Embedded System Programming
3D Labyrinth Game

CONTENTS

This Test Project proposal consists of the following documentation/files:

[1]. Task Board Schematic
[2]. World Skills CPU Board Schematic
[3]. STM32L0x2K6Tx Datasheet
[4]. STM32L0 Reference Manual
[5]. STM32L0xx HAL Driver's description
[6]. I2C serial interface Module datasheet for Character LCD
[7]. Shift Register_A2982 Datasheet
[8]. Row Driver_CD4022B Datasheet
[9]. Column Driver_74HC595 Datasheet
[10]. RG Dot Matrix Pinout
[11]. Project Files Task Phase 1
[12]. Project Files Task Phase 2

INTRODUCTION

DESCRIPTION OF PROJECT AND TASKS

The competition test project is a "blind" 3D labyrinth. The device's user interface consists of

- A 8x8 RG dot matrix display (CA)
- Six push buttons
- A RGB LED (CC)
- A piezo-buzzer and
- USB as COM port interface.

The idea of the game is to find a route from start to finish blindfolded. In other words the game does not show the player where the walls are. The player has to "feel" his way while playing. When the player hits a wall, the device beeps telling the player that a wall blocks the route in the direction the player tried to go. Every time the player hits a wall he gets a penalty point. When the player gets to the finish, the game displays his penalty points and starts a new game.

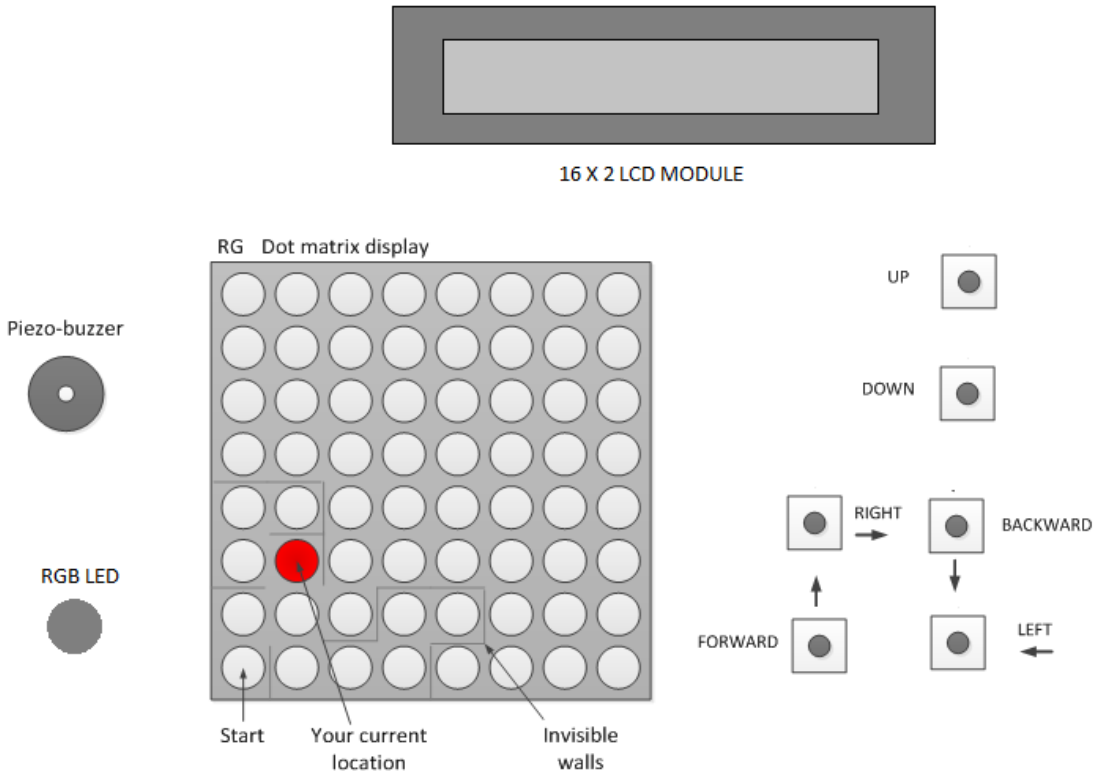


Figure 1: Component used in the project

The labyrinth is in 3D. You move from floor to floor using the up or down key OR Page Up or Page Down keys from PC keyboard via virtual COM terminal, but you must be in a position that allows up or down movement. Each floor is identified by a different led color (table 1). An RGB led is used here to denote the color.

Table 1: Floor colors

Floor	LED color
1	RED
2	GREEN
3	YELLOW
4	BLUE
5	VIOLET
6	CYAN
7	WHITE

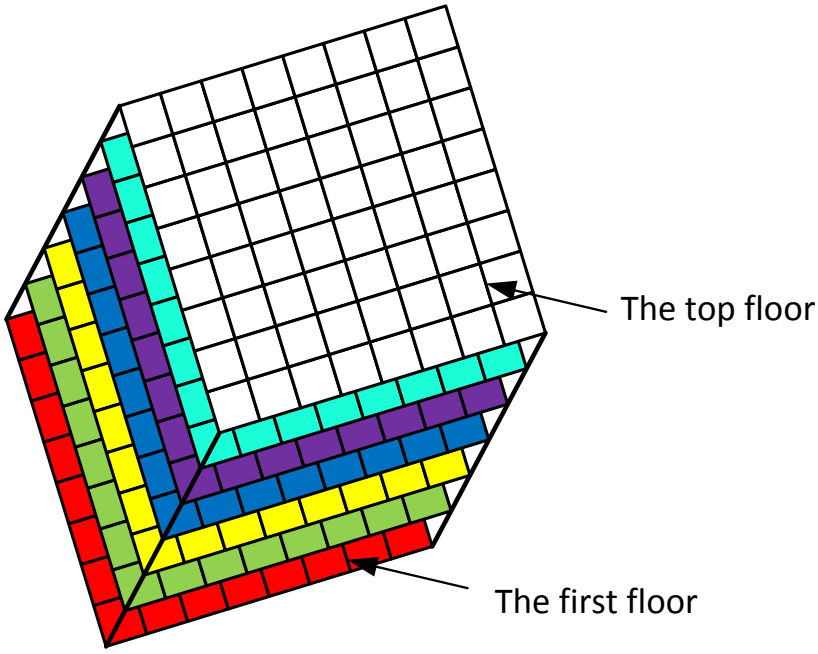


Figure 2: 3D Model for RG 8X8 Dot Matrix

The game starts on floor 1, row 0 and column 0. The game is over when you reach floor 7, row 7 and column 7.

INSTRUCTIONS TO THE COMPETITOR

PROGRAMMING ENVIRONMENT

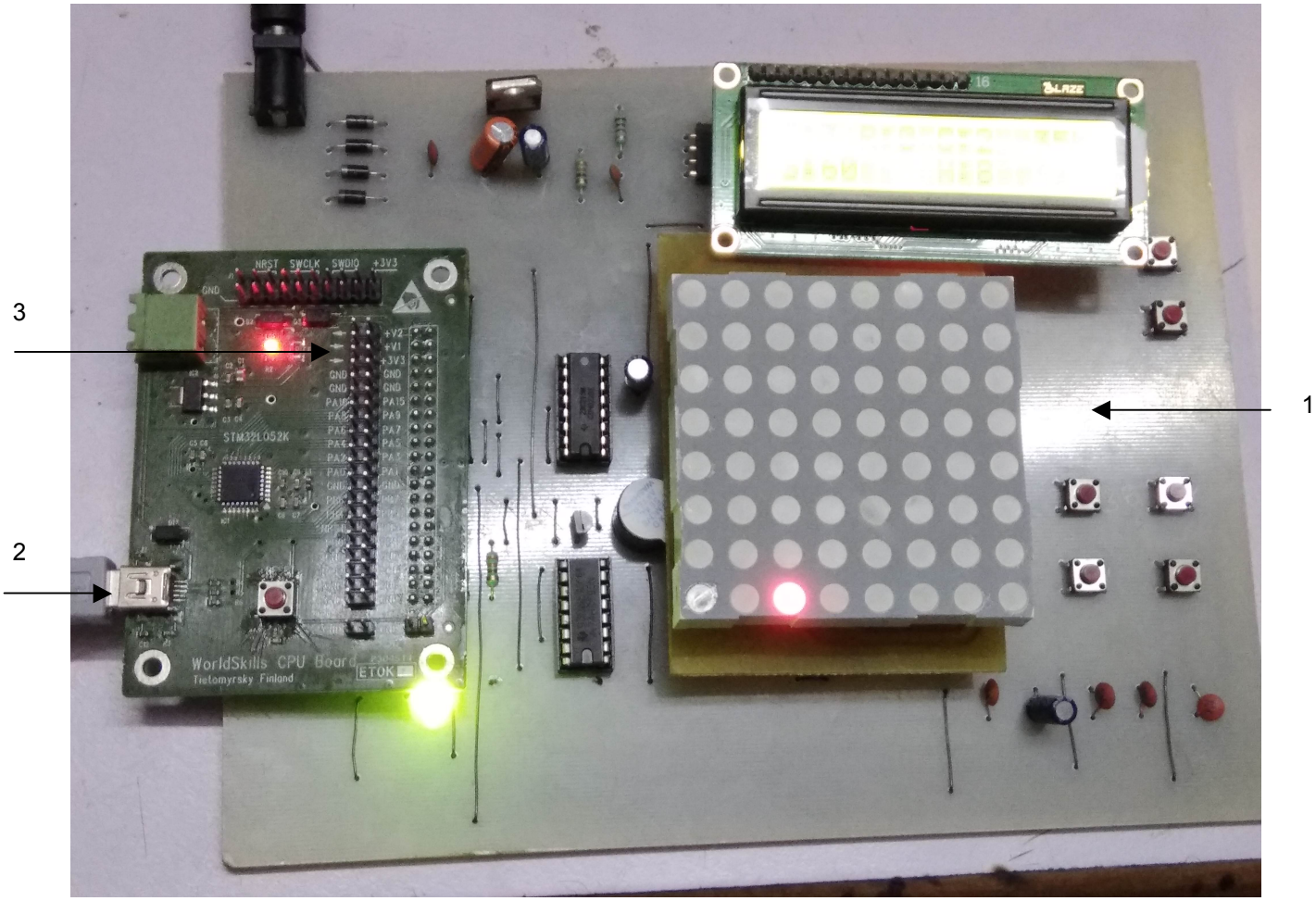


Figure 3: Programming setup

1	Task Board
2	USB
3	World Skills CPU Board

OVERVIEW OF PROGRAMMING ENVIRONMENT

- **STM32L052K CPU BOARD**

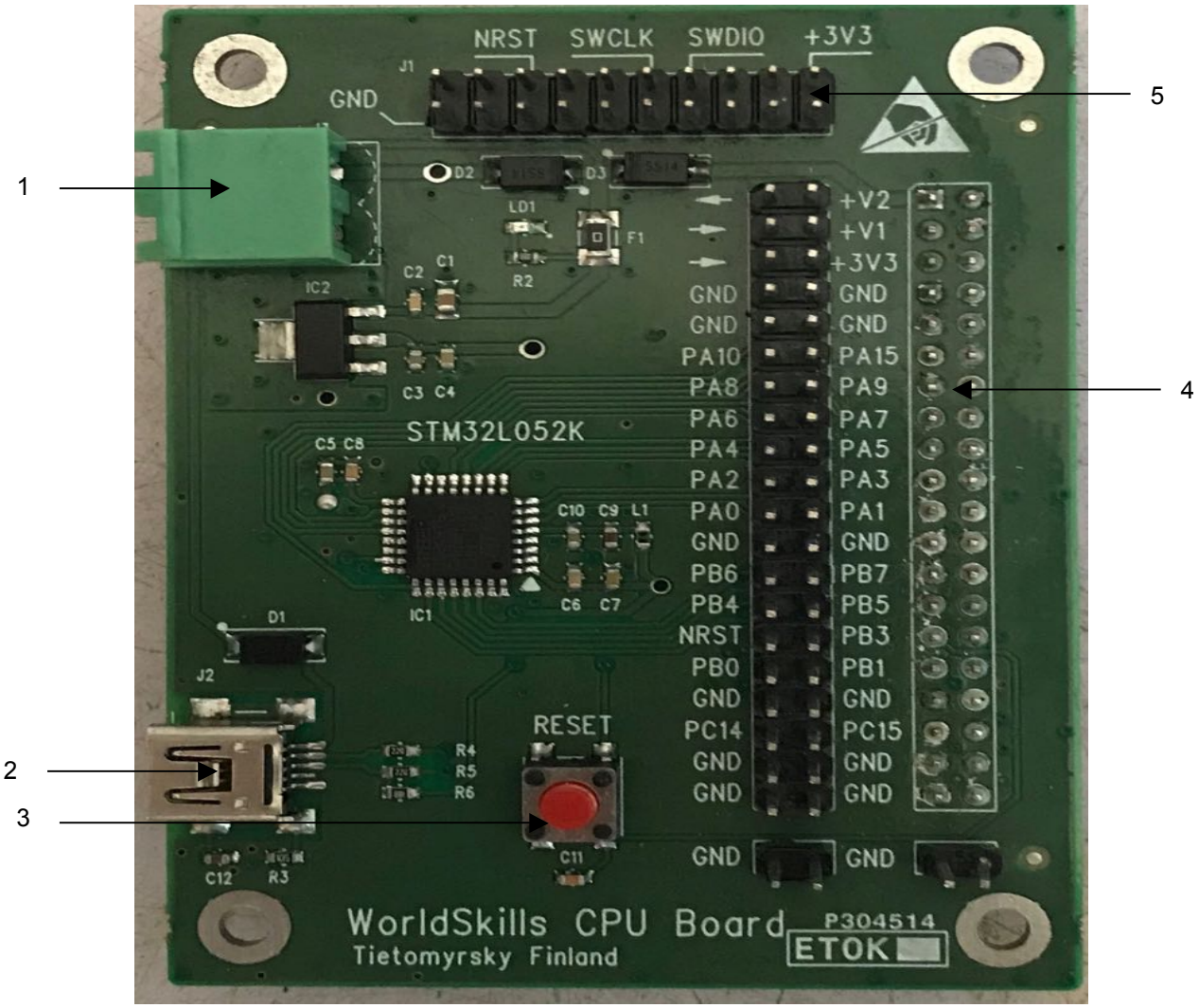


Figure 4: CPU Board

1	12 V DC
2	USB Connector
3	Reset Switch
4	CPU Connector
5	Programming Pins

- **TASK BOARD**

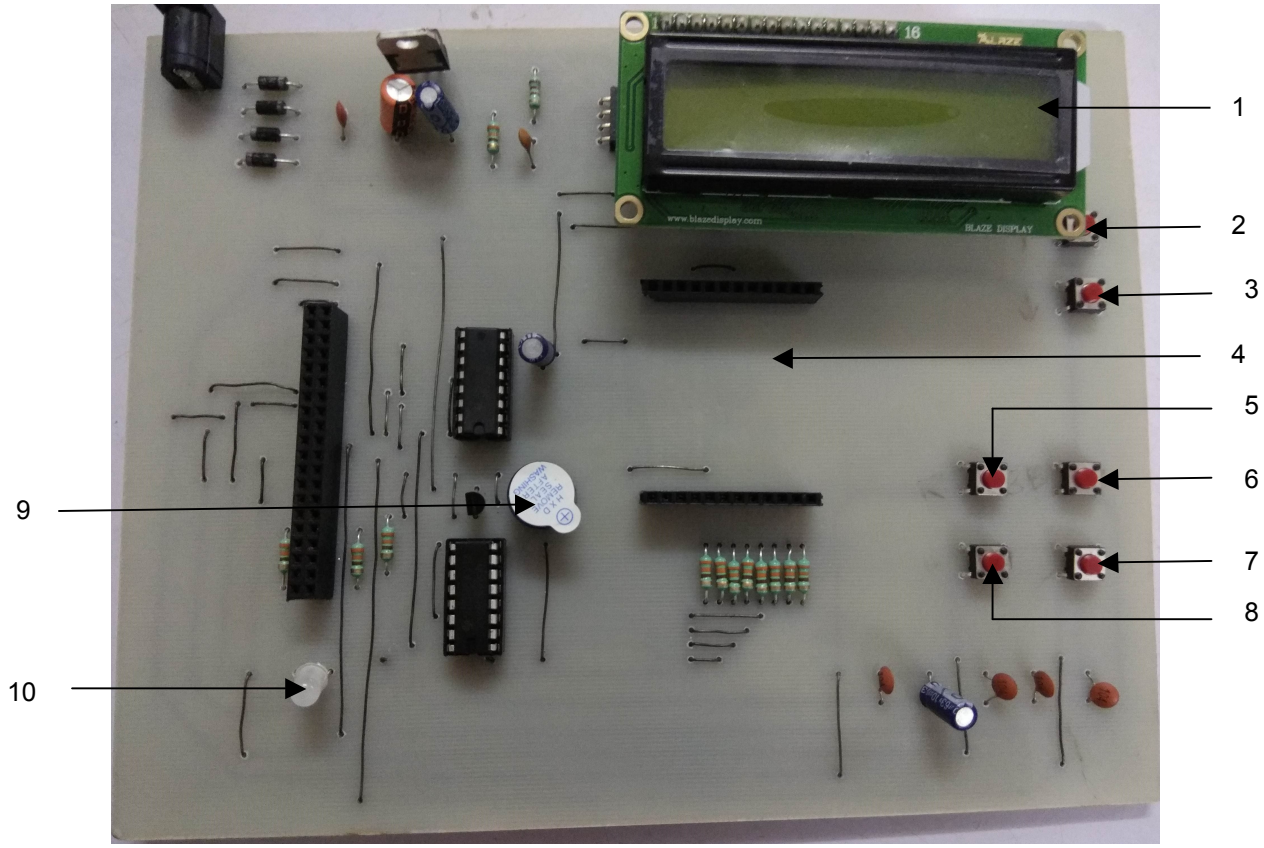


Figure 5: Task Board

1	16 X 2 LCD MODULE
2	BUTTON – UP
3	BUTTON – DOWN
4	RG DOT MATRIX
5	BUTTON – RIGHT
6	BUTTON – BACKWARD
7	BUTTON – LEFT
8	BUTTON – FORWARD
9	PIEZO – BUZZER
10	RGB LED

TABLE 2. SIGNALS BETWEEN CPU BOARD AND TASK BOARD

Conn. Pin	CPU GPIO	Data Direction	Signal Name	Notes
11	PA10	Output	COL_LOAD	Load shift register data to outputs
12	PA15	Output	COL_DATA	Shift register serial data input
13	PA8	Output	COL_RST	Reset shift register
14	PA9	Output	COL_CLK	Shift register clock input
15	PA6	Output	ROW_CLK	Increment counter value (count up)
16	PA7	Output	ROW_RST	Reset counter
17	PA4	Input	SW5	Button DOWN
18	PA5	Input	SW6	Button UP
19	PA2	Input	SW3	Button RIGHT
20	PA3	Input	SW4	Button BACKWARD
21	PA0	Input	SW1	Button FORWARD
22	PA1	Input	SW2	Button LEFT
28	PB5	Output	PZ	Piezo summer
30	PB3	Output	red_led	LED red
31	PB0	Output	blue_led	LED blue
32	PB1	Output	green_led	LED green

In table 2, there are all labyrinth board driving signals. Use signal names from table 2 with **HAL_GPIO_WritePin** and **HAL_GPIO_ReadPin** functions.

Here are examples how to use HAL_GPIO_WritePin function:

```
HAL_GPIO_WritePin(CTR_DN_GPIO_Port, CTR_DN_Pin,   GPIO_PIN_SET);

HAL_GPIO_WritePin(CTR_RST_GPIO_Port, CTR_RST_Pin, GPIO_PIN_RESET);
```

Function — Port — Pin — SET (1) or RESET (0)

Next is an example how read with HAL_GPIO_ReadPin function:

```
but = HAL_GPIO_ReadPin(BUT_READ_GPIO_Port, BUT_READ_Pin);
```

PROGRAMMING TASKS

This programming task will be done in two parts. Before you begin you will be able to see a finished presentation of the task.

You will then get a project file template. In this file all CPU Hardware Abstraction Layer (HAL) and General-Purpose Inputs/Outputs (GPIO) initializations have been done. There are also parts of code where you can find examples on how to use some library functions.

You get a precompiled HEX file to test the functionality of the program and after testing it you will hopefully understand it. Your task is to develop a similar program for the game device. The development will be done in multiple phases

The First part is a hardware dependent phase. Once you get the phase done, call a judge to check that the function performs as asked. Do not proceed to the second part until you have permission from the judge.

When you are done developing your program, return the source file, where you have written your name and other details.

In the second part you will get a new project file. In this project the previous hardware tasks in part-1 have been completed for you.

For both sections, you also receive demonstration files and projects. You can use these projects/files for downloading and viewing/testing for the required functionality of the tasks.

You can either load the demonstration .hex file using ST-LINK Utility

> Use ‘File->Open file.’ to load the ‘.hex’ file, and ‘Target->Program & Verify.’ to write the demonstration file to the task board.

Alternatively you can load the demonstration projects into Keil and download the demonstration code onto the task board.

Open the project file in Keil using 'Project-> Open Project.' and press the load icon 

, or use 'Flash->Download', or press F8.

PROGRAMMING PHASE 1

Load up the phase1 project in Keil

Phase 1.1: Push buttons and piezo-buzzer

With reference to the data sheets and schematics, complete the following function.

Table 3

BUTTON SW6(UP) or 'w' key of PC Keyboard	PIEZO-BUZZER ACTION	ON-TIME	OFF-TIME
Pressed	Beep twice	50ms	50ms

Table 4

FUNCTIONS	WORKING
static int8_t CDC_Receive_FS (uint8_t* Buf, uint32_t *Len)	This function is used to receive data from usb device. It is present in the usb_cdc_if.c file
void beep(void)	This function is used to generate beep sound

Phase 1.2: Turn on led's on the display

Add a test in the main loop to control the column drivers of 8 x 8 dot matrix, using the functions shown in Table 5.

Table 5

FUNCTIONS	WORKING
void shift_register_reset(void)	This function resets all the column driver shift registers.
void shift_register_clk_pulse(void)	This function applies one clock pulse to the column shift registers.
void shift_register_load_pulse(void)	This function loads the shift register data to its storage register outputs.

void shift_register_write(uint8_t data)	This function shifts in 8 bit data to the column shift register. This function should use the three previous shift register functions.

Test the function with these 8 bit values:

shift_register_write(0xFE); **// 1111 1110**

shift_register_write(0xEF); **// 1110 1111**

shift_register_write(0xCC); **// 1100 1100**

Phase 1.3 : Turn on any led on the display in a given row

Add a test in the main loop to control the row drivers of 8 x 8 dot matrix, using the functions shown in Table 6.

Table 6

FUNCTIONS	WORKING
void row_counter_reset(void)	This function reset row counter. After reset row 7 on the display is the active row.
void row_counter_clk_pulse(void)	This function applies one clock pulse to the row counter.
void set_row(uint8_t row)	This function selects the row that is given as a parameter.

Test your code to set a green dot to row 0 column 0 on the display.

Phase 1.4 : Turn on a led on the display given row, column and color

Add a test in the main loop to control the row drivers of 8 x 8 dot matrix, using the functions shown in Table 7.

Table 7

FUNCTIONS	WORKING
void clr_all_dots(void)	This function turns off all the led's on the display.
void set_dot(uint8_t row, uint8_t col, color_t color)	This function turns on a led on the display at given row, column and color.

Test your code to set violet led on row 3, column 4.

Phase 1.5 : Move led position and change color

Modify your existing program so that the led can "move" on the display with push buttons OR with **q**(up), **e**(down), **w**(forward), **s**(backward), **a**(Left) & **d**(Right) keys of PC keyboard via virtual COM terminal. If you try to move a led over the display edge, the program should beep twice. You must use switch de-bouncing method. This means if the button is pressed down, the led position will change immediately. If you keep the button pressed down the led position does not change. On releasing the button, led position will not change.

After you have made this function and tested the code of phase 1, call a judge and show it to them. You will be marked for phase 1 at this time.

You can move onto phase 2 without completing phase 1, but you cannot return to complete phase 1 later, and will only receive marks for phase 1 if shown to a judge at this stage.

PROGRAMMING PHASE 2

You can see presentation of finished program by loading the demonstration version of phase 2. You can view this at any time, but please use the correct Project for phase2 for this task.

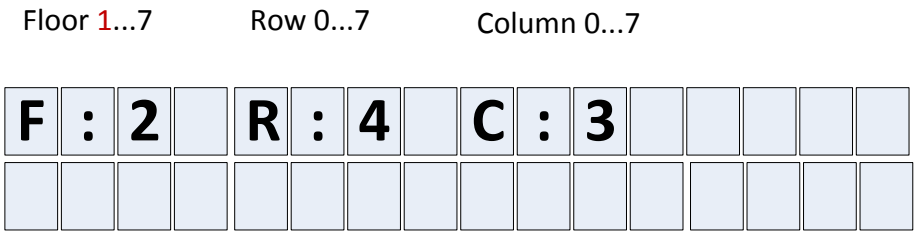
PROGRAM REQUIREMENTS

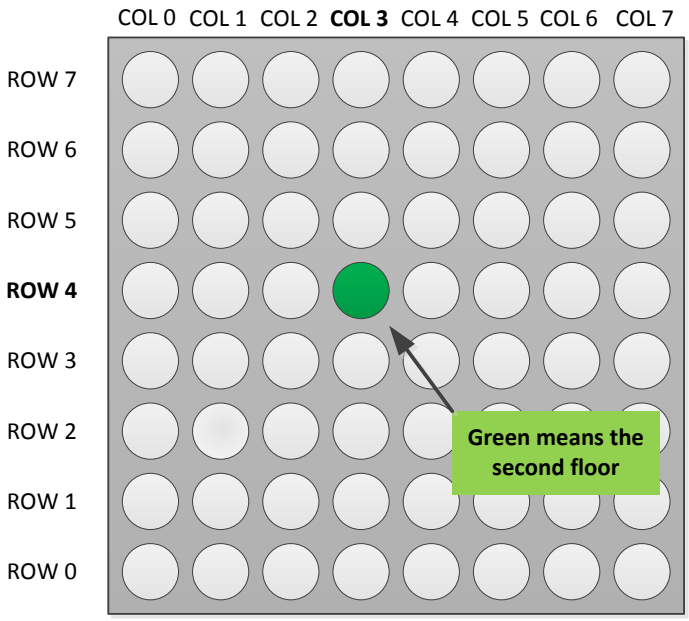
In the second phase you need to display the player's position, number of hits and number of steps.

Load part 2 project in Keil.

Phase 2.1 : Position display

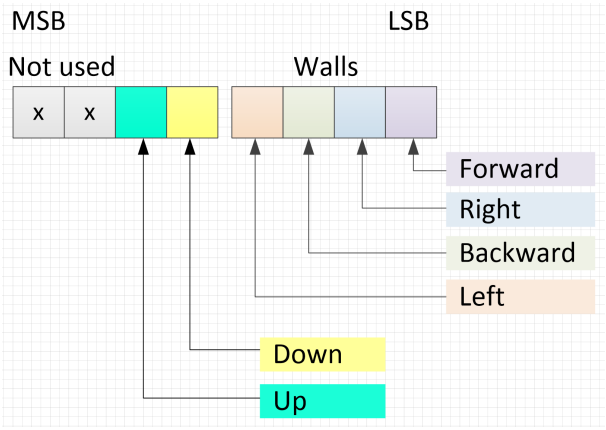
Modify your program so it displays floor number (1…7), row (0…7) and column (0…7) of your position in the labyrinth.





Phase 2.2 : Labyrinth walls

The project file you have includes a 3 dimensional array. Each location in the labyrinth has its own cell in the array. The six lower bits of the cell indicate which way you can't go. They correspond to the walls, floor and roof of the cell.



Bit value 0 = you can go this direction

Bit value 1 = you can't go this direction

Examples:

If the value of the cell is 0011 1101 = 0x3D, you can go only to the right direction.

If the value of the cell is 0011 0001 = 0x31, you can go from this cell left, backward and right.

If the value of the cell is 0010 1110 = 0x2E, you can go from this cell floor down and forward.

The code below shows the organization of the labyrinth floor arrays.

```
//                        F  R  C                        F = floor, R = row, C = colum
const uint8_t labyrinth[7][8][8] = {

//  col 0 col1  col2  col3  col4  col5  col6  col 7    row 0 of array is
//                                                     row 7 of labyrinth!

    0x39, 0x31, 0x37, 0x39, 0x35, 0x37, 0x39, 0x17, // row 7  FLOOR 1 (RED)
    0x3A, 0x3E, 0x39, 0x34, 0x33, 0x3B, 0x3C, 0x33, // row 6
    0x38, 0x31, 0x32, 0x1B, 0x38, 0x32, 0x3B, 0x3A, // row 5
    0x3A, 0x3A, 0x3A, 0x3A, 0x3A, 0x3E, 0x3C, 0x32, // row 4
    0x3E, 0x3E, 0x38, 0x32, 0x3A, 0x39, 0x31, 0x36, // row 3
    0x39, 0x31, 0x36, 0x3E, 0x3A, 0x3E, 0x3A, 0x3B, // row 2
    0x3A, 0x3A, 0x39, 0x33, 0x38, 0x35, 0x32, 0x3A, // row 1
    0x3E, 0x3C, 0x36, 0x3E, 0x3C, 0x37, 0x3C, 0x36, // row 0

    0x39, 0x31, 0x37, 0x39, 0x35, 0x37, 0x39, 0x27, // row 7  FLOOR 2 (GREEN)
    0x3A, 0x3E, 0x39, 0x34, 0x33, 0x3B, 0x3A, 0x3B,
    0x38, 0x31, 0x32, 0x2B, 0x38, 0x32, 0x3A, 0x3A,
    0x3A, 0x3A, 0x3A, 0x3A, 0x3A, 0x3E, 0x3C, 0x32,
    0x3E, 0x3E, 0x3A, 0x3A, 0x3A, 0x39, 0x31, 0x36,
    0x39, 0x31, 0x36, 0x3E, 0x3A, 0x3E, 0x3A, 0x3B,
    0x3E, 0x3A, 0x39, 0x33, 0x38, 0x35, 0x32, 0x3A,
    0x1D, 0x34, 0x36, 0x3E, 0x3C, 0x37, 0x3C, 0x36,

    0x1D, 0x33, 0x39, 0x35, 0x35, 0x35, 0x35, 0x33, // row 7  FLOOR 3 (YELLOW)
    0x3B, 0x3A, 0x3A, 0x39, 0x31, 0x35, 0x33, 0x3A,
```

These images are from Excel file which designed the labyrinth.

DATA

0x39,	0x31,	0x37,	0x39,	0x35,	0x37,	0x39,	0x17,
0x3A	0x3E,	0x39,	0x34,	0x33,	0x3B	0x3C	0x33,
0x38,	0x31,	0x32,	0x1B,	0x38,	0x32,	0x3B	0x3A,
0x3A	0x3A	0x3A	0x3A	0x3A	0x3E,	0x3C	0x32,
0x3E,	0x3E,	0x38,	0x32,	0x3A	0x39,	0x31,	0x36,
0x39,	0x31,	0x36,	0x3E,	0x3A	0x3E,	0x3A	0x3B,
0x3A	0x3A	0x39,	0x33,	0x38,	0x35,	0x32,	0x3A,
0x3E,	0x3C	0x36,	0x3E,	0x3C	0x37,	0x3C	0x36,
0x39,	0x31,	0x37,	0x39,	0x35,	0x37,	0x39,	0x27,
0x3A	0x3E,	0x39,	0x34,	0x33,	0x3B	0x3A	0x3B,
0x38,	0x31,	0x32,	0x2B	0x38,	0x32,	0x3A	0x3A,
0x3A	0x3A	0x3A	0x3A	0x3A	0x3E,	0x3C	0x32,
0x3E,	0x3E,	0x3A	0x3A	0x3A	0x39,	0x31,	0x36,
0x39,	0x31,	0x36,	0x3E,	0x3A	0x3E,	0x3A	0x3B,
0x3E,	0x3A	0x39,	0x33,	0x38,	0x35,	0x32,	0x3A,
0x1D,	0x34,	0x36,	0x3E,	0x3C	0x37,	0x3C	0x36,
0x1D,	0x33,	0x39,	0x35,	0x35,	0x35,	0x35,	0x33,
0x3B	0x3A	0x3A	0x39,	0x31,	0x35,	0x33,	0x3A,
0x38,	0x32,	0x3A	0x3E,	0x3A	0x39,	0x36,	0x3A,
0x3E,	0x3A	0x3C	0x17,	0x3A	0x3C	0x37,	0x3A,
0x39,	0x34,	0x31,	0x35,	0x30,	0x35,	0x35,	0x32,
0x3A	0x3D	0x30,	0x33,	0x3A	0x39,	0x37,	0x3A,
0x3C	0x37,	0x3E,	0x3E,	0x3A	0x3C	0x35,	0x36,
0x2D	0x35,	0x35,	0x35,	0x34,	0x35,	0x35,	0x37,
0x2D	0x33,	0x39,	0x35,	0x35,	0x35,	0x35,	0x33,
0x3B	0x3A	0x38,	0x35,	0x37,	0x39,	0x33,	0x3A,
0x38,	0x32,	0x3A	0x3D	0x33,	0x3A	0x3A	0x3A,
0x3E,	0x3A	0x38,	0x25,	0x36,	0x3E,	0x3A	0x3A,
0x39,	0x36,	0x3A	0x39,	0x35,	0x31,	0x30,	0x32,
0x3A	0x3D	0x32,	0x3A	0x3B	0x3E,	0x3A	0x3A,
0x3C	0x17,	0x3E,	0x3E,	0x38,	0x35,	0x36,	0x3E,
0x3D	0x35,	0x35,	0x35,	0x34,	0x35,	0x35,	0x17,
0x1D,	0x33,	0x39,	0x37,	0x3D	0x35,	0x35,	0x33,
0x3B	0x3A	0x3A	0x39,	0x33,	0x39,	0x35,	0x32,
0x38,	0x32,	0x3C	0x36,	0x3A	0x3C	0x37,	0x3A,
0x3E,	0x3C	0x33,	0x1D,	0x34,	0x35,	0x33,	0x3A,
0x39,	0x37,	0x38,	0x35,	0x35,	0x37,	0x38,	0x36,
0x3A	0x3D	0x34,	0x35,	0x31,	0x35,	0x36,	0x3B,
0x38,	0x25,	0x35,	0x37,	0x38,	0x35,	0x35,	0x36,
0x3C	0x35,	0x35,	0x37,	0x3C	0x35,	0x35,	0x27,
0x2D	0x33,	0x39,	0x35,	0x35,	0x31,	0x35,	0x33,
0x3B	0x3A	0x38,	0x35,	0x37,	0x3A	0x3B	0x3E,
0x38,	0x32,	0x3A	0x3D	0x33,	0x3C	0x34,	0x33,
0x3E,	0x3A	0x38,	0x25,	0x36,	0x39,	0x37,	0x3A,
0x39,	0x36,	0x3A	0x3D	0x33,	0x38,	0x31,	0x32,
0x3A	0x3D	0x30,	0x35,	0x36,	0x3E,	0x3A	0x3A,
0x3C	0x37,	0x3E,	0x3D	0x31,	0x35,	0x36,	0x3E,
0x1D,	0x35,	0x35,	0x35,	0x34,	0x35,	0x35,	0x37,
0x3D	0x33,	0x3D	0x35,	0x31,	0x31,	0x35,	0x37,
0x3B	0x38,	0x35,	0x35,	0x32,	0x3A	0x3B	0x3B,
0x38,	0x32,	0x39,	0x35,	0x32,	0x3C	0x34,	0x32,
0x3E,	0x3A	0x38,	0x37,	0x3A	0x39,	0x37,	0x3A,
0x39,	0x36,	0x3A	0x3D	0x32,	0x38,	0x35,	0x32,
0x3A	0x3D	0x30,	0x37,	0x3A	0x3E,	0x3B	0x3A,
0x3C	0x37,	0x3E,	0x3D	0x30,	0x35,	0x36,	0x3E,
0x2D	0x35,	0x35,	0x35,	0x34,	0x35,	0x35,	0x37,

Modify your existing program so that every time when you press a direction push buttons OR **q**(up), **e**(down), **w**(forward), **s**(backward), **a**(Left) & **d**(Right) keys of PC, the program checks if the movement is possible or not.

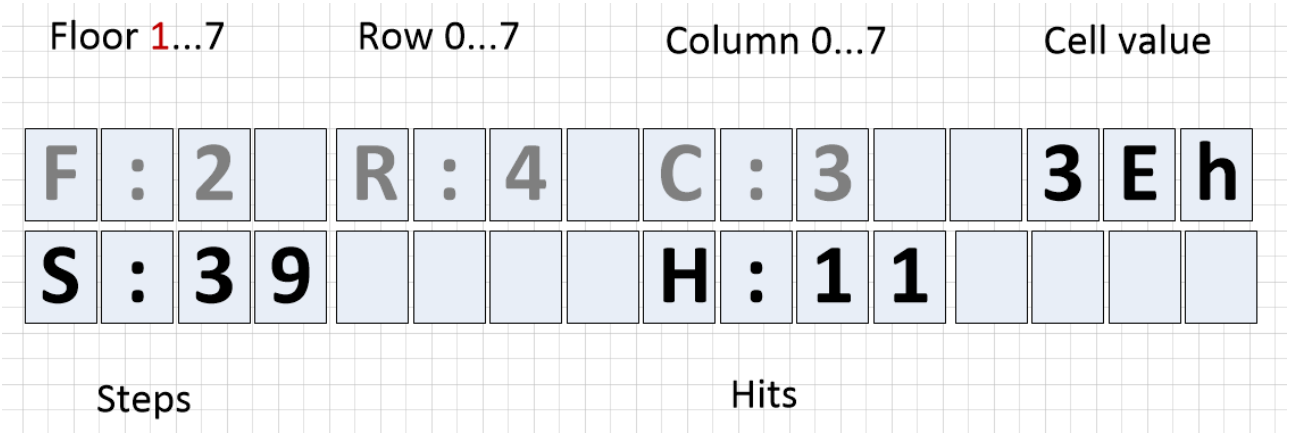
- If the movement is possible, move the LED to the new location.
- If the movement is not possible, beep twice
-

Phase 2.3 : Creating the penalty counter, step counter and cell value display

You need to add a penalty point and a step counter to your program. Every time when a player hits a wall, program increments the penalty counter. Every movement increments the step counter.

Also add "hint display" to show cell value of your current position. Display this information to the LCD display as picture below.

At this stage, every time on press of up or down buttons OR **q** or **e** keys of PC keyboard, the status of Floors, Rows, Columns, Cell Value, Steps and Hits has shown on LCD should be as it is there on virtual COM terminal via USB as COM interface.



Phase 2.4: Short cut to upper floor

Add to your program short cut to floor 7. You can use **j** button of your computer to jump floor 7 to any position you chose.

This helps you to test the labyrinth.

Phase 2.5 : Game over and new game

Add to your program the “Game Over effect”. The Game Over effect should reward for finishing the game. Be as interesting and creative as you can. You can use piezo and/or LCD display in your show.