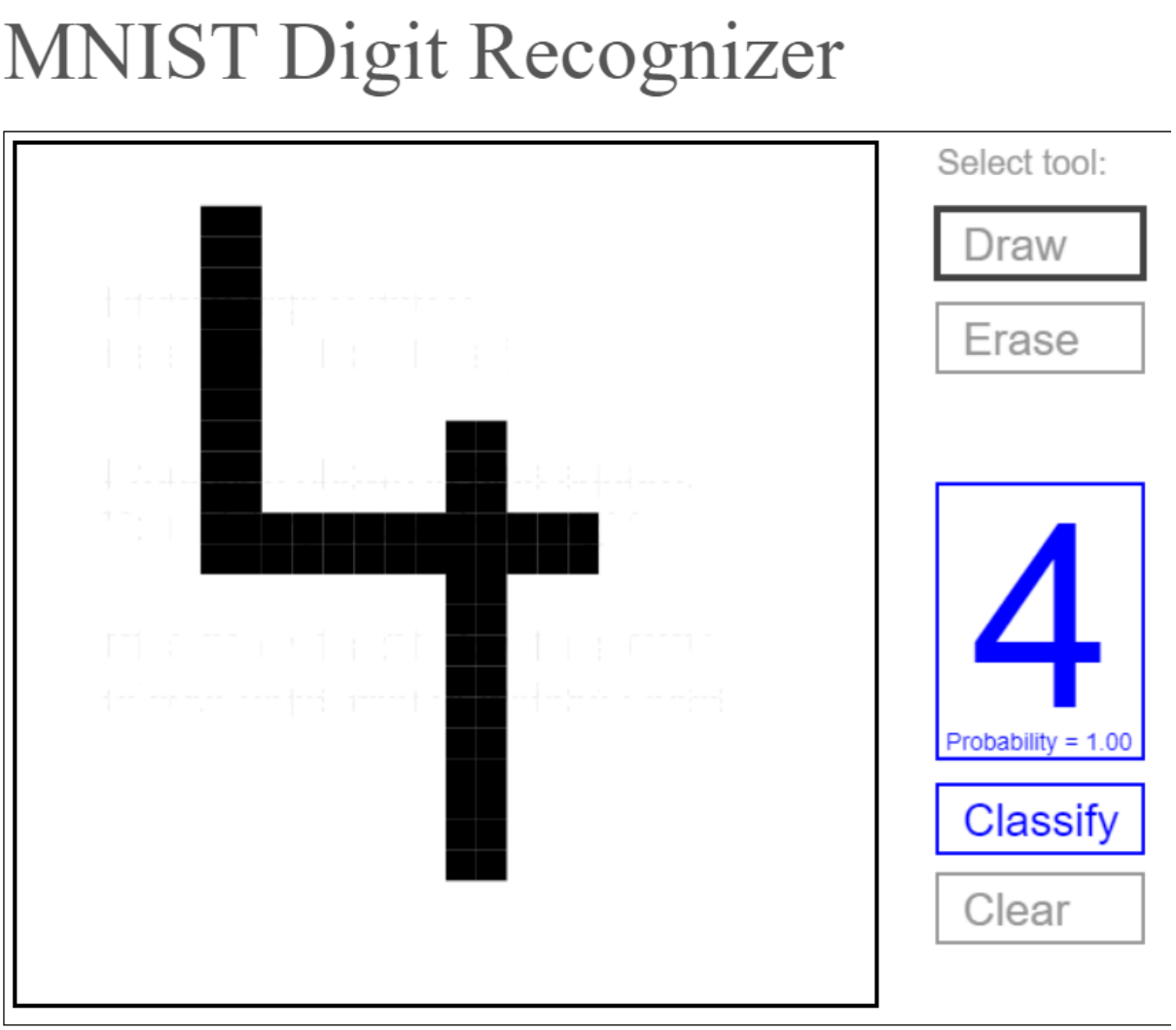
1. Cel ćwiczenia

W tym ćwiczeniu będziemy kwantyzować i analizować działanie przygotowanej przed zajęciami sieci neuronowej, które przedstawia bazę danych MNIST.

2. Baza danych MNIST

MNIST to popularny zestaw danych używany w dziedzinie uczenia maszynowego i rozpoznawania wzorców, szczególnie do trenowania i testowania algorytmów rozpoznawania obrazów. Składa się z ręcznie pisanych cyfr od 0 do 9, które są zapisane w formie obrazów o wymiarach 28x28 pikseli w odcieniach szarości. Zbiór zawiera 60 000 przykładów w zestawie treningowym oraz 10 000 przykładów w zestawie testowym. Ten dataset jest uważany za odpowiednik ‘Hello World’ w machine learningu i stanowi prosty wstęp do obsługi środowiska Vitis-AI.



https://www.ccom.ucsd.edu/~cdeotte/programs/MNIST.html

3. Przygotowanie do ćwiczenia

Sprawdź czy na twojej maszynie wirtualnej jest zainstalowany Python.

```
[Host] $ python3 —-version
```

Jeśli nie, zainstaluj za pomocą następujących komend.

```
[Host] $ sudo apt update

[Host] $ sudo apt install python3
```

Pobierz bibliotekę torch.

```
[Host] $ pip3 install torchvision
```

Pobierz z repozytorium pliki wymagane do przeprowadzenia ćwiczenia.

```
[Host] $ git clone
https://github.com/wierzba100/Mnist_Vitis_Ai.git
```

W repozytorium znajdują się między innymi pliki do tworzenia i testowania sieci **main.py** oraz **test.py**. Będziemy chcieli sprawdzić ich działanie. Uruchomiony skrypt main.py generuje wytrenowaną sieć mnist_model.pth (proces ten może zająć do ~10 minut). W przypadku skryptu test.py należy pamiętać, by zmienić ścieżkę do obrazu typu MNIST.

```
[Host] $ python3 main.py

[Host] $ python3 test.py
```

4. Uruchomienie Vitis-AI i dockera

Podobnie jak na poprzednich zajęciach, przejdź do folderu z repozytorium **Vitis-AI** i uruchom środowisko.

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

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

Następnie uruchom **dockera** za pomocą komendy.

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

Uruchom środowisko **conda** dla **vitis-ai-pytorch**.

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

5. Kwantyzacja modelu

Przypominając opis z pierwszego laboratorium, 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.

Teraz przydadzą nam się skrypty uruchamiane w punkcie 3. Przekopiuj folder **mnist/** do **~/Vitis-AI/** i przejdź do tego folderu. Następnie stwórz folder **model/** przekopiuj do niego wcześniej wytrenowaną sieć **mnist_model.pth**.

```
[Docker] $ cp -a /source/. /dest/

[Docker] $ cd mnist
```

Wykonaj kolejno następujące komendy. Ich objaśnienie znajduje się poniżej.

```
[Docker] $ python quant.py --quant_mode float --data_dir
mnist_data --model_dir model

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

[Docker] $ python quant.py --quant_mode calib --data_dir
mnist_data --model_dir model --subset_len 200

[Docker] $ python quant.py --model_dir model --data_dir
mnist_data --quant_mode test

[Docker] $ python quant.py --quant_mode test --subset_len 1
--batch_size=1 --model_dir model --data_dir mnist_data --deploy
```

--quant_mode float - oznacza, że model będzie wykorzystywał liczby zmiennoprzecinkowe (float)

--data_dir mnist_data - wskazuje na folder mnist_data, który zawiera bazę danych MNIST

--model_dir model - wskazuje na folder model, gdzie nasz model będzie zapisany

--inspect - umożliwia inspekcję modelu

--target DPUCZDX8G_ISA1_B4096 - oznacza docelowe DPU. W naszym przypadku jest to rodzaj DPUCZDX8G_ISA1_B4096

--quant_mode calib - ustawia tryb kwantyzacji w tryb kalibracji

--subset_len 200 - ogranicza zestaw danych do kalibracji do 200 próbek

--quant_mode test - uruchomienie w trybie testowym

--deploy - przygotowuje model do jego wdrożenia

--batch_size=1 - dane będą przetwarzane po jednym przykładzie na raz

6. Kompilacja modelu

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

Kompilator pobiera kwantyzowany modell i generuje wdrażany 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.

Skopiuj poniższy skrypt do obszaru roboczego:

```
[Docker] $ vai_c_xir -x quantize_result/Net_int.xmodel -a
/opt/vitis_ai/compiler/arch/DPUCZDX8G/KV260/arch.json -o Net_int
-n Net_int
```

7. Uruchomienie Zynq MPSoC

KROK 1: Podłącz kable do urządzenia KRIA

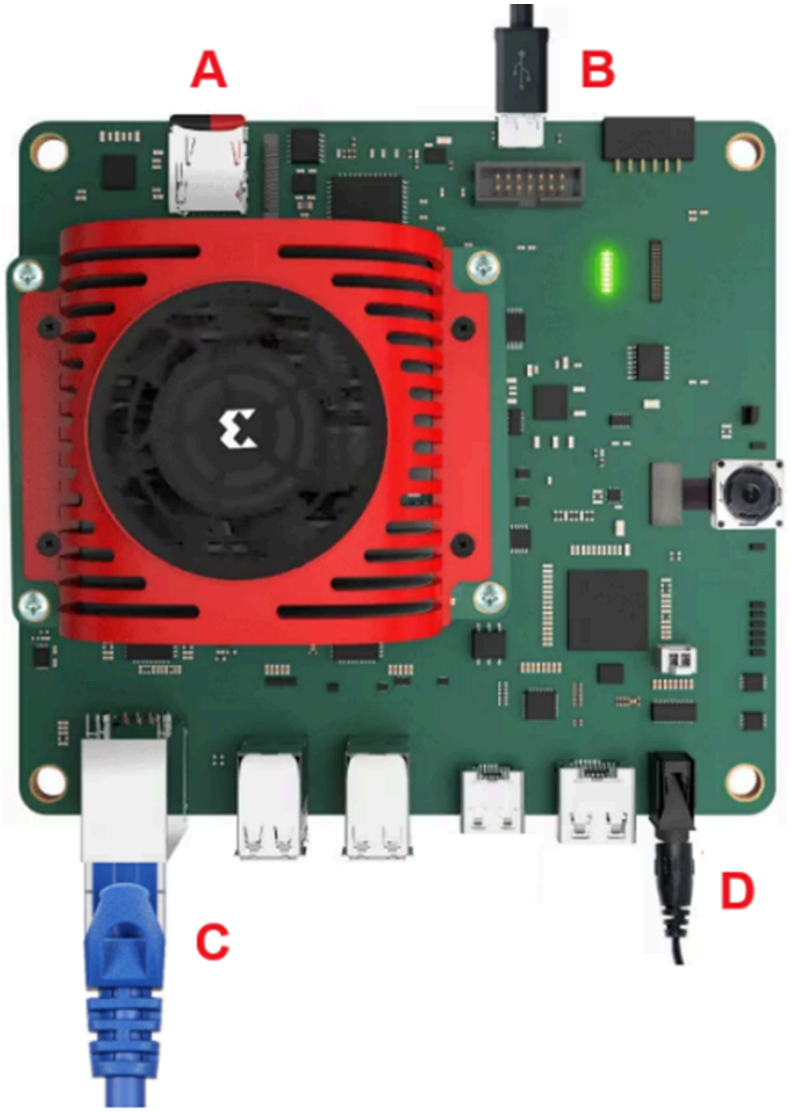
A. Włóż kartę microSD do gniazda J11

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 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.

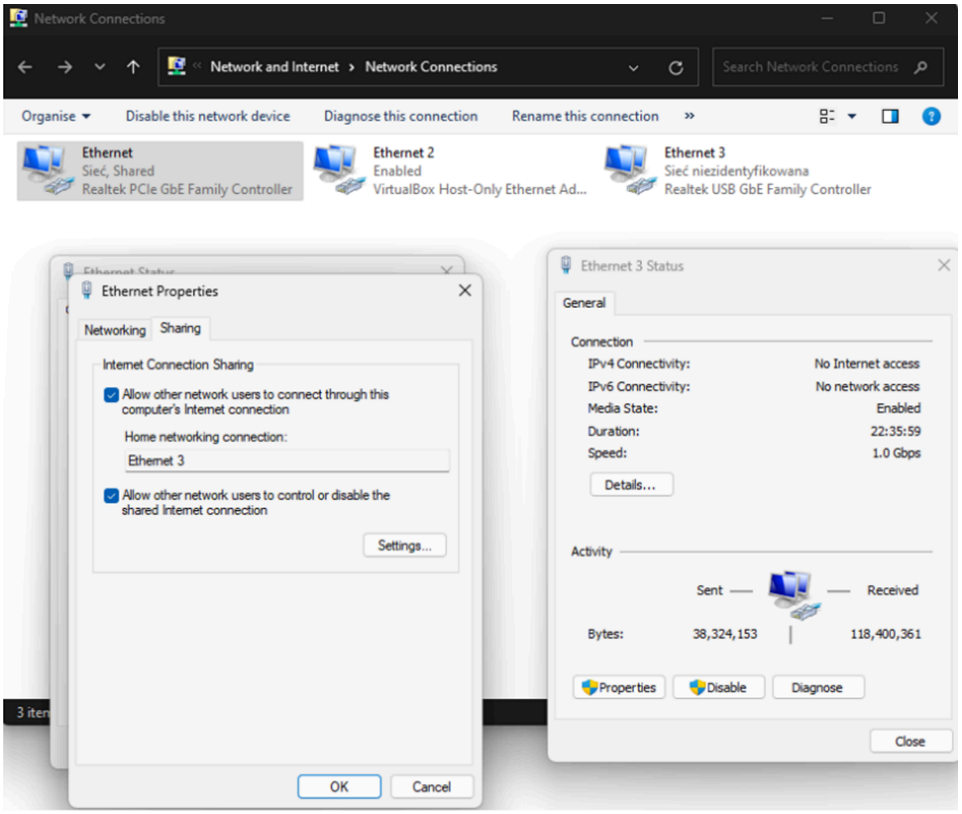
Dostępne są dwie opcje:

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:

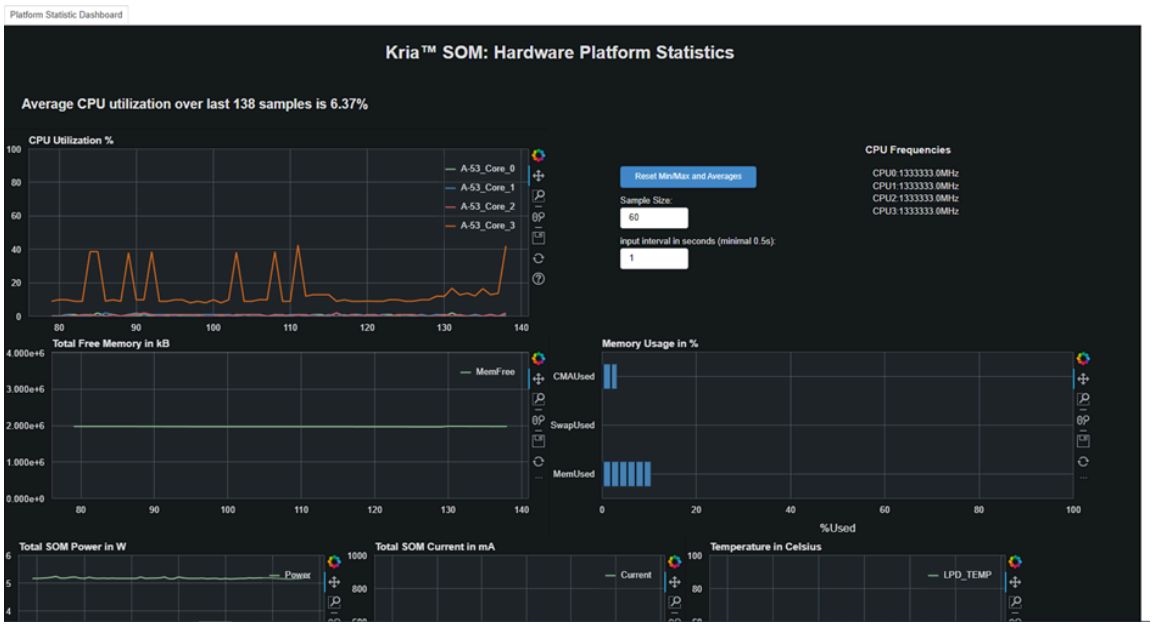
username: root, password: root

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
```

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



8. Wdrażanie modelu na Zynq MPSoC

Zaloguj się na płytkę za pomocą **SSH**:

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

Do folderu **Net_int/** przekopiuj skrypt **Mnist_Vitis_Ai/target_scripts/decode.py**. Następnie skopiuj pliki na Krię przy użyciu **scp**.

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

Na płytce przejdź do folderu **Net_int**.

```
[Target] $ cd /usr/share/vitis_ai_library/models/Net_int/
```

Uruchom sieć za pomocą komendy xdputil run <xmodel> [-i <subgraph_index>] <input_bin>. <input_bin> oznacza nasze dane wejściowe, a więc zdjęcie MNIST w formacie .npy.

```
[Target] $ xdputil run <xmodel> [-i <subgraph_index>]
<input_bin>
```

Nasza sieć jako argument przyjmuje tablicę .npy. W celu jej odpowiedniego przygotowania, należy ją pobrać lub stworzyć zdjęcie MNIST (28x28), a następnie przerobić je za pomocą skryptu **Mnist_Vitis_Ai/target_scripts/convert.py**.

Można wykorzystać przykładowe zdjęcia z bazy MNIST, znajdujące się pod poniższym linkiem: https://www.kaggle.com/datasets/alexanderyyy/mnist-png

Tak przygotowaną tablicę można przesłać na płytkę poprzez scp.

Uruchom sieć za pomocą komendy:

```
[Target] $ xdputil run Net_int.xmodel path_to_array/array.npy
```

Sieć ta powinna zwrócić wynik w postaci binarnej, a w terminalu powinien wyświetlić się komunikat: "dump output to 0.Net__Net_Linear_fc2__ret_fix.bin". W celu jego zdekodowania uruchamiamy poniższy skrypt:

```
[Target] $ python3 decode.py
```

W konsoli zwrócony zostanie wynik z poszczególnymi wartościami dla każdej z liczb. Im wyższa jego wartość, tym bardziej podobna jest liczba.