Prerequisites

This lab doesn't require any code from previous labs; however, it does expect you to have completed these, as common code repeated from previous labs will not be explained in detail again here. Out of your Arduino pack, you will need: 1 Arduino, 1 A3144 hall effect sensor, 1 LED, 1 $10K\Omega$ resistor, 1 220Ω resistor, and 5 jumper cables.

Objectives

The objective of today's lab is to learn how to use an interrupt to record a value of a sensor whose value is changed via an external event: `https://docs.arduino.cc/language-reference/en/functions/external-interrupts/attachInterrupt/`.

This worksheet will take you through the steps of how to connect a hall effect sensor to your Arduino, monitor when the value of the sensor has changed, and provide visual feedback to an end user of when the value has changed.

Hall effect sensors

In short, a hall effect sensor allows you to detect the presence of a magnet — the value given by the sensor changes when a magnet is near to it. This type of sensor has many applications, but most importantly for us, is it provides the facility to detect movement and ultimately how quick that movement was.

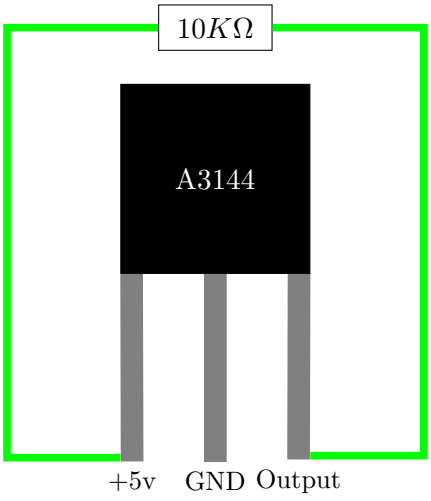

Figure 1: The wiring diagram of our hall effect sensor. The sensor has two bases on a trapezium. This wiring diagram is with the short base facing towards you.

The particular hall effect sensor you have has three legs: a *power* leg, a *ground* leg, and a *digital output* leg. The sensor is shaped like a trapezium. With the short base of the trapezium facing towards you (as depicted in Figure 1) and the legs facing south, the sensor is wired into the Arduino as follows: Arduino +5V→left leg, Arduino GND→middle leg, and Arduino Pin D2→right leg.

Interrupts

The purpose of today's lab, is to gain experience of when and how to use an *interrupt* in an Arduino. In previous labs, we have changed the value of an externally connected device using the command *digitalWrite*. You can read the value of a sensor using the command *digitalRead*; however, this method is not always appropriate when you consider when within the event timeline, the value of the sensor may change.

If we consider how an Arduino works in context of an event timeline. The Arduino runs the code within the 'setup' function, then runs the code in the 'loop' function continuously until it is turned off. If we were to put as the first line of the 'loop' function *digitalRead(PIN)*, we could capture the value of pin at the beginning of the loop. The Arduino would then execute the rest of the code within the 'loop' function, then start the 'loop' again, at which point we would then capture the current value of the pin again.

This design pattern works well in a system that is designed to monitor the gradual change of a value over a period of time — say capturing the temperature outside every five minutes over a twenty-four-hour period; however, this design pattern fails in a scenario where the sensor value could have changed between the two instances of the *digitalRead* logic. In this scenario you need a way of interrupting the Arduino to pause whatever it is doing, run the required code needed to record a value change, and then continue with its previous operations; an *interrupt*.

Because being able to interrupt the 'loop' function is a special case, there may be restrictions — depending on your Arduino — as to which pins this feature is enabled on. In the case of the Nano family of Arduino's, this is digital pins: 2 & 3 (`https://www.arduino.cc/reference/cs/language/functions/external-interrupts/attachinterrupt/`).

1 Creating the circuit

There are two circuits needed for this worksheet, the first is for the hall effect sensor (HES), and the second is for the visual display LED.
HES circuit: Arduino +5V→HES left leg, Arduino GND→HES middle leg, Arduino Pin D2→HES right leg. HES left→$10K\Omega$ resistor→HES right legs.
LED circuit: Arduino Pin D4→220Ω resistor→LED→Arduino GND.

2 System Functionality

The basis of this system is that we have an LED and a hall effect sensor — say placed near the crank of a bike — connected to our Arduino as described in Section 1. We set up the LED as we did in Lab 2 (but on a different pin) and then attach an interrupt to the pin connected with the hall effect sensor. Every time a magnet passes past the field of the hall effect sensor — say a magnet on the crank arm of a bike — the associated interrupt code is called, and it records the time in milliseconds of that event. Within the 'loop' function, we can then calculate if the last time the interrupt happened, is within a 500-millisecond window of the current time. If true, we can turn on the LED, if false, we turn off the LED.

The first thing we need to do is create a blank sketch, this is done by opening Arduino IDE, going to 'File'→'Examples'→'01.Basics'→'BareMinimum'. Once this has opened, we can now start to code up the system.

The code within this section should all be placed within the global scope. Like our previous labs, the first piece of code included is defining the external pins and our constants. In our scenario, this is declaring the '$CRANK\_PIN$' for our hall effect sensor, and then our '$CRANK\_ACTIVITY\_LED\_PIN$' for our visual feedback as to when the hall effect sensor has been triggered. The last part of code is defining the 500-millisecond window in which we will activate the LED:

```
#define CRANK_PIN 2
#define CRANK_ACTIVITY_LED_PIN 4
#define CRANK_PASS_ACTIVITY_DELAY 500
```

Next, we need two global variables: '*current_loop_time*' to store a consistent execution time in milliseconds across the 'loop' function, and a '*crank_interrupt_current_time*' to store the time in milliseconds of the last recorded interrupt:

```
unsigned long current_loop_time;
unsigned volatile long crank_interrupt_current_time;
```

Note: The reason we need to store a consistent time across the 'loop' function is that the 'loop' function may take 20 milliseconds to execute a full iteration of itself. If you request the current time (the *millis()* function) at the beginning of the code, compared to the end of the code, you would get a different value; therefore, to ensure you have a consistent 'current time' across an iteration of the 'loop' function, you store the value at the beginning of the iteration and then use that referenced value throughout the function.

3 Setup function

The code required for the setup function can be thought of as three steps: 1 - enable debugging tools, 2 - initialise global values, 3 - set the pins to the correct modes and attach the interrupt.

3.1 Enable debugging tools

Enable the debugging through the serial: (*hint:* see section 3.1 of Lab 3):

```
Serial.begin(9600);
delay(3000);
Serial.println("System Ready");
```

3.2 Initialise global variables

Both of the global system variables ('*current_loop_time*' & '*crank_interrupt_current_time*') have been created but have not been initialised; they exist, but they haven't been given any initial value. As both of these variables are of the data type *unsigned long* a logical value for initialisation would be zero (the first possible valid value):

```
current_loop_time = 0;
crank_interrupt_current_time = 0;
```

3.3 Pins

The last thing to do within the 'setup' function, is to set up the pins. The LED pin needs to be set to mode *OUTPUT* and then to ensure the LED isn't already on, write to the LED a *LOW* value to turn it off. Next, the hall effect sensor pin needs to be set to mode *INPUT* and an interrupt needs to be added to the pin. In code terms, the interrupt is a separate function that is only called when the interrupt is triggered. In our code, this is a function called '*crankInterrupt*' that we define in the next section of the worksheet. All we are doing in the 'setup' function, is

referencing the '*crankInterrupt*' function, we do not create the function itself.

There are several options when you attach the interrupt to define when the interrupt function is called. In our case we are using the *FALLING* mode (`https://docs.arduino.cc/language-reference/en/functions/external-interrupts/attachInterrupt/#parameters`) as our hall effect sensors report a *HIGH* value when no magnet is present, then the value then drops when a magnet is present:

```
// setup LED pin as output with LED off
pinMode(CRANK_ACTIVITY_LED_PIN, OUTPUT);
digitalWrite(CRANK_ACTIVITY_LED_PIN, LOW);
// setup crank pin as input and attach an interrupt
pinMode(CRANK_PIN, INPUT);
attachInterrupt(digitalPinToInterrupt(CRANK_PIN), crankInterrupt, FALLING);
```

4 Interrupt

As discussed above, we need a new function that appears in the global scope — like the 'setup' & 'loop' function — which can be called by the interrupt attached to the hall effect sensor pin. All this method does is record the current time in milliseconds to the '*crank_interrupt_current_time*' variable. As an interrupt is stopping everything else running on an Arduino, you try to keep the code inside the interrupt to the bare minimum required: in our case, this is simply recording the time. All the intensive computational logic is done within the 'loop' function — determining if the LED should be on and thus turning on or off the LED.

```
void crankInterrupt() {
  // record the current time of interrupt in milliseconds
  crank_interrupt_current_time = millis();
}
```

5 Loop function

The main logic of the loop can be divided into two steps: 1 - calculate if the last occurrence of the interrupt is within the 500-millisecond window, 2 - wait.

5.1 Calculate window

The first part for calculating the window is storing into the '*current_loop_time*' variable the current time in milliseconds. Next, you calculate if the current time and the last occurrence of the

interrupt are within the 500-millisecond window. If true, you turn on the LED with the standard *digitalWrite* method. We also then display a debug message so that if the LED isn't working, we can still see that the logic in the Arduino is working correctly. If the window has passed, we turn off the LED:

```
current_loop_time = millis();
if (current_loop_time - crank_interrupt_current_time < CRANK_PASS_ACTIVITY_DELAY) {
  digitalWrite(CRANK_ACTIVITY_LED_PIN, HIGH);
  Serial.println("Crank Active");
} else {
  digitalWrite(CRANK_ACTIVITY_LED_PIN, LOW);
}
```

5.2 Wait

The final logic within the 'loop' function is to wait 50 milliseconds before the next iteration of the 'loop' function:

```
delay(50);
```

Try it out on your Arduino

You now should be able to compile your sketch ('Sketch'→'Verify/Compile') and then upload to your Arduino ('Sketch'→'Upload'). You should now be able to pass the magnet across a side of the hall effect sensor, and every time you do, it should light up the LED. *Note:* the hall effect sensors are directional, so if the LED isn't lighting up, turn over the magnet and try again.

Help

If you are having issue compiling the project or don't understand where each section goes, you can see a complete version of the sketch with additional source code comments at `https://github.com/dsbrennan/mac-233-arduino-labs/blob/main/lab-4-hall-effect-sensor/lab-4-hall-effect-sensor.ino`.

Additionally, if you are wondering how to wire up the circuit required for this lab on the breadboard, the electronic wiring diagram is below.