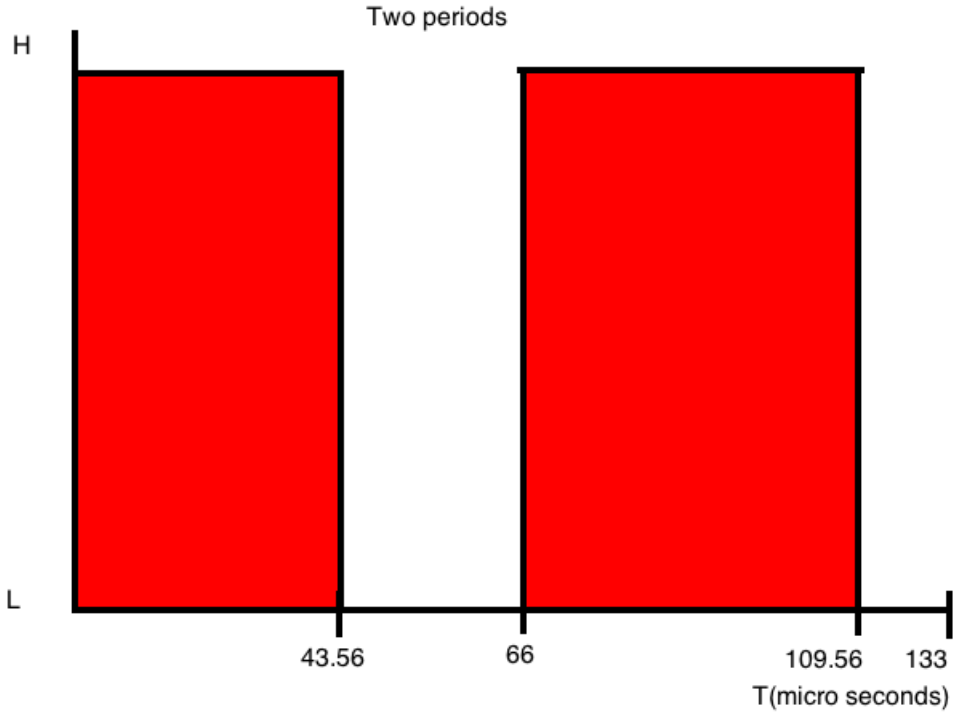
Aviles, Jean-Ralph
EEL3744
Section 1539
November 14th, 2015
Lab 7

Questions

1. Draw a 15 kHz square wave with 66% duty cycle.



2. For part A, what is the limiting factor for the precision of your frequency generation? Can your XMEGA generate some frequency ranges with higher precisions than other frequency ranges? Explain.
   The limiting factor for our frequency generation is the value of CCA. At its lowest, CCA can be 0. Using the equation in the appendix the higheset frequency we could get is 1Mhz, which is a period of $1\mu s$. I don't think the Xmega an generate more precise signals. We're getting too close to have enough clock cycles to do anything else.

3. How does the prescaler affect the way the TC system counts per clock cycle? Where are the counts stored?
   The prescaler sets how many clock ticks count for one TC count. The counts that make it through are stored in a device called a base counter.

4. Describe the difference(s) between the TCs Frequency Generation mode and its Single/Dual Slope PWM modes. Which mode(s) can be used to emulate the other(s)? How could you make a sine wave or other waveform using your XMEGA by using the timer system? Do you need to add any extra hardware? How can you produce these waveforms without extra hardware?
   The Frequency generation mode does a lot of the hard work for you, reseting the counter at a certain value, getting the timing right, while the PWM modes require a bit more intervention by the user. The PWM mode can be used to emulate the Frequency mode, but not the other way around as you can do things like change the duty cycle with the PWM mode. You can make a sine wave with a PWM system by slowly increasing the duty cycle of a signal over time to get a higher average value. You don't need any extra hardware you just need a quite involved software program to do all the math and duty cycle changes.

Problems Encountered

None this time. It went smoothly. Mainly because I used avr-gcc and never booted into my Windows VM.

Future Work/Applications

We can use the frequency generation capabilities of the Xmega to control other speakers to generate music or use PWM to control servos and motors.

Appendix

Part A

- I chose to use PORTE_PIN3 as the output port for my speaker.
- Since I'm using the Frequency Generation mode for the TC. CCA must be set to the correct value for the desired frequency. For a given frequency, we can calulate the CCA vaue required with by solving the formula given to us in the manual on page 172...

$$F_{\mathrm{FRQ}} = \frac{f_{\mathrm{clk_{per}}}}{2N(CCA+1)} \Rightarrow CCA = \frac{f_{\mathrm{clk_{per}}}}{2NF_{\mathrm{FRQ}}} - 1$$

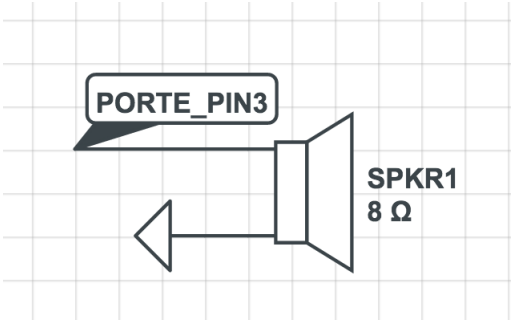
  Where F_{FRQ} is the desired frequency, CCA is the value of the CCA register, $F_{clk_{per}}$ is the system clock rate, and N is the clock prescaler.

  - For PartA to generate a 1760Hz signal, we would have to put a value of

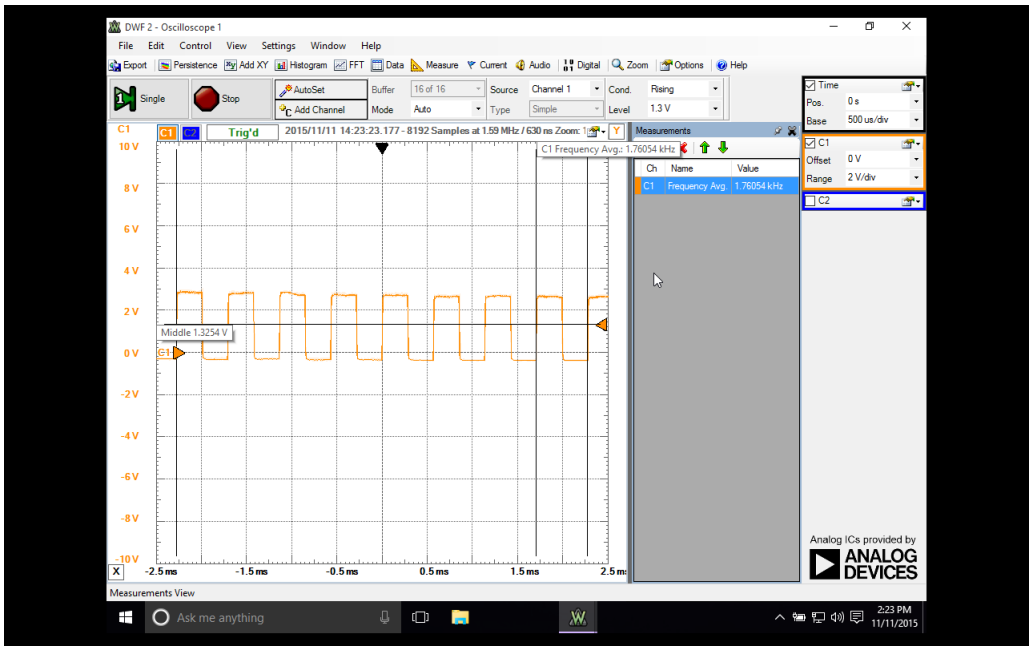
$$CCA = \frac{2 * 10^6}{2 * 1760} - 1 \Rightarrow CCA = 567$$

    into the CCA register.

- Wiring diagram for the speaker...



- Generated 1760Hz signal viewed on the DAD.



Part B

- CCA Values for Various Notes

Note	Frequency	CCA Value	Note	Frequency	CCA Value
$C_5\#$	523.25	1910.132	C_6	1046.5	954.566173
$C_5\#$	554.37	1802.849415	$C_6\#$	1108.73	900.9328421
D_5	587.33	1701.620333	D_6	1174.66	850.3101663
$D_5\#$	622.25	1606.071113	$D_6\#$	1244.51	802.5290998
E_5	659.25	1515.875237	E_6	1318.51	757.4318663
F_5	698.46	1430.721215	F_6	1396.91	714.8657322
$F_5\#$	739.99	1350.369613	$F_6\#$	1479.98	674.6848066
G_5	783.99	1274.526474	G_6	1567.98	636.7632368
$G_5\#$	830.61	1202.934458	$G_6\#$	1661.22	600.9672289
A_5	880	1135.363636	A_6	1760	567.1818182
$A_5\#$	932.33	1071.581597	$A_6\#$	1864.66	535.2907983
B_5	987.77	1011.381425	B_6	1975.53	505.1932747

Pseudocode/Flowcharts

Part A

```
def main():
    init()
    init_speaker()
    set_frequency(1760)
    while True:
        if read_switches() & 0x01:
            enable_speaker()
        else:
            disable_speaker()
```

Part B

```
def main():
    init()
    init_speaker()
    set_frequency(1760)
    while True:
        if read_switches() & 0x01:
            enable_speaker()
        else:
            disable_speaker()
```

Headers

Helpers

```
def delay_ms(ms):
  for i in range(0, ms):
    for j in range(0, 390):
      pass

def uint_string(x):
  return str(x)

def init():
  PORTH_DIR = 0b110111
  PORTH_OUTSET = 0b11011
  PORTJ_DIR = 0xFF
  PORTK_DIR = 0xFF
  EBI_CTRL = 0x01
  EBI_CS0_CTRLA = 0b0010101
  EBI_CS0_BASEADDR = 0x8000
  EBI_CS1_CTRLA = 0b100001
  EBI_CS1_BASEADDR = 0x470000
```

Keypad

```
KEYTAB = [
  ['1', '2', '3', 'A'],
  ['4', '5', '6', 'B'],
  ['7', '8', '9', 'C'],
  ['*', '0', '#', 'D']
]

def init():
  PORTF_DIR = 0xF0;
  PORTCFG_MPCMASK = 0x0F;
  PORTF_PIN0CTRL = 0x11;
  PORTF_INTCTRL = 0x03;
  PORTF_INT0MASK = 0x0F;
  PORTF_OUT = 0xF0;

char read_keypad(void) {
  for i in range(0, 4):
    PORTF_OUT = (0b0001 << i) << 4
    for j in range(0, 4):
      if PORTF_IN & (0b0001 << j) is not 0:
```

```
            return KEYTAB[i][j];
  return 'F';
}

ISR(PORTF_INT0_vect) {
  delay_ms(1)
  tmp = read_keypad();
  if tmp is not 'F':
    last_key = tmp;
  PORTF_INTFLAGS |= 0x01;
}
```

Lcd

```
cursor_pos = 0

def init():
  delay(40)
  lcd_write_cmd(0b111100)
  delay(40)
  lcd_write_cmd(0b111100)
  lcd_cmd(0b1111)
  lcd_on = True
  lcd_cmd(0b110)
  lcd_clear()

def lcd_write_c(c):
  if c is '\n':
    lcd_next_line()
  else:
    wait_lcd()
    mem_write(LCD_DATA, c)
    if cursor_pos is 16:
      cursor_pos = 40
      lcd_cmd(0x80 | cursor_pos)
    elif cursor_pos is 56
      cursor_pos = 0
      lcd_cmd(0x80)

def lcd_write_s(x):
  for c in x:
    lcd_write_c(c)

def lcd_cmd(cmd):
```

```
  wait_lcd()
  mem_write(LCD_CMD, cmd)

def lcd_clear():
  lcd_cmd(1)
  cursor_pos = 0

def lcd_busy():
  return mem_read(LCD_CMD) & 0x80 is not 0

def wait_lcd():
  while lcd_busy:
    pass

def lcd_reset_cursor():
  lcd_cmd(0b10)

def lcd_next_line():
  if cursor_pos < 40:
    cursor_pos = 40
    lcd_cmd(0x80 | cursor_pos)
  else:
    cursor_pos = 0
    lcd_cmd(0x80)

def lcd_hpos():
  if cursor_pos < 16:
    return cursor_pos
  else:
    return cursor_pos - 40

def lcd_vpos():
  if cursor_pos < 16:
    return 0
  return 1

def set_pos(h, v):
  if v is 0:
    cursor_pos = h
  else:
    cursor_pos = h + 40
  lcd_cmd(0x80 | cursor_pos)

def toggle_lcd():
  lcd_on ^= 1
```

```
  lcd_cmd(0b1111 ^ (lcd_on << 2))

def lcd_move_cursor(dir):
  switch (dir):
    case LEFT:
      if lcd_vpos() is 1 && lcd_hpos() is 0:
        set_pos(15, 0)
      elif lcd_hpos() != 0:
        set_pos(lcd_hpos() - 1, lcd_vpos())
    case RIGHT:
      if lcd_vpos() is 0 && lcd_hpos() is 15:
        set_pos(0, 1)
      elif lcd_hpos() != 15:
        set_pos(lcd_hpos() + 1, lcd_vpos())
    case UP:
      if lcd_vpos() is 1:
        set_pos(lcd_hpos(), 0)
    case DOWN:
      if lcd_vpos() is 0:
        set_pos(lcd_hpos(), 1)
```

Speaker

```
def init_speaker():
    PORTE_DIRSET = 0x08 # PIN3 Output
    TCE0_CTRLA = 0x01 # Prescaler = 1
    TCE0_CTRLD = 0x80 # Restart Waveform every Period
    TCE0_CTRLB = 0x01 # Waveform Generation

def play_note(note, duration):
    set_frequency(note)
    with speaker as s:
        # With statement automatically enables & disables speaker
        delay(duration)

def set_frequency(freq):
    TCE0_CCA = 1000000//freq - 1

def enable_speaker():
    TCE0_CTRLB |= 0x80

def disable_speaker():
    TCE0_CTRLB &= ~0x80
```

Programs

Part A

```
/*
* Lab7_PartA.c
* Generates 1760Hz Tone on Speaker
* Created: 11/11/2015 2:06:40 PM
* Author: Jean-Ralph Aviles
*/

#include <avr/io.h>

#include "helpers.h"
#include "speaker.h"

int main(void) {
  init();
  init_speaker();
  set_frequency(1760);
  while(1) {
    /* Speaker is Enabled when Switch0
     * is True */
    if (read_switches() & 0x01) {
      enable_speaker();
    } else {
      disable_speaker();
    }
  }
  return 0;
}
```

Part B

```
/*
* Lab7_PartA.c
* Generates 1760Hz Tone on Speaker
* Created: 11/11/2015 2:06:40 PM
* Author: Jean-Ralph Aviles
*/

#include <avr/io.h>

#include "helpers.h"
```

```c
#include "speaker.h"

int main(void) {
  init();
  init_speaker();
  set_frequency(1760);
  while(1) {
    /* Speaker is Enabled when Switch0
     * is True */
    if (read_switches() & 0x01) {
      enable_speaker();
    } else {
      disable_speaker();
    }
  }
  return 0;
}
```

Headers

Helpers

```c
#ifndef HELPERS_H
#define HELPERS_H

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include "ebi_driver.h"

#define  IOP_START    0x8000
#define  IOP_SIZE     0x2000
#define  SRAM32_START 0x472000
#define  SRAM32_SIZE  0x8000


void delay_ms(uint16_t ms) {
  for (uint16_t i = 0; i < ms; ++i) {
    for (uint16_t j = 0; j < 86; ++j) {
      asm volatile("nop");
    }
  }
}
```

```
char* uint_string(uint16_t n, char* s, int radix, int len) {
  for (int i = 0; i < len; ++i) {
    s[i] = '\0';
  }
  if (n == 0) {
    s[0] = '0';
    return s;
  }
  int i = 0;
  while (n > 0) {
    char tmp = (n % radix) + '0';
    if (tmp > '9') {
      tmp = tmp - '9' - 1 + 'A';
    }
    s[i++] = tmp;
    n /= radix;
  }
  len = i;
  for(i = i - 1; i >= len/2; --i) {
    char tmp = s[i];
    s[i] = s[len - 1 - i];
    s[len -1 - i] = tmp;
  }
  return s;
}


void init(void) {
  // Output address lines and chip selects
  PORTH_DIR = 0b110111; // /WE, /RE, /CS0, /CS1 outputs
  PORTH_OUTSET = 0b110011; // Output 1 to active low outputs
  PORTJ_DIR = 0xFF; // Set D7-D0 as outputs
  PORTK_DIR = 0xFF; // Set A15-A0 as outputs
  EBI_CTRL = 0x01; // Select 3 port SRAM ALE Mode
  // Switches and LEDS
  EBI_CS0_CTRLA = 0b0010101; // Set CTRLA to 8K size
  EBI_CS0_BASEADDR = (uint16_t)((IOP_START >> 8) & 0xFFFF); // Set upper
  // 32K SRAM + Block
  EBI_CS1_CTRLA = 0b100001; // Set CS1 to 64K
  EBI_CS1_BASEADDR = (uint16_t)((0x470000 >> 8) & 0xFFFF); // Set upper 1
  // Zero out LEDs
  __far_mem_write(IOP_START, 0x00);
}

uint8_t read_switches(void) {
```

```
  __far_mem_read(IOP_START);
}

void write_led(uint8_t data) {
  __far_mem_write(IOP_START, data);
}

void intmask_set(uint8_t mask) {
  PMIC_CTRL = mask;
}

void enable_int(void) {
  sei();
}

void disable_int(void) {
  cli();
}

void sleep_mode_set(uint8_t mode) {
  set_sleep_mode(mode);
}

void enable_sleep() {
  sleep_enable();
}

void sleep() {
  sleep_cpu();
}

void disable_sleep() {
  sleep_disable();
}

#endif
```

Keypad

```
#ifndef KEYPAD_H
#define KEYPAD_H

#include <avr/io.h>
#include <avr/interrupt.h>
```

```
#include "helpers.h"

char KEYTAB[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

static volatile char last_key = '\0';

void init_keypad(void);
char read_keypad(void);

void init_keypad(void) {
  PORTF_DIR = 0xF0;
  PORTCFG_MPCMASK = 0x0F;
  // High Level Interrupt on Key press.
  PORTF_PIN0CTRL = 0x11;
  PORTF_INTCTRL = 0x03;
  PORTF_INT0MASK = 0x0F;
  PORTF_OUT = 0xF0;
}

char read_keypad(void) {
  for (int i = 0; i < 4; ++i) {
    PORTF_OUT = (0b0001 << i) << 4;
    for (int j = 0; j < 4; ++j) {
      asm volatile("nop\nnop\nnop");
      if (PORTF_IN & (0b0001 << j)) {
        PORTF_OUT = 0xF0;
        return KEYTAB[i][j];
      }
    }
  }
  PORTF_OUT = 0xF0;
  return 'F';
}

ISR(PORTF_INT0_vect) {
  delay_ms(1); // Debounce
  char tmp = read_keypad();
  if (tmp != 'F') {
    last_key = tmp;
  }
```

```
  PORTF_INTFLAGS |= 0x01;
}

#endif
```

Lcd

```
#ifndef LCD_H
#define LCD_H

#include <avr/io.h>
#include "ebi_driver.h"
#include "helpers.h"

#define  LCD_CMD    0x471000
#define  LCD_DATA   0x471001

#define FALSE 0
#define TRUE 1
typedef int bool;

#define LEFT  0
#define RIGHT 1
#define DOWN  2
#define UP    3
typedef int dir;

static uint8_t cursor_pos;
static bool lcd_on = FALSE;

void init_lcd(void);
void lcd_cmd(short);
void lcd_clear(void);
bool lcd_busy(void);
void wait_lcd(void);
void lcd_write_c(char);
void lcd_write_s(char*);
void lcd_reset_cursor(void);
void lcd_next_line(void);
int lcd_hpos(void);
int lcd_vpos(void);
void set_pos(int h, int v);
void toggle_lcd(void);
void lcd_move_cursor(dir);
```

```
void init_lcd(void) {
  // Ensure it has been at least 40ms since start. See data sheet.
  delay_ms(40);
  // Two lines
  __far_mem_write(LCD_CMD, 0b111100);
  delay_ms(40);
  __far_mem_write(LCD_CMD, 0b111100);
  delay_ms(40);
  // Now, we can safely poll the BF Flag. See data sheet.
  lcd_cmd(0b1111); // Set Cursor and Display on
  wait_lcd();
  lcd_on = TRUE;
  lcd_cmd(0b110); // Cursor increments right and shifts
  lcd_clear();
}

void lcd_write_c(char c) {
  if (c == '\n') {
    lcd_next_line();
    return;
  }
  wait_lcd();
  __far_mem_write(LCD_DATA, (int)c);
  ++cursor_pos;
  if (cursor_pos == 16) {
    cursor_pos = 40;
    lcd_cmd(0x80 | cursor_pos);
  } else if (cursor_pos == 56) {
    cursor_pos = 0;
    lcd_cmd(0x80);
  }
}

void lcd_write_s(char* string) {
  while (*string != '\0') {
    lcd_write_c(*(string++));
  }
}

void lcd_cmd(short cmd) {
  wait_lcd();
  __far_mem_write(LCD_CMD, cmd);
}
```

```
void lcd_clear(void) {
  lcd_cmd(0b1);
  cursor_pos = 0;
}

bool lcd_busy(void) {
  delay_ms(5);
  return FALSE;
  int x = __far_mem_read(LCD_CMD);
  return (x & 0x80) != 0;
}

void wait_lcd(void) {
  do {
  } while(lcd_busy() != FALSE);
}

void lcd_reset_cursor(void) {
  lcd_cmd(0b10);
}

void lcd_next_line(void) {
  if (cursor_pos < 40) {
    cursor_pos = 40;
    lcd_cmd(0x80 | cursor_pos);
  } else {
    cursor_pos = 0;
    lcd_cmd(0x80);
  }
}

int lcd_hpos(void) {
  if (cursor_pos < 16) {
    return cursor_pos;
  } else {
    return cursor_pos - 40;
  }
}

int lcd_vpos(void) {
  if (cursor_pos < 16) {
    return 0;
  } else {
    return 1;
  }
```

```
}

void set_pos(int h, int v) {
  if (v == 0) {
    lcd_cmd(0x80 | h);
    cursor_pos = h;
  } else {
    if (h - lcd_hpos() == 1) {
      lcd_cmd(0x14);
      cursor_pos += 1;
    } else if (h - lcd_hpos() == -1) {
      lcd_cmd(0x10);
      cursor_pos -= 1;
    } else {
      cursor_pos = 40;
      lcd_cmd(0x80 | 40);
      for (int i = 0; i < h; ++i) {
        lcd_move_cursor(RIGHT);
      }
    }
  }
}

void toggle_lcd(void) {
  lcd_on ^= TRUE;
  lcd_cmd(0b1111 ^ (lcd_on << 2));
}

void lcd_move_cursor(dir d) {
  switch (d) {
    case LEFT:
      if (lcd_vpos() == 1 && lcd_hpos() == 0) {
        set_pos(15, 0);
      } else if (lcd_hpos() != 0) {
        set_pos(lcd_hpos() - 1, lcd_vpos());
      }
      break;
    case RIGHT:
      if (lcd_vpos() == 0 && lcd_hpos() == 15) {
        set_pos(0, 1);
      } else if (lcd_hpos() != 15) {
        set_pos(lcd_hpos() + 1, lcd_vpos());
      }
      break;
    case UP:
```

```
      if (lcd_vpos() == 1) {
        set_pos(lcd_hpos(), 0);
      }
      break;
    case DOWN:
      if (lcd_vpos() == 0) {
        set_pos(lcd_hpos(), 1);
      }
      break;
  }
}

#endif
```

Speaker

```
#ifndef SPEAKER_H
#define SPEAKER_H

#include <avr/io.h>

/* Returns CCA value for frequency, slightly tuned */
#define _frequency(x) ((int)((1000000)/(x))+10)

#define C5  523.25
#define C5S 554.37
#define D5  587.33
#define D5S 622.25
#define E5  659.25
#define F5  698.46
#define F5S 739.99
#define G5  783.99
#define G5S 830.61
#define A5  880.00
#define A5S 932.33
#define B5  987.77

#define C6  1046.50
#define C6S 1108.73
#define D6  1174.66
#define D6S 1244.51
#define E6  1318.51
#define F6  1396.91
#define F6S 1479.98
```

```
#define G6  1567.98
#define G6S 1661.22
#define A6  1760.00
#define A6S 1864.66
#define B6  1975.53

#define C7  2093.00
#define C7S 2217.46
#define D7  2349.32
#define E7  2637.02

typedef int note;

void init_speaker(void);
void play_note(note,uint16_t);
void set_frequency(float);
void enable_speaker(void);
void disable_speaker(void);

void init_speaker(void) {
  PORTE_DIRSET = 0x08;  // Pin3 Output
  TCE0_CTRLA = 0x01;    // Prescaler = CLK
  TCE0_CTRLD = 0x80;    // Restart Waveform
  TCE0_CTRLB = 0x01;    // FRQ Generation
}

void play_note(note n, uint16_t dur_ms) {
  set_frequency(n);
  enable_speaker();
  delay_ms(dur_ms);
  disable_speaker();
}

void set_frequency(float freq) {
  TCE0_CCA = _frequency(freq);
}

void enable_speaker(void) {
  TCE0_CTRLB |= 0x80;
}

void disable_speaker(void) {
  TCE0_CTRLB &= ~0x80;
}
```

```
#endif
```