MICROPROCESSOR SYSTEMS

ASSIGNMENT 2

Flynn Clyne

Parker Jack Kavanagh

Aditya Kharbanda

Raaghav Sawhney

Inam Syed

Introduction:

The objective of this project is to create a game aimed at teaching Morse code, utilizing the skills acquired throughout the module.

The game consists of multiple levels, each with their own features and difficulties. Upon initializing the board, a welcome message is displayed and the RGB LED is set to blue. A welcome banner, along with the group number, and the basic instructions for the gameplay, and a brief description of each level are displayed. In order to select a level, the morse code equivalent sequence to the level number must be entered, this pattern is provided on the initial UI.

Level 1: The first of these levels involves the user being presented with an alphanumeric character and its corresponding morse code. The user attempts to enter the shown morse code using the designated button, GP21. If the code is entered correctly, the code is correct, otherwise, it is not.

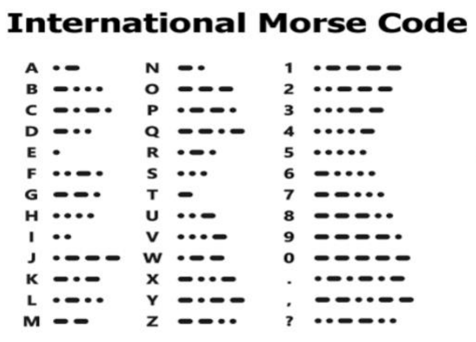
Level 2: The distinction between the first two levels is minimal. Level one provides both the characters along with their morse code equivalent, serving as an introduction to morse code. Level 2 doesn't include the morse equivalent, requiring players to memorize Morse code patterns, thus increasing the difficulty.

Level 3 (optional): This level is similar to level 1; however, players are presented with words instead of individual characters. Each word is accompanied by its Morse code counterpart.

Level 4 (optional): Serving as an advanced version of level #3, this level presents players with words without the corresponding Morse code. Players must deduce the Morse code for each word. This requires deeper understanding and knowledge of Morse code.

Each level necessitates the completion of five consecutive Morse sequences to progress. Upon reaching the final level, players receive a message indicating that the game has been beaten.

- GP21 Press for a short duration = results in a Morse "dot."
- GP21 Press for a longer duration = results in a Morse "dash."
- No input for more than 1 second = results in a "space" in the Morse sequence.
- No input for more than 2 second = current sequence is considered complete.



International Morse Code

A	•–	N	–•	1	•––––
B	–•••	O	–––	2	••–––
C	–•–•	P	•––•	3	•••––
D	–••	Q	––•–	4	••••–
E	•	R	•–•	5	•••••
F	••–•	S	•••	6	–••••
G	––•	T	–	7	––•••
H	••••	U	••–	8	–––••
I	••	V	•••–	9	––––•
J	•–––	W	•––	0	–––––
K	–•–	X	–••–	.	•–•–•–
L	•–••	Y	–•––	,	––••––
M	––	Z	––••	?	••––••

Figure 1: A-? morse code equivalent.

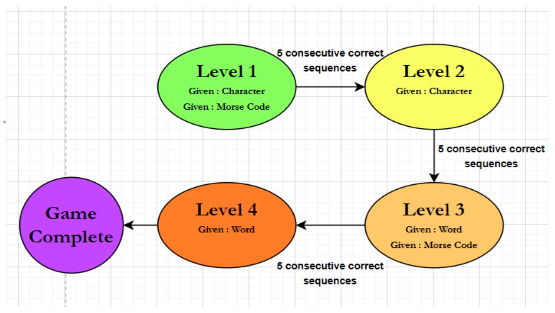


Figure 2: Flowchart of games' levels.

When commencing, the player is assigned three lives, this being the max number of lives available at any given moment. If mistakes are made when playing the game, a life will be deducted; however, if a code is entered correctly, a life is gained. This giving the user plenty of room for error.

The lives remaining are represented by the RPG LED, 3 lives being the colour green, 2, yellow, and 1 orange, indicating hazard as if another mistake is made, the game is over.

Upon completion of a level, as mentioned above, this is achieved by introducing 5 correct answers in a row, the stats of the player will be displayed, along with a level completed message. The stats displayed include the level completed as well as the number of attempts made.

As mentioned previously, if the players lives drop to 0, the game is lost. A game over message is displayed, along with the stats of the player, this being the number of stages played, the maximum level reached, and win and loss count.

If all four levels are completed, a "game complete" message is displayed on the UI along with the stats of the player, said stats are the same as the ones listed for when the player is defeated.

It must be noted that if the player doesn't enter an input for over 9 seconds, the game will end automatically, and they will have to restart.

Workflow:

For this project, each of the five group members were assigned with a role, depending on the role, the individual focused on different sections. The workflow for the group was distributed as follows:

- Project workflow owner – Aditya Kharbanda
- GitLab workflow owner – Flynn Clyne
- Project documentation owner – Parker Jack Kavanagh
- Project demonstration owner – Raaghav Sawhney
- Project code owner– Inam Syed
-

Role	**Responsibilities**
Project workflow owner	The project workflow owner was responsible for overseeing the overall organization of operations. This included effective communication to track deadlines and ensure timely achievement of milestones.
GitLab workflow owner	The GitLab workflow owner was tasked with creating the new shared project repository and overseeing the management of merge requests, ensuring their proper integration into the repository.

Project documentation owner	The project documentation owner was responsible for creating the layout of the project and ensuring that the final report is concise.
Project demonstration owner	The project demonstration is responsible for producing a concise 90-second video showcasing the final project developed by the group. This demonstration highlighted all project functionalities.
Project code owner	The project code owner was tasked with ensuring the code functions as intended. They also ensured that team members met their coding requirements.

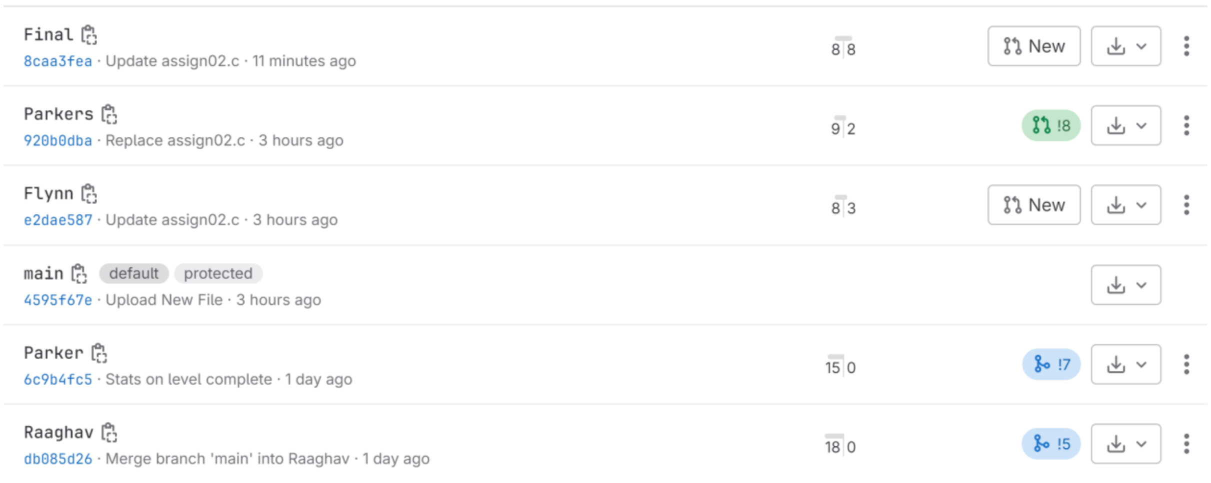
Figure 3: Table containing the Responsibilities of each team member.

GitLab Collaboration:

Gitlab was the collaborative platform used for this project. By using this platform, all of the group members were able to collaborate simultaneously; in addition to this, the tasks could be easily divided among members and issues could be flagged.

It was of great benefit to be able to create numerous branches where each individual could develop their code and test it. Once the member was satisfied with their work, a merge request could be made and upon approval, the code would be merged into the main branch.

A number of the designed branches can be seen in *image 4*, the branches displayed are merely examples of the branches made, these feature the information such as when the most recent commit was made, as well as how many times it has been merged into the main branch.



Final
8caa3fea · Update assign02.c · 11 minutes ago   8 | 8   New

Parkers
920b0dba · Replace assign02.c · 3 hours ago   9 | 2   !8

Flynn
e2dae587 · Update assign02.c · 3 hours ago   8 | 3   New

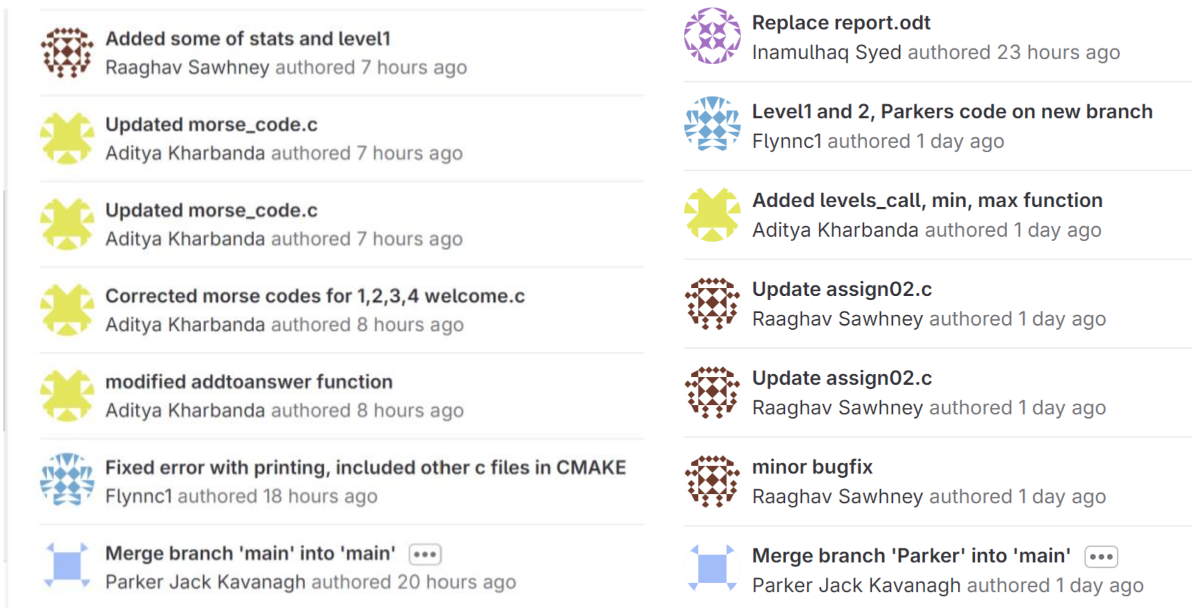
main   default   protected
4595f67e · Upload New File · 3 hours ago

Parker
6c9b4fc5 · Stats on level complete · 1 day ago   15 | 0   !7

Raaghav
db085d26 · Merge branch 'main' into Raaghav · 1 day ago   18 | 0   !5

Figure 4: Examples of active branches in the GitLab repository.

In addition, Gitlab provided features such as commit history, where all commits made are clearly documented and provide a timestamp as to when the commit was made. This can be seen in *figure 5.*


Added some of stats and level1
Raaghav Sawhney authored 7 hours ago

Updated morse_code.c
Aditya Kharbanda authored 7 hours ago

Updated morse_code.c
Aditya Kharbanda authored 7 hours ago

Corrected morse codes for 1,2,3,4 welcome.c
Aditya Kharbanda authored 8 hours ago

modified addtoanswer function
Aditya Kharbanda authored 8 hours ago

Fixed error with printing, included other c files in CMAKE
Flynnc1 authored 18 hours ago

Merge branch 'main' into 'main'
Parker Jack Kavanagh authored 20 hours ago

Replace report.odt
Inamulhaq Syed authored 23 hours ago

Level1 and 2, Parkers code on new branch
Flynnc1 authored 1 day ago

Added levels_call, min, max function
Aditya Kharbanda authored 1 day ago

Update assign02.c
Raaghav Sawhney authored 1 day ago

Update assign02.c
Raaghav Sawhney authored 1 day ago

minor bugfix
Raaghav Sawhney authored 1 day ago

Merge branch 'Parker' into 'main'
Parker Jack Kavanagh authored 1 day ago

Figure 5: GitLab commit history example from main branch.

TimeLine of the work:

This section of the report outlines the timeframe on which the workflow was produced, in addition, the GitLab history will be provided to reenforce the statements made.

Week	Achievements
Week 1	- Reviewed the project prompt and created the shared GitLab repository. - Conducted an in-person meeting to allocate roles to each team member.
Week 2	- Initial Setup of arm code developed: Buttons, GPIO ISR and Alarm. - Delegated tasks for members.
Week 3	- C code initiated: function for encoding and decoding. - RGB LED setup. - Report created.
Week 4	- Implementation of levels 1-4 completed. - Optimal UI created. - Report complete. - Video demonstration complete.

Figure 6: Table addressing the workflow per week.

Initiation
Planning
Execution
Monitoring and Controlling
Closure

Over the four weeks, the workflow was steady; however, during week 4 was when most of the project gained full form. Resulting in a fully functioning morse code game, including both the mandatory features, as well as the optional ones, this can be clearly seen in *figure 7* .

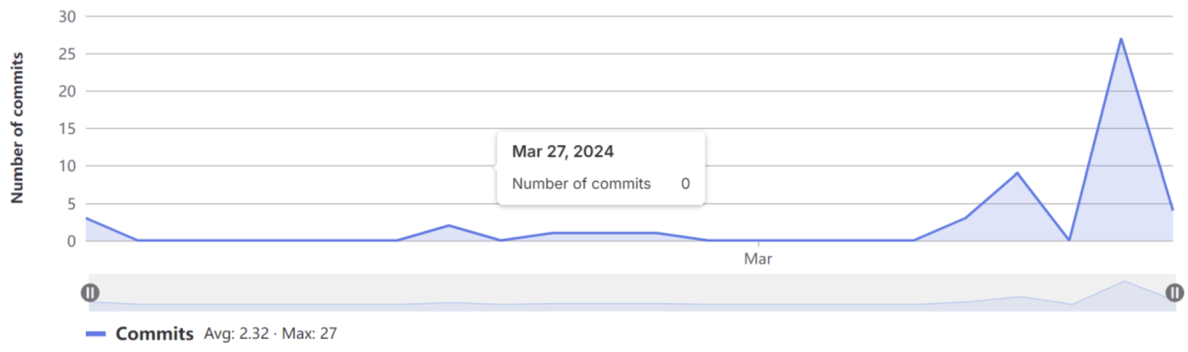


Figure 7: GitLab commit history on main branch (A spike during the last week).

It is to be noted that the figure above is merely an example of the workflow as multiple branches were being worked on during the development of the product.

Code Functionality:

An explanation of the product as a whole from software point of view is described in this section of the report. When needed, certain features won't be elaborated on due to simplicity.

- **C files:** various C files were created to avoid clustering the main file; the files created include welcome.c, morse.c and assign02.c.

  - Welcome.c, is a file that was created to display the initial user interface, the reason for which this was created was due to the welcome banner and initial instructions consisting of numerous print statements that would take up unnecessary room in the main c file and cause unnecessary cluster.

  - Morse_code.c, a file that consists of numerous arrays containing characters and strings, as well as arrays containing their equivalent morse code patterns. In addition, functions to encode and decode the characters and strings.

  - Assign02.c is the primary C file in this project, consisting of over 500 lines of code. This file uses numerous headers, including the header files created for the two previously mentioned C files. The layout is as follows:

    - Multiple global variables are declared, said variables will be used throughout the code, some used as counters, others as containers.

    - All basic I/O arm called functions are defined as well as the main entry point to said assembly code. Said functions include initialising GPIO pins, setting their direction, getting and setting the value of a pin and finally, enabling falling-edge and rising-edge interrupts for the pin.

- Functions to initialise the RGB LED are declared as well as a function that takes an input and sets the LED to a different colour depending on said input. Five modes are available: blue, green, yellow, orange, and red, each with their own meaning. Blue represents the availability to choose a level, no game has been initiated, green indicates that a level has been chosen, as well as the user having 3 lives remaining. Yellow represents 2 levels remaining and orange, one. Red indicates that all lives have been lost and the Game is over.

- Following three functions to display messages on the user interface are present, said functions could have been moved to another C file to reduce cluster, however this isn't of much concern as they aren't as extensive as the display created in the welcome.c file. The first of the three displays "Level Complete", along with information on the level that has been completed, and the number of attempts undertaken.
  The second function displays the statistics of the player once the game has ended, this being because the user lost, or the final level is beat. Accompanying the "stats" banner, statistics such as the games played, the level reached, the success count and the loss count are displayed.
  Finally, a function to display a game over banner was created, this is called once the game has been lost.

- Four level functions were created, each level as mentioned in the introduction consisting in different gameplay. Said functions were designed to embody the descriptions mentioned previously. This being the random generation of either characters or strings, and either displaying them along with their morse equivalent or not, depending on the difficulty level chosen.

- A function to deal with the level selected was implemented, creating a clean slate for the user to work with and displaying the corresponding level features. Functions such as clear array were created to clear the contents stored in the user answer array. In addition to a function to ensure that the first character is not NULL.

- To handle said answer array, a function that checks the answers was created. Said function acts adequately depending on it the input is correct or not, by adding or reducing the players lives as well as checking if the level is complete, and whether the player has any lives remaining.

- To assemble said answer, the users' input is added to the answer array, and for each input, the array iterated further. An initial flag is set, said flag checks if a level has been selected or not.

- Two functions to deal with the time parameter were created, the first of the two gets the start time, this being when an input is provided. Following, a function to determine the duration that the button is pressed for was created, said function is responsible of deciding if the input is a "dot", "dash" or "space".

- Finally, the main function, within this function all basic I/O functions are initialised, an offset for the program and the initial LED value are set. To

follow, the welcome banner is displayed and the main entry point to the assembly code is called.
The watchdog timer is enabled and set to 9 seconds, and an infinite while loop is present to ensure that the game is constantly operation and that the program is not exited.

- **Assembly file:** consisting of only one file of around 200 lines of code, it provides all of the assembly logic to obtain the desired result for named morse code game.

  Initially, all parameters are declared, this being the pin number, mask values, offset values, as well as any set time values. Following this, the entry point to the assembly language is found, within this main function, the buttons are initialised, the GPIO and Alarm ISR are installed, followed by the alarm being set. Within each of these calls, multiple branches are visited:

  - Init_buttons: this function initialises GP21 as an input as well as enabling the falling and rising edges. To do so, it calls functions found in the C files and passes the required parameters.
  - Install_gpio_isr: a subroutine that is used to install the GPIO interrupt service routine. From this function, the gpio_isr routine is visited where the watchdog timer is updated, and the routine responsible for setting the alarm is branched to. The remainder of this routine is responsible for determining the time the button is pressed for by comparing said times to one of the three possible input times, a dot, dash, or space. Once this is determined, the result is sent to the C code and worked with.
  - Install_alarm_isr: responsible for installing the alarm interrupt service handler.
  - Set_alarm_time: branched to from Install_gpio_isr, it essentially stores the time provided to it.
  - Alarm_isr: checks whether or not the button has been pressed and returns the time value to set_alarm_time.

Individual contribution:

In this section of the report, a brief description of what each member of the group carried out is expressed. More detailed declaration will be submitted in a separate document.

- **Flynn Clynn:** primary coder of the group, developed a great deal of the assembly code, as well as the C code. In addition, maintained the GitLab repository up to standard, creating branches to be worked on as well as identifying issues to be handled.
- **Parker Jack Kavanagh:** aided in the assembly coding, designed all UI C code, this being the banners displayed on each event. Helped with the logic of the C code as well as commenting on the code in Dioxygen format. Main contributor of the written report.
- **Aditya Kharbanda:** as the project workflow owner, posted found issues within the code onto GitLab, assisted with level 1-4 functions as well as assisting the demonstrator during the recording of the video.
- **Raaghav Sawhney:** tested and showcased the final product, contributed with important functions throughout the C code as well as working on the incorporation of the "dot" and "dash" character in the assembly code.
- **Inam Syed:** contributed with the creation of certain C functions, ex. encode/decode. Helped with the report documentation as well as the structure and layout. In addition to merging multiple videos to obtain a final demonstration worthy product.