LABORATORY EXERCISE 1: INTERFACING OF ANALOG SIGNALS, SENSORS AND ACTUATORS

INTRODUCTION

As explained in the introductory exercise, the overall objectives of this laboratory are:

- To provide practical experience of interfacing a variety of signal sources and sensors to the Arduino Mega 2560 using various peripheral boards
- To give further experience of using the Arduino's dialect of the C language
- To provide experience of the interfacing of a servo motor and incremental encoder using an H bridge, the LS7366R up/down counter and the Arduino itself
- To enable students to see in practice the waveforms associated with pulse width modulation and quadrature encoding.
- To give a "sneak preview" of the use of finite state machines and of the use of interrupts

SAFETY

In practice this is a low hazard laboratory, but it does involve the rotating shaft on a small motor. To avoid any risk of entrapment, **sleeves should be rolled up and long hair tied back**. No personal protective equipment is required.

EXERCISE 1 – INTERFACING WITH DIFFERENT SENSORS

1. Ensure the Arduino is disconnected from the PC at present.
2. Using the small screwdriver provided in your experiment kit, wire up the thermocouple to the high temperature sensor board, ensuring that the red lead is connected to + and the white lead to – on the green terminal block. **Take care not to over-tighten** as the terminal block and its attachment to the board are quite fragile (see the video guide, there are instructions).
3. Now connect the ribbon cable to the high temperature sensor board, and using appropriately coloured jumper leads from your kit, connect the other end of the ribbon cable to the Arduino as follows (and as shown on the diagram above):
   - **Black**: ground or common terminal, connect to GND on Arduino
   - **Red**: 5 V power supply (V_{CC}) for board, connect to 5V on Arduino
   - **White**: this is the output of the thermistor resistance measuring circuit, connect to analog input A1 on the Arduino
   - **Yellow**: this is the output of the thermocouple amplifier, connect to analog input A0.

4. **Get a demonstrator to check your wiring!**
5. Load the program you have modified from **TwoSensorsSkeleton.ino** into the Arduino IDE; it should have already been verified in your programming sessions.
6. Connect the Arduino via the USB cable and ensure that the correct port and board type are set by checking **Tool | Board** and **Tools | Port**.
7. Now compile and upload your program. Open the Serial Monitor from the Tools menu. You should see the room temperature; the thermocouple temperature and the displacement displayed every second.
8. When you have finished, disconnect the Arduino from the PC.
9. Disconnect the various items so the experiment is as you found it. You do not need this for the next two exercises.

EXERCISE 2 – PULSE WIDTH MODULATION: INTERFACING WITH A SERVOMOTOR

Servomotors of different types are widely used for positioning, for instance within machine tools and robotics. Unlike stepper motors, which (if desynchronisation and resonance do not occur) will move under open-loop control to a defined position and will stay there, servomotors rely on closed-loop control to achieve the desired position:

- Actual position is measured continuously and compared with the desired position.
- The difference (error) is used via a control algorithm (decision-making process) to determine the corrective action, typically in the form of a voltage or current which is fed into the motor.
- The current flowing through the motor results in a torque which causes the motor to move to correct for the error.

This process is repeated many times a second to achieve rapid and stable response to changes in the required position.

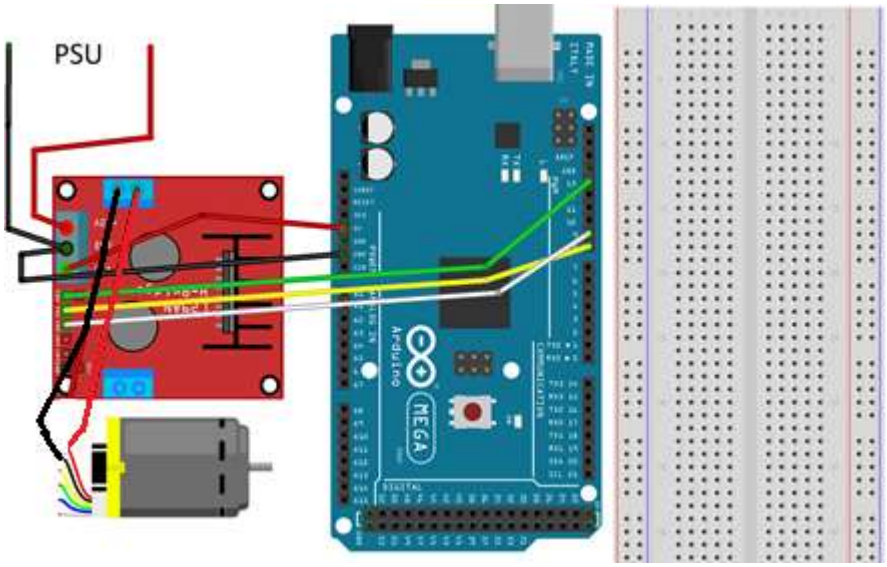
The interface to a servomotor requires the following:

- A means of varying the voltage and/or current fed to the motor – in the present case, this is achieved using pulse width modulation, which (conveniently) is the form of output provided by the Arduino when the loosely-named `analogWrite()` function is used.
- A means of measuring the position – in the present case, this is achieved using an incremental encoder – this will be examined in Exercise 3, “Quadrature Encoder Decoded”.

Within this lab, you will examine the forms of the signal going to and from the motor from this module under open-loop conditions – within lab 2 we will explore the motor’s closed-loop behaviour in more detail.

The lab itself involves driving the motor from an L298N H-bridge driver IC, which in turn is powered from a power supply.

1. Ensure the large bench power supply unit (PSU) is connected to the mains.
2. Switch the PSU on and ensure the ON/OFF button is in OFF. When it is OFF, the screen displays the set-point voltage and current limits.
3. Adjust the voltage to read 5V and the current limiter to read 0.5 A (see the video guide).
4. Turn ON the supply. The displays now read the actual voltage and current. The voltage should remain around 5V, but the current should be approximately zero as no load is connected.
5. Turn OFF the supply once again, and switch off the supply.
6. Ensure the Arduino is disconnected from the PC.



7. Connect the following between the motor and the H-bridge:
   - Motor **Red (positive)** – H bridge **OUT1**
   - Motor **Black (negative)** – H bridge **OUT2**

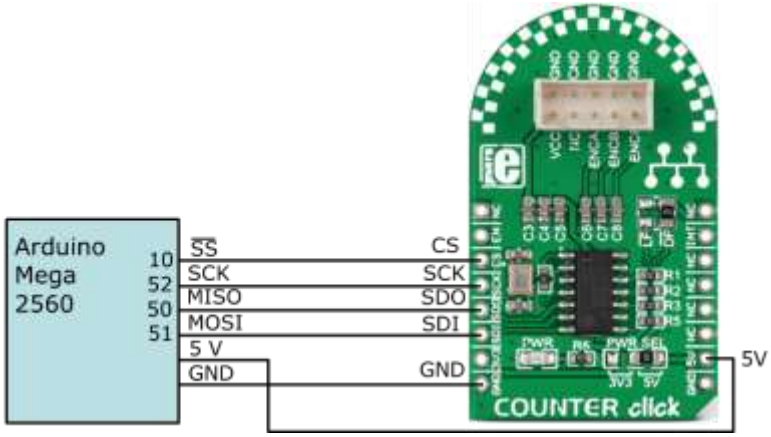
   Leave around 1-2 mm of the metal parts of the leads exposed, do not push the leads all the way in.
8. For the present time, plug the blue, green, white, and yellow motor leads into unconnected strips in the centre of the breadboard so they do not accidentally touch anything.
9. Connect the following:
   - PSU **Red (positive)** – H bridge **12V**
   - PSU **Black (negative)** – Breadboard **vertical rail for GND**
   - H bridge **GND** – Breadboard **vertical rail for GND**
   - Arduino **GND** – Breadboard **vertical rail for GND**
   - Arduino **5V** – Breadboard **vertical rail for 5V** (not the same as GND rail)
   - H bridge **5V** – Breadboard **vertical rail for 5V** (not the same as GND rail)
10. Plug the six-way connector into the L298 board with the green wire nearest the large terminal block:
    - H-bridge **Green (ENABLE)** – Arduino **13**
    - H-bridge **Yellow (IN1)** – Arduino **8**
    - H-bridge **White (IN2)** – Arduino **9**
11. Oscilloscope connections (refer to the video guide):
    - Oscilloscope **CH1 Signal** – H bridge **OUT1**
    - Oscilloscope **CH1 Reference** – H bridge **OUT2**
12. Ensure that the scope is switched on and set to 5V/div sensitivity on Channels 1 and 2 and 0.5 ms/div on the time base.
13. **Get a demonstrator to check your wiring! Also, if you do not understand how to operate the oscilloscope, ask a demonstrator.**
14. Reconnect the Arduino to the PC.
15. Load the program you have modified from **MotorEncoderSkeleton.ino** into the Arduino IDE; it should have already been verified in your programming sessions.
16. Switch the PSU on and ensure it is activated by pressing the small button to lock it in. Ensure you start at 5V.
17. Open the Serial Monitor and ensure that the box in the bottom right hand corner of the monitor reads **"Both NL and CR**" to ensure that your program will understand when a line of text has been sent.
18. Type different values of PWM duty cycle percentage in the range -100 to 100 into the input field. The motor should run forward or backwards for positive and negative values. Sketch carefully the

waveforms on channels 1 and 2 of the scope (which are the signals seen by each side of the motor with respect to ground) and the math trace (which is the potential difference across the motor) for (say) 60% and -60%. You will need to draw these neatly on your answer.

19. Now increase the PSU voltage up to 15V for higher speed. **Do not go higher than that, you may blow up the H-bridge.** Observe the traces again.
20. When finished, close the monitor, switch off the PSU and disconnect the Arduino but do not disconnect anything else.

EXERCISE 3 – QUADRATURE ENCODER DECODED

1. Ensure the power supply is switched off.
2. Plug the Counter Click quadrature counter board into the breadboard, around halfway along the breadboard.
3. Plug the encoder connection wires from the Counter Click board into separate transverse strips in the breadboard and connect the appropriate coloured wires from the motor to the Counter Click leads as follows, matching the colours:
    - Counter board **Blue** - Encoder **Blue (5V power supply)**
    - Counter board **Green** - Encoder **Green (power ground)**
    - Counter board **Yellow** - Encoder **Channel A**
    - Counter board **White** - Encoder **Channel B**
4. Connect the following (we will use the encoder output directly with the Arduino using the FSM you have developed and the individual channel timer):
    - Encoder **Channel A** – Arduino **2**
    - Encoder **Channel B** – Arduino **3**
    - Encoder **Channel A** – Arduino **47 (timer/counter input pin T5)**
5. Make the power and SPI serial communication connections between the Arduino and the Counter Click board as follows:
    - Counter **VCC (5V supply)** – Breadboard **vertical rail for 5V**
    - Counter **GND** – Breadboard **vertical rail for GND** (not the same as GND rail)
    - Counter **SCK (system clock)** – Arduino **52**
    - Counter **CS (chip select)** – Arduino **10**
    - Counter **SDI (slave data input)** – Arduino **51 (MOSI – Master Output Slave Input)**
    - Counter **SDO (slave data output)** – Arduino **50 (MISO – Master Input Slave Output)**



6. Leave the scope ground clip connected but re-connect Channel 1 scope probe to Channel A of the encoder (yellow lead) and Channel 2 scope probe to Channel B (white lead). **Please note that this has not been**

covered in the video guide. You know how to connect oscilloscope probes and view readings from the previous exercise.

7. **Get a demonstrator to check your wiring to avoid expensive damage!**
8. Re-connect the Arduino.
9. You should already have the working code loaded onto your Arduino from the previous exercise. If not, load the program you have modified from **MotorEncoderSkeleton.ino** into the Arduino IDE; it should have already been verified in your programming sessions.
10. Enter values 100 (to run motor fully forward) or -100 (fully in reverse) and see whether the encoder counts recorded by your state machine program agree with the values from the LS7366R hardware counter on the Counter Click board. Do you get any errors recorded? Note that your state machine code "polls" (looks at) the encoder inputs every time the `loop()` code executes, which is at a rate determined by software. Can you identify any reason why the microprocessor might miss long sequences of encoder pulses even though not many errors are recorded?
11. You should also note that the internal counter (using Timer 5, counting pulses received on pin 47) on the Arduino is giving a count value that should be a quarter of the value from the hardware counter. Does that behave at high speed running?
12. Carefully observe the way the numbers are changing when the motor is running forward and when it is running backward. What is different between the two?
13. How does the error look like at higher speeds? Is it different compared with lower speeds?
14. **BONUS:** Now comment out the line `updateEncoderStateMachine()` in `loop()` and uncomment the following two lines in `setup()`:

```
attachInterrupt(digitalPinToInterrupt(channelA),
updateEncoderStateMachine, CHANGE);

attachInterrupt(digitalPinToInterrupt(channelB),
updateEncoderStateMachine, CHANGE);
```

These call the `updateEncoderStateMachine()` function whenever there is a change of state on channel A or channel B from the encoder – in other words, exactly when (and only when) the state needs to change, regardless of anything else the Arduino is doing. Test this with motor PWM values up to (say) 30%. Then try at faster and faster speeds. Do you get any errors? Does something unexpected happen at high speeds? Why?

15. **BONUS:** A rather annoying feature of the system in its present form is that it makes an audible note (though this is often the case for pulse width modulation systems). This is because the timer-counter (Timer 0) is fed from a 16 MHz clock source which has been reduced by a prescale value of 64 to 250 kHz so counting up to (say) 255 results in a frequency well within the audible range (20 Hz – 20 kHz). Can you increase the frequency above the audible range by (for example) eliminating the prescaling so that Timer 0 is fed directly from the 16 MHz supply, without affecting any other settings? Hint: the information you need is given on pages 129-30 of the Atmega2560 datasheet.

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 16-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on rising edge

You will therefore need to add the following lines to **`Setup()`**:

```
TCCR0B = TCCR0B & 0xF8; // Clears CS00, CS01 and CS02 i.e. bits 0-2
TCCR0B = TCCR0B | 0x01; // Sets flag CS00 i.e. prescale value 1.
```

to clear three bits of register TCCR0B and set one of them. In practice this goes too far (giving a frequency of around 32 kHz, too fast for the driver circuits to react correctly to the pulses) and the system no longer works very effectively (the motor tends to run too slowly, and something else will be messed up entirely – why?).

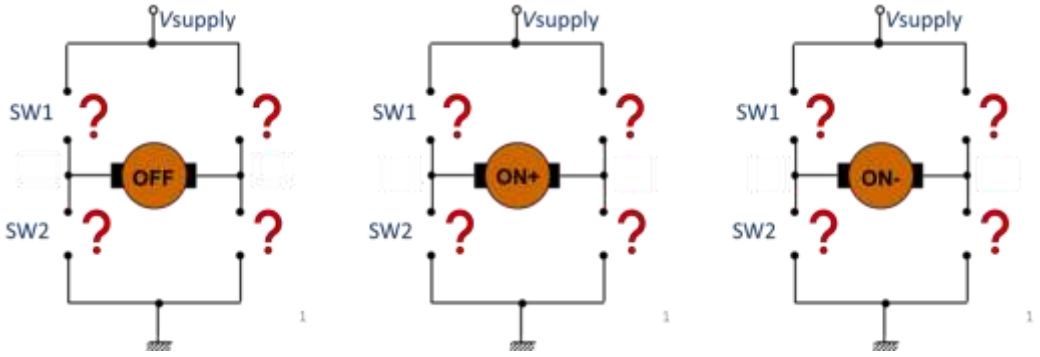
16. When you have completed the experiment, switch off the PSU, shut down the Arduino program, unplug the Arduino from the USB port and dismantle the experiment. Put the Counter Click board back in the kit box along with the screwdriver, LVDT, jumper leads etc. Finally put the experiment board back on top of the box divisions and close the lid.

COURSEWORK SUBMISSION

Well formatted and organised submission: A zip file containing the answers to your questions, ideally as a PDF, including listings of the programs you have written and submitted as preparatory work, including any corrections made in class. Also please include in the zip file all the programs themselves as Arduino source (.ino) files. **[20%].**

Questions (please limit your answers to a single side of an A4 page):

1. How does the ADC (Analog to Digital Converter) functionality work in an Arduino? What electrical signal (current, voltage, resistance, impedance etc.) is read on the pin and what is the digital value after conversion? Explain the relation between the physical signal and digital information. **[10%].**
2. The temperature sensor kit (thermocouple and thermistor daughter board) has a 3.3V amplifier, i.e., the output voltage is between 0 and 3.3V for the complete range of detected temperature (equation 4 in the programming lab sheet). Why do we still use 5V as the *Vref* when we convert the ADC-converted value back to it analog value (equation 1 in the programming lab sheet)? Why is this not the most optimum use of the ADC pin and how can we increase the resolution of the detected signal? **[10%].**
3. Please paste camera clicks of the PWM waveform (oscilloscope readings) as you observed in exercise 2 for the following duty cycle settings: 10%, 25%, 50%, 75%, 100%. What is the distinction between 100% PWM and all others? Clearly mark which region is the ON region and which region is the OFF region. Why is the voltage not zero in the OFF region? **[10%].**
4. Sketch the switch configuration for ON and OFF regions in the H-bridge circuit diagram below. What is the difference between ON (positive) and ON (negative)? What is the PWM frequency by default in an Arduino? What is the frequency of the audible note that we hear? Please provide evidence through the oscilloscope readings of the PWM waveform. **[10%].**



5. Please paste camera clicks of the quadrature signals (oscilloscope), when running in the positive and negative directions. Remember to mark the zero voltage datum marks. **[10%].**
6. Although it only counts a quarter of the pulses (as it only looks at rising edges of one pulse train, not all four transitions per cycle) the internal counter (Timer 5 configured to count pulses on pin 47) seems very reliable, at least under some circumstances. Could this be used as an alternative to the LS7366R quadrature decoder? Explain your answer. **[10%].**
7. When you implemented your quadrature decoder coding to run as software on the Atmega microprocessor, what shortcoming did you observe? Can you identify what the program might be doing when this problem occurs? **[10%]**
8. So far, in this laboratory, you have demonstrated how to drive a DC motor at variable speed, in the forward and reverse directions, and to measure how far it has rotated (again, either forwards or

backwards). How can these approaches be used to achieve accurate positioning of a motor, and what technique(s) from elsewhere in your studies (specifically, from MM2DYN/MMME2046) will you require to achieve this? (Note: you do NOT need to implement this approach within the present exercise, that is the subject of later study) **[10%]**