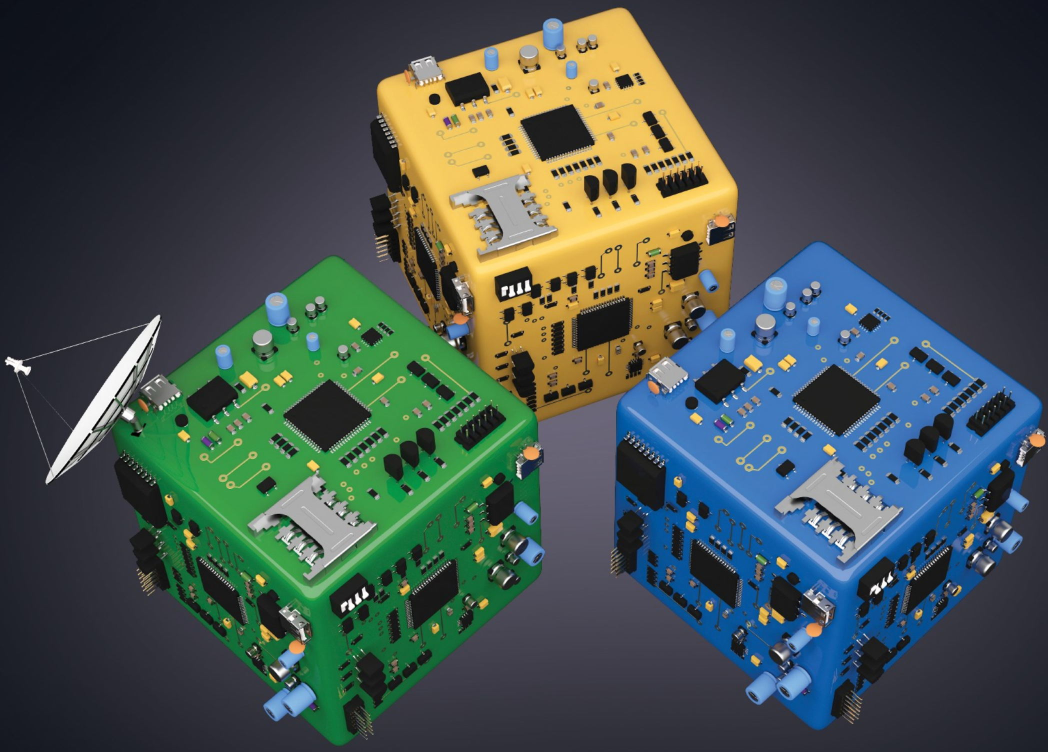
FLOWCODE6®

Simplifying Technology

Design and make electronic products 2



SE3810

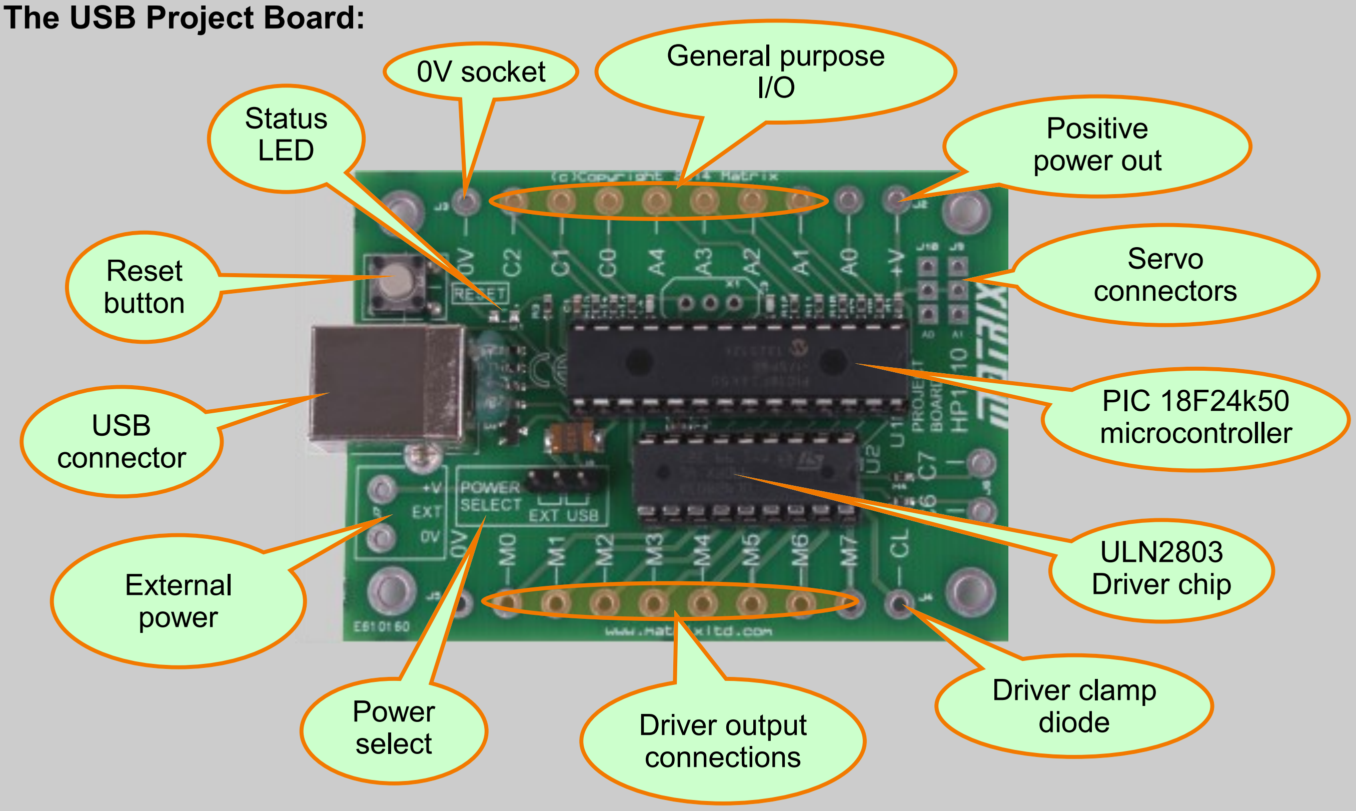
MATRIX

www.MatrixTSL.com

Contents

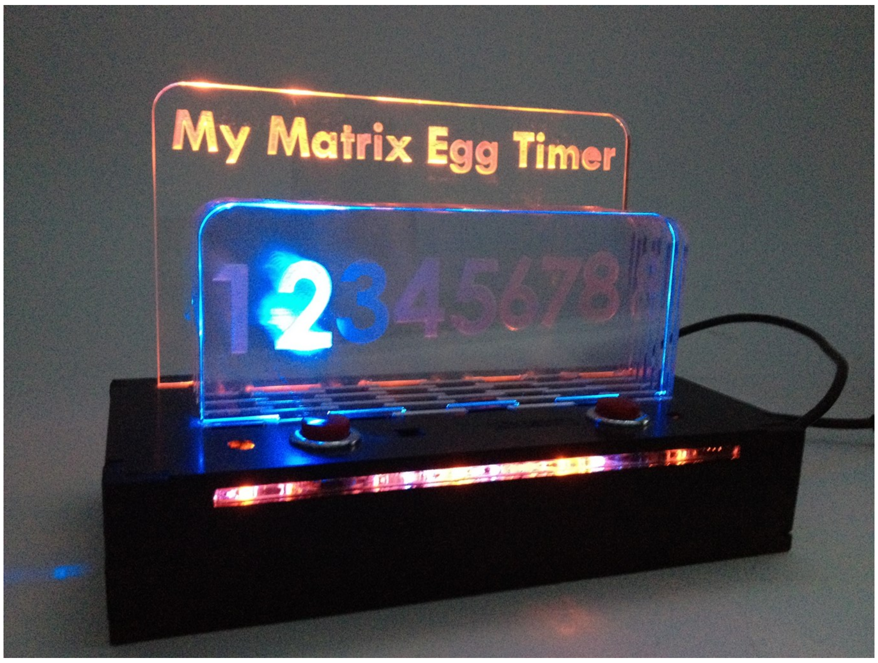
Introduction

The USB Project Board:



The hardware:

- The 'Power select' jumper position determines the power source for the hardware, either:
  - 5V delivered by the USB connection;
  - 2.3V to 5.5V from an external power supply.
- Input devices, such as switches and sensors, and low power output devices, like LEDs, can be soldered to the microcontroller terminals, labelled A0 to A4, C0 to C2, C6 and C7.
- Higher power devices, - motors and solenoids etc., can be soldered to the driver output terminals, labelled M0 to M7, which can handle currents up to 500mA from an external power supply.
- When inductive loads, such as motors and solenoids, are used, connecting the external positive power rail to the 'CL' terminal, makes use of the on-board diode protection to safe-guard the driver chip and microcontroller.
- The USB connector is used to connect to a computer to allow a program to be downloaded to the microcontroller.
- Once programming is completed:
  - the USB cable can be disconnected, unless it is used to deliver power to the hardware;
  - the Reset button is pressed to start the program running on the hardware.



What will we cover in this curriculum pack?:

- In this curriculum we will cover the creation of 3 typical D&T based projects, based on the Matrix USB Project Board. The examples will demonstrate how to implement techniques used in curriculum ‘Design and Make 1’, and show you how Flowcode can be used with 3rd party programs to improve the design cycle.
- The 3 examples covered in this curriculum will vary in complexity slightly, and each documented example will provide less detail that the previous one, to allow you to learn for yourself.
- In this curriculum pack we will demonstrate the creation of;
    - A binary clock
    - An egg timer
    - A 3D random dice value generator (simulation only)
- Once you have read through this pack you should be able to recreate the examples shown here, or hopefully you will be inspired to create your own projects.

- First, lets recap the layout of the USB Project Board and how we connect our devices to it. If you’ve read through curriculum pack ‘Design & Make 1’, and are confident you may skip this next section as it is a direct replica of what has been previously covered.

Connecting the hardware:

The diagrams show examples of different hardware attached to the USB Project Board. (There are other ways to do this!) To make them clearer, the USB Project Board is turned up-side-down.

The Project Board offers a set of terminals, labelled A0 to A4, C0 to C2 and C6 and C7, connected directly to the microcontroller ports. When these are configured as outputs, they deliver electrical power to the attached devices, but the current is limited to a few milliamps.

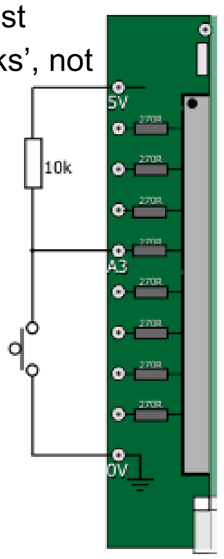
Alternatively, devices can be attached to the output terminals of the driver chip, the ULN 2803. These are labelled M0 to M7. Used in this way, the device must be powered from an external power supply, so that that the driver output 'sinks', not 'sources' current. (These concepts are explained shortly.)

Switch units:

The diagram shows a switch unit, (a push switch and resistor), connected to input a signal to port A, bit 3 of the microcontroller.

However, it could be connected to any of the microcontroller port bits.

It is connected to the '5V' and '0V' terminals, and so uses the USB Project Board as its source of power.

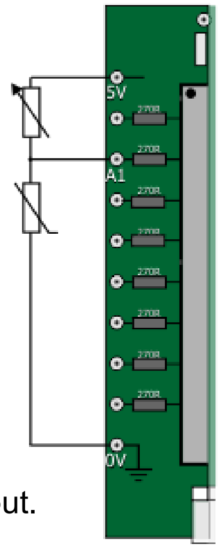


Sensing subsystems:

Here, a temperature-sensing unit, (a thermistor and a variable resistor), is connected to deliver an input signal to the system.

Although port A, bit 1 is used in the diagram, any microcontroller port bit could be used instead.

The sensing subsystem uses the USB Project Board as a power supply.

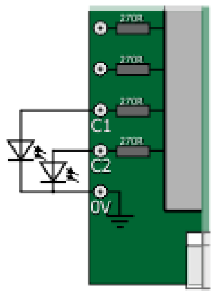


LEDs:

LEDs can be connected to any microcontroller terminal configured as an output.

The current-limiting resistors (270 ohm) are found on the USB board itself.

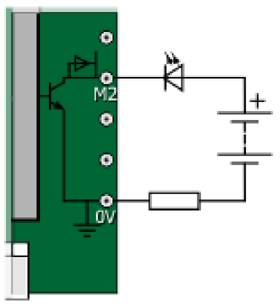
As the microcontroller acts as the source of current, the LED must be connected as shown, with its anode to the microcontroller terminal, and its cathode to 0V.

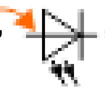


Connecting the hardware continued**...:**

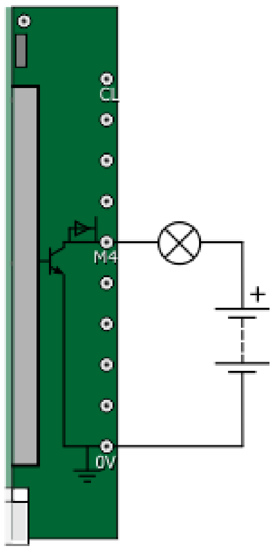
LEDs again:

This is an alternative way to connect a LED:



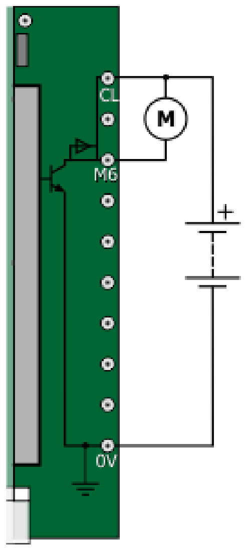
- the LED is powered from an external power supply;
- the anode,  , is connected to the positive terminal of the power supply;
- a resistor must be added to protect the LED;
- any 'M' terminal could be used;
- the transistor inside the ULN2803 chip acts as a switch between the LED and 0V. When the program sends a logic 1 signal to the driver, the transistor 'switches on' and the LED lights.

Lamps and indicators:



- An external power supply is needed.
- Normal filament lamps are not polarised - it doesn't matter which way round they are connected.
- As before, any 'M' terminal could be used.
- Again, the transistor inside the ULN2803 chip acts as a switch.

Motors and solenoids:



- These are 'inductive' loads - when they switch off, a fleeting, but very high, reverse voltage can be generated, which could damage the driver chip and microcontroller. By using the 'CL' terminal as shown, the protective diode inside the driver chip is used to 'short-circuit' this reverse voltage, and protect the chip.
- Notice how the external power supply is connected.
- These loads are not polarised - it doesn't matter which way round they are connected.
- As before, any 'M' terminal could be used.
- The transistor inside the ULN2803 chip again acts as a switch.

Connecting the hardware continued...:

Identifying resistors:

The resistors on the USB Project Board itself are tiny, surface mounted devices. 'Traditional' resistors have two metal leads attached allowing for easy soldering to other 'lead'ed components.

First of all, you must recognise which resistor to use.

Resistor Colour Code:

- Resistors often come with coloured bands across their body to show their resistance.

Black	Brown	Red	Orange	Yellow	Green	Blue	Purple	Grey	White
0	1	2	3	4	5	6	7	8	9

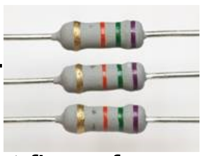
- Each colour represents a number, as shown in the table.
- To read the colour code, start from the opposite end to the gold or silver band:

The procedure:

- Write down the number shown by the first colour band, and then the second colour band.
- Add the number of 0's shown in the third band (e.g. for red, add two 0's.)
- The final band (gold, (5%) or silver (10%)) shows the tolerance (accuracy).

For example, the resistors in the photograph have a resistance of:

7 (purple) **5** (green) **000** (orange) = 75000Ω (or 75kΩ) with a 5% tolerance.



Sourcing and sinking current:

Electronic devices work when an electric current flows through them. This current flows from a source (often the positive terminal of a power supply) and flows around the circuit, before returning to the sink, (usually the negative terminal of the power supply).

In the systems covered in this course, current flows from, and returns to, a power supply, within the computer, (if we are using the USB Power Source option). From the component's perspective, however, the current is sourced from the USB Project Board, and flows back to it (as the sink).

For the LED, for example, the source is terminal 'C1', and the sink is 0V. For some of the circuits, that is not the case. The source will be the '+V' (5V) terminal, and the sink one of the terminals on the ULN32003 driver chip, (marked with 'M' on the board.)

Using the worksheets:

The worksheets that follow show how to connect and then control the hardware, using programs stored in the microcontroller

The Teacher Guide at the end of the module details all the steps needed to create these programs and test the systems. Please refer to this when necessary.

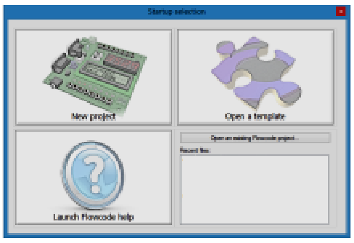
Step-by-step Guide to Flowcode 6 flowcharts:

1. Open Flowcode 6:

When the Flowcode application starts, you are presented with four options:

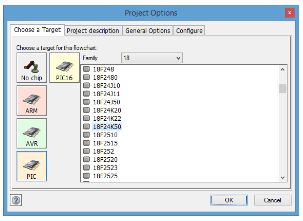
- New project;
- Open a template;
- Launch Flowcode Help;
- Open an existing Flowcode project.

Click on 'New project'.



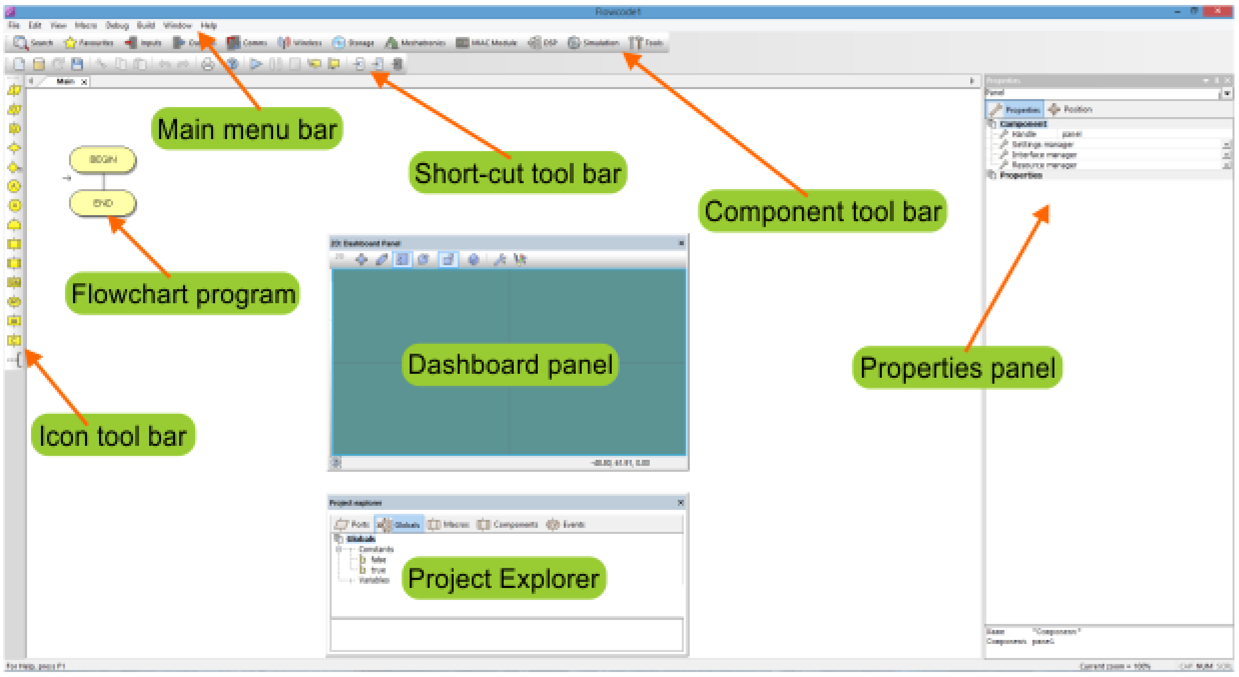
2. Select the target microcontroller:

The 'Project Options' screen opens, on the 'Choose a Target' tab.
Click on 'PIC', and for 'Family' choose 'Misc'.
Select the 'Matrix Project Board (18F24K50)' option.
Then click on 'OK'



3. Add the icons:

A new flowchart appears.



(We will not use the 'System Panel' in these programs. It may appear when the flowchart opens. If so, delete it by clicking on the '**X**' in the top right-hand corner of the panel, or by opening the 'View' menu and de-selecting 'System Panel'.)

Down the left-hand edge of the workspace is the strip of possible icons. Click and drag the ones you want to make up the flowchart .

Step-by-step Guide to the Worksheets - continued...:

4. **Configure each icon:**
   Double-click on each icon in turn, and use information like that given in the worksheets to complete the configuration dialogue box.
   Click on 'OK' after completing each.

5. **Open the Dashboard Panel:**
   Open the 'View' menu, and click on the 'Dashboard Panel' option.

6. **Add the components:**
   Items such as switches are found in the 'Inputs' tool bar.
   The LEDs and motors are found in the 'Outputs' tool bar.

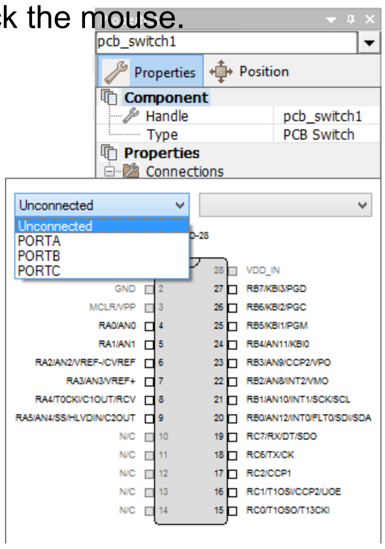
   Find the component you want and add it to the panel by:
   - clicking on it and dragging it onto the Dashboard panel;

   or
   - clicking on the down arrow to the left of its name, and selecting the ''Add to dashboard panel' option.

   Some components may be hidden, and require use of the 'Search' box.
   You can then 'click-and-drag' each component to its desired position.

   To zoom in or out, first click anywhere on the Dashboard panel.
   Then, hold down the 'Ctrl' key, and rotate the mouse wheel.
   This will cause the zoom level to change.

7. **Configure properties of components:**
   Move the cursor over the component, and right-click the mouse.
   The 'Properties' panel for the component appears.
   The important one for these worksheets is the 'Connections' property, shown on the right.



pcb_switch1

Component

Handle	pcb_switch1
Type	PCB Switch

Properties

Connections

Unconnected

   Click on the text alongside the 'Connection' item, (here "Unconnected") and a diagram of the chosen microcontroller chip appears.

   Select the pin to connect the component by:
   - selecting the port and bit from the two drop-down lists;

     or
   - clicking on the pin on the image of the chip.

Step-by-step Guide to the Worksheets - continued:

8. **Simulate the program:**

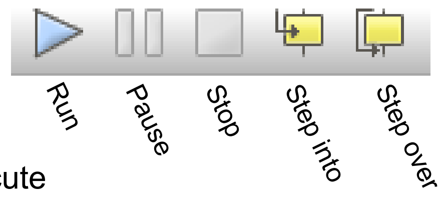
You can test whether your Flowcode program works by simulating it 'on-screen'.

To simulate a flowchart:

- click the 'Run' button on the main toolbar (or press **F5)**.

or

- select the 'Run' option from the 'Debug' menu;



Flowcode will go into simulation mode and will start to execute the program in the flowchart. Simulations can be paused or stopped by selecting either the 'Pause' or 'Stop' options from the 'Debug' menu or selecting them from the simulation section of the main toolbar.

Alternatively, you can simulate the flowchart step by step, using the 'Step Into' function by pressing **F8**. You can also 'Step Over' icons by pressing **Shift+F8.** A red rectangle indicates the **next** icon to be executed.

All these options can be accessed from the 'Debug' menu or by clicking the buttons on the main toolbar.

9. **Save the flowchart:**

Flowcharts must be saved before they can be downloaded to MIAC.

To save the flowchart:

- select either the 'Save' or 'Save As' option from the 'File' menu;

or

- click the button on the main toolbar.

10. **Connect the USB Project Board to the PC:**

Connect the USB socket on the board to the USB port of a computer. This allows the program to be transferred from the computer to the PIC microcontroller on the board.

11. **Compile the flowchart and transfer it to the chip:**

The next step is to compile the Flowcode program, (convert it into machine code,) and then transfer it to the microcontroller.

To do so:

- select the 'Compile to Chip' option from the 'Build' menu;

or

- click the 'Compile to Chip' icon on the toolbar.

12. **Test the program using the hardware:**

Press the 'Reset' switch on the USB Project Board.

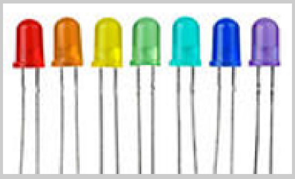
Now use the attached switches and LEDs etc. to confirm that the program does what you want it to do.

Project 1
The Binary Clock

Design and Make 2

We typically use either an analogue clock with hours, minutes and seconds hands, or a digital clock using LCD displays to tell the time.

But did you know that there are other ways to display the time?
In this example we will walk you through the creation of a Binary Clock, which uses only 12 LEDs to display the time of day.



The Binary Clock:

A binary clock is a method of representing the time of day on 3 sets of LEDs; Hours, Minutes and Seconds. In order to understand how this works we must first understand counting with binary notation.

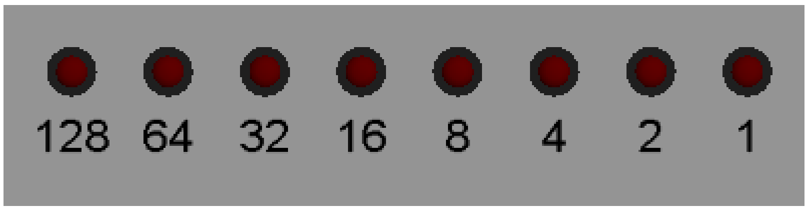
In the world of digital electronics everything is as simple as a ‘1’ or a ‘0’. If we want to turn something ON we set it to ‘1’, if we want to turn it OFF we set it to ‘0’. If all we ever wanted to do with our microcontroller was turn a single LED on or off, this might satisfy what we need. But what if we want a value that isn’t just on or off?

What if, for example, we want to count something like how many minutes have passed since the day started. For this we need to be able to store more than just 1’s and 0’s...or do we? Binary notation allows us to count up to any value using only 1 and 0, and here we will explain how:

A binary number is constructed of 1’s and 0’s as seen below. This image actually represents a value of 154 in ‘normal/ decimal numbers’. If that doesn’t make a lot of sense yet, don’t worry!

1 0 0 1 1 0 1 0

Let’s try look at our binary counting in a simpler way. Remember that we can turn an LED on or off by using a 1 or 0? 1 turns it ON, 0 turns it OFF. So instead of a possibly daunting binary number, we can swap all those 1’s and 0’s for a few LEDs which may help us understand.

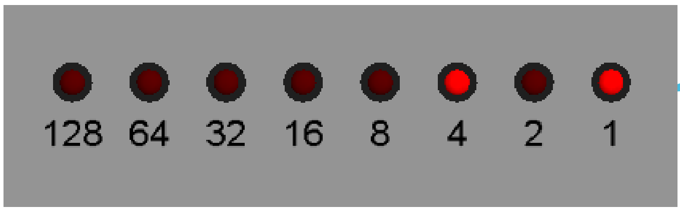


Project 1
The Binary Clock Cont...

Design and Make 2

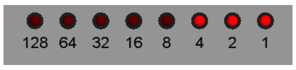
Binary Notation cont...

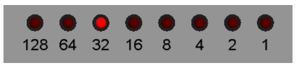
We see from the picture on the previous page that we have created an image of 8 LEDs. Each LED has a decimal value underneath it, but you will also notice these values are smallest on the right hand side. In Binary, we count right to left! Counting in binary is as simple as adding up a few values for all the LEDs that are turned ON. We will run through an example below, after which we have provided a few examples for you to work through.

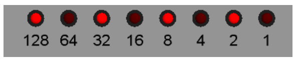


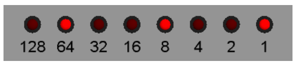
What value do we have here?
We simply count the LEDs that are turned ON;
4 + 1 = 5

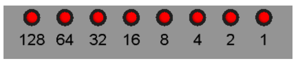
- Now that we have seen that binary counting is easy, we will provide you with a few examples. Calculate the value presented on the LEDs and write your answer next to it.

(a) 

(b) 

(c) 

(d) 

(e) 

- Imagine you were making a clock, and needed to count to 60 minutes and 12/24 hours. How many LEDs would you need for Minutes, and how many for Hours?

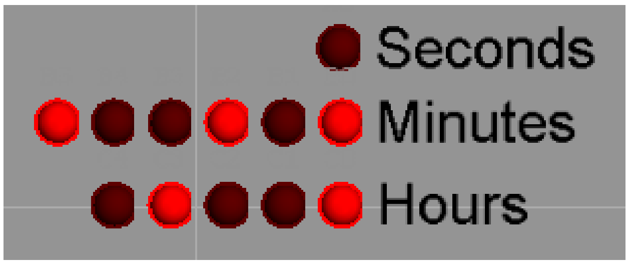
Answers:
(a) = 7
(b) = 32
(c) = 170
(d) = 73
(e) = 255
(f) = 6 and 5

Project 1
The Binary Clock Cont...

Counting with Binary and the LED Clock:

Now we understand how binary numbers are represented, we can start to see how the binary clock may work. Below is an image of a binary clock showing a time. You can see we have only 1 LED for seconds (in the final version this flashes for half a second ON and half a second OFF). We also have 6 LEDs for the minutes, and 5 for hours. 6 LEDs allow us to count to a value of 63; just enough for the maximum number of minutes we will count to. And 5 LEDs allow us to count to 31 which is enough to count to 12/24 hours.

- What time does the clock below display?



Answer:
09:37

Creating our Binary Clock:

- Now we understand how our clock will display the time on our LEDs we can begin to create the flowchart required for our clock. We will use the simulator within Flowcode to check it is working as we desired, and then we will download our program to our microcontroller to test it works in hardware.
- In Design & Make Electronic Products 1 (SE3844) we touched on creating variables to save data, creating Macros to implement ‘modularity’ and the use of components from our Flowcode toolbar. Therefore we will not go into detail in these areas in this curriculum pack.

Project 1

The Binary Clock Cont...

Seconds to Minutes to Hours:

One final step before we create our flowchart is to perform a bit of planning for our program. We have chosen a clock as it's fairly simple to create. Effectively, we need a counter which increases by a value of '1' every second. We then create variables to save the value of seconds, minutes and hours.

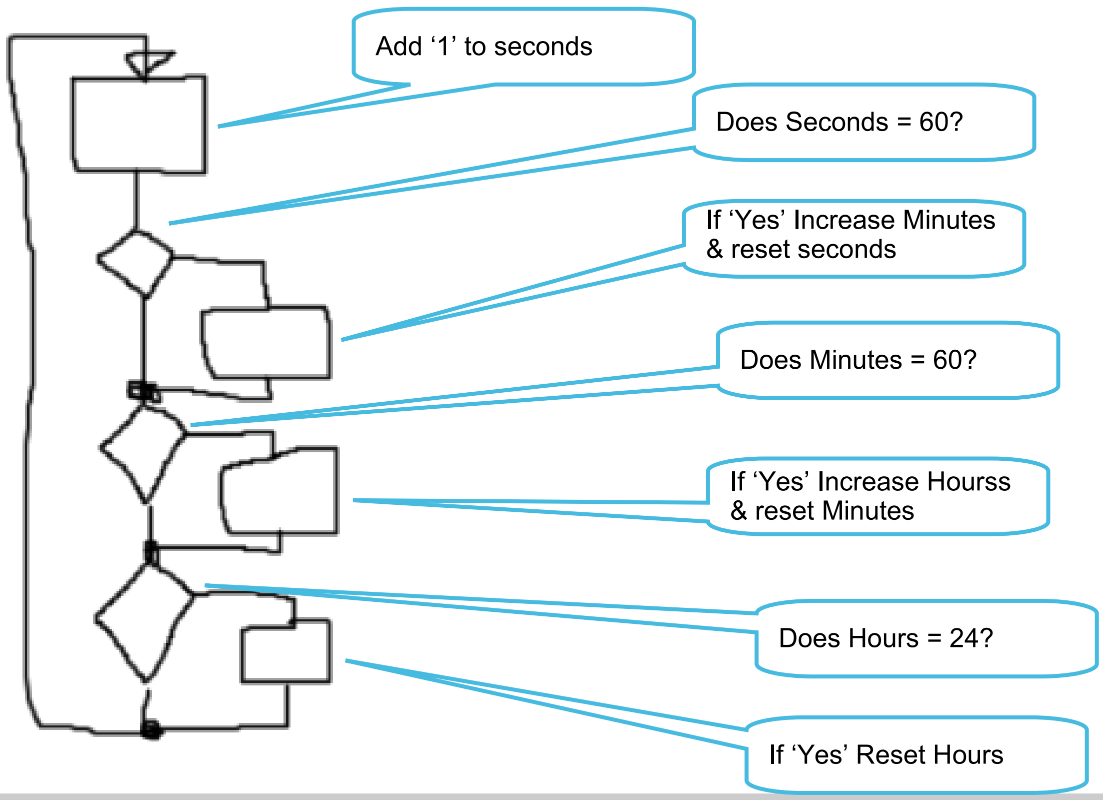
When our value of seconds reaches 60, we want to add '1' to minutes and reset our seconds to zero. When our minutes reaches 60 we want to add '1' to hours and reset our minutes. When hours reaches 12 or 24 (depending on whether you select a 12hr or 24hr clock) we want all clock variables to reset, and a new day begins.

Creating our variables:

Since our value for seconds, minutes or hours will never be more than 60 we can use a variable type 'byte' for each, as the maximum value a byte can hold is 255 (Think back to binary counting!). We could also use an integer, which would allow us to count to 65535, but we won't ever need that much space. Using a large variable when it is not required is much like using a really REALLY big piece of paper just to write one small word on it.

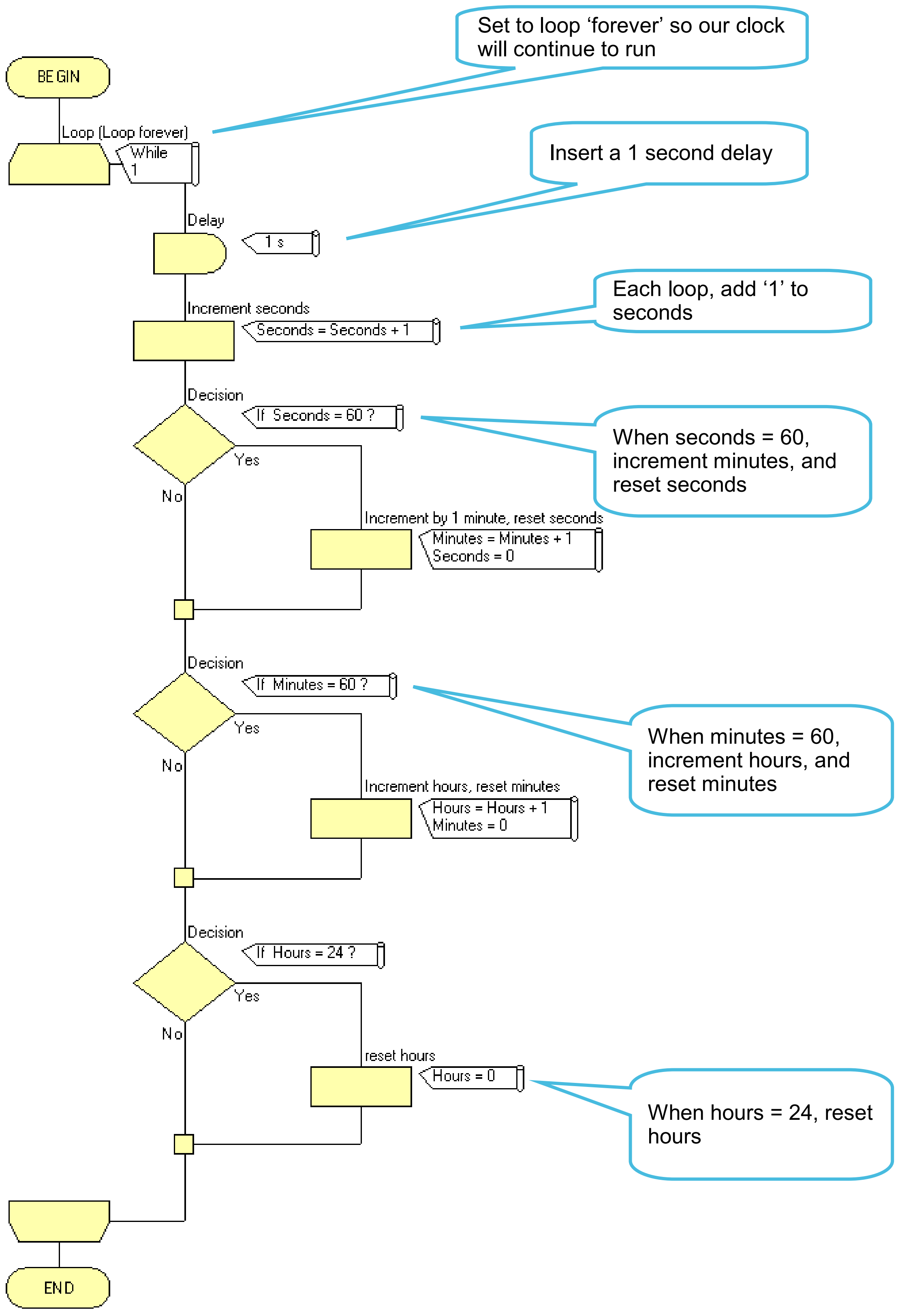
Binary Clock Version 1:

- Below is a quick sketch we made to show how simple our binary clock is! There's a few more details to add, but the sketched flowchart below shows the basic program.
- It's good practise to draw programs before you make them to plan how they will work.



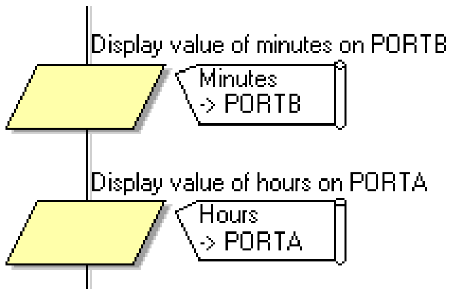
Basic Flowchart:

Below is a Flowcode flowchart of our program:


Set to loop 'forever' so our clock will continue to run
BEGIN
Loop (Loop forever)
While 1
Insert a 1 second delay
Delay
1 s
Each loop, add '1' to seconds
Increment seconds
Seconds = Seconds + 1
Decision
If Seconds = 60 ?
When seconds = 60, increment minutes, and reset seconds
Yes
No
Increment by 1 minute, reset seconds
Minutes = Minutes + 1
Seconds = 0
Decision
If Minutes = 60 ?
When minutes = 60, increment hours, and reset minutes
Yes
No
Increment hours, reset minutes
Hours = Hours + 1
Minutes = 0
Decision
If Hours = 24 ?
Yes
No
reset hours
Hours = 0
When hours = 24, reset hours
END

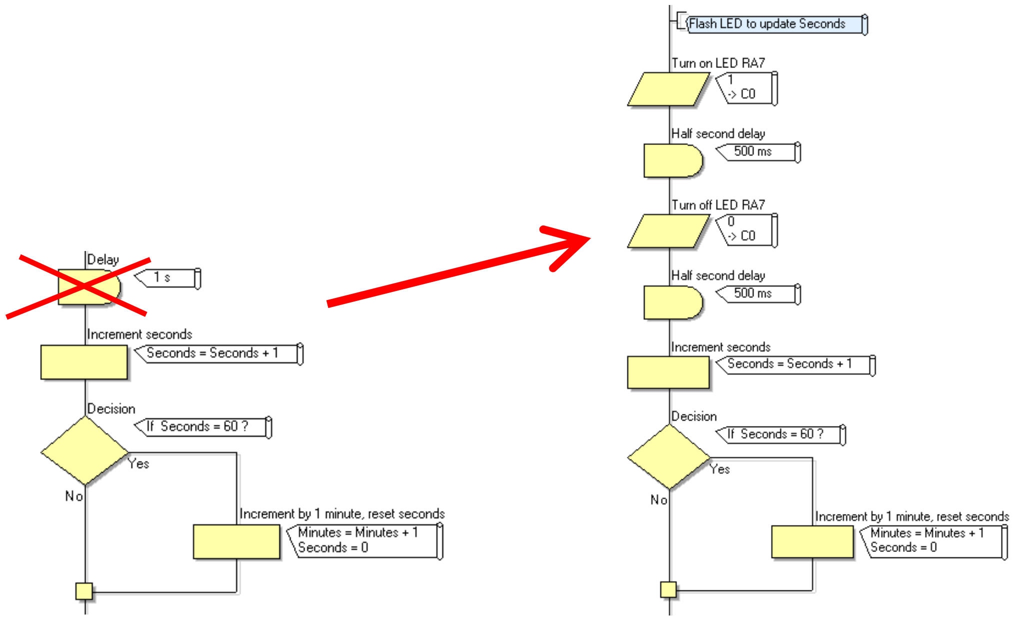
Basic Flowchart:

The flowchart in the previous page allows us to create a basic clock. However, we currently don't display any values on our LEDs, so it would be impossible for the end user to know what time it is. To display the time we write to the 'PORT' of the microcontroller. The image below shows how we display minutes and hours on our LEDs; it's as simple as 2 icons!


Display value of minutes on PORTB
Minutes -> PORTB
Display value of hours on PORTA
Hours -> PORTA

By writing our value of 'Minutes' and 'Hours' to PORTB and PORTA respectively, we will illuminate the appropriate LEDs to represent the time of the clock. The final section of code in the main program is to flash the 'seconds' LED. We have placed this code at the start of the flowchart, but it could equally be placed at the end.

- Note, we have slightly modified the original program from the previous page, by deleting the one second delay and changing it to two half second delays. Therefore the light flashes on for half a second, and off for half a second. Two images below demonstrate how we have modified the original program to flash the LED.

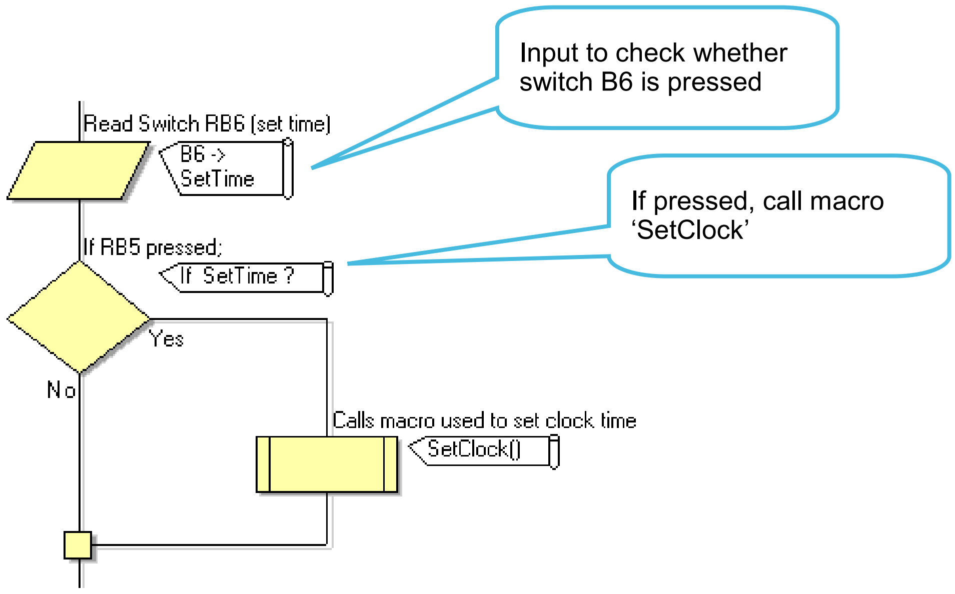


Additional code to set time:

Alongside telling the time, we also added some code to allow us to manually set the time. Without the ability to set the time we would have to ensure we power up the clock at exactly midnight, which is far from ideal. Also, unless a clock is perfectly accurate, which no clock is, we will occasionally need to set the time manually.

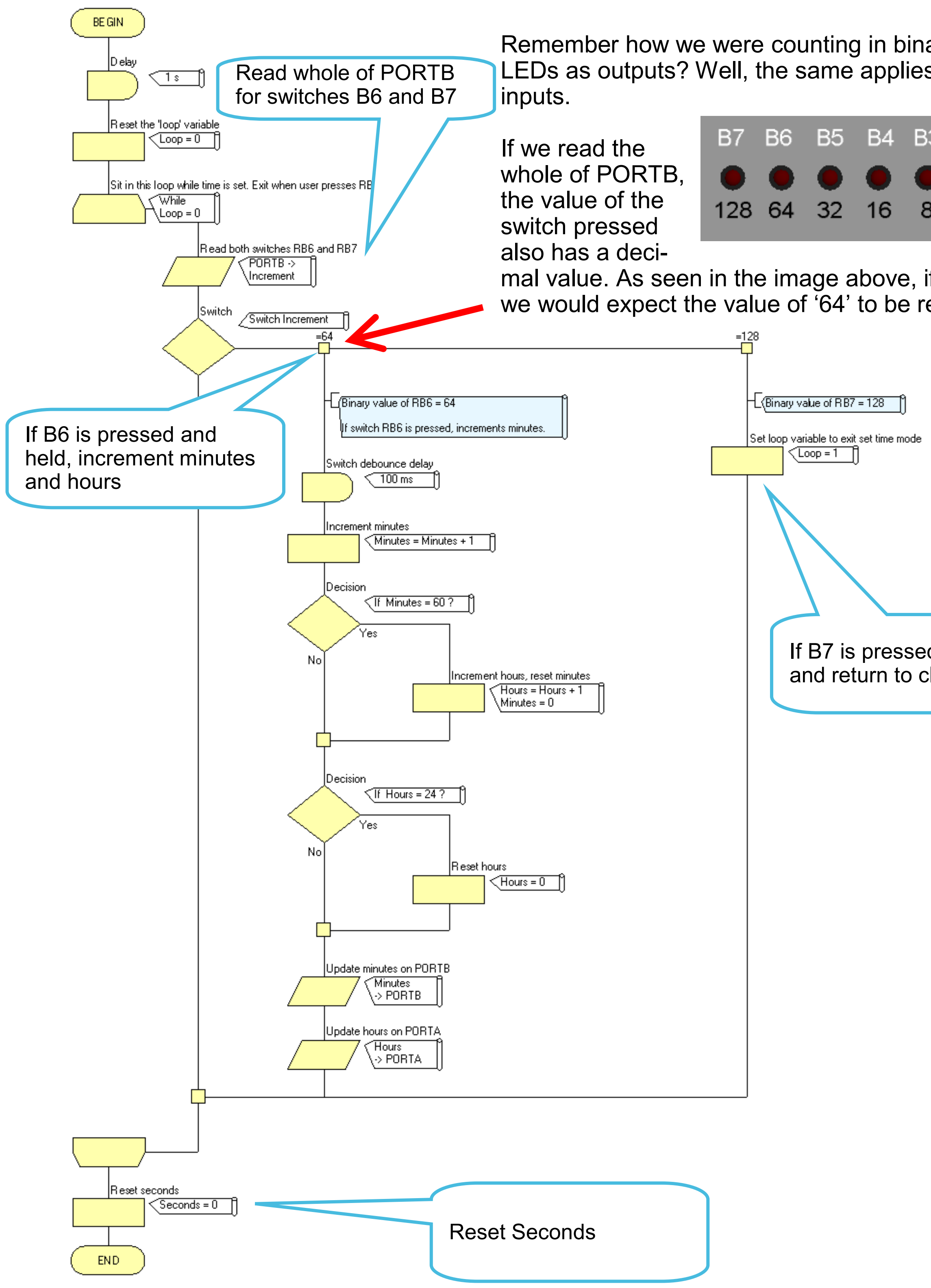
In order to set the time we have used 2 switches. In Flowcode we placed the switches on our simulation panel, and connected them to PORTB.6 and PortB.7. We configured the reading of the switch by using an 'Input' icon. With an 'If' statement we have configured our flowchart so that if switch B6 is pressed we call the Macro 'SetClock'. The code within SetClock is found on the following page.

- The code for entering the 'SetClock' mode is seen below, and lives in our 'Main' flowchart;

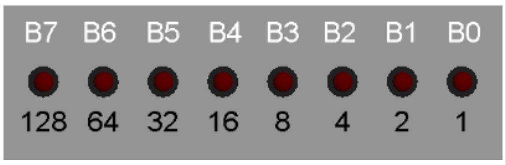

Input to check whether switch B6 is pressed
If pressed, call macro 'SetClock'
Read Switch RB6 (set time)
B6 -> SetTime
If RB5 pressed;
If SetTime ?
Yes
No
Calls macro used to set clock time
SetClock()

Additional code to set time:

- Below is the ‘SetClock’ macro, used to manually set the time of the clock:


BEGIN
Delay
1 s
Read whole of PORTB for switches B6 and B7
Reset the 'loop' variable
Loop = 0
Sit in this loop while time is set. Exit when user presses RB
While Loop = 0
Read both switches RB6 and RB7
PORTB -> Increment
Switch
Switch Increment
=64
=128
If B6 is pressed and held, increment minutes and hours
Binary value of RB6 = 64
If switch RB6 is pressed, increments minutes.
Switch debounce delay
100 ms
Increment minutes
Minutes = Minutes + 1
Decision
If Minutes = 60 ?
Yes
No
Increment hours, reset minutes
Hours = Hours + 1
Minutes = 0
Decision
If Hours = 24 ?
Yes
No
Reset hours
Hours = 0
Update minutes on PORTB
Minutes -> PORTB
Update hours on PORTA
Hours -> PORTA
Binary value of RB7 = 128
Set loop variable to exit set time mode
Loop = 1
If B7 is pressed exit loop and return to clock mode
Reset seconds
Seconds = 0
Reset Seconds
END

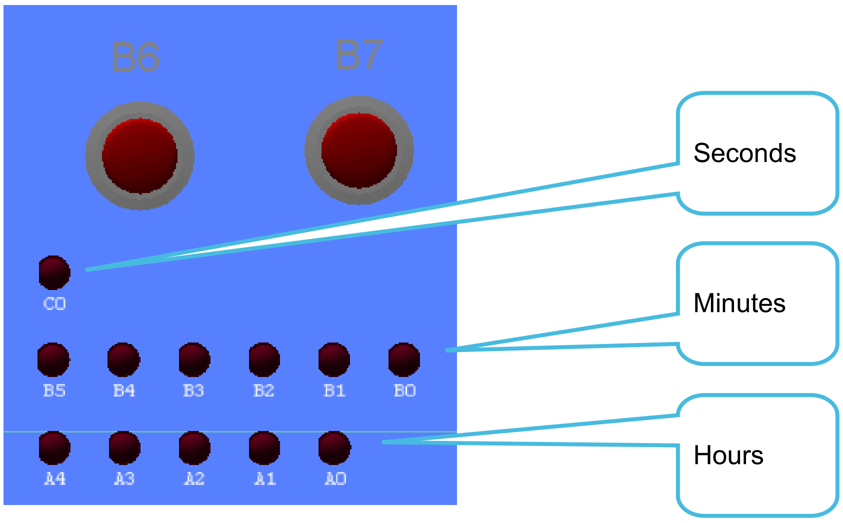
Remember how we were counting in binary to light up LEDs as outputs? Well, the same applies for reading inputs.



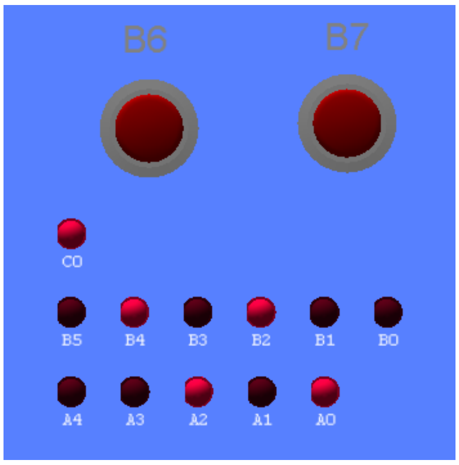
If we read the whole of PORTB, the value of the switch pressed also has a decimal value. As seen in the image above, if we press B6 we would expect the value of ‘64’ to be returned.

Simulating our Binary Clock:

- Until now we have just added components to our simulation panel and arranged them so they're easy to view as seen below.



One more check that we grasp the binary clock….

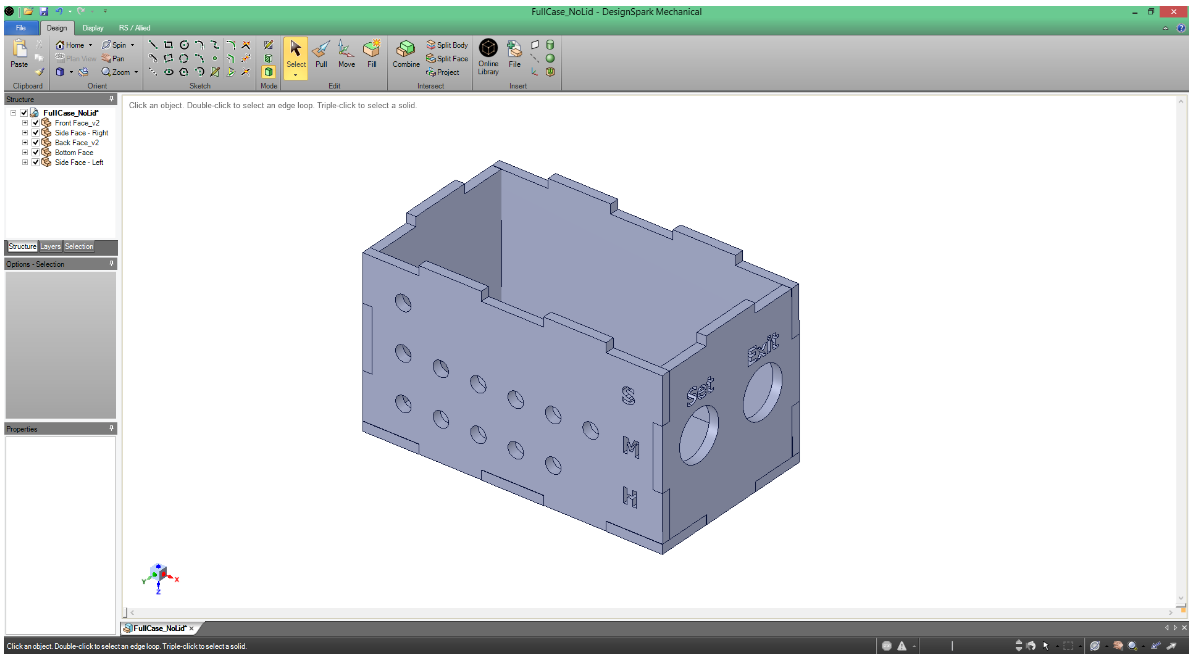


From what we have learnt previously, what time is the clock showing here in our running simulation?

Answers:
05:16

Importing 3D Models into Flowcode:

One feature of Flowcode is the ability to import 3D models from drawing packages. In this example we used DesignSpark Mechanical to create a simple box shape, which we could laser cut from 3mm clear acrylic. The DesignSpark Mechanical Assembly file can be seen below;

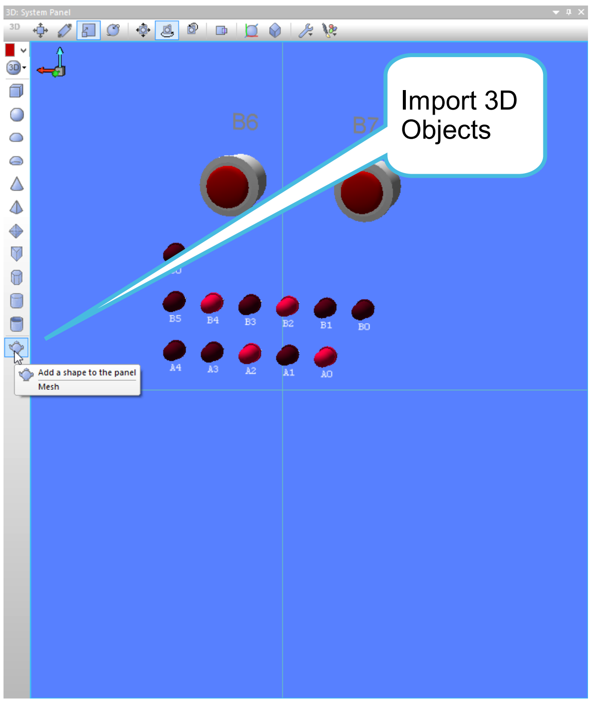

FullCase_NoLid - DesignSpark Mechanical
Structure
FullCase_NoLid*
Front Face_v2
Side Face - Right
Back Face_v2
Bottom Face
Side Face - Left
Click an object. Double-click to select an edge loop. Triple-click to select a solid.
S
M
H
Set
Exit

When we want to import a 3D object into Flowcode we must save it as an .OBJ file.

We import our 3D model using the little ‘teapot’ symbol on the left hand menu bar of the System panel, as indicated with the arrow.

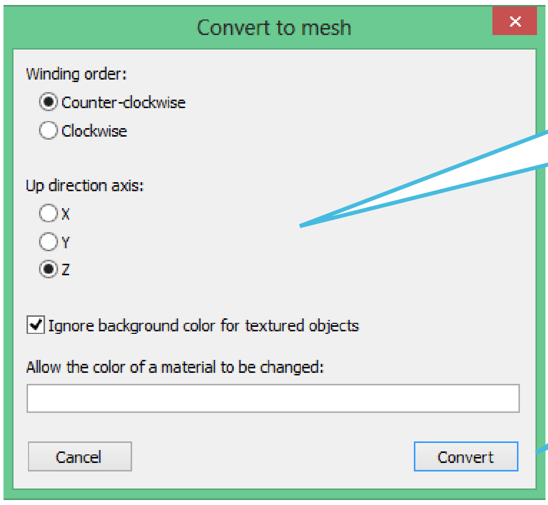
Drag the little teapot onto the simulation panel and a dialogue box will open allowing you to browse to the 3D design file you have created (.OBJ file)

Once you find your .OBJ file you will be presented with the dialogue box which is seen on the following page...



Importing 3D Models into Flowcode cont...

This dialogue box is used to convert the .OBJ file into a file that Flowcode recognises. The new file type will be a .MESH file. Keep all the default settings, unless you are an expert user!



Convert to mesh

Winding order:
- (●) Counter-clockwise
- ( ) Clockwise

Up direction axis:
- ( ) X
- ( ) Y
- (●) Z

☑ Ignore background color for textured objects

Allow the color of a material to be changed:

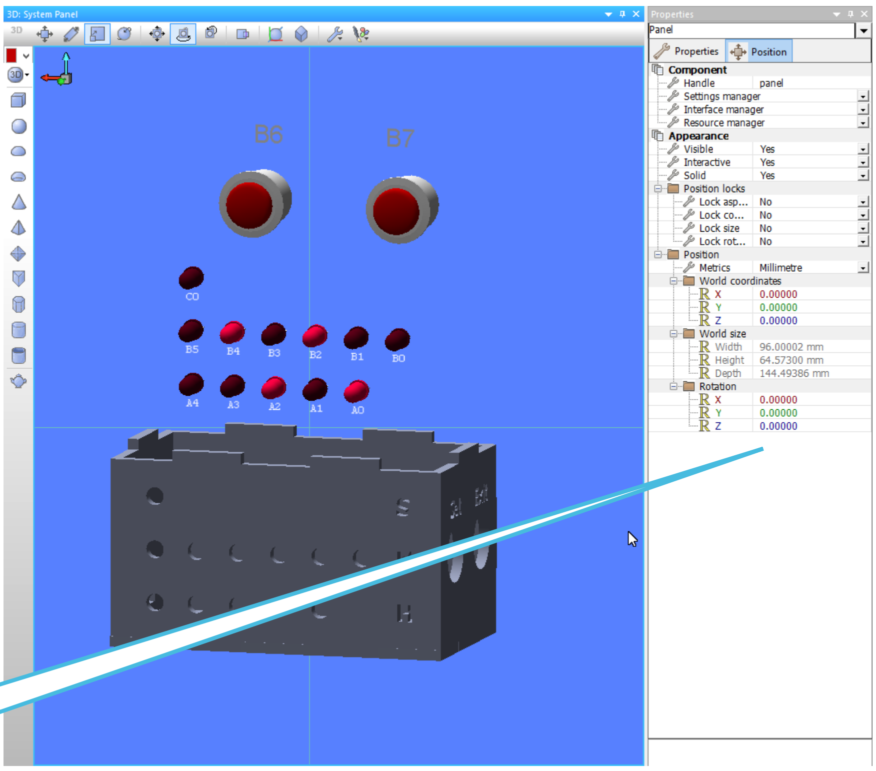
Leave all settings as default unless you are an 'expert' user

Click convert to import model

Once your 3D model has been imported you should see it appear on the simulation panel. You may need to move the object around to get it in the right place where you want it.

To move an item around the simulation panel, click on it with your left mouse button and drag it to your desired position.

You may also need to rotate the objects to align with the LEDs and switches already on the simulation panel. Any object can be rotated by modifying the parameters shown on the right side of this image.



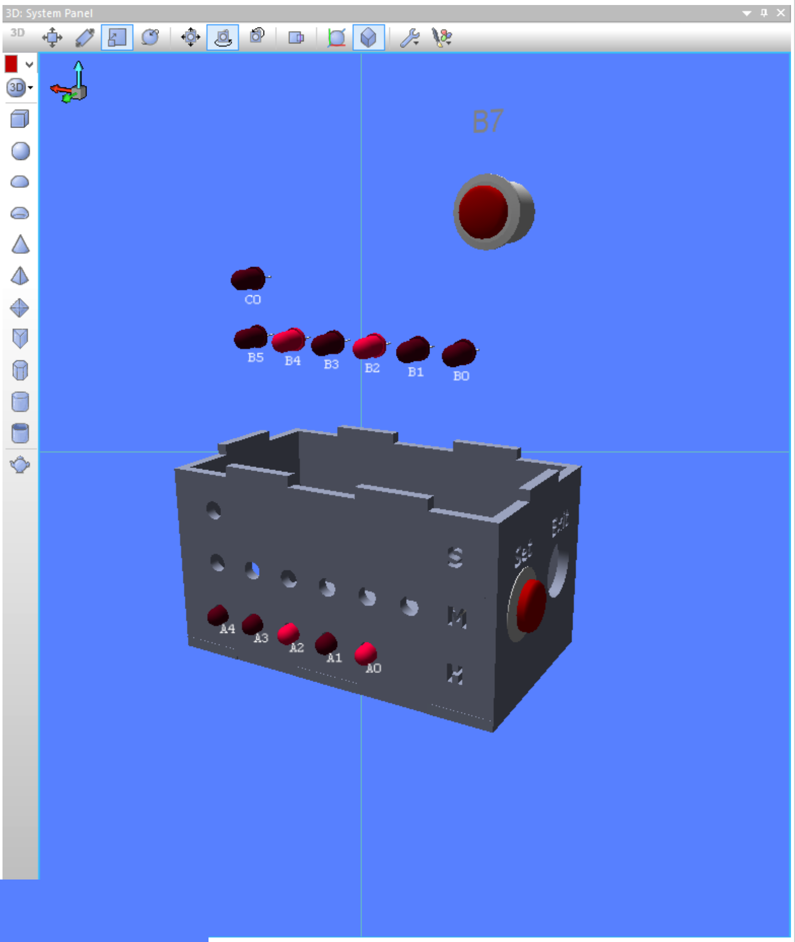
Component	
Handle	panel
Settings manager	
Interface manager	
Resource manager	
Appearance	
Visible	Yes
Interactive	Yes
Solid	Yes
Position locks	
Lock asp...	No
Lock co...	No
Lock size	No
Lock rot...	No
Position	
Metrics	Millimetre
World coordinates	
X	0.00000
Y	0.00000
Z	0.00000
World size	
Width	96.00002 mm
Height	64.57300 mm
Depth	144.49386 mm
Rotation	
X	0.00000
Y	0.00000
Z	0.00000

Use these parameters to rotate the objects on the simulation panel.

Finalising your 3D Model:

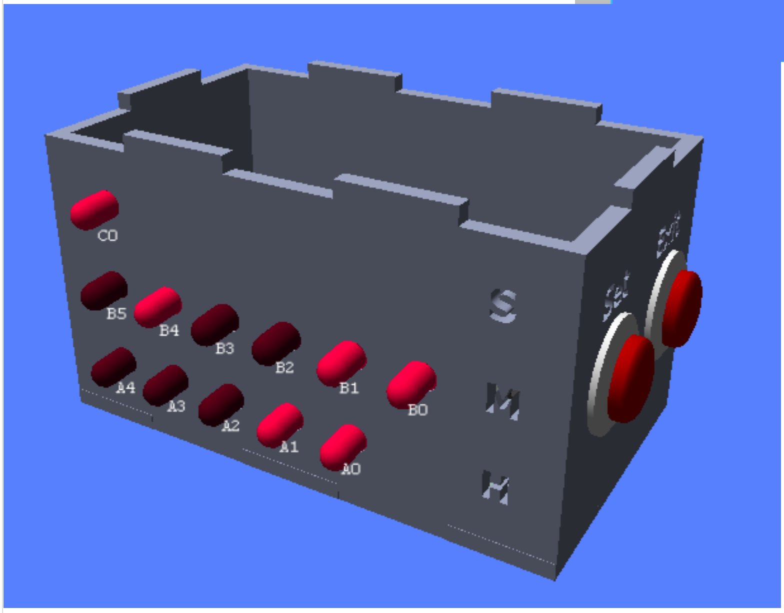
With our 3D model imported into our simulation panel, the final step is to configure our LEDs and switches so they are placed inside the case. There are two methods of moving our objects around the simulation panel. You may choose to use both methods:

1. Select the object by clicking on it, and dragging it around the simulation panel.
2. Select an object and modify its 'world coordinates' in the right hand properties box.



First, highlight the switch B6 and rotate it so it can be fitted in the side panel of the binary clock case.

In the image to the right we have also placed our bottom row of LEDs within the model. Again, by selecting them and dragging them into position.



To the left we can see our finished clock. Here we see one of the main advantages of the simulator in Flowcode; we were able to create a complete project and we still have not created any hardware. Now that we know our flowchart program is correct, and that our 3D model is also correct we can build our clock and have confidence that it will also work correctly.

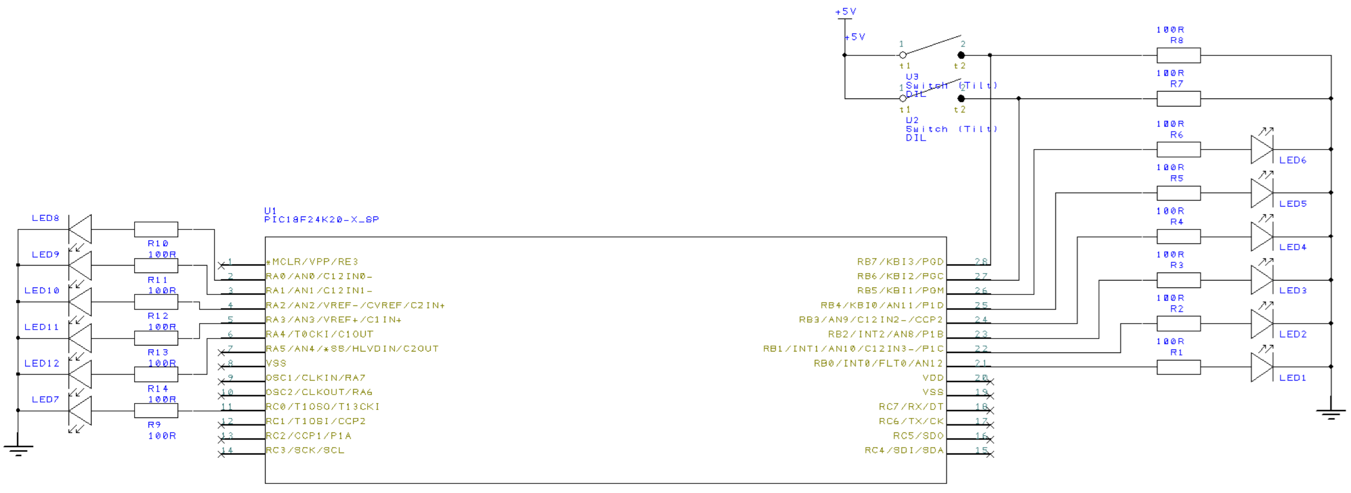
Project 1

The Binary Clock Cont...

The Final Step...let's build!:

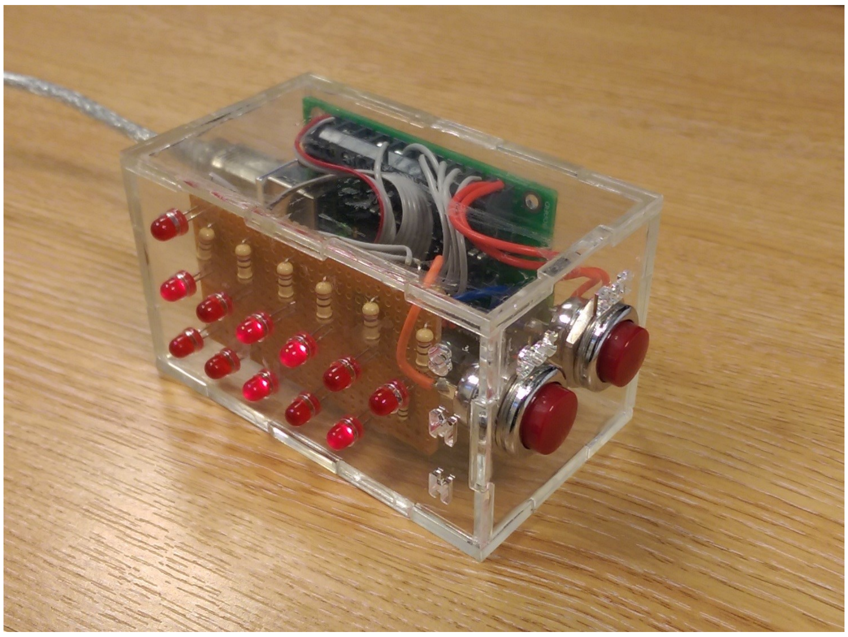
So, we have simulated our clock and imported our model and it all works as expected. But now we need to create and build our clock. For this example we used the Matrix HP1110 Project Board. A simple circuit schematic was created using DesignSpark PCB, and can be seen below which shows the connections directly from the microcontroller. The outputs from the Microcontroller are clearly labelled on the HP1110 project board.

In this example we didn't require the use of the motor driver chip as we are only powering low-power outputs. Instead, we removed the ULN2803 motor driver chip and instead placed small jumper wires between each pair of legs where the chip was on the project board. We powered our clock via a USB cable. These days most phone chargers can use standard USB cables to provide 5V, which our project board needs.



We created a veroboard circuit which held our LEDs and resistors, and also our pull down resistors for our switches. We soldered the LEDs directly in line with the holes of the case.

Here is the final product, a simple binary clock, complete with laser cut casing glued together. The file is the same we created in DesignSpark Mechanical.



Expansion Projects;

The project built here relies on two half second delays creating a 1s delay for our 'seconds' counter. This is not an accurate method of time keeping. The half second delays in the micro-controller might not be exactly half a second, they might be slightly more or slightly less. Therefore, over time the clock will start to tell the incorrect time.

Also, we have not accounted for the time taken to execute all the other programming commands in our clock; turning the LEDs on, checking if minutes = 60 etc. Lets say for example, that each command takes 1µs (microsecond) to execute. (In reality, they don't take this long, and each type of command takes a different length of time, but we are learning the principle!)

1µs is 0.000001s, which doesn't sound like a lot. However, if we have 10 commands in our program then each time our program runs through it will take 10µs longer than we planned. Over a period of 1 day there are 84 000 seconds. We can now calculate the inaccuracy of our clock over a 1 day period:

$$\text{Inaccuracy} = \text{No. of seconds} \times \text{Extra Delay}$$
$$\text{Inaccuracy} = 84\,000 \times 10\mu s = 0.864s$$

Therefore, each day we would be inaccurate by almost a complete second...not very accurate!

So how do we solve our inaccurate clock? There are at least two methods that Flowcode supports that you can investigate to make your clock more accurate:

1. Use a Timer Interrupt within the microcontroller.
    - A timer interrupt utilises an accurate counter within the microcontroller. Implement an interrupt with the 'INT' programming icon. → INT
1. Use a Real Time Clock (RTC) Module.
    - A RTC is a separate chip which can accurately tell the time. This time is then sent from this chip to our microcontroller so we could update the LEDs.

The use of either of these methods is down to you! Experiment in Flowcode and see if you can make your clock more accurate by adding either of the above 2 methods. Or perhaps there is another method you could use?

Project 2
The Egg Timer

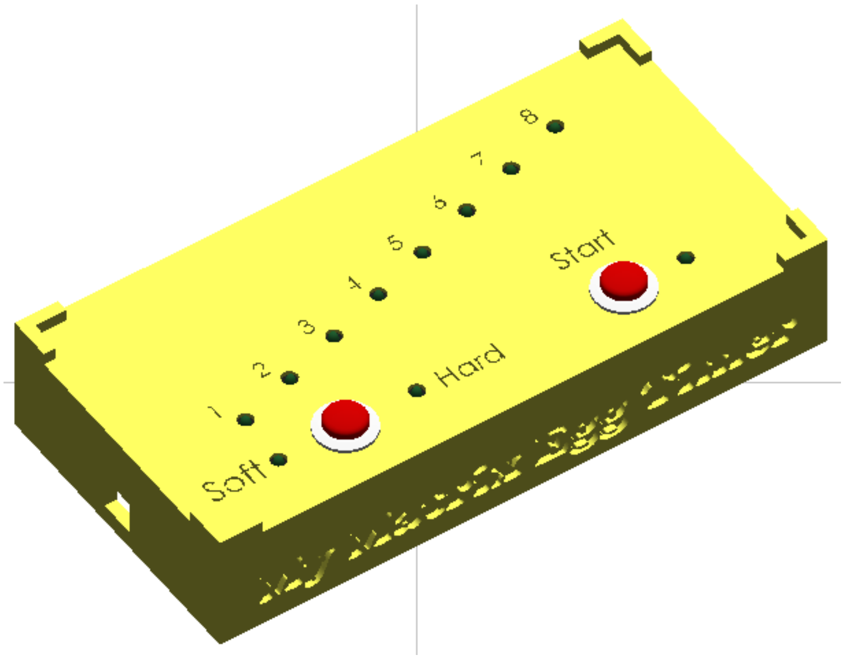
Design and Make 2

Whatever you do in life, you cannot escape time. In the kitchen getting exact timings help great chefs achieve consistent results. Here we will make an Egg Time, but you could extend this project to include a general timer and count down time compared to the way the binary clock counter up

Since you have already covered the concept of managing time in Project 1, we will not cover that here

The Egg Timer
In this example we want to create a egg time to control our cooking time. We will need to achieve 4 things in the project:

1. Decide whether we want the eggs soft or hard boiled
2. Start the timer
3. Show the time remaining
4. Show that our food is ready



How will this work?
We will use 2 switches. The first will select how long we cook for and the second will start the count down process.
We will use a number of LEDs to show the time remaining and then flash all LEDs to show that the count down has completed.

Choosing the right type of switch

In designing the egg time we decided to offer the option of soft boiled or hard boiled eggs. Based upon on you like your eggs you can then allocate an amount of time to each type. For our project we chose 4 minutes for soft and 8 minutes for hard.

Let us explore how to make the decision on whether we want soft or hard boiled eggs.

With the physical switch, there are several types of switches. You need to determine what switch you want to use, or is best for your project, and the develop your solution around this.

The main 2 types of switches are commonly know as push-to-make and latching switches. Unfortunately just from looking at the switches it is hard to tell

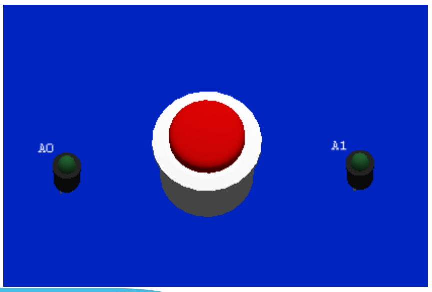
Push to Make	**Latching or Rocker switch**
A push to make switch allows electricity to flow between its two contacts when held in. When the button is released, the circuit is broken. This type of switch is also known as a Normally Open (NO) Switch. (Examples: doorbell, computer case power switch, calculator buttons, individual keys on a keyboard)	A latching switch is a switch that maintains its state after being activated. A push-to-make, push-to-break switch would therefore be a latching switch - each time you actuate it, whichever state the switch is left in will persist until the switch is actuated again. (Examples: light switch, car controls) In the images you can see 2 types of latching switches. One is clearly this with the rocker form of switch where as the other one latches during operation and the amount the button sticks out shows the state (sometime with LED lights on when active).

Project 2
The Egg Timer cont...

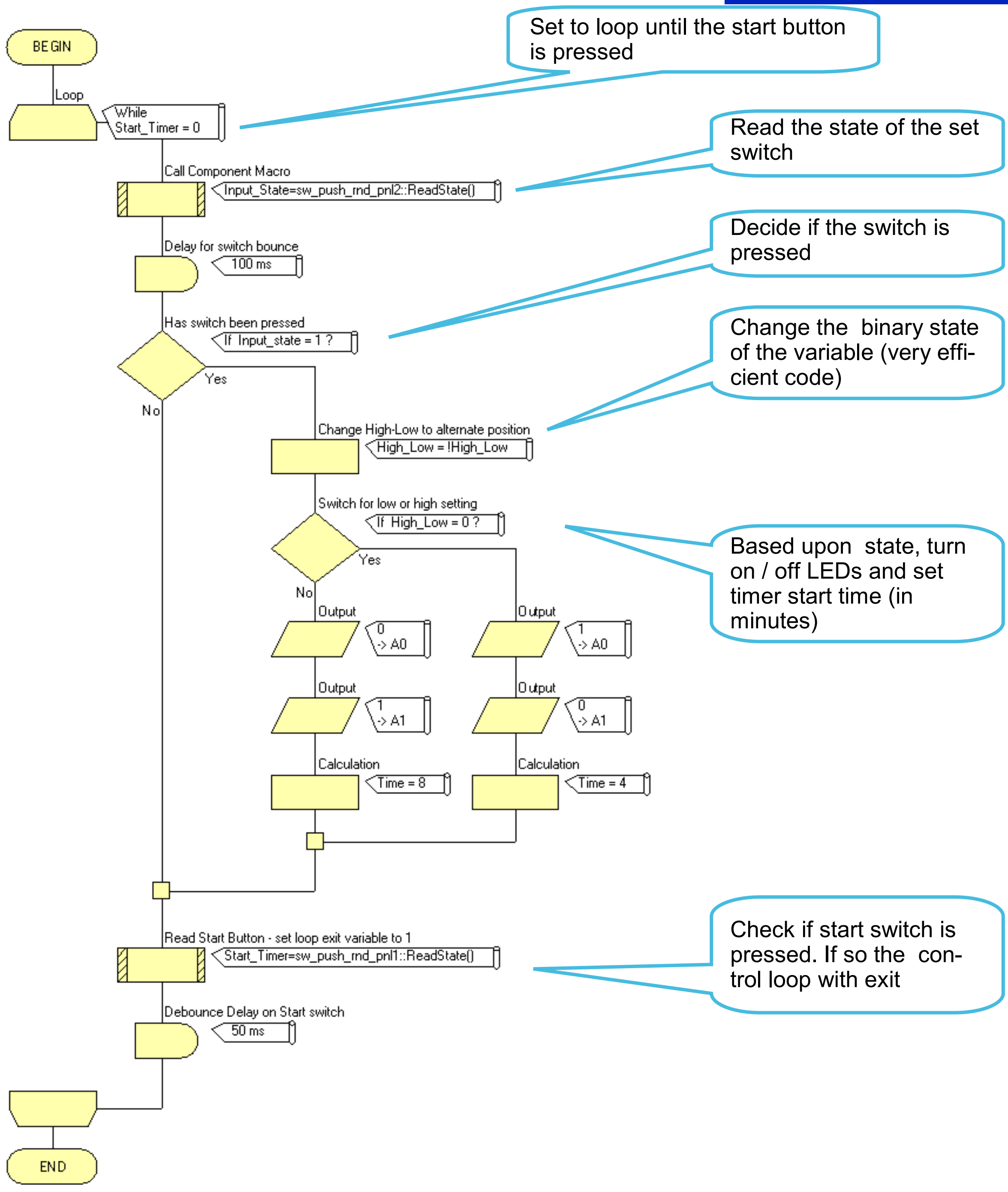
Design and Make 2

1. Setting the timer

For our egg time we decided to use a Push to Make switch and 2 LEDs to show whether it is soft or hard boiled. *You could extend this to soft, medium and hard boiled by using more variables and decision logic.*



Macro used to calculate whether the user wants hard or soft boiled.


BEGIN
Loop
While
Start_Timer = 0
Call Component Macro
Input_State=sw_push_rnd_pnl2::ReadState()
Delay for switch bounce
100 ms
Has switch been pressed
If Input_state = 1 ?
Yes
No
Change High-Low to alternate position
High_Low = !High_Low
Switch for low or high setting
If High_Low = 0 ?
Yes
No
Output
0 -> A0
Output
1 -> A1
Calculation
Time = 8
Output
1 -> A0
Output
0 -> A1
Calculation
Time = 4
Read Start Button - set loop exit variable to 1
Start_Timer=sw_push_rnd_pnl1::ReadState()
Debounce Delay on Start switch
50 ms
END
Set to loop until the start button is pressed
Read the state of the set switch
Decide if the switch is pressed
Change the binary state of the variable (very efficient code)
Based upon state, turn on / off LEDs and set timer start time (in minutes)
Check if start switch is pressed. If so the control loop with exit

2. Starting the timer

As you can see in the previous flowchart. We had a second switch that should be pressed to start the timer.

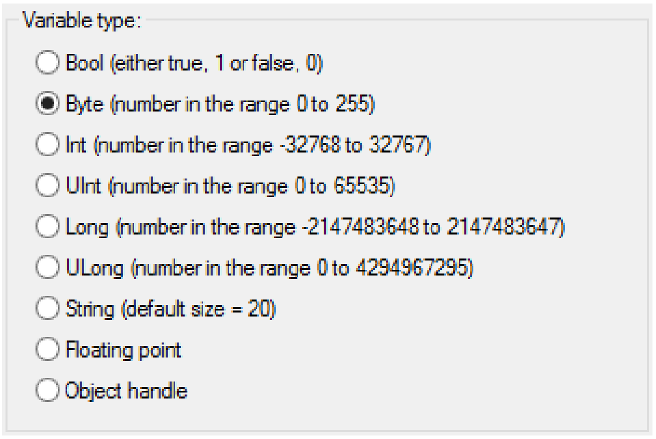
Questions:

A. Work through the loop construct to work out what the variable was called for the determining whether the start switch was pressed

____

B. Consider what type of variable this is. The better the choice of variable type, the more efficient your code

____



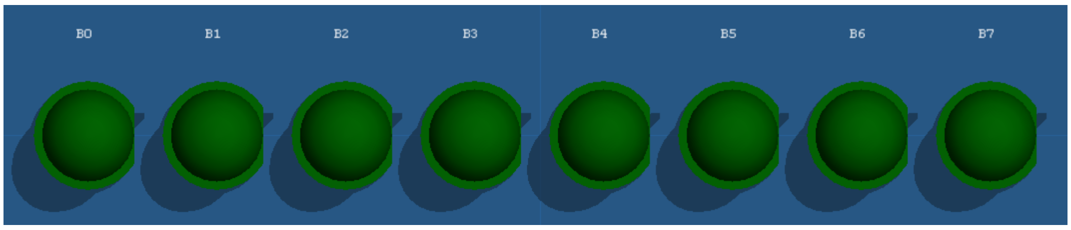
Variable type:

- ◯ Bool (either true, 1 or false, 0)
- ● Byte (number in the range 0 to 255)
- ◯ Int (number in the range -32768 to 32767)
- ◯ UInt (number in the range 0 to 65535)
- ◯ Long (number in the range -2147483648 to 2147483647)
- ◯ ULong (number in the range 0 to 4294967295)
- ◯ String (default size = 20)
- ◯ Floating point
- ◯ Object handle

3. Show the time remaining

Since we are only dealing with an egg timer and we decided that the highest time required was 8 minutes, we can use a binary notation on the pins of 1 port.

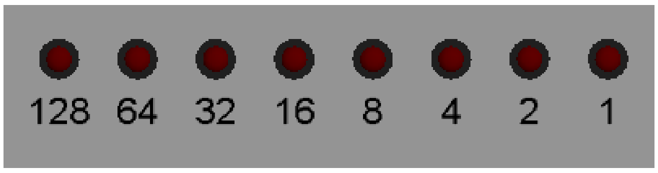
For our design we chose LEDs to display the binary notation, but we could have used a component called a 7 segment display. *You may want to investigate this and decide to use this type of component if you want to do more than 8 minutes or have a variable timer / alarm.*



There are 2 ways of dealing with this. A long winded way is to turn each LED on and then a complex loop of decisions to work out which one to turn on. However, you can use mathematics to also calculate the total number and set the LEDs at the same time by addressing the port.

Refresher on powers

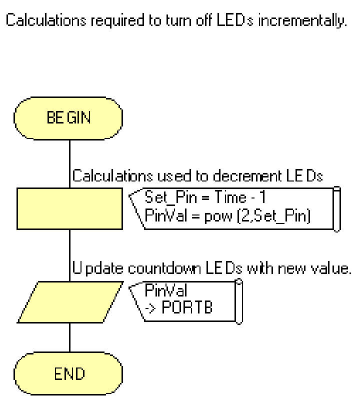
2 to the power of 0 = 1
2 to the power of 1 = 2
2 to the power of 2 = 4
2 to the power of 3 = 8
2 to the power of 4 = 16
…
2 to the power of 7 = 128



See how the pattern follows the binary numbers you can see in Project 1.

With this knowledge, every time we reduce the timer by a whole minute we can use this code.

The works by setting the value on the whole port but only 1 LED will come on

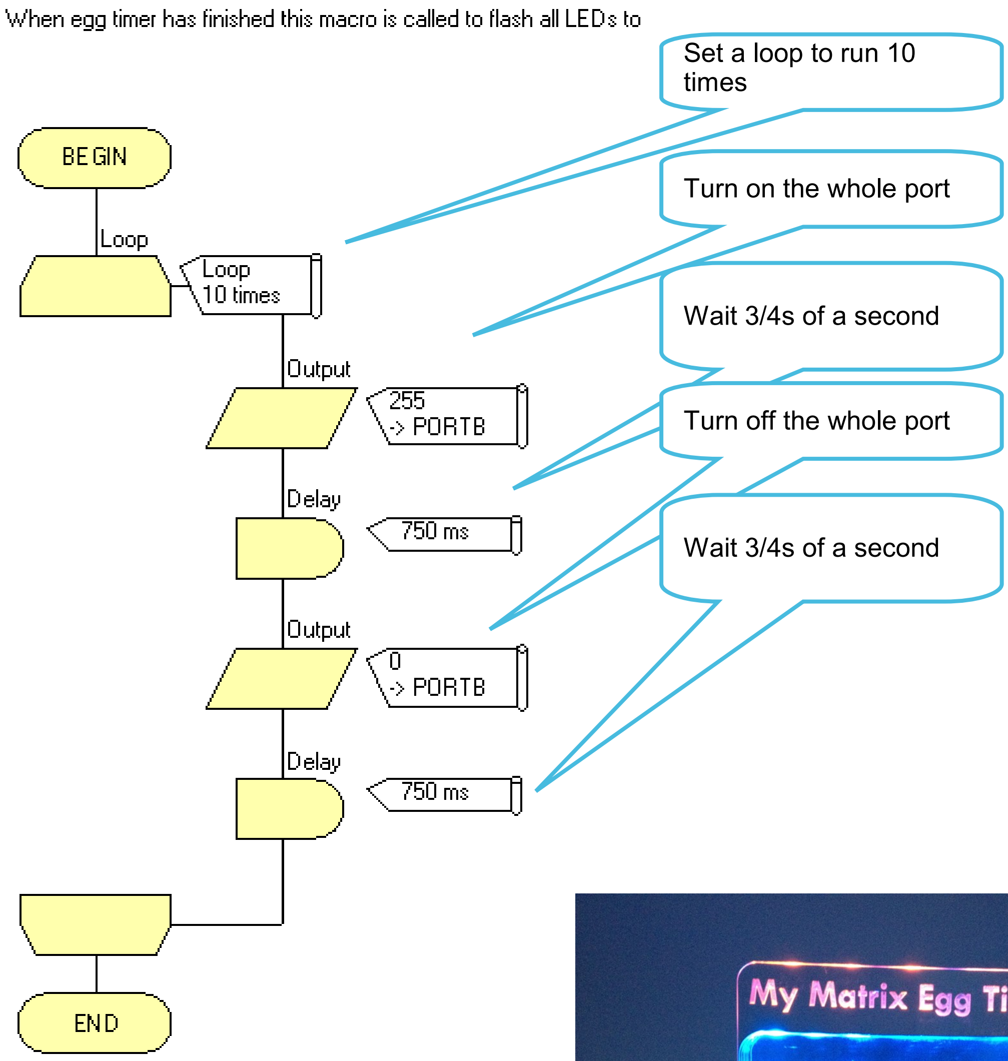

Calculations required to turn off LEDs incrementally.
BEGIN
Calculations used to decrement LEDs
Set_Pin = Time - 1
PinVal = pow (2,Set_Pin)
Update countdown LEDs with new value.
PinVal
-> PORTB
END

Project 2
The Egg Timer cont...

4. Show that the food is ready

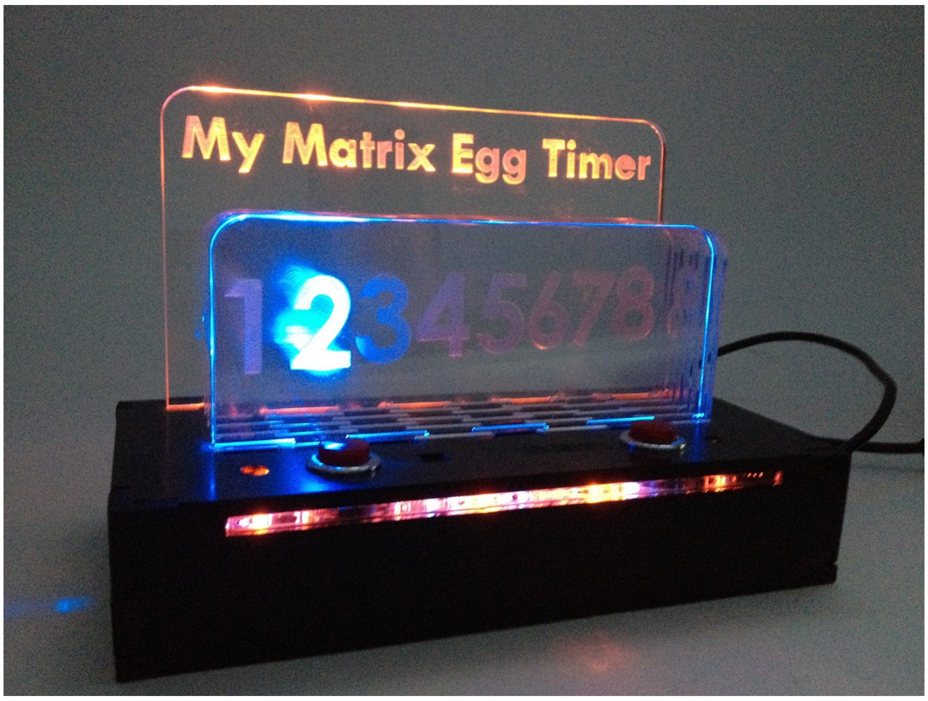
There are many ways an electronics device can get your attention. You could use a simple buzzer, flashing lights or even use a small motor to provide vibration. *You can even combine these to make a multi-sense egg timer.*

In our project, we chose to use the same count down LEDs and a flashing process.



5. Building your device

For the Matrix egg timer we decided to design the body of it in solid works and then cut this out on a laser cutter from various colours of Perspex. Some we made clear and generate a nice edge lit form using high power LEDs soldered on Vera board.



By using an engraving technique on the clear Perspex the bring LED lights up each number in turn. Here we have used 8 planes of Perspex with a small air gap between each panel to avoid light leakage across multiple numbers.

Things to think about with the electronics…

- Pull down resistors for your switches
- Total LED power vs your supply (particular important for high powered LEDs)
- Voltage dividing to get the right voltage for your particular colour
- Anode and cathode orientation of each LED
- Power supply to your device (here I have just used USB power)

One last thing…

When your device is working, can you time a second batch of eggs without rebooting the micro controller? Think how you need to ensure this is possible with your program code!

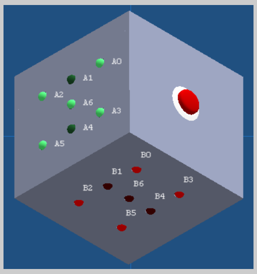
FLOWCODE 6

Project 3
Random Dice Generator

Design and Make 2

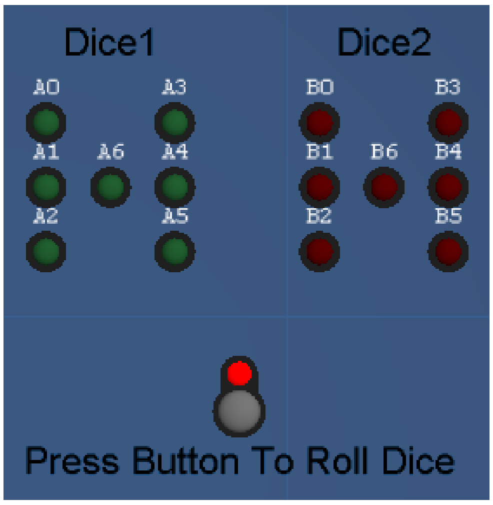
Boggle, Monopoly, Snakes & Ladders or Backgammon, they all use dice. In this project we will show you an overview of how to create an electronic random dice generator.

Since this is Project 3, we will show you in far less detail than previous projects and we will not show you how to create the device in hardware. But use it to inspire your own projects.



Random Dice Generator:

In this example we want to create a random dice generator. When a button is pressed, two 'sets' of LEDs will be used to display random numbers on the dice. We will only run through how to get the numbers displayed on the LEDs. The rest is up to you to work out how it should be done!

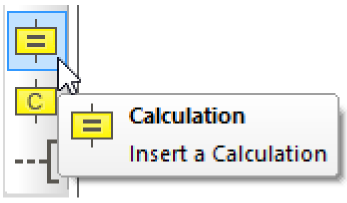


How will this work?

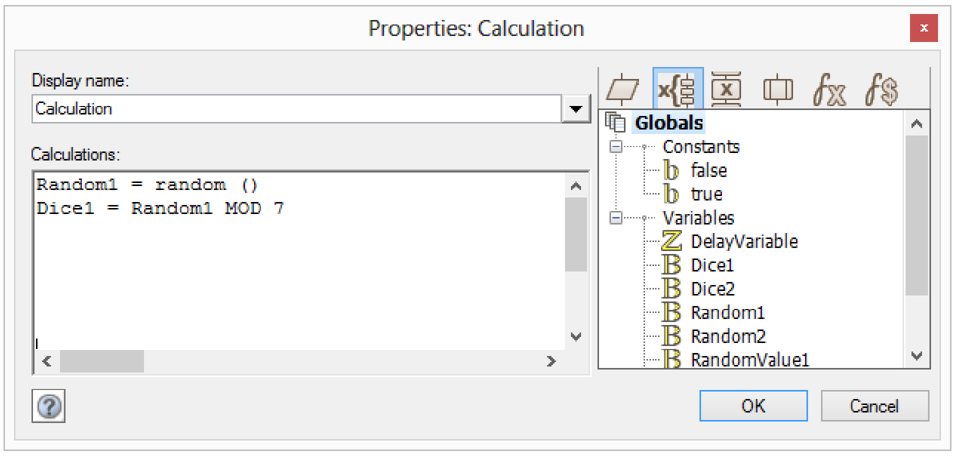
Each set of LEDs will be configured to match that of a dice formation, as seen in this image. When we press the switch we first want a sequence of random numbers to flash on the dice, before a final value is set and displayed.

Creating a random dice value;

Each time we press the button we want a random value to be presented on the dice. We achieve this using the 'random()' function within Flowcode. We create a variable called Dice1 which is used to store our random values. To generate a random value we use a calculation box;



With our calculation box on our flowchart we can use the random() function to generate our number.



```
Random1 = random ()
Dice1 = Random1 MOD 7
```

We first create a random number. The random() function will create a random value between 0 -255 (an 8-bit number). We save it to the variable 'Random1'.

However, this is no use to us as a dice can only display values 1-6. Therefore we need a second function to take our random number and limit it to a value no greater than 6. We achieve this using the Modulo (MOD) function. We will provide one example first, then you can work through the rest. We want to calculate X = 7 MOD 2.

Therefore, we want to divide 7 by the modulus (here the value 2) and see what value remains. We don't care how many times 7 can be divided by 2, but only the remainder value. In this example, the remainder after 7 is divided by 2 is '1'. Think about this, three 2's make 6, therefore the difference between 7 and 6 is 1. Try work out some of the other examples below;

(a) X = 9 MOD 2 ______________________________

____

(b) X = 50 MOD 8 ______________________________

____

(c) X = 37 MOD 4 ______________________________

____

(d) X = 251 MOD 7 ______________________________

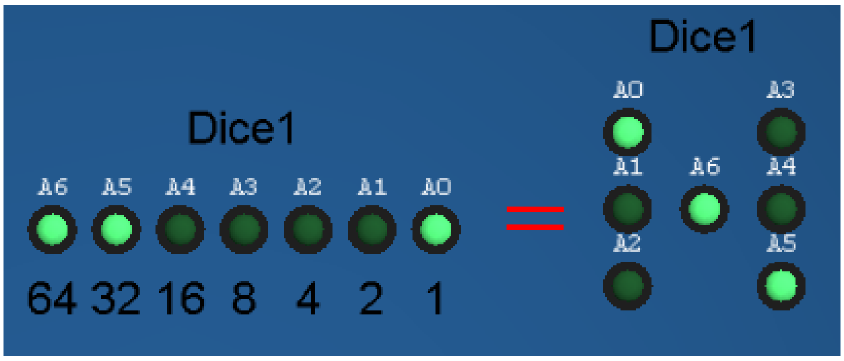
____

Answers:
(a) = 1
(b) = 2
(c) = 1
(d) = 6

Design and Make 2

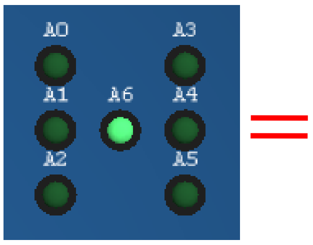
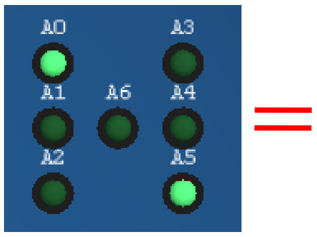
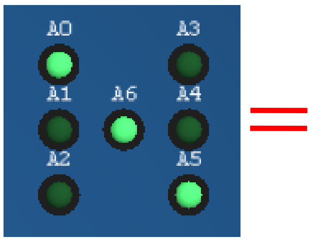
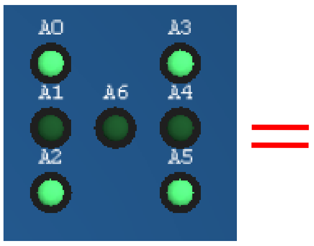
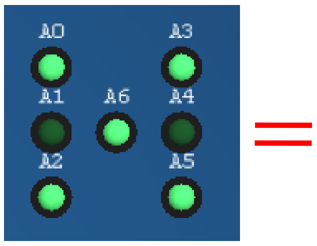
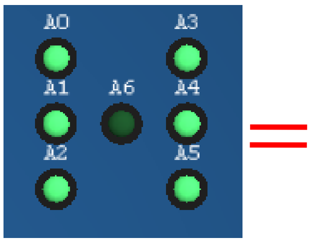
Now that we can create a random number we are almost there with our dice. In this section we will show you how we displayed our generated value on the LEDs of the dice. In 'Project 1 - The Binary Clock' we looked at writing a decimal value between 0-255 on 8 LEDs. We learnt that we can make any of the LEDs illuminate depending on the value we send between 0-255. In this example, we only use 7 LEDs, but the principle is the same. We write a value to the LEDs, but this time the aim is to create a pattern to display a value between 1-6.

In the example below we want to display the value '3', a diagonal line across the face of the dice. We placed our LEDs to replicate that of a dice pattern, as seen in the right of the image. So, from the example below what value should we write to our LEDs? (You will see that we have also aligned our LEDs in a line to make it a little easier to read the value).



Answer: 97

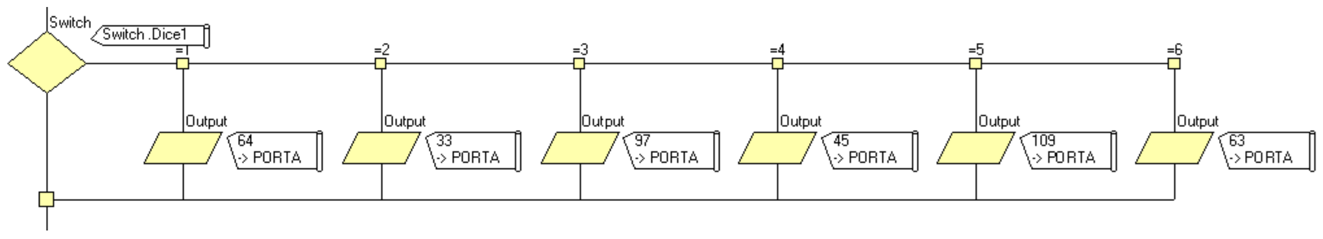
So, if we want to display a '3', we must write the value of 97 to the LEDs. Work through the examples below to calculate the values needed to display the other numbers on the dice:
(Answers are over the page)

On the previous page you should have worked out what values are required to switch on the correct pattern of LEDs to display the values 1-6. We can now check whether your answers were correct.

We used a ‘Switch’ statement to display our dice values. A switch statement is like a group of ‘if’ statements all combined into one programming function. In the example below we have configured it so that only when a value 1-6 is passed it will execute some code. In this case the only code that is executed is updating which LEDs are switched on, by writing a value to PORTA. We want our correct pattern of LEDs to illuminate, depending on the value sent.

On the previous page our example dice value was 3 which equated to the number 97 to be displayed correctly. Check your other values to see whether they match those by studying the switch statement below:



Now we can display dice patterns on a series of LEDs the rest of the project is over to you. Consider how you can improve the dice functionality.

- A Switch should be added so that the user can decide when to ‘roll the dice’.
- Maybe you can also add a second dice?
- Add some code which flashes the dice in a random pattern to simulate a dice roll, then after a few seconds finishes on a value.
- Add a buzzer?

The choice is yours!

Answers:
(1) = 64
(2) = 33
(3) = 97
(4) = 45
(5) = 109
(6) = 63