Term 4

General information

In this portfolio you can find folders which are created for weekly exercises, weekly projects and for final project. There are 4 folders named week {1-4}, where you can find all the progress and work which has been done for that week.

Each week has `./mini-projects` and `./test/demo` folders. In `mini-projects` folder you can find all the completed exercises. Some of the mini-projects include additional features and functionality. In `demo` folder you can find all the "demos". These are basically the example codes, that are provided by lecturers but enhanced further by me, to understand the logic better.

Additionally, there is a `./projects` folder which include all the projects, such as `Nim`, `Lunar Radar`, `Simon Says` and `Morse Trainer`. Each projects is written in its appropriate named C file.

Finally, there is `GHero` folder which is a source folder for guitar hero project. You can find the source code in `./GHero/src/main.c`

Libraries

You can find all the libraries in `./lib` folder. There are 8 libraries, each in their respective packages. In each of the library folder there is appropriate header filer. By checking header you will see which methods were required by different exercises and which are created additionally by me. Most of the libraries are interconnected, which means that some libraries functions are used to define other libraries functions.

More

You can find more detailed information about each folder in this portfolio by checking their respective `README.md` files.

Library Description

The `ADCLib` is responsible for enabling the ADC and using the potentiometer as its analog input value.

Defined function

- `void enableADC()`: enables the ADC, sets the configuration (reference voltage, prescaler, etc.)
- `void enableAutoSampling()`: enables auto-sampling for the ADC.

Library Description

The `buttonLib` is responsible for enabling button(s) as well as for checking their value at any given time.

Defined function

- `void enableButton(int button)`: Enables the button, sets the pull-up resistor as well the `ISR` for the buttons port and pin.
- `void enableAllButtons()`: Enables all buttons, sets the pull-up resistor as well the `ISR` for the buttons port and pin.
- `int buttonPushed(int button)`: Checks if the given button pushed.
- `uint8_t buttonsPushed()`: Checks if the given buttons pushed.
- `int buttonReleased(int button)`: Checks if the given button released.
- `uint8_t buttonsReleased()`: Checks if the given buttons released.

Library Description

The `buzzerLib` is responsible for functions related to buzzer. There are also predefined frequencies.

Defined function

- `void disableBuzzer()`: Disables the buzzer, by clearing the respective DDR pin.
- `void enableBuzzer()`: Enables the buzzer, by setting the respective DDR pin.
- `void playMusic(float* notes, int size, int duration)`: Plays musing using the array of frequency for given duration.
- `void playTone(float frequency, uint32_t duration)`: Plays given frequency for given duration.

Library Description

The `displayLib` library contains functions for writing number/strings/chars and other characters on LED display.

Defined function

- `void blankSegment(uint8_t segment)`: Clears the LED segment of the display.
- `void enableDisplay()`: Enables the display.
- `void writeNumberToSegment(uint8_t segment, uint8_t value)`: Writes number to the segment but doesn't wait.
- `void writeNumber(int number)`: Write number to the display. Max length is 9999.
- `void writeNumberAndWait(int number, int duration, float delay)`: Writes number to the display and waits. Max length is 9999.
- `void writeCharToSegment(uint8_t segment, char character)`: Writes char to the segment but doesn't wait.
- `void writeString(char* str)`: Writes string of maximum 4 characters.
- `void clearDisplay()`: Clears display's by turning it off.
- `uint8_t mapLetterWithHex(char letter)`: Maps letter with respective hex values.
- `void writeAlphabet()`: Writes the 26 letters in a loop.
- `void writeDashToSegment(uint8_t segment, uint8_t value)`: Writes up/middle/down dashes for the given segment.
- `void writeDifficultyAndWait(int difficulty, int duration)`: Final project specific function.
- `void writeDot(uint8_t segment)`: Writes dots to the given segment.
- `void writeLevelAndWait(int duration)`: Final project specific function.
- `void writeScoreAndWait(int score, int duration)`: Final project specific function.
- `void writeSongAndWait(int song, int duration)`: Final project specific function.
- `void writeStringifyNumberAndWait(char* number, int delay)`: Writes stringify number to LED display and waits for given time.
- `void writeStringifyNumbers(char* number)`: Writes stringify number to LED display.
- `void writeTime(int seconds)`: Writes time by given seconds.

Library Description

The `ISRLib` library contains functions for enabling and configuring ISR.

Defined function

- `void enablePinISR(char port, int button)`: Enables the ISR for given port and given button.
- `void enableMultiplePinsISR(char port, uint8_t pinsBinary)`: Enables the ISR for the given port and given buttons.

Library Description

The `ledLib` library contains functions to enable and control LEDs in various manners.

Defined function

- `void enableLed(int ledNumber)`: Enables given led by setting DDR. Internally turns off all leds.

- `void enableMultipleLeds(uint8_t pinsBinary)`: Enables multiple LEDs by setting DDR.
- `void enableAllLeds()`: Enables all LEDs by setting DDR.
- `void toggleLed(int ledNumber, int duration)`: Turn given led on and off one time.
- `void turnOnLed(int ledNumber)`: Turns on given led.
- `void turnOnMultipleLeds(uint8_t pinsBinary)`: Turns on multiple LEDs.
- `void turnOnAllLeds()`: Turns on all LEDs.
- `void turnOffLed(int ledNumber)`: Turns off given led.
- `void turnOffMultipleLeds(uint8_t pinsBinary)`: Turns off multiple LEDs.
- `void turnOffAllLeds()`: Turns off all LEDs.
- `void dimLed(int ledNumber, int percentage, double duration)`: Dims given led for given percentage for given time.
- `void fadeIn(int ledNumber, double duration)`: Fades in the given led for given duration.
- `void fadeOut(int ledNumber, double duration)`: Fades out the given led for given duration.
- `void ledChaos()`: Turns on/off randomly random LEDs.
- `void increaseLedBurnByPeriod(int ledNumber, int startDuration, int endDuration)`: Constantly makes the given LED brighter.
- `void increaseLedBurn(int ledNumber)`: Increase led burn for given led.
- `void toggleLedForGivenTime(int ledNumber, int times, int duration)`: Turn given led on and off contentiously by given duration.
- `void turnOnLedsConsecutively(int duration)`:Turns on LEDs consecutively one by one for given time.
- `void turnOnLedsRandomly(int duration)`: Turns on LEDs randomly for given time.
- `void turnOnLedsByAlphabet(const char *message)`: Morse Trainer specific function.
- `void turnOnLedsRandomlyByAlphabet(const char *message)`: Morse Trainer specific function.
- `int checkLedOn(int ledNumber)`: Check if the LED on.
- `int checkLedOff(int ledNumber)`: Checks if the LED off.
- `void countDown(int duration)`: Counts down the LEDs. Reverse of `void turnOnLedsConsecutively(int duration)`.

- `void dimAllLeds(int percentage, double duration)`: Dims all LEDs for given percentage for given time.
- `void disableLed(int ledNumber)`: Disable given LED.
- `void disableMultipleLeds(uint8_t pinsBinary)`: Disable multiple given LEDs.
- `void disableAllLeds()`: Disable all LEDs.
- `void displayScore(int score)`: Final project specific function.
- `void fadeInAllLeds(double duration)`: Fades in all LEDs for given duration.
- `void fadeOutAllLeds(double duration)`: Fades out all LEDs for given duration.
- `void turnOnAllLedsAndWait(int duration)`: Turns on all LEDs and waits for given time.
- `void turnOffAllLedsAndWait(int duration)`: Turns off all LEDs and waits for given time.

Library Description

The `songLib` library contains functions playing songs.

Defined structs

- `NOTE`: Defines object with `frequency` and `duration` fields.
- `SONG`: Defines object with `name`, `length` and `notes` fields.

Defined function

- `SONG* generateSong(char* name, uint16_t length)`: Generates songs with random notes and for random duration.
- `void playNote(NOTE* note)`: Plays given `NOTE`.
- `void playSong(SONG* song)`: Plays given `SONG`.
- `void deleteSong(SONG* song)`: Deletes song from HEAP.

Guitar Hero

This is a package where you can find the source code for the final project which is Nintendo "Guitar Hero" game. Various libraries' functions are used from `./lib` package to implement the logic of the game.

Main Features

- You can specify your username, which will be saved in case you had top 3 scores.
- You can define the starting level using Potentiometer and ADC. `./ADC.lib` library is the once responsible for enabling ADC and auto sampling. PC0 is the default pins which is listens to for input analog signal.
- You can choose one of the 7 songs you want to play. The songs have different notes and different length of notes.
- You can choose the difficulty level `Easy/Medium/Hardcore`. The difficulty level effects the speed by which the notes "fall down".
- LEDs identify how much it is left to level up.
- After the end of the game see on Serial Monitor the top 3 players' username and record score.

Core Logic Of The Game

The core logic is written in `Timer1` which updates the falling notes, by setting their corresponding variables either 0 or 1. As well as randomly selects segment of the LED display for the next note. `SxU`, `SxM`, `SxD` are responsible for passing the light state value for segment `x` and dash `U/M/D`. The `updateDisplay()` function uses these values to turn on/off the corresponding up, middle or down dashes for given `x` segment. The `SxUF`, `SxMF`, `SxDF` are responsible for passing the frequency associated which each note of the segment X and dash `U/M/D`. These variables are used by `triggerBuzzer()` function to play the frequency is the button is pushed on time. The `Bx` is variable for indicating if the given button `x` is pushed on time or not. The 'displayScore(int currentScore)' function is used to turn on/off the below 4 LEDs according to the current score.

The speed/difficulty of the game is being configured by altering `OCR1A` value, which is the top border, reaching which the display will be updated and dashes will "fall down". After each level up, this value is getting smaller according to the chosen configuration. The potentiometer sets the initial value for `OCR1A`.

The main goal was to use the Timer to update the LED display but at the same time keeps the operations executed in ISR simple.

Other functions and variables are pretty straight forward by just looking at it.

Libraries

In libraries additional functions are defined in order to abstract most of the code as well as make it more reusable. The additional functionality can be found under `//Extra` comment in libraries header filer.

Week 1

This week's exercises are mainly about LEDs which you can find in `ledLib.h`.

Week 2

This week includes `./mini-projects` folder where you can find exercises for using ISR and buttons. The most important one is the project 4 which uses both LEDs, buttons and ISRs.

Week 3

This week folder includes exercises for LED display and potentiometer, as well as C related memory management exercises. All exercises are straightforward and self-explanatory.

Week 4

This week's folder includes exercises about the Timer and Buzzer. It is crucial to check out this folder's `main.c`. Which includes extensively uses Timer for stopwatch.

Projects

Each of the projects file contains logic for the given project. In order to run the code, place the code in `main.c` file.

The projects follow the requirements of the assignments.