**Lab 2:**  Digital I/O and timing of outputs

Keerthi Pitchapati, 2076053 7-May-2024

Rhys Shah, 2133960

Assignment: ECE474 Lab 2

**Introduction**

In Lab 2 of the CSE/ECE 474 course at the University of Washington, we explored the intricacies of manipulating hardware registers and coordinating multiple tasks concurrently without relying on pre-existing libraries. This hands-on lab emphasized direct control over low-level hardware functions and introduced us to round-robin scheduling for task management. The primary activities involved programming an Arduino Mega board to drive an 8x8 LED matrix and a small speaker, integrating hardware programming with real-time audio and visual outputs.

We started by directly manipulating digital I/O pins to control LEDs, moving away from high-level functions like pinMode() and digitalWrite() to manually configure hardware registers. This direct control extended to generating precise audio tones using a 16-bit Timer/Counter, requiring us to delve deep into the microcontroller’s hardware capabilities. In addition to these individual tasks, the lab challenged us to synchronize these outputs—flashing LEDs and sound generation—demonstrating the complexity of managing concurrent processes in embedded systems.

By the end of this lab, we not only gained hands-on experience with basic electronic components and microcontrollers but also developed a foundational understanding of embedded system programming. This included tasks like configuring and controlling an LED matrix based on thumbstick movements, creating a user-interactive display, and synthesizing music with programmed audio tones, all orchestrated to operate seamlessly together. Through these exercises, we acquired practical skills and theoretical knowledge, preparing us for more advanced topics in digital and embedded systems.

**Experimental Results**

In this lab, we employed a variety of techniques and tools to meet the outlined learning objectives, engaging in a detailed approach across different segments of the lab:

1. Hardware Bit Manipulation: We began by setting up three LEDs with resistors ranging from 250 to 500 Ohms, connected to pins 47, 48, and 49 of the Arduino Mega. Instead of using the Arduino's built-in pinMode() and digitalWrite() functions for initialization and control, we manually configured the data direction register (DDR) and port data register (PORT). This allowed us to directly manipulate the I/O pins. We developed a function to toggle these LEDs sequentially, which involved setting and clearing bits in the PORT register to achieve the desired flashing pattern.
2. 16-Bit Timer/Counter for Tone Generation: The second part of the lab focused on using the 16-bit Timer/Counter of the Arduino to generate audio tones through a small speaker. We thoroughly reviewed the relevant sections of the ATMEGA2560 datasheet to understand the timer/counter configuration. Our task was to program the timer to generate a square wave at specified frequencies (400 Hz, 250 Hz, and 800 Hz), which we achieved by setting the appropriate waveform generation mode and toggling the compare match output mode bit. We ensured the output pin, OC4A, was set as an output through the DDR register and connected the pin to the speaker, carefully managing the duty cycle for a cleaner sound output.
3. Coordination of Concurrent Tasks: For the concurrent execution of tasks, we implemented a basic round-robin scheduler within the Arduino's loop() function. This setup allowed us to manage and sequence the operation of the LED flashing and tone generation tasks. We used global flags to control the start and stop of these tasks as per the scheduling requirements. This method demonstrated how multiple tasks could be managed and synchronized without external libraries, relying solely on careful timing and task management.
4. Interactive Display with Thumbstick-Controlled LED Matrix: The final part of the lab involved an interactive 8x8 LED matrix display. We interfaced an XY thumbstick to the Arduino and mapped its analog output to the matrix coordinates to control the movement of a dot across the display. The thumbstick’s variable voltage output was read through Arduino's analogRead() function, and we wrote additional code to convert these readings into matrix coordinates. This interactive setup required integrating the display control with real-time user input, showcasing an application of embedded systems in user interface design.

**Experimental Results**

1. Hardware Bit Manipulation

For this section of the lab, we were able to successfully demonstrate the ability to control LEDs using direct hardware manipulation, without relying on Arduino's built-in functions. We did this by directly setting and clearing bits in the DDR and PORT registers and controlling the blinking of LEDs on pins 47, 48, and 49. Each LED was programmed to light up sequentially for 333 milliseconds before switching to the next, in a continuous loop.

Test Results: The LEDs flashed in the correct order and with the precise timing required by the lab specifications. This was first tested using an oscilloscope to verify the timing accuracy. The LEDs lit up and turned off precisely at 333ms intervals, confirming that our manipulation of the hardware registers was accurate.

Setbacks: The main setback that we had was finding the virtual pin on the diagram. This was tough because it was difficult initially to map the diagram from the picture to our existing Arduino. The way we overcame this was by going through the diagram slowly, and mapping everything out together

2. 16-Bit Timer/Counter for Tone Generation

For this part of the lab, we utilized the 16-bit Timer/Counter to generate specific audio tones through a speaker. We programmed TimerCounter4 to output square waves at frequencies of 400 Hz, 250 Hz, and 800 Hz sequentially, each for one second, followed by one second of silence.

Test Results: The square waves generated matched the desired frequencies, as verified using an oscilloscope. The audio output from the speaker was clear, and each frequency change was distinct and as scheduled. The addition of a series resistor between the output pin and the speaker, as suggested, helped in reducing the harshness and distortion of the sound, thereby improving audio quality.

Setbacks: Our main setback with this task was ensuring that the square waves we were outputting were the correct frequencies. We got this to work by going through our code, and rethinking our delays, which allowed us to catch our initial mistakes.

3. Coordination of Concurrent Tasks

For this lab, we used a round robin scheduling approach. This approach allowed us to effectively manage multiple tasks, such as LED blinking and tone generation, without overlapping or interference. We implemented a control system using global flags that enabled switching between tasks according to the lab's schedule requirements.

Test Results: The tasks were executed seamlessly. The LEDs and the speaker functioned in the designated order and timing, demonstrating our ability to synchronize and coordinate concurrent operations effectively. The LEDs sequence and the tone generation tasks were successfully integrated, with transitions between tasks occurring smoothly without any glitches. As well, we needed to use the given array of notes to play the mary had a little lamb tune.

Setbacks: The biggest setback we had for this one was getting Task A and Task B working together. We used a series of flags and if statements, and experimented with different methods of attaining concurrent frequencies. As well, it was a setback finding that the period necessary for the mary had a little lamb task was different than the original sound. However, it just took a bit of testing to figure out what the prescaler that was used for the array/notes given was, and adjust them accordingly.

4. Interactive Display with Thumbstick-Controlled LED Matrix

The 8x8 LED matrix was controlled via a thumbstick, which moved a dot across the display based on the user's input. We mapped the thumbstick's analog output to the matrix coordinates to update the display in real-time.

Test Results: The dot moved fluidly across the matrix, responding accurately to the thumbstick movements. The response was immediate, showcasing the effective integration of the input device with the display output.

Setbacks: There was a huge setback initially, which was we had faulty hardware. With some testing using print statements, we found that our joystick was only able to reach as far was the value of 1, not 0. This meant we could not reach the final row of LEDs. Once we were able to switch this out for better hardware, it became much easier.

**Code Documentation**

1. Lab2-1.4.ino

Initialization Process: Lines 3-7: Configures pins 4, 5, and 7 of port L (DDRL) as outputs to drive LEDs.

Tasks: Lines 9-18: Executes a loop where each LED connected to port L is toggled sequentially with a 333ms delay between each toggle, using bitwise operations on PORTL.

1. Lab2-1.2.ino

Initialization Process: Lines 8-12: Sets up pins 47, 48, and 49 as outputs to control three different LEDs.

Tasks: Lines 14-22: In the main loop, each LED is turned on for 333ms and then turned off, creating a blinking effect across the three LEDs in sequence

1. Lab2\_2\_4

Initialization Process: Lines 8-16: Sets the port 4 bits to output mode and clears both bit A and B then puts port 4 into CTC mode with a prescaler of 1.

Tasks: Lines 14-22: In the main loop, it changes the OCR4A to various frequencies and a delay of 1000 ms to get the desired sound.

1. Lab\_2\_3\_1.ino

Initialization Process: Lines 8-20: Sets the port 4 bits to output mode and clears both bit A and B then puts port 4 into CTC mode with a prescaler of 1 just like part 2, and also sets up pin 47, 48, 49 for output.

Tasks: Lines 22-24: In the main loop, it calls function C, which calls function A and B, then delays. Lines 32-50: This is task A, which sets pin 47, 48 , 49 to blink with a delay of 333 ms as long as it is between 0 and 2 second time window. Lines 52-72: This is task B, which sets the OCR4A to various frequencies and a delay of 1000 ms to get the desired sound as long as it is within 0 to 4000 ms range.

1. Lab\_2\_3\_2.ino

Initialization Process: Lines 1-22: Sets the port 4 bits to output mode and clears both bit A and B then puts port 4 into CTC mode with a prescaler of 1 just like part 2, and also sets up pin 47, 48, 49 for output.

Tasks: Lines 23-25: In the main loop, it calls function C, which calls function A and B, then delays. Lines 61-79: This is task A, which sets pin 47, 48 , 49 to blink with a delay of 333 ms as long as it is between 0 and 2000 millisecond time window. Lines 81-100: This is task B, which sets the OCR4A to various frequencies and a delay of 1000 ms to get the desired sound as long as it is within 0 to 4000 ms window, which is how long the entire thing is. From lines 34- 60 is our concurrent task A and B. Because the period of each individual note of the sound was 1 second, we had it so the blinking lights had a delay of 333 ms like before, and once all of them had blinked (a period of 1000) we played one of the four notes. Every iteration of the loop would play the next note, and the fourth one would play nothing, for there to be a delay.

1. Lab\_2\_3\_3.ino

Initialization Process: Lines 1-22: Sets the port 4 bits to output mode and clears both bit A and B then puts port 4 into CTC mode with a prescaler of 1 just like part 2, and also sets up pin 47, 48, 49 for output. We also initialized the melody array, which contained the proper notes to play mary had a little lamb.

Tasks: Lines 23-25: In the main loop, it calls function C, which calls function A and B, then delays. Lines 61-79: This is task A, which sets pin 47, 48 , 49 to blink with a delay of 333 ms as long as it is between 0 and 2000 millisecond time window. Lines 106-122: This is task B, which sets the OCR4A to various frequencies equal to the melody notes, this runs for the entire song, which is 9 seconds. From lines 48- 80 is our concurrent task A and B. Because the period of each individual note of the sound in mary had a little lamb is 166 ms, one light blinking should happen every two notes. Thus, we had the delay set to 166 and on every second note, we changed the LED to blink on to the next one. Every second iteration of the loop would light the next LED, and every loop would play a new note. This would loop until it hits 10 seconds.

1. Lab\_2\_Part\_4.ino

Initialization Process: Lines 1-69: Sets up multiple digital output pins and configures Timer 4 for specific waveform generation to control sound frequencies. Sets the port 4 bits to output mode and clears both bit A and B then puts port 4 into CTC mode with a prescaler of 1 just like part 2. Also, it sets pin 12, 11, and 10 to output mode, and sets the CS pin to high. Lastly, it initializes the spiTransfer.

Tasks: Lines 74-83: records the analog read values for both x and y, and maps them onto the LED board, delays for 50 ms, and then turns the previously on LED off. Lines 96- 106: This sets the OCR4A to the various frequencies necessary to play Mary had a little lamb. This is done every 3 loops, as each note plays for 166, which is roughly 3 \* 50 ms, the delay used for the LED board. Lines 108- 126: Unused reference function for the mary had a little lamb melody. Lines 130- 145: is the SpiTransfer method, which shifts on port 12 and 11, and writes CS to high and low accordingly.

**Overall Performance Summary**

**Successes:**

Hardware Bit Manipulation:

We adeptly controlled the LEDs via direct manipulation of the DDR and PORT registers, ensuring each LED lit up sequentially for precisely 333 milliseconds. The use of an oscilloscope verified our timing accuracy, confirming our mastery over low-level hardware control.

16-Bit Timer/Counter for Tone Generation:

Our implementation of the 16-bit Timer/Counter successfully generated clear and distinct audio tones at the specified frequencies. Adjustments with a series resistor refined the sound quality, demonstrating our ability to fine-tune hardware outputs for improved results.

Coordination of Concurrent Tasks:

The round-robin scheduling effectively managed multiple concurrent tasks, maintaining synchronization without overlaps or errors. This showcased our ability to handle complex task coordination within an embedded system.

Interactive Display with Thumbstick-Controlled LED Matrix:

The real-time response of the LED matrix to thumbstick movements was seamless, illustrating our proficiency in integrating user input with visual output, an essential skill in user interface design.

**Challenges:**

A notable challenge was initially mapping the virtual pins on the Arduino diagram, which was crucial for the accurate setup of our hardware bit manipulation task. This was overcome by methodically analyzing the diagram and correlating it with our physical setup.

For the tone generation task, correctly setting the frequencies required careful recalibration of our timing logic within the code, underscoring the importance of precision in programming embedded systems.

Integrating Tasks A (LED control) and B (tone generation) posed initial difficulties in ensuring smooth transitions and simultaneous operations. Adjusting our control flags and refining the task scheduler were critical steps that helped resolve these issues.

SeThe primary setback in the interactive display was calibrating the sensitivity of the thumbstick to achieve smooth and accurate movement of the dot across the matrix. We addressed this by fine-tuning the analog-to-digital conversion process and optimizing the refresh rate of the matrix display.

**Teamwork Breakdown**

Teamwork Breakdown

The team for this lab consists of Keerthi and Rhys. We collaborated effectively to tackle the various components of Lab 2, dividing responsibilities for the lab. Rhys focused on the majority of the coding, debugging, and architecture. Rhys was involved in writing all the initial drafts of the code, implementing the logic for the manipulation of the DDR and PORT registers, the timer/counter for audio output, and integrating the round robin scheduler for the task managements. Rhys also helped monumentally with the architectural design, and helped with debugging as well. Keerthi was mainly involved with the hardware components, including the Arduino, the LED matrix, the speaker, and the thumbstick. In addition, she also helped with the architecture of the project, and helped Rhys with the architectural planning of the lab, and helped design the overall system configuration and workflow. She also helped with debugging.

**Discussion and Conclusions:**

This lab presented significant challenges, particularly in the direct manipulation of hardware registers and the synchronization of concurrent tasks using a round-robin scheduling method. Manipulating the DDR and PORT registers without the aid of Arduino libraries required a deep understanding of the microcontroller’s hardware, emphasizing the necessity of precise coding and configuration to ensure correct functionality.

The coordination of multiple processes, such as LED flashing and tone generation, without overlapping or interference, demanded a robust control system. Implementing global flags and meticulous timing was crucial for maintaining the seamless operation of each component, illustrating the complexities involved in managing concurrent tasks within embedded systems.

A notable aspect of the lab was the integration of real-time user input with the LED matrix. Achieving smooth and responsive movement of the dot across the display demonstrated effective real-time processing and hardware-software integration. This segment underscored the practical challenges of creating interactive user interfaces in embedded systems.

Overall, the lab sharpened our skills in embedded system programming, particularly in areas like debugging, optimizing system performance, and hardware-software integration. It highlighted the critical aspects of embedded system design, such as resource management and process synchronization, which are essential for the successful implementation of complex systems.

**Notes:**

* **Page numbers must be turned ON in the lower right corner of each page.**
* **Team members’ last names must appear in the header on each page.**