Caroline Hart

CSE 321 - Fall 2021

Project 3

Audio Detection Device

Table of Contents

Introduction

This project is an all-in-one audio feedback device. To account for disability, this device will be designed using universal design principles. Universal design principles are based on the idea that a device or service should be accessible to any operator. Oftentimes, devices and systems are not developed using principles of universal design. This means that there is a distinct amount of devices that a certain number of disabled people would not be able to fully operate. In many cases, devices operate on the notion that abled people are the norm, and designing for disabilities is considered outside the scope of the device's functionality [1]. According to Rosmaita, the best way to design for accessibility is to incorporate it at each and every step in the design process [1]. However, there is a distinct lack of mainstream accessible designs. This device's purpose is to bridge the gap between inaccessible design and disabled people by providing visual and vibration feedback when audio is detected.

Project Requirements

- Watchdog timer configured appropriately
- Synchronization Technique
- At least 1 direct bitwise driver configuration
- Critical Section protection for entire implementation
    - Make sure two threads don't try to access/modify the same memory at the same time
- A task/thread intentionally incorporated
- At least 1 interrupt

Specifications and Features

Watchdog

The Watchdog timer's purpose is to safeguard against the system getting stuck. It allows the system to reset itself into a safe state and reinitialize things. For this project, the watchdog timer was configured with the maximum allowed

time by the system [4]. This time was chosen because the system must wait for an audio signal before it can operate. Because the device's purpose when audio is not detected is to simply wait for an audio signal, it would be undesirable to have the system reset within a short period of time. Any time an audio signal is detected, the watchdog is once again reset to the maximum allowed time.

Synchronization

Common synchronization techniques include flags, semaphores, mutexes, messages, and mailboxes. Without the use of proper synchronization techniques, data that is either being stored or being transmitted may be corrupted. Therefore, it is important to ensure that events are synchronized. This project uses a semaphore to synchronize its tasks. The tasks consist of system initialization and the audio response. Because these tasks access the same data, it is critical to ensure that these data are only being accessed by one task at any given time. In this project, whenever shared data needs to be accessed, the semaphore is acquired. When the task is completed accessing the shared data, the semaphore is released so that the next task has the semaphore available to it. This ensures that the data will be preserved between tasks.

Bitwise Drivers

Bitwise drivers are used to control the function of input and output devices. This project contains bitwise drivers for the audio transducer, as well as the vibration motor. This allows the program to call driver functions to control the input/output devices, rather than making the necessary bitwise configurations each time the state of an input/output device needs to be altered.

Critical Section Protection

The critical section is an important part of the code to protect. For this project, the critical section was protected by removing access to global variables, and by implementing mutual exclusion. In order to achieve mutual exclusion, a semaphore was utilized.

Initially when beginning this project, a mutex was to be used to protect the critical section. However, after reading the Mbed OS documentation , it is evident

that the Mutex::lock() and Mutex::unlock() functions cannot be called from an ISR context [2]. Due to the nature of the project, having the audio transducer trigger an interrupt is an ideal solution, therefore, more research into critical section protection methods was conducted.

A semaphore was used to protect the critical section, because the Semaphore::try_acquire() method can be called from an ISR context [3]. The use of a semaphore in this project was significant because it prevents the audio interrupt from interrupting the LED and vibration motor response in the case that they are already responding to an audio signal.

Thread

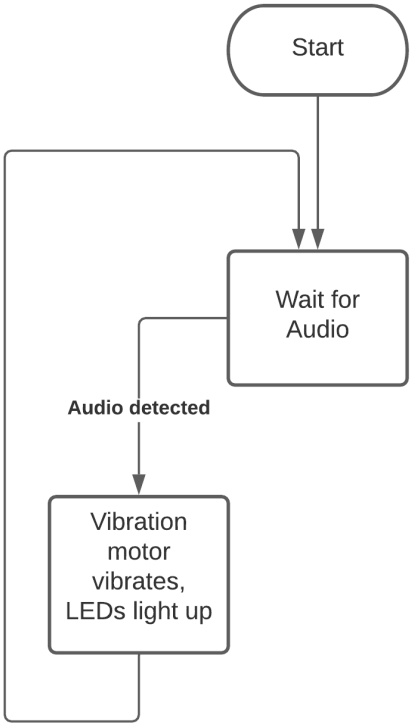
Threads are used to execute a sequence of instructions which are independent of other tasks or threads. In this project, a thread was used to control the response when audio is detected. The response thread is started when the audio interrupt is triggered, and handles the output response (LEDs and vibration motor).

Interrupt

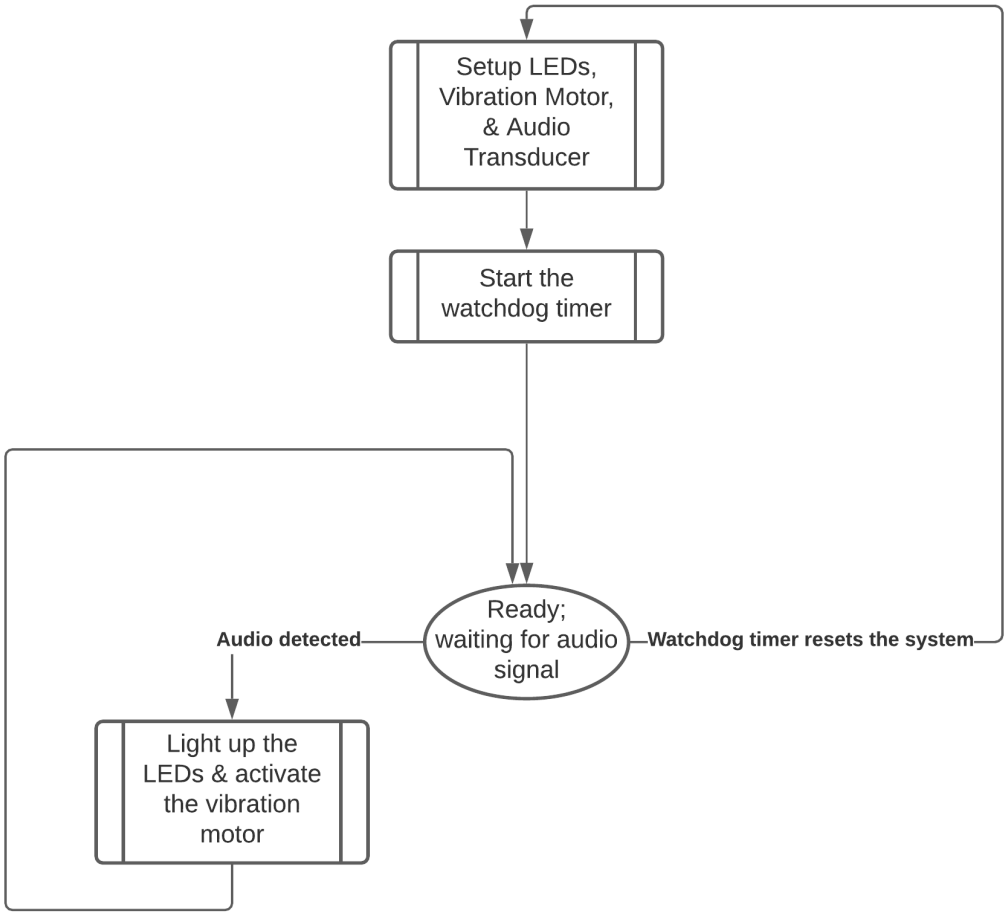
Interrupts are hardware triggered events used to interrupt the normal program flow. In this project, an interrupt was used to jump to the response thread when audio is detected. The use of an interrupt was influential in this design process, because when synchronization techniques were considered, there was a possibility that this interrupt may be removed. However, this did not happen, because using an interrupt to respond to audio was preferred over using a Mutex for synchronization and critical section protection.

Solution Development

Block Diagram



Diagram



Test Plan Instructions

The testing for this project was implemented throughout the entire course of the design process. Initially, the only configurations made were that of the input devices and output devices. At this point in the design process, only the bitwise drivers had been configured. Modular design is important to the testing process because it allows for the observation of how the system behaves at every step in the implementation process. After the input and output devices were tested for functionality, an interrupt was configured to trigger when audio is detected by the audio transducer. Next, the watchdog timer was implemented and tested. Then, in order to protect the critical section, a mutex was attempted

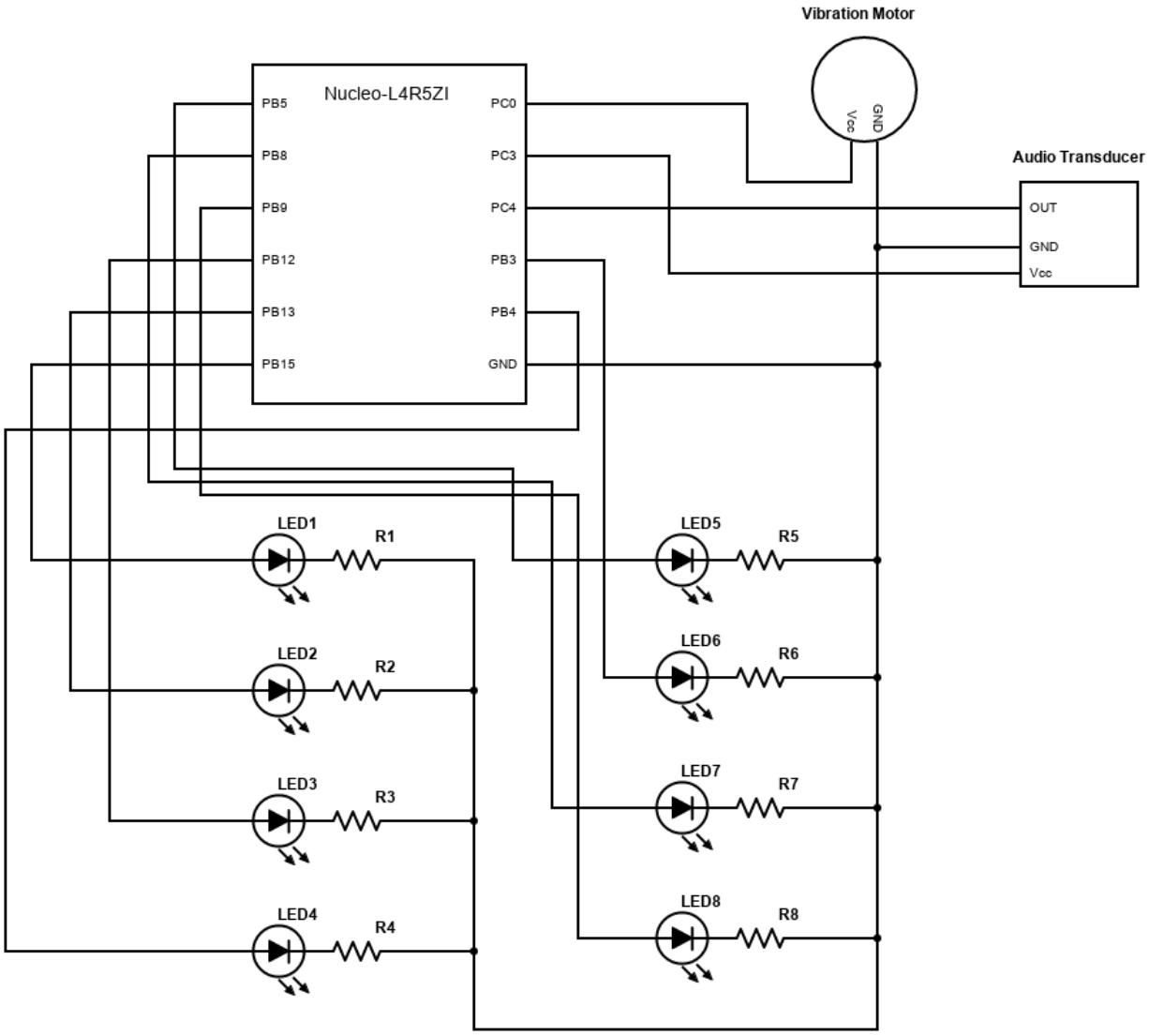
to be implemented. The modular design process was beneficial at this step because it was immediately clear that a mutex cannot be locked or unlocked from an interrupt. After conducting further research, a semaphore was implemented to account for synchronization and protection of the critical section.

BOM

- Nucleo-L4R5ZI
- Solderless Breadboard
- Jumper Wires (Male-Male)
- Jumper Wires (Male-Female)
- 8x LED
- 8x 100Ω Resistors
- Audio Transducer
- Vibration Motor
- Small Phillips Head Screwdriver (optional)

User Instructions

Schematic Diagram



Instructions to Build

1. **Gather materials (see BOM)**
2. **Ensure that the vibration motor is adhered to a surface. This will prevent the jumpers from being freed from the force of the vibration.**
3. **Connect the Audio Transducer**
    3.1. Connect three jumpers to the three pins of the audio transducer.
    3.2. Connect the jumper connected to “OUT” on the audio transducer to pin PC4 on the Nucleo.
    3.3. Connect the jumper connected to “GND” on the audio transducer to either the ground rail of the breadboard, or any GND on the Nucleo.
    3.4. Connect the jumper connected to “VCC” on the audio transducer to pin PC3 on the Nucleo.
4. **Connect the Vibration Motor**
    4.1. Connect the two wires of the vibration motor to two separate rows on the breadboard. This will help prevent the wires from being freed by the force of the vibration.
    4.2. Using a jumper, connect the row of the breadboard where the red wire (Vcc) of the vibration motor was connected to pin PC0 of the Nucleo.
    4.3. Using a jumper, connect the row of the breadboard where the blue wire (GND) of the vibration motor was connected to either the ground rail of the breadboard or any GND on the Nucleo
5. **Connect the LEDs**
    5.1. Place each of the 8 LEDs in its own row on the breadboard.
    5.2. Connect a resistor to the row of the breadboard where the negative terminal of the first LED is connected. Connect the other end of the resistor to the ground rail of the breadboard.
    5.3. Repeat step 5.2 for each of the 7 other LEDs and resistors
    5.4. Using a jumper, connect the positive terminal of LED 1 to pin PB15 of the Nucleo.
    5.5. Using a jumper, connect the positive terminal of LED 2 to pin PB13 of the Nucleo

5.6. Using a jumper, connect the positive terminal of LED 3 to pin PB12 of the Nucleo
5.7. Using a jumper, connect the positive terminal of LED 4 to pin PB4 of the Nucleo
5.8. Using a jumper, connect the positive terminal of LED 5 to pin PB5 of the Nucleo
5.9. Using a jumper, connect the positive terminal of LED 6 to pin PB3 of the Nucleo
5.10. Using a jumper, connect the positive terminal of LED 7 to pin PB8 of the Nucleo
5.11. Using a jumper, connect the positive terminal of LED 8 to pin PB9 of the Nucleo

6. **Insert the micro-USB cable into the USER PWR port of the Nucleo. Insert the other end of the cable into your computer. Ensure the connection between the Nucleo and the computer is complete by checking that the COM light on the Nucleo is now lit.**

Instructions to Use

1. **On your computer, launch Mbed Studio. Then, open the project from cse321-portfolio-cmhart2.**
2. **Ensure that "Project 3" is present in the directory**
3. **Ensure that the following files are present within Project 3**
    - CSE321_LED_Blinky.cpp
    - CSE321_LED_Blinky.h
    - CSE321_Project-3_main.cpp
    - CSE321_Project-3_main.h
    - CSE321_audio_transducer.cpp
    - CSE321_audio_transducer.h
    - CSE321_vibration_motor.cpp
    - CSE321_vibration_motor.h
4. **Ensure that Project 3 is selected as the active program.**
5. **Prepare a clean build of Project 3.**
6. **Run Project 3.**

7. **Wait for the message “ `--- SETUP COMPLETE ---` “ to display on the serial output**
8. **Drop something/knock next to the audio transducer. You may need to use a small phillips head screwdriver to adjust the sensitivity of the audio transducer.**
9. **You should see the LEDs light up as well as feel the vibration motor vibrate for a moment whenever a loud noise is detected.**

Implementation Outcomes

As an outcome of this project, the user will be able to implement the system and use it to interpret audio as both light and vibration. Many mainstream designs use audio to communicate information to the user (ex. a traditional timer emits noise when the time is up). This device intends to make these mainstream devices more accessible. In the case where the user is deaf, hard of hearing, or has difficulties with auditory processing, this device allows the user to use and benefit from mainstream devices that may otherwise be unusable to them.

Considerations for the Future

Shortfalls

Initially, I believed that the audio transducer was able to distinguish between different volumes. After testing the input and output devices, it became clear that the audio transducer is only capable of detecting audio above a threshold. This threshold is adjustable via an on-chip potentiometer, however this is not something that can be altered by the software. It was at this point that I discarded my original idea to implement a device which can distinguish between different volumes to produce different outputs. Instead, I designed a device that can detect audio past a reasonable threshold, which would produce only one response as a result.

General Improvement

One way to improve the system is to include multiple audio transducers. If each audio transducer was set to a different minimum threshold, then several different responses could theoretically be implemented. Special care would need to be taken to ensure that when the highest threshold for noise is reached (when all audio transducers detect audio) the interrupts do not conflict with each other, and only the intended response is run. This could be implemented by giving the events that trigger when the audio is loudest the highest priority, and giving the events that trigger when the audio is quietest the lowest priority. In addition, if several audio transducers were to be used, the responses could vary based on the level of audio detected. For example, if a very loud noise is present in the system's environment, the LEDs could cycle faster, and the vibration motor could be run at a high frequency/shorter period. If a quiet noise is present, the LEDs could cycle more slowly, and the vibration motor could be run at a lower frequency/longer period.

Development Timeline

November 19, 2021

- Created README
- Created initial file structure

November 22, 2021

- Designed driver for vibration motor
- Wrote driver for vibration motor
- Designed driver for audio transducer
- Designed LED response

December 2, 2021

- Wrote driver for audio transducer
- Wrote LED driver functions
- Tested I/O for functionality
- Resolved I/O issues
- Implemented audio triggered interrupt

December 7, 2021

- Added a Mutex
- Added a Watchdog timer
- Tested device functionality
- Resolved issues with the Watchdog
- Removed the Mutex; Added a Semaphore

December 8, 2021

- Tested device functionality
- Further resolved issues with the Watchdog
- Tested device functionality

December 10, 2021

- Updated README

References

[1] B.J. Rosmaita. "Accessibility Now! Teaching Accessible Computing at the Introductory Level," Assets '06: Proc. of the 8th Int. ACM SIGACCESS Conf. on Computers and Accessibility. Portland, OR, USA. Oct. 23-25, 2006. [Online] Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.563.5479&rep=rep1&type=pdf.

[2] Mbed OS. "Mutex Class," *os.mbed.com.* Accessed: Dec. 1, 2021. [Online]. Available: https://os.mbed.com/docs/mbed-os/v6.15/mbed-os-api-doxy/group__rtos__mutex.html#gaa81aed607133209dade63a226818224d.

[3] Mbed OS. "Semaphore," *os.mbed.com* Accessed: Dec. 3, 2021. [Online]. Available: https://os.mbed.com/docs/mbed-os/v6.15/apis/semaphore.html.

[4] Mbed OS. "Watchdog," *os.mbed.com.* Accessed: Dec. 1, 2021. [Online]. Available: https://os.mbed.com/docs/mbed-os/v6.15/apis/watchdog.html.

[5] STMicroelectronics. *RM0432 Reference manual.* (2021). [Online] Available: https://www.st.com/resource/en/reference_manual/dm00310109-stm32l4-series-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.