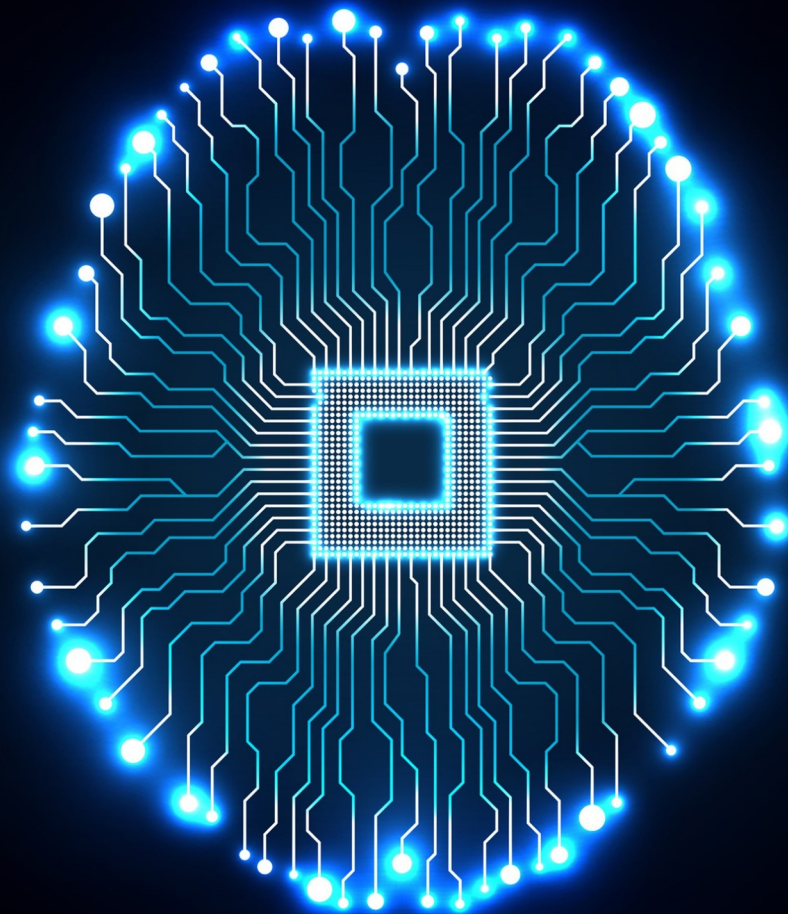


EBLOCKS2 FLOWCODE

Introduction to microcontroller programming

Suitable for BTEC National in Engineering unit 6,
Microcontroller Systems for Engineers



CP4375

MATRIX

www.matrixtsl.com

Copyright © 2018 Matrix Technology Solutions Ltd.

Section: Contents

| | | | |
|-------------------|------------------------------------|---|--------------|
| | Introduction | - | p. 3 |
| Section 1 | - Intro to Microcontrollers | - | p. 12 |
| Section 2 | - Using E-blocks | - | p. 26 |
| Section 3 | - Intro to Flowcode | - | p. 31 |
| Section 4 | - Flowcode - First program | - | p. 42 |
| Section 5 | - Flowcode - Examples | - | p. 50 |
| Section 6 | - Programming Exercises | - | p. 72 |
| Appendix 1 | - Arduino adjustments | - | p. 88 |
| Appendix 2 | - E-blocks 1 adjustments | - | p. 93 |
| | Index | - | p. 98 |

Section: Introduction

The aim of this course is to introduce you to the concepts of developing electronic systems using microcontrollers.

In doing so, it offers substantial coverage of **Unit 6 of the BTEC Level 3 National Extended Diploma in Engineering** (the precise mapping of the course to this unit is given on page 9).

On completing this course you will have learned:

- what a microcontroller is.
- how to construct circuits and systems based on microcontrollers.
- how to program microcontrollers.

Introduction

Before you start

This course is an introduction to microcontroller programming.

To get the full use out of this course we recommend you have the following:

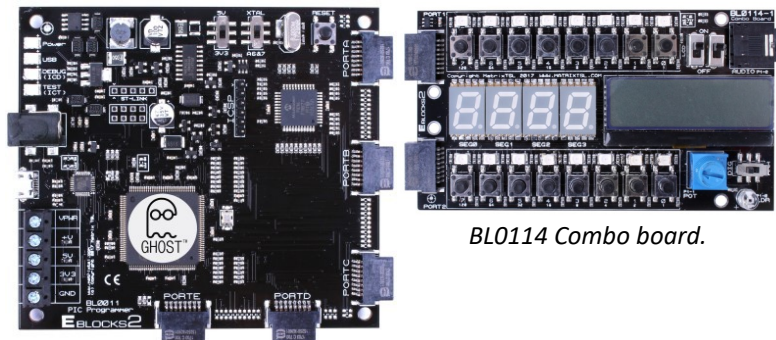
Flowcode

Flowcode is a software program which allows users to quickly and easily develop complex electronic systems in a simple manner, it works with a range of microcontrollers, including Microchip's 'PIC' microcontrollers (PIC MCUs), Arduino, and ARM. Flowcode itself is microcontroller neutral - it presents virtually the same user interface regardless of the microcontroller used. The differences are in the hardware and the way the program is downloaded and tested.

Hardware

It is always more rewarding when learning about microcontroller programming to see the programs execute on actual hardware, therefore we recommend that you have some hardware available to send and execute your created programs onto.

This course is mainly designed around the Matrix E-blocks2 hardware platform, typically the BL0011 programmer and the BL0114 Combo board, although separate E-blocks (LCD, switches, LEDs, etc) can also be used.



BL0011 programmer.

BL0114 Combo board.

While most of the course is designed around the E-blocks2, we also recognise that some people may be using either an Arduino or Eblocks1 devices. So in this course, whenever there is change in the instructions for Eblocks1 or Arduino changes, they will be displayed in the following colours:

Arduino users need an Arduino Uno and E-blocks Arduino Uno Shield (BL0055), as well as the Combo board.

E-blocks1 users will need the EB006v9 Multiprogrammer and the EB083 Combo board plus connecting wires.

Getting more Information

Flowcode

<https://www.flowcode.co.uk>

From here you can access:

- Flowcode—Getting Started Guide
- Flowcode Wiki
- A wide range of Flowcode examples

E-blocks

<https://www.matrixtsl.com/eblocks/resources>

In the E-blocks section you can get the follow resources:

- E-blocks USB Drivers
- E-blocks example files
- E-blocks User Guide

<https://www.matrixtsl.com/eblocks/boards>

From the boards pages:

- Specific datasheets for the boards
- Specific board examples

Other Help

<https://www.flowcode.co.uk/forums/>

The Matrix forum provides an in-depth community of well established, long-term users of Flowcode and new Flowcode users sharing ideas and solving problems and issues encountered whilst using the software.

<https://www.matrixtsl.com/learning/>

The Matrix 'Learning Centre' contains many different resources including articles, drivers, curriculum.

Introduction

Course Conventions

The following abbreviations are used in the course:

| Abbreviation | Meaning |
|--------------|---|
| ADC | Analogue to Digital Converter |
| ALU | Arithmetic Logic Unit |
| ASCII | American Standard Code for Information Interchange |
| CPU | Central Processing Unit |
| EEPROM | Electrically Erasable Programmable Read Only Memory |
| EPROM | Erasable Programmable Read Only Memory |
| GND | ground |
| Hex | hexadecimal |
| IDC | Insulation Displacement Connector |
| I/O | Input / Output |
| ISP | In-System Programming |
| JPEG | Joint Picture Expert Group (standard for images) |
| LCD | Liquid Crystal Display |
| LED | Light Emitting Diode |
| LVP | Low Voltage Programming |
| LDR | Light Dependent Resistor |
| LSB | Least Significant Bit |
| MSB | Most Significant Bit |
| NVRAM | Non-Volatile Random Access Memory |
| PIC | Peripheral Interface Controller |
| PROM | Programmable Read Only Memory |
| PSU | Power Supply Unit |
| RAM | Random Access Memory |
| RV1 | Resistor-Variable 1 |
| SPI | Serial Programmable Interface |
| XTAL | crystal |
| ZIF | Zero Insertion Force |
| +V | positive supply voltage |

The following conventions are used:

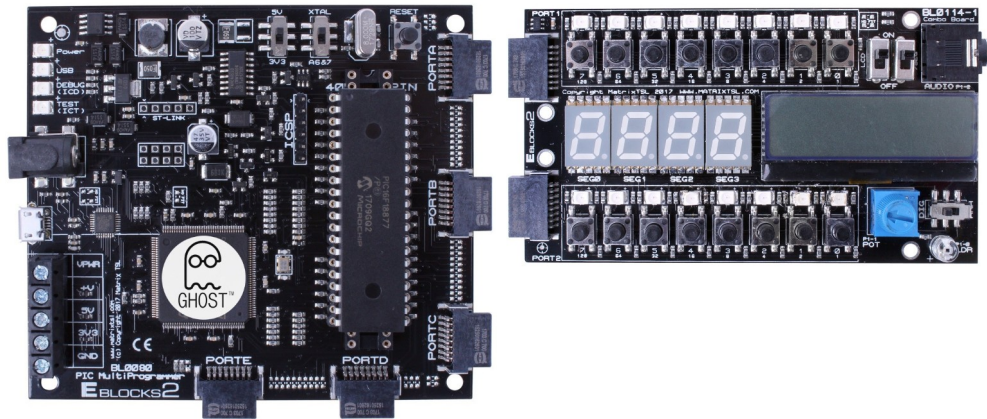
- Matrix products are capitalised on the first word.
For example: *Multiprogrammer* *Prototype board* *Flowcode*
- Flowcode menu instructions are presented as:
File > Open
- Abbreviations are spelled out in full the first time they are mentioned:
EPROM (Erasable Programmable Read Only Memory)
- European circuit symbols are used throughout the course.

Introduction

The hardware:

Most exercises use the BL0011 / BL0080 Multiprogrammer and BL0114 Combo board.

Most of the exercises can also be completed using the Arduino Uno Shield (BL0055). However, these require different PORT settings.



Hardware and software settings used to test most programs:

Hardware jumper settings:

| BL0011 / BL0080 | | Jumper settings |
|---------------------|--------------------------------|-----------------|
| Microchip PIC MCU | 16F18877 | |
| Target voltage | 5V | J15 |
| Voltage source | PSU | J11 |
| Programming Source | USB | J12,13,14: USB |
| Oscillator Selector | OSC | J18,19 |
| Port A E-block | E-blocks Combo board BL0114 | |
| Port B E-block | | |
| Port C E-block | | |
| Port D E-block | | |
| Port E E-block | | |

Flowcode and download settings:

| Menu | Name | Setting | E-blocks 1 |
|--|---------------------------|---------------|---------------|
| Build > Project Options... > Choose a Target | Family - PIC | 16F18877 | 16F1937 |
| Build > Project Options... > General Options | Clock speed (Hz) | 32 000 000 | 19 660 800 |
| | Simulation speed | Normal | Normal |
| Build > Project Options... > Configure | Oscillator | HS Oscillator | HS Oscillator |
| | Watchdog Timer Enable bit | Disabled | Disabled |

After completing this course, you will be able to:

- 01 Send different 8-bit codes to ports of the microcontroller.
- 02 Change the logic level of a one single pin.
- 03 Configure an output icon.
- 04 Use binary code.
- 05 Manipulate logic output levels.
- 06 Use LED's to display an output.
- 07 Compile a program to the PIC MCU.
- 08 Add a delay to slow down execution of a program.
- 09 Change the delay interval.
- 10 Configure a delay icon.
- 11 Control the speed of a microcontroller.
- 12 Use an oscilloscope to time events.
- 13 Use Connection Points to introduce unconditional branching in a program.
- 14 Introduce PWM as a means of controlling the brightness of LEDs.
- 15 Create an infinite loop.
- 16 Manipulate logic output levels.
- 17 Use LEDs to display an output.
- 18 Create and use a variable.
- 19 Configure a calculation icon to perform arithmetic and logic calculations.
- 20 Create and manipulate variables.
- 21 Perform calculations.
- 22 Use LEDs with current limiting resistors.
- 23 Create and use a 'running light' program, using the 'multiply-by-two' method.
- 24 Create and use a 'running light' program, using the 'shift-right' method.
- 25 Create and populate an array.
- 26 Create a conditional loop.
- 27 Input data from switches.
- 28 Use loops to create LED sequences.
- 29 Configure an input icon.
- 30 Configure decision icons and hence add conditional branching to a program.
- 31 Control the frequency at which LEDs flash.
- 32 Use LEDs to display output logic levels.
- 33 Use temporary memory.
- 34 Create, populate and manipulate string variables.
- 35 Control the display of text and numbers on a LCD.
- 36 Use a LCD as an output device for the PIC MCU.
- 37 Configure a Component macro for the LCD.
- 38 Input text and numbers from a keypad and display messages on the LCD.
- 39 Use ASCII code to transmit this data.
- 40 Use multiplexed inputs.
- 41 Configure a Component macro for the keypad.
- 42 Create data loggers, using 8-bit and 10-bit data from the ADC.
- 43 Configure an analogue input.
- 44 Enter data via switches.
- 45 Enter information from light and temperature sensors.
- 46 Configure and use the EEPROM.
- 47 Scroll through EEPROM data.
- 48 display text and numerical data on the LCD.
- 49 Use the E-blocks prototype board.
- 50 Use software macros to simplify the structure of a program.
- 51 Create software macros.
- 52 Use closed loop control.
- 53 Use PWM to control the brightness of LEDs.
- 54 Create and use 'single-pin' interrupts.
- 55 Create and use 'interrupt-on-change' (IOC) interrupts.
- 56 Use real time operation of a PIC MCU.
- 57 Create and use timer interrupts.
- 58 Use the prescaler to create accurate time intervals.
- 59 Trigger the timer using the crystal or an external event.

(Mapping to **Unit 6 of the BTEC Level 3 National Extended Diploma in Engineering** is given on page 9).

| A Investigate typical microcontroller system hardware | | Covered? |
|---|---|----------|
| A1 Control hardware | | |
| I/O capabilities – number, type (analogue/digital), ports | | ✓ |
| hardware specification – bus width, processor speed | | ✓ |
| memory – RAM, ROM | | ✓ |
| hardware features - interrupts, PWM | | ✓ |
| - stack | | ✗ |
| required peripherals | | ✓ |
| cost and accessibility | | ○ |
| ease of use | | ○ |
| software and programming language | | ✓ |
| operating voltages and power requirements | | ✓ |
| A2 Input devices | | |
| User input: | | |
| | digital – switches and buttons | ✓ |
| | analogue – control potentiometer | ✓ |
| Temperature | | |
| | temperature sensors | ✓ |
| | environmental sensor – temperature and humidity | ✓ |
| Light | | |
| | light-dependent resistor (LDR) | ✗ |
| | IR – phototransistor, photodiode or IR receiver | ✓ |
| Movement/orientation | | |
| | tilt switch | ✗ |
| Presence | | |
| | micro-switch | ✗ |
| | ultrasonic | ✓ |
| Input interfacing requirements | | |
| | signal conditioning | ✓ |
| | analogue-to-digital (ADC) conversion | ✓ |
| | modular sensor boards | ✓ |
| | PWM | ✓ |
| | serial communications | ✓ |
| | Inter-Integrated Circuit (I2C) | ✓ |
| A3 Output devices | | |
| Optoelectronic | | |
| | light-emitting diode (LED) – indicator and IR | ✓ |
| | 7-segment display | ✓ |
| | liquid crystal display (LCD) | ✓ |
| Electromechanical | | |
| | relay | ✓ |
| | direct current motor | ✓ |
| | servo | ✓ |
| Audio | | |
| | buzzer or siren | ✗ |
| | speaker or piezo transducer | ✗ |
| Output interfacing requirements | | |
| | power requirements and drivers | ✓ |
| | transistor output stage | ✗ |
| | relay | ✓ |
| | PWM | ✓ |
| | serial communications | ✓ |
| | I2C device interfacing | ✓ |
| A4 Selecting hardware devices and system design | | |
| A5 Assembling and operating a microcontroller system | | |

| | | |
|--|---|---|
| B Programming Techniques and Coding | | |
| B1 Programming techniques | | |
| Use of a programming development environment | | |
| | software operation | ✓ |
| | connecting to microcontroller hardware | ✓ |
| | creating and managing program files | ✓ |
| | syntax/error checking | ✓ |
| | simulation | ✓ |
| | compiling, downloading and live testing | ✓ |
| | monitoring/debugging | ◦ |
| | safe use of computer and display | ✗ |
| Coding practices: | | |
| | device set-up and program initiation | |
| | introductory comments | ✗ |
| | chip set-up | ✓ |
| | pin modes | ✓ |
| | Libraries | ✓ |
| | Declarations | ✗ |
| | efficient/effective code authoring | ✗ |
| | code syntax | ✗ |
| | in-line commenting | ✓ |
| | code organisation and structure | ✓ |
| B2 Coding constructs | | |
| Input/output | | |
| | digital – bit and port level read/write | ✓ |
| | analogue read/write, resolution, calibration | ✓ |
| | tone and sound generation | ✗ |
| | pulse and PWM | ✓ |
| | communication, including serial and I2C | ✓ |
| Program flow and control: | | |
| | calling libraries | ✓ |
| | subroutines and functions | ✓ |
| | control structure sequence iteration – if, else, switch, case, for, do, while, until and end | ✓ |
| | delays and timing | ✓ |
| | Interrupts | ✓ |
| Logic and arithmetic | | |
| | variables – data types (Boolean, character, byte, integer, word, float, long, double, string) | ✓ |
| | arrays | ✓ |
| | comparative operators: =, not =, <, >, < or =, > or =. | ◦ |
| | Boolean operators – AND, OR, NOT | ✓ |
| | logic using input condition – digital and analogue | ✓ |
| | arithmetic operations | ✓ |
| B3 Structured program design | | |
| | pseudo code | ✓ |
| | flowchart | ✓ |
| | decision table | ✗ |
| B3 Number systems | | |
| | bits, bytes | ✓ |
| | parallel and serial | ◦ |
| | binary to decimal conversion | ✓ |

| | |
|--|----------------|
| C System development cycle | Project |
| | |
| C1 Development processes | Project |
| Stages of the development process. | |
| | |
| C2 Documentation | Project |
| A portfolio of evidence produced throughout the development process. | |
| | |

Key

| Symbol | Meaning |
|---------|--|
| ✓ | Content covered by the course |
| • | Content partly covered by the course |
| ✗ | Content not covered by the course |
| Project | Content addressed through project work |

Section 1: Introduction to Microcontrollers

Microcontrollers are tiny devices used to control other electronic devices. They are found in a huge range of products. In automotive systems they can be found in engines, anti-lock brakes and climate control systems. In domestic electronics they can be found in TVs, VCRs, digital cameras, mobile phones, printers, microwave ovens, dishwashers and washing machines.

A **microcontroller** is a digital integrated circuit, consisting of a central processing unit, a memory, input ports and output ports.

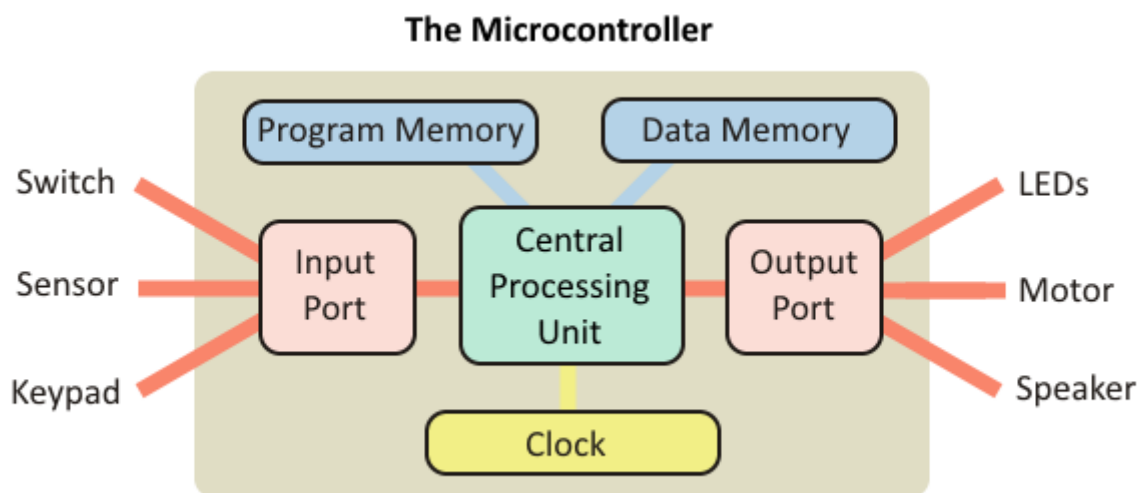
Microcontrollers

At their heart (or is it brain?) there is a **Central Processing Unit (CPU)**. This processes the digital signals, does calculations and logic operations, creates time delays, and sets up sequences of signals.

How does it know what to do? It is following a program of instructions, stored in Section of the memory, called the **program memory**, inside the microcontroller.

From time to time, the CPU needs to store data, and then later retrieve it. It uses a different area of memory, called the **data memory** to do this.

The clock synchronises the activities of the CPU. It sends a stream of voltage pulses into the CPU that controls when data is moved around the system and when the instructions in the program are carried out. The faster the clock, the quicker the microcontroller runs through the program. Typically, the clock will run at a frequency of 20MHz (twenty million voltage pulses every second).

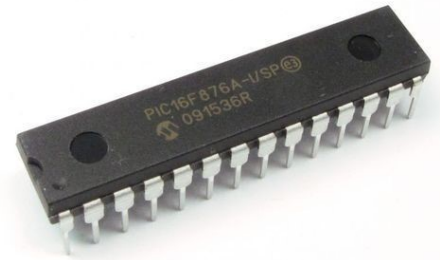


To talk to the outside world, the microcontroller has **ports** that input or output data in the form of binary numbers. Each port has a number of connections (often referred to as **bits**). An 8-bit port handles an 8-bit (or one byte) number.

Information from sensors is fed into the system through the input port(s). The microcontroller processes this data and uses it to control devices that are connected to the output port(s). The ports themselves are complex electronic circuits, not simply a bunch of terminals to hang components on.

Microcontrollers—PIC and AVR

The name PIC refers to a popular group of microcontrollers, produced by Arizona Microchip. AVR is another group of microcontrollers, also made by Microchip. The microcontroller on an Arduino board is actually an AVR chip.



When we use a microcontroller, we have to specify how we want the ports to behave. The ports are bi-directional, meaning that they can act as either input ports or output ports. When we write a program for a microcontroller, we start by configuring the ports, telling them whether they are to behave as input ports or output ports.

The input port can receive data (information) in one of two forms, as an analogue signal, or as a digital signal. It is important that we understand clearly the difference between these.

The Digital World

Much of our everyday information is described in numerical format.

For example:

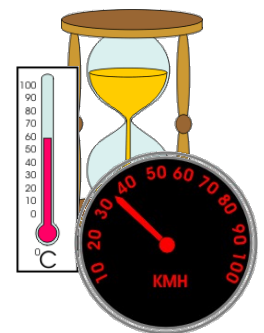
- "It is 2 o'clock."
- "The temperature outside is 21 degrees C."
- "The car was travelling at 48 kilometres per hour."

It is easy to understand data in this form.

For example, the table shows how the speed of a car changes over a period of time.

| Time in seconds | Speed in kilometres per hour |
|-----------------|------------------------------|
| 0 | 0 |
| 10 | 15 |
| 20 | 21 |
| 30 | 25 |
| 40 | 22 |
| 50 | 20 |
| 60 | 16 |

However, you might wonder what happened at time 35 seconds. Was the car moving faster or slower than 25 km/h at that moment?



The Analogue World

Now the information is given in the form of an analogy.

In other words, we use something that behaves in a similar way.

For example:

1. The hour glass egg timer:
The greater the time elapsed, the deeper the sand in the bottom of the egg timer.
2. The mercury-in-glass thermometer
The hotter it gets, the further the mercury moves up the tube.
3. The car speedometer
The higher the speed, the further the pointer moves around the dial.

The problem with analogue data is that you have to do some work to extract it.

For the speedometer, and thermometer, you have to work out where the pointer sits on the scale. On the other hand, it is easy to judge how the temperature of a body or speed of a car is changing. We see the mercury moving along the tube or the pointer moving around the dial.

Analogue Data

Many electronic sensors provide signals in analogue form. For example, a microphone provides an electrical 'copy' of a sound wave.

Another - the temperature sensor.

Here is the circuit diagram for one type of temperature sensor.

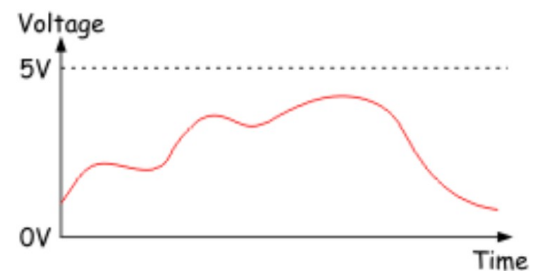
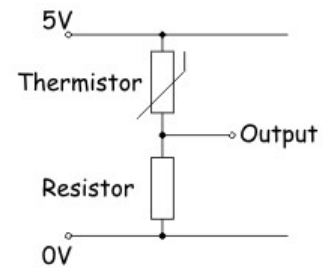
The output voltage increases when the temperature increases.

It is an analogue signal because the voltage copies the behaviour of the temperature.

An electrical analogue signal can have any voltage value, limited only by the power supply used.

In this case, the output of the temperature sensor could, in theory, go as high as 5V, or as low as 0V.

Over a period of time, the output voltage could change as shown in the diagram. This is an analogue signal.



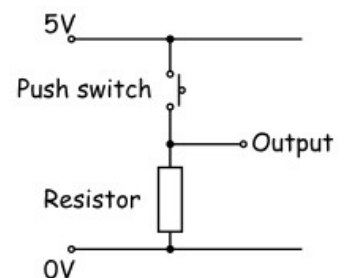
Digital Data

A digital signal carries its information in the form of a number. Electronic systems usually employ the binary number system, which uses only the numbers '0' and '1', coded as voltages. We could decide on the following code: '0' = 0V, '1' = 5V, for example.

Digital signals, then, have only two possible voltage values, usually the power supply voltage, or as close to it as the system can get, and 0V.

How can we enter these numbers into an electronic system?

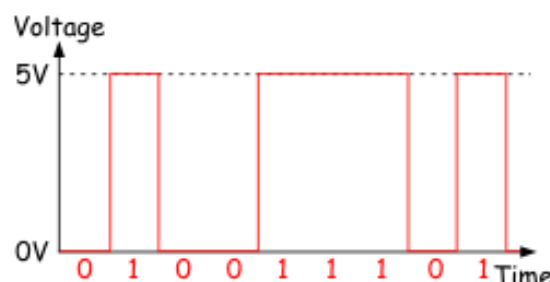
One (very slow) way would be to use a switch (an example of a digital sensor). The circuit diagram shows such a digital sensor.



- When the switch is open (not pressed), the output is 'pulled down' to 0V by the resistor. This output could represent the binary number '0'.
- With the switch closed (pressed), the output is connected to the positive supply, 5V in this case. This could represent the binary number 1.

(Note - if the positions of the switch and resistor were reversed, pressing the switch would put a logic 0 signal on the pin etc.)

The following diagram shows a more complex digital signal.



The nine bit binary number represented by the signal is given under the waveform.

Analogue to Digital Conversion

Much of our 'real world' data is **analogue**, but computers (including microcontrollers), only process **digital** data. Fortunately microcontrollers often contain a subsystem that can convert information from analogue format to digital format. This is called an **Analogue-to-Digital Converter** (usually shortened to '**ADC**' or '**A/D**').

The ADC inside a microcontroller divides the range of possible analogue voltages into equal steps. The lowest step is given the number '0', and the highest step is given the highest number that the A/D converter can handle.

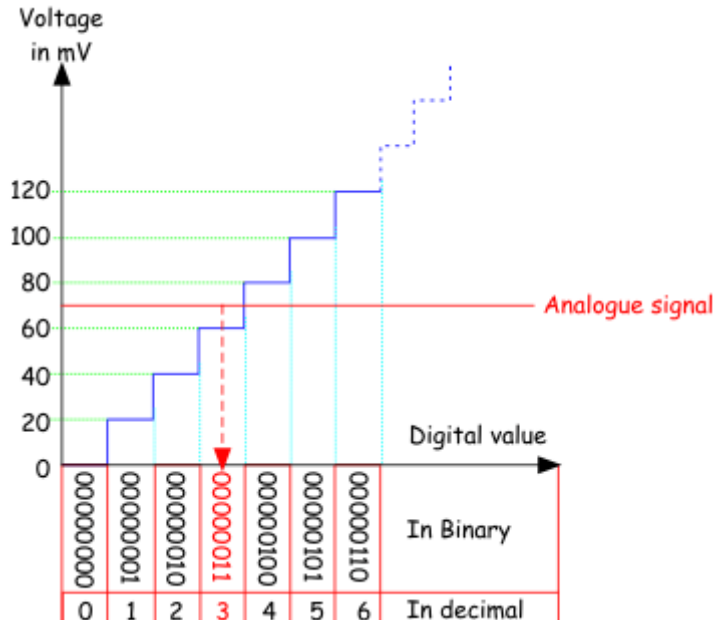
This highest number is determined by the resolution of the ADC, which, in turn, depends on number of **bits** the internal circuitry of the ADC can handle. Typical resolution of microcontroller ADCs is 8, 10 or 12 bit.

For example, if the biggest analogue voltage is 5V, and a microcontroller has an 8-bit ADC:

- the highest 8-bit number is 1111 1111 (= 255 in decimal).
- the first step is 0000 0000 (= 0 in decimal).
- meaning that there are 256 voltage levels.
- so stepping from one level to the next involves a voltage jump of $5V/256$, or about 20mV.

When this microcontroller processes an **analogue signal**, it first divides it by 20mV, to find out how many steps the signal includes. This gives the **digital** equivalent of the **analogue** signal.

The next graph illustrates this process.

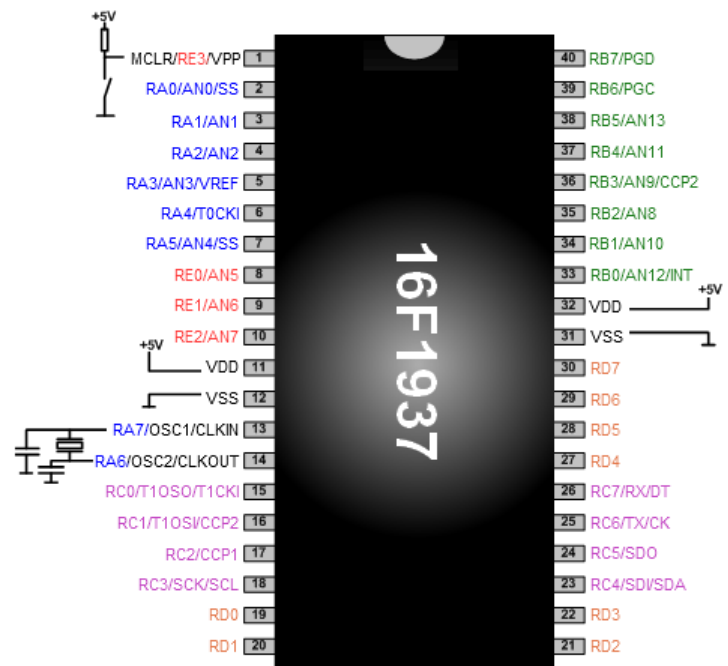


In our example, the converter outputs 0000 0000 for any analogue signal up to 20mV, outputs 0000 0001 for analogue signals between 20 and 40mV, and so on. The analogue signal shown in the graph produces an output of 0000 0011.

Inputting data into a Microcontroller

A microcontroller is a digital device, but data can be entered in both analogue and digital forms. Programmers choose whether pins on the microcontroller are used as analogue inputs, digital inputs or digital outputs. This flexibility leads to complex labelling.

The diagram shows the pinout for a typical 40pin PIC microcontroller, in this case it is the 16F1937 chip. It has five ports, known as A, B, C, D and E. The pins on port A are labelled RA0 to RA7; pins on port B are labelled RB0 to RB7 etc. Ports A, B, C and D have eight pins but port E has only four. For example, up to eight digital sensors can be connected to port A of the 16F1937. Pin 2 is marked as 'RA0/AN0', meaning that it can be used as bit 0 of port A (Register A bit 0) or as **AN**alogue input 0.



The function of each input / output pin is determined by setting the contents of internal registers, called 'data-direction' registers inside the microcontroller.

Pins RA6 and RA7 are also labelled as 'OSC1' and 'OSC2'. They can be connected to an external oscillator circuit or be used for digital input /output.

Analogue sensors must be attached to the pins labelled with an 'ANx' (ANalogue) label. These, found on ports A, B and E, can handle analogue signals between V_{DD} (5V) and V_{SS} (Gnd).

Most pins have alternative functions. For example pin 25 is labelled as 'RC6/TX/CK', meaning that it can be Register C bit 6, or the transmit (TX) pin of the internal serial interface, or the **ClocK** pin of the internal serial interface.

Fortunately Flowcode takes care of the internal settings that dictate pin functionality for you.

Outputting data

The microcontroller is a digital device - we have said that several times already. It outputs a digital signal. In most cases, we use this to turn something on and off - '0' = 'off' and '1' = 'On'.

For example:

Suppose that we set up port B as the output port (or let Flowcode do it for us). There are eight pins on port B, so we can switch eight devices on and off. It is important to plan how we connect these devices, as otherwise they might work the opposite way round.

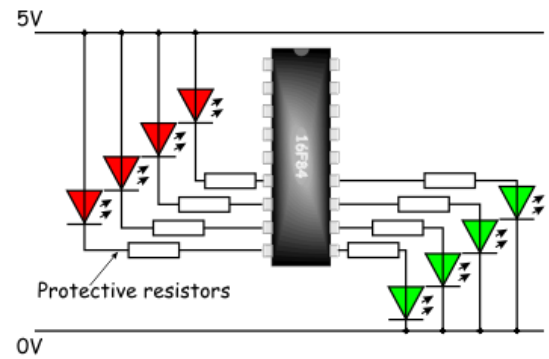
The diagram shows eight LEDs connected to port B of a PIC16F84 microcontroller:

- The four red LEDs are connected between the positive supply rail and the port B pins. For these LEDs, the microcontroller is **sinking** current.
- The four green LEDs are connected between the pins and the 0V rail. For these, the microcontroller is **sourcing** current.

Each red LED lights up when its pin is at a low voltage, outputting '0' in other words.

Each green LED lights when its pin is at a high voltage, outputting a '1'.

(There are limits as to how much current the ports can control. Typically, one output pin can manage up to 25mA. This is enough to drive LEDs and buzzers directly, but higher-powered devices will need additional circuitry to interface with the microcontroller -dealt with later. However, the maximum current for the whole port is around 100mA, so not all pins can output 25mA at the same time).



Current Limits

As you have seen, Flowcode has a simulation mode that allows you to attach LEDs to show the status of the pins on the microcontroller when they are used as outputs. The LED simulation function inside Flowcode can be set so current is sourced or sunk (look for "polarity" properties of components which can be "active high" or "active low"), but typically this workbook assumes that current is **sourced** from the PIC MCU - like the green LEDs in the diagram above.

At some stage, you will need to use the microcontroller pin specifications in order to use them as digital inputs, analogue inputs, or as digital outputs. In particular, there are limitations on the output capabilities of the device. Exceeding these limits, even for a short time, may cause permanent damage to the microcontroller.

Fortunately the E-block boards used on this course all have current limiting resistors which protect the microcontrollers. When using the prototype or patch boards, however, there is no such protection and care must be taken not to damage your device.

| | |
|---|-------|
| Maximum current sunk/sourced by any I/O pin | 25mA |
| Maximum current sunk by all ports | 200mA |
| Maximum current sourced by all ports | 140mA |
| Maximum current out of VSS (Gnd) pin | 95mA |
| Maximum current into V _{DD} (5V) pin | 70mA |

Storing Data

Electronic sub-systems that store data are known as 'memory'. They can store only digital data.

One item of data is stored in one location in the memory. This data could be the correct combination to disarm a burglar alarm, or the target temperature of a car engine block.

Each memory location has a unique address, a number used to identify the particular location. This means that we can draw up a map of the memory, showing what data is held in each location (the decimal version of the address is included to make the table easier to read).

| Address | | Data stored |
|------------|-----------|-------------|
| In decimal | In binary | |
| 0 | 000 | 11101001 |
| 1 | 001 | 00100101 |
| 2 | 010 | 10000101 |
| 3 | 011 | 11001101 |
| 4 | 100 | 01110100 |
| 5 | 101 | 00011011 |
| 6 | 110 | 11110011 |
| 7 | 111 | 10000101 |

Electronic systems only understand binary numbers. This very small memory has eight locations (notice that numbering normally starts at 0). It needs a 3-bit binary number to create unique addresses for each location. We can store items of data that are eight bits long (one **byte** - 1B). Our example memory could be called a 8 x 1B memory. Memory systems used in computers are much larger. Data is often stored as 32 bit numbers, allowing the use of much larger numbers. There are many more locations, too. A typical computer memory now has millions of memory locations.

Types of Memory

There are several types of electronic memory, each with a slightly different job to do. We can divide them into two main groups, **ROM** and **RAM**.

Read Only Memory (ROM)

These devices are normally only read (i.e. the contents are accessed but not changed / written,) during the running of a program.

- The contents are not volatile (the data remains stored even when the power supply is switched off).
- They are often used to store the basic programs, known as **operating systems**, needed by computers.
- The group includes:
 - **PROM (Programmable Read Only Memory)**
 - **EPROM (Erasable Programmable Read Only Memory)**
 - **EEPROM (Electrically Erasable Programmable Read Only Memory)**

A **PROM** is a one-shot device, which arrives blank, ready to receive data. Data can then be 'burned' into it, but only once. After that it behaves like a **ROM** chip that can be read many times but not altered.

With an **EPROM**, shining ultraviolet light through a window in the top of the chip erases the contents. New data can then be 'burned' into the memory. Some older microcontrollers operate in this way.

The **EEPROM** devices work in a similar way to an **EPROM**, except that the contents are erased by sending in a special sequence of electrical signals to selected pins. **Flash memory** is a form of **EEPROM**, widely used as the storage medium in digital cameras (the memory stick) and in home video games consoles.

Random Access Memory (RAM)

- RAM allows both read and write operations during the running of a program.
- The contents are volatile and disappear as soon as the power supply is removed.
(The exception is **NVRAM, Non-Volatile RAM**, where the memory device may include a battery to retain the contents, or may include an **EEPROM** chip as Section of the memory to store the contents during power loss).
- They are often used for the temporary storage of data or application programs.

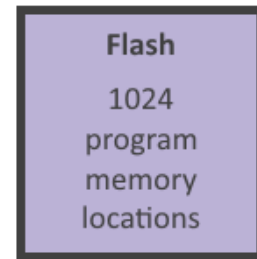
Microcontroller Memory

Microcontrollers have three separate areas of memory:

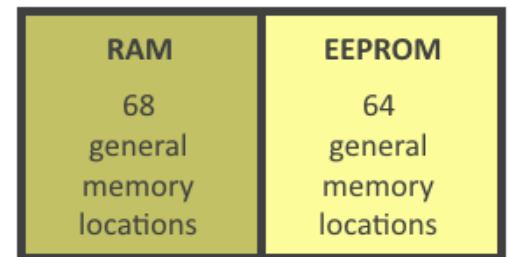
- program memory (**Flash**)
- user variable memory (**RAM**)
- **EEPROM**

The names give strong hints as to the purpose of the areas.

For the eighteen pin PIC16F84 the graphic illustrates the organisation of the memory.



Program Memory



Data Memory

Program memory is used to store the program.

In most microcontrollers, such as the 16F1937, this uses **Flash** technology, meaning that it can be programmed and cleared many times. Older PIC MCUs use PROM for the program memory so that many of these can be programmed only once.

Data memory is used to store data.

Section of this uses RAM and Section uses EEPROM.

The EEPROM allows us to preserve important data even if the power supply to the system is switched off.

For example, suppose that the microcontroller is used as a temperature controller that keeps an incubator at a set temperature. It might make sense to store the target temperature value in EEPROM so that we do not have to enter it into the system every time we switch the incubator on.

Programming

Microcontrollers are programmable devices. They do exactly what they are told to do by the program, and nothing else. A program is a list of instructions, along with any data needed to carry them out.

The only thing microprocessors understand is numbers. There's a problem because we don't speak in numbers, and they don't understand English.

There are two solutions, and both need some form of translator:

- Write the program in English, or something close to it, and then have the result translated into numbers.
- We can think through the program design in English and then translate it ourselves into a language that is similar to numbers, known as **assembler**. From there, it is a swift and simple step to convert into the numerical code that the microcontroller understands.

These two extremes are known as programming in a **high-level language** (something close to English) or in a **low-level language** (assembler).

The first is usually quicker and easier for the programmer, but takes longer to run the program, because of the need to translate it for the microcontroller.

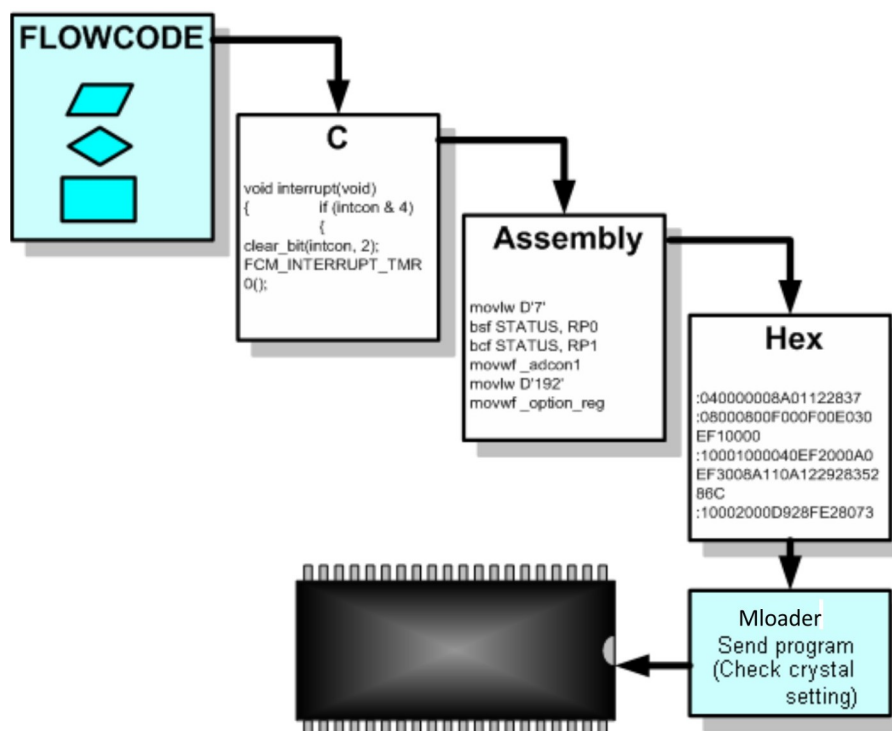
The second is much slower for the programmer, but ends up running very quickly on the microcontroller.

The Flowcode Process

Flowcode offers an easy way to program microcontrollers, as you will see. Once the flowchart is designed on-screen, one press of a button causes the software to translate it into numerical code. Flowcode passes the program through a number of processes before it gets sent into the microcontroller. The flowchart is processed:

- first into **C code**,
- then into **Assembler**,
- and finally into **hexadecimal numbers** or 'Hex', which the microcontroller 'understands'.

The Hex code is then sent into the microcontroller, using a subsidiary program called '**Mloader**'. When you select **Build > Project Options... Configure** from the Flowcode menu, you can control a number of options and configurations by setting the value of registers inside the device when you download a program.



The Hex code is 'burned' into the microcontroller program memory. Since Flash memory is used to form the program memory, the program is not lost when the microcontroller is removed from the programmer. This allows you to use it in a circuit. Equally, use of Flash memory means that you can reuse the microcontroller and overwrite the program memory with a new program.

Running the Program

As soon as the microcontroller is powered up and is supplied with clock pulses, it will start to run whatever program is stored in program memory (Flash).

When you press the reset button on the microcontroller programming board, the program restarts from the beginning.

During programming the microcontroller stops while the program is being loaded. When that is completed, it then restarts and runs the downloaded program.

Different types of Microcontroller

There are a large number of microcontroller devices available, from the humble 16F84 to larger more complex microcontrollers such as the 40 pin 16F1937. Different microcontrollers have different numbers of ports, or I/O pins, analogue inputs, larger memory, or advanced serial communications capabilities such as RS232 or SPI bus.

PIC16F18877 Architecture

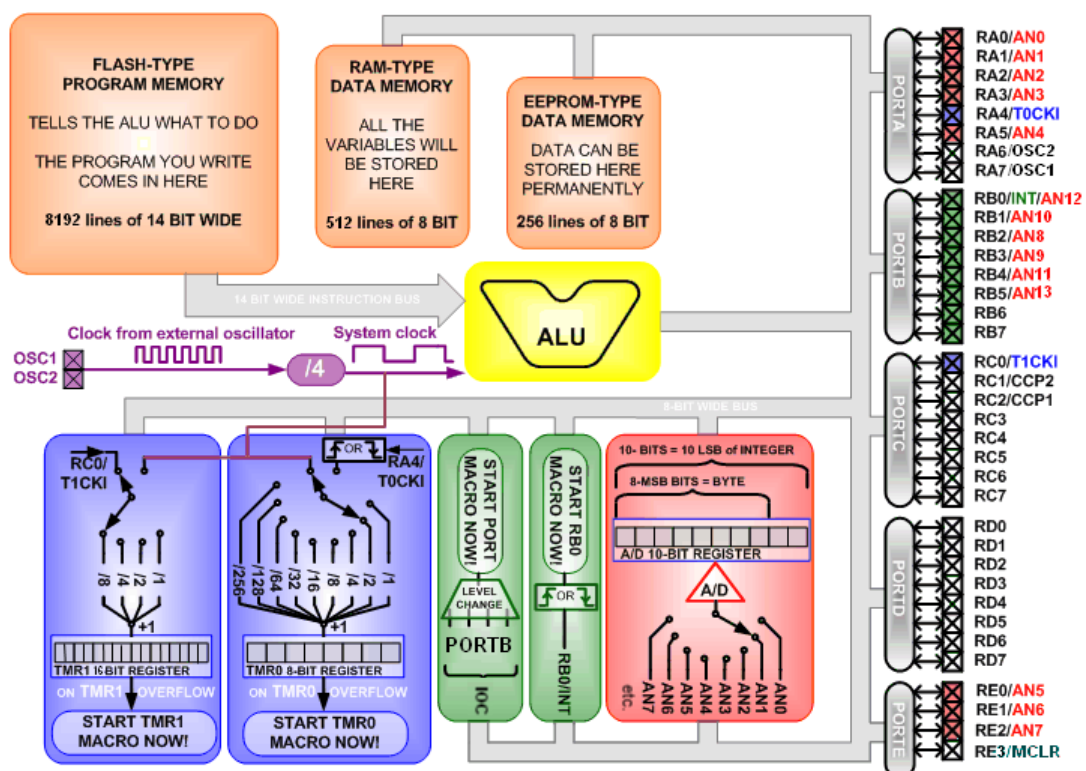
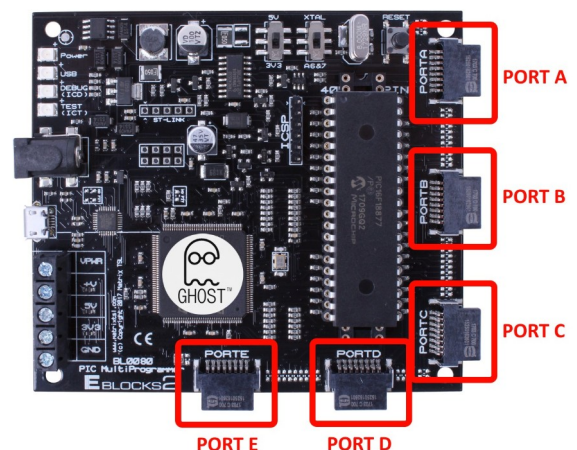
As an example we will use the **16F18877 microcontroller**. It is important that you understand a little more about what it does and how to use it. This section details the pins that are available on the **16F18877** and the connectors they use on the E-blocks programmer board.

At this point in a traditional programming course, you would be introduced in some detail to the various internal circuit blocks of the microcontroller. You would need this information to write code for the microcontroller in C or assembly code.

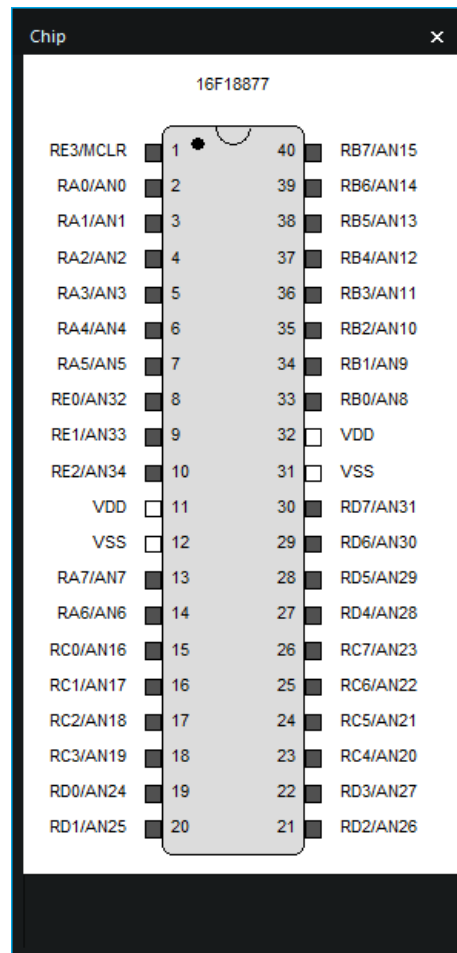
No need - Flowcode takes care of these details.

However, you do need to understand the input and output connections of the microcontroller, the memory available and the role of the other subsystems in the microcontroller.

Ports - The **16F18877** has five ports, labelled 'A' to 'E', connected to the rest of the microcontroller internals by an 8-bit bus system.



16F18877 Microcontroller Layout:



Memory

Flash

- Flash memory is used to store the program you write.
- This program is 'compiled' by the computer to binary code and then downloaded into the Flash memory of the microcontroller.
- You can read from, and write to it and it is retained, even after a power cut.
- The Flash memory contained in the **16F18877** can store up to 32768 program commands.

RAM

- Data from inputs, outputs, analogue inputs, calculations etc. is typically stored in 'variables' (values in the program that alter as it runs). RAM is where these are stored.
- This memory is erased every time the power gets cut or a reset occurs.
- It also contains system 'registers' which control and report the status of the device.
- The RAM memory in the **16F18877** can store up to 4096 bytes of data.

EEPROM

- EEPROM is where data can be permanently stored
- This memory is of the PROM-type - preserved every time the power cuts or a reset occurs.
- The EEPROM of the **16F18877** can store up to 256 bytes of data.

ALU

- The ALU (Arithmetic Logic Unit) is at the heart of the microcontroller's data processing.
- All data passes through this unit.
- The program in the Flash memory tells the ALU what to do.
- The ALU can send data to, and fetch data from all the separate blocks and ports in the microcontroller using the 8-bit wide data-bus.
- The ALU needs four external oscillator clock pulses to execute one whole instruction.
- How the ALU works is very complicated. Fortunately Flowcode programmers do not need to know how it works.

Timer 1 (TMR1)

- This timer interrupt is used to provide the microcontroller with exact timing information.
- It is 'clocked' either by the system clock or by an external clock on pin RCO.
- Either clock can be divided by 1, 2, 4 or 8 by configuring the Prescaler of TMR1 in Flowcode. The resulting output triggers TMR1 and increments the TMR1 register.
- TMR1 is a 16-bit register, which 'overflows' when it reaches '65536'.
- At the instant it overflows, it generates an interrupt and the TMR1 register is reset to '0'.
- This TMR1 Interrupt stops the main program immediately and makes it jump to the TMR1 macro.
- After this finishes, the main program continues from where it left off just before the interrupt.

For example:

| | |
|--|---------------|
| External clock oscillator frequency (crystal oscillator) | 19 660 800 Hz |
| System Clock (four clock pulses per instruction) | 4 915 200 Hz |
| Set prescaler to '8' (divides by 8) | 614 400 Hz |
| Overflow frequency when TMR1 = '65536' | 9.375 Hz |

Result: TMR1 interrupts the main program and execute the TMR1 macro 9.375 times per second.

Timer 0 (TMR0)

- This timer interrupt also provides the microcontroller with exact timing information.
- It is 'clocked' either by the system clock or by an external clock on pin RA4.
- This system clock runs exactly four times slower than the external oscillator clock.
- Either clock can be divided by 1, 2, 4 or 8, 16, 32, 64, 128, or 256 by configuring the Prescaler of TMR0 in Flowcode. The result triggers TMR0 and increment the TMR0 register.
- This TMR0 register is an 8-bit register, which overflows when it reaches 256.
- At the instant it overflows, it generates an interrupt and the TMR0 register is reset to 0.
- A TMR0 Interrupt stops the main program immediately and makes it jump to the TMR0 macro.
- After this finishes, the main program continues from where it left off just before the interrupt.

For example:

| | |
|--|---------------|
| External clock oscillator frequency (crystal oscillator) | 19 660 800 Hz |
| System Clock (4 clock pulses per instruction) | 4 915 200 Hz |
| Set prescaler to 256 (divides by 256) | 19 200 Hz |
| Overflow when TMR0 = 256 | 75 Hz |

Result: TMR0 interrupts the main program and execute the TMR0 macro 75 times per second.

RBO External Interrupt

- A logic level change on pin RB0 can be configured to generate an interrupt.
- It can be configured in Flowcode to react to a rising or to a falling edge on RB0.
- If set to react to a rising edge, when one occurs:
 - it immediately stops the main program;
 - the RB0 related macro is executed;
 - then the main program continues from where it left off just before the interrupt.

This happens every time a rising edge is detected at pin RB0.

PORT B External Interrupt

- A logic level change on any combination of pins on port B can generate an interrupt.
- This can be configured to occur on a rising or a falling edge, or both.
- When one of these interrupts occurs:
 - it immediately stops the main program;
 - the port B related macro is executed;
 - then the main program continues from where it left off just before the interrupt.

This happens every time a level change is detected on one of the pins selected on port B.

A/D

- The **16F18877** has 35 pins.
- It has only one 10-bit A/D converter.
- This implies that these fourteen analogue inputs can't all be read at the same time.
- A built-in analogue switch, configured in Flowcode, selects which inputs are sampled.
- After the **sample** instruction, the analogue switch points to the correct input and this is converted into a 10-bit binary value.
- In Flowcode, you can opt to use only the eight most-significant bits (MSB's) of this 10-bit value, by using the **GetByte** instruction, or to use the full ten bits by using the **GetInt** instruction. The ten bits will fill up the ten least-significant bits (LSB's) of the selected 16-bit integer variable.
- After this, the program can select to read another analogue input.

Busses

- A PIC and AVR microcontrollers are based on a Harvard architecture.
- This means that there are separate busses for instructions and for data.
- The data bus is 8-bits wide and connects every block and port together.
- The instruction bus is 14-bits wide and transports instructions, which are 14-bits long, from the program memory to the ALU.

Introduction to 'clocks'

Every microcontroller needs a clock signal to operate. Internally, the clock signal controls the speed of operation and synchronises the operation of the various internal hardware blocks.

In general, microcontrollers can be 'clocked' in several ways, using:

- an external crystal oscillator.
- 'RC' mode, where the clock frequency depends on an external resistor and capacitor.
- an internal oscillator.

The 'RC' mode exists partly historical and partly for reasons of economics. It was introduced as a low cost alternative to a crystal oscillator. It is fine for applications that are not timing critical, but is not covered in this course.

Section 2: Using E-blocks

E-blocks are small circuit boards that can easily connect together to form an electronic system. There are two kinds of E-Blocks. **Upstream** boards and **Downstream** boards.

A variety of boards can be combined to create a full system with downstream boards connected to upstream boards.

E-blocks are ideal companions to Flowcode software, allowing users to test and develop their Flowcode programs. Programs can be compiled directly to the boards, providing ideal development environments.

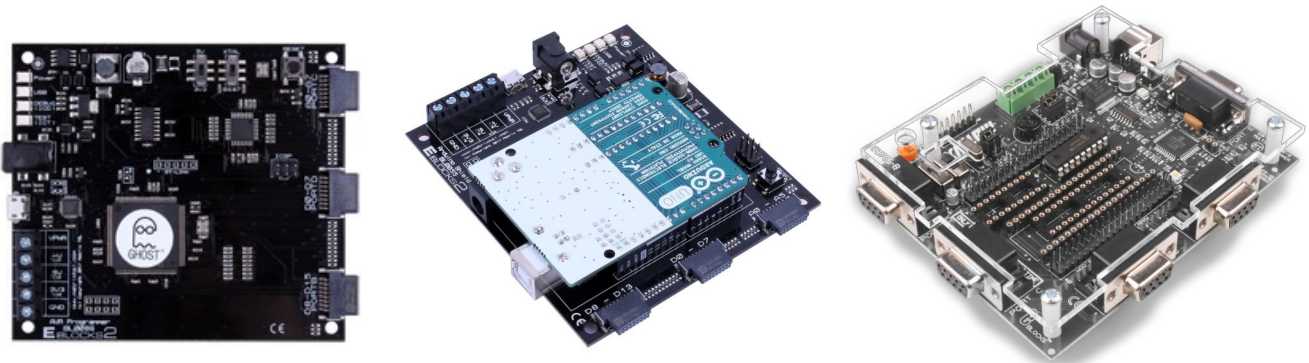
E-blocks consist of upstream boards and downstream boards.

Upstream boards

'Upstream' is a computing term indicating a board that controls the flow of information in a system. They are usually programmed in some way.

Any device which contains 'intelligence' and can dictate the direction of flow of information on the bus can be thought of as an 'upstream' device.

Examples include microcontroller boards, and Programmable Logic Device boards.

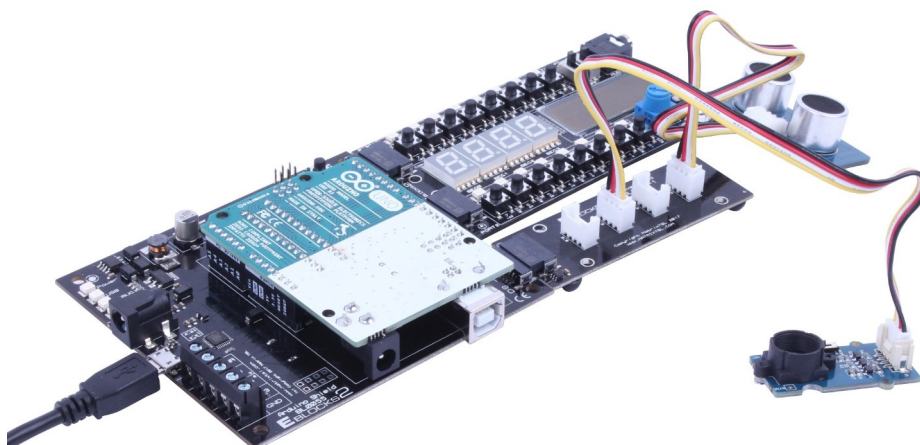


Downstream boards

'Downstream' boards are controlled by an 'upstream' board, but information can flow into or out of them. Examples include LED boards, LCD boards, RS232 boards etc.

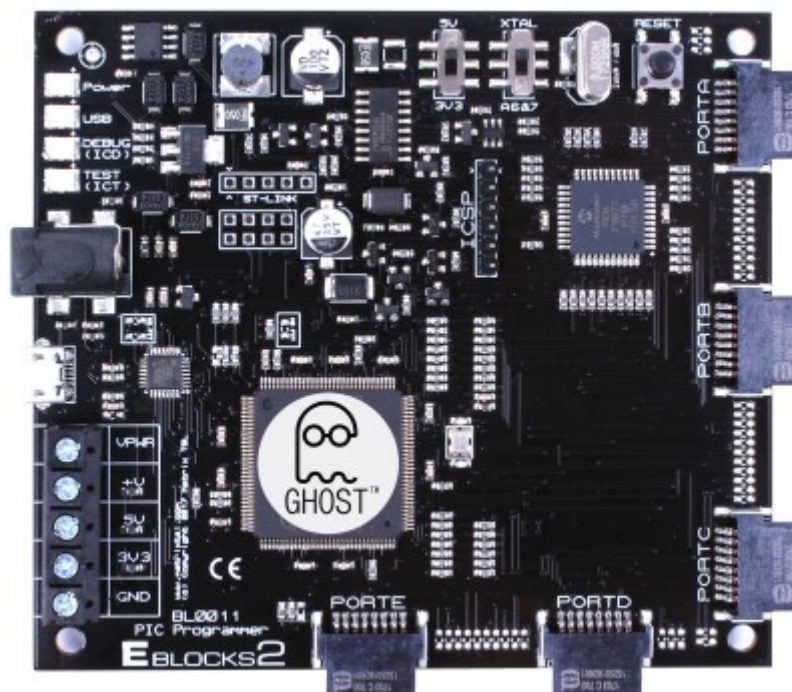


Upstream and downstream boards combined to form a full system, with the downstream boards plugging into the upstream 'intelligent' boards:



BL0011 PIC Programmer

- The board has five ports, labelled A to E.
- Ports 'B', 'C' and 'D' offer full 8-bit functionality.
- Port 'A' has 6-bit functionality (8-bit if the internal oscillator is selected).
- Port 'E' has 3-bit functionality.
- It can be powered from an external power supply, delivering 7.5V to 9V or from a USB supply.
- If the Reset switch is pressed, the program stored in the microcontroller will restart.
- The board is USB programmable via a programming chip. This takes care of communication between Flowcode and the microcontroller.
- The microcontroller executes one instruction for every four clock pulses it receives.
- (Note - a single instruction is NOT the same as a single Flowcode symbol, which is compiled into C and then into Assembly and probably results in a number of instructions).
- This course uses an 8MHz crystal which is multiplied up to 32Mhz internally.
- Switches allow the user to select a number of options.
- External power supply or USB power supply.
- Where the microcontroller uses an internal oscillator, all eight bits of port A can be used for I/O operation.
- Use of a PICKit3 tool from Microchip via ICSP header.
- Comes with a surface mounted PIC16F18877 device.
- Provides power to the downstream E-blocks boards via the port connectors.
- Contains the Matrix Ghost chip which allows for real time in-circuit debugging when combined with Flowcode.



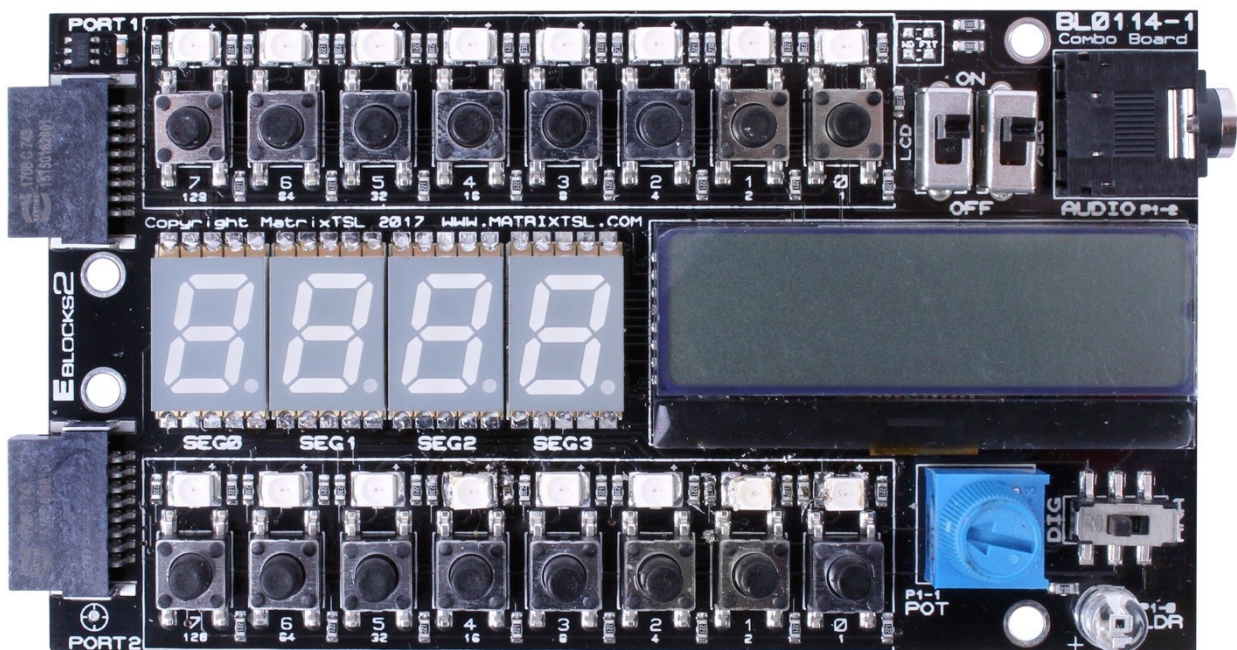
For Arduino programmer overview please refer to Appendix 1, SECTION A (page 89).

For E-blocks1 programmer overview please refer to Appendix 2, SECTION A (page 94).

BL0114 Combo Board

The board combines together on one compact board the functionality found on a number of individual E-blocks boards:

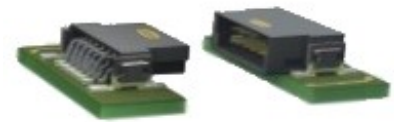
- BL0167 LED board (x2)
 - BL0169 LCD board
 - BL0145 Switch board (x2)
 - For this course, the port connectors attach to female connectors on ports A and B of the upstream board.
 - The board provides a set of eight switches and eight LEDs for port A and the same for port B.
 - With the main switch in the DIG position, port A is routed to its push switches (SA0 to SA7), to LEDs (LA0 to LA7) and to the quad 7-segment display.
 - With the main switch in the ANA position, port A is switched to the analogue sensor section of the board, so that pin RA0 is connected to the on-board light sensor and pin RA1 is connected to the potentiometer to give a variable output voltage, (simulating the action of an analogue sensing subsystem).
- Note: With the switch in the ANA position, the on-board switches and LEDs LA0 and LA1 will not operate.**
- Port B I/O pins are routed to its push switches (SB0 to SB7), to the LEDs (LB0 to LB7), to the quad 7-segment displays and to the LCD display.
 - The quad 7-segment display is turned on by switch '7SEG'. It is connected to both port A and B.
 - Port B is used to control the LED segments and the decimal point).
 - Port A, bits 0 to 3, select which display is activated.
 - The LCD is a 20 character x 4 lines module, turned on by switch 'LCD'. Normally a complex device to program, Flowcode takes care of the complexities, unseen by the user.



For E-blocks1 combo board (EB083) overview please see Appendix 1, SECTION B (page 95).

Connecting E-blocks together

E-blocks2 are built on a bus-based concept. Each E-block connects together with a 16 pin Har-flex connector, with the female ports attached to the 'intelligent' upstream boards and the male connectors attached to downstream boards.

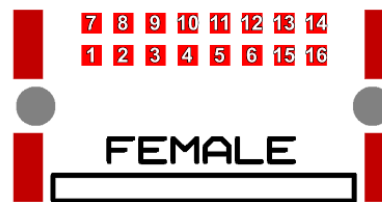


The diagram shows that the first three pins are used to transfer the power to the downstream board, pins 4,15 and 16 are reserved.

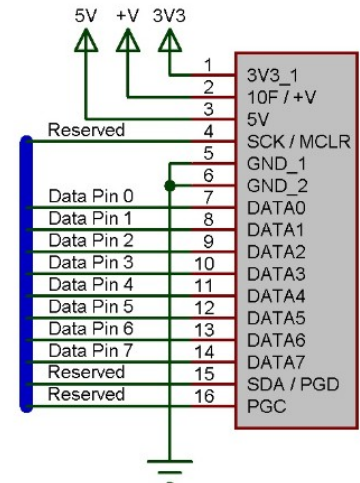
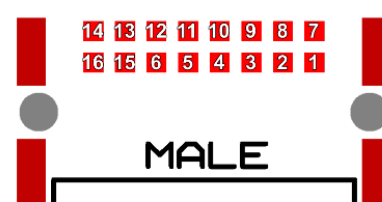
Pins 5 and 6 are connected to ground while pins 7-14 are the pins which transfer our 8 bits worth of data between the boards.

For E-blocks1 connections please see Appendix 2, SECTION C (page 96).

Upstream Connector - Female



Downstream Connector - Male

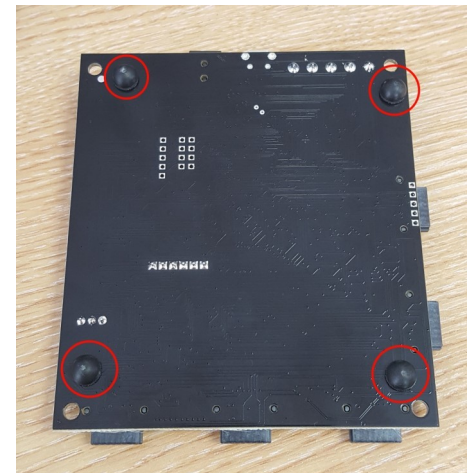


Using E-blocks on the bench

You do not need a backplane to use E-blocks - you can simply connect them together on the bench.

In each E-blocks package you will find a four small rubber feet to facilitate this. These provide a degree of protection for the E-blocks boards and help prevent short-circuits from tinned copper wire and other metal objects on the bench.

The disadvantage is that your E-blocks system is less portable as the connectors will be under more stress as the system is moved about.



Protecting E-blocks circuitry

Where possible, leaded components have been used for devices on E-blocks boards that are susceptible to electrical damage. This makes the task of replacing them simple should they be damaged.

To protect 'upstream' components, all 'downstream' E-blocks boards include protective resistors. Should errors occur when declaring the nature of port pins, e.g. an input declared as an output, no damage will be caused.

However there are circumstances where it is possible to cause damage:

- Care is needed when using screw terminal connectors and patch/prototype boards.
- Where possible, use protective resistors for the lines you need to connect when connecting two 'upstream' boards together with a gender changer E-block.
- Make sure you are earthed before handling E-blocks circuit boards to minimise the risk of static damage. If you have not got an antistatic wrist band, then touch a radiator or other earthed metal object.

Before making any changes to the E-blocks system, turn off the power supply.

Section 3: Introduction to Flowcode

Flowcode allows you to create microcontroller applications by dragging and dropping icons on to a flowchart to create programs. These can control external devices attached to the microcontroller such as LEDs, LCD displays etc.

Once the flowchart has been designed, its behaviour can be simulated in **Flowcode** before the flowchart is compiled, assembled and transferred to a microcontroller.

Introduction to Flowcode

This section allows those who are new to Flowcode to understand how it can be used to develop programs. It allows you to create programs step-by-step to learn about how Flowcode works. We advise that you work through every section to familiarise you with all of the options and features of Flowcode and introduce you to a range of programming techniques. As you work through each Section, please also refer to the Flowcode help file. The main Flowcode icons are introduced in turn.

Specifically in this section you will learn:

- how to use each Flowcode icon (except the C code icon).
- how the fundamental Components in Flowcode work - the LED, LCD, ADC, switch, 7-segment display, 7-segment quad display, keypad and EEPROM components.

What is Flowcode

The process

1. Create a new flowchart, specifying the microcontroller that you wish to target.
2. Drag and drop icons from the toolbar onto the flowchart and configure them to create the program.
3. Add external devices by clicking on the buttons in the components toolbar.
4. Edit their properties, including how they are connected to the microcontroller, and configure any macros they use.
5. Run the simulation to check that the program behaves as expected.
6. Transfer the program to the microcontroller by compiling the flowchart to C, then to assembler code and finally to object code.

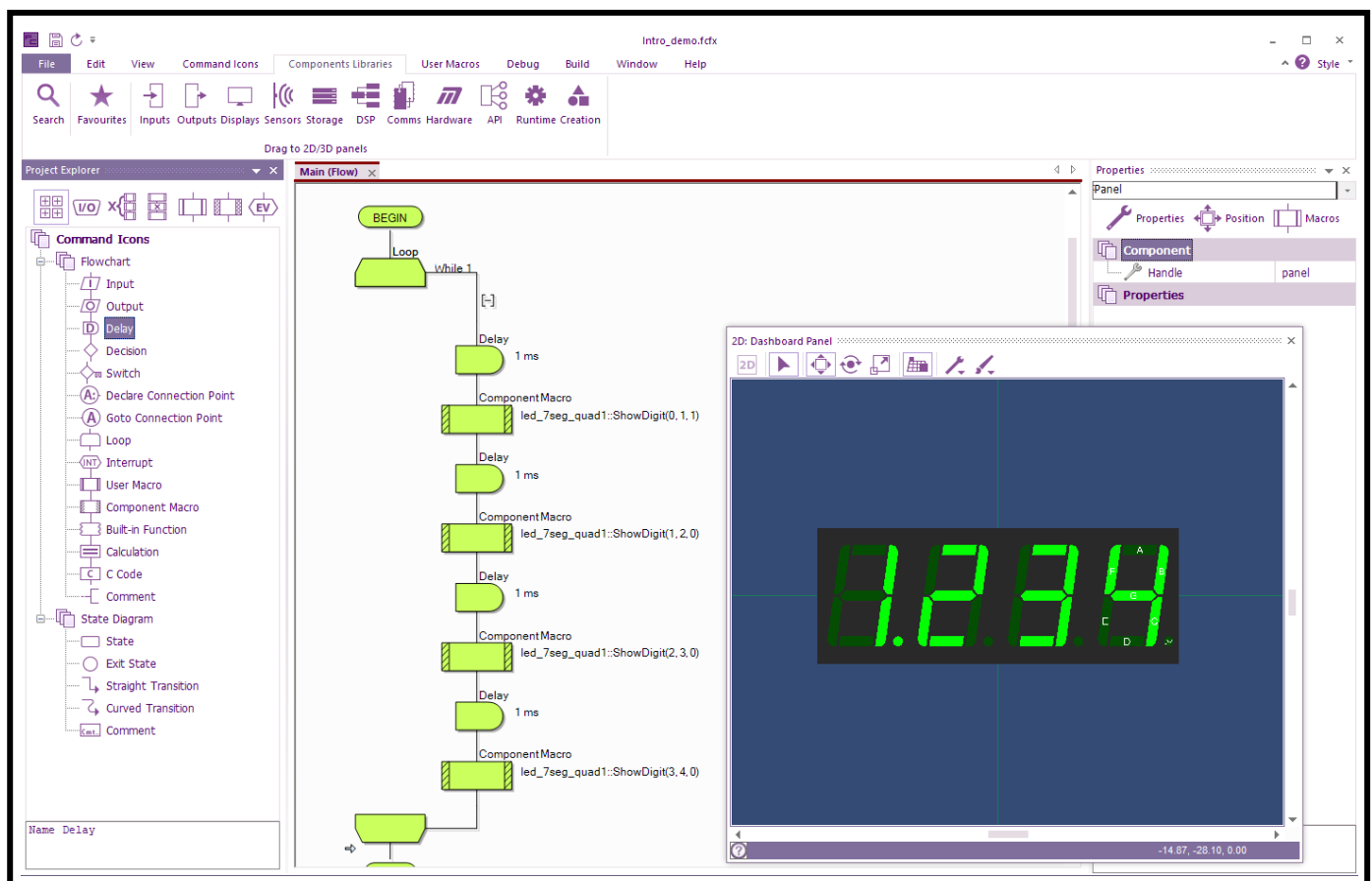
Flowcode overview

The Flowcode environment consists of:

- a main work area in which the flowchart windows are displayed.
- a number of toolbars that allow icons and components to be added to the flowchart.
- the System and Dashboard panels that display the attached components and provide basic drawing capabilities.
- the Project Explorer panel that shows project variables, macros and component macros.
- the Icon List panel that shows bookmarks, breakpoints and search results.
- windows that allow the status of the microcontroller to be viewed.
- windows that display variables and macro calls when the flowchart is being simulated.

Components Libraries toolbar

Connect external components to the microcontroller or use basic panel drawing commands. Components are grouped in different categories that appear as drop down menus. Click on a component and it will be added to the microcontroller and appear on the panel. The pin connections and properties of the component can then be edited.



A typical screen (testing the 7 segment display)



Command Icons

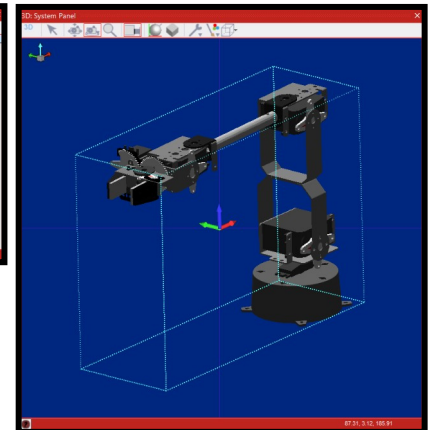
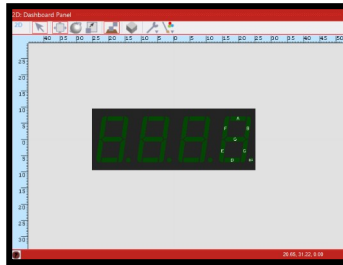
Drag-and-drop icons from this window onto the main flowchart window to create the flowchart application. Alternatively the icons are available in the Command Icons toolbar. This is the first tab of the Project Explorer, here docked left, but it can be undocked.

2D: Dashboard & 3D: System Panels

The components that you connect to the microcontroller will be displayed on one of these panels where you can interact with them when running a simulation.

They also provide basic drawing features like lines, shapes, and images, which can be used to build advanced professional-looking panels.

The **Dashboard Panel** is primarily for 2D use although it does offer a fixed 3D view. It is generally used as an interface showing the controls for interactive components.



The **System Panel** is the main 3D panel, offering many more features and options:

- full camera control.
- editable background environments with default Sky Dome and World Dome views.
- the option to use an image as the background.
- Shadow mode, offering both Tabletop and Object shadow options.

More details on these panels are found in the 'Flowcode - Getting Started Guide'.

(View > 3D: System Panel) / (View > 2D: Dashboard Panel)

Component Properties panel

All items on the panel, including the panel itself, have associated properties that are displayed in the Properties pane when the item is selected.

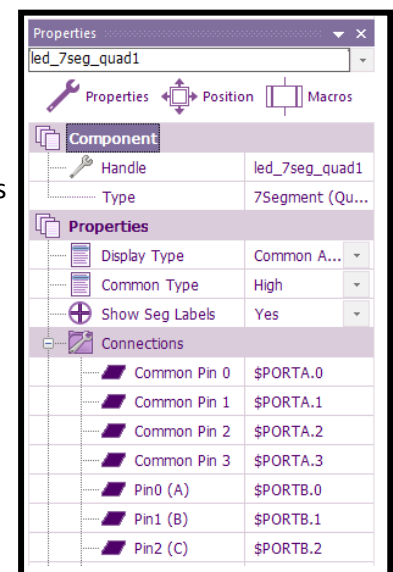
Some are read-only while others can be manipulated.

Some, like size and position, change as you interact with the item.

Others allow access to more advanced features of the selected item.

The Properties pane typically docks to the right hand side of the screen but looks like this when undocked:

(View > Component Properties)

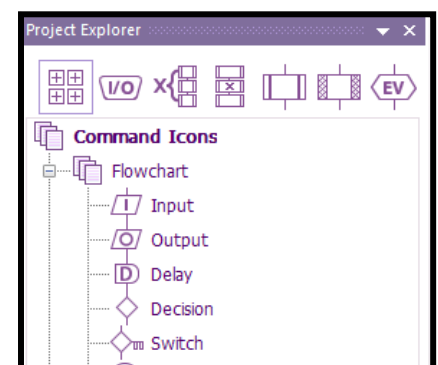


Project explorer

The buttons along the top of this panel allow you to select **Ports**, **Globals**, **Macros** and **Components**.

- **Ports** - variable names assigned to the microcontroller ports.
- **Globals** - any constants and variables that have been defined for use in the current project.
- **Macros** - user-created macros in the current program and allows the user to drag them into the current flowchart.
- **Component Macros** - lists components that are present in project

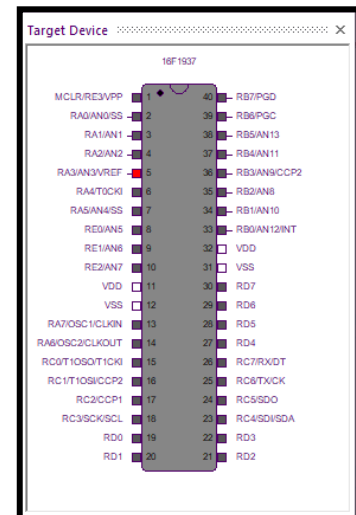
(View > Project Explorer)



Target Device window

The pinout for the currently selected microcontroller chip is displayed. When the flowchart is being simulated, the state of each microcontroller I/O pin is shown as red or blue, for 'high' or 'low' outputs respectively.

(View > Target Device)

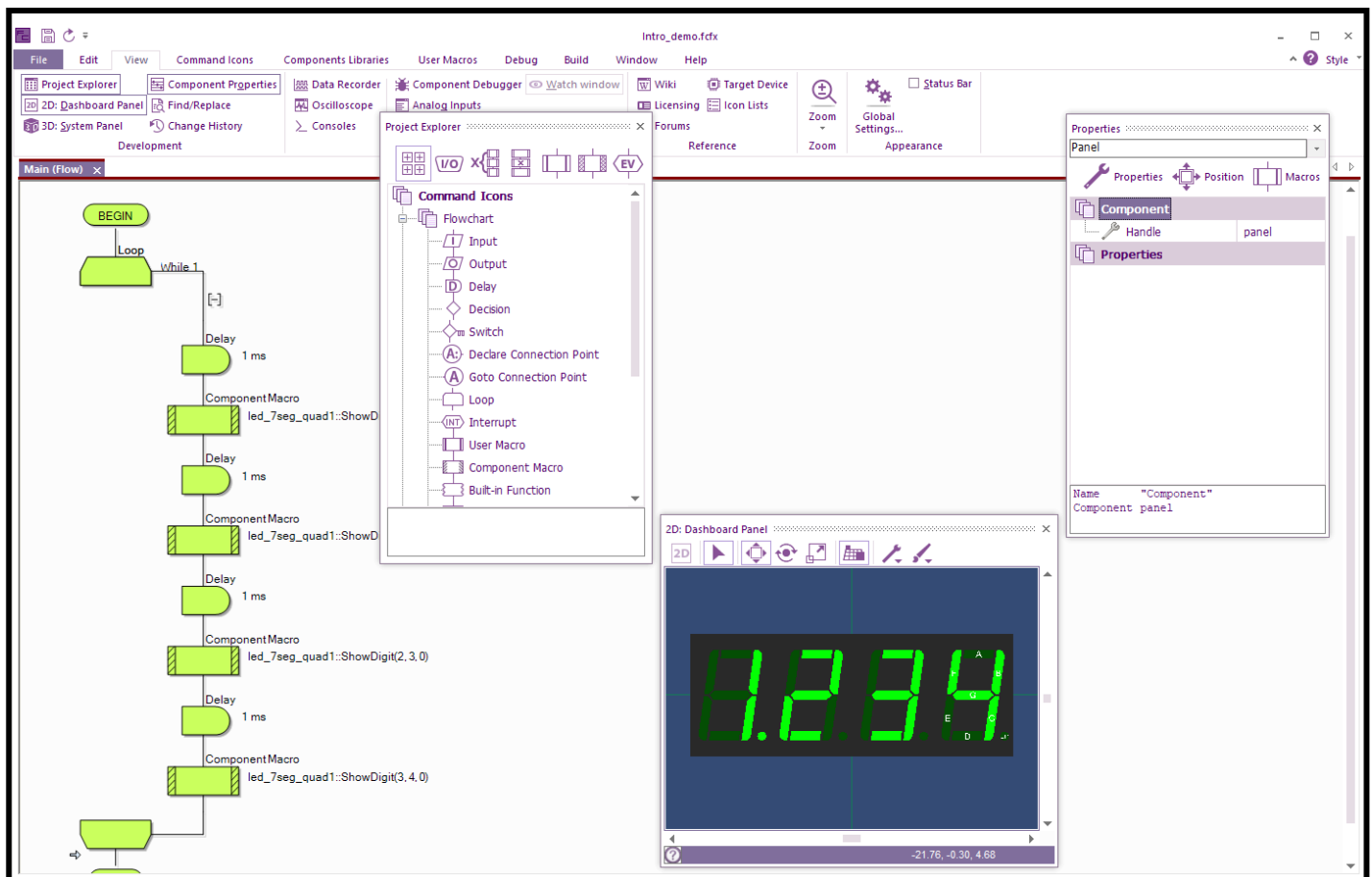


Docking and undocking the toolbars and panes

Toolbars and panes can be **undocked** from their default positions and either be left free **floating**, or **docked** to the top, bottom or the sides of the Flowcode window.

To undock a docked toolbar, simply click and hold on the toolbar **grab bars** (the speckled area at the top or side of the toolbar) and drag the toolbar to its new position.

To dock it again, double-click on the **grab bar**.

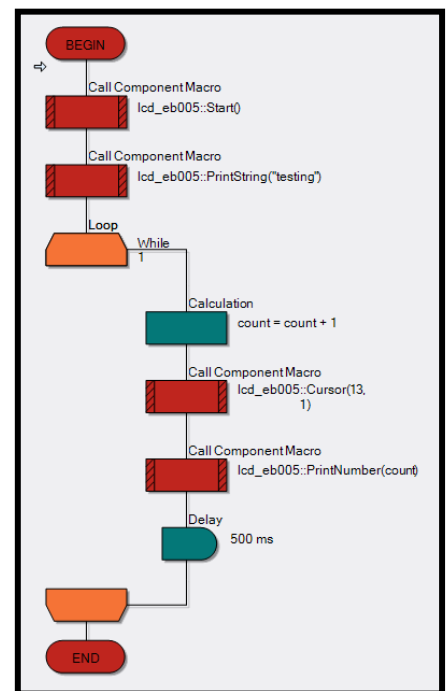
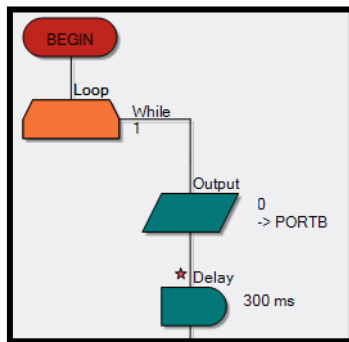


Example showing floating toolbars

Flowchart window

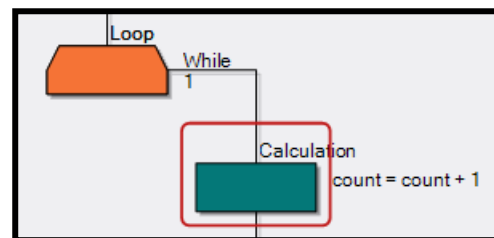
The icons that make up the flowchart are displayed in this main space. The text will change depending on properties selected, component macros called etc, The display names can be changed by the user to aid project organisation.

A red star alongside an icon indicates that the flowchart has not been saved in its current form.



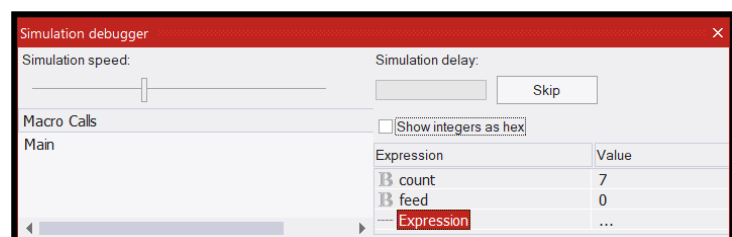
Simulation

When simulating a program in Flowcode a red rectangle around an icon indicates the icon to be executed next.



Simulation debugger

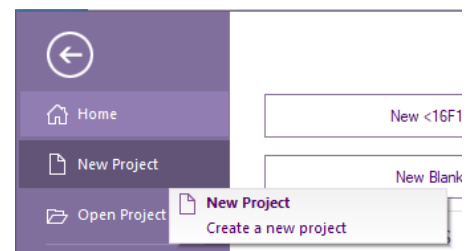
When simulating a flowchart, the current values of any variables used in the program appear in this window. These are updated after a command is simulated (unless the simulation is running at full speed - 'As fast as possible').



If you press the pause button during simulation, you can click on a variable in this window to change its value (allowing you to test the program under known conditions).

The window also shows the current macro being simulated under the **Macro Calls** section, useful when one macro calls another during the simulation process.

Please also see **Section 4 Flowcode First Program Page 43**



Starting a new flowchart

- Create a new flowchart by selecting **(File > New Project)**
- Select the microcontroller that you wish to target from the list presented.
- Click the “New Embedded Project” button

Opening an existing flowchart

There are a number of ways of opening an existing Flowcode flowchart:

- Select the Open option from the File menu **(File > Open Project)**
or
- Select the file from the list of most recently used files in the File menu.
or
- Double-click on a Flowcode (.fcfx) file in Windows Explorer to launch Flowcode and open the file.

Saving a Flowchart

To save a flowchart, select either the **Save** or **Save As** options from the File menu.

(File > Save / Save As).

Flowcharts must be saved before they can be compiled to C or transferred to a microcontroller.

Saving Flowchart Images

To save an image of the currently active flowchart, select **Save current Flowchart** from the **Save Image** sub-menu in the **File** menu **(File > Export > Save the current macro as an image).**

This function saves an image of the program to any file in the format chosen from the list:

- Bitmap (*.bmp)
- JPEG (*.jpg;*.jpeg)
- GIF (*.gif)
- PNG (*.png)

Note that the current zoom rate is used to determine the resolution of the image saved. If you need high quality images for printing then increase the zoom rate.

From the **Save Image** menu, you also have the option to save the current image of either the **Dashboard Panel** or the **System Panel**

(File > Export > Save the current 2D Dashboard as an image).

These images can be saved to any file format chosen from the list:

- Model (*.mesh)
- Bitmap (*.bmp)
- JPEG (*.jpg;*.jpeg)
- GIF (*.gif)
- PNG (*.png)Model (*.mesh)

The View menu

This dictates which panels and toolbars appear on the workspace, a useful feature when trying to simplify its appearance.

It also has a **Zoom** menu, which allows you to display more icons in the workspace window than when using the default zoom setting. The current zoom setting is displayed on the **Zoom** sub-menu, and on the right hand side of the status bar, at the bottom of the Flowcode window.

The size of each icon is dictated by the zoom level - for larger icons, zoom in - for smaller icons, zoom out. Use the Print Preview function to optimise the appearance of your flowchart on the paper.

The **Zoom** menu can also be accessed by right-clicking on the flowchart workspace.

Function key shortcuts:

- Increase Zoom (F3) - increases zoom size by 5%
- Decrease zoom (F2) - decreases zoom size by 5%
- Default zoom (F4) - set zoom to 75%
- Zoom to fit - Zooms to fit the whole flowchart into the current window
- Zoom to fit width - Zooms to fit the width of the flowchart into the width of the window.

Global Settings

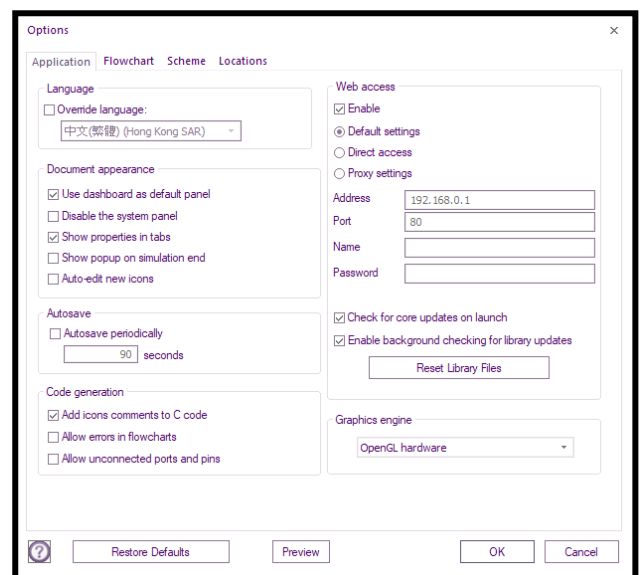
The **View** menu also includes a **Global Settings** for configuration of application and flowchart (**View > Global Settings**) Then select the appropriate Tab.

Application Tab

This tab enable setting of general application settings, such as language, document appearance, autosave feature, code generation options and web access.

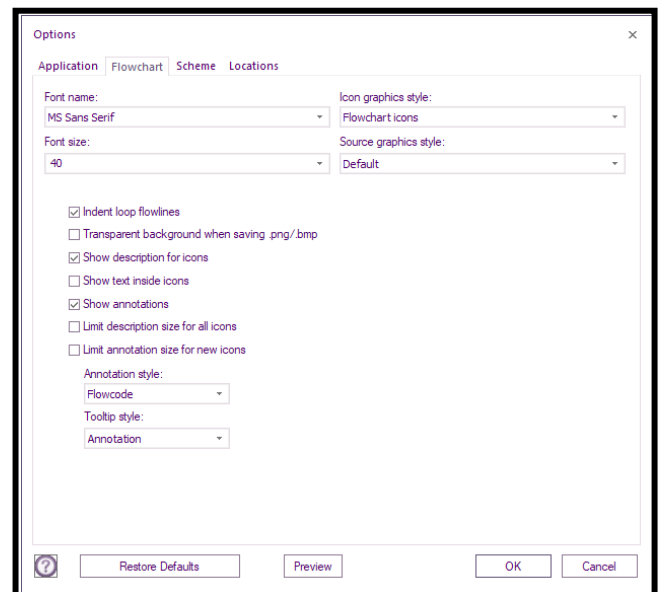
The OpenGL graphics engine can here be set as hardware or software mode.

The **Override language** option allows the user to override the default Flowcode language settings and to display Flowcode in a specified language. To do this, select the language from those available on the drop down list and restart Flowcode. It will do so in the selected language, provided the relevant language pack has been installed.



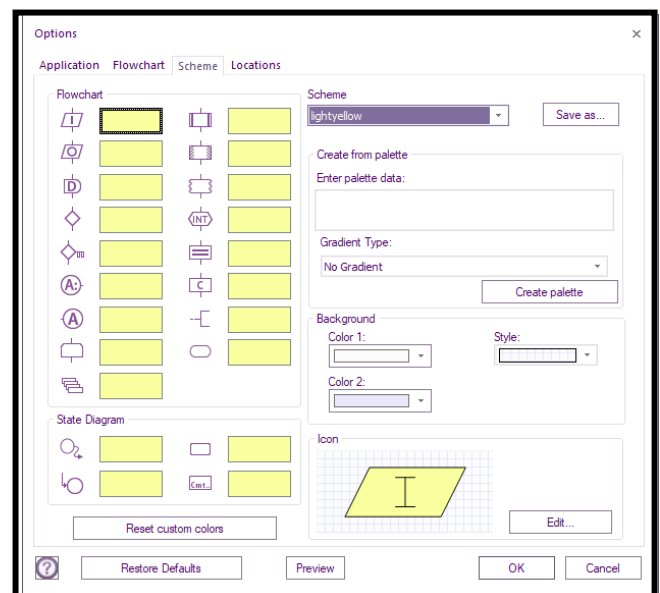
Flowchart Tab

This tab enable setting of flowchart display styles, text size and font. Annotation and tooltip style customizations.



Scheme Tab

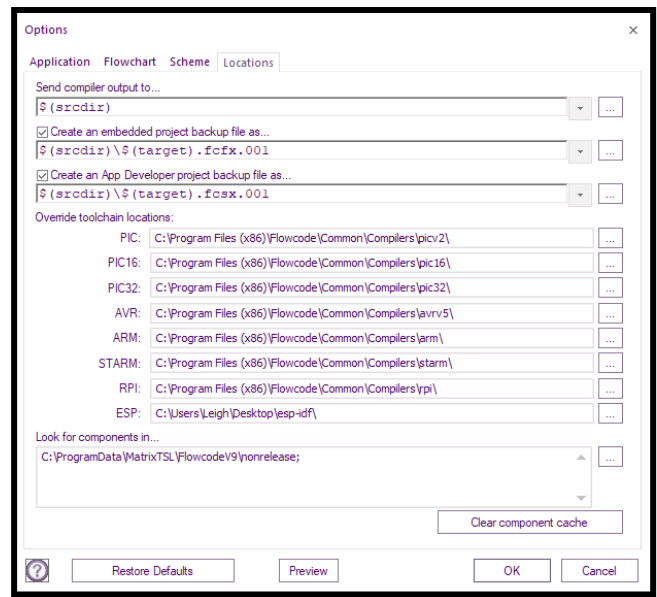
This tab contains the settings for changing the appearance of the flowchart, including icon colours and graphics, background colours and patterns etc.



Locations Tab

This tab enables the setting of backup filenames and the location of toolchain directories.

Additional directories can be added for the location of custom components.



View Windows (Simulation)

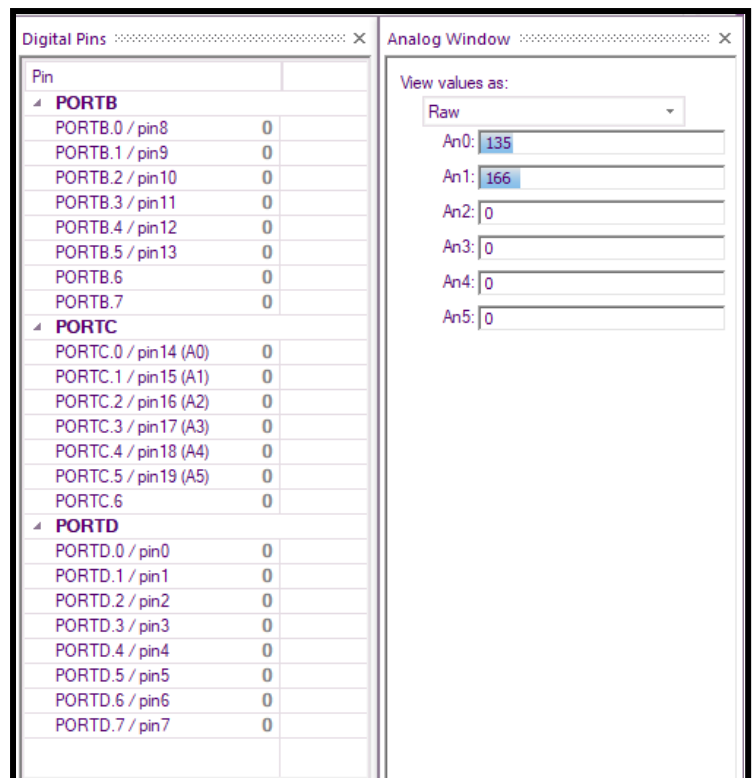
Analog Inputs and Digital Pins

Analog input values can be set and Digital pins monitored and set via the windows enabled from

View > Analog Inputs

and

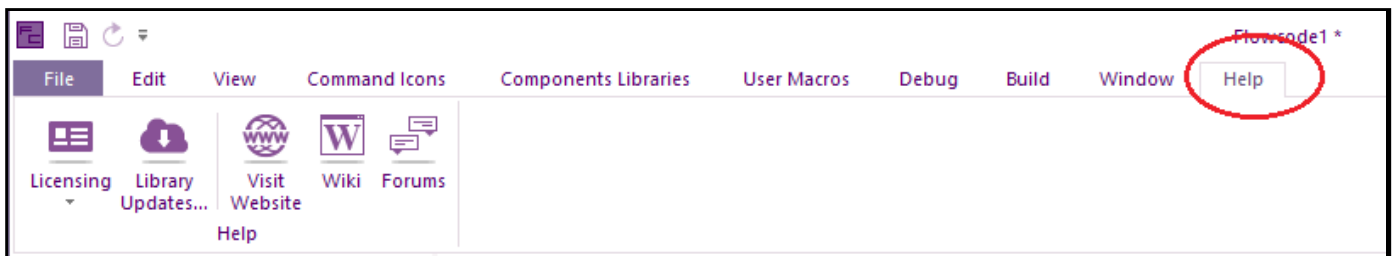
View > Digital Pins



Getting Help with Flowcode

Flowcode has within it and online an extensive **wiki** which can be accessed through the Help toolbar menu or via an internet browser and visiting this page:

<http://www.flowcode.co.uk/wiki/>



Additionally every single component within Flowcode has a page on the **wiki** which explains all the macros within it, and usually includes some examples as well.

To access the component help simply right-click your mouse on any component in either the 2D or 3D panel and select **Help**.

From here you can see:

- Explanation of the component
- Some examples of the component in use
- Macro references explaining what each macro plus the parameters and return values.

Library Updates

Flowcode components and target device information is kept up to date via an online system accessed from the Help menu (**Help > Library Updates**)

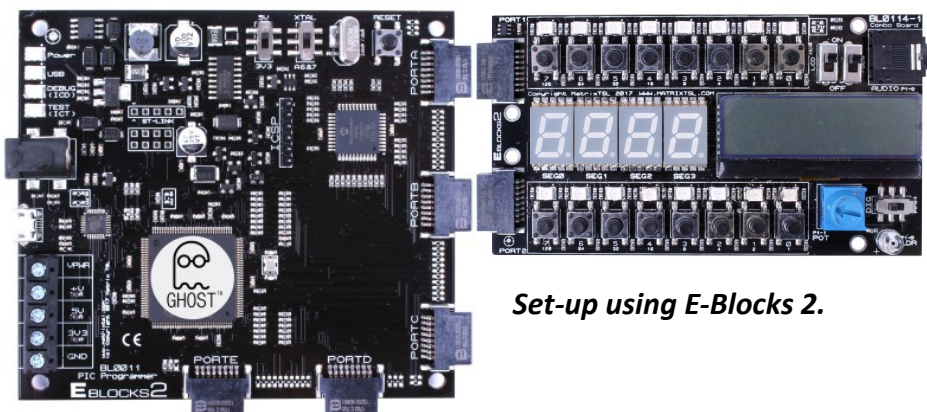
Section 4: Flowcode

First Program

Adding digital outputs - Light the LED

Create a program that lights an **LED** attached to the microcontroller.
This program introduces the topic of how to control a digital output.

The tutorial provides a clear, step by step approach enabling you to create your first program using Flowcode. It can be run in Flowcode's simulation mode before compiling to the board for testing and development.



Set-up using E-Blocks 2.

*Note: This tutorial refers to the port settings (ports A and B) as used with PIC.
For Arduino users, please use ports C and D as appropriate.
(Port C on the Arduino 'Maps' to Port A of the Combo board).*

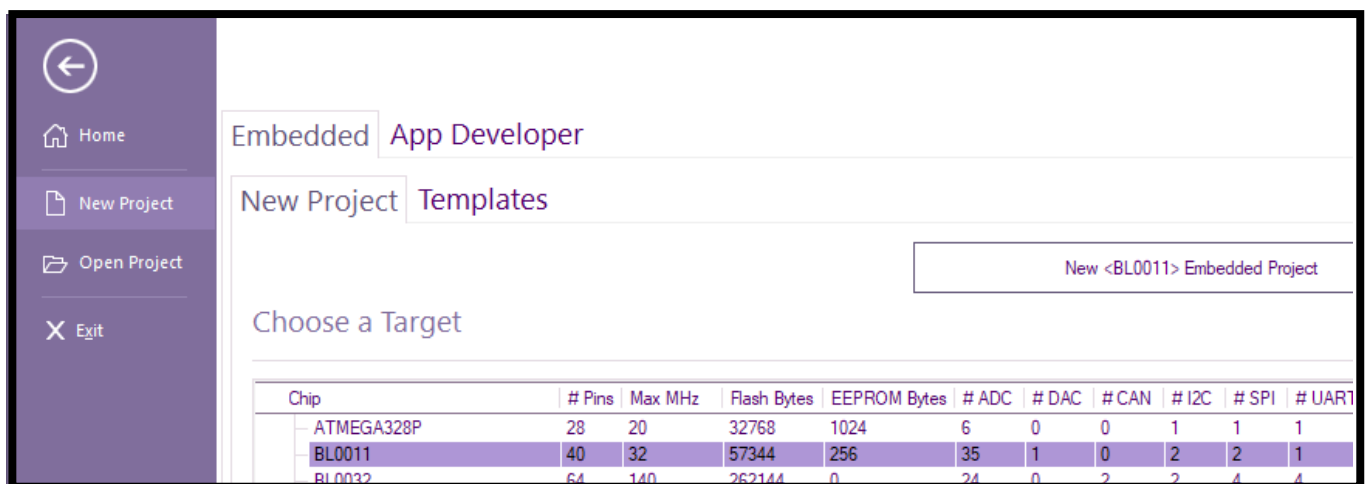
Starting a new project

Select 'New Project' at the welcome screen, or via the menu **(File > New Project)**
On the "Embedded" tab choose a target device or development board.

You will notice that the selection list includes details of the features and peripherals of each target. This is useful when selecting a device for a particular project.

For our new project, if we are using for example the MatrixTSL E-blocks2 PIC development board, choose the BL0011 target from the "Free targets" list.

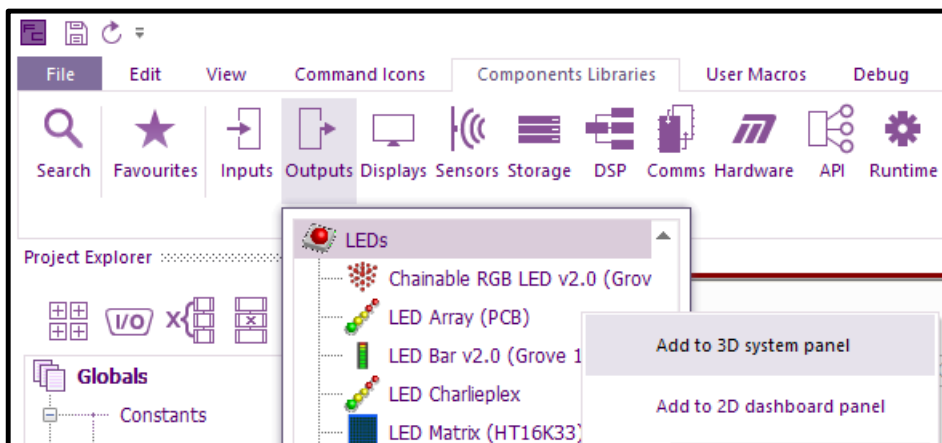
For Arduino users, please select an appropriate Arduino development board.



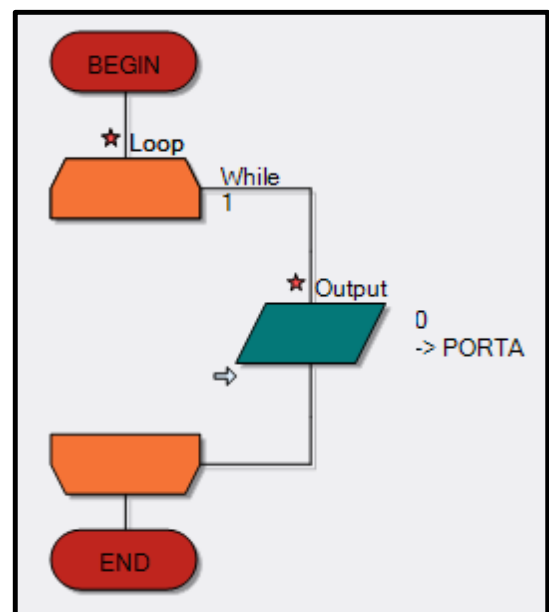
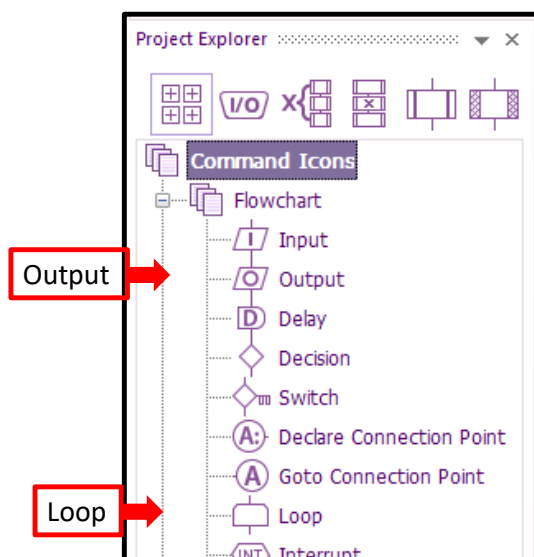
In this case the target choice also selects the correct 16F18877 device and presets the correct values for clock oscillator frequency and other settings.

Click on the "New <BL0011> Embedded Project" button to start the project.

TIP: The project target device can be changed later via the menu **(Build > Project Options)**



Add an LED Array (PCB) to the 3D system panel.
 The LED array can be found under **Outputs** in the Component Libraries Toolbar.
(Component Libraries > Outputs > LED Array (PCB) > Add to 3D system panel)



Create a Flowchart.

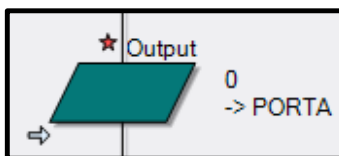
Move the cursor over the **Loop** icon, in the Icon toolbar. Click and drag it over to the work area. While dragging it, the normal cursor changes into a small icon. Move it in between the 'BEGIN' and 'END' icons. As you do so, an arrow appears showing you where the **Loop** icon will be placed. Release the mouse button to drop the icon in between the 'BEGIN' and 'END' boxes.

Add an **Output** icon within the loop on the flowchart in the same way.

TIP: The colours of the icons on your system may be different.

Changing port settings

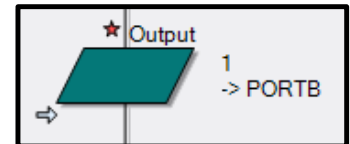
Double click on the Output icon that you've put in your flowchart and the **Properties** box will come up.



The 'Properties: Output' dialog box is shown. It has the following fields and options:

- Display name:** A dropdown menu with 'Output' selected.
- Port:** A dropdown menu with 'PORTB' selected.
- Variable or value:** A text input field with '1' entered.
- ☒ Use chip references
- ☐ Show advanced options
- Buttons: OK, Cancel

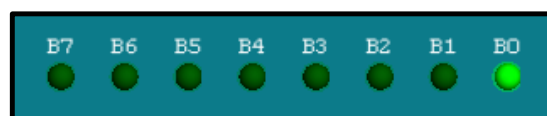
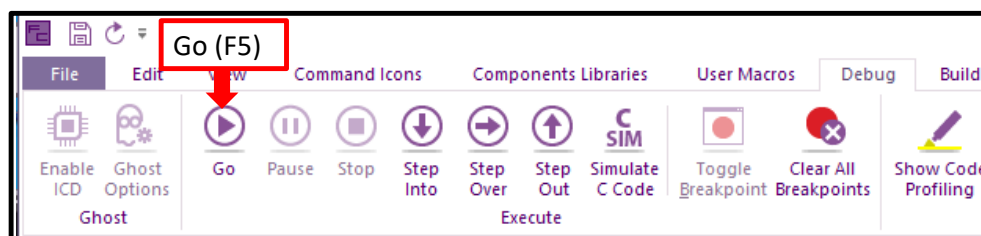
Select Port B.
Input a value of 1.



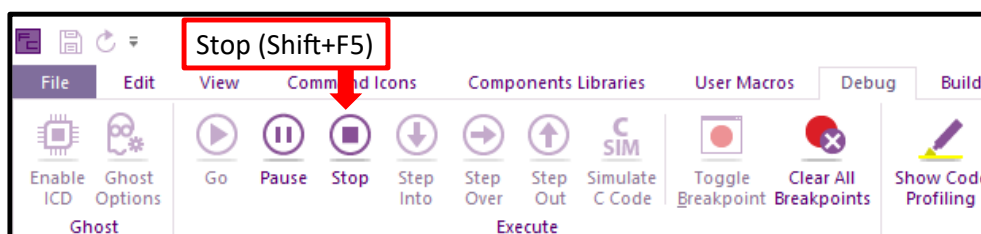
(You have done this because the LEDs in your 3D system panel are currently attached to port B, so we are sending the **Output** to the same port).

Run the simulation.

Select the **Go** icon from the **Debug** menu bar and the simulation of the LED will light up in the 3D system panel.

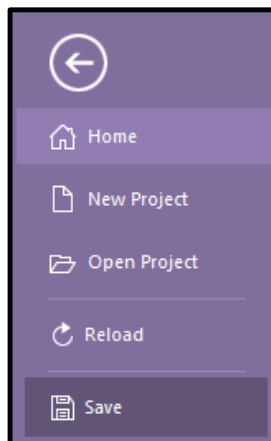


Simulation mode



TIP: Remember to stop your simulation before doing anything else.

(If Flowcode isn't doing as you expect, check that you haven't accidentally left your simulation running).

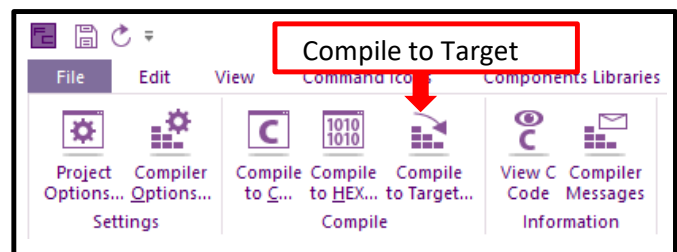


Save (Control+S)

Save your program (**File > Save**)

Connect your target development board to a power supply.
Connect the USB programming lead to your PC.

Click the **Compile to Target** from the **Build** menu as shown
(**Build > Compile to target**)

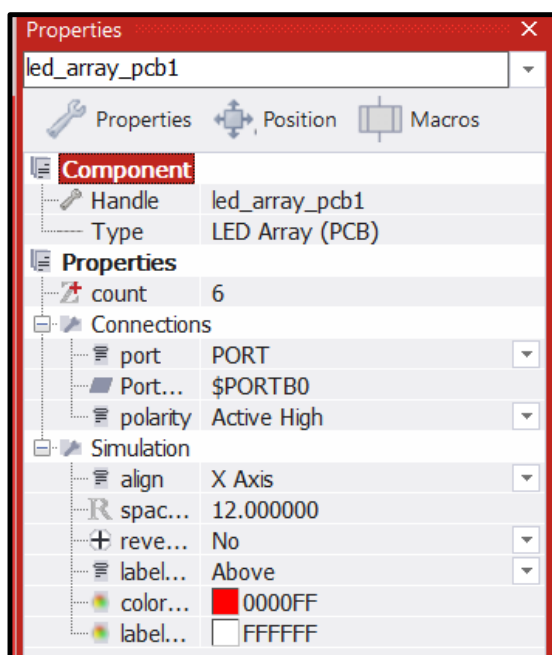


Changes to try after successfully lighting your LED.

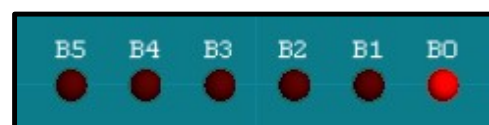
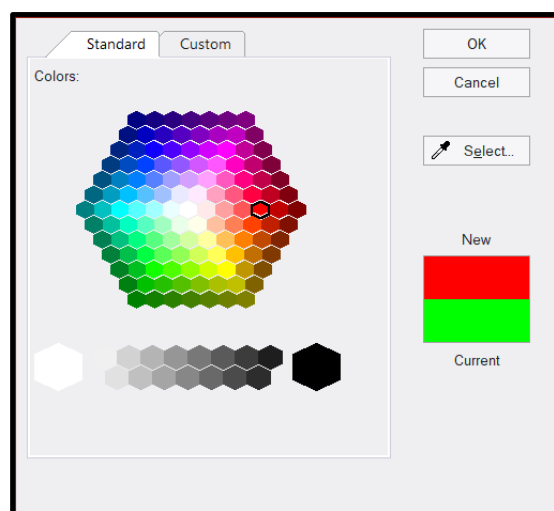
Highlight the image of the LED array in the 3D system panel and right click to select the **Properties**.

Here you can change the number of LEDs in your array by changing the value under **count**.

Try changing the colour of the LEDs in the simulation as shown below.



Property settings for 6 red LEDs.

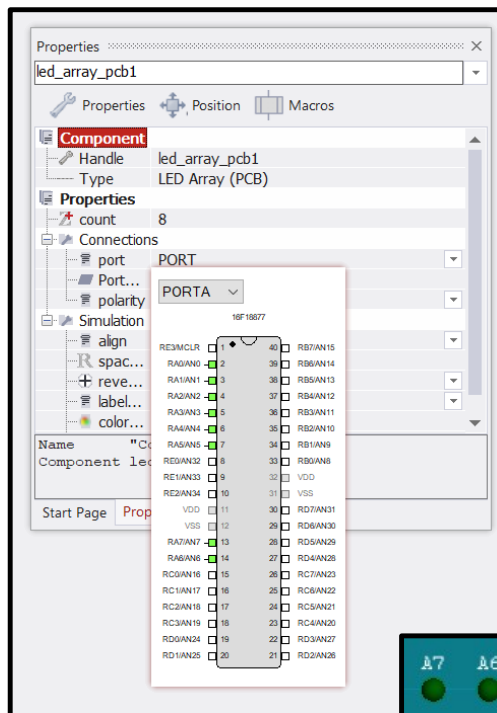
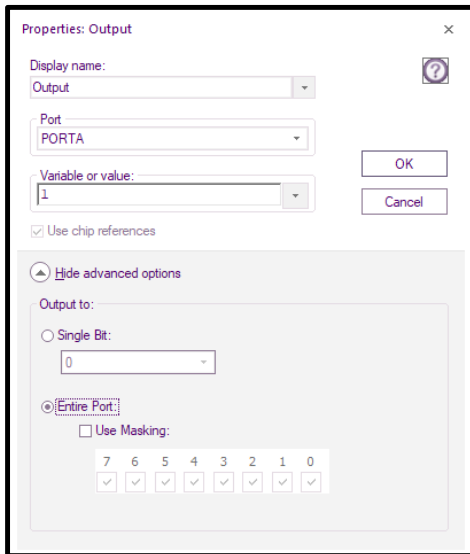
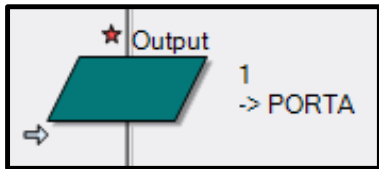


6 red LEDs in simulation.

Changing the port settings.

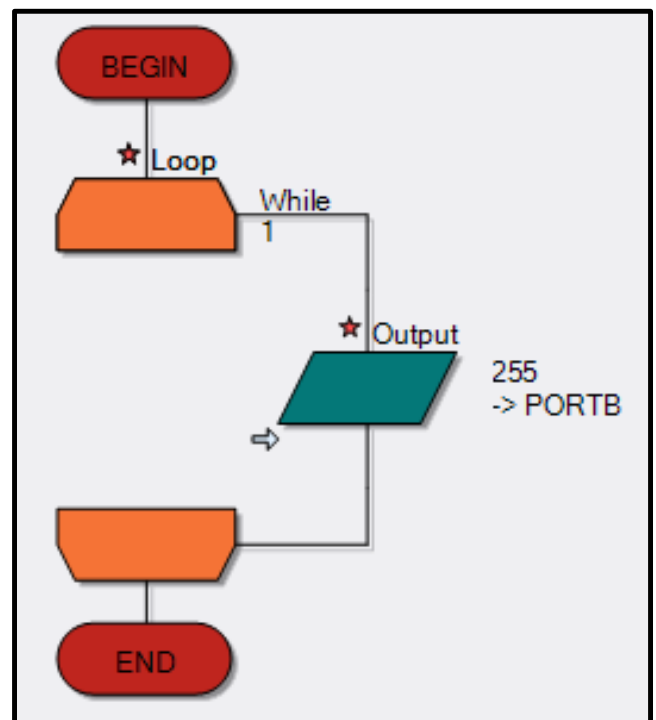
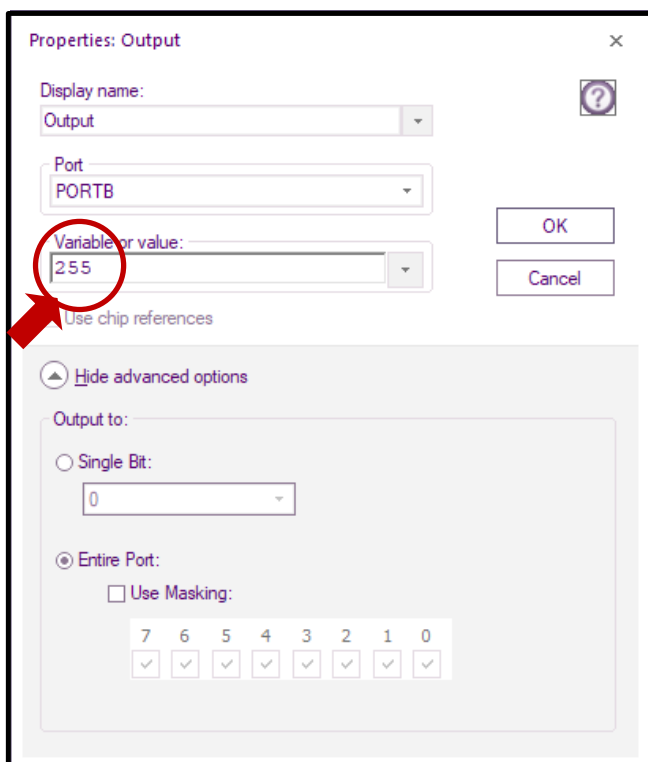
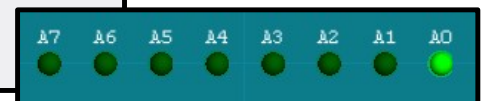
Bring up the Output icon properties (double click) and change the Port settings to **Port A**.

Highlight the image of the LED array in the 3D system panel and right click to select the **Properties**, and change the Port settings to **Port A**.



Run in simulation mode and then compile to chip. You should see the first LED of the other row light up.

Upper row LEDs



You can practise changing the ports by changing them back to **port B**.

Change the **value** from 1 to **255**. Test in simulation mode and then compile to chip (all 8 LEDs light up).

Experiment using other values. **(TIP: See Number Systems Worksheet).**

Binary Numbers

Digital electronic devices can't cope with decimal numbers. Instead, they use the binary system, which uses only two numbers 0 and 1. The number 1 could be represented by a high voltage signal, while number 0 could be a low voltage.

The decimal system uses ten numbers, 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. On reaching the last of these, 9, we start again with 0, but add another number in front. For example, after 8 and 9 comes 10, and after 18 and 19 comes 20 and so on. When we reach 99, both of these go back to 0s but with a 1 in front, to make 100.

| BINARY VALUE | | | | |
|--------------|---|---|---|---|
| 16 | 8 | 4 | 2 | 1 |

| Decimal | Same in binary |
|---------|----------------|
| 0 | 0 |
| 1 | 1 |
| 2 | 10 |
| 3 | 11 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| 8 | 1000 |
| 9 | 1001 |
| 10 | 1010 |

In **binary**, the same thing happens, but a lot more often, because it uses only 0s and 1s. Counting up starts with 0, then 1, then back to 0 with a 1 in front, making 10 (not ten - it's two) Next comes 11 (three) and start again with two 0s but with a 1 in front, to give 100 (four) and so on.

| Decimal | Same in binary |
|---------|----------------|
| 1 | 1 |
| 2 | 10 |
| 4 | 100 |
| 8 | 1000 |

Notice that each time the binary 1 moves one place to the left, it doubles in value of the number in decimal.

We can use this idea to convert between number systems.

TIP: In any binary number, the bit at the left-hand end, the Most Significant Bit (MSB), has the highest value. The one at the right-hand end, the Least Significant Bit (LSB), is worth least.

Hex Numbers

Hexadecimal, 'hex' for short, is a another system for representing numbers.

- A binary digit is either 0 or 1.
- A decimal digit varies between 0 and 10.
- A hex digit has sixteen possible states.

Sixteen states is a problem, as we have only the digits from 0 to 9. To get round this, we use the letters A to F to provide the additional six digits required.

Working with the binary number with eight digits is a handy convention as computers (and the PIC MCU) store information in groups of eight bits.

A single memory cell inside the PIC MCU can store a number ranging from 0000 0000 and 1111 1111. In decimal this range is 0 to 255. The equivalent in hex is 0 to FF.

TIP: You can enter a hex number into Flowcode by preceding it with '0x' in any of the dialogue boxes.

Coding Constructs - Number Systems

| Binary value | | | | | Decimal value |
|--------------|---|---|---|---|---------------|
| 16 | 8 | 4 | 2 | 1 | |
| | | | | 1 | 1 |
| | | | 1 | 0 | 2 |
| | | 1 | 0 | 0 | 4 |
| | 1 | 0 | 0 | 0 | 8 |
| so | | | | | |
| | | 1 | 1 | 1 | 7 |
| | 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | 0 | 20 |
| 1 | 1 | 1 | 1 | 1 | 31 |

A single memory cell inside a PIC device can store a number ranging from 0000 0000 and 1111 1111. In decimal this range is 0 to 255. The equivalent in hex is 0 to FF.

| Decimal | Binary | Hex |
|---------|----------|-----|
| 0 | 00000000 | 0 |
| 1 | 00000001 | 1 |
| 2 | 00000010 | 2 |
| 3 | 00000011 | 3 |
| 4 | 00000100 | 4 |
| 5 | 00000101 | 5 |
| 6 | 00000110 | 6 |
| 7 | 00000111 | 7 |
| 8 | 00001000 | 8 |
| 9 | 00001001 | 9 |
| 10 | 00001010 | A |
| 11 | 00001011 | B |
| 12 | 00001100 | C |
| 13 | 00001101 | D |
| 14 | 00001110 | E |
| 15 | 00001111 | F |

Tasks

- 1 Complete the table below by:
 - a) Shading in the LEDs that light, for the first three rows.
 - b) Working out what number produces the LED patterns shown in the last three rows.

| Number sent to output | LED array |
|-----------------------|--|
| 51 | B7 B6 B5 B4 B3 B2 B1 B0 ○ ○ ○ ○ ○ ○ ○ ○ |
| 204 | B7 B6 B5 B4 B3 B2 B1 B0 ○ ○ ○ ○ ○ ○ ○ ○ |
| 195 | B7 B6 B5 B4 B3 B2 B1 B0 ○ ○ ○ ○ ○ ○ ○ ○ |
| _____ | B7 B6 B5 B4 B3 B2 B1 B0 ● ● ● ● ● ● ● ● |
| _____ | B7 B6 B5 B4 B3 B2 B1 B0 ● ● ● ● ● ● ● ● |
| _____ | B7 B6 B5 B4 B3 B2 B1 B0 ● ● ● ● ● ● ● ● |

- 2 Use Flowcode to:
 - a) Check your work from the table above using Flowcode.
 - b) Enter a hex number into Flowcode by preceding it with '0x' in any of the dialogue boxes.

Can you light the same LED patterns using Hex?

Section 5: Flowcode Examples

All of these examples can be tried out using either a PIC or an Arduino microcontroller.

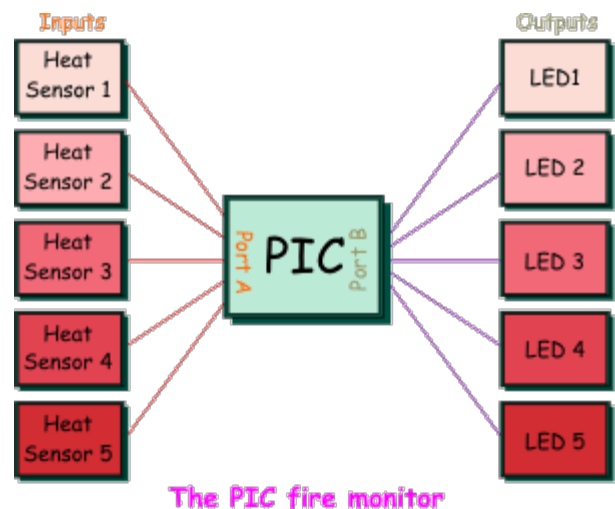
Arduino users should familiarise themselves with **Arduino Adjustments** in Appendix 1, (page 88) and adjust any port settings accordingly.

Example 1. Adding digital inputs - Where's the fire?

The scenario

A large building has a number of heat sensors in its fire alarm system. When there is a fire, the fire brigade needs to know where the fire is. In other words, they need to know which heat sensor has triggered the alarm.

The system is controlled by a PIC MCU. There are five heat sensors, connected as inputs to port A. Port B is set up as the output port and connected to a set of five LEDs. If a heat sensor detects a fire, the corresponding LED lights up.



Setting up the flowchart

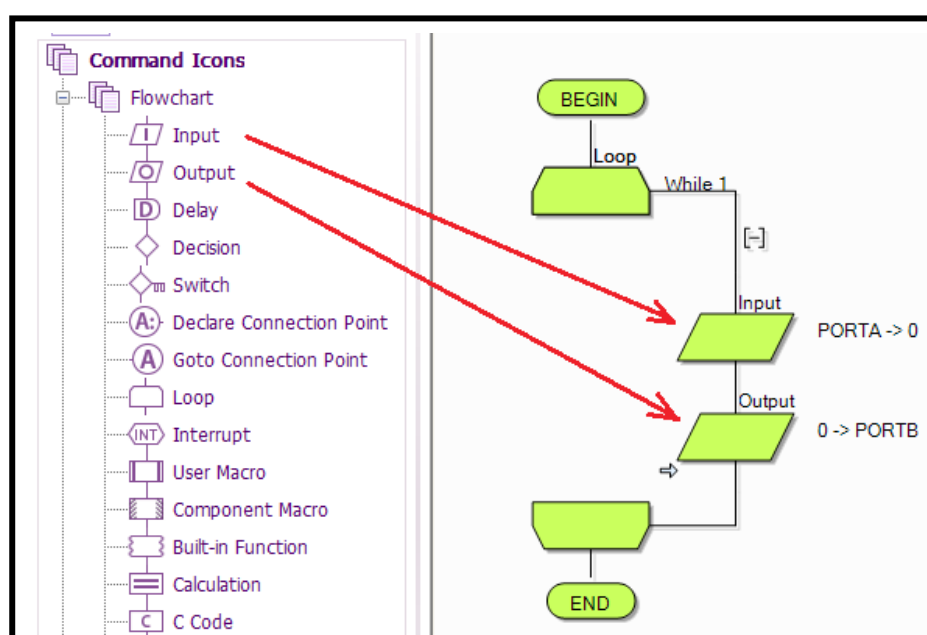
Open Flowcode and create a new project suitable for the board you are using.

Drag the **Loop** icon, the **Input** icon and the **Output** icon into your Flowchart from the icon toolbar to create a Flowchart as shown.

Set **Input** to port A and **Output** to port B.

For Arduino users, please use ports C and D as appropriate.


(Port C on the Arduino 'Maps' to Port A of the Combo board).



Creating the variables

- Right-click on the input icon, and select **Properties** from the menu. The **Input Properties** dialogue box appears, shown opposite. This allows us to add a **variable**. But what is a variable?
A **variable** is a place where we can store information, in particular, information that changes as our program runs. In this case, it is the number of the heat sensor that triggers the alarm. It might be sensor 1 that goes off, or sensor 5....

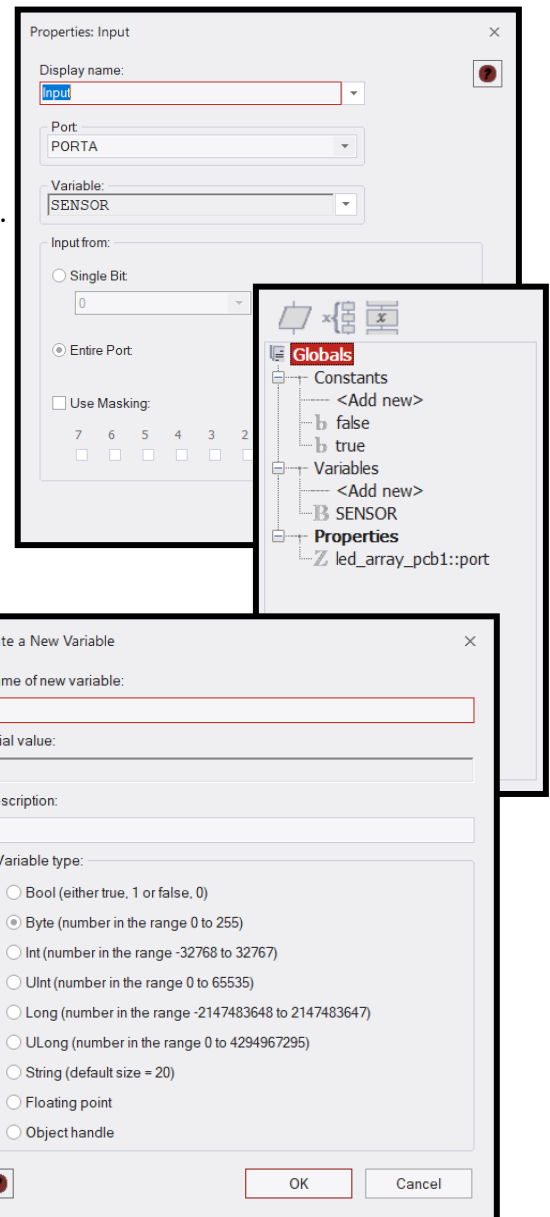
We are going to use a variable called **SENSOR** to store the information on which sensor has been triggered.

- Click on the arrow next to the **Variable** box.  You will see the next dialogue box.
- Now hover over the word **Variables** and the arrow appears. Click on it and select **Add new**. Another dialogue box, shown opposite, appears, offering a large choice of variable types. For now, accept the default type of **Byte**, a variable which can store numbers from 0 to 255. Type the name **SENSOR** as the name of the new variable and click on the 'OK' button. It now appears in the list of variables that the flowchart can use.
- Double-click on the name of the variable to use it, or alternatively click and drag the name into the variable box.

You now see the **Input Properties** box again.

Notice that you need to tell the system which port you are going to use to input the data the system needs. It is set to port A at the moment, and we are going to leave it that way.

In this case, the system needs to monitor the heat sensors and so each sensor will be connected to a different bit of port A. Click on 'OK' to close the Input Properties box.



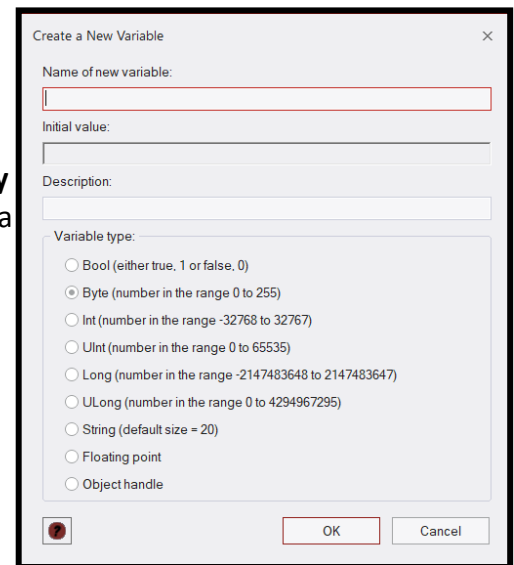
More on variables

In the previous section you added a variable to the program using the variable dialogue box.

Computer signals consist of streams of binary 0s and 1s on each wire. A group of eight wires can carry eight **bits**, (**binary digits**) simultaneously. This grouping of eight bits, known as a **byte** is used for much of the internal wiring inside microcontrollers and for the registers that hold and process data.

It is also used within memory subsystems. The contents of a memory register having eight bits can vary from 0 to 255.

A variable inside Flowcode can be configured to use just one memory register or more than one.



Flowcode variables:

Flowcode offers eight different types of variables:

- a **Bool (Boolean)** variable can either be 1 or 0 (true or false).
- a single register, known as a **Byte** variable, can store numbers from 0 to 255.
- a double register, known as an **Int** variable, can store numbers from -32768 to +32767.
- a double register can also be unsigned, when it is known as a **UInt** variable, which can store numbers from 0 to 65535.
- a quad register, known as a **Long** variable, can store numbers from -2147483648 to 2147483647.
- a quad register can also be unsigned, when it is known as a **ULong** variable, which can store numbers from 0 to 4294967295.

TIP: Use a **Byte** variable for simple counters and for variables that will not go above the value 255. It is the most economical in terms of memory space and also the fastest. Mathematical processes involving two bytes (often referred to as 16 bit arithmetic) take longer to execute. A multiple register, known as a **String** variable, can consist of a number of **Byte** variables - the default in Flowcode is 20.

Other variable issues

Floating point numbers (that contain a decimal point somewhere in them), can also be used, although they represent a much wider range of values than an integer. They suffer a loss of accuracy over large ranges.

Finally an **object handle** is used to reference a more complicated piece of data (such as a file, component or a block of text) whose internal format is not known.

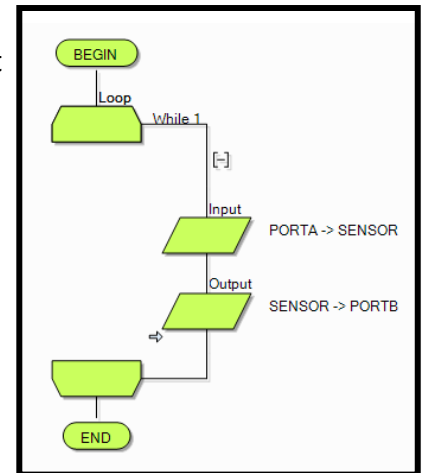
Why worry?

The number of registers inside a microcontroller is limited, and in larger applications the number and types of variables must be managed carefully to ensure that there are enough.

On downloading a program, the variables in Flowcode are implemented in the **Random Access Memory (RAM)** Section of the PIC MCU. In the 16F18877 there are 4096 Bytes of memory. This means you can have 4096 **Byte** variables, 2048 **Int** variables or 204 **Strings** each consisting of twenty **Bytes** or characters.

Setting up the outputs

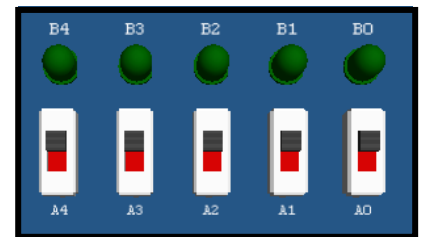
- Next, right-click on the Output icon, and select **Properties** or just double-click on it. The Output Properties box appears.
- Click on the arrow, next to the **Variable** box. You will see the **SENSOR** variable listed.
- Double-click on the word **SENSOR** or click and drag it to the **Variable** box.
- The **Output Properties** box now shows that the system is set to output whatever data is stored in the **SENSOR** variable. Change the port used to port B, by clicking on the arrow, in the port window, and then clicking on PORTB in the menu that opens.
- Click on 'OK' to close the Output Properties box.
- The flowchart should now look like this:



Notice the arrows in the icon annotations. They show that information will flow from port A into the flowchart, via **SENSOR**, (Input icon) and from the flowchart, via **SENSOR**, out to port B (Output icon).

Adding the LEDs

- Now click on the **Outputs** button and select the **LED Array (PCB)** icon. Click-and-drag it onto the System Panel.
- Change the **Count** property under the Simulation section to the value 5 by clicking on the box next to the Count property and using the keyboard to input the value.
- Click next to Port under the Connections section to open an interactive view of the chip, showing the compatible pins.
- Click on the drop-down menu and select the PORT B option. You have now connected the LEDs to the pins on port B.



(For Arduino users, please use ports C and D as appropriate).

Adding the switches

- You are going to use five switches to simulate the five heat sensors. The switch that is 'on' (closed) is the heat sensor that has triggered the fire alarm.
- Click on the **Inputs** button and select the **Switch Array (slide)**. Drag it into a suitable spot on the System Panel.
- Click on the box next to the **Count** property and change the value to 5. Check that the component is connected to PORTA.

Simulating the program

- Click once on the **Step Into** button. The **Simulation Debugger** window appears but ignore it for now.
- Move the cursor over one of the switches and click, to simulate detecting a fire. The switch graphic toggles to the closed position. Click the **Step Into** button a few more times to simulate the complete program.

The program is finished. You have just detected a fire, which turned on a heat sensor.

The LED array tells you, or the fire brigade, which sensor detected the fire.

Example 2. Using loops - Counting sheep

Counting sheep, badly at first, but without falling asleep!

The plan is straightforward - when a sheep passes through the gate, it breaks a light beam. This sends a pulse to a counting system, which then adds one to the total stored in the system.

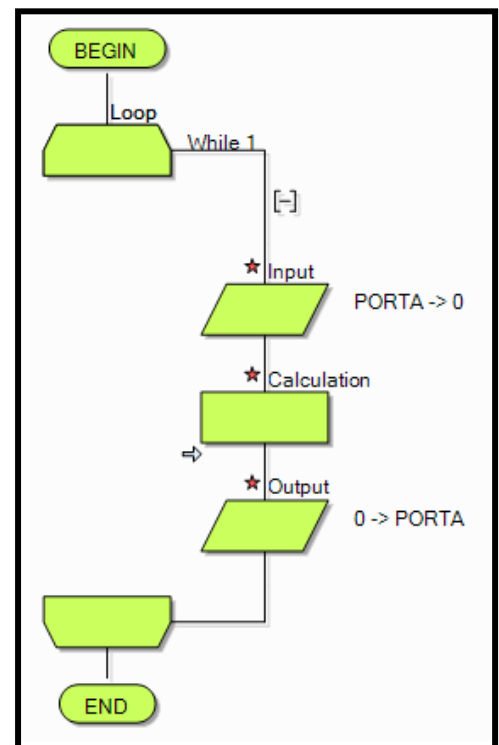
We display this total on the LED array.

(Note that Flowcode has a Beam Breaker component, based on the Collision Detector. Although this would do a far better job, for now we detect the light beam interruption using more basic methods).



Setting up the flowchart

- Launch Flowcode and start a new flowchart.
- Create the flowchart shown opposite.
- It contains a **Loop** icon and a **Calculation** icon.
- It contains an **Input** icon and an **Output** icon.

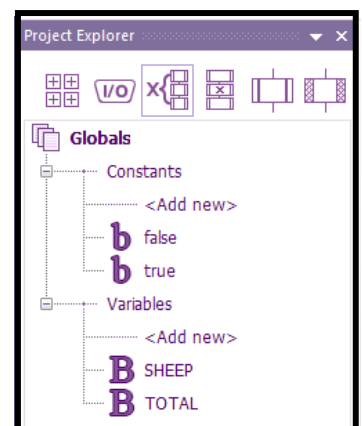


(For Arduino users, please use ports C and D as appropriate).

Creating the variables

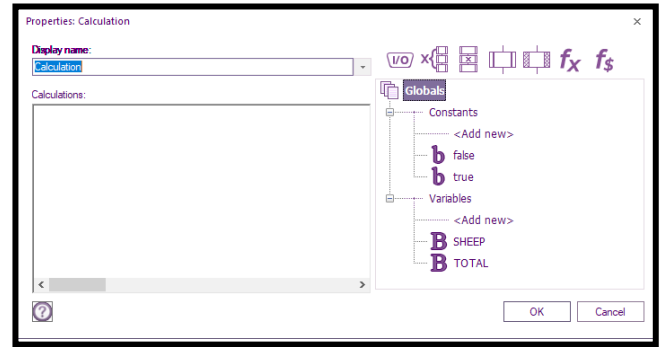
We are going to create two variables, one called **SHEEP** and the other called **TOTAL**.

- The **SHEEP** variable will show whether a sheep is present or not.
- The variable **TOTAL** will store the total number of sheep recorded so far.
- Click **View** on the menu bar, and select the Project Explorer (**View > Project Explorer**).
- Click on the **Globals** button at the top of the Project Explorer panel.
- Hover over **Variable** in the project explorer panel and click on the arrow to **Add new**. You now see the **Create a New Variable** dialogue box. Type in the name **SHEEP** and then click on 'OK'. You can leave the variable type as **Byte** as there will not be that many sheep.
- Create a variable named **TOTAL** in the same way.



Setting up the calculation

- Double-click on the **Calculation** icon to open the Properties dialogue box.
- Change the **Display name** to "New total".
- Create the calculation by typing the following in the **Calculations** window:
TOTAL = TOTAL + SHEEP
- We will simulate breaking the light beam using a switch on port A bit 0.
- The **Input** properties are set up to store whatever number appears on port A in the variable called **SHEEP**. Initially, that number is 0. When the switch is pressed, the number on port A and stored in the variable **SHEEP** is 1 (with only one switch, the biggest number we can create on port A is 1).
When the **Calculation** icon is executed, the number stored in the variable **SHEEP** is added to the **TOTAL** variable. Hence, when a sheep breaks the light beam, **TOTAL** is increased by 1. With no sheep present, **TOTAL** remains unchanged.
- Click on the 'OK' button, to close the dialogue box.



Configuring loop properties

- Double-click on the **Loop** icon to open its **Properties** dialogue box. (This shows the options for controlling the loop).
Next to the **Loop while** statement is the loop control text box, where you write the loop condition (the program continues looping until this condition is met).

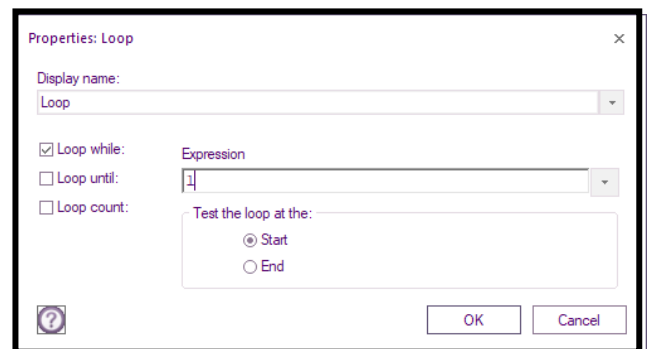
Examples of loop conditions:

- count = 10 (Loop runs as long as the variable 'count' = 10)
- count > 4 (Loop runs as long as the 'count' is greater than 4)
- count = preset (Loop runs as long as the 'count' is the same as the variable 'preset')

In all of these, looping continues as long as the condition in the **Loop while** text box is 'true'.

In programming 'true' has a special meaning. It is assigned a numerical value of 1 so that a test can determine if something is 'true'. Similarly 'false' is assigned the numerical value 0.

The default condition in the **Loop while** text box is 1 - this condition is always 'true' and so with this value, the loop will run forever. Programs usually contain a 'loop forever' structure. If they do not, the program will end suddenly and the computer will just sit there doing nothing.



When to test?

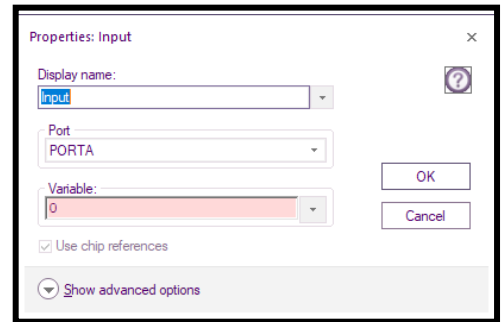
You can configure the properties to test the loop condition either at the start of the loop or at the end. Understanding this option is important. It can affect the number of times that the program will loop.

Loop for a set number of times

Sometimes, you just want to run a loop for a set number of iterations. To do this, check the **Loop count** box and enter the number of loops you want in the associated text box.

Setting up the input

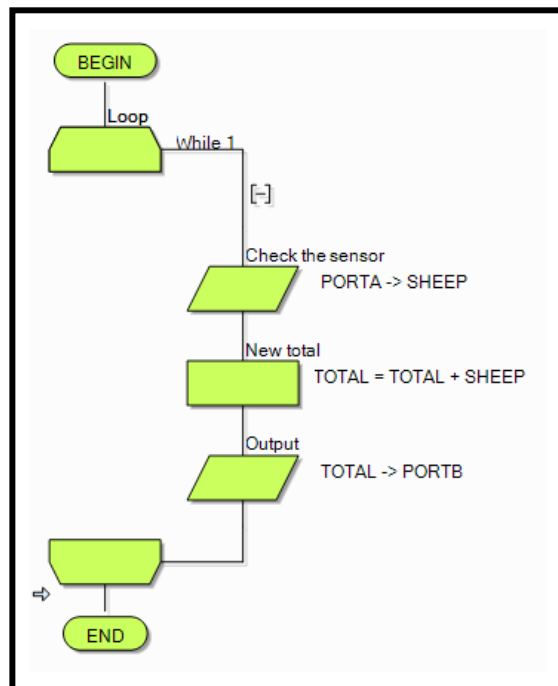
- Right-click on the **Input** icon, and select **Properties** from the menu, to see the following dialogue box:
- Change the display name. Double-click on **Input** in the **Display name** box and type "**Check the sensor**".
- Click on the arrow next to the **Variable** box to open the **Variable Manager**.
- Double-click on the word '**SHEEP**' to insert it into the **Variable** box.
- By default, the input is port A, which is what we want, ([Arduino use PORTC](#)), Click on 'OK' to close the dialogue box.



Setting up the output

- Double-click on the **Output** icon to open the output **Properties** dialogue box.
- Click on the arrow next to the **Variable** box.
- Double-click on the word **TOTAL** to insert it into the **Variable** box.
- In the output **Properties** box, change the port used to PORTB, ([Arduino use PORTD](#)).
- Click on 'OK' to close the dialogue box.

The flowchart should now look like this:



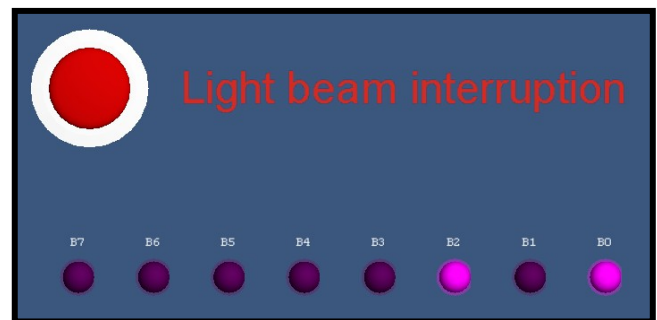
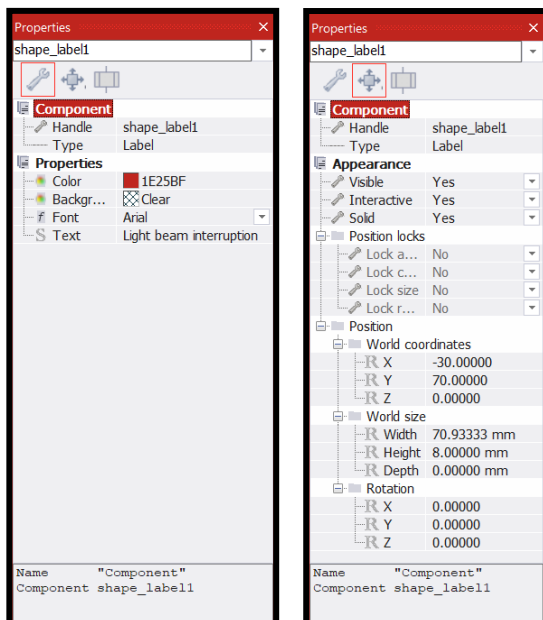
(For Arduino users, please use ports C and D as appropriate).

Adding the LED array

- Click once on the **Outputs** box and select the **LED Array** icon. Place it on the System Panel by moving the cursor over it and then 'clicking-and-dragging' it into position.
- Change the value of the **Count** property to 8 to set the number of LEDs in the array.
- Click the **Connections** property in the **Properties** pane. Select PORTB from the drop-down menu to connect the LEDs to the pins on port B.
- You can change the colour of the LED array in the **Colors** section.

Adding the switch

- A single push switch will represent the light beam sensor.
- Select **Switch (Push,Panel)** from **Component Libraries > Inputs**
Add or drag it onto the System Panel.
- On the **Properties** pane **Connections** section, check that the **Connection** property for the switch is **\$PORTA.0** i.e. the switch is connected to port A bit 0.
- Select **Label** from **Component Libraries > Creation**
- Click on the **Label** property in the **Properties** pane and replace the default text with "**Light beam interruption**".
- To adjust the size of the text, click on the **Position** tab and change the values of 'Width' and 'Height' under the 'World size' section. Move the text to a suitable position next to the switch.



Simulating the program

- Now run the simulation by clicking on the **Run** button.
- The **Simulation debugger** window appears - close it as it is not needed.
- Move the cursor over the switch and give the briefest mouse click you can.

What happens depends on how quickly you click, and how fast the PC works.

We want only the **B0 LED** to light, to show a total of 1 sheep. The program runs at high speed, however, and so keeps cycling through the **Input** and **Calculation** steps. As a result, before you have time to release the push switch, the total has incremented (increased by one) several times. This problem is explored in the next section.

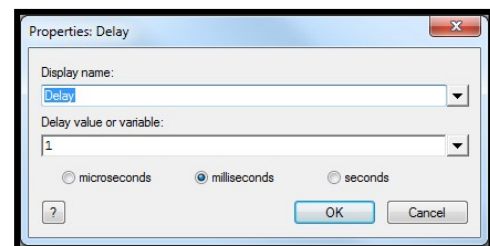
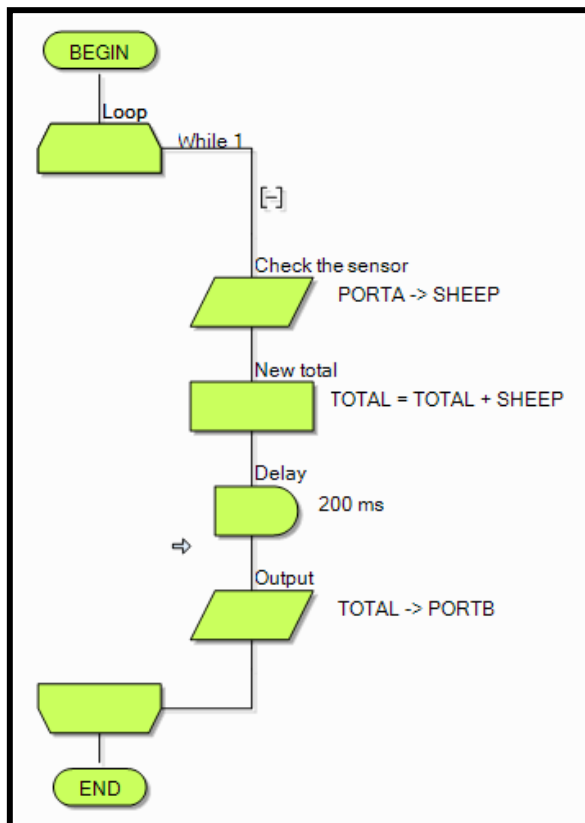
The solution: Adding a Delay

The problem - the program runs too fast!

Before we have time to release the switch, the program has run through several times, adding one to the total each time.

We need to slow it down by adding a delay.

- Move the cursor over the **Delay** icon.
- Drag it onto the main work area and drop it between the **Calculation** and the **Output** icons.

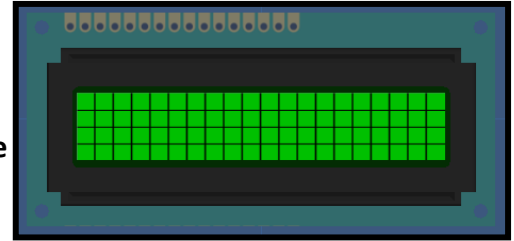


- Double-click on the **Delay** icon to open the **Properties** dialogue box.
- Change the value in the **Delay value or variable** box to 200 and then click on the 'OK' button. This causes a 200 millisecond (0.2 second) delay when the **Delay** icon is activated. In other words, the system just sits there and does nothing for 0.2 seconds.
- Now run the simulation again. Providing you don't keep it pressed for too long, you should find that the LED array shows an increase of 1 each time you press the switch.
- The program now works satisfactorily, providing the sheep rush through the light beam in less than 0.2 seconds. The delay could be increased to allow for slower sheep!

Note: This program shows the total number of sheep in binary format.

Example 3. The LCD display - Posting messages

Programs using the LCD display need to use the crystal oscillator. If necessary, in Flowcode, select **Build** from the main menu, then **Project Options** and finally the **Configure** tab. Select the **crystal oscillator** from the list of options (**Build > Project Options > Configure**).



LCD displays

Flowcode comes with a number of components that add commonly used subsystems to Flowcode, such as the LCD display, 7-segment display, and analogue inputs devices.

Here, we look at the LCD display, the basic text display subsystem on a range of electronics devices, from calculators to mobile phones. It can display text or numbers on one or more rows of the display.

In most programming languages, the LCD is one of the last things you learn, as it is quite a complicated device to program. However, Flowcode takes care of the complexities, making the LCD simple to use. The LCD display referred to here is the one used on the E-Blocks Combo board and on the LCD display - a four row, twenty character display.

Adding the LCD component

Before you can use the LCD, you need to add a LCD component to a Flowcode panel.

Select the **LCD (Generic, 20x4)** component from **Component Libraries > Displays** add it to the **System Panel**. A LCD display mimic will now appear on the panel.

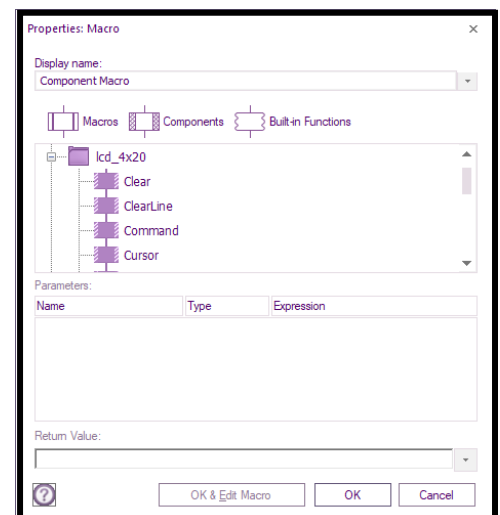
At the top of the **Properties** pane, the **Component** section identifies the component you have just selected. By default, the LCD is added to port B.

The LCD display requires five connections. It displays letters and numbers conveyed as serial data on this five wire bus. The techniques involved go beyond this tutorial. Fortunately, Flowcode has some embedded routines that take care of the complexities.

Drag a **Component Macro** icon onto the flowchart and open up the corresponding macro dialogue box by double-clicking on it.



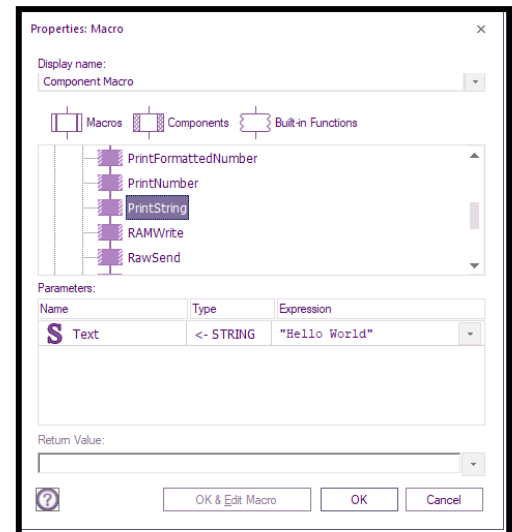
Now scroll through the **LCD** section in **Components** and select the macro called **Start**. This initiates the LCD, clears the display and gets it ready for action. We examine more LCD macros in the next couple of sections, but for now scroll through the available macros and take a quick look at each.



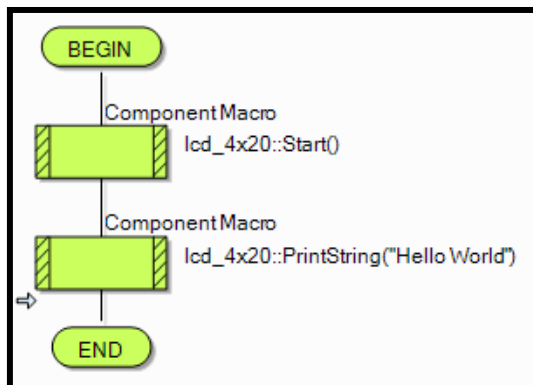
Writing messages

To display text on the LCD, simply type it in.

- Add another **Component Macro** to the flowchart and open the macro dialogue box.
- Select the LCD macro called **PrintString**. This requires a single parameter (item of data), 'Text' (the text to be printed).
- Type the text into the parameter box surrounded by quotation marks, e.g. **"Hello World"**



- Run the program and the text will be sent to the LCD display.



Other LCD functions

There are a number of other useful functions in the LCD macro list:

Clear - Clears the display and resets the cursor position (where the display prints next,) to '0,0' i.e. top left.

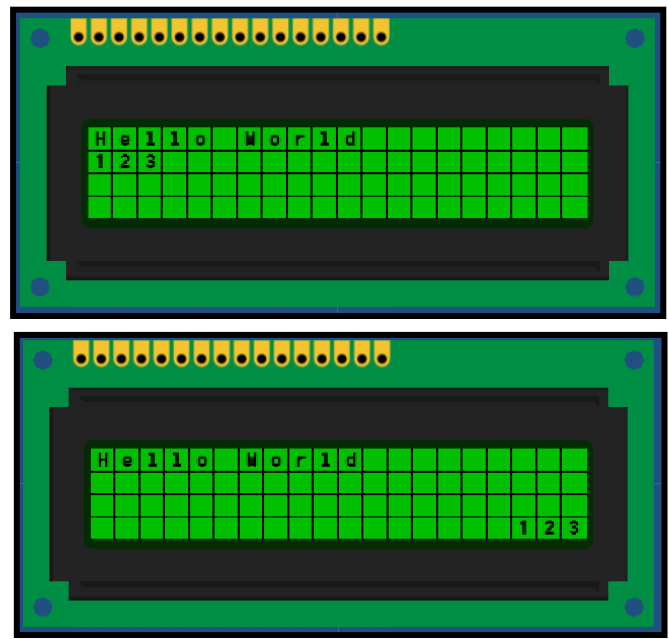
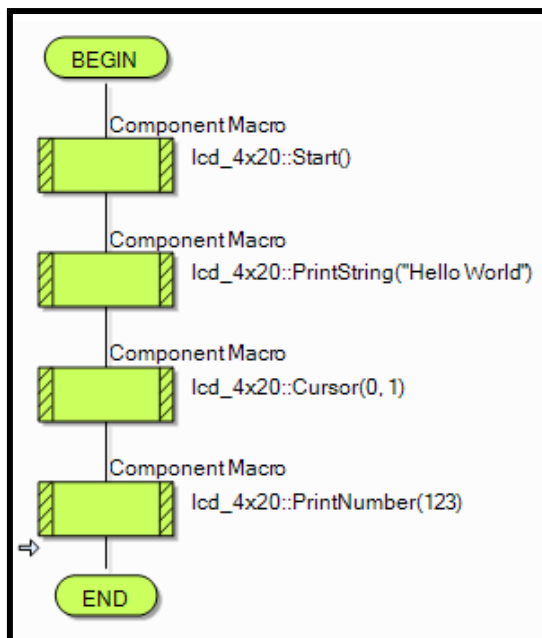
Cursor - Moves the cursor to the specified location. The two parameters, 'X' and 'Y' select the horizontal and vertical positions of the cell respectively. '0,0' is the top left cell, '0,1' the first cell on the second line, '3,2' the fourth cell on the third line etc.

PrintNumber - Works like 'PrintString' but prints a number instead of a string. It can be used with variables, or with actual numbers.

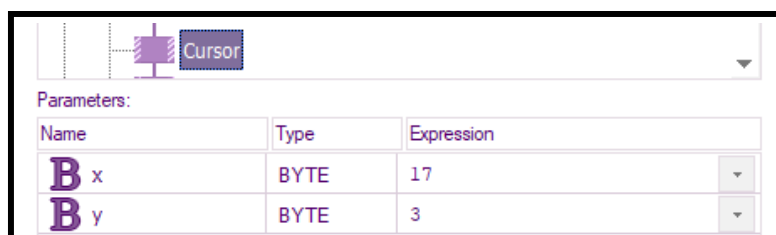
Using PrintNumber

Altogether we will add four **Component Macros** to the flowchart.

- To the first **Component Macro** add **Start**.
- To the second select **PrintString** and add **"Hello World"** (with quotation marks).
- To the third select **Cursor** and add **0,1** to the parameters.
- To the fourth select **PrintNumber** with the parameter value as **123**.
- Click **Run** to simulate the program.



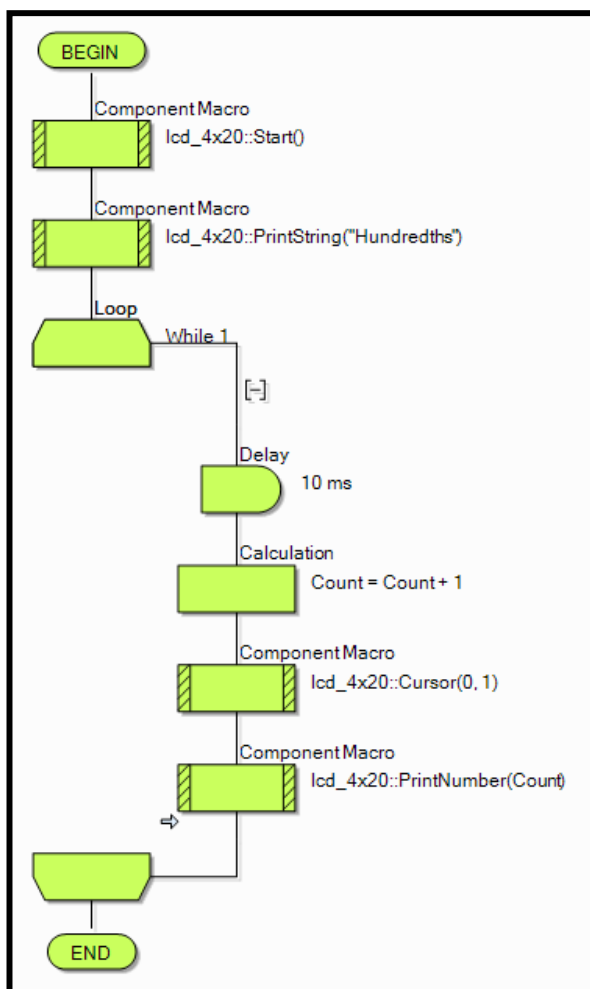
TIP: Try changing the values of the **Cursor** parameters and see where the numbers print. The 'y' value needs to be between 0 and 3 and the 'x' value needs to be between 0 and 19. (between 3 and 17 to see all three figures 1 and 2 and 3).



Example 4. Stopwatch

This example uses example 5 (Using PrintNumber) as a starting point.

- Expand the program from the previous example (**Using PrintNumber**) by dragging a **Loop** icon below the **PrintString Component Macro**.
- Change the text in the **PrintString Component Macro** to "**Hundredths**" (with quotation marks).
- Drag a **Calculation** icon into the **Loop**.
- Create a variable called **Count** as an **Int** type (Initial value 0).
- Double-click on the **Calculation** icon. In the **Calculations** text box type **Count = Count + 1** (This will add 1 to the value of variable count every time the icon is executed).
- Next drag another **Component Macro** into the **Loop**.
- Double-click the **Component Macro** and find **Cursor** under the LCD macros.
- Enter 0,1 as parameters to position the cursor on the first character of the second line.
- Next, drag another **Component Macro** onto the workspace.
- Select **PrintNumber** and enter **Count** as the parameter.
- Now, drag a **Delay** icon into the flowchart and set the delay to 10ms (which equals one hundredth of a second).
- The counter will count (approximately) the time elapsed in hundredths of seconds.



TIP: You can refine the program by clicking on each icon and entering comments to describe what the icon does.

It may seem like a lot of effort, but it can help with more complex programs.

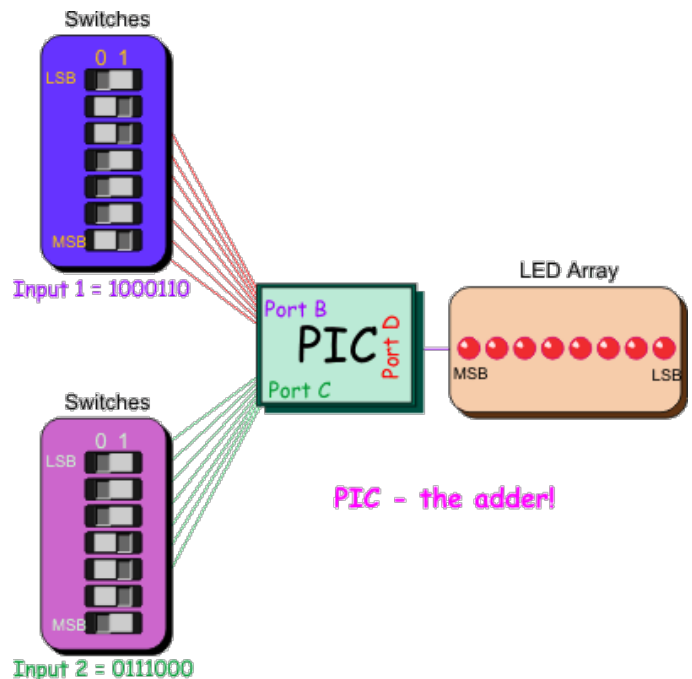
Example 5. Using binary numbers - A binary calculator.

In this section you build a binary adder - a system that makes the microcontroller add two numbers.

The simplest way to input a binary number is to use a set of switches attached to the input port.

To input two numbers, we need two sets of switches and two input ports.

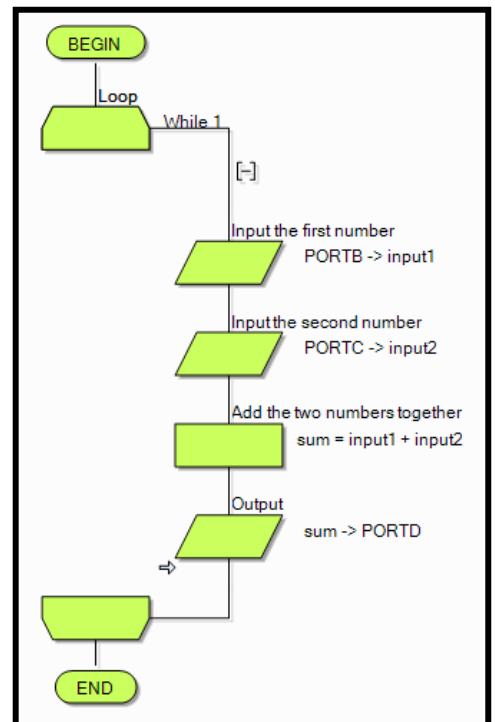
To see the result of the calculation, we will use an LED array, connected to the output port. We need a microcontroller with three ports.



Setting up the flowchart

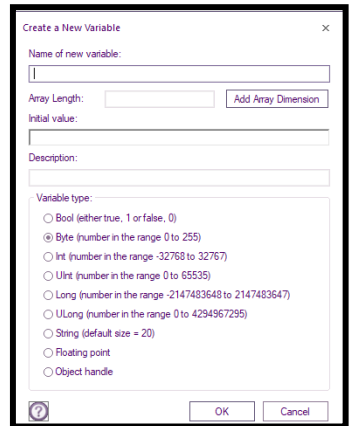
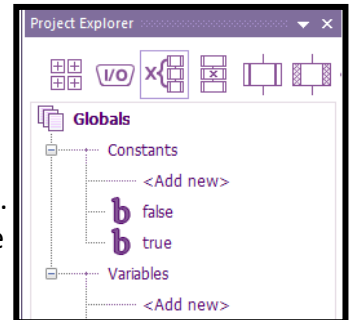
- Launch Flowcode and start a new flowchart.
- This time we take notice of this dialogue box: We need a microcontroller with at least three ports. Pull the slide bar down to find the 16F18877PIC MCU.
- Click on it to select it and then click on 'OK'.
- Click-and-drag a **Loop** icon.
- Click-and-drag an **Input** icon and drop it between the ends of the loop.
- Click and drag a second **Input** icon and drop it in between the ends of the loop.
- Click and drag an **Output** icon and drop it just below the **Input** boxes.
- Click and drag a **Calculation** icon, and place it in between the second **Input** icon and the Output icon.
- Your flowchart will look similar to this:

(the example image has descriptions and variables added).



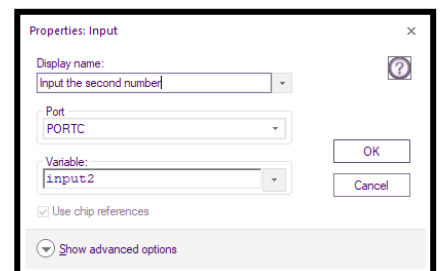
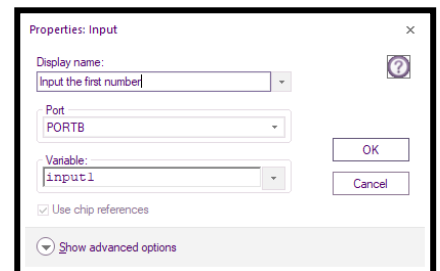
Creating the variables

- Click **View** on the menu bar and ensure that **Project Explorer** is checked (**View > Project Explorer**).
- Click on the **Globals** button at the top of the **Project Explorer** panel. We are going to create three **variables**, called **input1**, **input2** and **sum**. The first two store the numbers fed in from the switches. The variable **sum** stores the result of adding them together.
- Hover over **Variables** in the **Project Explorer** panel then click on the arrow that appears.
- Click **Add new** and the **Create a New Variable** dialogue box appears. Type in the name **input1**, and click on the 'OK' button - leave the variable type as **Byte**.
- Create variables, **input2** and **sum** in the same way.



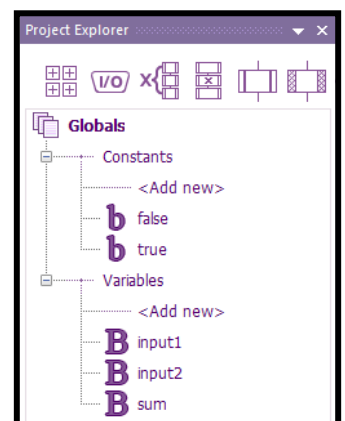
Setting up the inputs

- Right-click on the top **Input** icon, and select **Properties**. The **Properties: Input** dialogue box appears.
- Double-click on the word **Input** in the **Display name** box to highlight it.
- Type '**Input the first number**' to replace it. This will appear alongside the **Input** icon in the flowchart. (Adding labels like this helps users to understand what is happening).
- Click on the arrow next to the **variable** box to open the **Variable Manager**. This lists the three variables that you just created.
- Double-click on **input1** to use this variable in the input box.
- Back in the **Input Properties** dialogue box, click on the down arrow at the end of the port window and select **PORTB** to replace **PORTA**.
- Click on 'OK' to close the dialogue box.
- Double-click on the second **Input** icon, (a quicker way to open the **Properties** box.)



Configure this input to:

- display the label '**Input the second number**'
- use the variable **input2**
- use **PORTC**.
- Then close the dialogue box by clicking the 'OK' button.

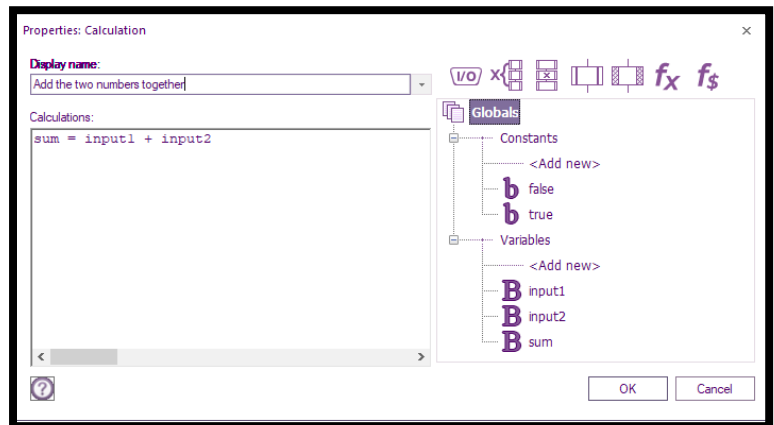


For Arduino users these two Ports will need to be set as follows:

- Input 1 set to **PORTC** (to use the Port A switches on the Combo board).*
- Input 2 set to **PORTD** (to use the Port B switches on the Combo board).*

Setting up the calculation

- Double-click on the **Calculation** icon to open the **Properties** dialogue box.
- Change the Display name: to '**Add the two numbers together**'.
- In the **Calculations** box insert:
sum = input1 + input2
(Either type this in directly, or drag in variables from the right window and then insert the = and + signs)
- Click on the 'OK' button, to close the dialogue box.

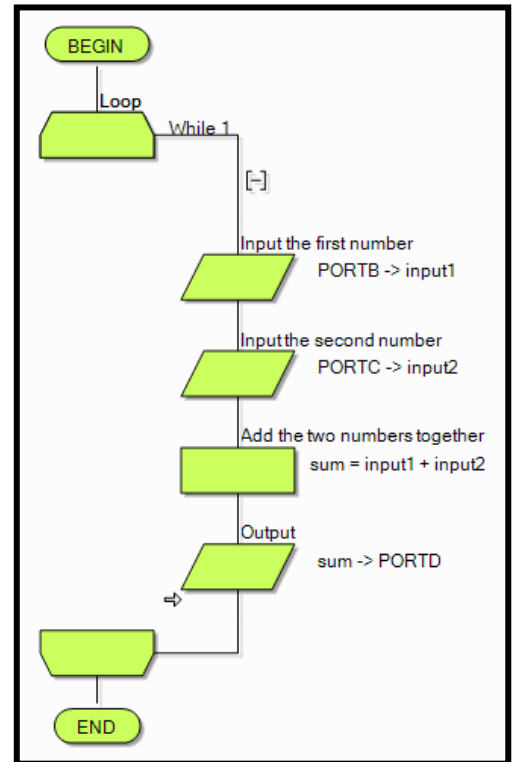


Setting up the output

- Double-click on the **Output** icon, to open the output **Properties** dialogue box.
- Click on the arrow next to the **Variable** box.
- Double-click on **sum** to insert it in the box.
- Back at the output **Properties** dialogue box:
 - change the port used to PORTD, ([Arduino PORTB](#))
 - click on 'OK' to close the dialogue box.

Adding an LED array

- Click on the **Outputs** tab and select **LED Array**
Place it in the middle of the System Panel by moving the cursor over the component and then clicking-and-dragging it into position (or right-clicking it and selecting Centre all objects).
- Click next to the **Count** property under the **Simulation** section on the **Properties** pane and change the number of **LEDs** to **seven**.
- Click next to the Port property and select PORTD from the drop-down menu to connect the LEDs to the pins on port D, ([Arduino PORTB](#)).
- Change the colour of the LED array to red (0000FF)



Adding the switches

Two sets of switches are used, one for each binary number. The output port has only eight bits. The biggest number it can output is 1111 1111 (= 255 in decimal). We are going to limit ourselves to inputting seven bit numbers meaning that the biggest number we can input is 111 1111 (= 127 in decimal). If we used bigger numbers, we would overflow the capacity of the output.

- Click on the **Inputs** tab, select **Switch Array (Slide)** and drag it onto the System Panel above the LED array.
- Open the **Properties** pane for the **Switch Array (Slide)**. Connect it to port B, using the arrow next to the Port property to open the drop down menu ([Arduino PORTC](#)).
- Add a second **Switch Array (Slide)** to the System Panel in the same way. Position it under the LED Array and connect it to PORTC ([Arduino PORTD](#)).



Slow simulation

As described earlier, Flowcode allows you to progress through the flowchart one step/icon at a time, to see the effect of each on the variables and on the output.

- There are three ways to simulate the program step-by-step:
 - Click on **Go** on the Debug toolbar and on the **Step Into** button (**Debug > Step Into**)
 - Press the **F8** function key on the keyboard.
 - Click on the **Step Into** button on the main toolbar in the simulation section.

Several things happen:

- a red rectangle appears around the **BEGIN** icon, showing that this is the current step.
- the **Simulation debugger** window appears (containing **Variables** and **Macro Calls**).
- the **Variables** section lists the three variables that you defined for this program, and shows their current values (all zero at the moment).

Ignore the Macro Calls section for the moment.

Now set up two numbers on the switch components.

- Move the cursor over the switch box connected to port B.
- Click on switches B0, B1, and B3, to activate them.

You have set up the binary number 000 1011 (= eleven in decimal.)

(Switch B6 gives the most significant bit and B0 the least significant bit).



- Set up the number 000 1111 (fifteen) on the switches connected to port C.
- Now Step Into to the next icon in the program by, for example, pressing F8 once more.
- The red rectangle moves on to the next icon, the Loop icon, but little else happens.
- Press F8 once again. The red rectangle moves on to the first Input icon.
- Press F8 again and the Variables box shows that the input1 variable now contains eleven (the result of the Input instruction just carried out).
- Press F8 again and the Variables section shows that input now contains fifteen.
- Press F8 again and the calculation is carried out. The sum variable stores the result.
- Press F8 again. The value stored in sum is transferred to the LED array.

It looks like:

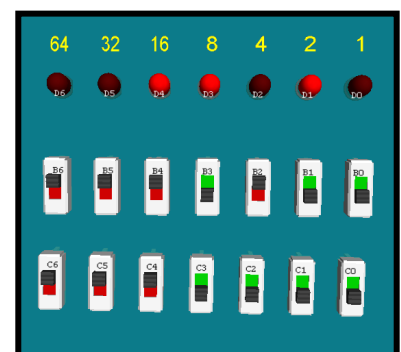


Reading from the most significant bit (D6) to the least significant bit (D0), the LED array shows the number 001 1010. In decimal, this is the number 26. No surprises there then.

- Repeat the same procedure using different numbers and step through the program to check what the sum of the numbers is.

TIP: Explore adding graphics to your binary calculator to make it easier to read.

Component Libraries > Creation to add digits above your LEDs.



Example 6. Binary logic in control.

Electronic systems can make decisions.

Very often, these are of the form "If this AND this is true, then..." or "If this OR this is true, then...". They rely on specific combinations of circumstances in order to take some particular action.

They are examples of using binary logic. The answer to the "If..." question is either "Yes" / "No", or "True" / "False", i.e. one of two possibilities (a binary solution). This answer could be expressed as a logic 0 or a logic 1 and electronically by a high voltage or a low voltage.

There is a class of digital electronic components, called **logic gates**, that perform exactly these decisions. The inputs and output are logic 0 or logic 1.

We can program Flowcode to make exactly the same decisions.



6A. Controlling a microwave oven

For reasons of safety, a microwave oven has a door sensor to make sure that the microwave generator will not operate if the door is open. Put another way, the generator operates if the door is closed **AND** one of the heating control switches is pressed.

We can build this condition into a Flowcode program.

Setting up the flowchart

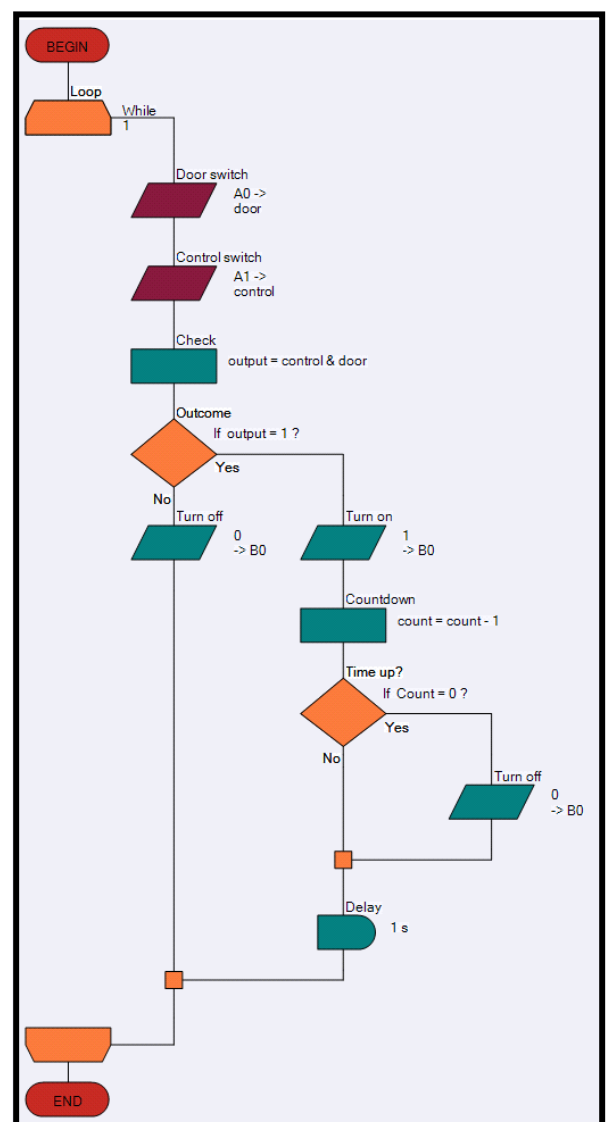
Launch Flowcode with a new flowchart. Create the flowchart shown opposite.

It uses:

- a **loop** icon
- two **input** icons
- three **output** icons
- two **decision** icons
- two **calculation** icons
- a **delay** icon.

Create four variables:

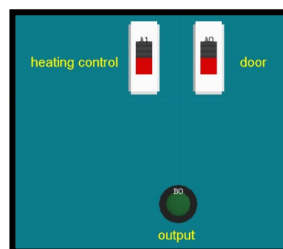
- **'door'** (to store the state of the door switch).
- **'control'** (to store the state of the on/off control switch)
- **'output'** (to control whether the microwave switches on or not)
- **'count'** (to monitor how many times the 1s delay has occurred. Give it an initial value of ten, so that the microwave oven will operate for 9s).
- Use the default configuration for the **loop** icon.
- Configure one input icon to store the state of the door switch (on port A bit 0) in the variable **'door'**.



- Configure the other **input** icon to store the state of the control switch (on port A bit 1) in the variable '**control**'.
- The upper calculation icon checks to see whether the door AND the control switch have been pressed.
Configure it using the equation **output = control & door**.
The **&** signifies the AND operation.
The result of this operation (0 or 1) is stored in the **variable 'output'**.
- The upper **decision** icon checks the value stored in '**output**'.
(**If output?** is shorthand for **If output=1?**)
Configure this **decision** icon.
- When the result of the **calculation** is 0, the program follows the 'No' route from the **decision** icon and the left-hand **output** icon is executed. This sends a logic 0 to the LED, ensuring that it (and the microwave generator) is switched off.

When the result of the calculation is 1, the program follows the 'Yes' route. The 'Turn on' **output** icon sends a logic 1 to the LED turning it on.
Configure both of these output icons.

- The lower **calculation** icon reduces the number stored in the variable '**count**' by one.
Configure it using the equation **count = count - 1**
- The initial value of 'count' is ten. Provided the number stored in 'count' has not reached zero, the program follows the 'No' route. Eventually, after looping enough times, the number stored does reduce to zero. The program then follows the 'Yes' route and executes the 'Turn off' output icon, which is configured in the same way as the other 'Turn off' icon, to switch off the microwave generator.



- Add a **switch array** to the System Panel. Configure it to have only two switches, one connected to port A, bit 0 and the other to port A, bit 1.
- Add an **LED** connected to port B, bit 0 to represent the microwave generator.
- Add labels to the System Panel to identify the components.
Position them using the **World coordinates** under the **Position** tab of the label properties.
- Now simulate the program step-by-step, using the F8 function key repeatedly.
- Check what happens for different combinations of switch states and interpret this in terms of the behaviour of the microwave oven. What happens, for example, if the door is opened while the microwave generator is operating?

For Arduino the Ports need to be set to PORTC and PORTD (equivalent to A and B on the Combo board).

Example 6. Binary logic in control. 6B Controlling the interior light in a car.

The interior light of a car can be controlled by another **Boolean logic** equation.

For simplicity, consider a two-door car with the following behaviour: The interior light turns on when one door (A) **OR** the other (B) is opened and stays on until the ignition switch (C) is turned on.

In Boolean-speak, we say that the light is on if (A OR B) **AND NOT** C is true.

Once again, we can build this condition into a Flowcode program.



Setting up the flowchart

Launch Flowcode and start a new flowchart.

Create the flowchart shown opposite, using:

- a **loop** icon.
- three **input** icons.
- two **output** icons.
- a **decision** icon.
- a **calculation** icon.

Create four **variables**:

- **door_A** (to store the state of the switch on door A).
- **door_B** (to store the state of the switch on door B).
- **ig_switch** (to store the state of the ignition switch).
- **output** (to control whether the interior light switches on or not).

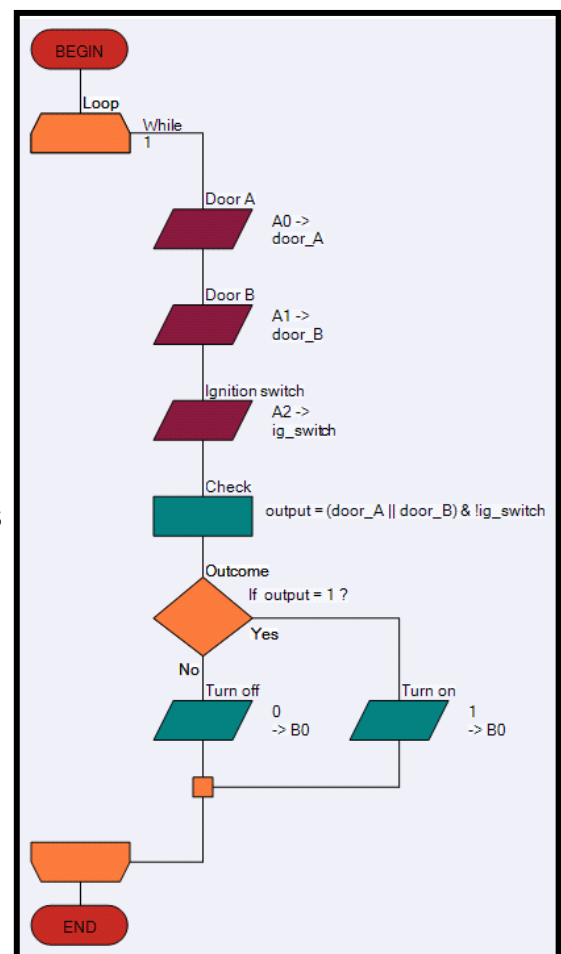
- Use the default configuration for the **loop** icon.
- Configure one **input** icon to store the state of the switch on door A (port A bit 0), in the **variable** 'door_A'.
- Configure one **input** icon to store the state of the switch on door B (port A bit 1) in the variable 'door_B'.
- Configure the other input icon to store the state of the ignition switch (port A bit 2) in the variable 'ig_switch'.

The calculation icon checks to see whether either door has been opened **AND** the ignition switch is **NOT** on.

- Configure it using the equation **output = (door_A || door_B) & !ig_switch**

The **||** signifies the **OR** operation and **!** the **NOT** operation. The result of the calculation is stored in the variable '**output**'.

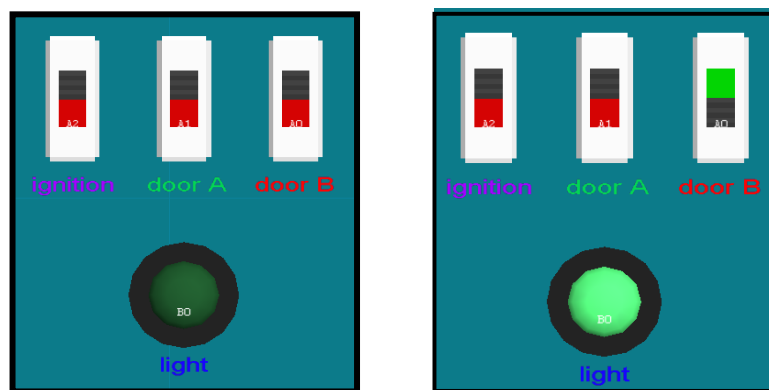
(For Arduino users, please use ports C and D as appropriate).



- The **decision** icon checks the value stored in '**output**'.
- Configure this **decision** icon.
- When the result of the **calculation** is 0, the program follows the 'No' route from the decision icon and the 'Turn Off' output icon is executed, ensuring that the light is switched off.
- When the result of the calculation is 1, the program follows the 'Yes' route. The 'Turn on' output icon sends a logic 1 to the LED turning it on.
- Configure both of these output icons.
- Add a **switch array** to the System Panel. Configure it to have three switches, one connected to port A, bit 0, one to portA, bit 1 and the other to port A, bit 2.
- Add an LED connected to port B, bit 0 to represent the interior light in the car.
- Add labels to the System Panel to identify the components and position them as shown in the diagram (**Component Libraries > Creation**)

Now simulate the program step-by-step, using the F8 function key repeatedly.

Check what happens for different combinations of open doors and ignition switch states. Interpret the behaviour in terms of the behaviour of the interior light. What happens, for example, if the door is opened and then closed shortly after? Is this behaviour correct?



Section 6: **Programming Exercises**

The **Programming Exercises** are presented here as flexible tasks suitable for further development.

Small, individual tasks can be developed into larger scale projects if desired. Try out the ideas, test them, experiment, develop your skills and see what you can create.

The aim of the exercises is to develop experience in using Flowcode and in the process, develop understanding of the programming terminology and techniques it embraces.

Programs can be tested by simulating them in Flowcode, but also downloaded to a microcontroller and tested on hardware. It is generally assumed that the programmer is using a Microchip PIC MCU though the exercises are equally applicable to other microcontrollers.

The section ends with further **Challenges**. These are even more open-ended and contain only a brief specification.

This exercise configures **Flowcode** to **output** specific digital signals to the **LED array**.

Background

- Section 1 - Introduction to microcontrollers.
- Section 2 - Using E-blocks.
- Section 4 - Flowcode First Project. Adding digital outputs - Light the LED
- Flowcode Wiki - Using masks.

Objectives

- Change the logic level of a one single pin of a port.
- Send different 8-bit codes to the port of a microcontroller.
- Configure an **Output** icon.
- Use **binary** code.
- Manipulate **logic** output levels.
- Use LEDs to display an **output**.
- Compile a program to a microcontroller.

Tasks

- 1 Create a Flowcode flowchart then see if you can:
 - a) add a single **Output** icon, configured to light all the LEDs of a port and run the simulation.
 - b) alter the parameters to light only the odd-numbered LEDs and run the simulation.
 - c) light only the even-numbered LEDs.
 - d) light only the high 'nibble' bits (4 to 7) of the chosen port.

Modify this program and see if you can:

- e) repeat the previous steps using **hexadecimal** rather than decimal numbering.
- f) only light the LED on bit 7, by sending an 8-bit value to the port.
- g) only light the LED on bit 7, using the '**single bit**' output method.
- h) only light the LED on bit 7, using the '**masking**' output method.

- 2 Write a program that uses at least twenty **Output** icons to write different values to port B, one after the other. Use all four methods in this exercise - hexadecimal, decimal, single bit and masking. Simulate the program and review the results.

(Save the program and download it to the microcontroller).

TIP: Restart the program a number of times by pressing the Reset button on the programmer board.

In this exercise, you learn how delays are used to slow down program execution. Microcontrollers work extremely quickly - typically executing about 5,000,000 assembly instructions, every second. A human can detect and understand only around three stable images per second. To allow a high-speed microcontroller to communicate with 'slow' humans, we sometimes need to slow it down by adding **Delay** instructions.

Background

- Section 1 - Introduction to microcontrollers.
- Section 2 - Using E-blocks.
- Flowcode Wiki - Loop icon properties.

Objectives

- Add a delay to slow down execution of a program.
- Change the delay interval.
- Configure a delay icon.
- Control the speed of a microcontroller.
- Use an **oscilloscope** to time events .

Tasks

1

Begin by opening the program created in the last exercise (*Exercise 1*).

- a) Add **Delay** icons and configure them so that the output states can be viewed comfortably even at 'HS oscillator' speed.
- b) Save the program and download it to the microcontroller, testing the program on the E-blocks boards.

Modify the length of the delays caused by the **Delay** icons.

- c) Start with a delay of 1s.
- d) Progressively reduce the delay until it is too fast for your eyes to detect the different output states.
- c) Download the program to the microcontroller every time and test it on E-blocks.
- d) Use an oscilloscope to measure the delays you set up in Flowcode.
- e) Make a detailed drawing of the oscilloscope image, complete with voltage and timing information and the delay time used in the Flowcode program.

TIP: Do not test this in simulation mode - simulation timing is not always accurate because it runs under a Windows operating system and not in 'real time'.

A **Connection Point**, or 'goto' instruction, is often used to create an infinite loop - to repeat a set of instructions over and over again (a better way to do this is to use a 'Loop' instruction). The advantage of a Connection Point is that it can be used to jump out of a loop to a certain location in the program. The idea of pulse-width modulation (PWM) is introduced as a means of controlling LED brightness.

Background

- Flowcode Wiki - Connection point icon properties.
- Section 1 - Introduction to microcontrollers.
- Section 2 - Using E-blocks.
- Section 4 - Flowcode First Project. Adding digital outputs - Light the LED.

Objectives

- Use **Connection Points** to introduce unconditional branching in a program.
- Introduce **PWM** as a means of controlling the brightness of LEDs.
- Create an **infinite loop**.
- Manipulate logic output levels.
- Use LEDs to display an output.

Tasks

- Write a program to see if you can:
 - Use **Delay**, **Output** and **Connection Point** icons to light the even and odd LEDs of an array alternately on and off. Use a 300ms interval between, in an **infinite loop**.
(Test the program at first 'step-by-step' and then continuously in the Flowcode simulator).
 - Use **Delay**, **Output** and **Connection Point** icons to flash the **high nibble** and **low nibble** LEDs alternately on and off, with a 300ms interval between, in an **infinite loop**.
 - use **Delay** and **Output** icons to flash all the LEDs of the array on and off with a 500ms interval in between, in an **infinite loop**.
 - Modify the program by changing the 'on' and 'off' times in such a way that the total ('on' + 'off') time is unchanged, e.g. on for 12ms and off for 8ms. What is the difference?
(Download programs to the microcontroller and test them).

TIP: Make the last delays very short and make the on and off times asymmetrical, (e.g. on for 8ms and off for 12ms).

This is a software PWM generator. When you run it, the intensity of the LEDs is lower. They flash on and off too fast for our eyes to observe. Instead, we see the intensity change.

- Write a program that:
 - Lights LEDs on the four most-significant bits (MSB), of an array and keeps them on.
 - Dims the intensity of the LEDs on the four least-significant bits (LSB), compared to the four MSB LEDs, using PWM.
 - Use an oscilloscope to examine the signal controlling one of the four LSB LEDs.

TIP: The MSB is the left-most bit and the LSB is the right-most bit.

Modern microcontrollers, like the PIC MCU or Arduino, are able to do simple mathematical tasks with 8-bit numbers at very high speed. As the calculations get more complex or the numbers rise above an 8-bit value, then the execution time lengthens dramatically. Flowcode allows complex calculations using up to 16-bit numbers and takes care of all the complexities. However, these may slow down execution of the program.

Background

- Variables - Example 1. Adding digital inputs - Where's the fire?
- Flowcode Wiki - Creating variables.
- Digital inputs - Example 1. Adding digital inputs - Where's the fire?
- Flowcode Wiki - Calculation icon properties.
- Section 1 - Introduction to microcontrollers.
- Section 2 - Using E-blocks.

Objectives

- Create and use a **variable**.
- Configure a **calculation** icon to perform arithmetic and logic calculations.
- Create and manipulate variables.
- Perform calculations.
- Use LEDs with current limiting resistors.

Tasks

- 1** Create a flowchart that:
 - a) uses a **variable** called 'counter' containing an initial value of '1'.
 - b) displays the value stored in the variable 'counter' on LEDs.
(simulate the program to test that it works).Modify your program by:
 - c) adding a **Calculation** icon to double the value stored in the variable 'counter';
 - d) displaying this new value on LEDs.
 - e) using an infinite loop to repeat these steps continuously with a 300ms delay between them.
What do you see? (This is called a 'running light').
 - f) replacing the 'multiply by 2' with 'counter = counter + 1'. What do you see now?
(You just programmed a binary counter).
- 2** Modify your program to display the result of the following calculations on the LEDs of port B:
 - a) $45 + 52$;
 - b) $45 \text{ AND } 52$;
 - c) $45 \text{ OR } 52$;
 - d) $\text{NOT } 45$;
 - e) $(1+3)*(6/2)$;
 - f) $\text{VAR2} = \text{VAR1} * 3$ (where variable 'VAR1' stores the number 18).
(On paper, check if the results are correct).

Repeating a set of instructions, for an exact number of times, **WHILE** or **UNTIL** a condition is met is one of the most powerful programming operations.

TIP: The slow simulation or 'Step Over' function in the Flowcode simulator is useful to debug complex programs.



Background

- Flowcode Wiki - Loop icon properties.
- Flowcode Wiki - Connection point icon properties.
- Flowcode Wiki - Creating variables.
- Section 1 - Introduction to microcontrollers.
- Section 2 - Using E-blocks.
- Section 4 - Flowcode First Project. Adding digital outputs - Light the LED.

Objectives

- Create and use a 'running light' program, using the 'multiply-by-two' method.
- Create and use a 'running light' program, using the 'shift-right' method.
- Create and populate an array.
- Create a conditional **loop**.

Tasks

- Write a program to:
 - make an 8-bit binary counter, using a **Loop** icon, to count **UP** from 0 to 255, then reset and repeat the count (display the counter value on the LEDs of port B).
 Modify your program to:
 - make the counter count **UP** from 0 to 255 and then count back **DOWN** to 0.

TIP: Use two loops inside an infinite loop so that the process repeats indefinitely.
(Download the program to the microcontroller and test it at full speed).

- Do you know KITT From Knight Rider or the Cylon robots from Battlestar Galactica? Write a program to make a simple 'running light' that runs from port B, bit 0 to port B bit 7 and then back to port B bit 0, repeatedly.
 - Try using the 'multiply-by-two' method.
 - Try using the 'shift right' method.
 Modify your program to create a 16-bit running light, using the LEDs from port A and B.
TIP: Use only loops, no decisions.
 (Download the program to the microcontroller and test it).

- Create a flowchart that contains an array of four variables, called 'Matrix[x]' which stores the following values: **Matrix[0]=129 Matrix[1]=66 Matrix[2]=36 Matrix[3]=24** (Display the outputs on the LEDs of port B).
 - Use two 'do-while' loops to create an infinite sequence:
Matrix[0]-Matrix[1]-Matrix[2]-Matrix[3]-Matrix[2]-Matrix[1]-Matrix[0]-Matrix[1]-..... ;
 - Refer to the four variables as '**Matrix[x]**' where 'x' is a separate variable, known as the index of the array.
(Download the program to the microcontroller and test it).

Adding digital **inputs** to a microcontroller circuit is quite easy but is a big step forward. This allows external signals to influence how the program reacts.

Background

- Section 1 - Introduction to microcontrollers.
- Section 2 - Using E-blocks.
- Section 4 - Flowcode First Project. Adding digital outputs - Light the LED.

Objectives

- Input data from **switches**.
- Use loops to create LED sequences.
- Configure an **input icon**.

Tasks

- 1** Write a program to show the status of the switches connected to a chosen port, on the LEDs connected to a different port. eg. when a switch is pressed connected to port A, the corresponding LED on port B lights.

Modify the program so that:

- a) the LED stays lit for 2s.
- b) when switch '0' is pressed, LED 1 is lit.
- b) when switch '1' is pressed, LED 2 is lit and so on.
- c) when switch '7' is pressed, nothing happens.

Explore as many combinations as you can.

(Download programs to the microcontroller and test them).

- 2** Write a program to create a counter that:
- a) contains two loops.
 - b) counts up when switch '0' is pressed.
 - c) counts down when switch '1' is pressed.
 - d) displays the count on the LED array of a suitable port.
- (Download programs to the microcontroller and test them).

- 3** Write a 'running light' program that:
- a) contains two loops.
 - b) causes the LEDs to 'run' left when switch '0' is pressed.
 - c) causes the LEDs to 'run' right when switch '1' is pressed.
 - d) displays the count on the LED array of a suitable port.
- (Download programs to the microcontroller and test them).

Earlier programs included simple decision-making, using loops and connection points. Now we look in detail at the **Decision** icon, widely known as the 'if...then...else' structure, probably the most widely used command line in any program.

Background

- Flowcode Wiki - Decision icon properties.
- Flowcode Wiki - Connection point icon properties.
- Section 1 - Introduction to microcontrollers.
- Section 2 - Using E-blocks.

Objectives

- Configure **Decision** icons and hence add conditional branching to a program.
- Control the **frequency** at which LEDs flash.
- Use LEDs to display output logic levels.
- Use temporary memory.

Tasks

- 1** Write a program that uses switches to produce a reversed sequence on the LEDs.
 - a) when switch '0' is pressed, LED 7 lights up.
 - b) when switch '1' is pressed, LED 6 lights up.and so on...
- 2** Write a program that creates an 8-bit counter, counting from '0' to '255' and then back to '0' repeatedly.
 - a) Use **Decision** icons instead of Loop icons.
 - b) Use two switches connected to a chosen port, bits 0 and 1.
 - c) Count up when switch '0' is pressed.
 - d) Count down when switch '1' is pressed.
 - e) Display the current count on the LEDs connected to a suitable port.(Download the program to the microcontroller and test it).
- 3** Write a program that counts from 0 to a value stored in a **variable** called 'count' when switch '0' is pressed and then waits until switch '1' is pressed before counting down to 0.
 - a) Use two switches connected to a chosen port, bits 0 and 1.
 - b) Use a different port for the LED array to display the current value of the count.(Download the program to the microcontroller and test it).
- 4** Write a program that makes eight LEDs flash on and off at a frequency of 1Hz, i.e. taking one second for an 'on-and-off' cycle. Use two switches connected to a suitable port, bits 0 and 1.
 - a) The LEDs should flash faster if switch '0' is pressed.
 - b) The LEDs should flash more slowly if switch '1' is pressed.(Download the program to the microcontroller and test it).

- 5** Write a program that makes all eight LEDs in an array light when a switch is pressed the first time and all go off when it is pressed again.

(Download the program to the microcontroller and test it).

- 6** A car has two interior lights. One is in the front of the car and one is in the rear. Write a program to simulate this scenario using LEDs and five switches to control them.

- Use switches 0, 1, 2, 3 to represent doors being open or closed.
- Use switch 4 to represent the boot (trunk) being open or closed.
- Light both LEDs when any door opens.
- Light only the 'rear' LED when the boot (trunk) is opened.

(Download the program to the microcontroller and test it).

**TIP: Assume that the switches are closed when the doors are open.
This may be easier to simulate with 'push-to-make' switches.**

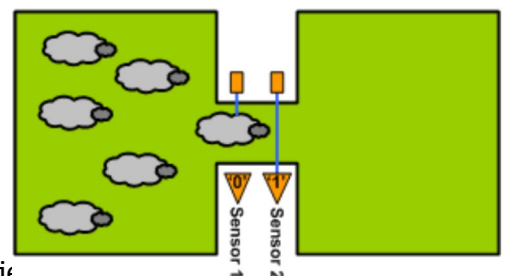
- 7** A car's steering wheel has switches on it that control the external lights. Write a program to simulate the control of the lights.

- Use a switch to control the left direction-indicator (choose a relevant LED), which flashes on for 250ms and then off for 250ms repeatedly until the switch is released.
- Use another switch to control the right direction-indicator (choose a relevant LED), in the same way.
- Use two LEDs as brake lights controlled by a switch which light up for as long as it's pressed.
- Create headlights which light when a switch is pressed and stay on until it is pressed again.
- Finish off with a pair of foglights in the same way.

**TIP: Don't attempt to write this program all at once. Divide it into subsections and solve each separately before putting them all together.
To make it easier, use the labelling feature of Flowcode to label switches and LEDs.**

- 8** Six sheep are allowed to wander between two fields. There are two sensors between the fields. Write a program that counts and displays the number of sheep in each field. Simulate this scenario using two switches to represent the sensors.

- Show the results in binary form on the LED array (use four LEDs for the west field and four for the east field).
- Use two switches to represent the sensors.



TIP: Assume that each sheep is longer than the gap between the sensors.

Think about the various scenarios that could happen. A sheep might trigger a sensor and then back out. Can a sheep trigger both sensors and then back out? When does a sheep count as being in the east field?

Using LEDs to display outputs can be limiting.

The LCD is an alternative way to display data, both letters and numbers, for 'non binary' humans.

Background

- Section 1 - Introduction to microcontrollers.
- Section 2 - Using E-blocks.
- Example 3. The LCD display - Posting messages

Objectives

- Create, populate and manipulate **string variables**.
- Control the display of text and numbers on an **LCD**.
- Use an LCD as an output device for the microcontroller.
- Configure a **Component macro** for the **LCD**.

Tasks

- 1 Write a program that displays the text "Hello World" in the centre of the bottom line of the **LCD**.
- 2 Write a program that shows an increasing count (decimal) on the LCD screen.
Modify the program so that it counts up when a switch is pressed and counts down when a different switch is pressed (use **Loops** or **Decisions**).
- 3 Write a program to show the status of the switches attached to the first port. Every time a switch is pressed, the corresponding LED of the second port lights up and the value of the **decimal** equivalent is displayed on the LCD.
- 4 Write a program to show the status of the switches attached to the first port on the LEDs of the second port and on the top line of the LCD and then:
 - a) multiply this **binary** number by 100.
 - b) display the result on the bottom line of the LCD, with "[x 100 =]" displayed in front of it.
- 5 Write a program that scrolls the lines of text given below, one line at a time. Initially, the text is centred on the bottom line of the display for 2s. Then it moves up to be centred on the top line for 2s, to be replaced on the bottom line by the next line of text, and so on.

Text:

"There are only"
"10 kinds"
"of people"
"Those who"
"understand"
"BINARY"
"and those who"
"DON'T."

(Enclose the program in an **infinite loop** and test on the **LCD**).

A numeric **keypad** is used in many electronic devices, and in some (eg. mobile phone), it is used as a numeric keypad and also as a way to type text instead of numbers. There are twelve buttons on the keypad, yet the keypad is connected to the microcontroller by only eight lines. This problem is solved by using **multiplexing**.

Background

- LCD - Exercise 8 - Programming LCDs.
- Flowcode Wiki - String manipulation functions.
- Section 1 - Introduction to microcontrollers.
- Section 2 - Using E-blocks.

Objectives

- Input text and numbers from a keypad and display messages on the LCD.
- Use **ASCII code** to transmit this data.
- Use **multiplexed inputs**.
- Configure a **Component macro** for the **keypad**.

Tasks

- Display numbers that are pressed on the **keypad** on the **LCD**.

 - Display one number one at a time for as long as the button on the keypad is pressed.
 - Can you re-write this program without using the Keypad **Component macro**?
 - Extend this program to display numbers that are pressed on the keypad one after another on the top row of the LCD.

See if you can refine the program to:

 - clear the display when '#' is pressed.
 - display a maximum of fifteen characters and display a warning on the bottom row of the LCD when this maximum is exceeded.
- Write a program to:

 - add together two numbers less than '9999' entered via the **keypad**.
 - Display the two numbers, the '+' and '=' and the resulting sum on the top row of the LCD.
 - Display a warning on the bottom row when '9999' is exceeded.
- Write a program for a simple guessing game, where:

 - a player needs to guess a number between '0' and '9'.
 - the secret number is pre-programmed into the microcontroller.
 - the LCD displays, on the top row, the latest guess entered via the **keypad**.
 - the LCD displays a message, on the bottom row, indicating whether the guess is too high or too low.

Extend this program so that the secret number is in the range '0' to '255'.

Extend the program again so that the secret number is in the range '0' to '9999'.
- Write a program to use the **keypad**, as on a mobile phone, to input text to the microcontroller.

 - Use **ASCII code** to transmit the data.
 - Use the character '*' for a space.
 - Clear the display when '#' is pressed.
 - Display a message on the bottom row when the text has more than ten characters.

The 16F18877 PIC MCU accepts 35 separate analogue inputs. Newer devices may have even more. An **analogue** signal on one of these inputs can be translated into a 10-bit digital binary number. We can choose to use only the eight most-significant-bits of this 10-bit number or to use the full 10-bit number. Be aware that working with 10-bit numbers in an 8-bit microcontroller like the PIC MCU, needs careful program writing.

Background

- LCD - Exercise 8 - Programming LCDs
- Flowcode Wiki - String manipulation functions
- Section 1 - Introduction to microcontrollers
- Section 2 - Using E-blocks

Objectives

- Create data loggers, using 8-bit and 10-bit data from the ADC.
- Configure an analogue input.
- Enter data via switches.
- Enter information from light and temperature sensors.
- Configure and use the **EEPROM**.
- Scroll through **EEPROM** data.
- Display text and numerical data on the LCD.
- Use the E-blocks prototype board.

Tasks

- 1 Write a program to display an 8-bit number, equivalent to the **analogue** input voltage from the light sensor on the Sensor board. Try connecting a voltmeter to measure the **analogue** input voltage. (Save the following programs and download them to the microcontroller for testing).
- 2 Modify the program from Task 1 to display data from the '**pot**' on the Sensor board. Try to convert the ADC 8-bit output into a voltage reading between 0 and 5V, making it as accurate as the 8-bit mode allows. Use a **voltmeter** to measure the analogue input voltage.
- 3 Modify program 2 to display, on the LCD, a 10-bit number equivalent to the **analogue** input voltage from the '**pot**' on the Sensor board. Use a voltmeter to measure the analogue input voltage. Try to convert the ADC 10-bit output into a voltage reading between 0 and 5V, making it as accurate as the 10-bit mode allows. Use a **voltmeter** to measure the **analogue** input voltage.
- 4 Write a program to monitor the lighting in a room over a 24 hour period:
 - using the analogue signal from the light sensor on the Sensor board
 - storing light measurements on the **EEPROM**.
 - sampling at the highest rate possible, given that the PIC MCU has 256 bytes of **EEPROM** memory on board.
 - and displaying each sample with its sample number, on the LCD.
 - by scrolling forwards through the samples by pressing switch '0' or scrolling backwards by pressing switch '1'.

TIP: Increase sampling rate so that you don't have to spend 24 hours in testing.

In code-based programming languages, like **C** and **BASIC**, a **software macro** would be called a **subroutine** or **function** or **procedure**. As programs get bigger, they use certain combinations of instructions over and over again. These programs become harder to understand and read. Routines that are re-used can be put into a **software macro**, which can be called whenever it is needed in the main program. Making use of these software macros lightens up the main program and makes it much easier to read.

Background

- Flowcode Wiki - **Software macro** icon properties
- Section 2 - Using E-blocks

Objectives

- Use software macros to simplify the structure of a program.
- Create **software macros**.
- Use closed loop control.
- Use PWM to control the brightness of LEDs.

Tasks

1

Write a program that selects and runs one of three different programs by using two switches.

- a) switch '0' selects one of three programs (which you developed earlier).
 - 'X': an 8-bit binary up-counter, displayed on the LEDs.
 - 'Y': an 8-bit binary down-counter, displayed on the LEDs.
 - 'Z': an 8-bit bidirectional 'running light', displayed on the LEDs.
- b) the LCD displays a text message identifying the selected program.
- c) switch '1' activates the chosen program when pressed.
- d) the three programs are placed in **software macros**.

Modify program 1 so that:

- e) If switch '0' is pressed while one of the three software is running, execution stops immediately and focus returns to the main loop and waits for a new selection.

Modify program 1 again so that:

- f) if switch '0' is pressed while one of the three software is running, execution stops and returns to the main loop, as before, but it stores the value displayed on the LEDs.
- g) when the next selection is made, that macro starts the LEDs from where the previous one left off, making the transition between them smoother.

(Download programs to the microcontroller and test them).

In earlier exercises, the microcontroller did not necessarily react to inputs straight away because it was busy doing something else. The external interrupt features of microcontrollers solve this problem. On a 16F18877, the external interrupts are on pin 'RB0' - a single pin interrupt and on port B as an 'interrupt on change (IOC)'. If these interrupts are initialized correctly, then a change on port B can cause the program to stop execution immediately and switch to executing the appropriate interrupt macro. We then have what is called a 'real time' execution.

Background

- Section 2 - Using E-blocks.
- Flowcode Wiki.

Objectives

- Create and use **single-pin interrupts**.
- Create and use **interrupt-on-change (IOC) interrupts**.
- Use real time operation of a microcontroller.

Tasks

- 1 Write a program to time how many seconds have passed since a program was reset and displays the result on an LCD. Use a variable called **count** whose value is displayed on the LEDs (don't use an interrupt). Use a 1s delay. A rising edge on pin RB0 should call a macro that adds one to **count**.

Re-design this program using an **interrupt** (single-pin) on RB0.

Now re-design it using both kinds of **external interrupt** so that:

- a) triggering the single-pin interrupt increments '**count**' ($\text{count} = \text{count} + 1$)
- b) triggering the IOC interrupt decrements '**count**' ($\text{count} = \text{count} - 1$)

- 2 Write a program to make an electronic dice that:

- a) counts from 1 to 12.
- b) display the result on the LCD.
- c) starts 'rolling' when switch 0 is pressed.
- d) stops 'rolling' when switch 0 is pressed again.

TIP: The LCD should display numbers from 1 to 12, one after the other, over and over again rapidly, at 20 ms intervals (much too fast to see with a human eye).

Modify this program so that:

- e) the dice keeps 'rolling' as long as switch 0 is held down.
- f) stops 'rolling' when the switch is released.
- g) at that point displays the number on the LCD.

- 3 Write a program to make a reaction timer that :

- a) lights all LEDs initially.
- b) keeps them lit for around 6s.
- c) switches them off and starts a timer.
- d) stops the timer when the player presses switch 0.
- e) then displays the resulting 'reaction time' on the LCD.
(Use a variable that is incremented every 10ms.)

Modify program 3 to limit the time allowed to the size of the used variable and displays a message on the LCD when this size is exceeded. (Include a trap to prevent cheating by simply holding down switch 0 continuously).

The other type of interrupt function in Flowcode is the timer interrupt. These allow you to perform software tasks at precisely predetermined time intervals - a really useful feature when developing time critical applications and clocks.

Background

- Flowcode Wiki - What is a 7-segment display?
- Section 2 - Using E-blocks.

Objectives

- Create and use timer interrupt.
- Use the prescaler to create accurate time intervals.
- Trigger the timer using the crystal or an external event.

Timer arithmetic:

The 16F18877 has several timers, but we look at only two: **TMR0** (Timer 0) and **TMR1** (Timer 1).

- **TMR0** can be triggered by the crystal or by a transition on the **T0CKI** pin **RA4**.
- The internal clock has a frequency of crystal clock frequency/4, i.e. $19,660,800/4 = 4,915,200\text{Hz}$.
- The **TMR0** prescaler can be set from 1:2 to 1:256. For this exercise, set it to 1:256, so that every 256 clock pulses cause the TMR0 to increase by 1. This happens at a frequency of $4.915.200/256 = 19.200\text{Hz}$.
- Every time this 8-bit timer 'overflows' (reaches 256), it generates an interrupt. This happens with a frequency of $19.200/256 = 75\text{Hz}$, so that the main program is stopped 75 times per second and so the **timer interrupt** macro is executed 75 times per second.
- Instead of using the crystal, this timer can also be 'clocked' by an external event, as when measuring motor speed etc.
- **TMR1** can be triggered by the crystal oscillator or by a transition on the **T1CKI** pin **RC0**. (Its operation is similar to that of TMR0, except that it uses different prescaler values).

Tasks

- 1 Write a program to produce a precise 'seconds' timer that displays the result on the LCD and starts when the microcontroller is reset. Use a 1s delay. Don't use a timer interrupt.
(Download this program to the microcontroller and test it using your watch).
Rewrite the program using a timer interrupt.
- 2 Write a program to create a basketball timer that starts when switch 0 is pressed and displays the time elapsed on the LCD. Make the LEDs flash on and off when 30s has elapsed (the time allowed for the team with the ball to make a goal attempt).
TIP: Use a **single-bit** interrupt on pin **RB0** to start the timing.)
- 3 Write a program to produce a precise clock that displays the time elapsed since the last reset, in hours, minutes and seconds on the LCD (test with a watch).
Modify this program so that:
 - a) switch '0' stops the clock when pressed the first time.
 - b) switches '1', '2' and '3' can be used to change the displayed time to the actual time.
 - c) switch '0' restarts the clock when pressed a second time.
- 4 Write a program to produce a timer that counts down from 01:00:00 to 00:00:00 in seconds and then lights all the LEDs.
(Download to the microcontroller and test it with your watch).

- 1** Develop a dimmer for all the LEDs that reacts to measured light intensity.

 - The light sensor monitors the light intensity in a room. When this intensity drops, a control circuit sends more power to the lights in that room and vice versa when the intensity increases.
 - Use the LEDs to simulate the dimmed lights. This is done by programming a **software PWM** output to all LEDs on port B. When the PWM 'on' time increases, the LEDs get brighter, etc.
 - Put this program in a **timer interrupt** macro.
 - The main **loop** monitors the **analogue** input from the **light sensor** on the **sensor board**, on port C. When the light sensor detects less light, the LEDs need to shine brighter. The opposite should happen with the LEDs when the light level, measured by the light sensor, intensifies.
 - The period of the PWM signal stays a constant 20ms at all times, set using a timer interrupt.
 - Download this program to the microcontroller and test it. If you have a **2 channel oscilloscope**, measure the **analogue input** of the light sensor on one and the PWM output to one of the LEDs on the other.
 - Using a similar approach, develop a temperature controller for an incubator. The BL0129 **Grove Sensor board** can be used with the **Grove Temperature sensor** module. Use the LEDs to simulate the action of a heater.
- 2** Three judges vote on variety acts in a X-factor-like game show. When two or more judges vote 'Yes', the act progresses to the next round.

 - Design a program to combine the judges' votes into a pass/fail verdict.
 - Create two LED light sequences, one to indicate pass and the other fail.
- 3** Design an automatic watering system for a sealed terrarium (glass plant container). **Use the Grove Temperature and Humidity sensor** module to sense when the terrarium needs watering.

 - The output device is a motor-driven pump that runs for a set period of time once triggered. There should be a 'rest' period after watering before the system can operate again.
- 4** Create a combination lock, using the BL0138 **Keypad board** to input a four-digit 'PIN'.

 - Add a feature that 'locks out' a user after three unsuccessful attempts.
 - Modify it to prevent further access to the system for a period of time such as ten seconds.
 - Use the LCD display to show the numbers selected on the **keypad** and the number of attempts made.
- 5** Develop a proximity switch for a security light using the **Grove Ultrasonic Ranger sensor** module. The system should switch on four lights (12V lamps) when a person approaches within one metre of the sensor and so makes use of the BL083 **Relay board**.
- 6** Use the **Grove Infrared Receiver sensor** module to time the swing of a pendulum without impeding it.
- 7** Design a system to drive the **DC motor** (and sensor) on the **Actuators** training panel at a steady speed.

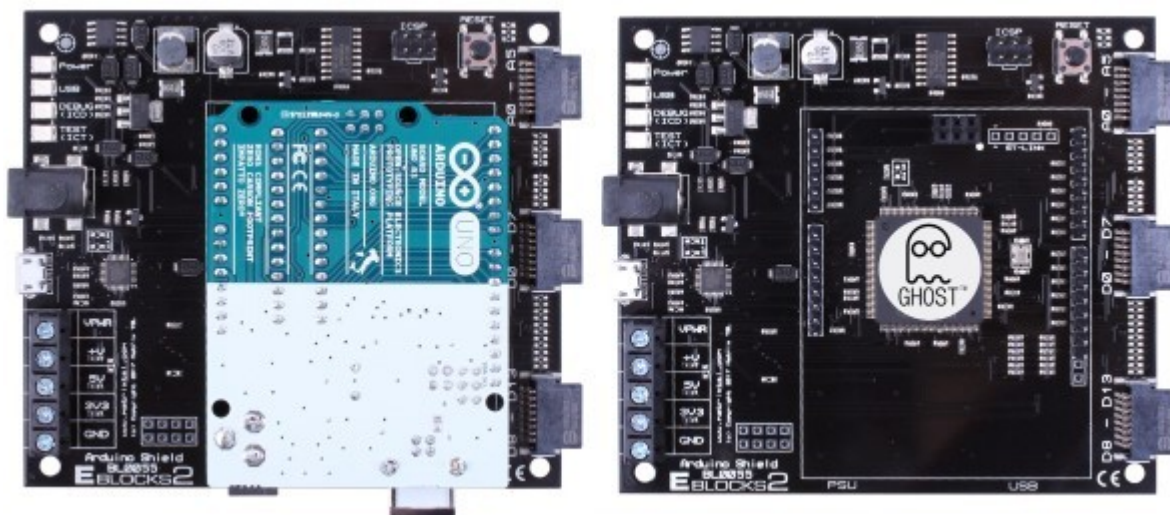
 - Add a feature to modify this set speed.
- 8** Design a system to drive the **stepper motor** on the **Actuators** training panel so that it rotates, in $1\frac{1}{2}$ steps, through one complete circle and then reverses back to its initial position in the same manner.

Appendix 1: **Arduino Adjustments**

ARDUINO: SECTION A

BL0055 Arduino Shield

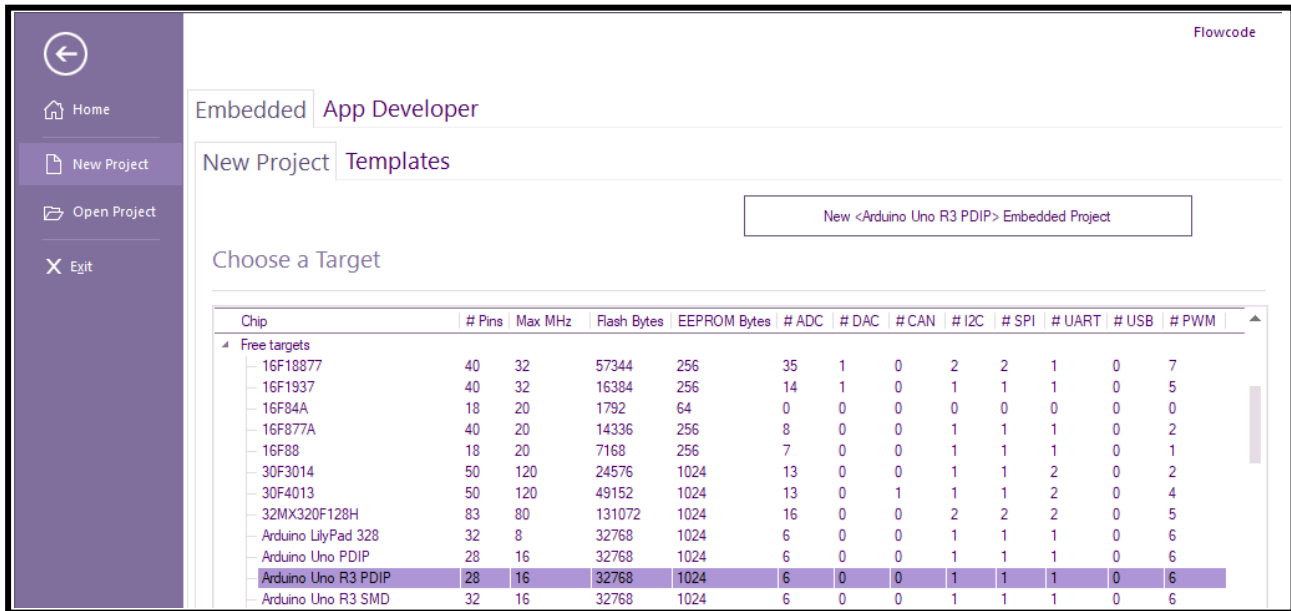
- The board has three ports, labelled **A0-A5**, **D0-D7** and **D8-D13**.
- Port **D0-D7** offers full 8-bit functionality.
- Port **A0-A5** and **D8-D13** has 6-bit functionality.
- It can be powered from an external power supply, delivering 7.5V to 9V or from a USB supply.
- If the Reset switch is pressed, the program stored in the Arduino will restart.
- The board is USB programmable via a programming chip. This takes care of communication between **Flowcode** and the **Arduino** device.
- The Arduino executes one instruction for every clock pulse it receives.
- (Note - a single instruction is NOT the same as a single Flowcode symbol, which is compiled into C and then into Assembly and probably results in a number of instructions).
- This device uses a **16MHz crystal**.
- The board will detect whether External power supply or USB power supply should be used.
- Use of the AVR ISP tool from Microchip via the ICSP header.
- Usually supplied with an Arduino Uno device.
- Provides power to the downstream E-blocks boards via the port connectors.
- Contains the Matrix Ghost chip which allows for real time in-circuit debugging and pin monitoring when combined with Flowcode.



ARDUINO: SECTION B

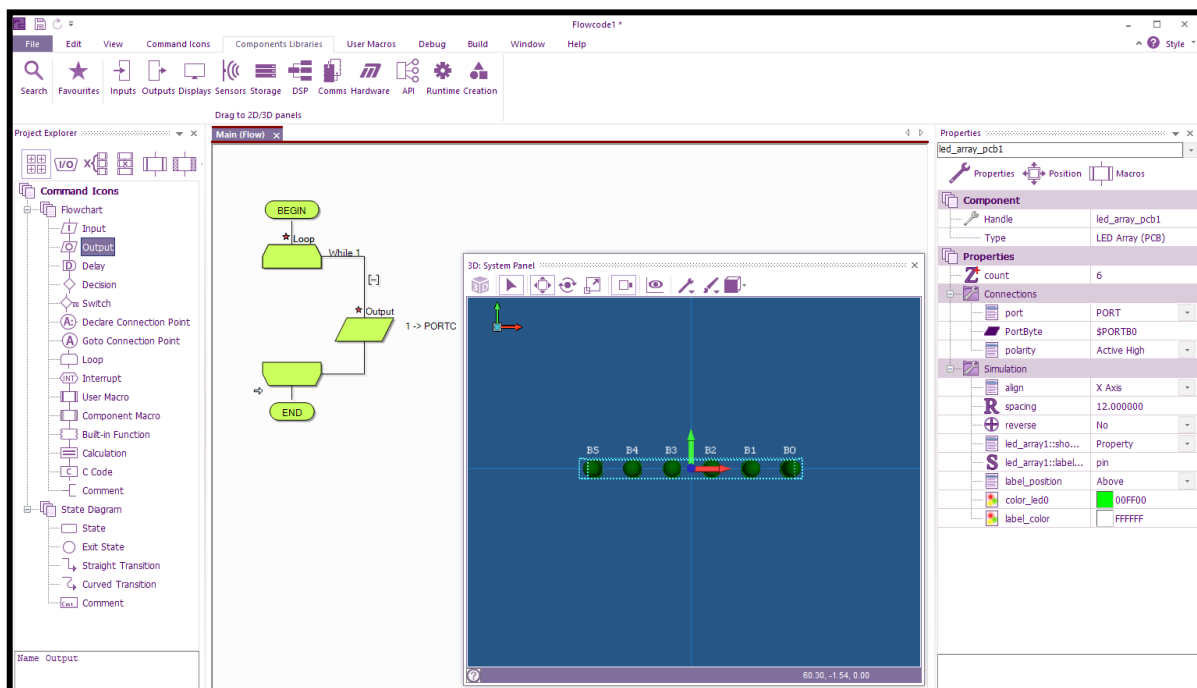
Selecting Arduino in Flowcode

On opening Flowcode, you are presented with the 'Welcome' screen. Click on **New Project**.



Select **Arduino Uno R3 PDIP** from the Free targets list. Click **"New <Arduino..."** button above

This brings up the standard Flowcode environment. A flowchart can now be developed into a program that can be tested within the Flowcode simulation mode, or saved and compiled to the Arduino board.



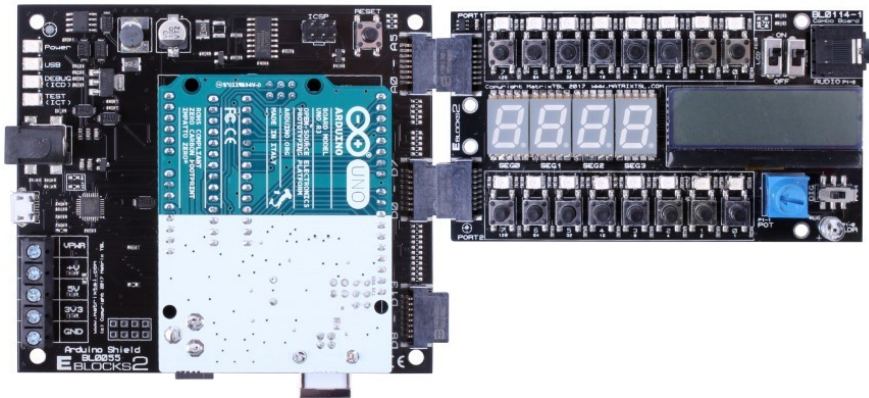
Follow the Examples and Exercises, taking Port changes into consideration where required. E.g. Above is how **Flowcode First Program** (Page 42) would look to an Arduino user.

Here, Arduino users are using **PORTC** instead of **PORTA**.

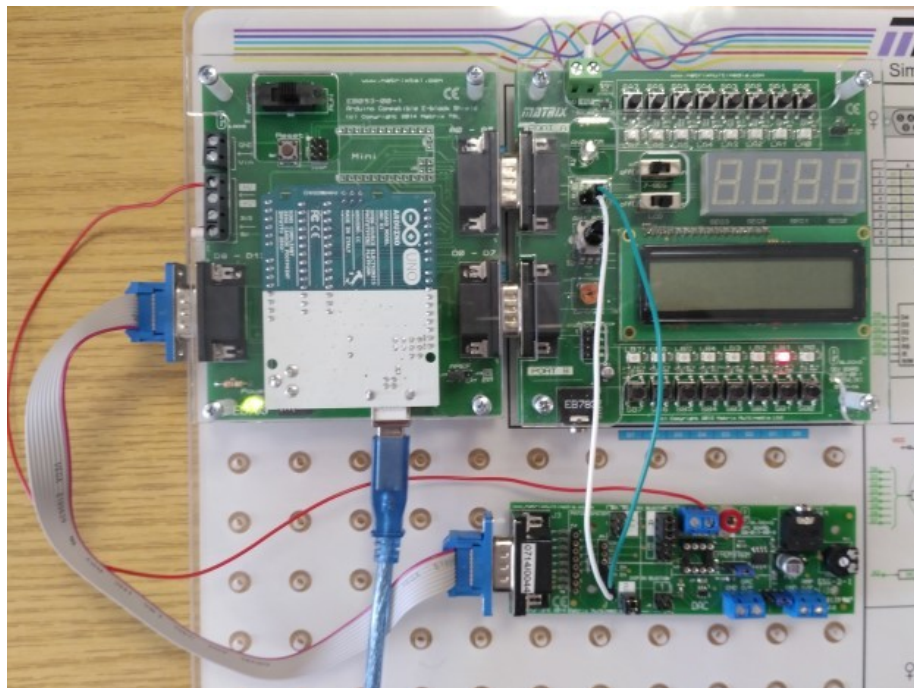
(PORTC on the Arduino 'Maps' to PORTA of the Combo board)

ARDUINO: SECTION C**E-blocks2:**

Eblocks2 uses the 'Click' boards for its SPI connections. Using the BL0106 'Click' board E-block, you can put the board into the (D8-D13) port as shown in the picture below:

**E-blocks1:****Using with the SPI E-block (EB013)**

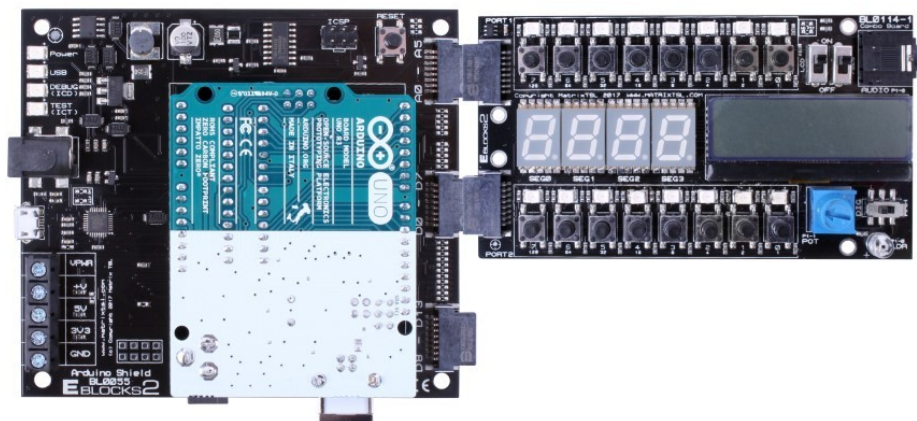
On the SPI E-block move the "CHIP EN SELECTION" jumper to setting "2" and connect DAC_EN and NVM_EN to the centre pins of the "PORT A MODE SELECT" jumper (J14) of the EB083 Combo Board:



ARDUINO: SECTION D

Setting up the hardware:

This diagram shows you how to set up the E-blocks hardware with Arduino. Plug your Arduino into the BL0055 board as shown, then the combo board into the ports labelled **(A0-A5)** and **(D0-D7)**.



Note: Despite having two hardware port connections between the EB0114 Development board and the BL0055 Shield, the Arduino Uno can only provide 6 general purpose I/O connections on port C, (A0-A5). Therefore, LEDs '6' and '7' and switches '6' and '7' on Port 1 of the Development board, cannot be used with the Arduino Uno.

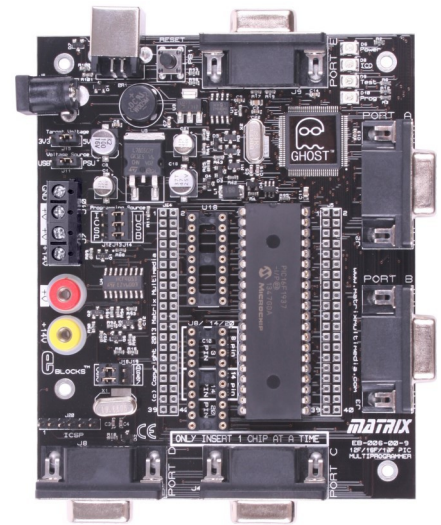
In order to program the Arduino Uno board directly from within Flowcode, you must ensure that the appropriate drivers are installed. We recommend you visit the Arduino site and download the latest drivers from there.

Appendix 2: **E-blocks1**

Adjustments

E-BLOCKS1: SECTION A**EB006 Multiprogrammer**

- The board has five ports, labelled 'A' to 'E'.
- Ports 'B', 'C' and 'D' offer full 8-bit functionality.
- Port 'A' has 6-bit functionality, (8-bit if the internal oscillator is selected).
- Port 'E' has 3-bit functionality.
- It can be powered from an external power supply, delivering 7.5V to 9V or from a USB supply.
- As pointed out elsewhere, no 5V connection is provided in the D-type connector. This must be provided by an additional connection to 'downstream' boards, using the screw terminals on the Multiprogrammer board. The ground connection is provided through pin 9 of the D-type connector.



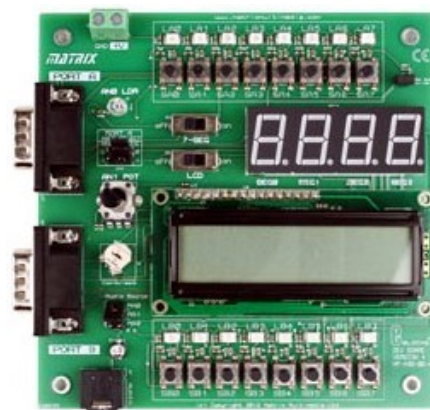
IMPORTANT - DO NOT connect the 14V screw terminal to a 5V 'downstream' screw terminal.

- If the Reset switch is pressed, the program stored in the PIC MCU will restart.
- There is a 1:1 mapping between pins on the D-type connector and those on the ports, (e.g. Pin1 is connected to PB0, Pin 2 to PB1 etc.)
- The board is USB programmable via a programming chip. This takes care of communication between Flowcode' and the PIC MCU.
- The PIC MCU executes one instruction for every four clock pulses it receives.
- (Note - a single instruction is NOT the same as a single Flowcode symbol, which is compiled into C and then into Assembly and probably results in a number of instructions.)
- This course uses a 19,660,800Hz crystal. The advantages of this frequency include:
 - standard Baud rate (19200) is obtainable by dividing it by 1024;
 - it can be further divided by 256 to give 75 Hz.
- Jumpers allow the user to select a number of options:
 - external power supply or USB power supply;
 - where the PIC MCU uses an internal oscillator, all eight bits of port A can be used for I/O operation;
 - use of an professional ICD2 (In-Circuit Debugger) tool from Microchip.

E-BLOCKS1: SECTION B**EB083 Combo Board**

The board combines together on one compact board the functionality found on a number of individual E-blocks boards:

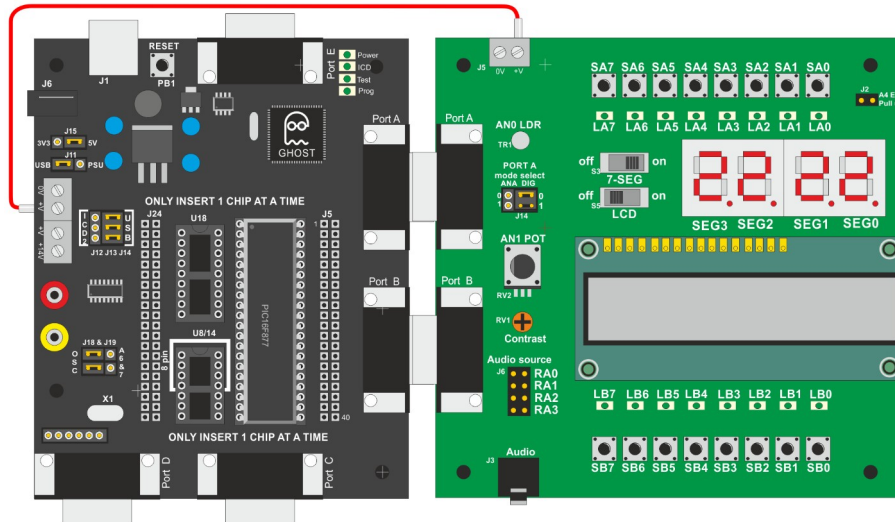
- EB004 LED board (x2)
- EB005 LCD board;
- EB007 Switch board (x2)
- EB008 Quad 7-segment Display board.
- For this course, the D-type connectors attach to female connectors on ports A and B of the Multiprogrammer.
- The board provides a set of eight switches and eight LEDs for port A and the same for port B.
- With J14 links in the 'Digital' position, port A is routed to its push switches (SA0 to SA7), to LEDs (LA0 to LA7) and to the quad 7-segment display.
- With J14 links in the 'Analogue' position, port A is switched to the analogue sensor section of the board, so that pin RA0 is connected to the on-board light sensor and pin RA1 is connected to the potentiometer to give a variable output voltage, (simulating the action of an analogue sensing subsystem).
- (With these links in this position, the on-board switches and LEDs LA0 and LA1 will not operate.)
- Port B I/O pins are routed to its push switches (SB0 to SB7), to the LEDs (LB0 to LB7), to the quad 7-segment displays and to the LCD display.
- The quad 7-segment display is turned on by switch S3. It is connected to both port A and B.
 - Port B is used to control the LED segments and the decimal point).
 - Port A, bits 0 to 3, select which display is activated.
- The LCD is a 16 character x 2 lines module, turned on by switch S5. Normally a complex device to program, Flowcode takes care of the complexities, unseen by the user.



E-BLOCKS1: SECTION C

Setting up the hardware:

This diagram shows you how to set up the E-blocks hardware.



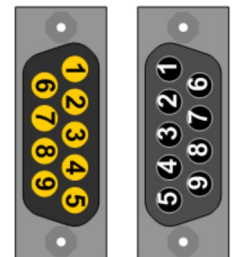
How E-blocks connect to each other

E-blocks are built on a bus-based concept. Each E-block D-type connector uses nine pins, allowing transfer of eight bits of data and 'ground'.

In the E-blocks system, some E-blocks are 'upstream' and some are 'downstream'. 'Upstream' devices have 'intelligence', are usually programmable and control the flow of information to 'downstream' devices. In the E-blocks system, 'upstream' devices connect using 9-way D-type **sockets**, whereas 'downstream' devices connect using 9-way D-type **plugs**.

The diagrams show how the pins are numbered on the plugs and sockets. On both plugs and sockets, bit 0 is delivered on pin 1, bit 7 on pin 8 and pin 9 is designated 0V.

Where two 'upstream' devices need to be connected together a gender changer or 'Insulation Displacement Connector' (IDC) cable with two IDC sockets on can be used.



E-BLOCKS1: SECTION D

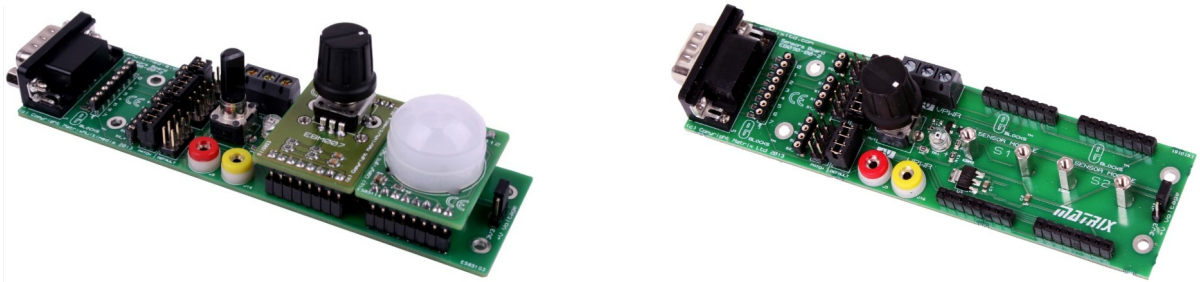
Understanding the patch system

Most 'downstream' E-blocks include a patch system that gives you flexibility in the connections that are made between 'upstream' and 'downstream' E-blocks. The default connections are optimized for ease of connection between PIC MCU 'upstream' and E-Blocks 'downstream' boards. Other microcontrollers may require different connections, facilitated by the patch system.

The patch system has two parts:

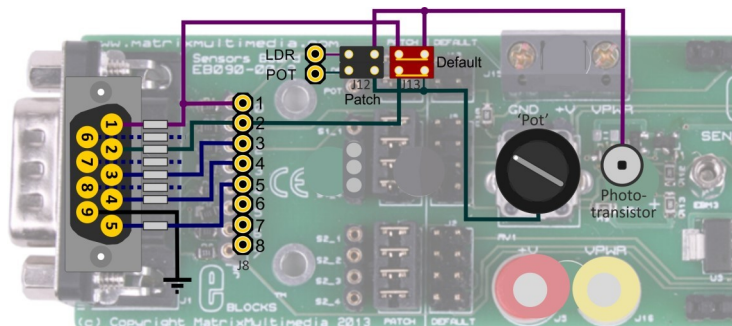
- jumper links that select between default connections and the patch system,
- the patch connectors themselves.

The photographs show an EB090 sensors motherboard with and without sensors attached to it.

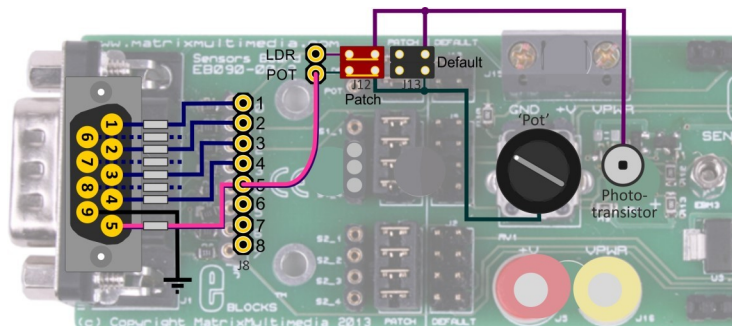


The next two diagrams show how to use the patch system on this board.

In the upper diagram, jumpers J12 and J13 are set to the 'default' position. This routes signals from the light sensor and 'pot' to pins 1 and 2 of the D-type plug, respectively.



However, when using a different microcontroller, you might want the signal from the 'pot' to appear on pin 5 of the D-type plug. To achieve this, move jumpers J12 and J13 to the 'patch' position and add a wire from the 'pot' socket on 'J11' to pin 5 on J8, as shown in the lower diagram.



Section: **Index**

Introduction

| | |
|---|---|
| About this Course | 3 |
| What you will need | 4 |
| Course Conventions | 6 |
| Learning Objectives | 8 |
| Mapping to BTEC Level 3 National Extended Diploma in Engineering - Unit 6 | 9 |

Section 1

| | |
|---|----|
| What is a microcontroller? | 12 |
| Microcontrollers | 13 |
| The Digital World | 14 |
| The Analogue World | 14 |
| Analogue Data | 15 |
| Digital Data | 15 |
| Analogue to Digital Conversion | 16 |
| Inputting data into a microcontroller | 17 |
| Outputting data | 17 |
| Current Limits | 18 |
| Storing Data | 18 |
| Types of Memory | 19 |
| Read Only Memory (ROM) | 19 |
| Random Access Memory (RAM) | 19 |
| PIC Memory | 20 |
| Programming | 20 |
| The Flowcode Process | 21 |
| Running the Program | 21 |
| Different types of Microcontroller | 22 |
| PIC16F1887 Architecture | 23 |
| PORT A | 22 |
| PORT B | 22 |
| PORT C | 22 |
| PORT D | 22 |
| PORT E | 22 |
| Memory: | 23 |
| ALU: | 24 |
| Timer 1 (TMR1): | 24 |
| Timer 0 (TMR0): | 24 |
| RBO External Interrupt: | 25 |
| PORT B External Interrupt: | 25 |
| A/D: | 25 |
| Busses: | 25 |
| Introduction to clocks | 25 |

Section 2

| | |
|-------------------------------------|----|
| Downstream boards | 27 |
| Upstream boards | 27 |
| BL0011 PIC Programmer | 28 |
| BL0114 Combo Board | 30 |
| Connecting E-blocks together | 30 |
| Using E-blocks on the bench | 30 |
| Protecting E-blocks circuitry | 30 |

Section 3

| | |
|--|----|
| Introduction to Flowcode | 32 |
| What is Flowcode | 32 |
| Flowcode overview | 32 |
| The toolbars and panels | 33 |
| Icons toolbar | 33 |
| Components toolbar | 33 |
| Menu and simulation toolbar | 33 |
| Dashboard & System Panels | 34 |
| Properties pane | 34 |
| Project explorer | 34 |
| Icon list | 34 |
| Chip window | 35 |
| Docking and undocking the toolbars and panes | 35 |
| Flowchart window | 36 |
| Simulation Debugger window | 36 |
| Starting a new Flowchart | 37 |
| Opening an existing Flowchart | 37 |
| Saving a Flowchart | 37 |
| Saving Flowchart Images | 37 |
| The View menu | 38 |
| Global settings | 38 |
| Application Tab..... | 38 |
| Flowchart Tab | 39 |
| Scheme Tab | 39 |
| Locations Tab | 40 |
| View Analog Inputs | 40 |
| View Digital Pins | 40 |
| Getting Help With Flowcode | 41 |

Section 4

| | |
|--|----|
| Adding digital outputs - Light the LED | 42 |
| The hardware | 42 |
| Setting up a New Project..... | 43 |
| Adding LEDs | 44 |
| The flowchart | 44 |
| Running the Simulation | 45 |
| Changing LED properties | 46 |
| Changing the output | 46 |
| Saving the Program | 46 |
| Compiling to Chip | 46 |
| Changing port Settings | 47 |
| Binary Numbers | 48 |
| Converting Numbers | 48 |
| Working in Hex | 48 |
| Hex in Flowcode | 48 |
| Coding Constructs - Number Systems Worksheet | 49 |

Section 5

| | |
|--|-----------|
| Example 1. Adding digital inputs - Where's the fire? | 51 |
| Setting up the flowchart | 51 |
| Creating the variables | 51 |
| More on variables | 53 |
| Flowcode variables | 53 |
| Other variable issues | 53 |
| Why worry? | 53 |
| Setting up the outputs | 54 |
| Adding the switches | 54 |
| Simulating the program | 54 |
| Example 2. Using loops - Counting sheep | 55 |
| Setting up the flowchart | 55 |
| Creating the variables | 55 |
| Setting up the calculation | 56 |
| Configuring loop properties | 56 |
| When to test? | 56 |
| Loop for a set number of times | 56 |
| Setting up the input | 57 |
| Setting up the output | 57 |
| Adding the LED array | 57 |
| Adding the switch | 58 |
| Simulating the program | 58 |
| The solution: Adding a Delay | 59 |
| Example 3. The LCD display - Posting messages | 60 |
| LCD displays | 60 |
| Adding the LCD component | 60 |
| Writing messages | 61 |
| Other LCD functions | 61 |
| Using PrintNumber - an example: | 61 |
| Example 4. A stopwatch | 63 |
| Example 5. Using binary numbers - A binary calculator | 64 |
| Setting up the flowchart | 64 |
| Creating the variables | 65 |
| Setting up the inputs | 65 |
| Setting up the calculation | 66 |
| Setting up the output | 66 |
| Adding a LED array | 66 |
| Adding the switches | 66 |
| Slow simulation | 67 |
| Example 6. Binary logic in control | 68 |
| A. Controlling the microwave oven | 68 |
| Setting up the flowchart | 68 |
| B. Controlling interior light in a car | 70 |
| Setting up the flowchart | 70 |

Section 6

| | |
|--|----|
| Exercise 1 - Creating Outputs..... | 73 |
| Exercise 2 - Using Delays | 74 |
| Exercise 3 - Using Connection Points | 75 |
| Exercise 4 - Performing Calculations | 76 |
| Exercise 5 - Using Loops | 77 |
| Exercise 6 - Inputting Data | 78 |
| Exercise 7 - Making Decisions | 79 |
| Exercise 8 - Programming LCDs | 81 |
| Exercise 9 - Using the Keypad | 82 |
| Exercise 10 - Analogue Inputs and the EEPROM | 83 |
| Exercise 11 - Using Software Macros | 84 |
| Exercise 12 - Using External Interrupts | 85 |
| Exercise 13 - Using Timer Interrupts | 86 |
| Additional Challenges | 87 |

Version Control

| Version | Author | Date | Changes |
|---------|--------|------------|--|
| 1.0 | JV | 28/10/2016 | Document creation. |
| 2.0 | RT | 20/04/2018 | Update re. E-blocks2 and general revamp. |
| 3.0 | LM | 24/02/2021 | Update re. Flowcode v9 |



Matrix Technology Solutions Ltd.
33 Gibbet Street
Halifax
HX1 5BA

t: +44 (0) 1422 252380
e: sales@matrixtsl.com

www.matrixtsl.com

CP4375