Microprocessor Systems Assignment 2

The task is to use a mixture of C code and ARM assembly to build a simple game that will teach a player Morse Code. The player should interact with the game by pressing the **GP21** button on the *MAKER-PI-PICO* board for a short duration to input a Morse "dot" and a longer duration to input a Morse "dash". If no new input is entered for at least 1 second after one-or-more "dots" or "dashes" have been input, then it should be considered a "space" character. If no new input is entered for at least another 1 second after that, then the sequence should be considered compete and passed to the game portion of the code for pattern matching. The game should have a minimum of two levels. The first level for matching individual characters (with the expected Morse pattern provided) and the second for matching individual characters (without the expected Morse pattern provided).

Basic gameplay specifications:

1. When the application starts, the UART console should display a welcome message banner that includes the group number as well as some brief instructions on how to play the game. The player should be presented with a minimum choice of two difficulty levels (chosen by entering the equivalent Morse code pattern for the level number) to start the game and the RGB LED should be set to the colour blue to indicate that a game is not in progress.
2. When the game starts, it should present the player with an alphanumeric character (A-Z/0-9). The player should attempt to enter the equivalent Morse code sequence for the character using the **GP21** button. If the player enters the correct Morse code sequence, then the answer is correct, otherwise it is incorrect.
3. The two basic levels should work as follows:
    1. **Level #1:** Individual characters with their equivalent Morse code provided.
    2. **Level #2:** Individual characters without their equivalent Morse code provided.
    3. **For extra credit:**
       Level #3: Complete words (strings of characters) with their equivalent Morse code provided
       Level #4: Complete words (strings of characters) without their equivalent Morse code provided
4. The sequence that the player inputs should be shown as both a Morse code pattern as well as the alphanumeric character equivalent that it decodes to (if it does decode to a character). If it does not decode to an alphanumeric character, then "?" should be shown instead.
5. When a level is chosen and the game begins, the player should start with three lives and the RGB LED should be green.
    1. Each time the player inputs a sequence that is not accurate to the expected sequence that is provided, a life should be taken away and the RGB LED colour updated to match.

    2. Each time a sequence that is input by the player is exactly accurate to the expected sequence then a life should be added (up to the maximum of three) and the RGB LED colour updated to match.
    3. If the number of lives reaches zero, then the game is over and the RGB LED should indicate red.
6. To progress to the next level, the player should enter 5 fully correct sequences in a row during the current level. If the player is already at the final level, then they have completed the game and the application should display a suitable message.
7. The RP2040 Watchdog Timer should be used to trigger a reset of the Raspberry Pi Pico and return it to the application welcome message banner if the game is in a running state and no input is detected for the maximum timeout of the Watchdog Timer (approximately 9 seconds according to the datasheet).

Instructions:

The application can be primarily written in C; however, the button event detection handler and the Morse code input buffer logic portion should be written entirely in ARM assembly.

CONSOLE

The serial console can be initialised directly using "**stdio_init_all()**" from the "**main()**" C function to the default setting of 8-N-1 at 115200 baud. All messages to the console can be sent using the "**printf()**" function from either the C or the ASM portion of the code.

RGB LED

Using the "pico-apps/examples/ws2812_rgb" project as a starting point, implement suitable functions in C that will wrap around the PIO controller logic for the RGB LED on the *MAKER-PI-PICO* board and allow the colour to be set to at least: "red", "blue", "orange", "green" or "off".

GPIO

Write and install a GPIO interrupt handler that will detect falling-edge (button presses) and rising-edge (button releases) events for the **GP21** push-button on the *MAKER-PI-PICO* board. Every time the button is pressed or released, the handler should store the current system timestamp to a shared memory data-segment each time the button is pressed or released and should calculate the duration for which the button was pressed as well as the interval between button presses (the amount of time when the button was not pressed). The code should be written in ARM assembly apart from the following C functions: "**stdio_init_all()**", "**printf()**", "**gpio_set_irq_enabled()**" and the asm_gpio_*() functions from the previous labs.