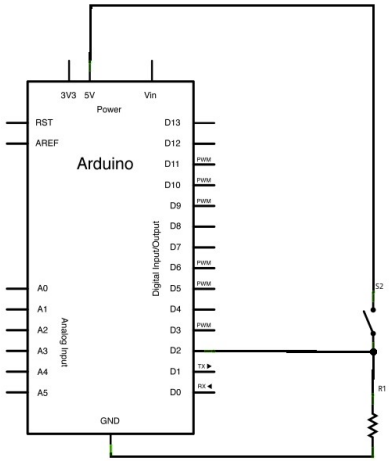
Week 3 – Let's recap weeks 1 & 2

- By now you should be starting to have a pretty good idea of the Arduino's capabilities
  - digitalWrite sets a pin to 0 or 5v to control LEDs or Relays
  - analogWrite doesn't produce a real analog voltage but uses PWM to simulate one – fine for dimming LEDs or controlling motors
  - AnalogRead reads an analog voltage and produces a digital value in the range 0-1023
  - Libraries are very well supported
    - Lots of built-ones
    - Plenty of libraries on the Arduino playground or on the web
    - Install is straightforward
      - normally can all be done from Sketch/Inlcude Library in the IDE
    - Most libraries come with really good Examples
  - The Arduino documentation is excellent and easy to use
  - Many projects just require finding the appropriate library, copying the example and writing a little 'glue' code

But what about digital input?

- Just as digital output can put 0v or 5v on a pin, digital input can read the value on a pin
  - 5v will return a value of 1 (or HIGH) and 0v a 0 (or LOW)
- To set a pin as digital input we can use
  - pinMode(INPUT) – this is default so not strictly needed
    - But good coding practice to make it clear to anyone reading
  - pinMode(INPUT_PULLUP)
    - Sets a weak pullup to 5v
- We then just connect a button to ground (or 5v)

For pinMode(INPUT) we need a pull-up
or pull-down resistor to hold the pin at 5v or 0v



For pinMode(INPUT_PULLUP) an 20kΩ resistor internal in the Arduino holds the pin at 5v, so no external resistor is needed

pinMode(INPUT) with no external pull-up or pull-down resistor means the pin will be floating and is randomly HIGH or LOW when read

Let's try it: the DigitalReadSerial example #1

```
int pushButton = 4;

void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
  delay(1);        // delay in between reads for stability
}
```

Note the line in RED
Until now we have declared all our variables in the header section (as we do with pushButton)
We can declare variables when we want to use them but note:

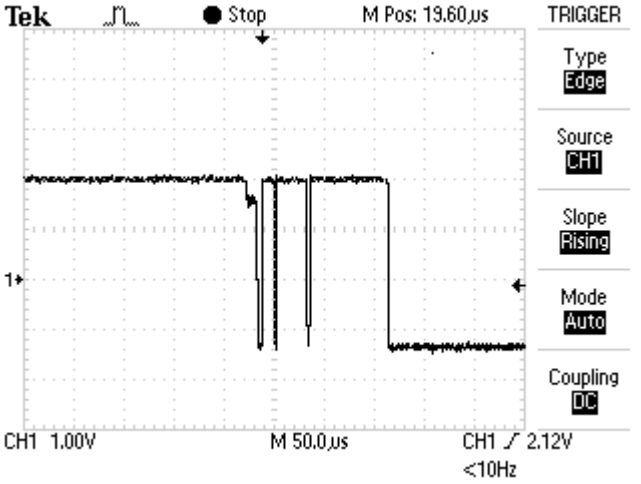
- buttonState must be declared before we use it for the first time
- We still need to say what type of variable it is – e.g. an int
- It is only accessible inside the loop function whilst pushButton is accessible everywhere
  - buttonState is a LOCAL variable, pushButton is a GLOBAL variable

Let's try it: the DigitalReadSerial example #2

- So – hook up a simple switch between pin 4 and ground and try DigitalReadSerial
  - From the Examples/1. Basic section
- Start up the Serial Monitor and look at the output
  - Random 0s and 1s
  - Unaffected by the button press
  - This is because we are using pinMode(INPUT) without and external pull up/down resistor
  - We could add a resistor to hold pin 2 high or:
- Change the pinMode to pinMode(INPUT_PULLUP)
  - Now notice the difference when the button is pressed
- Now try something different, make the LED on pin 13 light when the button is pressed
  - Don't cheat by looking at Examples/2. Digital/Button

There's always some problem

- Buttons don't make nice clean contacts but 'bounce'
- Arduino's can read the state of the button very quickly
  - And so catch these bounces
  - Making it look like multiple button presses



- So we often need to de-bounce button reads
  - We read the button again after a short interval
  - If it is still in the same state then it is a real press/release
  - Otherwise it might just be noise from the button bouncing
- The same technique can be used to determine long and short button presses
  - Elecraft makes great use of these
  - Buttons provide different functions depending on length of press

millis()

- We shouldn't use delay() in loops with switches
  - A switch press won't be detected during a delay
  - So we have to do it manually
    - Using millis() which returns the milliseconds clock value since start-up
    - This is an unsigned long so is 32 bits long
      - Which means it will roll-over in just under 50 days
    - The code that follows will assume the Arduino will not run continuously for > 50 days
    - If it does we need to add some additional code
  - Since millis() returns a long we must always use longs when doing calculations with it

Debouncing a switch – sample code

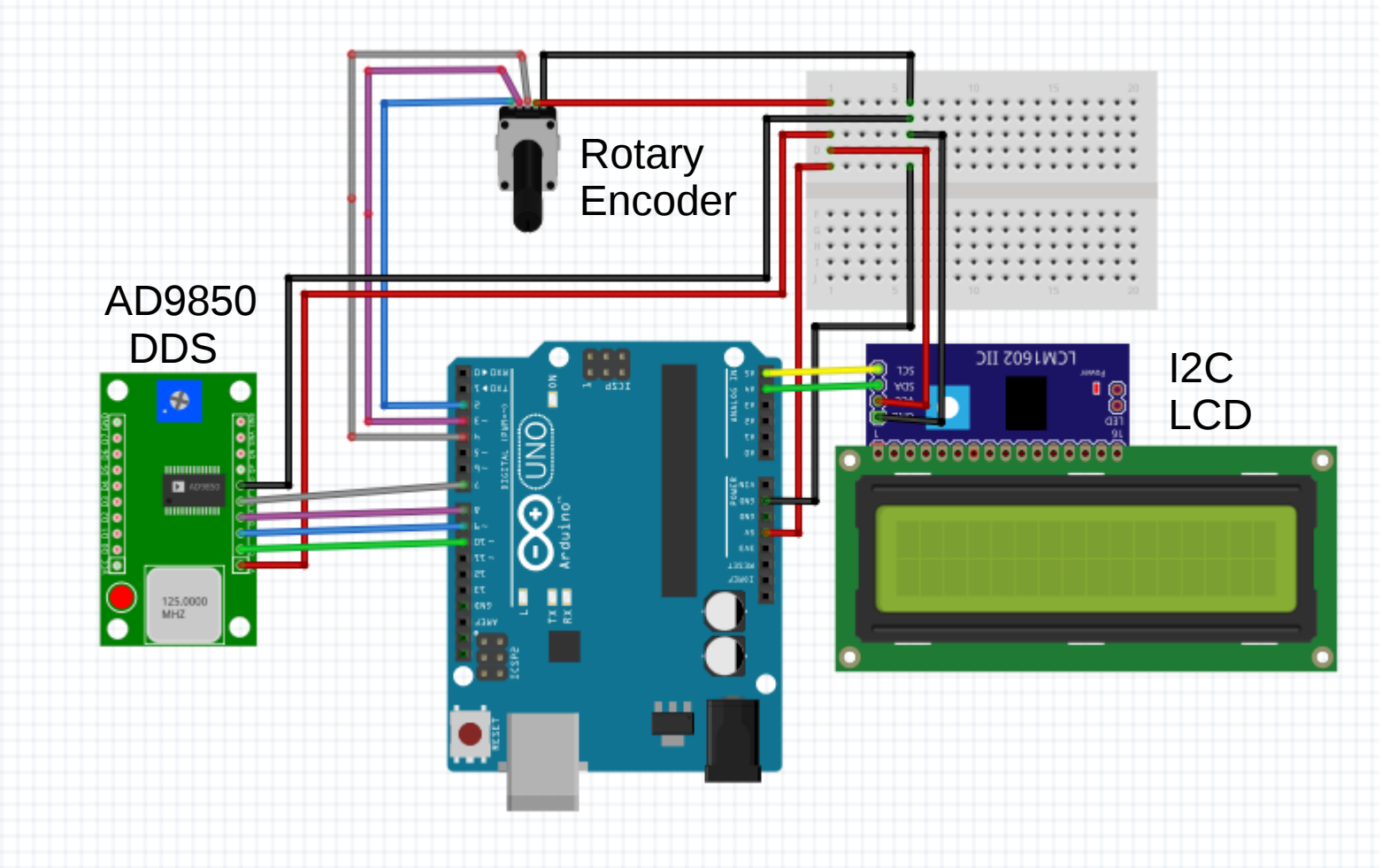
```
const int btnPin=4;                                  // Define the button pin

int buttonState;                                     // The button's true debounced state
int prevTmpState = HIGH;                             // temp button state while debouncing
long lastChange;                                     // last time button changed
const long debounceDelay =50;                        // debounce time

void setup() {
  Serial.begin (9600);
  pinMode(btnPin,INPUT_PULLUP);                      // make sure the button works
}

void loop() {
  int tmpState=digitalRead(btnPin);                  // read current state of the button
  if (tmpState != prevTmpState) {                    // has it changed?
    lastChange = millis();                           // yes – note the time
    prevTmpState=tmpState;                           // and record it's new state
  } else {                                           // no – still the same
    if (millis()-lastChange>debounceDelay){          // has it stayed the same for a while
      buttonState=tmpState;                          // yes – it's not just a bounce
      prevTmpState=buttonState;                       // so record it as true button press
    }
  }
  Serial.println(buttonState);
}
```

That's all folks!

- This course is intended to be an appreciation of Arduinos not a programming tutorial
- There is a LOT more programming syntax we haven't covered
- Important pieces of syntax to learn next
  - for loops allow you to repeat things multiple times
  - Functions – you've used functions such as lcd.print('text')
    - But creating your own is great when you want to do something in multiple places in your code
    - And great for breaking up your code into manageable 'baby' steps
    - Sean's homework answer is a perfect example!
- If you are interested in learning more code
  - Buy an Arduino book – I like Arduino Cookbook (O'Reilly) about £20
    - Although a quick Google for it did find a (illegal) PDF copy on the web if you want to try before you buy
  - Ask for help – Me, Sean and several others at the club have programming experience
  - Use the excellent Arduino website and its documentation, examples etc.
- Another interesting follow-up – learn Python
  - Python is (IMHO) a much better first programming language to learn
  - It is the native language for the Raspberry Pi – do we want a Raspberry Pi appreciation course?
  - It is all I program in these days

A no(t much) programming DDS project

0-30MHz without soldering



Done in Baby Steps

A plagiarist's toolbox

- Learning to copy and paste text from an example into your code is invaluable
- Keyboard is quicker and easier than using cut and paste from edit menu
- To cut or copy text
  - Select the text with the mouse
  - Ctrl-c will copy the text
  - Ctrl-x will cut it – handy if you are moving text around
- To paste text
  - Move the cursor to where you want to paste
    - You may want to hit enter to create a blank line
  - Ctrl-v to paste
- To overwrite text
  - Highlight the text you want overwritten
  - Ctrl-v to paste and overwrite it
- To recover from accidental overwrites, cuts or pastes in the wrong place
  - Ctrl-z will restore the situation be undoing the last action
- Save the good file regularly
  - Ctrl-s will save

Step 1 – get the rotary encoder working

1) Find a library supporting KY-040 Rotary Encoders
   a) Look on Arduino Playground
   b) Find a really good library at http://www.pjrc.com/teensy/td_libs_Encoder.html
   c) Install it using sketch/Include Library/Add .ZIP library from the IDE menu
2) Find the Examples/Basic program
   a) Change pins to map to the project's encoders pin
   b) Test it!

#1 - Rotary Encoder test code

Comments edited for the presentation

```
#include <Encoder.h>   // include the header file

Encoder myEnc(2, 3);   // define the pins used

void setup() {
  Serial.begin(9600);  // set up serial monitor
  Serial.println("Basic Encoder Test:");
}

long oldPosition  = -999;

void loop() {
  long newPosition = myEnc.read(); //get encoder value
  if (newPosition != oldPosition) {
    oldPosition = newPosition;     // if it has changed
    Serial.println(newPosition);   // print it to the monitor
  }
}
```

Step 2 – change frequency

- OK so the rotary encoder works and increments when turned clockwise and decrements counter-clockwise
- Lets change that into a varying frequency

  1) Pick an initial frequency (say 10MHz)

  2)Pick a step size (say 100KHz)

  3)Do the maths

    - Freq = initialfreq + encoder_value * stepsize
- Also let's improve the code
  - Using const int to define the encoder pins

#2 – Change frequency via the encoder

```
#include <Encoder.h>   // include the header file

const int ENC_CLK_PIN = 2;                  // encoder clock pin
const int ENC_DT_PIN  = 3;                  // encoder DT pin
const unsigned long initFreq = 10000000;    // initial frequency to display
const unsigned long stepSize = 100000;      // step to change freq

unsigned long freq = initFreq;              // actual current frequency

Encoder myEnc(ENC_CLK_PIN, ENC_DT_PIN);     // define the pins used

void setup() {
  Serial.begin(9600);                       // set up serial monitor
  Serial.println("Basic Encoder Test:");
}

long oldPosition  = -999;

void loop() {
  long newPosition = myEnc.read();        //get encoder value
  if (newPosition != oldPosition) {
    oldPosition = newPosition;            // if it has changed
    freq=initFreq+newPosition*stepSize;   // calculate the frequency
    Serial.println(freq);           // and print it to the monitor
  }
}
```

Step 2 - Add the LCD

- We already have the library installed for an I2C LCD
  - If we hadn't we could have found one via the playground
  - And installed it using the normal Add .ZIP Library
- We already have some working sample code for the LCD
  - If we did our homework!
  - Or we could refer back to last week's slides
- All we need do is copy in the relevant parts
  - And change Serial.println to lcd.print
    - **Note** lcd.print NOT lcd.println
  - Oh but lcd.print just prints over what was previously on the display
    - So if we go from 10000000 down to 9900000 then the final 0 of 10000000 will still show
    - So we print a single space after the frequency to avoid this problem
- We could leave the Serial monitor code in too help debug later
  - But I'll remove it to make the code easier to show in the presentation
- I'll also change the encoder reads to use a better variable name and show the frequency every time round the loop
  - again to reduce code for ease of presentation

#3 - Show the frequency on the LCD

```
#include <Encoder.h>                          // include the encoder header file
#include <Wire.h>                             // and the I2C header file
#include <LiquidCrystal_I2C.h>                // and the LCD header file

const int ENC_CLK_PIN = 2;                    // encoder clock pin
const int ENC_DT_PIN  = 3;                    // encoder DT pin
const unsigned long initFreq = 10000000;      // initial frequency to display
const unsigned long stepSize = 100000;        // step to change freq

Encoder myEnc(ENC_CLK_PIN, ENC_DT_PIN);       // define the pins used
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

unsigned long freq = initFreq;                // actual current frequency
long encoderCt=0;                             // value read from the encoder

void setup() {
  lcd.begin(16,2);                      // send initialisation sequence to the LCD
}

void loop() {
    encoderCt = myEnc.read();                 //get encoder value
    freq=initFreq+encoderCt*stepSize;         // calculate the frequency
    lcd.setCursor(0,0);                       // set the LCD cursor to start of line 1
    lcd.print(freq);                          // print freq to monitor
    lcd.print(" ");                           // overwrite any trailing char with space
}
```

Step 4 – Add the DDS

- You know the drill by now

  1) Find an appropriate library via the playground

  2)In this case nothing found so search Google

  - Found https://github.com/F4GOJ/AD9850/blob/master
  - Actually none of the libraries are perfect

  3) Install using normal Add .ZIP Library

  4)Find the example (AD9850.ino)

  5)Comments on the example

  - Note the setfreq function takes 2 parameters - freq and phase
  - while(1) at the end creates a tight infinite loop
    - So effectively stops the loop and we only run the code in the loop once
  - double variables are exactly the same as float – can use either

#4 - Run the DDS example

Simplifed slightly for the ease of presentation

```
#include <AD9850.h>

const int W_CLK_PIN = 7;
const int FQ_UD_PIN = 8;
const int DATA_PIN = 9;
const int RESET_PIN = 10;

double freq = 10000000;
double trimFreq = 124999500;
int phase = 0;

void setup(){
  DDS.begin(W_CLK_PIN, FQ_UD_PIN, DATA_PIN, RESET_PIN);
  DDS.calibrate(trimFreq);
}

void loop(){
  DDS.setfreq(freq, phase);
  delay(10000);
  DDS.down();
  delay(3000);
  while(1);
}
```

Copy and Paste DDS code

- Just copy the code we need into our program that displays frequency on the LCD
- Tidy up a little
- We can ignore calibration for now
- And also always make phase = 0

Yes – it really is that easy

#5 – a complete oscillator!

```
#include <AD9850.h>                    // Add the AD9850 header file
#include <Encoder.h>                   // and the rotary encoder header file
#include <Wire.h>                      // and the I2C header file
#include <LiquidCrystal_I2C.h>         // and the LCD header file

const int ENC_CLK = 2;                 // Encoder Clock pin
const int ENC_DT = 3;                  // Encoder Data pin
const int W_CLK_PIN = 7;               // DDS Clock pin
const int FQ_UD_PIN = 8;               // DDS Frequency Update Pin
const int DATA_PIN = 9;                // DDS Data pin
const int RESET_PIN = 10;              // DDS Reset pin

Encoder enc(ENC_CLK, ENC_DT);          // define the pin maps for the encoder and LCD
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

const long initFreq = 10000000;        // Initial frequency
const long stepSize = 100000;          // Step size that freq changes per encoder change
long encoderCt = 0;                    // Encoder value
long freq = initFreq;                  // Frequency

void setup() {
  lcd.begin(16, 2);                    // send initialisation sequence to the LCD
  DDS.begin(W_CLK_PIN, FQ_UD_PIN, DATA_PIN, RESET_PIN); //and set up the DDS
}

void loop() {
  encoderCt = enc.read();              // get the current value from the encoder
  freq = initFreq + (encoderCt * stepSize); // convert it to a frequency
  lcd.setCursor(0, 0);                 // set the LCD back to beginning of line 1
  lcd.print(freq);                     // write the frequency
  lcd.print(" ");                      //clear any trailing '0'
  DDS.setfreq(freq, 0);                // and set the oscillator (the extra 0 is for phase)
}
```

Critique

- OK it works and would suffice but it's not very good
  - We can keep turning the encoder and generate frequencies out of range (even negative frequencies)
    - Need to check for range limits and reset the encoder if it goes past
    - If encoderCt>maxAllowed then write the max value to the encoder
  - We set the oscillator every time round the loop not when the frequency actually changes
    - Easy – reinstate the encoder newPostition/OldPosition code from #1
    - Better – use a pushbutton so we only oscillate when we've finished dialing the desired frequency
  - The oscillator is uncalibrated
    - Need to use the library's calibrate function
  - The step size is fixed so doesn't allow granularity
    - A simple prototype solution would be just to recompile with a different initial frequency and step size each time
    - Better to add a button and write code to change the step size when the button is pressed
      - This is quite complex code
      - We could use the button on the encoder

Fixing the encoder limits

```
…
…
const long minFreq= 10000;              // Define the minimum frequency allowed
const long maxFreq = 30000000;          // and the maximum
…
…
void loop() {
  long tmpEncoder = enc.read();         // Read the encoder value into a temp
  long tmpFreq = initFreq + (tmpEncoder * stepSize); // and calc a temp freq
  if ((tmpFreq < minFreq) or (tmpFreq > maxFreq)) {  // tmp freq out of range?
    enc.write(encoderCt);                           // yes – write the last good val
  }                                                 // back to the encoder
  else {                                            // the tmp freq is in range
    encoderCt=tmpEncoder;                           // so store the good enc value
    freq=tmpFreq;                                   // and set the frequency
  }
…
…
```

Button press to set frequency

```
…
const int btnPin=4;                          // Define the button pin
…
int buttonState;                             // The button's true debounced state
int prevTmpState = HIGH;                     // temp button state while debouncing
long lastChange;                             // last time button changed
const long debounceDelay = 50;               // debounce time
…
void setup() {
  pinMode(btnPin,INPUT_PULLUP);              // make sure the button works
…
…
void loop() {
  int tmpState=digitalRead(btnPin);          // read current state of the button
  if (tmpState != prevTmpState) {            // has it changed?
    lastChange = millis();                   // yes – note the time
    prevTmpState=tmpState;                   // and record it's new state
  } else {                                   // no – still the same
    if (millis()-lastChange>debounceDelay){  // has it stayed the same for a while
      buttonState=tmpState;                  // yes – it's not just a bounce
      prevTmpState=buttonState;              // so record it as true button press
    }
  }
…
if (buttonState==HIGH) {                     // is button pressed?
        DDS.setfreq(freq, 0);                // yes - set the oscillator
  }
…
```

Hold on a sec – we've seen nearly all of this code before!
It's been directly copied from the Button Debounce slide

Stop! This is getting to complex

- If we add a debounced button and bounded frequency the code starts to get very complex
- This is the time to step away and start to write functions
- Make a high level logical flow of the program steps and then encapsulate the code needed to perform each step in a separate function
- Each function should be self contained and easy to test

A logical Signal Generator

- Initialise everything and do a welcome screen
- Get the freq based on bounded encoder values
- Print the frequency on the LCD
- If the freq has changed then turn off the oscillator (and say so on the LCD)
- Check for the debounced button state
- If the button has just been pressed (i.e. it was previously unpressed)
    - If the oscillator is running turn if off (and say so on the LCD)
    - If the oscillator is stopped turn it on at the frequency from the encoder (and say so on the LCD)

The Index Of Possibilities

If you don't know what something can do, you don't know how you can use it

- This course has NOT been an attempt to turn you all into competent C programmers on the Arduino
  - Really, this has only been a tiny taster of programming
- But to give you an idea of what the Arduino can do
  - Digital and Analog I/O
  - Time sequencing
  - Simple interface to many modules using libraries
  - Easy and rapid no solder prototyping
- The goals were primarily:
  - Next time you have a project you think about whether an Arduino might form part of the solution
  - You have enough basic understanding to either try something simple or to be able to discuss your requirements with someone in the club who can help write the code

Hopefully these goals were achieved

Hopefully you understood at least some of the course

Hopefully there was the odd moment of excitement and enjoyment