Technologie Internetu Rzeczy

PYNQ

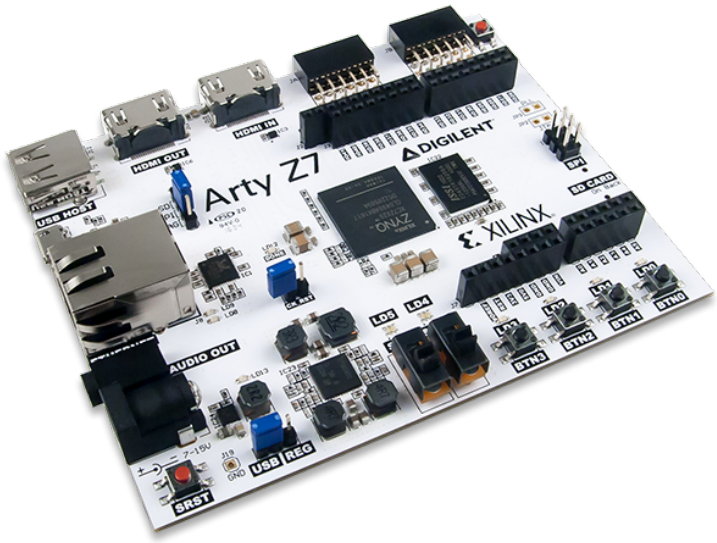
Wykonali: Mikołaj Dłuś, Tomasz Karkocha, Piotr Szczerbiński

Wstęp

Temat zadania

Nasze zadanie polegało na opracowaniu PoC (Proof of Concept) dla platformy PYNQ wykorzystując płytkę ARTY-Z7.

ARTY-Z7



Zdjęcie 1: Wygląda płytki Arty-Z7

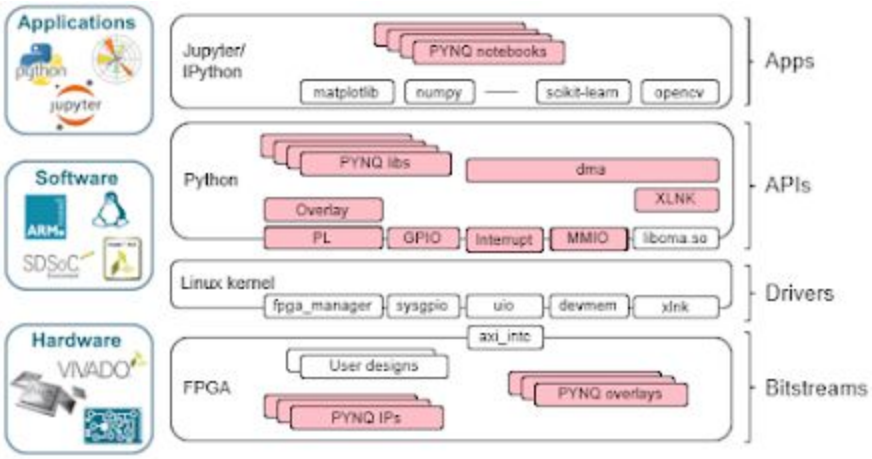
Wykorzystywana płytka ARTY-Z7 posiada następujące specyfikacje:

- 2-rdzeniowy procesor z rdzeniem ARM Cortex-A9
- Wbudowany kontroler pamięci DDR3
- Wbudowane interfejsy: SPI, UART, CAN, I2C
- **Zasoby programowalne odpowiadające układowi FPGA Artix-7**
- 512 MB pamięci DDR3 16-bit, przepustowość 1050 Mb/s
- 16 MB pamięci Flash z interfejsem Quad-SPI, zaprogramowanej fabrycznie
- **Port wejściowy i wyjściowy HDMI**
- Możliwość programowania przez JTAG, QuadSPI lub za pomocą karty microSD
- Porty Arduino/chipKIT Shield

Wymienione wyżej specyfikacje są tymi najważniejszymi (posiada jeszcze parę).

Platforma PYNQ

PYNQ jest open-source projektem stworzonym przez Xilinx® dzięki której można łatwo projektować systemy wbudowane z Xilinx Zynq® Systems on Chips (SoCs).
Wykorzystując język programowania Python możemy tworzyć ładne i szybkie aplikacje.



Rysunek 1: Przedstawienie architektury platformy PYNQ

Wartym nadmienienia plusem platformy jest wbudowany w system Jupyter Notebook dzięki któremu można tworzyć wydajnie programy w Pythonie.

PYNQ-Z1 vs ARTY-Z7

Dokumentacja platformy PYNQ'u opisuje swoje funkcje na podstawie płytki PYNQ-Z1 która jest identyczna co wykorzystywana w tym zadaniu ARTY-Z7 z paroma wyjątkami:

- PYNQ-Z1 posiada wejście mikrofonu.
- ARTY-Z7 posiada przycisk resetu.

Warto nadmienić że oprogramowanie napisane na PYNQ-Z1 będzie działało na ARTY-Z7 i odwrotnie (nie uwzględniając oprogramowania wykorzystującego wejście mikrofonu).

Zadanie

Opis zadania

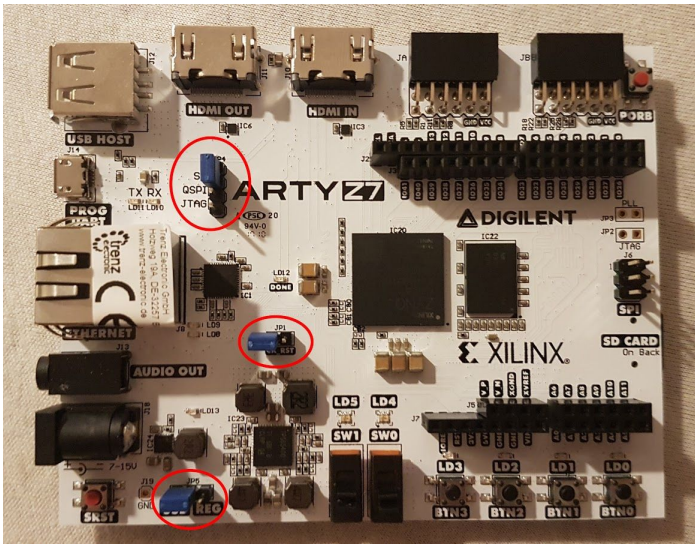
By opracować PoC dla platformy PYNQ postanowiliśmy wykorzystać dwa porty hdmi by móc na bieżąco filtrować obraz przy pomocy funkcji dostarczanych przez bibliotekę OpenCV. Wykorzystując specjalny overlay stworzony przez Xilinx który dostarcza funkcje filtrujące obraz 2D z biblioteki OpenCV tak by układ fpga je wykonywał a nie procesor, chcemy pokazać jaka jest ogromna różnica gdy pewną czynność wykonuje procesor a specjalnie zaprogramowany układ fpga.

Wstępna konfiguracja płytki

By móc zacząć zabawę z naszą płytką, pierw musimy zaopatrzyć się w kartę microSD o **pojemności przynajmniej 8GB** i kabel Ethernet oraz kabel usb-microUSB.

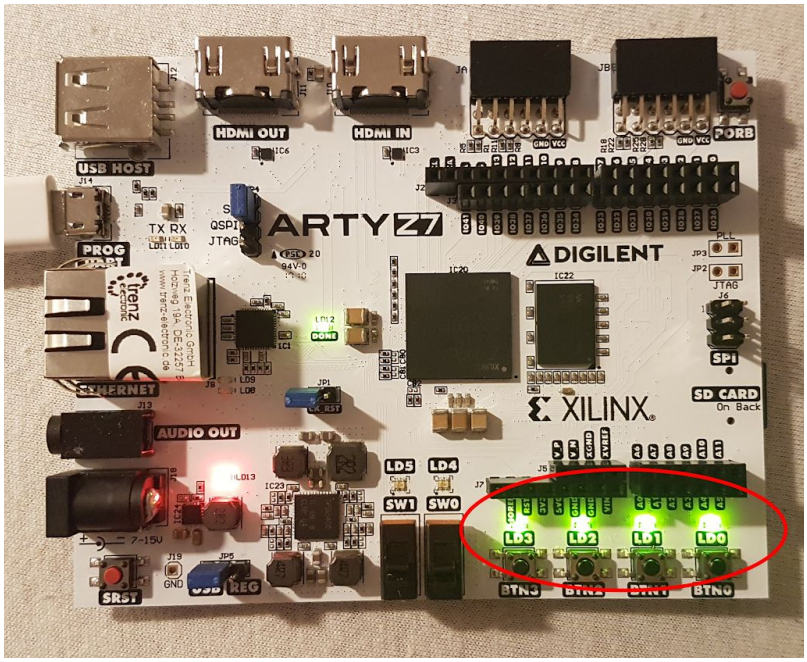
Pierwszym krokiem jest pobranie obrazu PYNQ i jego zamontowanie na naszej karcie SD.

- Obraz można pobrać ze strony http://www.pynq.io/board.html i wybierając opcję **PYNQ-Z1 v<najnowsza wersja> PYNQ image**, następnie zarejestrować się na stronie i pobrać obraz.
- Po pobraniu obrazu montujemy go na naszej karcie wykorzystując np. win32 Disk Imager (https://sourceforge.net/projects/win32diskimager/).
- Po wypaleniu obrazu ISO na karcie należy ją włożyć do płytki w slot dla karty SD.
- Przed podłączeniem płytki do zasilania należy się upewnić że wszystkie trzy zworniki są ustawione tak jak na zdjęciu poniżej



Zdjęcie 2: Prawidłowe ustawienie zworników na płytce

- Górny zwornik służy do ustalenia w jaki sposób płytka ma się bootować, takie ustawienie powoduje bootowanie z karty SD.
- Dolny zwornik służy do ustalenia z jakiego źródła ma płytka odbierać zasilanie, w tym momencie jest ustawiony na zasilanie z wejście micro USB.
- Po podłączeniu zasilania trzeba będzie odczekać parenaście sekund by płytka się zbootowała.
- Gdy diody przy SW0, SW1 zaczną migać na niebiesko i przy BTN0, BTN1, BTN2 oraz BTN3 na zielono i następnie zapalą się na zielono (tak jak na zdjęciu poniżej) to wiemy że płytka jest gotowa do działania.



Zdjęcie 3: Stan płytki mówiący o jej gotowości do działania.

- Teraz wystarczy podłączyć laptopa lub PC'ta kablem ethernet z płytką i możemy już korzystać z płytki.

Łączenie się do płytki

Płytka ma statycznie ustawiony adres IP na **192.168.2.99**.
Do płytki można się połączyć na dwa sposoby:

- Przez ssh (np. programem Putty) podając jako adres: **192.168.2.99:22**
  i jako login podając **xilinx** z hasłem **xilinx**.
- Poprzez wcześnie wspomniany Jupyter Notebook - uruchamiamy dowolną przeglądarkę i wchodzimy na adres **192.168.2.99:9090**, jeśli notebook poprosi nas o login i hasło to wpisujemy: **xilinx** oraz **xilinx**.

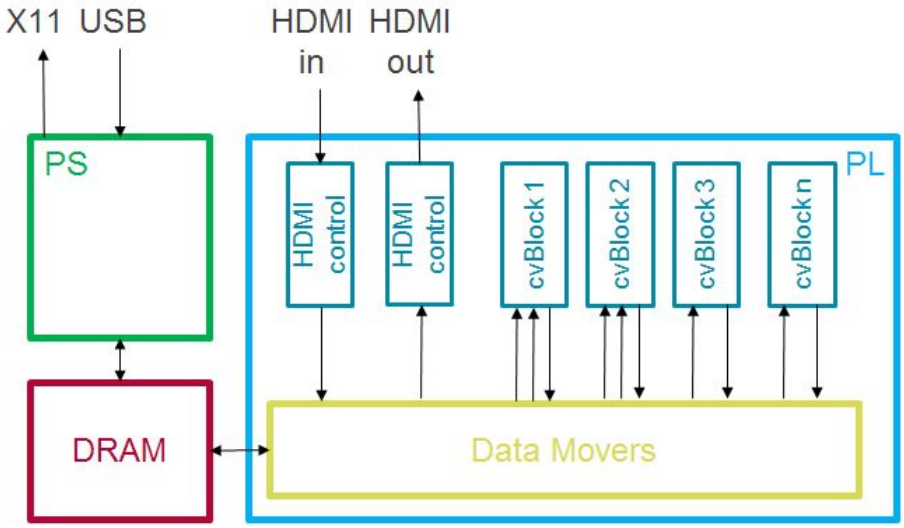
Na płytce powinny już znajdować się różne tutoriale prezentujące podstawowe możliwości urządzenia oraz platformy PYNQ.

Overlay do filtrów

By móc zobaczyć różnicę działania pomiędzy tym jak szybko przetwarza obraz fpga a procesor należy pobrać specjalny overlay (https://github.com/Xilinx/PYNQ-ComputerVision).
Jeśli płytka nie jest podłączona do internetu to wtedy należy przenieść pobrane repozytorium z PC do płytki wykorzystując np. WinSCP i następnie łącząc się przez ssh (Putty) wykonać komendę:

```
$ sudo pip3 install <nazwa katalogu repozytorium>
```

Wtedy biblioteka wraz z przykładami powinna się zainstalować na urządzeniu



Rysunek 2: Schemat używanego overlay.

Patrząc na Rysunek 2 widzimy czemu na tym overlay'u filtrowanie będzie szybsze, mianowicie kontrola portów HDMI jest zaprogramowana w PL (Programmable Logic) czyli FPGA i co najważniejsze FPGA posiada także bloki (cvBlock) które służą do wykonywania funkcji z biblioteki OpenCV.

Zawartość notebook'a

Cały notebook można znaleźć na: https://github.com/Grucha8/TIR-Project

Przed uruchomieniem wszystkiego należy się upewnić że do płytki są podłączone kable hdmi (oba), w przeciwnym wypadku inicjalizacja portów może się nie udać.

Inicjalizacja płytki oraz fpga

Ładowanie overlay'u

Ładujemy specjalny overlay który pozwoli nam używać funkcji openCV na układzie fpga, co przyspieszy ich działanie.

```
# Load filter2D + dilate overlay
from pynq import Overlay
bareHDMI = Overlay("/usr/local/lib/python3.6/dist-packages/"
                   "pynq_cv/overlays/xv2Filter2DDilate.bit")
import pynq_cv.overlays.xv2Filter2DDilate as xv2

# Load xlnk memory mangager
from pynq import Xlnk
Xlnk.set_allocator_library("/usr/local/lib/python3.6/dist-packages/"
                           "pynq_cv/overlays/xv2Filter2DDilate.so")
mem_manager = Xlnk()
```

Konfiguracja hdmi

```
hdmi_in = bareHDMI.video.hdmi_in
hdmi_out = bareHDMI.video.hdmi_out

from pynq.lib.video import *
hdmi_in.configure(PIXEL_GRAY)
hdmi_out.configure(hdmi_in.mode)

hdmi_in.cacheable_frames = False
hdmi_out.cacheable_frames = False

hdmi_in.start()
hdmi_out.start()
```

Konfiguracja parametrów wejścia i wyjścia dla hdmi

```
mymode = hdmi_in.mode
print("My mode: " + str(mymode))

height = hdmi_in.mode.height
width = hdmi_in.mode.width
bpp = hdmi_in.mode.bits_per_pixel
```

Filtrowanie obrazu przy pomocy procesora

Funkcja dzięki której będziemy móc zmieniać dynamicznie filtry obrazu. Do demonstracji działania wykorzystaliśmy 6 filtrów.

```
import numpy as np

def setKernelAndFilter3x3(kernelName):
    global kernel_g

    kernel_g = {
        'Laplacian high-pass': np.array([[0.0,1.0,0.0],[1.0,-4.0,1.0],
                                         [0.0,1.0,0.0]],np.float32),
        'Gaussian high-pass': np.array([[-0.0625,-0.125,-0.0625],
                                        [-0.125,0.75,-0.125],
                                       [-0.0625,-0.125,-0.0625]],np.float32),
        'Average blur':  np.ones((3,3),np.float32)/9.0,
        'Gaussian blur': np.array([[0.0625,0.125,0.0625],
                                   [0.125,0.25,0.125],
                                   [0.0625,0.125,0.0625]],np.float32),
        'Sobel ver': np.array([[1.0,0.0,-1.0],[2.0,0.0,-2.0],
                               [1.0,0.0,-1.0]],np.float32),
        'Sobel hor': np.array([[1.0,2.0,1.0],[0.0,0.0,0.0],
                               [-1.0,-2.0,-1.0]],np.float32)
    }.get(kernelName, np.ones((3,3),np.float32)/9.0)
```

Funkcja dzięki będziemy mogli dynamicznie zmieniać czy filtrowanie ma być przeprowadzane przez procesor czy fpga.

```
def setProcessing(whichProcessing):
    global processing_g

    processing_g = {
        'Fpga processing': 'fpga',
        'Procesor processing': 'procesor'
    }.get(whichProcessing)
```

Główna funkcja

Główna funkcja która jest odpowiedzialna za pobranie obrazu, nałożeniu filtru i wypuszczeniu przetworzonego obrazu na wyjście.

```
import cv2

def imageProccesing(fps):
    outframe = hdmi_out.newframe()   # Pobieramy klatkę strukturę klatki by móc ją później
wykorzystać
    inframe = hdmi_in.readframe()    # Pobranie klatki z wejścia
    # Nałożenie filtru
    if processing_g == 'fpga':
        xv2.filter2D(inframe, -1, kernel_g, dst=outframe, borderType=cv2.BORDER_CONSTANT)
# Filtr przy użyciu fpga
    elif processing_g == 'procesor':
        cv2.filter2D(inframe, -1, kernel_g, dst=outframe,  borderType=cv2.BORDER_CONSTANT)
# Filtr przy użyciu procesora

    font                   = cv2.FONT_HERSHEY_SIMPLEX
    bottomLeftCornerOfText = (1020,1060)
    fontScale              = 1
    fontColor              = (255,255,255)
    lineType               = 2

    cv2.putText(outframe,str("%.2f" % fps),
    bottomLeftCornerOfText,
    font,
    fontScale,
    fontColor,
    lineType)
    hdmi_out.writeframe(outframe)    # Wypuszczenie przetworzonego obrazu
    inframe.freebuffer()             # Wyczyszczenie bufora
```

Klasa która umożliwia nam dynamiczne zmienianie filtra

Sama klasa jest dość prosta gdyż jest to wątek który w pętli wykonuje przetwarzanie obrazu dopóki ktoś jej nie zastopuje metodą `stop()`

```
import threading
import time
import statistics
from IPython.display import display, clear_output


class MainLoop(threading.Thread):
    def __init__(self):
        super(MainLoop, self).__init__()
        self.flag = True

    def run(self):
        num_frames = 60
        frame = 0
        frame_table = [0] * 60
        table_sum = 0
        fps = 0

        while self.flag:
            start = time.time()
            imageProccesing(fps)
            end = time.time()
            elapsed = end - start
            table_sum += elapsed
            table_sum -= frame_table[frame]
            frame_table[frame] = elapsed
            frame = (frame + 1) % 60

            fps = 1 / (table_sum / 60)



    def stop(self):
        self.flag = False
```

W pętli obliczana jest także ilość klatek na sekundę, która jest wyświetlana na ekranie w danej powyżej głównej funkcji.

Rozpoczęcie działania programu

Po rozpoczęciu możemy zmieniać dynamicznie jakim filtrem chcemy zmienić obraz. Do tego służy nam specjalny interaktywny widget ipython.

Czy ma przetwarzać procesor czy fpga

```
from ipywidgets import interact, interactive, fixed, interact_manual
from ipywidgets import IntSlider, FloatSlider
import ipywidgets as widgets

interact(setProcessing, whichProcessing = ['Fpga processing', 'Procesor
processing']);
```

Jaki filtr

```
interact(setKernelAndFilter3x3, kernelName
         = ['Sobel ver','Sobel hor','Laplacian high-pass','Gaussian
high-pass','Average blur', 'Gaussian blur',]);
```

Rozpoczęcie przetwarzania

```
t = MainLoop()
t.start()
```

Zatrzymanie działania programu

```
t.stop()
```

Zamknięcie hdmi

```
hdmi_out.close()
hdmi_in.close()
```

Podsumowanie

Jak widać filtrowanie obrazu przy pomocy fpga jest ponad 10 krotnie szybsze niż używanie procesora. Pokazuje nam to że zaprogramowanie układu fpga dla pewnego danego zadania może okazać się bardzo efektywne dla działania naszego systemu.

Wykorzystane materiały

https://reference.digilentinc.com/reference/programmable-logic/arty-z7/reference-manual
https://pynq.readthedocs.io/en/latest/getting_started/pynq_z1_setup.html
https://pynq.readthedocs.io/en/latest/getting_started.html
https://pynq.readthedocs.io/en/latest/index.html
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html
https://github.com/Xilinx/PYNQ-ComputerVision