

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Informatyki, Elektroniki i Telekomunikacji

RAPORT Z PROJEKTU

Histogram obrazu na platformie Kria KV260 Vision AI Starter Kit

Autor:

Andrzej Zadęcki, Piotr Mosurek

Prowadzący:

Paweł Russek

Data:

17.06.2025

Spis treści

1. Cel projektu	2
2. Opis funkcjonalności finalnej postaci IP	2
3. Porównanie dostępnych wersji IP	2
4. Testowanie IP.....	3
4.1 Symulacja IP w Vitis 2022.1	3
4.2 Tworzenie overlaya	3
4.3 Testy w Jupyter Notebook.....	6
5. Podsumowanie	8
6. Spis rysunków.....	9

1. Cel projektu

Celem projektu było stworzenie i rozwój bloku IP przeznaczonego do wyznaczenia histogramu obrazu przekazywanego za pomocą DMA. Projekt został zaimplementowany i przetestowany na platformie Kria KV260 Vision AI Starter Kit, z wykorzystaniem narzędzi Vitis HLS 2022.1 oraz Vivado 2022.1.

Zaprojektowane IP funkcjonuje w środowisku PYNQ (Jupyter Notebook) poprzez wykorzystanie overlayów wygenerowanych z plików .xsa, które powstały w Vivado.

Wszystkie niezbędne pliki do uruchomienia projektu dostępne są w repozytorium GitHub: <https://github.com/Cirrhush/KriaImageHistogram>

2. Opis funkcjonalności finalnej postaci IP

- Wejście: 32-bitowy sygnał w formacie AXI-Stream.
- Dane wejściowe wysyłane są ze wsparciem sygnałów kontrolnych (tvalid, tlast) pozwalając na prawidłowe synchronizowanie i zakończenie transferu danych.
- Wyjście: Histogram 256-binowy.
- Sterowanie: Parametry obrazu: Width, Height.

3. Porównanie dostępnych wersji IP

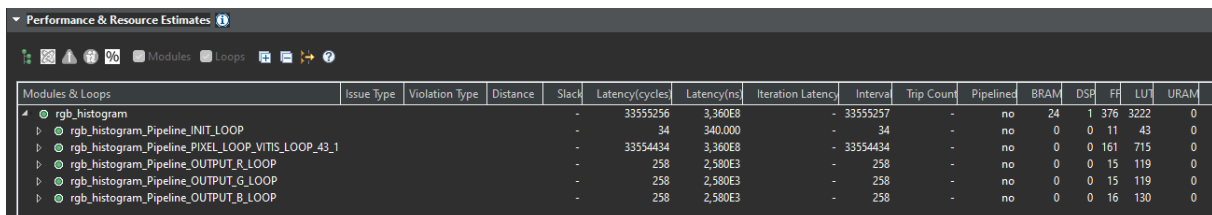
Nazwa wersji	Dane wejściowe	Dane wyjściowe	Obsługa RGB	Rozdzielczość obrazu	Przetwarzanie	Przykładowy czas przetwarzania obrazu
histzinho	AXI-Stream 8-bit	AXI-Stream 32-bit	Nie	128x128	Obrazy greyscale, jeden kanał	-
hist1024	AXI-Stream 8-bit	AXI-Stream 32-bit	Tak	1024x1024	Sekwencyjnie, kanały RGB	-
varHist	AXI-Stream 8-bit	AXI-Stream 32-bit	Tak	Dynamicznie przekazywana	Sekwencyjnie, kanały RGB	17,97ms (obraz 500x500)
rvarHist	AXI-Stream 32-bit (8 bitów nie używane)	AXI-Stream 32-bit	Tak	Dynamicznie przekazywana	Równolegle, kanały RGB	5,53ms (obraz 500x500)

Wersja „rvarHist” stanowi finalną wersję bloku IP. Wykonuje histogram równolegle dla każdego kanału RGB, co w porównaniu z wersją „varHist” pozwala osiągnąć nawet ~3,25 razy szybszy czas wykonania histogramu.

4. Testowanie IP

4.1 Symulacja IP w Vitis 2022.1

Przeprowadzono testy dla obrazu 2000x1500 z losową wartością pixeli. IP przeszło symulację, syntezę oraz eksport. Wyeksportowany IP został dodany do projektu w Vivado 2022.1.



Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
rgb_histogram				-	33555256	3.360E8	-	33555257	-	no	24	1	376	3222	0
rgb_histogram_Pipeline_INIT_LOOP				-	34	340.000	-	34	-	no	0	0	11	43	0
rgb_histogram_Pipeline_PIXEL_LOOP_VITIS_LOOP_43_1				-	33554434	3.360E8	-	33554434	-	no	0	0	161	715	0
rgb_histogram_Pipeline_OUTPUT_R_LOOP				-	258	2.580E3	-	258	-	no	0	0	15	119	0
rgb_histogram_Pipeline_OUTPUT_G_LOOP				-	258	2.580E3	-	258	-	no	0	0	15	119	0
rgb_histogram_Pipeline_OUTPUT_B_LOOP				-	258	2.580E3	-	258	-	no	0	0	16	130	0

Rysunek 1 - Wynik syntezy w Vitis

4.2 Tworzenie overlaya

Podczas tworzenia projektu w zakładce *Default Part* wybrana została płytką **Kria KV260 Vision AI Starter Kit**. Następnie do projektu zostało dołączone IP wygenerowane w Vitis HLS.

Block diagram składa się z:

- Zynq UltraScale+ MPSoC **zynq_ultra_ps_e_0**
- rgb_histogram_v1.0 **rgb_histogram_0**
- AXI Direct Memory Access **axi_dma_0**
- 2 x AXI Interconnect **axi_interconnect_0** **axi_interconnect_1**
- Processor System Reset **rst_ps8_0_99M**

Poszczególne elementy block diagramu zostały skonfigurowane następująco:

- **zynq_ultra_ps_e_0:**
 - Master Interface – AXI HPM0 FPD AXI HPM1 FPD
 - Slave Interface – AXI HP0 FPD
- **axi_dma_0:**
 - Enable Scatter Gather Engine – Wyłączone
 - Width of Buffer Length Register – 26
 - Read Channel – Allow Unaligned Transfers
 - Write Channel – Allow Unaligned Transfers
- **axi_interconnect_0:**
 - Number of Slave Interfaces – 2
 - Number of Master Interfaces – 2
- **axi_interconnect_1:**
 - Number of Slave Interfaces – 2
 - Number of Master Interfaces – 1

Component Name

☐ Enable Asynchronous Clocks (Auto)

☐ Enable Scatter Gather Engine

☐ Enable Micro DMA

☐ Enable Multi Channel Support

☐ Enable Control / Status Stream

Width of Buffer Length Register (8-26) bits

Address Width (32-64) bits

☒ Enable Read Channel

Number of Channels
 Memory Map Data Width
 Stream Data Width
 Max Burst Size

☒ Allow Unaligned Transfers

☒ Enable Write Channel

Number of Channels
 Memory Map Data Width
 Stream Data Width
 Max Burst Size

☒ Allow Unaligned Transfers

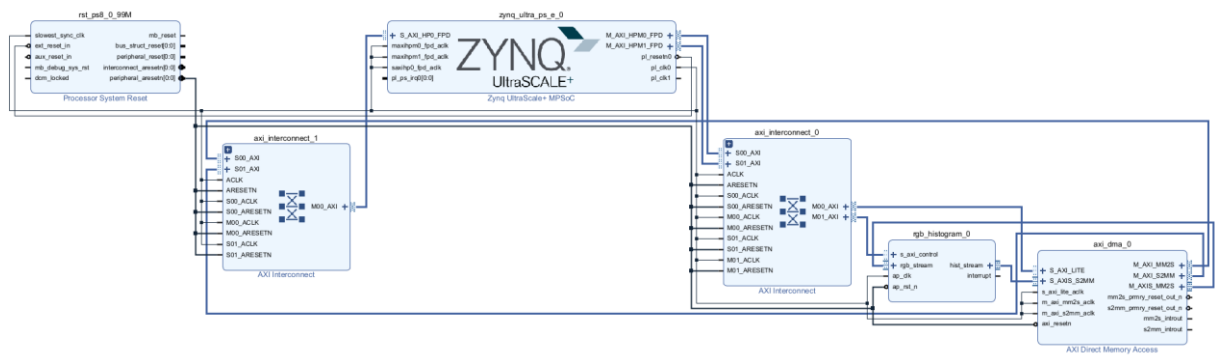
Rysunek 2 - Konfiguracja AXI DMA

Po skonfigurowaniu wykonano następujące połączenia:

- **zynq_ultra_ps_e_0 M_AXI_HPM0_FPD - S00_AXI axi_interconnect_0**
- **zynq_ultra_ps_e_0 M_AXI_HPM1_FPD - S01_AXI axi_interconnect_0**
- **zynq_ultra_ps_e_0 S_AXI_HP0_FPD - M00_AXI axi_interconnect_1**
- **axi_interconnect_0 M00_AXI - S_AXI_LITE axi_dma_0**
- **axi_interconnect_0 M01_AXI - s_axi_control rgb_histogram_0**
- **axi_interconnect_1 S00_AXI - M_AXI_MM2S axi_dma_0**
- **axi_interconnect_1 S01_AXI - M_AXI_S2MM axi_dma_0**
- **rgb_histogram_0 hist_stream - S_AXIS_S2MM axi_dma_0**
- **rgb_histogram_0 rgb_stream - M_AXIS_MM2S axi_dma_0**

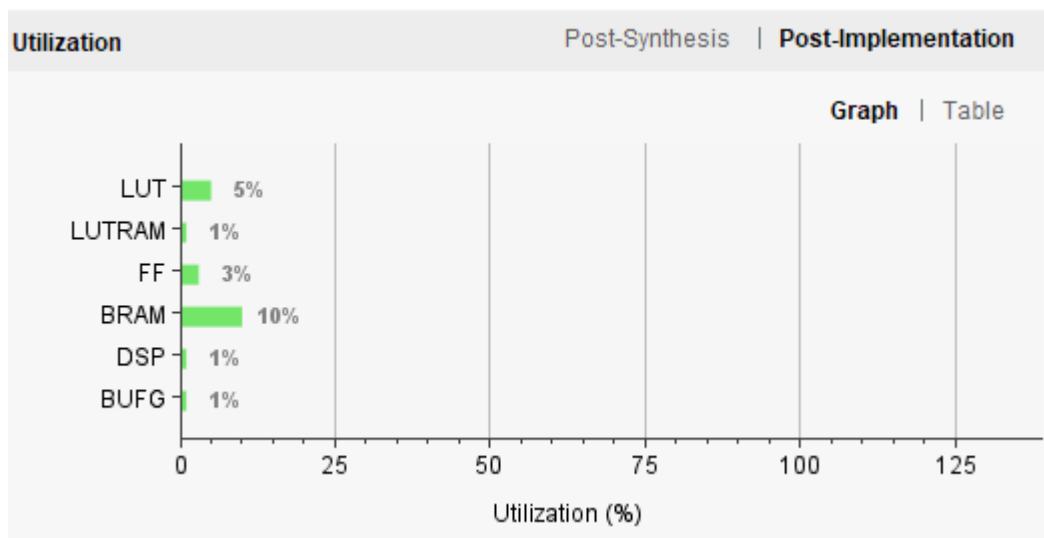
Następnie wybrano Run Connection Automation – All Automation.

Ostatnim etapem było stworzenie wrappera HDL, wygenerowanie bitstreamu i wyeksportowanie hardwareu.



Rysunek 3 - Block diagram

Otrzymano następujące zużycie zasobów:



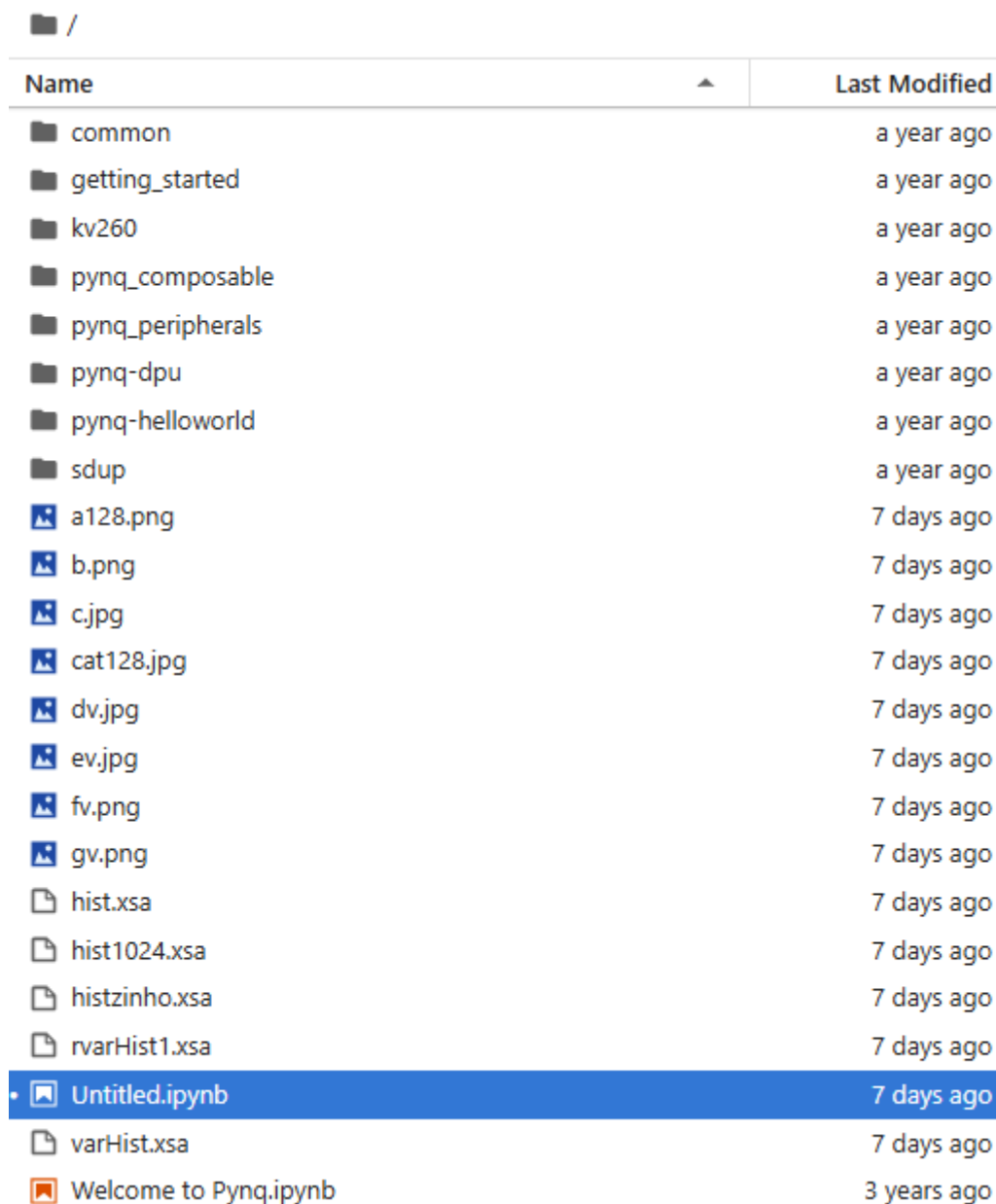
Rysunek 4 - Zużycie zasobów graf

Utilization				Post-Synthesis	Post-Implementation
				Graph Table	
Resource	Utilization	Available	Utilization %		
LUT	5881	117120	5.02		
LUTRAM	358	57600	0.62		
FF	6497	234240	2.77		
BRAM	14	144	9.72		
DSP	1	1248	0.08		
BUFG	2	352	0.57		

Rysunek 5 - Zużycie zasobów tabela

4.3 Testy w Jupyter Notebook

Przygotowany Jupyter Notebook posiada osobne komórki przygotowane dla każdego dostępnego overlaya.



Name	Last Modified
common	a year ago
getting_started	a year ago
kv260	a year ago
pynq_composable	a year ago
pynq_peripherals	a year ago
pynq-dpu	a year ago
pynq-helloworld	a year ago
sdup	a year ago
a128.png	7 days ago
b.png	7 days ago
c.jpg	7 days ago
cat128.jpg	7 days ago
dv.jpg	7 days ago
ev.jpg	7 days ago
fv.png	7 days ago
gv.png	7 days ago
hist.xsa	7 days ago
hist1024.xsa	7 days ago
histzinho.xsa	7 days ago
rvarHist1.xsa	7 days ago
Untitled.ipynb	7 days ago
varHist.xsa	7 days ago
Welcome to Pynq.ipynb	3 years ago

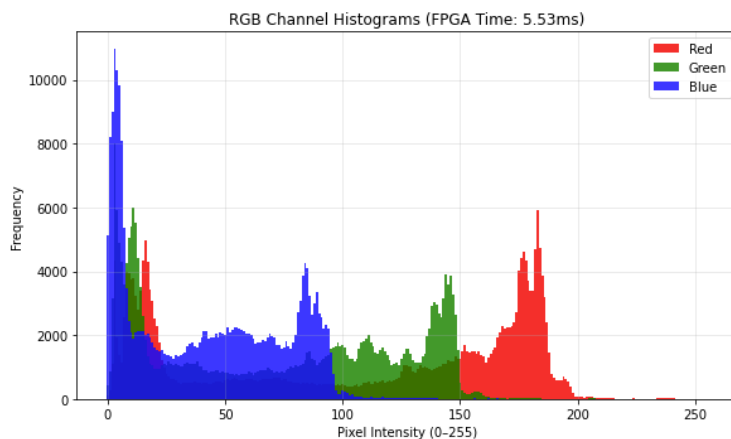
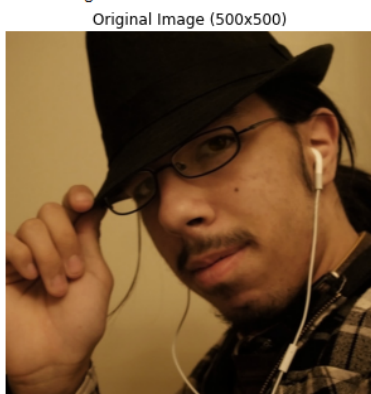
Rysunek 6 - Folder Jupyter Notebook

```
#Variable Histogram RGB in HW
from pynq import Overlay
overlay = Overlay("rvarHist1.xsa")
overlay?
```

Rysunek 7 - Przykładowa komórka ładująca overlay

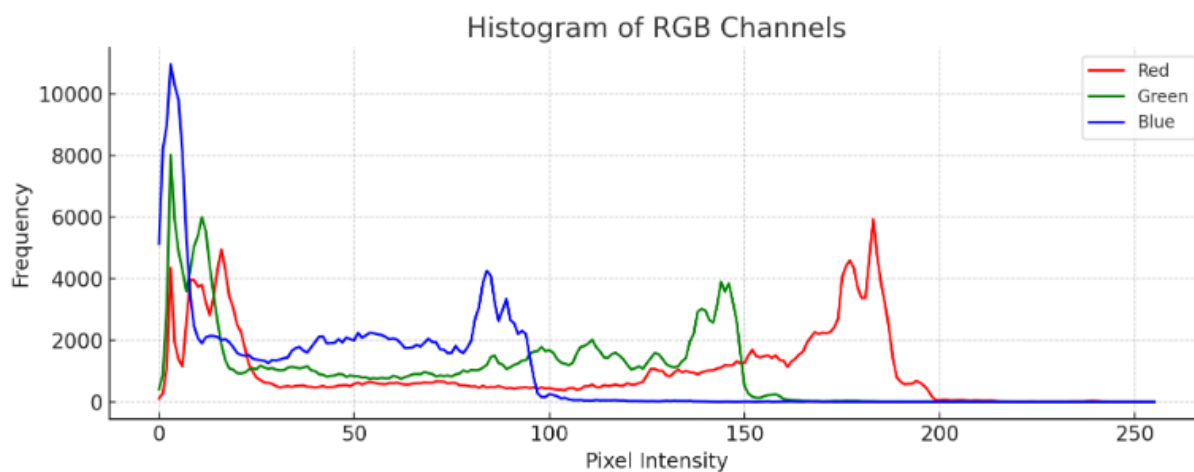
Testy zostały przeprowadzone dla różnych obrazów pod względem rozmiaru oraz zawartości kanałów. Zgodność zwracanych wartości została porównana z rozwiązaniem softwareowym.

```
Starting image processing...
Loading FPGA overlay...
Loading image...
Computing histograms...
Generating visualization...
```



```
Histogram Statistics:
Red Channel:
  Max bin: 183 (count=5931)
  Min bin: 244 (count=0)
  Total pixels: 250000
Green Channel:
  Max bin: 3 (count=8025)
  Min bin: 219 (count=0)
  Total pixels: 250000
Blue Channel:
  Max bin: 3 (count=10963)
  Min bin: 177 (count=0)
  Total pixels: 250000
Resources released
Processing complete
```

Rysunek 8 - Output dla przykładowego zdjęcia



Rysunek 9 - Wynik softwareowy histogramu

5. Podsumowanie

Udało się opracować i przetestować blok IP do wyznaczania histogramu obrazu, działający na platformie Kria KV260 Vision AI Starter Kit. Finalna wersja IP „rvarHist”, została zoptymalizowana pod względem czasu wykonywania dzięki równoległemu przetwarzaniu kanałów RGB i korzystaniu z 32-bitowego wejścia danych. Ostateczna wersja IP została zintegrowana w środowisku PYNQ, umożliwiając wykonywanie histogramów z wykorzystaniem Jupyter Notebooka.

Projekt wykazuje poprawność zarówno na poziomie symulacji, jak i testach sprzętowych.

Potencjalne kierunki dalszego rozwoju obejmują:

- Przetwarzanie histogramu w przestrzeni barw innej niż RGB (YUV, LAB, HSV).
- Wykonywanie histogramu dla obrazu wideo dostarczanego w czasie rzeczywistym z kamery.

6. Spis rysunków

Rysunek 1 - Wynik syntezy w Vitis	3
Rysunek 2 - Konfiguracja AXI DMA	4
Rysunek 3 - Block diagram	5
Rysunek 4 - Zużycie zasobów graf.....	5
Rysunek 5 - Zużycie zasobów tabela	5
Rysunek 6 - Folder Jupyter Notebook.....	6
Rysunek 7 - Przykładowa komórka ładująca overlay	6
Rysunek 8 - Output dla przykładowego zdjęcia	7
Rysunek 9 - Wynik softwareowy histogramu.....	7