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

List of Figures

Listings

1 Introduction

The goal of laboratory ten is to familiarize the student with the Jupyter Notebook and debugging of hardware in Vivado.

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.

- 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.

```
1 from .myrgbled import MYRGBLED
```

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

```
1 self.rgbleds = ([None] * 4) + [pynq.lib.MYRGBLED(i)
2                                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
```

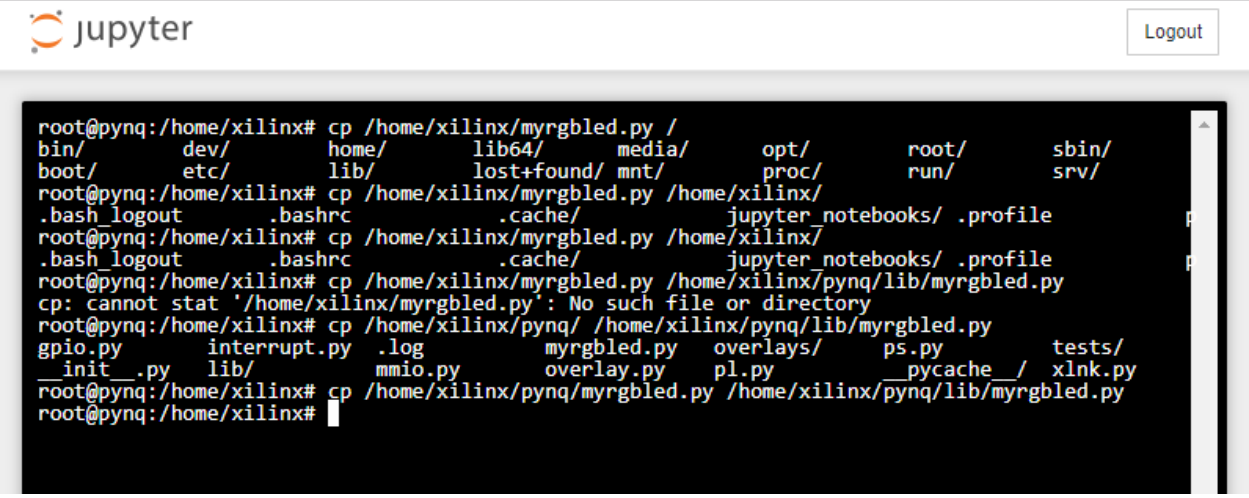

Figure 1: Jupiter Notebook terminal, copy a file.

Notice: The original functions of the driver remind. This provides back works 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

designed GUI is shown in Figure 2. 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.

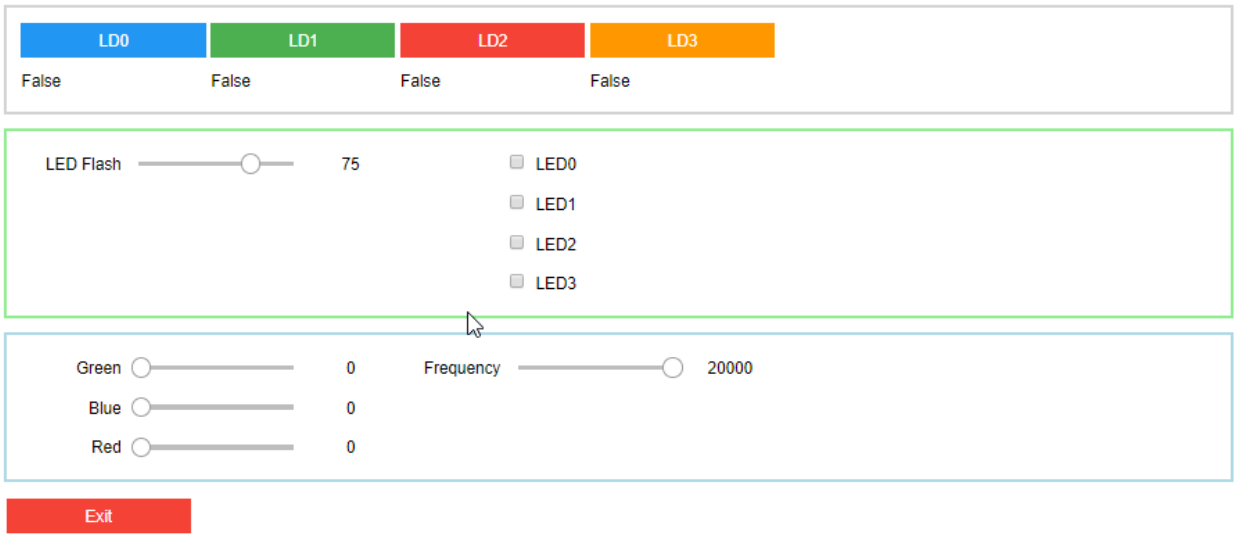

Figure 2: Jupiter Notebook GUI to control the green LEDs and a RGB LED.

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 3. The source code is shown in Listing 9 in Section 4.2 of the appendix.

Figure 3: Groove LED Bar program output.

2.3 Part III - Music Synthesizer

3 Conclusion

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
```

```
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 5: Jupyter Notebook file __init__ saved as *.py file.

```
#   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"
```

```
__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 6: 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

```

```
133         """
134 #         if color not in range(8):
135 #             raise ValueError("color should be an integer value from 0 to 7.")
136         new_val = (MYRGBLED._rgbleds_val ) | (RGB_RED << ((self.index-RGBLEDS_START_INDEX)
    *3))
137 #         print(MYRGBLED._rgbleds_val)
138 #         print(new_val)
139 #         print( (RGB_RED << ((self.index-RGBLEDS_START_INDEX)*3)) )
140         self._set_rgbleds_value(new_val)
141 
142     def red_off(self):
143         """Turn off a single RGB LED with color value red.
144 
145         Parameters
146         ----------
147         color : int
148            Color of RGB specified by a 3-bit RGB integer value.
149 
150         Returns
151         -------
152         None
153 
154         """
155 #         if color not in range(8):
156 #             raise ValueError("color should be an integer value from 0 to 7.")
157 
158         new_val = (MYRGBLED._rgbleds_val ) &~ (RGB_RED << ((self.index-RGBLEDS_START_INDEX)
    *3))
159         self._set_rgbleds_value(new_val)
160 
161     def green_on(self):
162         """Turn on a single RGB LED with color value green.
163 
164         Parameters
165         ----------
166         color : int
167            Color of RGB specified by a 3-bit RGB integer value.
168 
169         Returns
170         -------
171         None
172 
173         """
174 #         if color not in range(8):
175 #             raise ValueError("color should be an integer value from 0 to 7.")
176 
177         new_val = (MYRGBLED._rgbleds_val ) | (RGB_GREEN << ((self.index-RGBLEDS_START_INDEX)
    *3))
178         self._set_rgbleds_value(new_val)
179 
180     def green_off(self):
181         """Turn off a single RGB LED with color value green.
182 
183         Parameters
184         ----------
185         color : int
186            Color of RGB specified by a 3-bit RGB integer value.
187 
188         Returns
189         -------
190         None
191 
192         """
193 #         if color not in range(8):
194 #             raise ValueError("color should be an integer value from 0 to 7.")
195 
196         new_val = (MYRGBLED._rgbleds_val ) &~ (RGB_GREEN << ((self.index-RGBLEDS_START_INDEX
    )*3))
```

```
197         self._set_rgbleds_value(new_val)
198
199     def blue_on(self):
200         """Turn on a single RGB LED with color value blue.
201
202         Parameters
203         ----------
204         color : int
205            Color of RGB specified by a 3-bit RGB integer value.
206
207         Returns
208         -------
209         None
210
211         """
212 #         if color not in range(8):
213 #             raise ValueError("color should be an integer value from 0 to 7.")
214
215         new_val = (MYRGBLED._rgbleds_val ) | (RGB_BLUE << ((self.index-RGBLEDS_START_INDEX)
    *3))
216         self._set_rgbleds_value(new_val)
217
218     def blue_off(self):
219         """Turn off a single RGB LED with color value blue.
220
221         Parameters
222         ----------
223         color : int
224            Color of RGB specified by a 3-bit RGB integer value.
225
226         Returns
227         -------
228         None
229
230         """
231 #         if color not in range(8):
232 #             raise ValueError("color should be an integer value from 0 to 7.")
233
234         new_val = (MYRGBLED._rgbleds_val ) &~ (RGB_BLUE << ((self.index-RGBLEDS_START_INDEX)
    *3))
235         self._set_rgbleds_value(new_val)
236
237     def status(self):
238         rgb_mask = 0x7 << ((self.index-RGBLEDS_START_INDEX)*3)
239         return ((MYRGBLED._rgbleds_val )& ~rgb_mask)
240
241     def rgb_on(self, color):
242         """Turn on a single RGB LED color.
243
244         Parameters
245         ----------
246         color : int
247            Color of RGB specified by a 3-bit RGB integer value.
248                Blue  = 1
249                Green = 2
250                Red = 4
251
252         Returns
253         -------
254         None
255
256         """
257         if color not in [1, 2, 4]:
258             raise ValueError("color should be an integer value from 1, 2, and 4.")
259
260         new_val = (MYRGBLED._rgbleds_val ) | (color << ((self.index-RGBLEDS_START_INDEX)*3))
261         self._set_rgbleds_value(new_val)
262
```

```
263     def rgb_off(self, color):
264         """Turn off a single RGB LED color.
265 
266         Parameters
267         ----------
268         color : int
269             Color of RGB specified by a 3-bit RGB integer value.
270                 Blue  = 1
271                 Green = 2
272                 Red = 4
273         Returns
274         -------
275         None
276 
277         """
278         if color not in [1, 2, 4]:
279             raise ValueError("color should be an integer value from 1, 2, and 4.")
280 
281         new_val = (MYRGBLED._rgbleds_val ) &~ (color << ((self.index-RGBLEDS_START_INDEX)*3)
    )
282         self._set_rgbleds_value(new_val)
283 
284     def pwm(self, color, duty_cycle, frequency):
285     """PWM for single RGB LED color.
286 
287     Parameters
288     ----------
289     color : int 1, 2 or 3
290       Color of RGB specified by a 3-bit RGB integer value.
291         Blue  = 1
292         Green = 2
293         Red = 4
294     duty_cycle : int between 0 and 100
295       Duty cycle is an integer value between 0 and 100 %
296       ____+____+____+____+____+ is a duty cyle of 50 %
297     frequency : int
298       Frequency defines the length of the intervall
299 
300         Returns
301         -------
302         None
303 
304         """
305         if color not in [1, 2, 4]:
306             raise ValueError("color should be an integer value from 1, 2, and 4.")
307 
308         self.rgb_on(color)
309         time.sleep( duty_cycle / frequency )
310         self.rgb_off(color)
311         time.sleep( (100-duty_cycle) / frequency)
312 
313     def write(self, color):
314         """Set the RGBLED state according to the input value.
315 
316         Parameters
317         ----------
318         color : int
319             Color of RGB specified by a 3-bit RGB integer value.
320 
321         Returns
322         -------
323         None
324 
325         """
326         self.on(color)
327 
328     def read(self):
329         """Retrieve the RGBLED state.
```

```
330
331         Returns
332         -------
333         int
334             The color value stored in the RGBLED.
335
336         """
337         return (MYRGBLED._rgbleds_val >>
338                 ((self.index-RGBLEDS_START_INDEX)*3)) & 0x7
339
340     @staticmethod
341     def _set_rgbleds_value(value):
342         """Set the state of all RGBLEDs.
343
344         Note
345         ----
346         This function should not be used directly. User should call
347         `on()`, `off()`, instead.
348
349         Parameters
350         ----------
351         value : int
352             The value of all the RGBLEDs encoded in a single variable.
353
354         """
355         MYRGBLED._rgbleds_val = value
356         MYRGBLED._mmio.write(RGBLEDS_XGPIO_OFFSET, value)
```

Listing 7: Jupyter Notebook file myrgbled saved as *.py file.

```
1
2 # coding: utf-8
3
4 # ## LED Ctrl RGB
5 #
6 # Use buttons and sliders to control the LEDs on the board.
7 #
8 # The program is started by select Menubar -> Cell -> Run All
9 #
10 # Cell -> Current Outputs -> Toggle Scrolling
11 #
12
13 # In[1]:
14
15
16 import time
17 from pynq.overlays.base import BaseOverlay
18
19 import ipywidgets as widgets
20 from IPython.display import display
21 from multiprocessing import Process
22 from multiprocessing.sharedctypes import Value
23
24 base = BaseOverlay("base.bit")
25
26
27 # ### Define functions here
28 # Function decision() provides the computaion of the win and loss with average and a consol
    ouput accordingly.
29 # #### Colors RGB LED No 4 and 5
30 #     off = 0     blue = 1     green = 2     tÃ¼rkies = 3     red = 4     purple = 5     yellow =
    6
31 #     white = 7
32 #
33
34 # In[19]:
35
36
```

```
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):
```

```
    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 8: 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 9: Part II - LED Groove Bar Python code.

4.3 Python code Listings Part III - Music Synthesizer

```

## Music Sytheziser

Place description


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

```
from time import sleep
```


```python
%%microblaze base.PMODA
#include "xparameters.h"
#include "timer.h"
#include "circular_buffer.h"
#include "gpio.h"
#include "pmod_grove.h"

// Mailbox commands
#define CONFIG_IOP_SWITCH       0x1
#define PLAY_TONE               0x3
#define PLAY_DEMO               0x5

// Speaker channel
#define SPEAKER_CHANNEL 1

// 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 playNote(char note, int duration) {

    char names[] = { 'c',  'd',  'e',  'f',  'g',  'a',  'b', 'C',  'D',  };
    int tones[] = { 1916, 1700, 1519, 1432, 1275, 1136, 1014, 956, 702 };
    int i;

    // play the tone corresponding to the note name
    for (i = 0; i < 8; i++) {
        if (names[i] == note) {
          playTone(tones[i], duration);
        }
    }
}

void melody_demo(void) {
    // The number of notes
    int length = 15;
    char notes[] = "ccggaagffeeddc ";
    int beats[] = { 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 4 };
    int tempo = 300;
    int i;

    for(i = 0; i < length; i++) {
```

```
        if(notes[i] == ' ') {
            delay_ms(beats[i] * tempo);
        } else {
            playNote(notes[i], beats[i] * tempo);
        }
        // Delay between notes
        delay_ms(tempo / 2);
    }
}
#include "gpio.h"
#include "timer.h"
#include "circular_buffer.h"
#include <pyprintf.h>
#include <pmod_grove.h>


// Work on 8-bit mode
#define CONFIG_IOP_SWITCH           0x1
#define RESET                       0x3
#define WRITE_LEDS                  0x5
#define SET_BRIGHTNESS              0x7
#define SET_LEVEL                   0x9
#define READ_LEDS                   0xB

/*
 * 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 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;

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);
    //pyprintf(" --> ledbar init done\n");
}

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

    data_internal = data;

    clkval = 0;
```

```
147     gpio_write(gpio_data, 0);
148     // First toggle the clock 8 times
149     for (i = 0; i < 8; ++i) {
150          clkval ^= 1;
151          gpio_write(gpio_clk, clkval);
152     }
153 
154     // Working in 8-bit mode
155     for (i = 0; i < 8; i++){
156         /*
157          * Read each bit of the data to be sent LSB first
158          * Write it to the data_pin
159          */
160         data_state = (data_internal & 0x80) ? 0x00000001 : 0x00000000;
161         gpio_write(gpio_data, data_state);
162         clkval ^= 1;
163         gpio_write(gpio_clk, clkval);
164 
165         // Shift Incoming data to fetch next bit
166         data_internal = data_internal << 1;
167     }
168 }
169 
170 void latch_data(){
171     int i;
172     gpio_write(gpio_data, 0);
173     delay_ms(10);
174 
175     // Generate four pulses on the data pin as per data sheet
176     for (i = 0; i < 4; i++){
177         gpio_write(gpio_data, 1);
178         gpio_write(gpio_data, 0);
179     }
180 }
181 
182 u16 reverse_data(u16 c){
183     /*
184      * Function to reverse incoming data
185      * Allows LEDbar to be lit in reverse order
186      */
187     int shift;
188     u16 result = 0;
189 
190     for (shift = 0; shift < 16; shift++){
191         if (c & (0x0001 << shift))
192             result |= (0x8000 >> shift);
193     }
194 
195     // 10 LSBs are used as LED Control 6 MSBs are ignored
196     result = result >> 6;
197     return result;
198 }
199 
200 void set_bits(u16 data){
201     int h,i;
202     int data_internal = data;
203 
204     for(h=0; h<10; h++){
205         ledbar_state[h] = HIGH;
206     }
207 
208     send_data(GLB_CMDMODE);
209 
210     for (i = 0; i < 10; i++){
211         if ((data_internal & 0x0001) == 1) {
212             send_data(ledbar_state[i]);
213         } else {
214             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 ledbar_read(){
    int h;
    u16 bits;
```

```
    bits = 0x0000;
    for(h=0; h<10; h++){
        if (current_state[h] != 0x00) {
            bits |= 0x0001 << h;
        }
    }
    bits = bits & 0x03FF;
    return bits;
}
```

```python
def music_synt():
    # initialize GPIO
    #
    buzzer_init()
    ledbar_init()
    # The number of notes
    length = 15
    notes_key = {'c': 99, 'd':100, 'e':101, 'f':102, 'g':103, 'a':97, 'b':98, 'C': 67, 'D': 68 , ' ': 32}

    # A-Team
    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, 8 ]
    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(10, 1, 1)
        else:
            #print('else: ',chr(notes[index]), int(beat * tempo))
            playNote(notes_key[notes[index]], int(beat * tempo))
            set_level(list(notes_key.keys()).index(notes[index])+1, 1, 1)
        # Delay between notes
        time.sleep(tempo / 300);
```

```python
if __name__ == '__main__':
    # synthesice music and visualize with LED Bar
    for i in range(3):
        music_synt()
```

```python

#ledbar_init()
#a = ledbar_read(void)
#set_bits(0b100000001 & 0xFFFF)
#set_level(5,1,1)
```

```python
#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))
'''
```

Listing 10: Jupyter Notebook file MusicSynthesizer saved as *.py file.