



Robotics Course

Using ezCircuit™ Designer & CoreChart™

Doc Version 1.4 (updated July 2008)

(For eRacer_16m PCB Ver 1.0 and 1.1)



elabtronics

INNOVATIVE CONTROL SOLUTIONS

Office: 51 Byron Place, Adelaide, South Australia 5000

ABN 64 405 458 383

Email: enquiries@elabtronics.com

Web: <http://www.elabtronics.com>

Telephone: +61 8 8231 5966

Fax: +61 8 8231 5266

Table of Contents

1. Introduction.....	3
2. Basic Soldering Workshop.....	4
3. LEDFUN Soldering Project.....	9
4. eRacer_16m Mechanical Construction Manual.....	11
5. eRacer_16m Electronics Construction Manual.....	23
6. Software Workshop Part 1- ezCircuit Designer.....	36
7. Software Workshop Part 2 - CoreChart.....	57
8. Software Workshop Part 3 - eRacer_16m Inputs.....	84
9. Software Workshop Part 4 - eRacer_16m Outputs.....	106
10. eLabtronics Robotic Line Following Competition.....	122
11. Appendix - Further ezCircuit Designer projects.....	123
12. Quick Reference eRacer_16m Port Operation Table.....	129

Introduction

ezCircuit Designer (ezCD) is extremely simple to use to design microchip applications. CoreChart (CC) is an industry strength graphical software developed to simplify the programming of microchips. These leading edge ICT technologies provide the students an opportunity to learn electronics in the context of science and mathematics studies.

On completion of this course, students will be ready to use ezCircuit Designer and CoreChart to develop their own projects with the re-usable eLab16m controller on the eRacer_16m robot.

This course incorporates a number of cross-curriculum contents for different year levels from junior to high schools. It also prepares the students for tertiary level learning in the context of real life industrial applications.

Learning Outcome

This course provides students with an opportunity to gain knowledge and competency in PIC microcontrollers. The students will build and program the eRacer_16m robot. They will gain an understanding of mechanical construction, basic soldering, basic electronic component recognition, electronic circuit construction and programming of the PIC microcontroller which forms the brain of the eRacer_16m robot.

The students will appreciate the critical role of microchips in modern society.

Long term outcome

The aim of the eRacer_16m robotics course is to encourage schools, Tafe and Universities to work together in a Robotics Peer Mentoring program which entails collaborative projects with industry partners.

This widely acclaimed collaborative program is to spark student interest in Science, Technology and industry sought after people and communication skills. It helps the students to identify a pathway to study Science and Technology at Higher Education.

The Peer Mentoring program provides opportunity for Tafe and University engineering students of all year levels in electronics, electrical, microelectronics, mechatronics, computer systems, telecommunication and advanced manufacturing to work with teachers and students in secondary schools.

Copyright & Disclaimer

Original material used in this publication is copy right to eLabtronics and the authors. Permission to reproduce original material must be sought from eLabtronics.

eLabtronics reserves the right to make changes to the product described. The information in this manual has been presented in good faith. eLabtronics and its contributing writers accept no liability for any consequential loss, injury or damage resulting from any omissions or errors which may occur. The specifications presented in this product are not to be deemed as forming a contract either in full or in part.

Web site: www.elabtronics.com

email: enquiries@elabtronics.com

Basic Soldering Workshop

The LEDFUN soldering workshop provides students with basic soldering experience and component recognition.

The students will also learn to handle electronics components with caution which are sensitive to excessive temperature, static electricity, polarity etc.

A poor solder joint, a damaged component due to incorrect component handling or a simple error with component polarity could slow down the progress of the project because it will be time consuming to identify the faults and correcting the errors.

Please ask if you are unsure!

Basic Soldering information

OHSW

1. Always wear safety glasses when cutting wires or soldering
2. Keep work area neat and tidy
3. Wash hands after soldering as solder contains lead and lead is poisonous
4. Any burn should immediately be put under cold running water for at least 10 minutes

Preparing to solder

1. Use an iron with a tip size $\frac{2}{3}$ of the solder pad width
2. Wipe the tip clean on a wet solder sponge
3. Set the temperature to 350 Degrees
4. Bend the leads of the components to avoid stress fractures in the components See **Figure 1**



Figure 1: how to bend the component lead

5. Lead termination **Figure 2**

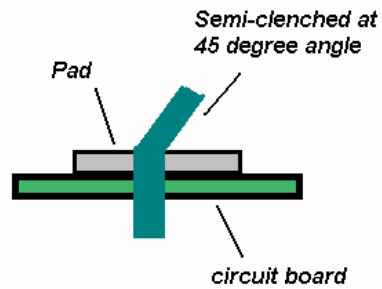


Figure 2: lead termination

6. Semi-clenched see **Figure 3**

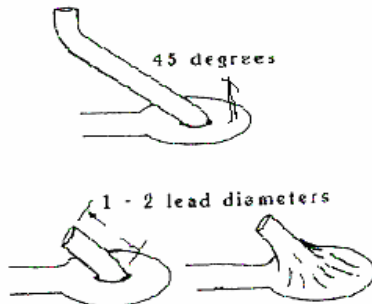


Figure 3: semi-clenched solders

7. Obviously not all components can be semi-clenched, so leave these straight through.

Soldering a joint

1. Solder is commonly made from 60% tin and 40% lead
2. The object being soldered should be kept perfectly still until it has set
3. If a joint is being re-done use of flux (tree rosin) may be required as it helps the solder flow better and spread over the pad rather than sit in a little ball.
4. Before soldering cut the leads at 1x Diameter of the pad
5. Place the iron so it is touching the pad and the lead

6. Put a small amount of solder on the iron between the pad and the lead, this forms a heat bridge. **Figure 4**

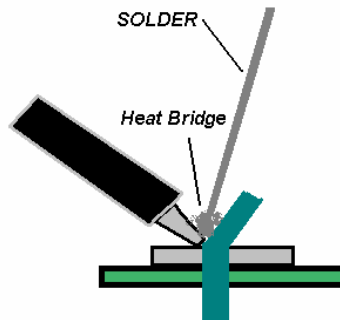


Figure 4: the heat bridge is highlighted

7. When the solder starts to flow, run it over the pad, then up to the tip of the lead to seal the end
8. **Figure 5.** The joint should look like this when finished, it should have a convex and shiny appearance



Figure 5: a good solder joint

9. A joint should only take 2-5 seconds to make, longer than that can damage the board and the pad

De-soldering and removing components

1. To de-solder a component use of either a solder sucker, (ask for a demonstration), solder wick, or copper braid and flux is required.
2. To use copper braid, put the copper braid on the joint.
3. Put the tip of the iron on the copper braid and the solder should be sucked into the braid. Do not keep the heat on for more than 5 seconds as you could burn the board

Electrostatic discharge

The human body generates static electricity, sometimes without you noticing it e.g. rubbing your feet on the carpet. Components can be damaged by static electricity. Try not to handle chips and component boards unless grounded. By touching a large metal object that is in direct contact with the ground helps. Hold parts and boards by the edges. It is for this reason that computer parts arrive in special anti-static bags, anti-static mats, sprays and wrist straps are also available.

Soldering Glossary

What is a Dry Joint?

A dry joint occurs when a joint is not properly soldered, thus creating a hollow space around the component leg and the board. This will result in an unstable circuit because it will often become an open circuit.

Re-solder the components and make sure the solder builds up a nice cone shape with no holes in the middle. Always heat up the pad AND the lead before applying the solder.

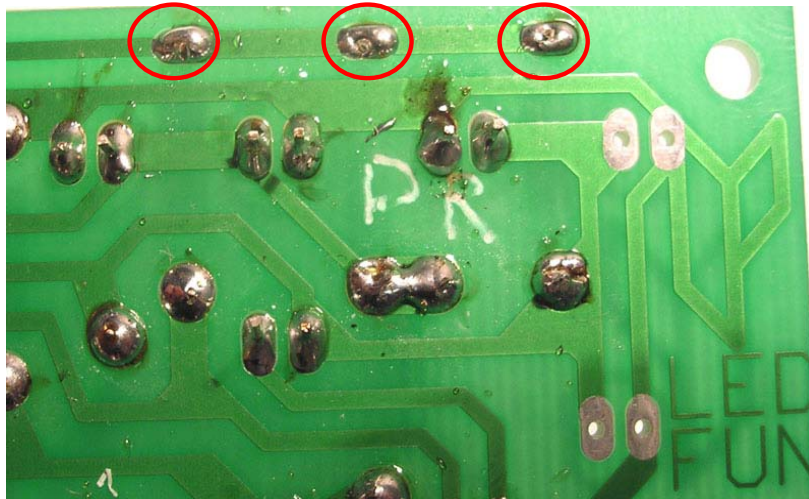


Figure 6: Dry joints are shown in red

Wrong polarity on the Light Emitting Diodes LEDs

It is very important to have LEDs in the right way or else the LED will not turn on when it should and vice versa. By convention, the longer leg is the positive leg. The shorter negative leg should be on the side of the LED that has a flat edge.

Broken Tracks

A broken track or lifted track on the Printed Circuit Board causes problems. Normally this occurs around the solder pads when too much heat is applied to remove the components. Remove the solder on the joints as much as possible before removing the components.

Not the same type of LEDs

Besides the difference in colours, LEDs are different in many other ways, e.g. the current rating, efficiency, size, appearance and more. Be sure to use LEDs with the same characteristic to ensure an even flow of power.

Disconnected wires

This means that the wire is disconnected from the solder joint or the solder joint is broken. This results in electricity not flowing to the required components. To avoid this, twist and tin the wires before soldering them onto the PCB. Make sure there is enough solder to cover the junction point.

Short Circuit

A Short Circuit occurs when the positive terminal is connected directly to the negative terminal. Be sure to check all solder joints are not crossing over to nearby solder pads, which may cause a short circuit and damage or “cook” components.

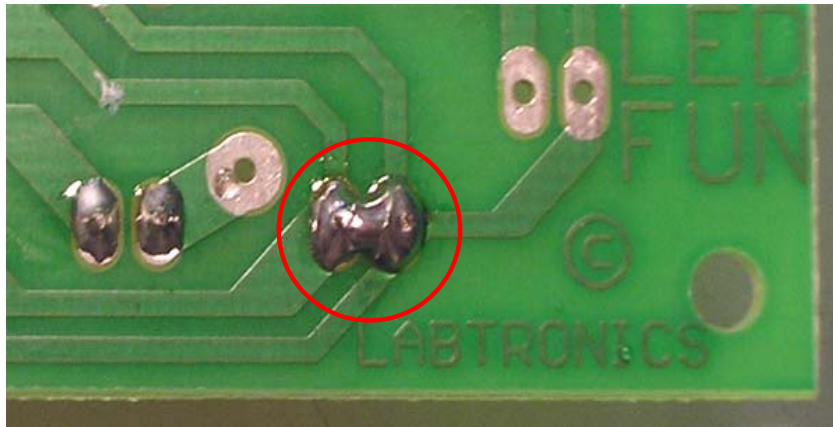


Figure 7: short circuited joints

LEDFUN 8-in-1 Micro-controller Kit

Micro-controllers, or “computers on single chips”, play a crucial role in most industries such as defence, telecommunication, food processing, bio-electronics, ICT, automotive and consumer products. Micro-controllers form the brains of equipment and processes of industries. In late 2006, Microchip Technology Inc (USA) manufactured its 5th Billionth PIC micro-controller chip.

LEDFUN is a low cost electronic kit based on the PIC micro-controller. It is designed to introduce electronics as a fun and exciting subject. After LEDFUN, CoreChart and ezCircuit Designer will take the learning experience to a different level of excitement. These breakthrough technologies spark off creative ideas to use electronics to solve problems in climate changes, pollution and the conservation of energy, water and the environment. See www.elabtronics.com/energysave

The LEDFUN kit demonstrates some basic elements of practical electronics as well as some advanced concepts.

Special care is taken to minimize problems for first time users in the recognition, placement and soldering of components. Documentation is clear and informative.

LEDFUN Kit: 1 PCB 7cm x 3½ cm, tinned, solder masked and top component overlay.

Components: 7 Red LEDs, 6 Resistors, 1 Capacitor, 1 Diode, 1 Piezo Buzzer, 1 Push Button Switch, 1 Battery Holder (3 x AA Batteries not supplied), 1 IC Socket-8 pin, 1 PIC chip.

MODES (8 Modes are set according to Mode Table in the Instruction For Construction section)

MODE 1: Random LED selector. Randomly select one of 6 possible choices.

TO USE: 6 LEDs flash quickly with sound while the push button switch is pressed. On button release only one LED lights up randomly.

USES: Make your own game e.g. every LED represents a move or a function on a board game.

MODE 2: Variable Chaser. 6 LED chaser with sound. Three LED patterns with variable speed.

Patterns:

1. Single LED Chaser – One LED chases another and then loops back to start.
2. Progressive – Each LED progressively lights up and stays on. When all LEDs light up the sequence starts again.
3. Strobe – All LEDs flash on and off at the same time.

TO USE: To vary speed and pattern press switch and hold. The first pattern will start slow and go faster and faster until the pattern changes. Release switch to maintain a speed and pattern.

MODE 3: Binary Counter with Alarm. Count time can be set between 1 – 64 seconds.

TO USE: To set the count time press switch and hold. The Binary Counter will continue counting up to a maximum of 64 seconds. When the desired count time is reached, release the switch. Press and release the switch and the Binary Counter will count down. When the count reaches zero, an alarm siren is set off and the LEDs will flash for 5 seconds.

USES: The Binary Counter can be used as a timer, a door bell, a security alarm or use in “pass the parcel” party games.

MODE 4: Reaction Game – Climb the 6 LED Ladder to test your reaction time.

TO USE: When you hear a “click” and an LED turns on, press the button quickly to keep the LED on. The next LED up the Ladder will turn on and you must press the button quickly again to keep it on as well – until you get to the top. Then all the LEDs will flash and the buzzer will sound. Next the first LED will start to light again but this time your reaction will have to be faster. How many levels can you reach? Have Fun!

MODE 5: Blinking Face – 7 LED face smiles & blinks randomly. It reacts if the switch is pressed.

TO USE: Watch face winks and smiles. Press button to make buzzer sound and face smile. **USES:** Make into a stunning pendant, broach or use as a face on a robot or a doll. You could even use it as an interesting night light for the baby.

MODE 6: Doorbell or Alarm – All LEDs are off until button is pressed. A face with 7 LEDs blinks & smiles. Siren sounds. Makes an interesting door bell – even for deaf people. It could be used as a “Smiley face” Alarm.

TO USE: When the switch is pressed, Smiley face alerts you to someone at the door and sounds the alarm.

MODE 7: Memory / Sequence Game – Remember and repeat a sequence of tones.

TO USE: Press and release button to start. The buzzer creates a series of high and low tones. The challenge is to remember and repeat the tones. The high tone is generated by pressing the button for a short duration and the low tone with a longer duration. When you have the correct sequence the face smiles and the buzzer plays a tune. The sequences get longer and harder.

MODE 8: Electronic Dice using 7 LEDs.

TO USE: To start press the push button switch. The dice “spins” randomly until the button is released. The number of LEDs that light up is the result of the “spin”.

INSTRUCTION FOR CONSTRUCTION:

Solder 270 Ohm Resistors to R1, R5 & R6.

Solder 270 Ohm Resistors as per Mode Table.

A. Solder Diode to D1. Note polarity.

B. Solder Capacitor to C1 (any direction for monolithic capacitor).

C. Solder 8 pin DIL IC Socket to IC1. Align the notches on IC socket and on the PCB label. (To anchor IC socket in the PCB, bend the 4 corner pins. Do not cut pins).

D. Solder Red LEDs as per MODE TABLE. Note polarity, the short leg is negative and is inserted in hole next to the flat edge. Mount the LEDs 5 or 6 mm off the board.

E. Solder battery holder wires to BAT on PCB. Note polarity, +ve (red wire) and –ve (black wire).

F. Cut the red and black wires in two halves for the piezo buzzer and the push button switch. Strip the ends to about 3mm & twist each end tightly.

G. Solder the red wire to the middle silver section of the piezo buzzer metal disc – the black wire to the outer golden section of the disc. Solder the other ends of the wires to B1 on the PCB.

H. Solder wires onto the switch terminals and the other ends to S1 on the PCB (any direction).

I. Insert pre-programmed PIC12C508 IC into DIL IC Socket. Align the notch on the IC with the notch on the IC socket.

J. Insert 3 x AA Batteries (not supplied) in the battery holder. Note polarity. Test the mode you have selected for the LEDFUN. If you have a problem, please check the construction. More LEDFUN information is available on eLabtronics web site under “Resources / Downloads”.

MODE TABLE

MODE	270 Ohm	LEDs
1	none	L1 to L6
2	R4 Only	L1 to L6
3	R3 Only	L1 to L6
4	R3 & R4	L1 to L6
5	R2 Only	L2,L4,L8 to L12
6	R2 & R4	L2,L4,L8 to L12
7	R2 & R3	L2,L4,L8 to L12
8	R2, R3 & R4	L2 to L4,L7,L9 to L11

eLabtronics: 51 Byron Place, Adelaide, South Australia, 5000 Tel: (08) 8231-5966 Fax: (08) 8231-5266

Email: enquiries@elabtronics.com <http://www.elabtronics.com>

eRacer_16m Mechanical Construction Manual

Tools required

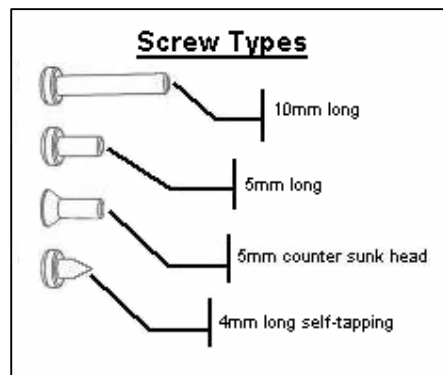
A small bench vice, a small cross-head (Philips) screw driver and a pair of small long-nose pliers.

Important Note

It is important to go through each task step by step in a sequential manner. Further information is available in the eRacer_16m robot CD.

Four plastic wheels assembly

Note that some of the components may be stored inside the battery compartment while others are in a compartmentalized plastic bag. Take each component out only when it is needed as there are a number of small parts which could go missing. Keep the work surface tidy and uncluttered at all times. Follow the instructions sequentially and examine the diagrams carefully to avoid mistakes or breakage.



1. Fit the metal 'L' shaped bracket onto the wheel mount using a 4mm self-tapping screw.

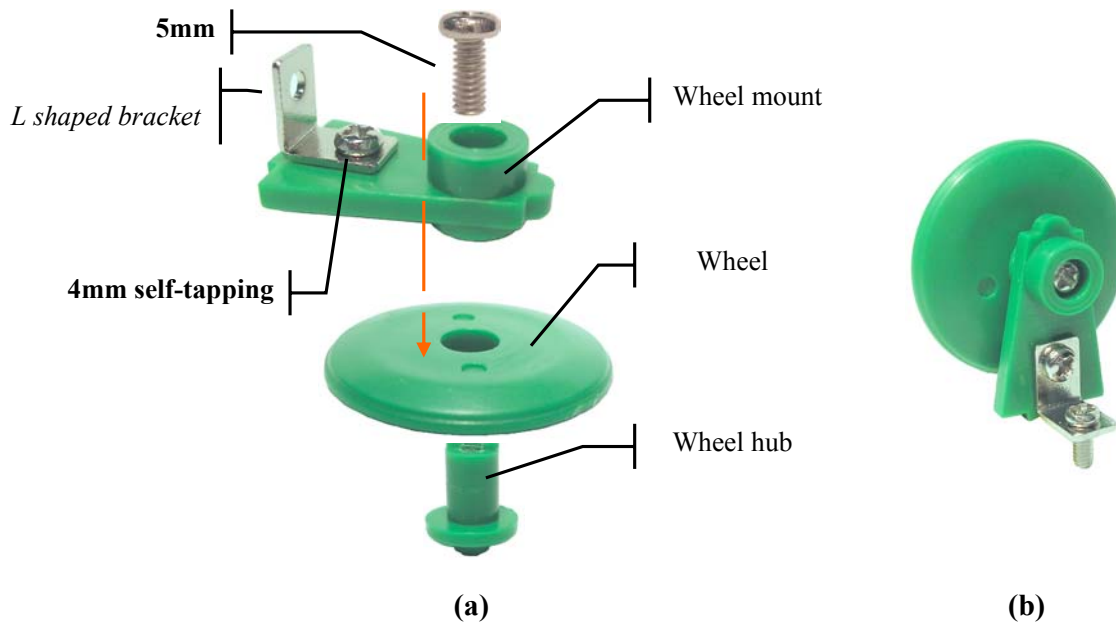


Figure 8 (a) and (b): The small wheel construction and the completed wheel.

2. Put the wheel hub through the wheel and align carefully the wheel mount **notch** with the wheel hub **notch** as shown in Figure 8(a). Fit together with a 5 mm screw.
3. Assemble the remaining three small wheels the same way as above.
4. Fit the four assembled wheels to the eRacer_16m base with the wheels facing outside as shown in Figure 9.

Battery Contacts Assembly

Important note: If the battery contacts come assembled check that these are fitted on the left side as shown in Figure 9. The two double contacts are on the left when the base is positioned as shown in Figure 9. If not, carefully take out all the battery contacts and change them over as shown.

5. Position the eRacer_16m base upside down in the **orientation** shown in Figure 9 below (**the side with the two notches is the *front*!**). Open the battery compartment cover by sliding it. The cover is **not** detachable!

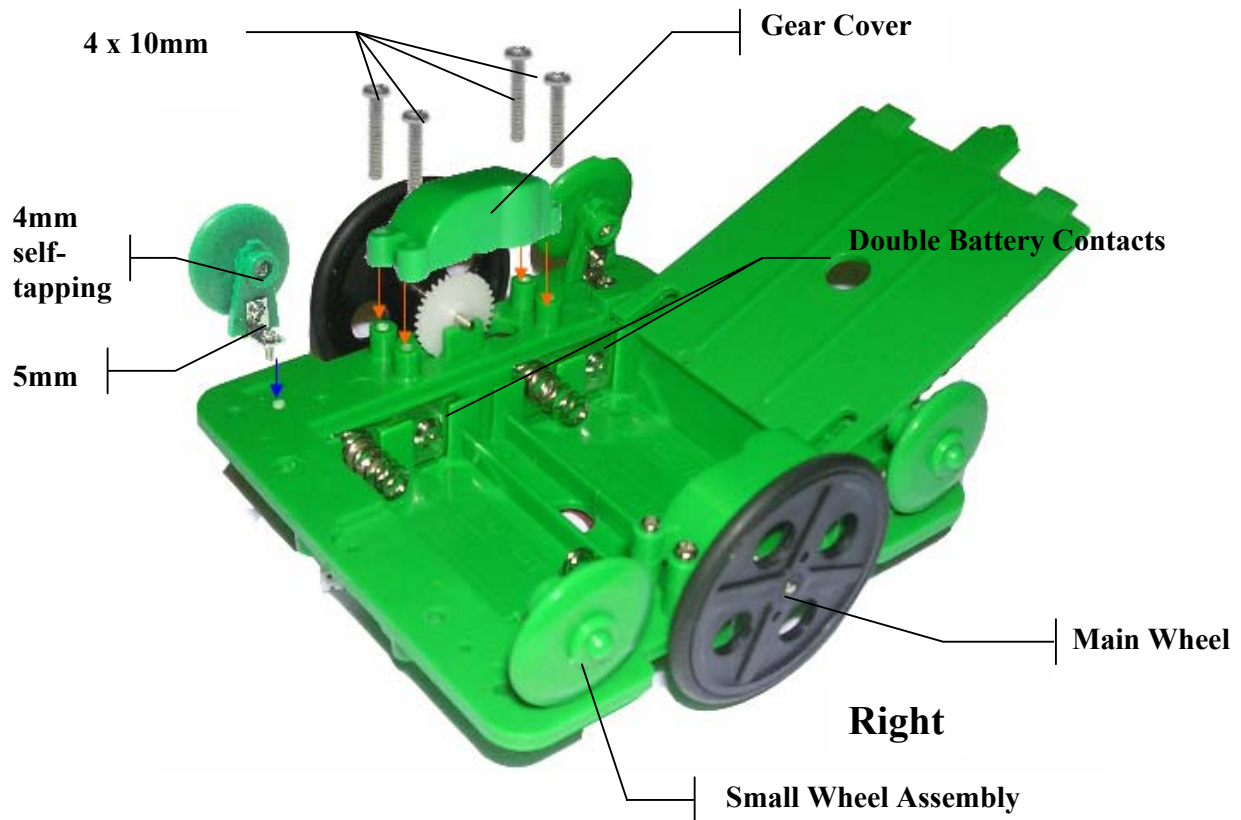


Figure 9: the eRacer_16m mechanical base

6. Ensure each of the **double** battery contacts are inserted into the grooves on the **Left** of the eRacer_16m as shown.
7. Insert the four **single** battery contacts on the **RIGHT** of the eRacer_16m battery compartment (not visible in the diagram) in the right order. Check that the two contacts *without* the spring are lined up with the '+' battery terminals marked on the battery compartment.

Main Wheel Assembly

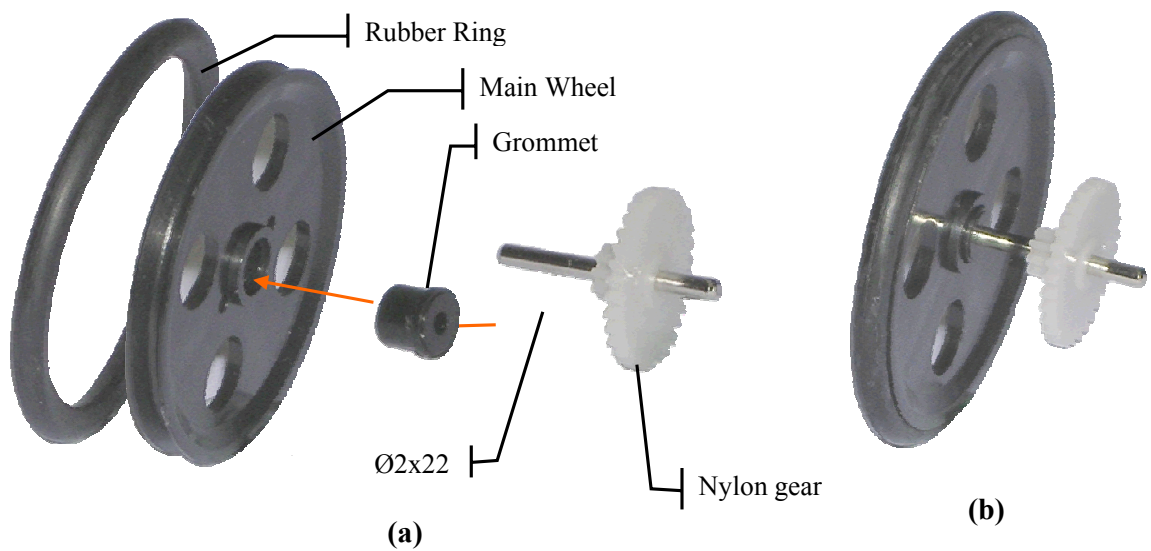
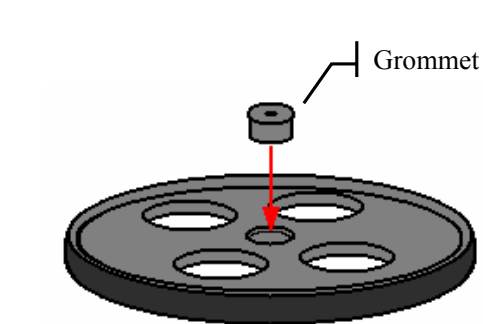


Figure 10 (a) and (b): show the main wheel assembly.

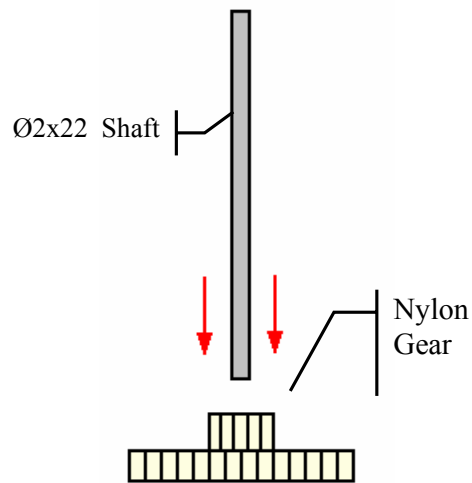
8. Mount the rubber ring onto each of the two main wheels if they are not already assembled.
9. Assemble the main wheel according to steps a) b) c) and d) on the next page.

Main wheels assembly

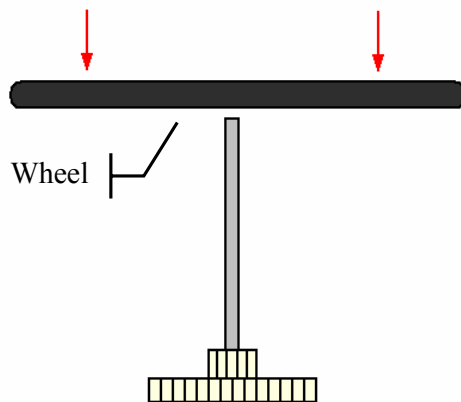
(a), (b), (c), & (d) show the steps for the main wheel assembly.



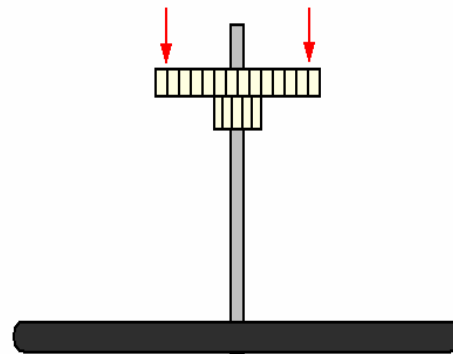
(a) Insert the grommet into the wheel first to avoid cracking it.



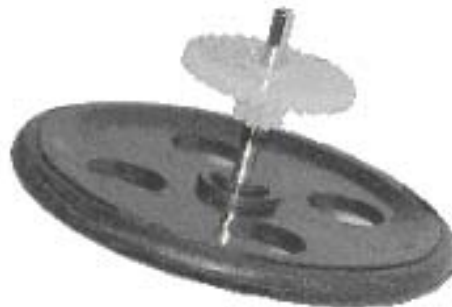
(b) Slide the shaft into the nylon gear.



(c) Using the gear to stand the shaft up, slide the wheel onto the shaft.



(d) Push the nylon gear down the shaft so that the flat side is 4mm from the end of the shaft.



**Completed
Main Wheel**

10. **Note:** Push the grommet onto the main wheel first and **not** the shaft so that it is easier to fit the grommet to the shaft next. (*Make sure it is the correct side of the wheel*). Use a **small bench vice** or **hammer** to gently fit the shaft (already mounted with nylon gear) into the grommet in the main wheel. *The outer edge of the grommet should line up with the end of the shaft – Refer to Figure 10 (b).*
11. Place each wheel into position and fit the gear covers with the 10mm screws. *The “bulging” on the gear covers has to face each other (see Figure 9).* (Note that there is a left and right hand gear cover).
12. Solder a blue (or green) wire between Tag 2 and Tag 3 of the Battery Tags (see Figure 16 Top View of eRacer_16m Base in the Motor and gear test section).
Note: It is difficult to solder this wire after the Motors are mounted.

Motor Assembly

Note: Skip 13 and 14 if the terminal wires are already soldered onto the motors.

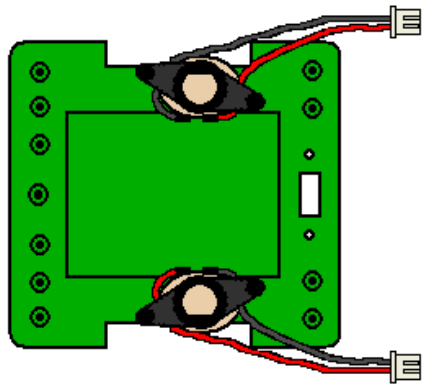
13. Put in short lengths of heat shrink over the Black and Red wires that come with a white two way connector. Solder the Red and Black wires to the each of the motor terminal tabs (see Figure 11a and 11b).
14. Cover each of the motor terminal connections with the heat shrink and gently apply heat by rubbing the soldering iron tip over the heat shrink. This will secure the terminals and prevent movement that may cause the wire to break off from the motor.
15. Push the nylon worm gear gently onto the motor shaft so that about 3 to 5 mm of the shaft is visible between the motor and the worm gear.



Figure 11a



Figure 11b



Insert the motors into respective sockets with the black and red wires facing inward. Secure the motors down with the motor brace. Fasten with the 10mm screws as shown below.

Ensure there is no gap between the motor brace and the eRacer_16m base before securing it with the screws. Some force may be required to squeeze the two together. If there is a gap, the motor will not sit on its place correctly and therefore will not turn the main wheel.

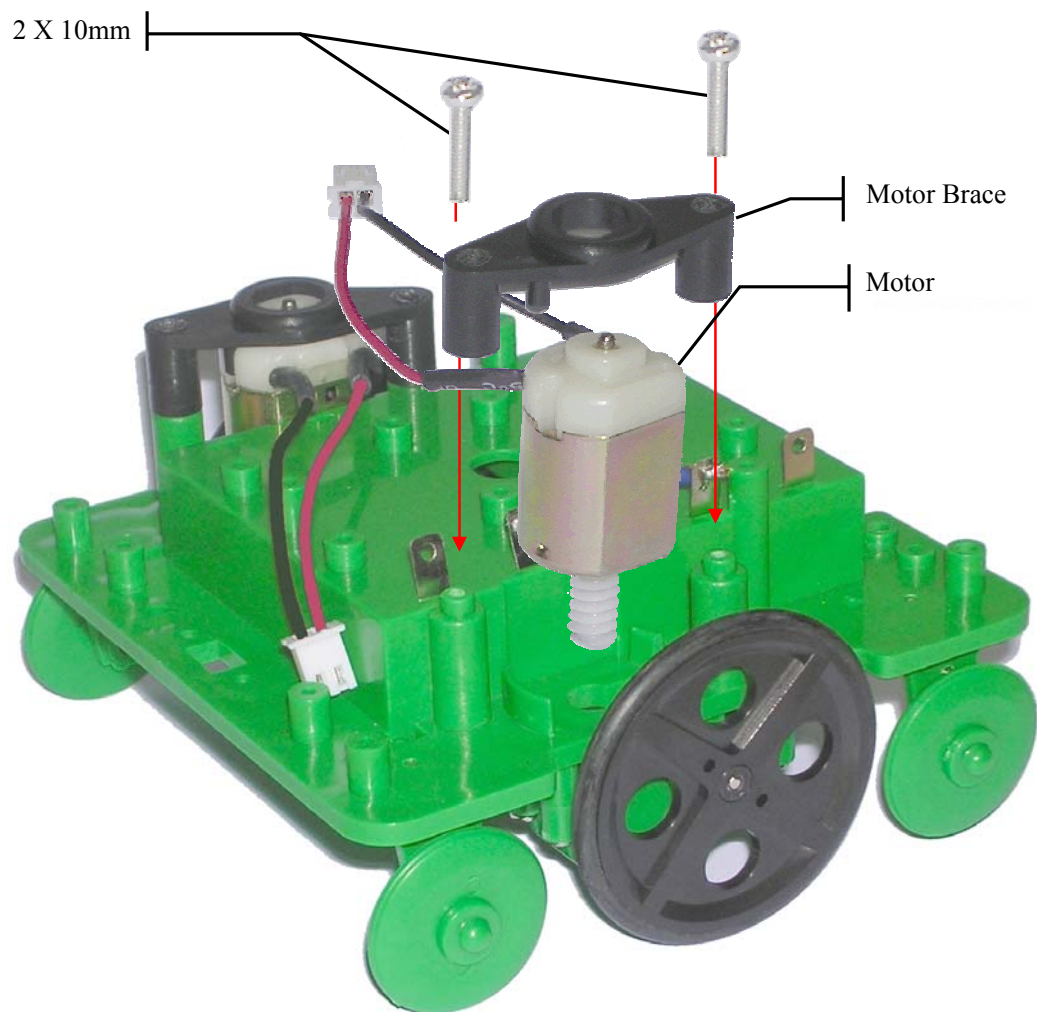


Figure 12: attaching the motors

Note: Lubricate the gears with a tiny drop of sewing or similar machine oil on the worm gear. This will increase the speed of the eRacer_16m and prolong the life span of the gear. As a result the batteries may last longer.



Figure 13: Views of completed mechanical base for eRacer_16m

On completion of the eRacer_16m mechanical construction it is important to check that the mechanical parts are functioning properly.

Quality Control check list

The four small plastic wheels

- ☐ The wheel hub and the wheel mount are screwed firmly and the wheel does not wobble too much.
- ☐ The L-shaped metal bracket is screwed firmly holding the small wheels onto the eRacer_16m board. Do not over tighten as this may damage the parts.
- ☐ Each wheel can rotate freely. If not check that the wheel mount and the wheel hub is fitted correctly as shown in Figure 10.

Battery Contacts

- ☐ Battery cover can be closed and opened without any obstruction.
- ☐ The two double battery contacts are placed correctly.
- ☐ The four single battery contacts are placed correctly.

Main Wheels

- ☐ Both wheels have the rubber rings on.
- ☐ The flat side of the nylon gear is about 4.5 mm from the end of the shaft.
- ☐ The end of the shaft is aligned with the outer surface of the wheel.
- ☐ The gap between the outer side of gear cover and the wheel should be about the same for each side. This allows the eRacer_16m can travel in a straight line.
- ☐ The gear covers are screwed on firmly.

Motor

- ☐ The motor brace is fastened and there is no gap between the brace and the eRacer_16m base.
- ☐ There is a gap of about 3mm between the worm gear and the motor.

Functionality test

Motor and gear tests

Insert 4 new AA batteries carefully (check direction) into the battery compartment.



Figure 14a

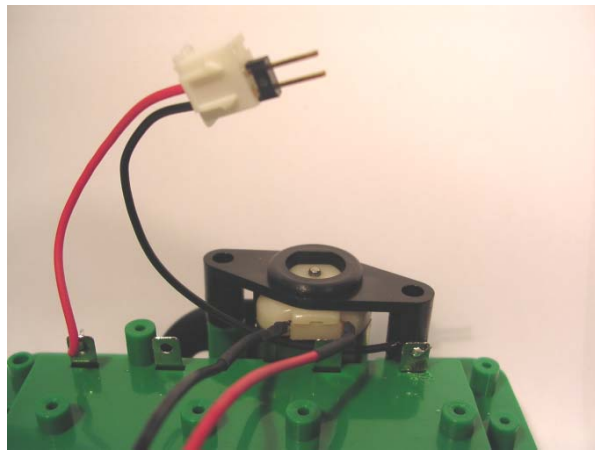


Figure 14b

Insert a *two pin header* into the *three way white battery connector socket* lining up the *pins with the wires* (Figure 14a / 14b). Connect the battery connector socket with the two pin header onto the two way white motor socket (Figure 15). The motor under test should turn. Note the direction.

Disconnect and reverse the connection between the battery connector socket and the same motor socket. The same motor should turn in reverse direction.

Repeat the test for the other motor and note the motor turns smoothly in both directions. Ensure no wires are near the wheels of the eRacer_16m.

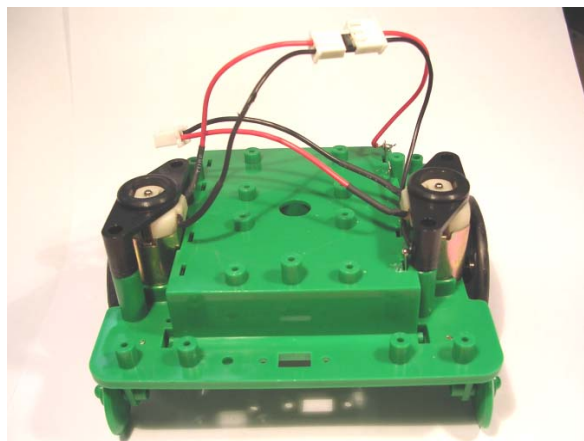


Figure 15: eRacer_16m mechanical base with motors installed

The eRacer_16m sensors PCB is on this end of the robot

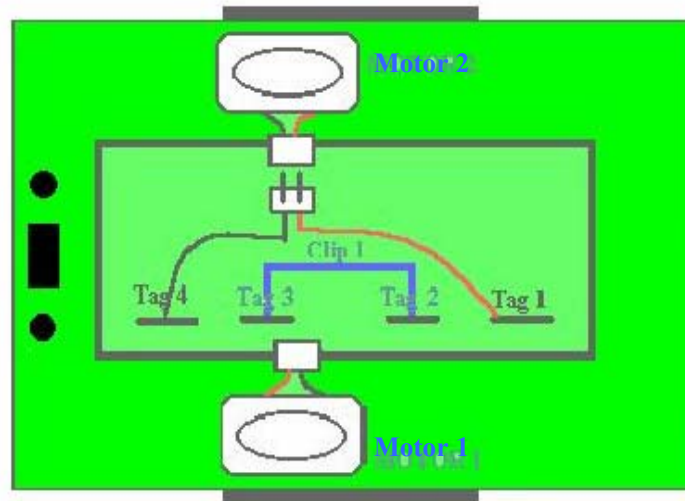


Figure 16: Top View of the eRacer_16m Base

Final check

- ☐ Both wheels are checked to rotate smoothly in both directions. If not check that the batteries are inserted correctly. If possible measure the voltage across the motor leads (with a digital multi-meter) and this should be around 6 volts.
- ☐ Put a tiny drop of oil to the worm gear if this is not done already. If a wheel is still not rotating try loosening its motor brace. If this still does not work the motor may have to be replaced.
- ☐ To ensure the gears are working properly there should be no “crunchy” noise when the wheels are turning. If there is a crunchy noise check the gap between the gear and the motor. Adjust this until the noise disappears.
- ☐ Check that both the wheels are rotating at about the same speed. If not check that each worm gear has a 3 mm gap. If the gap is correct then try loosening the motor brace a little bit. If the wheels are still not turning at about the same speed you may have to replace the motor.

If the wheels are turning properly the mechanical section of the eRacer_16m is now completed. Remove the test leads and proceed to the electronic section.

Component Identification



RESISTOR

Black Stripe
(Negative Lead)



DIODE

**LED
(Light Emitting Diode)**



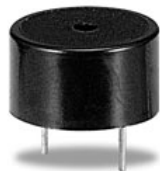
Flat side

Negative Lead

Positive Lead



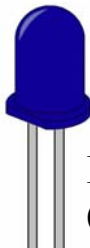
**Electrolytic
Capacitor**



Piezo Buzzer
(Do not remove
sticker)



IR Receiver



**IR LED
(Transmitter)**



Crystal



Heat-Shrink

Ceramic Capacitors



= 1nF



= 10nF



= 100nF



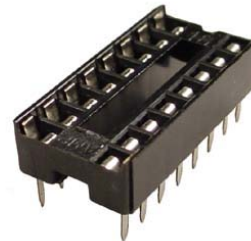
OR



= 22pF



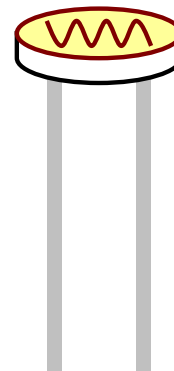
Transistor



IC Socket



**10 Pin
Programming
Header**



**LDR
(Light-
Dependent
Resistor)**

eRacer_16m Electronics Construction Manual

Introduction

This section is divided into three parts. Each part consists of soldering in components, quality check and functional test. These steps ensure that the components are fitted correctly and working properly.

At this stage it is assumed that:

- You have completed the *eRacer_16m Mechanical Construction Manual* and the eRacer_16m mechanical base is functioning properly.
- You have also completed a LEDFUN soldering workshop and that you are confident to solder electronic components onto a printed circuit board (PCB).
- You have knowledge on some basic electronics components (such as resistors, capacitors, LEDs etc.) or your teacher / mentor / friend will provide guidance in identifying their basic functions as you proceed through this module.

The eLab16m controller on the eRacer_16m is re-usable for other applications. The eLab16m has been used in Water Saving Projects, digital clocks, electronic counters, robotics competition timer, large 7 segment displays etc.

So do a good job with your eRacer_16m – be proud of the quality of your work and most importantly, HAVE FUN!

Further information on this section is available in the CD provided. Make sure you go through each task step by step and always double-check your work.

Note:

DO NOT OPEN ALL THE COMPONENT PACKS.

The component pack is separated into 5 parts.

Only open the part of component pack when you need to.

Part I Solder components

Note: Components such as transistor, buzzer and electrolytic capacitor are temperature sensitive and can be damaged if over heated. Some electronic components are polarity sensitive. Refer to Basic Soldering Workshop.

Open the first section of the component pack.

Solder in the Links

1. Solder tinned Link wires into LINK1 and LINK2. **Note:** LINK1 is located next to R4 and LINK2 next to R10.

Solder in the resistors

2. 10k Ω (*brown, black, black, red, brown*) into R1.
3. 390 Ω (*orange, white, black, black, brown,*) into R2, R6, R7, R8, R9.

Note: If you have trouble identifying the resistors, use a digital multi-meter (DMM) to confirm the values.

Solder in the Transistor

4. BC337 Transistor (Bend the middle leg slightly towards the curved surface of the transistor.) into Q1. Leave about 3mm of lead between the transistor and the board.

Solder in the capacitors

5. 100nF (104) ceramic capacitors into C5.

Solder in the Piezo Buzzer

6. Solder in the round black Piezo Buzzer into Buzzer label on PCB. (*Note direction: the '+' long lead must line up with '+' on the board*). Do not peel off the white plastic cover from the buzzer.

Solder in the red LEDs

7. Insert two red LEDs into LED1 and LED2. (*Note polarity of LED: Flat side of LED or the shorter leg is negative.*) These should not be confused with the super bright Green LED with the clear lens or an InfraRed IR LED which is “bluish”.
8. Solder the two red LEDs in LED1 and LED2.

Solder in the lower line tracking LEDs

9. Place a small piece of masking tape on the top of the PCB covering the 8 holes corresponding to LED4, LED5, LDR1 and LDR2.



Figure 17: Grey area shows masking tape to hold LEDs and LDRs for soldering

10. Prepare heat shrinks to the length of the shorter Red LED5 leg. Insert the heat shrinks into the longer positive legs of the Red LED5 and the clear lens Green LED4. Insert both LEDs into the corresponding PCB holes with the longer legs piercing through and the shorter legs **just** piecing the masking tape on the PCB. The masking tape will hold the LEDs in place.
11. Check that the flat edge of LEDs matches up with the LED labels on the PCB.
12. Solder the LEDs to the PCB as shown. Cut off the excessive LED legs protruding from the PCB. *(Leave the masking tape until the LDRs are soldered on to the PCB).*



Figure 18: Soldering the LEDs

Solder in the 10 pin male IDC shrouded header

13. Solder the 10-pin shrouded header into '10PIN' on the PCB. Make sure the header shroud 'slot' matches the 'slot' or white line marked on the PCB.

Quality Control Check list for Part I

Checked by:

Date of check:

Components

- ☐ 10k Ω resistors (brown, black, black, red, brown) in R1
- ☐ 390 Ω resistor (orange, white, black, black, brown) in R2, R6, R7, R8, R9
- ☐ Tinned Link wire in LINK1 and LINK2
- ☐ Bright Red LEDs in LED1, LED2 and LED5
- ☐ Bright Green LED (clear to green) in LED4
- ☐ 10 pin Male IDC header (shrouded) in 10PIN
- ☐ BC337 Transistor in Q1 (next to R2)
- ☐ Black Round Piezo buzzer in "Buzzer"
 - ☐ Correct polarity
 - ☐ White plastic tag still in place.
- ☐ 100nF (104) capacitor in C5
- ☐ Component leads cut to approx 3mm from solder joint

Quality Inspection

- ☐ No overheating / dry joints
- ☐ 10 pin header in 10 Pin

Insert the batteries the right way round and leave them in the battery compartment for the rest of the course. Make sure the battery power is switched on (using the ON / OFF sliding switch) before you do any testing.

Switch off the battery power as soon as each test is completed.

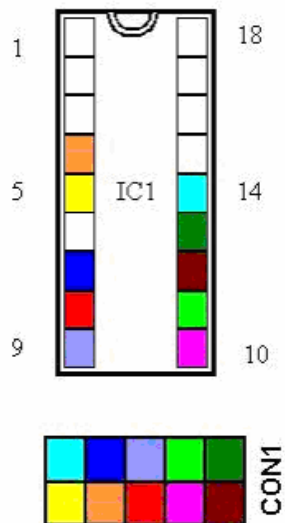
Functionality test

You will need a digital multi-meter (DMM). Set it to DC voltage measurement.

NOTE: IF THE PIC16F819 CHIP IS IN THE eLab16m CONTROLLER BOARD THEN ASK YOUR TEACHER OR MENTOR TO REMOVE IT!!!

- ☐ IC1 is the 18-pin PIC16F819 chip on the eLab16m board. Remove the PIC16F819 chip in order to perform the next tests. Connect the eRacer_16m board to the eLab16m board using the 10 pin ribbon connector (Refer to Figure 21)
- ☐ Place the DMM probes across **pin 5** and **pin 14** of **IC1 socket** with the **positive red** probe of DMM on **pin 14**. The reading should be around 5.5 volts.
- ☐ Short pin 14 and pin 17 LED1 should light up.
- ☐ Short pin 14 and pin 18 LED2 should light up
- ☐ Short pin 14 and pin 3 LED4 should light up
- ☐ Short pin 14 and pin 13 LED5 and the small yellow LED on the eLab16m control board should light up
- ☐ Short pin 14 and pin 1 Buzzer should sound.

Remove the ribbon connectors before going on to the next stage.



Part II

Open the second section of the component pack

Solder in the resistors

14. 33k Ω (*orange, orange, black, red, brown*) into R3, R15.

Solder in the capacitors

15. 10nF (103) ceramic capacitors into C1 and C2

Solder in the 16 pin male IDC shrouded header

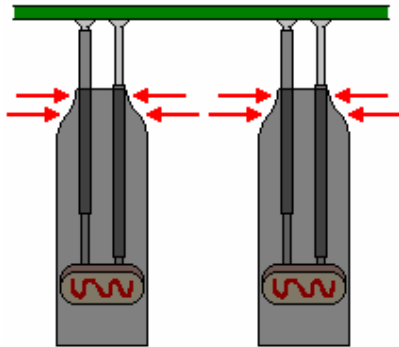
16. Solder the 16-pin shrouded header into '16PIN' on the PCB. If there is a black line on the 16PIN label, match the shroud 'slot' with the black line. If the black line is not present, match the shroud 'slot' with the white line. The shroud 'slot' should face towards Infrared Transmitter marked IRT on the PCB.

Solder in the 2 pin male IDC shrouded header

17. Solder the 2-pin shrouded header into 'Left' and 'Right' labels on the PCB. Make sure the header shroud 'slot' faces towards the edge of the PCB.

Solder in the LDRs (Light Dependent Resistors)

18. Insert heat shrinks the same lengths as the LEDs into one leg of the two LDRs. Insert both LDRs into the corresponding PCB holes with the legs piercing through masking tape on the PCB. Adjust the height to about the LEDs height. The masking tape will hold the LDRs in place.



19. Solder the LDRs to the board. Use a larger heat shrink to cover the **body** of each of the LDRs but leave the sensor ends uncovered. The large heat shrink **must not be shrunk too tight** so that it can be adjusted up and down the LDRs.

20. If necessary apply hot glue to the legs of LDRs and super bright LEDs to strengthen the connections as shown below.

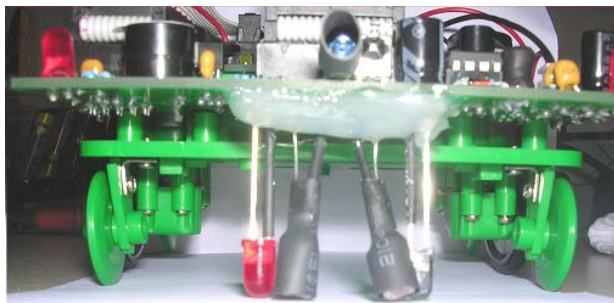


Figure 19: front view of the eRacer_16m with the LEDs and LDRs in place.

Quality Control Check list for Part II

Checked by:

Date of check:

Components

- ☐ 33k Ω resistors (orange, orange, black, red, brown) in R3 and R15
- ☐ 10nF capacitors in C1 and C2
- ☐ Light Dependent Resistor (LDR) in LDR1, LDR2
- ☐ 16 pin male IDC header (shrouded) in 16PIN
- ☐ 2 pin male IDC header in “Left” and “Right”.

Quality Inspection

- ☐ 16 pin male IDC header (shrouded) in 16PIN
 - ☐ Long length pins facing Upwards (unsoldered)
 - ☐ Open end of the header facing towards IRT
- ☐ LDR in LDR1 and LDR2
 - ☐ Both LDRs on bottom side of PCB
 - ☐ Thin heatshrink (1mm, 2cm) on one leg of both LDRs
 - ☐ Thick heatshrink(5mm, 2cm) on each LDR to allow light to enter only through their ends
 - ☐ LDR legs not tangled up
- ☐ 2 pin male IDC header in “Left” and “Right”.
 - ☐ Long length pins facing Upwards (unsoldered)
 - ☐ Open end of the headers facing towards nearest edge of the board

Functionality test

Slide the switch to “ON” position for this test.

You will also need to connect the eRacer_16m board to the eLab16m board using the 10pin and 16pin ribbon connectors (Refer to Figure 21)

- ☐ Put a short wire across pin 4 and 1 of IC1 socket. Sound will emit from Buzzer when push button is not pressed. Sound will cease when push button is pressed.
- ☐ Place a short wire across pin 2 and 1 of IC1 socket. Sound will emit from Buzzer when LDR2 is facing a light source like the ceiling light. Sound will cease when LDR2 is covered (place finger on LDR2 lens).
- ☐ Place a short wire across pin 12 and 1 of IC1 socket. Sound will emit from Buzzer when LDR1 is facing a light source like the ceiling light. Sound will cease when LDR1 is covered (place finger on LDR1 lens).

Part III

Solder in the resistors

21. 10 Ω (brown, black, black, gold, brown) into R4.
22. 220 Ω (red, red, black, black, brown) into R13
23. 4.7k (yellow, violet, black, brown, brown) into R5 and R12
24. 2.2k Ω (red, red, black, brown, brown) into R11
25. 10k (brown, black, black, red, brown) into R10 and R14

Note: If you have trouble identifying the resistors, use a digital multi-meter (DMM) to confirm the values.

Solder in the 1N4148 diodes

26. There are 5 diodes
 - a. 3 1N4148 signal diodes (glass capsule with a black stripe)
 - b. 1 Zener diode (glass capsule with a white stripe)
 - c. 1 1N5819 diode (bigger black capsule with a grey stripe)

Solder the 1N4148 signal diode into D1, D2, and D3 only after making sure it is in the right direction. (*Make sure the black stripe on the diode matches the white strip marking on the board*). Note: The smaller diode may be in the second section of parts. Do not open the whole section - remove only the diode.

Solder in the Zener diode

27. Solder the Zener diode in D4. Match the white stripe on the Zener with the PCB.

Solder in the 1N5819 diode

28. Solder the diode in D5. Match the grey stripe on the diode with the PCB.

Solder in the IC socket

29. Solder the 8-pin IC socket in IC1. Match the socket notch with the PCB near L1.

Solder in the Radial Inductor

30. The Radial Inductor looks like an Electrolytic Capacitor. It has a black rubber cover. Solder the Radial Inductor (1/2 amp 100 μ H) in L1. **Note:** The Radial inductor does not have polarity. It can be mounted in any direction you choose.

Solder in the capacitors

Electrolytic Capacitors are the large “drink can” looking components while the Monolithic Capacitors are flat and yellow in colour. The values of both the Electrolytic and Monolithic Capacitors may be found on the components itself.

31. Solder the (330 μ F/25V) Electrolytic Capacitor into C3. Note polarity.
32. Solder the (100 μ F/25V) Electrolytic Capacitor into C4. Note polarity.
33. Solder the (100nF) Monolithic Capacitor (104) in value to C7
34. Solder the (1nF) Monolithic Capacitor (102) in value to C8

Solder in the Transistor

35. BC337 Transistor (Bend the middle leg slightly towards the curved surface of the transistor.) into Q2. Note: Q2 is located in between R13 and C7.

Solder in the Infrared IR Transmitter and Receiver

36. Solder the Infrared Transmitter LED (tinted blue LED) into IRT. The longer leg is positive. Line up the flat edge of LED lens with the IRT label. Leave the leads at least 10mm long and bend it forward as shown.

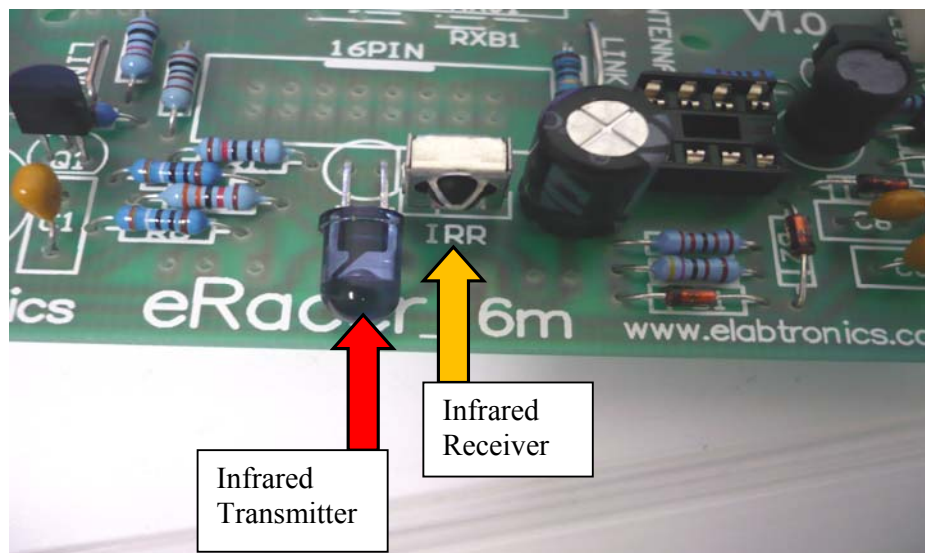


Figure 20

37. Insert a 3cm length heat shrink into the IR LED. Leave about 3 mm over the top of the IR LED. This will ensure light only comes out from the exposed end. Use the soldering iron to carefully heat the heat shrink **around the IR LED legs**.
38. Solder the 3-pin **IR Receiver** (the component in a metal casing) into **IRR**. Make sure the IRR receiver faces towards the nearest edge on the board.

Insert the Timer IC

39. Insert the 8 pin Timer IC into the 8-pin IC socket. Check direction is correct.

Mount the completed PCB onto the eRacer_16m robot

40. Use the 4 small screws to tighten and secure the eRacer_16m PCB board onto the eRacer_16m Robot.

Connect the Ribbon Connectors

41. The eLab16m controller board and eRacer_16m board are connected by the 16 pin and 10 pin ribbon connectors. The connection for the 10 pin ribbon connector is straight forward. Follow the steps carefully with the 16 pin connector described below.

Ensure the 16 pin (2 rows of 8 pins) ribbon connector is connected to the 2 rows of the 24 pins (3 rows of 8 pins) header pins closest to the two IC chips on the eLab16m board.

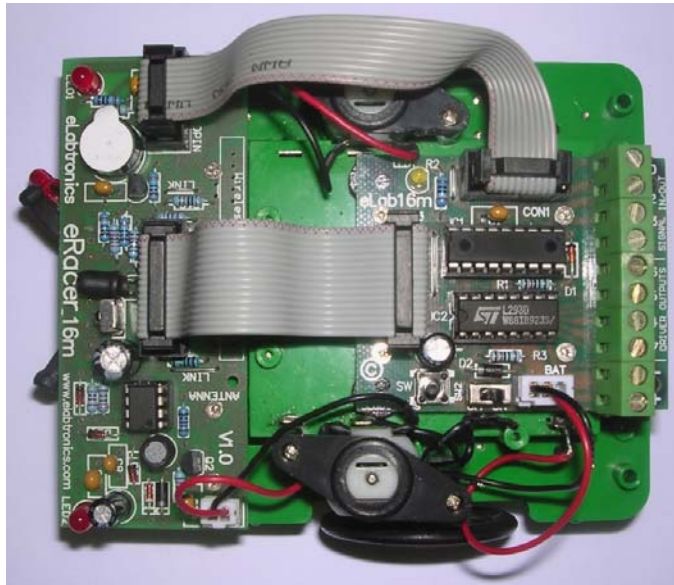


Figure 21

Quality Control Check list for Part III

Checked by:

Date of check:

Components

- ☐ 10 Ω resistor (brown, black, black, gold, brown) in R4
- ☐ 220 Ω resistor (red, red, black, black, brown) in R13
- ☐ 4.7k Ω resistor (yellow, violet, black, brown, brown) in R5
- ☐ 2.2k Ω resistor (red, red, black, brown, brown) in R11
- ☐ 4.7k Ω resistor (yellow, violet, black, brown, brown) in R12
- ☐ 10k Ω resistor (brown, black, black, red, brown) in R10 and R14
- ☐ 100nF (104) capacitor in C7
- ☐ 1nF (102) capacitor in C8
- ☐ 1N4148 diodes in D1, D2, D3
- ☐ 4.7V/200mA zener diode in D4
- ☐ 1N5819 diode in D5
- ☐ BC337 Transistor in Q2
- ☐ 8-pin IC Socket into IC1
- ☐ Timer IC is inserted into the 8-pin IC Socket
- ☐ IR LED (blue tint) in IRT
 - ☐ Flat side of IR LED lined up with IRT symbol on PCB
 - ☐ IR LED facing off the edge of PCB
 - ☐ Heatshrink placed over IR LED, allowing only light to escape out the end
 - ☐ IR LED legs not tangled up

Quality Inspection

- ☐ No overheating / dry joints
- ☐ IR sensor in IRR
 - ☐ IR sensor facing the nearest PCB edge

Functionality test

- ☐ Remove the PIC16F819 chip if it is in the IC socket in order to do the next tests.
- ☐ Ensure that the left motor leads are connected to “Left” on the eRacer_16m board.
 - ☐ The left motor 2-pin female connector fits snugly into the “Left” 2-pin male shrouded header.
- ☐ Ensure that the right motor leads are connected to “Right” on the eRacer_16m board.
 - ☐ The right motor 2-pin female connector fits snugly into the “Right” 2-pin male shrouded header.
- ☐ Short pin 14 and pin 16. Left motor turns clockwise
- ☐ Short pin 14 and pin 15. Right motor turns clockwise
- ☐ Short pin 14 and pin10. Left motor turns anti-clockwise
- ☐ Short pin 14 and pin 11. Right motor turns anti-clockwise
- ☐ If all tests are satisfactory then insert the rest of the PCB mounting screws and tighten.

Software Workshop Part 1- ezCircuit Designer

Introduction

Having built and functional tested the eRacer_16m robot it is useful to show how ezCircuit Designer is used to design the electronic control circuits of the robot.

ezCircuit Designer also generates a number of important design documents such as a schematic diagram of the circuits, a part list, a test procedure for the circuits, a CoreChart test program to test the output circuits.

The schematic file can be exported to a PCB program such as Proteus.

The CoreChart test program forms the template of the control program for the user to program the eRacer_16m robot.

A critical function of ezCircuit Designer is that it automatically generates the microcontroller configuration settings. This is done when the user connects the Input and Output circuits to the PIC microcontroller. The microcontroller configuration settings form an important of the CoreChart test program.

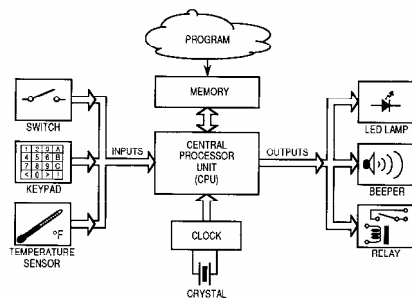
CoreChart is an intuitive microcontroller programming language for the PIC microcontroller. The PIC chip controls the eRacer_16m robot.

Important features of the PIC microcontroller:

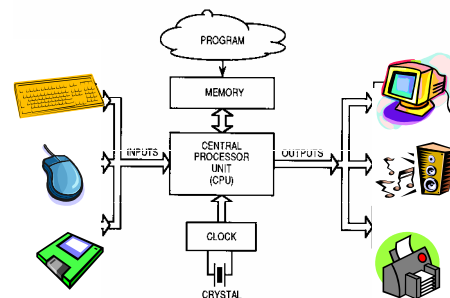
Before using ezCircuit Designer it is important to understand some key features of the PIC microcontroller e.g. Port Structures and how these Ports are configured.

A PIC microcontroller is a small stand-alone computer system designed for the purpose of control whereas a personal computer (PC) is primarily designed to process large amounts of data quickly. The PIC chip has all of the components on a single chip, such as CPU, RAM, ROM, inputs and outputs.

A typical Microcontroller system



A simplified Personal Computer system



Note the similarity of a microcontroller and personal computer system

Introduction to Port Structure of the eRacer_16m Robot

The PIC16F819 chip on the eRacer_16m robot has 2 Ports or independent work areas, namely Port A and Port B. Each of the 2 ports has 8 pins. Each of these pins can be set as an input circuit e.g. power source, switch or an output circuit e.g. buzzer, motor etc.

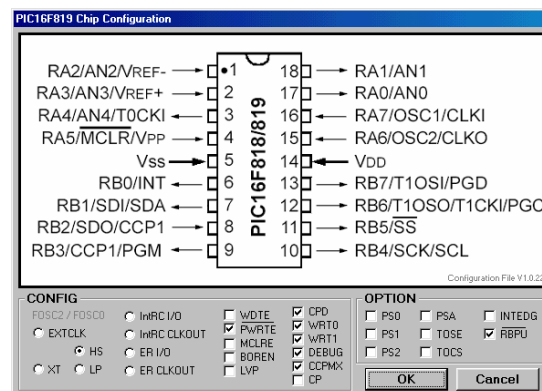
The pin configuration for the eRacer_16m is automatically generated by ezCircuit Designer as shown in the Table of Port Operation below.

Port	Bit	Control	Function
B	0	(not used)	
B	1	(not used)	
B	2	IR Receiver	Input
B	3	IR LED Transmitter	Output
B	4	Left Motor Forward	Output
B	5	Right Motor Reverse	Output
B	6	LDR1	Input
B	7	LED5	Output
A	0	RED LED1	Output
A	1	RED LED2	Output
A	2	Buzzer	Output
A	3	LDR2	Input
A	4	LED4	Output
A	5	Push Button	Input
A	6	Right Motor Forward	Output
A	7	Left Motor Reverse	Output

Table of Port Operation for eRacer_16m

You can take a look at the chip configuration but do not change any settings!

In CoreChart, double click on the **START** icon and a chip configuration window will pop up (see below). Note the directions of the arrows are consistent with the Table. The outward arrows are for Outputs and inward arrows for Inputs. Click the **Ok** button to close the chip configuration window. **Warning: DO NOT CHANGE ANY SETTINGS!**

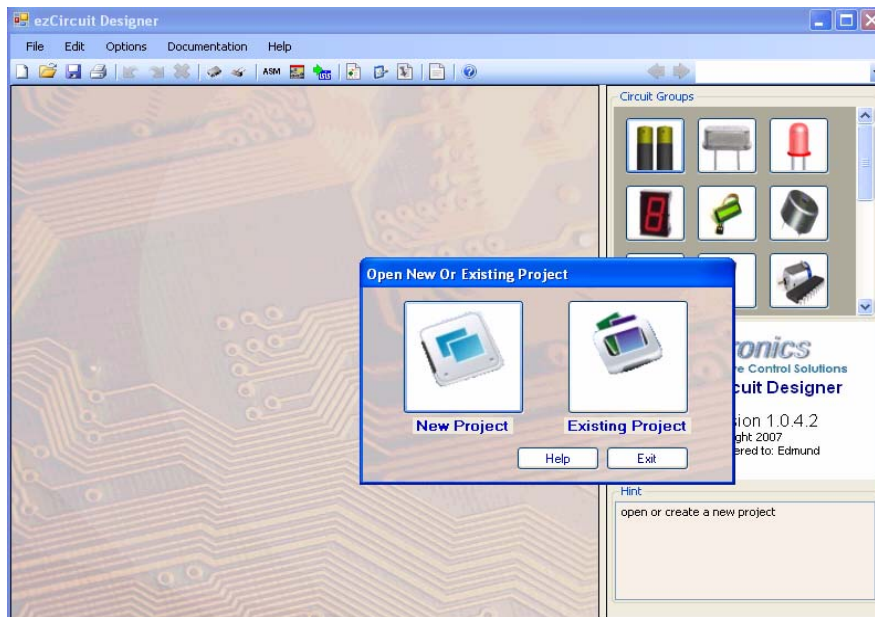


PIC16F819 chip configuration generated by ezCircuit Designer

Practical 1

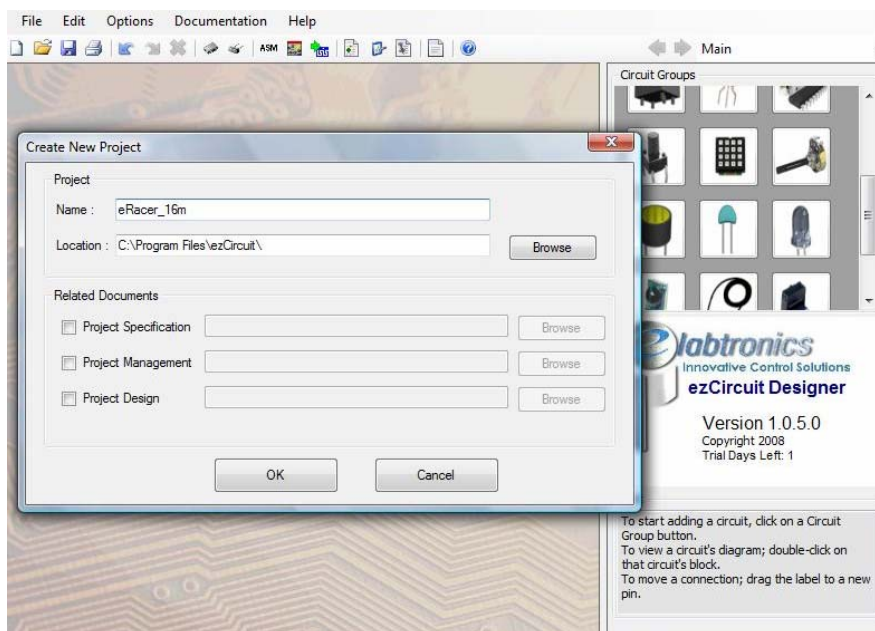
Using ezCircuit Designer to design the input / output circuits of the eRacer_16m robot

Open ezCircuit Designer. Select “New Project”

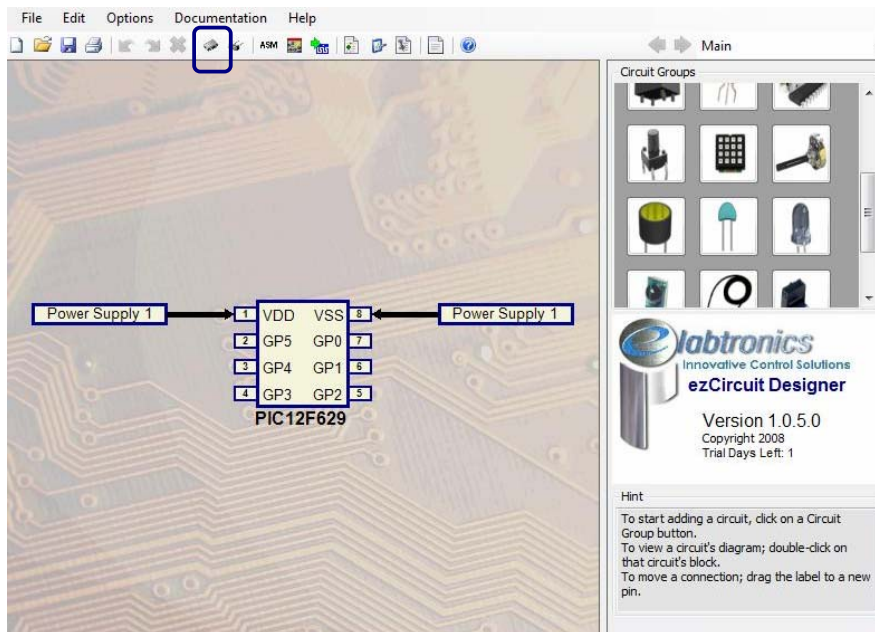


A new window will pop up.

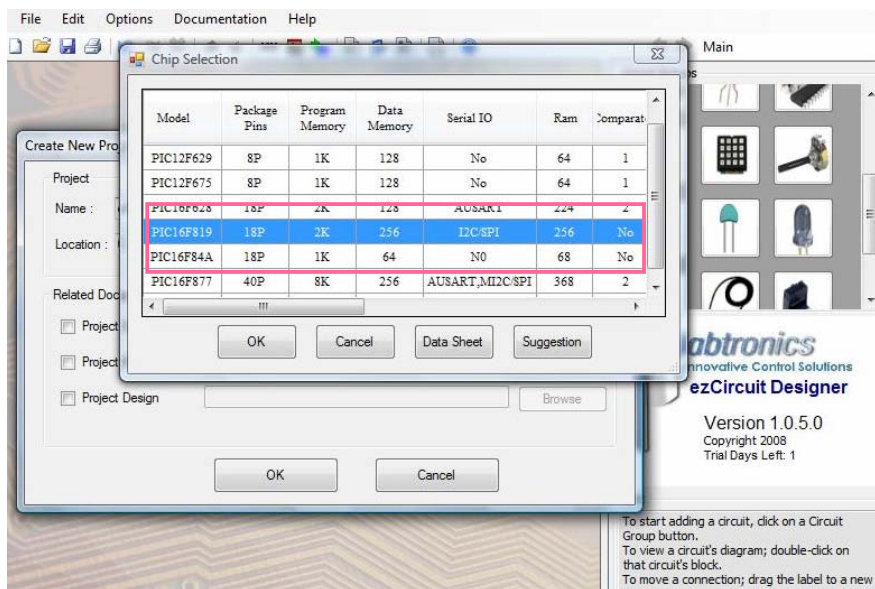
Save the project name as “eRacer_16m”



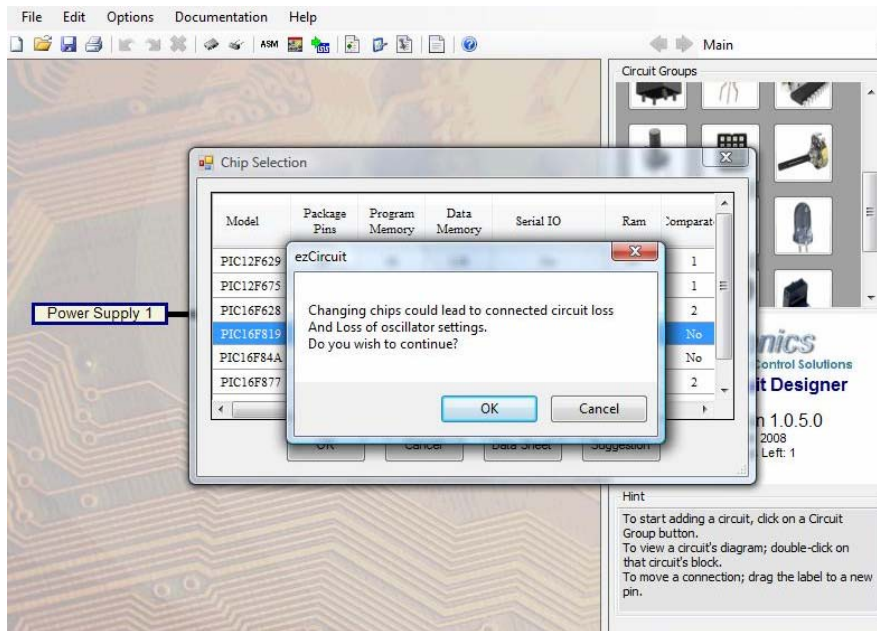
Click on the Chip selection button shown circled in blue.



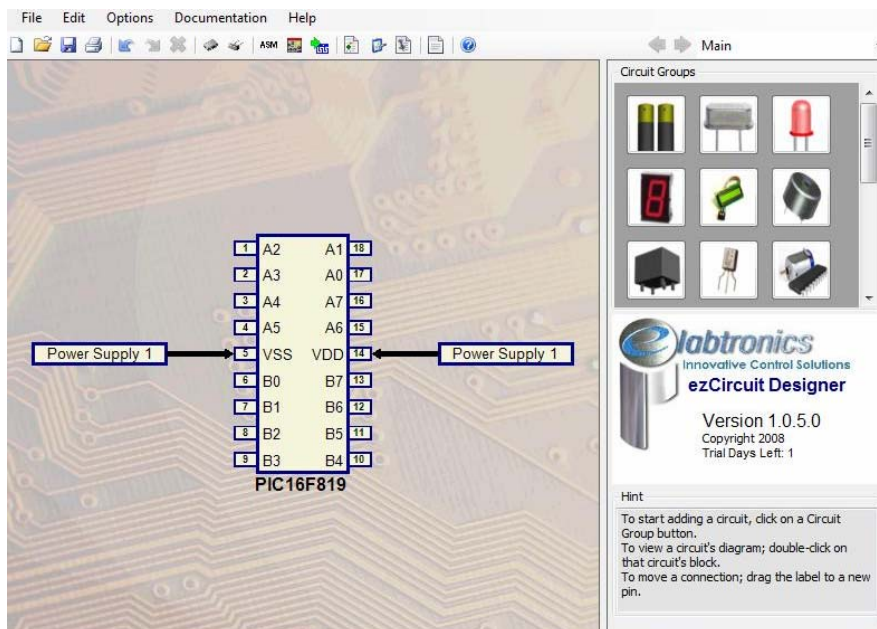
A new window pops up, select PIC16F819 and click “OK”



Another window will come up with a warning message about the effects of changing the chip. Click “OK”.



At this stage the screen should look like this



You are now ready to design a new ezCircuit Designer project with the selected chip.

The next step is to connect input and output circuits to the chip.

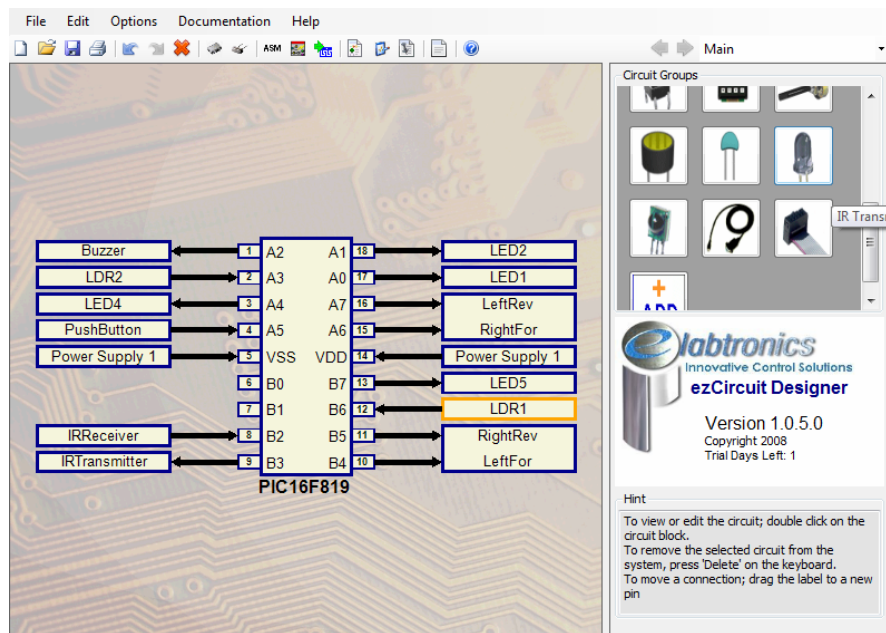
The selected chip and connections are shown on the left of the screen.

The smaller window on the top right is called the “Circuit Groups” window. Input and output circuits are selected from this window for connection to the chip.

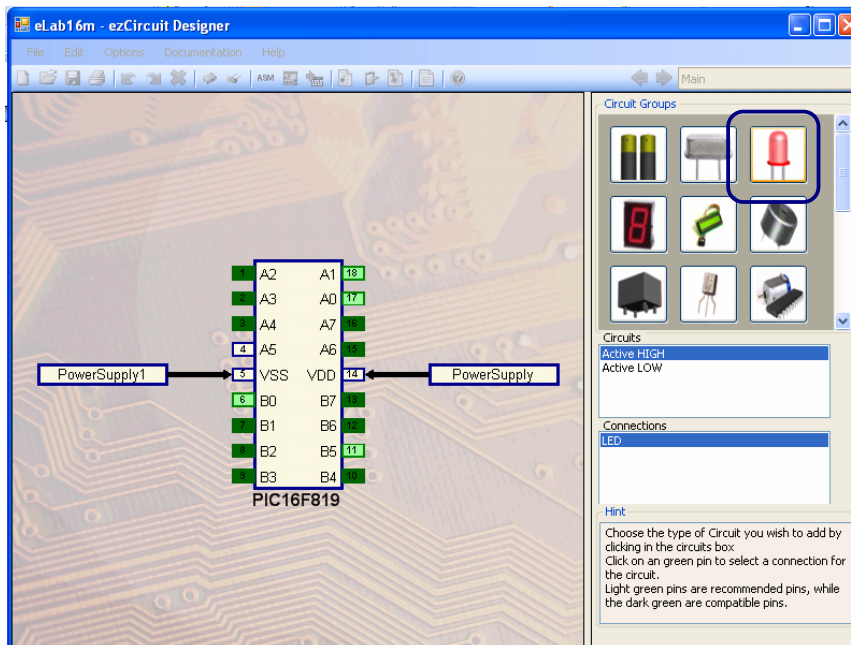
Note1: When a circuit in the Circuit Groups window is selected by a click, all the possible connection points on the chip turn green. For example, click on a LED circuit, all the connection points will turn green except A5 (Pin 4) and VSS and VDD.

Note2: By ‘hovering’ the cursor over a circuit in the Circuit Groups window, the circuit name and function will appear. This will help to identify a circuit.

The objective here is to build the circuit displayed in the figure below.



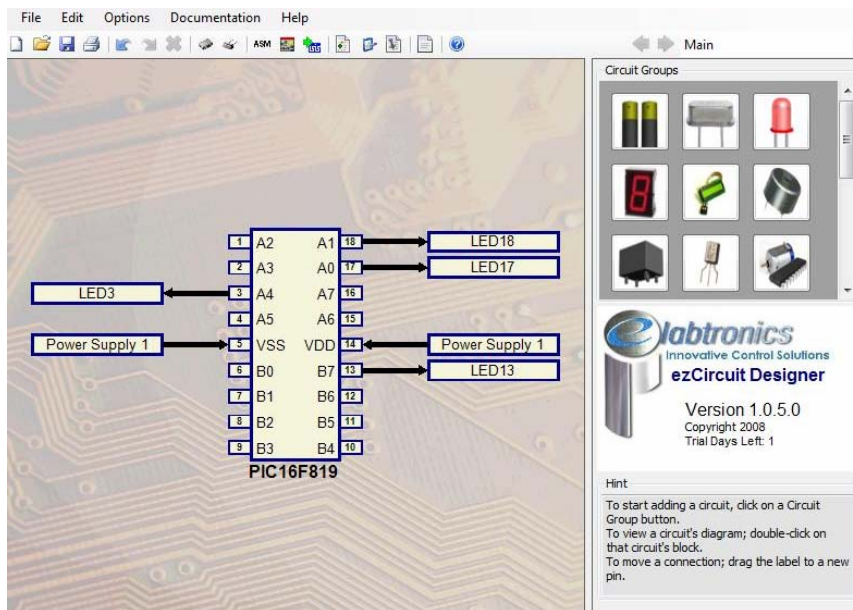
Click on the LED in the Circuit Groups window. Select **Active HIGH** in the **Circuits** window.



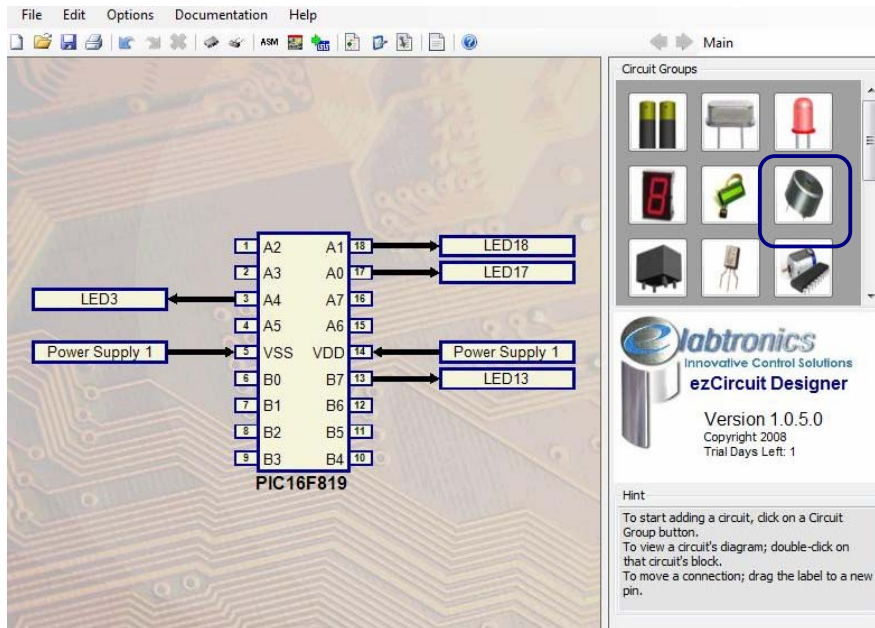
Click on A1 (Pin 18) to connect the LED to Pin 18.

ezCircuit Designer assigns a name to the LED. The circuit can be renamed later.

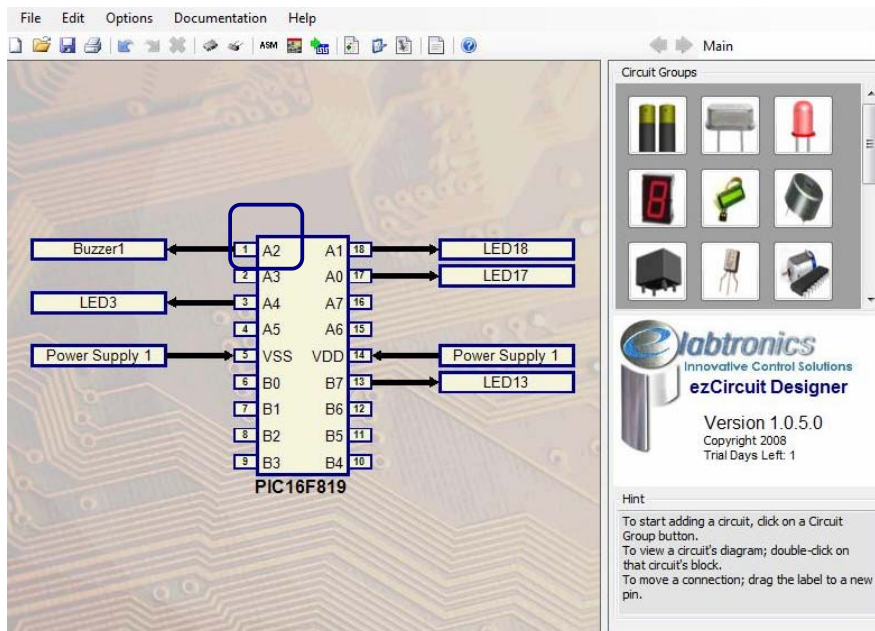
Repeat this step to connect LEDs to A0 (Pin 17), B7 (Pin 13), and A4 (Pin 3)



Click on the buzzer in the Circuit Groups window.

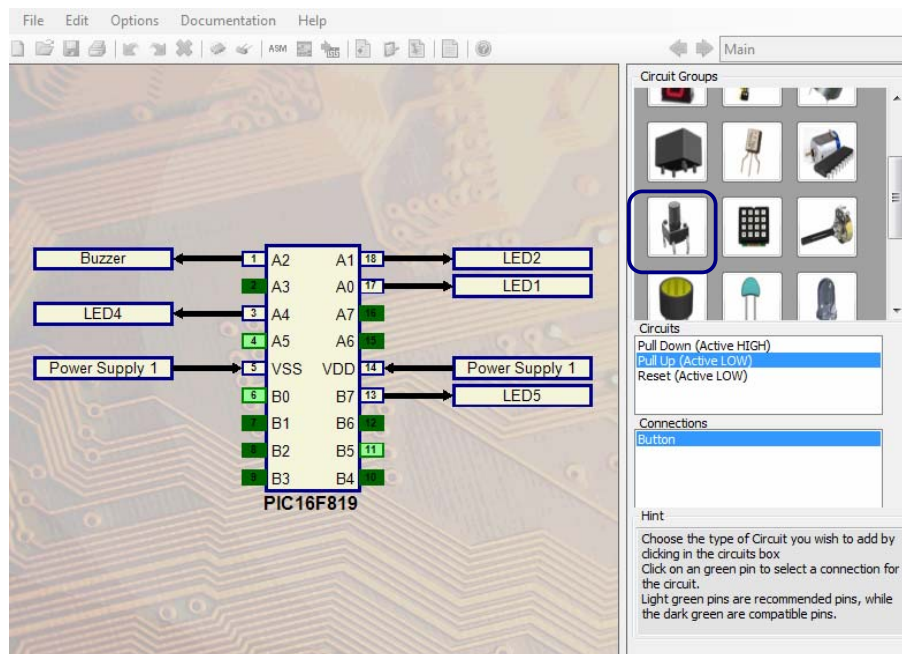


Connect the Buzzer to the microcontroller by clicking on A2 (Pin 1).

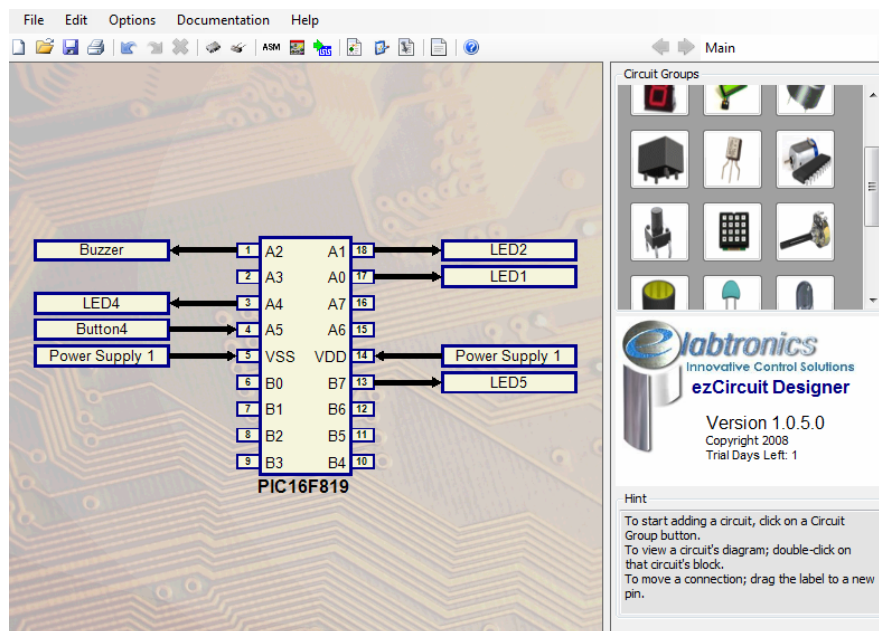


Select the push button from the Circuit Groups window.

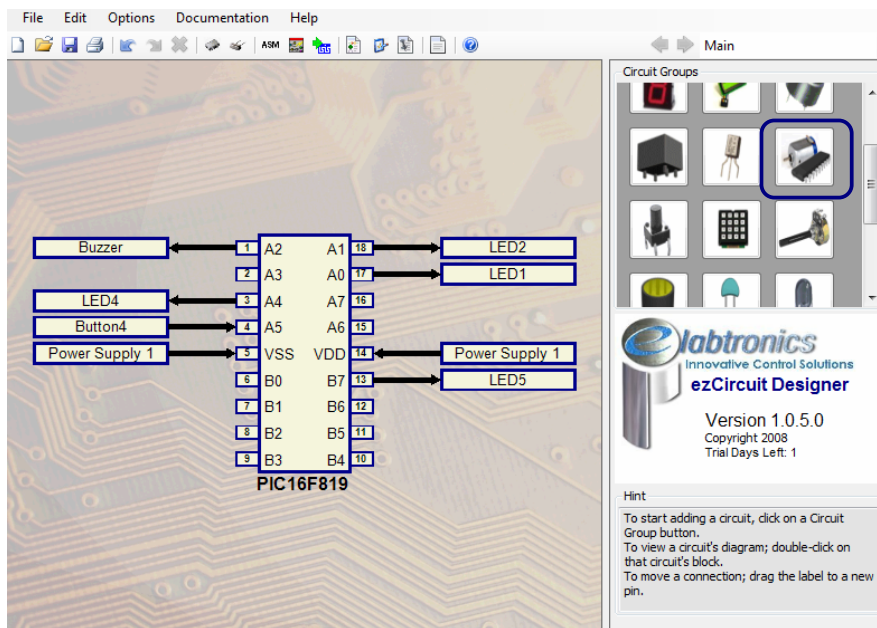
Select **Pull Up (Active LOW)** in the **Circuits** window.



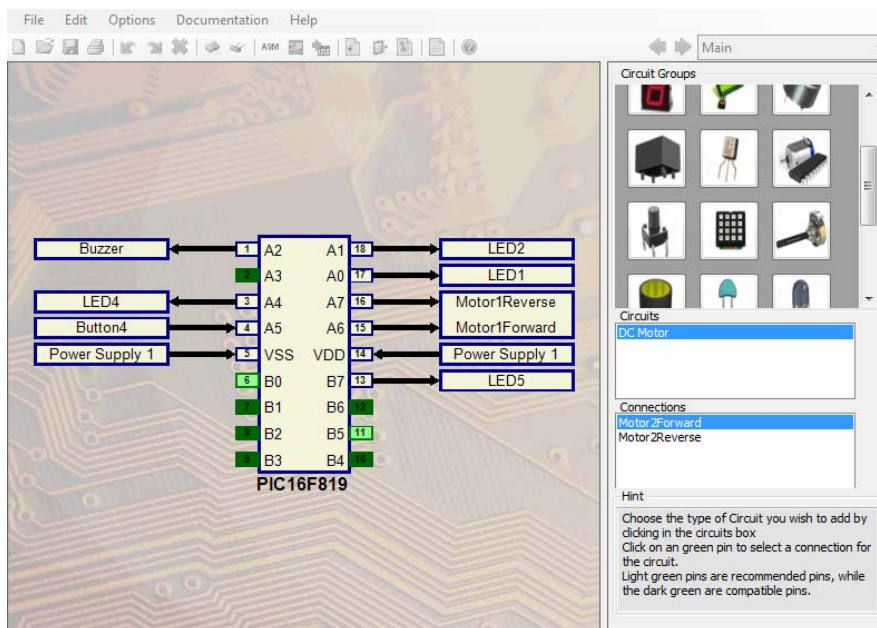
Connect the push button to A5 (Pin 4).



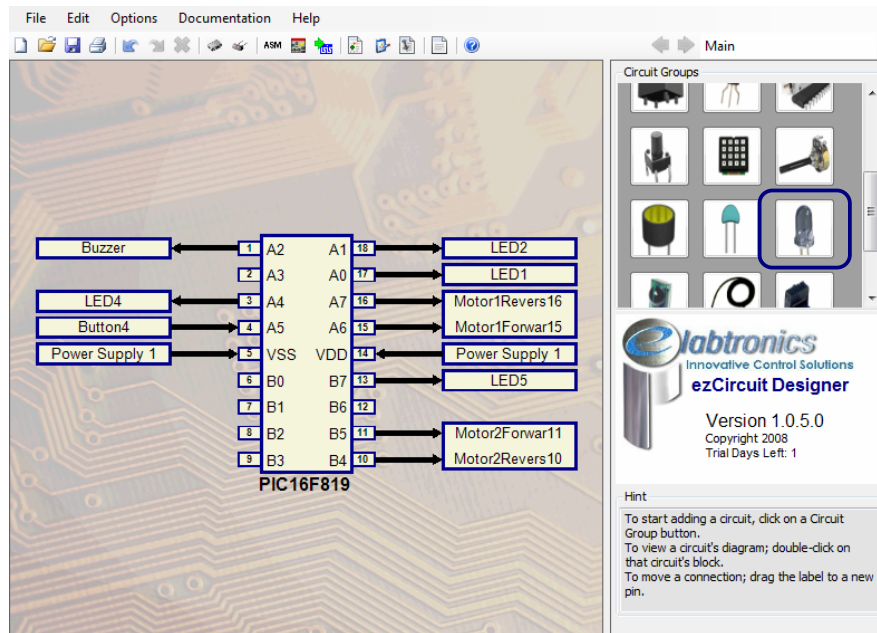
Click on the Motor in the Circuit Groups window.



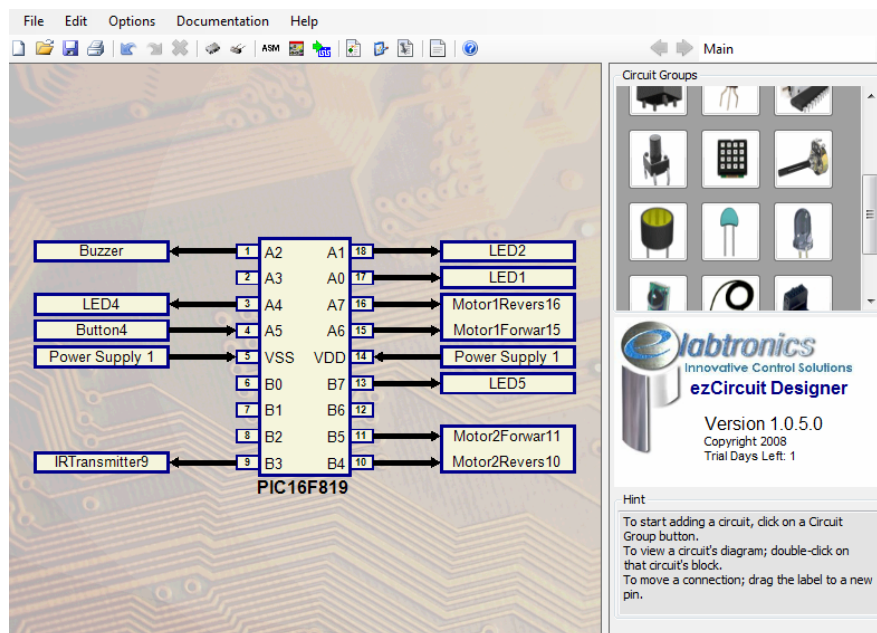
Connect the motor to A6 & A7 (Pin 15 & Pin16). Repeat this step for second motor at B5 & B4 (Pin 11 & Pin10)



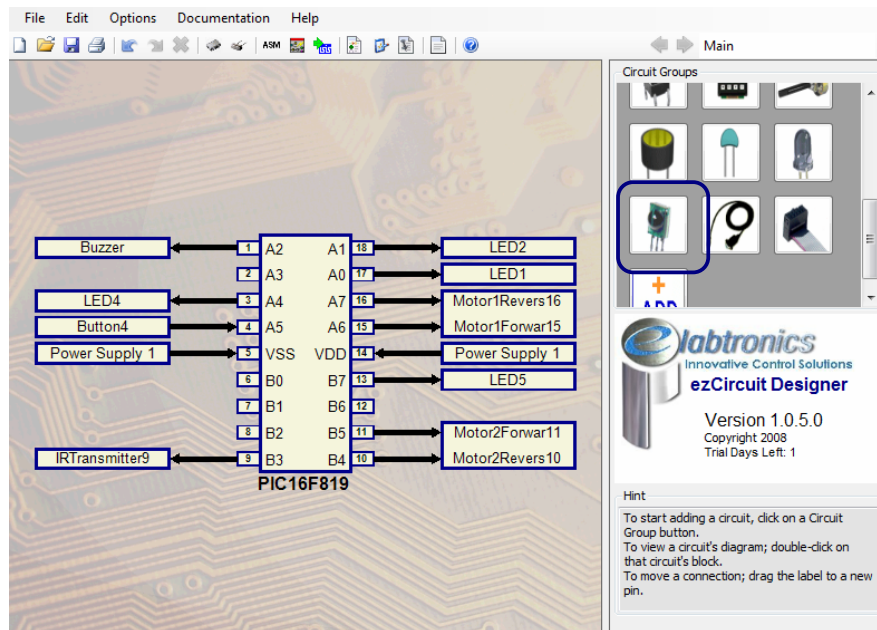
Select the IR Transmitter LED (bluish colour) in the Circuit Groups window.
Note: This is not a normal LED. The normal LED is shown in red colour.



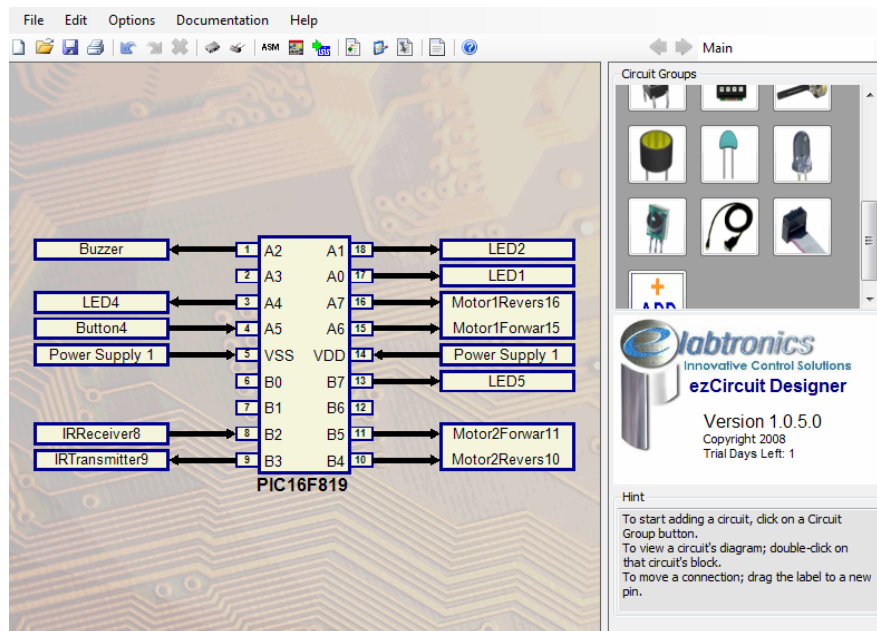
Connect the IR Transmitter to B3 (Pin 9).



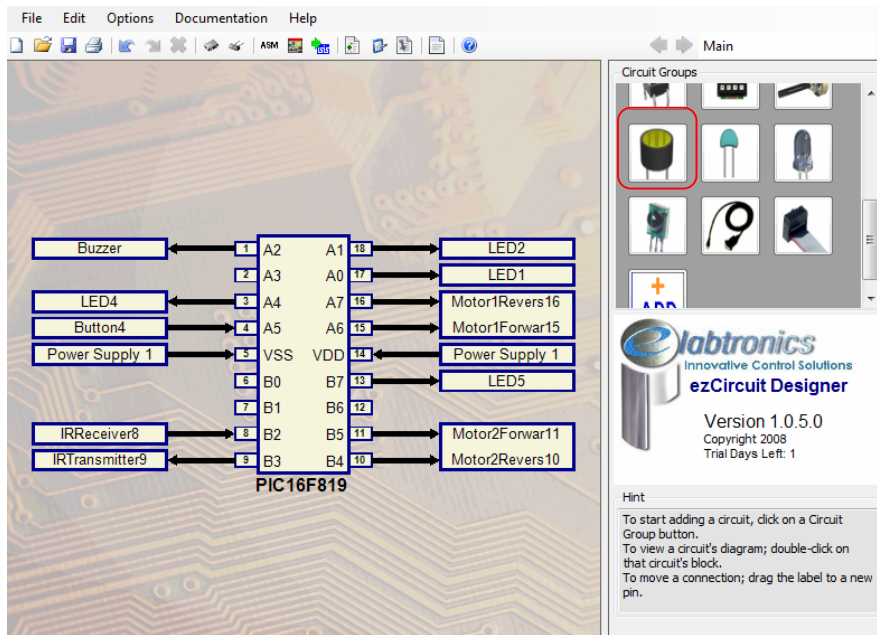
Select the IR Receiver in the Circuit Groups window



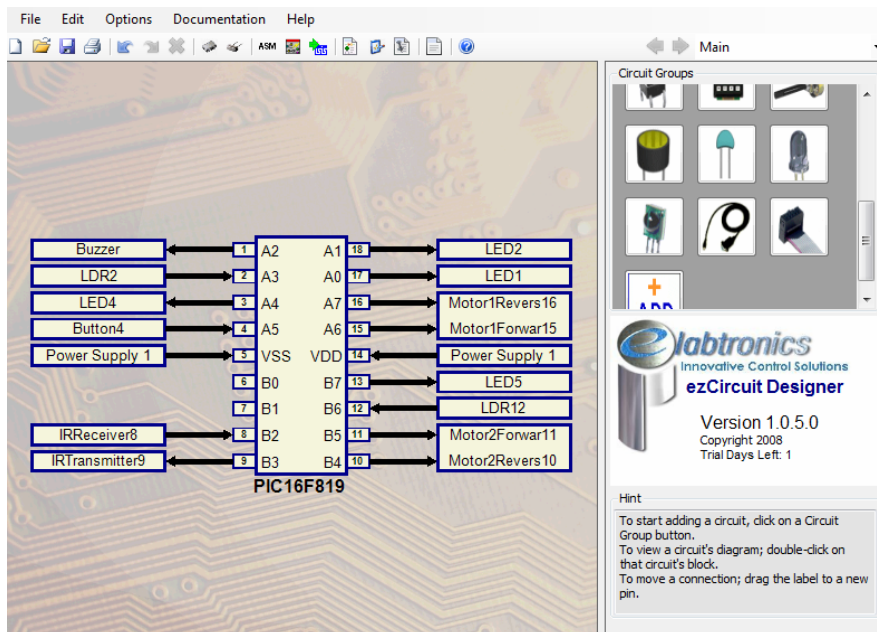
Connect the IR Receiver to B2 (Pin 8)



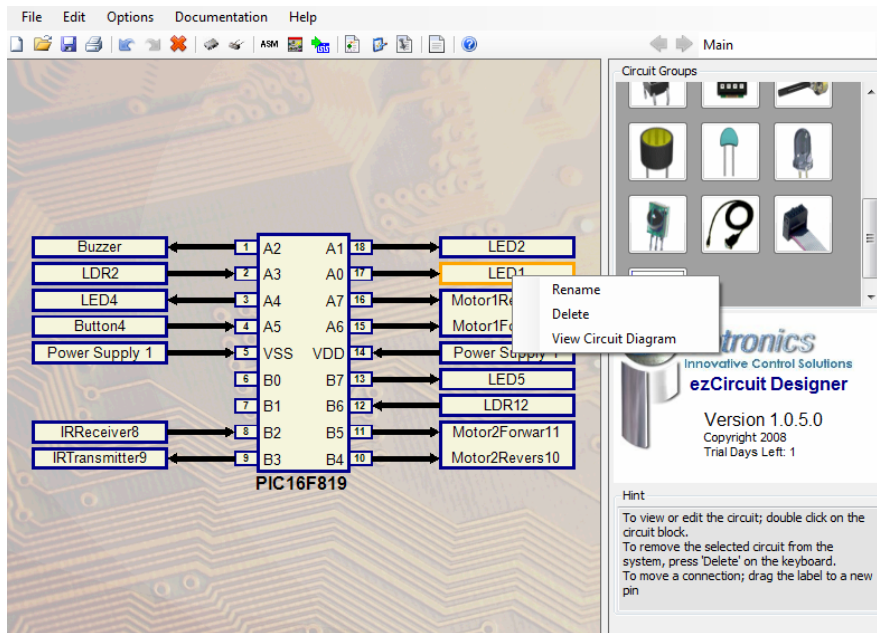
Select the Light Dependent Resistor (LDR) from Circuit Groups. The default Active High setting is correct.



Connect the LDR to A3 (pin 2). Connect another LDR to B6 (pin 12).



Screen showing all the connected circuits



Rename circuits:

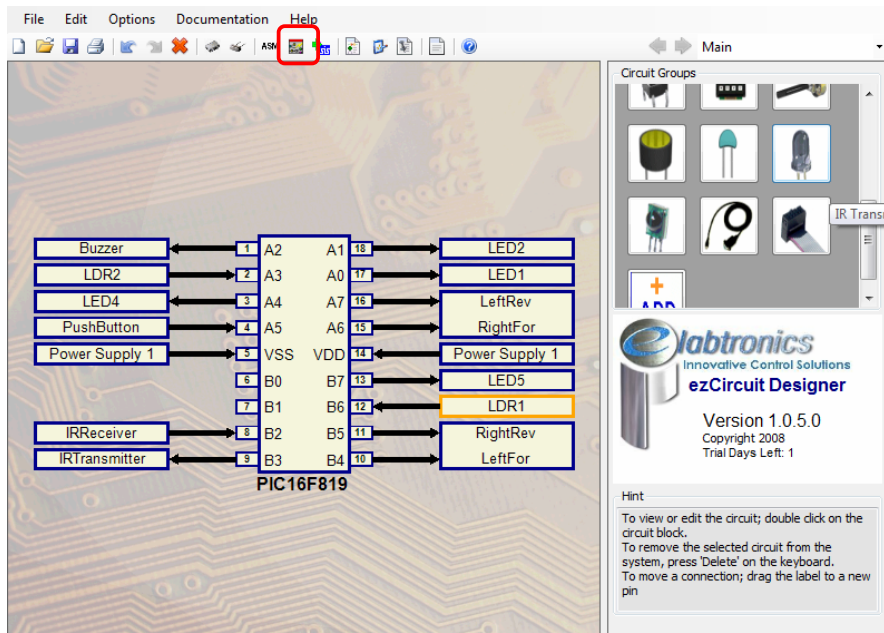
To rename a circuit, right click on the circuit, a box will pop up as shown above.

Use the “Backspace delete” key to clear the name in the box. **Do not** use the ‘Delete’ key on the keyboard when you want to make changes to the name. The ‘Delete’ key on the keyboard is used for deleting Circuit blocks connected to the chip!

You cannot rename a circuit to a name that is already used.

For example Pin 3 is named LED4. To rename the circuit at Pin 18 to LED4, it is necessary to first rename the assigned name LED4 in Pin 3 to another name. Now rename Pin 18 to LED4.

The design of the eRacer_16m robot circuits using ezCircuit Designer is now completed. The next section shows the use of the CoreChart test program to test the circuits.



CoreChart test program

ezCircuit Designer generates a circuit test program written in CoreChart assembly language which we will later call eRacer16SetUp.bst. See below for explanation of CoreChart. To export this circuit test program to CoreChart, click on the CoreChart icon, shown in a red circle, on the ezCircuit Designer menu bar.

A CoreChart window will be opened as shown below.

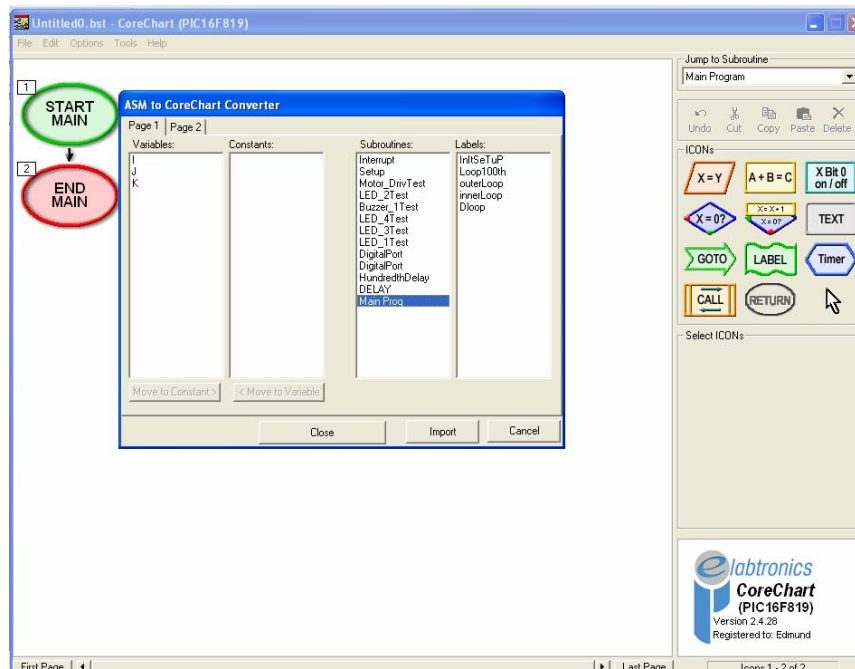
Note on CoreChart:

CoreChart is an intuitive microcontroller programming language. It is a graphical assembly language which operates at the maximum speed of the PIC microcontroller. The instruction codes are represented by icons to minimise syntax problems.

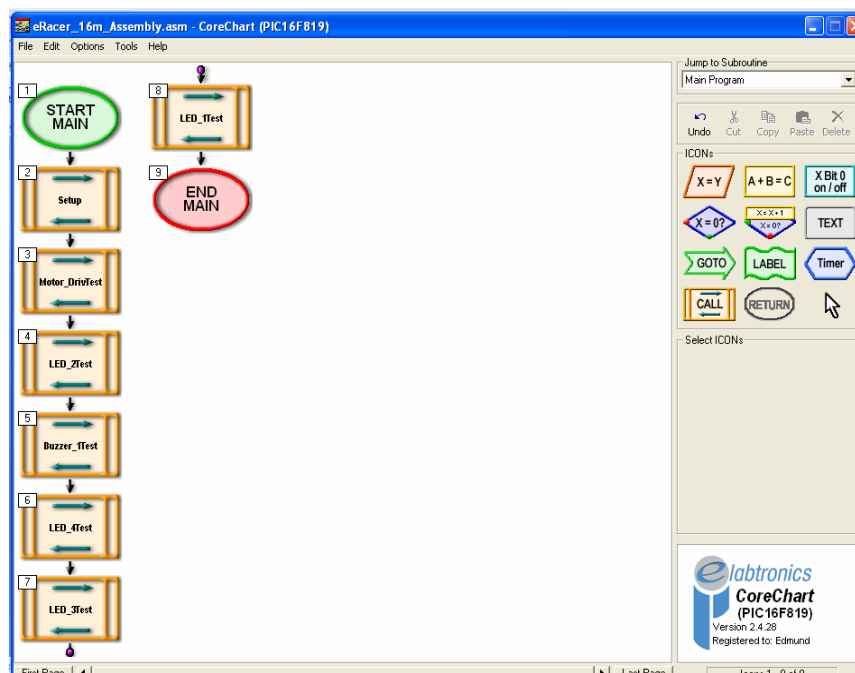
A CoreChart window has 3 sections: the programme section, icon section and icons properties section.

Please refer to the Help section in the CoreChart Menu Bar. It contains important user information about CoreChart operation.

Click on the “Import” button to import all subroutines and labels.



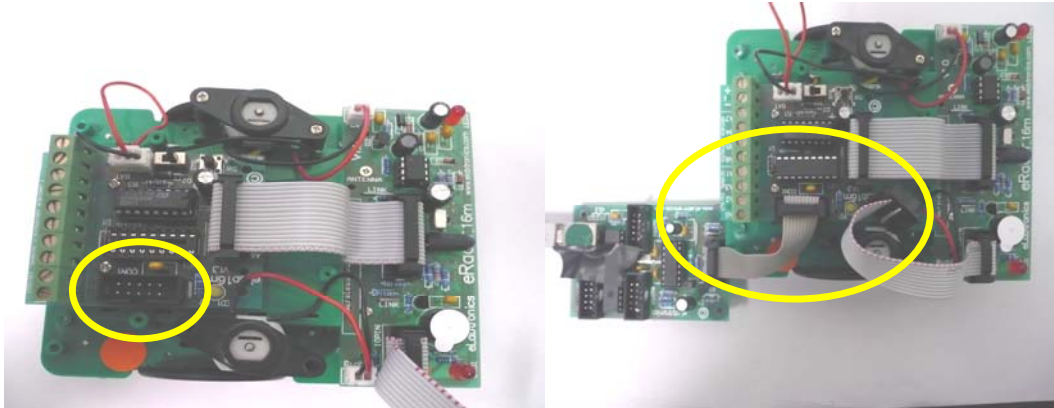
When all subroutines and labels are imported from ezCircuit Designer a new CoreChart screen is displayed. The CoreChart program displayed is the CoreChart test program to test the output circuits created with ezCircuit Designer.



Running CoreChart test program

Step 1: Connect the eRacer_16m to the PC

Next we connect the eRacer_16m to the PC via the USB programmer.



The eRacer_16m is connected to the USB programmer as shown. To do this, unplug the 10-pin cable from the eRacer_16m connector shown. Next connect the USB programmer 10-pin grey ribbon cable onto the eRacer_16m 10-pin connector.

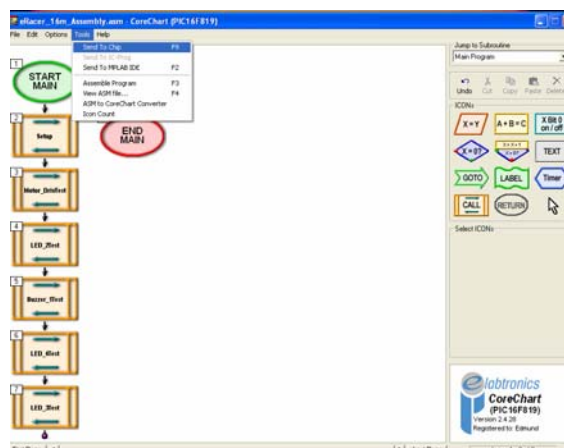
Note1: The PIC16F819 chip is programmed via the USB programmer. Ensure that the PIC16F819 chip is in the eLab16m board. If not note direction and plug the chip into the IC socket carefully.

Note2: Use the USB cable supplied with the programmer. A poor quality USB cable, or cable extensions may result in program transfer errors.

When the eRacer_16m is connected to the PC ready for programming, make sure that the eRacer_16m is switched **ON**. The ON/OFF switch is on the eLab16m board.

Step 2: Programming the eRacer_16m with CoreChart test program

Go to **Tools** and click **Send To Chip** or simply press 'F9'. **CoreChart** will assemble the program into machine codes and transfer the codes and control to the USB programmer. The program will then be transferred to the PIC chip on eRacer_16m.

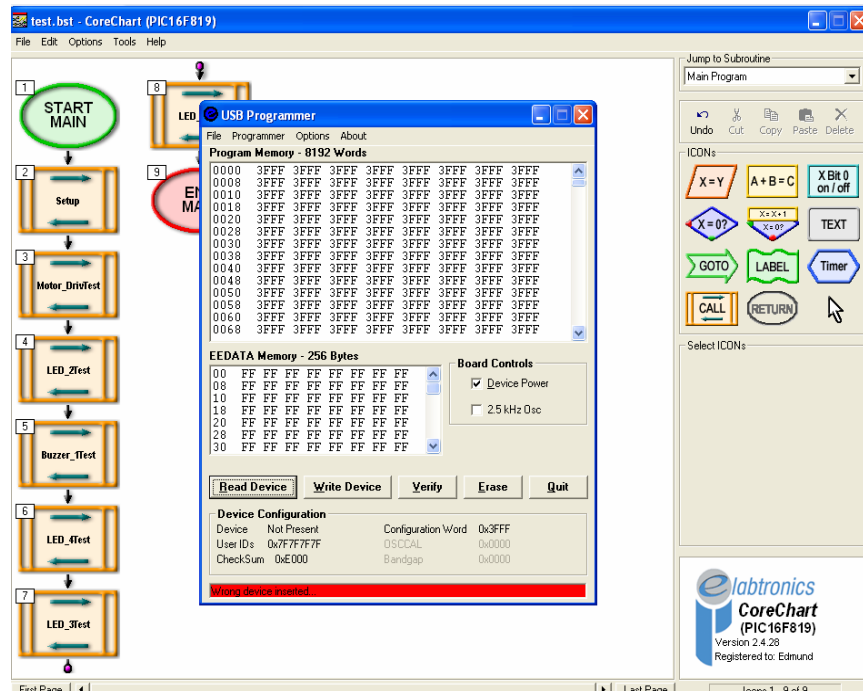


While the program is being transferred, a USB programmer window will pop up.

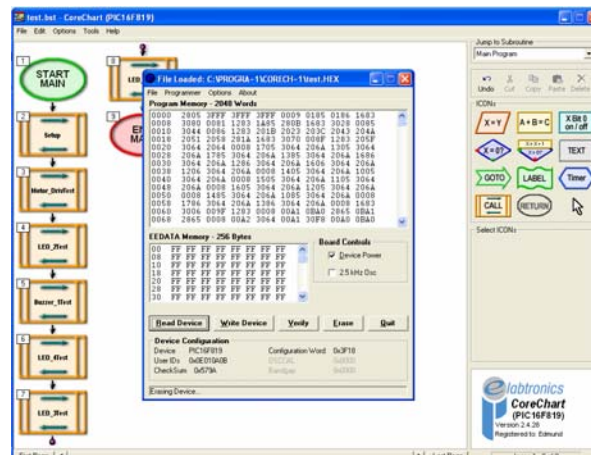
During program transfer to the eRacer_16m, a bar is displayed on the USB programmer window to show the transfer in progress. If a bar is either not displayed and or not moving click the **“Write Device”** button.

If the program transfer is successful the bar at the bottom of the USB programmer window turns green and displays the message **“Write Successful”**

If the program transfer is unsuccessful, the bar will turn red and displays a different message.



Note: If the program transfer fails, check if the eRacer_16m is switched on. Also check if the USB cable is connected to the programmer and to the CPU.



Save a copy of this CoreChart test program as **eRacer16mSepUp.bst**. Later CoreChart workshops will make use of this program because it contains the essential CoreChart settings for the eRacer_16m project.

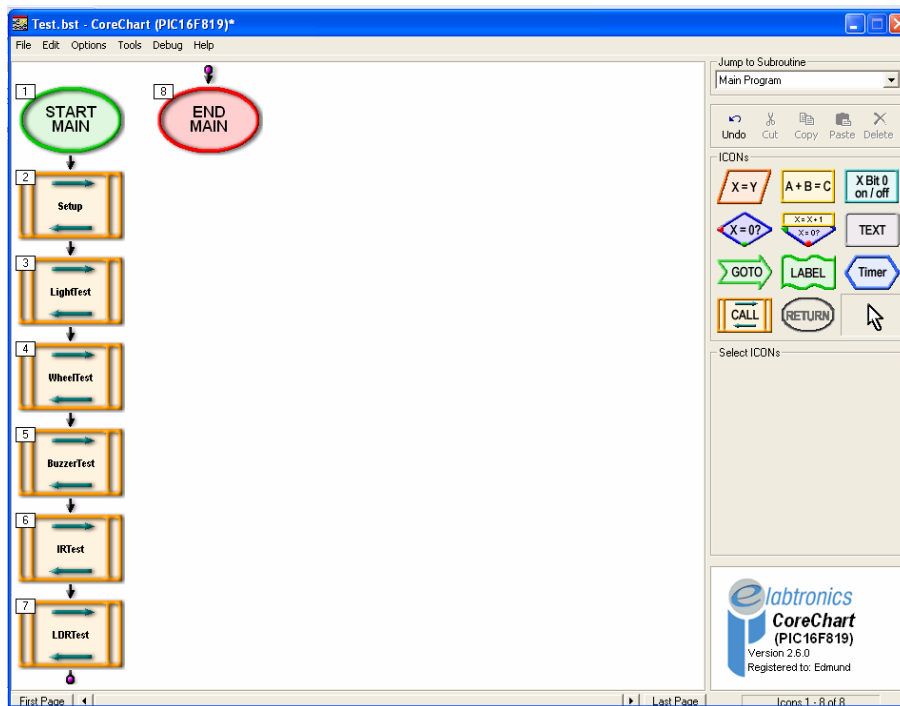
Note: When programming is finished and the USB programmer removed, remember to re-connect the 10-pin ribbon from the eRacer_16m board to the eLab16m board.

Test Inputs Using Output circuits with Test.bst provided

After all the **output circuits** are confirmed to be working, we need to make sure that the inputs are tested using the output circuits.

Do not close the CoreChart program yet!

Go to file→ select Open→ click on eRacer_16m file→ open the program **Test.bst**



Screen showing the test program.

Transfer the file to the eRacer_16m. Press the 'F9' key on the keyboard or **refer to Step 2 on page 52 to send the file to the chip**. When programming is completed remove the USB programmer and reconnect the 10-pin ribbon connector to the eLab16m board.

To test the eRacer_16m, make sure the ON/OFF switch is ON.

Click on the reset button to move from one test to the next.

LED1 will light up when the eRacer_16m is reconnected and switched on.

Click the reset button once. LED1 will turn off and LED2 will turn on

Click the reset button again and LED2 will turn off and LED4 will turn on.

Click the reset button again and LED4 will turn off and LED5 will turn on.

Click the reset button and LED5 will turn off, Left motor will move forward.

Click the reset button and Left motor will move backward

Click the reset button and Right motor will move forward

Click the reset button and Right motor will move backward

Click the reset button and Buzzer will sound

Click the reset button and LED1 & LED2 will switch on.

Next put an object in front of the eRacer_16m to block the IR receiver and LED1 and LED2 will switch off immediately.

Click the reset button to test the LDRs. LED1 and LED2 will light up when both LDR1 and LDR2 face a light source with sufficient light.

When LDR1 is covered, LED2 will remain on while LED1 turns off. The opposite happens when LDR2 is covered.

Quality Control Check list

Checked by:

Date of check:

Functionality test

- ☐ **LED1.** LED1 should light up
- ☐ **LED2.** LED2 should light up
- ☐ **LED4.** LED4 should light up
- ☐ **LED5.** LED5 should light up
- ☐ **Left Motor (Forward).** Left Motor should rotate clockwise
- ☐ **Left Motor (Reverse).** Left Motor should rotate anticlockwise
- ☐ **Right Motor (Forward).** Right Motor should rotate anticlockwise soon after program “test” is loaded into eRacer_16m.
- ☐ **Right Motor (Reverse).** Right Motor should rotate clockwise
- ☐ **Buzzer.** Sound should emit from buzzer when push button is pressed once again
- ☐ **Infra red responding**
- ☐ **LDR1 responding**
- ☐ **LDR2 responding**

Please note that it will take a few seconds to see the rotating wheels and the blinking of the LEDs.

Software Workshop Part 2 - CoreChart

Programming eRacer_16m Robot Outputs

At this stage the eRacer_16m robot is built and functionally tested. Input / Output circuits for the robot have been designed using ezCircuit Designer. Output circuits are tested with a test program based on CoreChart generated by ezCircuit Designer.

This workshop introduces CoreChart programming of the eRacer_16m PIC16F819 microcontroller using simple **output** circuit examples such as LED and Buzzer.

In order to program using CoreChart we need to understand the eRacer_16m hardware.

eRacer_16m Hardware

Let us take a closer look at the PIC microcontroller and how it operates.

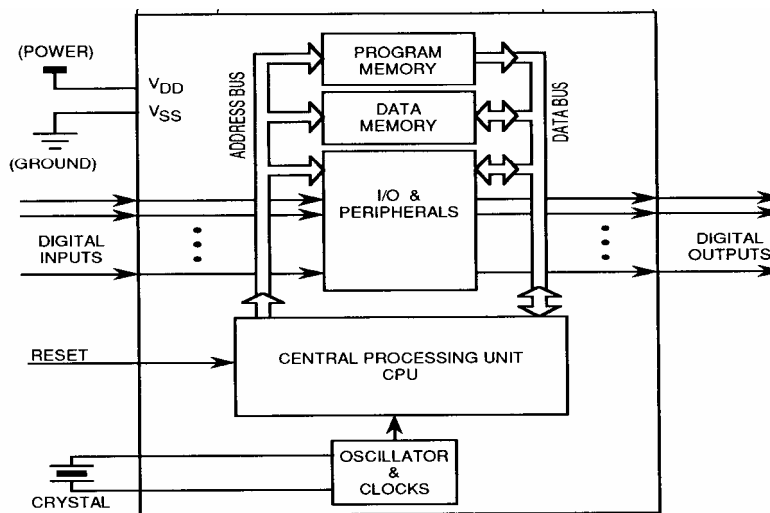


Figure 22: Block diagram of a microcontroller.

Power

The PIC16F819 requires 5V DC to work. Voltages of 4.5 to 6.0 volts DC from three or four 1.5V dry cells are sufficient to run the chip.

Even though the PIC chip consumes only 1 mA - even less at low clock speeds, the power supply must be sufficient for the chip to drive LEDs or other high-current devices.

A 0.1 μ F capacitor is connected close to the PIC chip from V+ (*pin 14*) to ground (*pin 5*). It needs to be close in order to protect the chip and adjacent components from electrical noise. This capacitor is labeled as C5 on the eRacer_16m PCB.

Reset

The microcontroller clear (MCLR) pin (*pin 4*) is normally connected to V+ (*pin 14*) through a 10K resistor. Grounding the pin momentarily will reset the PIC.

A **normally open** reset button, between **MCLR and ground**, is added to act as a circuit-reset switch.

Clock

The PIC16F819 microcontroller has an internal clock set to run at 8MHz (8 million cycles per second). For most operations, each instruction in a program will take 4 clock cycles to be executed. This means that 2 instructions will be executed in 1 millionth of a second i.e. 0.5µs for each instruction.

The ability of the PIC chip to take fixed internal clock cycles for each instruction is particularly useful when programming specific time intervals in real time applications.

It takes 4 clock cycles to execute one instruction.

$$\text{INSTRUCTIONS per second} = \frac{f_{osc}}{4}$$

f_{osc} = The frequency of the microcontroller internal clock.

f_{osc} = 8Mhz, the microcontroller executes 2 million instructions per second.

Digital Electronics / Binary numbers

The word “digital” refers to electronics that use “digits” for processing information. “Digits”, or numbers, could be 3 or 6 or even 2. In fact we normally use digits anywhere from 0 to 9.

Electronics that are not digital are referred to as “analogue”. An example of an “analogue” signal is a voltage signal. For example, the voltage may vary anywhere between 0 to 5 Volts.

It is easier to build digital computers such as the PIC chip on the eRacer_16m that operates on just 2 digits, such as 0 and 1. The circuits can then be either on or off, corresponding to 0V or 5V. This system of using just 2 digital states to count is called the binary number system.

The digits are called **bits** (*binary digits*). Just like decimal numbers, binary numbers are made larger by putting more digits together. Below is a table of decimal numbers up to ten and their binary equivalents.

Decimal	0	1	2	3	4	5	6	7	8	9	10
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010

Assembly Language

The PIC microcontroller reads binary instructions from the program memory, one after another, and does whatever those instructions say. Each instruction consists of 14 bits.

If you could **see** the bits as binary ones and zeroes, a program might look like this:

```
11000000000000
00000001100110
110000000000001
00000010000110
10100000000100
```

Technicians used to write binary codes like these to program early computers. Binary codes are very hard for human beings to comprehend, as they do not represent the “normal” way that we communicate.

For this reason, *assembly language* was invented over forty years ago for different types of CPU. What assembly languages have in common is that the instructions are abbreviated by readable texts (*mnemonics*) such as GOTO and programmer assigned labels can represent memory locations.

Today the eLabtronics CoreChart graphical assembler has replaced the text assembler instructions with icons to make it more intuitive to program microcontrollers. It is important to note that CoreChart is NOT a graphical interface. It is a graphical assembly language like any text based assembly language operating at maximum speed and efficiency of the microcontroller.

PIC Outputs

The output from a microcontroller is in the form of a digital signal (*logic 0 or 1*) placed on the required port connection (*I/O line*).

Each of the PIC16F819 microcontroller pins could be seen as a switch because it is either on (5Volts) or off (0 Volts). Hence the microcontroller pins will switch the connected output circuits on or off.

Output Devices

In a personal computer, output devices such video display and printer are used to communicate information. Microcontrollers use simpler devices such as lamps, light emitting diodes (*LEDs*), opto-couplers, relays, and switching transistors.

Relays, switching transistors and opto-couplers are in turn used to interface to higher power applications such as heaters, electric motors and solenoid devices.

Light Emitting Diode (LED)

LEDs used in electronics, are a semiconductor **diode** that emits infrared or visible light (electro luminescence) when charged with an electric current. Visible light LEDs are used in many electronic devices as indicator lamps. They can be arranged into a matrix, to make alphanumeric displays.

The LED's chemical compound is formed by the combination of the elemental periodic table groups III-V, to create a semiconductor compound that is related to gallium arsenide. Compared to a laser **diode**, an LED also consumes little power but has a simpler configuration and uses a wider bandwidth.

Piezo Buzzer

Piezoelectrics are materials that generate a voltage when they are subjected to mechanical pressure; conversely, when subjected to an electromagnetic field, they exhibit a change in dimension. Many piezoelectric devices are made of the same ceramic materials as capacitor dielectrics.

Practical 2

Editing a CoreChart program

In this practical you will learn how to edit and rewrite the functionality test program from Practical 1.

There may be occasions when you write or modify existing programs many times, i.e. developing different versions of the same program. You may also be “debugging” programs written by someone else!

It is therefore important to have a **file management strategy** *and* a **modular** approach in programming.

If not it be difficult to remember **accurately** what you have done previously. It will be even more difficult for someone else to debug your programs.

File management strategy

First create a new folder under your name, ideally one for each different project. When modifying an existing program use the **same file name** followed by your name.

For example, if you are about to modify the program **test** and your name is Jane. Save this program as **test.Jane1** and use it as your working program instead of the original program. You may wish to name further alteration to this program as **test.Jane2**. Including your name will make it easier to identify the file later.

Modular programming

Test.bst program for functionality test in the previous chapter is an example of Modular programming.

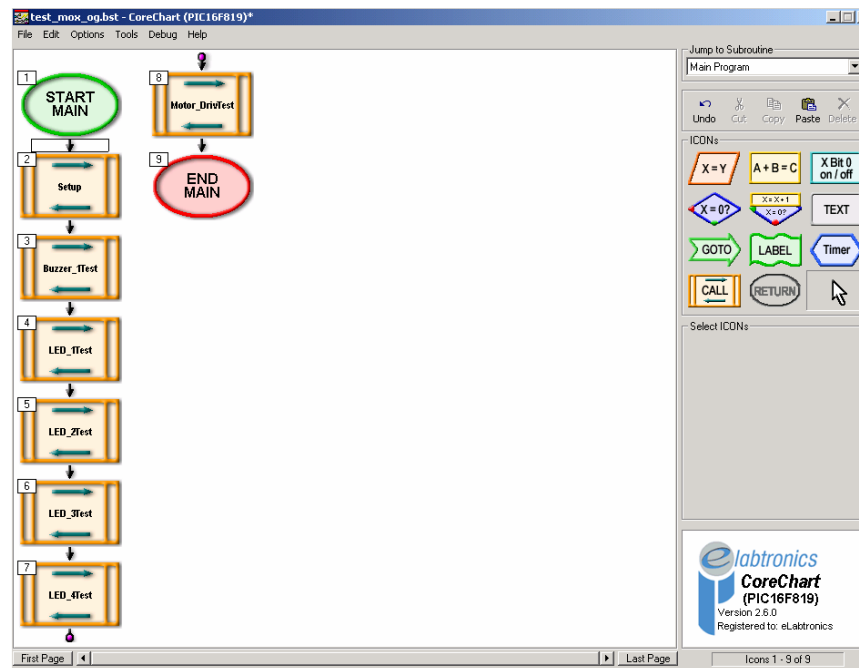
Modular programming can be further illustrated by comparing the **Program1.bst** as supplied in the eRacer_16m Programs folder to **eRacer16mSetUp.bst**

Open **eRacer16mSetUp.bst** CoreChart test program as saved in Practical 1.

Open the program Program1.bst in the eRacer_16m Programs folder.

Compare eRacer16mSetUp program to Program1.

Question: *Which program is easier to understand for a novice programmer?*



The **eRacer16mSetUp** program is easier to understand as it has a modular form with each task written as a **subroutine**. The program has 6 tests: 4 LED tests, a motor test and a buzzer test.

The contents of the **subroutines** can be displayed by double-clicking on the icons. Open the **eRacer16mSetUp** program and double click on the **CALL** icons.

Subroutines are covered in greater depth later.

Writing each task as a subroutine makes it easier to debug or to check for program faults. Subroutines such as **turn motor left, right, forward or backward** are already written. Access them by clicking on the **CALL** icons in the menu box on the right.

On the other hand **program1** is harder for the novice programmer to follow. The program is presented in its entirety with the **overall program missing** even though comments are included.

This program actually performs a simpler task - turn on permanently 4 light emitting diodes (LEDs) and then sound the buzzer on and off continuously.

Program1 will become increasingly difficult to follow as more tasks are included and the program becomes larger.

Modify pre-written CoreChart programs

1. CoreChart programming is a process of picking and placing icons in a logical order and entering parameters where required. Open **eRacer16mSetUp** and save it as **BuzzerOnOff** (or save it as BuzzerOnOff.your name)
2. Modify **BuzzerOnOff** by deleting all the subroutines except the Setup and Buzzer_1Test subroutines. This is done by clicking the mouse pointer on the icon to be deleted, press **delete** and then **return**. Open the Buzzer_1Test subroutine by double clicking on it. Insert a Label icon (green flag shaped icon) over icon number 2 in the Buzzer_1Test subroutine. Next, insert a GOTO icon (green arrow shaped icon) above icon number 8. Click **save** to save the program.
3. Send the modified program to the programmer as in Practical 1. Test your program. Notice that the buzzer turns on and off at a different rate.
4. Next, open the Buzzer_1Test subroutine again and modify the program by changing the parameter **w** to other values e.g. double the value. Test the altered program. Observe the change in the sound from the buzzer.
5. Modify **test.bst**. Open the **test.bst** program. Change the sequence or the order of the instructions. Save it under a new name (e.g. test.yourname).
6. Test the program. Send the program to the programmer, remove the cable and press the push button repeatedly to check the changes to the program.

Note:

The microcontroller will execute a program in the way it is programmed. It is only as clever as the program that is written.

Practical 2a

Testing the outputs

Step 1: Open eRacer16mSetUp.bst

Startup the **CoreChart** program, open the **eRacer16mSetUp.bst** program that you created in Practical 1. **eRacer16mSetUp.bst** has been designed to be a test program, since it cycles through the port outputs.

Step 2: Send the program to PIC

Make sure that the cables are connected the same as in previous practical and that the **eRacer_16m** is switched on. Select **Send to chip** from the **Options** menu. A dialogue box should appear that indicates that the **program** was **successfully verified**.

Step 3: Check all the outputs of the PIC

When the **eRacer_16m** is successfully programmed, press the reset button once. LED1 lights up for about 3 seconds and then it switches off for 3 seconds. Then LED2, LED4 and LED5 will take turn to turn on and off as in LED1.

The buzzer turns on for about 3 seconds. The left motor spins forward for 3 seconds and then spins in reverse for 3 seconds. Finally the right motor will spin forward for 3 seconds and then spins in reverse for 3 seconds.

Note that the order of these steps may differ.

Step 4: Modifying the program

In each of the LED test, the buzzer test and the motor test there is a delay subroutine called **HundredthDelay**. Modify the delay number used in this program (w=100) by changing the number 100 to 50. Check the results by sending the program to chip.


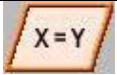
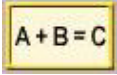
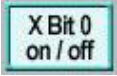
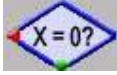
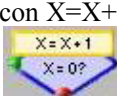




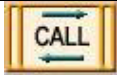




Observe the changes.

What are the changes?

Practical 2b

Starting your own program

Useful Information for Practical 2b

CoreChart Symbol	Meaning
START MAIN 	Beginning of program
Assignment icon 	Allocating a value to a Character
Calculate icon 	Performing a calculation
Reset icon 	Mode of operation, bits on=1, off=0
Decision icon $X=0?$ 	If loop (insert after symbols are between START MAIN and END MAIN)
Decision with counter icon $X=X+1$ 	If loop (insert after symbols are between START MAIN and END MAIN)
TEXT box 	Program comments
GOTO icon 	Program 'repeat from LABEL' Command
LABEL icon 	Label of function
TIMING icon 	Time of when a command is to be carried out
CALL subroutine icon 	Calling a subroutine function
SUB INTERRUPT within CALL subroutine icon 	CALL operation calls a Subroutine function
RETURN from subroutine icon 	Within SUB INTERRUPT
Return to MAIN 	Return From Subroutine to MAIN program
END MAIN 	Returning from a subroutine Function
	End of program

Step 1: Open eRacer16mSetup.bst

Startup the **CoreChart** program, open the eRacer16mSetup.bst from your file by clicking on **File->Open>drive>your folder>eRacer16mSetup.bst**

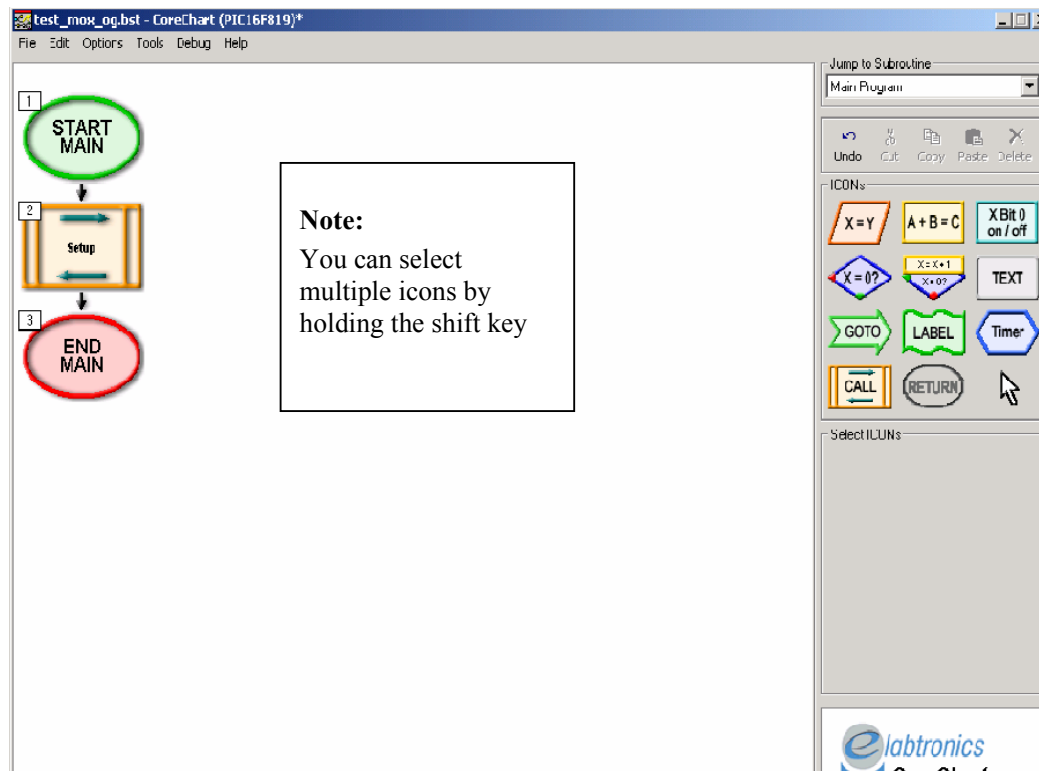
Step 2: Create a template file

Save this program to be re-used later. Save this file as **start.bst**.

Keep the Setup icon; delete the rest between the **START MAIN** and **END MAIN** icons.

The reason for re-using the file is because it contains the essential CoreChart settings and some subroutines that are required later.

Note: It is not possible to delete the **LABEL** icons until the **GOTO** icons have been deleted. Delete these **GOTO** icons first then the **LABEL** icons and lastly delete the other icons.



Note: Many of the ports on the microcontroller have multiple functions. The Setup subroutine is necessary to initialize the PIC16F819 when using only digital inputs and outputs.

Step 3: Save new file

Now there should be 1 icon. Select the **Text** icon and a box will appear below. Enter the program name “**;ON LED PROGRAM**”. The text description is “**; turn on a LED**”. *Each comment line must be preceded with a semicolon.* Save the modified program as “**on.bst**” (**File->Save As...**).

The screenshot shows the CoreChart (PIC16F819) software interface. On the left, a flowchart is displayed with four steps: 1. START MAIN (green oval), 2. ON LED PROGRAM (rectangle with text "; turn on a LED"), 3. Setup (rectangle with two horizontal arrows), and 4. END MAIN (red oval). A red arrow points from the "ON LED PROGRAM" step to the "TEXT" icon in the toolbar. The toolbar on the right contains various icons for operations like X=Y, A+B=C, X Bit 0 on/off, X=X+1, X=X-1, GOTO, LABEL, Timer, CALL, and RETURN. Below the toolbar is a "Select ICONs" section. At the bottom right, the eLabtronics CoreChart (PIC16F819) logo and version information (Version 2.6.0, Registered to: eLabtronics) are visible.

Note

You must precede each line of comments with a semi colon (;)
If not a program error will occur.

Insert the following icons as shown.



Set / Reset Icon



Label Icon



Timing Icon



Goto Icon

Procedure to insert the icons above:

Step 1: Select a Set / Reset Icon in the ICON Properties box situated below ICONs

Set – Click on down arrow and select device to set e.g. **LED1**

Set to **On**

Place above **End** icon

Step 2: Select a Label Icon in the ICON Properties box

In the “**Label Name**” box enter the label name e.g. **Here**

Place above **End** icon

Step 3: Select a **Timing Icon** in the ICON Properties box

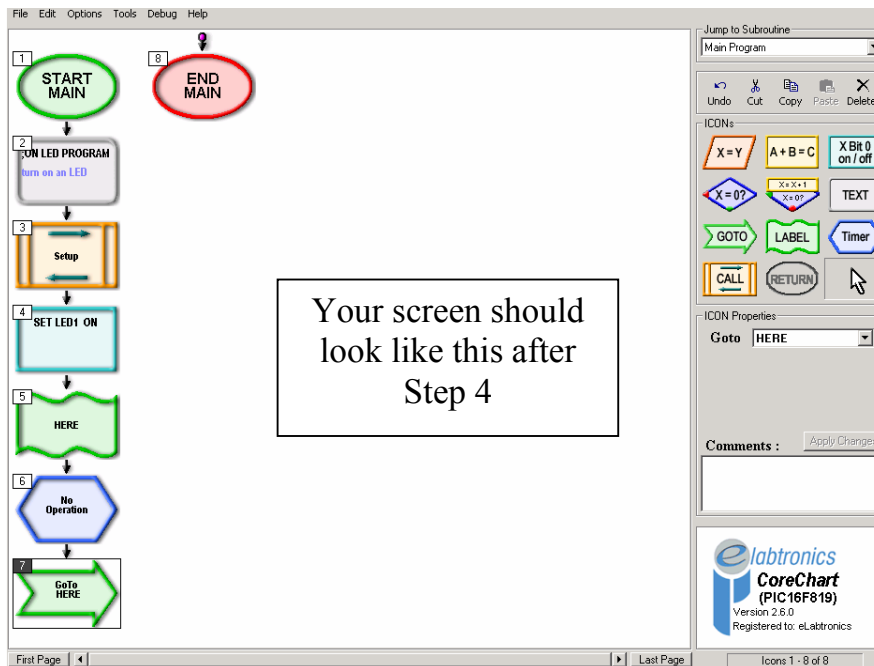
Set Timing icon drop down box – **No Operation**

Place above **End** icon

Step 4: Select a **Goto Icon** in the ICON Properties box

Set the Goto icon drop down box to – **Here**

Place above **End** icon



Step 5: Test the program

Save the program into your own folder and then run it. This is done by selecting **Options->Send to chip** or hit the **<F9>** shortcut key. When the program is run the red LED on the right (labeled as LED1 on PCB) should light up.

Step 6: Extend the program

Now we will make the other red LED (labeled as LED2) on the PCB turn on. Changing icon 4 to read Set LED2 ON does this. Click on icon 4 and the properties box will appear. Change the **Set property** to **LED2**.

Set LED2 ON

Save and recompile (**Send to chip**) then run the program. The Red LED on the left (labeled as LED2 on PCB) should turn on.

Step 7: Modify the program

Now changes will be made to cause both the LEDs turn on at the same time. This will be done by adding another icon under icon 4 which should now read **Set LED1 ON** as a result of Step 6. Click on the Set icon on the **ICONS** menu and the properties box will appear. Change the ICON Properties **Set LED1 ON**.

Set LED1 ON

Click on the arrow between Icon 4 and Icon 5 to insert the new icon.

Save and recompile (**Send to chip**) then run the program. Both the left and right LEDs should turn on.

Useful Information for Practical 2c

The following are important information about programming in CoreChart before starting Practical 2c.

When to Use a Repeating Instruction for Loops and Delays

Decision Icon

CoreChart uses the Decision icon to perform the IF function.

Decision icon X=0?	Decision, Test bit
--------------------	--------------------



The Decision Icon tells the program to skip the next icon when the result of a specified variable bit is either clear or set.

f = variable;
b = bit number.

CoreChart Instruction List			PIC Instruction	Comments/ description
1	2	3		
Variable	(Bit) 0 to 7	ON	btfss, f,b	Bit test variable, skip if set
Variable		OFF	btfsc, f,b	Bit test variable, skip if clear

Count and Decide Icon

CoreChart uses the Count and Decide icon to perform the LOOP function.

Decision with counter icon X=X+1	Decision and Count then Decide, Test if zero
----------------------------------	--



Decision Icon with counter. Count and Decide.

f = variable;
d = destination (variable or register w).

CoreChart Instruction List			PIC Instruction	Comments/ description
1	2	3		
Variable	Variable	+	incfsz, f,d	Increment variable, skip if zero
Variable	Variable	-	decfsz, f,d	Decrement variable, skip if zero

How the PIC makes Decisions

The PIC is a form of RISC (*Reduced Instruction Set Computer*) processor. By contrast the Pentium has hundreds of instructions. Each takes several clock cycles to execute. This makes the Pentium a CISC (*Complex Instruction Set Computer*) processor. The eRacer_16m uses the PIC16F819 chip.

The advantage of the PIC16FXXX series is it has only 35 instructions. Every instruction that does not involve a “jump” executes in just one instruction cycle.

In order to simplify the PIC instruction set and to limit instructions to one instruction cycle, there is no “IF” statement. Instead the PIC has several instructions that test for a condition and skip the next instruction if it is true.

An example of how this works is demonstrated below.

ICON	PROPERTIES
IF	PORTA Bit 0 OFF
Goto	m1
Assignment	w = 10

The icons translate to:

if PORTA Bit 0 is OFF then Goto m1
else w = 10

By interleaving Conditions and Goto the following instructions will rotate the bits of PORT B to the left if PORT A Bit 0 is 1 and to the right if PORT A Bit 0 is 0.

ICON	PROPERTIES
IF	PORTA Bit 0 OFF
Goto	M1
Calculate	PORTB>> PORTB
Goto	M2
LABEL	M1
Calculate	PORTB<< PORTB
LABEL	M2

Comparing Bytes

Note: PIC16FXXX does not have a byte compare (CMP) instruction as available in other PIC chips and other assembly language. Bytes are compared by subtracting them and then checking whether the result is zero in PIC16FXXX.

When a calculation is made the STATUS register will record if the result is zero (*Bit Z*) or the result has overflowed (*Bit C*) past the 8 bits of one register. The carry bit (*C*) can also indicate if a result is positive or negative. The STATUS register is 8 bits long. It can be used to select register banks as well as to check whether there has been a power-down or time-out.

Practical 2c

Flashing LED example

Step 1: Prepare for a new program

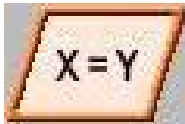
Start up **CoreChart**. Open the program **start.bst** from your files and save the program as **onoff.bst** (File, Save as, Select your folder).

Step 2: Insert comments

Select the **TEXT** icon and add comments to the program e.g. “**;ON / OFF LED FLASHER**”. The description is “**;To flash an LED**”. Insert below the **START** icon. Save the modified program.

*(Note: Each comment line in the **Text** icon must begin with a semicolon “;” otherwise an error will occur when you try to send it to the PIC chip).*

New Icons



Assignment Icon

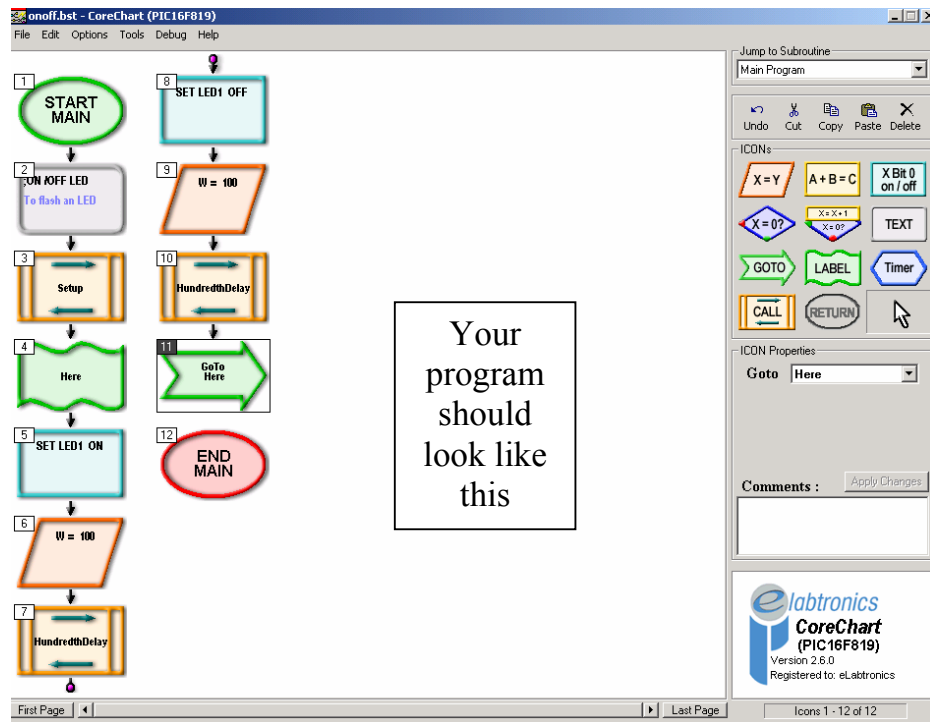


Call Icon

Step 3: Add new Icons

Change the program by inserting the following icons above the **END** icon.

ICON	PROPERTIES
Label	Here
Set	LED1 ON
Assignment	w = 100
Call	HundredthDelay
Set	LED1 OFF
Assignment	w = 100
Call	HundredthDelay
Goto	Here



Step 4: Run the program

Save the program and send to the PIC chip as in **Practical 2**. The Red LED on the right (labeled as LED1 on PCB) should now flash.

Step 5: Change length of delay

Change the program by changing the value of **W** in the Assignment icon to a **larger** number, up to 255. Save and send to the PIC chip. The LED should now **flash slower** since there is a longer delay between the On and Off states.

Essential Extension Questions

- How does entering different value for “W” in each delay effect the timing?
- How does adding another delay directly after the existing ones affect the result?

Step 6: Turn on other PORTs

Make **LED4** or **LED5** flash. Describe what happens.

Practical 2d

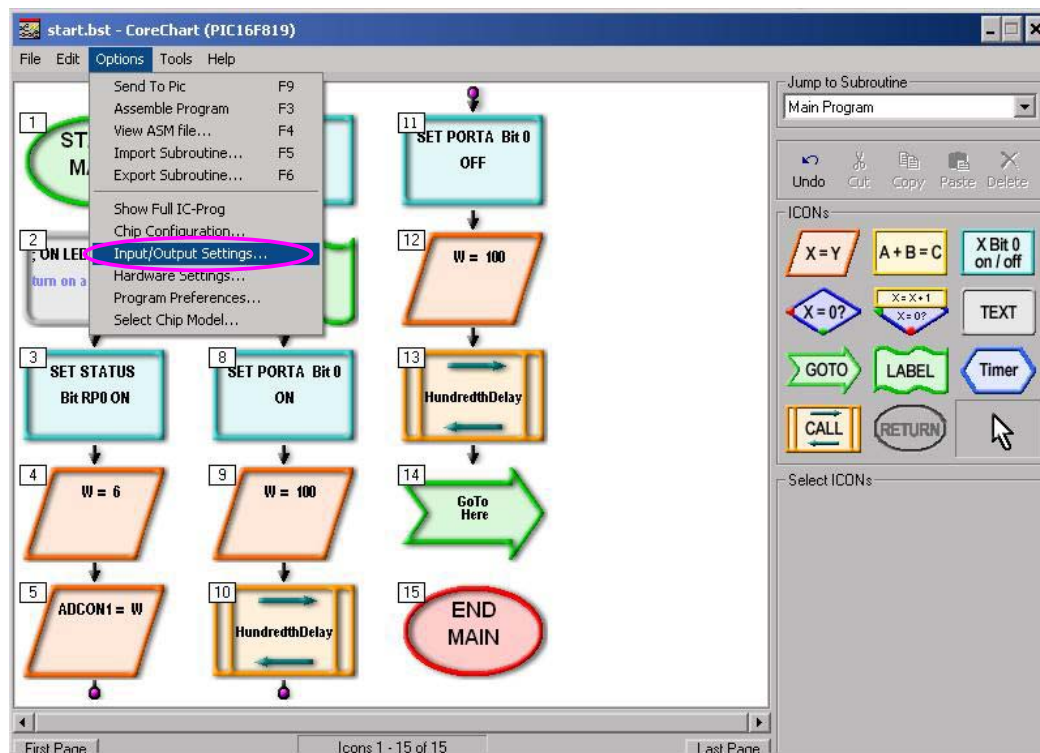
Bit Variable Example

Note: Even though ezCircuit Designer configures the CoreChart bit variables for eRacer_16m, this exercise shows how bit variables function.

When a Port is connected to multiple components it can become confusing. Labeling each bit with its function will reduce this confusion. **Bit Variable** in CoreChart allows any bit including PORT bits to be given a name to help to simplify programming. The names of the Port pins can also be the same as the Bit Variable names.

There are two ways to create a **Bit Variable**. One way is to select the **Input/Output Settings** under the **Options** menu.

Open the **onoff.bst** program that you have just done to change the program.



Note: In this Option it is only possible to label the bits of the ports but not the variable. The advantage is that it allows for change in the directions of the port pins.

Note: The second approach will be shown in a later Chapter.

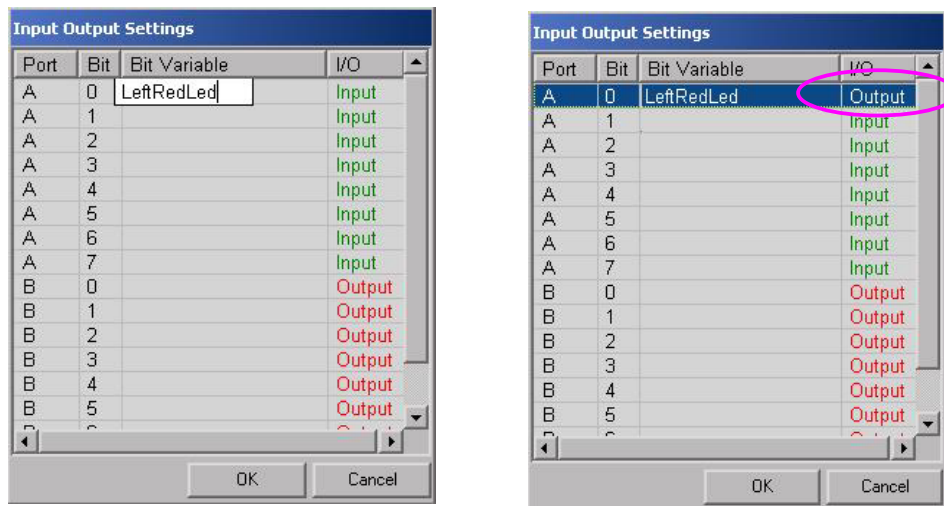
ezCircuit Designer automatically assigns bit variable names to all used pins so we do not need to enter bit variable names manually. Skip Step1 and continue with Step 2. Step 1 is only to show how bit variables are configured in CoreChart if ezCircuit Designer is not used.

Step 1:

Once the **Options** and **Input/Output Settings** have been selected a table will pop up as shown below. Select a Port and a Port Bit which you wish to label. In this case select the Port A Bit 0 as the Bit Variable. Type in the name in the Bit Variable column.

Refer to Port Table for port operations of eRacer_16m.

It is possible to also alter the pin directions from Input to Output by just double clicking on the **Output** or **Input** under **I/O**. When this is completed the entry click ok.



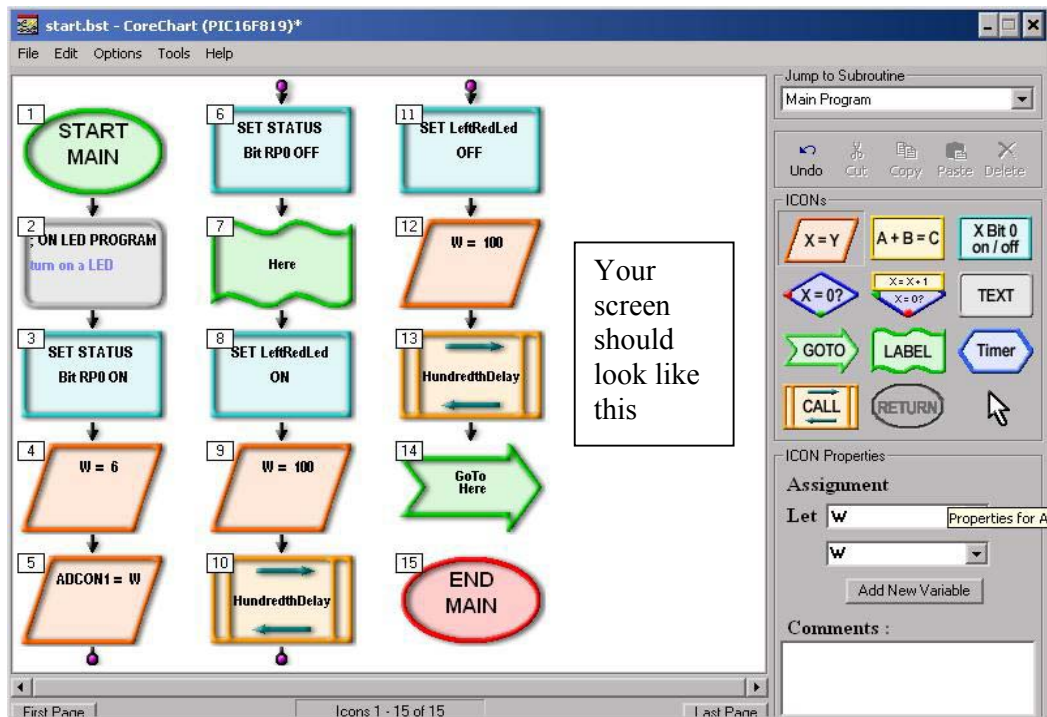
Step 2:

The bit variable has now been successfully created.

Click on the **Set/Reset icon**. In the **ICONS Properties box** scroll through **“Set”** to find LeftRedLed.

Replace **“PORTA bit 0 On”** icon with **“LeftRedLed ON”**. Also replace **“PORTA bit 0 OFF”** icon with **“LeftRedLed OFF”**. Send program to chip.

The LEDs should flash as before.



Exercise:

Label all the Ports with the Bit Variable names shown in the Port table below. On completion update your program using the new Bit Variables that have been just created.

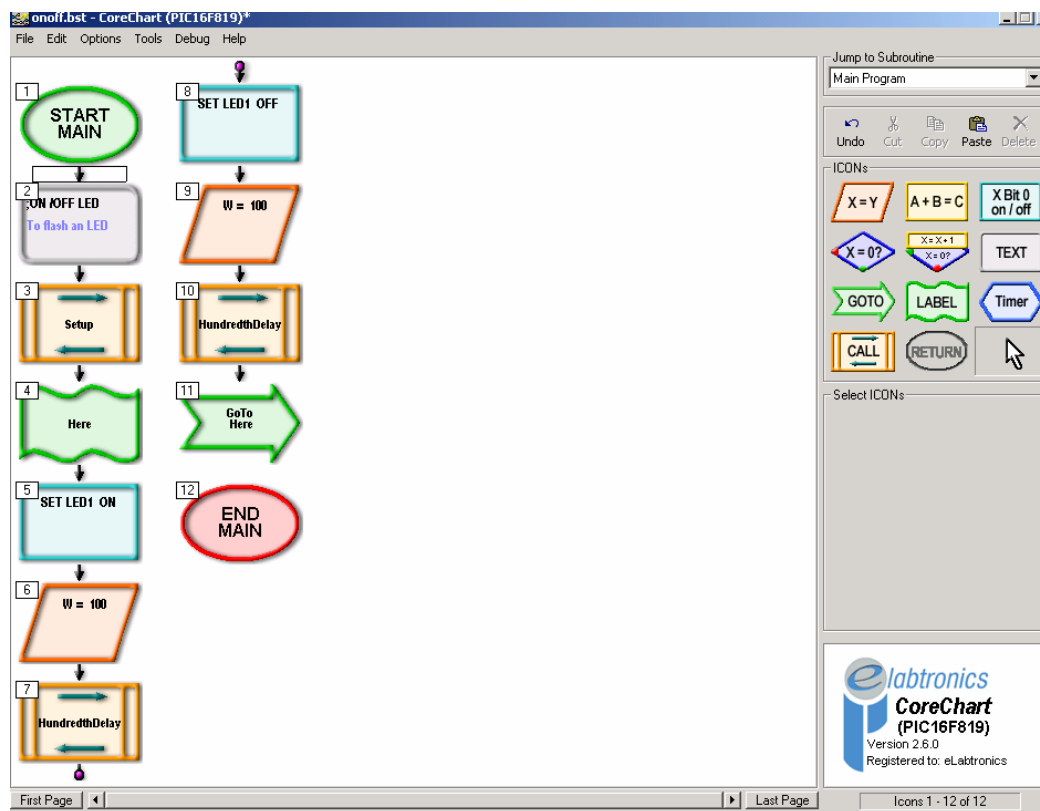
Port Table

Port	Bit	Control	Function
B	0	(not used)	
B	1	(not used)	
B	2	IR Receiver	Input
B	3	IR LED Transmitter	Output
B	4	Left Motor Forward	Output
B	5	Right Motor Reverse	Output
B	6	LDR1	Input
B	7	LED5	Output
A	0	RED LED1	Output
A	1	RED LED2	Output
A	2	Buzzer	Output
A	3	LDR2	Input
A	4	LED4	Output
A	5	Push Button	Input
A	6	Right Motor Forward	Output
A	7	Left Motor Reverse	Output

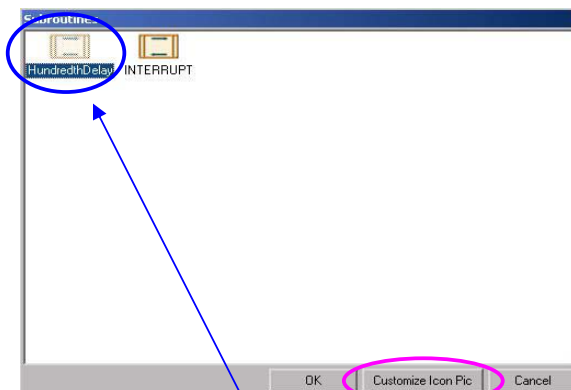
Practical 2e

Customising Icons with Pictures

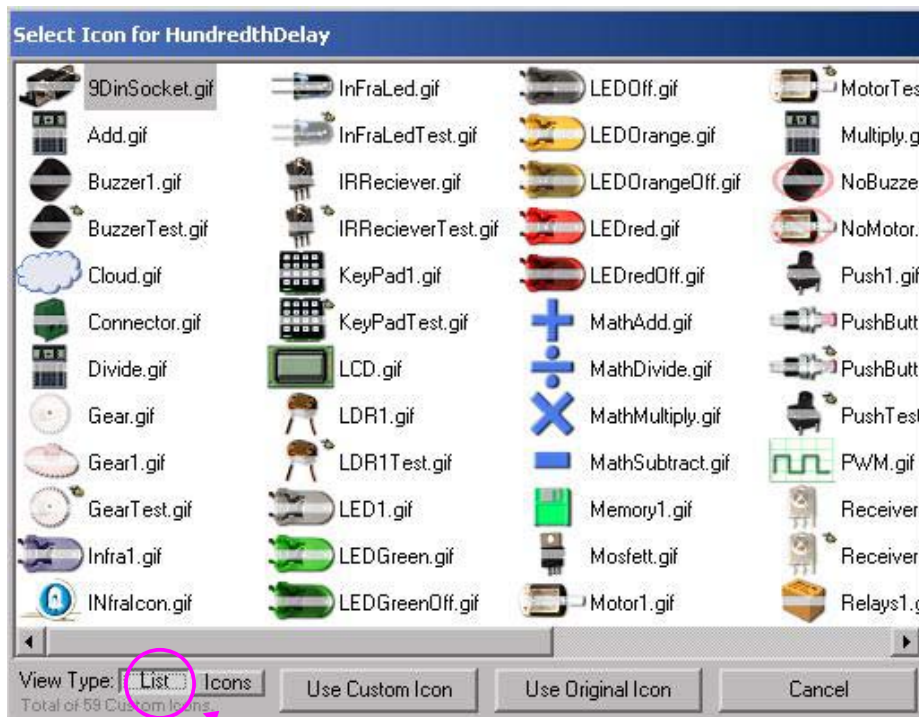
A useful feature of CoreChart is that it allows for customisation of subroutine icons with icon pictures of your choice. This allows for easy identification of the function of the subroutine with a picture closely related to the function. Open the program from Practical 2c – LED flashing example: **onoff.bst**.



Select the “HundredthDelay” subroutine. Now click on the CALL icon in the **ICONS** menu. A window of all the subroutines used by this program will appear.



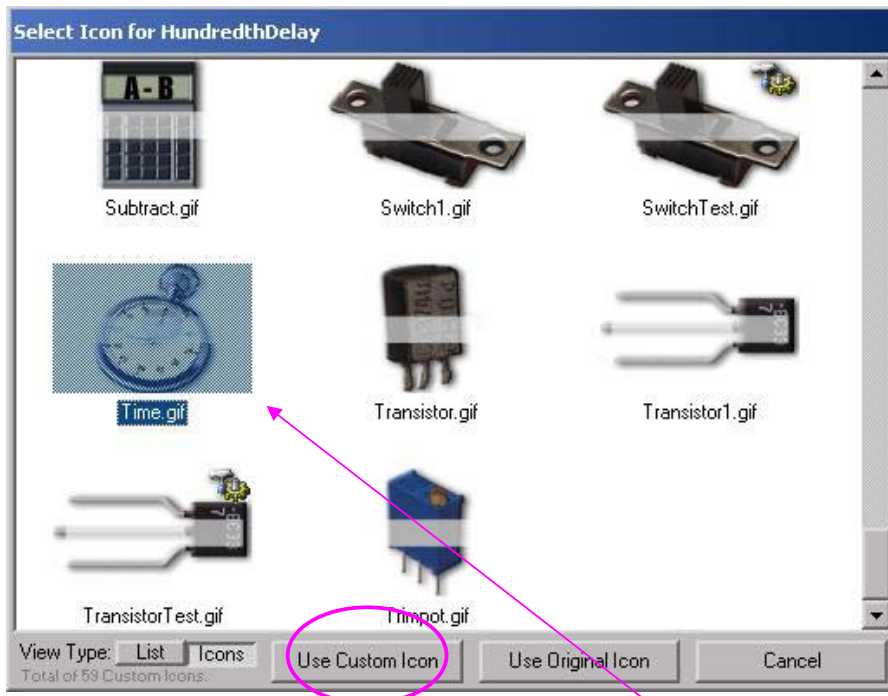
Make sure the “HundredthDelay” Icon is chosen as shown above. Click on “Customize Icon Pic”



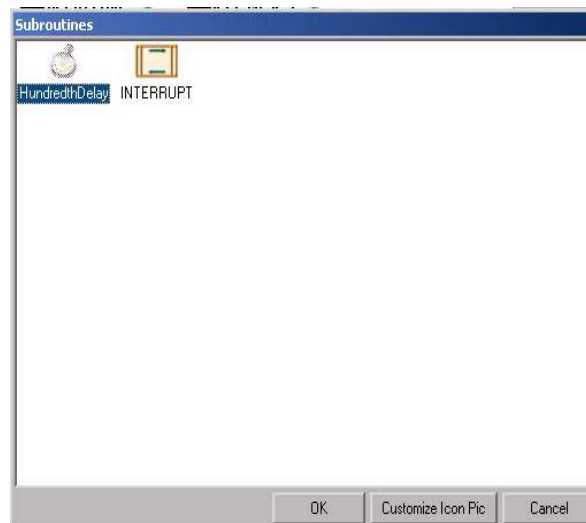
Two View Types are available for Customized Icons. The **List View** or the **Icons View**.



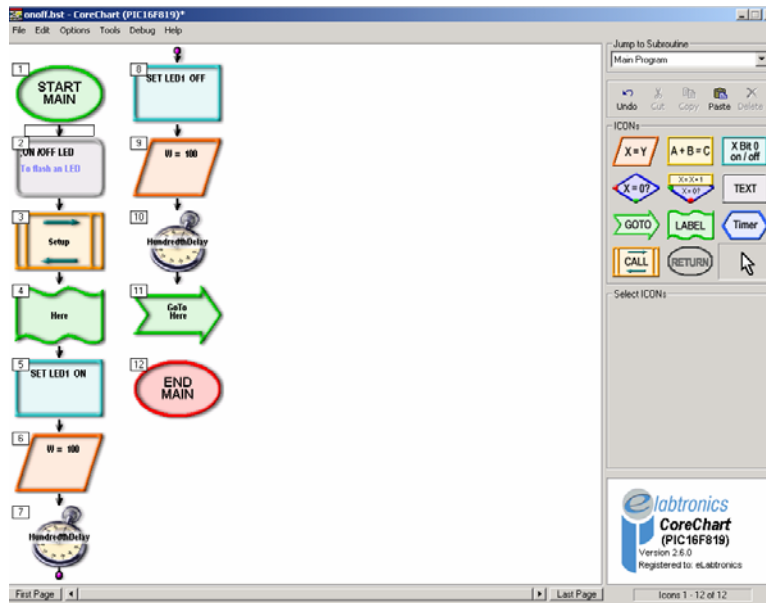
Choose **Icons**



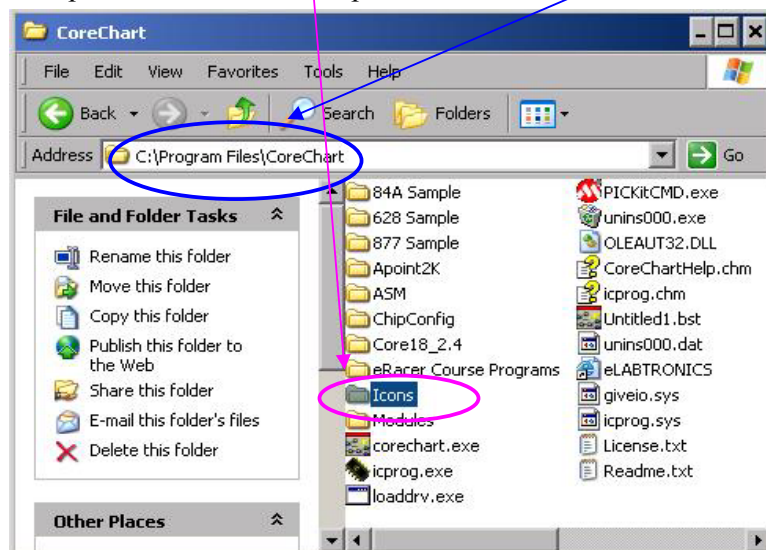
Scroll down the window to browse for an icon named “Time”. Double click on the icon to select the icon and click on the “Use Custom Icon” to customize the subroutine. To reverse the change, select the “Use Original Icon” button.



Notice that after the customized icon is selected, the subroutine will be substituted with the new icon. Click **OK** and return to the program.



The program should look like this when done. When creating icons ensure that the picture is in “gif” file format with a resolution of 60 x 39 pixel. To use the created icon, place it inside the **Icons** folder under **Program Files\CoreChart** directory. Other icon pictures should be also present in this folder.



Editing Subroutines

To open a subroutine in the main program, double click on the selected subroutine **CALL** icon. This is now in the selected subroutine. Further subroutine icons can be entered into the selected subroutine the same way as icons have been added into the main program.

To return to the main program double click on the **Return From Subroutine** icon.

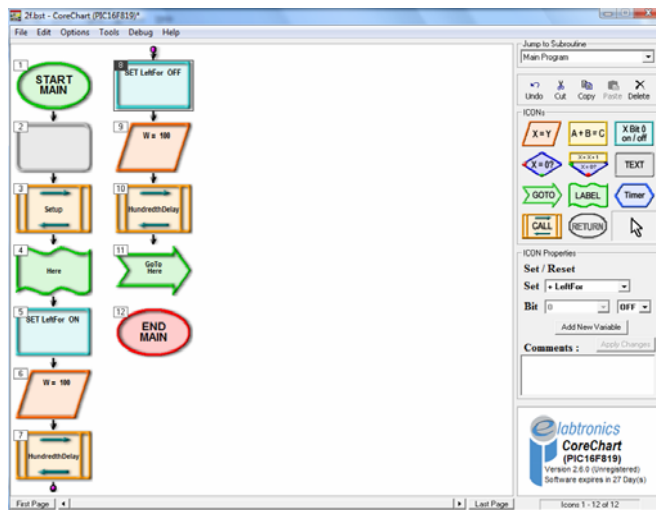
Practical 2f

Use of Debug Feature in CoreChart

CoreChart has an inbuilt debugger emulator called Debug which is an extremely useful aide for novice programmers. As an emulator it enables step by step instruction execution which is interactive with the hardware. The common simulator has no hardware interaction. The emulator displays the actual PIC chip memory values as processed on the hardware. Program bugs as well as problems in the hardware can be easily identified particularly with more complex programs.

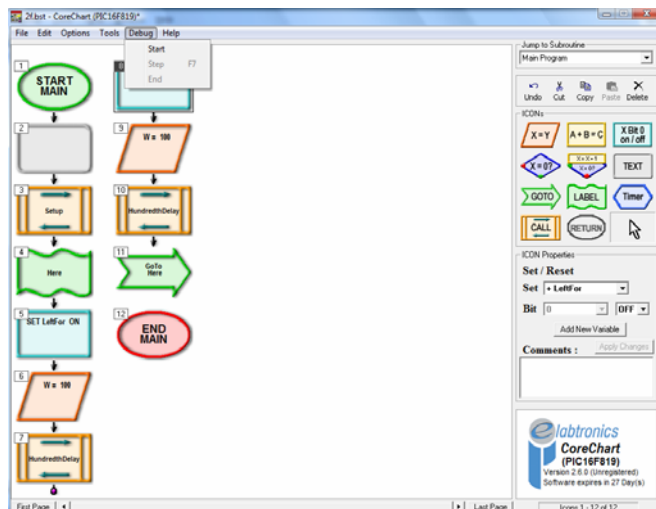
Step 1:

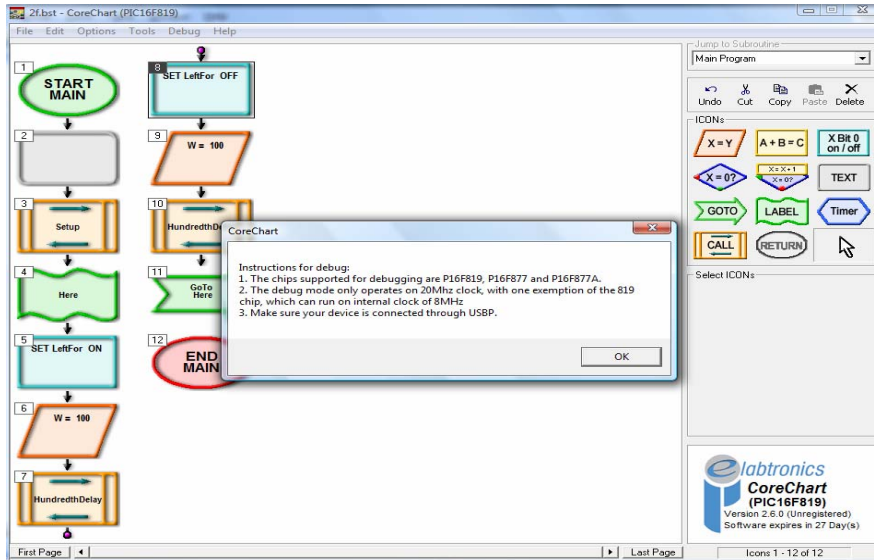
Make sure the USB programmer is connected to the eLab16m board and connected to the PC via USB. Open the program from **Practical 2e**. Replace the SET LED1 ON and SET LED2 OFF icons with SET LeftFor ON and SET LeftFor OFF respectively.



Step 2:

From the toolbar select Debug as shown below and click on Start.



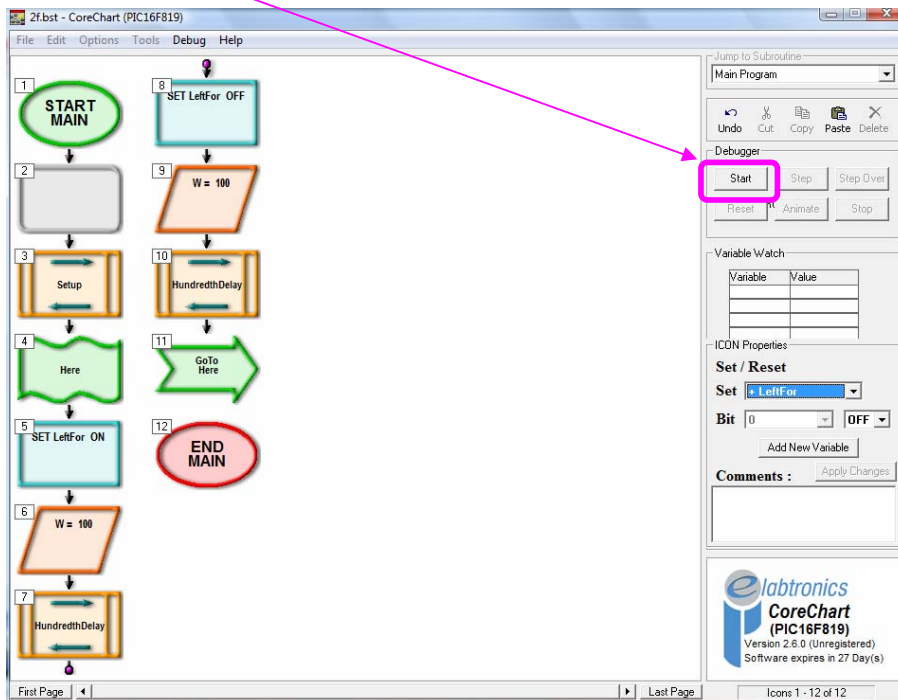


The following prompt will appear on the screen, simply click OK and the data will automatically be sent to the chip.

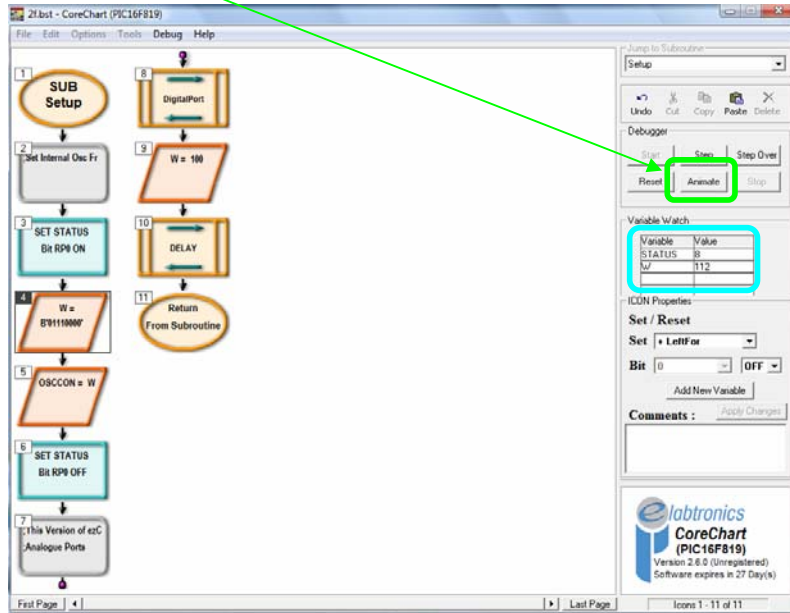
Step 3:

Select Quit when this is successfully done.

Now select **Start** from the Debugger panel.

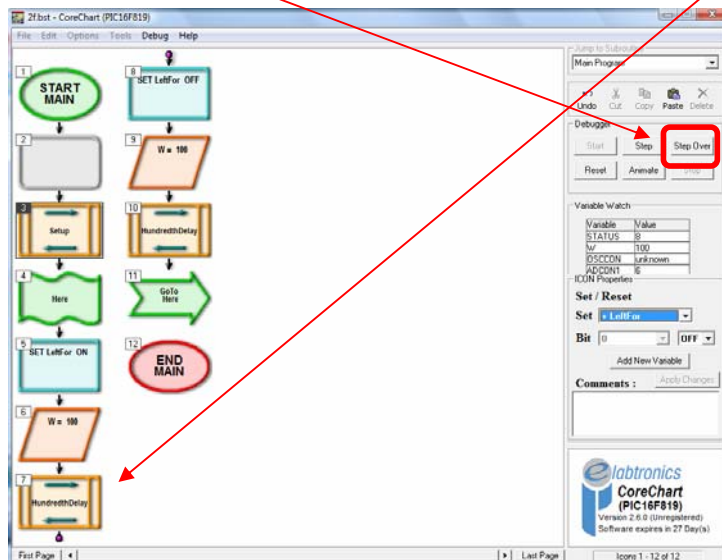


Notice in the panel below that a table now displays the value of each variable in the PIC chip memory. To find the values of each variable press the Step button. The Animate button will step through the program automatically, approx one instruction per second. Errors in the program and hardware could be identified this way.



Step 4:

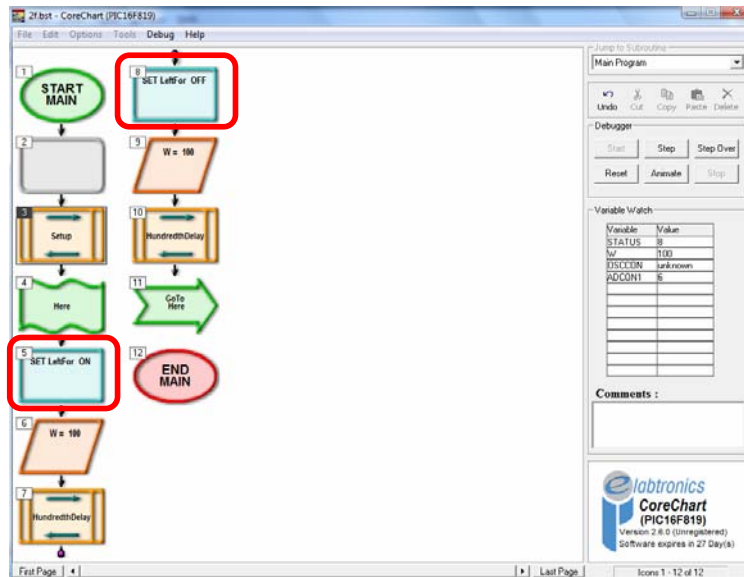
Press Reset button to reset the instruction pointer to the beginning of the program. The press the Step button step through the program. If there is a **delay** icon press the **Step Over** button to bypass it otherwise the debugger will be stuck in the delay loop for a long time until the delay counter cycles are complete.



Notice how the values of each variable changes depending on the instructions.

Continue to step through the program until the SET LeftFor ON icon.
What happens when the debugger steps on to the SET LeftFor ON and SET LeftFor OFF icons?

Notice that the left motor turns forward when the debugger steps onto the SET LeftFor ON icon and turns off when on SET LeftFor OFF icon. If this was not the case, inspect the hardware connections and the program codes for errors.



To stop the debugger, go to the Debug toolbar and select End.

CoreChart will now prompt the user to send the current program to the PIC chip. Select Yes and test the eRacer_16m as in previous exercises.

The Debug tool has the ability to read inputs and control the outputs on the eRacer_16m. It therefore helps to identify program bugs and hardware problems.

This exercise was intended as an introduction to the debugging tool to demonstrate the interaction between the debugger and the eRacer_16m.

Software Workshop Part 3 - eRacer_16m Inputs

Programming eRacer_16m Inputs with CoreChart

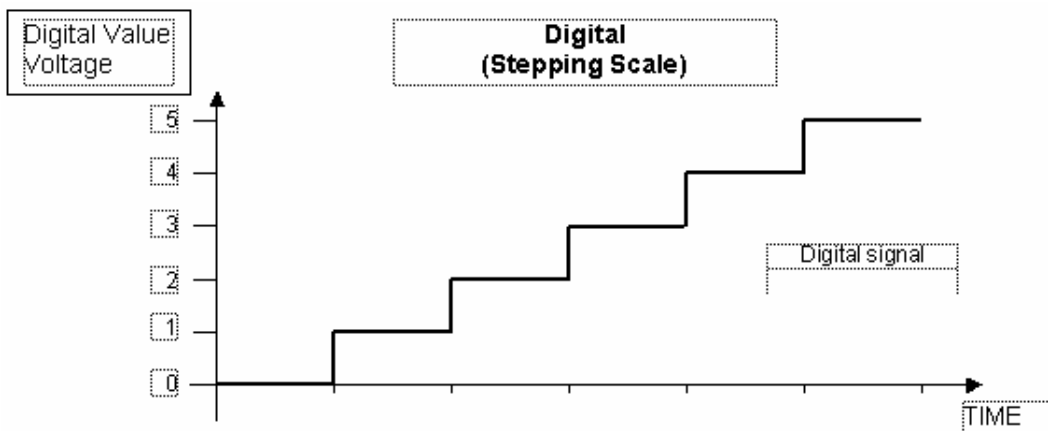
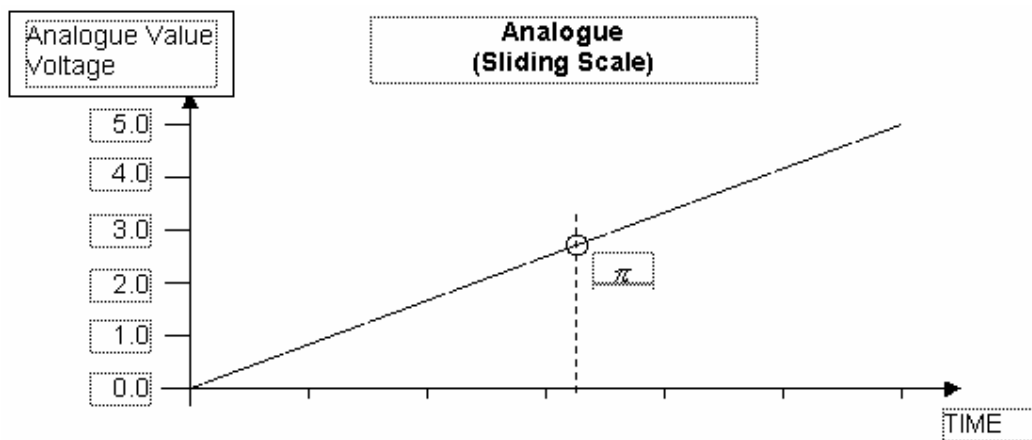
This section discusses the input circuits such as a switch, Infrared Transmitter / Receiver, Light Dependent Resistor etc to the eRacer_16m robot PIC microcontroller. Input signal conditioning techniques are included.

Digital and Analogue representation

Some input signals are analogue and some are digital. For the PIC microcontroller to work the analogue input signals have to be converted to digital values.

Below is an example of an analogue value in the range 0.0 to 5.0 and a digital value in the range 0 to 5.

An analogue value could be 2.6 or 2.6452951 depending on the accuracy. The number of possible analogue values in the analogue scale below is infinite and solely depends on the resolution of the time scale. However, a digital value can only exist at six distinct values: 0, 1, 2, 3, 4, or 5.



What is inside a PIC...the Memory

To understand how to write a CoreChart program to sense and process the inputs let us take a closer look at the structure of the PIC microcontroller.

The figure below shows the most important parts inside the PIC16F819 microcontroller. We have now seen that the PIC is a tiny but complete computer chip. It has a central processing unit (*CPU*), program memory (*PROM*), working memory (*RAM*), and input-output ports.

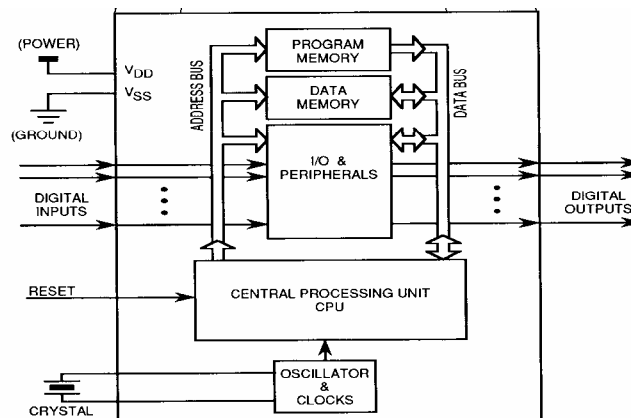


Figure 23: Block diagram of the PIC16F819 chip

The figure shows the PIC16F819 is essentially a single-chip computer. The CPU is of course the 'brain' of the computer. It reads and executes instructions from the PROM. As instructions are interpreted it can store and retrieve *data* in the RAM.

Some CPUs make a distinction between registers located within the CPU and RAM located outside it. **The PIC chip does not.** The general-purpose RAM is also known as 'file registers'. The PIC16F819 has 256 bytes of general-purpose RAM.

CoreChart defines these file registers automatically. They configure the assembler by putting it into PIC16F819 mode and defining the variables. In **CoreChart** variables can be added from the **Variables** menu and will automatically be assigned the next free register.

Besides the general-purpose memory there is a special 'working register' or 'w register' where the CPU holds the data that it is working on.

There are also several special-function registers. They control the operation of the PIC chip.

The special-function registers are divided into 4 register banks. The access to the register banks is controlled by Register Select Bits: RP0 and RP1.

The program memory of the PIC16F819 consists of flash EPROM. It can be recorded and erased electrically and it retains its contents when the power is off. Many other PIC chips require ultraviolet light for erasure and some are not erasable. The PIC16F819 is erasable and electrically reprogrammable over 100,000 times.

The Reset button is a Push button located next to the left motor on the eLab16m. The Reset button is used to reset the chip or to step through a program in the chip.

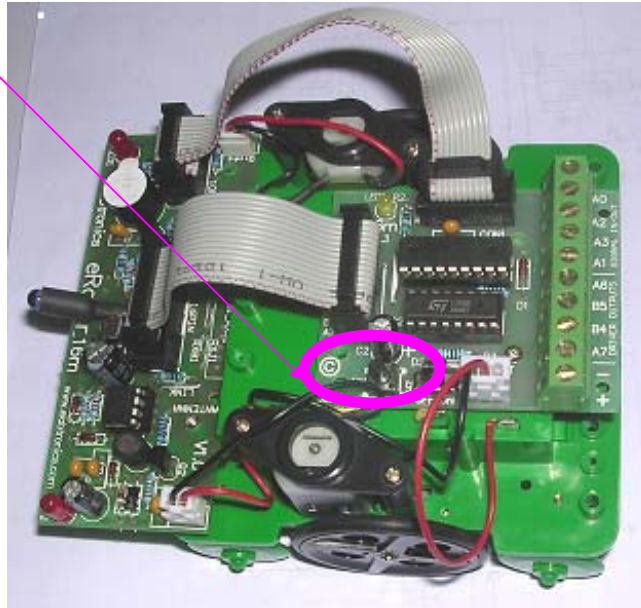
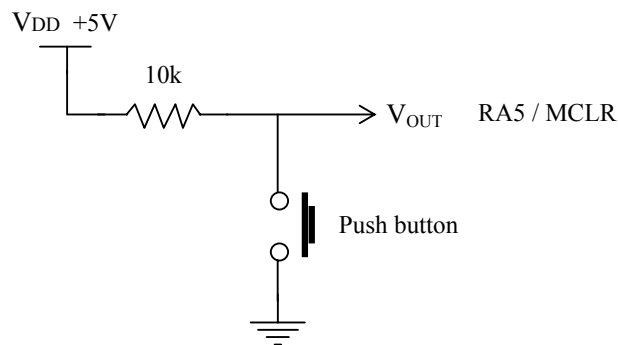


Figure 24: photo illustrating the location of the reset button

Push Button Input

The circuit connection of the Push button on the eRacer_16m is shown in the schematic diagram below.



When Push button is open, $V_{OUT} = 5\text{ V}$. The MCLR input will register as a Logic 1.
When Push button is closed, $V_{OUT} = 0\text{ V}$. The MCLR input will register as Logic 0.

Practical 3a

Testing for Inputs

Step 1: Prepare for new program

Start up **CoreChart** program. Open **start.bst** and save as **press.bst** (Refer to **Software Workshop Part 2- CoreChart Practical 2**).

Step 2: Insert description

Insert a TEXT icon to include your name and the program name **PRESS ON LED**. The description is “**To turn on a LED by pressing a button**”. Save the modified program.

Step 3: Insert new instructions

Insert the following icons.

ICON	PROPERTIES
LABEL	Here
SET	LED4 OFF
IF	ResetButton OFF
SET	LED4 ON
Assignment	W = 20
CALL	HundredthDelay
GOTO	Here

Step 4: Run the program

Save the program and send it to the PIC chip. When the Push button is pressed the Superbright Green LED (LED4) should turn on.

Step 5: Make LED flash

Change the program by inserting “**W = 20** and **Call HundredthDelay**” before the **IF** icon. Re-save program and send to PIC. When the Push button is pressed the Superbright Green LED should flash. Can you work out why?

(Answer: After the Superbright Green LED is turned off, a short time delay is now introduced before it is turned on again immediately).

Step 6: Turn Buzzer On and Off

Change the program so that the Buzzer is switched On and Off when the Push button is pressed. *(Hint: Change the bit that was turned On and Off in the last program.*

Practical 3b

Waiting for an event

Often there is a wait for an event to occur before progression to another section of a program. The following example waits for a switch to be pressed before changing outputs to PORTB.

Step 1: Prepare for new program

Start up **CoreChart** program. Open the **start.bst** program and save it as **wait.bst** (Refer to **Software Workshop Part 2- CoreChart Practical 2**).

Step 2: Insert description

Insert a TEXT icon to include your name and the program name ROBOT WAIT. The description is “Making the robot wait until a button is pressed”. Save the modified program.

Step 3: Insert new instructions

Insert the following icons.

ICON	PROPERTIES
IF	ResetButton OFF
GOTO	START
SET	LED4 ON
LABEL	LABEL1
IF	ResetButton ON
GOTO	LABEL1
SET	LED4 OFF
SET	LED5 ON

The eRacer_16m waits for the Push button to be pressed. When the Push button is pressed the state of the Superbright RED on the eRacer_16m will change.

Infrared (IR)

Object Detection Using Infrared Light

Infrared light is not visible to the human eyes. However infrared light is extremely useful in a wide range of applications.

A very useful application of infrared light is in the detection of objects. This is widely used in hospital operating theatres and in the food preparation industry e.g. a doctor waves his hand past an infrared sensor to turn on and off the water tap in order to avoid the risk of contaminating his hands on the taps.

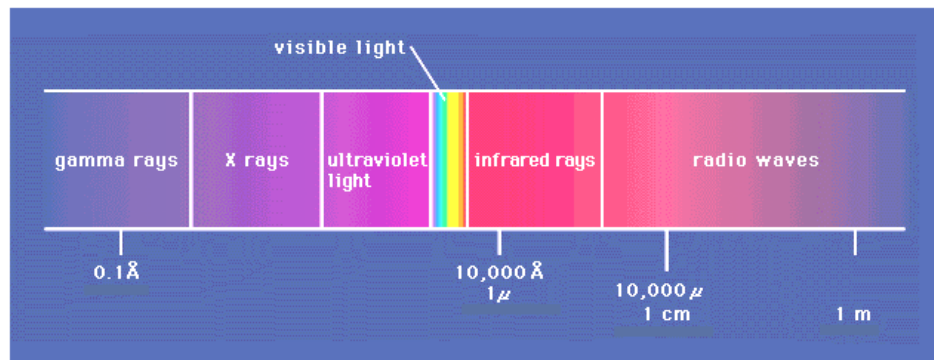


Figure 1: The electromagnetic spectrum.

Infrared IR LEDs are used in opto-electronics such as in auto-focus cameras and television remote controls and as light sources in some long-range optical-fibre communication systems.

The eRacer_16m robot uses its IR LED sensor to determine whether an object is in its path in order for it to try to avoid crashing onto some obstacles.

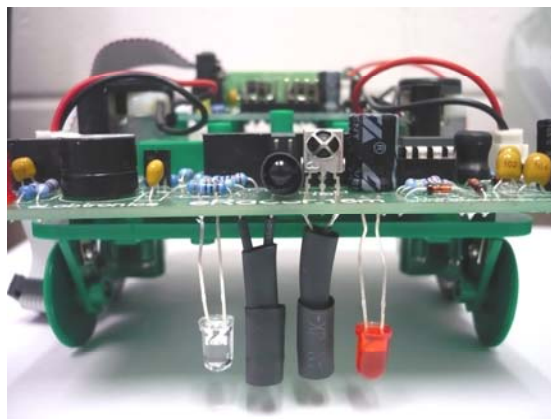
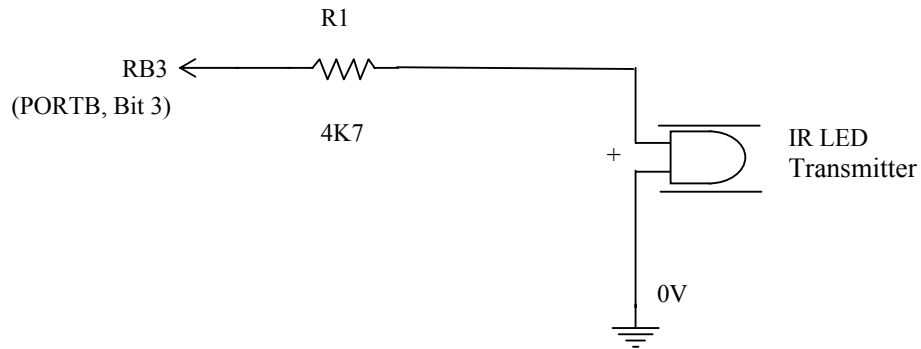


Figure 25: Front view of the eRacer_16m

Infrared (IR) – Transmitter and Receiver

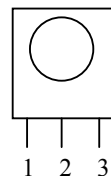
A circuit diagram of the IR LED Transmitter connected to RB3 of the microcontroller.



The resistor value determines the strength of light emitted from the IR Transmitter.

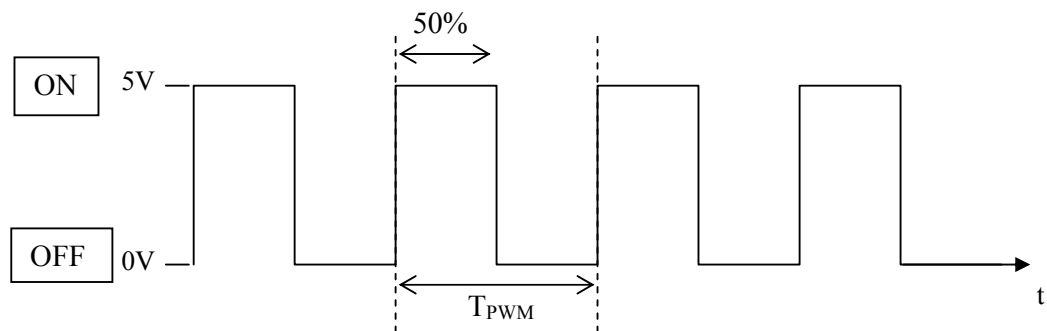
The IR Receiver has three pins.

Front View



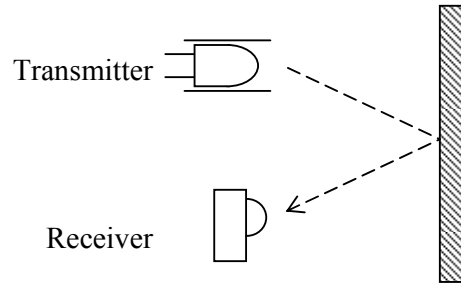
- 1 – signal output
- 2 – negative / ground connection
- 3 – 5 volt power supply

For the IR Receiver to operate properly a square waveform must be supplied to the IR LED Transmitter at a frequency of 38kHz. This is generated by a PWM signal from pin RB3 of the PIC16F819 chip. The duty cycle should be 50 percent.



$$T_{PWM} = \frac{1}{38\text{kHz}}$$

The infrared light emitted from the Transmitter is reflected off objects. The IR Receiver detects the deflected light.



When an *active low* IR Receiver detects IR light the signal pin (#1) will go low (0V). Otherwise the signal pin (#1) will stay high (5V).

Infrared on the eRacer_16m

An infrared IR LED transmitter and an IR Receiver are mounted on the front of the eRacer_16m. As a result the eRacer_16m is able to detect an object in front of it.

The IR LED Transmitter and the IR Receiver work in a similar way to the Super Bright LED and the LDR combination. The infrared Receiver can be thought of as a switch that is turned on by an infrared light similar to the LDR that is turned on by normal light.

Turning on the infrared LED is more complex.

A subroutine called PWM is needed.

Practical 3c

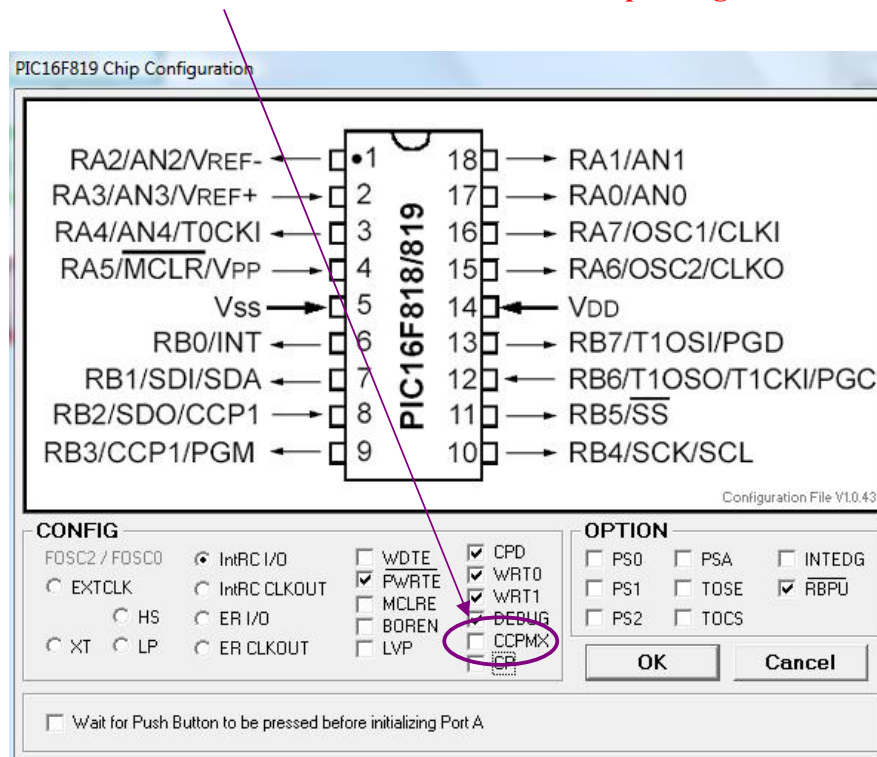
Testing the IR on eRacer_16m

Step 1:

Open **eRacer16mSetUp.bst** in CoreChart. This program is a template which contains the configuration settings for PIC16F189 chip on the eRacer_16m. These settings are already automatically generated when the ezCircuit Designer export the test program to CoreChart. Save this program as **ir.bst**. Delete all other icons between START MAIN and END MAIN except the **Setup** subroutine.

However for the IR Transmitter to operate, the PWM section of the PIC chip must be set correctly i.e. the CCPMX setting **must not** be selected.

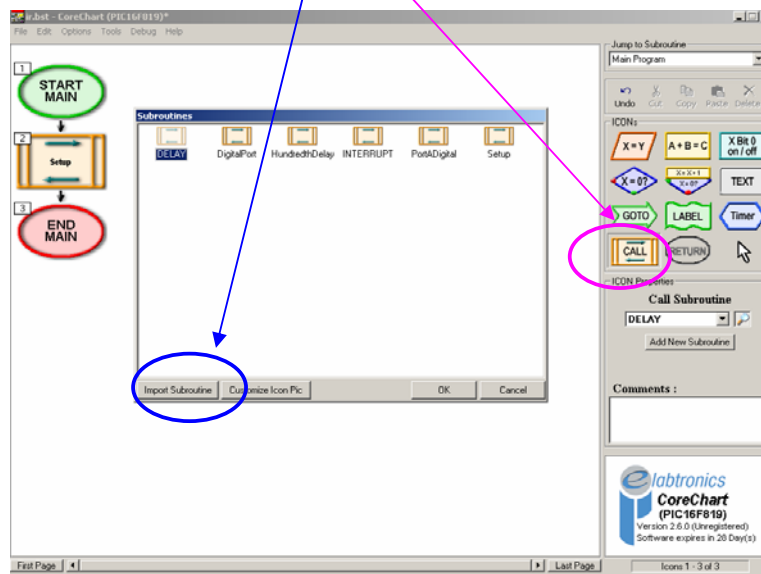
In order to verify this, double click the START MAIN icon to open the PIC16F819 chip configuration window and make sure the radio button next to CCPMX is not selected. Click OK to exit the chip configuration window.



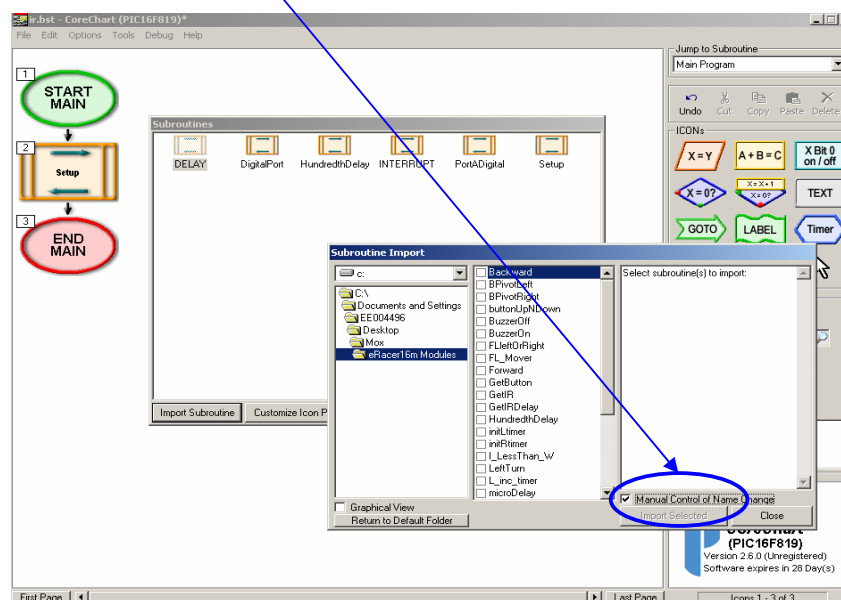
Step 2:

Import **PWM_On**, **PWM_Off**, **PWM_Setup_Off** and **GetIRDelay** subroutines from the eRacer_16m Modules folder into ir.bst. You will need these subroutines for the IR detector. Follow steps 1 to 3 below to import the subroutines.

In the program **ir.bst**, click the **CALL** icon in the **Icons** window. A **Subroutines** window will come up. Click on **Import Subroutine** button in the Subroutines window and a **Subroutine Import** window will come up.

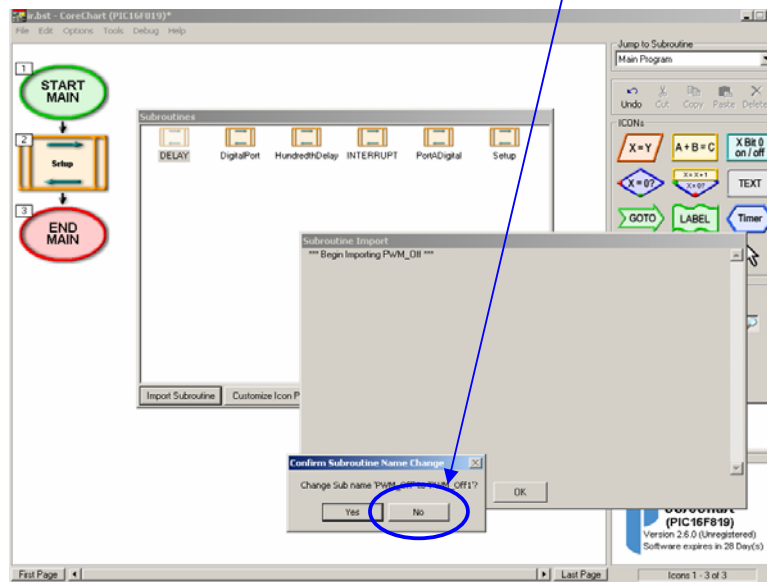


In the **Subroutine Import** window, select the location of the subroutine to be imported, and then select the radio button next to the subroutine to import (Note you can select several subroutines to export them all at once). Select the radio button next to **Manual Control of Name Change**. Click **import selected** and a **Confirm Subroutine Name Change** window will pop up.

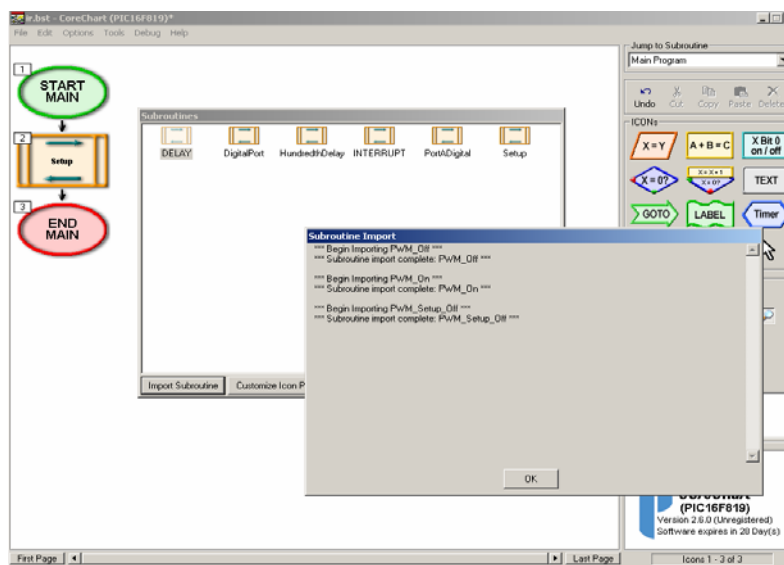


Note: If you do not select “Manual Control of Name Change” option when importing a subroutine, any variable or subroutine name that may already be in use in the program will be automatically renamed.

In the **Confirm Subroutine Name Change** window click **No** to reject any variable or subroutine name changes.



Click **Ok** to exit the Subroutine Import window when finished.



Step 3:

Create a new subroutine by selecting the CALL icon. Click on the “Add New Subroutine” button. Type the name “GetIR” in the text box and click “Add Subroutine”. Close the subroutine window to return to the main program.

Step 4:

Insert a CALL to the “GetIR” subroutine in the main program. Open the “GetIR” subroutine by double clicking on the CALL icon and insert 2 Labels into the subroutine. Label them as “GetIR_Yes” and “GetIR_End”. Insert the following icons before the GetIR_Yes label:

CALL	PWM_On
CALL	GetIRDelay
IF	IRReceiver Off
GoTo	GetIR_Yes
ASSIGN	erResult = 0
GoTo	GetIR_End

Note: A new variable “erResult” will need to be created by clicking the “Add New Variable” button.

Step 5:

Insert the following icon between the GetIR_Yes label and the GetIR_End label:

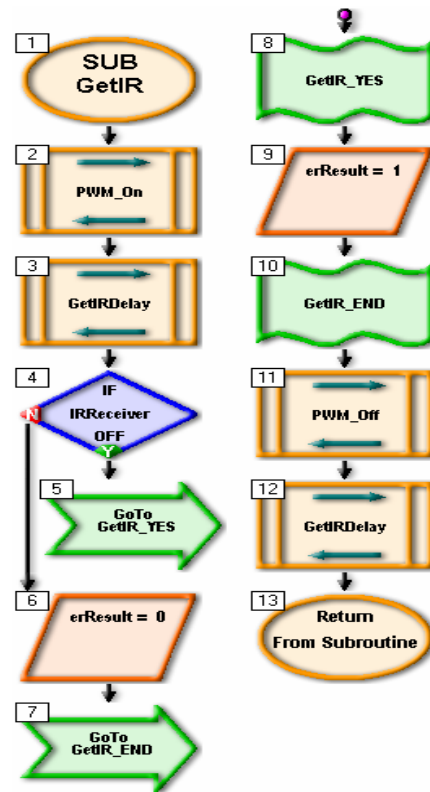
ASSIGN erResult = 1

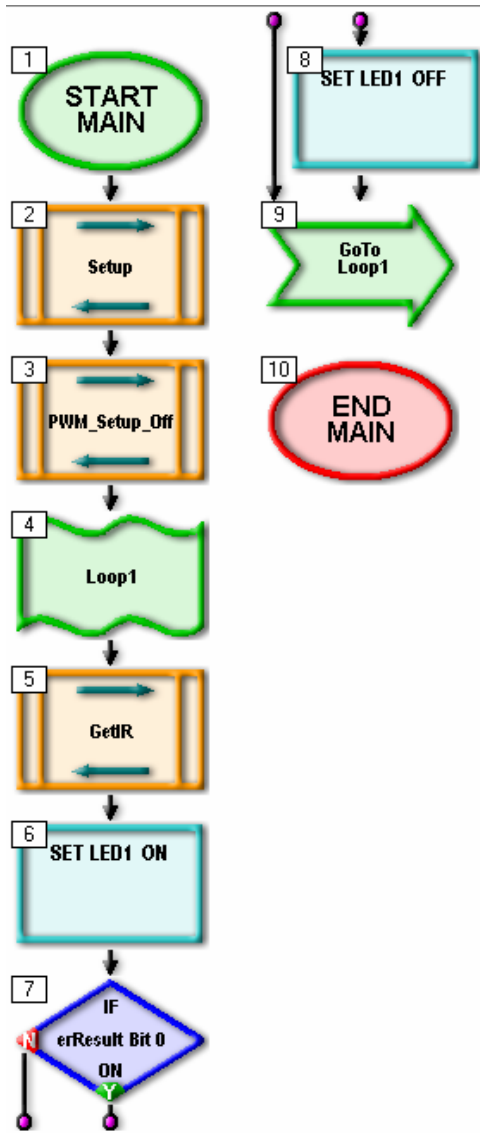
Step 6:

Insert the following icons after the GetIR_End label:

CALL PWM_Off
CALL GetIRDelay

The **GetIR** Subroutine should look like the CoreChart program on the right.





Step 7:

Double click the **Return from Subroutine** icon to return to the main program. Insert the following icons before the CALL GetIR icon:

CALL
LABEL

PWM_Setup_Off
Loop

Step 8:

Insert the following icons after the GetIR icon:

SET	LED1	On
IF	erResult Bit 0	On
SET	LED1	Off
GoTo	Loop	

Your Main program should look like the CoreChart program on the left

Step 9:

Save the program and send it to PIC.

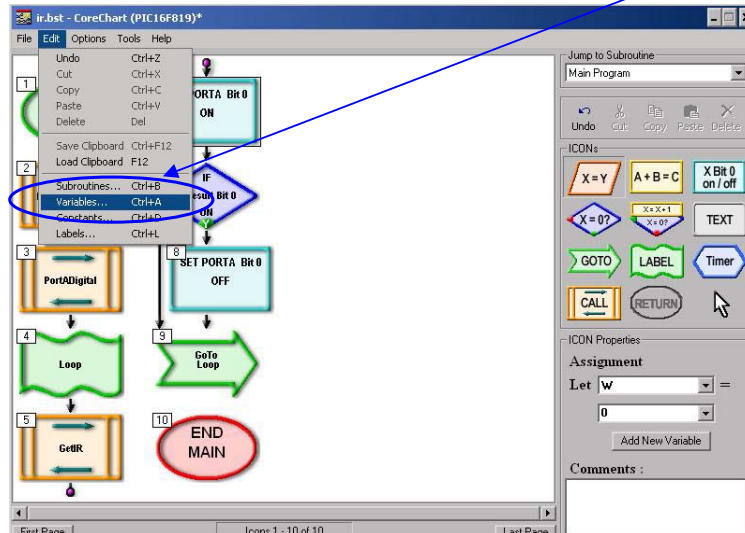
The subroutine **GetIR** starts by switching on the infrared pulse width modulation PWM and delaying long enough for the IR sensor to detect any reflected infrared signal. Port B bit 2 is then tested and the variable **erResult** is assigned the value 1 if a signal is received and the value 0 if a signal is not received.

The Main program simply reads the value of erResult Bit 0 and turns on a red LED if the signal was received.

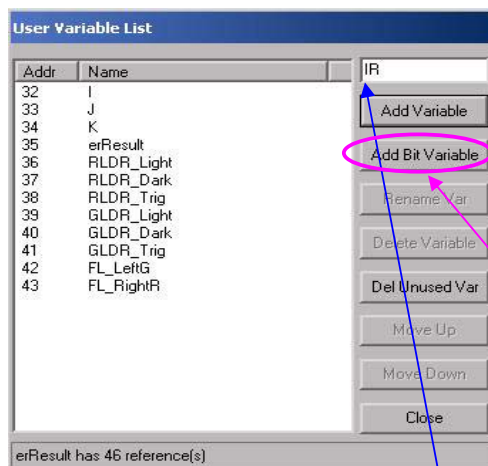
Practical 3c Extension

Bit Variable

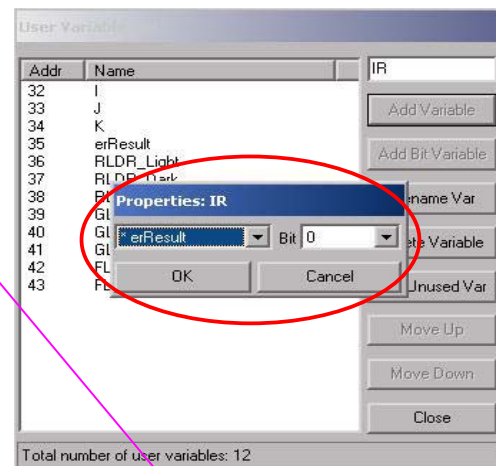
This exercise is to demonstrate the second method of editing/creating Bit Variables through **Variables** that is under the **Edit** menu (or just simply click on **Add New Variable** on Icons). This allows you not only to label Bits on PORTs but also Bits on other Variables, making programming much easier. (Use **ir.bst** and save as **irbits.bst**).



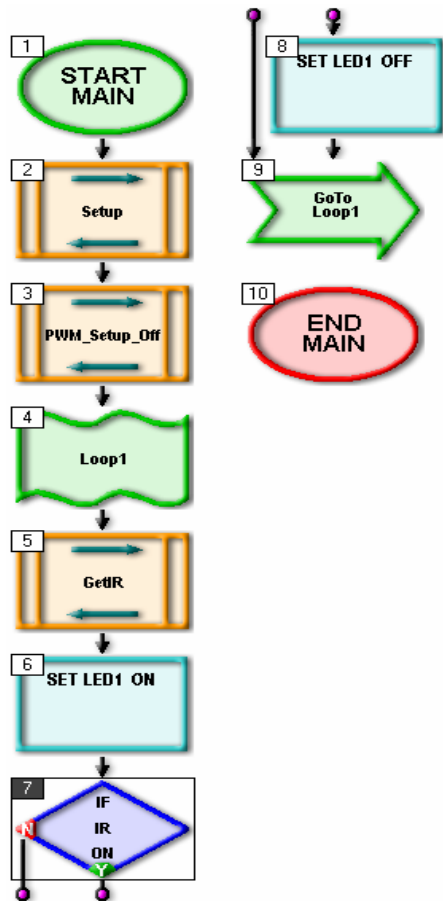
Step 1



Step 2



Type in the name of the new Variable **“IR”** and then click on **“Add Bit Variable”**. A small Properties window will pop up. Choose the Port or Variable you would like to label and the specified Bit. For example choose **“erResult Bit 0”** and click OK when you have finished. A new Bit Variable will be created. Using this method of creating **“Bit Variable”** you can label any Bit of any Variable.



Replace the existing “**erResult Bit 0**” with the **new Bit Variable “IR”** you have just created. The program should look like the CoreChart program on the left.

Using Infrared Remote Control on eRacer_16m

Introduction

This section will make use of the various CoreChart subroutines and commands to make the eRacer_16m move about by infrared remote controls e.g. TV remote and others.

Procedure

Start up CoreChart and open **eRacer16mSetup.bst**. Save this program as **IRfollower.bst**. Delete all other icons between START MAIN and END MAIN except the **Setup** subroutine.

1. Insert a text box and add a description about the program.

Basically the eRacer_16m will continue to move forward until it receives an infrared signal when it will pivot either left or right depending on your program.

2. Import subroutines called: GetButton, PWM_Setup_Off, Forward, GetIR, PivotRight and Stop. This can be done by opening the Options toolbar and selecting Import Subroutine. The subroutines are in the “eRacer16m” folder.
3. Add the following instructions below the text box:

ICON	PROPERTIES
LABEL	ButtonDelay
CALL	GetButton
IF	erResult Bit 0 ON
GOTO	ButtonDelay
CALL	PWM_Setup_Off
LABEL	SenseAgain
SET	LED1 on
SET	LED2 on
CALL	Forward
LABEL	IRSensing
CALL	GetButton
LABEL	OUT

Insert the following between the CALL **GetButton** and LABEL **OUT** icons:

IF	erResult Bit 0 ON
GOTO	OUT
CALL	GetIR

Next insert the following between the **GetIR** and **OUT** icons:

LABEL	IRTest_End
-------	------------

Then insert the following between the **GetIR** and **IRTest_End** icons:

IF	erResult Bit 0 ON
GOTO	IRTest_End
GOTO	IRSensing

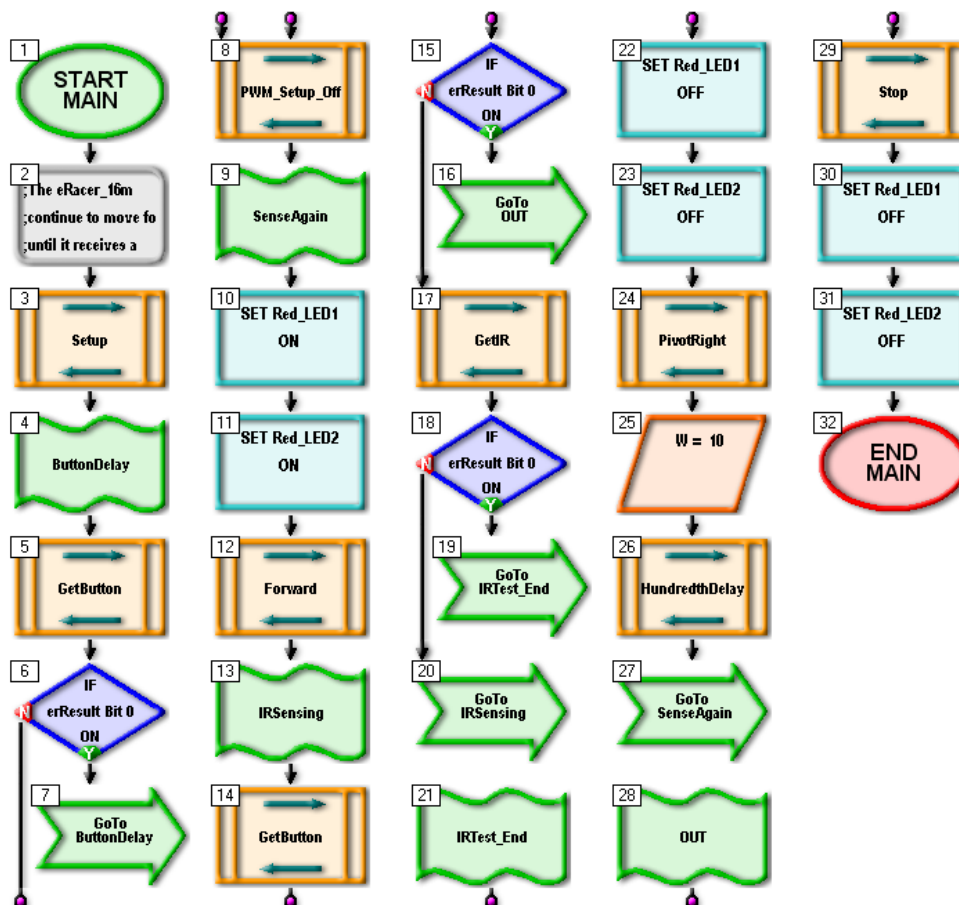
Insert the following under the LABEL **IRTest_End** icon:

SET	LED1 off
SET	LED2 off
CALL	PivotRight (or PivotLeft, but NOT both)
ASSIGNMENT	W = 10
CALL	HundredthDelay
GOTO	SenseAgain

4. Finally insert the following under the LABEL **OUT** icon:

CALL	STOP
SET	LED1 off
SET	LED2 off

5. Save the program. It should now look like this:



6. Ensure that the chip is set to allow PWM operation (see Practical 3c Step1) i.e. make sure that the CCPMX box is not ticked in the Configuration box otherwise the IR transmitter will not operate.

Send program to PIC chip.

7. Test the program. The eRacer_16m should now continuously move forward and pivot left (or right) when it senses an obstacle in front of it.

Use an infrared source e.g. a TV remote control and point it at the eRacer_16m. The eRacer_16m will pivot either left or right according to the program.

You can stop the eRacer_16m at anytime by pressing and holding the reset button.

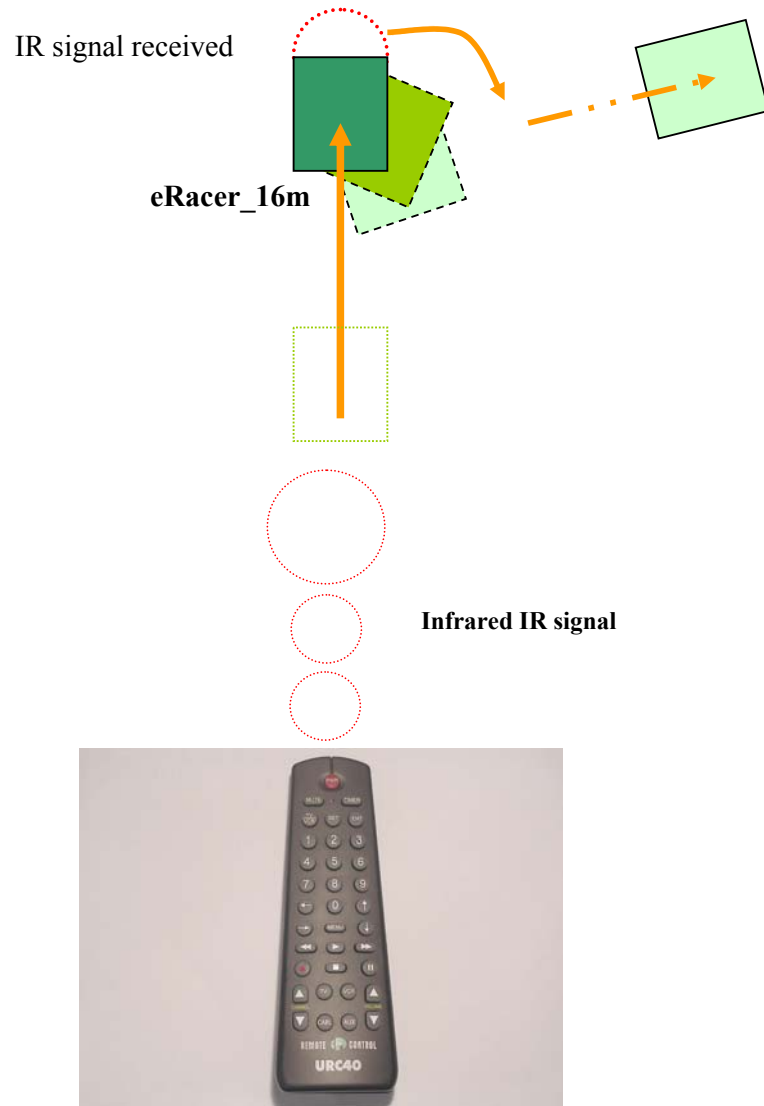


Figure 26: illustration of the remote causing the eRacer_16m to pivot

Try to navigate through an obstacle course with this simple IR wireless control.

Try out different angles and distances to see if the infrared signal can still be detected!

As an extension modify the program so that the eRacer_16m senses two different infrared signals in order to pivot left on one signal and right on the other.

Trouble shooting tips:

1. If the eRacer_16m is moving forward but does not respond to an infrared signal:
 - a. Check that the infrared source is actually emitting a signal. Check the remote control batteries etc.
 - b. Check that the infrared receiver on the eRacer_16m is working correctly. Refer to the eRacer_16m Electronics Construction Manual.
2. If the eRacer_16m continues to turn around in circles with the red LEDs blinking it could be detecting a stray infrared source continuously. Ensure possible interfering IR sources are deactivated.
3. If the eRacer_16m does not move check your program. Check the eRacer_16m can be programmed. Remember to turn on the eRacer_16m when programming.
4. If the eRacer_16m does not program check the manual for hints and tips.

Light Dependent Resistor LDR

An LDR is a resistor that responds to variations in the light that it is exposed to. On the eRacer_16m the LDRs are used for line following and colour differentiating programs.

Two Light Emitting Diodes (LEDs) and two Light Dependent Resistors (LDRs) are mounted next to each other on the track side or under side of the eRacer_16m PCB.

When light shines on the LDR the LDR sends a signal to the microcontroller. The LDR acts as a switch. A CoreChart program could be written such that IF LDR ON, move eRacer_16m forward.

Using this method it is possible to get the eRacer_16m to follow a set pattern. The LED shines on the ground at an angle. If the ground is light in colour and reflective, the light from the LED will be reflected from the surface to the LDR. This causes the LDR to send a signal to the microcontroller.

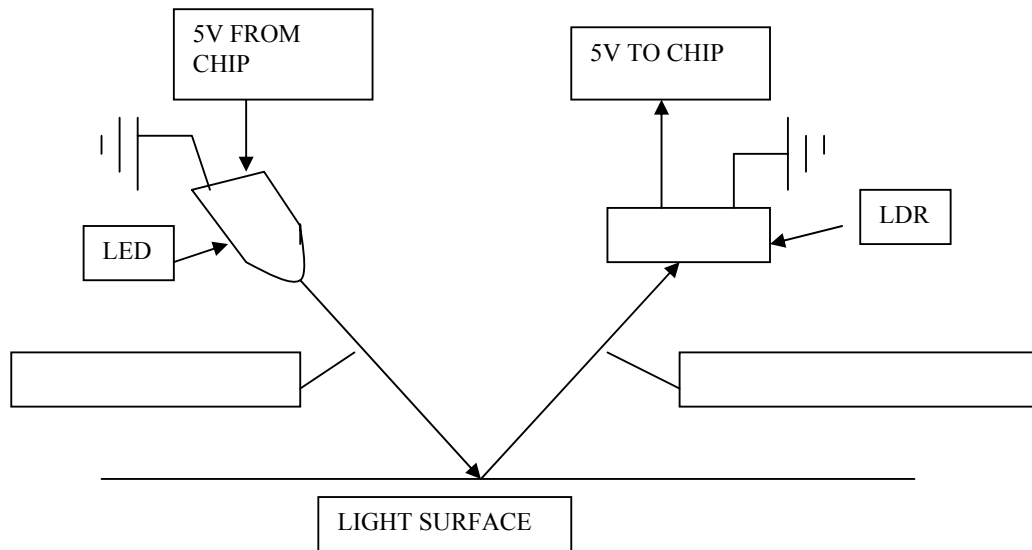


Figure 27: Light reflected from LED to LDR

If however the LED shines onto a dark surface such as a dark carpet then little or no light will be reflected to the LDR and no signal will be sent to the microcontroller.

Analogue to Digital Conversion (ADC)

The LDR can be an On Off switch for Light or Dark condition.

Using the ADC the LDR analogue signals can be converted to digital signals which correspond to a range of light conditions.

The PIC16F819 has an in-built analogue to digital converter (ADC). The ADC converts the analogue signals from the LDRs to 10 bit digital values.

The PIC16F819 has 5 ADC pins i.e. PORT A0, A1, A2, A3 and A4. The analogue voltage signals measured are converted to numbers between 0 and 1023.

Practical 3d

Testing the LDRs on the eRacer_16m

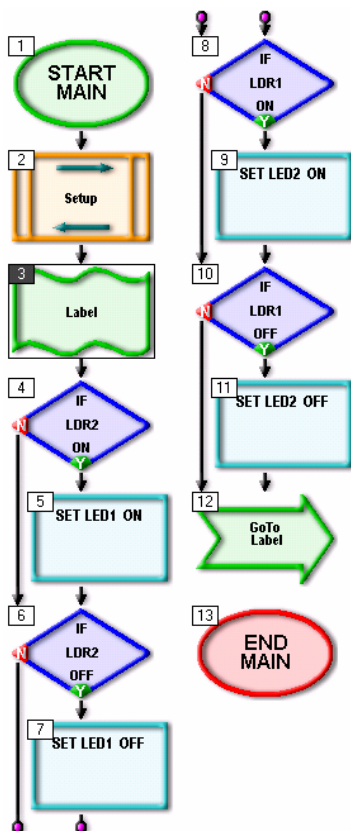
Step 1:

Start CoreChart and open the program **LDRfollower.bst**

Step 2:

Insert the following icons:

Label	Label
If	LDR2 On
Set	LED1 On
If	LDR2 Off
Set	LED1 Off
If	LDR1 On
Set	LED2 On
If	LDR1 Off
Set	LED2 Off
Go To	Label



Step 3:

Check that the LDR test program looks like the figure on the left.

Save the program and send it to the chip.

Step 4:

The LEDs indicate the status of the LDRs.

If there is light on the LDR then the LED on same side of the eRacer_16m will turn on. If there is no light on the LDR then the LED will turn off.

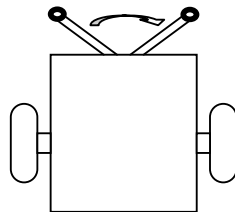
Cover each LDR in turn and observe the status of the LEDs.

To change the LDR sensitivity, adjust the length of heat shrink shroud covering the LDR.

Step 5:

Alter the test program by adding in motor control in the same location in the program that the LEDs turn on and off.

Instead of pointing to the ground, adjust the LDRs so that they are pointing forward and at about 90 degrees to each other. This will enable the eRacer_16m to go towards the brighter area and avoid shadows.



Run the eRacer_16m and observe the reaction.

It is likely you need to adjust the LDR sensitivity. The more even the LDR sensitivity, the better the reaction to the small difference in the light levels.

Software Workshop Part 4 - eRacer_16m Outputs

Introduction

This workshop discusses how CoreChart controls the eRacer_16m robot movement which is determined by the speed and direction of the two motors. The importance of time delays is also discussed.

The two motors on the eRacer_16m robot are connected to a L293 motor driver chip. The driver chip uses the small voltage signals from the PIC microcontroller to control the motor.

The CoreChart program sets specified microcontroller PORT Bits to activate the motor driver chip to rotate the eRacer_16m motors in directions set out in the Table of Port operation for eRacer_16m[®]

eRacer_16m L293 Driver Chip Motor Control

The PIC16F819 microcontroller chip current output is too small to turn the motors. Hence we need to use the L293 driver chip. This chip has four “buffers”, one for each motor terminal. Each of these buffers has two pins. The input pin is connected to the PIC16F819 microcontroller chip and the output pin is connected to a motor terminal.

These buffers output the same voltage as the input but are able to supply larger current.

Each motor has two terminals powered by two buffers of the motor driver chip. These two buffers receive signals from two pins of the PIC16F819 microcontroller.

For the motor to spin one of these two pins must be on while the other is off. The motor will not turn if both pins are on or both pins are off.

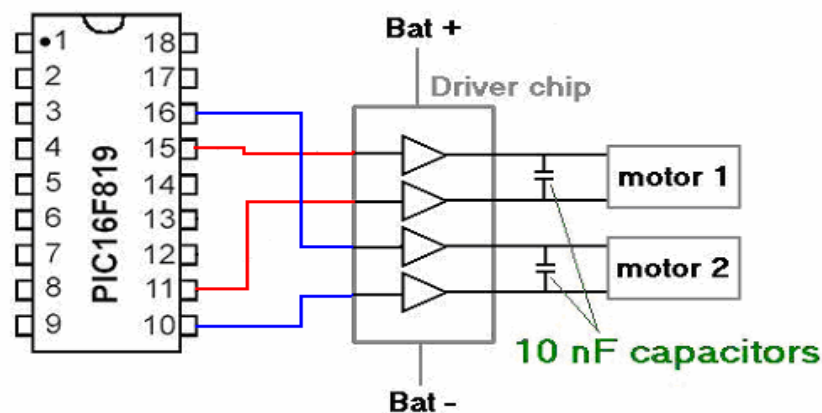


Figure 28: connection between the controller and motors

Outputs from the PIC Microcontroller

- Each Bit in PORT A or PORT B of the PIC chip can output a maximum of 25mA. There is a limit of 200mA per Port of 8 Bits in total.
- Therefore a need a driver to increase the current to drive motors. The L293 driver chip is connected to the designated Port Pins to drive motors.

Table of Port operation for eRacer_16m[©]

Port	Bit	Control	Function
B	0	(not used)	
B	1	(not used)	
B	2	IR Receiver	Input
B	3	IR LED Transmitter	Output
B	4	Left Motor Forward	Output
B	5	Right Motor Reverse	Output
B	6	LDR1	Input
B	7	LED5	Output
A	0	RED LED1	Output
A	1	RED LED2	Output
A	2	Buzzer	Output
A	3	LDR2	Input
A	4	LED4	Output
A	5	Push Button	Input
A	6	Right Motor Forward	Output
A	7	Left Motor Reverse	Output

Practical 4a

Programming the eRacer_16m to Move and Stop

This practical discusses the CoreChart program that moves the eRacer_16m forward for a set amount of time and then stop. The eRacer_16m program switches on the wheels for a set amount of time and then switches off.

Step 1:

Open the file “**movement.bst**” in CoreChart.

Step 2:

Insert the following icons in the Main CoreChart program:

CALL		ButtonUpNDown
SET		MotorLeftForw On
SET		MotorRightForw On
SET		MotorLeftRev Off
SET		MotorRightRev Off
ASSIGN		W = 50
CALL		HundredthDelay
SET		MotorLeftForw Off
SET		MotorRightForw Off
GOTO		START

Step 3:

Save the program in your own file name and send it to the PIC.

Note that the program will not start until the Push button is pressed.

Step 4:

Alter the length of the delays and observe the new patterns that the eRacer_16m traces.

Step 5:

Change the sequence of the motor control in each section of the program and observe the changes to the patterns that the eRacer_16m traces.

Delays

In this section we discuss time delays and why they are necessary in CoreChart programs. We will then show how to construct delay subroutines.

The PIC Microcontroller uses an internal clock to control the execution of program commands. Essentially the internal clock is a digital pulse changing at a constant rate.

The PIC16F819 internal clock pulses at a rate of 8,000,000 cycles per second or 8,000,000 Hertz (or 8 Megahertz or 8 MHz). The period of each clock cycle is $1/8,000,000$ seconds or 0.000000125 seconds (or 0.125 microseconds or 0.125 μ s).

In most cases the PIC16F819 takes 4 clock cycles to execute 1 instruction. Therefore the microcontroller takes **0.5 μ s** to execute **each CoreChart icon** or instruction. Please refer to Figure 29.

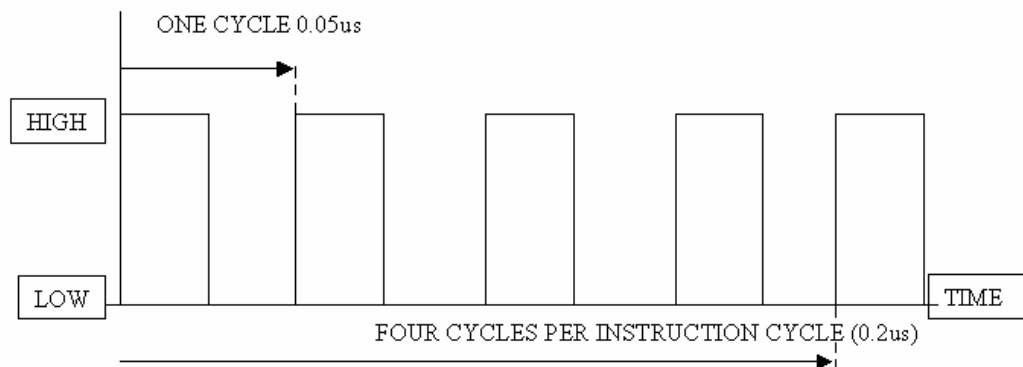


Figure 29: depiction of the delays

The instructions are executed at such a high speed it is sometimes necessary to include a time delay in the program.

Time delays are used in the Flashing LEDs CoreChart program. Here is why.

There are 4 instructions in this example i.e. turn LED on, turn LED off, goto loop to turn LED on again. The goto command is equivalent to two read instructions.

There are 2 instructions between LED on and LED off which is about 1 μ s. At this speed the LED will look as if it is constantly on to the human eye.

However if a suitable time delay is inserted between each LED turn on and LED turn off cycle, the human eye will then be able to see the turning on and off of the LED.

Types of Delays

A time delay is implemented in one of two ways viz. by inserting an icon that does nothing or a delay subroutine that counts. Each time the microcontroller **does nothing** for **one instruction cycle** it pauses for **0.5 μ s**. Therefore a time delay is made up of a multiple of 0.5 μ s.

No Operation

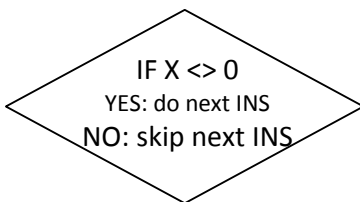
A **no operation** instruction is inserted in the program by clicking on the Timer icon and selecting **no operation**. This provides a specific delay of 0.5 μ s.

Delay Subroutine

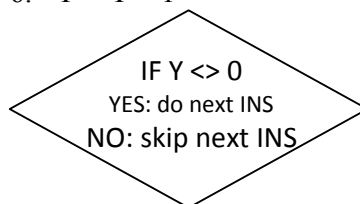
The delay subroutine assigns a value to the W register that sets the length of the delay.

The outer loop will continue to loop until the value of Y in W register ($Y=W$) changes to 0. For each outer loop, the inner loop is executed 255 times. The inner loop has a variable X. When the delay subroutine is called X is equal to zero. However the first arithmetic operation on X makes it equal to 255.

1. SUB DELAY
2. $Y = W$
3. LOOP
4. $X = X - 1$



5. GOTO LOOP
6. $Y = Y - 1$

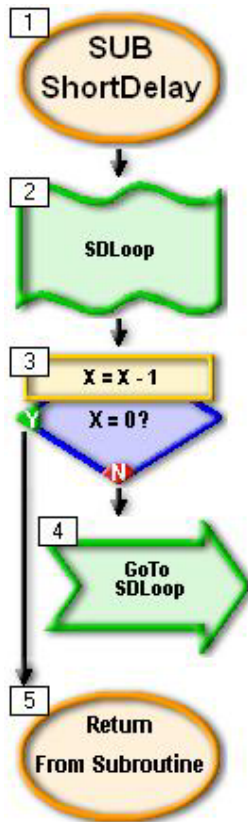


7. GOTO LOOP
8. RETURN

Note: INS = instruction

CoreChart Delay Subroutine Example

The delay subroutine in this CoreChart example uses the Count and Decide icons in between a Label and a GoTo instruction as shown in the diagram.



In this delay subroutine the register X is an 8 bit register that counts from 0 to 255 and then back to 0 again. Note that this delay is a fixed delay i.e it will always delay for the same amount of time except for the first time when it is CALLED. This is because on the first CALL the value assigned to X is unknown. Therefore it will loop an unknown number of times (0 to 255) until it reaches zero.

The next time the delay subroutine is CALLED the value in the X register will start at zero. The next value of X in the register is 255 after decrement by 1 and then 254, 253, until it reaches zero again at which point it returns from the subroutine.

Each delay count iteration consists of 3 CoreChart icons that take $1.5\mu\text{s}$ to complete i.e. $0.5\mu\text{s}$ for the Count instruction and $1\mu\text{s}$ for the GoTo instruction.

The length of time for this delay subroutine is:

$$256 \times 1.5\mu\text{s} = 384\mu\text{s}.$$

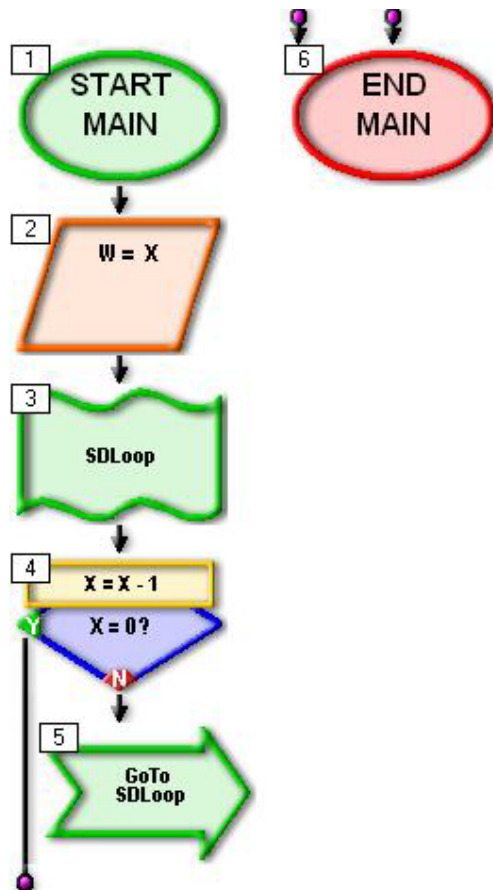
One of the template example programs that come with CoreChart is a subroutine that consists of a loop inside a loop. It counts to 256 and this count is repeated 256 times.

The length of time for this Delay subroutine is:

$$((256 \times 1.5\mu\text{s}) + 1.5\mu\text{s}) \times 256 = 0.1 \text{ s}$$

Variable Time Delays

It is possible to create a “variable time” Delay subroutine. The delay time can be changed by changing the value of W register BEFORE CALLing the Delay Subroutine.



The value of W can be chosen from the range 0 to 255 inclusive.

When this delay subroutine is called in the program the variable W will be assigned the current value of X.

If $W = N$ before this subroutine is called then execution of this delay will assign the value of N to X. Therefore it will loop an N number of times until it reaches zero and returns from the subroutine.

The length of time for this delay subroutine is:

$$N \times (1.5\mu s)$$

Practical 4b

Delay Subroutine to Create Sounds of Varying Frequencies

In Practical 2c we created a program to turn the LED On and Off by using Time Delay subroutine. Similarly we can use Time Delay subroutines to turn a Buzzer On and Off.

If you turn On and Off the Buzzer slowly then you can hear the Buzzer turn On and Off. When the Buzzer is turned On and Off very fast only a single tone will be heard.

Hence if the Time Delay between switching On and Off (the Period) is decreased sufficiently then a single tone will be heard. The shorter the Time Delay the higher the Frequency or the Pitch of the Buzzer.

The Period is calculated by adding the On time and the Off time of the Buzzer. The Frequency is the number of Periods that fit into one Second. The Frequency is then the INVERSE of the Period. ($F = 1 / T$ The unit of F is Cycle Per Second and the unit of T is Second).

The following program produces a tone when the Push Button is pressed.

Step 1:

Open the file pitch.bst. Select the CALL icon. Create a new subroutine and call it 'shortdelay'.

Select the 'shortdelay' subroutine and insert it in a new Main program.

Step 2:

Open the 'shortdelay' subroutine and insert the following instructions:

Note: You need to create the Variable "X". Open the "User Variable List" by clicking on Edit / Variables at the top of the menu bar. Type in "X" in the top right hand box. Click on "Add Variable" and close the "User Variable List".

ICON	FUNCTION
ASSIGNMENT	$X = W$
LABEL	sdloop
COUNT AND DECIDE	$X = X - 1$
GOTO	sdloop

Step 3:

Double click on "Return from Subroutine" icon to go back to the Main program.

Step 4:

Insert the following icons in the Main program:

ICON	FUNCTION
ASSIGNMENT	W = 100
LABEL	here
SET OFF	Buzzer1
CALL	shortdelay
IF OFF	ResetButton
SET ON	Buzzer1
CALL	shortdelay
GOTO	here

Step 5:

Save the program and send it to PIC.

Step 6:

Change the value of $W = 100$ to $W = 15$. What effect does this have on the pitch? Can you calculate the frequency of the pitch? What is the threshold frequency of human hearing?

Practical 4c

Sliding the Tones of the Buzzer

The next exercise explores what happens if we slowly change the Time Delay while the Buzzer is On. In this case the pitch would change from high to low or low to high producing a WHHOOOOP sound

Step 1:

Delete all the icons between START and END except the **Setup** icon from the **pitch.bst** program and save it as **slide.bst** so that we can reuse the 'shortdelay' subroutine.

Step 2:

Select the CALL icon and create a new subroutine named '**play**'. Select the '**play**' subroutine and insert it in the **slide.bst** Main program.

Step 3:

Open the '**play**' subroutine and enter the following:

ICON	FUNCTION
SET	Buzzer1 ON
CALL	shortdelay
SET	Buzzer1 OFF
CALL	shortdelay

Step 4:

Double click on "Return from Subroutine" to go back to the Main program.

Step 5:

Click on the "Assignment" icon and click "Add New Variable" and add a new variable called '**slide**'.

Step 6:

Insert the following icons in the Main program:

ICON	FUNCTION
ASSIGNMENT	$W = 1$
ASSIGNMENT	$SLIDE = 0$
ASSIGNMENT	$Y = 0$ (add another variable Y as in step 5)
LABEL	sound
COUNT AND DECIDE	$Y = Y - 1$
CALL	play
CALCULATE	$SLIDE = SLIDE + 1$
ASSIGNMENT	$W = SLIDE$
GOTO	sound

Step 7:

Send the program to PIC. You should hear a sliding sound of decreasing pitch.

How long does it take slide through all the 256 tones? How could the tones be made to slide upwards?

Change the icons in Main program to:

ICON	FUNCTION
ASSIGNMENT	$W = 1$
ASSIGNMENT	$SLIDE = 0$
ASSIGNMENT	$Y = 0$
ASSIGNMENT	$CHANGE = 0$ (add variable CHANGE as in step 5)
LABLE	sound
COUNT AND DECIDE	$Y = Y - 1$
CALL	play
COUNT AND DECIDE	$CHANGE = CHANGE + 1$
GOTO	sound
CALCULATE	$SLIDE = SLIDE + 1$
ASSIGNMENT	$W = SLIDE$
GOTO	sound

How long does it take this altered program to cycle through all 256 tones?

CHALLENGE:

Modify slide.bst to only operate when Push button is pressed as in Practical 3a.

Practical 4d

Pulse Width Modulation PWM Controls eRacer_16m Motor Speed

This practical explains the control of the eRacer_16m motor speed by using Time Delay subroutines. There are no hardware changes to the eRacer_16m.

The eRacer_16m motor speed can be varied if we can make the motor start and stop for very short duration of time through Time Delay subroutines in the program. If there are no Time Delays in the CoreChart program the eRacer_16m motors will rotate at maximum speed.

If we make the motor start and stop for very short duration of time through Time Delay subroutines, the result may be a net forward movement at a fraction of the maximum possible speed.

The actual speed depends on the length of the delays and the relationship between the ON time and OFF time. For this to work, the OFF time must be less than the ON time.

$$\text{Actual speed} \cong \frac{\text{ONtime}}{\text{ONtime} + \text{OFFtime}} * \text{MaximumSpeed}$$

Step 1:

Open the “start.bst” program. Use the CALL icon to create new subroutines named ‘OnDelay’ and ‘MotorDelay’. Select ‘MotorDelay’ from the subroutine list and insert it in the main.

Refer to Subroutines section of Software Workshop Part 2 – CoreChart.

Step 2:

Open the ‘MotorDelay’ subroutine and insert the following instructions:

You will need to add the Variable ‘Speed’ by clicking on the Assignment icon and inserting a new Variable.

ICON	FUNCTION
LABEL	Loop2
ASSIGNMENT	W=20
ASSIGNMENT	X=W
LABEL	Loop1
COUNT AND DECIDE	X = X –1?
GOTO	Loop1
COUNT AND DECIDE	Speed = Speed–1?
GOTO	Loop2

Step 3:

Insert the following instructions in the main program:

ICON	FUNCTION
LABEL	here
SET	MotorLeftForw ON
SET	MotorRightForw ON
ASSIGNMENT	W = 10 (* Ontime of the motors)
ASSIGNMENT	Speed = W
CALL	MotorDelay
SET	MotorLeftForw OFF
SET	MotorRightForw OFF
ASSIGNMENT	W = 1 (*Offtime of the motors)
ASSIGNMENT	Speed = W
CALL	MotorDelay
GOTO	here

Step 4:

Save the program in your file as **motorspeed.bst** and send it to the PIC.

Step 5:

Changing the value of W alters the motor speed. The ratio of Ontime / Offtime of the motors determine the speed.

W is an 8-bit register that has a maximum value of 255. Therefore the maximum value for Ontime or Offtime for W is 255.

If the value of W is less than 4 the motors may not rotate because the time between motor on and motor off is too short. Increase the value of W or start motor manually.

Practical 4e

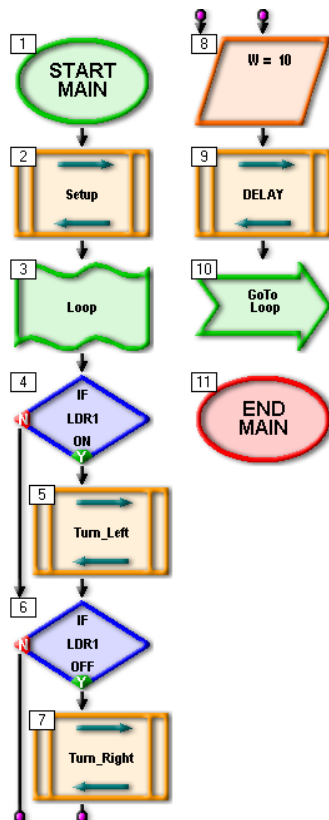
Light Dependent Resistor LDR To Control eRacer_16m Movement

Step 1:

Open the file “eRacer16mSetup.bst” in CoreChart created in Practical 1.
Delete all subroutines except “Setup”.

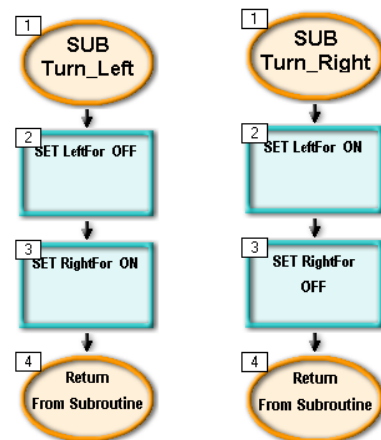
Step 2:

Create new subroutines named ‘Turn_Left’ and ‘Turn_Right’.
Add the icons shown in the figure below:



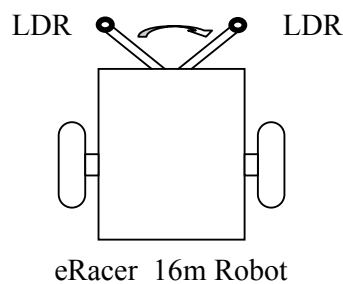
Step 3:

Add the icons shown in the figure below to subroutines “Turn_Left” and “Turn_Right” :



These modifications to the program cause the eRacer_16m robot to move in accordance to the LDRs. By positioning the LDRs to face forward and at right angles to each other this program moves the eRacer_16m towards the light.

Test this by placing the robot into a dark room and shining a torch onto the LDRs. The robot will move towards the torchlight. When a shadow is cast over the LDRs the robot will move away from the shadow to the brighter area.



Making LDRs Detect Colours

The LDRs can be made to detect colours by using different colour cellophane. For example the LDRs can detect the colour lines of a robotic competition map if they are covered with colour cellophane as shown below.

Place a small piece of red cellophane over the face of an eRacer_16m LDR and fold it over the legs.

Slide a heatshrink about 1 cm long over the LDR so that the edge of the heatshrink is level with the face of the LDR.

Apply heat so that the heatshrink presses the cellophane onto the LDR. Repeat the above with green cellophane onto the other eRacer_16m LDR.

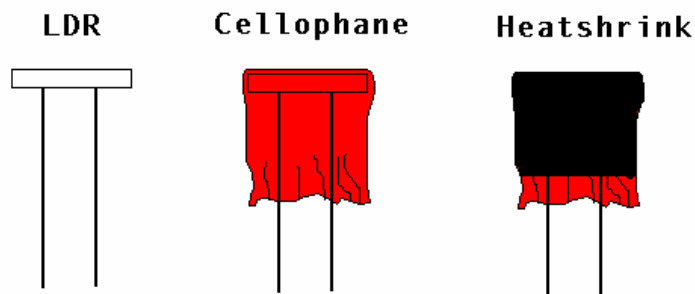


Figure 30: Putting cellophane over the LDRs

Practical 4f

Avoid Objects

The eRacer_16m robot uses the Infrared Transmitter and Receiver to detect and avoid objects in its path. The following CoreChart subroutine demonstrates this eRacer_16m feature.

Step 1:

Open “**ir.bst**” and save as “**avoid.bst**”.

Step 2:

Delete the 3 icons after the GetIR icon

Step 3:

Create new subroutines called: Backwards, Left_Turn and Forward. This can be done pressing ctrl+B or opening the Edit toolbar and selecting Subroutines.

Step 4:

Create a subroutine called “**Dodge**” and insert the following icons:

CALL	Backwards
ASSIGN	W=20
CALL	HundredthDelay
CALL	Left Turn
ASSIGN	W = 50
CALL	HundredthDelay

Step 5:

After the CALL GetIR icon insert the following:

IF	erResult bit 0 On
CALL	Dodge
IF	erResult bit 0 Off
CALL	Forward

Step 6:

Send program to chip.

Refer to the eLabtronics Robotic Line Following Competition section of this eRacer_16m manual and program your eRacer_16m to follow the line.

eLabtronics Robotic Line Following Competition

Aim

This is a fun race! Each **eRacer_16m** robot will be given one try to go as far and as fast as it can. Our race track is a black project paper sheet mat with a white line twisting and turning from start to finish. The track starts out with gentle turns and then it gets more complex towards the finish.

Robot Specifications

Robots must be self-operated i.e. programmed to run autonomously.

All robots must be checked and recorded by the competition officials before the beginning of the Line Following Contest. No allowance will be made for adjustments after the Line Following Contest has begun.

Participants who qualify for the Line Following Finals may bring their robots back to their schools after the preliminaries and make any desired adjustments. All robots will be re-checked by the competition officials before the Line Following Finals.

Competition Rules

When a “Start” command is given, the contestant will start his/her robot.

The robot cannot be controlled by or interfered with by the participant or anybody else during the competition.

A timer will be started when the “Start” command is given.

The timer will be stopped if the robot leaves the Line. The distance will be marked at the place where the robot left the Line.

The timer will be stopped if the robot stops making forward progress. For example, if the robot turns and begins following the Line in the opposite direction, the timer will be stopped. The distance will be marked at the place where the robot turned around.

A three-minute time limit will be enforced. If the robot has not completed the Line in three minutes, the distance will be marked at the point where the robot is at that time.

The robot must go straight through any four-way intersections, if they exist in the competition. If the robot goes either right or left at a four-way intersection, the timer will be stopped and the distance will be marked.

The timer will be stopped when the robot crosses the end of the Line.

The winner will be declared as the robot that travelled the longest distance along the Line. If more than one robot reaches the end of the Line, the winner will be the robot that completes the Line in the shortest time.

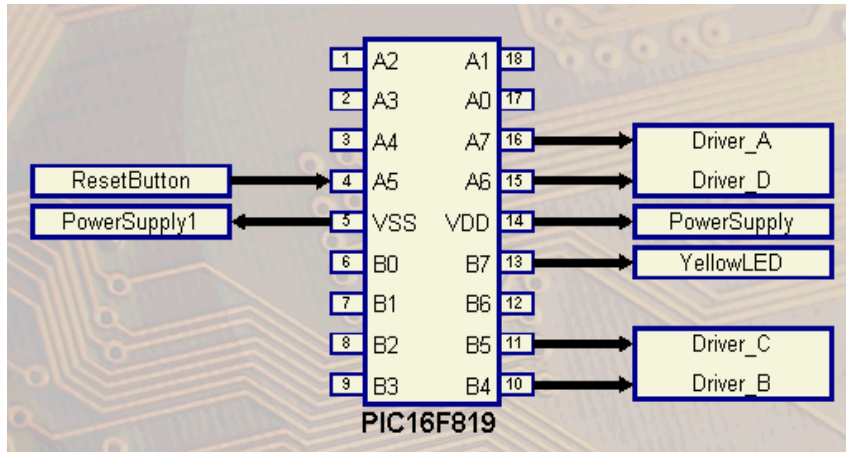
Hints and Tips

The Competition black paper has a shiny white Line using white electrical tape. If you are not using white electrical tape to test your robot, you may need to change its configuration to detect the black edge along the Line.

Appendix Further ezCircuit Designer projects

As you would have discovered by now the eLab16m controller on the eRacer_16m can be re-used for further projects. Students can design their own project applications using ezCircuit Designer and the eLab16m controller. The following six eLab16m projects are found in the eLabtronics ezCircuit Designer Starter Pack.

Project 1: LED Flash Example



Learning Outcome

This example introduces the user to the ezCircuit Designer, CoreChart and the reusable eLab16m mini controller board.

The “LED Flash Example” quickly tests the installation of ezCircuit Designer and CoreChart Software, the USBP Programmer and the eLab16m controller board.

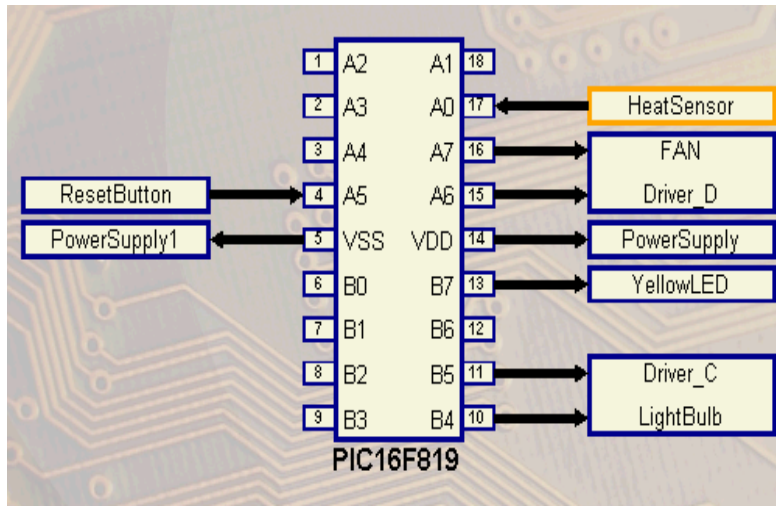
The aim is to encourage the user to design PIC microcontroller electronic projects.

A typical example: a team of Secondary School students used ezCircuit Designer, CoreChart and eLab16m to design, build, document and program a touch pad device for autistic children to communicate through an I-pod.

Project 2: Thermo Fan Controller

Learning Outcome

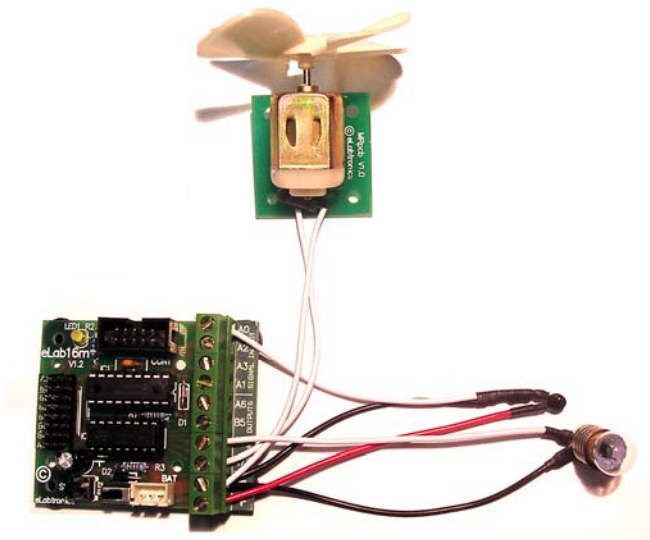
This workshop gives the user a hands-on experience on how to design, build, document and program a temperature controller



Temperature controller

A temperature controller or regulator is commonly found in air conditioner, fan coolers for car engines and inside computers to cool down the microprocessors.

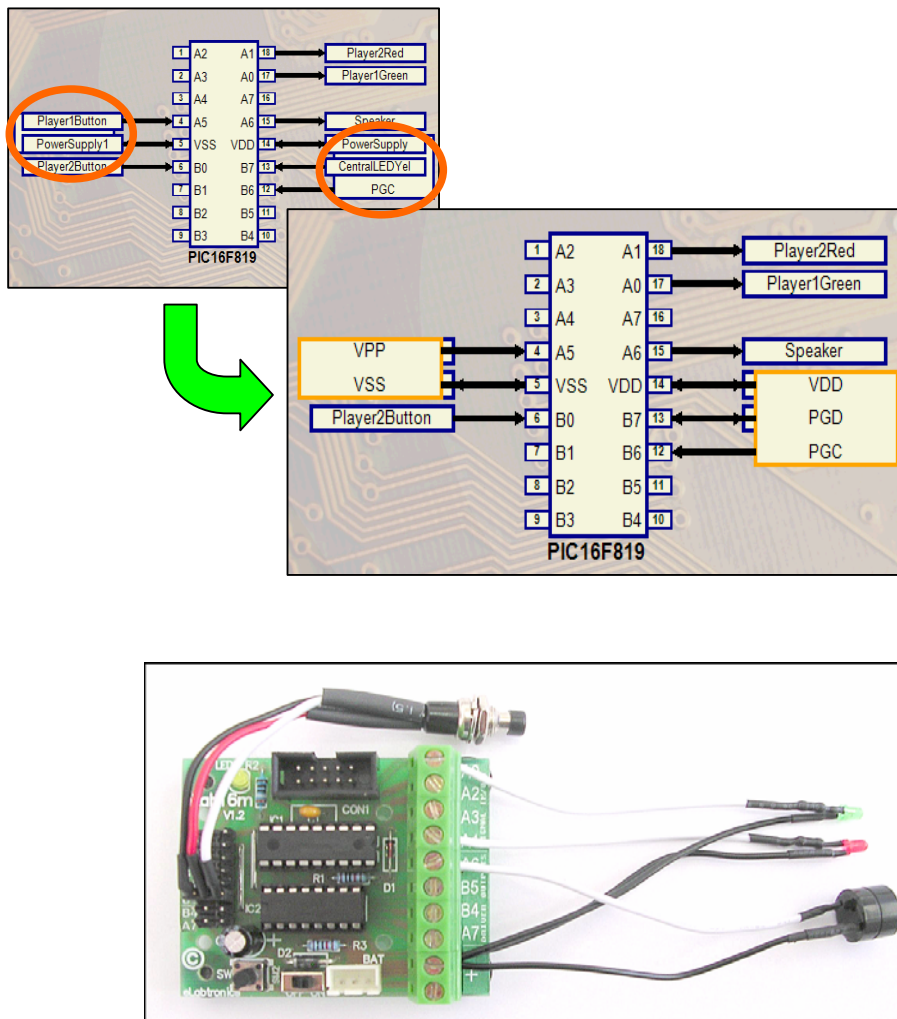
In this project, the thermistor will be the temperature sensor, the light blub will be the heater and the fan is the cooler.



Project 3: Reaction game

This 3 part project gives the user a hands-on experience on how to design, build, document and program a two player Reaction Game.

1. Part 1: Design Project
2. Part 2: Build Project
3. Part 3: Program Chip

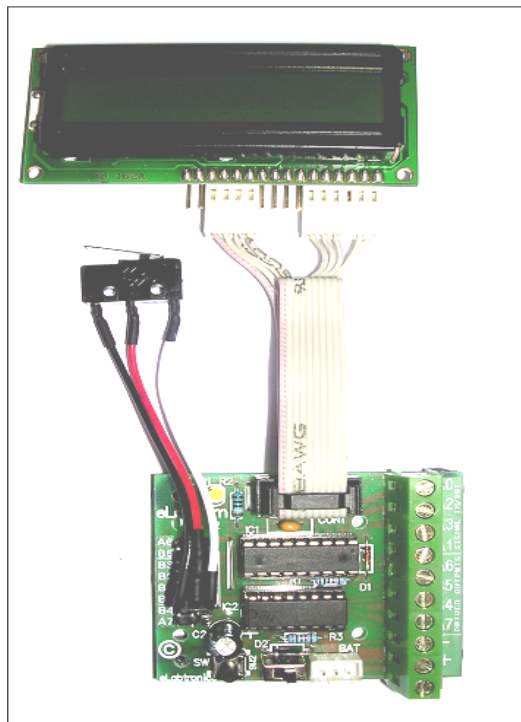
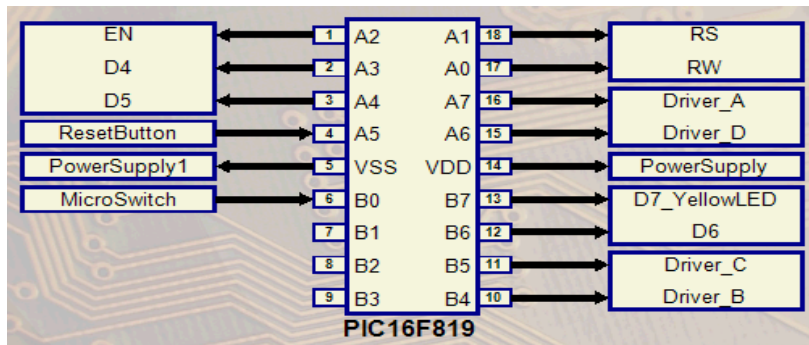


The 2 player Reaction Game is built by adding circuits to the pre-built eLab16m reusable Controller Board.

This project instills the 'ureka effect' in the young creative students because he can claim 'I am the inventor'! He / she has designed and built his / her very own electronic game! ☺

Project 4: LCD Micro-Switch Counter

Design micro-switch counter with liquid crystal display (LCD)



Learning Outcome

This workshop gives the user a hands-on experience on how to design, build, document and program an Electronic Counter using a micro-switch and display the count result on a Liquid Crystal Display (LCD).

Micro-switch Counter with a LCD Display

Generally, it is a challenge to program LCD displays. The following practical explains the steps to build a micro-switch electronic counter and display the count result on an LCD. Electronic counters form an important part of electronic control systems.

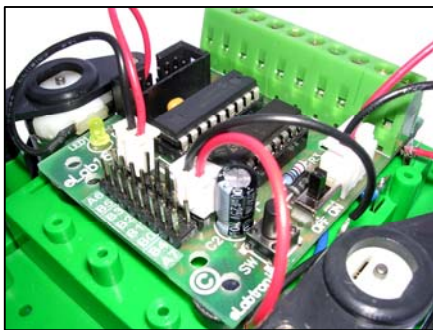
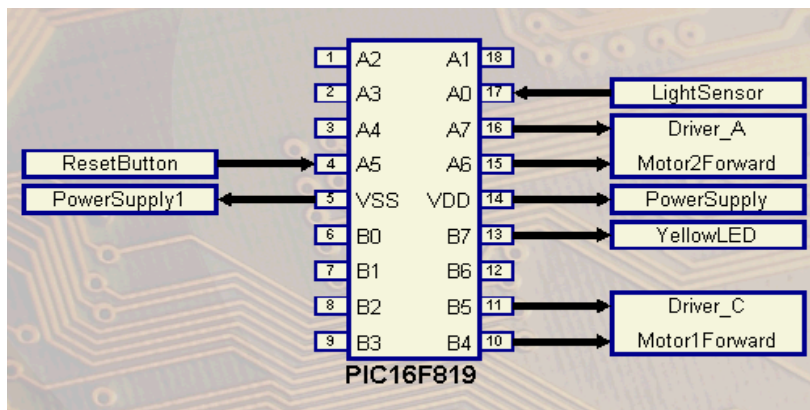
Project 5: LDR Line Following Robot

Design a Line Following Robot using a Light Dependent Resistor (LDR)

Learning Outcome

This project gives the user a hands-on experience on how to use ezCircuit Designer (ezCD), CoreChart and the eLab16m to design, build, document and program a “Line Following Robot”.

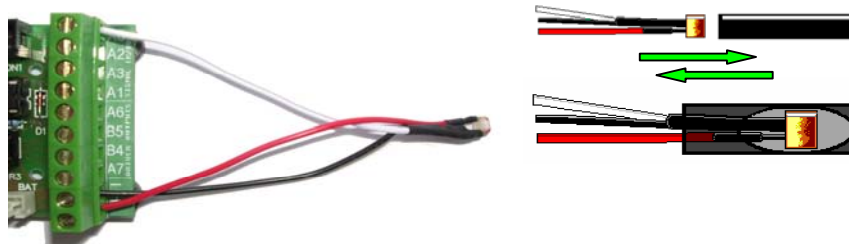
The aim is to encourage the user to design other PIC microcontroller electronic projects.



A typical example: A team of Secondary School students used ezCircuit Designer (ezCD), CoreChart and eLab16m to design, build, document and program a touch pad device for autistic children to communicate with an I-Pod.

Line Following Robot

Building a robot is no longer rocket science. With only one LDR (Light Dependent Resistor) and two motors, a simple robot can be built to follow a line.



Project 6: A/D LDR Light Level switch

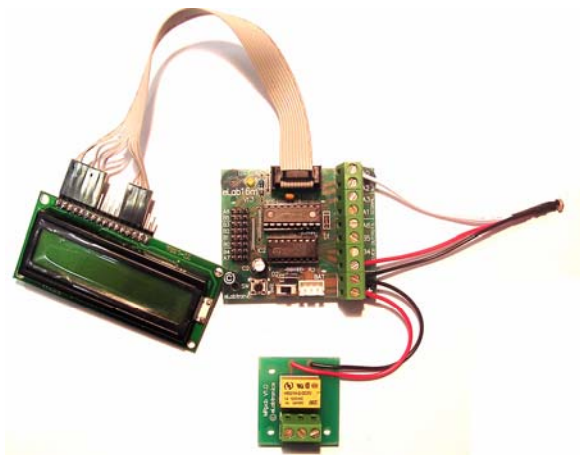
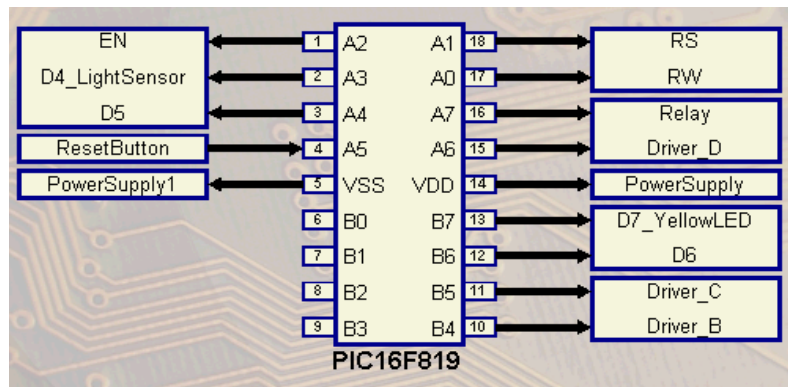
Design Analogue Light Detector

(Project 6 requires Full CoreChart & ezCircuit Designer licences)

Learning Outcome

This workshop gives the user a hands-on experience on how to design, build, document and program an electronic light detector and an introduction to Analogue to Digital A/D conversion.

An example of A/D is the conversion of voltage levels to digital values for the use by microcontrollers.



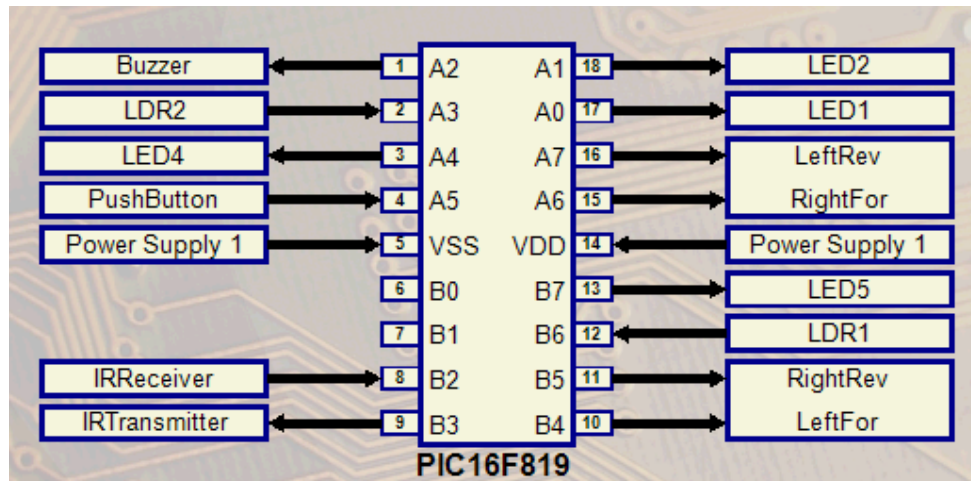
Analogue Light Detector

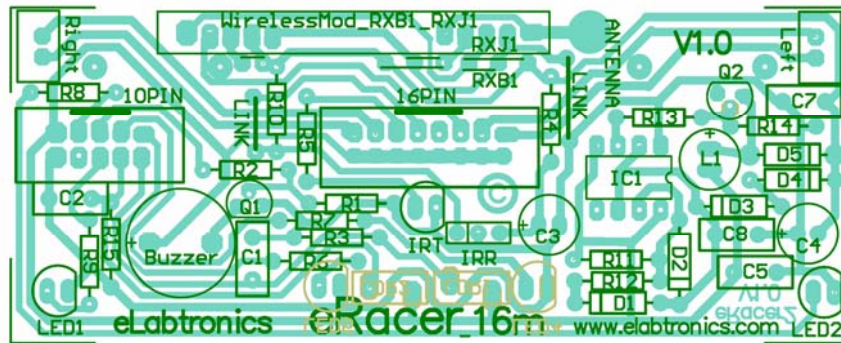
An analogue light detector measures the light level of the surrounding. For example, an automatic window shutter extends or retracts depending on the light level in the room.

Quick Reference

Port	Bit	Control	Function
B	0	(not used)	
B	1	(not used)	
B	2	IR Receiver	Input
B	3	IR LED Transmitter	Output
B	4	Left Motor Forward	Output
B	5	Right Motor Reverse	Output
B	6	LDR1	Input
B	7	LED5	Output
A	0	RED LED1	Output
A	1	RED LED2	Output
A	2	Buzzer	Output
A	3	LDR2	Input
A	4	LED4	Output
A	5	Push Button	Input
A	6	Right Motor Forward	Output
A	7	Left Motor Reverse	Output

Table of Port Operation for eRacer_16m

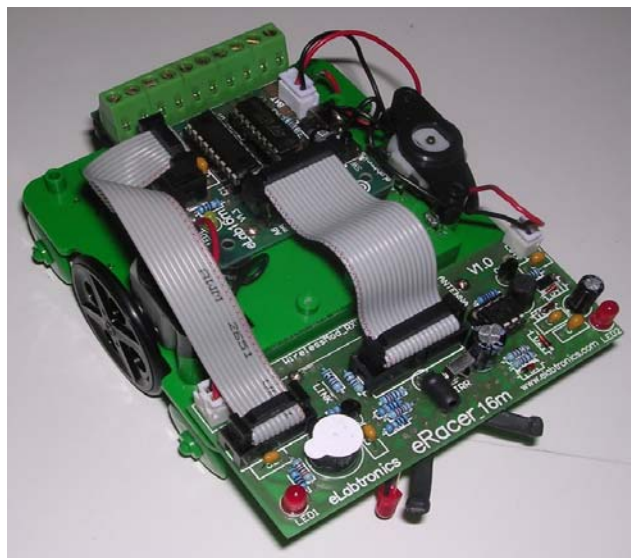




eRacer_16m bare PCB



eRacer_16m assembled PCB



eRacer_16m assembled robot