Assignment #02

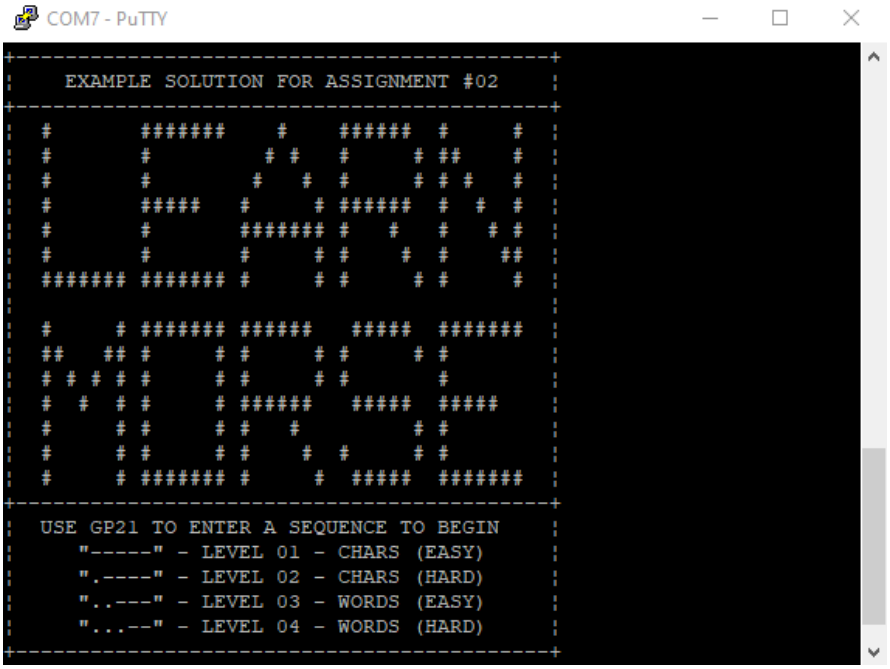
Creating a Game to Teach Morse Code

Assignment Type:	Group
Assignment Duration:	4 weeks
Marks (of course total):	30%
Submission Method:	https://tcd.blackboard.com
Submission Deadline:	23:59 on Friday the 7th of April 2023

Introduction

This group project is designed to be as much about how you work together in a group as it is about the coding assignment that you must complete. While the bulk of the grading scheme will be focussed on meeting the coding requirements set out in the next section, marks will also be awarded for: how well the project was run; good usage of Git as a group; the quality of the final project report and the application demonstration video.

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:
    a. **Level #1:** Individual characters with their equivalent Morse code provided.
    b. **Level #2:** Individual 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.
    a. 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.
    b. 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.
    c. 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).

Optional features for extra marks:

1. Add statistics at the end of each level that is successfully completed or when the player runs out of lives indicating how well they did. For example, the number of correct/incorrect answer and total overall accuracy.
2. Add two additional difficulty levels to the game that allow for inputting and matching individual words and not just individual characters:
    - **Level #3:** Individual words with their equivalent Morse code provided.
    - **Level #4:** Individual words without their equivalent Morse code provided.

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.

Team Roles

You should allocate the following individual project roles amongst yourselves at the start of the project. The individual ownership roles should be noted as part of the project report along with a breakdown of project and coding tasks that the individual team members were responsible for. Some project teams will have 5 members, and some will have 4 – where there is a team of 4, the responsibilities of the "Project code owner" should be split amongst all 4 team members equally and this should be noted in the project report.

Project workflow owner

As the project workflow owner, you are responsible for the overall organisation of how the project should be run. You should work with the other functional owners to ensure the overall project is on target, everyone is in regular communication and the individual project tasks have all been allocated and are on track for completion by the milestone dates. It is recommended you use the GitLab built-in "Issues" management feature where you can collaborate with your project team members to define, set, manage and track overall project milestones and individual tasks in a highly visible way to ensure the overall project and required deliverables get completed in time.

GitLab workflow owner

As the GitLab workflow owner, you are responsible for creating a new shared GitLab project repository for your team, providing access to your project team members, and managing code merge requests to the repository. The other project team members should clone this repository directly so they can work on their allocate code portions individually and make regular pull requests (using Git feature branches) back to the project repository. You should ensure that all pull requests get correctly merged back to this repository. At the end of the project, you should submit an overview of the Git history (e.g., a screenshot of the change log, statistics on number of merge requests accepted and from who, etc.) for the documentation owner to include in the final project report.

Project documentation owner

As the project documentation owner, you are responsible for compiling the required documentation and final project report from your team. Each member of the project team should individually document their contributions to the project as part of the report however you are responsible for setting up the initial template and structure of the report in a location that other team members can access. You should also ensure that the final report reads as a single unified document when complete. The document should be a maximum of 10 pages and should be roughly structured as follows:

- **Introduction** – describing the task and how the application works (at a high-level)
- **Workflow** – describing breakdown/ownership of tasks and how they were tracked
- **GitLab** – describing how collaboration on the project application code was managed
- **Code** – describing implementation of the application, including design decisions, etc.

Project demonstration owner

As the project demonstration owner, you are responsible for creating a short (**90 sec max**) video demonstration of the completed code running on a *MAKER-PI-PICO* project board. The demonstration should showcase all the requested functionality that was set out in the **Introduction** and **Instructions** sections of this document.

Project code owner

As the project code owner, you are responsible for ensuring that the overall application functions as expected. Everyone in the team should contribute to implementing the code required for the application, however as the other team members have additional roles to perform you should (in agreement with the other members of your project team) undertake a slightly large share of coding tasks.

Project Deliverables

D1 – A declaration document from each person outlining the aspects of the project they did
D2 – All source code and CMakeLists files for the group project (*.c, *.h, *.S, *.pio, *.txt)
D3 – All build output collateral for the group project (*.elf, *.uf2, *.dis)
D4 – The project report document and Doxygen documentation for the group project
D5 – The short video demonstration showcasing the project code running on the hardware

Grading Scheme

Complete all the steps in the instructions section and then upload the required deliverables specified below to the "assign02" assignment in Blackboard before the assignment deadline. Each group will be able to submit the result three times, but the last attempt will be marked. Please be aware of the submission limit and the assessment criteria.

It is assumed that all team members will contribute to the project equally and, as a result, will receive equal marks for the project – if one or more team members do not participate in the project or it is deemed by consensus that they do not perform their fair-share of the work, it is important to let us know so that we can investigate balancing the marking allocation fairly between all of the project team members as appropriate.

This assignment is worth a total of **30% of your year-end results for the CSU232021 module** and the marking scheme is defined as follows.

	INCOMPLETE [0%]	COMPETENT [50%]	PROFICIENT [100%]
SUBMISSION [5%]	The assignment was not submitted and/or the group does not have a valid reason.	Some deliverables are missing, or assignment was submitted after the deadline.	All the deliverables for this assignment have been submitted before the deadline.
BASIC FEATURES [40%]	An insufficient number of basic features were implemented or don't work as expected.	Most of the features were implemented but the application does not work as expected.	All the basic game-play features were fully implemented and verified to work.
OPTIONAL FEATURE [10%]	No attempt was made to implement any of the optional features.	One of the optional features was fully implemented.	Both of the optional features were fully implemented.
PROJECT FLOW [10%]	There was no (or very little) attempt made to define or track project milestones.	A reasonable effort was made to define and track the project milestones.	The project was divided into clearly defined and tracked milestones.
REPOSITORY FLOW [10%]	There are either no or very few code commits to the group repository and/or the commits are badly commented.	Git feature branches and merge requests in use however, there are few commits or poorly commented commits.	Git feature branches and merge requests in use along with adequate, well commented commits.
DOCUMENTATION [10%]	The final project report was of poor quality or was not submitted.	The project report was not of good quality or was missing content.	The project report was of good quality and met all requirements.
DEMONSTRATION [10%]	The project showcase video was of poor	The project showcase video did not clearly	The project showcase video did clearly

	quality or was not submitted.	demonstrate the game functionality.	demonstrate the game functionality.
OVERALL QUALITY [5%]	The submitted source code was badly structured or has not been well commented.	The submitted source code was adequately structured along with reasonable comments.	The submitted source code was structured well and with clear commenting.