

# Slot Machine

Biplav Karna

2020



# Introduction

Slot machine is a game device easily available at every game parlour and casino. Figure 1 shows a slot machine. It has 3 spinning wheels, lever to start the spin, place to insert coin, display to show progress. It has some lighting effects and sound effects.



Figure 1: General Slot Machine, PHOTO SRC: pixabay.com

This can be mimicked with avr 2560 with LCD screen, buttons, leds and speaker/buzzer.



# Design

Slot machine is implemented using below components

- MEGA 2560 board
- 16 \* 2 LCD
- 10 LEDs
- 4 \* 4 Keypad (using only 2 button)
- supporting connections and components

The circuit diagram of the implementation is shown in the figure 2. PA0-PA7 are used for data bus, and PG0-PG2 are used for control bus for LCD interfacing. PB0-PB7 are used for LEDs. PD1-PD0 are connected to two buttons for spin and bet functions. PD1-PD0 are used as external interrupts.

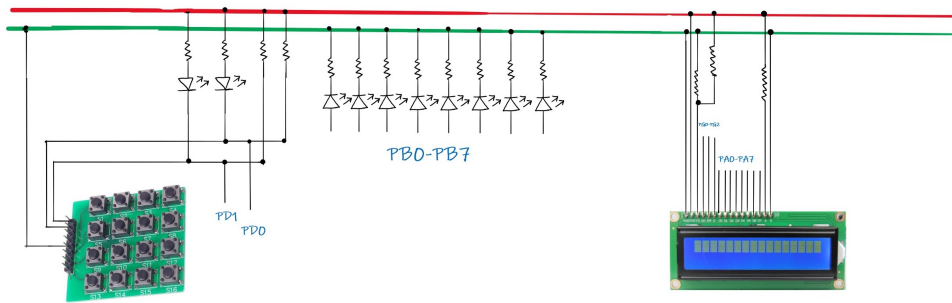


Figure 2: Circuit Diagram

The life-size slot machine's spin and bet levers are realized by buttons. The life-size screen is realized by LCD. The LCD is divided into four sections:- bet section, balance section, winning amount section and wheel section. Figure 3 shows the division of section on LCD. The lighting effect is realized with 8 leds. The initial balance is set to 2000. Default bet value is 1. The three wheels of slot machine is represented by 3 led characters in the wheel section of LCD. Spin button is given higher priority interrupt then the bet button. Timer 1 is used to check the activity time out. Activity time out is set to 15 seconds.



Figure 3: LCD Display

Timing requirements for the system are

- the response on button pressed should reflect on LCD within 1 sec.
- the wheel spinning should not be so fast that the change of symbols on the screen is not observable.
- the wheel spinning should not be so slow that player gets enough time to stop by seeing symbol at screen.

#### Game Logic:

- pressing bet button increases bet
- pressing spin button toggles the spin functionality
- when spin is stopped and the wheels match the reward pattern, the balance is increased with reward
- when spin is stopped and the wheels don't match the reward pattern, the bet value is deducted from balance
- player wins the game, if the balance is reached to max value 10,000
- player loses the game, if the balance becomes zero.

#### Reward Patterns:

- $\pi_{sym\ sym} = bet * 5$

- $$$$ = bet * 10$
- $YYY = bet * 20$
- $### = bet * 30$
- $\Sigma\Sigma\Sigma = bet * 50$
- $\pi\pi\pi = bet * 100$





# Testing

Black box and white box testing is done for the project. Codes were modified to replicate the desired testing scenarios. The list of test cases are listed below in tabular form.

Testing Id	Test Case	Method	Results and Inference
1	Spin Button Pressed Response <ul style="list-style-type: none"> <li>Spin Start on pressing</li> <li>Spin Stop on pressing</li> </ul>	Manual black box testing. Checked update of spin section on LCD	Passed
2	Spin button should not work immediately after spin is stopped. Spin button should be activated only after the winning result and balance are updated on screen.	Added delays in the code to verify	Passed
3	Bet button pressing should increase bet value. After reaching maximum value, the value should be the minimum value.	Pressing bet button and checking the updated values on the LCD	Passed
4	Bet button should be inactive when spin is on. The bet value should not change when spinning is on	Pressing bet button while spin is on.	Passed
5	Balance should increase by proper reward value and decrease by bet value.	Modified code with less permutation to check proper mapping of reward amount of the pattern.	Passed
6	The balance should not underrun minimal value of 0 and should not overrun maximum value of 10000	Modified code with large reward values to check ceil overrun. Modified code with large bet and zero reward to check underrun.	Passed
7	Winning value should match specified value matching with the pattern.	Modified code to slow turn the spin, so that it can be stopped at desired pattern.	Passed
8	Timing bench marking between spin stop and update of LCD with wheel position, balance and winning reward.	Adding timers in code to get time difference. Activating timer in ISR of bet button and stopping the timer after update of LCD.	To be determined

The Worst Case Execution Time WCET can be considered for the time of response when the spin is stopped and the values get updated. Considering 1Mz clock speed,  $37 \mu s$  write and command time for LCD, WCET can be calculated from code walkthrough. In function `SM_SpinWheel()` in file `SlotMachine.c`, a while loop runs when spin is on, i.e. **`while(SPIN_ON == spinReels)`**. It is considered the main routine is exactly at this point when ISR routine for spin button is pressed. The longest code path till the LCD update occurs give the WCET. No reward pattern is considered for the longest code traversal. The calculated WCET rounds off to 104 ms.



## Source Listing

**Config.h**

```

1  /*
2   * Config.h
3   *
4   * Created: 21/09/2020 17:11:31
5   * Author: 230712
6   */
7
8
9  #ifndef CONFIG_H_
10 #define CONFIG_H_
11
12 #include <avr/io.h>
13 #define F_CPU 1000000UL
14 #include <util/delay.h>
15 #define LCD_IO_CMD_PORT_LETTER G
16 #define LCD_IO_CMD_PORT_RS 0
17 #define LCD_IO_CMD_PORT_RW 1
18 #define LCD_IO_CMD_PORT_EN 2
19 #define LCD_IO_DATA_PORT_LETTER A
20 #define KEYPAD_PORT_LETTER D
21 #define LIGHTS_PORT_LETTER B
22
23 #define KEY_DEBOUNCE_WAIT 100
24
25 #define INPUT_MODE 0x00
26 #define OUTPUT_MODE 0xFF
27
28 #define ALL_BITS 0xFF
29 #define CLEAR_BITS 0x00
30
31
32 #define PORT_LETTER_TO_DD(PORT_LETTER)  _PORT_LETTER_TO_DD(
33     PORT_LETTER)
34 #define _PORT_LETTER_TO_DD(PORT_LETTER)  DDR ## PORT_LETTER
35
36 #define PORT_LETTER_TO_PORT(PORT_LETTER)
37     _PORT_LETTER_TO_PORT(PORT_LETTER)
38 #define _PORT_LETTER_TO_PORT(PORT_LETTER)  PORT ##
39     PORT_LETTER
40
41
42 #define PORT_LETTER_TO_PIN(PORT_LETTER)  _PORT_LETTER_TO_PIN(
43     PORT_LETTER)
44 #define _PORT_LETTER_TO_PIN(PORT_LETTER)  PIN ## PORT_LETTER
45
46
47  /*
48   * VERY IMPORTANT
49   * To do a 16-bit write, the high byte must be written
50   * before the low byte.
51   * For a 16-bit read, the low byte must be read before the
52   * high byte.
53   */

```

```
47      It is important to notice that accessing 16-bit registers  
      are atomic operations.  
48      If an interrupt occurs between the two instructions  
      accessing the 16-bit register,  
49      and the interrupt code updates the temporary register by  
      accessing the same or any  
50      other of the 16-bit Timer Registers, then the result of  
      the access outside the  
51      interrupt will be corrupted. Therefore, when both the  
      main code and the interrupt  
52      code update the temporary register, the main code must  
      disable the interrupts during  
53      the 16-bit access.  
54      */  
55  
56  
57  #define OPER_16_BIT_START  { unsigned char _oper_16_bit_sreg  
      = SREG; cli();  
58  
59  #define OPER_16_BIT_END    SREG = _oper_16_bit_sreg; }  
60  
61  #endif /* CONFIG_H */
```

**LcdLibrary.h**

```

1  #ifndef _LCD_LIBRARY_H
2  #define _LCD_LIBRARY_H
3
4
5  /*
6
7
8  This library is for QC1602A LCD
9  pin configuration:
10 1 : vss
11 2 : vdd
12 3 : v0
13 4 : RS
14 5 : R/W
15 6 : E
16 7-14 : D0-D7 (data bus )
17 15 : A (back light 5 v)
18 16 : K (back light 0 v)
19
20
21 PORT 1(0->5) will be connected to 1->6
22 PORT 2(0->7) will be connected to 7->14
23 */
24 #include <stdbool.h>
25 #include "config.h"
26
27 #if !defined(LCD_IO_CMD_PORT_LETTER) || !defined(
    LCD_IO_DATA_PORT_LETTER)
28 #error "LCD_IO_CMD_PORT_LETTER_and_LCD_IO_DATA_PORT_LETTER_
    not_defined"
29 #endif
30
31 #define HASH_SYMBOL 0x23
32 #define DOLLAR_SYMBOL 0x24
33 #define SUMMATION_SYMBOL 0xF6
34 #define PI_SYMBOL 0xF7
35 #define YEN_SYMBOL 0x5C
36
37 // RS = 0 and RW= 0 for command
38 #define CLEAR_CMD      0b00000001 //write time 1.52 ms
39 #define HOME_CMD       0b00000010 //write time 1.52 ms
40
41
42
43 // Display ON/OFF      0b00001DCB
44 // D =1, Display ON
45 // C=1, Cursor ON
46 // B=1, Cursor Position On
47 #define DISPLAY_CMD 0b00001000 //write time 37 us
48 #define DISPLAY_CMD_DISPLAY_ON_BIT 2
49 #define DISPLAY_CMD_CURSOR_ON_BIT 1
50 #define DISPLAY_CMD_CUR_POS_ON_BIT 0

```



```

51
52
53 //Cursor or Display Shift 0b0001(S/C)(R/L)XX
54 // set cursor shift without changing DDRAM data
55 #define DISPLAY_SHIFT_CMD 0b00010000
56 #define DISPLAY_SHIFT_CMD_SHIFT_CONTROL_BIT 3
57 #define DISPLAY_SHIFT_CMD_DIRECTION_BIT 2
58
59
60 // Function Set 001(DL)NFX
61 //DL = interface data 8/4 bits
62 // N = Number of Line
63 // 00H to 4FH in one line mode
64 // 00H to 27H in 1st line , 40H to 67H in 2nd line
65 // F = Font Size 5*11 / 5*8
66
67 #define FUNCTION_CMD 0b00100000
68 #define FUNCTION_CMD_INTERFACE_BIT 4
69 #define FUNCTION_CMD_LINE_BIT 3
70 #define FUNCTION_CMD_FONT_BIT 2
71
72
73
74 // set CGRAM address 0b01(AC5-AC0)
75 #define CGRAM_CMD 0b01000000
76
77 //set DGRAM address 0b1(AC6-AC0)
78
79 #define DGRAM_CMD 0b10000000
80
81
82
83
84 /*
85 For read RS = 1, RW=1
86 For Write RS = 1, RW=0
87 For reading busy flag RS=0,RW = 1
88 */
89
90 #define LCD_D0 0
91 #define LCD_D1 1
92 #define LCD_D2 2
93 #define LCD_D3 3
94 #define LCD_D4 4
95 #define LCD_D5 5
96 #define LCD_D6 6
97 #define LCD_D7 7
98
99 #define LCD_BF LCD_D7
100
101
102 #define LCD_ROW_1 0
103 #define LCD_ROW_2 1
104

```

```

105 #define LCD_IO_CMD_DD PORT_LETTER_TO_DD(
        LCD_IO_CMD_PORT_LETTER)
106 #define LCD_IO_CMD_PORT PORT_LETTER_TO_PORT(
        LCD_IO_CMD_PORT_LETTER)
107 #define LCD_IO_CMD_PIN PORT_LETTER_TO_PIN(
        LCD_IO_CMD_PORT_LETTER)
108 #define LCD_IO_DATA_DD PORT_LETTER_TO_DD(
        LCD_IO_DATA_PORT_LETTER)
109 #define LCD_IO_DATA_PORT PORT_LETTER_TO_PORT(
        LCD_IO_DATA_PORT_LETTER)
110 #define LCD_IO_DATA_PIN PORT_LETTER_TO_PIN(
        LCD_IO_DATA_PORT_LETTER)
111
112 extern bool gTwoLineMode;
113 /*
114 #define LCD_IO_1_DD LCD_IO_1_DD(LCD_IO_1ST_PORT_LETTER)
115 #define LCD_IO_1_DD(PORT_LETTER) LCD_IO_1_DD(PORT_LETTER)
116 #define LCD_IO_1_DD(PORT_LETTER) DDR ## PORT_LETTER
117
118 #define LCD_IO_1_PORT LCD_IO_1_PORT(LCD_IO_1ST_PORT_LETTER)
119 #define LCD_IO_1_PORT(PORT_LETTER) LCD_IO_1_PORT(
        PORT_LETTER)
120 #define LCD_IO_1_PORT(PORT_LETTER) PORT ## PORT_LETTER
121
122 #define LCD_IO_1_PIN LCD_IO_1_PIN(LCD_IO_1ST_PORT_LETTER)
123 #define LCD_IO_1_PIN(PORT_LETTER) LCD_IO_1_PIN(
        PORT_LETTER)
124 #define LCD_IO_1_PIN(PORT_LETTER) PIN ## PORT_LETTER
125
126 #define LCD_IO_2_DD LCD_IO_2_DD(LCD_IO_2ND_PORT_LETTER)
127 #define LCD_IO_2_DD(PORT_LETTER) LCD_IO_2_DD(PORT_LETTER)
128 #define LCD_IO_2_DD(PORT_LETTER) DDR ## PORT_LETTER
129
130 #define LCD_IO_2_PORT LCD_IO_2_PORT(LCD_IO_2ND_PORT_LETTER)
131 #define LCD_IO_2_PORT(PORT_LETTER) LCD_IO_2_PORT(
        PORT_LETTER)
132 #define LCD_IO_2_PORT(PORT_LETTER) PORT ## PORT_LETTER
133
134 #define LCD_IO_2_PIN LCD_IO_2_PIN(LCD_IO_2ND_PORT_LETTER)
135 #define LCD_IO_2_PIN(PORT_LETTER) LCD_IO_2_PIN(
        PORT_LETTER)
136 #define LCD_IO_2_PIN(PORT_LETTER) PIN ## PORT_LETTER
137
138 */
139
140 // #include <string.h>
141
142 /*
        *****
143
144 bool LCD_Init(bool twoLineMode, bool largeFontMode);
145 Arguments : 2
146 Inputs : 2

```

```

147  twoLineMode           :=  1(false) or 2(true) line mode
148  largeFontMode         :=  (false) = 5*8 pixels , (true) =
      5*11 pixels
149  Output : None
150  Return : bool          :=  success (true) and failure (
      false)
151
152  *****/
153  bool LCD_Init(bool twoLineMode ,
154                bool largeFontMode);
155
156
157  void LCD_Set_CMD_Port_Out(unsigned bitsToWrite);
158  void LCD_Set_CMD_Port_In(unsigned bitsToWrite);
159  unsigned LCD_Read_CMD_Port(unsigned bitsToBeRead);
160  void LCD_Write_CMD_Port(unsigned bitsToWrite , bool setReset);
161  void LCD_Write_Command(unsigned char commandValue);
162  void LCD_Write_Data(unsigned char dataValue);
163  void LCD_Wait();
164  void LCD_Enable();
165  void LCD_Disable();
166  void LCD_Write_String(char text []);
167  void LCD_Display_ON_OFF(bool displayON , bool cursorON , bool
      cursorPositionON);
168  void LCD_Clear();
169  void LCD_Home();
170  void LCD_ShiftDisplay(bool shiftDisplayON , bool
      directionRight );
171  void LCD_SetCursorPosition(unsigned char columnPosition /*0 -
      40 */, unsigned char rowPosition /*0 for top row, 1 for
      bottom row*/);
172
173
174
175  #endif

```

## KeyPad.h

```
1  #ifndef _KEY_PAD_H
2  #define _KEY_PAD_H
3
4  #include "Config.h"
5  #include <avr/interrupt.h>
6
7  #define KEYPAD_DD PORT_LETTER_TO_DD(KEYPAD_PORT_LETTER)
8  #define KEYPAD_PORT PORT_LETTER_TO_PORT(KEYPAD_PORT_LETTER)
9  #define KEYPAD_PIN PORT_LETTER_TO_PIN(KEYPAD_PORT_LETTER)
10
11 #define KEY_SPIN 0
12 #define KEY_BET 1
13 #define KEY_BET_MAX 2
14
15 extern unsigned char buttonPressed;
16 void KP_Init();
17 void KP_Enable_Spin();
18 void KP_Enable_Bet();
19 void KP_Enable_Bet_Max();
20 void KP_Disable_Spin();
21 void KP_Disable_Bet();
22 void KP_Disable_Bet_Max();
23 #endif
```

## SlotMachine.h

```

1  #ifndef _SLOT_MACHINE_H
2  #define _SLOT_MACHINE_H
3
4  #include <stdint.h>
5  #include <stdbool.h>
6
7  /*
      *****
8  MACROs defined for initial values and maximum values
9  MACROs for different sections of lcd start column and rows
      are defined
10 MACROs for fixed texts to be displayed into sections are
      defined
11 *****
      */
12
13 #define START_BALANCE 2000
14 #define MAX_BET 3
15 #define MIN_BET 1
16
17 #define SPIN_ON 1
18 #define SPIN_OFF 0
19
20 #define MAX_WIN_BALANCE 10000
21
22 #define REEL_CURSOR_ROW LCD_ROW_2
23 #define REEL_CURSOR_COL 12
24
25 #define WIN_CURSOR_ROW LCD_ROW_2
26 #define WIN_CURSOR_COL 4
27
28 #define PLAYER_BALANCE_CURSOR_ROW LCD_ROW_1
29 #define PLAYER_BALANCE_CURSOR_COL 10
30
31 #define PLAYER_BET_CURSOR_ROW LCD_ROW_1
32 #define PLAYER_BET_CURSOR_COL 4
33
34 #define REEL_TEXT_LEFT 0x7E
35 #define REEL_TEXT_LEFT_ROW LCD_ROW_2
36 #define REEL_TEXT_LEFT_COL 11
37
38 #define REEL_TEXT_RIGHT 0x7F
39 #define REEL_TEXT_RIGHT_ROW LCD_ROW_2
40 #define REEL_TEXT_RIGHT_COL 15
41
42 #define WIN_TEXT "Won: "
43 #define WIN_TEXT_ROW LCD_ROW_2
44 #define WIN_TEXT_COL 0
45
46 #define PLAYER_BALANCE_TEXT "Bal: "
47 #define PLAYER_BALANCE_TEXT_ROW LCD_ROW_1

```

```

48 #define PLAYER_BALANCE_TEXT_COL 6
49
50 #define PLAYER_BET_TEXT "Bet:"
51 #define PLAYER_BET_TEXT_ROW LCD_ROW_1
52 #define PLAYER_BET_TEXT_COL 0
53
54
55 #define IDLE_TEXT "Press_To_Start"
56 #define IDLE_TEXT_ROW LCD_ROW_1
57 #define IDLE_TEXT_COL 0
58
59 #define YOU_WON_TEXT "YOU_WON"
60 #define YOU_WON_TEXT_ROW LCD_ROW_1
61 #define YOU_WON_TEXT_COL 5
62
63 #define GAMEOVER_TEXT "GAMEOVER"
64 #define GAMEOVER_TEXT_ROW LCD_ROW_1
65 #define GAMEOVER_TEXT_COL 4
66
67 #define SPIN_DELAY 100
68 #define DISPLAY_BANNER_DELAY 10000
69
70 #define IDLE_TIME_OUT_VALUE 10
71
72 /*
       *****
73  MACROS for reward values are defined
74
75  *****
       */
76 #define DOLLAR_REWARD 10
77 #define YEN_REWARD 20
78 #define HASH_REWARD 30
79 #define SUMMATION_REWARD 50
80 #define PI_REWARD 100
81
82 #define DOUBLE_MATCH_REWARD 5
83
84 /*
       *****
85  Enum for states defined
86  *****
       */
87
88 typedef enum _gameState {SM_INIT, SM_USER_WAIT,
      SM_IDLE_TIMER_START, SM_SPIN, SM_IDLE} GameState;
89
90 /*
       *****
91  structure for player specific data
92  Bet is set to volatile as it gets updated in ISR

```

```

93  ****
94      */
95  typedef struct _playerData {
96      uint16_t Balance;
97      volatile uint16_t Bet;
98  } PlayerData;
99
100
101  /*
102      ****
103
104      Structure Game related Data
105      currently stopGame variable is not used
106      smState is updated in ISR so it is make volatile
107      ****
108      */
109
110  typedef struct _gameData {
111      PlayerData playerData;
112      uint16_t winValue;
113      volatile bool stopGame;
114      unsigned wheel1Pos;
115      unsigned wheel2Pos;
116      unsigned wheel3Pos;
117      volatile GameState smState;
118  } GameData;
119
120  volatile unsigned char spinReels;
121  extern volatile unsigned int idleTimeOut;
122  extern GameData gGameData;
123
124  /*
125      ****
126
127      Functions for game logic
128      ****
129      */
130
131  void SM_InitGameData();
132
133  void SM_ToggleSpin();
134  void SM_BetMax();
135  void SM_IncreaseBet();
136  void SM_Init();
137  uint16_t SM_WinValue();
138  void SM_UpdateLCDPlayerBet();
139  void SM_UpdateLCDPlayerBalance();
140  void SM_UpdateLCDReels();
141  void SM_UpdateLCD();
142  void SM_UpdateLCDValue();
143  void SM_UpdateLCDTexts();

```

```

140 void SM_UpdateLCDWinValue();
141 void SM_SpinWheel();
142 void SM_StopWheel();
143 void SM_Winner();
144 void SM_GameOver();
145 void SM_Idle();
146 void SM_InitialiseIdleTimer();
147 void SM_EnableIdleTimer();
148 void SM_DisableIdleTimer();
149
150 /*
    *****

151 Functions for different lighting effects
152 *****
    */
153 void SM_BetPressedLights();
154 void SM_SpinPressedLights();
155 void SM_SpinningLights();
156 void SM_UserWaitLights();
157 void SM_WinnerLights();
158 void SM_GameOverLights();
159 void SM_BetWonLights();
160 void SM_SystemBusyLights();
161 void SM_IdleLights();
162 void SM_LightsOff();
163
164
165 #endif

```



## LcdLibrary.c

```

1  #include "LcdLibrary.h"
2
3  bool gTwoLineMode = false;
4
5  bool LCD_Init(bool twoLineMode, bool largeFontMode)
6  {
7      LCD_Set_CMD_Port_Out(ALL_BITS);
8      LCD_Write_CMD_Port(ALL_BITS, false); //clear bits
9      LCD_IO_DATA_DD = OUTPUT_MODE; // configure i/o port 2 for
        output
10     LCD_IO_DATA_PORT = CLEAR_BITS; // clear i/o port 1
11
12     unsigned char Command_value = FUNCTION_CMD | (1<<
        FUNCTION_CMD_INTERFACE_BIT);
13     if(true == twoLineMode)
14     {
15         gTwoLineMode = true;
16         Command_value |= (1 << FUNCTION_CMD_LINE_BIT);
17     }
18     else
19     { // One-line mode
20         if(true == largeFontMode)
21         {
22             Command_value |= (1 <<FUNCTION_CMD_FONT_BIT);
23         }
24     }
25
26     LCD_Write_Command (Command_value);
27
28
29     return true;
30 }
31
32
33 void LCD_Set_CMD_Port_Out(unsigned bitsToWrite)
34 {
35     if (bitsToWrite == ALL_BITS)
36     {
37         LCD_IO_CMD_DD |= (1 << LCD_IO_CMD_PORT_RS ) | (1 <<
            LCD_IO_CMD_PORT_RW) | ( 1 << LCD_IO_CMD_PORT_EN);
38     }
39     else
40     {
41         // ensure other bits are not changed
42         LCD_IO_CMD_DD |= bitsToWrite & ( (1 <<
            LCD_IO_CMD_PORT_RS ) | (1 << LCD_IO_CMD_PORT_RW)
            | ( 1 << LCD_IO_CMD_PORT_EN) );
43     }
44 }
45
46 void LCD_Set_CMD_Port_In(unsigned bitsToWrite)
47 {

```

```

48
49     if (bitsToWrite == ALL_BITS)
50     {
51         LCD_IO_CMD_DD &= ~ ( (1 << LCD_IO_CMD_PORT_RS ) | (1
                    << LCD_IO_CMD_PORT_RW) | ( 1 <<
                    LCD_IO_CMD_PORT_EN) ) ;
52     }
53     else
54     {
55         // ensure other bits are not changed
56         LCD_IO_CMD_DD &= ~ (bitsToWrite & ( (1 <<
                    LCD_IO_CMD_PORT_RS ) | (1 << LCD_IO_CMD_PORT_RW)
                    | ( 1 << LCD_IO_CMD_PORT_EN) ) );
57     }
58
59 }
60
61 unsigned LCD_Read_CMD_Port(unsigned bitsToBeRead )
62 {
63     LCD_Wait();
64     unsigned readVal = 0;
65     LCD_Set_CMD_Port_In(bitsToBeRead);
66     readVal = LCD_IO_CMD_PIN & ( (1 << LCD_IO_CMD_PORT_RS ) |
        (1 << LCD_IO_CMD_PORT_RW) | ( 1 <<
        LCD_IO_CMD_PORT_EN) );
67     return readVal;
68
69 }
70
71 void LCD_Write_CMD_Port(unsigned bitsToWrite, bool setReset)
72 {
73     if (setReset)
74     {
75         LCD_IO_CMD_PORT |= bitsToWrite & ( (1 <<
                    LCD_IO_CMD_PORT_RS ) | (1 << LCD_IO_CMD_PORT_RW)
                    | ( 1 << LCD_IO_CMD_PORT_EN) );
76     }
77     else
78     {
79
80         LCD_IO_CMD_PORT &= ~ (bitsToWrite & ( (1 <<
                    LCD_IO_CMD_PORT_RS ) | (1 << LCD_IO_CMD_PORT_RW)
                    | ( 1 << LCD_IO_CMD_PORT_EN) ));
81     }
82 }
83
84 void LCD_Wait()
85 {
86     // Retain the command port(port 1) values as it is
87     // set data port (port 2) to input mode
88     //LCD_Set_CMD_Port_In( (1 << LCD_IO_CMD_PORT_RS) );
89     LCD_Write_CMD_Port( (1<< LCD_IO_CMD_PORT_RW), true);
90     LCD_Write_CMD_Port( (1<< LCD_IO_CMD_PORT_RS), false);
91

```

```

92     LCD_IO_DATA_DD = INPUT_MODE;
93     unsigned dataBus_val = (1 << LCD_BF);
94     //check if the LCD is busy
95     // read DB7, BF (busy flag) of LCD
96     while (dataBus_val & (1 << LCD_BF))
97     {
98         LCD_Enable();
99         dataBus_val = LCD_IO_DATA_PIN;
100        LCD_Disable();
101    }
102 }
103
104 void LCD_Enable()
105 {
106     //PORTG |= 0b00000100;
107     LCD_Write_CMD_Port( (1<< LCD_IO_CMD_PORT_EN), true);
108 }
109
110
111 void LCD_Disable()
112 {
113     //PORTG &= 0b11111011;
114     LCD_Write_CMD_Port( (1<< LCD_IO_CMD_PORT_EN), false);
115 }
116
117
118 void LCD_Write_Command(unsigned char commandValue)
119 {
120     LCD_Wait();
121     //LCD_Set_CMD_Port_Out(ALL_BITS);
122     LCD_Write_CMD_Port( (1 << LCD_IO_CMD_PORT_RS) , false);
123     LCD_Write_CMD_Port( (1 << LCD_IO_CMD_PORT_RW) , false);
124     LCD_Enable();
125     LCD_IO_DATA_DD = OUTPUT_MODE;
126     LCD_IO_DATA_PORT = commandValue;
127     LCD_Disable();
128 }
129
130 void LCD_Write_Data(unsigned char dataValue)
131 {
132     LCD_Wait();
133     //LCD_Set_CMD_Port_Out(ALL_BITS);
134     LCD_Write_CMD_Port( (1 << LCD_IO_CMD_PORT_RS) , true)
135     ;
136     LCD_Write_CMD_Port( (1 << LCD_IO_CMD_PORT_RW) , false
137     );
138     //PORTG |= 0b00000001; // Set Register Select HIGH for
139     //data mode (PortG bit0)
140     //PORTG &= 0b1111101;
141     LCD_Enable();
142     LCD_IO_DATA_DD = OUTPUT_MODE;
143     LCD_IO_DATA_PORT = dataValue;
144     LCD_Disable();
145 }

```

```

143
144 void LCD_Write_String(char text[])
145 {
146     unsigned idx = 0;
147     while (text[idx] != '\0')
148     {
149         LCD_Write_Data(text[idx]);
150         idx++;
151     }
152 }
153
154 void LCD_Clear()           // Clear the LCD display
155 {
156     LCD_Write_Command (CLEAR_CMD);
157     _delay_ms(2);         // takes 1.5 ms to complete, so wait
158 }
159
160 void LCD_Home()           // Set the cursor to the 'home'
                             position
161 {
162     LCD_Write_Command (HOME_CMD);
163     _delay_ms(2);         // takes 1.5 ms to complete, so wait
164 }
165
166 void LCD_Display_ON_OFF(bool displayON, bool cursorON, bool
                             cursorPositionON)
167 {
168     unsigned commandValue = DISPLAY_CMD;
169     if (displayON) commandValue |= (1 <<
        DISPLAY_CMD_DISPLAY_ON_BIT);
170     if (cursorON) commandValue |= (1 <<
        DISPLAY_CMD_CURSOR_ON_BIT);
171     if (cursorPositionON) commandValue |= (1 <<
        DISPLAY_CMD_CUR_POS_ON_BIT);
172
173     LCD_Write_Command(commandValue);
174 }
175
176
177
178 void LCD_ShiftDisplay(bool shiftDisplayON , bool
                             directionRight )
179 {
180     unsigned commandValue = DISPLAY_SHIFT_CMD;
181     if (shiftDisplayON) commandValue |= (1<<
        DISPLAY_SHIFT_CMD_SHIFT_CONTROL_BIT);
182     if (directionRight) commandValue |= (1 <<
        DISPLAY_SHIFT_CMD_DIRECTION_BIT);
183     LCD_Write_Command(commandValue);
184 }
185
186 void LCD_SetCursorPosition(unsigned char columnPosition /*0 -
        40 */, unsigned char rowPosition /*0 for top row, 1 for
        bottom row*/)

```

```
187 {
188 // Function Set 001(DL)NFX
189 //DL = interface data 8/4 bits
190 // N = Number of Line
191 // 00H to 4FH in one line mode
192 // 00H to 27H in 1st line , 40H to 67H in 2nd line
193 // F = Font Size 5*11 / 5*8
194 //set DGRAM address 0b1(AC6-AC0)
195     if (true == gTwoLineMode)
196     {
197         LCD_Write_Command(DGRAM_CMD | (0x40 * rowPosition +
198             columnPosition ));
199     }
200     else
201     {
202         LCD_Write_Command(DGRAM_CMD | columnPosition );
203     }
```

**KeyPad.c**

```
1  #ifndef _KEY_PAD_H
2  #define _KEY_PAD_H
3
4  #include "Config.h"
5  #include <avr/interrupt.h>
6
7  #define KEYPAD_DD PORT_LETTER_TO_DD(KEYPAD_PORT_LETTER)
8  #define KEYPAD_PORT PORT_LETTER_TO_PORT(KEYPAD_PORT_LETTER)
9  #define KEYPAD_PIN PORT_LETTER_TO_PIN(KEYPAD_PORT_LETTER)
10
11 #define KEY_SPIN 0
12 #define KEY_BET 1
13 #define KEY_BET_MAX 2
14
15 extern unsigned char buttonPressed;
16 void KP_Init();
17 void KP_Enable_Spin();
18 void KP_Enable_Bet();
19 void KP_Enable_Bet_Max();
20 void KP_Disable_Spin();
21 void KP_Disable_Bet();
22 void KP_Disable_Bet_Max();
23 #endif
```

## SlotMachine.c

```

1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4  #include "SlotMachine.h"
5  #include "LcdLibrary.h"
6  #include "KeyPad.h"
7
8
9  #define LIGHTS_DD PORT_LETTER_TO_DD(LIGHTS_PORT_LETTER)
10 #define LIGHTS_PORT PORT_LETTER_TO_PORT(LIGHTS_PORT_LETTER)
11 #define LIGHTS_PIN PORT_LETTER_TO_PIN(LIGHTS_PORT_LETTER)
12
13 GameData gGameData;
14 volatile unsigned int idleTimeOut = 0;
15
16 //char spinWheelValues[] = { DOLLAR_SYMBOL, '0', '1', YEN_SYMBOL
    , '3', '4', PI_SYMBOL, HASH_SYMBOL, '5', '6', SUMMATION_SYMBOL
    , '2', '7', '9', '8' };
17 char spinWheelValues[] = { DOLLAR_SYMBOL, YEN_SYMBOL, PI_SYMBOL
    , HASH_SYMBOL, SUMMATION_SYMBOL };
18 //char spinWheelValues[] = { DOLLAR_SYMBOL, YEN_SYMBOL,
    PI_SYMBOL };
19 #define spinWheelValuesLength sizeof(spinWheelValues)/sizeof(
    char)
20
21 void SM_InitGameData()
22 {
23     gGameData.smState = SM_INIT;
24     gGameData.playerData.Bet = 1;
25     gGameData.playerData.Balance = START_BALANCE;
26     gGameData.winValue = 0;
27     spinReels = SPIN_OFF;
28     gGameData.stopGame = false;
29     srand((unsigned int) rand());
30     gGameData.wheel1Pos = rand() % spinWheelValuesLength;
31     srand((unsigned int) gGameData.wheel1Pos);
32     gGameData.wheel2Pos = rand() % spinWheelValuesLength;
33     srand((unsigned int) gGameData.wheel2Pos);
34     gGameData.wheel3Pos = rand() % spinWheelValuesLength;
35 }
36
37 void SM_Init()
38 {
39     LIGHTS_DD = 0xFF;
40     SM_SystemBusyLights();
41     SM_InitGameData();
42     SM_UpdateLCD();
43     KP_Init();
44     SM_InitialiseIdleTimer();
45     KP_Enable_Spin();
46     KP_Enable_Bet();
47     gGameData.smState = SM_USER_WAIT;

```

```

48     SM_UserWaitLights();
49     //     KP_Enable_Bet_Max();
50 }
51
52
53 uint16_t SM_WinValue()
54 {
55     if( (gGameData.wheel1Pos == gGameData.wheel2Pos) && (
56         gGameData.wheel1Pos == gGameData.wheel3Pos) )
57     {
58         switch (spinWheelValues[gGameData.wheel1Pos])
59         {
60             case DOLLAR_SYMBOL:
61                 return DOLLAR_REWARD * gGameData.playerData.Bet;
62             case YEN_SYMBOL:
63                 return YEN_REWARD * gGameData.playerData.Bet;
64             case HASH_SYMBOL:
65                 return HASH_REWARD * gGameData.playerData.Bet;
66             case SUMMATION_SYMBOL:
67                 return SUMMATION_REWARD * gGameData.playerData.Bet;
68             case PI_SYMBOL:
69                 return PI_REWARD * gGameData.playerData.Bet;
70             default:
71                 return 0;
72         }
73     }
74     if ( (spinWheelValues[gGameData.wheel1Pos] == PI_SYMBOL
75         && gGameData.wheel2Pos == gGameData.wheel3Pos) ||
76         ( spinWheelValues[gGameData.wheel2Pos] ==
77             PI_SYMBOL && gGameData.wheel1Pos == gGameData
78             .wheel3Pos) ||
79         ( spinWheelValues[gGameData.wheel3Pos] ==
80             PI_SYMBOL && gGameData.wheel1Pos == gGameData
81             .wheel2Pos) )
82     {
83         return DOUBLE_MATCH_REWARD * gGameData.playerData.Bet;
84     }
85     return 0;
86 }
87
88 void SM_UpdateLCDPlayerBet()
89 {
90     char lcdString[10] = {'\0'};
91     sprintf(lcdString, "%d", gGameData.playerData.Bet);
92     LCD_SetCursorPosition( PLAYER_BET_CURSOR_COL ,
93         PLAYER_BET_CURSOR_ROW);
94     LCD_Write_String(lcdString);
95 }

```



```

90
91 void SM_UpdateLCDPlayerBalance()
92 {
93
94     char lcdString[10]= {'\0'};
95     sprintf(lcdString, "%5d", gGameData.playerData.Balance);
96     LCD_SetCursorPosition( PLAYER_BALANCE_CURSOR_COL ,
97         PLAYER_BALANCE_CURSOR_ROW);
98     LCD_Write_String(lcdString);
99 }
100 void SM_UpdateLCDReels()
101 {
102     char reelValues[4]= {'\0'};
103     reelValues[0] = spinWheelValues[gGameData.wheel1Pos];
104     reelValues[1] = spinWheelValues[gGameData.wheel2Pos];
105     reelValues[2] = spinWheelValues[gGameData.wheel3Pos];
106     LCD_SetCursorPosition( REEL_CURSOR_COL , REEL_CURSOR_ROW);
107     LCD_Write_String(reelValues);
108 }
109
110 void SM_UpdateLCDWinValue()
111 {
112
113     char lcdString[10]= {'\0'};
114     sprintf(lcdString, "%3d", gGameData.winValue);
115     LCD_SetCursorPosition( WIN_CURSOR_COL , WIN_CURSOR_ROW);
116     LCD_Write_String(lcdString);
117 }
118
119 /*
120      *****
121      SM_SpinWheel() is the main function for game play
122      - When the state is SM_USER_WAIT, idle time out is started
123      - If spin is on then bet button is disabled and ider timer is
124        disabled.
125      - three wheels spin in three different speeds
126      - when wheel is stopped the bet button is disabled until the
127        results are updated on the lcd
128      - different light effects are set based on event and
129        functionality
130      *****
131      */
132
133 void SM_SpinWheel()
134 {
135     KP_Enable_Spin();
136     int count = 1;
137     if (gGameData.smState == SM_USER_WAIT)
138     {
139         SM_EnableIdleTimer();
140         gGameData.smState = SM_IDLE_TIMER_START;
141     }

```

```

137     if(SPIN_OFF == spinReels) return;
138     SM_DisableIdleTimer();
139     gGameData.smState = SM_SPIN;
140     KP_Disable_Bet();
141     KP_Disable_Bet_Max();
142     SM_LightsOff();
143     while(SPIN_ON == spinReels)
144     {
145         KP_Enable_Spin();
146         gGameData.wheel1Pos = ( gGameData.wheel1Pos + 1 ) %
            spinWheelValuesLength;
147         if (count % 2 == 0) gGameData.wheel2Pos = ( gGameData.
            wheel2Pos + 1 ) %spinWheelValuesLength;
148         if (count % 3 == 0) gGameData.wheel3Pos = ( gGameData.
            wheel3Pos + 1 ) %spinWheelValuesLength;
149         if (count >= 3)
150         {
151             count = 1;
152         }
153         else
154         {
155             count++;
156         }
157         SM_UpdateLCDReels();
158         SM_SpinningLights();
159         _delay_ms(SPIN_DELAY);
160     }
161     KP_Disable_Spin();
162
163
164     gGameData.winValue = SM_WinValue();
165     if (gGameData.winValue == 0 )
166     {
167         if (gGameData.playerData.Balance >= gGameData.
            playerData.Bet)
168         {
169             gGameData.playerData.Balance -= gGameData.
                playerData.Bet;
170         }
171         else
172         {
173             gGameData.playerData.Balance = 0;
174         }
175         if ( 0 == gGameData.playerData.Balance)
176         {
177
178             SM_GameOver();
179             _delay_ms(DISPLAY_BANNER_DELAY);
180             SM_InitGameData();
181             LCD_Clear();
182         }
183     }
184     else
185     {

```

```

186     SM_BetWonLights();
187     gGameData.playerData.Balance += gGameData.winValue;
188     if (gGameData.playerData.Balance >= MAX_WIN_BALANCE)
189     {
190         gGameData.playerData.Balance = MAX_WIN_BALANCE;
191         SM_Winner();
192         gGameData.smState = SM_IDLE;
193         _delay_ms(DISPLAY_BANNER_DELAY);
194         SM_InitGameData();
195         LCD_Clear();
196     }
197
198 }
199 SM_UpdateLCD();
200 KP_Enable_Spin();
201 KP_Enable_Bet();
202 KP_Enable_Bet_Max();
203 gGameData.smState = SM_USER_WAIT;
204 SM_UserWaitLights();
205 }
206
207 void SM_StopWheel()
208 {
209     spinReels = SPIN_OFF;
210 }
211
212
213 void SM_UpdateLCD()
214 {
215     SM_UpdateLCDTexts();
216     SM_UpdateLCDValue();
217 }
218
219
220 void SM_UpdateLCDValue()
221 {
222     SM_UpdateLCDPlayerBet();
223     SM_UpdateLCDPlayerBalance();
224     SM_UpdateLCDReels();
225     SM_UpdateLCDWinValue();
226 }
227
228 void SM_UpdateLCDTexts()
229 {
230     LCD_SetCursorPosition( WIN_TEXT_COL , WIN_TEXT_ROW);
231     LCD_Write_String(WIN_TEXT);
232
233     LCD_SetCursorPosition( PLAYER_BET_TEXT_COL ,
234                             PLAYER_BET_TEXT_ROW);
235     LCD_Write_String(PLAYER_BET_TEXT);
236
237     LCD_SetCursorPosition( PLAYER_BALANCE_TEXT_COL ,
238                             PLAYER_BALANCE_TEXT_ROW);
239     LCD_Write_String(PLAYER_BALANCE_TEXT);

```

```
238
239     LCD_SetCursorPosition(REEL_TEXT_LEFT_COL,
240                           REEL_TEXT_LEFT_ROW);
241     LCD_Write_Data(REEL_TEXT_LEFT);
242     LCD_SetCursorPosition(REEL_TEXT_RIGHT_COL,
243                           REEL_TEXT_RIGHT_ROW);
244     LCD_Write_Data(REEL_TEXT_RIGHT);
245 }
246
247 void SM_ToggleSpin()
248 {
249     if( SPIN_ON == spinReels )
250     {
251         spinReels = SPIN_OFF;
252     }
253     else
254     {
255         spinReels = SPIN_ON;
256     }
257 }
258 void SM_BetMax()
259 {
260     gGameData.playerData.Bet = MAX_BET;
261 }
262 void SM_IncreaseBet()
263 {
264     if (gGameData.playerData.Bet == MAX_BET)
265     {
266         gGameData.playerData.Bet = MIN_BET;
267     }
268     else
269     {
270         gGameData.playerData.Bet += 1;
271     }
272 }
273
274 void SM_Idle()
275 {
276
277     LCD_Clear();
278     LCD_SetCursorPosition( IDLE_TEXT_COL , IDLE_TEXT_ROW);
279     LCD_Write_String(IDLE_TEXT);
280     gGameData.smState = SM_IDLE;
281     SM_IdleLights();
282 }
283
284 void SM_Winner()
285 {
286
287     LCD_Clear();
288     LCD_SetCursorPosition( YOU_WON_TEXT_COL ,
289                           YOU_WON_TEXT_ROW);
```

```

289     LCD_Write_String(YOU_WON_TEXT);
290     SM_WinnerLights();
291 }
292
293 void SM_GameOver()
294 {
295     LCD_Clear();
296     LCD_SetCursorPosition( GAMEOVER_TEXT_COL ,
297                             GAMEOVER_TEXT_ROW);
297     LCD_Write_String(GAMEOVER_TEXT);
298     SM_GameOverLights();
299 }
300
301 void SM_EnableIdleTimer()
302 {
303     // check last 3 bits , if all is zero then enable
304     if((TCCR1B & 0x07) == 0x00)
305     {
306         //reset timer value as it may be some intermediate
307         value when stopped
308         idleTimeOut = 0;
309         OPER_16_BIT_START
310         TCNT1H = 0b00000000; // Timer/Counter count/value
311         registers (16 bit) TCNT1H and TCNT1L
312         TCNT1L = 0b00000000;
313         OPER_16_BIT_END
314         TCCR1B = 0b00001101; // pre-scalar 1024
315     }
316 }
317
318 void SM_DisableIdleTimer()
319 {
320     TCCR1B = 0b00001000;
321 }
322
323 void SM_InitialiseIdleTimer() // Configure to
324 generate an interrupt after a 1-Second interval
325 {
326     TCCR1A = 0b00000000; // Normal port operation (OC1A,
327     OC1B, OC1C), Clear Timer on 'Compare Match' (CTC)
328     waveform mode)
329     TCCR1B = 0b00001000; // CTC waveform mode, initially
330     stopped (no clock)
331     TCCR1C = 0b00000000;
332
333     // For 1 MHz clock (with 1024 prescaler) to achieve a 1
334     second interval:
335     // Need to count 1 million clock cycles (but already
336     divided by 1024)
337     // So actually need to count to (1000000 / 1024 =) 976
338     decimal, = 3D0 Hex
339     OPER_16_BIT_START
340     OCRAH = 0x03; // Output Compare Registers (16 bit)
341     OCR1BH and OCR1BL

```

```

332     OCR1AL = 0xD0;
333     OPER_16_BIT_END
334
335
336     TIMSK1 = 0b00000010;    // bit 1 OCIE1A          Use '
                               Output Compare A Match' Interrupt, i.e. generate an
                               interrupt
337     // when the timer reaches the set value (in the OCR1A
                               register)
338 }
339
340 void SM_BetPressedLights()
341 {
342     LIGHTS_PORT = 0b00000011;
343 }
344
345 void SM_SpinPressedLights()
346 {
347     LIGHTS_PORT = 0b11000000;
348 }
349
350 void SM_SpinningLights()
351 {
352     if (LIGHTS_PIN == 0x00)
353     {
354         LIGHTS_PORT = 0b10000000;
355     }
356     else
357     {
358         LIGHTS_PORT >>=1;
359     }
360 }
361
362 void SM_UserWaitLights()
363 {
364     LIGHTS_PORT = 0b00011000;
365 }
366
367 void SM_WinnerLights()
368 {
369     LIGHTS_PORT = 0b10101010;
370 }
371 void SM_GameOverLights()
372 {
373     LIGHTS_PORT = 0b10000001;
374 }
375
376 void SM_BetWonLights()
377 {
378     LIGHTS_PORT = 0b01100110;
379 }
380
381 void SM_SystemBusyLights()
382 {

```

```
383     LIGHTS_PORT = 0xFF;
384 }
385
386 void SM_IdleLights()
387 {
388     LIGHTS_PORT = 0x00;
389 }
390
391 void SM_LightsOff()
392 {
393     LIGHTS_PORT = 0x00;
394 }
395
396
397 /*
398 ISR(INT2_vect) // Interrupt Handler for H/W INT 0
399 {
400     KP_Disable_Bet_Max();
401     SM_BetMax();
402     _delay_ms(80); // Short delay to debounce the push-
                     button
403     KP_Enable_Bet_Max();
404 }
405
406 */
```

**Main.c**

```

1  /*
2   * SlotMachineAtmelProject.c
3   *
4   * Created: 21/09/2020 16:06:22
5   * Author : 230712
6   */
7  #include "LcdLibrary.h"
8  #include "SlotMachine.h"
9
10 #include <avr/io.h>
11 #include <avr/interrupt.h>
12 #include <string.h>
13
14 int main(void)
15 {
16     LCD_Init(true, false);
17     LCD_ShiftDisplay(false, false);
18     LCD_Display_ON_OFF(true, false, false);
19     LCD_Clear();
20     LCD_Home();
21     SM_Init();
22     sei();
23     while (1)
24     {
25         SM_SpinWheel();
26     }
27 }
28
29 ISR(INT0_vect) // Interrupt Handler for H/W INT 0
30 {
31     if(gGameData.smState == SM_IDLE)
32     {
33         gGameData.smState = SM_USER_WAIT;
34         LCD_Clear();
35         SM_UpdateLCD();
36         _delay_ms(KEY_DEBOUNCE_WAIT);
37         EIFR |= 0b00000011;
38         return;
39     }
40     KP_Disable_Spin();
41     //KP_Disable_Bet();
42     SM_ToggleSpin();
43     SM_SpinPressedLights();
44     _delay_ms(KEY_DEBOUNCE_WAIT); // Short delay to
45     //debounce the push-button
46     SM_UserWaitLights();
47     EIFR |= 0b00000011;
48     idleTimeOut = 0;
49     //KP_Enable_Spin();
50     //KP_Enable_Bet();
51 }

```



```

52
53
54
55 ISR(INT1_vect) // Interrupt Handler for H/W INT 0
56 {
57     if(gGameData.smState == SM_IDLE)
58     {
59         gGameData.smState = SM_USER_WAIT;
60         LCD_Clear();
61         SM_UpdateLCD();
62         _delay_ms(KEY_DEBOUNCE_WAIT);
63         EIFR |= 0b00000011;
64         return;
65     }
66     KP_Disable_Bet();
67     KP_Disable_Spin();
68     SM_IncreaseBet();
69     SM_UpdateLCDPlayerBet();
70     SM_BetPressedLights();
71     _delay_ms(KEY_DEBOUNCE_WAIT); // Short delay to
        debounce the push-button
72     SM_UserWaitLights();
73     EIFR |= 0b00000011;
74     idleTimeOut = 0;
75     KP_Enable_Bet();
76     KP_Enable_Spin();
77 }
78
79
80
81 ISR(TIMER1_COMPA_vect) // TIMER1_CompareA_Handler (Interrupt
        Handler for Timer 1)
82 {
83     idleTimeOut++; // Increment the number of elapsed seconds
        while the timer has been running
84     if (idleTimeOut >= IDLE_TIME_OUT_VALUE)
85     {
86         idleTimeOut = 0;
87         SM_DisableIdleTimer();
88         SM_Idle();
89     }
90 }

```



## Critical evaluation and Conclusion