Assignment 01 - "Give Me the Binary"

"Embedded Systems e Internet of Things" final report

Lorenzo Cinelli

October 22, 2024

Contents

Chapter 1

Analysis

1.1 Description

The software required is a game called "Turn on the binary code" that has to be implemented in an embedded system. The Goal is to convert a random decimal number shown on the LCD into a binary number within T seconds as many times as possible by turning on the proper leds.

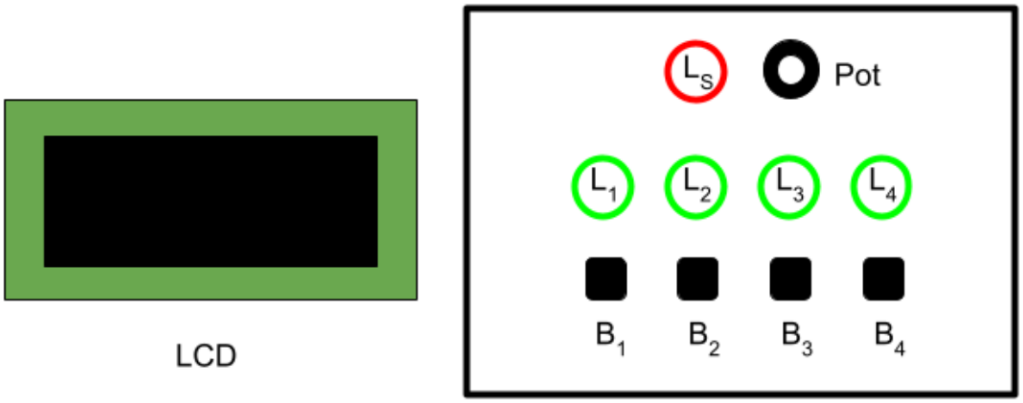

Figure 1.1: Gaming board

The game board is composed by four leds representing the binary digits (0..15). Each led L_i has its corresponding button B_i that can change the state of the led (L_1 is the most significant bit, L_4 is the less significant bit). Each game involves multiple rounds. If the player composes the number correctly, a score - starting from zero - is increased and the game goes on to another round, but the time T of the game is reduced by some factor F. If the player does not compose the correct number within the given time, the

red led L_s is turned on for 1 second and the game ends, displaying the final score on the LCD.
Before starting the game, the potentiometer *Pot* device can be used to set the difficulty D level, that is in the range between 1 and 4 (1 easiest, 4 most difficult). The level must affect the value of the factor F (the more difficult the game is, the greater the factor F must be).

Here is the complete description for the assignment.

1.2 Requirements

- The LCD display has to show different texts depending on the state of the game.
- If the player doesn't interact with the game for some time, the board will power down to minimize electric consumption.
- The player can set the difficulty of the game - before the game starts - by turning the potentiometer.
- During the gameplay leds will change state depending if the player presses their corrisondent button.
- When the player loses the game, his final score is shown while the red led is turned on for 1 second, and then the game will restart.
- If the board is in power down mode, it can be awoken at any moment by pressing any button.

Chapter 2

Design

2.1 Architecture

The application is based on a super loop architecture combined to an event driven architecture.
It is not a pure event driven application because there is the super loop always running, doing something based on which state the application is at. Events are used to switch the state of the application. Base operations for the game are executed during the state switch, while the super loop does polling on timers and executes secondary operations, which are not critical to the operation of the application.

2.2 Detailed design

The design of the application is an event driven architecture on top of a super loop core. Events are triggered by timers and interrupts. Interrupts are triggered by buttons, while timers are managed by the super loop. Each event may change the state of the game. The game has six states: *START, BEGIN, PLAY, SHOW-SCORE, GAME-OVER, SLEEP.*

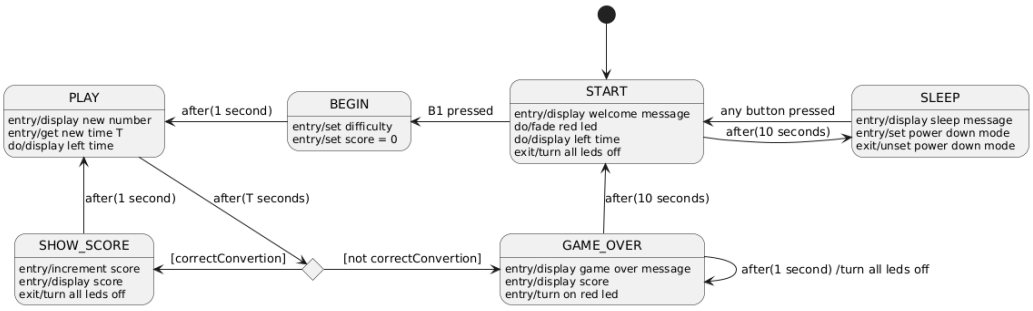

Figure 2.1: States Diagram

2.2.1 State: *START*

This is the initial state, where the player can start the game by pressing B_1 after setting the difficulty using the *Pot*, otherwise after 10 seconds the game will change state to *SLEEP* powering down the board. In the LCD is displayed a welcome message while the red led is pulsing.

2.2.2 State: *BEGIN*

This state last for 1 second after the player has pressed B_1 displaying 'Go' on the LCD, to setup time and the first number. Then the state will be set to *PLAY*.

2.2.3 State: *PLAY*

In this state the player has T seconds to turn on the proper leds to correctly convert the number shown on the LCD. If the binary number corresponds to the decimal number the state will be set to *SHOW-SCORE*, otherwise to *GAME-OVER*.

2.2.4 State: *SHOW-SCORE*

This state displays on the LCD the current score for 1 second, calculates the next number and the next time T and then sets the state to *PLAY*.

2.2.5 State: *GAME-OVER*

This state displays on the LCD a game over message, the final score for 10 seconds and turns on the red led for 1 second. At the end of the 10 seconds sets the state to *START*.

2.2.6 State: *SLEEP*

In this state the board is set to power down mode to minimize the electrical consumption. It can be awoken by pressing any one of the buttons which generates an interrupt to turn on the board. Once turn on the state is set to *START*.

Chapter 3

Development

3.1 Development notes

The development is based on libraries to delegated and make more reusable code. In order to have a more portable code that is detached from the runtime environment, the core of the game is in "game.h" library and not in the main file "GBM.ino". The main file only calls two functions: gameSetup() to setup the game and gamePlay() in the super loop, then all other functions are delegate to the core library. The core is surrounded by some utility libraries, one for each peripheral, and in addition one for the interrupt event queue and one for timer management.

3.1.1 "game.h"

The library is the core of the game.

Two enumerations are defined:

- *enum states*: it tracks the state of the game. Possible values are: *START, BEGIN, PLAY, SHOW_SCORE, GAME_OVER, SLEEP*.
- *enum difficulties*: it tracks the difficulty of the match. Possible values are: *EASY, MEDIUM, HARD, IMPOSSIBLE*.

The library offers the following functions:

- *void setupInitialTime()*: setups the initial number of seconds T to play depending on the selected difficulty.
- *void setupScore()*: initializes the score variable to 0.

- *void scoreIncrement()*: increments the score by 1.
- *char* getNextNum()*: returns a new random number between 0 and 15 converted to a char array.
- *bool checkScore()*: returns *true* if the player has correctly converted the decimal number into a binary number, *false* otherwise.
- *char* getScore()*: returns the actual score converted to a char array.
- *void nextGameTime()*: returns the next time T that the player has for the next turn.
- *void gameSetup()*: setups global variables and calls all the setup functions in utility libraries, such as leds, the lcd screen, the queue, the state, buttons interrupts and the random function.
- *void gamePlay()*: checks if the queue is empty, and if it's not empty then executes the first node. At the end, depending on the state, it shows some text on the lcd and polls the proper timer, occasionally displaying the remaining time on the lcd.

3.1.2 "button_utils.h"

The library manages buttons' interrupts and the interrupt handler. It defines also buttons' physical pin on the board. It uses the library "EnableInterrupt.h". The bouncing problem is managed with a debounce time.
The interrupt event queue has been implemented to reduce the number of instructions in the interrupt handler. In this way the interrupt adds in queue the function that have to be executed after pressing the button.

The library offers the following functions:

- *void buttonPinSetup()*: it setups buttons' pins on the board, setting their mode to *INPUT*.
- *void buttonInterruptSetup(Queue* queue)*: it attach an interrupt for each button's pin.

3.1.3 "lcd_utils.h"

This library manages the lcd screen using the "LiquidCrystal_I2C.h" library. In particular the lcd monitor uses the I2C protocol to communicate with the

board.

The library offers the following functions:

- *void lcdSetup()*: it setups the lcd screen.
- *void printLcd(int x, int y, char* text)*: it prints the string *text* (char array) starting from the coordinates (x, y) of the lcd screen.
- *void startScreen()*: it displays the *state* = *START* screen on the lcd.
- *void beginScreen()*: it displays the *state* = *BEGIN* screen on the lcd.
- *void playScreen()*: it displays the *state* = *PLAY* screen on the lcd.
- *void scoreScreen()*: it displays the *state* = *SHOW_SCORE* screen on the lcd.
- *void gameOverScreen()*: it displays the *state* = *GAME_OVER* screen on the lcd.
- *void sleepScreen()*: it displays the *state* = *SLEEP* screen on the lcd.

3.1.4 "led_utils.h"

This library manages leds and defines their physical pin on the board. The PWM technique (), in combo with the "TimerOne.h" library is used to have the pulsing effect. PWM emulates an analog output with square waves at certain frequencies. "TimerOne" library attaches an interrupt to *Timer1* executing a function each x microseconds. The pulsing speed of the red led depends from the current difficulty selected by the potentiometer, so the faster the pulsing is, the harder the game is.

The library offers the following functions:

- *void ledPinSetup()*: it setups leds' pins on the board, setting their mode to *OUTPUT*.
- *void fadeBlink(int speed)*: it changes the speed of the pulsing changing the initialized time x of *Timer1*.
- *void fadeInitialize(int pin)*: it attaches an interrupt that execute the fade function (on a certain pin) to *Timer1*.
- *void lightRedLed()*: it turns on the red led.

- *void changeLedState(int pin)*: it changes the state of the *pin* pin (if it was ON, it will be turn off and vice versa).
- *void turnAllLedOff()*: it turns all leds off, including the hypotetical pulsing one, setting his intensity to 0.
- *bool isLedOn(int pin)*: it returns *true* if the *pin* led (only among the four green leds) is on, otherwise return *false*.

3.1.5 "potentiomeret_utils.h"

This library manages the potentiometer and defines its pin on the board. It uses the library "Arduino.h".

The library offers the following function:

- *int potentiometerMapRead(int pin, int min, int max)*: reads the analog value returned by the potentiometer and returns the value mapped between $[min, max]$.

3.1.6 "queue_utils.h"

This library manages the interrupt event queue. It creates a linked list with pointers for the head and the tail of the queue to reduce computational effort. It uses "stdlib.h" to manage dynamically allocations. Because of the small memory capacity of the board, the queue dimension is limited to 10 nodes.

Defines two structures:

- *struct Node*: it is the single component of the queue. It contains a pointer *next* to the next node, a pointer *func* to the function that has to be performed and a short parameter *var* that stands for *variants* for the function to execute.
- *struct Queue*: it is the full queue, that contains two pointers: *head* for the first node and *tail* for the last. In addition has a short parameter to track the size.

The library offers the following functions:

- *Queue* initialize()*: it allocates dynamically and returns a *Queue* object with null pointers and dimension 0.
- *void clear(Queue* Q)*: it deallocates all nodes inside the queue Q.

- *void enqueue(Queue* Q, void(*func)(states* state, short var), short var)*: it allocates dynamically a node and adds in the queue Q tail.
- *void dequeue(Queue* Q)*: it deallocates - if present - the first element of the queue Q.
- *Node* getNext(Queue* Q)*: it returns - if present - the first element of the queue Q.

3.1.7 "timer_utils.h"

This library manages game's timers. It uses the "Timer.h" and "avr/sleep.h" libraries. Timers are managed by polling and not by interrupts not to have the risk of losing events in the case that a timer expires at the same time of a button being pressed, and for a well known problem in *Timer1* that makes it inaccurate.

The library offers the following functions:

- *void stopSleepTimer()*: it stops the sleep timer, which is meant to wait 10 seconds and then put the board in power down mode.
- *float sleepTimer(states* state)*: it starts the sleep timer and returns how much time is left from the initial 10 seconds. Once the time is expired the board is setted in power down mode and the state setted to *SLEEP*.
- *float gameTimer(states* state, long time)*: it starts the game timer and returns how much time is left from the initial T seconds. Once the time is expired it checks if the user correctly converted the number and shows the score, setting the state in *SHOW_SCORE* or *GAME_OVER* depending on the check.
- *float scoreTimer(states* state)*: it starts the score timer and returns how much time is left from the initial 1 second. Once the time is expired it sets the state in *PLAY* calculating the new number and the new time T.
- *float overTimer(states* state)*: it starts the score timer and returns how much time is left from the initial 10 seconds. Once the time is expired sets the state to *START* to restart the game.

Appendix A

User guide

Once the circuit is built and the software uploaded on the board, to play the game one needs to connect the board to some power source and follow the instruction on the lcd monitor. More in detail, if the board is in sleep mode (power down mode) one needs to press any button to awake it. When awake select the difficulty using the potentiometer and press B_1 button within 10 seconds. Then press the button to turn on the correspondent led to convert the decimal number shown on the lcd display to a binary number, with leds that represents bits from most significant B_1 on the left to least significant B_4 on the right. Have fun!