Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Embedded System Design
Course Number:	COE718
Semester/Year (e.g.F2016)	F2023

Instructor:	Dr. Gul Khan

Assignment/Lab Number:	Project
Assignment/Lab Title:	Media Center Final Report

Submission Date:	December,1,2023
Due Date:	December,1,2023

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Soubra	Karim	500966625	03	KS

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

COE718 Final Project Report

Karim Soubra

***Abstract*—The final project of COE718 is the culmination of the theory and lab work presented in the course. The final project consists of designing an interactive media center with the following functionality: A photo gallery, An MP3 player, Game. The media center would utilize different features on the board in order to meet the functionality requirements. The photo gallery would display multiple picture that can be cycled between them using the on board joystick. The MP3 player would utilize the USB port to communicate with the lab computer as well as the onboard speaker. Two games are design in order to meet the functionality of an interactive game. The first game requires coordination from the user to hit a target that is auto-generated each round , the second game is a memory game , where the user must remember where the "x" is located. Both games have a built in point system that reset when a user loses or quits.**

***Index Terms*—GLCD, LED, ADC, Keil uVision**

I. Introduction

The final project consists for designing an interactive media center that utilizes the capabilities of the cortex M3. The media center must consist of a main menu, photo gallery, MP3 player, and game. The design of the media center utilizes various concepts covered throughout the course and it is up to the designer as to which concept will be used to implement the media center. The main menu screen is used to select which task the media center must perform. The MP3 player uses its on board speaker to play audio from the lab computer, the photo gallery displays pictures on the GLCD, and the game is design to be an interactive task between the media center and the user. Different features on the board are used to implement the media center in an interactive manner.

II. Related Work

The development of the media center is built upon prerequisites covered through past labs. These labs include tasks that explain how to use certain functions on the board and to fully utilize the Cortex M3.

A. Introduction to Keil uVision & ARM Cortex M3

Lab 1 introduced the capabilities of Keil uVision and Arm Cortex M3, and introduced how to use the MCB1700 dev board. In lab 1 , the various features of the MCB1700 dev board were explored , alongside the capabilities of Keil uVision. Figure 1 shows the files given in lab 1 , that are used to explore the board's and keil uVision's capabilities. The LED.c file given is used to control the LEDs and to explore how the LEDs are enabled and disabled in Keil uVision. The KBD.c file given contains example on how to use the joystick on the MCB1700 dev board. The IRQ.c file contains an interrupt handler to handle the timer interrupt. The GLCD_SPI_LPC1700.c is used to control the GLCD on the MCB1700 board. The example code are used within the main function located in blinky.c . [1] Overall, lab 1 serves as an introduction that teaches the user how to control different features on the MCB1700 board, which serves as the basis for any work that is to be done on the board.

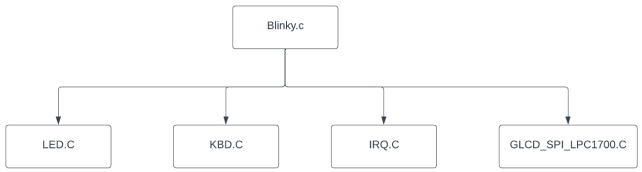

Fig. 1. Lab 1 example programs

Lab 1 introduces ways to turn on and off LEDs , using the LED.c files. As well as ways to manipulate the GLCD . The GLCD_SPI_LPC1700.c provides ways to clear the GLCD, display photos , and print to the GLCDs . The GLCD provides support for a range of various colors that can be set using the GLCD set background and set text color functions. The KBD.c provided functions that can get the direction of the joystick using the get button function.

Lab 1 also introduces the Keil uVision software,and its capabilities, that is used to program the MCB1700 board. The Keil uVision software is a powerful tool that enables the programmer to design and test software that is to be used on the MCB1700 board. Figure 2 shows Keil uVision's built in debugger. The debugger is a powerful tool that allows the programmer to examine their code during run time , this is tool provides a programmer with another lens to look at their code. The debugger comes with a watch window to observe different instance variables , a register window to display the CPU registers as well as the status register , a Disassembly window to observe the generated assembly language of your code, and a performance analyzer for multi threaded programs. [1] Another, important feature of the debugger is the ability to simulate and run program without needing the physical board , therefore speeding up development time.

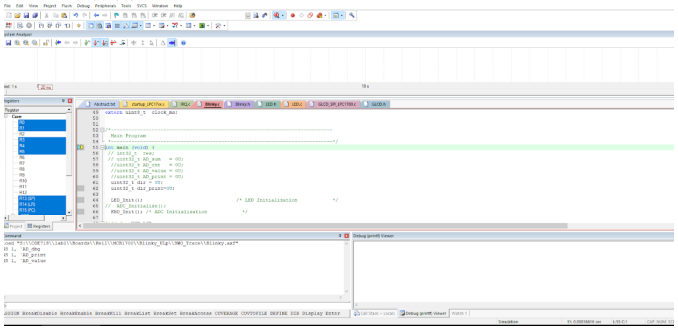

Fig. 2. Keil uVision debugger

B. Exploring Cortex M3 Features for Performance Efficiency

Lab 2 explored the Cortex M3's capabilities. The Cortex M3 has built in features that improved code density, code performance , and improve memory access times. Lab 2 explores the barrel shifter, bit banding, and conditional execution features found on the Cortex M3 [2]. The two example files given in lab 2 are cond_ex.c and bitband.c . Both files are used to demonstrate the features found in the Cortex M3. Figure 3 shows the code block diagram used in lab 2.

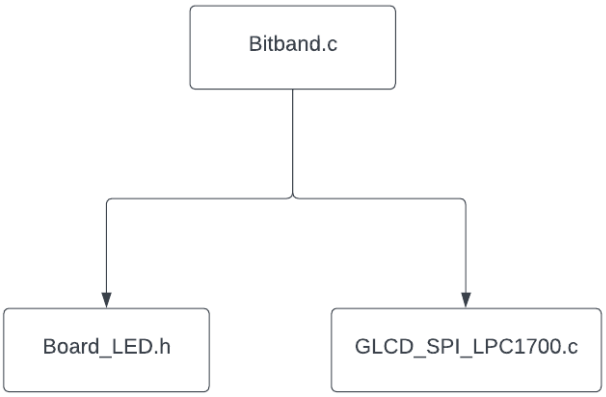

Fig. 3. Code block diagram

The bitband.c file contains examples of using bit band in a program. Bit banding provides an efficient way of accessing bits in memory. Bit banding enables the Cortex M3 to access specific bits in memory rather than reading the whole register, masking, then storing the register [2]. The second example file cond_ex.c contains examples of the uses of barrel shifting and conditional execution. The barrel shifter enables the Cortex M3 to do an ALU and shift or rotate instruction in one clock cycle therefore speeding up its execution and improving code density, many instruction that would take multiple lines of assembly would take just one with the Cortex M3 [2]. The conditional execution feature also increases code density and makes the Cortex M3 more efficient in executing instructions since using conditional instruction removes the need for branch instruction and as a result the Cortex M3's pipeline does not need to flushed due to pipelining hazards, therefore resulting in better performance. Lab 2 also shows how to enable the compiler to take advantage of barrel shifting and conditional execution. The compiler must be set to the maximum level of optimization , which is Level 3 (-O3) and the check box for "optimize for time" must be checked [2].

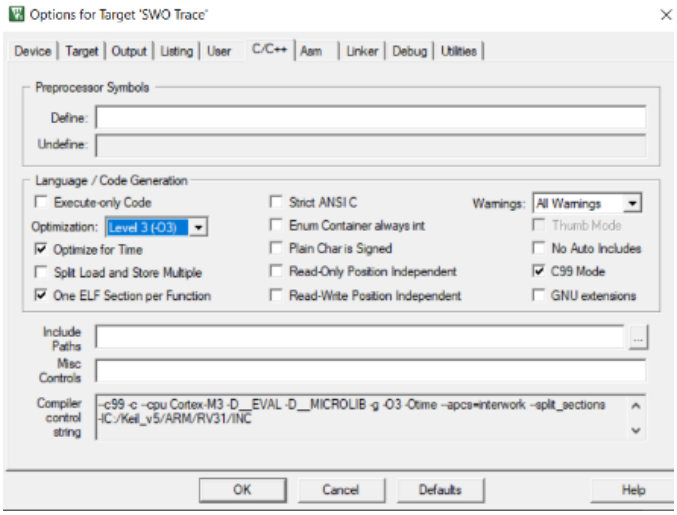

Fig. 4. Compiler setup for optimization

Figure 5 shows the assembly generated after setting up the compiler to take advantage of barrel shifting and conditional execution. The "S" suffix is invoked indicating that the compiler is using barrel shifting and conditional execution.

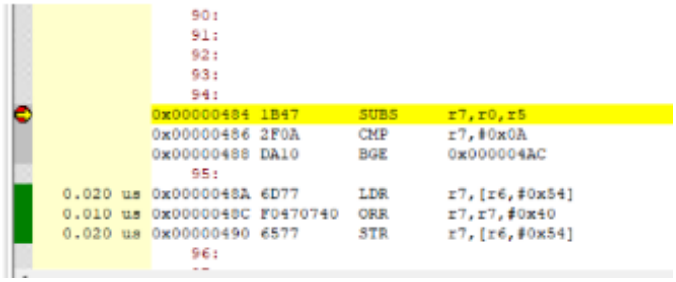

Fig. 5. Compiler setup for optimization

III. Methodology

There are multiple design methodologies to choose from when it comes to designing the media center, therefore there is no one right answer, rather there are optimized "good" designs and less optimized "bad" designs. Whatever methodology chosen, the end goal must be that the media center accomplishes all functionalities required. The requirements for the media center is as follows, the media center must be interactive with the user. The media center must have implemented at least one game. The media center must have a photo gallery, with the ability for the gallery to display multiple photos, and an MP3 player that connects to the lab computer via USB and start playing audio via the onboard speaker. Some implicit requirements of the media center is that the user interface must be friendly and intuitive to the user. Figure 6 shows the general high level architecture of the media center's design.

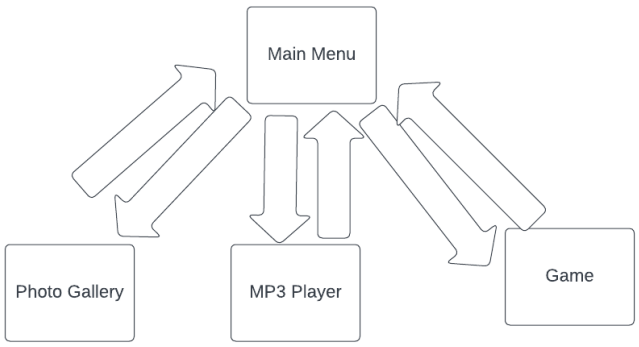

Fig. 6. Media center architecture

Figure 6 shows a high level overview of the design chosen for the media center. The main menu acts as a central hub for the user. The user can go back and forth between the photo gallery, MP3 Player, and game from the main menu. The each block within figure 6 indicates that although the user can switch between functionality, each functionality is fully independent. For example, once the user is in the photo gallery all the logic and variables needed to run the photo gallery is within the photo gallery function, the same logic applies to the MP3 player and game. Each block represents a separate screen the user can access.

Figure 7 shows an alternate design for the media center. The main difference in the alternate design of the media center is that the user can move to any screen from any screen without the need to go through the main menu and selecting which screen to go to next. Ultimately, the design in figure 6 was chosen over the design in figure 7 due to the simplicity of the design in figure 6 compared to the design in figure 7.

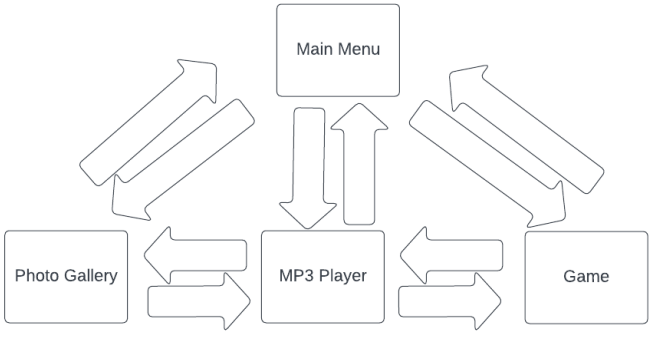

Fig. 7. Media center alternate architecture

IV. Design

Designing the media center began by chooses a board that is capable of implementing the functionality that the media center requires ,and designing the algorithms required to perform and use the hardwaare on a development board that would enable an interactive experience using the media center.

A. Hardware

The board used to implement the media center is the MCB1700 board , since much of the related work done was also on the MCB1700. The MCB1700 board houses a Cortex M3 CPU and an NXP LPC1768 microcontroller [1]. Figure 8 shows a high level view of the MCB1700 development board. The hardware features that are used to implement and design the media center will be covered in the upcoming design sections.

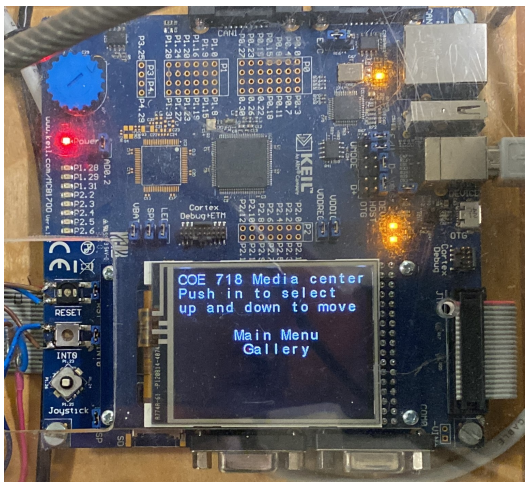

Fig. 8. MCB1700 board

1) LEDs: The MCB1700 board contains 8 LEDs, only some of which are utilized to relay extra information to the user while using the media center. The main use cases for the LEDs in the media center is for implementing both games as the LEDs light up depending on the number of lives a user still has remaining in the game. Figure 9 shows the LEDs on the MCB1700 board.

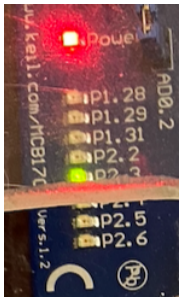

Fig. 9. LEDs

The LEDs on the MCB1700 board are enabled and disabled using a set of GPIO pins. Setting the associated GPIO pin high will turn on the LEDs and setting the GPIO pin low will turn off the LEDs. However , the LEDs must be initialized first since I/O pins will be pulled high by default upon a reset and as a result all the LEDs will be on [3]. The pins associated with the LEDs can be found on written on the board and can be seen in figure 9, the pins are P1.28,P1.29,1.31,2.2,2.3,2.4,2.5, and 2.6. Furthermore , the LEDs must be initialized via the PCONP register [1].

2) GLCD: The MCB1700 comes with a Color QVGA TFT LCD (GLCD) [4]. The GLCD is used to display the current functionality on the screen when the user is using the media center. The GLCD is used so that the media center's user interface can be displayed and to enable the media center to be interactive. Figure 10 shows the Color QVGA TFT LCD found on the MCB1700 board.

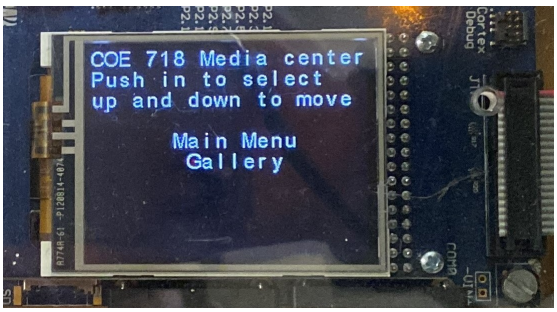

Fig. 10. Color QVGA TFT LCD (GLCD)

The GLCD in figure 10 is a 3.2 inch LCD display, using the Thin Film Transistor(TFT) display technology, this is used to improve the GLCD's image qualtiy [5] [6]. The GLCD communicates with the board via SPI. During initialization CPOL is set to 1 and CPHA is set to 1 also. The LCD must be enabled in the PCONP register as well during initialization [1].

3) Joystick: The joystick on the MCB1700 board is used to take in the user's input and as a means for the user to control the media center. The joystick has 5 different directions of movement up,right,down,left, and select [1]. The select input occurs when the user pushes the joystick in. Figure 11 shows the joystick on the MCB1700 board.

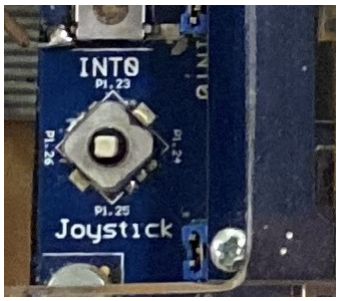

Fig. 11. Joystick

The joystick is initialized via the PCONP register. The MCB1700 board receives inputs from the joystick through a set of GPIO pins connected to the joystick with the direction of the GPIO pins set as an input [1].

4) Speaker: The speaker is used to play audio from the lab computer. The speaker is located below the GLCD screen. The speaker uses the turn pot to change the volume of the speaker. Turning the pot to the left raises the volume and turning the pot to the right lowers the volume. This is possible due to the ADC that takes in the input. The ADC is enabled via the PCONP register,and can measure voltage values between V_{REFN} to V_{REFP} [3] . Turning the pot results in the voltage being measured by the ADC changing which results in the volume being raised or lowered.

Furthermore , the MCB1700 speaker is connected to the lab computer via USB. The MCB1700 has a USB device controller that controls data communication via USB. The Serial Interface engine in the USB controller implements the USB protocol with the data being received via the USB analog transceiver. The USB port communicates with the rest of the board via DMA , with the DMA engine connected to the rest of the board via the AHB bus [3].

B. Algorithms

1) Main menu: Figure 12 shows the algorithm used to design the main menu. The main menu runs in an infinite loop polling the joystick to check for user input.

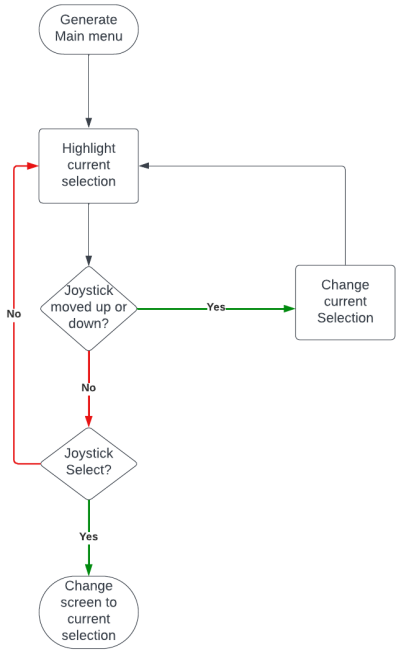

Fig. 12. Main menu Algorithm

The main menu runs in a constant loop until a user selects a new screen. In figure 12 when the main menu is generated , this marks the beginning of the main menu algorithm. The first process after generating the main menu would be to highlight the current selection, during the first iteration of the algorithm the first option is highlighted by default. After highlight the current section, the joystick is checked for a user's input. If the joystick moved up or down , this would indicate that the user is browsing through all the possible selections in the media center , with each up or down movement with the joystick a new selection is highlighted by updating the current selection. However, if the joystick the joystick did not move up or down , a secondary check is performed in order to check whether the user is try to select the currently highlighted option. If the user has not selected the current option , the algorithm will

loop back to the start by highlighting the current option. If the user has selected a the currently highlighted optoin, the loop breaks and the new screen is generated. The possible option the user can select from are the Photo gallery, MP3 player, Game1, and Game2.

2) Photo Gallery: The photo gallery's main function is to listen to the user's input and display an array of photos, one after the other. The user can enter the photo gallery by selecting it in the main menu. Figure 13 shows the photo gallery's implementation.

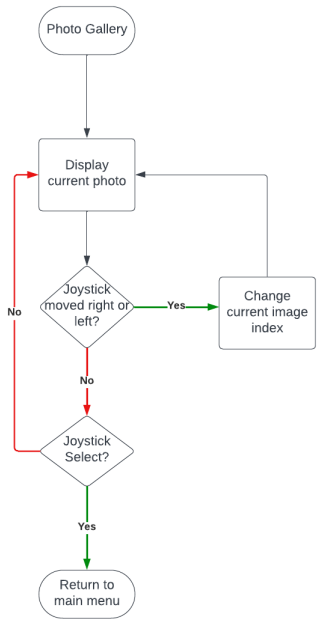

Fig. 13. Photo gallery Algorithm

The photo gallery algorithm runs in a infinite loop until the user exits the photo gallery. When the photo gallery's screen is generated, the inifinite loop begins. The first process in the loop is to display the current photo. In the first iteration of the photo gallery the first image is displayed by default. After displaying the photo , the joystick is checked for the user's input. If the user pushes the joystick left or right then the image index will be changed and the display current photo process will display the updated image. The user can go back and forth between images in this implementation of the photo gallery. However , if the user did not push the joystick left or right , a secondary check is done in order to determine whether the user is trying to exit the photo gallery. If the joystick is pushed in, which is the select input, the photo gallery loop breaks and the user is returned to the main menu.

3) MP3 Player: The MP3 player is the second feature that is implemented in the media center. The onboard speaker is used to play audio from lab computer. The MCB1700 board and the lab computer are connected via USB. In order for the speaker to work , the USB connection between the board and lab computer must be initialized. Figure 14 shows the implementation of the onboard speaker into the media center.

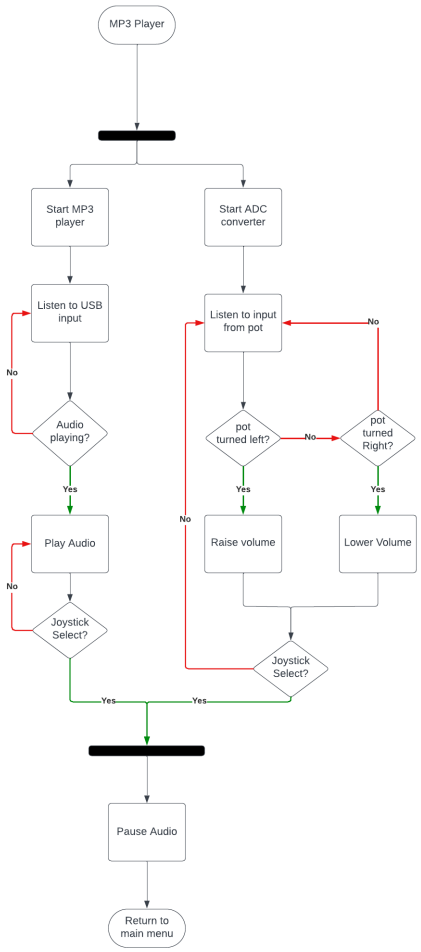

Fig. 14. MP3 Algorithm

For the MP3 player to be implemented , there are two processes that running in parallel. Both processes are running in the form of interrupts. Beginning with the process on the left side. The left process is an interrupt that executes periodically. The left process starts the MP3 player and then listens to the USB input from the lab computer. After listening to the USB port , if there is no audio being played by the lab computer then it will keep listening to the USB port and not play anything through the speakers. If there is audio being played through the lab computer , then the play audio process begins and plays the audio through the speakers, while the audio is playing the joystick is checked if the user pushes in the joystick to quit. If the user quits ,then the pause audio process executes effectively pausing the ADC interrupt and the USB interrupt and stopping the audio.Next, the user will be sent back to the main menu.

The right side of figure 14 shows the process for raising and lowering the volume of the speaker. This process occurs in parallel with the speaker, therefore giving the user the ability

to decrease or increase the volume in real time. After starting the ADC converter , the interrupt constantly reads the turnings pot's voltage which determines the current volume. Turning the pot left or right , changes the voltage level being read by the ADC converter and as a result the volume is raised or lowered. When listening to the input from the pot , a check occurs in order to determine which direction the pot is being turned towards. If the pot turned left then the volume is raised and if the pot turned right the volume is lowered. After lowering or raising the volume , the joystick is checked, in order to determine if the user is trying to exit. If the user pushes the joystick in , both parallel process end and the audio is paused before returning to the main menu.

4) Game 1, Hit the Target: The first game , named Hit the Target , utilizes the joystick and GLCD mechanisms to randomly generate levels the user can interact with and gain points. An example of a level generated can be seen in figure 15.

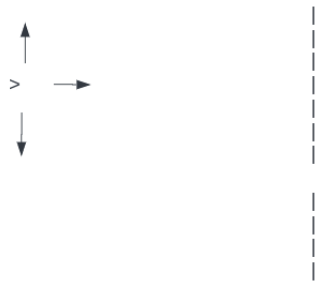

Fig. 15. Game level design

The user can move the player in three directions up, down, and right. The user moves up and down to aim the arrow and pushes the joystick right to fire with the aim of hitting the target. If the user hits the target, a point will be gained and if the user misses , a life will be lost. The user starts with zero points and three lives. The implementation of the Hit the target game can be found in figure 16. The target in the game refers to the gap in the wall that can be seen in figure 15.

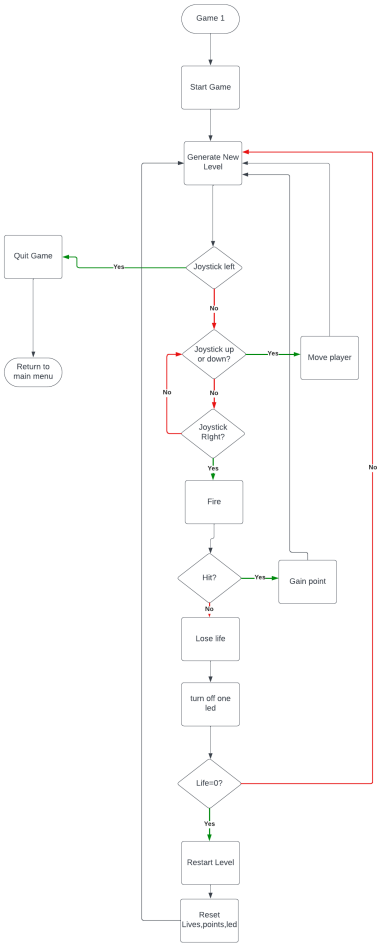

Fig. 16. Game1 algorithm

The game, in figure 16, begins with the start game process and the game keeps running until the user quits the game. In the start game process, the required LEDs are turned on and the lives and points are set to three and zero respectively. After the start game process , a new level is generated then the game will start checking the joystick's input. If the user pushes the joystick in then the quit game process begin , which starts the process of ending the game before finally returning to the main menu. If after generating the level the user does not quit, the next step would be to check if the user is trying to move the player up and down. The game checks whether the joystick is pushed up or down and moves the player accordingly. Moving the player results in regenerating the level with the player's updated position. A second check is done to determine whether the user is pushing the joystick to right, which indicates that the user is done aiming and is wants to hit the target. If the game does indeed detect the user pushed the joystick right after aiming, then the arrow on the screens flies to the right towards the hole in the wall seen in figure 15, the game then detects whether it is a hit or miss. If the game has not detected that the user pushed the joystick to the right then the game will keep listening to the joystick's input until the user has either pushed the joystick up or down, or pushed the joystick

to the right. After firing the arrow , the game detects whether the arrows hits or misses the target. If the arrow hits then the player gains a point and a new level is generated with a new target. However , if the player misses then the player loses a life and one LED turns off represent the one lost life. The LEDs represent a visual aid for the user in order to keep track of the number of lives left, if all three LEDs are on then the user has three lives remaining and if all LEDs are off then the user has no lives remaining. After turning off the LED , the game checks whether the user ran out of lives if the user ran out of lives the game restarts by resetting the player's position and then resets the user's stats in the reset process. In the reset process the user's lives are reset and the user's points is set back to zero. The game keeps going and restart every time the user runs out of lives until the user quits by pushing in the joystick.

5) Game 2,Mine Seeker: The second game implemented in the media center is mine seeker. It is a memory based game where the user needs to memorize the location of the mine before it disappears and then trying to find it by maneuvering throughout the screen trying to find it. The game works with a points system and lives system. The user begins with three lives and zero points. Each miss results in a life being taken and each time a user finds the mine then a point is gained. Figure 17 shows the game's implementation in the media center.

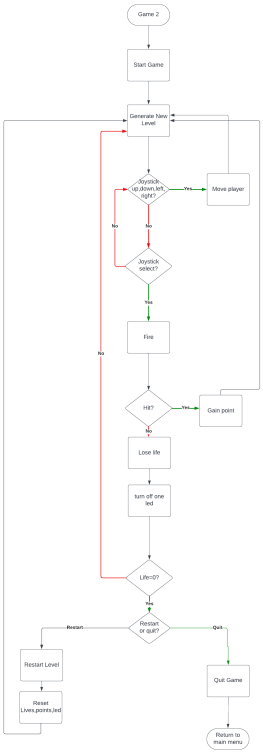

Fig. 17. Game2 algorithm

The user enters the game from the main menu. The game begins by initializing the points, lives, and LEDs in the start game process. After the start game process, a new level is generated. During this time , the location of the mine is displayed on screen for the user . After the level generation, the mines disappear and the user must use the joystick in order to find the Mine's . The first check is used to detect whether the user is trying to maneuver around the screen , if the user pushes the joystick in any direction or than in , this will result in moving the player and generating a new frame with the player's updated location. If the joystick is pushed in , this indicates that the user wants to confirm selection and check if a mine is located under the player's location. Since the game requires inputs from all 5 directions of the joystick, the user can only quit when the user loses all their lives and the game is over. After pushing the joystick in , the game checks whether the current location of the user and compares it with the current location of the mine on the screen in the fire process. After the fire process , the game checks whether the user hit the correct location or missed. If the user hit the correct location , the user will gain a point and an new level is generated. However, if the user misses then the game will deduct one life from the user and turn off one LED, indicating that the user lost a life. After deducting a life, the game checks whether the user ran out of lives if the user ran out of lives. If the user still has lives remaining then a new level will be generated, otherwise a game over screen will be displayed, giving the user the option to restart the game or quit. If the user quits then the quit game process initiates and the user is returned to the main menu. If the user chooses to restart then the game restarts the level and resets all the user's data in the game. The game will reset the user's lives back to 3 , reset the user's points back to 0, and turn on the LEDs according to the new lives the user received.

C. Software

The software used to implement the project is Keil uVision, this software is required in order to program and operate the MCB1700 board [4]. Figure 18 shows an example of the Keil uVision user interface.

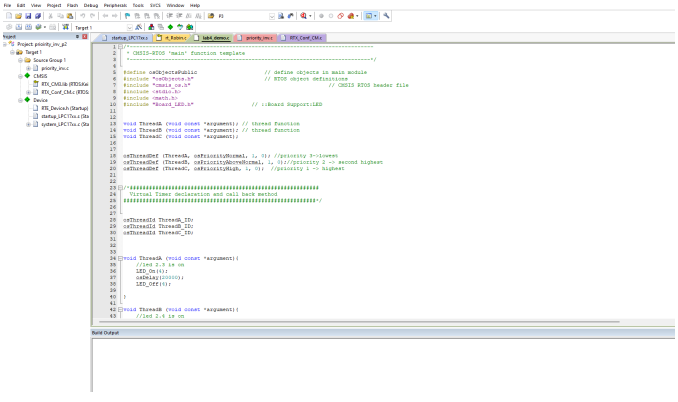

Fig. 18. Keil uVision user interface

The keil uVision user interface consists of a project directory on the left, the main screen where the program is written and an output buil log. The keil uVision also houses a simulator that simulates the MCB1700 board without the

need to have the physical board , which helps speed up development. Finally, the Keil uVision software has a debugger that is useful during development and design of your code.

V. Results

A. Main menu

Figure 19 shows the main menu implemented on the board. The main menu has four options Photo gallery, MP3 player, Game , and Game2. The options are hidden and appear as the user scrolls through the main menu. The instructions for the main menu are printed above the main menu options. The instructions include pushing the joystick up and down to move ,and to push the joystick in to select.

Fig. 19. Main menu

In figure 19, the user is currently hovering over the Game option and if the user were to move the joystick up or down a different option would be highlighted. The user pushes the joystick in, in order to select the currently highlighted option. Figure 10 shows the main menu with a different option highlighted.

B. Gallery

Figure 20 shows the main menu when the user selects the photo gallery functionality. The main menu highlights that the current option is the gallery and the user pushes the joystick in, in order to select the photo gallery.

Fig. 20. Photo gallery

Figure 21 shows the user entering the photo gallery. By default, when entering the photo gallery, the media center displays the first photo. In this implementation, the first photo is always going to be Mario. Above the photos in the photo gallery are instructions for the user, in order for the user to know how to interact with the photo gallery. The instruction indicate to the user that pushing the joystick left or right will switch the photo and pushing the joystick in will result in the user returning to the home screen. The user can push the joystick in and quit the photo gallery at any time. Pushing the joystick to the right will result in the next image displaying and pushing the joystick to the left will result in the previous image appearing. If there are no previous images, the photo gallery will remain on the same image.

Fig. 21. Photo 1

Figure 22 shows the second photo in the photo gallery. This photo appears when the user pushes the joystick to the right. Pushing the photo to the left will result in the Mario photo reappearing. The second photo is that of Luigi. Pushing the joystick right one more time will result in the photo gallery displaying the third and final image, which is the Manchester United Soccer team's logo.

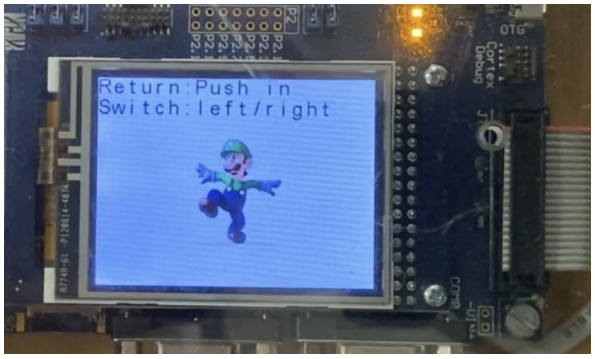

Fig. 22. Photo 2

Figure 23 shows the third and final photo displayed. At the final photo, if the user pushes the joystick right, the photo will not change since the Manchester United team logo is the final picture. However, pushing the joystick left will result in the photo gallery switch back to the previous photos of Mario and Luigi.

Fig. 23. Photo 3

Finally, after seeing all the photos, the user can quit and return to the main menu by pressing the joystick in.

C. MP3 Player

The on board speaker is activated once the user select the MP3 player in the main menu. Figure 24 shows the MP3 player option highlight on the main menu.

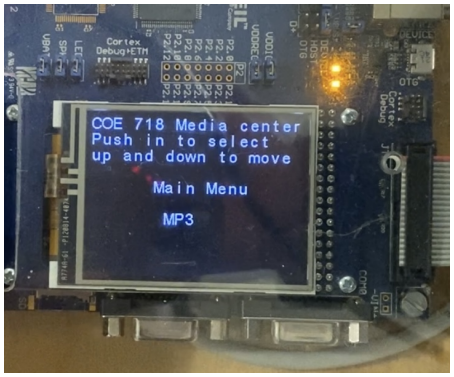

Fig. 24. MP3 Player

When the user selects the MP3 player, the onboard speaker is initialized and begins listening to the USB input. The audio that is played through the lab computer will now be rerouted to the MCB1700 on board speaker. The volume can be adjusted with the turn pot. Figure 25 shows the GLCD when the user has entered the MP3 player. Furthermore, in the top right of figure 25 two LEDs turn on indicating that the USB connection is enabled and that the onboard speaker has been initialized.

Fig. 25. MP3 Player start screen

The MP3 player will keep playing audio until the user exits the MP3 player. The user can exit the MP3 player at any time by pushing the joystick in. Figure 26 shows the result when the user exits the MP3 player. The MP3 player displays a good bye message before return the user to the main menu.

Fig. 26. MP3 Player exit screen

D. Game 1,Hit the Target

Figure 27 shows the game 1 option highlighted in the main menu. The user can select game 1 by pushing the joystick in.

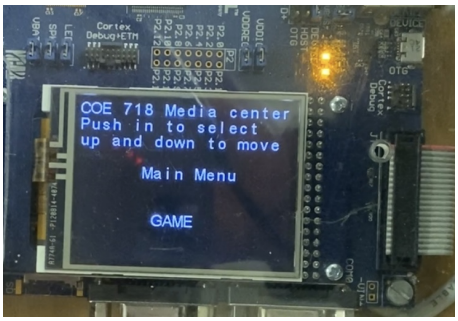

Fig. 27. Main menu game option

When entering the game from the main menu. The game instructions are displayed to the user in order for the user to understand how to use the controls in the game. Figure 28 shows the instructions displayed for the Hit the Target game. The instruction include pushing the joystick right to fire, move the player by pushing the joystick up and down, and to push the joystick in to quit. The game instructions are displayed at the start of every new game, with the instructions being displayed on screen for a few seconds in order to ensure the user has plenty of time to read the instructions.

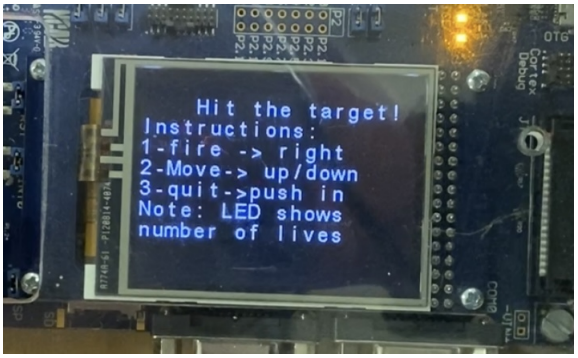

Fig. 28. Game instruction

Figure 29 shows the playing field of the game displayed on the GLCD. The game level is generated after the instruction page. The player always starts at the same position, meanwhile the gap in the wall in figure 29 is randomized at every turn. The player must aim the arrow in between the gap in order to hit the target.

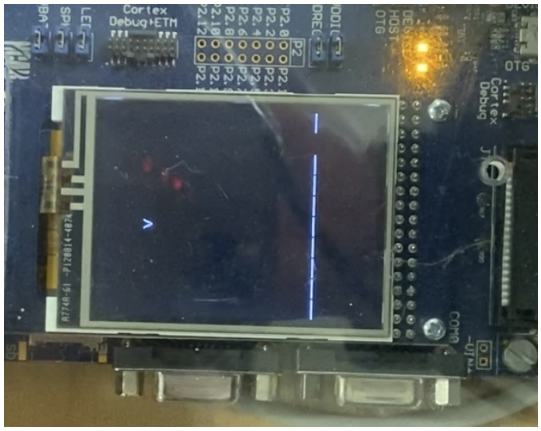

Fig. 29. Game level

Figure 30 shows the LEDs that are used to display the number of lives to the user. At the start of every new game , the LEDs are reset. In figure 30, the three LEDs that are on represent that the user has 3 lives remaining.

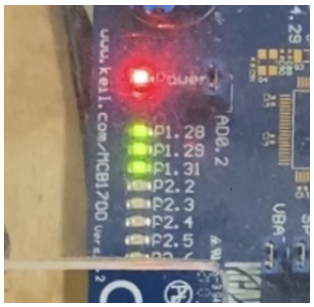

Fig. 30. Game LEDs

Figure 31 shows the fire functionality in the game. The user would push the joystick right , after aiming the arrow by moving up or down. The arrow moves to the right towards the wall. In the example of figure 31 the player misses the gap.

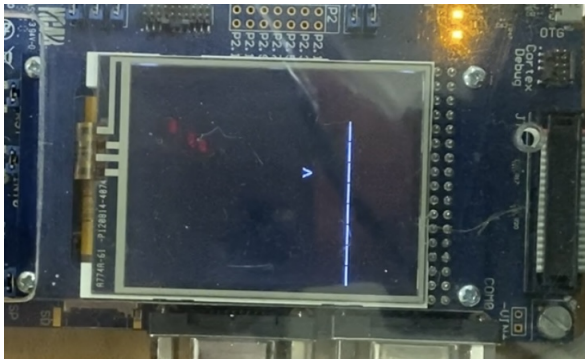

Fig. 31. Fire functionality

Figure 32 shows the result of a miss in the game. A miss screen is displayed , along with the number of lives remaining and the amount of points the user currently has before generating a new level for the user.

Fig. 32. Miss screen

Along with the miss screen, the game also updated the number of LEDs turned on in order to represent the number of lives the player has remaining. Initially, at the start of the game , the user can observe three LEDs on as seen in figure 30, but due to the player missing in the first turn, only two LEDs are on. This tells the user that there is only two lives remaining.

Fig. 33. Updated LEDs

Figure 34 shows the hit screen. This screen only appears when the player successfully hit the target. The hit screen shows the new tally of points, since hitting the target increases the player's points tally by 1.

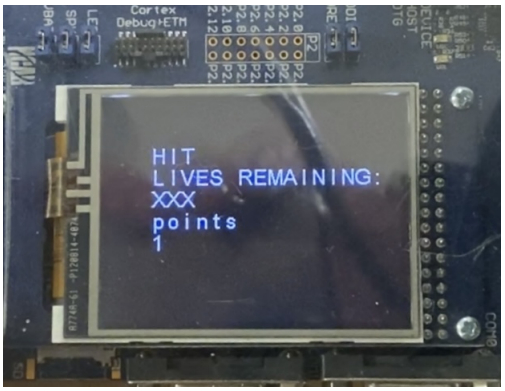

Fig. 34. Hit screen

Figure 35 shows the game over screen. This screen only occurs in the game when the user runs out of lives. After the game over screen, the game will restart. The game will keep going until the user quits by pushing in the joystick.

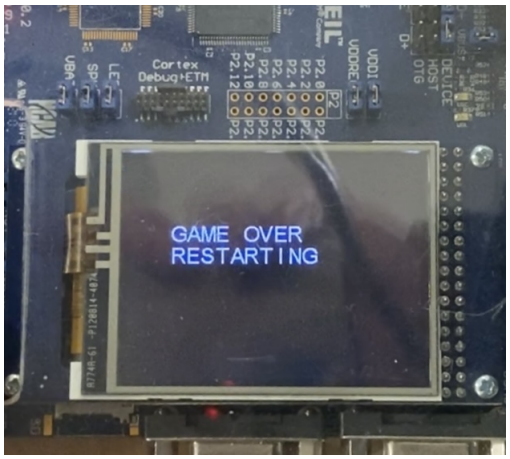

Fig. 35. Game over screen

E. Game 2,Mine Seeker

Figure 36 shows the game 2 option highlighted on the main menu. The user can enter game 2 by pressing the joystick in in order to select the game 2 option on main menu.

Fig. 36. Game 2 main menu

When first entering the Mine Seeker game, the instructions page is displayed on the screen so that the user understands how to play the game. Figure 37 shows the instruction page for the Mine Seeker game. The instruction are as follows move player with joystick, and push joystick in to search

Fig. 37. Game 2 Instructions page

When the game is done displaying the instructions page , the game level will be generated. For the first few periods of generating the game level, the user is able to see where the mine is located. The user must memorize where the mine is located since the mine's location is randomized and can change each round. Figure 38 shows the initial generation of the level.

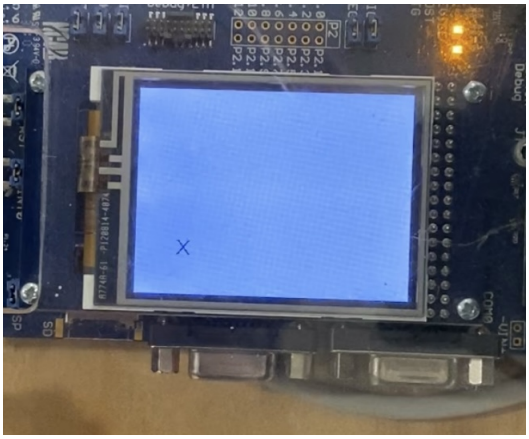

Fig. 38. Game 2 level generation

After generating the level, the screen goes dark and the mine disappears. The user must navigate around the screen and try to find the hidden mine. The user navigates the screen using the joystick and when the user is hovering over a position it thinks the mine is located in, then the user can search that spot by pushing the joystick in. If the user misses then the number of LEDs on decreases. Similar to game one , the LEDs are used to tell the user how many lives are remaining. With each miss, the number of LEDs on decrease by one , the user starts with three lives, and with each miss the user looses a life. Figure 39 shows the game field after generating the level. The game field shows the number of points the user currently has. The number of points is increased by one with each hit.

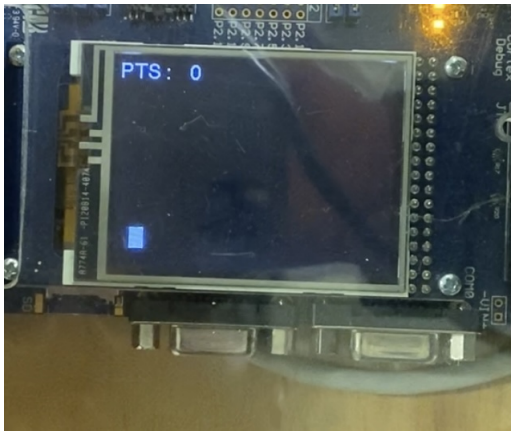

Fig. 39. Game 2 playing field

Figure 40, shows the game over screen. This screen pops up when the user has no more lives remaining. Since, the game requires the use of all 5 of the joystick's directions the only way to quit the game is after losing all three lives. If the user pushes the joystick right then the game restarts, when the game restarts, the number of points is reset to zero and the number of lives will be set to three. If the user pushes the joystick left, the user will be brought back to the main menu.

Fig. 40. Game 2 game over screen

VI. Conclusion

In conclusion, the media center has been a great opportunity to combined everything learned in the previous labs in order to build some larger than the some of its parts. The media center implemented different theory learned throughout the course, such as barrel shifting, conditional execution. As well as, utilizing many of the board's features that were covered in the previous labs such as using the joystick, GLCD, and LEDs

VII. Append A: Code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "Blinky.h"
#include "LPC17xx.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"

#include "usbdmain.h"




char text[10];
char text_l[10];
uint8_t screen_sel;


uint32_t new_dir = 0U;
uint32_t current_dir=0U;
int8_t lives;
uint16_t points;
bool audio_initial_start=false;
uint8_t hole_x=0;
uint8_t hole_y=0;
int8_t current_sel=0;

extern unsigned char luigi[];
extern unsigned char mario[];
extern unsigned char manu[];

void main_menu(void);
void change_screen(int i);
void highlight_option(int i);
void photo_gallery(void);
void switch_pic(int i);
void hit_the_target_game(void);
void instruction(void);
void lives_counter(int hit);
int gen_level(void);
int fire(int player_position, int hole_position);
int move_player(int i,int dir);




void mine_seeker(void);
void seeker_inst(void);

int main (void) {


    GLCD_Init();
    LED_Init();
    KBD_Init();

    for(int i=0;i <=7;i++){

   LED_Off(i);
    }

    screen_sel=5;



  while (1) {

   change_screen(screen_sel);


  }
}


void change_screen(int i){
 GLCD_Clear(Black);
 switch(i){

   case 0:

  switch_pic(0);
  photo_gallery();
  break;
   case 1:
  GLCD_SetTextColor(White);
  GLCD_DisplayString(4, 0, __FI, " MP3 player!
");


  if(!audio_initial_start){

    for(int i=0;i<1000;i++);
    audio_start();

    audio_initial_start=true;
  }else{


    audio_resume();
  }



  GLCD_DisplayString(4, 0, __FI, "
Good Bye!      ");
  for(int i=0;i<100000000;i++);
  GLCD_SetTextColor(Black);
  screen_sel=5;
  break;
   case 2:
    hit_the_target_game();
    break;
   case 3:
  GLCD_SetBackColor(Blue);
  GLCD_SetTextColor(White);
  mine_seeker();
  break;
   default:
  main_menu();
  break;
 }

}

void main_menu(){
   uint8_t current_sel=0;
   highlight_option(current_sel);
   bool exit=false;
   for(;;){
```

```
  new_dir=get_button();
  switch(new_dir){
    case KBD_UP:
     if(current_sel==0){
    current_sel=0;
     }else{
    current_sel--;
     }
     highlight_option(current_sel);
     break;
    case KBD_SELECT:
   screen_sel=current_sel;
   exit=true;
   break;
    case KBD_DOWN:
     if(current_sel==3){
    current_sel=3;
     }else{
    current_sel++;
     }
     highlight_option(current_sel);
   break;

  }
  if(exit){
    break;
  }

   }

}


void highlight_option(int i){


 GLCD_SetBackColor(Black);
 GLCD_SetTextColor(White);
 GLCD_DisplayString(0, 0, __FI,
 \ "COE 718 Media center");
 GLCD_DisplayString(1,0,__FI,
 \"Push in to select   ");
 GLCD_DisplayString(2,0,__FI,
 \"up and down to move ");
 GLCD_DisplayString(3,0,__FI,"      ");
 GLCD_DisplayString(4, 0, __FI, "
Main Menu      ");
 GLCD_DisplayString(9,0,__FI,"      ");

 switch(i){
   case 0:
     GLCD_SetTextColor(White);
     GLCD_DisplayString(5, 0, __FI, "
Gallery   ");
     GLCD_SetTextColor(Black);
     GLCD_DisplayString(6, 0, __FI, "
MP3    ");
     GLCD_SetTextColor(Black);
     GLCD_DisplayString(7, 0, __FI, "
GAME      ");
     GLCD_SetTextColor(Black);
     GLCD_DisplayString(8, 0, __FI, "
GAME2     ");
     break;
   case 1:
     GLCD_SetTextColor(Black);
     GLCD_DisplayString(5, 0, __FI, "
Gallery   ");
     GLCD_SetTextColor(White);
     GLCD_DisplayString(6, 0, __FI, "
MP3    ");
     GLCD_SetTextColor(Black);
     GLCD_DisplayString(7, 0, __FI, "
GAME      ");
     GLCD_SetTextColor(Black);
     GLCD_DisplayString(8, 0, __FI, "
GAME2     ");
     break;
   case 2:
    GLCD_SetTextColor(Black);
    GLCD_DisplayString(5, 0, __FI, "
Gallery   ");

    GLCD_SetTextColor(Black);
    GLCD_DisplayString(6, 0, __FI, "    MP3
");

    GLCD_SetTextColor(White);
    GLCD_DisplayString(7, 0, __FI, "
GAME      ");
    GLCD_SetTextColor(Black);
    GLCD_DisplayString(8, 0, __FI, "
GAME2     ");
    break;
   case 3:
     GLCD_SetTextColor(Black);
     GLCD_DisplayString(5, 0, __FI, "
Gallery   ");
     GLCD_SetTextColor(Black);
     GLCD_DisplayString(6, 0, __FI, "
MP3    ");
     GLCD_SetTextColor(Black);
     GLCD_DisplayString(7, 0, __FI, "
GAME      ");
     GLCD_SetTextColor(White);
     GLCD_DisplayString(8, 0, __FI, "
GAME2     ");
    break;



 }


}
void photo_gallery(){

   uint8_t curr_pic=0;
   bool exit=false;
   new_dir=0U;
   current_dir=0U;
   for(;;){

  new_dir=get_button();
  if(current_dir!=new_dir){
    current_dir=new_dir;

    switch(current_dir){
   case KBD_LEFT:
     curr_pic--;
     if(curr_pic==0){
    curr_pic=0;
     }
     switch_pic(curr_pic);
     break;
   case KBD_RIGHT:
     curr_pic++;
```

```
      if(curr_pic >2){
     curr_pic=2;
      }
      switch_pic(curr_pic);
      break;
    case KBD_SELECT:

      exit=true;
      break;
     }
     if(exit){

    screen_sel=5;
    break;
     }

  }
 }
}

void switch_pic(int i){

  GLCD_Clear(White);
  GLCD_SetBackColor(White);
  GLCD_SetTextColor(Black);
  GLCD_DisplayString(0,0,__FI,"Return:Push in
");
  GLCD_DisplayString(1,0,__FI,"
Switch:left/right");
  switch(i){
    case 0:
     GLCD_Bitmap (  115, 70, 100,
104, mario);
     break;
    case 1:

     GLCD_Bitmap (  115, 70, 100,
121, luigi);
     break;
    case 2:

      GLCD_Bitmap (  70, 55, 180,  183, manu);
      break;
    default:

     GLCD_Bitmap (  115, 70, 100,
121, luigi);
     break;


 }


}

void hit_the_target_game(){

  int hit_or_miss;
  bool exit_game=false;
  new_dir=0U;
  current_dir=0U;

  instruction();
  for(;;){

   GLCD_SetBackColor(Black);
   GLCD_SetTextColor(White);

   points=0;
   lives=3;
   for(int i=0;i<3;i++){

     LED_On(i);
   }
   bool restart_screen=false;
   for(;;){

     GLCD_Clear(Black);
     uint8_t hole = gen_level();
     uint8_t player_y=5;
     GLCD_DisplayString(player_y,3, __FI, ">");
     for(;;){
    new_dir=get_button();
    if(new_dir!=current_dir){
      current_dir=new_dir;

      if(new_dir==KBD_UP){ //move up
     player_y=move_player(player_y,0);
      }else if (new_dir == KBD_RIGHT){

    // return 1 if success ,0 to loose lives
     hit_or_miss=fire(player_y,hole);
     fire(player_y,hole);
     lives_counter(hit_or_miss);
     if(lives<0){
        restart_screen=true;
     }
     break;
      }else if (new_dir ==KBD_DOWN){
     player_y=move_player(player_y,1);
      }
      else if (new_dir ==KBD_SELECT){
     exit_game=true;
     //screen_sel=5;
     for(int i=0;i<3;i++){

       LED_Off(i);
     }
     break;

      }
    }
     }

     if(exit_game || restart_screen){

    if(restart_screen){

      GLCD_Clear(Black);


      GLCD_DisplayString(3, 3, __FI,
"LIVES REMAINING:");
      GLCD_DisplayString(4, 3, __FI,
"XXX");
      GLCD_DisplayString(5, 3, __FI,
"points:");
      GLCD_DisplayString(6, 3, __FI,
"0");
      for(int i=0;i<100000000;i++);
    }
    break;
     }
   }
   if(exit_game){
     screen_sel=5;
     break;
```

```
    }


  }


}

void instruction(){
   GLCD_Clear(Black);
   GLCD_SetBackColor(Black);
   GLCD_SetTextColor(White);
   GLCD_DisplayString(1, 0, __FI, "
Hit the target!   ");
   GLCD_DisplayString(2, 0, __FI, "
Instructions:    ");
   GLCD_DisplayString(3, 0, __FI, "
1-fire -> right   ");
   GLCD_DisplayString(4, 0, __FI, "
2-Move-> up/down");
   GLCD_DisplayString(5, 0, __FI, "
3-quit->push in");
   GLCD_DisplayString(6,0,__FI,"
Note: LED shows   ");
   GLCD_DisplayString(7,0,__FI,"
number of lives   ");

   for(int i=0;i<100000000;i++);

}
void lives_counter(int hit){

  char text[20];
  GLCD_Clear(Black);
  GLCD_SetBackColor(Black);
  GLCD_SetTextColor(White);

  if(hit==0 && lives>=0){


   lives--;
 if(lives>=0){
  sprintf(text,"%d",points);
  GLCD_DisplayString(2, 3, __FI,
"MISS");
  GLCD_DisplayString(3, 3, __FI,
"LIVES REMAINING:");
  GLCD_DisplayString(5, 3, __FI,
"points:");
  GLCD_DisplayString(6, 3, __FI,
(unsigned char *)text);

  switch(lives){
    case 0:
   LED_Off(lives);
   GLCD_DisplayString(4, 3, __FI, "0");
   break;
    case 1:
   LED_Off(lives);
   GLCD_DisplayString(4, 3, __FI, "X");
   break;
    case 2:
   LED_Off(lives);
   GLCD_DisplayString(4, 3, __FI, "XX");
   break;

  }

 }
  }else if(hit==1 && lives>=0){
  points++;
  sprintf(text,"%d",points);
  GLCD_DisplayString(2, 3, __FI,
"HIT");
  GLCD_DisplayString(3, 3, __FI,
"LIVES REMAINING:");
  GLCD_DisplayString(5, 3, __FI,
"points");
  GLCD_DisplayString(6, 3, __FI,
(unsigned char *)text);
  switch(lives){
    case 0:
   GLCD_DisplayString(4, 3, __FI, "0");
   break;
    case 1:
   GLCD_DisplayString(4, 3, __FI, "X");  \
   break;
    case 2:
   GLCD_DisplayString(4, 3, __FI, "XX");
   break;
    case 3:
   GLCD_DisplayString(4, 3, __FI, "XXX");
   break;

  }


  }
  if(lives<0){

 GLCD_Clear(Black);

 GLCD_DisplayString(3, 3, __FI, "GAME OVER");
 GLCD_DisplayString(4, 3, __FI, "RESTARTING");
 for(int i;i<3;i++){

     LED_On(i);
 }
  }


  for(int i=0;i<100000000;i++);

}

int fire(int player_position,
int hole_position){

  for(int i=4; i<17;i++){
 if(player_position==hole_position){
   GLCD_DisplayString(player_position, i-1,
__FI, " ");
   GLCD_DisplayString(player_position, i,
__FI, ">");
 }else{
   if(i<=12){
  GLCD_DisplayString(player_position, i-1,
__FI, " ");
  GLCD_DisplayString(player_position, i,
__FI, ">");
   }
 }
  }

  if(player_position==hole_position){
 GLCD_DisplayString(player_position,17,
__FI, " ");
 return 1;
```

```
  }else{
 GLCD_DisplayString(player_position, 12,
__FI, ">");
 return 0;
  }
}
int move_player(int i,int dir){
  int curr_pos=i;
  if(dir==1){ // moves player down
 if(i==9){
   i=9;
 }else{
   i++;
 }

  }else if(dir==0){
 if(i==0){
   i=0;
 }else{
   i--;
 }
  }

  GLCD_DisplayString(curr_pos, 3, __FI, " ");
  GLCD_DisplayString(i,3,__FI,">");

  return i; //return updated position
}

int gen_level(){
  int hole=rand()%10;

  for(int i=0 ; i <=9;i++){


 if(i!=hole){
   GLCD_DisplayString(i,15,__FI,"|");
 }

  }
  return hole;
}



void mine_seeker(){
  GLCD_Clear(White);

  GLCD_SetBackColor(Black);
  GLCD_SetTextColor(White);
  points=0;
  lives=3;
  new_dir=0U;
  current_dir=0U;
  seeker_inst();



  bool quit_flag=false;
  bool restart_flag=false;
  char text[20];

  for(int i=0;i<3;i++){
  LED_On(i);
  }
  sprintf(text,"%d",points);

  for(;;){
 if(restart_flag){
   points=0;
   lives=3;
   sprintf(text,"%d",points);
   seeker_inst();
   restart_flag=0;
   for(int i=0;i<3;i++){
  LED_On(i);
   }
 }




 hole_x=rand()%8;
 hole_y=rand()%21;
 while(hole_x==0){
   hole_x=rand()%8;
 }
 while(hole_y==0){
  hole_y=rand()%21;
 }

 uint8_t current_x=0;
 uint8_t current_y=1;
 uint8_t new_y=0;
 uint8_t new_x=0;


 GLCD_Clear(White);

 GLCD_SetBackColor(White);
 GLCD_SetTextColor(Black);
 GLCD_DisplayString(hole_y,hole_x,__FI,"X");

 for(int i=0;i<100000000;i++);
 GLCD_Clear(Black);

 GLCD_SetBackColor(Black);
 GLCD_SetTextColor(White);

 GLCD_DisplayString(0, 0, __FI, "PTS:");
 GLCD_DisplayString(0, 5, __FI,
(unsigned char *)text);

 for(;;){
   GLCD_SetBackColor(White);
   GLCD_SetTextColor(White);
   new_dir=get_button();
   if(new_dir!=current_dir){
      current_dir=new_dir;
      if(new_dir==KBD_UP){ //move up
     new_y=current_y-1;

     if(new_y<=0 || current_y<=0){
       new_y=1;
       current_y=1;
     }
     new_x=current_x;

     GLCD_SetBackColor(Black);
     GLCD_SetTextColor(White);
     GLCD_DisplayString(current_y,
current_x,__FI," ");
     current_y=new_y;
     current_x=new_x;
     GLCD_SetBackColor(White);
     GLCD_SetTextColor(White);
```

```
      GLCD_DisplayString(current_y,
current_x,__FI," ");
       }else if (new_dir == KBD_RIGHT){
      new_y=current_y;
      new_x=current_x+1;
      GLCD_SetBackColor(Black);
      GLCD_SetTextColor(White);
      GLCD_DisplayString(current_y,
current_x,__FI," ");
      current_y=new_y;
      current_x=new_x;
      GLCD_SetBackColor(White);
      GLCD_SetTextColor(White);
      GLCD_DisplayString(current_y,
current_x,__FI," ");
       }else if (new_dir ==KBD_DOWN){
      new_y=current_y+1;
      new_x=current_x;
      GLCD_SetBackColor(Black);
      GLCD_SetTextColor(White);
      GLCD_DisplayString(current_y,
current_x,__FI," ");
      current_y=new_y;
      current_x=new_x;
      GLCD_SetBackColor(White);
      GLCD_SetTextColor(White);
      GLCD_DisplayString(current_y,
current_x,__FI," ");
       }
       else if (new_dir ==KBD_SELECT){
      if(current_x==hole_x && current_y==hole_y){
        restart_flag=false;
        points++;
        sprintf(text,"%d",points);
        break;
      }else{
        lives--;
        LED_Off(lives);

        if(lives<0){

        GLCD_SetBackColor(Black);
        GLCD_SetTextColor(White);
        GLCD_DisplayString(0,0,__FI,"
Game Over   ");
        GLCD_DisplayString(1,0,__FI,"
Quit? Y:Left   ");
        GLCD_DisplayString(2,0,__FI,"
Restart? Y:Right ");
        GLCD_DisplayString(3,0,__FI,"      ");
        GLCD_DisplayString(4,0,__FI,"      ");
        GLCD_DisplayString(5,0,__FI,"      ");
        GLCD_DisplayString(6,0,__FI,"      ");
        GLCD_DisplayString(7,0,__FI,"      ");
        GLCD_DisplayString(8,0,__FI,"      ");
        GLCD_DisplayString(9,0,__FI,"      ");

        for(;;){
          uint32_t new_dir2=get_button();
          if(new_dir2==KBD_LEFT){
         quit_flag=true;
         for(int i=0;i<3;i++){

           LED_Off(i);
         }
         break;
          }else if(new_dir2==KBD_RIGHT){
         restart_flag=true;
         break;
          }


        }
        break;
        }

      }


       }else if(new_dir==KBD_LEFT){
      new_y=current_y;
      new_x=current_x-1;
      GLCD_SetBackColor(Black);
      GLCD_SetTextColor(White);
      GLCD_DisplayString(current_y,
current_x,__FI," ");
      current_y=new_y;
      current_x=new_x;
      GLCD_SetBackColor(White);
      GLCD_SetTextColor(White);
      GLCD_DisplayString(current_y,
current_x,__FI," ");

       }
     }
  }
  if(quit_flag){
    screen_sel=5;
    break;
  }

  }
}


void seeker_inst(){

  GLCD_DisplayString(0,0,__FI,"
Mine Seeker!  ");
  GLCD_DisplayString(1,0,__FI,"
intructions:   ");
  GLCD_DisplayString(2,0,__FI,"1-move with
joystick");
  GLCD_DisplayString(3,0,__FI,"2-Push
joystick in  ");
  GLCD_DisplayString(4,0,__FI,"to search   ");
  GLCD_DisplayString(5,0,__FI,"Note: LED shows
");
  GLCD_DisplayString(6,0,__FI,"number of lives
");
  GLCD_DisplayString(7,0,__FI,"      ");
  GLCD_DisplayString(8,0,__FI,"      ");
  GLCD_DisplayString(9,0,__FI,"      ");
  for(int i=0;i<100000000;i++);
}
```

REFERENCES

[1] "Lab 1: Introduction to Keil uVision ARM Cortex M3," Course Website, 2023, november,23,2023. [Online]. Available: https://www.ecb.torontomu.ca/%7Ecourses/coe718/F22-labs/lab1.pdf

[2] "Lab2: Exploring ARM Cortex M3 Features," Course Website, 2023, november,23,2023. [Online]. Available: https://www.ecb.torontomu.ca/%7Ecourses/coe718/F22-labs/lab2.pdf

[3] "LPC17xx User manual," Course Website, 2023, november,23,2023. [Online]. Available: https://www.ecb.torontomu.ca/%7Ecourses/coe718/Data-Sheets/Cortex-M3/NXP%20LPC17XX%20User%20Manual.pdf

[4] "armkeil," Keil Website, 2023, november,23,2023. [Online]. Available: https://www.keil.com/arm/mcb1700/

[5] “Embedded artists,” website, 2023, november,23,2023. [Online]. Available: https://www.embeddedartists.com/products/3-2-inch-qvga-lcd-color-display/
[6] “indelmatic,” website, 2023, november,23,2023. [Online]. Available: https://www.inelmatic.com/blog/news-1/post/what-is-a-tft-screen-30