Musical keyboard with display, Lab 5

Embedded System Design - CS3813301
Group 6
Tobias Vera - tobiasvera2011@gmail.com (F10915126),
Gerardo Fisch - gerarfisch@gmail.com (F10915108)
December 7th, 2021

I. PURPOSE

The purpose of this lab is to introduce the students to waveform generation using interrupts by displaying info unique message via an 8x8 LED dot matrix (1588BS) display.

II. PROCEDURE

- Connect the 1588BS dot matrix display to Arduino, use a series resistor at least 220 Ω per LED (row or column, not both).
- Connects multiple push buttons or a keypad to the Arduino, suggest use a pullup VCC and a 1K or greater resistor per button.
- Use assembly or C to decode the buttons pressed, program the counter / timer to generate a certain sound and display the sound (Note, frequency, etc..) via the LED dot matrix display.
- Program the Arduino

III. EXPERIMENTAL DATA

A. Schematic of the Circuit

The schematic of the circuit can be seen on the Figure 1. From the schematic it can be seen that the pins used for this circuit are the pins from 0 to 7 for the 8x8 LED Display, the pins 8 to 10 for the 74HC595 Shift Register, the pins 11 to 13 for the buttons, and the pin A0 for the buzzer. A potentiometer was connected to the buzzer. From the schematic of the Arduino UNO[1], it can be seen that PORT B, PORT C and PORT D are used on the circuit, since the pins from 0 to 7 correspond to the PORT D, the pins from 8 to 13 correspond to the PORT B, and the pin A0 correspond to PORT C. In addition, the GND and 5V pins were used to power the 74HC595 Shift Register, the buttons and the buzzer.

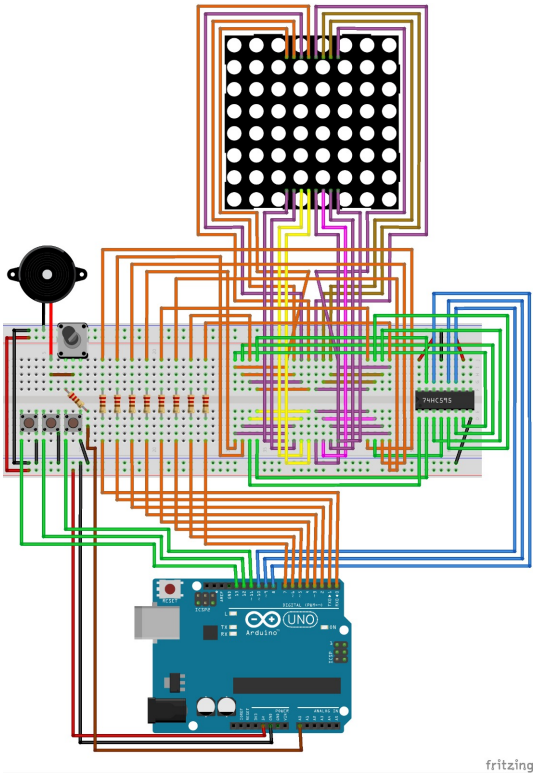

Fig. 1. Schematic of the circuit

B. Photo of the wired-up circuit

Based on the schematic of the circuit, the same circuit was reproduced. The circuit can be seen on Figure 2.

C. Assembly Code

The Assembly Code can be seen on the Appendix. A flow diagram of the code can be seen on the Figure 3 and 4.

D. Results

The Figures 5, 6, 7 and 8 show the final result.

IV. QUESTIONS

1) What are the pros and cons of using a hardware timer versus software timer?
The advantage of using timers to realize a delay is that they provide a way to allow async counting. Using a software delay forces the controller to put all its resources into processing some kind of loop (incrementing a variable until a given value) and thus blocking the rest of the code execution path.
A software delay is easier to implement and may be sufficient if it's just a very short delay which is not significantly interrupting any other task in the main sequential code processing path. Furthermore, the timers may be in use for some other hardware related tasks like PWM generation and may not be "free" to be configured according to the delay requirements.
Another use case would be some initial delay that is required

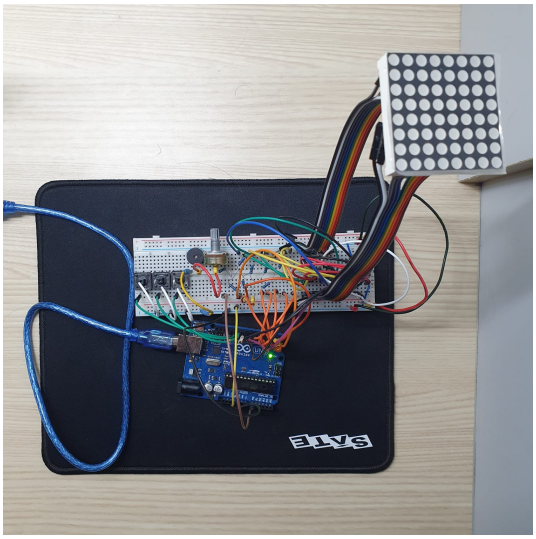

Fig. 2. Wired-up circuit

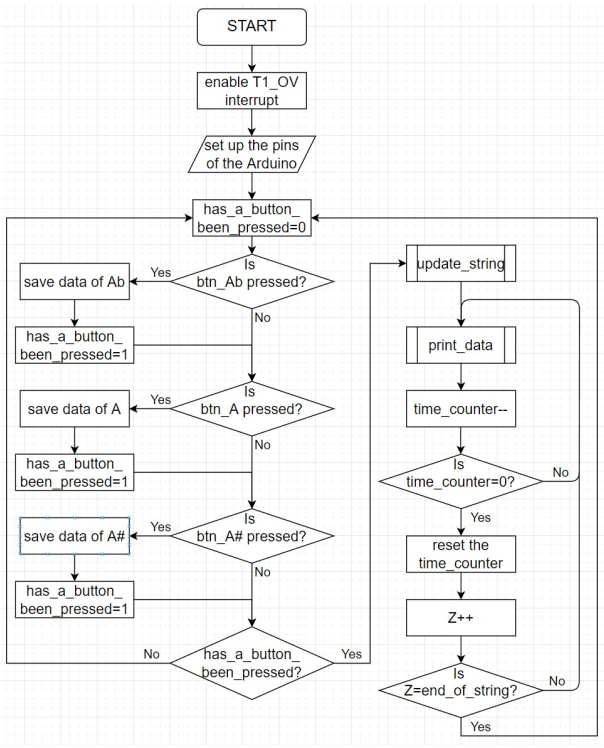

Fig. 3. Flow Diagram of the Main Program

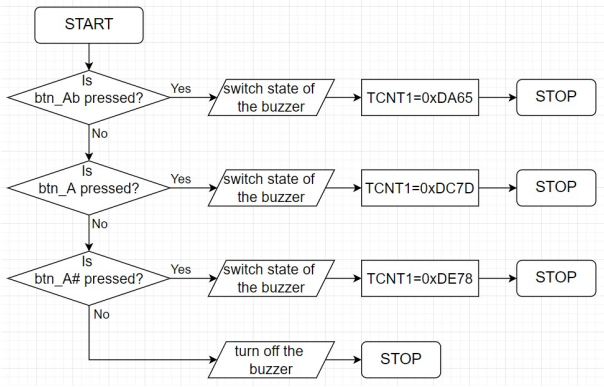

Fig. 4. Flow Diagram of the Timer1 Overflow Interrupt

before the main loop is running. There would be no need to use a hardware delay in that case.

A software delay doesn't require interrupts to be globally enabled, while it's a requirement for timer-based delays (at least for the common use case).

2) What would happen if you only used a software timer?

A software timer forces the controller to put all its resources into processing the loop, which doesn't allow any kind of parallel processing of tasks.

3) What pre-scale factors did you use?

No pre-scale factor was used for this system.

4) What happens if you press multiple buttons at a time? Think about the waveform type being sent by Arduino and what can and cannot be done.

By the way this system is programmed, if multiple buttons are pressed at a time, their corresponding interrupts are activated, but the system process the interrupts by an order of priorities, being the Ab tone button the interrupt with the highest priority.

V. DISCUSSION

The use of hardware timers is a great knowledge for the systems that will be implemented from now on. The advantage of using timers to realize a delay is that they provide a way to allow async counting, which is not possible when using software delays because all the resources are put to process the delay.

However, the implementation of software delays is easier, and can be convenient for small delays which doesn't affect the pipeline of the code so much.

Even the implementation of the square wave generations could have been done using software delays. This is because the delay used in the square wave generation is a very small task which doesn't affect the rest of the code so much.

Although only three notes were implemented this time, this system is coded in a way that it is very easy to add any number of notes and scale it to a whole musical keyboard, if the number of pins is not a problem.

The waveform generation is done through the Timer1 in normal mode, no prescaler.

In order to generate a square wave of a specific frequency, a 16-bit number which is introduced to the TCNT1H and TCNT1L registers must be calculated.

This calculation can be done like following:

1) 1 / frequency = period of the square wave.
2) 1/2 of it for the high and low portions of the pulse.
3) 65536 – (half wave period / clock period) = input for the TCNT1x registers.

As an example, the calculation of the Ab tone (831 Hz) is done as follows:

1) 1/831 = 1.20 ms
2) 1.20 ms / 2 = 0.601 ms

Fig. 5. Example of the 8x8 LED showing Ab.

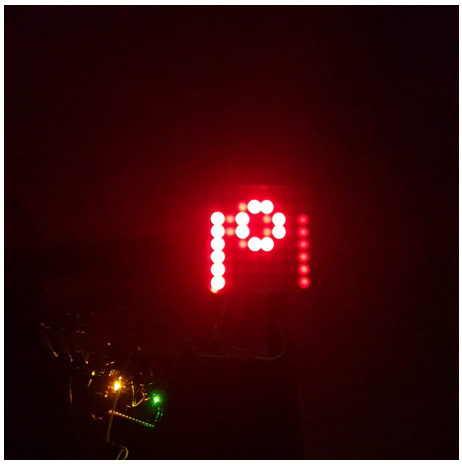

Fig. 6. Example of the 8x8 LED showing °.

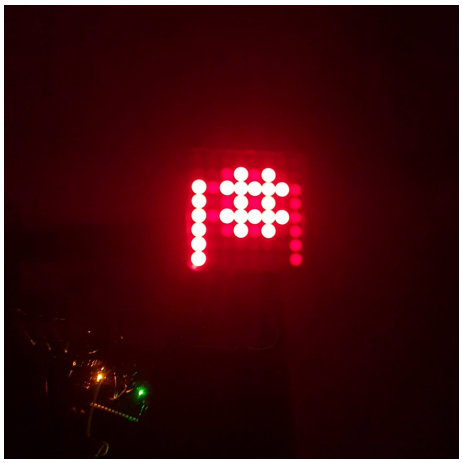

Fig. 7. Example of the 8x8 LED showing #.

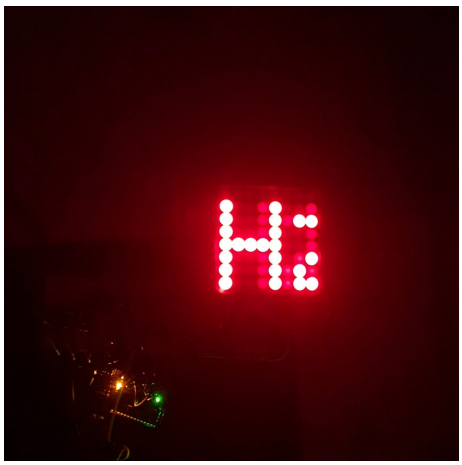

Fig. 8. Example of the 8x8 LED showing Hz.

3) 65536 – (0.601 ms / 62.5 ns) = 55909 = 0xDA65 (Hex)
4) TCNT1H = 0xDA and TCNT1L = 0x65

Note: the clock period is calculated as 1 / crystal oscillator frequency. For the ATMega328P: 1 / 16 Mhz = 62.5 ns. The calculation for each note can be seen on Table I.

VI. CONCLUSION

Advantage of timer in microcontroller is that developers can accomplish lots of timing/ counting/ delay related operations very elegantly. Whole philosophy behind their use is let hardware and registers of timer/counter peripheral do the job. However, software delay is still a good option for simple tasks that don't significantly interrupt any other task and can save a lot of programming time. It is the job of the programmer to determine which option to implement depending on the circumstances of the code.
The project went as expected, achieving the requirements dictated by the instructor, and no significant problem was noticed.

Tone (Frequency Hz)	Square wave period	Half wave period	Input for TCNT1
Ab (831 Hz)	1.20 ms	0.601 ms	55909 (0xDA65 Hex)
A (880 Hz)	1.13 ms	0.568 ms	56445 (0xDC7D Hex)
A# (932 Hz)	1.07 ms	0.536 ms	56952 (0xDE78 Hex)

TABLE I
CALCULATIONS FOR EACH NOTE.

REFERENCES

[1] https://www.arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf

APPENDIX

```
; Seting up the directions
.ORG 0x00          ; location for reset
  jmp setup
.ORG 0x001A        ; location for Timer1 overflow
  jmp T1_OV_ISR    ; jump to ISR for Timer1
; main program for initialization and keeping CPU busy
.ORG 0x40

.equ tone_letter = 0x200      ; memory where the note from the button pressed
.equ tone_sign = 0x201        ; will be saved
.equ freq_hundreds = 0x202
.equ freq_decimals = 0x203
.equ freq_units = 0x204

.equ buzzer = 0      ; PIN of the buzzer
.equ Ab = 3          ; PIN of the button corresponding to Ab
.equ A = 4           ; PIN of the button corresponding to A
.equ As = 5          ; PIN of the button corresponding to A#
.equ latchPin = 1    ; Latch pin of 74HC595 is connected to pin 9 (9 - 8 = 1)
.equ clockPin = 2    ; Clock pin of 74HC595 is connected to pin 10 (10 - 8 = 2)
.equ dataPin = 0     ; (8 - 8 = 0)

.equ tone_letter_print = 0x42      ; location of the string that corresponds to the tone letter
.equ tone_sign_print = 0x48        ; location of the string that corresponds to the tone sign
.equ freq_hundreds_print = 0x4E    ; location of the string that corresponds to the hundreds of the freq
.equ freq_decimals_print = 0x54    ; location of the string that corresponds to the decimals of the freq
.equ freq_units_print = 0x5A       ; location of the string that corresponds to the units of the freq

.equ param1 = 0x17C                ; memory space that will be used as a parameter for "functions" (call)

.def end_of_string = r29
.def has_a_button_been_pressed = r22

.equ U_0 = 0b01111110
.equ U_1 = 0b00000001
.equ U_2 = 0b00000001
.equ U_3 = 0b00000001
.equ U_4 = 0b01111110

.equ P_0 = 0b01111111
.equ P_1 = 0b01000100
.equ P_2 = 0b01000100
.equ P_3 = 0b01000100
.equ P_4 = 0b00111000

.equ T_0 = 0b01000000
.equ T_1 = 0b01000000
.equ T_2 = 0b01111111
.equ T_3 = 0b01000000
.equ T_4 = 0b01000000

.equ N_0 = 0b01111111
.equ N_1 = 0b00010000
.equ N_2 = 0b00001000
.equ N_3 = 0b00000100
.equ N_4 = 0b01111111

.equ S_0 = 0b00110010
.equ S_1 = 0b01001001
.equ S_2 = 0b01001001
.equ S_3 = 0b01001001
.equ S_4 = 0b00100110

.equ H_0 = 0b01111111
.equ H_1 = 0b00001000
.equ H_2 = 0b00001000
.equ H_3 = 0b00001000
.equ H_4 = 0b01111111

.equ A_0 = 0b00111111
.equ A_1 = 0b01001000
.equ A_2 = 0b01001000
.equ A_3 = 0b01001000
```

```
.equ A_4 = 0b00111111

.equ z_0 = 0b00100011
.equ z_1 = 0b00100101
.equ z_2 = 0b00101001
.equ z_3 = 0b00110001
.equ z_4 = 0b00000000

.equ n0_0 = 0b01111111
.equ n0_1 = 0b01000001
.equ n0_2 = 0b01000001
.equ n0_3 = 0b01000001
.equ n0_4 = 0b01111111

.equ n1_0 = 0b00000000
.equ n1_1 = 0b00100000
.equ n1_2 = 0b01111111
.equ n1_3 = 0b00000000
.equ n1_4 = 0b00000000

.equ n2_0 = 0b01001111
.equ n2_1 = 0b01001001
.equ n2_2 = 0b01001001
.equ n2_3 = 0b01001001
.equ n2_4 = 0b01111001

.equ n3_0 = 0b01001001
.equ n3_1 = 0b01001001
.equ n3_2 = 0b01001001
.equ n3_3 = 0b01001001
.equ n3_4 = 0b01111111

.equ n4_0 = 0b01111000
.equ n4_1 = 0b00001000
.equ n4_2 = 0b00001000
.equ n4_3 = 0b00001000
.equ n4_4 = 0b01111111

.equ n5_0 = 0b01111001
.equ n5_1 = 0b01001001
.equ n5_2 = 0b01001001
.equ n5_3 = 0b01001001
.equ n5_4 = 0b01001111

.equ n6_0 = 0b01111111
.equ n6_1 = 0b01001001
.equ n6_2 = 0b01001001
.equ n6_3 = 0b01001001
.equ n6_4 = 0b01001111

.equ n7_0 = 0b01000000
.equ n7_1 = 0b01000000
.equ n7_2 = 0b01000111
.equ n7_3 = 0b01001000
.equ n7_4 = 0b01110000

.equ n8_0 = 0b01111111
.equ n8_1 = 0b01001001
.equ n8_2 = 0b01001001
.equ n8_3 = 0b01001001
.equ n8_4 = 0b01111111

.equ n9_0 = 0b01111001
.equ n9_1 = 0b01001001
.equ n9_2 = 0b01001001
.equ n9_3 = 0b01001001
.equ n9_4 = 0b01111111

.equ dash = 0b00001000
.equ point = 0b00000001
.equ space = 0b00000000

.equ degree_0 = 0b00110000
.equ degree_1 = 0b01001000
.equ degree_2 = 0b01001000
```

```
.equ degree_3 = 0b00110000

.equ hash_0 = 0b00101000
.equ hash_1 = 0b01111100
.equ hash_2 = 0b00101000
.equ hash_3 = 0b01111100
.equ hash_4 = 0b00101000

.equ b_0 = 0b01111111
.equ b_1 = 0b00001001
.equ b_2 = 0b00001001
.equ b_3 = 0b00001111


setup:
  sbi DDRC, buzzer      ; set up buzzer as output
  cbi DDRB, Ab          ; set up button of Ab as input
  cbi DDRB, A           ; set up button of A as input
  cbi DDRB, As          ; set up button of A# as input
  sbi PORTB, Ab         ; pull-up enabled
  sbi PORTB, A          ; pull-up enabled
  sbi PORTB, As         ; pull-up enabled

  ldi r16, 0xFF
  out DDRD, r16         ; set up the row of the 8x8 LED as output ports
  ldi r16, 0x00
  out PORTD, r16        ; clear the display

  sbi DDRB, latchPin    ; set up the latch pin as output
  sbi DDRB, clockPin    ; set up the clock pin as output
  sbi DDRB, dataPin     ; set up the data pin as output

  sbi PORTB, dataPin    ; datapin = 1
  sbi PORTB, clockPin   ; the shifter receice '1' eight times
  nop                   ; due to the 8 instructions of nop
  nop                   ; (8 cycles)
  nop
  nop
  nop
  nop
  nop
  cbi PORTB, clockPin   ; the shifter stop receiving data
  ; this was done to have the shifter register equals 0xFF

  ; setting up the Timer1
  ldi r20, 0x00
  sts TCCR1A, r20       ; WGM11:10 = 00
  ldi r20, 0x01
  sts TCCR1B, r20       ; WGM13:12 = 00, Normal mode, prescaler = 1
  ldi r20, (1<<TOIE1)
  sts TIMSK1, r20       ; activate T1_OV_ISR
  sei                   ; activate global interrupts

fixed_string:
  ; The string will be stored into the memory starting at 0x100
  ldi ZH, 0x01
  ldi ZL, 0x00

  ldi r16, space
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16

  ldi r16, U_0
  st Z+, r16
  ldi r16, U_1
  st Z+, r16
  ldi r16, U_2
  st Z+, r16
  ldi r16, U_3
  st Z+, r16
```

```
ldi r16, U_4
st Z+, r16
ldi r16, space
st Z+, r16

ldi r16, P_0
st Z+, r16
ldi r16, P_1
st Z+, r16
ldi r16, P_2
st Z+, r16
ldi r16, P_3
st Z+, r16
ldi r16, P_4
st Z+, r16
ldi r16, space
st Z+, r16

ldi r16, T_0
st Z+, r16
ldi r16, T_1
st Z+, r16
ldi r16, T_2
st Z+, r16
ldi r16, T_3
st Z+, r16
ldi r16, T_4
st Z+, r16
ldi r16, space
st Z+, r16

ldi r16, P_0
st Z+, r16
ldi r16, P_1
st Z+, r16
ldi r16, P_2
st Z+, r16
ldi r16, P_3
st Z+, r16
ldi r16, P_4
st Z+, r16
ldi r16, space
st Z+, r16

ldi r16, dash
st Z+, r16
st Z+, r16
st Z+, r16
ldi r16, space
st Z+, r16

ldi r16, N_0
st Z+, r16
ldi r16, N_1
st Z+, r16
ldi r16, N_2
st Z+, r16
ldi r16, N_3
st Z+, r16
ldi r16, N_4
st Z+, r16
ldi r16, space
st Z+, r16

ldi r16, T_0
st Z+, r16
ldi r16, T_1
st Z+, r16
ldi r16, T_2
st Z+, r16
ldi r16, T_3
st Z+, r16
ldi r16, T_4
st Z+, r16
ldi r16, space
```

```
    st Z+, r16

    ldi r16, U_0
    st Z+, r16
    ldi r16, U_1
    st Z+, r16
    ldi r16, U_2
    st Z+, r16
    ldi r16, U_3
    st Z+, r16
    ldi r16, U_4
    st Z+, r16
    ldi r16, space
    st Z+, r16

    ldi r16, S_0
    st Z+, r16
    ldi r16, S_1
    st Z+, r16
    ldi r16, S_2
    st Z+, r16
    ldi r16, S_3
    st Z+, r16
    ldi r16, S_4
    st Z+, r16
    ldi r16, space
    st Z+, r16

    ldi r16, T_0
    st Z+, r16
    ldi r16, T_1
    st Z+, r16
    ldi r16, T_2
    st Z+, r16
    ldi r16, T_3
    st Z+, r16
    ldi r16, T_4
    st Z+, r16
    ldi r16, space
    st Z+, r16
    st Z+, r16

variable_string:
    ; This is the part of the string that changes when a button (note) is pressed
    ; ZL = ZL + 5, X_ ___ Hz
    ldi r16, 0xFF
    st Z+, r16
    st Z+, r16
    st Z+, r16
    st Z+, r16
    st Z+, r16
    ldi r16, space
    st Z+, r16

    ; ZL = ZL + 5, _x ___ Hz
    ldi r16, 0xFF
    st Z+, r16
    st Z+, r16
    st Z+, r16
    st Z+, r16
    st Z+, r16
    ldi r16, space
    st Z+, r16

    ; ZL = ZL + 5, __ X__ Hz
    ldi r16, 0xFF
    st Z+, r16
    st Z+, r16
    st Z+, r16
    st Z+, r16
    st Z+, r16
    ldi r16, space
    st Z+, r16

    ; ZL = ZL + 5, __ _X_ Hz
```

```
  ldi r16, 0xFF
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  ldi r16, space
  st Z+, r16

  ; ZL = ZL + 5, __ __X Hz
  ldi r16, 0xFF
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  ldi r16, space
  st Z+, r16

  ldi r16, H_0
  st Z+, r16
  ldi r16, H_1
  st Z+, r16
  ldi r16, H_2
  st Z+, r16
  ldi r16, H_3
  st Z+, r16
  ldi r16, H_4
  st Z+, r16
  ldi r16, space
  st Z+, r16

  ldi r16, z_0
  st Z+, r16
  ldi r16, z_1
  st Z+, r16
  ldi r16, z_2
  st Z+, r16
  ldi r16, z_3
  st Z+, r16
  ldi r16, z_4
  st Z+, r16
  ldi r16, space
  st Z+, r16

  mov end_of_string, ZL      ; r29 stores the end of the string

  ldi r16, space
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16
  st Z+, r16

get_data:
  ldi has_a_button_been_pressed, 0x00    ; reset the flag
  sbis PINB, Ab                          ; if Ab button is pressed
  call generateToneAb                    ; generate its tone
  sbis PINB, A                           ; if A  button is pressed
  call generateToneA                     ; generate its tone
  sbis PINB, As                          ; if As button is pressed
  call generateToneAs                    ; generate its tone

  sbrs has_a_button_been_pressed, 0      ; if the flag is 0,
  rjmp get_data                          ; go back to call data
  call update_string                     ; else update the string

  ldi ZL, 0
  ldi r25, 255  ; registers used as time
  ldi r26, 255  ; counters

loop:
  call print_data
```

```
  subi r25, 1
  sbci r26, 0
  cpi r25,0x00              ; Compare time_counter with 0
  breq shift_by_one         ; if time_counter == 0 go to shift_by_one
  rjmp loop                 ; else repeat the loop

  shift_by_one:
    ldi r26, 255            ; reset the time counter
    inc ZL                  ; update the Z pointer (This shifts the output by 1)
    cp ZL, end_of_string    ; compare Z with end_of_string
    breq get_data           ; if equal go back to get_data
    rjmp loop               ; else continue printing

delay_50us:
  ldi  r27, 2
    ldi  r28, 9
  L1: dec  r28
    brne L1
    dec  r27
    brne L1
    ret

update_string:  ; this function change the values of the "variable_string" to the corresponding note
  lds r16, tone_letter
  sts param1, r16
  ldi ZL, tone_letter_print
  call set_note

  lds r16, tone_sign
  sts param1, r16
  ldi ZL, tone_sign_print
  call set_note

  lds r16, freq_hundreds
  sts param1, r16
  ldi ZL, freq_hundreds_print
  call set_digit

  lds r16, freq_decimals
  sts param1, r16
  ldi ZL, freq_decimals_print
  call set_digit

  lds r16, freq_units
  sts param1, r16
  ldi ZL, freq_units_print
  call set_digit

  ret

generateToneAb:       ; the values corresponding to Ab are stored
  ldi r18, 8
  sts freq_hundreds, r18
  ldi r18, 3
  sts freq_decimals, r18
  ldi r18, 1
  sts freq_units, r18
  ldi r18, 'A'
  sts tone_letter, r18
  ldi r18, 'b'
  sts tone_sign, r18
  ldi has_a_button_been_pressed, 0x01     ; flag = 1
  ret

generateToneA:       ; the values corresponding to A  are stored
  ldi r18, 8
  sts freq_hundreds, r18
  ldi r18, 8
  sts freq_decimals, r18
  ldi r18, 0
  sts freq_units, r18
  ldi r18, 'A'
  sts tone_letter, r18
  ldi r18, 'o'
```

```
  sts tone_sign, r18
  ldi has_a_button_been_pressed, 0x01    ; flag = 1
  ret

generateToneAs:      ; the values corresponding to As are stored
  ldi r18, 9
  sts freq_hundreds, r18
  ldi r18, 3
  sts freq_decimals, r18
  ldi r18, 2
  sts freq_units, r18
  ldi r18, 'A'
  sts tone_letter, r18
  ldi r18, 's'
  sts tone_sign, r18
  ldi has_a_button_been_pressed, 0x01    ; flag = 1
  ret

print_data:
  ld r21, Z
  out PORTD, r21

  cbi PORTB, dataPin
  sbi PORTB, clockPin    ; upload '0' to the shifter
  cbi PORTB, clockPin
  cbi PORTB, latchPin    ; the content of the  Shift Register is copied into the Storage/Latch Register
  sbi PORTB, latchPin    ; latch register = 0b01111111

  call delay_50us

  ldd r21, Z+1
  out PORTD, r21

  sbi PORTB, dataPin
  sbi PORTB, clockPin    ; upload '1' to the shifter
  cbi PORTB, clockPin
  cbi PORTB, latchPin    ; the content of the  Shift Register is copied into the Storage/Latch Register
  sbi PORTB, latchPin    ; latch register = 0b10111111

  call delay_50us

  ldd r21, Z+2
  out PORTD, r21

  sbi PORTB, clockPin    ; upload '1' to the shifter
  cbi PORTB, clockPin
  cbi PORTB, latchPin    ; the content of the  Shift Register is copied into the Storage/Latch Register
  sbi PORTB, latchPin    ; latch register = 0b11011111

  call delay_50us

  ldd r21, Z+3
  out PORTD, r21

  sbi PORTB, clockPin    ; upload '1' to the shifter
  cbi PORTB, clockPin
  cbi PORTB, latchPin    ; the content of the  Shift Register is copied into the Storage/Latch Register
  sbi PORTB, latchPin    ; latch register = 0b11101111

  call delay_50us

  ldd r21, Z+4
  out PORTD, r21

  sbi PORTB, clockPin    ; upload '1' to the shifter
  cbi PORTB, clockPin
  cbi PORTB, latchPin    ; the content of the  Shift Register is copied into the Storage/Latch Register
  sbi PORTB, latchPin    ; latch register = 0b11110111

  call delay_50us

  ldd r21, Z+5
  out PORTD, r21

  sbi PORTB, clockPin    ; upload '1' to the shifter
```

```
    cbi PORTB, clockPin
    cbi PORTB, latchPin     ; the content of the  Shift Register is copied into the Storage/Latch Register
    sbi PORTB, latchPin     ; latch register = 0b11111011

    call delay_50us

    ldd r21, Z+6
    out PORTD, r21

    sbi PORTB, clockPin     ; upload '1' to the shifter
    cbi PORTB, clockPin
    cbi PORTB, latchPin     ; the content of the  Shift Register is copied into the Storage/Latch Register
    sbi PORTB, latchPin     ; latch register = 0b11111101

    call delay_50us

    ldd r21, Z+7
    out PORTD, r21

    sbi PORTB, clockPin     ; upload '1' to the shifter
    cbi PORTB, clockPin
    cbi PORTB, latchPin     ; the content of the  Shift Register is copied into the Storage/Latch Register
    sbi PORTB, latchPin     ; latch register = 0b11111110

    call delay_50us
    ret

set_note:
    lds r16, param1
    cpi r16, 'A'        ; compare param1 with the values 'A','b','o','s' to decide
    breq print_A        ; which char to show
    cpi r16, 'b'
    breq print_b
    cpi r16, 'o'
    breq print_o
    cpi r16, 's'
    breq print_hash
    print_A:
      ldi r16, A_0
      st Z+, r16
      ldi r16, A_1
      st Z+, r16
      ldi r16, A_2
      st Z+, r16
      ldi r16, A_3
      st Z+, r16
      ldi r16, A_4
      st Z+, r16
      rjmp switch_end
    print_b:
      ldi r16, b_0
      st Z+, r16
      ldi r16, b_1
      st Z+, r16
      ldi r16, b_2
      st Z+, r16
      ldi r16, b_3
      st Z+, r16
      ldi r16, space
      st Z+, r16
      rjmp switch_end
    print_o:
      ldi r16, degree_0
      st Z+, r16
      ldi r16, degree_1
      st Z+, r16
      ldi r16, degree_2
      st Z+, r16
      ldi r16, degree_3
      st Z+, r16
      ldi r16, space
      st Z+, r16
      rjmp switch_end
    print_hash:
      ldi r16, hash_0
```

```
    st Z+, r16
    ldi r16, hash_1
    st Z+, r16
    ldi r16, hash_2
    st Z+, r16
    ldi r16, hash_3
    st Z+, r16
    ldi r16, hash_4
    st Z+, r16
  switch_end:
    ret

set_digit:
  lds r16, param1
  cpi r16, 0x00       ; compare param1 with values between 0-9 to decide
  breq print_0        ; which number to show
  cpi r16, 0x01
  breq print_1
  cpi r16, 0x02
  breq print_2
  cpi r16, 0x03
  breq print_3_long_jump
  cpi r16, 0x04
  breq print_4_long_jump
  cpi r16, 0x05
  breq print_5_long_jump
  cpi r16, 0x06
  breq print_6_long_jump
  cpi r16, 0x07
  breq print_7_long_jump
  cpi r16, 0x08
  breq print_8_long_jump
  cpi r16, 0x09
  breq print_9_long_jump
  print_3_long_jump:
    jmp print_3
  print_4_long_jump:
    jmp print_4
  print_5_long_jump:
    jmp print_5
  print_6_long_jump:
    jmp print_6
  print_7_long_jump:
    jmp print_7
  print_8_long_jump:
    jmp print_8
  print_9_long_jump:
    jmp print_9
  print_0:
    ldi r16, n0_0
    st Z+, r16
    ldi r16, n0_1
    st Z+, r16
    ldi r16, n0_2
    st Z+, r16
    ldi r16, n0_3
    st Z+, r16
    ldi r16, n0_4
    st Z+, r16
    rjmp switch_end2
  print_1:
    ldi r16, n1_0
    st Z+, r16
    ldi r16, n1_1
    st Z+, r16
    ldi r16, n1_2
    st Z+, r16
    ldi r16, n1_3
    st Z+, r16
    ldi r16, n1_4
    st Z+, r16
    rjmp switch_end2
  print_2:
    ldi r16, n2_0
    st Z+, r16
```

```
  ldi r16, n2_1
  st Z+, r16
  ldi r16, n2_2
  st Z+, r16
  ldi r16, n2_3
  st Z+, r16
  ldi r16, n2_4
  st Z+, r16
  rjmp switch_end2
print_3:
  ldi r16, n3_0
  st Z+, r16
  ldi r16, n3_1
  st Z+, r16
  ldi r16, n3_2
  st Z+, r16
  ldi r16, n3_3
  st Z+, r16
  ldi r16, n3_4
  st Z+, r16
  rjmp switch_end2
print_4:
  ldi r16, n4_0
  st Z+, r16
  ldi r16, n4_1
  st Z+, r16
  ldi r16, n4_2
  st Z+, r16
  ldi r16, n4_3
  st Z+, r16
  ldi r16, n4_4
  st Z+, r16
  rjmp switch_end2
print_5:
  ldi r16, n5_0
  st Z+, r16
  ldi r16, n5_1
  st Z+, r16
  ldi r16, n5_2
  st Z+, r16
  ldi r16, n5_3
  st Z+, r16
  ldi r16, n5_4
  st Z+, r16
  rjmp switch_end2
print_6:
  ldi r16, n6_0
  st Z+, r16
  ldi r16, n6_1
  st Z+, r16
  ldi r16, n6_2
  st Z+, r16
  ldi r16, n6_3
  st Z+, r16
  ldi r16, n6_4
  st Z+, r16
  rjmp switch_end2
print_7:
  ldi r16, n7_0
  st Z+, r16
  ldi r16, n7_1
  st Z+, r16
  ldi r16, n7_2
  st Z+, r16
  ldi r16, n7_3
  st Z+, r16
  ldi r16, n7_4
  st Z+, r16
  rjmp switch_end2
print_8:
  ldi r16, n8_0
  st Z+, r16
  ldi r16, n8_1
  st Z+, r16
  ldi r16, n8_2
```

```
    st Z+, r16
    ldi r16, n8_3
    st Z+, r16
    ldi r16, n8_4
    st Z+, r16
    rjmp switch_end2
  print_9:
    ldi r16, n9_0
    st Z+, r16
    ldi r16, n9_1
    st Z+, r16
    ldi r16, n9_2
    st Z+, r16
    ldi r16, n9_3
    st Z+, r16
    ldi r16, n9_4
    st Z+, r16
  switch_end2:
    ret

; ISR for Timer1
.ORG 0x400
T1_OV_ISR:
  sbis PINB, Ab
  rjmp Ab_pressed
  sbis PINB, A      ; if A  button is pressed
  rjmp A_pressed    ; generate its tone
  sbis PINB, As     ; if As button is pressed
  rjmp As_pressed   ; generate its tone
  ; if any button is being pressed
  cbi PORTC, buzzer
  reti
  Ab_pressed:
    in r20, PORTC
    com r20
    out PORTC, r20
    ldi r20, HIGH(0xDA65)
    sts TCNT1H, r20
    ldi r20, LOW(0xDA65)
    sts TCNT1L, r20
    reti
  A_pressed:
    in r20, PORTC
    com r20
    out PORTC, r20
    ldi r20, HIGH(0xDC7D)
    sts TCNT1H, r20
    ldi r20, LOW(0xDC7D)
    sts TCNT1L, r20
    reti
  As_pressed:
    in r20, PORTC
    com r20
    out PORTC, r20
    ldi r20, HIGH(0xDE78)
    sts TCNT1H, r20
    ldi r20, LOW(0xDE78)
    sts TCNT1L, r20
    reti
```