GRAND VALLEY
STATE UNIVERSITY®
SEYMOUR AND ESTHER PADNOS
COLLEGE OF ENGINEERING
AND COMPUTING

EGR680 High Level Implementation on FPGA

Final Project

PYNQ Embedded Design using Jupyter Notebooks

Professor: Dr. C. Parikh

Student: Dimitri Häring

December 02, 2018

Contents

1 Introduction **3**

2 Design **4**

2.1 Part I - RGB LED Driver 4

2.2 Part II - LED Groove Bar 7

2.3 Part III - Music Synthesizer 7

2.3.1 MicroBlaze Softcore for PMODA 8

2.3.2 C code in MicroBlaze to Play a Melody 8

2.3.3 Python melody 10

2.3.4 MicroBlaze wrapper behavior 12

3 Conclusion **12**

4 Appendix **13**

4.1 Python code Listings Part I - RGB LED Driver 13

4.2 Python code Listings Part II - LED Groove Bar 25

4.3 Python code Listings Part III - Music Synthesizer 26

List of Figures

1 Xilinx PYNQ-Z1 development board SoC [1]. 3

2 PYNQ-Z1 block diagram of the MicroBlaze subsystem [2]. 3

3 Jupiter Notebook terminal, copy a file. 5

4 Jupiter Notebook GUI to control the green LEDs and a RGB LED. 7

5 Groove LED Bar program output. 7

6 GUI for an simple music synthesizer that allows the composer to design a melody. 10

7 GUI for an simple music synthesizer that allows the composer to design a melody. 11

8 Inconsistent wrapping of C functions. 12

Listings

1 Python code changed on line 45 of file __init__.py. 4

2 Python code changed on line 99 of file base.py. 4

3 Jupiter Notebook terminal, copy a file. 4

4 RGB LED driver PWM method. 5

5 RGB LED driver PWM overwiev. 6

6 Magic microblaze command. 8

7 Magic microblaze C code for buzzer and ledbar initialization 8

8 Magic microblaze C code for playing a note. 8

9 Magic microblaze C code for playing the A-Team theme. 9

10 Magic microblaze C code for playing the A-Team theme. 9

11 Part I - Jupyter Notebook file __init__ saved as *.py file. 13

12 Part I - Jupyter Notebook file base saved as *.py file. 14

13 Part I - Jupyter Notebook file myrgbled saved as *.py file. 16

14 Part I - Jupyter Notebook file LED_ctrl_myrgbled saved as *.py file. 21

15 Part II - LED Groove Bar Python code. 25

16 Part III - Jupyter Notebook file MusicSynthesizer saved as *.py file. 26

1 Introduction

The goal of final project is to familiarize the student with the groove connector and the possible interface methods provided with MicroBlaze and low level C code. The student will also write a software pulse width modulation (PWM). The outlined tasks are build on the PYNQ platform shown in Figure 1. The MicroBlaze system is placed in between of the peripherals and the Zynq processing system (PS) as shown in Figure 2. The MicroBlaze is in fact an from Xilinx developed softcore.

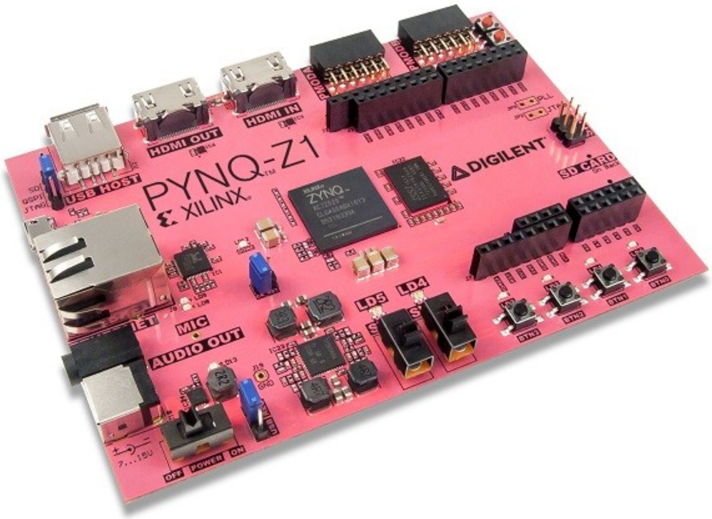


Figure 1: Xilinx PYNQ-Z1 development board SoC [1].

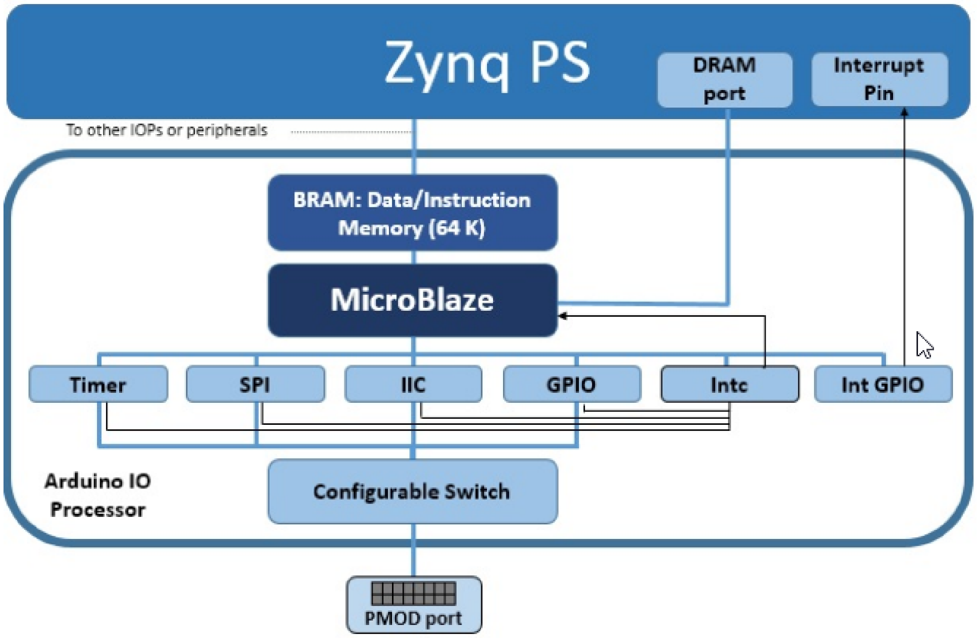


Figure 2: PYNQ-Z1 block diagram of the MicroBlaze subsystem [2].

2 Design

In this section the design and decisions that where made to achieve the laboratory are discussed.

2.1 Part I - RGB LED Driver

From project description, create the RGB LED color mixer using the ipywidgets integer or float slider. In theory, you should be able to create an infinite amount of colors with the combination of red, blue, and green LEDs. However, with digital electronics, we are limited to the width of the driving data bus or processing system to create the colors. The Project description states the following [3]

- Create individual methods for enabling/disabling RED, GREEN and BLUE color of the RGB LED.
- PWM functionality for color mixing of the RGB LED.
- Create 3 color mixing sliders using ipywidgets. One slider per color (R-G-B).
- Create toggle buttons for all four of the green LEDs using ipywidgets.
- Create a way to set a flashing rate of the green LEDs using ipywidgets.
- Create a neat and organized GUI for all of the LED functionality.

First make a back up of all files that are to be modified. The files saved are __init__.py, base.py, and rgbled.py. This is done with connecting a network drive to the Python Productivity for ZYNQ (PYNQ) platform and copy the files over to an computer. The reason to do so is that the files can not be accessed over the web browser. Listening 1 shows the changes made in the init file. This file imports the renamed class myrgbled by startup of the kernel. Listing 2 shows how the base file is modified so that instead of the original rgbled driver the modified driver myrbgbled is used.

```python
from .myrgbled import MYRGBLED
```

Listing 1: Python code changed on line 45 of file __init__.py.

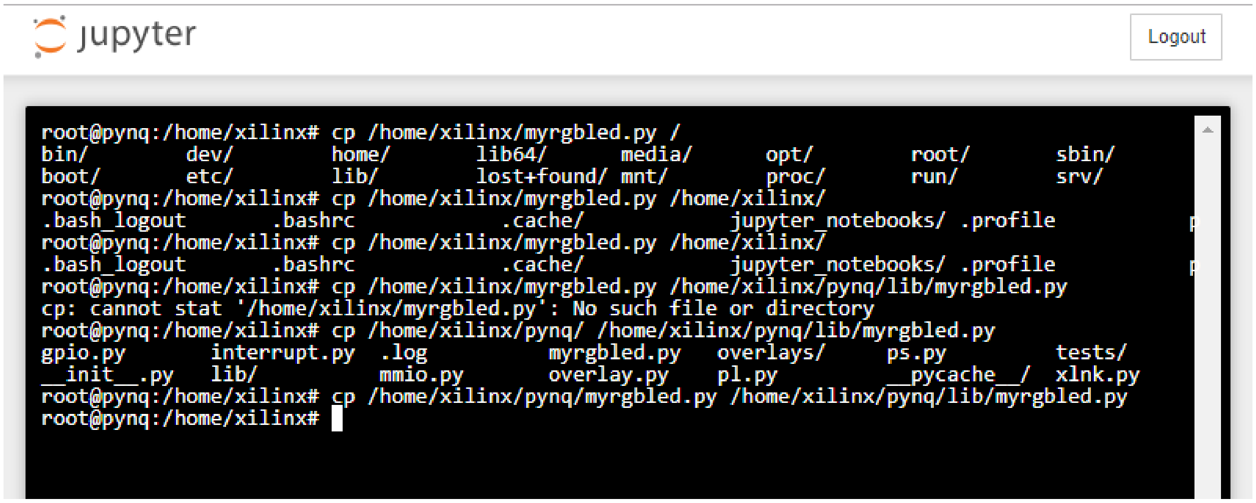
```python
self.rgbleds = ([None] * 4) + [pynq.lib.MYRGBLED(i)
                                for i in range(4, 6)]
```

Listing 2: Python code changed on line 99 of file base.py.

To save the on the computer modified driver file back on to the PYNQ platform an trick is needed due to the restricted access rights of the /lib folder. Listing 3 shows how the terminal command that is used to copy (cp) the file from folder /PYNQ to /PYNQ/lib because the file can be copyed into folder /PYNQ.

Listing 3: Jupiter Notebook terminal, copy a file.

```
root@pynq:/home/xilinx# cp /home/xilinx/pynq/myrgbled.py /home/xilinx/pynq/lib/myrgbled.py
```



```
root@pynq:/home/xilinx# cp /home/xilinx/myrgbled.py /
bin/        dev/        home/       lib64/      media/      opt/        root/       sbin/
boot/       etc/        lib/        lost+found/ mnt/        proc/       run/        srv/
root@pynq:/home/xilinx# cp /home/xilinx/myrgbled.py /home/xilinx/
.bash_logout       .bashrc            .cache/            jupyter_notebooks/ .profile           p
root@pynq:/home/xilinx# cp /home/xilinx/myrgbled.py /home/xilinx/
.bash_logout       .bashrc            .cache/            jupyter_notebooks/ .profile           p
root@pynq:/home/xilinx# cp /home/xilinx/myrgbled.py /home/xilinx/pynq/lib/myrgbled.py
cp: cannot stat '/home/xilinx/myrgbled.py': No such file or directory
root@pynq:/home/xilinx# cp /home/xilinx/pynq/ /home/xilinx/pynq/lib/myrgbled.py
gpio.py       interrupt.py  .log          myrgbled.py   overlays/     ps.py         tests/
__init__.py   lib/          mmio.py       overlay.py    pl.py         __pycache__/  xlnk.py
root@pynq:/home/xilinx# cp /home/xilinx/pynq/myrgbled.py /home/xilinx/pynq/lib/myrgbled.py
root@pynq:/home/xilinx# 
```

Figure 3: Jupiter Notebook terminal, copy a file.

Notice: The original functions of the driver remind. This provides back words compatibility for previous written programs.

To the driver a method for pulse width modulation (PWM) is added, shown in Listing 4. As inputs of the pwm() method a color can be defined which is either value 1, 2, or 4. Define a duty circle and a frequency to define the pulse length and period of the generated signal.

```
def pwm(self, color, duty_cycle, frequency):
    """PWM for single RGB LED color.

    Parameters
    ----------
    color : int 1, 2 or 3
    Color of RGB specified by a 3-bit RGB integer value.
    Blue  = 1
    Green = 2
    Red = 4
    duty_cycle : int between 0 and 100
    Duty cycle is an integer value between 0 and 100 %
    ____+———+____+———+____+ is a duty cyle of 50 %
    frequency : int
    Frequency defines the length of the intervall

    Returns
    -------
    None

    """
    if color not in [1, 2, 4]:
    raise ValueError("color should be an integer value from 1, 2, and 4.")

    try:
        self.rgb_on(color)
        time.sleep( duty_cycle / frequency )
        self.rgb_off(color)
        time.sleep( (100-duty_cycle) / frequency)
    except ZeroDivisionError:
        print "division by zero!"
```

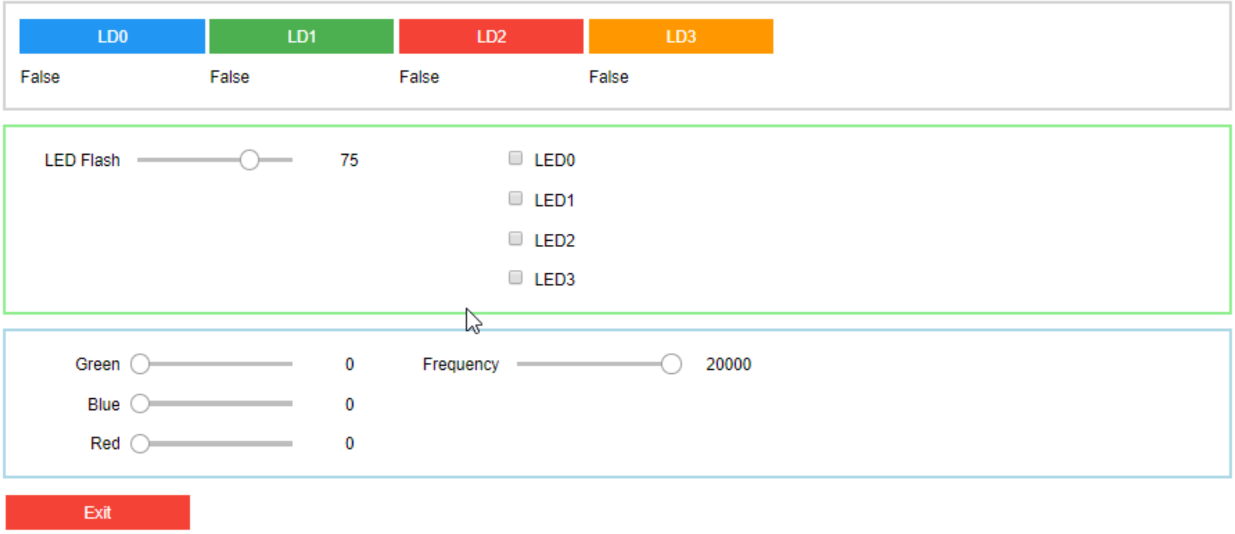
Listing 4: RGB LED driver PWM method.

The pwm() method is used in the program in an independent process so it can be run in a while loop and does not interfere with the running graphical user interface (GUI) which operates event driven, the code to build processes is shown in Listing 5.

```python
from multiprocessing import Process
from multiprocessing.sharedctypes import Value

def run_pwm2():
  # prvides PWM for RGB LED
  try:
    while( 1 ):
      if red_duty.value != 0:
      base.rgbleds[4].pwmd(red.value, red_duty.value, frequency.value)
      if green_duty.value != 0:
      base.rgbleds[4].pwmd(green.value, green_duty.value, frequency.value)
      if blue_duty.value != 0:
      base.rgbleds[4].pwmd(blue.value, blue_duty.value, frequency.value)
      # terminate process
      if exit.value:
      break
  except KeyboardInterrupt:
    raise

# running pwm in seperate process
try:
  p_pwm = Process(target=run_pwm2, args=(), name='pwm2')
  p_pwm.start()
except:
  raise

# running led flash in seperate process
try:
  p_led_flash = Process(target=run_leds, args=(), name='led_flash')
  p_led_flash.start()
except:
  raise
```

Listing 5: RGB LED driver PWM overwiev.

The designed GUI is shown in Figure 4. The first sections controls the four green LEDs and returns the status of LED0 to LED3. The light green framed section controls the green LEDs flashing rate with the slider to the right. The four check boxes are used to enable or disable the flashing of the green led. The third section controls the RGB LED LD4 and allows color mixing with the three sliders to the right, changes the PWMs duty cycle. The slider to the left controls the frequency of the PWM. At the end is an red exit button which is used to terminate the running processes properly so that the program can be restarted without restarting or interrupting the kernel.



False False False False

LED Flash 75

☐ LED0
☐ LED1
☐ LED2
☐ LED3

Green 0
Blue 0
Red 0
Frequency 20000

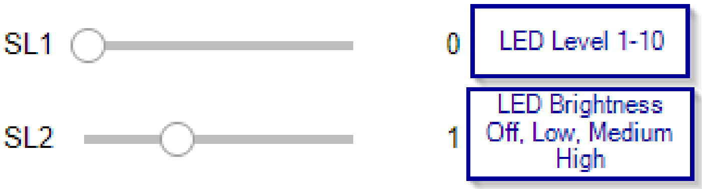
Figure 4: Jupiter Notebook GUI to control the green LEDs and a RGB LED.

It turns out that running each single color of the RGB LED is an issue if done so with three processes. The problem is that due to different duty cycles it can occur that one process invokes as example blue twice and then red only once and maybe green is not called at all. This causes the RGB color to fade in different colors then to be a stable color mixing. Therefore, the approach was changed to one process which runs the three colors sequentially. Turns out, this method works quite well. Due to the fact that two different process where used and each process has an independent Heap global variables aren't shared anymore and the class Value had to be used to synchronize those. For a larger project certainly a queue would be more appropriate to handle values between processes. An different approach would be instead of using processes to start two threads which would have the same Heap but might come with the price of decreased performance. By using a process it is important to implement a small delay from time.sleep.

The full code is shown in the appendix Section 4.1.

2.2 Part II - LED Groove Bar

The Groove LED Bar can be turned on in level increments from 0 to 10 where 0 is off and 10 all segments on. The brightness of the leds can be defined independently with an value from 0 to 3 where 0 is off, 1 is low, 2 is medium, and 3 is led brightness high. As graphical user interface (GUI) two integer slider are used SL1 and SL2 as shown in Figure 5. The source code is shown in Listing 15 in Section 4.2 of the appendix.



SL1 0

SL2 1

Figure 5: Groove LED Bar program output.

2.3 Part III - Music Synthesizer

To build a music synthesizer first the overlay and necessary python functions are loaded in the first cell.

2.3.1 MicroBlaze Softcore for PMODA

Second, a magic cell is build with the magic MicroBlaze command, shown in Listing 6. The microblaze cell works as a C wrapper for python as well as it allows to write C code and function which will be compiled, flashed, and executed on the MicroBlaze softcore.

The cell compiles, flashes, and executes C code on the MicroBlaze softcore. Each peripheral outlet, as they are PMODA, PMODB, and ARDUINO has there own MicroBlaze.

The cell uses the PMODA MicroBlaze to execute driver code which allows the programmer to invoke C functions directly in python code. This works of because of cell magic, where the MicroBlaze cell wraps the C function to make it accessible for python.

Due to the fact that there is only one MicroBlaze per outlet a notebook can only have one cell per each MicroBlaze. if there more the first code will be compiled, flashed, and executed. As the the second MicroBlaze cell with the same outlet is run the C code of that cell is compiled, flashed, and executed on the MicroBlaze.

```
%%microblaze base.PMODA
```

Listing 6: Magic microblaze command.

2.3.2 C code in MicroBlaze to Play a Melody

The MicroBlaze cell is used to run C code that builds the drivers for the connected peripherals to the PMODA. In this case the Groove connector is connected to the PMODA and is equipped with the Groove LED Bar on connector G1 and the Groove Buzzer is connected to G4.

The init functions for buzzer and LEDs had to be written or modified because usually the driver would be compiled and run as program on the platform. Therefore,general purpose input outputs (GPIO)s had to be initialized directly as shown in Listing 7. Interesting is that the led bar init has an counter intuitive GPIO assignment where someone would assume that clock (clk) would be pin A but pin A is the data pin and pin B is the clock pin.

```
void buzzer_init(){
    pb_speaker = gpio_open(PMOD_G4_A);
    gpio_set_direction(pb_speaker, GPIO_OUT);
}
void ledbar_init(){
   gpio_clk = gpio_open(PMOD_G1_B);
   gpio_data = gpio_open(PMOD_G1_A);
   gpio_set_direction(gpio_clk, GPIO_OUT);
   gpio_set_direction(gpio_data, GPIO_OUT);
}

```

Listing 7: Magic microblaze C code for buzzer and ledbar initialization .

To be able to play 10 tunes the buzzer function playNote() had to be expanded with one extra tune which is the lower B and the the maximum value of the for loop had to be increased from 8 to 10 as shown in Listing 8. In addition, the set_bits() function is used to turn on the appropriate value on the ledbar that corresponds to the note which is buzzed.

```
void playNote(char note, int duration) {

  char names[] = { 'B', 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C', 'D' };
  int tones[] = { 2010, 1916, 1700, 1519, 1432, 1275, 1136, 1014, 956, 836 };
  int i;

  // play the tone corresponding to the note name
  for (i = 0; i < 10; i++) { // haringd changed to 10
    if (names[i] == note) {
      set_bits(reverse_data(0b00000000001 << i));
      playTone(tones[i], duration);

    }
  }
}
```

Listing 8: Magic microblaze C code for playing a note.

To play a melody the function melody_demo() is changed with an individual melody the 'A-Team' theme, shown in Listing 9. The notes and beats array are changed accordingly. To turn of the ledbar the function set_bits(0) is used if condition which handles a pause where note is played. Further adjustments where made to the tempo variable which is set to seventy two. The length is defined by the size function of the notes array. To improve a stable use by all users it is recommended to check that the length of beats and notes are equal. Note, this is done in the python version of the program.

```
void melody_demo(void) {
  // The number of notes to play
  int length = 20;

  /*  A–Team theme */
  char notes[] = {' ','C','C','g','C','f','g','c','e','g','C','g','D','C','b','a','g','f','g',' ',\
    'C','C','g','C','e','f','d','g','c','e','f','a','b','b','a',' ','f','c','a',\
    ' ','d','f','g','C','g','f',' ','g','f','f','e','c','B','c',' ',\
    'e','e','d','e',' ','d',' ','e',' ','d',' ','d','a','g',
    'e','e','d','e',' ','d',' ','c',' ','c',' ','c','d', ' '};
  int beats[] = {  8,  3,  1,  2, 18,  2,  8, 10,  1,  1,  2,  2,  2, 18,  3,  1,  1,  3,
    16,  1, \
    3 , 1 , 2 , 18, 2 , 2 , 2 , 2 , 16, 3 , 1 , 2 , 50, 2 , 2 , 2 , 2 , 8 , 8, \
    8 , 3 , 1 , 2 , 18, 2 , 2 , 2 , 8 , 8 , 2 , 2,  2,  2,  16, 2, \
    2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2,  8,  8, \
    2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2,  16,  1 };

  length = sizeof(notes);
  int tempo = 73;
  int i;

  for(i = 0; i < length; i++) {
    if(notes[i] == ' ') {
      set_bits((0b00000000000));
      delay_ms(beats[i] * tempo);
    } else {
      playNote(notes[i], beats[i] * tempo);
    }
    // Delay between notes
    delay_ms(tempo / 2);
  }
}
```

Listing 9: Magic microblaze C code for playing the A-Team theme.

For simple use a main function named c_music_play() is programmed that initializes the ledbar and the buzzer GPIOs. A one second blink of five leds shows the initialization was successful. The melody function is called to play the melody. The described code is shown in Listing 10.

```
void c_music_play(){
  buzzer_init();
  ledbar_init();
  set_bits(0b1111100000);
  delay_ms(1000);
  melody_demo();
}
```

Listing 10: Magic microblaze C code for playing the A-Team theme.

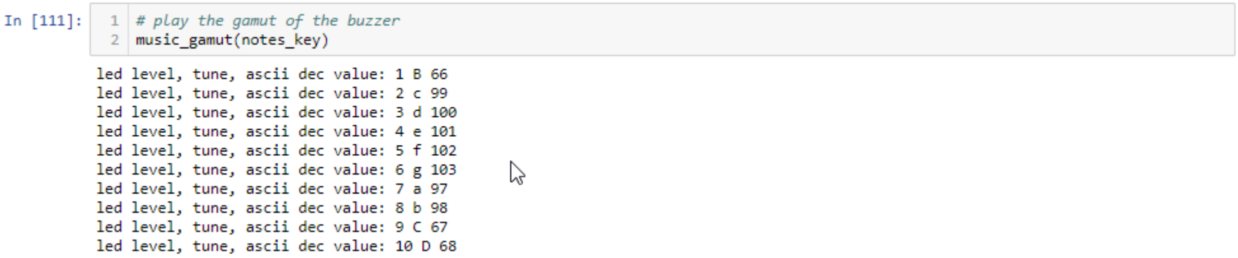
After executing the MicroBlaze cell which compiles, flashes, and executes the code onto the MicroBlaze, builds python wrappers for the C functions so they can be called in python in a cell of the Jupiter notebook, the function c_music_play() can be used in the next cell with pressing Shift+Return the melody sounds on the buzzer and the led lights up to the corresponding note.

2.3.3 Python melody

In addition to the C implementation the same function was build with python by using the basic C driver functions provided by the magic MicroBlaze cell.

Now that python is used a dictionary seems appropriate to access notes which can be made as a list as well.

To check that the LEDs are ordered and inconsistency with the led bar a gamut is programmed that also prints out the current values in console, shown in Figure 6.

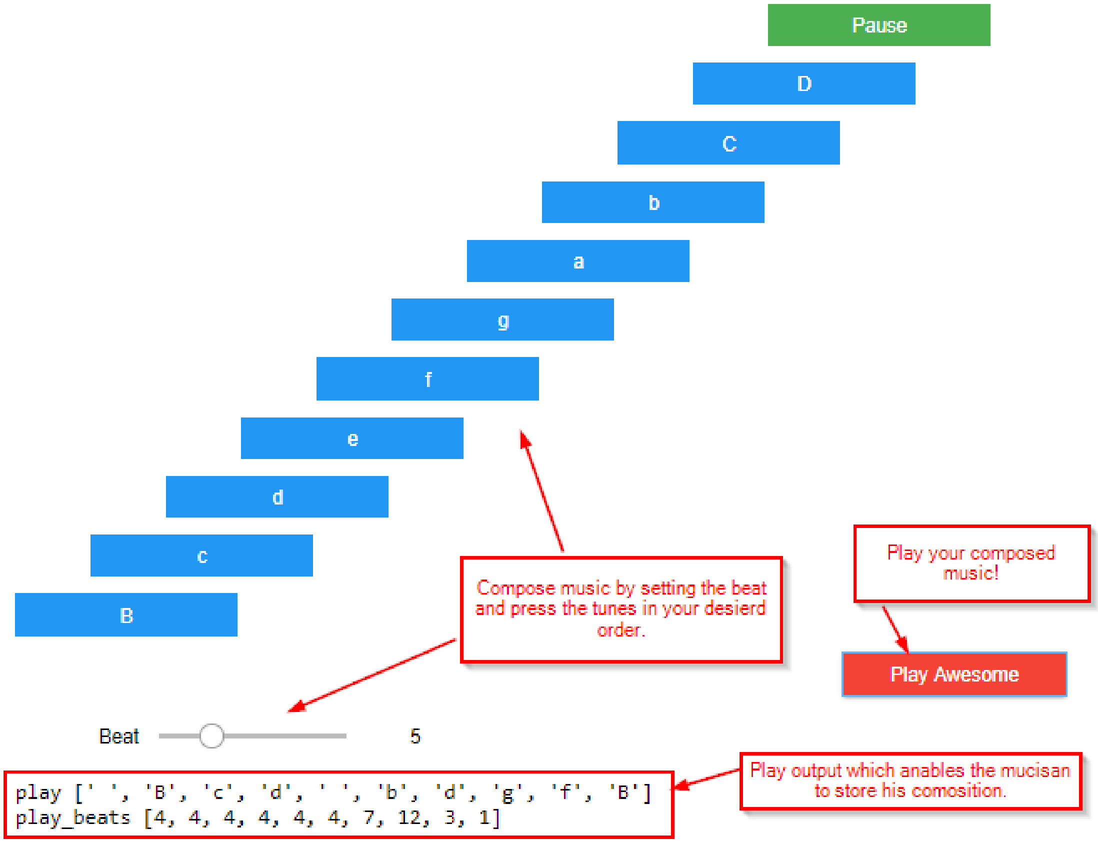


```
In [111]:
# play the gamut of the buzzer
music_gamut(notes_key)
```

```
led level, tune, ascii dec value: 1 B 66
led level, tune, ascii dec value: 2 c 99
led level, tune, ascii dec value: 3 d 100
led level, tune, ascii dec value: 4 e 101
led level, tune, ascii dec value: 5 f 102
led level, tune, ascii dec value: 6 g 103
led level, tune, ascii dec value: 7 a 97
led level, tune, ascii dec value: 8 b 98
led level, tune, ascii dec value: 9 C 67
led level, tune, ascii dec value: 10 D 68
```

Figure 6: GUI for an simple music synthesizer that allows the composer to design a melody.

Due to the simple handling of the high programming language in terms of GUIs a cooler approximation of an music synthesizer was build, shown in Figure 7. The GUi allows the user to pres any combination of notes and pause with the desired beat to build his own melody. The melody build is printed out below the GUI as the 'Play Awesome' button is pressed. This allows the user to copy paste his composition so no valuable tunes are lost. further work would be a clear button or a play awesome log book (donations welcome).



Compose music by setting the beat and press the tunes in your desierd order.

Play your composed music!

Play output which anables the mucisan to store his comosition.

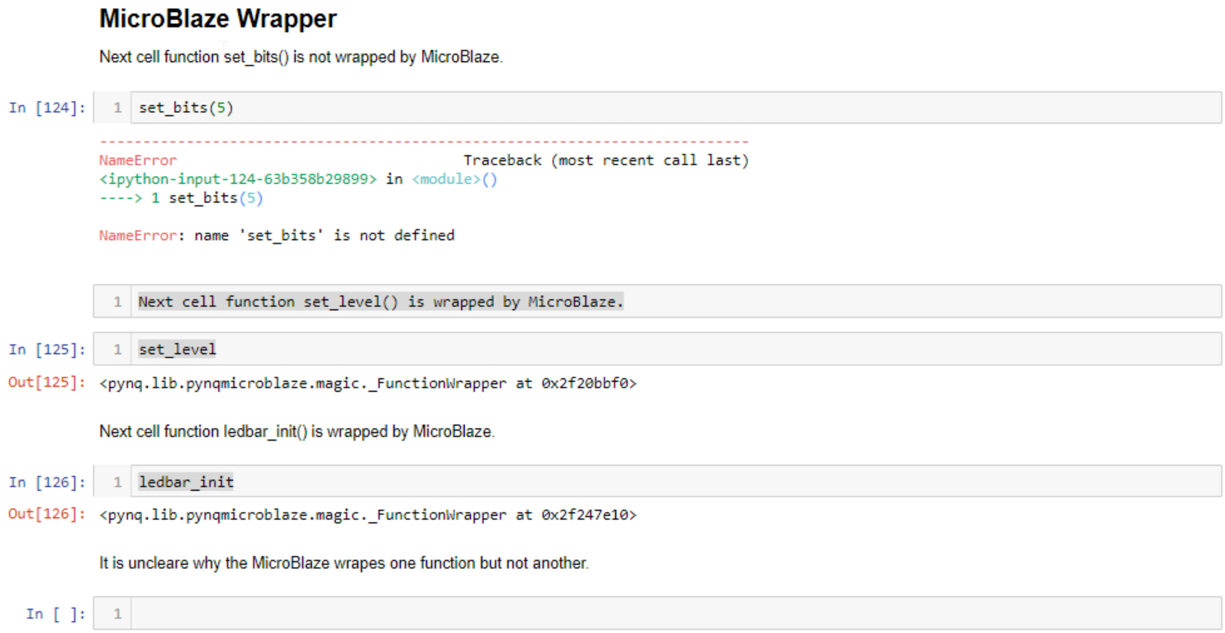
```
play [' ', 'B', 'c', 'd', ' ', 'b', 'd', 'g', 'f', 'B']
play_beats [4, 4, 4, 4, 4, 4, 7, 12, 3, 1]
```

Figure 7: GUI for an simple music synthesizer that allows the composer to design a melody.

The complete code for Part III can be found in Listing 16 or Section 4.3 of the appendix.

2.3.4 MicroBlaze wrapper behavior

By the programming the MicroBlaze showed interesting behavior in form that not every function was wrapped. Although, tremendous efforts where made to enlighten the mystery, no conclusive statement could be made why a function is wrapped or not. Figure 8 shows the an example of wrapped functions and a function that would be assumed to be wrapped but is not



MicroBlaze Wrapper

Next cell function set_bits() is not wrapped by MicroBlaze.

```
In [124]:  set_bits(5)

           ---------------------------------------------------------------------------
           NameError                                 Traceback (most recent call last)
           <ipython-input-124-63b358b29899> in <module>()
           ----> 1 set_bits(5)

           NameError: name 'set_bits' is not defined

           Next cell function set_level() is wrapped by MicroBlaze.

In [125]:  set_level
Out[125]:  <pynq.lib.pynqmicroblaze.magic._FunctionWrapper at 0x2f20bbf0>
```

Next cell function ledbar_init() is wrapped by MicroBlaze.

```
In [126]:  ledbar_init
Out[126]:  <pynq.lib.pynqmicroblaze.magic._FunctionWrapper at 0x2f247e10>
```

It is uncleare why the MicroBlaze wrapes one function but not another.

```
In [ ]:
```

Figure 8: Inconsistent wrapping of C functions.

3 Conclusion

The final project introduces the use of the groove peripherals and the vestal approaches to interface with those. First, the on board leds and RGB led is used to flash and build a color synthesizer that is controlled over a user friendly GUI. Second, python is used to interface the Groove LED bar by which two sliders allows to change the brightness and the level on what value shall be shown. Third, the MicroBlaze is used to use low level C language to build a custom driver for the Groove LED Bar and the Groove Buzzer. The C code is executed in an softcore that interfaces directly with the PMODA connector. With the use of high level python language a simple music synthesizer was build with an user friendly GUI. The C functions of the MicroBlaze can be invoked directly in python,

4 Appendix

The appendix contains code listening and other large information parts that contain partial or complete relevance to the reports topic.

4.1 Python code Listings Part I - RGB LED Driver

```
#   Copyright (c) 2016, Xilinx, Inc.
#   All rights reserved.
#
#   Redistribution and use in source and binary forms, with or without
#   modification, are permitted provided that the following conditions are met:
#
#   1.  Redistributions of source code must retain the above copyright notice,
#       this list of conditions and the following disclaimer.
#
#   2.  Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in the
#       documentation and/or other materials provided with the distribution.
#
#   3.  Neither the name of the copyright holder nor the names of its
#       contributors may be used to endorse or promote products derived from
#       this software without specific prior written permission.
#
#   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
#   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
#   THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
#   PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
#   CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
#   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
#   PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
#   OR BUSINESS INTERRUPTION). HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
#   WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
#   OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
#   ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


# from .audio import Audio
# from .video import HDMI
# from .video import Frame
# from .dma import DMA
# from .trace_buffer import Trace_Buffer
# from .usb_wifi import Usb_Wifi

from .pynqmicroblaze import PynqMicroblaze
from .pynqmicroblaze import MicroblazeRPC
from .pynqmicroblaze import MicroblazeLibrary
from .axigpio import AxiGPIO
from .dma import DMA
from .dma import LegacyDMA
from .led import LED
from .myrgbled import MYRGBLED
from .switch import Switch
from .button import Button

from .arduino import Arduino
from .arduino import Arduino_DevMode
from .arduino import Arduino_IO
from .arduino import Arduino_Analog
from .arduino import Arduino_LCD18

from .pmod import Pmod
from .pmod import Pmod_DevMode
from .pmod import Pmod_ADC
from .pmod import Pmod_DAC
from .pmod import Pmod_OLED
```

```python
from .pmod import Pmod_LED8
from .pmod import Pmod_IO
from .pmod import Pmod_IIC
from .pmod import Pmod_DPOT
from .pmod import Pmod_TC1
from .pmod import Pmod_TMP2
from .pmod import Pmod_ALS
from .pmod import Pmod_Cable
from .pmod import Pmod_Timer
from .pmod import Pmod_PWM

from .logictools import LogicToolsController
from .logictools import Waveform
from .logictools import BooleanGenerator
from .logictools import PatternGenerator
from .logictools import TraceAnalyzer
from .logictools import FSMGenerator

from . import video
from . import audio
from . import dma

__author__ = "Graham Schelle"
__copyright__ = "Copyright 2016, Xilinx"
__email__ = "pynq_support@xilinx.com"
```

Listing 11: Part I - Jupyter Notebook file __init__ saved as *.py file.

```python
#   Copyright (c) 2017, Xilinx, Inc.
#   All rights reserved.
#
#   Redistribution and use in source and binary forms, with or without
#   modification, are permitted provided that the following conditions are met:
#
#   1.  Redistributions of source code must retain the above copyright notice,
#       this list of conditions and the following disclaimer.
#
#   2.  Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in the
#       documentation and/or other materials provided with the distribution.
#
#   3.  Neither the name of the copyright holder nor the names of its
#       contributors may be used to endorse or promote products derived from
#       this software without specific prior written permission.
#
#   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
#   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
#   THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
#   PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
#   CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
#   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
#   PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
#   OR BUSINESS INTERRUPTION). HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
#   WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
#   OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
#   ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


import pynq
import pynq.lib
import pynq.lib.video
import pynq.lib.audio
from .constants import *
from pynq.lib.logictools import TraceAnalyzer


__author__ = "Peter Ogden"
__copyright__ = "Copyright 2017, Xilinx"
```

```python
__email__ = "pynq_support@xilinx.com"


class BaseOverlay(pynq.Overlay):
    """ The Base overlay for the Pynq-Z1

    This overlay is designed to interact with all of the on board peripherals
    and external interfaces of the Pynq-Z1 board. It exposes the following
    attributes:

    Attributes
    ----------
    iop_pmoda : IOP
         IO processor connected to the PMODA interface
    iop_pmodb : IOP
         IO processor connected to the PMODB interface
    iop_arduino : IOP
         IO processor connected to the Arduino/ChipKit interface
    trace_pmoda : pynq.logictools.TraceAnalyzer
        Trace analyzer block on PMODA interface, controlled by PS.
    trace_arduino : pynq.logictools.TraceAnalyzer
        Trace analyzer block on Arduino interface, controlled by PS.
    leds : AxiGPIO
         4-bit output GPIO for interacting with the green LEDs LD0-3
    buttons : AxiGPIO
         4-bit input GPIO for interacting with the buttons BTN0-3
    switches : AxiGPIO
         2-bit input GPIO for interacting with the switches SW0 and SW1
    rgbleds : [pynq.board.RGBLED]
         Wrapper for GPIO for LD4 and LD5 multicolour LEDs
    video : pynq.lib.video.HDMIWrapper
         HDMI input and output interfaces
    audio : pynq.lib.audio.Audio
         Headphone jack and on-board microphone

    """

    def __init__(self, bitfile, **kwargs):
        super().__init__(bitfile, **kwargs)
        if self.is_loaded():
            self.iop_pmoda.mbtype = "Pmod"
            self.iop_pmodb.mbtype = "Pmod"
            self.iop_arduino.mbtype = "Arduino"

            self.PMODA = self.iop_pmoda.mb_info
            self.PMODB = self.iop_pmodb.mb_info
            self.ARDUINO = self.iop_arduino.mb_info

            self.audio = self.audio_direct_0
            self.leds = self.leds_gpio.channel1
            self.switches = self.switches_gpio.channel1
            self.buttons = self.btns_gpio.channel1
            self.leds.setlength(4)
            self.switches.setlength(2)
            self.buttons.setlength(4)
            self.leds.setdirection("out")
            self.switches.setdirection("in")
            self.buttons.setdirection("in")
            self.rgbleds = ([None] * 4) + [pynq.lib.MYRGBLED(i)
                                           for i in range(4, 6)]

            self.trace_pmoda = TraceAnalyzer(
                self.trace_analyzer_pmoda.description['ip'],
                PYNQZ1_PMODA_SPECIFICATION)
            self.trace_arduino = TraceAnalyzer(
                self.trace_analyzer_arduino.description['ip'],
                PYNQZ1_ARDUINO_SPECIFICATION)
```

Listing 12: Part I - Jupyter Notebook file base saved as *.py file.

```
#   Copyright (c) 2016, Xilinx, Inc.
#   All rights reserved.
#
#   Redistribution and use in source and binary forms, with or without
#   modification, are permitted provided that the following conditions are met:
#
#   1.  Redistributions of source code must retain the above copyright notice,
#       this list of conditions and the following disclaimer.
#
#   2.  Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in the
#       documentation and/or other materials provided with the distribution.
#
#   3.  Neither the name of the copyright holder nor the names of its
#       contributors may be used to endorse or promote products derived from
#       this software without specific prior written permission.
#
#   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
#   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
#   THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
#   PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
#   CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
#   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
#   PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
#   OR BUSINESS INTERRUPTION). HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
#   WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
#   OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
#   ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
#   edited by Dimitri Haring
#   date 12/04/2018

from pynq import MMIO
from pynq import PL
import time

__author__ = "Graham Schelle"
__copyright__ = "Copyright 2016, Xilinx"
__email__ = "pynq_support@xilinx.com"


RGBLEDS_XGPIO_OFFSET = 0
RGBLEDS_START_INDEX = 4
RGB_CLEAR = 0
RGB_BLUE = 1
RGB_GREEN = 2
RGB_CYAN = 3
RGB_RED = 4
RGB_MAGENTA = 5
RGB_YELLOW = 6
RGB_WHITE = 7


class MYRGBLED(object):
    """This class controls the onboard RGB LEDs.

    Attributes
    ----------
    index : int
        The index of the RGB LED, from 4 (LD4) to 5 (LD5).
    _mmio : MMIO
        Shared memory map for the RGBLED GPIO controller.
    _rgbleds_val : int
        Global value of the RGBLED GPIO pins.
    """
    _mmio = None
    _rgbleds_val = 0

    def __init__(self, index):
        """Create a new RGB LED object.

        Parameters
        ----------
        index : int
            Index of the RGBLED, from 4 (LD4) to 5 (LD5).

        """
        # print("Changed LED Driver to MYRGBLED.") # debugging only
        if index not in [4, 5]:
            raise ValueError("Index for onboard RGBLEDs should be 4 or 5.")

        self.index = index
        if MYRGBLED._mmio is None:
            base_addr = PL.ip_dict["rgbleds_gpio"]["phys_addr"]
            MYRGBLED._mmio = MMIO(base_addr, 16)

    def on(self, color):
        """Turn on a single RGB LED with a color value (see color constants).

        Parameters
        ----------
        color : int
           Color of RGB specified by a 3-bit RGB integer value.

        Returns
        -------
        None

        """
        if color not in range(8):
            raise ValueError("color should be an integer value from 0 to 7.")

        rgb_mask = 0x7 << ((self.index-RGBLEDS_START_INDEX)*3)
        new_val = (MYRGBLED._rgbleds_val & ~rgb_mask) | \
                  (color << ((self.index-RGBLEDS_START_INDEX)*3))
        self._set_rgbleds_value(new_val)

    def off(self):
        """Turn off a single RGBLED.

        Returns
        -------
        None

        """
        rgb_mask = 0x7 << ((self.index-RGBLEDS_START_INDEX)*3)
        new_val = MYRGBLED._rgbleds_val & ~rgb_mask
        self._set_rgbleds_value(new_val)

    def red_on(self):
        """Turn on a single RGB LED with color value red.

        Parameters
        ----------
        color : int
           Color of RGB specified by a 3-bit RGB integer value.

        Returns
        -------
        None

        """
#         if color not in range(8):
#             raise ValueError("color should be an integer value from 0 to 7.")
        new_val = (MYRGBLED._rgbleds_val ) | (RGB_RED << ((self.index-RGBLEDS_START_INDEX)*3))
#         print(MYRGBLED._rgbleds_val)
#         print(new_val)
#         print( (RGB_RED << ((self.index-RGBLEDS_START_INDEX)*3)) )
        self._set_rgbleds_value(new_val)

    def red_off(self):
        """Turn off a single RGB LED with color value red.

        Parameters
        ----------
        color : int
           Color of RGB specified by a 3-bit RGB integer value.

        Returns
        -------
        None

        """
#         if color not in range(8):
#             raise ValueError("color should be an integer value from 0 to 7.")

        new_val = (MYRGBLED._rgbleds_val ) &~ (RGB_RED << ((self.index-RGBLEDS_START_INDEX)*3))
        self._set_rgbleds_value(new_val)

    def green_on(self):
        """Turn on a single RGB LED with color value green.

        Parameters
        ----------
        color : int
           Color of RGB specified by a 3-bit RGB integer value.

        Returns
        -------
        None

        """
#         if color not in range(8):
#             raise ValueError("color should be an integer value from 0 to 7.")

        new_val = (MYRGBLED._rgbleds_val ) | (RGB_GREEN << ((self.index-RGBLEDS_START_INDEX)*3))
        self._set_rgbleds_value(new_val)

    def green_off(self):
        """Turn off a single RGB LED with color value green.

        Parameters
        ----------
        color : int
           Color of RGB specified by a 3-bit RGB integer value.

        Returns
        -------
        None

        """
#         if color not in range(8):
#             raise ValueError("color should be an integer value from 0 to 7.")

        new_val = (MYRGBLED._rgbleds_val ) &~ (RGB_GREEN << ((self.index-RGBLEDS_START_INDEX)*3))
        self._set_rgbleds_value(new_val)

    def blue_on(self):
        """Turn on a single RGB LED with color value blue.

        Parameters
        ----------
        color : int
           Color of RGB specified by a 3-bit RGB integer value.

        Returns
        -------
        None

        """
#        if color not in range(8):
#            raise ValueError("color should be an integer value from 0 to 7.")

        new_val = (MYRGBLED._rgbleds_val ) | (RGB_BLUE << ((self.index-RGBLEDS_START_INDEX)*3))
        self._set_rgbleds_value(new_val)

    def blue_off(self):
        """Turn off a single RGB LED with color value blue.

        Parameters
        ----------
        color : int
           Color of RGB specified by a 3-bit RGB integer value.

        Returns
        -------
        None

        """
#        if color not in range(8):
#            raise ValueError("color should be an integer value from 0 to 7.")

        new_val = (MYRGBLED._rgbleds_val ) &~ (RGB_BLUE << ((self.index-RGBLEDS_START_INDEX)*3))
        self._set_rgbleds_value(new_val)

    def status(self):
        rgb_mask = 0x7 << ((self.index-RGBLEDS_START_INDEX)*3)
        return ((MYRGBLED._rgbleds_val )& ~rgb_mask)

    def rgb_on(self, color):
        """Turn on a single RGB LED color.

        Parameters
        ----------
        color : int
           Color of RGB specified by a 3-bit RGB integer value.
               Blue  = 1
               Green = 2
               Red = 4

        Returns
        -------
        None

        """
        if color not in [1, 2, 4]:
            raise ValueError("color should be an integer value from 1, 2, and 4.")

        new_val = (MYRGBLED._rgbleds_val ) | (color << ((self.index-RGBLEDS_START_INDEX)*3))
        self._set_rgbleds_value(new_val)

    def rgb_off(self, color):
        """Turn off a single RGB LED color.

        Parameters
        ----------
        color : int
           Color of RGB specified by a 3-bit RGB integer value.
               Blue  = 1
               Green = 2
               Red = 4
        Returns
        -------
        None

        """
        if color not in [1, 2, 4]:
            raise ValueError("color should be an integer value from 1, 2, and 4.")

        new_val = (MYRGBLED._rgbleds_val ) &~ (color << ((self.index-RGBLEDS_START_INDEX)*3))
        self._set_rgbleds_value(new_val)

    def pwm(self, color, duty_cycle, frequency):
    """PWM for single RGB LED color.

    Parameters
    ----------
    color : int 1, 2 or 3
      Color of RGB specified by a 3-bit RGB integer value.
        Blue  = 1
        Green = 2
        Red = 4
    duty_cycle : int between 0 and 100
      Duty cycle is an integer value between 0 and 100 %
      ____+----+____+----+____+ is a duty cyle of 50 %
    frequency : int
      Frequency defines the length of the intervall

        Returns
        -------
        None

        """
        if color not in [1, 2, 4]:
            raise ValueError("color should be an integer value from 1, 2, and 4.")

        self.rgb_on(color)
        time.sleep( duty_cycle / frequency )
        self.rgb_off(color)
        time.sleep( (100-duty_cycle) / frequency)

    def write(self, color):
        """Set the RGBLED state according to the input value.

        Parameters
        ----------
        color : int
            Color of RGB specified by a 3-bit RGB integer value.

        Returns
        -------
        None

        """
        self.on(color)

    def read(self):
        """Retrieve the RGBLED state.

        Returns
        -------
        int
            The color value stored in the RGBLED.

        """
        return (MYRGBLED._rgbleds_val >>
                ((self.index-RGBLEDS_START_INDEX)*3)) & 0x7

    @staticmethod
    def _set_rgbleds_value(value):
        """Set the state of all RGBLEDs.

        Note
        ----
        This function should not be used directly. User should call
        `on()`, `off()`, instead.

        Parameters
        ----------
        value : int
            The value of all the RGBLEDs encoded in a single variable.

        """
        MYRGBLED._rgbleds_val = value
        MYRGBLED._mmio.write(RGBLEDS_XGPIO_OFFSET, value)
```

Listing 13: Part I - Jupyter Notebook file myrgbled saved as *.py file.

```

# coding: utf-8

# ## LED Ctrl RGB
#
# Use buttons and sliders to control the LEDs on the board.
#
# The program is started by select Menubar -> Cell -> Run All
#
# Cell -> Current Outputs -> Toggle Scrolling
#

# In[1]:


import time
from pynq.overlays.base import BaseOverlay

import ipywidgets as widgets
from IPython.display import display
from multiprocessing import Process
from multiprocessing.sharedctypes import Value

base = BaseOverlay("base.bit")


# ### Define functions here
# Function decision() provides the computaion of the win and loss with average and a consol ouput accordingly.
# #### Colors RGB LED No 4 and 5
#       off = 0     blue = 1     green = 2     tÃ¼rkies = 3     red = 4     purple = 5     yellow = 6
#       white = 7
#

# In[19]:


```

```
def all_led_off():
    # turn all led's off
    for led in base.leds:
        led.off()
    base.rgbleds[4].off()
    base.rgbleds[5].off()

def on_button0_clicked(b):
    if bt_led_state0.value == 0:
        bt_led_state0.value = 1
        base.leds[0].on()
    else:
        bt_led_state0.value = 0
        base.leds[0].off()
    ldStatus0.value = '' + ('False' if base.leds.read() & int('0001',2) == 0 else 'True')

def on_button1_clicked(b):
    if bt_led_state1.value == 0:
        base.leds[1].on()
        bt_led_state1.value = 1
    else:
        bt_led_state1.value = 0
        base.leds[1].off()
    ldStatus1.value = '' + ('False' if base.leds.read() & int('0010',2) == 0 else 'True')

def on_button2_clicked(b):
    if bt_led_state2.value == 0:
        base.leds[2].on()
        bt_led_state2.value = 1
    else:
        bt_led_state2.value = 0
        base.leds[2].off()
    ldStatus2.value = '' + ('False' if base.leds.read() & int('0100',2) == 0 else 'True')

def on_button3_clicked(b):
    if bt_led_state3.value == 0:
        base.leds[3].on()
        bt_led_state3.value = 1
    else:
        bt_led_state3.value = 0
        base.leds[3].off()
    ldStatus3.value = '' + ('False' if base.leds.read() & int('1000',2) == 0 else 'True')

def on_button4_clicked(b):
    exit.value = 1

def handle_slider0_change(change):
    green_duty.value = change.new

def handle_slider1_change(change):
    blue_duty.value = change.new

def handle_slider2_change(change):
    red_duty.value = change.new

def handle_slider3_change(change):
    frequency.value = change.new

def handle_slider4_change(change):
    led_freq.value = change.new

def handle_check0_change(LED0):
    led0_check.value = int(LED0)

def handle_check1_change(LED1):
    led1_check.value = int(LED1)

def handle_check2_change(LED2):
```

```python
    led2_check.value = int(LED2)

def handle_check3_change(LED3):
    led3_check.value = int(LED3)

def led_control(which_led, bt_status, check_status):
    if check_status and bt_status != 0:
        base.leds[which_led].toggle()
    else:
        if bt_status:
            base.leds[which_led].on()
        else:
            base.leds[which_led].off()

def run_leds():
    # function to run LED output with flash function in process
    while( 1 ):
        # LED control
        led_control(0, bt_led_state0.value, led0_check.value)
        led_control(1, bt_led_state1.value, led1_check.value)
        led_control(2, bt_led_state2.value, led2_check.value)
        led_control(3, bt_led_state3.value, led3_check.value)

        # update LED status
        ldStatus0.value = '' + ('False' if base.leds.read() & int('0001',2) == 0 else 'True')
        ldStatus1.value = '' + ('False' if base.leds.read() & int('0010',2) == 0 else 'True')
        ldStatus2.value = '' + ('False' if base.leds.read() & int('0100',2) == 0 else 'True')
        ldStatus3.value = '' + ('False' if base.leds.read() & int('1000',2) == 0 else 'True')

        # defines interval time
        time.sleep(led_freq.value/100)

        # terminate process
        if exit.value:
            break

def run_pwm2():
    # prvides PWM for RGB LED
    try:
        while( 1 ):
            if red_duty.value != 0:
                base.rgbleds[4].pwmd(red.value, red_duty.value, frequency.value)
            if green_duty.value != 0:
                base.rgbleds[4].pwmd(green.value, green_duty.value, frequency.value)
            if blue_duty.value != 0:
                base.rgbleds[4].pwmd(blue.value, blue_duty.value, frequency.value)
            # terminate process
            if exit.value:
                break
    except KeyboardInterrupt:
        raise

def run_gui():
    # setup GUI and displays it

    button0.on_click(on_button0_clicked)
    button1.on_click(on_button1_clicked)
    button2.on_click(on_button2_clicked)
    button3.on_click(on_button3_clicked)
    button4.on_click(on_button4_clicked)

    slider0.observe(handle_slider0_change, names='value')
    slider1.observe(handle_slider1_change, names='value')
    slider2.observe(handle_slider2_change, names='value')
    slider3.observe(handle_slider3_change, names='value')
    slider4.observe(handle_slider4_change, names='value')

    check0.observe(handle_check0_change)
    check1.observe(handle_check1_change)
    check2.observe(handle_check2_change)
    check3.observe(handle_check3_change)

    # display LED toggle controls
    left_box = widgets.VBox([button0, ldStatus0])
    right_box = widgets.VBox([button1, ldStatus1])
    left1_box = widgets.VBox([button2, ldStatus2])
    right1_box = widgets.VBox([button3, ldStatus3])
    box = widgets.HBox([left_box, right_box, left1_box, right1_box])
    box.layout.border='solid 2px lightgray'
    box.layout.padding='10px 10px 10px 10px'
    display(box)

    # display LED flash controls
    left3_box = widgets.VBox([slider4])
    right3_box = widgets.VBox([check0, check1, check2, check3])
    box3 = widgets.HBox([left3_box, right3_box])
    box3.layout.border='solid 2px lightgreen'
    box3.layout.padding='10px 10px 10px 10px'
    display(box3)

    # display RGB controls
    left2_box = widgets.VBox([slider0, slider1, slider2])
    right2_box = widgets.VBox([slider3])
    box1 = widgets.HBox([left2_box, right2_box])
    box1.layout.border='solid 2px lightblue'
    box1.layout.padding='10px 10px 10px 10px'
    display(box1)

    # Exit Button
    display(button4)



# ### Start progarm

# In[20]:


if __name__ == '__main__':
    # Gui variables
    button0 = widgets.Button(description="LD0", button_style='primary')
    button1 = widgets.Button(description="LD1", button_style='success')
    button2 = widgets.Button(description="LD2", button_style='danger')
    button3 = widgets.Button(description="LD3", button_style='warning')
    button4 = widgets.Button(description="Exit", button_style='danger')

    ldStatus0 = widgets.Label(value='False')
    ldStatus1 = widgets.Label(value='False')
    ldStatus2 = widgets.Label(value='False')
    ldStatus3 = widgets.Label(value='False')

    check0 = widgets.interactive(handle_check0_change, LED0=False)
    check1 = widgets.interactive(handle_check1_change, LED1=False)
    check2 = widgets.interactive(handle_check2_change, LED2=False)
    check3 = widgets.interactive(handle_check3_change, LED3=False)

    slider0 = widgets.IntSlider(min=0, max=100, value=0, description='Green')
    slider1 = widgets.IntSlider(min=0, max=100, value=0, description='Blue')
    slider2 = widgets.IntSlider(min=0, max=100, value=0, description='Red')
    slider3 = widgets.IntSlider(min=30, max=20000, value=20000, description='Frequency')
    slider4 = widgets.IntSlider(min=10, max=100, value=75, description='LED Flash')

    # LED variables
    led_freq = Value('i', 75)
    led0_check = Value('i', 0)
    led1_check = Value('i', 0)
    led2_check = Value('i', 0)
    led3_check = Value('i', 0)
    bt_led_state0 = Value('i', 0)
    bt_led_state1 = Value('i', 0)
    bt_led_state2 = Value('i', 0)
    bt_led_state3 = Value('i', 0)

    # RGB gloabal variables
    blue = Value('i', 1)
    green = Value('i', 2)
    red = Value('i', 4)
    blue_duty = Value('i', 0)
    green_duty = Value('i', 0)
    red_duty = Value('i', 0)
    frequency = Value('i', 20000)

    # terminate process
    exit = Value('i', 0)

    # turn all led's off
    all_led_off()

    # LED show of
    for x in range(3):
        base.leds[x].on()
        base.leds[x+1].on()
        base.rgbleds[4].rgb_on(2**x);
        base.rgbleds[5].rgb_on(2**x);
        time.sleep(1)
        all_led_off()

    # start GUI
    run_gui()

    # running pwm in seperate process
    try:
        p_pwm = Process(target=run_pwm2, args=(), name='pwm2')
        p_pwm.start()
    except:
            raise

    # running led flash in seperate process
    try:
        p_led_flash = Process(target=run_leds, args=(), name='led_flash')
        p_led_flash.start()
    except:
            raise

    #print('Am I blocked?') # debug only
```

Listing 14: Part I - Jupyter Notebook file LED_ctrl_myrgbled saved as *.py file.

4.2 Python code Listings Part II - LED Groove Bar

```

# coding: utf-8

# # Part II - LED Groove Bar
# Demonstrates how the LED Groove Bar level is set with slider SL1. The brightness can be
    chosen in four levels with slider SL2.
#
# LED Bar Brightness
#  - 0 = off
#  - 1 = low
#   - 2 = medium
#   - 3 = hight

# In[2]:


# Steup the PYNQ board
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")

from pynq.lib.pmod import Grove_LEDbar
from pynq.lib.pmod import PMOD_GROVE_G1 # Import constants
import ipywidgets as widgets
from IPython.display import display

# For delays
from time import sleep

# Global values
g_ledBrightness  = 3
g_leds = 0

# defined functions
def handle_slider1_change(change):
    global g_leds
    ledbar.write_level(change.new, g_ledBrightness, 1)
    g_leds = change.new
def handle_slider2_change(change):
    global g_ledBrightness
    g_ledBrightness = change.new
    ledbar.write_level(g_leds, change.new, 1)
   # ledbar.write_brightness(ledbar.read(), change.new)


# Instantiate Grove LED Bar on PMODA and on Pmod2Grove G1
ledbar = Grove_LEDbar(base.PMODA, PMOD_GROVE_G1)
ledbar.reset()

# Flash 2 extreme LEDs of the LED Bar in a loop, dubbiging only
# for i in range(5):
#     ledbar.write_binary(0b1000000001)
#     sleep(0.5)
#     ledbar.write_binary(0b0000000000)
#     sleep(0.5)

# GUI
slider1 = widgets.IntSlider(min=0, max=10, value=0, description='SL1')
slider2 = widgets.IntSlider(min=0, max=3, value=0, description='SL2')

slider1.observe(handle_slider1_change, names='value')
slider2.observe(handle_slider2_change, names='value')

display(slider1, slider2)
```

Listing 15: Part II - LED Groove Bar Python code.

4.3 Python code Listings Part III - Music Synthesizer

```

## Music Sythesizer

The music synthesizer plays the A-Team theme with the Groove Buzzer and visualizes the played note on the Groove LED Bar. Tu generate the tone or turn on a LED Bar Level the microblaze is used to run C code on it which is used to run the drivers for both external components.

To run the program scroll down to the music_gamut cell no. 8 and execute in menulist Cell -> Run All Above
```

~~~

To just here and see the playable tones execute cell no. 8 with the music_gamut() function.


```python
# Steup the PYNQ board
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")
import time
import ipywidgets as widgets
from IPython.display import display
```

MicroBlaze Softcore for PMODA
The following cell compiles, flashes, and executes C code on the MicroBlaze softcore. Each perihareal outlet as they are PMODA, PMODB, and ARDUINO has there own MicroBlaze.

The following cell uses the PMODA MicroBlaze to execute driver code which allows the pogrammer to invoke C functions directly in python code. This works of because of cell magic where the microblaze cell wrappes the C funtion to make it accessable for python.

Due to the fact that there is only one microblace per outlet a notebook can only have one cell per each microblaze. if there more the first code will be compiled, flashed, and executed. As the the second microplace cell with the same outlet is run the C code of that cell is compiled, flashed, and executed on the microblaze.


```python
%%microblaze base.PMODA

/*
* Code imported from pmod_groove_ledbar.c file
*/
#include "xparameters.h"
#include "timer.h"
#include "circular_buffer.h"
#include "gpio.h"
#include "pmod_grove.h" // file added to have correct

/*
 * Green-to-Red direction contains slight transparency to one led distance.
 * i.e. A LED that is OFF will glow slightly if a LED  beside it is ON
 */
#define GLB_CMDMODE                  0x00
#define HIGH                         0xFF
#define LOW                          0x01
#define MED                          0xAA
#define OFF                          0x00

/*
 * gpio devices ledbar for clock and data
 */
gpio gpio_clk;
gpio gpio_data;

/*
 * LED state, Brightness for each LED in
 * {Red, Orange, Green, Green, Green, Green, Green, Green, Green, Green}
 */
char ledbar_state[10] = {OFF, OFF, OFF, OFF, OFF, OFF, OFF, OFF, OFF, OFF};
char current_state[10] = {OFF, OFF, OFF, OFF, OFF, OFF, OFF, OFF, OFF, OFF};

// Current Level
int level_holder = 0;

// Current direction: 0 => Red-to-Green, 1 => Green-to-Red
int prev_inverse = 0;

~~~

```

// The driver instance for GPIO Devices
gpio pb_speaker;

void buzzer_init(){
    pb_speaker = gpio_open(PMOD_G4_A);
    gpio_set_direction(pb_speaker, GPIO_OUT);
}

void generateTone(int period_us) {
    // turn-ON speaker
    gpio_write(pb_speaker, 1);
    delay_us(period_us>>1);
    // turn-OFF speaker
    gpio_write(pb_speaker, 0);
    delay_us(period_us>>1);
}

void playTone(int tone, int duration) {
    // tone is in us delay
    long i;
    for (i = 0; i < duration * 1000L; i += tone * 2) {
        generateTone(tone*2);
    }
}

void ledbar_init(){
    gpio_clk = gpio_open(PMOD_G1_B);
    gpio_data = gpio_open(PMOD_G1_A);
    gpio_set_direction(gpio_clk, GPIO_OUT);
    gpio_set_direction(gpio_data, GPIO_OUT);
}

void send_data(u8 data){
    int i;
    u32 data_state, clkval, data_internal;

    data_internal = data;

    clkval = 0;
    gpio_write(gpio_data, 0);
    // First toggle the clock 8 times
    for (i = 0; i < 8; ++i) {
         clkval ^= 1;
         gpio_write(gpio_clk, clkval);
    }

    // Working in 8-bit mode
    for (i = 0; i < 8; i++){
        /*
         * Read each bit of the data to be sent LSB first
         * Write it to the data_pin
         */
        data_state = (data_internal & 0x80) ? 0x00000001 : 0x00000000;
        gpio_write(gpio_data, data_state);
        clkval ^= 1;
        gpio_write(gpio_clk, clkval);

        // Shift Incoming data to fetch next bit
        data_internal = data_internal << 1;
    }
}

void latch_data(){
    int i;
    gpio_write(gpio_data, 0);
```

```c
    delay_ms(10);

    // Generate four pulses on the data pin as per data sheet
    for (i = 0; i < 4; i++){
        gpio_write(gpio_data, 1);
        gpio_write(gpio_data, 0);
    }
}

void set_bits(u16 data){
    int h,i;
    int data_internal = data;

    for(h=0; h<10; h++){
        ledbar_state[h] = HIGH;
    }

    send_data(GLB_CMDMODE);

    for (i = 0; i < 10; i++){
        if ((data_internal & 0x0001) == 1) {
            send_data(ledbar_state[i]);
        } else {
            send_data(0x00);
            ledbar_state[i] = 0x00;
        }
        data_internal = data_internal >> 1;
    }
    // Two extra empty bits for padding the command to the correct length
    send_data(0x00);
    send_data(0x00);


    latch_data();
    // Store LEBbar state for reading purpose.
    for(h=0; h<10; h++){
        current_state[h] = ledbar_state[h];
    }
}

void set_led_brightness(u16 data, char set_brightness[]){
    int h,i;
    int data_internal = data;

    for(h=0; h<10; h++){
        ledbar_state[h] = set_brightness[h];
    }

    send_data(GLB_CMDMODE);

    for (i = 0; i < 10; i++){
        if ((data_internal & 0x0001) == 1) {
            send_data(ledbar_state[i]);
        } else {
            send_data(0x00);
            ledbar_state[i] = 0x00;
        }
        data_internal = data_internal >> 1;
    }
    // Two extra empty bits for padding the command to the correct length
    send_data(0x00);
    send_data(0x00);

    latch_data();
    // Store LEBbar state for reading purpose.
    for(h=0; h<10; h++){
        current_state[h] = ledbar_state[h];
    }
```

```
}

void set_level(int level, int intensity, int inverse){
    int h,i;
    int prev_inv ;

    prev_inv = prev_inverse;

    // Clear LED states from previous writes
    if (inverse != prev_inv) {
        for(h=0; h<10; h++){
            ledbar_state[h] = OFF;
        }
    }

    if (inverse == 0) {
        // Execute when direction is Red-to-Green
        if (level < level_holder) {
            for(h=level_holder-1; h>level-1; h--){
                ledbar_state[h] = OFF;
            }
        }
        for(h=0; h<level; h++)
        {
            if (intensity == 1) {
                ledbar_state[h] = LOW;
            } else if (intensity == 2) {
                ledbar_state[h] = MED;
            } else if (intensity == 3) {
                ledbar_state[h] = HIGH;
            } else {
                ledbar_state[h] = OFF;
            }
        }
        for(h=level; h>10; h++){
            ledbar_state[h] = OFF;
        }
    } else if(inverse == 1) { // Execute when direction is Red-to-Green
        if (level < level_holder) {
            for(h=0; h>=10-level; h++)
            {
                ledbar_state[h] = OFF;
            }
        }
        for(h=9; h>=10-level; h--)
        {
            if (intensity == 1) {
                ledbar_state[h] = LOW;
            } else if (intensity == 2) {
                ledbar_state[h] = MED;
            } else if (intensity == 3) {
                ledbar_state[h] = HIGH;
            } else {
                ledbar_state[h] = OFF;
            }
        }
        if (level != 10) {
            for(h=10-level-1; h>=0; h--)
            {
                ledbar_state[h] = OFF;
            }
        }
    } else { // Execute when direction is Invalid Integer
        for(h=0; h<10; h++){
            ledbar_state[h] = OFF;
        }
    }

    send_data(GLB_CMDMODE);

    for (i = 0; i < 10; i++){
        send_data(ledbar_state[i]);
    }
    // Two extra empty bits for padding the command to the correct length
    send_data(0x00);
    send_data(0x00);

    // Two extra empty bits for padding the command to the correct length
    latch_data();
    // Store LEBbar Indication level for resetting level
    level_holder= level;
    // Store LEBbar direction for resetting direction
    prev_inverse = inverse;
    // Store LEBbar state for reading purpose.
    for(h=0; h<10; h++){
        current_state[h] = ledbar_state[h];
    }
}

u16 reverse_data(u16 c){
    /*
     * Function to reverse incoming data
     * Allows LEDbar to be lit in reverse order
     */
    int shift;
    u16 result = 0;

    for (shift = 0; shift < 16; shift++){
        if (c & (0x0001 << shift))
            result |= (0x8000 >> shift);
    }

    // 10 LSBs are used as LED Control 6 MSBs are ignored
    result = result >> 6;
    return result;
}

void playNote(char note, int duration) {

    char names[] = { 'B', 'c',  'd',  'e',  'f',  'g',  'a',  'b', 'C',  'D'  };
    int tones[] = {  2010, 1916, 1700, 1519, 1432, 1275, 1136, 1014, 956, 836 };
    int i;

    // play the tone corresponding to the note name
    for (i = 0; i < 10; i++) { // haringd changed to 9
        if (names[i] == note) {
          set_bits(reverse_data(0b00000000001 << i));
          playTone(tones[i], duration);

        }
    }
}

void melody_demo(void) {
    // The number of notes to play
    int length = 20;

    /* Melody demo */
//  char notes[] = "ccggaagffeeddc ";
//  int beats[] = { 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 4 };

    /*  A-Team theme */
    char notes[] = {' ','C','C','g','C','f','g','c','e','g','C','g','D','C','b','a','g','f','g', ' ',\
                    'C','C','g','C','e','f','d','g','c','e','f','a','b','b','a',' ','f','c','a',\
```

~~~
                    ' ','d','f','g','C','g','f',' ','g','f','f','e','c','B','c',' ',\
                    'e','e','d','e',' ','d',' ','e',' ','d',' ','d','a','g',
                    'e','e','d','e',' ','d',' ','c',' ','c',' ','c','d', ' '};
    int beats[] = {  8,  3,  1,  2, 18,  2,  8, 10,  1,  1,  2,  2,  2, 18,  3,  1,  1,  3, 16,  1, \
                     3 , 1 , 2 , 18, 2 , 2 , 2 , 2 , 16, 3 , 1 , 2 , 50, 2 , 2 , 2 , 2 , 8 , 8, \
                     8 , 3 , 1 , 2 , 18, 2 , 2 , 2 , 8 , 8 , 2 , 2,  2,  2,  16, 2, \
                     2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2,  8,  8, \
                     2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2,  16,  1 };

    /* gamut */
//  char notes [] = {' ', 'B', 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C', 'D', ' '};
//  int beats [] = {  4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4};

    length = sizeof(notes);
    int tempo = 73;
    int i;

    for(i = 0; i < length; i++) {
        if(notes[i] == ' ') {
            set_bits((0b00000000000));
            delay_ms(beats[i] * tempo);
        } else {
            playNote(notes[i], beats[i] * tempo);
        }
        // Delay between notes
        delay_ms(tempo / 2);
    }
}
void c_music_play(){
    buzzer_init();
    ledbar_init();
    set_bits(0b1111100000);
    delay_ms(1000);
    melody_demo();
}


```

## C Implemented Play
The function c_music_play() calls the C implemented melody with note visualitation on the led bar.


```python
c_music_play()
```

## Python Implemented Play
The cells below using C functions written from the MicroBlaze cell to implemented melody with note visualitation on the led bar.

NOTICE: The advantage of seberating the code at this spot is that aftre the python libarays are load and the microblaze is flashed the python code followed can be executed and debug independantly. This saves a lot of time and in case there is an buzzer involved it saves your ears as well.


```python
def play_melody(notes, beats, notes_key, tempo):
    if len(notes) != len(beats):
        return print('Error: Notes and beats must be of same length!')
    #tempo = 124/1.7
    for index, beat in enumerate(beats):
        if notes[index] == ' ':
            time.sleep(beat * tempo/1000)
~~~

```
            set_level(0, 3, 1)
        else:
      #      set_level(list(notes_key.keys()).index(notes[index]), 3, 1) # not needed due to C implementation
            # my_func( list(notes_key.keys()).index(notes[index]) ) # not working because not wrapped
            playNote(notes_key[notes[index]], int(beat * tempo))
            # print( list(notes_key.keys()).index(notes[index]), notes_key[notes[index]], notes[index])
        # Delay between notes
        time.sleep(tempo / 300);

def music_synt(tempo, notes_key):
    # initialize GPIO, just to enusers GPIO init
    buzzer_init()
    ledbar_init()

    # A-Team part 1, main
    notes = [' ','C','C','g','C','f','g','c','e','g','C','g','D','C','b','a','g','f','g', ' ']
    beats = [  8,  3,  1,  2, 18,  2,  8, 10,  1,  1,  2,  2,  2, 18,  3,  1,  1,  3, 16,  1 ]
    play_melody(notes, beats, notes_key, tempo)

def music_synt2(tempo, notes_key):
    # A-Team part 2
    notes = ['C','C','g','C','e','f','d','g','c','e','f','a','b','b','a',' ','f','c','a' ]
    beats = [ 3 , 1 , 2 , 18, 2 , 2 , 2 , 2 , 16, 3 , 1 , 2 , 50, 2 , 2 , 2 , 2 , 8 , 8  ]
    play_melody(notes, beats, notes_key, tempo)

def music_synt3(tempo, notes_key):
    # A-Team part 3
    notes = [' ','d','f','g','C','g','f',' ','g','f','f','e','c','B','c',' ' ]
    beats = [ 8 , 3 , 1 , 2 , 18, 2 , 2 , 2 , 8 , 8 , 2 , 2,  2,  2,  16, 2 ]
    play_melody(notes, beats, notes_key, tempo)

def music_synt4(tempo, notes_key):
    # A-Team part 4
    notes = ['e','e','d','e',' ','d',' ','e',' ','d',' ','d','a','g' ]
    beats = [ 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2,  8,  8]
    play_melody(notes, beats, notes_key, tempo)

def music_synt5(tempo, notes_key):
    # A-Team part 5
    notes = ['e','e','d','e',' ','d',' ','c',' ','c',' ','c','d', ' ' ]
    beats = [ 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2,  16,  1]
    play_melody(notes, beats, notes_key, tempo)

def A_Team(tempo, notes_key):
    for i in range(1):
        music_synt(tempo, notes_key)
        music_synt2(tempo, notes_key)
        music_synt3(tempo, notes_key)
        music_synt4(tempo, notes_key)
        music_synt5(tempo, notes_key)
    music_synt(tempo, notes_key)

def music_gamut(notes_key):
    # initialize GPIO, to be able to run it independently
    buzzer_init()
    ledbar_init()

    # A-Team
    notes = [' ', 'B', 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C', 'D', ' ']
    beats = [  4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4]
    if len(notes) != len(beats):
        return print('Error: Notes and beats must be of same length!')
    tempo = 124/0.5
```

~~~
    for index, beat in enumerate(beats):
        if notes[index] == ' ':
            time.sleep(beat * tempo/1000)
            set_level(0, 3, 1)
        else:
       #     set_level(list(notes_key.keys()).index(notes[index]), 3, 1)  # not needed due to C implementation
            playNote(notes_key[notes[index]], int(beat * tempo))
            print( 'led level, tune, ascii dec value:',
                  list(notes_key.keys()).index(notes[index]),
                  notes[index],
                  notes_key[notes[index]]
                )
        # Delay between notes
        time.sleep(tempo / 200);
```


```python
if __name__ == '__main__':
    # synthesice music and visualize with LED Bar
    tempo = 124/1.7
    notes_key = { ' ': 32, 'B':66, 'c': 99, 'd':100, 'e':101, 'f':102, 'g':103, 'a':97, 'b':98, 'C': 67, 'D': 68 }
    play = [       ' ', 'B'   ]
    play_beats = [   4,    4 ]
    beat_slide = 4

    # play A-Team theme
    A_Team(tempo, notes_key)
```


```python
# play the gamut of the buzzer
music_gamut(notes_key)
```


```python
# test celll

#playTone(int(1700), int(600))
#print(chr(99))
#notes_key = {'a':97, 'b':98, 'c': 99, 'd':100, 'e':101, 'f':102, 'g':103, ' ': 32, 'C': 67, 'D': 68 }
#notes = ['c', 'd', 'e', 'f', 'g', 'a', 'b', 'C', 'D', ' ', ' ', ' ', ' ', ' ' ]
#for note in notes:
#    playNote(notes_key[note], int(600))
#playNote(notes_key['b'], int(600))
#playNote(notes_key['C'], int(600))
#playNote(notes_key['D'], int(600))
```

## Python GUI
Make your own instend custom music. Us the beat slider to set a beat for a tone and add atone by pressing the coresponding button. It will print your beat and play. Beat log, comming soon! Donations are welcome :)


```python
def play_note(note, beats ):
    global notes_key
    #notes_key = note_key
    # A-Team
    notes = [note ]
    beats = [ beats]
    if len(notes) != len(beats):
~~~

```python
        return print('Error: Notes and beats must be of same length!')
    tempo = 124/1.7
    for index, beat in enumerate(beats):
        if notes[index] == ' ':
            time.sleep(beat * tempo/1000)
            set_level(0, 3, 1)
        else:
            #set_level(list(notes_key.keys()).index(notes[index]), 3, 1)
            playNote(notes_key[notes[index]], int(beat * tempo))

        # Delay between notes
        time.sleep(tempo / 300);


def on_buttonB_clicked(b):
    play_note('B',4)
    play.append('B')
    play_beats.append(beat_slide)
def on_buttonc_clicked(b):
    play_note('c',4)
    play.append('c')
    play_beats.append(beat_slide)
def on_buttond_clicked(b):
    play_note('d',4)
    play.append('d')
    play_beats.append(beat_slide)
def on_buttone_clicked(b):
    play_note('e',4)
    play.append('e')
    play_beats.append(beat_slide)
def on_buttonf_clicked(b):
    play_note('f',4)
    play.append('f')
    play_beats.append(beat_slide)
def on_buttong_clicked(b):
    play_note('g',4)
    play.append('g')
    play_beats.append(beat_slide)
def on_buttona_clicked(b):
    play_note('a',4)
    play.append('a')
    play_beats.append(beat_slide)
def on_buttonb_clicked(b):
    play_note('b',4)
    play.append('b')
    play_beats.append(beat_slide)
def on_buttonC_clicked(b):
    play_note('C',4)
    play.append('C')
    play_beats.append(beat_slide)
def on_buttonD_clicked(b):
    play_note('D',4)
    play.append('D')
    play_beats.append(beat_slide)
def on_buttonPause_clicked(b):
    play_note(' ',4)
    play.append(' ')
    play_beats.append(beat_slide)
def on_buttonRun_clicked(b):
    global play, play_beats, notes_key, tempo
    print('play', play)
    print('play_beats', play_beats)
    play_melody(play, play_beats, notes_key, tempo)
def handle_slider_change(change):
    global beat_slide
    beat_slide = change.new
#def handle_sliderTempo_change(change):
#    global tempo
```

~~~
#    tempo = change.new
#def gui():
buttonB = widgets.Button(description="B", button_style='primary')
buttonc = widgets.Button(description="c", button_style='primary')
buttond = widgets.Button(description="d", button_style='primary')
buttone = widgets.Button(description="e", button_style='primary')
buttonf = widgets.Button(description="f", button_style='primary')
buttong = widgets.Button(description="g", button_style='primary')
buttona = widgets.Button(description="a", button_style='primary')
buttonb = widgets.Button(description="b", button_style='primary')
buttonC = widgets.Button(description="C", button_style='primary')
buttonD = widgets.Button(description="D", button_style='primary')
buttonPause = widgets.Button(description="Pause", button_style='success')
buttonRun = widgets.Button(description="Play Awesome", button_style='danger')
slider = widgets.IntSlider(min=1, max=16, value=4, description='Beat')
#sliderTempo = widgets.IntSlider(min=40, max=150, value=70, description='Tempo')

buttonB.on_click(on_buttonB_clicked)
buttonc.on_click(on_buttonc_clicked)
buttond.on_click(on_buttond_clicked)
buttone.on_click(on_buttone_clicked)
buttonf.on_click(on_buttonf_clicked)
buttong.on_click(on_buttong_clicked)
buttona.on_click(on_buttona_clicked)
buttonb.on_click(on_buttonb_clicked)
buttonC.on_click(on_buttonC_clicked)
buttonD.on_click(on_buttonD_clicked)
buttonPause.on_click(on_buttonPause_clicked)
buttonRun.on_click(on_buttonRun_clicked)
slider.observe(handle_slider_change, names='value')
#slider.observe(handle_sliderTempo_change, names='value')

buttonPause.layout.margin = "0 0 0 500px"
buttonD.layout.margin = "0 0 0 450px"
buttonC.layout.margin = "0 0 0 400px"
buttonb.layout.margin = "0 0 0 350px"
buttona.layout.margin = "0 0 0 300px"
buttong.layout.margin = "0 0 0 250px"
buttonf.layout.margin = "0 0 0 200px"
buttone.layout.margin = "0 0 0 150px"
buttond.layout.margin = "0 0 0 100px"
buttonc.layout.margin = "0 0 0 50px"
buttonB.layout.margin = "0 0 0 0"
buttonRun.layout.margin = "0 0 0 550px"

display(buttonPause)
display(buttonD)
display(buttonC)
display(buttonb)
display(buttona)
display(buttong)
display(buttonf)
display(buttone)
display(buttond)
display(buttonc)
display(buttonB)
display(buttonRun)
display(slider)
#display(sliderTempo)
```

## MicroBlaze Wrapper

Next cell function set_level() is wrapped by MicroBlaze.


```python
set_level
~~~

~~~~
```

Next cell function ledbar_init() is wrapped by MicroBlaze.


```python
ledbar_init
```

It is uncleare why the MicroBlaze wrapes one function but not another.

Next cell function set_bits() is not wrapped by MicroBlaze.


```python
set_bits(5)
```
~~~~

Listing 16: Part III - Jupyter Notebook file MusicSynthesizer saved as *.py file.

References

[1] *XUP PYNQ*. [Online]. Available: https://www.xilinx.com/support/university/boards-portfolio/xup-boards/XUPPYNQ.html (visited on 12/04/2018).

[2] "Python productivity for Zynq (Pynq) Documentation Release 2.2 Xilinx", Tech. Rep., 2018. [Online]. Available: https://media.readthedocs.org/pdf/pynq-testing/image%7B%5C_%7Dv2.2/pynq-testing.pdf.

[3] C. Parikh, "EGR 680 Fall 2018 Final Project PYNQ Embedded Design using Jupyter Notebooks", p. 6, 2018. [Online]. Available: http://www.egr.gvsu.edu/%7B~%7Dparikhc/Chirag%7B%5C_%7DEGR680.html.