Politécnico
Castelo Branco
Polytechnic University

Microcontrollers and Microprocessors

Laboratory work 1 - Piano

Diana-Patricia Toader - n°20241130
Karla-Maria Boga - n° 20241159
Timeea Goța - n° 20241160

April 2025

1. Introduction

This lab work introduces a digital piano with two modes based on the position of a switch. When switch 1 is set to 0, the piano allows users to play seven musical notes. When switch 1 is set to 1, it autonomously performs the "Happy Birthday" song.

2. Hardware

The hardware design implements a digital piano capable of operating in two modes: interactive and autonomous. The system utilizes various components to achieve its functionality (Figure 1): microcontroller, inputs components(buttons, switch), output components(LCD, buzzer).

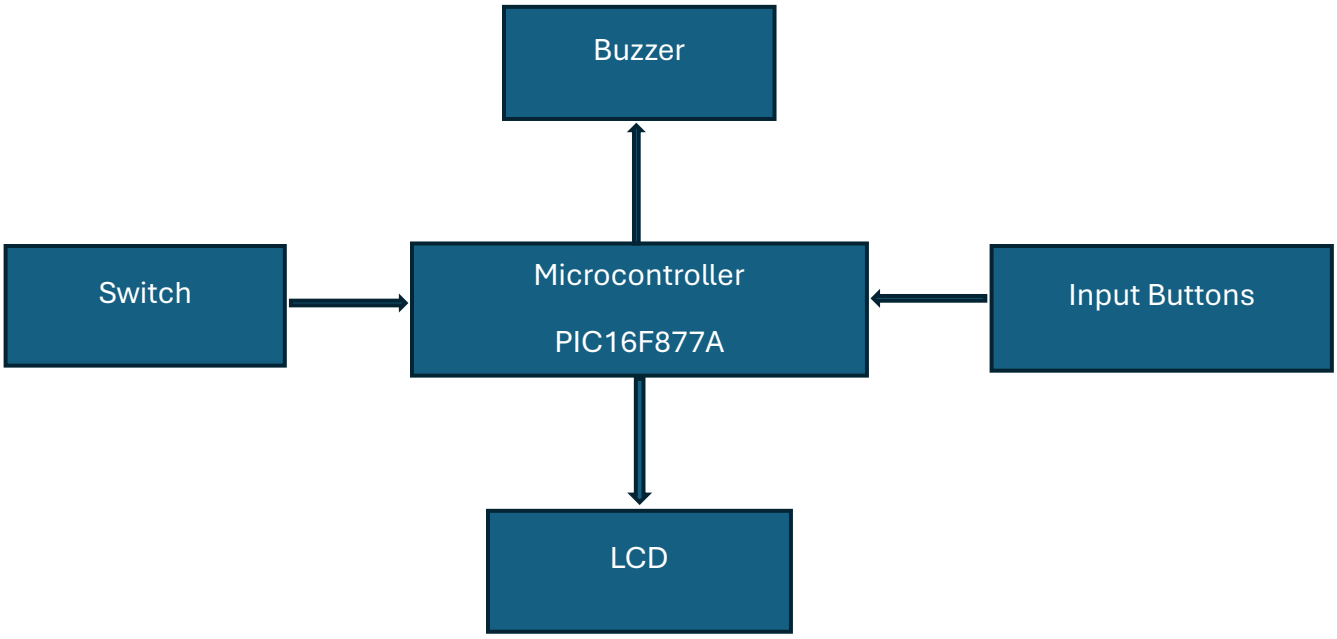
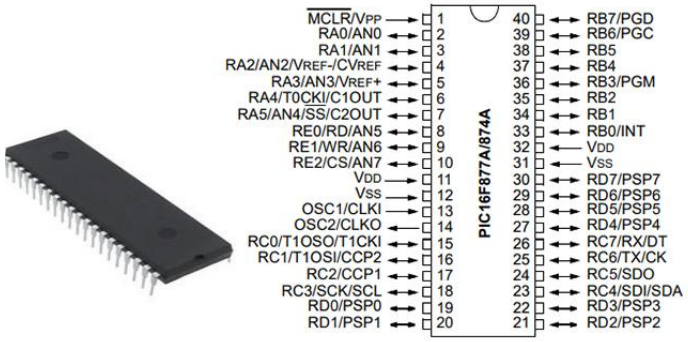


Figure 1. Block Diagram

In the center of the project is the **PIC16F877A microcontroller** (Figure 2), which acts as the central processing unit, which coordinates the interaction between the hardware components. It detects the state of the switch, processes button inputs, controls the buzzer to generate sounds, and manages the display to provide real-time feedback



.

Figure 2. PIC16F877A microcontroller with pinnout

The schematic (Figure 3) represents the physical layout and functional interconnection of the system's components, providing a clear visualization of how the switch, buttons, buzzer, and display interact with the PIC16F877A microcontroller. The schematic illustrates signal flow, pin connections, and the logical relationships between components.

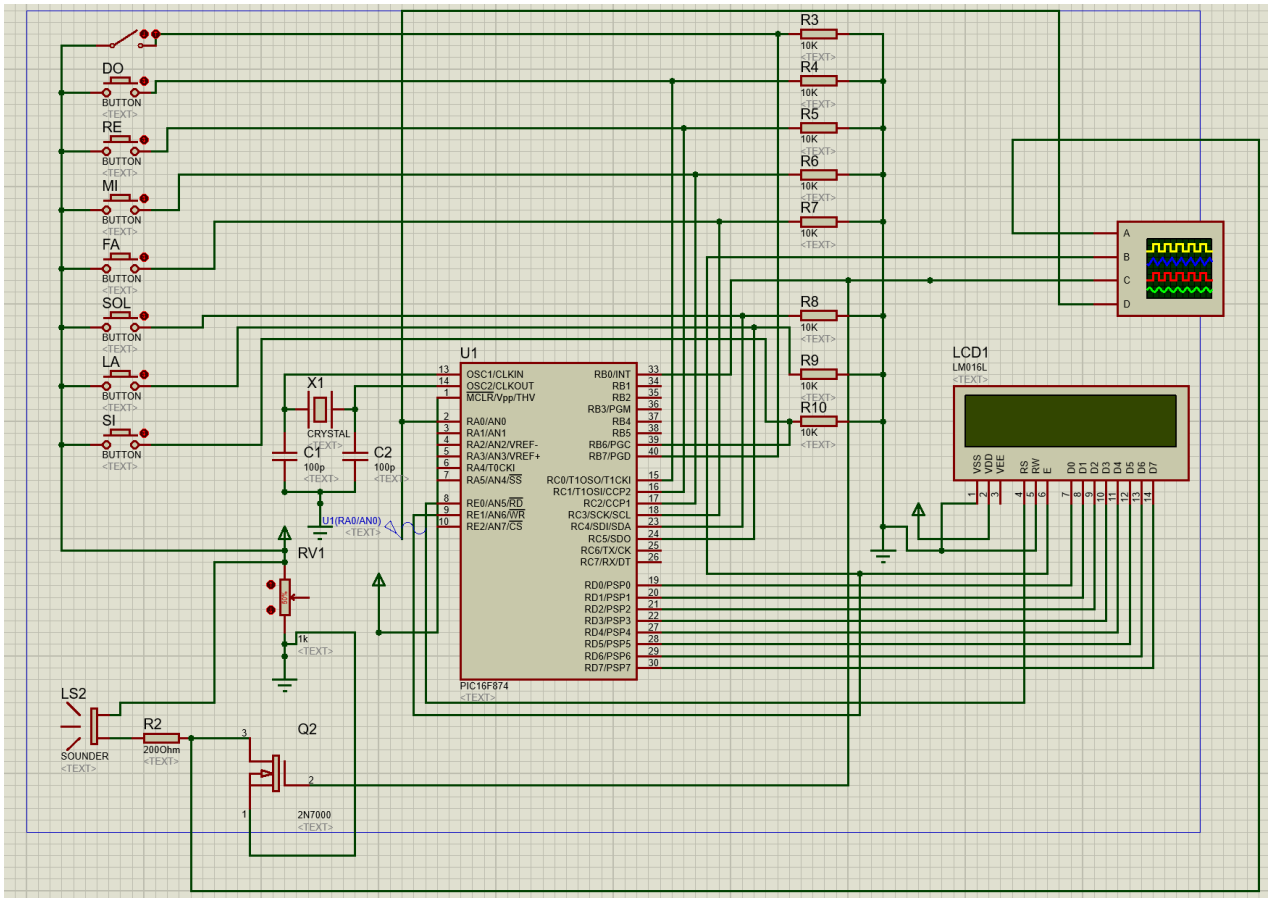


Figure 3. Schematic of the board

Components

The switch plays a crucial role in determining the piano's mode of operation. Connected to **PIN_B7**, it checks its state to set the mode. When the switch is set to 0, it activates the interactive *mode*, allowing manual input via the buttons. Conversely, when the switch is set to 1, the system automatically triggers the pre-programmed "Happy Birthday" melody to play without user intervention. This functionality provides a dynamic and dual-purpose feature for the digital piano.

The buttons are essential for the interactive mode and are mapped to specific pins, including **PIN_C0, PIN_C1, PIN_C2, PIN_C3, PIN_C4, PIN_C5,** and **PIN_B6.** Each button corresponds to a specific musical note, such as DO, RE, MI, and so on. When pressed, the buttons send signals to the microcontroller, which processes these inputs to produce the corresponding sound through the buzzer. This enables users to create and play melodies in real time.

The buzzer (LS1), connected to **PIN_B0**, functions as the sound output device of the system. It generates the audible tones, whether they are produced based on the buttons pressed by the user in interactive mode or as part of the pre-programmed melody in autonomous mode. The buzzer is key to translating electrical signals into musical sounds.

The display (LCD1), connected to **PIN_E0** (RS) and **PIN_E1** (Enable), serves as the system’s visual interface, providing feedback to the user. It indicates the current mode of operation, whether interactive or autonomous, and can display other relevant information about the piano’s status. This enhances usability by offering clear and intuitive visual guidance throughout the operation of the digital piano.

1. Software

The software component of the digital piano project governs the system's behavior, managing input processing, sound generation, and display feedback. Written for the PIC16F877A microcontroller, the code integrates the switch, buttons, buzzer, and LCD into a cohesive system. The software defines functions for each mode of operation, ensuring real-time responsiveness in interactive mode and seamless playback in autonomous mode. It serves as the backbone of the piano's functionality, executing precise control over all components. In Figure 4 we can observe the flow of the main program.

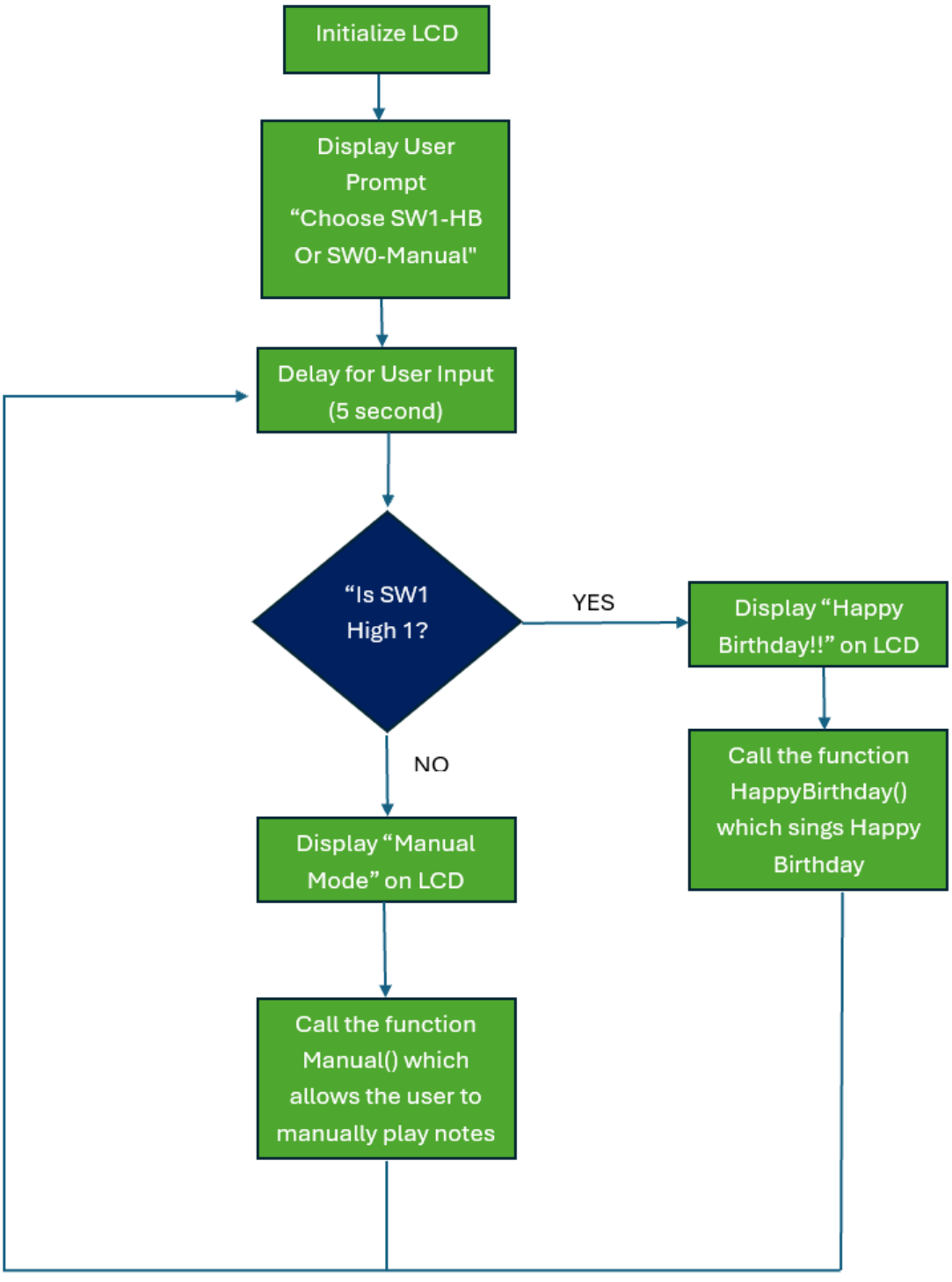


Figure 4. Flow Diagram for the main program

Basic functions

The functions *activa(), transfere_carac(),* and *transfere_inst()* manage the transfer of information to the LCD.

The *activa()* function (Figure 5.a) briefly sets the Enable pin high for 20 milliseconds and then lowers it, signaling the LCD to process data.

In the *transfere_carac()* function (Figure 5.b), the LCD is switched to character mode, and the character data is sent to the data pins. The activa() function is called to prompt the LCD to process the character.

The *transfere_inst()* function (Figure 5.b) operates similarly but sets the LCD to instruction mode instead. The instruction data is sent to the data pins, and activa() ensures the LCD processes the instruction effectively

```
16  void activa()
17  {
18    output_high(ENABLE);
19    delay_ms(10);
20    output_low(ENABLE);
21  }
```

Figure 5.a

```
24  void transfere_carac(int8 carac)
25  {
26    output_high(RS);
27    output_d(carac);
28    activa();
29  }
```

Figure 5.b

```
31  void transfere_inst(int8 inst)
32  {
33    output_low(RS);
34    output_d(inst);
35    activa();
36  }
37
```

Figure 5.c

In the *init_LCD()* function (Figure 6), the LCD is configured to instruction mode, and instructions are sent to the data pins. The activa() function is then called to prompt the LCD to process the instructions.

Instructions:

- 56 sets the LCD to operate in 8-bit mode with 2 lines.
- 1 clears the display and rests the cursor the initial position (top left)
- 12 turns on the display without a visible cursor
- 2 moves the cursor to the initial position without clearing the display

```
39  void init_LCD()
40  {
41     instrucao=56;
42     transfere_inst(instrucao);
43     instrucao=1;
44     transfere_inst(instrucao);
45     instrucao=12;
46     transfere_inst(instrucao);
47     instrucao=2;
48     transfere_inst(instrucao);
49  }
50
```

Figure 6. init_LCD() function

The *write_word()* function (Figure 7) is designed to display a word on the LCD, character by character. It iterates through each character in the given string and transfers it to the LCD, ensuring the word is displayed sequentially and clearly.

```
51  void write_word(char word[])
52  {
53     for(i=0; i<strlen(word); i++)
54        {
55           caracter = word[i];
56           transfere_carac(caracter);
57           delay_ms(500);
58        }
59        delay_ms(500);
60  }
61
```

Figure 7. write_word()

The functions of the piano

The *nota()* function (Figure 8) is designed to simplify the generation of musical notes. It takes an integer parameter that represents the desired note and uses a switch statement to determine the corresponding action. For example for DO, the pin connected to the buzzer (PIN_B0) is set high for half the period (976 microseconds) using delay_us(1953) and then set low for the remaining half. This creates the complete waveform corresponding to the frequency of DO, generating the correct pitch for the note. Adjusting the delays for each note allows the nota() function to control the buzzer's output frequency and produce different musical pitches effectively. For each note, the periods were derived from the table shown in Figure 9.

```
63  void nota(int note)
64  {
65    switch (note)
66    {
67      case 1: // Do
68        output_high(PIN_B0);
69        delay_us(1953);
70        output_low(PIN_B0);
71        delay_us(1953);
72        break;
73
74      case 2: // Re
75        output_high(PIN_B0);
76        delay_us(1709);
77        output_low(PIN_B0);
78        delay_us(1709);
79        break;
```

Figure 8. nota() function

Nota	Dó	Ré	Mi	Fá	Sol	Lá	Si
Frequencia Hz	256	293	329	366	402	439	475
Periodo s	0,0039	0,00342	0,003	0,0027	0,0025	0,0023	0,0021
Periodo us	3906	3418	3038	2734	2486	2279	2103
T/2 us	1953	1709	1519	1367	1243	1139	1052

Figure 9. Note Frequencies and Periods Table

Manual mode

Manual mode lets users play musical notes by interacting with buttons or switches connected to designated microcontroller pins.

The *Manual()* function (Figure 10) handles this process by detecting button presses and associating each with a specific note. When a button is pressed, the note's name is shown on the LCD screen while the buzzer produces the corresponding sound. This function ensures responsive, real-time interaction, allowing users to create and play music seamlessly.

```
137  void Manual()
138  {
139    if(input(PIN_C0))
140    {
141    cuv=" DO";
142    write_word(cuv);
143    }
144    while (input(PIN_C0)) {
145      nota(1); // Do
146    }
```

Figure 10. Manual() function

In the diagram below (Figure 11), the flow of the HappyBirthday() function is illustrated. The process begins with the function initialization, followed by the sequential playback of musical notes using the nota() function. Each note, such as DO, RE, or FA, is repeated a set number of times as per the predefined melody logic, with pauses introduced to maintain proper rhythm. At key points in the flow, the function checks the state of the switch (SW1). If the switch is detected to be in the "Low" state (0), the playback is interrupted, and the melody stops. Otherwise, the function proceeds through the sequence of notes until the entire song is completed. This systematic representation highlights the function's looping structure, decision-making points, and the interplay between note generation and timing to deliver the "Happy Birthday" melody.

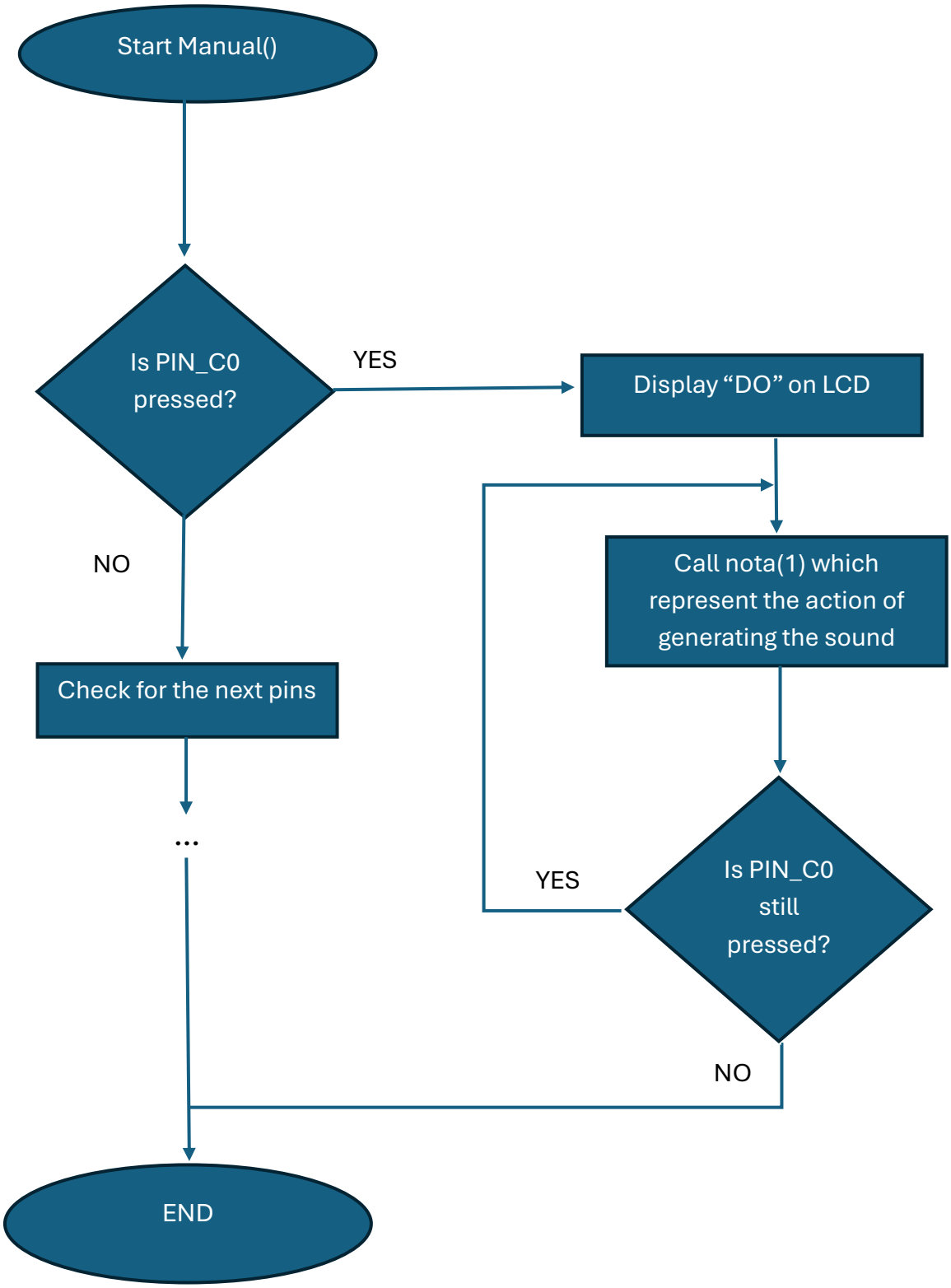


Figure 11. Flow Diagram for Manual() function

Happy Birthday mode

The HappyBirthday() function (Figure 12) generates the melody for the "Happy Birthday" song using the nota() function and a looped timing mechanism. Each musical note is repeated several times to define its duration, with pauses inserted between notes to maintain rhythm. The function checks the state of the switch connected to PIN_B7, allowing playback to stop if the switch is turned off. Notes are sequenced and structured in sections to replicate the tune accurately. The melody relies on the nota() function to produce each sound, ensuring the proper frequencies and timing for a cohesive musical output.

The number of repetitions for each note is determined using the formula:

(T_note / 2) * 2 * k=250ms

where T_note / 2 is derived from the table (Figure 9) and represents half the period of the corresponding musical note. This ensures the melody is played accurately with the appropriate timing.

```
155  void HappyBirthday()
156  {
157    for(i = 1; i <= 64; i++) // k = 64
158      nota(1); // Do
159    delay_us(100000);
160
161    for(i = 1; i <= 64; i++) // k = 64
162      nota(1); // Do
163    delay_us(100000);
164
165    for(i = 1; i <= 73*2; i++) // k = 73
166      nota(2); // Re
167    delay_us(100000);
168
169    for(i = 1; i <= 64; i++) // k = 64
170      nota(1); // Do
171    delay_us(100000);
172
173    for(i = 1; i <= 91; i++) // k = 91
174      nota(4); // Fa
175    delay_us(100000);
176
177    for(i = 1; i <= 82*3; i++) // k = 82
178      nota(3); // Mi
179    delay_us(100000);
180
181    for(i = 1; i <= 64; i++) // k = 64
182      nota(1); // Do
183    delay_us(100000);
```

Figure 12. HappyBirthday() function

Main function

The main() function is the central control hub of the program. It begins by initializing the LCD display through the init_LCD() function, configuring it to show messages to the user. After initialization, the program sets instructions for the LCD to display two prompts: "Choose SW1-HB Or SW0-Manual." (Figure 14). This message is designed to guide the user in selecting their desired mode. A delay is added to give the user time to make a decision.

Figure 14. First message showed on LCD

The program then enters an infinite loop, continuously checking the state of the switches for input. If the switch is 1, the program enters Happy Birthday mode. In this mode, the LCD displays the message "Happy Birthday" (Figure 15) and the HappyBirthday() function is called to play the melody.

Figure 15. HappyBirthday Message on LCD

If the button is not pressed, the program enters Manual mode. In this case, the LCD shows "Manual Mode" (Figure 16) and the Manual() function is executed, allowing the user to play musical notes interactively using buttons.

Figure 16. Manual Message on LC

Code:

```
#include <16F877A.h>
#include <string.h>

#fuses NOWDT,HS, NOPUT, NOPROTECT, NODEBUG, NOLVP, NOCPD, NOWRT, BROWNOUT

#include <16F874.h>
#include <string.h>

#fuses NOWDT,HS, NOPUT, NOPROTECT, NODEBUG, NOLVP, NOCPD, NOWRT, BROWNOUT

#use delay(clock=20000000)
#define RS    PIN_E0
#define ENABLE PIN_E1


int8 caracter; // Variable used for transmiting characters to LCD
int8 instrucao; // Vriable used for transmiting instructions to LCD
int16 i;
char cuv[20];  // Variable used for notes (DO, RE...)

void activa()   // Enable LCD
{
 output_high(ENABLE);
 delay_ms(10);
 output_low(ENABLE);
}


void transfere_carac(int8 carac)  // Function for character to be sent to the data pins
{
 output_high(RS);
 output_d(carac);
 activa();
}


void transfere_inst(int8 inst)  // Function for instruction to be sent to the data pins
{
 output_low(RS);
 output_d(inst);
 activa();
}


void init_LCD()  // Function for LCD initialization
{
```

```
  instrucao=56;  // Sets the LCD to operate in 8-bit mode with 2 lines.
  transfere_inst(instrucao);
  instrucao=1;  // Clears the display and rests the cursor the initial position (top left)
  transfere_inst(instrucao);
  instrucao=12;  // Turns on the display without a visible cursor
  transfere_inst(instrucao);
  instrucao=2;   // Moves the cursor to the initial position without clearing the display
  transfere_inst(instrucao);
}

void write_word(char word[])  // Display a word on the LCD, character by character
{
  for(i=0; i<strlen(word); i++)
    {
      caracter = word[i];
      transfere_carac(caracter);
    }
    delay_ms(250);
}


void nota(int note)  // Generates musical notes based on the input note parameter
{
 switch (note)  // To determine the corresponding note frequency
 {
   case 1: // Do
    output_high(PIN_B0);  // Set the buzzer pin high to generate sound
    delay_us(1953);      // Delay for half the period of the DO note (1953 microseconds)
    output_low(PIN_B0);   // Set the buzzer pin low
    delay_us(1953);      // Delay for the remaining half of the period
    break;

   case 2: // Re
    output_high(PIN_B0);  // Set the buzzer pin high
    delay_us(1709);      // Delay for half the period of the RE note
    output_low(PIN_B0);   // Set the buzzer pin low
    delay_us(1709);      // Delay for the remaining half of the period
    break;

   case 3: // Mi
    output_high(PIN_B0);  // Set the buzzer pin high
    delay_us(1519);      // Delay for half the period of the MI note
    output_low(PIN_B0);   // Set the buzzer pin low
    delay_us(1519);      // Delay for the remaining half of the period
    break;

   case 4: // Fa
    output_high(PIN_B0);  // Set the buzzer pin high
```

```
    delay_us(1367);      // Delay for half the period of the FA note
    output_low(PIN_B0);   // Set the buzzer pin low
    delay_us(1367);      // Delay for the remaining half of the period
    break;

   case 5: // Sol
    output_high(PIN_B0);  // Set the buzzer pin high
    delay_us(1243);      // Delay for half the period of the SOL note
    output_low(PIN_B0);   // Set the buzzer pin low
    delay_us(1243);      // Delay for the remaining half of the period
    break;

   case 6: // La
    output_high(PIN_B0);  // Set the buzzer pin high
    delay_us(1139);      // Delay for half the period of the LA note
    output_low(PIN_B0);   // Set the buzzer pin low
    delay_us(1139);      // Delay for the remaining half of the period
    break;

   case 7: // Si
    output_high(PIN_B0);  // Set the buzzer pin high
    delay_us(1052);      // Delay for half the period of the SI note
    output_low(PIN_B0);   // Set the buzzer pin low
    delay_us(1052);      // Delay for the remaining half of the period
    break;

   case 8: // Do2
    output_high(PIN_B0);  // Set the buzzer pin high
    delay_us(947);       // Delay for half the period of the DO2 note
    output_low(PIN_B0);   // Set the buzzer pin low
    delay_us(947);       // Delay for the remaining half of the period
    break;

   case 9: // La#
    output_high(PIN_B0);  // Set the buzzer pin high
    delay_us(1064);      // Delay for half the period of the LA# note
    output_low(PIN_B0);   // Set the buzzer pin low
    delay_us(1064);      // Delay for the remaining half of the period
    break;

   default:  // Handle invalid input
    // Invalid note - No action taken
    break;
  }
}

void Manual()
{
```

```
if(input(PIN_C0))  // Check if the button connected to PIN_C0 is pressed
{
 cuv = " DO";      // Set the note name to "DO"
 write_word(cuv);   // Display "DO" on the LCD
}
while (input(PIN_C0)) // While the button is pressed
{
 nota(1);         // Play the "DO" note
}

if(input(PIN_C1))    // Check if the button connected to PIN_C1 is pressed
{
 cuv = " RE";      // Set the note name to "RE"
 write_word(cuv);   // Display "RE" on the LCD
}
while (input(PIN_C1))
{
 nota(2);         // Play the "RE" note
}

// Repeat the same logic for other buttons and notes
if(input(PIN_C2))
{
 cuv = " MI";
 write_word(cuv);
}
while(input(PIN_C2))
{
 nota(3);         // Play the "MI" note
}

if(input(PIN_C3))
{
 cuv = " FA";
 write_word(cuv);
}
while(input(PIN_C3))
{
 nota(4);         // Play the "FA" note
}

if(input(PIN_C4))
{
 cuv = " SOL";
 write_word(cuv);
}
while(input(PIN_C4))
{
```

```
  nota(5);          // Play the "SOL" note
 }

 if(input(PIN_C5))
 {
  cuv = " LA";
  write_word(cuv);
 }
 while(input(PIN_C5))
 {
  nota(6);          // Play the "LA" note
 }

 if(input(PIN_B6))
 {
  cuv = " SI";
  write_word(cuv);
 }
 while(input(PIN_B6))
 {
  nota(7);          // Play the "SI" note
 }
}

void HappyBirthday()
{
 for(i = 1; i <= 64; i++) // Repeat 64 times for the "DO" note (k=64)
  nota(1); // Play "DO"
 delay_us(100000);  // Delay for rhythm

 for(i = 1; i <= 64; i++)
  nota(1); // Play "DO" again
 delay_us(100000);

 for(i = 1; i <= 73*2; i++) // Repeat k times multiplied by 2 (tempos)
  nota(2); // Play "RE"
 delay_us(100000);

 for(i = 1; i <= 64; i++)
  nota(1); // Play "DO"
 delay_us(100000);

 for(i = 1; i <= 91; i++)
  nota(4); // Play "FA"
 delay_us(100000);

 for(i = 1; i <= 82*3; i++)
  nota(3); // Play "MI"
```

```
 delay_us(100000);

 if(input(PIN_B7) == 0) // Check if the switch is turned off
  goto sss;          // Exit the melody if the switch is off

 // Repeat similar logic for the rest of the melody
 for(i = 1; i <= 64; i++)
  nota(1); // Play "DO"
 delay_us(100000);

 for(i = 1; i <= 64; i++)
  nota(1); // Play "DO"
 delay_us(100000);

 for(i = 1; i <= 132*2; i++)
  nota(8); // Play "DO2"
 delay_us(100000);

 // Add additional repetitions for the "Happy Birthday" tune
 for(i = 1; i <= 110; i++)
  nota(6); // Play "LA"
 delay_us(100000);

sss:
 delay_us(100000); // Exit point for stopping the melody
}

void main()
{
 set_tris_d(0x00);     // Set all PORTD pins as output
 init_LCD();          // Initialize the LCD

 char wordHB[20] = "Happy Birthday!!";
 char wordMM[20] = "Manual";
 char user[20] = "Choose SW1-HB";
 char user2[20] = "Or SW0-Manual";

 // Display the user prompt
 instrucao = 128;
 transfere_inst(instrucao);
 write_word(user);

 instrucao = 192;
 transfere_inst(instrucao);
 write_word(user2);

 delay_ms(5000);       // Allow the user time to decide
```

```
  while(1)
  {
    instrucao = 128;
    transfere_inst(instrucao);

    instrucao = 1;
    transfere_inst(instrucao);  // Clear the LCD

    if(input(PIN_B7))    // Check the state of the switch
    {
      write_word(wordHB); // Display "Happy Birthday!!"
      HappyBirthday();    // Play the "Happy Birthday" melody
    }
    else
    {
      write_word(wordMM); // Display "Manual Mode"
      Manual();          // Enable manual note play
    }
  }
}
```