ECSE 444 – Final Project Report

Camilo Berdugo Salcedo
camilo.berdugosalcedo@mail.mcgill.ca
261116161

Teddy El-Husseini
teddy.el-husseini@mail.mcgill.ca
261036957

Michel El-Hallal
michel.el-hallal@mail.mcgill.ca
261104859

Rafael Reis
rafael.reis@mail.mcgill.ca
261037134

I. Introduction

Modern embedded systems increasingly combine real-time sensing, user interaction, and multimedia feedback, making them ideal for exploring timing-critical applications. In this project, we want to design and implement a reflex-based reaction game on the STM32L4S5I board that challenges users to respond quickly and accurately to a series of audio and visual cues. Our system will integrate multiple peripherals: LED, push-button, DAC speaker, DFSDM microphone, I2C accelerometer, QSPI Flash memory, and UART. These peripherals will be used to create a functional and interactive game. To optimize performance, we will be using a FreeRTOS-based multi-threaded design.

II. Design Problem

The main challenge of this project is to build a reflex-based game that feels responsive, accurate, and fun to interact with. The system must generate four types of cues: three audio tones and one LED signal. The system should also detect four different user reactions: tilting the board left and right, making a sound, or pressing the button. Each cue has a short reaction window, so the system needs to read the accelerometer, microphone, and button inputs reliably. The synchronization of these components (audio playback, sensor monitoring, scoring, UART messages, and overall game flow) can quickly become tricky to manage. That's why we will be using FreeRTOS to keep a clean and low latency code. This will be done by handling cue generation, input monitoring, scoring, and game logic all in parallel. The design problem is therefore to structure all these hardware and software components into a coordinated real-time system that delivers precise timing, accurate reaction detection, multi-modal inputs, deterministic real-time behavior, and a smooth gameplay experience.

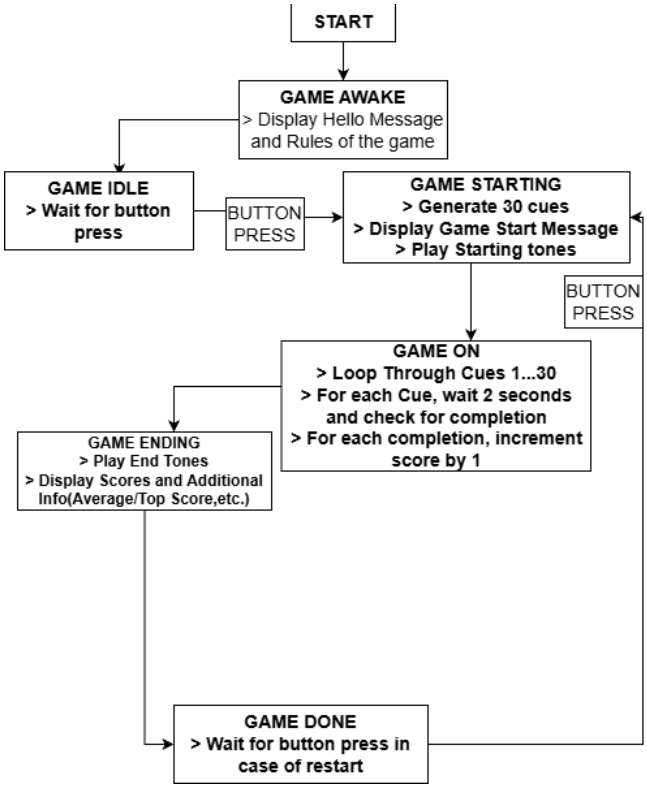
III. System Overview (Proposed Solution)

A. High-level description

Our embedded application will:

1. Display game rules via UART.
2. Wait for a button press to start the game.
3. Play a 3-tone countdown indicating the start of the game.
4. Present **30 cues** at fixed intervals (3 distinct audio cues + 1 LED cue).
5. Capture user responses to those cues using button press, microphone loudness, and board tilt.
6. Score PASS/MISS for each cue.
7. Print results in UART.
8. Update the high score (if needed) and average score in QSPI flash.
9. Print high score, average score and games played from QSPI flash.
10. Play a 3-tone countdown indicating the end of the game.
11. Check for a new button press to restart the game.

B. State machine diagram



IV. System Architecture and Components

A. Hardware Components

- **Button (GPIO / EXTI):** Used to start/restart the game and to respond when the LED cue appears. Configured with an interrupt for immediate reaction detection.
- **LED (GPIO):** Provides visual cues in-game.

- **DAC + TIM2 + DMA:** Drives the speaker to generate clean sine-wave audio tones for the three sound cues. TIM2 provides stable timing, and DMA enables low-overhead continuous playback.
- **DFSDM Microphone:** Captures sound so the system can detect when the user makes a loud noise.
- **I2C Accelerometer:** Measures board orientation to detect tilt-left and tilt-right responses.
- **UART1:** Outputs game rules, cue feedback ("PASS"/"FAIL"), the final score at the end of the session, and the data gathered in the memory.
- **QSPI flash memory:** Used to keep track of data like high score, average score, and games played so we can display it on game-end.

B. Software Components

- **Finite State Machine (FSM):**
  Controls the overall game flow through States: `AWAKE`, `IDLE`, `STARTING`, `ON`, `ENDING`, `DONE`.
- **Cue Scheduler:**
  Randomly generates the sequence of 30 cues and manages the timing of each one, including the 2-second response window.
- **Response Detection:**
  Maps each cue to its required user action:
  - LED cue → button press
  - Sound cue → mic amplitude threshold
  - Tilt cue → accelerometer orientation
- **Audio Generation Module:**
  Produces sine-wave tones using a precomputed lookup table and circular DMA transfers triggered by TIM2.
- **Timing and Debouncing:**
  The button is debounced inside the EXTI interrupt, and system timing (cue delays and reaction windows) is handled using HAL_GetTick().

V. Design Approach

A. Parameter Selection

Several parameters were chosen through experimentation to ensure the game feels both responsive and playable. The cue delay was set to 2000ms, which provides good difficulty and usability. Other delays could be used to increase or decrease difficulty, but we found that 2000ms had a good balance: it took on average 1 second for each of our team members to process a cue and then took between 0.2 seconds and 1 to answer with a movement.

The three audio cues of frequencies were picked so that they are clearly distinguishable through the small speaker. After some testing we found the three frequencies we wanted to use (600, 800 and 1000 Hz). The lowest pitch tells the user to tilt the board right left while the highest pitch tells the user to tilt right. The middle sound corresponds to the "make a sound" cue.

The microphone was set using amplitude measurements similar to those used in Lab 3. To set the threshold for the microphone, we looked at the raw values of amplitude and chose one that clearly shows that the mic picked up a sound played not far from the board. We started from a very high threshold of 0.5 million and immediately noticed on the "Live expressions" debug screen that screaming into the board makes us barely reach 150,000 on the threshold scale. We then dialed the threshold down to 25000 before realizing this was too low and picked up noise from the room. Finally, we settled on our final value of 50000 which will detect someone talking at the board or snapping their finger close to it, but not random noise from the room.

Similarly, the threshold for the accelerometer was done experimentally: we looked at the accelerometer sensor x[0] values in the live expressions debug screen as we titled the board left and right. This allowed us to see that ±500 on the x scale is a good balance between an actual tilt of the board and an accidental one (or staying still).

B. Evaluation of System

To evaluate the system, we have employed unit testing followed by integration testing. In unit testing, we tested each module (i.e. each cue being heard and the ability to score a point from it) individually and determined the parameters that we felt were best suited for each component. The DAC output was checked with an oscilloscope, and button interrupts were confirmed using UART prints. The DFSDM microphone readings were examined with the captured amplitude values, and accelerometer readings were checked by tracking changes in [X Y Z] data during physical movement. After that, we tested QSPI Flash by writing and reading data. To confirm that the write operation was successful, we read right after writing and compared the read and write values, which matched. After each unit test, we did integration testing, which consisted of making sure that when combined with the main system, the sub-systems would not have conflicts nor break the game logic. The full finite state machine was tested step-by-step to confirm proper transition and detection of each cue type. Timing was validated using HAL_GetTick() to ensure consistent cue spacing and proper audio playback.

The FreeRTOS-based architecture helped in integration testing, since each unit test could be its own thread, and there were little dependencies between threads.

VI. Results and Behavior Observed

The system we implemented performed successfully across all major objectives. All hardware peripherals were correctly integrated, and the FreeRTOS-based architecture enabled responsive, low-latency cue presentation and input detection. We measured the input detection latency and got < 1ms for the push-button and between 10-12ms for the microphone and accelerometer inputs, confirming that real-time responsiveness.

The full FSM was executed exactly as intended. Each transition occurred deterministically based on button events and timing conditions. Across repeated game sessions, all 30 cues per game were delivered at the expected 2000 ± 3ms intervals. There were no unexpected jumps or stalls between

states. The system consistently returned to the IDLE state after each game finished, allowing a new game to happen.

The DAC + TIM2 + DMA audio pipeline produced stable sine-wave tones at 600, 800 and 1000 Hz. Oscilloscope measurements showed that the actual real values of the frequencies were approximately 605, 790, and 1008Hz, corresponding to an error of about 1-1.4%.
The tones may not have been the best sounding musically, but they were clearly distinguishable on the speaker. The DMA circular buffer ensured uninterrupted playback. Countdown tones (start and end of game) played correctly each time.

All four user input types were validated through repeated test cycles. Across multiple full-game tests, the system consistently applied the rule that the first detected input determines the outcome for that cue. If the user performs the wrong action for a cue, even briefly, and then immediately corrects themselves, the system still records the response as a "Fail". Once an incorrect input is detected during a cue window, the outcome cannot be reversed.

LED cue – Button Press:
The EXTI interrupt enabled almost instant detection, with a measured latency of 0.3-0.6ms. And debouncing ensured no false triggers. In 30 test cases of falsely pressing the button during a wrong cue resulted in a successful output of “Fail”.

Sound Cue 3 – Microphone amplitude:
The DFSDM microphone reliably detected when a user made a sound that was above the experimentally chosen threshold. Across 20 trials, 18/20 cues we correctly detected, and no false positives were detected during silent intervals. The closer the sound made was to the board/sensor, the better the detection got.

Sound Cues 1&2 – Accelerometer (left and right):
Orientation thresholds of ±500 on the X-axis correctly mapped to left and right tilts. The accelerometer correctly and consistently responded to the physical rotation of the board, meaning it allows the user to score correctly but also fail if the board is tilted the wrong way or tilted during a non-accelerometer related cue. Out of 50 tests, it scored 40/40 correct classifications and 10/10 mis-tilts.

Across trials, the reaction window operated correctly and closed after 2000ms ± 3ms across 50 trials. Late inputs past this window, i.e. when the next cue already started, would make the current cue fail 100% of the time and affect the next cue if the user was not careful.

UART served as the main communication interface and was stable throughout all testing and debugging. Operating at 115200 baud, it successfully displayed: Start message, cue announcements, pass/miss results, final game score summary, high-score, average score and number of games played retrieved from QSPI Flash. Across more than 500 transmitted “data” during testing, no data corruption or missed UART transmissions occurred.

The QSPI memory operations worked correctly for long-term data storage. Values written to flash (high-score, average score and games played) were read back and verified. We tested if data persisted between 20 game sessions, and it did. The “Write” time had an average of 1-2ms, while the “Read” time had an average of 0.3-0.4ms.

The FreeRTOS consists of four Tasks. The first task was Game Logic, which ran periodically every 5ms. It encapsulates the FSM, scoring and accelerometer reading and handling. The second task, Microphone, consisted of reading the microphone values and determining whether or not a sound was played. It would sample the DFSDM values at approximately a 10-12ms interval. The third task was Flash; it read and wrote from and to the memory but only when required and remained inactive otherwise, minimizing CPU usage. Our fourth task was Audio playback. Depending on where we are in the main game logic, this task would (or not) play the sounds. We estimated that the overall CPU load during gameplay remained below 25%. This indicated that the system always had enough processing time available and never came close to overloading the CPU.

VII. Design Alternatives

Throughout development, several design alternatives were tested and considered. Their performance was compared using latency, accuracy, and reliability as measures.

The first design alternative would advance to the next cue immediately after a correct user input was detected, instead of waiting for the whole 2000ms response window. The goal of this design was to make the game more exciting and interactive. But this created a reliability problem: When two tilt-based cues occurred consecutively, the board would register the tilt from the first cue into the second one because there was no delay. And this would cause an unintended FAIL.

A second alternative was to allow users to correct an incorrect input within the same cue window. The system would not lock the outcome after detecting the first input. However, this introduced a potential abuse of fairness and our scoring system. Users could exploit the system by continuously tilting the board left and right, spamming the button and making sounds. This made the game trivial, so the system was instead designed to record only the first input detected during a cue window.

A third design alternative we considered was using multiple sensors for certain cues. Each action would need to be answered by using two or more sensors. For example, a tilt cue would require both an accelerometer for tilt reading and a push-button press, while a microphone cue might require detecting both a sufficient amplitude and a rotation of the board. This method was ultimately rejected because it made user interactions confusing and inconsistent. It also increased false negatives, as users could not perform both required actions simultaneously and with the exact requirements for reliable detection.

Finally, we also considered changing the microphone queue to be silent. Therefore, when there was no sound output and no LED light, the user was supposed to speak into the microphone. However, this queue alternative was confusing for the majority of users, because they would confuse it with the LED queue. Each one of us tested the game with this alternative, and the results for the microphone queue were as follows: [3/8, 2/7, 1/6, 2/9]. We can observe that most of the microphone queues failed. Since it is a quick reaction time game, and having two queues that have no sound in them led to confusion among many users, we decided to remove this feature from our game.

VIII. Recommendations

After finishing this project, there are several things that we would have done differently to be more efficient.

We began by writing our code directly in the main.c file, specifically inside the loop section. After the whole system was implemented, we transitioned the code into threads. If we had to start over the project, we would recommend writing the code directly into threads.

Additionally, if we had more time to complete the project, we would suggest using sounds like a human voice “Left” and “Right” for the tilt queues, instead of using different frequency notes. These notes can be hard to differentiate for the user in the first few games, but with some practice, it is easy to remember them. So, it would be a recommendation to help new users adapt to the game.

IX. Teamwork

All team members contributed equally throughout the project. We collaborated closely on design decisions, debugging, parameter tuning, and implementation, ensuring that every component was developed and tested together.

X. Acknowledgements

We would like to thank our TAs for their exceptional support throughout the labs and project. Their guidance, troubleshooting help, and clear explanations were essential in helping us complete this project successfully.

XI. Conclusion

This project successfully implemented a real-time, reflex-based reaction game on the STM32L4S5I platform using FreeRTOS and multiple onboard peripherals. All cues were generated reliably, user inputs were detected with low latency, and game performance remained consistent across testing. The system met all functional objectives, demonstrated stable real-time behavior, and provided an engaging interaction experience.