HSOP25 (smd)
*Ceramic capacitors:*
AVX-SE045C106KAR 50V, 10 µF ceramic capacitor
*Resistance wire*
e.g. 5.65 Ohm/m (isachrom 60) at www.conrad.com number 421201 (or Conrad at your country)

**Matrix Multimedia:**
http://www.matrixmultimedia.com/mmforums/viewtopic.php?f=29&t=8921
Comparator interrupt
http://www.matrixmultimedia.com/mmforums/viewtopic.php?f=29&t=7792
Moving average

**Elector Forum**
http://www.elektor.nl/forum/forum/algemene-forums/microcontrollers/directe-aansturing-mosfet-door-microcontroller.1985092.lynkx : on the PCB (in Dutch)

**Theo van Hoof**
http://www.theovanhoof.be/
http://www.modularcircuits.com/blog/
http://modularcircuits.tantosonline.com/blog/articles/h-bridge-secrets/

**Author**
jan.lichtenbelt@lichtenbelt.nl : Jan Lichtenbelt
http://treintjes.smugmug.com/Other/Electronica-Electronics : this project
http://lichtenbelt.com/jan/trein/kleintjestoom.htm : garden rail

# Appendix

## Set Internal Oscillator Frequency

The internal oscillator frequencies used are 16 MHz and 32 MHz. In the Oscillator Control Register OSCCON, the Internal Oscillator Frequency Select bits IRF <6:3> are 1111 for 16 MHz, or:

```
osccon=0x78;        // 16 Mhz osc.
```

For 8 MHz IRF<6:3> should be 1110. To multiply this frequency with 4 to 32 MHz, the Software PLL Enable SPILLEN bit 7 should be 1 (this makes the frequency setting independent of PLL value in the config word), or:

```
osccon= 0xF0;       // 32 MHz osc.
```

These frequencies can be set, facultatively, to a maximum value with the Frequency Tune Bits TUN <5:0> in the Oscillator Tuning OSCTUNE register. The maximum is reached for TUN=011111, or:

```
osctune=0x1F;       // maximum frequency
```

## Interrupt-On-Change on RC pulses

To measure the period time of the RC pulses a timer counter is used. This times will be start on rising edges of the pulse and the counter ends at the falling edge of the pulse.

Remarks:
a. There is a special timer used (called *Timer9*), see chapter Timers and Accuracy.
b. Here we use the procedure to set (st_bit) and clear (cr_bit) special bits of a register instead of the whole register as done in Annex I.

1. To enable this special interrupt, both the Global Interrupt Enable GIE bit 7 and the IOC enable bit3 should be set to 1 in the Interrupt Control Register INTCON, or:

```
st_bit(intcon,GIE);          // global interrupt enable
st_bit(intcon,IOCIE);        // enable IOC (interrupt on change)
```

2. We start first detecting the rising edge of the RC pulse. Therefor one of the Interrupt-On-Change (IOC) PortA Positive Edge bits IOCAP<5:0> has to be set in the IOC PortA Positive Edge register IOCAP, and because we use RA3 as IOC port,

```
st_bit(iocap,IOCAP3);       // rising edge RA3
```

with the number 3 for bit 3 and RA3.

3. The next step is to reset the interrupt flags and enable this IOC interrupt. If a IOC interrupt occurs the IOC Flag will be set both for the IOC Interrupt Flag bit IOCIF in INTCON and for that particular port in the IOC PortA Flag Register IOCAF. After re-setting these flags the interrupt will be enabled for a positive or negative pulse (depending on setting IOCAP or IOCAN). At last we has to set the IOC enable IO-CIE bit 3 in the INTCON register (see also point 1):

```
cr_bit(intcon,IOCIF);      // reset interrupt flags for all RA's
cr_bit(iocaf,IOCAF3);      // reset particular interrupt flag RA3
st_bit(intcon,IOCIE);      // enable IOC (interrupt on change)
```

4. As soon as we detected the rising edge, we stop this procedure and replace it with detecting on the falling edge with:

```
cr_bit(iocap,IOCAP3);      // disable rising edge RA3
st_bit(iocan,IOCAN3);      // set falling edge RA3
```

with IOCAN the equivalent of IOCAP but for falling edges.

5. Stopping this interrupt can be done simple by clearing/disabling the global interrupt GIE bit and the IOC interrupt bit in INTCON:

```
cr_bit(intcon,IOGIE);      // disable global interrupts
cr_bit(intcon,IOCIE);      // disable IOC (interrupt on change)
```

6. The interrupt handler will be:
Enable Code:

```
st_bit(intcon,GIE);          // global interrupt enable
st_bit(iocap,IOCAP3);        // rising edge RA3
cr_bit(intcon,IOCIF);        // reset interrupt flags for all RA's
cr_bit(iocaf,IOCAF3);        // reset interrupt flag RA3
st_bit(intcon,IOCIE);        // enable IOC (interrupt on change)
```

Diable Code:

```
cr_bit(intcon,IOGIE);        // disable global interrupts
cr_bit(intcon,IOCIE);        // disable IOC (interrupt on change)
```

Handler Code:

```
if (iocaf.IOCAF3)            // if interrupt RA3 flag is set
{
FCM_%n();                    // call selected macro
cr_bit(intcon,IOCIF);        // reset interrupt flag for all RA's
cr_bit(iocaf,IOCAF3);        // reset interrupt flag for RA3
}
```

The interrupt macro used
In this IOC_Interrupt macro the detection of rising edge will be exchanged with the detection of falling edge and visa versa. The interrupt counter9 is stored into the variables Temp (pulse repetition period time) and RC_Time_On (pulse time)

```
if (iocap.IOCAP3)      // rising edge RA3 ?
{
cr_bit(iocap,IOCAP3);      // disable rising edge RA3
st_bit(iocan,IOCAN3);      // set falling edge RA3
```

```
FCV_TEMP= FCV_INTERRUPT_COUNTER9; // save interrupt counter9
FCV_INTERRUPT_COUNTER9=0;        // reset interrupt counter9
}
else                  // falling edge
{
cr_bit(iocan,IOCAN3);     // disable falling edge RA3
st_bit(iocap,IOCAP3);     // set rising edge RA3

FCV_RC_TIME_ON= FCV_INTERRUPT_COUNTER9; // save interrupt
counter9
}
```

8. Change falling edge detection into rising edge detection
After the detection of the pulse period time Temp, the falling edge detection will be changed into rising edge detection:

```
cr_bit(iocan,IOCAN3);     // disable falling edge RA3
st_bit(iocap,IOCAP3);     // set rising edge RA3
```

## Alternate PWM output pins

The PWM output CCP1/P1A is standard on the RA2 pin. Using the Pin Select CPP1SEL bit 1 in the Alternate Pin Function Control Register APFCON, the output can be set to RA5.

```
apfcon= 0x00;     // PWM output to RA2
```

or

```
apfcon= 0x01;     // PWM output to RA5
```

## Current/temperature limit control

To control the current through the motor, the voltage over a shunt will be compared to a reference voltage. This comparator shuts down the PWM output if the shunt voltage is to large. Facultative an auto restart can be used also. See figure 4 with the schematic set up. The external component, the shunt resistance can be chosen freely, dependent on the expected currents and the possible reference voltages. However the shunt voltage may not be larger then 1 Volt, leaving 4 volts for source/ gate voltages. Reference voltage, comparator, auto-shutdown and auto-restart are parts of the microcontroller. This sense input can be used not only for currents limit but also for any other voltage sense, like e.g. the temperature warning signal (high) as output of the ADP3624 bridge driver.

**a) Reference Voltage for the Comparator**

The positive and negative voltage of the Digital-Analog-Convertor(DAC) must be selected for the reference voltage. This DAC must be enabled. This all is done in the Voltage Reference Control Register 0 DACCON0.

The DAC Enable DACEN bit 7 must be set to 1,
The DAC Positive Source Select DACPSS bits <3:2> will be 00 for V+=Vdd
The DAC Negative Source Select DACNSS bit 0 will be 0 for V-=Vss

```
daccon0= 0x08;   // DAC enabled, V+ = Vdd and V- = Vss
```

With 5 DAC Voltage Output Select DACR bits <4:0> of the DAC Voltage Reference Register 1, the output voltage of the DAC can be selected between 0 and 1/2V+ in 16 steps. With V+ = 5V and V-=0, Vref= 5*DACR/32= 0.15625xDACR Volt.

```
daccon1=FCV_DACR;   // DACR gets a value in the INIT macro.
```

***b) Comparator settings***

Two registers are used for the comparator settings: Comparator C1 Control Register 0 and 1, respectively CM1CON0 and CM1CON1. The first one to enable the comparator and selection in which case the output becomes high. The second one to select the Input+ and Input_ of the comparator.
CM1CON0 settings:

The Comparator Enable bit 7 C1ON is set to 1
Comparator Output Polarity Select C1POL bit 4 must be set to 1
The Comparator Speed/Power Select C1SP bit 2 must be set to 'normal' =1

```
cm1con0= (CM1CON0 & 0x40) | 0x94;
        // enable comparator normal mode
        //  and set polarity
        //  & 0x40 = do not change bit 6
        //  | 0x94 =  set bits 2,4 and 7
```

CM1CON0 **output** (this is the flag for auto-shutdown the PWM):

Comparator Output C1OUT bit 6 will become high if the inputs of the comparator V+ becomes < V- (with C1POL=1)

**Used in Annex V Auto-Shutdown**

CM1CON1 settings:

Comparator Positive Input Channel Select C1PCH bits <5:4> =01 for DAC (Vref) as input+
Comparator Negative Input Channel Select C1NCH bit 0 = 0 connects C1NO- pin or RA1 pin to Input- (this is the sense input)

```
cm1con1= 0x10;   // set comparator IN+ = Vref and IN- = RA1 (sense)
```

## Auto-shutdown and auto-restart PWM

The auto-shutdown of the PWM is regulated in the CCP1 Auto-shutdown Control Register CCP1AS. If activated the PWM will be set to 0%, while the program is still going on. Rest of this auto-shutdown can be done by repower the microcontroller or use the auto-restart procedure.

The CCP1 Auto-shutdown Source Select CC1PAS<6:4> should be 001= comparator C1 output low (this text can be read on two way. But the shutdown occurs if C1 output gets high)

```
ccp1as= 0x10; // Select C1 output to trigger the auto-shutdown
```

The CCP1 Auto-Shutdown Status CCP1ASE bit 7 can be used in an interrupt. The interrupt can be:

Enable code:

```
ccp1as= 0x10; // Select C1 output to trigger the auto-shutdown
```

Disable code:

```
ccp1as= 0x00;  // auto-shutdown is disabled
```

Handler code:

```
if (ccp1as.CCP1ASE)  // CCP1ASE bit get's 1 if auto-shutdown is activated
{
FCM_%n();            // call selected macro
}
```

What to do depends on the time of a peak current compared to the pulse-on-time. Here examples are given for 2 extreme conditions, very short peak and very long peaks

a) Short peak currents

With auto-restart the PWM start directly is Vsense becomes < Vref. If (very) short peak currents activate the auto-shutdown, the result will be that only the rising edge of these peaks will occur and which will results that the motor effectively does not restart. If these short peaks are acceptable, the best way is to integrate the current with a RC circuit in the Vsense. The Vsense will not become > Vref.

b) Long peak currents

If accidental high currents are expected e.g. motor forced to stop, the best way is to wait a while and test if the shut-down conditions are removed which make an (auto-) restart possible.
In this interrupt macro a time delay can be set.

c) Bridge temperature

If a temperature warming is available, like the ADP3624 bridge, this can be easily used with the auto restart. The temperature warning will disappear after a while when temperature drops due to the auto-shutdown. Some hysteresis is used in this type of bridge.

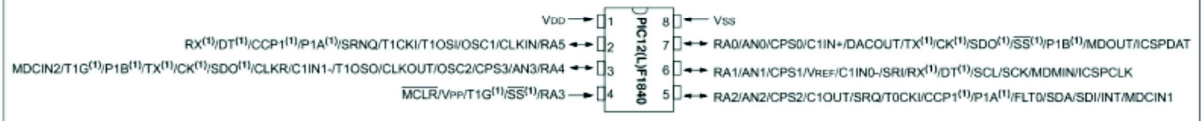
To Auto-restart the PWM, The PWM Restart Enable P1RSEN bit 7 in the Enhanced PWM Control Register PWM1CON must be set.

```
pwm1con= pwm1con | 0x80;  // bit7=1 enable auto restart
```

## Pin settings PIC12F1840

**FIGURE : 8-PIN DIAGRAM FOR PIC12(L)F1840**

**PDIP, SOIC, DFN**




**Note 1:** Pin function is selectable via the APFCON register.

**TABLE : 8-PIN ALLOCATION TABLE (PIC12(L)F1840)**

| I/O | 8-Pin PDIP/SOIC/DFN | A/D | Reference | Cap Sense | Comparator | SR Latch | Timers | ECCP | EUSART | MSSP | Interrupt | Modulator | Pull-up | Basic |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RA0 | 7 | AN0 | DACOUT | CPS0 | C1IN+ | — | — | P1B(1) | TX(1) CK(1) | SDO(1) SS(1) | IOC | MDOUT | Y | ICSPDAT ICDDAT |
| RA1 | 6 | AN1 | VREF | CPS1 | C1IN0- | SRI | — | — | RX(1) DT(1) | SCL SCK | IOC | MDMIN | Y | ICSPCLK ICPCLK |
| RA2 | 5 | AN2 | — | CPS2 | C1OUT | SRQ | T0CKI | CCP1(1) P1A(1) FLT0 | — | SDA SDI | INT/ IOC | MDCIN1 | Y | — |
| RA3 | 4 | — | — | — | — | — | T1G(1) | — | — | SS(1) | IOC | — | Y | MCLR VPP |
| RA4 | 3 | AN3 | — | CPS3 | C1IN1- | — | T1G(1) T1OSO | P1B(1) | TX(1) CK(1) | SDO(1) | IOC | MDCIN2 | Y | OSC2 CLKOUT CLKR |
| RA5 | 2 | — | — | — | — | SRNQ | T1CKI T1OSI | CCP1(1) P1A(1) | RX(1) DT(1) | — | IOC | — | Y | OSC1 CLKIN |
| VDD | 1 | — | — | — | — | — | — | — | — | — | — | — | — | VDD |
| VSS | 8 | — | — | — | — | — | — | — | — | — | — | — | — | VSS |

**Note 1:** Pin function is selectable via the APFCON register.

*Table annex VI. Green pins used during on board programming ICSP. Red pins used during operation. RA4 is used for LED output.*

RS232 output (Tx) or LED1 output on RA0. Shunt voltage on RA1 (-Input of the comparator). PWM output on RA2 or RA5. RC-input on RA3 (Interrupt-On-Change). Remark: A better interrupt performance gives the INT input. But this pin RA2 is already used for the PWM. RA4 used for LED2.

Programming on the board (green pins) is possible (ICSP), but set the PPP options to raise Vdd *before* Vpp (PPP options can be found under the 'options..' knob of the chip configuration window).

## Filtering

In general filtering can be done by averaging data. Here is used a moving average, done on a special way [*Matrix Multimedia*] That is a part of the moving average (MA) is replaced by a part of a new value of the data read.
MA(new)= MA(old) – MA(old)/k + Data(new)/k with k=1,2,3…..
To simplify the program only k= 2^n is used with n=0,1,2,3….. (k=1,2,4,8…)

If new data change suddenly from 0 to 1 (step function) the moving average will reach the end value depending on the damping factor k. A special case is k=1 which is no filtering and the moving average reaches directly the end value.

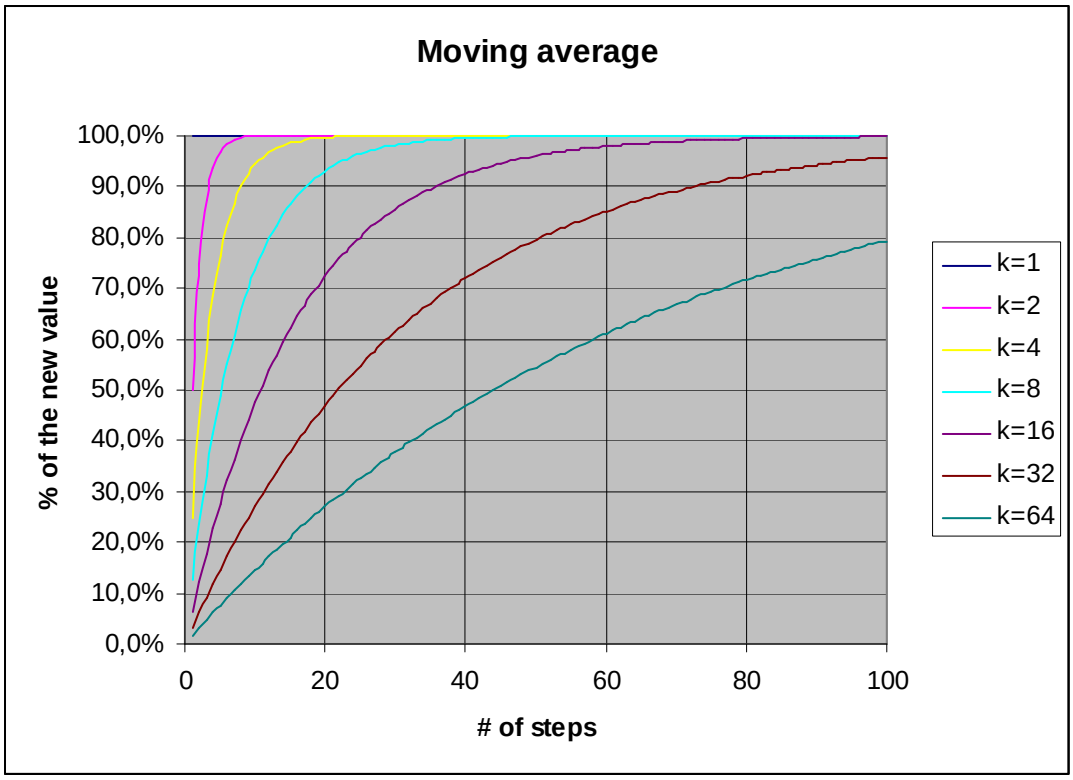



*Figure Annex VIII. Moving average of a change from 0 to 1 as function of the number of steps.*

In the next table the number of steps needed to reach a particular percentage of the end value, are given for different damping factors k.

| % of end value | k 1 | 2 | 4 | 8 | 16 | 32 | 64 | Remarks |
|---|---|---|---|---|---|---|---|---|
| **50%** | | 1 | 2 | 5 | 11 | 22 | 44 | RC-time |
| **90%** | | 3 | 8 | 17 | 36 | 71 | 160 | |
| **95%** | | 4 | 10 | 22 | 45 | 92 | 190 | Statistically often used |
| **99%** | | 7 | 15 | 32 | 66 | 144 | 290 | Advised in this program |
| **99,9%** | | 10 | 23 | 49 | 101 | 205 | 413 | Advised in this program |
| **100%** | 1 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | The new value |

*Table Annex VIII. The number of steps needed to reach a percentage of the end value. Remark: ∞ = infinite is just theoretically the number of steps needed to reach 100%.*

In a program moving average can be used:

To remove noise on the RC-signal period times

For damping the output PWM signals

See details in the filtering chapters VI.

## Electronic details

Some advices about electronic parts, see below.

### *a) Shunt*

Shunt resistances are not always easily available in small values and for small powers. An alternative is to use resistance wire and made the shunt yourself. Resistance wire is available at Conrad […] from 2.5 Ohm/m and higher. I used Isachrom 60 with a resistance of 5.65 Ohm/m. With a Vsense of 0.15625 Volt (see Annex IV.1) and 1A current limit a resistance of 0.15 Ohm is needed, which is equal to the wire length of 0.15625/5.65= 2.7 cm. The disadvantage is that these wires do not solder very well. This can give an additional resistance.

### *b) Soldering SMD components*

Soldering SMD IC's with traditional soldering iron can be done in the following way. Soldering just one pin, until the IC is in the correct position. Next solder all other pins, do not worry if solder comes between two pins. Take desolder wick and take the surplus of solder away. Take care not to long soldering the IC to prevent over heating of the IC.

## Further reading

Below are some links to other resources and articles on related subjects, and technical documentation relating to the hardware used for this project...

| | |
|---|---|
| Flowcode: | http://www.matrixmultimedia.com/flowcode.php |
| E-blocks | http://www.matrixmultimedia.com/eblocks.php |
| Learning Centre: | http://www.matrixmultimedia.com/lc_index.php |
| User Forums: | http://www.matrixmultimedia.com/mmforums |
| Product Support: | http://www.matrixmultimedia.com/sup_menu.php |