Rozpoczęcie pracy w środowisku Vitis AI

1. Opis środowiska

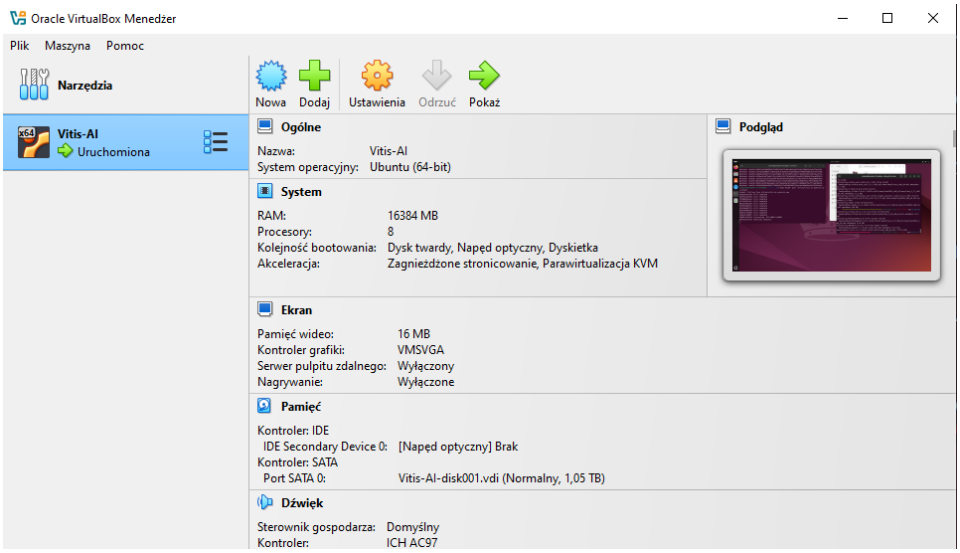
Vitis AI to zintegrowane środowisko programistyczne, które służy do optymalizacji i wdrażania sztucznej inteligencji na adaptowalnych platformach sprzętowych AMD. Został stworzony z myślą o umożliwieniu realizacji aplikacji AI, takich jak rozpoznawanie obrazów, przetwarzanie danych w czasie rzeczywistym czy analiza wideo, przy użyciu programowalnych układów FPGA oraz zintegrowanych systemów. Vitis AI oferuje zoptymalizowane IP, narzędzia, biblioteki, modele oraz zasoby, takie jak przykładowe projekty i samouczki, które wspierają użytkownika w całym procesie tworzenia.

2. ResNet18

W trakcie zajęć będziemy korzystać z sieci **ResNet18** (Residual Network 18) jest to popularna sieć neuronowa używana głównie do zadań związanych z rozpoznawaniem obrazów, takich jak klasyfikacja obiektów czy wykrywanie obiektów. ResNet18 ma 18 warstw i jest wykorzystywana w wielu zadaniach związanych z analizą obrazów, oferując dobrą równowagę między dokładnością a złożonością. Jest jedną z najprostszych dostępnych sieci.

3. Klonowanie maszyny wirtualnej

Vitis AI działa w środowisku Linux w związku z czym jesteśmy zmuszeni do korzystania z maszyny wirtualnej. Uruchom program **Oracle VirtualBox** i sklonuj istniejącą już maszynę z Ubuntu. Nazwij ją **Vitis-Ai**.



Login: student

Hasło: student

UWAGA Podczas wykonywania niektórych poleceń może wyskoczyć komunikat o braku danego pakietu, należy wtedy doinstalować wymagany program sugerowaną przez Ubuntu komendą. Pamiętaj o używaniu **sudo** jeżeli wykonywana komenda zwróci błąd o braku uprawnień

4. Przygotowanie środowiska

Otwórz terminal i sklonuj repozytorium do folderu **home/student/**. Prawdopodobnie będzie trzeba doinstalować **gita**.

```
[Host] $ git clone https://github.com/Xilinx/Vitis-AI
```

Po pomyślnym pobraniu repozytorium wykonaj w nim komendę:

```
[Host] $ cd Vitis-AI
[Host] $ git checkout 3.0
```

W procesie kompilacji naszego modelu będziemy korzystać z **Dockera**. Vitis-Ai daje możliwość budowania kontenerów w różnych architekturach i frameworkach. Do kwantyzacji zaleca się stosowanie Dockera z GPU, ale komputery w laboratorium nie posiadają dedykowanych kart graficznych. W związku z tym jesteśmy zmuszeni do korzystania z Dockera typu CPU z frameworkiem PyTorch.

Vitis AI Docker Container Build Options

DOCKER_TYPE (-t)	TARGET_FRAMEWORK (-f)	Desired Environment
cpu	pytorch	PyTorch cpu-only
	tf2	TensorFlow 2 cpu-only
	tf1	TensorFlow 1.15 cpu-only
gpu	pytorch	PyTorch CUDA-gpu
	opt_pytorch	PyTorch with AI Optimizer CUDA-gpu
	tf2	TensorFlow 2 CUDA-gpu
	opt_tf2	TensorFlow 2 with AI Optimizer CUDA-gpu
	tf1	TensorFlow 1.15 CUDA-gpu
	opt_tf1	TensorFlow 1.15 with AI Optimizer CUDA-gpu
rocm	pytorch	PyTorch ROCm-gpu
	opt_pytorch	PyTorch with AI Optimizer ROCm-gpu
	tf2	TensorFlow 2 ROCm-gpu
	opt_tf2	TensorFlow 2 with AI Optimizer ROCm-gpu

Zainstaluj **Dockera**

```
[Host] $ sudo snap install docker
```

Zweryfikuj instalację poniższymi komendami:

```
[Host] $ docker --version
[Host] $ docker run hello-world
```

Pobierz i uruchom najnowszą wersję Vitis AI Docker za pomocą następujących poleceń:

```
[Host] $ docker pull xilinx/vitis-ai-pytorch-cpu:latest
```

5. Uruchomienie Vitis-Ai

Przejdź do folderu z repozytorium:

```
[Host] $ cd home/student/Vitis-AI
```

Ustaw środowisko za pomocą komendy:

```
[Host] $ cd Vitis-AI/board_setup/mpsoc
[Host] $ sudo chmod u+r+x host_cross_compiler_setup.sh
[Host] $ ./host_cross_compiler_setup.sh
```

Po instalacji wykonaj:

```
[Host] $ source ~/petalinux_sdk_2022.2/environment-setup-
cortexa72-cortexa53-xilinx-linux
```

Skrypt ten zawiera istotne informacje, takie jak ścieżki do kompilatorów, bibliotek oraz innych narzędzi używanych w procesie budowy.

Uruchom dockera za pomocą komendy

```
[Host] $ cd ../..
[Host] $ ./docker_run.sh xilinx/vitis-ai-pytorch-cpu:latest
```

Aktywuj środowisko **conda** dla **vitis-ai-pytorch**

```
[Docker] $ conda activate vitis-ai-pytorch
```

6. Kwantyzacja Modelu

Kwantyzacja zmniejsza precyzję wag i aktywacji sieci, aby zoptymalizować wykorzystanie pamięci i wydajność obliczeniową przy zachowaniu akceptowalnego poziomu dokładności. Operacja ta jest kosztowna obliczeniowo i wymaga dużej przepustowości pamięci, aby

spełnić wymagania aplikacji w zakresie niskich opóźnień i wysokiej przepustowości. Techniki kwantyzacji i pruningu są stosowane w celu rozwiązania tych kwestii przy jednoczesnym osiągnięciu wysokiej wydajności i wysokiej efektywności energetycznej przy niewielkim pogorszeniu dokładności. Vitis AI Quantizer przyjmuje model zmiennoprzecinkowy jako dane wejściowe i wykonuje wstępne przetwarzanie, a na koniec kwantyzuje wagi / odchylenia i aktywacje do danej szerokości bitowej.

```
[Docker] $ mkdir -p resnet18/model
```

Pobierz zestaw danych ImageNet 1000 (mini) z serwisu Kaggle (**UWAGA** Aby pobrać plik należy się zarejestrować). Ta paczka jest podzbiorem zestawu danych ILSVRC 2012-2017 i obejmuje 1000 klas obiektów oraz zawiera 1 281 167 obrazów treningowych, 50 000 obrazów walidacyjnych i 100 000 obrazów testowych. Aby uzyskać dostęp do tego zestawu danych, należy utworzyć konto Kaggle. Przenieś pobrany plik **archive.zip** do utworzonego folderu **/Vitis-AI/resnet18** i rozpakuj go przy użyciu komendy **unzip**. Po tej operacji folder **resnet18** powinien wyglądać następująco:

```
├── archive.zip
│
├── model
│
└── imagenet-mini
    ├── train              # Training images folder. Will not be used in this tutorial.
    │   └── n01440764      # Class folders to group images.
    └── val                # Validation images that will be used for quantization and evaluation of the floating point model.
        └── n01440764
```

Pobierz wstępnie wytrenowany model resnet18 z PyTorch do środowiska docker i zapisz go w folderze **model** . Jest to model zmiennoprzecinkowy (**FP32**), który zostanie skwantyfikowany do precyzji **INT8** w celu wdrożenia na urządzeniu docelowym.

```
[Docker] $ cd resnet18/model
[Docker] $ wget https://download.pytorch.org/models/resnet18-
5c106cde.pth -O resnet18.pt
```

Skopiuj przykładowy skrypt kwantyzacji Vitis AI ResNet18 do obszaru roboczego. Skrypt ten zawiera wywołania Quantizer API, które zostaną wykonane w celu kwantyzacji modelu.

```
[Docker] $ cd ..
[Docker] $ cp
../src/vai_quantizer/vai_q_pytorch/example/resnet18_quant.py ./
```

Struktura folderu **resnet18** powinna wyglądać następująco

```
├── archive.zip
│
├── model
│   └── resnet18.pth          # ResNet18 floating point model downloaded from PyTorch.
│
├── imagenet-mini
│   ├── train                 # Training images folder. Will not be used in this tutorial.
│   │   └── n01440764         # Class folders to group images.
│   └── val                   # Validation images that will be used for quantization and evaluation of the floating point model.
│       └── n01440764
│
└── resnet18_quant.py         # Quantization python script.
```

Uruchom poniższe polecenie, aby ocenić dokładność modelu zmiennoprzecinkowego.

```
[Docker] $ python resnet18_quant.py --quant_mode float --data_dir
imagenet-mini --model_dir model
```

Należy zauważyć, że zgłoszona dokładność jest podobna do dokładności **top-1 / top-5: 69,9975 / 88,7586**

Następnie uruchommy Inspektora modeli, aby potwierdzić, że ten model powinien być kompatybilny z docelową architekturą DPU.

```
[Docker] $ python resnet18_quant.py --quant_mode float --inspect -
-target DPUCZDX8G_ISA1_B4096 --model_dir model
```

Uruchom poniższe polecenie, aby rozpocząć kwantyzację. Ogólnie rzecz biorąc, do kwantyzacji wymagane jest 100-1000 obrazów, a liczbę iteracji można kontrolować za pomocą argumentu ładowania danych **subset_len**. W tym przypadku 200 obrazów jest przekazywanych przez sieć, a obrazy te są wybierane losowo z zestawu obrazów walidacyjnych. Należy pamiętać, że wyświetlane straty i dokładność, które są wyprowadzane z tego procesu, nie są reprezentatywne dla ostatecznej dokładności modelu.

```
[Docker] $ python resnet18_quant.py --quant_mode calib --data_dir
imagenet-mini --model_dir model --subset_len 200
```

Na większości maszyn hosta polecenie to powinno zakończyć się w czasie krótszym niż 1 minuta, nawet w przypadku Dockera wykorzystującego tylko procesor CPU. Jeśli wykorzystasz Dockery CUDA lub ROCm na kompatybilnej maszynie, proces kwantyzacji zostanie znacznie przyspieszony. Przyjrzyjmy się wynikom:

```
[Docker] $ cd quantize_result
[Docker] $ ls
```

Jeśli polecenie zostanie uruchomione pomyślnie, wygenerowany zostanie katalog wyjściowy **quantize_result**, zawierający dwa ważne pliki:
-**ResNet.py** - Skwantyzowany model w formacie vai_q_pytorch.
-**Quant_info.json** - Kroki kwantyzacji tensorów. Zachowaj ten plik do oceny skwantyzowanego modelu

Aby ocenić dokładność skwantyzowanego modelu, wróć do katalogu **/resnet18** i uruchom następujące polecenia. Należy pamiętać, że na komputerach z procesorem tylko CPU wykonanie tego polecenia zajmie trochę czasu (~20 minut). Jeśli się spieszysz, możesz pominąć ten krok i przejść do następnego.

```
[Docker] $ cd ..
[Docker] $ python resnet18_quant.py --model_dir model --data_dir
imagenet-mini --quant_mode test
```

Należy zauważyć, że zgłaszana dokładność będzie podobna do dokładności **top-1 / top-5: 69,1308 / 88,7076**. Utrata dokładności spowodowana kwantyzacją wynosi mniej niż **1%**.

Aby wygenerować skwantyzowany plik **.xmodel**, który zostanie następnie skompilowany dla DPU, uruchom następujące polecenie z argumentami **batch_size** i **subset_len** ustawionymi na 1. W przypadku eksportu modelu oba te parametry powinny być ustawione na 1, ponieważ wielokrotne iteracje nie są wymagane.

```
[Docker] $ python resnet18_quant.py --quant_mode test --subset_len
1 --batch_size=1 --model_dir model --data_dir imagenet-mini --
deploy
```

Wynikowy model **resnet18_pt.xmodel** można teraz znaleźć w folderze **resnet18/resnet18_pt**.

7. Kompilacja modelu

Kompilator Vitis AI kompiluje operatory grafów jako zestaw mikrokodowanych instrukcji, które są wykonywane przez DPU. W tym kroku skompilujemy model ResNet18, który skwantyfikowaliśmy w poprzednim kroku.

Kompilator pobiera kwantyzowany **INT8.xmodel** i generuje wdrażalny **DPU.xmodel**, uruchamiając poniższe polecenie. Należy pamiętać, że należy zmodyfikować polecenie, aby określić odpowiedni plik **arch.json** dla celu. W przypadku celów MPSoC znajdują się one w folderze **/opt/vitis_ai/compiler/arch/DPUCZDX8G** wewnątrz kontenera Docker.

```
[Docker] $ cd /workspace/resnet18
[Docker] $ vai_c_xir -x quantize_result/ResNet_int.xmodel -a
/opt/vitis_ai/compiler/arch/DPUCZDX8G/KV260/arch.json -o
resnet18_pt -n resnet18_pt
```

Jeśli kompilacja się powiedzie, plik **resnet18_pt.xmodel** powinien zostać wygenerowany zgodnie z określoną architekturą DPU.

Utwórz nowy plik w folderze **resnet18_pt** w wybranym edytorze tekstu i nadaj mu nazwę **resnet18_pt.prototxt**. Skopiuj i wklej następujące linie kodu:

```
model {
  name : "resnet18_pt"
  kernel {
       name: "resnet18_pt_0"
       mean: 103.53
       mean: 116.28
       mean: 123.675
       scale: 0.017429
       scale: 0.017507
       scale: 0.01712475
  }
  model_type : CLASSIFICATION
  classification_param {
       top_k : 5
       test_accuracy : false
       preprocess_type : VGG_PREPROCESS
  }
}
```

Plik **.prototxt** jest plikiem konfiguracyjnym Vitis™ AI, który ułatwia jednolite zarządzanie konfiguracją parametrów modelu.

Możemy teraz wdrożyć skwantyzowany i skompilowany model na obiekcie docelowym.

8. Uruchomienie Zynq MPSoC

KROK 1: Podłącz kable do urządzenia KRIA

A. Włóż kartę microSD do gniazda J11

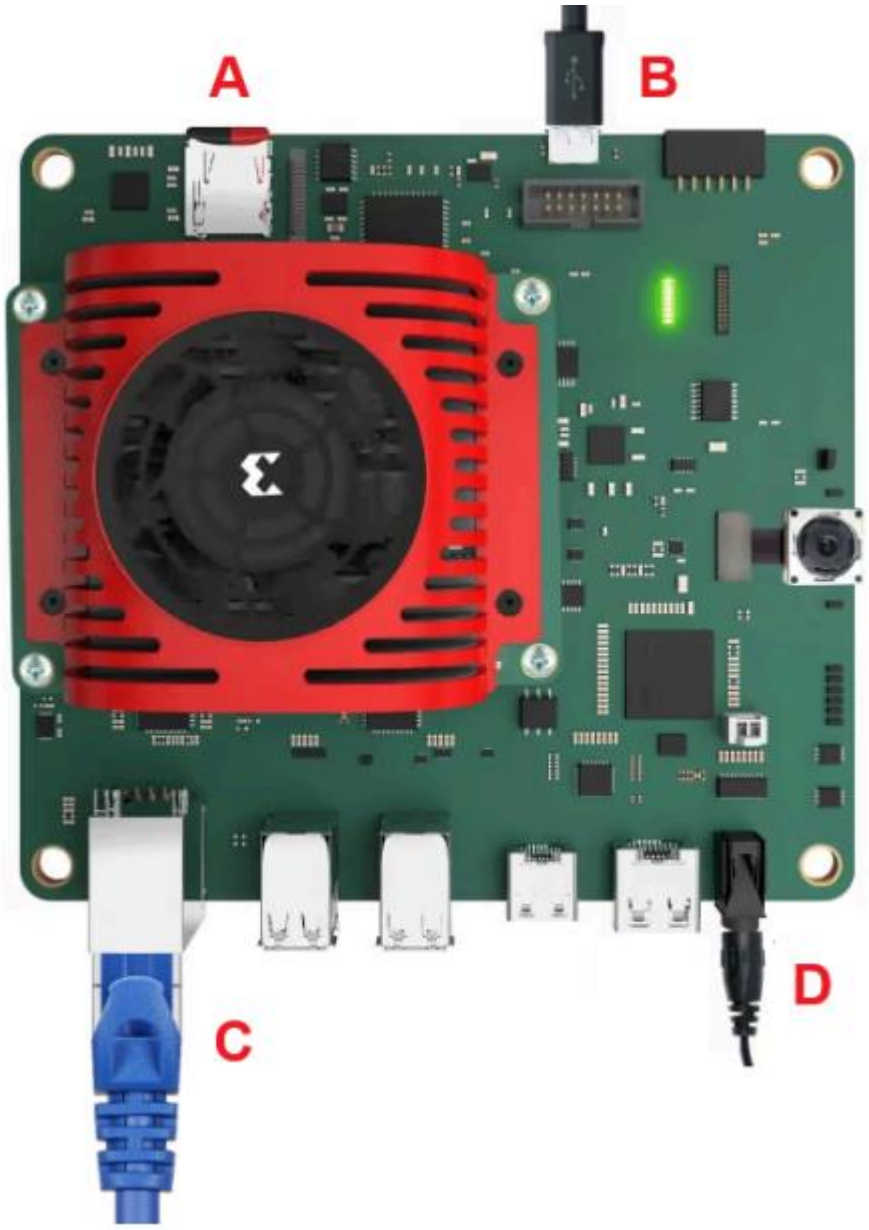
B. Podłącz kabel micro-USB do złącza J4

C. Podłącz kabel RJ45

D. Podłącz zasilanie do J12

OSTRZEŻENIE: Po podłączeniu zasilania wentylator znajdujący się na płycie KRIA uruchomi się z pełną prędkością. Wentylator zwolni

zwolni po poprawnym uruchomieniu systemu operacyjnego Ubuntu, po około 3 minutach.



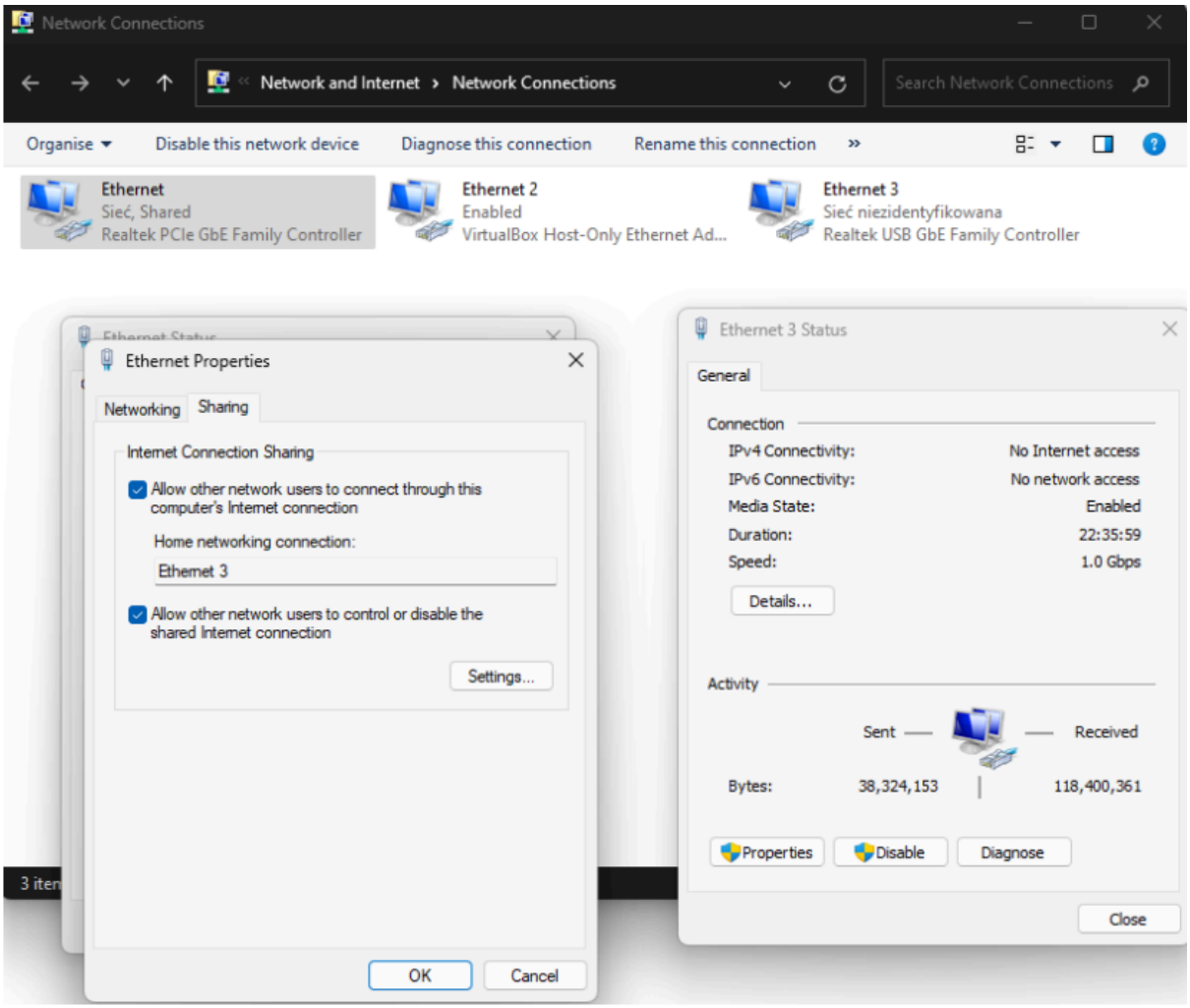
KROK 2: Podłączenie do sieci

Aby uzyskać dostęp do Krii z poziomu PCta musi być ona podłączona do internetu. Dostępne są dwie opcje:

Opcja 1: Podłącz KRIA za pomocą RJ45 bezpośrednio do komputera (wbudowana karta sieciowa lub klucz sprzętowy USB<->Ethernet dołączony do zestawu laboratoryjnego)

Opcja 2: Podłącz KRIA bezpośrednio do routera
W komputerze musi być dostępny port Ethernet, a użytkownik musi mieć uprawnienia do konfigurowania interfejsu sieciowego.

UWAGA: W przypadku **1 opcji** upewnij się, że port Ethernet do którego podłączona jest FPGA odbiera pakiety. W poniższym przykładzie Ethernet to połączenie komputera z siecią LAN, a Ethernet 3 to połączenie z KRIA. Wejdź w **View Network Connections**, aby ustawić **Internet Connetion Sharing**. W razie dalszych problemów najlepszym rozwiązaniem jest wyłączenie i włączenie (disable and enable) udostępniania sieci w zakładce Właściwości sieci Ethernet (Properties tab).



KROK 3: Otwórz terminal szeregowy USB
Za pomocą terminala można sprawdzić połączenie sieciowe płyty. **PuTTY** jest jedną z aplikacji, której można użyć. Aby otworzyć terminal, należy znać **port COM** urządzenia (Device Manager). Użyj prędkości **115200 baudrate**. Aby się zalogować, użyj danych:

username: root

password: root

Możesz sprawdzić **HOSTNAME** i **adres IP** płyty za pomocą **ifconfig**

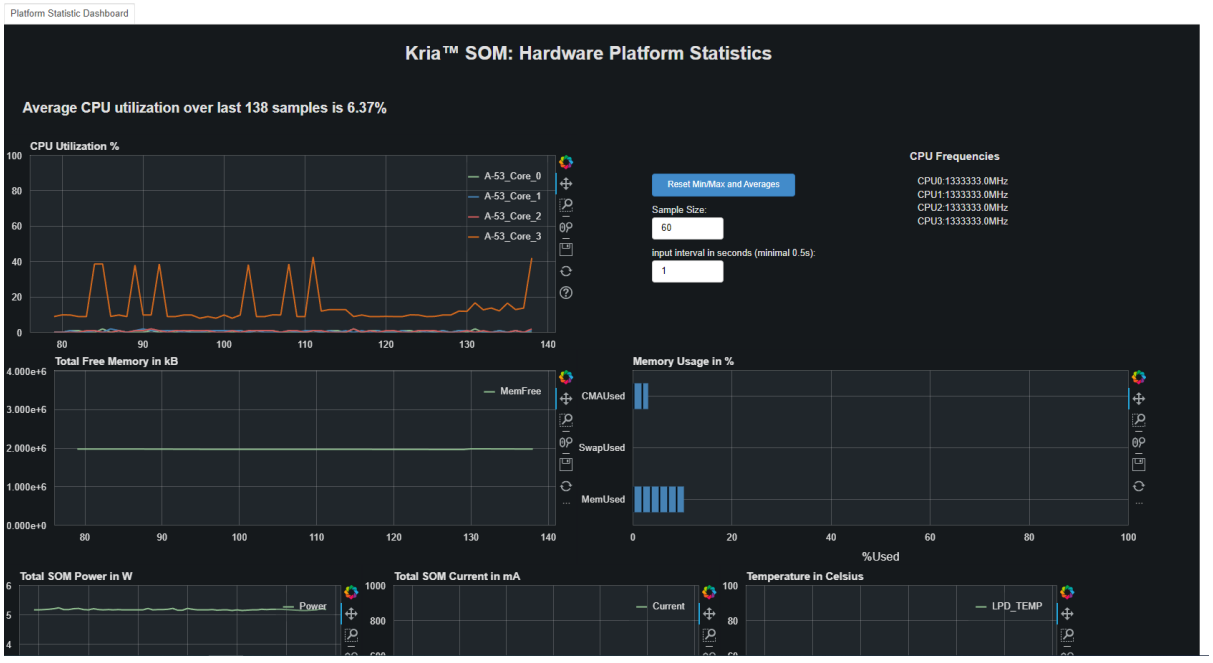
```
[Target] $ ifconfig
```

```
root@xilinx-k26-kv:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.13  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::20a:35ff:fe15:332f  prefixlen 64  scopeid 0x20<link>
        ether 00:0a:35:15:33:2f  txqueuelen 1000  (Ethernet)
        RX packets 417803  bytes 543828278 (518.6 MiB)
        RX errors 0  dropped 7592  overruns 0  frame 0
        TX packets 205059  bytes 24393034 (23.2 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 38
```

Znając IP urządzenia możesz połączyć się teraz przez **SSH:**

```
[Host] $ ssh -X root@[TARGET_IP_ADDRESS]
```

Dodatkowo możesz sprawdzić panel główny Krii z informacjami o aktualnym zużyciu zasobów na stronie: **http://<ip_address>:5006/kria-dashboard**



9. Wdrażanie modelu na Zynq MPSoC

Pobierz folder **resnet18_pt** z hosta do celu za pomocą **scp** następującym poleceniem:

```
[Docker] $ scp -r resnet18_pt
root@[TARGET_IP_ADDRESS]:/usr/share/vitis_ai_library/models/
```

Model zostanie umieszczony w folderze **/usr/share/vitis_ai_library/models/** wraz z innymi przykładami modeli Vitis-AI.

Pakiety **vitis_ai_library_r3.0.0_images.tar.gz** i **vitis_ai_library_r3.0.0_video.tar.gz** zawierają obrazy testowe i filmy, które można wykorzystać do oceny naszego skwantyzowanego modelu i innych wstępnie zbudowanych przykładów biblioteki Vitis-AI na targecie. Pobierz paczki:

```
[Docker] $ cd /workspace
[Docker] $ wget
https://www.xilinx.com/bin/public/openDownload?filename=vitis_ai_l
ibrary_r3.0.0_images.tar.gz -O
vitis_ai_library_r3.0.0_images.tar.gz
[Docker] $ wget
https://www.xilinx.com/bin/public/openDownload?filename=vitis_ai_l
ibrary_r3.0.0_video.tar.gz -O vitis_ai_library_r3.0.0_video.tar.gz
```

Skopiuj pliki na Krie przy użyciu **scp**:

```
[Docker] $ scp -r vitis_ai_library_r3.0.0_images.tar.gz
root@[TARGET_IP_ADDRESS]:~/
[Docker] $ scp -r vitis_ai_library_r3.0.0_video.tar.gz
root@[TARGET_IP_ADDRESS]:~/
```

Rozpakuj pliki na płytce:

```
[Target] $ tar -xzvf vitis_ai_library_r3.0.0_images.tar.gz -C
~/Vitis-AI/examples/vai_library/
[Target] $ tar -xzvf vitis_ai_library_r3.0.0_video.tar.gz -C
~/Vitis-AI/examples/vai_library/
```

Wejdź do katologu sampla i go skompiluj:

```
[Target] $ cd ~/Vitis-AI/vai_library/samples/classification
[Target] $ ./build.sh
```

Uruchom aplikację testową dla pojedynczego obrazu:

```
[Target] $ ./test_jpeg_classification resnet18_pt ~/Vitis-
AI/examples/vai_library/samples/classification/images/003.JPEG
```

Obrazek **003.JPEG** wygląda następująco:

Wynik naszej skwantyzowanej sieci:

```
root@xilinx-k26-kv:~/Vitis-AI/examples/vai_library/samples/classification# ./test_jpeg_classification resnet18_pt ~/Vitis-AI/example
s/vai_library/samples/classification/images/003.JPEG
WARNING: Logging before InitGoogleLogging() is written to STDERR
I1013 04:49:09.732903 87325 demo.hpp:1193] batch: 0     image: /home/root/Vitis-AI/examples/vai_library/samples/classification/image
s/003.JPEG
I1013 04:49:09.733121 87325 process_result.hpp:24] r.index 286 cougar, puma, catamount, mountain lion, painter, panther, Felis conco
lor, r.score 0.999173
I1013 04:49:09.733534 87325 process_result.hpp:24] r.index 287 lynx, catamount, r.score 0.000709587
I1013 04:49:09.733669 87325 process_result.hpp:24] r.index 290 jaguar, panther, Panthera onca, Felis onca, r.score 4.53624e-05
I1013 04:49:09.733880 87325 process_result.hpp:24] r.index 291 lion, king of beasts, Panthera leo, r.score 1.66879e-05
I1013 04:49:09.734059 87325 process_result.hpp:24] r.index 282 tiger cat, r.score 1.66879e-05
```

Źródła:

https://xilinx.github.io/Vitis-AI/3.0/html/docs/install/install.html
https://xilinx.github.io/Vitis-AI/3.0/html/docs/quickstart/mpsoc.html
https://docs.docker.com/engine/install/ubuntu/
https://community.element14.com/products/roadtest/b/blog/posts/amd-xilinx-kria-kv260-vision-ai-starter-kit-software
https://xilinx.github.io/Vitis-AI/3.0/html/docs/install/install.html
https://github.com/Xilinx/Vitis-AI-Tutorials
https://github.com/Xilinx/Vitis-AI