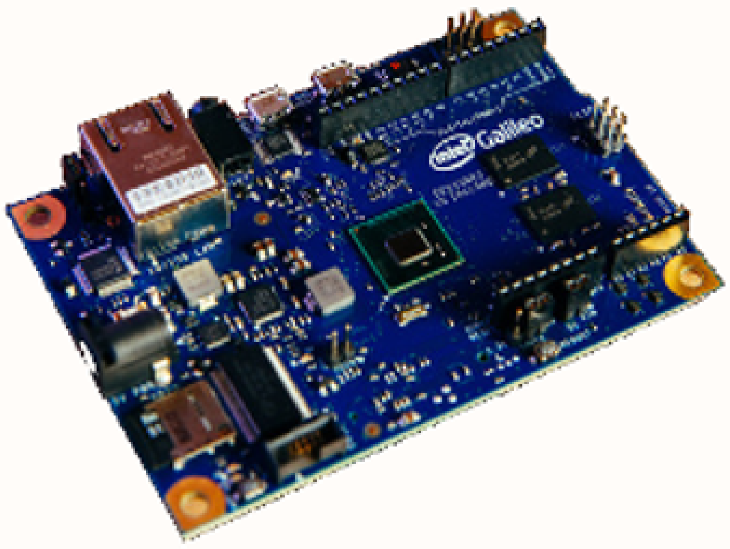


Arduino Activities with the Intel Galileo

Paul Chubb

In creating a work like this, the author is well aware that in many cases they are building on the understandings and work of others freely shared on the Internet. The author acknowledges the contributions of the many in the Arduino communities.

Table of Contents

Introduction

These activities were written as part of a project between Intel, CSIRO and ACT schools – this booklet specifically in the context of the Technology Faculty of Lyneham High School. They are designed to slowly move the student from the simplest circuit to fairly complex circuits that utilise almost all the digital pins on a Galileo. They are designed to allow a teacher to simply supply the students with activity sheets and then concentrate on supporting the student learning.

At the time of writing the National Curriculum for Digital Technology was still in draft. These activities are written against that draft for the two bands covering years 7 to 10. Nothing in this booklet covers all learning outcomes of the curriculum, however the addition of processes around the tasks such as the familiar design, produce, evaluate process or the alternative Demming process that includes reflection on what the student has done, together with other teaching, allow these activities to form a basis for meeting this curriculum. Students, depending on their level and background, may require support, such as code walkthroughs and teaching of digital electronic and coding concepts, to improve their understanding.

Because the audience consists of several different ages and can be used with different levels of student competence, the sample code supplied has been moved to an appendix. This allows the teacher to decide whether to supply the code to the student as soft copy for cut and paste, hard copy for typing and correcting or to not supply it at all.

The text of the activities includes loose pseudocode of the algorithms the author used to write the sample code. In addition, computing and digital electronics concepts are given to the student in information boxes. Where coding is more esoteric or the learning value of the code is low, the author has supplied the code in code tables with the activities.

The licensing of this booklet: Creative Commons Attribution, Share Alike, means that the teacher has a license to use the material in anyway that they choose as long as they attribute and share the content under the same terms. This frees the teacher to print, copy, store or publish to their hearts content.

Hardware and Software

Hardware Requirements

Part	per kit
220 ohm Resistor	3
470 ohm Resistor	8
10K ohm Resistor	5
Red LED	1
RGB LED	1
4 digit 7 Segment display	1
PIR Sensor such as dfrobots SEN0171	1
NPN Transistors (eg BC548s)	4
Piezo Buzzer	1
Push Button	1
Galileo Board	1
Hook up wire	15
Bread board	1
USB 2.0 type A to micro B cable	1

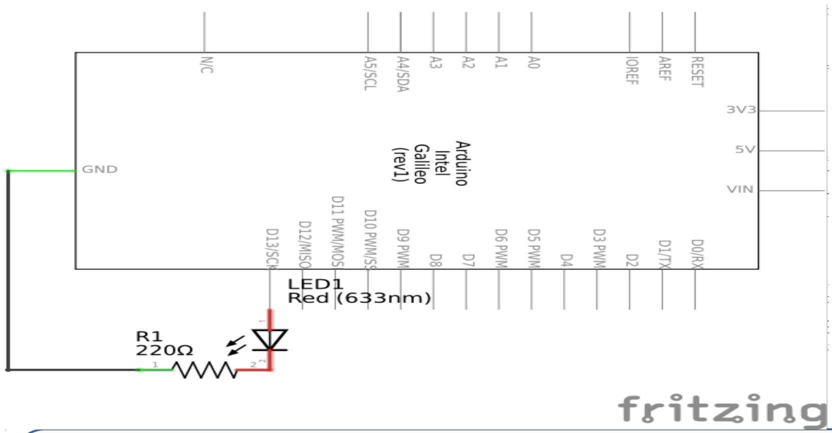
Software Requirements

Fritzing Version 0.9.0 Beta or above which includes the Galileo	Only required if the student need to easily create their own schematics
Arduino IDE 1.53 Intel version or above which supports the Galileo	This is available on the Intel download site. I expect that the main line of the Arduino IDE will soon support the Galileo.
A good code editor such as Gedit	While it is possible to edit code directly in the IDE, it is not a particularly productive process when doing the first cut of code. Windows, which most students will be working with, doesn't supply an appropriate programmer's editor. Gedit is a simple editor which includes syntax highlighting for C and, being free, is a good choice for this purpose.

Activity 1: Blink

In this initial activity you will connect an LED to the Galileo and write a sketch to make it blink.

Schematic



Parts List

- Intel Galileo
- LED of any colour
- Resistor ~220ΩW

Using LEDs

LEDs like all diodes only allow current to run in one direction (arrow in the symbol). This means that if you place them the wrong way around, they will not light up. The longer leg connects to the +ve side and the shorter to the –ve side. If your LED doesn't light up try flipping it around.

Software Sketch

Operations

To make the LED blink you need to:

- Label the pin you are using
- Initialise the pin to do OUTPUT
- Loop around:
    - Toggling the pin high
    - Pausing
    - Toggling the pin low
    - Pausing

Sketches

A basic Arduino programme (called a sketch) consists of two functions:

a) setup() - Place all code to setup for your programme here.
b) Loop() – this function is repeatedly called. This is where your code goes or is called from.

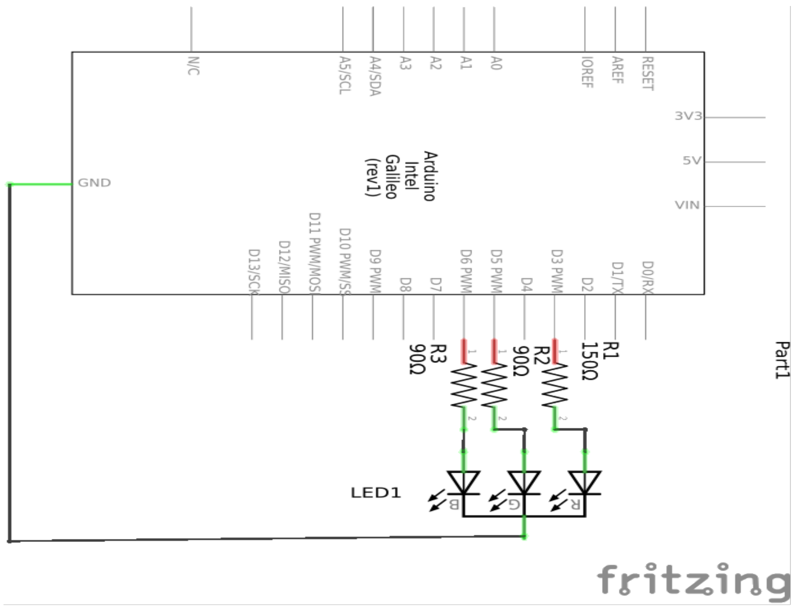
Functions

Function	Example	Purpose
assignment	int iLedPin = 13;	Replace a number with a meaningful label to make it easier to read.
pinMode(<pinnumber>, INPUT | OUTPUT);	pinMode(13, OUTPUT);	Set a pin to either send or receive.
digitalWrite(<pinnumber>, LOW | HIGH);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
delay(<milliseconds>);	delay(1000);	Wait for a period of time.

Activity 2: RGB Blink

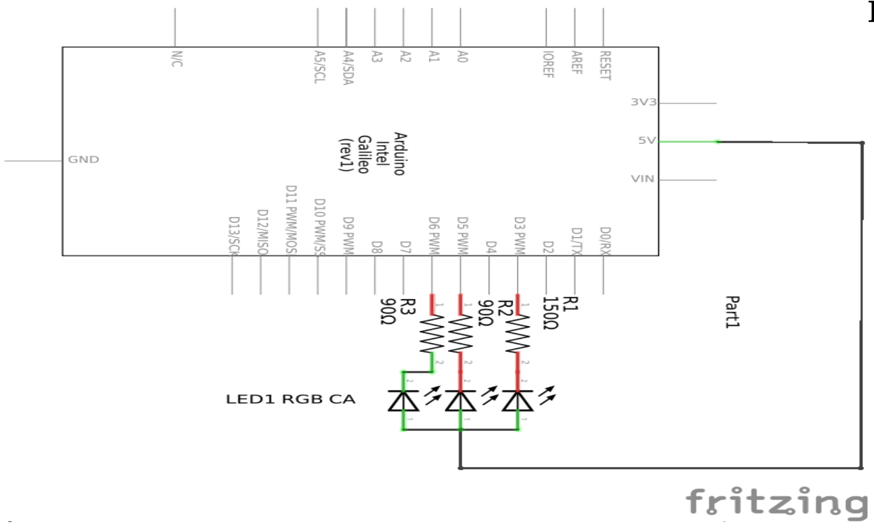
In this activity you will build on your work in activity one by using an RGB LED rather than a single LED. Choose your parts list and schematic based on which type of RGB LED you have.

Schematic



Parts List

1. Intel Galileo board
2. RGB LED (common cathode)
3. 3x220Ω resistors



Parts List

1. Intel Galileo board
2. RGB LED (common anode)
3. 3x220Ω resistors

LED Resisters

Different LEDs require different resistors. To calculate the minimum required:

V_F = (Forward) Voltage of LED
I_F = (Forward) Current of LED
V_S = Supply Current (5v or 3.3v)

$R = (V_S - V_F)/ I_F$

RGB LEDs

RGB LEDs combine red, green and blue LEDs to allow you to have any colour from off (black) to white. They come in two types: common cathode and common anode. Common cathode means they are connected on the negative or ground side. Common anode means they are connected on the positive or power side. For common anode, you toggle the pin low to switch on the light, and high for common cathode.

Software Sketch

Operations

Rewrite the blink sketch from activity 1 to light each LED in turn with a pause between them. **Note if you have a common-anode RGB LED then you will have to switch the pin low before it switches high.**

To make each LED blink you need to:

- Label the pin you are using
- Initialise the pin to do OUTPUT
- Loop around:
    - For each colour:
        - Toggling the pin high
        - Pausing
        - Toggling the pin low
        - Pausing

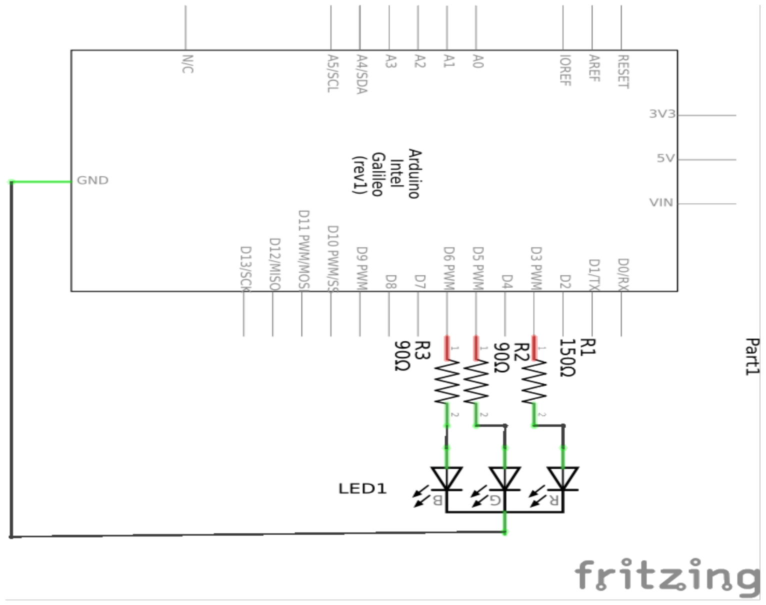
Functions

Function	Example	Purpose
assignment	int iLedPin = 13;	Replace a number with a meaningful label to make it easier to read.
pinMode(<pinnumber>, INPUT | OUTPUT);	pinMode(13, OUTPUT);	Set a pin to either send or receive.
digitalWrite(<pinnumber>, LOW | HIGH);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
delay(<milliseconds>);	delay(1000);	Wait for a period of time.

Activity 3: Counting by Colours

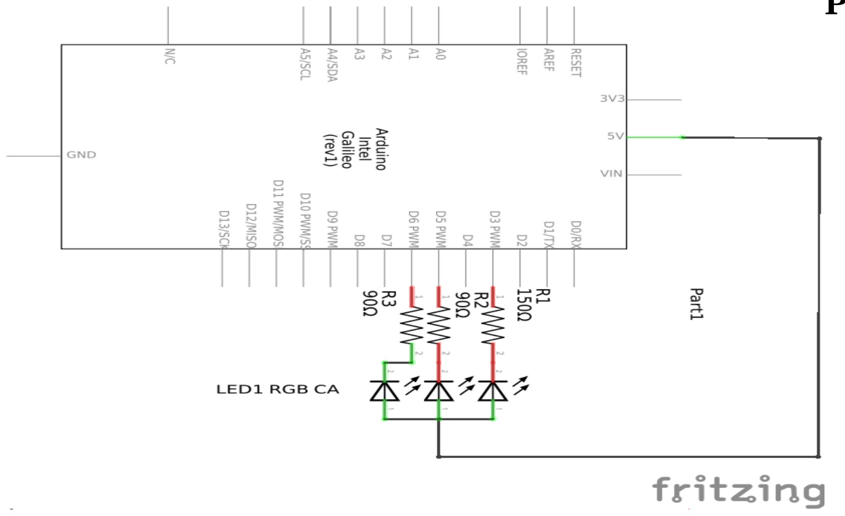
In this activity you will use an RGB LED to simulate a counting number. If you count up in decimal, after you reach 9 you add one to the next column making 10. In this activity you will add "ones" of brightness and once you reach a maximum will add a "one" to the next colour.

Schematic



Parts List

- Intel Galileo board
- RGB LED (common cathode)
- 3x220Ω resistors



Parts List

- Intel Galileo board
- RGB LED (common anode)
- 3x220Ω resistors

Bitwise Operations

One quick and easy way to pull a specific 8 bits out of 32 is to use a bitmask and bit operators. You use a number with each bit of the portion you want set and a bitwise "and" (&).to extract it. Using shift left (<<) or shift right (>>) moves the bits you now have to where you want them. For example we have a long which is four bytes:

```
    10110110 11110011 00010011 11001110 & 00000000 00000000 11111111 00000000
=   00000000 00000000 00010011 00000000
```

but we now want that value in the far right position so we shift it right 8 places

```
    00000000 00000000 00010011 00000000 >> 8
=   00000000 00000000 00000000 00010011
```

Software Sketch

Operations

We use an adding algorithm to change the colours. To do this, you will use some bitwise operations. These functions extract the bits you need for you.

To count in colours:

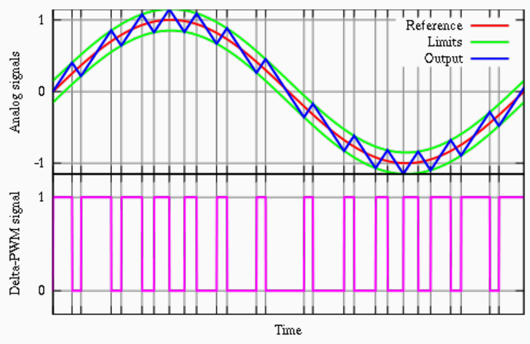
- Setup the three pins to output
- Loop
    - Add a step value to your colour
    - If the colour is now greater than lLimit, set it to zero.
    - Analog write the value of the first byte of colour suitably corrected to the blue pin.
    - Analog write the value of the second byte of colour suitably corrected to the green pin.
    - Analog write the value of the third byte of colour suitably corrected to the red pin.

Byte Extractors

```
// this is the maximum number you count to
long lLimit = 16777216;


byte GetFirstByte(long lNumber) {
long lFirstByteMask = 255;

   return lFirstByteMask & lNumber;
}

byte GetSecondByte(long lNumber) {
long lSecondByteMask = 65280;

   return (lSecondByteMask & lNumber) >> 8;
}

byte GetThirdByte(long lNumber) {
long lThirdByteMask = 16711680;

   return (lThirdByteMask & lNumber) >> 16;
}
```

Pulse Width Modulation (PWM)

Pulse Width Modulation is a way to control things like LED brightness or motor speed by simulating different voltages on a digital pin. Only certain digital pins on Arduino compatible boards, including the Galileo, support PWM. Doing an analogWrite to one of these pins causes that pin to send out a square wave which simulates a voltage. The square wave switches the pin from high (usually 5v) to low (0v) very fast. The highs and lows are fast enough to average the voltage. So the more highs in the square wave, the higher the simulated voltage, the more lows, the lower the simulated voltage. The device – say an LED – attached to the pin will then brighten or dim accordingly.



CC-BY, SA Cyril Buttay, wikimedia

The term “dutycycle” is used to describe the amount of time that the voltage is high. So for a dutycycle of 25%, 75% of the time the voltage will be low and 25% of the time the voltage would be high.

Functions

Function	Example	Purpose

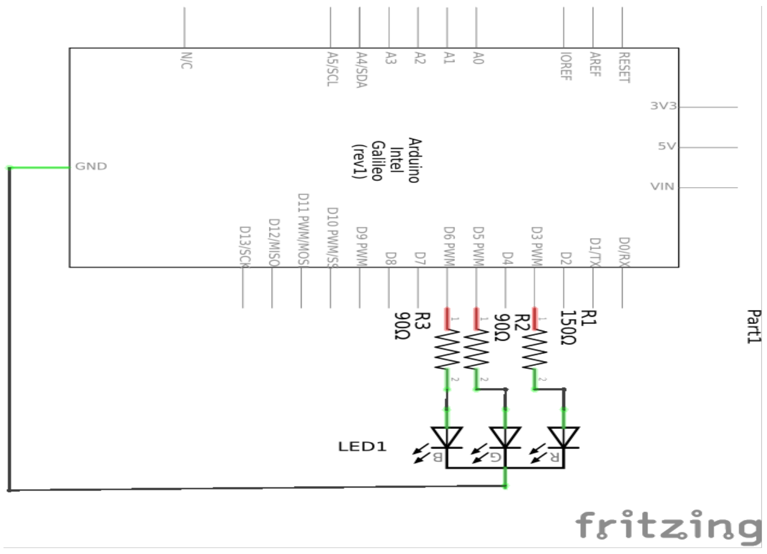
if (<condition>) <statement>;	if (lColour >= lLimit)
	    lColour = 0;
		Test something and act on the result.

analogWrite(<pin number>, <speed>);	analogWrite(iBluePin, bBlue);	Start PWM on a pin at a specified speed.
<variable> +=<value>	lColour += bStep;	Add value to the contents of variable and assign the result to variable
digitalWrite(<pinnumber>, LOW | HIGH);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
delay(<milliseconds>);	delay(1000);	Wait for a period of time.
pinMode(<pinnumber>, INPUT | OUTPUT);	pinMode(13, OUTPUT);	Set a pin to either send or receive.

Activity 4: Mood Lamp

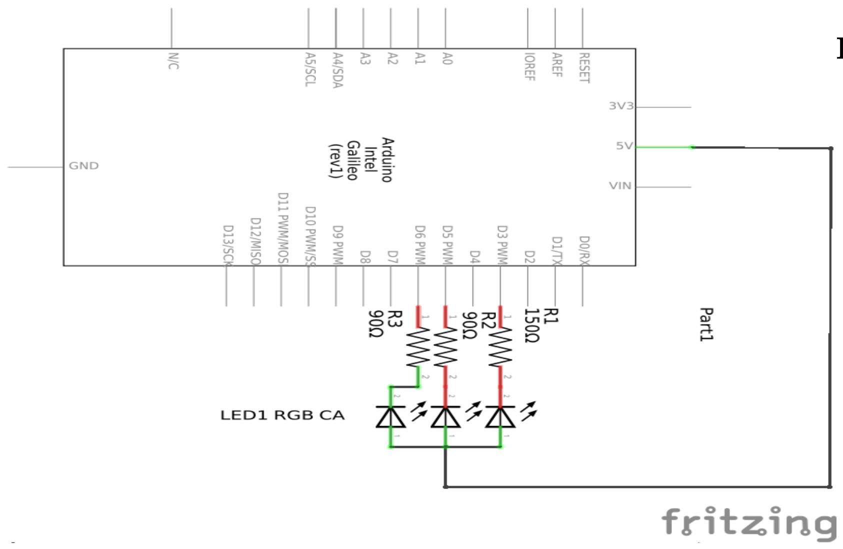
A mood lamp randomly fades between different colours. For best results with your mood light you may need to tune your resistors so that each light is the same intensity. Secondly if the RGB LED is clear you may need to diffuse it. Either scratch the surface of the LED with fine sandpaper or wet and dry paper (ask your teacher first) or wrap a bit of paper around it. Something like cooking greaseproof paper works well.

Schematic



Parts List

- Intel Galileo board
- RGB LED (common cathode)
- 3x220Ω resistors



Parts List

- Intel Galileo board
- RGB LED (common anode)
- 3x220Ω resistors

Software Sketch

Operations

You will need the following functions:

SetTarget()

- call random to get a random colour
- set a target for each of red, green and blue by extracting the third, second and first byte of your random target respectively
- Set the direction from where each colour is to each target – positive 1 for counting up, and -1 for counting down.

Setup()

- Prepare each pin for output
- seed the random number generator by doing an analogread from an unconnected pin
- set the targets

loop()

- if we have reached our target colour
    - Write out our target colour on each pin
    - delay to admire the colour
    - set the next target
- check each of the colours and if it hasn't reached its target add its direction variable
- write each pin
- delay a small amount of time to slow it all down

Pseudo Random Numbers

Computers don't really do random numbers. A random number generator tends to generate numbers that seem at first glance to be random but really follow a pattern. This is why when you use shuffle on your music player, often the song selection seems similar.

To help avoid this problem, most random number generators have the ability to be started with a truly random number. Arduino's "randomSeed" takes a random number and starts the generator and thus gets a more truly random sequence of numbers.

Getting a truly random number is in itself a challenge. Real world applications have been known to use radio wave static or even weather patterns. In this case we use the uncertain analog value on an unconnected pin. This is easy and gives a fairly random number.

Functions

Function	Example	Purpose

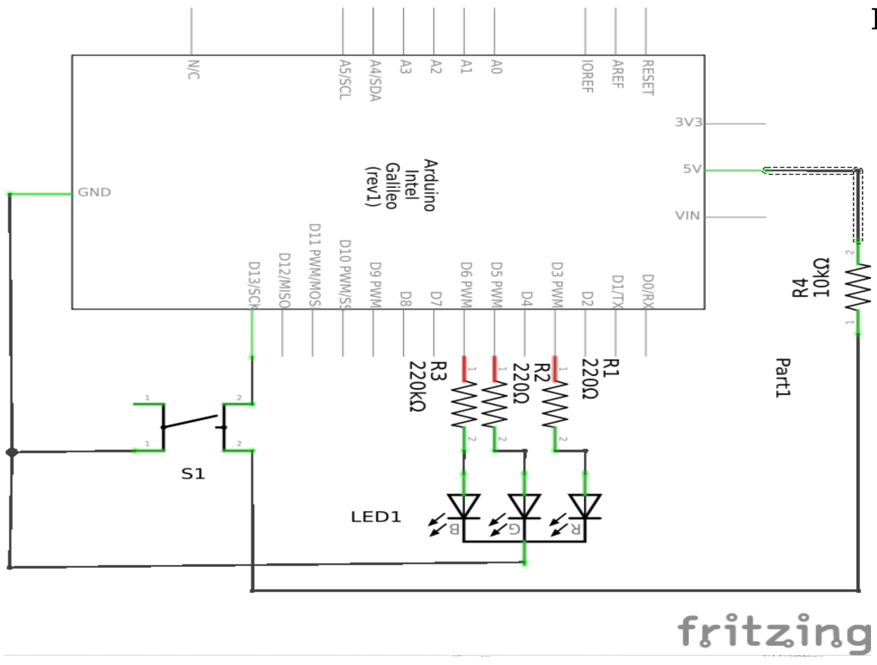
if (<condition>) <statement>;	if (lColour >= lLimit)
	    lColour = 0;
		Test something and act on the result.

analogWrite(<pin number>, <speed>);	analogWrite(iBluePin, bBlue);	Start PWM on a pin at a specified speed.
<variable> +=*<value>*	lColour += bStep;	Add value to the contents of variable and assign the result to variable
digitalWrite(<pinnumber>, LOW | HIGH);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
delay(<milliseconds>);	delay(1000);	Wait for a period of time.
randomSeed(<seed value>);	randomSeed(analogRead(0));	Sets the random number generator to be really random by supplying a seed
pinMode(<pinnumber>, [INPUT | OUTPUT]);	pinMode(13, OUTPUT);	Set a pin to either send or receive.
long random(<uppervalue + 1>);	lTarget = random(lLimit);	Provides a pseudo random number between 0 and <uppervalue>
int analogRead(<pin number>);	iVal = analogRead(0);	Reads the voltage on a specified pin as a digital number.

Activity 5: Light Switching

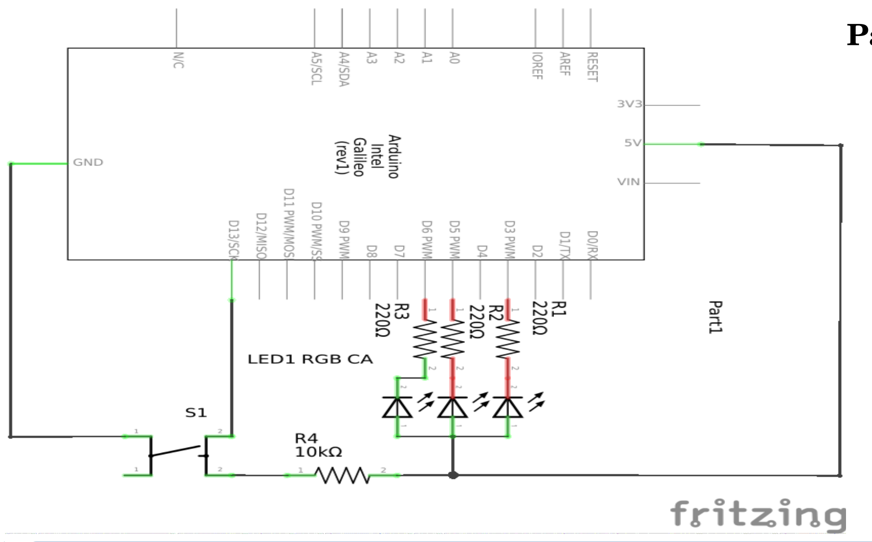
In this activity you will create software and hardware to switch between the different colours of an RGB LED.

Schematic



Parts List

- Intel Galileo board
- RGB LED (common cathode)
- 3x220Ω resistors
- 1x10KΩ
- Push button

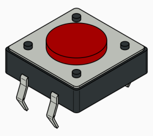


Parts List

- Intel Galileo board
- RGB LED (common anode)
- 3x220Ω resistors
- 1x10KΩ
- Push button

Push Buttons

These buttons make a connection between the two pins on one side with the two pins on the other side. Straighten the pins for best results in a breadboard.



Pull up and pull down resistors

Digital circuits such as the Intel Galileo read ones and zeros by checking the voltage. If the voltage is not what is expected then the result is uncertain. Pull up and pull down resistors force the voltage to expected levels when moving between one and zero. Generally pull up and pull down resistors are 10K Ω. This value fits most situations.

For example: a circuit with pull up resistor connected to a 5v source will force the voltage to 5v when the circuit is not connected to ground. The effect of ground is much more powerful than the effect of the voltage across the pull up resistor so when connected to ground, the voltage in the circuit will be 0v.

A pull down resistor works in a similar way except it pulls the voltage to 0v except when a 5v source is connected.

Software Sketch

Operations

To Setup:

- Set the mode to output on each LED pin.
- Set the mode to input on the switch pin.
- Set the LED to red on to start with.
- Set other LED pins to off

To Cycle:

- Debounce the switch and check it is stable
- check if the switch has changed
    - if the switch is on
        - Turn the current colour off
        - if the current colour is red, set it to green
        - if the current colour is green set it to blue
        - if the current colour is neither red nor green set it to red
        - Turn the current colour on.
    - Remember what state the button was in.

Debouncing

From the Galileo's viewpoint, switches don't change cleanly. During a change they will bounce from on to off and back again several times. Code needs to test for a stable state before acting on the switch state.

One way is to read the switch, wait for a time, say 50ms, then read it again and check if it is the same. If it is then we can believe the switch.

Functions

Function	Example	Purpose

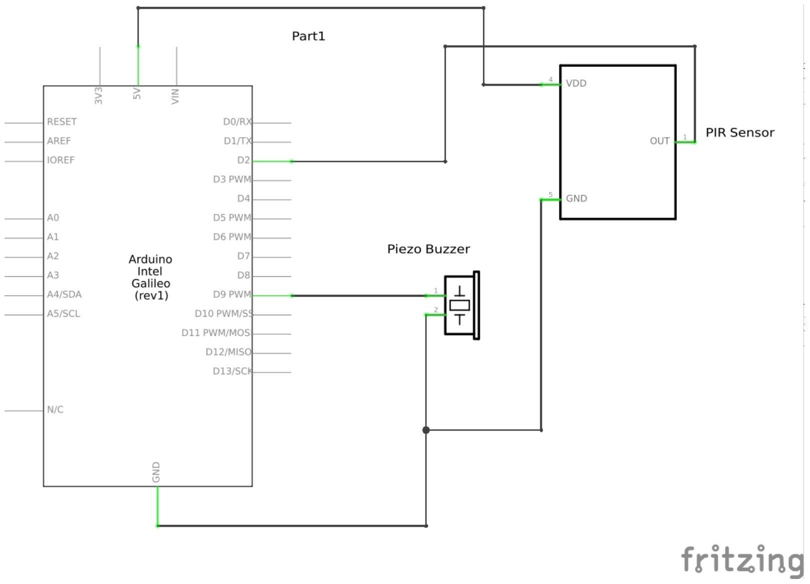
if (<condition>) <statement>;
else <statement>;
	if (lColour >= lLimit) lColour = 0;
	else lColour = 1;
		Test something and act on the result.

digitalWrite(<pinnumber>, LOW | HIGH);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
pinMode(<pinnumber>, INPUT | OUTPUT);	pinMode(13, OUTPUT);	Set a pin to either send or receive.
[LOW | HIGH] digitalRead(<pinnumber>);	If (digitalRead(13) == HIGH) lColour = 0;	Reads whether a pin is high or low

Activity 6: Infrared Alarm

In this activity you will use a passive infrared motion detector (PIR) and a piezo buzzer to make a sound when someone moves.

Schematic



Parts List

- Intel Galileo
- Piezo Buzzer
- PIR such as dfrobot's SEN0171W

Software Sketch

Operations

To make a noise when the motion detector is activated:

- Set the buzzer pin to output and the sensor pin to input
- Loop and:
  - check to see if the sensor pin is high.
  - If the sensor pin is high
    - analogWrite the buzzer pin at a selected duty cycle (1 - 254).
    - Delay a while
    - analog write the buzzer pin with 0.

Piezo Buzzers

These are very simple noise makers but can be made softer or louder using the PWM duty cycle. The maximum volume is reached at about 50%. Their tone or frequency can be changed and melodies can be played using such commands as tone().

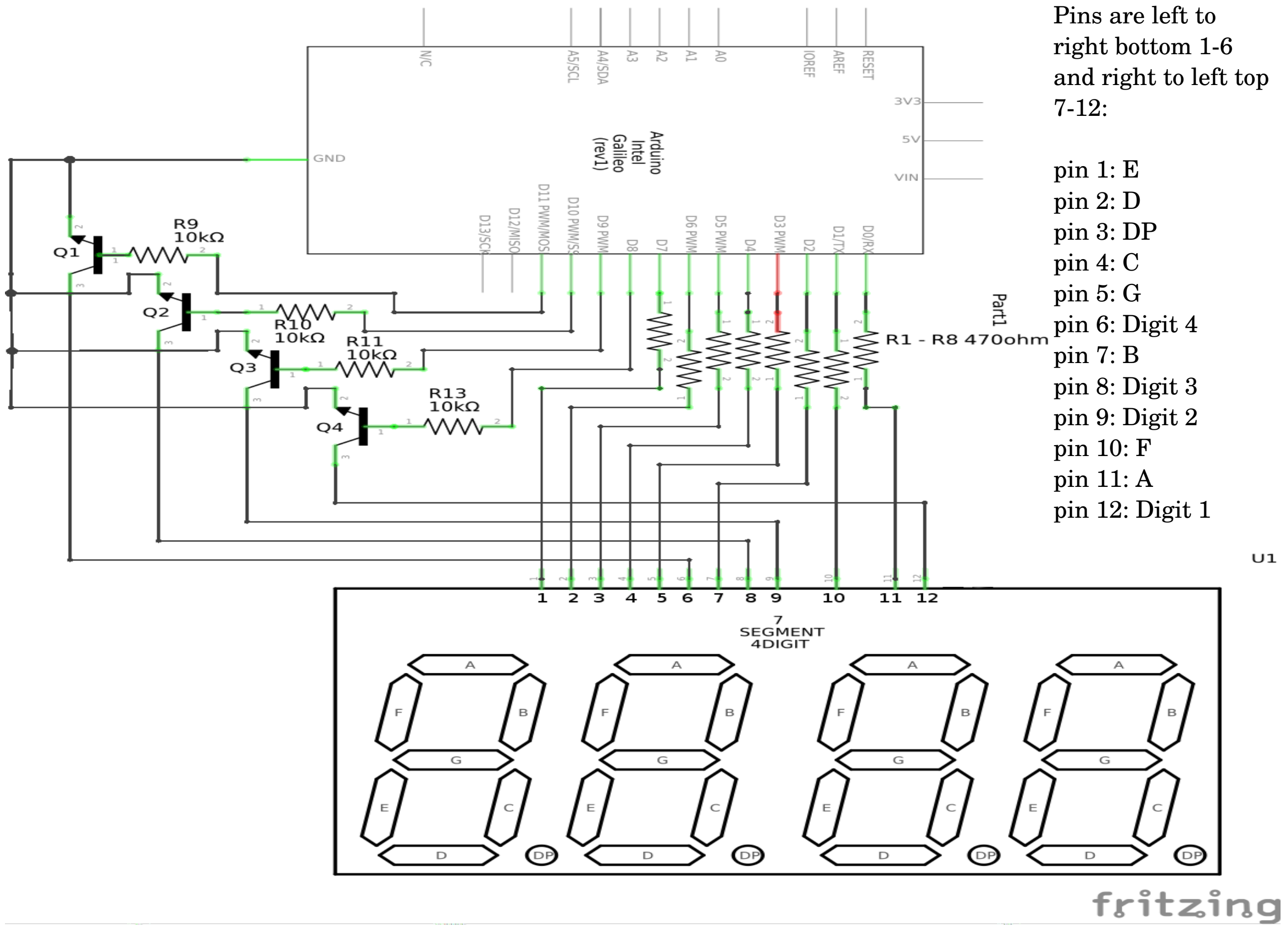
Functions

Function	Example	Purpose
analogWrite(<pin number>, <speed>);	analogWrite(iBluePin, bBlue);	Start PWM on a pin at a specified speed.
delay(<milliseconds>);	delay(1000);	Wait for a period of time.
pinMode(<pinnumber>, [INPUT | OUTPUT]);	pinMode(13, OUTPUT);	Set a pin to either send or receive.

Activity 7: LED Display

In this activity you will create hardware and software to cycle through all LEDs on a 4 digit, 7 segment common cathode display.

Schematic



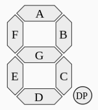
5461AS
Pins are left to right bottom 1-6 and right to left top 7-12:

pin 1: E
pin 2: D
pin 3: DP
pin 4: C
pin 5: G
pin 6: Digit 4
pin 7: B
pin 8: Digit 3
pin 9: Digit 2
pin 10: F
pin 11: A
pin 12: Digit 1

Parts:

- Intel Galileo board
- 4 digit, 7 segment display such as 5461AS
- 8x470Ω resistors
- 4x10KΩ resistors
- 4xNPN Transistors such as 548s

7 Segment Displays

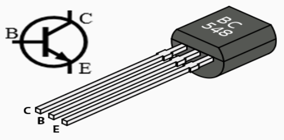


CC-BY, SA: Wikipedia Commons

These consist of one LED per segment that make up the digit, often with a common anode or cathode. Where they are connected into multi digit displays, the common anode or cathode for each digit is attached to one pin while the individual segments share the same pins for that segment on each digit. To switch on a segment, you select the common anode or cathode pin on that digit, then the common segment pin.

Transistors

We are using transistors as tiny switches to handle the current coming from each digit of the display. A pin of the Galileo can probably not handle (sink) the required amount of current to control a digit so we switch the current through a transistor.



Software Sketch

Operations

Create Global variables:

- Create an array of integers holding the pins numbers connected to segments.
- Create an array of integers holding the pin numbers controlling digits

Setup():

- Use a for loop to set the mode on each segment pin to output.
- Use a for loop to set the mode on each digit pin to output.

Loop():

1. For each digit in the digit array
   - make the digit pin high
   - for each segment in the segment array
     - make the segment pin high
     - delay for say 500ms
     - make the segment pin low
   - make the digit pin low.

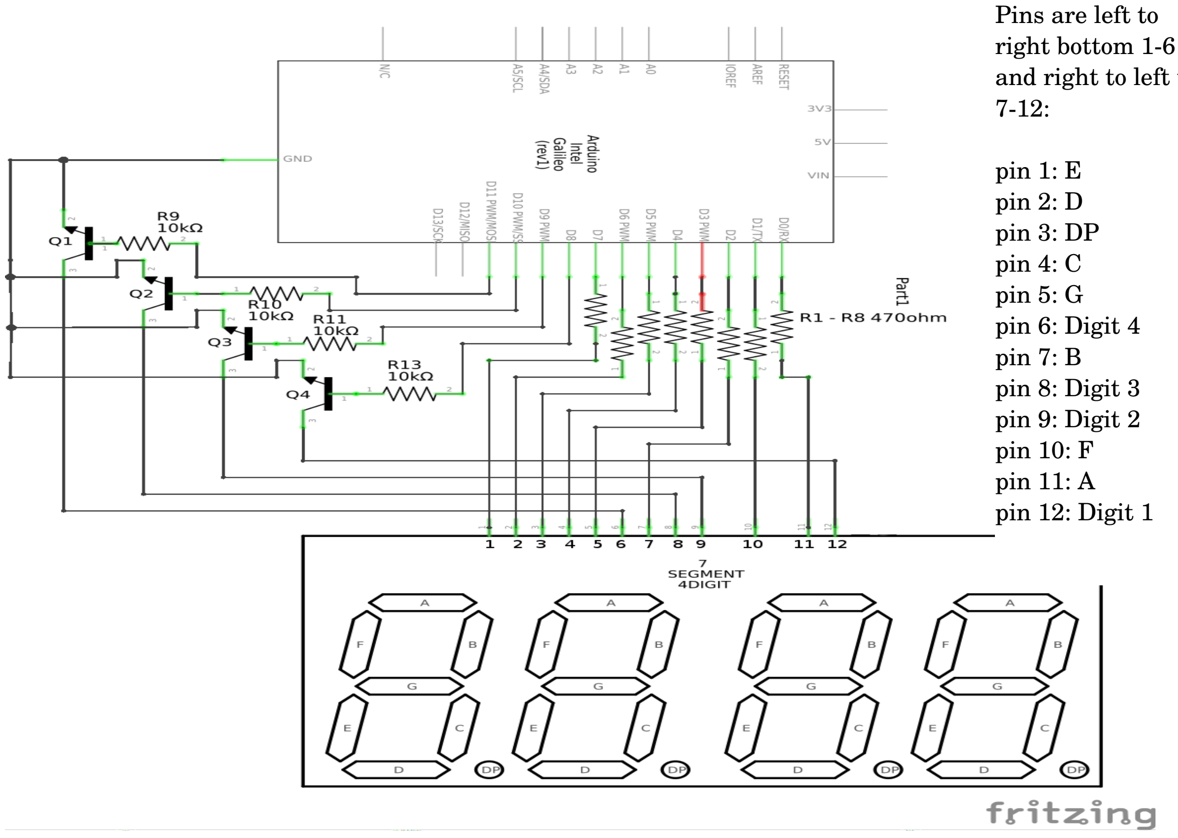
Functions

Function	Example	Purpose
<type><varname>[] = {<element>,<element>,...};	int iaSegs[] = {0, 2, 4. 6, 7, 1, 3, 5};	Create an array with some values
digitalWrite(<pinnumber>, LOW | HIGH);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
pinMode(<pinnumber>, INPUT | OUTPUT);	pinMode(iaSegs[i],OUTPUT);	Set a pin to either send or receive.
for ([<Start value>]; [<end test]; [<change op>]) <statement>;	for (i = 0; i < 50; i++) { … }	For loop. Starts with i equal to 0, stops when i is equal to 50 and i increases by 1 each loop.
delay(<milliseconds>);	delay(500);	Waits the specified number of milliseconds

Activity 8: Marquee

A marquee in computer terms is a scrolling area of text. In this exercise you will attempt to scroll a message across your 4 digit display. However some letters are impossible to form and you only have four digits – good luck.

Schematic



5461AS

Pins are left to right bottom 1-6 and right to left top 7-12:

pin 1: E
pin 2: D
pin 3: DP
pin 4: C
pin 5: G
pin 6: Digit 4
pin 7: B
pin 8: Digit 3
pin 9: Digit 2
pin 10: F
pin 11: A
pin 12: Digit 1

Software Sketch

Parts:

- Intel Galileo board
- 4 digit, 7 segment display such as 5461AS
- 8x470Ω resistors
- 4x10KΩ resistors
- 4xNPN Transistors such as 548s

Operations

- Create Global variables:
- Create an array of integers holding the pins numbers connected to segments.
- Create an array of integers holding the pin numbers controlling digits
- Create an array of highs and lows to store your message (see note).
- Create a number of times a digit is displayed in a location say 25 times.

- Setup():

- Use a for loop to set the mode on each segment pin to output.
- Use a for loop to set the mode on each digit pin to output.

What we are doing is selecting 4 letters from our message and moving our selection down the array one letter for each time through. So we need to keep track of where we are in the array and use the digit pin number to select the next letter: digit 0 will get the current location, digit 1 will get the letter after the current location etc. To control speed and to ensure we see the digit before it is gone, we need to display each digit a number of times – 25 times worked for me.

Example Message Array

```
// USB id for Galileo is 8086:babe
int L = LOW;
int H = HIGH;
int iaMessage[][8] = {
            {L, L, L, L, L, L, L, L},     // all off
            {L, L, L, L, L, L, L, L},     // all off
            {L, L, L, L, L, L, L, L},     // all off
            {L, L, L, L, L, L, L, L},     // all off
            {L, L, L, L, L, L, H, L},     // dash
            {L, L, L, L, L, L, H, L},     // dash
            {L, L, L, L, L, L, H, L},     // dash
            {L, L, L, L, L, L, H, L},     // dash
            {L, L, H, H, H, H, H, L},     // B
            {H, H, H, H, H, L, H, L},     // A
            {L, L, H, H, H, H, H, L},     // B
            {H, L, L, H, H, H, H, L},     // E
            {L, L, L, L, L, L, H, L},     // dash
            {L, L, L, L, H, H, L, L},     // I
            {H, L, H, H, L, H, H, L},     // S
            {L, L, L, L, L, L, H, L},     // dash
            {L, L, H, H, H, H, H, L},     // B
            {H, H, H, H, H, L, H, L},     // A
            {H, L, L, H, H, H, L, L},     // C
            {L, L, H, H, H, L, H, L},     // O
            {L, L, H, L, H, L, H, L},     // N
            {L, L, L, L, L, L, L, H},     // .
            {L, L, L, L, L, L, L, H},     // .
            {L, L, L, L, L, L, L, H},     // .
};
```

- Loop():
    - For each digit in the digit array
        - make the digit pin high
        - get the corresponding letter checking whether we wrap to the beginning of the message.
        - for each segment in the segment array
            - Make the segment pin high
            - delay for say 1ms
            - Make the segment pin low
        - make the digit pin low.
    - If we have displayed the letter enough times, move our current message location along and wrap if necessary

Persistence of Vision

In TV, films, games and several other things, the device draws and redraws little bits very fast. Fast enough so that the eye is fooled into believing that there has been either no change or a smooth change. In this activity you will draw each segment of each digit one after another. You will switch off each digit to move to the next on. Because you are doing this really fast, it looks like the digit remains lit although it does flicker. With more different code, it is possible to remove this flicker. Try editing your code and replacing delay() with delayMicroseconds().

Functions

Function	Example	Purpose
<type><varname>[] = {<element>,<element>,...};	int iaSegs[] = {0, 2, 4. 6, 7, 1, 3, 5};	Create an array with some values
digitalWrite(<pinnumber>, LOW | HIGH);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
pinMode(<pinnumber>, INPUT | OUTPUT);	pinMode(iaSegs[i],OUTPUT);	Set a pin to either send or receive.
for ([<Start value>]; [<end test]; [<change op>]) <statement>;	for (i = 0; i < 50; i++) { … }	For loop. Starts with i equal to 0, stops when i is equal to 50 and i increases by 1 each loop.
delay(<milliseconds>);	delay(500);	Waits the specified number of milliseconds

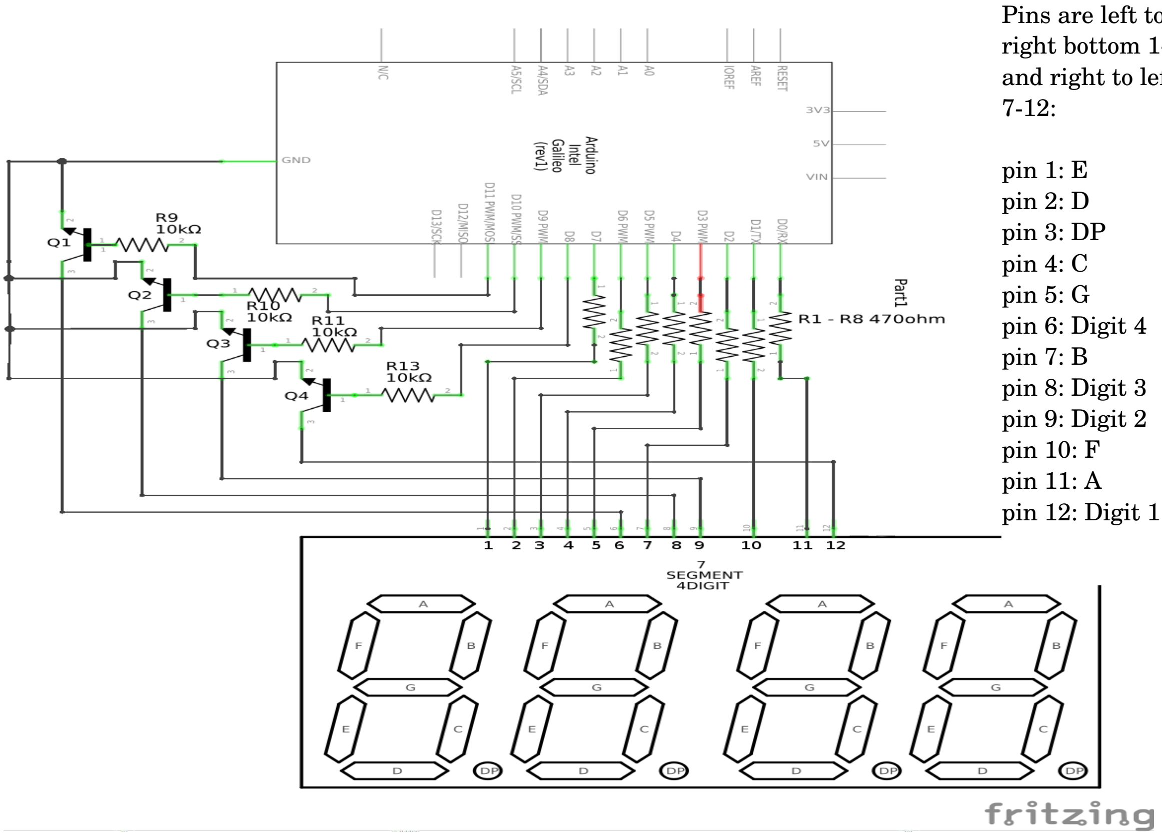
<type><varname>[]
<type><varname>[<number>]
<type><varname>[][<number>]
	Char caString[] = {“a”, “b”, “c”, “d”};
	int iaNumbers[10];
	int iaSegments[][2] ={{1, 2}, {3, 4}};
		Declaring arrays of variables. Think of a string of boxes or a chess board. When no number is specified as the dimension, they have to be initialised as shown. Two lots of square brackets mean two dimensional arrays. More dimensions are possible.

delayMicroseconds(<microseconds>)	delayMicroSeconds(10);	Delay for a number of microseconds.

Activity 9: Real Time Clock

In this activity you will use the built in real time clock on the Galileo and the 4 digit 7 segment LED display to create a digital clock

Schematic



5461AS

Pins are left to right bottom 1-6 and right to left top 7-12:

pin 1: E
pin 2: D
pin 3: DP
pin 4: C
pin 5: G
pin 6: Digit 4
pin 7: B
pin 8: Digit 3
pin 9: Digit 2
pin 10: F
pin 11: A
pin 12: Digit 1

Parts:

- Intel Galileo board
- 4 digit, 7 segment display such as 5461AS
- 8x470Ω resistors
- 4x10KΩ resistors
- 4xNPN Transistors such as 548s

Software Sketch

Operations

- Create Global variables:
- Create an array of integers holding the pins numbers connected to segments.
- Create an array of integers holding the pin numbers controlling digits
- Create an array of highs and lows to store the digits and dp (see note).

Pipes, Date and Linux

Galileo runs linux under the covers. Linux provides a rich environment for you to use. The date command is a good example. Linux uses date to set and get the date and time. /bin/date 114100014 sets the date to 14/11/2014 and the time to 10:00am (MMDDhhmmYY). Normally you would use date from the command line but we want to use it in our code, so we use a pipe to call it. A pipe passes information to a command and then gets what it returns back to your code. See the code box....

Setup()

- Start serial comms at 9600 BAUD.
- Open a pipe to the /bin/date command to set the date and time to the current date and time, then close the pipe (see note).
- Set the pins controlling the digits and the segments to output.

Loop()

- Open a pipe to the /bin/date command to read back the current time, then close it.
- Use a for loop to walk through the digits and return from the /bin/date command.
    - Switch the digit pin high
    - take this digit from the time string and convert it to a number.
    - Use a for loop to walk through the segments to set the number based on the current time digit.
        - Set the current segment according to the segment array for this number
        - Wait for a millisecond
        - Set the segment to low
    - If this is the second digit ie number 1
        - set the decimal point to high
        - wait for a milliseconds
        - set the decimal point to low
    - Switch the digit pin low.

Numeral Array

```
int iaNumbers[][8] = {
                        {H, H, H, H, H, H, L, L},  // 0
                        {L, H, H, L, L, L, L, L},  // 1
                        {H, H, L, H, H, L, H, L},  // 2
                        {H, H, H, H, L, L, H, L},  // 3
                        {L, H, H, L, L, H, H, L},  // 4
                        {H, L, H, H, L, H, H, L},  // 5
                        {H, L, H, H, H, H, H, L},  // 6
                        {H, H, H, L, L, L, L, L},  // 7
                        {H, H, H, H, H, H, H, L},  // 8
                        {H, H, H, L, L, H, H, L},  // 9
                        {L, L, L, L, L, L, L, H},  // DP
};
```

Set the date and time

```
// use date command to set the date and time numbers:
//MMDDhhmmYY
char *cpStartTime = "/bin/date 1114100014";

{ .... }

// in setup()
char *cmd = cpStartTime;
char buf[64];
FILE *ptr;

  // start serial comms
  Serial.begin(9600);

  // set the time
  if ((ptr = popen(cmd, "r")) != NULL)
  {
    while (fgets(buf, 64, ptr) != NULL)
    {
      Serial.print(buf);
    }
  }
  (void) pclose(ptr);
```

Get the time

```
// in loop()
char *cmd = cpRetTime;
char buf[64];
FILE *ptr;

  // read back the time
  if ((ptr = popen(cmd, "r")) != NULL) {
    while (fgets(buf, 64, ptr) != NULL) {
      Serial.print(buf);
    }
  }
  (void) pclose(ptr);
```

Information Storage

While everything is stored as binary, ones and zeros, we have to make sure that one computer's ones and zeros mean the same as another computer's ones and zeros. To do this we have agreed codes and protocols. To store and transmit information two traditional codes are ASCII (American Standard Code for Information Interchange) and EBCDIC (an IBM sponsored competitor). These encode character information and a few control codes. Nowadays things are a bit more complicated with codes designed for many different uses and languages.

In this activity you get a time that is numbers as ASCII characters and you want to use it as numbers as numbers. There are several ways to translate ASCII characters of numbers to numbers. I used a simplistic solution of simply taking away the ASCII value of zero (48).

Functions

Function	Example	Purpose
digitalWrite(<pinnumber>, LOW | HIGH);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
pinMode(<pinnumber>, INPUT | OUTPUT);	pinMode(iaSegs[i],OUTPUT);	Set a pin to either send or receive.
for ([<Start value>]; [<end test]; [<change op>]) <statement>;	for (i = 0; i < 50; i++) { … }	For loop. Starts with i equal to 0, stops when i is equal to 50 and i increases by 1 each loop.
delay(<milliseconds>);	delay(500);	Waits the specified number of milliseconds

<type><varname>[]
<type><varname>[<number>]
<type><varname>[][<number>]
	char caString[] = {“a”,“b”,“c”,“d”};
	int iaNumbers[10];
	int iaSegments[][2] ={{1,2},{3,4}};
		Declaring arrays of variables. Think of a string of boxes or a chess board. When no number is specified as the dimension, they have to be initialised as shown. Two lots of square brackets mean two dimensional arrays. More dimensions are possible.

<fileptr>popen(<cmd>, <fileflags>);
pclose(<fileptr>);
	ptr = popen(“/bin/date +%k%M”);
	pclose(ptr);
		popen opens a pipe to a linux command that is run and the output returned via the ptr.

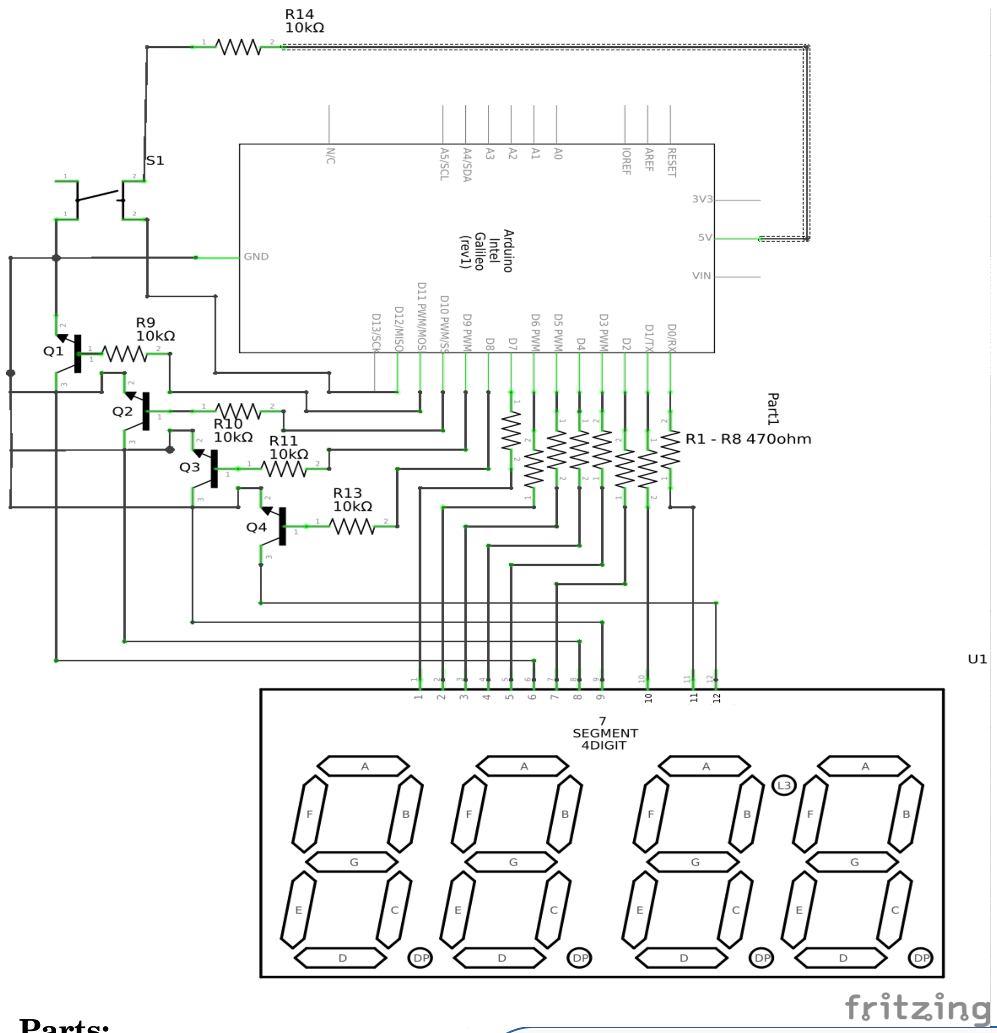
[<string>|<null>]fgets(<buffer>, <length>, <fileptr>);	If (fgets(buf, 64, ptr) == NULL) Serial.print(“finished);	fgets attempts to read a string up to one character less than the stated length. When it finishes it returns NULL.

Serial.begin(<bitrate>);
Serial.print(<string>);
	Serial.begin(9600);
	Serial.print(buf);
		Serial.begin starts serial communications from the Galileo to the computer. Serial.print writes the string back to the computer to the serial monitor.

Activity 10: Poker Machine

In this activity you will create a simple poker machine. When you press a button, the Galileo will display – with suitable effects – a random set of numbers on a four digit display.This is perhaps the most complex activity in both code and electronics.

Schematic



5461AS

Pins are left to right bottom 1-6 and right to left top 7-12:

pin 1: E
pin 2: D
pin 3: DP
pin 4: C
pin 5: G
pin 6: Digit 4
pin 7: B
pin 8: Digit 3
pin 9: Digit 2
pin 10: F
pin 11: A
pin 12: Digit 1

Parts:

- Intel Galileo board
- 4 digit, 7 segment display such as 5461AS
- 8x470Ω resistors
- 5x10KΩ resistors
- push button
- 4xNPN Transistors such as 548s

Structures

Structures, sometimes called records in other computer languages, are a way of gathering together variables that relate to a specific thing. This makes accessing the variables very convenient and improves the readability of code. For example this structure has all the control variables of a digit.

```
struct DIGIT {
  int *paThisNumber;   /
  int iDepth;
  int iBuilding;
  int iCurrent;
  int iTimesSpun;
  int iTimesDisp;
  int iDigit;
} sDigit;
```

Pointers and Pointer Arithmatic

Pointers are one of the hardest things to learn. Think of a row of bags with labels and handles. They are easy to find and pick up. A pointer is a way of adding a handle and label to a variable.

- &iDigit returns a pointer to iDigit

- *aPtr returns what aPtr is pointing at

- aPtr->anElement returns a member of the structure that aPtr is pointing at.

Adding a one to a pointer causes that pointer to point to the next variable, in the bags analogy, the next bag in the row. Say you had a pointer to an array. This would mean that aPtr+1 would point to the next element of that array. It works in a similar manner for other numbers and subtraction.

Software Sketch

Operations

- Create Global variables:
- Create an array of integers holding the pins numbers connected to segments.
- Create an array of integers holding the pin numbers controlling digits
- Create an array of highs and lows to store the digits and dp (see note on activity
- Create constants (defines) for the states of your state machine: Waiting, Building, Built and Prestart.
- Create an array of structures to manage the four digits.

Setup()

- Set the switch pin to input
- Set the segment and digit pins to output
- initialise the digit structure array
- Set each digit to a state of "Prestart"
- Seed the random number generator by reading an empty pin

Start()

Parameters: a pointer to a digit structure, an offset into the number array, the digit number (0 – 3)

- Store
    - a pointer to the number array entry with the highs and lows required for this digit
    - The depth we have displayed ie 0
    - The state ie Waiting
    - The current segment we have built to ie 0
    - The times we have spun ie 0
    - The times we have displayed ie 0
    - The digit number

Scramble()

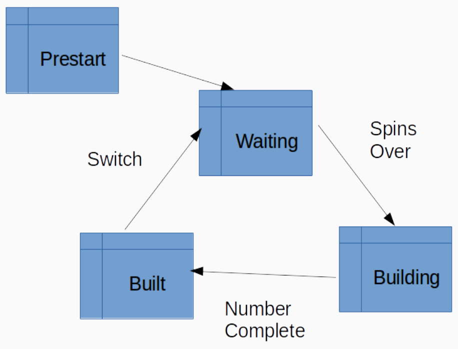
Parameters: a pointer to a digit structure
- For this digit:

- Check if we have displayed all the segments if we have start again and increment the number of spins.
- Open the display digit for output
- if our state is Built or Building
    - use a for loop to switch each segment high or low up to the current depth
    - After all have been switched high or low according to the number array, delay for a millisecond
- If the state is Waiting and we have spun enough times, change our state to Building.
- If our state is not Built
    - switch on the segment indicated by current
    - increment times displayed
    - delay for a millisecond
    - if we have displayed this stuff enough times
        - increment current
        - if the state is Building, increment depth
        - If the depth is at the end of the segments, change our state to Built
    - Set the times displayed to zero
- Close all the segments and this digit

Loop()

- check a digit to find out our state
- If our state is either Built or Prestart
    - Check the switch state
    - If the switch state is not the same as the last switch state
        - store our current time (milliseconds)
        - set Last switch value to current switch value
    - if the current time (milliseconds) is greater than the stored time plus the debounce time
        - If the switch value is low
            - restart each digit structure with a random number under 10
- Call scramble for each digit in the digit structure array.

Finite State Machines



A finite state machine is a way of looking at certain types of problems that make them easier to solve. You define different states or conditions and how the programme gets from one to another. This gives a structure to your programme and the problem you are solving and also makes debugging easier. In the picture we have four states: Prestart ie before we have started, Waiting in which we display pretty patterns, Building in which we morph the patterns into a number and Built in which we display the number. To get from Prestart and Built to Waiting we look for a switch press. To get from Waiting to Building we complete a number of pretty patterns and from Building to Built we have to complete displaying the number.

Functions

Function	Example	Purpose

<type><varname>[]
<type><varname>[<number>]
<type><varname>[][<number>]
	Char caString[] = {“a”, “b”, “c”, “d”};
	int iaNumbers[10];
	int iaSegments[][2] ={{1, 2}, {3, 4}};
		Declaring arrays of variables. Think of a string of boxes or a chess board. When no number is specified as the dimension, they have to be initialised as shown. Two lots of square brackets mean two dimensional arrays. More dimensions are possible.

assignment	int iLedPin = 13;	Replace a number with a meaningful label to make it easier to read.
delay(<milliseconds>);	delay(1000);	Wait for a period of time.
digitalWrite(<pinnumber>, [LOW | HIGH]);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
for ([<Start value>]; [<end test]; [<change op>]) *<statement>;*	for (i = 0; i < 50; i++) { … }	For loop. Starts with i equal to 0, stops when i is equal to 50 and i increases by 1 each loop.

if (<condition>) <statement>;
else <statement>;
	if (lColour >= lLimit) lColour = 0;
	else lColour = 1;
		Test something and act on the result.

int analogRead(<pin number>);	iVal = analogRead(0);	*digitalWrite(<pinnumber>, LOW | HIGH);*
long random(<uppervalue + 1>);	lTarget = random(lLimit);	Provides a pseudo random number between 0 and <uppervalue>
pinMode(<pinnumber>, [INPUT | OUTPUT]);	pinMode(13, OUTPUT);	Set a pin to either send or receive.
randomSeed(<seed value>);	randomSeed(analogRead(0));	Sets the random number generator to be really random by supplying a seed
defines or pseudo constants	#define ANUMBER 10	A way to create a named constant in C. Note that there is no semicolon at the end of the line.

struct <name> '{ ' {<variable>}+ '}';	struct DIGIT {
	  int *paThisNumber;
	  int iDepth;
	  int iBuilding;
	  int iCurrent;
	};
		A structure that gathers variables that relate to a common thing for better handling and readability.

pointers	pThisDigit->paThisNumber = &iaNumbers[iOffset][0];	Pointers are used to point to other things as a way of manipulating those things.
millis()	lWaited = (long)(millis());	Get the elapsed time in milliseconds since starting.

Appendix: Function List

Function	Example	Purpose
[<string> | <null>]fgets(<buffer>, <length>, <fileptr>);	If (fgets(buf, 64, ptr) == NULL) Serial.print("finished);	fgets attempts to read a string up to one character less than the stated length. When it finishes it returns NULL.

<fileptr>popen(<cmd>, <fileflags>);
pclose(<fileptr>);
	ptr = popen("/bin/date +%k%M");
	pclose(ptr);
		popen opens a pipe to a linux command that is run and the output returned via the ptr.

<type><varname>[]
<type><varname>[<number>]
<type><varname>[][<number>]
	Char caString[] = {"a", "b", "c", "d"};
	int iaNumbers[10];
	int iaSegments[][2] ={{1, 2}, {3, 4}};
		Declaring arrays of variables. Think of a string of boxes or a chess board. When no number is specified as the dimension, they have to be initialised as shown. Two lots of square brackets mean two dimensional arrays. More dimensions are possible.

<variable> +=<value>	lColour += bStep;	Add value to the contents of variable and assign the result to variable
analogWrite(<pin number>, <speed>);	analogWrite(iBluePin, bBlue);	Start PWM on a digital pin at a specified speed.
assignment	int iLedPin = 13;	Replace a number with a meaningful label to make it easier to read.
defines or pseudo constants	#define ANUMBER 10	A way to create a named constant in C. Note that there is no semicolon at the end of the line.
delay(<milliseconds>);	delay(1000);	Wait for a period of time.
delayMicroseconds(<microseconds>)	delayMicroSeconds(10);	Delay for a number of microseconds.
digitalWrite(<pinnumber>, [LOW | HIGH]);	digitalWrite(13, LOW);	Set a pin to either low voltage or high voltage.
for ([<Start value>]; [<end test]; [<change op>]) *<statement>;*	for (i = 0; i < 50; i++) { … }	For loop. Starts with i equal to 0, stops when i is equal to 50 and i increases by 1 each loop.

if (<condition>) <statement>;
else <statement>;
	if (lColour >= lLimit) lColour = 0;
	else lColour = 1;
		Test something and act on the result.

int analogRead(<pin number>);	iVal = analogRead(0);	*digitalWrite(<pinnumber>, LOW | HIGH);*
long random(<uppervalue + 1>);	lTarget = random(lLimit);	Provides a pseudo random number between 0 and

		<uppervalue>
millis()	lWaited = (long)(millis());	Get the elapsed time in milliseconds since starting.
pinMode(<pinnumber>, [INPUT | OUTPUT]);	pinMode(13, OUTPUT);	Set a pin to either send or receive.
Pointers	pThisDigit->paThisNumber = &iaNumbers[iOffset][0];	Pointers are used to point to other things as a way of manipulating those things.
randomSeed(<seed value>);	randomSeed(analogRead(0));	Sets the random number generator to be really random by supplying a seed

Serial.begin(<bitrate>);
Serial.print(<string>|<number>);
Serial.println(<string>|<number>);
	Serial.begin(9600);
	Serial.print(buf);
	Serial.println(buf);
		Serial.begin starts serial communications from the Galileo to the computer.
		Serial.print writes the string back to the computer to the serial monitor.
		Serial.println is the same as print but includes a carriage return.

struct <name> '{ ' {<variable>}+ '}';	struct DIGIT {
	  int *paThisNumber;
	  int iDepth;
	  int iBuilding;
	  int iCurrent;
	};
		A structure that gathers variables that relate to a common thing for better handling and readability.

Appendix: Sketches

Activity 1 Sketch: Blink

Blink

```
/*
** Title: Blink
**
** Author: Paul Chubb
**
** Purpose: Make an LED on pin 13 to blink on and off
**
*/

int iLedPin = 13;
int iWaitTime = 1000; // milliseconds

void setup() {

  pinMode(iLedPin, OUTPUT);

}

void loop() {

  digitalWrite(iLedPin, HIGH);
  delay(iWaitTime);
  digitalWrite(iLedPin, LOW);
  delay(iWaitTime);
}
```

Activity 2 Sketch: RGB Blink

RGB Blink

```
/*
** Title: RGB Blink
**
** Author: Paul Chubb
**
** Purpose: Make an RGB LED blink through its colours
*/
int iRedPin = 3;
int iGreenPin = 5;
int iBluePin = 6;

int iOn = LOW;                  // High for c-cathode, Low for c-anode
int iOff = HIGH;                // Low for c-cathode, High for c-anode
int iDirection = OUTPUT;        // OUTPUT always
int iWaitTime = 1000;           // milliseconds

void setup() {

  // setu up the pins for action
  pinMode(iRedPin, iDirection);
  pinMode(iBluePin, iDirection);
  pinMode(iGreenPin, iDirection);

  // arduino starts with pin high so may need to switch off the light
  digitalWrite(iRedPin,iOff);
  digitalWrite(iGreenPin, iOff);
  digitalWrite(iBluePin, iOff);

}

void loop() {

  // Red light
  digitalWrite(iRedPin, iOn);
  delay(iWaitTime);
  digitalWrite(iRedPin,iOff);
  delay(iWaitTime);

  // Green light
  digitalWrite(iGreenPin, iOn);
  delay(iWaitTime);
  digitalWrite(iGreenPin, iOff);
  delay(iWaitTime);

  // Blue light
  digitalWrite(iBluePin, iOn);
  delay(iWaitTime);
  digitalWrite(iBluePin, iOff);
  delay(iWaitTime);

}
```

Activity 3 Sketch: Counting with Colours

Counting with Colours

```
/*
** Title: Counting with Colours
**
** Author: Paul Chubb
**
** Purpose: use PWM and an addition algorithm that will control the colours of an RGB LED
*/

// Hardware Variables
int iRedPin = 3;
int iGreenPin = 6;
int iBluePin = 5;

int iOn = LOW;                    // High for c-cathode, Low for c-anode
int iOff = HIGH;                  // Low for c-cathode, High for c-anode
int iDirection = OUTPUT;          // OUTPUT always

// Control Variables
long lColour = 0;
byte bStep = 1;                   // probably work best with 1 or a multiple of 2
int iAnode = -255;                // 0 for common cathode, -255 for common anode
int iWaitTime = 10;               // milliseconds
```

```
long lLimit = 16777216;                    // this is the maximum number you need to count to

/*
** GetFirstByte()
**
** Returns the first byte in a four byte number
*/
byte GetFirstByte(long lNumber) {
long lFirstByteMask = 255;

   return lFirstByteMask & lNumber;
}

/*
** GetSecondByte()
**
** Returns the second byte in a four byte number
*/
byte GetSecondByte(long lNumber) {
long lSecondByteMask = 65280;

   return (lSecondByteMask & lNumber) >> 8;
}

/*
** GetThirdByte()
**
** Returns the third byte in a four byte number
*/
byte GetThirdByte(long lNumber) {
long lThirdByteMask = 16711680;

   return (lThirdByteMask & lNumber) >> 16;
}

void setup() {

  // setup the pins for action
  pinMode(iRedPin, iDirection);
  pinMode(iBluePin, iDirection);
  pinMode(iGreenPin, iDirection);

  // arduino starts with pin high so may need to switch off the light
  digitalWrite(iRedPin,iOff);
  digitalWrite(iGreenPin, iOff);
   digitalWrite(iBluePin, iOff);

}

void loop() {

  lColour += bStep;
  if (lColour >= lLimit) lColour = 0;
  analogWrite(iBluePin, abs(iAnode + GetFirstByte(lColour)));
   analogWrite(iGreenPin, abs(iAnode + GetSecondByte(lColour)));
  analogWrite(iRedPin, abs(iAnode + GetThirdByte(lColour)));
  delay(iWaitTime);
}
```

Activity 4 Sketch: Mood Light

RGB Mood Light

```
*
** Title: Mood light
**
** Author: Paul Chubb
**
** Purpose: use PWM, random  and fade algorithm to control the colours of an RGB LED
*/

// Hardware Variables
int iRedPin = 3;
int iGreenPin = 6;
int iBluePin = 5;

int iOn = LOW;                          // High for c-cathode, Low for c-anode
int iOff = HIGH;                        // Low for c-cathode, High for c-anode
int iDirection = OUTPUT;                // OUTPUT always

// Control Variables
long lTarget = 0;
byte bBlueStep = 1;                     // blue direction step +ve count up, -ve count down
byte bGreenStep = 1;                    // green direction step
byte bRedStep = 1;                      // red direction step
int iAnode = -255;                      // 0 for common cathode, -255 for common anode
int iWaitTime = 3000;                   // milliseconds
int iSpeed = 10;                        // control fade speed - higher = slower
long lLimit = 16777216;                 // this is the maximum number you need to count to
byte bBlue = 0;                         // current blue value
byte bGreen = 0;                        // current green value
byte bRed = 0;                          // current red value
byte bBlueTarget = 0;                   // blue target value
byte bGreenTarget = 0;                  // green target value
byte bRedTarget = 0;                    // red target value

/*
** GetFirstByte()
**
** Returns the first byte in a four byte number
*/
byte GetFirstByte(long lNumber) {
long lFirstByteMask = 255;

   return lFirstByteMask & lNumber;
}

/*
** GetSecondByte()
**
** Returns the second byte in a four byte number
*/
byte GetSecondByte(long lNumber) {
long lSecondByteMask = 65280;

   return (lSecondByteMask & lNumber) >> 8;
}
```

```
/*
** GetThirdByte()
**
** Returns the third byte in a four byte number
*/
byte GetThirdByte(long lNumber) {
long lThirdByteMask = 16711680;

   return (lThirdByteMask & lNumber) >> 16;
}

/*
** SetTarget()
**
** Get a random number and setup target control values
**
*/
void SetTarget() {

  // get our target
  lTarget = random(lLimit);

  // set the individual colour targets
  bBlueTarget = abs(iAnode + GetFirstByte(lTarget));
  bGreenTarget = abs(iAnode + GetSecondByte(lTarget));
  bRedTarget = abs(iAnode + GetFirstByte(lTarget));

  // set the directions
  if (bBlue > bBlueTarget) bBlueStep = -1;
  else bBlueStep = 1;
  if (bGreen > bGreenTarget) bGreenStep = -1;
  else bGreenStep = 1;
  if (bRed > bRedTarget) bRedStep = -1;
  else bRedStep = 1;
}

void setup() {

  // setup the pins for action
  pinMode(iRedPin, iDirection);
  pinMode(iBluePin, iDirection);
  pinMode(iGreenPin, iDirection);

  // arduino starts with pin high so may need to switch off the light
  digitalWrite(iRedPin,iOff);
  digitalWrite(iGreenPin, iOff);
  digitalWrite(iBluePin, iOff);

  // set the seed so numbers will be random using noise on an unconnected pin
  // set the initial random colour target
  randomSeed(analogRead(0));
  SetTarget();
}

void loop() {

  if ((bBlue == bBlueTarget) && (bGreen == bGreenTarget) && (bRed == bRedTarget)) {
    // got to the end colour so show it, wait and choose another
    analogWrite(iBluePin, bBlue);
    analogWrite(iGreenPin, bGreen);
    analogWrite(iRedPin, bRed);
    delay(iWaitTime);
    SetTarget();
  } else {
    // update each colour that isn't there yet
    if (bBlue != bBlueTarget) bBlue += bBlueStep;
    if (bGreen != bGreenTarget) bGreen += bGreenStep;
    if (bRed != bRedTarget) bRed += bRedStep;
    analogWrite(iBluePin, bBlue);
    analogWrite(iGreenPin, bGreen);
    analogWrite(iRedPin, bRed);
    delay(iSpeed);
  }
}
```

Activity 5 Sketch: RGB Toggle

RGB Toggle

```
/*
** Title: RGB Toggle
**
** Author: Paul Chubb
**
** Purpose: use a push switch to cycle through the colours of an RGB LED
*/
int iSwitch = 13;
int iRedPin = 3;
int iGreenPin = 6;
int iBluePin = 5;
int iCurrentColour = iRedPin;
int iButtonState = HIGH;
int iOn = LOW;                     // High for c-cathode, Low for c-anode
int iOff = HIGH;                   // Low for c-cathode, High for c-anode
int iDebounce = 50;                // milliseconds to wait for switch to settle

void setup() {

  // setup the pins for either input or output
  pinMode(iSwitch,INPUT);
  pinMode(iRedPin, OUTPUT);
  pinMode(iBluePin, OUTPUT);
  pinMode(iGreenPin, OUTPUT);

  // start with red
  digitalWrite(iRedPin, iOn);
  digitalWrite(iGreenPin, iOff);
  digitalWrite(iBluePin, iOff);
}

void loop() {

// read and debounce the switch
int iPinVal = digitalRead(iSwitch);
delay(iDebounce);
int iPinVal2 = digitalRead(iSwitch);
```

```
  // are we in a stable state?
  if (iPinVal == iPinVal2) {
    if (iPinVal != iButtonState) {
      if (iPinVal == LOW) {
        // change colour
        digitalWrite(iCurrentColour, iOff);
        if (iCurrentColour == iRedPin) iCurrentColour = iGreenPin;
        else if (iCurrentColour == iGreenPin) iCurrentColour = iBluePin;
        else iCurrentColour = iRedPin;
        digitalWrite(iCurrentColour, iOn);
      }
      iButtonState = iPinVal;
    }
  }
}
```

Activity 6 Sketch: Infrared Alarm

Alarm

```
/*
** Title: Movement Alarm
**
** Author: Paul Chubb
**
** Purpose: Using a passive Infrared motion detector, sound when something moves
*/

int iBeepPin = 9;
int iSensorPin = 2;

void setup() {

  pinMode(iBeepPin, OUTPUT);
  pinMode(iSensorPin, INPUT);
}

void loop() {
int iMovement;

   // check the motion sensor
   iMovement = digitalRead(iSensorPin);
   if (iMovement == HIGH) {
     analogWrite(iBeepPin, 20);      // Almost any value can be used except 0 and 255
     delay(1000);         // wait for a delayms ms
     analogWrite(iBeepPin, 0);      // 0 turns it off
   }

}
```

Activity 7 Sketch: 4 Digit 7 Segment Display

Seven Seg 4 digit tester

```
/*
** Title: Seven Seg 4 digit tester
**
** Author: Paul Chubb
```

```
**
** Purpose: Run through each segment to check that it is working
*/

// segment array - A => G, DP
int iaSegs[] = {0, 2, 4, 6, 7, 1, 3, 5};
int iaDigs[] = {8, 9, 10, 11};
int iSegBound = 8;
int iDigBound = 4;

// other vars
int iDigOpen = HIGH;
int iDigClose = LOW;
int iSegOpen = HIGH;
int iSegClose = LOW;
int iWaitTime = 500;

void setup() {
int i;

  // set all pins to output
  for (i = 0; i < iSegBound; i++)
      pinMode(iaSegs[i],OUTPUT);

  for (i = 0; i < iDigBound; i++)
      pinMode(iaDigs[i],OUTPUT);
}

void loop() {
int iDigInc, iSegInc;

  // walk through each digit
  for (iDigInc = 0; iDigInc < iDigBound; iDigInc++) {
     // switch on this digit
     digitalWrite(iaDigs[iDigInc], iDigOpen);
     // walk through each segment
     for (iSegInc = 0; iSegInc < iSegBound; iSegInc++) {
        // switch on this segment and wait
        digitalWrite(iaSegs[iSegInc], iSegOpen);
        delay(iWaitTime);
        digitalWrite(iaSegs[iSegInc], iSegClose);
     }
     digitalWrite(iaDigs[iDigInc], iDigClose);
   }
}
```

Activity 8 Sketch: Marquee

Marquee

```
/*
** Title: Marquee
**
** Author: Paul Chubb
**
** Purpose: scroll text and numbers on a 4 digit seven segment display
*/

// segment array - A => G, DP
```

```
int iaSegs[] = {0, 2, 4, 6, 7, 1, 3, 5};
int iaDigs[] = {8, 9, 10, 11 };
int iSegBound = 8;
int iDigBound = 4;

// message array
int L = LOW;
int H = HIGH;
int iDisplays = 0;
int iJustRight = 25;
int iMessageBound = 24;
int iMessageInc = 0;

// USB id for Galileo is 8086:babe
int iaMessage[][8] = {
                    {L, L, L, L, L, L, L, L},     // all off
                    {L, L, L, L, L, L, L, L},     // all off
                    {L, L, L, L, L, L, L, L},     // all off
                    {L, L, L, L, L, L, L, L},     // all off
                    {L, L, L, L, L, L, H, L},     // dash
                    {L, L, L, L, L, L, H, L},     // dash
                    {L, L, L, L, L, L, H, L},     // dash
                    {L, L, L, L, L, L, H, L},     // dash
                    {L, L, H, H, H, H, H, L},     // B
                    {H, H, H, H, H, L, H, L},     // A
                    {L, L, H, H, H, H, H, L},     // B
                    {H, L, L, H, H, H, H, L},     // E
                    {L, L, L, L, L, L, H, L},     // dash
                    {L, L, L, L, H, H, L, L},     // I
                    {H, L, H, H, L, H, H, L},     // S
                    {L, L, L, L, L, L, H, L},     // dash
                    {L, L, H, H, H, H, H, L},     // B
                    {H, H, H, H, H, L, H, L},     // A
                    {H, L, L, H, H, H, L, L},     // C
                    {L, L, H, H, H, L, H, L},     // O
                    {L, L, H, L, H, L, H, L},     // N
                    {L, L, L, L, L, L, L, H},     // .
                    {L, L, L, L, L, L, L, H},     // .
                    {L, L, L, L, L, L, L, H},     // .
};

// other vars
int iDigOpen = HIGH;
int iDigClose = LOW;
int iSegOpen = HIGH;
int iSegClose = LOW;
int iWaitTime = 1;

void setup() {
int i;

  // set all pins to output
  for (i = 0; i < iSegBound; i++)
      pinMode(iaSegs[i],OUTPUT);

  for (i = 0; i < iDigBound; i++)
      pinMode(iaDigs[i],OUTPUT);
}
```

```
void loop() {
int iDigInc, iLetter, iSegInc;
  // walk through each digit
  for (iDigInc = 0; iDigInc < iDigBound; iDigInc++) {
     // switch on this digit
     digitalWrite(iaDigs[iDigInc], iDigOpen);
     // write the letter for this digit
     iLetter = iMessageInc + iDigInc;
     if (iLetter >= iMessageBound) iLetter -= iMessageBound;
     for (iSegInc = 0; iSegInc < iSegBound; iSegInc++) {
        // switch on/off each segment
        digitalWrite(iaSegs[iSegInc],iaMessage[iLetter][iSegInc]);
        delay(iWaitTime);
        digitalWrite(iaSegs[iSegInc],iSegClose);
     }
     digitalWrite(iaDigs[iDigInc], iDigClose);
   }
   // move to the next letter and wrap
   iDisplays++;
   if (iDisplays > iJustRight) {
      iDisplays = 0;
      iMessageInc++;
      if (iMessageInc >= iMessageBound) iMessageInc = 0;
   }

}
```

Activity 9 Sketch: Digital Clock

Digital Clock

```
/*
** Title: Real Time Clock
**
** Author: Paul Chubb
**
** Purpose: Access the embedded linux OS to set then get the time and display on seven segment display
*/

// time commands
char *cpStartTime = "/bin/date 1114100014";     // use date command to set the date and time numbers:
                                                //MMDDhhmmYY
char *cpRetTime = "/bin/date +%k%M";            // use the date command to get back the current time
int iAsciiZero = 48;

// segment array - A => G, DP
int iaSegs[] = {0, 2, 4, 6, 7, 1, 3, 5};
int iaDigs[] = {8, 9, 10, 11 };
int iSegBound = 8;
int iDigBound = 4;

// message array
int L = LOW;
int H = HIGH;
int iaNumbers[][8] = {
                    {H, H, H, H, H, H, L, L},   // 0
                    {L, H, H, L, L, L, L, L},   // 1
                    {H, H, L, H, H, L, H, L},   // 2
                    {H, H, H, H, L, L, H, L},   // 3
```

```
                                    {L, H, H, L, L, H, H, L},   // 4
                                    {H, L, H, H, L, H, H, L},   // 5
                                    {H, L, H, H, H, H, H, L},  // 6
                                    {H, H, H, L, L, L, L, L},    // 7
                                    {H, H, H, H, H, H, H, L},  // 8
                                    {H, H, H, L, L, H, H, L},   // 9
                                    {L, L, L, L, L, L, L, H},      // DP
};

// other vars
int iDigOpen = HIGH;
int iDigClose = LOW;
int iSegOpen = HIGH;
int iSegClose = LOW;
int iWaitTime = 1;            // wait one millisec between turning on a seg and off. Without the time it usually doesn't
have time to light up

void setup() {
char *cmd = cpStartTime;
char buf[64];
FILE *ptr;
int i;

  // start serial comms
  Serial.begin(9600);

  // set the time
  if ((ptr = popen(cmd, "r")) != NULL)
  {
    while (fgets(buf, 64, ptr) != NULL)
    {
      Serial.print(buf);
    }
  }
  (void) pclose(ptr);

  // set all pins to output
  for (i = 0; i < iSegBound; i++)
       pinMode(iaSegs[i],OUTPUT);

  for (i = 0; i < iDigBound; i++)
       pinMode(iaDigs[i],OUTPUT);
}

void loop()
{
char *cmd = cpRetTime;
char buf[64];
FILE *ptr;
int iDigInc, iSegInc, iNum;

  // read back the time
  if ((ptr = popen(cmd, "r")) != NULL) {
    while (fgets(buf, 64, ptr) != NULL) {
      Serial.print(buf);
    }
  }
  (void) pclose(ptr);
  // display on leds
  for (iDigInc = 0; iDigInc < iDigBound; iDigInc++) {
      // switch on this digit
      digitalWrite(iaDigs[iDigInc], iDigOpen);
      // write the letter for this digit
      // turn ascii character string into numbers by deleting an ascii zero
      iNum = buf[iDigInc] - iAsciiZero;
      for (iSegInc = 0; iSegInc < iSegBound; iSegInc++) {
         // switch on/off each segment
         digitalWrite(iaSegs[iSegInc],iaNumbers[iNum][iSegInc]);
         delay(iWaitTime);
         digitalWrite(iaSegs[iSegInc],iSegClose);

      }
      if (iDigInc == 1) {
        digitalWrite(iaSegs[iSegBound - 1],iSegOpen);
        delay(iWaitTime);
        digitalWrite(iaSegs[iSegBound - 1],iSegClose);
      }
      digitalWrite(iaDigs[iDigInc], iDigClose);
   }
}
```

Activity 10 Sketch: Poker Machine

Poker Machine

```
/*
** Title: Poker Machine
**
** Author: Paul Chubb
**
** Purpose: using a button, get the Galileo to emulate a poker machine (aka one armed bandit, slot machine)
*/

// pins etc
// segment array - A => G, DP
int iaSegs[] = {0, 2, 4, 6, 7, 1, 3, 5};
int iaDigs[] = {8, 9, 10, 11 };
int iSegBound = 8;
int iDigBound = 4;
int iSwitch = 12;

// message array
int L = LOW;
int H = HIGH;
int iaNumbers[][8] = {
                            {H, H, H, H, H, H, L, L},  // 0
                            {L, H, H, L, L, L, L, L},  // 1
                            {H, H, L, H, H, L, H, L},  // 2
                            {H, H, H, H, L, L, H, L},  // 3
                            {L, H, H, L, L, H, H, L},  // 4
                            {H, L, H, H, L, H, H, L},  // 5
                            {H, L, H, H, H, H, H, L},  // 6
                            {H, H, H, L, L, L, L, L},  // 7
                            {H, H, H, H, H, H, H, L},  // 8
                            {H, H, H, L, L, H, H, L},  // 9
                            {L, L, L, L, L, L, L, H},  // DP
};
```

```
// other vars
int iDigOpen = HIGH;
int iDigClose = LOW;
int iSegOpen = HIGH;
int iSegClose = LOW;
int iWaitTime = 5;          // wait one millisec between turning on a seg and off. Without the time it usually doesn't
                            //have time to light up
long lDebounce = 50;        // time to wait for switch to debounce
int iLastSwVal = HIGH;  // default value is high
long lMark = 0;             // last time we got a switch change

// state control defines
#define WAITING 0           // spinning my lights before displaying the number
#define BUILDING 1          // building from spin to a number
#define BUILT 2             // displaying a number
#define PRESTART 3          // just spinning before we start
#define MAXSPINS 3          // max number of spins before number
#define STEPS 5             // control how fast the lights swivel

// digit state structs
struct DIGIT {
  int *paThisNumber;        // template for this number
  int iDepth;               // where we are up to drawing it
  int iBuilding;            // state: prestart to waiting to building to built to waiting
  int iCurrent;             // current segment to draw when spinning
  int iTimesSpun;           // times we have spun up to MAXSPINS
  int iTimesDisp;           // times we have displayed the current image up to steps times
  int iDigit;               // Which display digit we are
};

struct DIGIT saDisplay[4];

/*
** start()
**
** initialise the passed digit struct
**
*/
void start(struct DIGIT *pThisDigit, int iOffset, int iDig) {
  int iSegInc;

  pThisDigit->paThisNumber = &iaNumbers[iOffset][0];
  pThisDigit->iDepth = 0;
  pThisDigit->iBuilding = WAITING;
  pThisDigit->iCurrent = 0;
  pThisDigit->iTimesSpun = 0;
  pThisDigit->iTimesDisp = 0;
  pThisDigit->iDigit = iDig;
}


/*
** setup()
**
** initialise the pins, digit structs and the random number generator
**
*/
void setup() {
```

```
int i;

  // set sw pin to input
  pinMode(iSwitch,INPUT);

  // set all pins to output
  for (i = 0; i < iSegBound; i++)
      pinMode(iaSegs[i],OUTPUT);

  for (i = 0; i < iDigBound; i++)
      pinMode(iaDigs[i],OUTPUT);

  // start the digits;
  for (i = 0; i < iDigBound; i++) {
      start(saDisplay+i,i, i);
      saDisplay[i].iBuilding = PRESTART;
  }

  // set our random generator with a truly random seed ie the noise on a pin
  randomSeed(analogRead(0));
}

/*
** scramble()
**
** do the various displays, move through the states etc for the passed digit
**
*/
void scramble(struct DIGIT *pThisDigit) {
int i;

    // wrap the segment value and open this digit for writing and count the spins
    if (pThisDigit->iCurrent > iSegBound) {
       pThisDigit->iCurrent = 0;
       pThisDigit->iTimesSpun++;
    }
    digitalWrite(iaDigs[pThisDigit->iDigit], iDigOpen);


   // if building or built write all segments before the current one
   // assert: if built then iDepth will be at the bound
   if ((pThisDigit->iBuilding == BUILT) || (pThisDigit->iBuilding == BUILDING)) {
      for (i = 0; i < pThisDigit->iDepth; i++) {
         digitalWrite(iaSegs[i],pThisDigit->paThisNumber[i]);
      }
      delay(iWaitTime);
   }

   // check moving to building
   if ((pThisDigit->iBuilding == WAITING) && (pThisDigit->iTimesSpun >=MAXSPINS)) {
       pThisDigit->iBuilding = BUILDING;
   }

   // write the current one if we aren't built
   if (pThisDigit->iBuilding != BUILT) {
       digitalWrite(iaSegs[pThisDigit->iCurrent],iSegOpen);
       pThisDigit->iTimesDisp++;
       delay(iWaitTime);
       if (pThisDigit->iTimesDisp > STEPS) {
         // move everything along a step
         pThisDigit->iCurrent++;
         if (pThisDigit->iBuilding == BUILDING) pThisDigit->iDepth++;
         if (pThisDigit->iDepth >= iSegBound) {
           pThisDigit->iDepth = iSegBound;
           pThisDigit->iBuilding = BUILT;
         }
         pThisDigit->iTimesDisp = 0;
       }
   }


   // shutdown and prepare for the next time
   for (i = 0; i < iSegBound; i++)
      digitalWrite(iaSegs[i],iSegClose);

   // close this digit
   digitalWrite(iaDigs[pThisDigit->iDigit], iDigClose);
}

/*
** loop()
**
** debounce our switch and start the run if it is switched, run scramble
**
*/
void loop() {
int i;
int iSwVal;
long lWaited;

  // only allow switch if we have finished the spin
  if ((saDisplay[0].iBuilding == BUILT) || (saDisplay[0].iBuilding == PRESTART)) {

    // check my switch
    iSwVal = digitalRead(iSwitch);

    // check to see if the button has changed if it has restart the clock
    if (iSwVal != iLastSwVal) {
      lMark = (long)millis();
      iLastSwVal = iSwVal;
    }

    // only act if we are a real debounced value ie hasn't changed for a while
    lWaited = (long)(millis()) - lMark - lDebounce;
    if (lWaited >= 0) {
      // only act if the switch is on
      if (iSwVal == LOW) {
         // get a random value (0-9) into each digit and ready to start
          for (i = 0; i < iDigBound; i++)
           start(saDisplay+i,random(9), i);
      }
    }
  }

  // spin the dials
  for (i = 0; i < iDigBound; i++) {
      scramble(saDisplay+i);
  }

}
```

Appendix: Curriculum Map

At time of writing the Australian National Curriculum for Digital Technologies was still in draft. The following maps use the outcome designators from that draft curriculum. In almost all cases, the outcome is only partially covered by any one activity. In many cases more of the outcome can be achieved by requiring standardised development processes around the activities and additional teaching. Only the outcomes that are covered by the activity are listed.

Year 7 and 8

Curriculum Outcome	Relevant Activities
Investigate how data are transmitted and secured in wired, wireless and mobile networks, and how the specifications of hardware components impact on network activities (ACTDIK023)	9
Investigate how digital systems represent text, image and audio data in binary (ACTDIK024)	3, 4, 6, 7, 8, 9, 10
Design algorithms represented diagrammatically and in English, and trace algorithms to predict output for a given input and to identify errors (ACTDIP029)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Implement and modify programs with user interfaces involving branching, iteration and functions in a general-purpose programming language (ACTDIP030)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Plan and manage projects, including tasks, time and other resources required, considering safety and sustainability (ACTDIP033)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Acquire data from a range of sources and evaluate authenticity, accuracy and timeliness (ACTDIP025)	5, 6, 10
Define and decompose real-world problems taking into account functional requirements and economic, environmental, social, technical and usability constraints (ACTDIP027)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Year 9 and 10

Curriculum Outcome	Relevant Activities
Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases (ACTDIP040)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Implement modular programs, applying selected algorithms and data structures including using an object-oriented programming language (ACTDIP041)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Plan and manage projects using an iterative and collaborative approach, identifying risks and considering safety and sustainability (ACTDIP044)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Precisely define and decompose real-world problems, taking into account functional and non-functional requirements and including interviewing stakeholders to identify needs (ACTDIP038)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10