

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

Turn In and Demo Requirements:

[\[LINK\]](#)

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.

In their demonstration, 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) Turn in a demo video and lab writeup according to Lab Guidance Doc.

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](#) (Spring 21 version still valid) 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)
```

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

```
#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.
 - 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
 - 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()`.

Serial Interface (non-RAW)

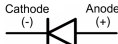
Consult this [background document](#) by Brody. You may use `shiftOut()` function.

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

Consult this [background document](#) by Ishaan. Sample code called `LED_matrix.ino` is provided in the 474 Google Drive “Lab 2” Folder. You will not need to modify the `spiTransfer()` function.^{3,4}

Raw 8x8 matrix

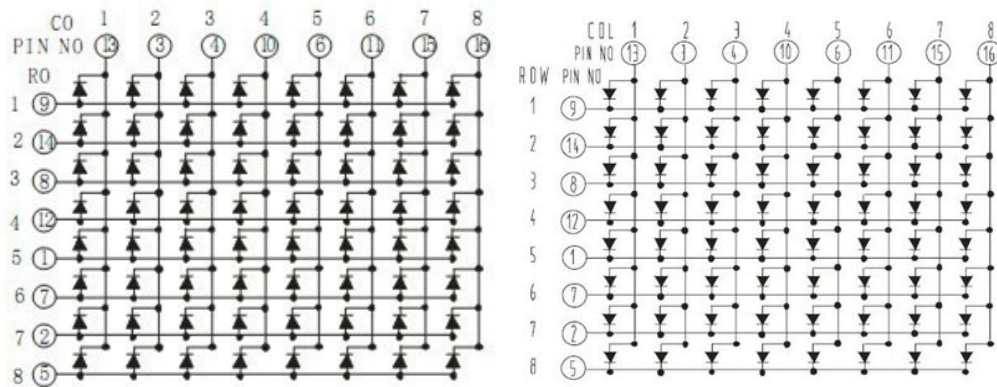
For the display alone (no chips), you connect a pin to each row and another pin to each column. Drive a row to (e.g. low, see below) and then you can light pixels in the row by driving the column lines.

Review Diode terminology: the positive side when forward biased (conducting and emitting light) is the Anode: . The negative side is the Cathode.

There are many different schematics for these 8x8 displays. Check yours CAREFULLY.

³Revised 27-Apr-21

⁴Note: code updated 4/29 to use only 2 bytes instead of 16bytes, functionality was not affected



A Common Anode 8x8 led display (Right: type 1088BS, “CO” = col, “RO” = row) and a Common Cathode Display (Left). **Note:** pinouts may vary. Check your model number carefully.

Use your DMM to verify one diode: (procedure may vary depending on your specific type of LED matrix).

1. Set your DMM to diode setting.
2. Connect red (positive) to pin 9 and black (negative) to pin 13 of the LED matrix (change pin numbers according to your schematic to get a CORNER LED).
3. If LED in the corner lights and conducts, your display matches the Common Cathode display on the left (type 1088BS) but be sure you carefully identify the correct schematic and pinout for your specific device.
4. If you have to reverse red and black from DMM, then you have a common anode device. Pinouts vary a lot for different “1088” 8x8 displays so check yours carefully.

Complete this worksheet to wire your display pins to the Arduino Mega. For the Arduino pins, you have many options. Select appropriate pins according to

- 1) Convenient wiring runs:
 - a) Short distance from Arduino pin to LED pin
 - b) Sequential LED pins go to sequential Arduino pins (neatness)
- 2) The arduino pins can be used as OUTPUT pins (i.e. they do not conflict with other necessary functions)..
- 3) The correct LED matrix pins depend on your specific model.

Examples in light green for Row 1 and Row 2 LED matrix pins are for the type 1088BS specifically. Connect to arduino pins to make clean wiring between Arduino and 8x8 LED matrix. Arduino connector pin Examples in yellow can be changed as you wish.

Function	Arduino Mega "Digital" connector pin, wire color, 2560 Port Code	LED Matrix Pin (example for type 1088BS only!)
Row 1 enable (Low)	47 B	9
Row 2	48 Y PB 3	14
Row 3		
Row 4		
Row 5		
Row 6		
Row 7		
Row 8 enable (Low)		
Col 1 Drive (High)		
Col 2		
Col 3		
Col 4 Drive		
Col 5 Drive		
Col 6 Drive		
Col 7 Drive		
Col 8 Drive		

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.

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](#).

REQUIRED for non-RAW matrix, extra credit for RAW matrix⁴ : 5pts

Integrate the interactive Display as a task with Task B. **Demonstrate** simultaneous output of “Mary had a little lamb” with smooth function of the thumbstick/8x8display.

Special Grading Criteria

- The following is extra-credit if you used the “RAW” matrix, otherwise, it is mandatory: “Integrate the interactive Display as a task with Task B. **Demonstrate** simultaneous output of “Mary had a little lamb” with smooth function of the thumbstick/8x8display. “ This is because completing the dot-matrix task using the non-RAW matrix does not access the LEDs at the register level, and some work has been abstracted/given to you.

⁴Revised 27-Apr-21

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 };
```