cs281: Introduction to Computer Systems

Lab08 – Interrupt Handling and Stepper Motor Controller

Overview

The objective of this lab is to introduce ourselves to the Arduino interrupt capabilities and to use them to build a stepper motor controller. The interrupt handler will allow the Arduino to ”interrupt” a currently executing thread (a loop) to respond to an important event. This critical features allows the Arduino to monitor and respond to its external environment even while already performing a different task. Stepper motors are used in everything from robotics, commercial applications, manufacturing – they are instrumental in allowing a computer controller to effect physical change in the environment. A controller is needed to provide the correct sequence of commands to move and position a stepper motor.

You are to write a lab report for *the last part of this project only*. See details below. Your report is due Friday (in class) following the lab.

Arduino Interrupts

The Arduino gets input from the external world by reading values on its input pins. However, if the Arduino is currently running a long program, it cannot also simultaneously read input values. It might miss an important input signal while it is busy doing something else. The interrupt capability is essential in overcoming this problem. Essentially, an interrupt can be configured so that if an external signal is received, the Arduino can pause its current program, respond to the interrupt signal, and then continue its normal execution.

The two important aspects of Arduino interrupts are the interrupt input and the interrupt handler. The Arduino Uno board has two interrupt inputs, INT0 and INT1, which are mapped to hardware pins 2 and 3 respectively. These pins can be configured to respond to a rising edge pulse (transition from 0 to 1), a falling edge pulse (1 to 0), or a low level (input value of 0). The setup routine can be used to configure an interrupt detection on one of these input pins, set the type of interrupt (rising, falling, low) and to register an interrupt handler. The interrupt handler is a function that is called when an interrupt is received.

Imagine that the ”loop”function is busy executing, and an interrupt is detected on the registered input pin. The loop algorithm then stops wherever it is. The Arduino calls your interrupt handler routine and executes it completely. Then the Arduino resumes executing the loop function at the point it left off.

We will build a simple circuit and write a simple interrupt method to test this capability.

1. Wire your Arduino to power and ground as normal.
2. Wire output pin 13 to one of your logic monitors. We will toggle this output on and off according to the interrupt.
3. Wire your breadboard pushbutton (PB1) to the input pin 2 on the Arduino; use the NC connection on the button – normally closed = low level when the button isn't pressed. This will be our interrupt signal. Make sure your pushbutton has a pullup resistor [1]

[1] You should have a resister wired from the pushbutton NC to +5v. This will pull up the value on the button when it is open. It also helps to debounce the signal so that it won't toggle back and forth quickly while you are pushing.

4. Write the following program and upload/run it on the Arduino. Experiment pressing the button and record the observed behavior.

Question: How is the interrupt signal fundamentally different than reading a value using `digitalRead()`? List all the ways they differ both in operation and in program implementation.

```
#define PIN_OUT 13
#define PIN_INTERRUPT 2
#define PAUSE 100

int light = 0;

void buttonPressed()
{
    Serial.println("Interrupt captured");
    light = 1 - light;
    digitalWrite(PIN_OUT,light);
    Serial.print("\t Light = ");
    Serial.println(light);
}

void setup()
{
    Serial.begin(9600);
    pinMode(PIN_OUT, OUTPUT);
    cli();   // disable all interrupts
    pinMode(PIN_INTERRUPT, INPUT_PULLUP);
    attachInterrupt(0, buttonPressed, FALLING);
    sei();  // enable interrupts
    light = 0;
    digitalWrite(PIN_OUT,light);
}

void loop()
{
    int silly = 1;

    while ( silly <= 100 )
    {
       Serial.print(silly);
       Serial.println(" ways to waste time with this program.");
       delay(PAUSE);
       silly++;
    }
}
```

Stepper Motors

Stepper motors are typically used for tasks that require precise positioning or a small/measured number of rotations. Stepper motors are more precise both in small movements and in exact positioning than servo motors, but they are also more complex in design and more complex to control. And whereas DC brushless motors typically spin rapidly when voltage is applied, a stepper motor requires a sequence of pulses to make it move. By controlling the timing and order of the sequence, you can ”move” your stepper motor in very small and highly precise increments. Stepper motors are also closed loop systems meaning that once you position your motor, you can trust that it achieves its intended position without having to rely on sensors to verify its movements. The tradeoff for this precision is a slight increase in the complexity of running a stepper motor over other motors. We will build a simple stepper motor controller in this part of the lab. **It is important that you follow the wiring instructions very carefully; a crossed wire here can burn out your motor or cause other damage – these parts are sensitive to incorrect inputs. Carpenters use the mantra: ”Measure twice, cut once”. We will adapt this to mean ”check wiring twice, then turn on power”.**

Obtain your stepper motor and observe the 5 colored wires attached to it. The red wire is a voltage reference wire; the other four colors are used in pairs to control internal electromagnets in the motor and cause it to spin. We need to send the following set of pulses to the stepper motor in this exact sequence:

Blue In 1	Pink In 2	Yellow In 3	Orange In 4
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	0	0
1	1	0	0
1	0	0	0
1	0	0	1

This sequence of pulses will turn the internal magnets on and off in such a way to rotate the motor a step for each input.

1. You will need a 28BYJ-48 5V Stepper Motor. This will be given to you.
2. You will also need a motor auxiliary board, a ULN2003a board. But this board is not labeled (in English) – the professor will give this to you.
3. You will need some additional wiring harnesses that will also be given to you.
4. Wire the Arduino to power and ground if you have not already done so.
5. On the ULN board, find the two power pins. They are labeled + and - and have 5-12 near them. Use the female to male wires (different than the normal ones) to conned the - pin to Ground on the breadboard. Use another female to male wire to connect the + pin to +5v on the breadboard. Double check that you have them right.

6. Use your 4-wire harness to connect to the ULN pins. Connect the blue wire to IN1, green to IN2, yellow to IN3, and orange to IN4. Note these correspond to the colors on the stepper motor except that we had to match green on the wiring harness to pink on the stepper motor.

7. The other end of this 4-way wiring harness will now be connected to the Arduino. Use the appropriate colored wires in your Arduino kit to connect blue to Arduino pin 7 (use a blue wire), green to pin 6, yellow to pin 5 and orange to pin 4.

ULN PINS	4-way Wiring Harness	Arduino Inputs
IN1	blue	7
IN2	green	6
IN3	yellow	5
IN4	orange	4

   Again, double check all these connections to be sure they are wired correctly.

8. We will now connect the stepper motor. Plug the plastic clip at the end of the wiring harness on the stepper motor into the socket on the ULN board. Notice the notches on the clip and socket so that the harness goes in only one way. The color order on the stepper motor harness should match the color order of the 4-way wiring harness except that pink/green are matched and there is no corresponding connection for the red wire.

9. From the class webpage, obtain the `StepperMotor.h` and `StepperMotor.cpp` files. Put them on your laptop. Open them with a text editor and read through them. You will see a stepper motor class that implements the sequence of pulses in the table above.

10. You now need to create a library where you can upload the stepper motor files. This library needs to be in a specific place and format within your Arduino application folder on your laptop.

    On your laptop, create a new folder in the location Arduino → libraries. Name this folder `StepperMotor`, the same as the files and the class. Copy your `StepperMotor.h` and `StepperMotor.cpp` files into this folder.

11. In your Arduino application, open a new sketch and enter the program listed below. Once your program is complete, go to the Arduino menu, choose Sketch→Import Library. You should see your "StepperMotor" library at the bottom of this list. Select it. This will automatically insert a

    ```
    #include <StepperMotor.h>
    ```

    at the top of your program.

12. Now compile and upload. You should see your stepper motor rotating in the clockwise direction (when viewed behind the shaft) for a few seconds and then it will stop. You can reload your program to run it again.

```
// This program will run a stepper motor for a few seconds
#include <StepperMotor.h>

// 4 pins of the stepper motor board
```

```
#define _PIN1 7
#define _PIN2 6
#define _PIN3 5
#define _PIN4 4

StepperMotor stepper(_PIN1, _PIN2, _PIN3, _PIN4);
int run_once;

void setup()
{
    Serial.begin(9600);
    Serial.println("Running");
    stepper.setPeriod(5);
    run_once = 0;
}

void loop()
{
    if ( run_once == 0 )
    {
        stepper.move(1024);
        stepper.stop();
        run_once = 1;
     }
}
```

Question: What does the input parameter to the `stepper.move()` command do? What happens if you change this value? What happens if you make it negative?

Question: What does the input parameter to the `stepper.setPeriod()` command do? What happens if you change this value? (Try 1 for a value).

Question: What do you observe about the extent of rotation? What do you conclude about how many steps complete a full rotation?

Project Design

You are to design a project to achieve the following design criteria. You will have to decide what hardware and software to build to complete the project. Try to keep the project as simple and elegant as possible.

Design Criteria

1. You are designing a circuit for a child's robotic toy. The toy cycles between four "states": idle - forward - idle - backward. There is a button on the toy. Pressing the button advances the toy into the next state (backward transitions to the start idle state). When the toy is first turned on, it must start in the first idle state.

2. You must use the PB1 on your breadboard to simulate the button on the toy.

3. You must use your stepper motor as the drive mechanism for the toy (imagine the shaft connected to a drive wheel on the robotic toy). You do not have to build any other aspects of the toy (no wheels or plastic robot shells or anything else).

4. Step away from the breadboard and laptop and spend some time designing the solution with pencil/paper and your mind. What hardware components will you need? What software will you use? How will you keep track of the state of the toy? There are probably parts of this solution that could be done in hardware or software. There is no single right solution for this project, but there are definitely "better" solutions. Spend time thinking BEFORE you jump prematurely into the build phase and construct a suboptimal design.

5. You want your design to be:

   (a) Elegant: simple, clean, efficient.

   (b) Robust: repeatable, dependable, insensitive to errors.

   (c) Well documented: the other project members (the ones designing the child's toy) need to know your design.

6. Now decide how to divide up the tasks to different lab partners. Go to work and build your project. Illustrate for your professor / lab assistant.

Your lab report for this project is only for the last part. You are to write a design document in which you outline your overall design, present the independent hardware and software components, and talk about how your design meets the three criteria listed above.