Development of Camera Module for Microbolometer Based Infrared Imaging Applications

Kerem Şahin e-mail keremsahin@sabanciuniv.edu
Electronics Engineering

Jiyan Baran Bükun e-mail jiyanbaran@sabanciuniv.edu
Electronics Engineering

Oğuz Şensoz e-mail oguzsensoz@sabanciuniv.edu
Electronics Engineering

Muhtasham Oblokulov e-mail muhtasham@sabanciuniv.edu
Electronics Engineering

Prof. Yaşar Gürbuz
Electronics Engineering

Dr. Ömer Ceylan
Electronics Engineering

Dr. Melik Yazıcı
Electronics Engineering

Abstract

Infrared imaging technology has a very large usage area. The applications of IR imaging can be divided into two main field such as military applications and civilian applications. For military applications unmanned aerial vehicles (UAV), missile applications, forward-looking infrared cameras (FLIR) can be good examples. On the other hand, this technology is also using in civil applications such as industrial solutions, medical analysis, astronomy and automotive industry.
In our project the main goal is developing a new camera module that include microbolometer based infrared imaging sensor and FPGA with appropriate optics. Microbolometer is IR light-sensitive structure and when IR light drops on microbolometer, the resistance values change. These changes are

realized by ROIC and delivered to the FPGA. FPGA process the information provided by ROIC and it gives a video as an output.

Keywords: Readout integrated circuits (ROIC), microbolometer, infrared imaging, FPGA

1 Introduction

Infrared light is firstly introduced in 1880 by William Herschel. Herschel was using triangular prism in order to differentiate sunlight into it's spectrums. After putting thermometer under every part of triangular prism he realized that the level of mercury in thermometer which is under the red light increase much more than the others. Herschel realize that there should be another type of light that is invisible, and he named this type as calorific lights. The term of "infrared" get into literature in late of 19th century.

Every object radiates infrared lights because of the temperature in their structures. The amount of temperature, that any object has, determine the wavelength of emitted lights. Most of the time these radiations are not visible or in other words the wavelength of the radiated light is out of the range of visible lights. For instance, infrared lights have longer wavelength (700nm-1mm) and it is not possible to observe these lights with our eyes. There are some special instruments developed that provide an opportunity to make some observations on infrared light. Night-vision goggles and infrared cameras can be some good examples. These instruments allow us to see infrared waves that are emitting from warm object such as humans and animals.("Infrared Waves | Science Mission Directorate," n.d.)

Infrared imaging technique is also using for examinations of stars, nebulous, planets which have a respectively colder surface than the other space objects. This improvement leads to discovery of several nebulous, cool stars and many other space objects. By infrared imaging we can measure the change of the number of photons in blood and with this technique we may have some opinions about several illnesses. Another way to detect illnesses by using IR imaging is examining the symmetric part of the body with thermal camera. Since these parts have same structure it should be same amount of radiation. If the amount of radiation differs with that symmetric parts then it could be an illness in that part of the body.(Dergipark)

IR imaging also has a crucial importance in military applications. There are a lot of infrared cameras on fighter jets or unmanned aerial vehicles that provide a strong vision to monitor an area that is not visible. Also, in missile applications there are some IR detectors using in order to find the exact place of target in every condition. Forward looking infrared cameras are other instruments that usually using for surveillance. By using FLIR cameras pilot or driver will have clear vision even in some conditions that do not allow to see the way clearly.

In this project the main purpose is developing a microbolometer based infrared camera module. The system can be examined under four main structures. System starts with optics that are using for coupling infrared radiation and focus them on the infrared light-sensitive detector. After IR light has been sensed by detector the values of resistance that are rely on microbolometer change. As the values of resistance change, the current passing through readout integrated circuit (ROIC) will also change. These changes are collecting by ROIC, converted to digital signals (by the internal ADC of ROIC) and stored in RAM as 12-bit data. By using Python programming language data transferred from the RAM to the FPGA. FPGA is processing and color mapping these data's. As an outcome of the project we get an infrared video.

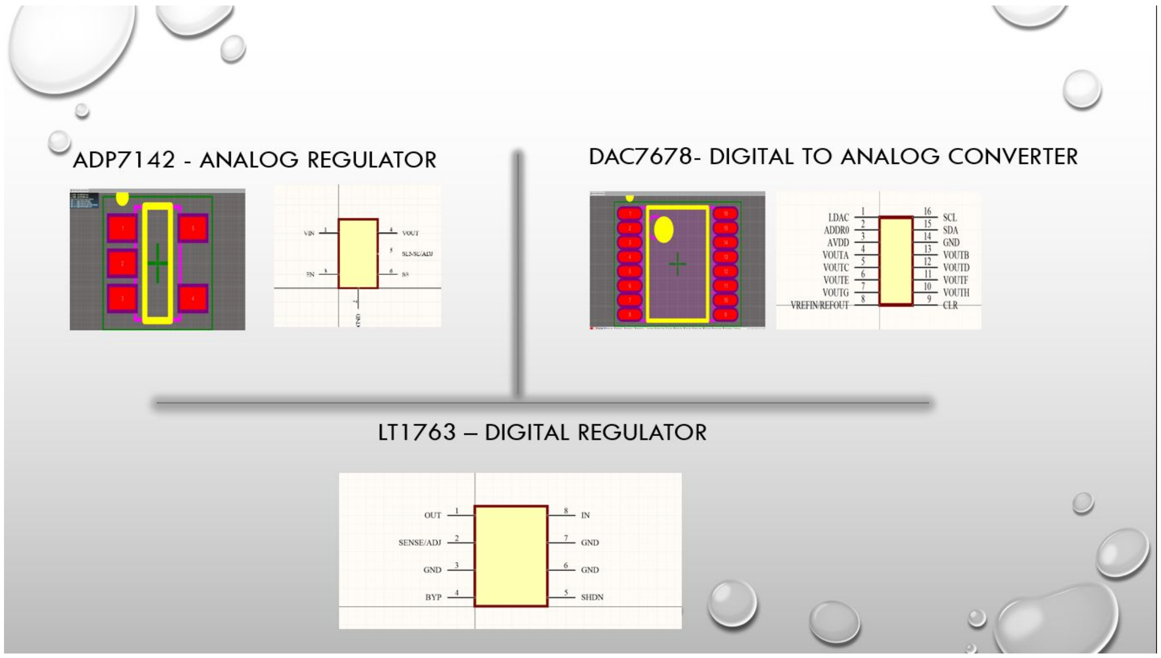
Applications and Implementations

In this part the design of power board and software applications will be explained in details.

1.1 PCB design of Power Board

One of the main component of infrared camera is power board. Power board is providing bias voltages to the system in order to activate readout circuit and FPGA. Bias voltages means constant voltages that allows components to work in their maximum capability. There are two analog regulator and one digital regulator take part on the power board. ADP7142 is the analog regulator that has been used in this system. ADP7142 has a 5V output voltage as it's default property. One of the ADP7142 has been used as a power source of DAC (digital-to-analog converter). In order to run the DAC properly 5.2V bias voltage was required. Due to this reason a voltage divider circuit has been established and added to the ADP7142. By this way analog regulator produce output voltage by the amount of 5.2V instead of 5V. Other analog regulator is producing bias voltage to microbolometer. LT1763 has 3.3V output voltage as it's default property. This digital regulator is also using for providing bias voltages to the microbolometer and ROIC. Front side of power board has a connection between microbolometer and FPGA will plug in to the back side of board. So there are 4 headers in different location of the board.

Altium Designer software has been used to design the board. Normally most of the components are already have their libraries in Altium. However the analog regulators and digital-to-analog converter that have been used in project were not introduced to software before. So their libraries created manually. In order to create a library in Altium, schematic and pcb drawing must be provided to the program. In schematic of a component the circuit that component work with it take place. Connections of components and names of the pins are also indicated in schematic. After schematic is completed pcb design of component can start. In pcb design components are fully introduced. In this process informations about dimensions of the component have been given by looking at the footprint of them in datasheet.



There are some other components such as capacitors and resistors. The model and type of these components have already been indicated in the datasheet.
For instance;
ADP7142 is working with:

- 2 capacitor of 2.2uF
- 2 resistors of 1K and 10K

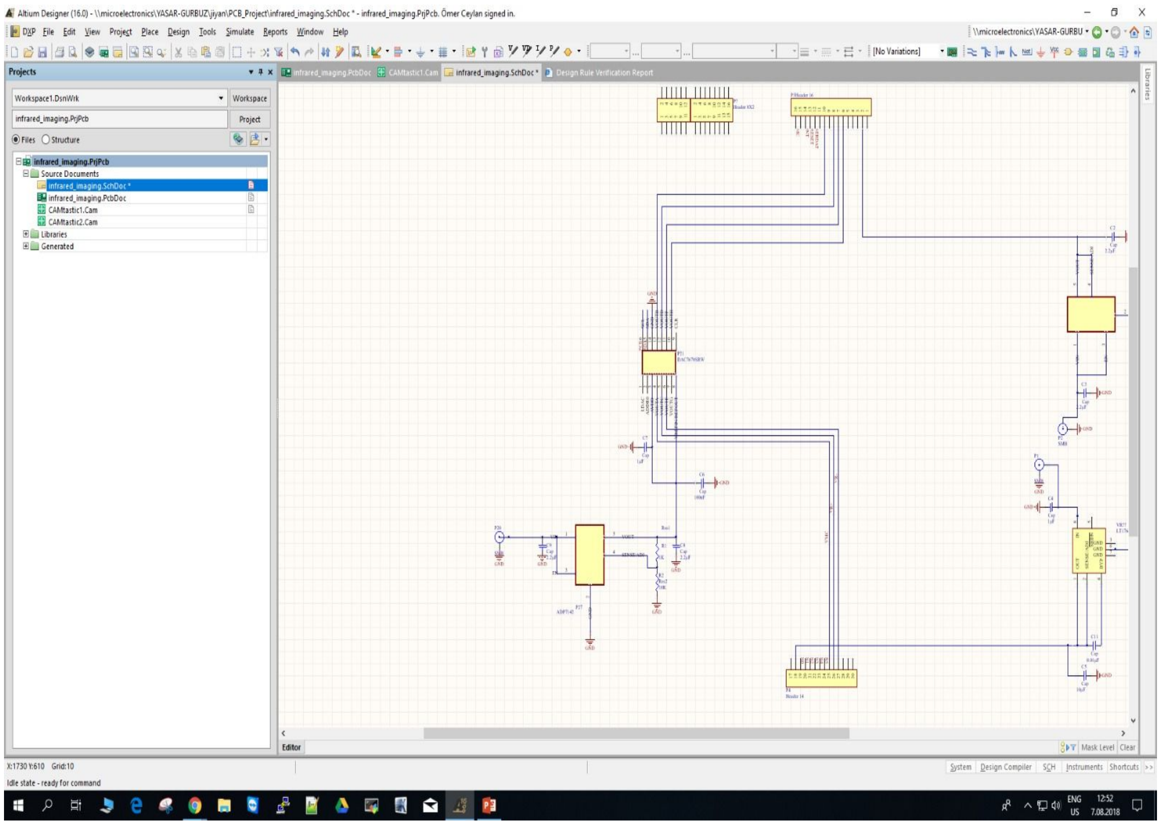
LT1763 is working with:

- 3 capacitors of 0.01uF, 1uF, 10uF

DAC7142 is working with:

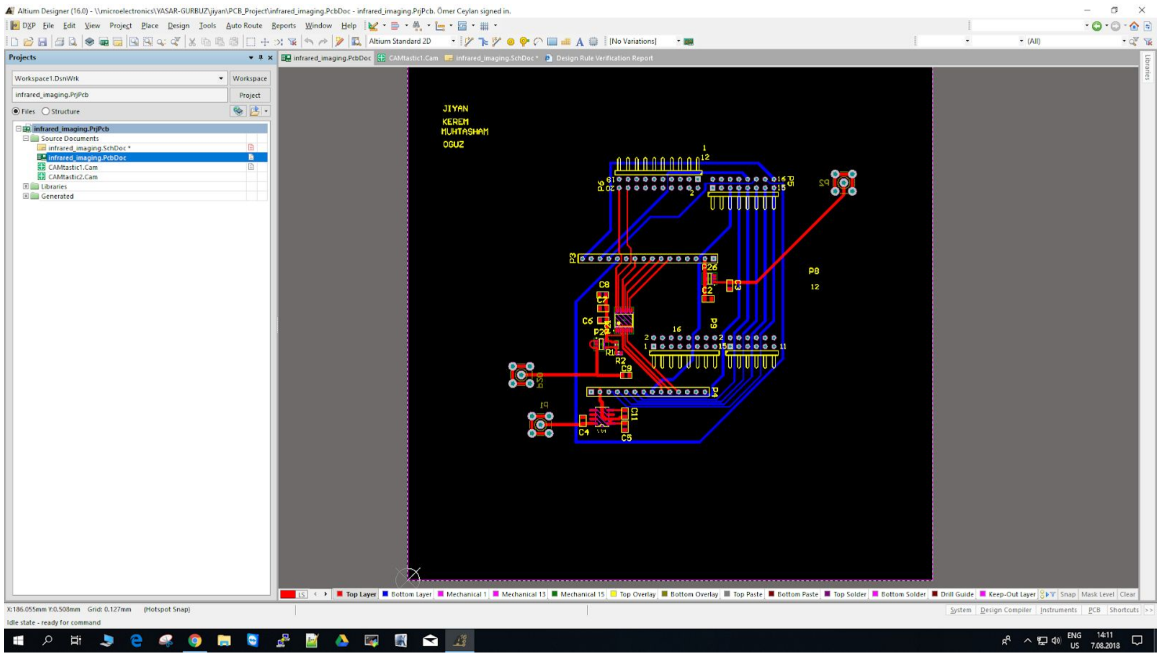
- 2 capacitors of 100nF and 1uF

Since software includes every capacitors and resistors that are required for this project, their libraries used automatically.


Altium Designer (16.0) - \\microelectronics\YASAR-GURBUZ\jiyan\PCB_Project\infrared_imaging.SchDoc * - infrared_imaging.PrjPcb. Ömer Ceylan signed in.
DXP File Edit View Project Place Design Tools Simulate Reports Window Help
Projects
Workspace1.DsnWrk
Workspace
infrared_imaging.PrjPcb
Project
Files Structure
infrared_imaging.PrjPcb
Source Documents
infrared_imaging.SchDoc *
infrared_imaging.PcbDoc
CAMtastic1.Cam
CAMtastic2.Cam
Libraries
Generated
infrared_imaging.PcbDoc
CAMtastic1.Cam
infrared_imaging.SchDoc *
Design Rule Verification Report
Editor
X:1730 Y:610 Grid:10
Idle state - ready for command
System Design Compiler SCH Instruments Shortcuts

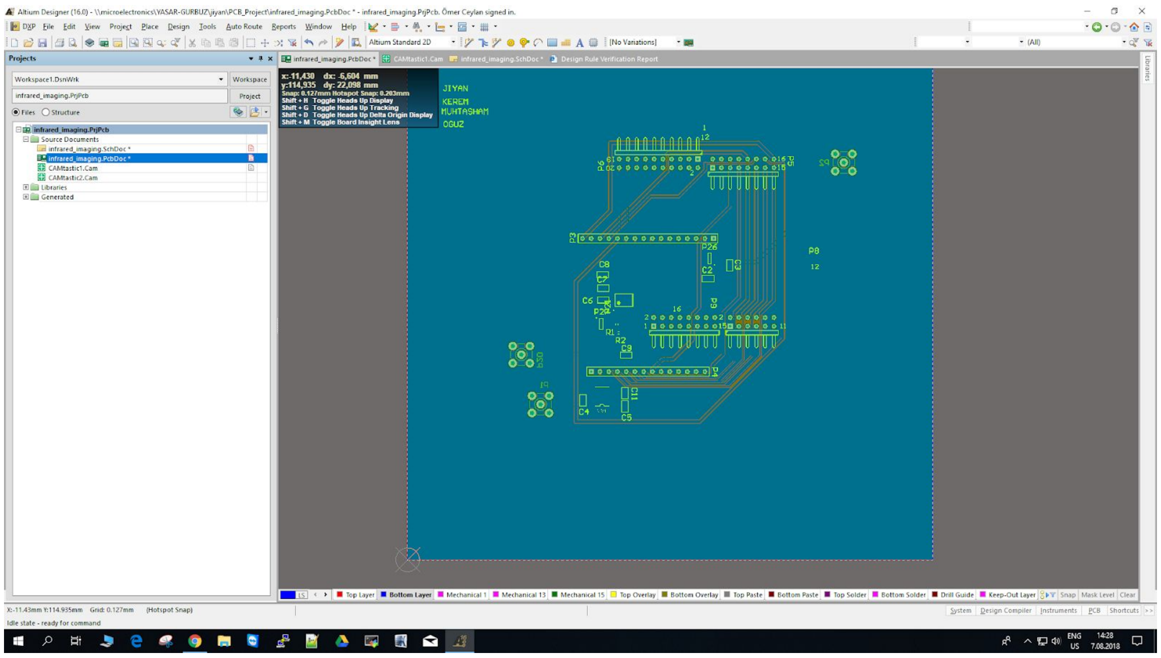
Schematic Design Of PCB

After the schematic design of the system is done, we import all the changes to the file that will be considered as the real structure of the board. This step is call “designation of the layout”. All the connections have been defined in schematic. So that while importing changes from schematic to the pcb layout the connections are also importing as well. There are some rules to be considered in order to avoid any problem that decrease boards performance. For instance thickness of the wires or distance between components are important factors. Connections in pcb layout must be done by considering these rules.

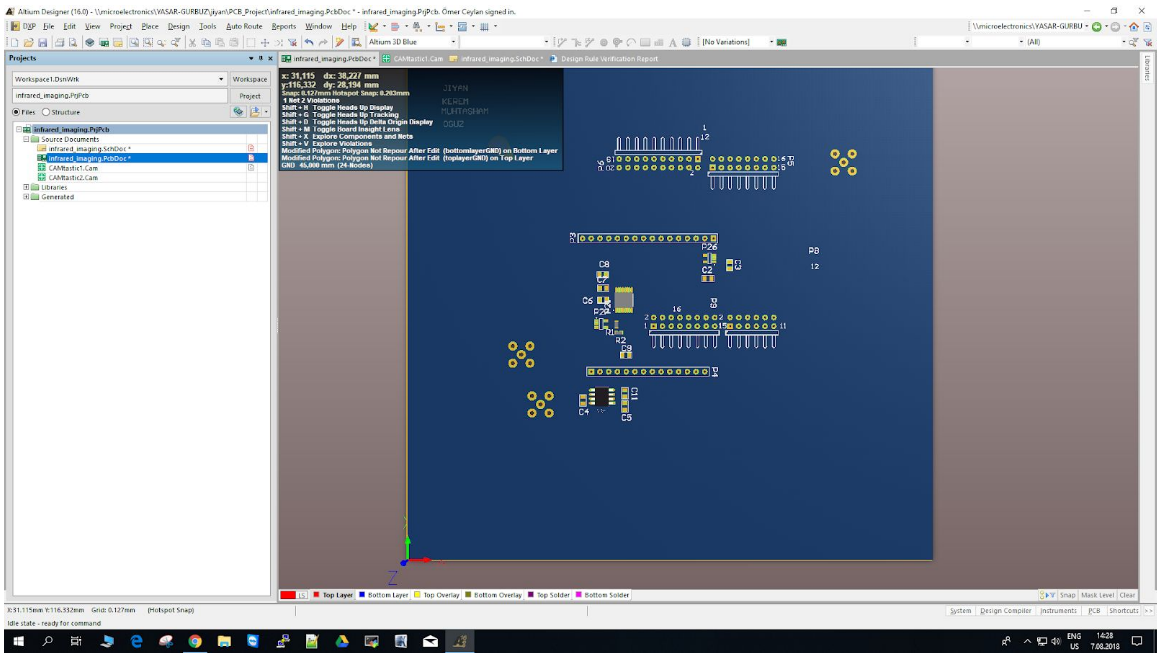


PCB Layout

Every component has a pin that is connected to the ground. In PCB design all of the board defining as a ground in order to avoid individual grounds. This process called polygon pour. It is an automatic application done by software.



Last step of pcb design is using rule check wizard. In order to run system properly there should not be any mistake on the board. So that the warnings that are showing by the rule check wizard corrected by project members.
Finally the board become ready to be printed. For this purpose LPFK S63 board printer has been used.



```
x: 31,115   dx: 38,227 mm
y:116,332   dy: 28,194 mm
Snap: 0.127mm Hotspot Snap: 0.203mm
 1 Net 2 Violations
Shift + H  Toggle Heads Up Display
Shift + G  Toggle Heads Up Tracking
Shift + D  Toggle Heads Up Delta Origin Display
Shift + M  Toggle Board Insight Lens
Shift + X  Explore Components and Nets
Shift + V  Explore Violations
Modified Polygon: Polygon Not Repour After Edit  (bottomlayerGND) on Bottom Layer
Modified Polygon: Polygon Not Repour After Edit  (toplayerGND) on Top Layer
GND  45,000 mm  (24-Nodes)
```

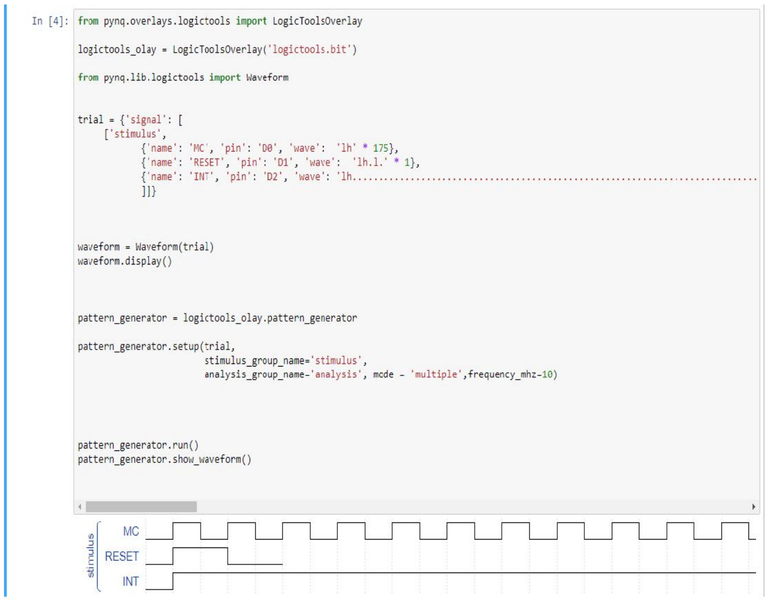
2.2 Generating Clock Signals From FPGA

There is a part called sequencer inside the readout circuit(ROIC) of the microbolometer. This sequencer takes different external clock signals as input and generates internal signals for a synchronous ROIC operation. The necessary clocks to operate the sequencer are Master Clock(MC), Integration Phase(INT) and RESET.

To generate the clock signals that are discussed we use PYNQ-Z1 board and programmed it using Jupyter Notebook via ethernet interface. According to PYNQ official website: "The Xilinx® Zynq® All Programmable device is an SOC based on a dual-core ARM® Cortex®-A9 processor (referred to as the *Processing System* or **PS**), integrated with FPGA fabric (referred to as *Programmable Logic* or **PL**). The *PS* subsystem includes a number of dedicated peripherals (memory controllers, USB, Uart, IIC, SPI etc) and can be extended with additional hardware IP in a *PL* Overlay.

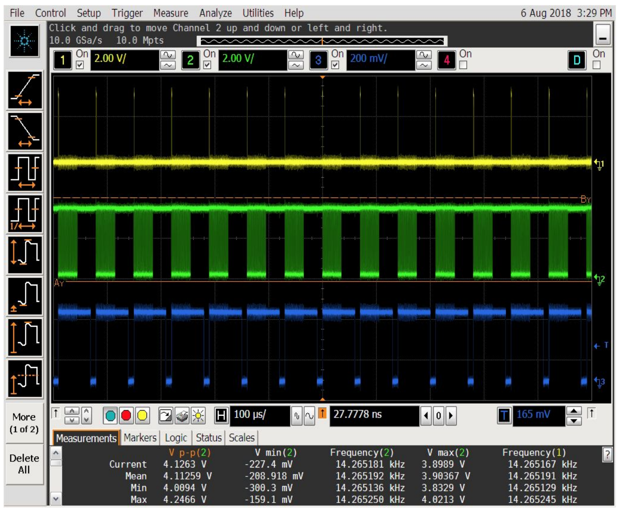
Overlays, or hardware libraries, are programmable/configurable FPGA designs that extend the user application from the Processing System of the Zynq into the Programmable Logic. Overlays can be used to accelerate a software application, or to customize the hardware platform for a particular application." ("PYNQ Overlays — Python productivity for Zynq (Pynq) v1.0," n.d.)

Hence for clock generation purpose we used the Logictools overlays' Pattern Generation subgroup.After examining example notebooks and libraries to learn syntax we were able to generate clock sequence (Figure 1), and also measured it via digital Oscilloscope (Figure 2).When we set desired period parameters the board was not able to generate the sequence due to limit of storing wavelength of reset signal.After, our supervisors advised us to solve this issue with an alternative way.They suggested to generate pattern using Finite State machine subgroup of logic tools overlay.

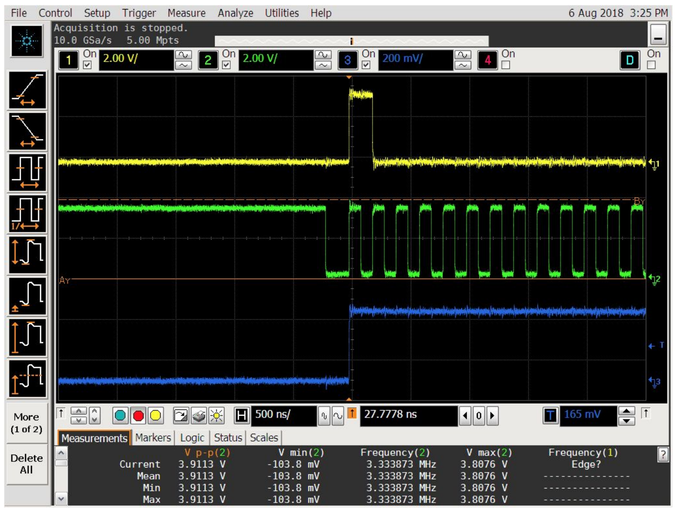


```
In [4]: from pynq.overlays.logictools import LogicToolsOverlay

        logictools_olay = LogicToolsOverlay('logictools.bit')

        from pynq.lib.logictools import Waveform


        trial = {'signal': [
            ['stimulus',
                 {'name': 'MC', 'pin': 'D0', 'wave':  'lh' * 175},
                 {'name': 'RESET', 'pin': 'D1', 'wave':  'lh.l.' * 1},
                 {'name': 'INT', 'pin': 'D2', 'wave': 'lh.............................................................................
                 ]]}



        waveform = Waveform(trial)
        waveform.display()



        pattern_generator = logictools_olay.pattern_generator

        pattern_generator.setup(trial,
                                stimulus_group_name='stimulus',
                                analysis_group_name-'analysis', mode - 'multiple',frequency_mhz-10)




        pattern_generator.run()
        pattern_generator.show_waveform()
```

Figure 1. Patterns which were generated initially.



```
6 Aug 2018  3:29 PM
Click and drag to move Channel 2 up and down or left and right.
10.0 GSa/s   10.0 Mpts
1 2.00 V/   2 2.00 V/   3 200 mV/
H 100 µs/   27.7778 ns   T 165 mV

              V p-p(2)     V min(2)      Frequency(2)     V max(2)     Frequency(1)
   Current    4.1263 V     -227.4 mV     14.265181 kHz    3.8989 V     14.265167 kHz
      Mean    4.11259 V    -208.918 mV   14.265192 kHz    3.90367 V    14.265191 kHz
       Min    4.0094 V     -300.3 mV     14.265136 kHz    3.8329 V     14.265129 kHz
       Max    4.2466 V     -159.1 mV     14.265250 kHz    4.0213 V     14.265245 kHz
```

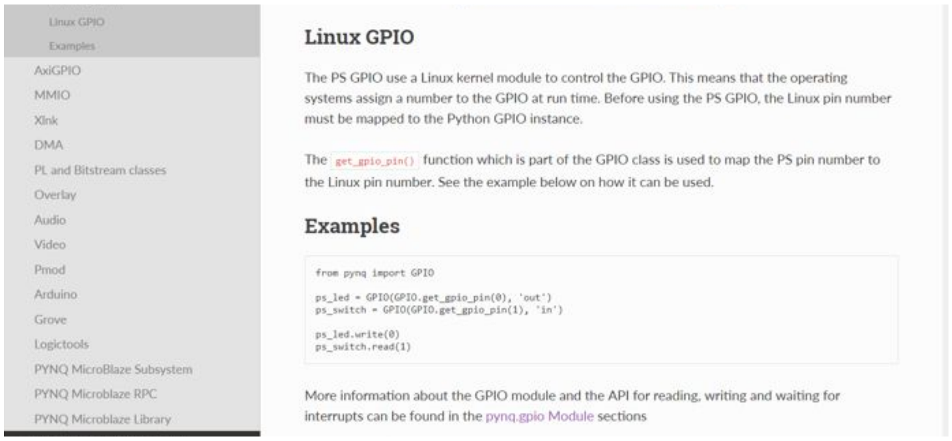


```
6 Aug 2018  3:25 PM
Acquisition is stopped.
10.0 GSa/s   5.00 Mpts
1 2.00 V/   2 2.00 V/   3 200 mV/
H 500 ns/   27.7778 ns   T 165 mV

              V p-p(2)     V min(2)      Frequency(2)     V max(2)     Frequency(1)
   Current    3.9113 V     -103.8 mV     3.333873 MHz     3.8076 V        Edge?
      Mean    3.9113 V     -103.8 mV     3.333873 MHz     3.8076 V     ---------------
       Min    3.9113 V     -103.8 mV     3.333873 MHz     3.8076 V     ---------------
       Max    3.9113 V     -103.8 mV     3.333873 MHz     3.8076 V     ---------------
```

Figure 2. Measuring and verifying patterns in oscilloscope.

2.3 Reading 12-Bit Data From Microbolometer

A major part in this project is to program the FPGA board using python libraries provided by PYNQ to stream video output from HDMI OUT pin. For this purpose we first want to read 12-bit data coming from microbolometer to FPGA I/O pins. Each 12-bit data represents one pixel and when we do reading 320 times we will get the row of a frame. While reading the data we also want to store it to DRAM. After reading 240 rows we will have one frame in DRAM ready to be used as output from HDMI OUT port. For reading the data we think we can use GPIO(general purpose input output) library of PYNQ, we may create a GPIO instance with giving GPIO number and direction(input or output) information. For now we created an GPIO instance for test purposes and tried to write "1"(true) value to D0 pin but when we measured the voltage from I/O pins we couldn't measure voltage difference. We will both continue to work for finding and correcting our mistakes and at the same time research for new tools or libraries to be able to read data and store it into DRAM.



Linux GPIO

The PS GPIO use a Linux kernel module to control the GPIO. This means that the operating systems assign a number to the GPIO at run time. Before using the PS GPIO, the Linux pin number must be mapped to the Python GPIO instance.

The `get_gpio_pin()` function which is part of the GPIO class is used to map the PS pin number to the Linux pin number. See the example below on how it can be used.

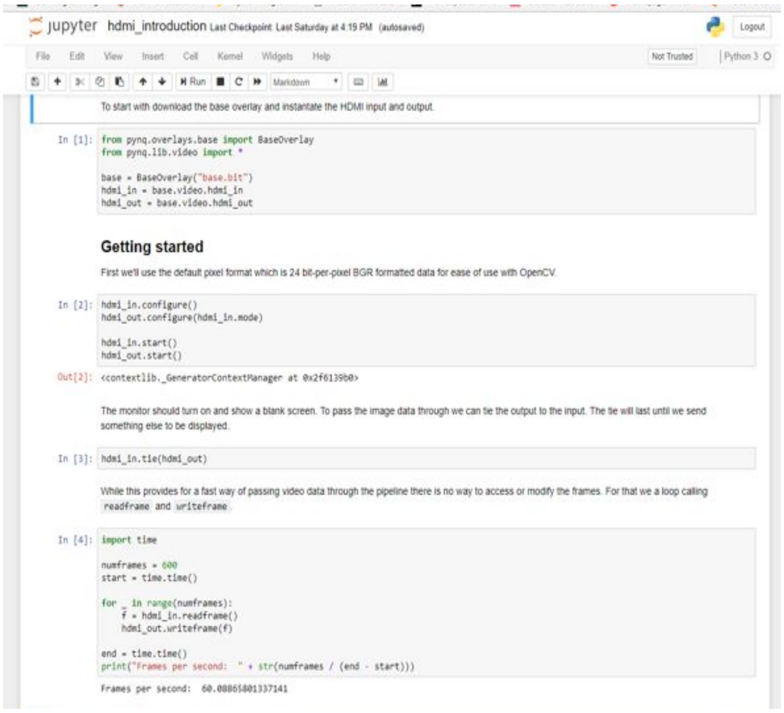
Examples

```
from pynq import GPIO

ps_led = GPIO(GPIO.get_gpio_pin(0), 'out')
ps_switch = GPIO(GPIO.get_gpio_pin(1), 'in')

ps_led.write(0)
ps_switch.read(1)
```

More information about the GPIO module and the API for reading, writing and waiting for interrupts can be found in the pynq.gpio Module sections

2.4 Output Video From HDMI Port

To get the video as an output we plan to display the data in DRAM frame by frame. This process will be done by using Video Module of PYNQ. Until now we see and examined code samples on how to read frames from HDMI IN and output the frame to HDMI OUT. In the examples HDMI IN and OUT instances were created by Video library and these instances were just tied up with a function to pass the frames easily.It has been understand most part of the code sample but we weren't able to find how to do this operation for frames that stored in DRAM. We found DMA library to input/output data from DRAM but we were not able to find how to pass our data in DRAM to HDMI OUT port. We will continue to work and research on this issue to be able to get video output from HDMI OUT.


jupyter hdmi_introduction Last Checkpoint: Last Saturday at 4:19 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown

To start with download the base overlay and instantiate the HDMI input and output.

```
In [1]: from pynq.overlays.base import BaseOverlay
        from pynq.lib.video import *

        base = BaseOverlay("base.bit")
        hdmi_in = base.video.hdmi_in
        hdmi_out = base.video.hdmi_out
```

Getting started

First we'll use the default pixel format which is 24 bit-per-pixel BGR formatted data for ease of use with OpenCV.

```
In [2]: hdmi_in.configure()
        hdmi_out.configure(hdmi_in.mode)

        hdmi_in.start()
        hdmi_out.start()

Out[2]: <contextlib._GeneratorContextManager at 0x2f6139b0>
```

The monitor should turn on and show a blank screen. To pass the image data through we can tie the output to the input. The tie will last until we send something else to be displayed.

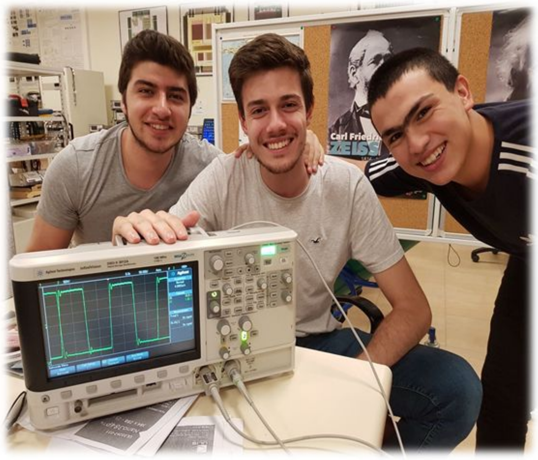
```
In [3]: hdmi_in.tie(hdmi_out)
```

While this provides for a fast way of passing video data through the pipeline there is no way to access or modify the frames. For that we a loop calling `readframe` and `writeframe`.

```
In [4]: import time

        numframes = 600
        start = time.time()

        for _ in range(numframes):
            f = hdmi_in.readframe()
            hdmi_out.writeframe(f)

        end = time.time()
        print("Frames per second:  " + str(numframes / (end - start)))

        Frames per second:  60.08865801337141
```

Conclusion and Future Work

- Power board has been designed.
- Altium Software has been studied
- FPGA programming have been studied in order to generate clock signals, read data from RAM and colormap them.
- By implementing studies and using Video and DMA overlays/libraries we will hopefully obtain infrared video



References

PYNQ Overlays — Python productivity for Zynq (Pynq) v1.0. (n.d.). Retrieved from http://pynq.readthedocs.io/en/v2.1/pynq_overlays.html

Infrared Waves | Science Mission Directorate. (n.d.). Retrieved from https://science.nasa.gov/ems/07_infraredwaves

Retrieved from

http://dergipark.gov.tr/download/article-file/200904