Real-time LED Music Visualizer

Team: GoRTU
Jisoo Kim, Jiewen Sun, Pierre Karashchuk, Baihong Jin

Mentor: Michael Zimmer
Motivation

● **LED Music Visualizer**
  ○ Generating sound with square waves / MIDI
  ○ 7 LED strips representing different notes
  ○ Corresponding LED strips glowing with the music

● **RTU: FlexPRET**
  ○ Efficient
  ○ Precise timing control
  ○ Multitasking
Results
WorkFlow

FlexPRET Specification

```
elf executable
01 10044155 00
00 00000000 10
01 01001100 10
00 00001000 00
0000 10031212 0100
1001 10000001 0001
0000 00000000 1000
```

LabVIEW™

FPGA

MicroBlaze overlay software code

Build specification

MyRIO

NeoPixel

Speaker

MIDI
FlexPRET

- Timing control is not an easy task on conventional processors
  - Using timed interrupt as well as other complex mechanisms
- FlexPRET - Better timing control enabled from the architecture level
  - Exploiting cycle-level accuracy (~10ns on 100MHz FlexPRET)
  - Better isolation between different threads
  - More user-friendly programming interfaces

```c
time = get_time(); // Get current time
delay_ns(10000);
delay_until(time+100000);
```
Sound Generation

- Method 1: Generate sound with square wave
  - toggling GPIO pin with certain frequencies for different notes
  - using arrays for period and duration of each note
Sound Generation

- Method 2: Generate sound on computer using MIDI
  - send bytes according to MIDI protocol
    - asynchronous serial interface
    - fixed to 31.25 kbit/sec bitrate
    - currently using MIDI channel 3
    - send least significant bit first
  - sequence of 3 bytes for note on/off
    - note on/off + MIDI channel (e.g. 0x93)
    - note pitch (e.g. 0x40)
    - velocity 0-126 (can be translated into volume)

Note on for channel 3:
0x93 -> 1001 0011
actual sequence: 0 1100 1001 1
Song Pattern Generation

- Songs are represented as sequence of notes and durations (in second)
- Using a python script, we convert the pattern into several arrays where each contains information for duration and notes in different format (e.g. number of cycles, period in nanoseconds, MIDI pitch representation)

A4 0.70
Mute 0.05
A4 1.5
G4# 0.75
G4 0.75

```cpp
unsigned int note[SONG_LENGTH] = {1136363, 50000000, 1136363, 1203948, 1275510};
unsigned int duration[SONG_LENGTH] = {154, 1, 330, 155, 147};
unsigned int duration_ns[SONG_LENGTH] = {349999958, 50000000, 749999910, 373224035, 374999940};
char note_byte[SONG_LENGTH] = {0x45, 0, 0x45, 0x44, 0x43};
```
A4 0.70
Mute 0.05
A4 1.5
G4# 0.75
G4 0.75

unsigned int note[SONG_LENGTH] = {1136363, 50000000, 1136363, 1203948, 1275510};
unsigned int duration[SONG_LENGTH] = {154, 1, 330, 155, 147};
unsigned int duration_ns[SONG_LENGTH] = {349999958, 50000000, 749999910, 373224035, 374999940};
char note_byte[SONG_LENGTH] = {0x45, 0, 0x45, 0x44, 0x43};
NeoPixel Spec

Data transfer time (TH+TL=1.25μs±600ns)

<table>
<thead>
<tr>
<th></th>
<th>0 code, high voltage time</th>
<th>0.35μs</th>
<th>±150ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0H</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1H</td>
<td>1 code, high voltage time</td>
<td>0.7μs</td>
<td>±150ns</td>
</tr>
<tr>
<td>T0L</td>
<td>0 code, low voltage time</td>
<td>0.8μs</td>
<td>±150ns</td>
</tr>
<tr>
<td>T1L</td>
<td>1 code, low voltage time</td>
<td>0.6μs</td>
<td>±150ns</td>
</tr>
<tr>
<td>RES</td>
<td>low voltage time</td>
<td>Above 50μs</td>
<td></td>
</tr>
</tbody>
</table>

Source: WS2812 Datasheet

Source: Tim’s Blog
https://cpldcpu.wordpress.com/2014/01/14/light_ws2812-library-v2-0-part-i-understanding-the-ws2812/

Real tolerance: +/- ~215 ns
Neopixel Timing
Neopixel Timing

How many times can we set LEDs between notes?
~30us per LED
17*6 = 102 LEDS
102 * 30 = 3060 us = ~3ms per update
~100ms per note so about 33 updates

Using updated timing, we can send signals 7 times faster, so we can 33*7 = 231 updates
Simulator
Neopixel Driver
Acknowledgement

Sincere appreciation towards our Professors, GSIs, and especially our nice mentor - Michael Zimmer.

Thank you for listening. Go Bears!