Lab 2 CSE/ECE474 Sp 2023

Prof. Vikram Iyer[1] University of Washington

Digital I/O and timing of outputs

Introduction

In this lab, we will drive some digital outputs to flash an 8x8 matrix of LEDs and make a tone on your speaker. Emphasis will be placed on the correct timing of multiple tasks by verification of light motion and audio tone. Timing and frequency output will be verified on the oscilloscope.

Learning Objectives

With successful completion of this lab, the student will be able to

- Manipulate hardware registers/ bits, without the use of existing libraries, to perform low-level hardware functions.
- Coordinate multiple concurrent tasks with round-robin scheduling

Equipment

1) Arduino Mega Microcontroller board, breadboards, wires
2) External Arduino power supply (recommended)
3) 3 LEDs and 250-500 Ohm resistors
4) Small 8-Ohm Speaker
5) 8x8 LED Matrix
6) 2-way thumbstick control

Technical Requirements

The lab will be performed and written up in teams of two. As a member of the two-person lab team, you are responsible to:

1) Work with your partner to complete the lab, writeup, and demo video.
2) Understand all wiring and code in your solution. (if code is divided up, the author/debugger of the code may have a greater understanding of the code, but “I don’t know because my partner wrote that code” is NOT an acceptable answer.)
3) Turn in at least one submission per group, and list your partner(s) in the submission comments. Report header/title page must include both team member names and student #s.

Turn in requirements: each team must show achievement of the above Learning Objectives by

1) Performing all demonstrations in **BOLD** below. Audio tones must be “smooth” not glitchy.
2) Submit a lab writeup according to the Lab Guidance Doc.

[1] Thanks for valuable guidance from Prof. Blake Hannaford

Required Procedures

1. **Hardware bit manipulation**
   For this part (part 1.1-1.3) you may use Arduino functions `pinMode()`, `digitalWrite()` and `delay()`, unless instructed otherwise (e.g. part 1.4).

   1.1. Wire up 3 LEDs with 500-250 Ohm series resistors (exact R-value doesn't matter) on pins 47, 48, 49.[1]

   1.2. Create code to initialize these pins as outputs (using Arduino's `pinMode()` function), and **demonstrate** flashing them on and off in a sequential pattern (using Arduino `digitalWrite()`) with a period of 1 second (and no other functions). LEDs should flash in order: 47-->48-->49-->47-->48 … etc, each LED on for 333ms.

   1.3. Identify the ATMEGA2560 Port number (cap letter) and port bit numbers corresponding to pins 47-49 (your LEDs). See the "Arduino MEGA Hardware Guide ECE474".

   1.4. Redo part 1.2 **without using the `pinMode()` & `digitalWrite()`** functions. See the hardware doc for background on how to directly set up and use digital I/O pins. Satisfaction of this part requires code that:

   1.4.1. Uses the defined macros such as **DDR3, PORT3,** etc. registers that control the pins requested in 1.1.

   1.4.2. Uses these definitions to **demonstrate** flashing the LEDs in the same pattern of 1.2

2. **16-Bit Timer/Counter**

   2.1. Review the lecture material on the 16-bit Timer/Counters and related material in the hardware guide doc above and the description of TimerCounter4 in the 2560 datasheet.

   2.2. Identify the (pre-defined) #defines for all registers and bits you need for TimerCounter4 to output a square wave. You may define your own macros if you prefer. For example,

```
// (see page 56 of data sheet)
#define TIMER4_ON_BIT  PRTIM4
```

   2.3. Write a task that starts the timer such that it generates a square wave on pin **OC4A** at a specified frequency. **Hints**:

   2.3.1. Choose an appropriate waveform generation mode (Table 17-2).

   2.3.2. Do not enable interrupts.

   2.3.3. Set or clear all required register bits using the technique of Section 1.4 above. Best to write a bit_set() and bit_clr() or bit_set_clear() function of your own. When setting or clearing specific register bits, be sure to not change *other* register bits.

[1] If you have an important reason to change these pin numbers you may do so, but document your reason in the report.

2.3.4. Be sure to properly initialize the output pin you need. Make sure OC4A is set up as an output via DDR4 so that its pin can drive the speaker.

2.3.5. PWM modes can be used if set for 50% duty cycle. Connect the output pin to the speaker. See the “Pro-tip” below.

2.4. **Demonstrate** the ability to program the 16-bit Timer/Counter to directly output the following tone sequence on your speaker (cycle it).

2.4.1. 400 Hz for 1 sec
2.4.2. 250 Hz for 1 sec
2.4.3. 800 Hz for 1 sec
2.4.4. Silence for 1 sec

It should sound like THIS.

Pro Tip: Driving a speaker with a 5V square wave is not exactly “Hi-Fidelity sound”. Specifically, a bunch of harmonics will be generated because 1) it’s a square wave, 2) we are driving the speaker hard and the speaker’s cone will slam into the end of its motion range, severely distorting the sound. In my tests with a very cheap 8 Ohm speaker, sometimes the harmonics are so strong that the frequency subjectively sounds wrong (especially when playing a tune as in 2.4 above). For example, if the 2nd harmonic is too strong on a 400Hz signal, it will sound like 800Hz. Experiment with a series resistor between the output pin and your speaker to get a softer, less harsh sound with fewer harmonics (by driving the speaker with less current). The Oscilloscope can really help you out here.

3. Concurrent Tasks

3.1. Define three tasks:

3.1.1. Task A: (LED Sequence) Same as part 1.4.2 above
3.1.2. Task B: (Timer tone output) Same as 2.4 above
3.1.3. Task C: Control the operation of Task A and Task B as follows:

3.1.3.1. Task A for 2 seconds
3.1.3.2. Task B by itself with no LEDs for 4 seconds
3.1.3.3. No outputs for 1 second
3.1.3.4. Repeat the above steps

3.2. **Demonstrate** simultaneous operation of Task A and Task B. Modify Task C to operate A&B as follows (You may modify Task A and B if necessary to achieve this):

3.2.1.

3.2.1.1. Task A runs alone for 2 sec.
3.2.1.2. Task B alone for one music cycle.
3.2.1.3. Task A and Task B run at the same time for 10 sec.
3.2.1.4. No tasks for 1 sec.
3.2.1.5. Repeat

Thanks for valuable guidance from Prof. Blake Hannaford

3.2.2. Use the Arduino `loop()` function to call the tasks in order (a basic round-robin scheduler). Declare and clearly document some flags (global variables visible to all 3 tasks) so that Task C can "signal" Tasks A&B when to start and stop.

3.3. Modify your part 3.2 to change Task B to play the tune "Mary Has a Little Lamb" (see Appendix below for the specific frequencies) and **Demonstrate** recognizable glitch-free music.

4. **Interactive Display**

Add an interactive XY LED display matrix which moves a dot around in response to thumbstick inputs.
Your accessory kit should have a generic XY thumbstick and an 8x8 LED matrix like the 1088BS model here.

See VIDEO.

There are two common types of 8x8 LED matrices sold as Arduino Accessories:

- "Raw" 8x8 matrix (exposes one pin per row and one pin per column for a total of 16 connections.)
- Serial interface 8x8 matrix. Includes one or more chips to allow you to control all 64 LEDs with fewer (than 16) connections but requiring added software.

Note: for each of the following options, you may not use pinMode() or digitalWrite().

SPI 8x8 Matrix (e.g. MAX7219 dot matrix) (non-RAW)

Consult this background document by Ishaan. Sample code called LED_matrix.ino is on the course website. You will not need to modify the `spiTransfer()` function.[3,4]

Write a function that can set or clear a single LED at a specified row and column. Use it in an otherwise empty Arduino sketch to test your wiring.

Now wire the thumb stick. Under the hood, the thumb stick consists of two variable voltage dividers, one senses X deflection and the other Y deflection. Plug the thumbstick into your solderless breadboard and connect its X and Y output pins to two Arduino Analog inputs. Use the arduino `analogRead()` function to read the X and Y values. Analog inputs convert 0-5VDC to a 10 bit int range from 0-1023. Write a function to convert this range to 0-7 (the row or column indices by which the dot will move).

You might want to test your thumbstick reading / mapping function separately by using the `Serial.print()` function to print converted row/column data back to your PC over the USB port.

Thanks for valuable guidance from Prof. Blake Hannaford

Now combine reading the thumbstick with flashing the LED at the appropriate Row/Col. Specifically, in `loop()`

1) Read voltages from thumbstick and convert to row and column
2) Turn on the LED at {row,col}
3) Delay 50ms
4) Turn off the LED at {row,col}

Demonstrate ability to reach all 64 LEDs with smooth and rapid thumbstick motion. See VIDEO.

Thanks for valuable guidance from Prof. Blake Hannaford

Turn In Requirements

CSE474 rev 1.0 Spring 2023

Learning Objectives

With successful completion of this lab, the student will be able to

- Manipulate hardware registers/bits, without the use of existing libraries, to perform low-level hardware functions.
- Coordinate multiple concurrent tasks with round-robin scheduling.

(refer to Overall Lab Guidance Doc. Spring 2023)

Report Turn In:

Turn in a PDF report following the document template documents. Include a section detailing each partner's contributions, and be specific. This PDF should be turned in as a standalone document, not in a .zip file.

In-Lab Demo:
You may combine all steps into a single code file. Compile it multiple times with commenting out code to achieve the below specifications. You might also consider running each code block in an if-statement and changing a `#define` symbol to select functions.

Demo must include:

1. Sequential Flashing of LEDs in Part 1.4, with each LED on for 333ms. Show where the code uses the DDR and PORT registers.
2. Speaker tone output in Part 2.4. Show where the code uses the 16-bit Timer/Counter registers.
3. Operation of Task C as outlined in Part 3.2.1.
4. Operation of 3.2.1 but with Task B playing "Mary Has a Little Lamb"
5. Working LED matrix dot controller with thumbstick
6. Simultaneous output of "Mary had a little lamb" with smooth function of the thumbstick/8x8display.

Be prepared to answer the following questions.

7. How could you implement the pinMode() and digitalWrite() functions?
8. How do you change the frequency of the square wave generated on OC4A?
9. How do you achieve simultaneous operation of Task A and B in Task C despite using a round-robin scheduler?
10. How do you determine which index to move the dot to in your LED matrix dot-controller?
11. Any other questions at the discretion of the instructor/TA

Thanks for valuable guidance from Prof. Blake Hannaford

Code Turn In:

Turn in ALL code in .ino and .c files (if applicable).Submit a zipped folder containing all .ino and .c files, and keep Arduino files in their sketch folder. Comment out (but do not delete) code which might be required for intermediate steps but is not required for the final code. This will help us give you partial credit!

Appendix: “Mary Had a Little Lamb” [Source of this snippet]

For our purposes pay attention to the frequency of each note. You do not have to use the note encoding scheme (i.e. “c” = 261Hz, in `int melody[]`) below. It’s nice though. (**Warning**: Arduino C can get a bit messed up with 1-character #defines like “d”. I got a better result changing them to something like `#define NOTE_d.`) Also, for better playback of the song (i.e. to make the song more recognizable), hack your task to include a brief silence pause after each note (I think that is what they are doing with “R” in this example)..

```
#define c 3830 // 261 Hz
#define d 3400 // 294 Hz
#define e 3038 // 329 Hz
#define f 2864 // 349 Hz
#define g 2550 // 392 Hz
#define a 2272 // 440 Hz
#define b 2028 // 493 Hz
#define C 1912 // 523 Hz
#define R 0

int melody[] = { e, R, d, R, c, R, d, R, e, R,e, R,e, R,d, R,d, R,d, R,e, R,g,
R,g, R,e, R,d, R,c, R,d, R,e, R,e, R,e, R,e, R,d, R,d, R,e, R,d, R,c, R,c };
```

Thanks for valuable guidance from Prof. Blake Hannaford