ECE-3130-001 — Microcomputer Systems

Final Project Report

Digital Jukebox System

Team Members:
Lucas Taylor
Shane Shankar
Steven Jamal Crocker
Nate Wiser
James Tyler
Campion Coombe

Submitted: November 26, 2025

TABLE OF CONTENTS

INTRODUCTION

The purpose of this project was to design and build a small “jukebox” system using an STM32 microcontroller. The user can choose a song on the LCD using pushbuttons, select a song, and play it through a buzzer. The system also includes a custom mode that allows a user to record a short song of his own with the keypad and play it back.

Features include

I) Created an LCD menu with seven total options, including six preset songs and a “Record Synth” mode

II) Used two pushbuttons to scroll through and select menu items

III) Played multiple melodies on the buzzer using software-generated square waves

IV) Implemented a keypad-based melody recorder with basic tempo control

V) Added support code for a 7-segment display through a shift register

Some challenges we encountered included adjusting the timing for musical notes, the limited simplicity of the buzzer sound, and combining the song-playing logic with the 7-segment display code.

Overall, this project represents a fully functional jukebox system and demonstrates several skills learned throughout the semester, including LCD interfacing, pushbutton debouncing, SysTick usage, and buzzer.

PROJECT SPECIFICATIONS AND DESCRIPTION

Functional Description

The jukebox system uses an STM32 microcontroller and an LCD to show a menu of seven options: Imperial March, Mario Theme, Jingle Bells, Harry Potter, Nokia, LG, and Record Synth. These options let the user pick one of six preset songs or record a custom melody.

The system is controlled with two pushbuttons:

I) SW4 (PB9) moves to the next item in the menu

II) SW5 (PB8) selects the highlighted item

When the user chooses one of the preset songs:

I) The LCD shows “Playing Song X” and the song’s name

II) The buzzer on PC9 plays the melody using software-timed notes

III) Pressing the select button again stops the song early

When Record Synth is selected, the system switches to a simple recording mode. First, the user chooses a tempo using keypad keys 1–3. After that, the keypad is used to enter notes:

I) Keys 1–8 enter musical notes (roughly C4–C5)

II) Key 0 enters a rest

III) The user then picks the note length (short, medium, long) with keys 1–3

The system stores up to 16 notes. Pressing the * key ends recording early. After recording is done, the system plays the custom melody back automatically.

The code also includes support for a 7-segment display using a shift register. A prepared (but commented-out) SysTick_Handler shows how a timer value could be updated and sent to the 7-segment display during each SysTick interrupt.

Block Diagram

The system is made up of several main hardware blocks connected to the STM32 microcontroller.

STM32 Microcontroller Board

The STM32 runs all program code and system.

LCD

The LCD is driven using a shift-register-style interface on pins PA5, PA10, and PB5.

It shows the menu options, instructions, and status messages during operation.

Keypad

I) Columns (PB1–PB4) are set as outputs

II) Rows (PB8–PB11) are set as inputs

The keypad is used in record mode for entering notes, note lengths, and tempo.

Pushbuttons SW4 and SW5

I) SW4 (PB9): Scroll

II) SW5 (PB8): Select

Buzzer

The buzzer is connected to PC9, and using code, create different sound frequencies.

7-Segment Display + Shift Register

The display is controlled through PA5, PB5, and PC10 using the Write_SR_7S() and Init_7Seg() functions.

It was intended to show a two-digit timer value, although this feature is commented out in the final version.

DETAILED IMPLEMENTATION

Pseudocode

MAIN PROGRAM

```
1. Turn everything on
   HAL, LED, 7-seg display, LCD, buzzer, keypad
   Set up the button interrupts for SW4 and SW5
2. Set the tempo to Medium
3. Play a quick startup beep
4. Show the main menu, starting at option 1
5. Loop forever
{
      If the SCROLL button flag is set
{
          Move to the next menu option
          Wrap back to 1 after 7
          Refresh the menu on the LCD
}
      If the SELECT button flag is set
{
          If the option is 1 through 6
              Play that song
```

        Return to the menu
}

    If the option is 7
{
        Show the tempo choices
        Wait for keypad input 1, 2, or 3
        Update the tempo setting
        Record a custom melody
        Return to the menu
}
}

Tone Generation

playTone

  Repeat for however many cycles the note needs
  Turn the buzzer ON for a short time
  Turn the buzzer OFF for a short time

This creates the square wave sound for the note.

Interrupt Handler

When a button interrupt happens

  If SELECT (SW5) was pressed
}
    Set the select flag
    If the program is in the middle of a song
    Stop the song early
{
  If SCROLL (SW4) was pressed
{
    Set the scroll flag
}

Song Playback

play_song

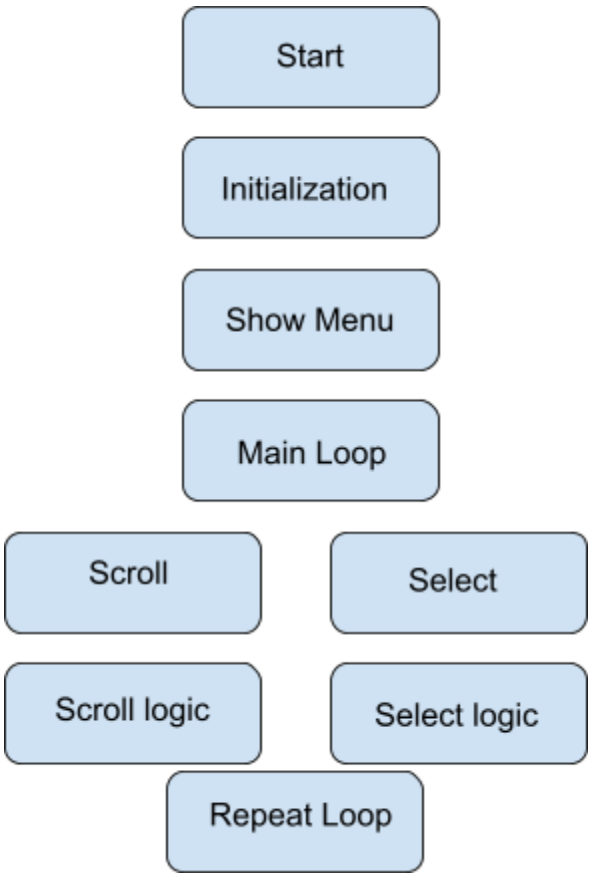
Show "Playing Song" on the LCD

Go through each note in the song
Play the tone for that note
Wait for its timing

If SELECT gets pressed again
{
Stop the song right away
Exit the loop
}
Pause when finished

Flowcharts

Main Menu Flow



Interface Design

LCD Interface

The LCD runs in 4-bit mode using a shift register and three main control pins:

I) PA5 – clock.

II) PA10 – latch.

III) PB5 – serial data.

LCD_Init handles the normal startup steps and sets the display to 4-bit, 2-line mode.

LCD functions include:

I) Write_Instr_LCD(code) – sends LCD commands.

II) Write_Char_LCD(code) – writes one character.

III) Write_String_LCD(char *) – prints a whole string.

Keypad Interface

The system uses a keypad:

I) PB1–PB4 are the column outputs

II) PB8–PB11 are the row inputs

Read_Keypad works by pulling columns high, checking which row goes high, and then scanning to figure out which key was pressed.

Key codes include:

I) 0–9 → number keys

II) 14 → “*” (used to end recording)

III) Remaining keys are used internally

Menu

Two pushbuttons control the menu:

I) SW5 (PB8) – Select

II) SW4 (PB9) – Scroll

Both use pull-downs and trigger EXTI interrupts on a rising edge.

The interrupt handler sets flags (swScrollFlag, swSelectFlag) while lock variables help prevent bouncing and double presses.

When a song is playing, pressing SW5 clears the select lock so the song can stop early.

Buzzer Interface

The buzzer is on PC9, configured as a push-pull output.

Square waves are made by toggling the pin at different speeds inside playTone, which creates different note pitches.

Microcontroller Utilization

GPIO

Port A

I) PA1 – LED0 output (used for debugging or optional blinking).

II) PA5 – Shift-register clock line (used by LCD and the 7-segment display).

III) PA10 – LCD latch/control signal.

Port B

I) PB1–PB4 – Keypad column outputs.

II) PB5 – Shift-register data line.

III) PB8, PB9 – Keypad rows and also inputs for pushbuttons SW5 and SW4 (EXTI).

IV) PB10, PB11 – Additional keypad row inputs.

Port C

I) PC9 – Buzzer output.

II) PC10 – 7-segment latch signal.

EXTI (External Interrupts)

I) EXTI line 8 → PB8 (SW5 select button)

II) EXTI line 9 → PB9 (SW4 scroll button)

Both interrupts use rising-edge triggers and are handled in EXTI9_5_IRQHandler.

Memory Usage

I) Six preset songs are stored in flash as arrays of note structures.

II) The custom melody and variables used during runtime are stored.

III) The limit of 16 custom steps helps manage memory and keeps the recording process easy for the user.

Software Description

Song Tables

Each melody is stored as an array of Note items:

I) miiMelody[] – Imperial March

II) marioMelody[] – Mario theme

III) jingleMelody[] – Jingle Bells

IV) hpMelody[] – Harry Potter theme

V) nokiaMelody[] – Nokia ringtone

VI) LGMelody[] – LG washing machine melody

Each note includes:

I) A delay-based “frequency” value

II) How long the note is held

III) How long to pause afterward

Tone Generation

playTone(int frequency, int cycles) creates square waves in software by toggling the buzzer pin. The macro NOTE_DELAY(f) converts a note frequency into an approximate delay value for timing.

Menu and Control Logic

ShowMainMenu(selection) updates the LCD with the correct menu text for options 1–7.

play_song(num) selects the correct melody array and displays the song name on the LCD while playing it.

Synth / Custom Melody Functions

keyToFrequency(key) converts keypad keys into note values (keys 1–8 are notes, key 0 is a rest).

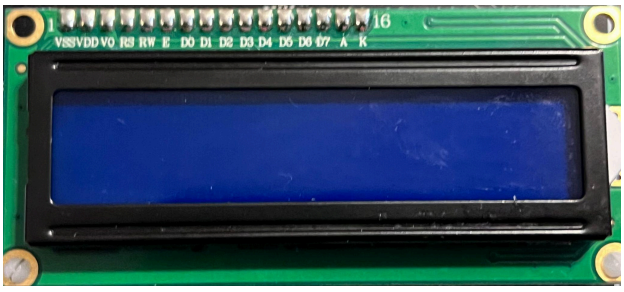
setTempoPreset(preset) sets timing values for slow, medium, or fast playback.

record_custom_melody() lets the user record up to 16 notes, each with its own length.

play_custom_melody() plays the recorded melody if at least one note exists.

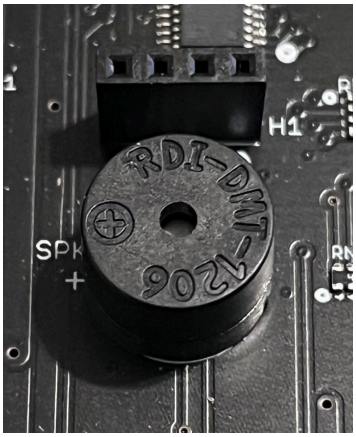
Hardware Illustrations

LCD

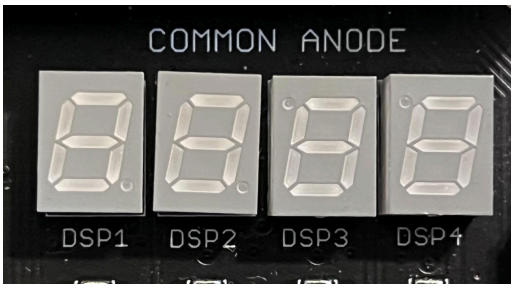


Keypad

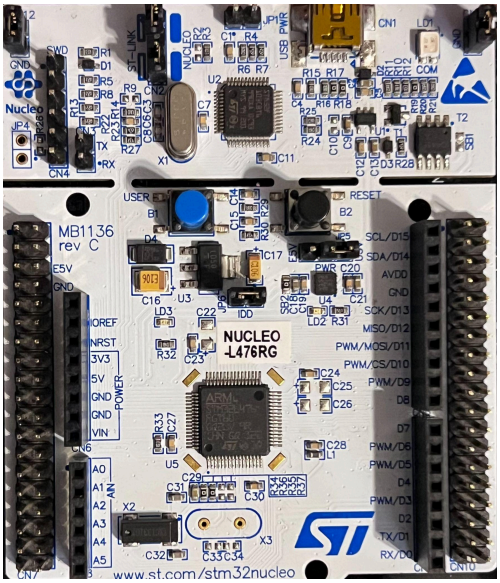
Buzzer



7-Segment Display



STM32 microcontroller



ANALYSIS AND TESTING

Functionality and Test Results

Menu Navigation

I) Verified SW4 scroll cycled through options 1–7 and wrapped correctly

II) Verified SW5 select entered the correct mode

III) Confirmed the debounce logic

Song Playback (Options 1–6)

I) Played each preset melody and checked that pitches were correct for delay-based timing
II) Verified that the rhythm and note order matched expectations
III) Tested the stop feature by pressing select during playback and confirmed the song stopped early

Record Synth

I) Tested slow, medium, and fast tempo presets and confirmed timing changes
II) Recorded multiple test melodies using keys 0–8 with different note lengths
III) Verified that pressing * ended recording early
IV) Confirmed playback followed the exact order and timing of the recorded notes

Public Health, Safety, and Welfare Considerations

I) Exposed wires and component leads on the breadboard can cause minor poke or scratch risks.

II) There is a small chance of electrical shock from touching open connections.

These issues can be prevented by placing the entire circuit inside a protective enclosure or project box.

Global, Cultural, Social, Environmental, and Economic Factors

I) Different cultures can select songs that reflect their own musical styles and traditions.
II) The STM32 board, LCD, and keypad are relatively inexpensive, but the user still needs a laptop to program and run the system.

DESCRIPTION OF THE TEAMWORK EXPERIENCE

The team held several meetings to discuss project ideas and review what each member had completed. After considering different options, we chose a project idea suggested by Jamal. Once the project direction was set, we divided the work among team members.

Jamal and Shane worked on the main LCD menu and the custom song recording system. Jamal, Lucas, James, and Nate were responsible for creating and testing the preset songs. The 7-segment display was implemented by Shane and Campion.

We shared updates through email and a group chat, where we exchanged code, reported progress, and helped each other solve problems as they came up. This regular communication helped keep the project on track.

After the system was finished, James, Nate, and Campion worked together to write the project report.

APPENDIX A – SOURCE CODE

```
/* USER CODE BEGIN Header */

/**

  ******************************************************************************

  * @file           : main.c

  * @brief          : Main program body

  ******************************************************************************

  * @attention

  *

  * Copyright (c) 2022 STMicroelectronics.

  * All rights reserved.

  *

  * This software is licensed under terms that can be found in the LICENSE file

  * in the root directory of this software component.

  * If no LICENSE file comes with this software, it is provided AS-IS.

  *

  ******************************************************************************

  */
```

```
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/

#include "main.h"



void SystemClock_Config(void);


void Init_LED0(void);

void EXTI_PB8_Init(void);

void Delay(unsigned int);

void SysTick_Initialize (uint32_t);

void SysTick_Handler(void);

void Init_7Seg(void);

void Write_SR_S7( uint8_t, uint8_t);



uint8_t tick = 1;

uint8_t time = 0;


/* ---------- Prototypes ---------- */

void Delay(unsigned int n);


void Init_buzzer(void);


void Write_String_LCD(char*);

void Write_Char_LCD(uint8_t);

void Write_Instr_LCD(uint8_t);

void LCD_nibble_write(uint8_t, uint8_t);

void Write_SR_LCD(uint8_t);

void LCD_Init(void);


void Keypad_Init(void);

uint8_t Read_Keypad(void);
```

```
void play_song(int num);


// Synth helpers
void record_custom_melody(void);
void play_custom_melody(void);
int  keyToFrequency(uint8_t key);
void setTempoPreset(int preset);


// Switch inits  (SW5 = PB8 select, SW4 = PB9 scroll)
void EXTI_PB8_Init(void);
void EXTI_PB9_Init(void);


void ShowMainMenu(int selection);


/* ---------- Note "frequencies" as DELAY COUNTS ----------
   We use an inverse relationship: delay ~= K / realHz
   so higher piano frequency -> smaller delay -> higher pitch.
---------------------------------------------------------------- */


#define NOTE_DELAY(f)   (50000 / (f))   // K = 50000, tweak if you want faster/slower tone


// Piano-ish base freqs from your chart, then inverted by NOTE_DELAY
#define C4_NOTE    NOTE_DELAY(262)   // ~261.63 Hz
#define CS4_NOTE   NOTE_DELAY(277)   // C#4/Db4
#define D4_NOTE    NOTE_DELAY(294)
#define DS4_NOTE   NOTE_DELAY(311)
#define E4_NOTE    NOTE_DELAY(330)
#define F4_NOTE    NOTE_DELAY(349)
#define FS4_NOTE   NOTE_DELAY(370)
#define G4_NOTE    NOTE_DELAY(392)
#define GS4_NOTE   NOTE_DELAY(415)
#define A4_NOTE    NOTE_DELAY(440)
#define AS4_NOTE   NOTE_DELAY(466)
#define B4_NOTE    NOTE_DELAY(494)
```

```
#define C5_NOTE    NOTE_DELAY(523)

#define D5_NOTE    NOTE_DELAY(587)

#define E5_NOTE    NOTE_DELAY(659)

#define F5_NOTE    NOTE_DELAY(698)

#define G5_NOTE    NOTE_DELAY(784)

#define A5_NOTE    NOTE_DELAY(880)

#define B5_NOTE    NOTE_DELAY(900)


// 3rd octave notes (one octave below your current main range)

#define C3_NOTE    NOTE_DELAY(131)   // ~130.81 Hz

#define D3_NOTE    NOTE_DELAY(147)   // ~146.83 Hz

#define E3_NOTE    NOTE_DELAY(165)   // ~164.81 Hz

#define F3_NOTE    NOTE_DELAY(175)   // ~174.61 Hz

#define G3_NOTE    NOTE_DELAY(196)   // 196 Hz

#define A3_NOTE    NOTE_DELAY(220)   // 220 Hz

#define B3_NOTE    NOTE_DELAY(247)   // ~246.94 Hz

#define DS3_NOTE   NOTE_DELAY(156)   // ~155.56 Hz  (D#3 / Eb3)

#define AS3_NOTE   NOTE_DELAY(233)   // ~233.08 Hz  (A#3 / Bb3)

#define FS3_NOTE   NOTE_DELAY(185)   // ~185 Hz     (F#3 / Gb3) for HP bass

#define GS3_NOTE            NOTE_DELAY(208)   // ~207.65 Hz


typedef struct

{

    int frequency; // actually: delay count used in inner loop

    int cycles;    // how long to play (outer loop)

    int pause;     // delay after the note

} Note;


/* ---------- SONGS ---------- */


/* Song 1: Imperial March    main riff

   First time: low (3rd octave)

   Second time: same riff, but higher (4th octave)

*/
```

```
Note miiMelody[] = {
    // ---- Low riff (3rd octave) ----
    // G G G  |  D# Bb G   D# Bb G   (classic opening)
    { G3_NOTE,  180, 40 }, { G3_NOTE,  180, 40 }, { G3_NOTE,  220, 80 },
    { DS3_NOTE, 150, 40 }, { AS3_NOTE, 320, 80 },
    { G3_NOTE,  220, 40 }, { DS3_NOTE, 150, 40 }, { AS3_NOTE, 320, 80 },
    { G3_NOTE,  320, 120 },

    // ---- Same riff, but higher (4th octave) ----
    { D4_NOTE,  180, 40 }, { D4_NOTE,  180, 40 }, { D4_NOTE,  220, 80 },
    { F4_NOTE, 150, 40 }, { CS4_NOTE, 320, 80 },
    { E3_NOTE,  180, 40 }, { DS4_NOTE, 150, 40 }, { AS4_NOTE, 320, 80 },
    { G4_NOTE,  340, 160 },
};
#define MII_MELODY_LENGTH (sizeof(miiMelody) / sizeof(miiMelody[0]))

/* Song 2: Super Mario Bros.   intro, 4th octave, longer notes (your version) */
Note marioMelody[] = {
    // Phrase 1: E E E  C E  G   |   G (held)
    // Using E4/C4/G4 so it's higher than 3rd octave but not squeaky like 5th.
    { E4_NOTE, 150, 25 }, { E4_NOTE, 150, 25 }, { E4_NOTE, 230, 50 },
    { C4_NOTE, 150, 25 }, { E4_NOTE, 150, 25 }, { G4_NOTE, 320, 80 },
    { G4_NOTE, 360, 120 },

    // Phrase 2: C G E  A B Bb A
    // All in 4th octave so it matches the first phrase.
    { C4_NOTE, 150, 25 }, { G4_NOTE, 150, 25 }, { E4_NOTE, 150, 25 },
    { A4_NOTE, 150, 25 }, { B4_NOTE, 150, 25 }, { AS4_NOTE, 150, 25 }, { A4_NOTE, 260, 80 },

    // Phrase 3: G E G A  F G E C D C
    // Still 4th octave, keep final note as held resolution.
    { G4_NOTE, 150, 25 }, { E4_NOTE, 150, 25 }, { G4_NOTE, 150, 25 }, { A4_NOTE, 260, 60 },
    { F4_NOTE, 150, 25 }, { G4_NOTE, 150, 25 }, { E4_NOTE, 150, 25 },
    { C4_NOTE, 150, 25 }, { D4_NOTE, 150, 25 }, { C4_NOTE, 260, 120 },
```

```
};

#define MARIO_MELODY_LENGTH (sizeof(marioMelody) / sizeof(marioMelody[0]))


/* Song 3: Jingle Bells   EXACTLY your version */

Note jingleMelody[] = {

   // E E E | E E E | E G C D E

   { E4_NOTE, 90, 25 }, { E4_NOTE, 90, 25 }, { E4_NOTE, 180, 60 },

   { E4_NOTE, 90, 25 }, { E4_NOTE, 90, 25 }, { E4_NOTE, 180, 60 },


   { E4_NOTE, 90, 25 }, { G4_NOTE, 90, 25 }, { C4_NOTE, 90, 25 },

   { D4_NOTE, 90, 25 }, { E4_NOTE, 220, 80 },


   // F F F F | E E E | G G F D C

   { F4_NOTE, 90, 25 }, { F4_NOTE, 90, 25 }, { F4_NOTE, 90, 25 }, { F4_NOTE, 180, 60 },


   { E4_NOTE, 90, 25 }, { E4_NOTE, 90, 25 }, { E4_NOTE, 220, 80 },


   { G4_NOTE, 90, 25 }, { G4_NOTE, 90, 25 }, { F4_NOTE, 90, 25 },

   { D4_NOTE, 90, 25 }, { C4_NOTE, 220, 80 },

};

#define JINGLE_MELODY_LENGTH (sizeof(jingleMelody) / sizeof(jingleMelody[0]))


/* Song 4: Harry Potter (Hedwig's Theme)   your shorter riff */

Note hpMelody[] = {

   // Phrase 1: B E G F#

   { B3_NOTE, 220, 40 }, { E4_NOTE, 220, 40 }, { G4_NOTE, 220, 40 }, { FS4_NOTE, 420, 80 },


   // Phrase 2: E B A F#

   { E4_NOTE, 220, 40 }, { B4_NOTE, 220, 40 }, { A4_NOTE, 420, 40 }, { FS4_NOTE, 420, 80 },


   // Phrase 3: E G F# D#

   { E4_NOTE, 220, 40 }, { G4_NOTE, 220, 40 }, { FS4_NOTE, 220, 40 }, { DS4_NOTE, 420, 80 },


   // Phrase 4: F B E B (little closing lick)
```

```
    { F4_NOTE, 220, 40 }, { B3_NOTE, 220, 40 },
};
#define HP_MELODY_LENGTH (sizeof(hpMelody) / sizeof(hpMelody[0]))

/* Song 5: Nokia */
Note nokiaMelody[] = {
                        // Phrase 1: E G F G
                        { E4_NOTE, 180, 60 }, { G4_NOTE, 90, 50 }, { F4_NOTE, 90, 50 }, { G4_NOTE, 220, 500 },
                        // Phrase 2: E E F G A G F
                        { E4_NOTE, 180, 60 }, { E4_NOTE, 90, 50 }, { F4_NOTE, 90, 50 }, { G4_NOTE, 90, 50 }, { A4_NOTE, 90, 50 }, { G4_NOTE, 90, 50 }, { F4_NOTE, 90,
50 },
                        // Phrase 3: E D E E
                        { E4_NOTE, 180, 60 }, { D5_NOTE, 180, 50 }, { E5_NOTE, 180, 50 }, { E4_NOTE, 180, 500 },
                        // repeat of first three phrases
                        { E4_NOTE, 180, 60 }, { G4_NOTE, 90, 50 }, { F4_NOTE, 90, 50 }, { G4_NOTE, 220, 500 },

                        { E4_NOTE, 180, 60 }, { E4_NOTE, 90, 50 }, { F4_NOTE, 90, 50 }, { G4_NOTE, 90, 50 }, { A4_NOTE, 90, 50 }, { G4_NOTE, 90, 50 }, { F4_NOTE, 90,
50 },

                        { E4_NOTE, 180, 60 }, { D5_NOTE, 180, 50 }, { E5_NOTE, 180, 50 }, { E4_NOTE, 180, 500 },
                        // Phrase 7: D F E F
                        { D4_NOTE, 180, 60 }, { F4_NOTE, 90, 25 }, { E4_NOTE, 90, 25 }, { F4_NOTE, 220, 500 },
                        // Phrase 8: E F G A G F
                        { E4_NOTE, 180, 60 }, { E4_NOTE, 90, 50 }, { F4_NOTE, 90, 50 }, { G4_NOTE, 90, 50 }, { A4_NOTE, 90, 50 }, { G4_NOTE, 90, 50 }, { F4_NOTE, 90,
50 },
                        // Phrase 9: E E F D E
                        { E4_NOTE, 125, 250 }, { E4_NOTE, 250, 250 }, { F4_NOTE, 90, 25 }, { D4_NOTE, 90, 25 }, { E4_NOTE, 180, 300 },
                        // Phrase 10: E F E D C
                        { E5_NOTE, 180, 50 }, { F5_NOTE, 180, 50 }, { E5_NOTE, 180, 50 }, { D5_NOTE, 180, 50 }, { C5_NOTE, 180, 50 }
};

#define NKA_MELODY_LENGTH (sizeof(nokiaMelody) / sizeof(hpMelody[0]))

Note LGMelody[] = {
                        // Phrase 1: C# F# F D#
                        { CS4_NOTE, 300, 50 }, { FS4_NOTE, 100, 50 }, { F4_NOTE, 100, 50 }, { DS4_NOTE, 100, 50 },
```

```
                        // Phrase 2: C# A#
                        { CS4_NOTE, 300, 50 }, { AS3_NOTE, 300, 50 },
                        // Phrase 3: B C# D# G# A# B
                        { B3_NOTE, 100, 13 }, { CS4_NOTE, 100, 13 }, { DS4_NOTE, 100, 13 }, { GS3_NOTE, 100, 13 }, { AS3_NOTE, 100, 25 }, { B3_NOTE, 100, 13 },
                        // Phrase 4: A# C#
                        { AS3_NOTE, 300, 50 }, { CS4_NOTE, 300, 50 },
                        // Phrase 5: C# F# F D#
                        { CS4_NOTE, 300, 50 }, { FS4_NOTE, 100, 50 }, { F4_NOTE, 100, 50 }, { DS4_NOTE, 100, 50 },
                        // Phrase 6: C# F#
                        { CS4_NOTE, 300, 50 }, { FS4_NOTE, 300, 50 },
                        // Phrase 7: F# G# F# F D# F F#
                        { FS4_NOTE, 100, 50 }, { GS4_NOTE, 100, 50 }, { FS4_NOTE, 100, 50 }, { F4_NOTE, 100, 50 }, { DS4_NOTE, 100, 50 }, { F4_NOTE, 100, 50 }, { FS4_NOTE, 400, 50 },


                        };

#define LG_MELODY_LENGTH (sizeof(LGMelody) / sizeof(LGMelody[0]))
/* ---------- Custom Synth Storage ---------- */


#define MAX_CUSTOM_STEPS 16

Note customMelody[MAX_CUSTOM_STEPS];
int  customLength = 0;

int baseCycles = 150;
int basePause  = 50;

/* ---------- Menu & Flags ---------- */

volatile uint8_t swScrollFlag = 0; // SW4 PB9
volatile uint8_t swSelectFlag = 0; // SW5 PB8

// NEW: simple locks to kill bounce / double-triggers
volatile uint8_t scrollLock = 0;
```

```
volatile uint8_t selectLock = 0;


int currentSelection = 1;  // 1..5 (1 4 songs, 5=record)


/* ---------- Buzzer Tone ---------- */


void playTone(int frequency, int cycles)
{
    for (int i = 0; i < cycles; i++)
    {
        for (volatile int a = 0; a < frequency; a++)
        {
            GPIOC->ODR |= (1 << 9); // PC9 ON
        }
        for (volatile int a = 0; a < frequency; a++)
        {
            GPIOC->ODR &= ~(1 << 9); // PC9 OFF
        }
    }
    GPIOC->ODR &= ~(1 << 9); // ensure OFF
}


/* ---------- Synth Helpers ---------- */


int keyToFrequency(uint8_t key)
{
    // 0 = rest, 1 8 = C4..C5
    switch (key)
    {
        case 1:  return C4_NOTE;
        case 2:  return D4_NOTE;
        case 3:  return E4_NOTE;
        case 4:  return F4_NOTE;
        case 5:  return G4_NOTE;
```

```
        case 6:  return A4_NOTE;
        case 7:  return B4_NOTE;
        case 8:  return C5_NOTE;
        case 0:  return 0;       // rest
        default: return 0;
    }
}

/* SHORTER synth note lengths */
void setTempoPreset(int preset)
{
    if (preset == 1)      // slow
    {
        baseCycles = 160;
        basePause  = 55;
    }
    else if (preset == 2) // medium
    {
        baseCycles = 110;
        basePause  = 40;
    }
    else if (preset == 3) // fast
    {
        baseCycles = 70;
        basePause  = 25;
    }
}

/* ---------- Play Fixed Songs ---------- */

void play_song(int num)
{
    LCD_Init();
```

```
if (num == 1)
{
    Write_Instr_LCD(0x80);
    Write_String_LCD("Playing Song 1 ");
    Write_Instr_LCD(0xC0);
    Write_String_LCD("Imperial March ");

    for (unsigned int i = 0; i < MII_MELODY_LENGTH; i++)
    {
        playTone(miiMelody[i].frequency, miiMelody[i].cycles);
        HAL_Delay(miiMelody[i].pause);
                                                    if (selectLock == 0)
                                                    {
                                                        break;
                                                    }
    }
}
else if (num == 2)
{
    Write_Instr_LCD(0x80);
    Write_String_LCD("Playing Song 2 ");
    Write_Instr_LCD(0xC0);
    Write_String_LCD("Mario Theme    ");

    for (unsigned int i = 0; i < MARIO_MELODY_LENGTH; i++)
    {
        playTone(marioMelody[i].frequency, marioMelody[i].cycles);
        HAL_Delay(marioMelody[i].pause);
                                                    if (selectLock == 0)
                                                    {
                                                        break;
                                                    }
    }
}
```

```
else if (num == 3)
{
    Write_Instr_LCD(0x80);
    Write_String_LCD("Playing Song 3 ");
    Write_Instr_LCD(0xC0);
    Write_String_LCD("Jingle Bells   ");

    for (unsigned int i = 0; i < JINGLE_MELODY_LENGTH; i++)
    {
        playTone(jingleMelody[i].frequency, jingleMelody[i].cycles);
        HAL_Delay(jingleMelody[i].pause);
                                                                    if (selectLock == 0)
                                                                    {
                                                                            break;
                                                                    }
    }
}
else if (num == 4)
{
    Write_Instr_LCD(0x80);
    Write_String_LCD("Playing Song 4 ");
    Write_Instr_LCD(0xC0);
    Write_String_LCD("Harry Potter   ");

    for (unsigned int i = 0; i < HP_MELODY_LENGTH; i++)
    {
        playTone(hpMelody[i].frequency, hpMelody[i].cycles);
        HAL_Delay(hpMelody[i].pause);
                                                                    if (selectLock == 0)
                                                                    {
                                                                            break;
                                                                    }
    }
}
```

```
                    else if (num == 5)
{
    Write_Instr_LCD(0x80);
    Write_String_LCD("Playing Song 5 ");
    Write_Instr_LCD(0xC0);
    Write_String_LCD("Nokia          ");

    for (unsigned int i = 0; i < NKA_MELODY_LENGTH; i++)
    {
        playTone(nokiaMelody[i].frequency, nokiaMelody[i].cycles);
        HAL_Delay(nokiaMelody[i].pause);
                                                        if (selectLock == 0)
                                                        {
                                                            break;
                                                        }
    }
}

                    else if (num == 6)
{
    Write_Instr_LCD(0x80);
    Write_String_LCD("Playing Song 6 ");
    Write_Instr_LCD(0xC0);
    Write_String_LCD("LG Machine     ");

    for (unsigned int i = 0; i < LG_MELODY_LENGTH; i++)
    {
        playTone(LGMelody[i].frequency, LGMelody[i].cycles);
        HAL_Delay(LGMelody[i].pause);
                                                        if (selectLock == 0)
                                                        {
                                                            break;
                                                        }
```

```
        }
    }
    HAL_Delay(300);
}

/* ---------- Record Custom Melody (with live preview) ---------- */

void record_custom_melody(void)
{
    customLength = 0;
    uint8_t key;

    LCD_Init();
    Write_Instr_LCD(0x80);
    Write_String_LCD("Recording...   ");
    Write_Instr_LCD(0xC0);
    Write_String_LCD("*=End 0-8=Note ");

    while (customLength < MAX_CUSTOM_STEPS)
    {
        key = Read_Keypad();

        // '*' = end (key 14)
        if (key == 14)
            break;

        int freq = keyToFrequency(key);

        // Only keys 0 8 are allowed (0 = rest)
        if (freq == 0 && key != 0)
        {
            LCD_Init();
            Write_Instr_LCD(0x80);
            Write_String_LCD("Use 0-8 only   ");
```

```
    HAL_Delay(400);
    LCD_Init();
    Write_Instr_LCD(0x80);
    Write_String_LCD("Recording...   ");
    Write_Instr_LCD(0xC0);
    Write_String_LCD("*=End 0-8=Note ");
    continue;
}

// preview note
if (freq != 0)
{
    playTone(freq, baseCycles);
    HAL_Delay(basePause);
}
else
{
    HAL_Delay(basePause);
}

// Ask for length: 1 = short, 2 = medium, 3 = long
LCD_Init();
Write_Instr_LCD(0x80);
Write_String_LCD("Len 1=S 2=M 3=L");
Write_Instr_LCD(0xC0);
Write_String_LCD("Pick length    ");

uint8_t lenKey;
int lenUnit;

HAL_Delay(100);
while (1)
{
    lenKey = Read_Keypad();
```

```
            if (lenKey == 1 || lenKey == 2 || lenKey == 3)
            {
                lenUnit = lenKey;
                break;
            }
        }

        customMelody[customLength].frequency = freq;
        customMelody[customLength].cycles    = baseCycles * lenUnit;
        customMelody[customLength].pause     = basePause  * lenUnit;
        customLength++;

        LCD_Init();
        Write_Instr_LCD(0x80);
        Write_String_LCD("Step saved     ");
        HAL_Delay(200);
        LCD_Init();
        Write_Instr_LCD(0x80);
        Write_String_LCD("Recording...   ");
        Write_Instr_LCD(0xC0);
        Write_String_LCD("*=End 0-8=Note ");
    }

    LCD_Init();
    Write_Instr_LCD(0x80);
    Write_String_LCD("Done Recording ");
    HAL_Delay(500);

    play_custom_melody();
}

/* ---------- Play Custom Melody ---------- */

void play_custom_melody(void)
```

```
{
    if (customLength == 0)
    {
        LCD_Init();
        Write_Instr_LCD(0x80);
        Write_String_LCD("No song saved  ");
        HAL_Delay(700);
        return;
    }

    LCD_Init();
    Write_Instr_LCD(0x80);
    Write_String_LCD("Playing Synth  ");
    HAL_Delay(300);

    for (int i = 0; i < customLength; i++)
    {
        int f   = customMelody[i].frequency;
        int cyc = customMelody[i].cycles;
        int pau = customMelody[i].pause;

        if (f == 0)
            HAL_Delay(pau);
        else
        {
            playTone(f, cyc);
            HAL_Delay(pau);
        }
    }

    LCD_Init();
    Write_Instr_LCD(0x80);
    Write_String_LCD("Done Synth     ");
    HAL_Delay(500);
```

```
}

/* ---------- Menu ---------- */

void ShowMainMenu(int selection)
{
   LCD_Init();
   Write_Instr_LCD(0x80);
   Write_String_LCD("Use SW4/5 Menu ");

   Write_Instr_LCD(0xC0);
   switch (selection)
   {
      case 1: Write_String_LCD("1) ImperialMar"); break;
      case 2: Write_String_LCD("2) Mario Theme "); break;
      case 3: Write_String_LCD("3) Jingle Bells"); break;
      case 4: Write_String_LCD("4) HarryPotter "); break;
                                                  case 5: Write_String_LCD("5) Nokia       "); break;
                                                  case 6: Write_String_LCD("6) LG          "); break;
      case 7: Write_String_LCD("7) Record Synth"); break;
      default:Write_String_LCD("Invalid option "); break;
   }
}

int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* Configure the system clock */
                Init_LED0();
                EXTI_PB8_Init();
                Init_7Seg();
                //SystemClock_Config();
                //SysTick_Initialize(36665);
```

```
                                LCD_Init();
Init_buzzer();
Keypad_Init();

EXTI_PB8_Init(); // SW5 select
EXTI_PB9_Init(); // SW4 scroll

setTempoPreset(2); // medium tempo

// Startup test beep so we know buzzer is alive
playTone(E4_NOTE, 400);
HAL_Delay(400);

ShowMainMenu(currentSelection);

while (1)
{
    if (swScrollFlag)
    {
        // lock out further scroll interrupts until we're done debouncing
        scrollLock = 1;
        swScrollFlag = 0;

        while (GPIOB->IDR & (1 << 9)) { } // wait release
        Delay(200);                       // debounce

        currentSelection++;
        if (currentSelection > 7) currentSelection = 1;
        ShowMainMenu(currentSelection);

        scrollLock = 0;
    }

    if (swSelectFlag)
```

```
{
    // lock out further select interrupts until song/record is done
    selectLock = 1;
    swSelectFlag = 0;

    while (GPIOB->IDR & (1 << 8)) { } // wait release
    Delay(200);                       // debounce

    if (currentSelection >= 1 && currentSelection <= 6)
    {
        play_song(currentSelection);
        ShowMainMenu(currentSelection);
    }
    else if (currentSelection == 7)
    {
        LCD_Init();
        Write_Instr_LCD(0x80);
        Write_String_LCD("Tempo 1=S 2=M 3=F");
        Write_Instr_LCD(0xC0);
        Write_String_LCD("Pick tempo     ");

        uint8_t tkey;
        HAL_Delay(100);
        while (1)
        {
            tkey = Read_Keypad();
            if (tkey == 1 || tkey == 2 || tkey == 3)
            {
                setTempoPreset(tkey);
                break;
            }
        }

        LCD_Init();
```

```
            Write_Instr_LCD(0x80);
            Write_String_LCD("Tempo Set      ");
            HAL_Delay(400);

            record_custom_melody();
            ShowMainMenu(currentSelection);
          }

          selectLock = 0;
       }
    }



}
void SysTick_Initialize (uint32_t ticks)
                {
                SysTick->LOAD = ticks - 1;    // Set reload register
                                SysTick->VAL = 0;             // Reset the SysTick counter value

                                // Select processor clock to internal: 1 = processor clock; 0 = external clock
   SysTick->CTRL |= 0x04;  //SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;

                                // Enable counting of SysTick
   SysTick->CTRL |= 0x1; //SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;

                                // Enables SysTick interrupt, 1 = Enable, 0 = Disable
   SysTick->CTRL |= 0x1<<1; //SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;
}

//================================================

/*void SysTick_Handler(void)
{ // SysTick interrupt service routine
GPIOA->ODR^=(1<<1);
```

```
}
*/

void EXTI9_5_IRQHandler(void)
		{
  if (EXTI->PR1 & (1 << 8)) // PB8 SW5
  {
    EXTI->PR1 |= (1 << 8);   // clear pending
    if (!selectLock)        // only set flag if not locked
      swSelectFlag = 1;
                                                        if (selectLock == 1)                            // using selectLock like a state for stopping the song
                                                        {
                                                                selectLock = 0;
                                                        }
  }
  if (EXTI->PR1 & (1 << 9)) // PB9 SW4
  {
    EXTI->PR1 |= (1 << 9);   // clear pending
    if (!scrollLock)        // only set flag if not locked
      swScrollFlag = 1;
  }
		}

void EXTI_PB8_Init(void)
{
		//uint32_t temp;
		//===== (1) Configure PB8 input
		RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;/* enable GPIOB clock */
		//Configure PB8 pin input
		 GPIOB->MODER &= ~(0x03<<(2*8));
		 GPIOB->OTYPER &=~(0x01<<8);
		 GPIOB->PUPDR &=~(0x03<<(2*8));
		 GPIOB->PUPDR |=(0x02<<(2*8));
		// ===== (2) Connect External Line to the GPI
```

```
		//enable the clock of SYSCFG
		RCC->APB2ENR |= 0x00000001;
		//RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
		//clear the 4 bits of the EXIT5
		//RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
			SYSCFG->EXTICR[2] &= ~SYSCFG_EXTICR3_EXTI8_PB;
			SYSCFG->EXTICR[2] |= SYSCFG_EXTICR3_EXTI8_PB;
		// ===== (3) Rising trigger selection
		// 0 = trigger disabled, 1 = trigger enabled
			 EXTI->RTSR1 |= (1<<8);  //EXTI->RTSR |= EXTI_RTSR_RT8;

		// ===== (4) Enable interrupt
		// 4.1 NVIC enable bit
		NVIC->ISER[0U] = 1<<23;  //OR NVIC_EnableIRQ(EXTI9_5_IRQn);
		// 0 = marked, 1 = not masked (enabled)
			EXTI->IMR1 |= (1<<8);  //EXTI->IMR |= EXTI_IMR_IM8;
}

void EXTI_PB9_Init(void)
{
   RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;

   GPIOB->MODER &= ~(0x03<<(2*9));
   GPIOB->OTYPER &= ~(0x01<<9);
   GPIOB->PUPDR &= ~(0x03<<(2*9));
   GPIOB->PUPDR |=  (0x02<<(2*9));

   RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
   SYSCFG->EXTICR[2] &= ~SYSCFG_EXTICR3_EXTI9_PB;
   SYSCFG->EXTICR[2] |=  SYSCFG_EXTICR3_EXTI9_PB;

   EXTI->RTSR1 |= (1<<9);
   EXTI->IMR1  |= (1<<9);
```

```
    NVIC->ISER[0] = 1<<23;
}

/* ---------- Buzzer GPIO ---------- */

void Init_buzzer()
{
    uint32_t temp;
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN; /* enable GPIOC clock*/

    temp = GPIOC->MODER;
    temp &= ~(0x03<<(2*9));
    temp |=  (0x01<<(2*9));  // PC9 output
    GPIOC->MODER = temp;

    temp=GPIOC->OTYPER;
    temp &=~(0x01<<9);
    GPIOC->OTYPER=temp;

    temp=GPIOC->PUPDR;
    temp&=~(0x03<<(2*9));
    GPIOC->PUPDR=temp;
}

/* ---------- Keypad + LCD + Delay (from lab) ---------- */

uint8_t Read_Keypad()
{
    uint8_t a;

    GPIOB->ODR|=(1<<1);
    GPIOB->ODR|=(1<<2);
    GPIOB->ODR|=(1<<3);
    GPIOB->ODR|=(1<<4);
```

```
while((GPIOB->IDR &(0x1<<8))==0 &&
      (GPIOB->IDR &(0x1<<9))==0 &&
      (GPIOB->IDR &(0x1<<10))==0 &&
      (GPIOB->IDR &(0x1<<11))==0)
{
}

Delay(25);  /*debouncing*/

while(1){
    GPIOB->ODR&=~(1<<1);
    GPIOB->ODR&=~(1<<2);
    GPIOB->ODR&=~(1<<3);
    GPIOB->ODR&=~(1<<4);

    /* Scan Col 0   PB1 = high*/
    GPIOB->ODR|=(1<<1);
    Delay(2);

    if((GPIOB->IDR &(0x1<<8))!=0)
        {a=1; break;}
    if((GPIOB->IDR &(0x1<<9))!=0)
        {a=4; break;}
    if((GPIOB->IDR &(0x1<<10))!=0)
        {a=7; break;}
    if((GPIOB->IDR &(0x1<<11))!=0)
        {a=14; break;}

    /* Scan Col 1 */
    GPIOB->ODR&=~(1<<1);
    Delay(2);
    GPIOB->ODR|=(1<<2);
    Delay(2);
```

```
if((GPIOB->IDR &(0x1<<8))!=0)
    {a=2; break;}
if((GPIOB->IDR &(0x1<<9))!=0)
    {a=5; break;}
if((GPIOB->IDR &(0x1<<10))!=0)
    {a=8; break;}
if((GPIOB->IDR &(0x1<<11))!=0)
    {a=0; break;}

/* Scan Col 2 */
GPIOB->ODR&=~(1<<2);
Delay(2);
GPIOB->ODR|=(1<<3);
Delay(2);

if((GPIOB->IDR &(0x1<<8))!=0)
    {a=3; break;}
if((GPIOB->IDR &(0x1<<9))!=0)
    {a=6; break;}
if((GPIOB->IDR &(0x1<<10))!=0)
    {a=9; break;}
if((GPIOB->IDR &(0x1<<11))!=0)
    {a=15; break;}

/* Scan Col 3 */
GPIOB->ODR&=~(1<<3);
Delay(2);
GPIOB->ODR|=(1<<4);
Delay(2);

if((GPIOB->IDR &(0x1<<8))!=0)
    {a=10; break;}
if((GPIOB->IDR &(0x1<<9))!=0)
```

```
            {a=11; break;}
        if((GPIOB->IDR &(0x1<<10))!=0)
            {a=12; break;}
        if((GPIOB->IDR &(0x1<<11))!=0)
            {a=13; break;}
    }

    GPIOB->ODR|=(1<<1);
    Delay(2);
    GPIOB->ODR|=(1<<2);
    Delay(2);
    GPIOB->ODR|=(1<<3);
    Delay(2);
    GPIOB->ODR|=(1<<4);
    Delay(2);

    while(!(((GPIOB->IDR &(0x1<<8))==0) &&
            ((GPIOB->IDR &(0x1<<9))==0) &&
            ((GPIOB->IDR &(0x1<<10))==0) &&
            ((GPIOB->IDR &(0x1<<11))==0) ))
    {}
    Delay(25);
    return(a);
}

void Keypad_Init()
{
    uint32_t temp;
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;

    /* rows PB11, PB10, PB9, PB8 input */
    temp = GPIOB->MODER;
    temp &= ~(0x03<<(2*11));
    temp &= ~(0x03<<(2*10));
```

```
temp &= ~(0x03<<(2*9));

temp &= ~(0x03<<(2*8));

GPIOB->MODER = temp;


temp=GPIOB->OTYPER;

temp &=~(0x01<<11);

temp &=~(0x01<<10);

temp &=~(0x01<<9);

temp &=~(0x01<<8);

GPIOB->OTYPER=temp;


temp=GPIOB->PUPDR;

temp&=~(0x03<<(2*11));

temp&=~(0x03<<(2*10));

temp&=~(0x03<<(2*9));

temp&=~(0x03<<(2*8));

GPIOB->PUPDR=temp;


/* Col 0 to 3 are PB1, PB2, PB3, PB4 outputs */

temp = GPIOB->MODER;

temp &= ~(0x03<<(2*1));

temp|=(0x01<<(2*1));

temp &= ~(0x03<<(2*2));

temp|=(0x01<<(2*2));

temp &= ~(0x03<<(2*3));

temp|=(0x01<<(2*3));

temp &= ~(0x03<<(2*4));

temp|=(0x01<<(2*4));

GPIOB->MODER = temp;


temp=GPIOB->OTYPER;

temp &=~(0x01<<1);

temp &=~(0x01<<2);

temp &=~(0x01<<3);
```

```
    temp &=~(0x01<<4);
    GPIOB->OTYPER=temp;

    temp=GPIOB->PUPDR;
    temp&=~(0x03<<(2*1));
    temp&=~(0x03<<(2*2));
    temp&=~(0x03<<(2*3));
    temp&=~(0x03<<(2*4));
    GPIOB->PUPDR=temp;
}

void LCD_Init()
{
    uint32_t temp;

    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;

    /*PA5 and PA10 outputs*/
    temp = GPIOA->MODER;
    temp &= ~(0x03<<(2*5));  temp|=(0x01<<(2*5));
    temp &= ~(0x03<<(2*10)); temp|=(0x01<<(2*10));
    GPIOA->MODER = temp;

    temp=GPIOA->OTYPER;
    temp &=~(0x01<<5);
    temp &=~(0x01<<10); GPIOA->OTYPER=temp;

    temp=GPIOA->PUPDR;
    temp&=~(0x03<<(2*5));
    temp&=~(0x03<<(2*10)); GPIOA->PUPDR=temp;

    /*PB5 output*/
    temp = GPIOB->MODER;
```

```
    temp &= ~(0x03<<(2*5));
    temp|=(0x01<<(2*5));
    GPIOB->MODER = temp;

    temp=GPIOB->OTYPER;
    temp &=~(0x01<<5);
    GPIOB->OTYPER=temp;

    temp=GPIOB->PUPDR;
    temp&=~(0x03<<(2*5));
    GPIOB->PUPDR=temp;

    /* LCD controller reset sequence */
    Delay(20);
    LCD_nibble_write(0x30,0);
    Delay(5);
    LCD_nibble_write(0x30,0);
    Delay(1);
    LCD_nibble_write(0x30,0);
    Delay(1);
    LCD_nibble_write(0x20,0);
    Delay(1);

    Write_Instr_LCD(0x28);
    Write_Instr_LCD(0x0E);
    Write_Instr_LCD(0x01);
    Write_Instr_LCD(0x06);
}

void LCD_nibble_write(uint8_t temp, uint8_t s)
{
    if (s==0){
        temp=temp&0xF0;
        temp=temp|0x02;
```

```
        Write_SR_LCD(temp);


        temp=temp&0xFD;
        Write_SR_LCD(temp);
    }
    else {
        temp=temp&0xF0;
        temp=temp|0x03;
        Write_SR_LCD(temp);


        temp=temp&0xFD;
        Write_SR_LCD(temp);
    }
}


void Write_String_LCD(char *temp)
{
    int i=0;


    while(temp[i]!=0)
    {
        Write_Char_LCD(temp[i]);
        i=i+1;
    }
}


void Write_Instr_LCD(uint8_t code)
{
    LCD_nibble_write(code&0xF0,0);


    code=code<<4;
    LCD_nibble_write(code,0);
}
```

```
void Write_Char_LCD(uint8_t code)
{
    LCD_nibble_write(code&0xF0,1);

    code=code<<4;
    LCD_nibble_write(code,1);
}

void Write_SR_LCD(uint8_t temp)
{
    int i;
    uint8_t mask=0b10000000;

    for(i=0; i<8; i++)
    {
        if((temp&mask)==0)
            GPIOB->ODR&=~(1<<5);
        else
            GPIOB->ODR|=(1<<5);

        GPIOA->ODR&=~(1<<5);
        GPIOA->ODR|=(1<<5);
        Delay(1);

        mask=mask>>1;
    }

    GPIOA->ODR|=(1<<10);
    GPIOA->ODR&=~(1<<10);
}

void Init_LED0()
```

```
{
	uint32_t temp;
	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;;	/* enable GPIOA clock */

	temp = GPIOA->MODER;
	temp &= ~(0x03<<(2*1));
	temp|=(0x01<<(2*1));
	GPIOA->MODER = temp;

	temp=GPIOA->OTYPER;
	temp &=~(0x01<<1);
	GPIOA->OTYPER=temp;

	temp=GPIOA->PUPDR;
	temp&=~(0x03<<(2*1));
	GPIOA->PUPDR=temp;
}

//====================================
void Write_SR_7S(uint8_t temp_Enable, uint8_t temp_Digit)
{

	switch (temp_Digit)
	{
		case 0:
			temp_Digit = 0xC0;
			break;

		case 1:
			temp_Digit = 0xF9;
			break;

		case 2:
			temp_Digit = 0xA4;
```

```
                break;


        case 3:
                temp_Digit = 0xB0;
                break;


        case 4:
                temp_Digit = 0x99;
                break;


        case 5:
                temp_Digit = 0x92;
                break;


        case 6:
                temp_Digit = 0x82;
                break;


        case 7:
                temp_Digit = 0xF8;
                break;


        case 8:
                temp_Digit = 0x80;
                break;


        case 9:
                temp_Digit = 0x90;
                break;
}
int i;
uint8_t mask=0b10000000;
for(i=0; i<8; i++)
{
```

```
                        if((temp_Digit&mask)==0)
                                GPIOB->ODR&=~(1<<5);
                        else
                                GPIOB->ODR|=(1<<5);
                /* Sclck */
                                GPIOA->ODR&=~(1<<5);
                                Delay(1);
                                GPIOA->ODR|=(1<<5);
                                Delay(1);
                                mask=mask>>1;
                }
                mask=0b10000000;
                for(i=0; i<8; i++)
                {
                        if((temp_Enable&mask)==0)
                                GPIOB->ODR&=~(1<<5);
                        else
                                GPIOB->ODR|=(1<<5);
                /* Sclck */
                        GPIOA->ODR&=~(1<<5);
                /*Delay(1);*/
                        GPIOA->ODR|=(1<<5);
                /*Delay(1); */
                        mask=mask>>1;
                }
                /*Latch*/
                        GPIOC->ODR|=(1<<10);
                        GPIOC->ODR&=~(1<<10);
}


void Init_7Seg(void)
{
                uint32_t temp;
                RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
```

```
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;

RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;


//

temp = GPIOA->MODER;

temp &= ~(0x03<<(2*5));

temp|=(0x01<<(2*5));

GPIOA->MODER = temp;


temp=GPIOA->OTYPER;

temp &=~(0x01<<5);

GPIOA->OTYPER=temp;


temp=GPIOA->PUPDR;

temp&=~(0x03<<(2*5));

GPIOA->PUPDR=temp;


//

temp = GPIOB->MODER;

temp &= ~(0x03<<(2*5));

temp|=(0x01<<(2*5));

GPIOB->MODER = temp;


temp=GPIOB->OTYPER;

temp &=~(0x01<<5);

GPIOB->OTYPER=temp;


temp=GPIOB->PUPDR;

temp&=~(0x03<<(2*5));

GPIOB->PUPDR=temp;


//

temp = GPIOC->MODER;

temp &= ~(0x03<<(2*10));
```

```
                temp|=(0x01<<(2*10));
                GPIOC->MODER = temp;

                temp=GPIOC->OTYPER;
                temp &=~(0x01<<10);
                GPIOC->OTYPER=temp;

                temp=GPIOC->PUPDR;
                temp&=~(0x03<<(2*10));
                GPIOC->PUPDR=temp;
}
void Delay(unsigned int n)
{
                int i;
    for (; n > 0; n--)
        for (i = 0; i < 136; i++) ;
                }

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
```

```
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_5;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/*void SysTick_Handler(void)
{
  // USER CODE BEGIN SysTick_IRQn 0
        if (tick > 0)
                tick--;
        else
        {
                tick = 20;
                time++;
```

```
                    if (time > 60)
                        time = 0;
                }
                Write_SR_7S(2,(time/10));
                Write_SR_7S(1, (time%10));
                Write_SR_7S(4, 0xc0);
}*/


/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
```

```
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```