Lab #6 - Lab Write-up and documentation

Course	ECEN2370 Embeded Software Eng
Date	@May 5, 2024
Type	Assignment
Status	Completed

Project description

This was my final project for ECEN 2370 - Embedded Systems. It is an implementation of the known and loved game of Tic-Tac-Toe using a STM32F429i Discovery Board. This project relies on the effective use of the HAL (Hardware Abstraction Layer) in order to correctly access and manipulate the peripherals from the board in order to implement the game functionality. The game allows for 2 playing modes, single player and multiplayer respectively, as well as accurate tracking of playing time and total record of wins.

Project scope

This project scope covered virtually all the topics learned throughout the course.

The breakdown is as follows:

1. **Utilized Peripherals**:
    - **RNG (Random Number Generator)**: Used for random moves in the game.
    - **Timers**: Used for timing game events.
    - **Interrupts**: Used for handling user input and game reset logic.

- **LCD Screen**: Displayed the intro screen, game board, and game result screen.

2. **Tic-Tac-Toe Game**:
   - Standard 3×3 game board.
   - User selects either single player or multiplayer mode.
   - Implemented X's and O's.
   - Game ends when someone gets 3 in a row or if there is a stalemate.
3. **Additional Requirements Implemented**:
   - Created two screens besides the game board screen:
     - One for selecting 1the game mode.
     - One for displaying game results, time elapsed, and current record (W-L-Ties).
   - Implemented AI logic for 1 player using the RNG (brute force possibilities).

Timeline

The timeline was followed religiously since the start of the project given the large scope of it, and the short amount of time we had to complete it.

1. **April 19-20:**
   - ☑ ~~Project setup i.e peripheral configuration.~~
   - ☑ ~~Set up the LCD screen and basic game structure.~~
2. **April 22-24:**
   - ☑ ~~Implement the intro screen (1-2 player select).~~
   - ☑ ~~Set up the game board and basic game logic.~~
   - ☑ ~~Basic 2 player functionality~~
   - ☑ ~~Project Check-In #1 (4/23).~~
3. **April 25-27:**
   - ☑ ~~Ensure all possible endgames (W-L-T).~~
   - ☑ ~~Finish two-player functionality.~~
   - ☑ ~~Perform 2 player testing.~~
   - ☑ ~~Start AI logic for 1 player game.~~
   - ☑ ~~Debug AI logic for 1 player game.~~
4. **April 28-29:**
   - ☑ ~~Continue testing and debugging.~~
   - ☑ ~~Implement end game screen~~

- [x] ~~Implement interrupt button reset~~
- [x] ~~Implement Timer usage~~

5. **April 30 - May 1:**
   - [x] ~~Finish End Game Logic~~
   - [x] ~~Integration Testing~~
   - [x] ~~Debug Complete Functionality~~
   - [x] ~~Code(5/1 @ 11:59pm).~~
6. **May 2 - 5:**
   - [x] ~~Lab Write-up (5/5).~~
   - [x] ~~Lab documentation (5/5).~~

Work breakdown

- `Game.h/c` Fileset to implement complete game logic:
  - **Game State management:** when selecting game mode, playing either 1 or 2 players, and end conditions met to terminate the game.
  - **Parsing user's input:** in the form of mode selection, a selection of a cell to make a move on the game, or reading the pressing of the user button to reset the game.
  - **LCD integration:** calling the appropriate LCD functions at the right time in order to provide a seamless gaming experience.
- `LCD_Driver.h/c` Modification:
  - **Creation of Game specific functions:** created the 3 separate functions to display the 3 different required screens, the Intro game screen where the user could select the game mode, the playing screen that displayed the board, and the end game screen that displayed the win record the time elapsed, and the state of the match only after an end condition was met.
  - **Modified included functions:** Modified functions from the starter code in order to satisfy the project's needs such as: drawing horizontal lines, drawing rectangles with specific with and length, and displaying complete words.
- `Button_Driver.h/c` Modification:
  - **Configure GPIO for button interrupt mode:** Implemented a correct initialization of the GPIO peripheral needed to use the blue user button on interrupt mode.
  - **Handle button interrupts for resetting game:** Implemented the correct logic to handle the interrupt once it was called in order to reset the game state completely, independent of where the current logic was i.e. user could reset the game at any stage.
- `RNG.h/c` Implementation:

    - **Initialize and configure RNG peripheral for single-player mode:** Correctly initialize the RNG peripheral so as to use it for the computer moves.
    - **Generate random numbers for automated moves:** Effective use of HAL to fetch a random number and parse it correctly for it to be used for the computer moves.
- `Application.c` Integration & debugging:
    - **Fully integrate all the game functionality:** Majority of the errors throughout the project stemmed from integration errors such as reading the game state flags correctly as they got set i.e. checking when the game was over, checking when the player changed, transitioning from the welcome screen to the game board and start reading user input as such, and many more.
    - **Debug:** Using the STMCube IDE effectively to keep track of where the program failed, reading the correct registers to check that the right pins where being updated successfully, eliminating infinite loops within the game logic, terminating functions, setting flags correctly, etc.

Testing strategy

- **Unit Testing:**
    - Tested all created and modified components independently such as the game logic functions for parsing player input, making a player move, checking for win conditions, LCD drawing functions for the three game screens, button interrupt handler for resetting the game state, and RNG fetching for making computer moves.
- **Integration Testing:**
    - Verify that the peripherals where being initialized, modified, and cleared correctly.
    - Tested integration of function calls and overall game logic.
    - Verified the correct logic stream was being followed for the game logic.
- **System Testing:**
    - Made end-to-end testing of the complete program to ensure proper functionality and good user experience for both playing modes.
- **Regression Testing:**
    - Re-test existing functionality every time new code was added or modified.

Use cases

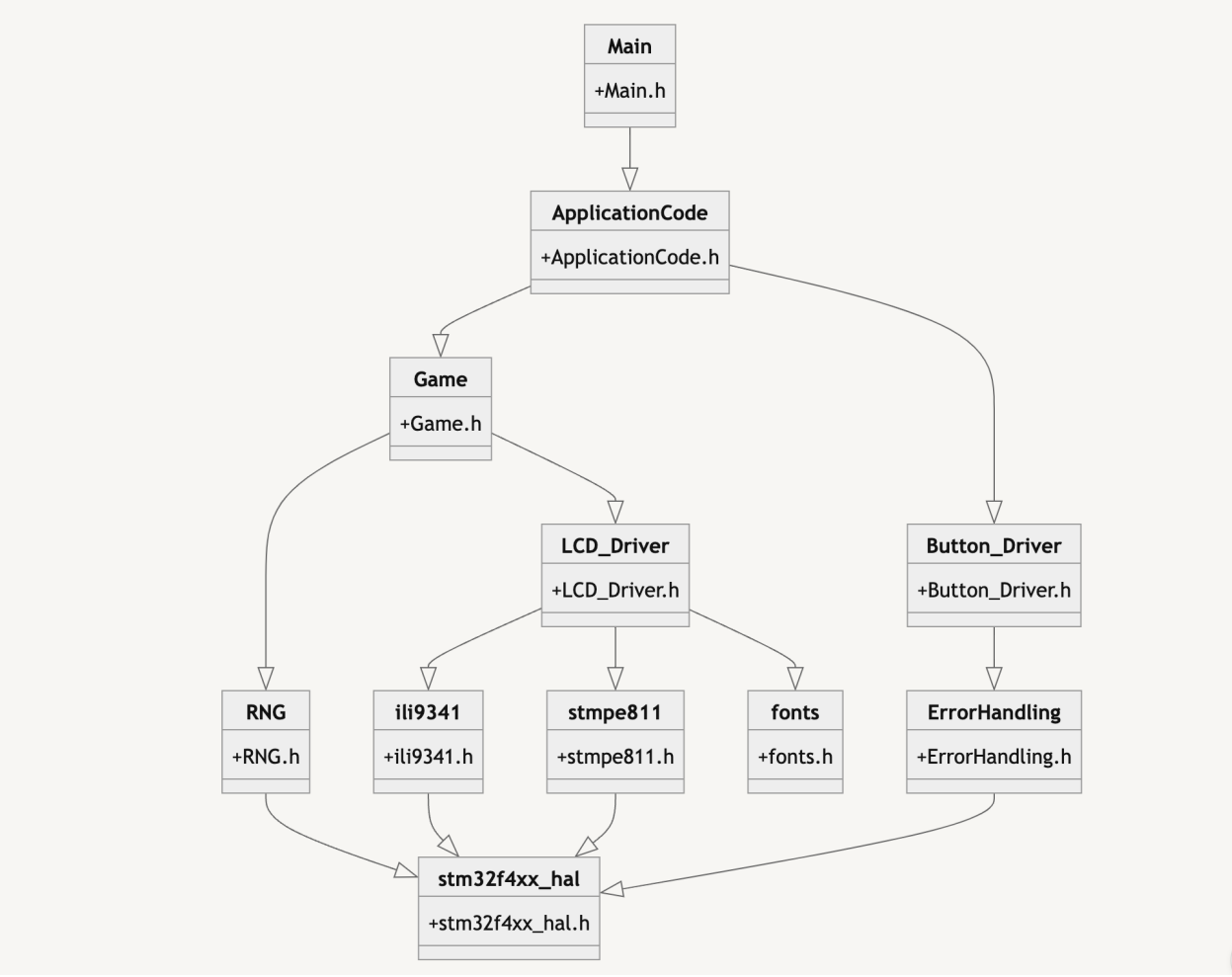
1. **Single Player Mode:**
    - User is presented with welcome screen and selects *single-player* mode.
    - User makes moves by touching desired cell on the board
    - The computer “thinks” and responds after a short amount of time.
    - Game ends when either player or computer wins or the board is full.

2. **Multiplayer Mode:**
    - User is presented with welcome screen and selects *Multiplayer* mode.
    - Two users play against each other, taking turns to make moves.
    - Game ends when either player wins or the board is full.

Project documentation

- Coding documentation should consist of two things: a document(s) for your code and a “how-to” guide explaining how to utilize your project.
    - The document for your code should describe essential functions and their purpose, your coding hierarchy will also be a part of this

1. Coding Hierarchy



2. Essential Functions

This are the functions that I spent the most time on, and the most important ones in order for the game to function correctly.

2.1. Integration Handler (ApplicationCode.c)

- `ApplicationInit():` Initialize all the appropriate peripherals for the game to function correctly such as the monitor handles for printf functionality, LCD screen, the RNG, and the user button.
- `LCD_IntroScreen_Polling():` It will implement a polling method to get the user input for the welcome screen, where the user decided which playing mode to select. It parses said input and sets the appropriate flag for the game logic. Once the game mode is selected, the function calls the appropriate function to continue the game. **Notice: the function does not terminate once the user selects a game mode, but rather it creates a nested function call to start the game. The logic of this function is such that when the end game flag is raised, this nested function calls will terminate gracefully.*
- `LCD_Touch_Polling_Demo():` This function was modified to fit the purposes of this project, since it already implements a successful polling logic to get the user inputs from the LCD display, it now reads the said input and sends it to the appropriate game function for it to be parsed and interpreted as a move. This function runs until the end game flag is raised.
- `LCD_EndGame():` This function is responsible for passing the correct arguments to the game logic, and then enters an infinite loop that will run until the user invokes an external interrupt by pressing the button and thereby resetting the game logic successfully.
- `EXTI0_IRQHandler():` This function is the IRQ handler for the user button, and it calls the game reset function. This function sets the end game flag that allows all the other nested calls to terminate gracefully.

2.2. Game Logic (game.c)

- `GAME_init()`: Initializes the game board and its variables.
- `GAME_readMove(uint16_t x, uint16_t y)`: Takes in an x and y coordinates as arguments, it maps it out to a given cell on the board. The full LCD screen is utilized for this.
- `GAME_makeMove(uint16_t row, uint16_t col)`: Takes in a row and column as argument to update the game board data structure, 2D char array. It will only update the board if the cell the user selected is empty. Otherwise it will not count that as a valid input for that current user, and will wait for the user to select a valid cell.
- `GAME_AIMove():` Uses the `RNG` to produce a random number and make a valid computer move on the board.
- `GAME_checkWin()`: Checks for a win condition on the game board.
- `GAME_reset()`: Resets the game to its initial state.

2.3. LCD Driver (LCD_Driver.c)

- `LCD_Init()`: Initializes the LCD screen and necessary peripherals using HAL.

- `LCD_Draw_TicTacToe()`: Draws the Tic-Tac-Toe grid on the LCD.
- `LCD_Draw_Welcome_Screen():` Draws the initial screen for the user to select the playing mode.
- `LCD_Draw_EndGame(char x, uint32_t played, char *wins):` Takes in the end game state as a char, the total time played, and the record of the wins. This arguments are updated by `applicationCode.c` and `game.c`. Given the right arguments, it will display the end game screen to the user with the above mentioned data.

2.4. Random Number Generator (RNG.c)

- `RNG_Init()`: Initializes the RNG peripheral using HAL.
- `RNG_GetRandomNumber()`: Generates a random number using HAL.

2.5. User Input (Button.c)

- `Button_Init()`: Initializes GPIO pins for the user button using HAL.
- `btnInitIntMode()`: Initializes GPIO pins for button input in interrupt mode using HAL.

3. How to Utilize the Tic-Tac-Toe Project

1. Set up the STM32F429i board with necessary peripherals such as the LCD screen, the user button, the system timers, the GPIO peripherals, and the I2C3 serial communication. No new peripherals where used for this project.
2. Set up the correct debugging configuration that enables the printf functionality within STM32CubeIDE environment.
3. Compile and flash the project onto the board, the code should compile with **0 errors, and 0 warnings.** All where handled properly.
4. Follow the game logic accordingly, by selecting the game mode and playing the game to completion, reset the current game at any stage by pressing the blue button.
5. Enjoy playing Tic-Tac-Toe on your STM32F429i discovery board against your friends or a very bad computer!

Struggles and Obstacles and how you overcame them

There were several struggles and obstacles I encountered throughout this project. However, thanks to previous experience, I dedicated a good amount of time at the beginning to ensure I had a realistic timeline and a good plan if attack. This helped me stay on track throughout the short amount of time we had to complete this project and minimized the struggles as much as possible. Still, some inherent obstacles faced where:

- Understanding and integrating new LCD screen peripheral with HAL. Even though I starter code was provided, I spent a considerable amount of time just familiarizing myself with this peripheral and the given code. Modifying and breaking functions constantly was part of the process. Thankfully, the GDB debug functionality of STM32CubeIDE was extremely helpful for stepping through the functions and noting where the errors occurred and why.
- Peripheral initialization and management. This project was relatively simple in that we only used 2 new peripherals for the project, `RNG & LCDT` whilst the rest was material that had already been covered in previous labs. Moreover, even the use of this peripherals follows the same principle as the ones we had already used. Sifting through the HAL code to find the right `structs` and function calls to manage the peripherals was no knew challenge.
- **Overall Integration:** This was the biggest obstacle I faced during this project. Being able to integrate the simplicity of the tic-tac-toe game with the complexity of the STM32F42i discovery board. The pieces to both of these problems where simple, putting them together was the real challenge. Making sure the right functions where being called at the right time, with the right arguments. Terminating flows where they ought to terminate. And just dealing with weird bugs that stemmed from this integration.

What you learned about this project

There are several outcomes from this project. Effectively putting together not only all the topics we learned in class, but moreover all the principles I've learned throughout my extremely short coding lifetime. Some of the most notable things I learned from this project where:

- Gained proficiency in using HAL for the STM32F429i discovery board which I plan on extrapolating to any STM32 microcontroller development with a HAL.
- Improved my understanding of hardware-software interactions in embedded systems.
- Enhances my problem-solving skills in debugging and managing complex systems.

What would you do differently if given a fresh "restart" for this project

As mentioned before, something I feel I did very well was the initial phase of the project; the scope and timeline part. In all honesty, I realistic timeline with feasible goals was empirical for the successful completion of this project. Without it, I would have more likely stray of course and end up cramming and rushing development which would have inevitably lead to nastier bugs and errors in logic.

I for one, **would not do anything differently if I re-started this same project.** I believe took the right steps at the right time which allowed me to get stuck early, learn early, and successfully resolve all the problems I had in my code way before the deadline.

- My plan was thorough and the time allocated sufficient for both testing and debugging.

- The tasks I gave myself where small enough to be manageable components, and large enough to create meaningful contributions when completed

If I really had to say something I would do differently now that I am on the other side, it would be to choose to make my own personal project, since the reason why I chose the given tic-tac-toe was because I did not believed I would be able to do anything more advanced. Looking back at it, if I had taken the same steps I took to complete this project, I could have gotten away with creating something more meaningful for myself.

How could your project be improved?

There are several ways this particular project could be improved. Some of them are:

- Improve the existing 1 player game mode where the computer actually tries to win rather than just guessing where to respond.
- Implement additional features such as player statistics tracking and difficulty levels for single-player mode where the computes can go from random moves, to somewhat random moves where it will roughly compare the best move, to an even more complex analytics program where the computers tries to find the best possible move.
- Greatly enhance user interface design for a more pleasing visual, improving backgrounds, button paddings, fonts, and overall color design and resolution (improved by software).
- Including some other peripheral to enhance user experience, such as the LED’s.
- Optimize code for a better performance and resource utilization on the STM32F429i board itself.