Contactless Morse Code Education Device

Chul Jin Kwag*, Yazdan Zinati *, Himel Saha*, David Deng*

Team 9

*Department of Electrical and Computer Engineering, McGill University, Montreal, Qc, Canada

Chul.kwag@mail.mcgill.ca

Yazdan.zinati@mail.mcgill.ca

himel.saha@mail.mcgill.ca

zhiyun.deng@mail.mcgill.ca

I. Design Problem

Morse code is an important communication method requiring little bandwidth and low complexity. Indeed, the famous SOS is still in use to signal distress worldwide. To introduce the code to new users and allow people to practice without touching a telegraph key (the input switch of a telegraph device), we seek to implement a morse code education device. The ideal device would feature a contactless telegraph key and enable bidirectional, full-duplex conversion from text to morse code and from morse code to text. The contactless acquisition of user inputs is critical during pandemic times and for accessibility purposes. Furthermore, digitally implementing this previously analog functionality allows us to introduce a vast array of features and extra functionality. These functionalities include using the speaker to translate morse code to text and using the LED to show the received text in morse code. They make this product more accessible to the visually or hearing impaired.

II. Final Design

The project is a morse code education system, consisting of a contact-less input device (the board) and a terminal on a computer linked by USB cable.

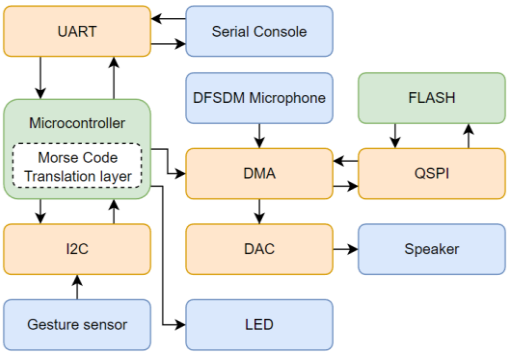

Figure 1: System and components diagram

As can be seen in Figure 1, the system gets a character input from a user via USART, and the character is then read out loud out on the speaker using the corresponding audio segments stored in the Flash via QSPI. At the same time as the audio is being played, the corresponding morse code is flashed on LED2 of the board. The user could then get familiar with the morse code by associating the blinking sequence with pronounced letters. The second part of the project involved creating a contactless telegraph key using an I2C distance sensor. When combined, the two parts of this project enables 2-way conversion between morse code and text. A person can practice *inputting* morse code by waving his hand near the device in Morse code patterns and verifying the text output in the terminal. The features implemented are DAC speaker, I2C sensor (distance sensor), and QSPI flash. We implemented low power operation for the performance enhancement. When the pushbutton the pressed, the board goes into sleep mode (HAL_PWR_EnterSLEEPMode), which stops the core but keeps all peripherals running. Keeping peripherals running reduces the delay seen by the user when waking up the device from sleep. This wake-up is accomplished by a high-priority interrupt triggered by the pushbutton on the EXTI lines and resumes the core operations.

The Morse code translation layer is a function that takes text as input and produces the associated morse code. In addition, it converts motion in front of the distance sensor to text.

The device receives characters from the terminal via UART and processes them using our translation layer. However, our initial design made use of the ISR triggered by the reception of a character through UART. Unfortunately, it was unwieldy to manage the separate task of making the LED blink according to the corresponding morse code. To address this and to enable simultaneous input and output to the terminal, our design required the use of FreeRTOS which resolved this issue. FreeRTOS enables simultaneous two-way conversions using threading. The overall system makes use of two FreeRTOS tasks to get readings from distance sensor (morse code to text) and text from computer to board (text to morse code). This ensures that the two tasks are not blocking each other or the CPU under any condition.

The main implementation of receiving a character through an interrupt, converting it to corresponding morse code and blinking the LED was moved to a task of the operating system. This task is handled independently of the distance-sensor-based morse code creation. Although an interrupt was used to receive individual characters, the task has been implemented in polling mode. The task regularly polls a Boolean flag. When a character is sent from the terminal to the board, an interrupt is triggered. Inside the interrupt callback function, the value of the flag is changed to the one that the task is looking for. Since the task is constantly polling, as the callback function is triggered, it can execute the functions defined inside the condition. The task proceeds to convert the received character to morse code and blink the LED accordingly.

Instead of HAL_Delay() and stopping the execution of all the threads, osDelay() was used to pause the execution of that task. This helped us achieve the blinking effect. In addition, the program looks for the corresponding preprocessed audio data in the QSPI flash for that character received. The data was then passed to the DAC to play the recorded audio of the letter. After one execution is successful, the task alters the flag condition

initially altered by the receiving callback function to prevent the repeated audio playback and LED blinks.

The distance-sensor-based telegraph key works by detecting the presence of a hand in front of the sensor and counting the duration of the event. Its functionality and usability crucially depend on the selection of two parameters. This process is elaborated in section IV below.

III. Components and Rationales

The hardware components of our system include the following: I2C interface, Time-of-Flight and gesture detection sensor (VL53L0X), Quad-SPI NOR Flash memory (N25Q128A13EF840F), DAC, ADC, Speaker, USART, QSPI, DMA controller, LED (LED2), Discovery kit (B-L4S5I-IOT01A), various timers, and DFSDM Microphone.

The Time-of-Flight and gesture detection sensor is considered the primary component of the contactless telegraph key. In general, we limited our design components to those that were readily available on the board. There are various buses and interfaces (i.e. I2C, SPI, USART) available on board to allow communication with associated peripherals. In terms of software components, we have decided to make use of the software packages that we had used for the labs, plus a few more for added functionalities. They help simplify interactions with various peripherals. More precisely, the board support package for the QSPI interface as well as the flash module were leveraged to facilitate operations in memory such as block erase, read and write. We disabled the Flash reset from the BSP Init method for the Flash so as to preserve our audio data in between runs. Of course, there are also the 3 main software components of our solution which are the Morse code translation layer, the software expansion package for the gesture detection sensor, and the BSP for the USART interface. Since there is no BSP for the gesture detection sensor that goes with our existing board, we had to adapt external software (X-CUBE-53L0A1) designed for the X-NUCLEO-53L0A1 expansion board to our use case. Among other things, the adaptation included modifying platform files to include the I2C read address of our board. The APIs provided by external packages allowed more flexible control of the sensors and greatly simplified the implementation of the telegraph key and console input features. The distance sensor software package was also tweaked to accommodate its use in an RTOS, such as by modifying references to HAL_Delay() as necessary. The Morse code translation layer functions by using an in-memory lookup table. The system iteratively goes through every entry in the table to find the character corresponding to the incoming morse code. Conversely, to convert from text to morse code the same process is executed in reverse for each character.

IV. Parameters and Evaluation

There are various parameters that must be chosen for the system to function correctly. Two of them relate to the operation of the distance/gesture sensor. The device regularly reads data from the distance sensor, and if the distance is below a certain threshold (i.e., a hand is above the sensor and the telegraph key is in the "telegraph key pressed" state), it starts a counter that increments on the next reading. When the distance sensor no longer detects the hand, it goes to the "telegraph key released" state. Using the counter value, it can determine the duration of the previous "press" and is able to translate the operation to either a "dot" or a "dash." The distance threshold and the counter (duration) threshold are the two parameters necessary to be chosen.

1) *The distance threshold*: The parameter was chosen with high input speed as a goal. For example, during testing, it was found that moving one's hand horizontally in front of the sensor to operate the telegraph key was unnatural for users. Instead, it is more efficient to have the operator move his hand vertically, in a manner similar to a real telegraph operator. As a result, the distance threshold at which the telegraph key triggers was chosen to be approximately 10cm. At this distance, the user can comfortably hover his hand above the sensor without triggering a "press". To perform a "press", it is sufficient to slightly lower the fingers below the 10cm mark. This allows for the rapid "press" and "release" of the telegraph key.
2) *The duration threshold*: This parameter was chosen to allow novice learners to comfortably input morse code. To choose this parameter, our team members' natural paces of operating the switch were recorded. We asked our team members to operate multiple dots and dashes in sequence. Then, we set the duration threshold to be the average between the average "dot" duration and the average "dash" duration. The parameter selection contributes towards the goal of the device—to educate new morse code users. A device designed for users with more expertise in morse code would have used shorter duration thresholds.

Testing was conducted in stages. The first stage was component testing. First, we tested that the correct morse code is detected by the system for every single gesture input. This verified that the distance sensor noise is at acceptable levels. We went through multiple morse sequences, starting from shorter ones (such as the one-character sequence '.') and gradually moving to longer ones. We iteratively went through every code in the code book to see if any code is especially vulnerable to failure.

As expected, characters with longer Morse code sequences are more prone of misinput. The characters that are most prone to errors during testing are the characters C (-.-.), H(….), and P (.--.). Transitioning between dots and dashes is difficult for new users. Additionally, when inputting characters consisting of multiple dots, users sometimes miscount them. Overall, we observed a decrease in accuracy when the length of Morse code encoding increases.

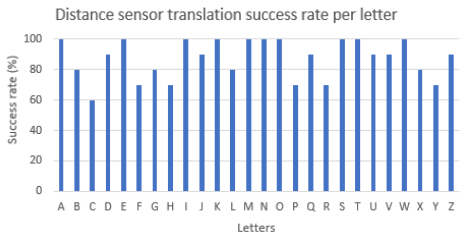

Figure 2: Distance sensor translation success rate per letter

We also verified that the code is transformed to the appropriate character via the transformation method by outputting the translated ASCII characters to the terminal. Our final test involved going through all letters and verifying that each letter is output into speaker correctly and that the letter is audible from approximately 20 cm away. We have also evaluated the use of Flash memory with the DFSDM Microphone. The number of consecutive writes without errors was 10, beyond which the consistency and accuracy fell. We

then proceeded to perform integration tests in which we tested the system as a whole, including accuracy tests of specific sequences using the distance sensor as well as input speed tests which then culminated in qualitative feedback.

V. RESULTS, OBSERVED BEHAVIOUR AND CHALLENGES

All the milestones have been implemented and work as expected independently. The distance sensor is used to produce "." and "-" morse codes based on set thresholds for detection distance and length. This, in conjunction with the morse code translation layer, the desired character can be produced by hand gesture and displayed onto the terminal as seen in Figure 2. Using FreeRTOS, an individual task is created that makes the distance sensor poll for input, without blocking the CPU for other processes, which is then passed onto the translation layer. Given a character or a morse code, the translation layer gives the corresponding morse code or character.

Figure 3: Terminal input to board and output Morse code

During preliminary testing, a team member with about 5 minutes' training was able to type an 11-letter sequence (34 morse code signals) with an accuracy of 81% (8 out of 11 characters correct). We subsequently conducted more thorough testing of the telegraph key. We asked three participants to input a 48-letter sequence using the distance sensor. Those three participants both have more than 10 minutes of training. They are provided with a reference Morse code chart during the input process. They were instructed to perform the input at any comfortable pace. Between the three participants, the input took 4 minutes, averaging 7.5 seconds per character. Participants made an average of 5 errors in this 48-letter sequence (89% accuracy).

The qualitative feedback received involved the inconvenience of having to use a "guide" ruler to know where to place a hand. In addition, participants suggested that having auditory or tactile feedback on whether a press has been registered would be very helpful to their accuracy. All participants agreed that this device is helpful and motivating for learning Morse code. They especially appreciate the contactless nature of the device, as they think it brings a degree of freshness to the text encoding method often seen as old and unfashionable. The communication from the terminal to the board is fault free, giving the correct morse code output for a text 100% of the time. The testing of each way of communication implicitly puts the translation layer to test, which passes the test 100% of the time. For each character input to the terminal, we manually verified that the LED generates the correct blinking sequence.

Finally, we have the speaker functionality implementation which requires DAC, DFSDM (microphone) and QSPI (for storing pre-recorded sounds of letters). All elements in the chain worked as expected with the DFSDM microphone used to record audio of the letters into the flash at specific addresses which were then read via the DMA to the DAC and output on the speaker. The flash memory issue we encountered previously was resolved as expected by writing a single audio segment to the flash at a time and by writing to every second block instead of consecutive blocks. Furthermore, the clarity of the audio recordings was significantly improved by playing the audio of the letters through Google Translate and recording that versus using our voice. Also, we had much greater control over the quality of the recordings by not recording all segments consecutively and rerecording each letter until we were satisfied. We also used the B-L4S5I-IOT01A board which features double the amount of RAM compared to the B-L475E-IOT01A to be able to record longer segments of audio from the DFDSM microphone.

In terms of our performance enhancement which was low-power operations, we achieved a power saving of 12 mA, starting from a baseline of 23 mA power drawn at the JP5 jumper, which measures the IDD current when the program was running with all of its features active, to 11 mA when at sleep. The jumper used to measure the current is indicated in the diagram below of Figure 3. It was found through the reference documentation of the board.

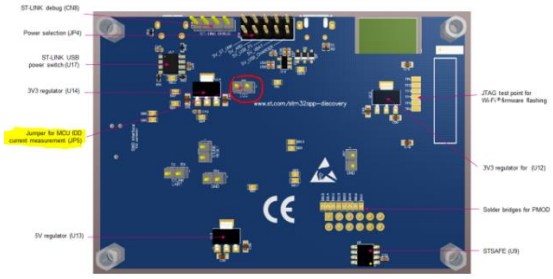

Figure 4: Discovery board test pad and jumper diagram

We measured the current using a Sperry DM-4100A multimeter as pictured in Figure 4, connected serially to the pins of JP5 using alligator clips thus allowing us to measure current draw of the board.

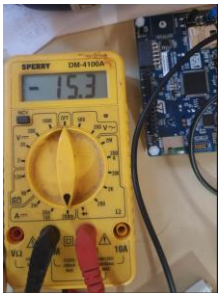

Figure 5: Current measurement setup using JP5 (IDD)

When the board was not in low power mode, the current draw varied between 20 and 23 mA depending on whether the DAC speaker was playing audio or if there was input from the terminal being processed by the core. When the board was placed to sleep using the push-button there was little delay (~0.6 s) for the current to drop and settle to 11.4 mA with some variation (within 0.3 mA) between runs. Furthermore, the wake-from-interrupt using the pushbutton also had very little

delay (~0.1 s) which was important for usability and a reason why the sleep mode was implemented over other alternatives.

VI. Alternatives

Alternatives that were tested were mostly for the power saving optimization. We researched and tested multiple lower power modes for the board. Indeed, the STM32 platform offers multiple ways of conserving energy by putting either the microprocessor or the entire board to low power mode by using different HAL commands. We had elected to use the sleep mode of the board, which stops the Cortex core but leaves the peripherals running, as it did not require a reconfiguration of the timers which was required in other low power modes. The alternatives that we explored were the other 2 low power modes provided through HAL commands: stop mode and standby mode. Stop mode pauses the core as well as putting the timers and other peripherals such as the flash, DAC, I2C and USART to sleep. This specific mode which was enabled using the HAL_PWR_EnterSTOPMode method enabled us to achieve a current consumption of 9.2 milliamps which is a ~2 mA saving relative to the sleep mode and the time taken to get to this current was about the same (~0.6 s) while waking up took longer (~0.7 s). However, we opted to not use this mode since we had to reconfigure the timer after wake-up which added complexity while not providing sufficient benefits. We also had to consider the timing of the wake-up from sleep of the peripherals. While this was handled by the method, we also believed that the tradeoff of additional power conservation versus the availability of the peripherals at core wake-up was worth it. The other mode that was standby (HAL_PWR_EnterSTANDBYMode) which turns off the core and peripherals and wipes the SRAM as well as the registers, turning off the 1.2V domain. This also resulted in additional power saving in which we recorded a current draw of only 0.1 mA which is a ~11 mA saving compared to sleep mode. However, sleep mode was chosen, as the drawbacks outweighed the benefits for our use case. Indeed, the usability of the board dramatically decreases with this solution as the wakeup from standby amounts to a system RESET with the program essentially restarting and thus adding a significant delay between the start of the wakeup using the pushbutton and the actual usability of the morse code functions from either the terminal or the distance sensor. We observed a power settling time of ~0.8 s and a wake-up time of ~1.3 s. The last alternative we explored was the use of the FreeRTOS low power functions which is the tickless idle mode which places the microcontroller in low power when the OS is idle. We determined early on in our research of the mode that this would not be compatible with our implementation, as to see a significant power saving the OS must have large idle times during which the microcontroller would be placed in a low power state. However, since our tasks run with low os delays we would not see the benefits and in fact would have seen increases in power consumption as the microcontroller is set to a low power mode and then exited frequently.

VII. Recommendations

With the short timeline of the project, we had to make a few cost-benefit analyses when it came to designing and implementing features. For instance, the distance I2C sensor is continuously read by polling. The text from terminal to board functionality polls for a flag modification by the USART interrupt. The use of interrupts, in this case, would be more efficient regarding power consumption and processor utilization. We did implement our program as part of the OS, so polling the sensor wouldn't be a blocking operation, but using interrupts will improve our project. One of our features is the low power mode that can be entered and exited by pressing the push buttons on the board. A worthy addition to this feature would be an auto-sleep mode where the board would enter this mode upon detecting a set period of inactivity. Furthermore, we have many ways to improve the minimum-viable product implemented for this class. Firstly, we intend to extend morse code communication to two boards over WiFi or Bluetooth instead of between a computer and a board. A benefit of this new implementation would be wireless communication without needing a personal computer, significantly reducing the cost to our users. Secondly, we can encrypt the morse code sent between the boards with AES or RSA and error detection/correction. For aesthetic purposes, the development boards would come in an enclosure, only exposing the parts that the user interacts with, such as the LED, speaker, distance sensor, and push buttons. Team members had two different development boards and had to work on separate modules simultaneously. Having different boards resulted in issues regarding integration, as project-autogenerated files would be distinct and not applicable from one board model to another. This made continuous integration of code challenging needing manual code conflict resolution. Possible solutions would been to separate tasks such that modules developed on separate boards would less often need to be integrated.

In addition, to incorporate the qualitative feedback given by user participants, adding auditory feedback on a telegraph key press (for example, adding a light sound when a “press” is detected) will improve user experience.

VIII. Conclusion

In this project, we implemented, integrated, and tested modules composing a Contactless Morse Code Education Device. To develop this device, we used a wide range of features and peripherals available, including DAC, ADC, speaker, DFSDM microphone, I2C distance sensor, Quad-SPI NOR Flash, USART, SPI, DMA controller, and LED. To further optimize the product, we used FreeRTOS to allow simultaneous text-to-morse (computer to board) and morse-to-text (board to computer) conversion and implemented a low-power mode. We completed our tasks ahead of both demos and presented a viable product in the initial demonstration. Due to the product mainly being conducted for the initial demo, we received helpful feedback, which we incorporated into our work for the final demo. To validate the device for the demo, we conducted an array of tests, from a unit test of individual components and modules to testing input peripherals such as the distance sensor or the USART input for input speed and error rate. These tests often revealed tradeoffs that led us to continuously calibrate and fine-tune our device precisely. For instance, to find the optimate tradeoff between error rate and input speed for the distance sensor, we followed an iterative process for calibration. We finally asked independent users to test the product and rate various aspects, including ease of use. The product received positive feedback and can be extended in the future, as summarized in the previous section.