Embedded Software
CS 145/145L

Caio Batista de Melo
Announcements (2022-04-28)

- Project 2 is due tomorrow!
- Mid-quarter course evaluation is due on Saturday for extra credit on Project 3
  - Completely anonymous, please provide your honest feedback :)
  - Will replace the early submission extra credit for P3
  - [https://evaluations.eee.uci.edu/takeLanding/WTWYYF](https://evaluations.eee.uci.edu/takeLanding/WTWYYF)
Project 3

Design an embedded computer centered around the ATMega32 microcontroller. For input: use a keypad; For output: use an LCD and a speaker.

Write a C program that implements a music player. Your music player should be able to play musical notes stored in its memory.

https://canvas.eee.uci.edu/courses/45047/assignments/929272

Project 3 is short! It’s due next week (2022-05-06)!!
Basic requirements (100%)

- Plays a sound which is neither croaky nor severely distorted. (We understand this is digital music). Player should support start/stop through button press. (65% for functionality + 30% for quality).
- LCD displays the name of the song currently playing (5%)

Extra credits (20%)

- Implements pitch control (at least 3 levels) (5%)
- Implements tempo control (at least 3 levels) (5%)
- Plays multiple songs, and supports user selection between each song (5%)
- Complete the Mid-quarter evaluation (5%)

https://canvas.eee.uci.edu/courses/45047/assignments/929272
Sound is an alternating signal! Its frequency determines the pitch!

Frequency (F) = 1/P

20Hz to 20 KHz

- The speaker’s whole diaphragm changes according to the voltage applied.
- Thus vibration of pressure (technically) or sound is generated by alternating this voltage.
- But our AVR cannot provide a purely analog signal as shown.
- We have voltages in the digital nature in the form of 0s and 1s.
Use of Digital Signal

Sound

The momentum of diaphragm’s motion will help it oscillate.
Initial Layout

AVR

GPIO

PB3

SPEAKER
Some improvements in the Layout

- The speaker has an impedance of its own
- The capacitor charges during the positive cycle and the charging speed is decided by the R and C combination
- In the negative cycle or 0 cycle in digital terms, the capacitor discharges again as per the R and C network
- Thus smoothening of the square wave takes place
For extra credit you probably need a keypad. Standard project could use a single button.
• Notes can be defined as a combination of frequency and duration

• Musicians abstract this out using symbols.
  - E.g.,: A, ½ Time

• It is believed that most of the musics on the planet can be played using 12 frequencies and their variations.

Song for Project 3

MUSIC

Song for Project

Sequence of Notes

Frequency (Hz), for example 440HZ

Duration(Seconds), for example 2 seconds
Decoding Notes

(440 Hz, 1 sec)
(466 Hz, 2 sec)
(490 Hz, 0.5 sec)
Musical Notes Resources

https://en.wikipedia.org/wiki/Musical_note
https://www.szynalski.com/tone-generator/

https://www.musictheory.net/lessons/11
Example Music

HINT: You can try searching for the song you want + “midi” to try to find a note sequence. Example: to find the above one, I searched for “shooting stars midi”
https://www.google.com/search?q=shooting+stars+midi
typedef enum {
    A, As, B, C, Cs, D, Ds, E, F, Fs, G, Gs
} Note;
**Code Layout**

```c
typedef enum {
    W, H, Q, E
} Duration;
```
typedef enum {
    A, As, B, C, Cs, D, Ds, E, F, Fs, G, Gs
} Note;

typedef enum {
    W, H, Q, E
} Duration;

typedef struct {
    Note note;
    Duration duration;
} PlayingNote;

Anything wrong?
Two enums with the same value!
```c
typedef enum {
    A, As, B, C, Cs, D, Ds, Ee, F, Fs, G, Gs
} Note;

typedef enum {
    W, H, Q, Ei
} Duration;

typedef struct {
    Note note;
    Duration duration;
} PlayingNote;
```
PlayingNote shooting_stars[] = {
    {Ds, W},
    /* Wait for half */
    {Ds, H},
    {Ee, H},
    /* Wait for half */
    {B, Q},
    /* Wait for quarter */
    {Gs, Q}
    /* Keep going... */
};
Main Function

```c
int main () {
    while (1) {
        play_song(shooting_stars, N);
    }
    return 0;
}
```

The sequence we defined previously.

Number of notes in our song.

Play it forever!

How do we add a button input? What changes are needed for multiple songs?
play_song function

```c
void play_song(const PlayingNote song[], int length) {
    int i;
    for (i = 0; i < length; i++) {
        play_note(&song[i]);
    }
}
```

Can we do a loop like the one for strings? 

\[ \text{while (note = *song++)} \]

Why not?
```c
void play_note(const PlayingNote* note) {
    int i, k;
    for (i = 0; i < k; i++) {
        SET_BIT(PORTB, 3);
        wait(TH);
        CLR_BIT(PORTB, 3);
        wait(TL);
    }
}
```

Create $k$ ups and downs

$F = 1 / P$ (you know $F$)

$P = TH + TL$

$TH = TL$

$k = \text{Duration} / P$
Notes Frequencies

https://en.wikipedia.org/wiki/Musical_note
# Notes Frequencies

<table>
<thead>
<tr>
<th>Note</th>
<th>Offset</th>
<th>Frequency (Hz)</th>
<th>Period (s)</th>
<th>TH / TL (s)</th>
<th>Wait (1ms resolution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>440.00</td>
<td>0.002272727273</td>
<td>0.001136363636</td>
<td>1</td>
</tr>
<tr>
<td>A#</td>
<td>1</td>
<td>466.16</td>
<td>0.002145168892</td>
<td>0.001072584446</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>493.88</td>
<td>0.002024769814</td>
<td>0.001012384907</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>523.25</td>
<td>0.001911128216</td>
<td>0.000955564108</td>
<td>0</td>
</tr>
<tr>
<td>C#</td>
<td>4</td>
<td>554.37</td>
<td>0.001803864832</td>
<td>0.000901932415</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>587.33</td>
<td>0.001702621678</td>
<td>0.000851310839</td>
<td>0</td>
</tr>
<tr>
<td>D#</td>
<td>6</td>
<td>622.25</td>
<td>0.001607060866</td>
<td>0.000803530433</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>659.26</td>
<td>0.001516863471</td>
<td>0.000758431735</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>8</td>
<td>698.46</td>
<td>0.001431728466</td>
<td>0.000715864232</td>
<td>0</td>
</tr>
<tr>
<td>F#</td>
<td>9</td>
<td>739.99</td>
<td>0.001351371722</td>
<td>0.000675685860</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
<td>783.99</td>
<td>0.001275525055</td>
<td>0.000637762527</td>
<td>0</td>
</tr>
<tr>
<td>G#</td>
<td>11</td>
<td>830.61</td>
<td>0.001203935334</td>
<td>0.000601967667</td>
<td>0</td>
</tr>
</tbody>
</table>

Cannot tell them apart!

Maybe we can have a finer timer?
<table>
<thead>
<tr>
<th>Note</th>
<th>Offset</th>
<th>Frequency (Hz)</th>
<th>Period (s)</th>
<th>TH / TL (s)</th>
<th>Wait (0.1ms resolution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>440.00</td>
<td>0.002272727273</td>
<td>0.001136363636</td>
<td>11</td>
</tr>
<tr>
<td>A#</td>
<td>1</td>
<td>466.16</td>
<td>0.002145168892</td>
<td>0.001072584446</td>
<td>11</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>493.88</td>
<td>0.002024769814</td>
<td>0.001012384907</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>523.25</td>
<td>0.001911128216</td>
<td>0.000955564108</td>
<td>10</td>
</tr>
<tr>
<td>C#</td>
<td>4</td>
<td>554.37</td>
<td>0.001803864832</td>
<td>0.000901932415</td>
<td>9</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>587.33</td>
<td>0.001702621678</td>
<td>0.000851310839</td>
<td>9</td>
</tr>
<tr>
<td>D#</td>
<td>6</td>
<td>622.25</td>
<td>0.001607060866</td>
<td>0.000803530433</td>
<td>8</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>659.26</td>
<td>0.001516863471</td>
<td>0.000758431735</td>
<td>8</td>
</tr>
<tr>
<td>F</td>
<td>8</td>
<td>698.46</td>
<td>0.001431728466</td>
<td>0.000715864232</td>
<td>7</td>
</tr>
<tr>
<td>F#</td>
<td>9</td>
<td>739.99</td>
<td>0.001351371722</td>
<td>0.000675685860</td>
<td>7</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
<td>783.99</td>
<td>0.001275525055</td>
<td>0.000637762527</td>
<td>6</td>
</tr>
<tr>
<td>G#</td>
<td>11</td>
<td>830.61</td>
<td>0.001203935334</td>
<td>0.000601967667</td>
<td>6</td>
</tr>
</tbody>
</table>

Still can’t tell some apart!
Maybe we can change frequencies?
# Notes Frequencies - Down an Octave

<table>
<thead>
<tr>
<th>Note</th>
<th>Offset</th>
<th>Frequency (Hz)</th>
<th>Period (s)</th>
<th>TH / TL (s)</th>
<th>Wait (0.1ms resolution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>220.00</td>
<td>0.00454545454545</td>
<td>0.002272727273</td>
<td>23</td>
</tr>
<tr>
<td>A#</td>
<td>1</td>
<td>233.08</td>
<td>0.004290337785</td>
<td>0.002145168892</td>
<td>21</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>246.94</td>
<td>0.004049539628</td>
<td>0.002024769814</td>
<td>20</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>261.63</td>
<td>0.003822256433</td>
<td>0.001911128216</td>
<td>19</td>
</tr>
<tr>
<td>C#</td>
<td>4</td>
<td>277.18</td>
<td>0.003607729664</td>
<td>0.001803864832</td>
<td>18</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>293.66</td>
<td>0.003405243357</td>
<td>0.001702621678</td>
<td>17</td>
</tr>
<tr>
<td>D#</td>
<td>6</td>
<td>311.13</td>
<td>0.003214121733</td>
<td>0.001607060866</td>
<td>16</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>329.63</td>
<td>0.003033726941</td>
<td>0.001516863471</td>
<td>15</td>
</tr>
<tr>
<td>F</td>
<td>8</td>
<td>349.23</td>
<td>0.002863456932</td>
<td>0.001431728466</td>
<td>14</td>
</tr>
<tr>
<td>F#</td>
<td>9</td>
<td>369.99</td>
<td>0.002702743443</td>
<td>0.001351371722</td>
<td>14</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
<td>392.00</td>
<td>0.00255105011</td>
<td>0.001275525055</td>
<td>13</td>
</tr>
<tr>
<td>G#</td>
<td>11</td>
<td>415.30</td>
<td>0.002407870669</td>
<td>0.001203935334</td>
<td>12</td>
</tr>
</tbody>
</table>

Can tell all most of them apart!

For our use-case it’s probably ok :)

But you could use a finer resolution!

How do you get these values in your code?
# Frequency Mapping

<table>
<thead>
<tr>
<th>Note</th>
<th>Offset</th>
<th>Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>220.00</td>
</tr>
<tr>
<td>A#</td>
<td>1</td>
<td>233.08</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>246.94</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>261.63</td>
</tr>
<tr>
<td>C#</td>
<td>4</td>
<td>277.18</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>293.66</td>
</tr>
<tr>
<td>D#</td>
<td>6</td>
<td>311.13</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>329.63</td>
</tr>
<tr>
<td>F</td>
<td>8</td>
<td>349.23</td>
</tr>
<tr>
<td>F#</td>
<td>9</td>
<td>369.99</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
<td>392.00</td>
</tr>
<tr>
<td>G#</td>
<td>11</td>
<td>415.30</td>
</tr>
</tbody>
</table>

1. Store only the original frequency (220Hz) and use the formula \(2^{(n/12)} \times 220\);

   or

2. Store these values as constants and use them as needed
   a. Could also store period, TH, number of waits, etc.

Which approach is better?

*It depends on your application!*
AVR Resolution

```c
void avr_wait(unsigned short msec) {
    TCCR0 = 3;
    while (msec--) {
        TCNT0 = (unsigned char)(256 - (XTAL_FRQ / 64) * 0.001);
        SET_BIT(TIFR, TOV0);
        while (!GET_BIT(TIFR, TOV0));
    }
    TCCR0 = 0;
}
```

Check our slides about timers!
Make a new function or fix existing code that uses the 1ms resolution (e.g., lcd_init)
See you next time :) 

Q & A