POZNAN UNIVERSITY OF TECHNOLOGY ACADEMIC JOURNALS
No 106 Electrical Engineering 2021

DOI 10.21008/j.1897-0737.2021.106.0003

Robert SMYK*, Paweł KOWALSKI*

PRZEGLĄD METOD SZYBKIEGO PROTOTYPOWANIA ALGORYTMÓW UCZENIA MASZYNOWEGO W FPGA

W artykule opisano możliwe do wykorzystania otwarte narzędzia wspomagające szybkie prototypowanie algorytmów uczenia maszynowego (ML) i sztucznej inteligencji (AI) przy użyciu współczesnych platform FPGA. Przedstawiono przykład szybkiej ścieżki przy realizacji toru wideo wraz z implementacją przykładowego algorytmu przetwarzania w trybie na żywo.

SŁOWA KLUCZOWE: algorytmy AI, FPGA, szybkie prototypowanie w FPGA.

1. WSTĘP

Uczenie maszynowe (ML) to w pewnym sensie implementacja algorytmów sztucznej inteligencji (AI) w urządzeniach lub w konkretnych środowiskach, co dalej w bardzo optymistycznym ujęciu umożliwia adaptacyjne poprawianie właściwości użytkowych takiego urządzenia. Istotne jest przy tym osiągnięcie poprawy właściwości bazując podczas pracy algorytmu na analizie danych pochodzących ze środowiska, bez ingerencji człowieka. Bardzo duże zainteresowanie wdrażaniem AI w różnych dziedzinach można częściowo wytłumaczyć dynamicznym rozwojem technologii wytwarzania układów elektronicznych przy stałym podnoszeniu ich wydajności obliczeniowej i obniżaniu kosztów. Przekłada się to na możliwości oferowania konsumentom stosunkowo tanich i bardzo wydajnych miniaturowych komputerów, czyli platform, na których oprogramowanie zawierające algorytmy sztucznej inteligencji może pracować. Ciekawostkę może stanowić fakt, że w zasadzie układ mogący pełnić funkcję miniaturowego i stosunkowo wydajnego komputera można nabyć w cenie dużo niższej niż posiłek w szanującym się lokalu. Przykładowo współczesny mikrokontroler 32-bitowy serii STM32H7 [1] charakteryzujący się wydajnością na poziomie 1327 DMIPS (ang. Dhrystone Million Instructions Per Seconds) i 3224 punktów w teście CoreMark [2], kosztuje mniej niż 8USD za pojedynczy układ. Dodatkowym i bardzo istotnym czynnikiem pobudzającym zainteresowanie AI jest łatwość implementacyjna. Stoi za tym dostępność otwartych środowisk, takich

* Politechnika Gdańska.

jak Tensorflow [11], zawierający przyjazne w użyciu narzędzia wspomagające prototypowanie rozwiązań ML w schemacie "od początku do końca", czyli od zaprojektowania do zakodowania algorytmu. Wspomniany Tensorflow stanowi kompleksową platformę oprogramowania udostępnianego na licencji otwartej z przeznaczeniem do uczenia maszynowego. Platforma ta z jednej strony pozwala naukowcom na wdrażanie nowych algorytmów a inżynierom na stosunkowo łatwe tworzenie i wdrażanie aplikacji ML. Dodatkowo, producenci sprzętu jak STMicroelectronics, Texas Instruments, Intel, Xilinx i inni starają się tworzyć pakiety oprogramowania, które pośredniczą podczas implementacji algorytmu w konkretnym środowisku sprzętowym producenta. Przykładowo STMicroelectronics oferuje STM32Cube.AI, które umożliwia konwertowanie algorytmu AI do zoptymalizowanego kodu, który można uruchomić na rzeczywistym mikrokontrolerze STM32 [3].

Mimo dostępności różnych platform sprzętowych, które teoretycznie nadają się do implementacji algorytmów ML, najpowszechniej wykorzystywane są silniki GPU albo środowiska FPGA [4]. W przypadku GPU (ang. Graphics Processing Unit) kompletne środowisko uruchomieniowe musi być jednak oparte o CPU (ang. Central Processing Unit), typowo jest to stacja robocza PC. Istnieją co prawda układy typu SoC (ang. System on Chip) integrujące GPU i CPU, ale ich moc obliczeniowa jest o kilka rzędów niższa od rozwiązań dyskretnych (odseparowane CPU i GPU), np Tesla P40 (40 INT8 TOP/s). Okazuje się, że współczesne układy FPGA dorównują topowym układom GPU pod względem specyficznych właściwości przetwarzania. Przykładowo układ Ultrascale+ XCVU13P oferuje podobną do wymienionego GPU zdolność przetwarzania operandów typu INT na poziomie 38.3 INT8 TOP/s. Układy FPGA wyposażone są w lokalne bloki pamięci BRAM, które są wykorzystywane w aplikacjach ML w roli lokalnych pamięci podręcznych (Cache). Znane są również opracowania, w których wykazuje się, że rozwiązania bazujące na FPGA potrafią być niemalże dziesięciokrotnie sprawniejsze energetycznie, niż rozwiązania o takim samym przeznaczeniu bazujące na GPU [5]. W takim ujęciu można powiedzieć, że FPGA z dedykowaną implementacją sprzętową może stanowić kompletne dyskretne rozwiązanie sprzętowe, które nadaje się do wbudowania np. w urządzeniu brzegowym pracującym w strukturze IoT [6, 7].

Podstawową zaletą prototypowania ML na GPU jest natywna dostępność środowisk programistycznych i bibliotek, takich jak wymieniony Tensorflow. Moduł z układem GPU stanowi zwykle integralny składnik stacji roboczej PC, na której bardzo łatwo można skonfigurować dedykowane kompilatory lub środowiska ze zintegrowanymi bibliotekami i narzędziami. Dla kontrastu w przypadku wykorzystywania FPGA do prototypowania algorytmów ML nie istnieje takie bezpośrednie i integralne połączenie stacji roboczej PC z układem FPGA. Dla większości inżynierów prototypujących systemy wbudowane jest oczywiste, że oprogramowanie przygotowuje się na PC, natomiast uruchamia się

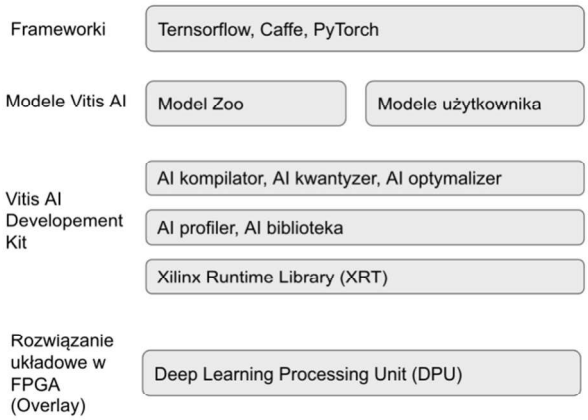
je na odseparowanej platformie FPGA czy z mikrokontrolerem. Producenci FPGA obecnie mają w ofercie rozwiązania hybrydowe integrujące w jednym układzie strukturę programowalną PL (ang. Programmable Logic) oraz strukturę umożliwiająca wykonywanie programu PS (ang. Processing System), czyli CPU. Przykładowo, układy z serii Xilinx Zynq Ultrascale posiadają oprócz struktury FPGA, o zasobach sprzętowych porównywalnych z FPGA Virtex 7, również 4 rdzeniowe CPU klasy ARM Cortex A5x pozwalające na uruchomienie systemu operacyjnego takiego jak Petalinux [8]. Pozwala to uzyskać środowisko sprzętowe w jednym układzie SoC, które ma strukturę rekonfigurowalną, ale dzięki temu, że można uruchomić na nim oprogramowanie umożliwia bezpośrednią interakcję użytkownika-programisty, na podobnej zasadzie jak z wykorzystaniem stacji PC z wbudowanym GPU. W tym przypadku jednostkę GPU lub inny akcelerator można zaimplementować w strukturze FPGA (PL). Warto również wspomnieć o tym, że istnieją rozwiązania pozwalające na zintegrowanie FPGA ze stacją roboczą. Przykładem może być tu platforma Xilinx Alveo, która umożliwia uruchamianie akceleratorów sprzętowych pozwalających na swobodny przepływ danych z wykorzystaniem PCIe/DMA między CPU/RAM w stacji roboczej [9]. Tego typu rozwiązania mają zastosowanie do budowy rekonfigurowalnych rozwiązań dedykowanych do pracy w chmurze. W dobie rozwiązań internetu rzeczy dyskretna platforma FPGA jest postrzegana, jako urządzenie krawędziowe (ang. Edge) a platforma osadzona w serwerze klasy PC (jak wspomniana Alveo) jest uznawana za element infrastruktury sprzętowej chmury (ang. Cloud) [10].

Niestety uruchamianie aplikacji z wykorzystaniem FPGA wymaga oprócz znajomości typowych narzędzi programowych z zakresu ML również wiedzy specjalistycznej z zakresu projektowania sprzętu, znajomości architektur systemów komputerowych a także umiejętności poruszania się w środowisku Linux. Z pozoru wydaje się, że wykorzystanie FPGA do ML jest zadaniem dużo trudniejszym niż w przypadku klasycznej konfiguracji wykorzystującej PC i GPU. W praktyce istnieją narzędzia oferowane przez producenta na zasadzie dystrybucji otwartego oprogramowania. Większość z nich to biblioteki oraz pakiety oprogramowania znane od lat deweloperom. Istota rzeczy tkwi jednak w zrozumieniu przebiegu kolejnych czynności, jakie należy wykonać od zaplanowania i wyboru modelu do implementacji w FPGA. Żargonowo ścieżka postępowania w celu implementacji jest określana terminem „workflow".

W artykule opisano skróconą ścieżkę do prototypowania ML przy użyciu układów FPGA w środowisku Xilinx Vitis AI. W punkcie 2 opisano logikę dostępnych narzędzi oraz tzw. „workflow". W punkcie 3 opisano przykłady uruchomienia toru przetwarzania ML bezpośrednio w środowisku FPGA. W punkcie 4 opisano podejście do opracowania własnego modelu.

2. NARZĘDZIA SZYBKIEGO PROTOTYPOWANIA AI W FPGA

Środowisko deweloperskie Xilinx Vitis AI [10] stanowi zbiór zintegrowanego oprogramowania pomocnego przy implementacji algorytmów sztucznej inteligencji przy użyciu platform FPGA. Posiada ono strukturę warstwową (rys. 1), z jednej strony integruje narzędzia programowe służące do przygotowania modeli a z drugiej strony pozwala na wygenerowanie struktury sprzętowej modelu, którą można zaimplementować w FPGA. Przedstawiony na ilustracji schemat można traktować także, jako „workflow" przy opracowywaniu aplikacji AI w omawianym środowisku. W Vitis AI wkomponowano popularne dostępne biblioteki wspomagające projektowanie sztucznych sieci neuronowych. Istnieje możliwość wykorzystania innymi: Caffe, Tensorflow i PyTorch [11]. Dzięki temu możliwe jest opracowanie własnych modeli użytkownika lub wykorzystanie istniejących (Xilinx Model Zoo), przygotowanych przez Xilinx. W dostępnej od kilku miesięcy Vitis AI w wersji 1.3 wbudowanych jest przeszło 90 gotowych modeli, które mogą zostać wykorzystane w aplikacji bazującej na technikach głębokiego uczenia. Warto zaznaczyć, że oferowane modele są w pełni imeplementowalne sprzętowo w strukturze układu FPGA. W celu implementacji wybranego modelu minimalnym wymaganiem, jakie należy spełnić, jest wygenerowanie zbioru wag sieci neuronowej. W ramach Xilinx Runtime Developement Kit udostępniono zbiór głównych narzędzi, które pozwalają na opracowanie modelu AI i następnie wygenerowanie jego struktury możliwej do zaimplementowania w FPGA.



Rys. 1. Warstwowa struktura środowiska Xilinx Vitis AI

Sama struktura sprzętowa jednostki przetwarzającej i implementującej wygenerowany model jest zunifikowana dla wszystkich modeli. Struktura ta złożona jest z modułów sprzętowych typu IP syntezowanych do formy tzw. struktury na poziomie rejestrów RTL (ang. Register Transfer Level) modelującej przepływ

sygnałów od wejść do wyjść. Sprzętowa jednostka przetwarzania została nazwana DPU (ang. Deep Learning Processing Unit) [12]. Pełni ona rolę koprocesora zdolnego przetwarzać w trybie współbieżnym zoptymalizowane instrukcje modelujące funkcjonowanie sieci neuronowej.

Z punktu widzenia pracy inżyniera opracowującego rozwiązanie sprzętowe ML przy użyciu Xilinx FPGA platforma Vitis AI rozdziela się na dwa podstawowe środowiska: lokalne na stacji PC oraz w pewnym sensie zdalne na urządzeniu krawędziowym FPGA lub w chmurze z wykorzystaniem platformy PCIe-FPGA. Środowisko na urządzeniu krawędziowym pełni funkcję uruchomieniową dla opracowanych implementacji ML i jest instalowane lokalnie w obrazie systemu operacyjnego klasy Linux pracującego na układzie SoC FPGA. W tym przypadku zwykle jest to Petalinux dedykowany dla rozwiązań Xilinx'a lub wersja Ubuntu dla systemów wbudowanych ARM. Vitis AI oprócz środowiska uruchomieniowego zawiera DPU dystrybuowany w formie pliku binarnego ładowanego do struktury FPGA oraz zbiór modeli rezydujących w plikach '.xmodel'. W dużym uproszczeniu wykorzystanie DPU sprowadza się do wywołania go z poziomu API Python lub C/C++ a następnie przekierowania do niego strumienia danych wejściowych i odprowadzenia danych wyjściowych. Przykładowo dane wejściowe do DPU mogą być strumieniem pochodzącym z kamery USB podłączonej do SoC FPGA a dane wyjściowe mogą zostać przekształcone do strumienia kompatybilnego z HDMI/Display Port. Okazuje się, że jest to jeden z typowych wariantów testowania algorytmów ML w omawianym środowisku. Innym wariantem jest lokalny odczyt np. pliku graficznego, przekształcenie go do postaci kompatybilnej z wejściem DPU a następnie odprowadzenie informacji z wyjścia DPU. Należy wspomnieć, że DPU na wyjściu udostępnia zbiór danych w podobnej postaci jak typowa sieć neuronowa.

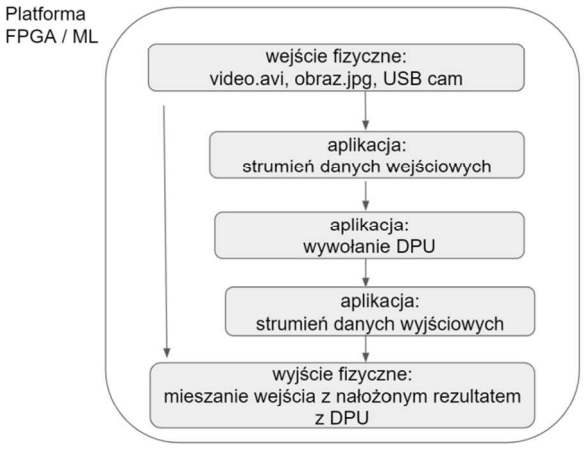
Vitis AI uruchamiane na stacji PC pełni funkcję głównego środowiska deweloperskiego umożliwiającego trening, kwantyzację czy kompilację modelu. W tym przypadku jest ono uruchamiane jako kontener Docker z wydzielonymi mikrośrodowiskami Python dla konkretnej biblioteki ML (rys. 2). Należy wspomnieć, że każda ze znanych bibliotek ML wymaga nieco odmiennej konfiguracji oraz różnego drzewa zależności bibliotek wspomagających, co uzasadnia ideę wydzielania mikrośrodowisk.

Jak pokazano na ilustracji z rys. 2 początkowy etap 'workflow' Vitis-AI wymusza wybranie konkretnej opcji biblioteki, w tym przypadku pokazano aktywację mikrośrodowiska Caffe. Dalej możliwa jest praca z konkretnym modelem, obejmująca np. pozyskanie jego plików źródłowych oraz kompilację dla konkretnej platformy FPGA. Czynności te zostaną skrótowo omówione w punkcie 3. W kolejnym drugim punkcie omówiona zostanie na przykładzie procedura uruchamiania implementacji konkretnego modelu Vitis AI w urządzeniu brzegowym FPGA.

Rys. 2. Vitis AI uruchomione lokalnie w formie kontenera Docker z aktywnym mikrośrodowiskiem Python/Caffe

3. PRZYKŁAD TORU PRZETWARZANIA AI W FPGA

Typowy schemat toru przetwarzania przy użyciu Vitis AI (rys. 3) może składać się ze źródła danych, które fizycznie może być zrealizowane w postaci dołączonej kamery USB.



Rys. 3. Schemat przykładowego toru przetwarzania strumienia wideo w FPGA przy użyciu Vitis AI

W środowisku Linux (Petlalinux) sterowniki typowych kamer USB są już wbudowane, co zwalnia z konieczności ich implementacji czy instalacji. Dane wejściowe mogą być również czytane z pliku video lub jako obraz statyczny. Rezultat wyjściowy pochodzący z DPU stanowi zwykle zbiór współczynników określających stopień klasyfikacji obiektu wejściowego. Aby można było taki rezultat wykorzystać wymagana jest dodatkowa programowa interpretacja. Przykładowo, rezultat wykrycia twarzy może zostać zwizualizowany poprzez naniesienie na obraz wejściowy zawierający pierwotny kształt twarzy dodatkowo kształtu prostokątnego okalającego twarz. Zwykle w środowiskach takich jak Petalinux zawarte są również sterowniki urządzeń wyjściowych video, dlatego też rezultat można z poziomu aplikacji przekazać na wyjście wideo.

Środowisko Vitis AI jest dystrybuowane w postaci otwartych źródeł przy użyciu platformy Github. Jest ono również dostępne w formie odpowiednio przygotowanego obrazu zawierającego dystrybucję Petalinux wraz ze zintegrowaną biblioteką Vitis AI w wersji 1.x. Obrazy są dostępne dla płyt deweloperskich Xilinx ZCU102 i ZCU104 oraz Avnet Ultra96v2. Demonstracje przykładów przeprowadzono przy użyciu platformy ostatniej z wymienionych. Wykorzystano gotowy obraz Petlalinux z dostępnym dedykowanym akceleratorem DPU w najnowszej wersji 1.4.1 (rys. 4). Należy wspomnieć, że istnieje możliwość skompilowania składników Petalinux oraz wszystkich wymaganych bibliotek. Jest to jednak zadanie stosunkowo czasochłonne i raczej rekomendowane dla specyficznych konfiguracji sprzętowych, dla których nie możliwe jest pobranie gotowego obrazu systemu.

```
root@u96v2-sbc-base-2020-2:~# dexplorer --whoami
[DPU IP Spec]
IP  Timestamp            : 2020-11-02 15:15:00
DPU Core Count           : 1

[DPU Core Configuration List]
DPU Core                 : #0
DPU Enabled              : Yes
DPU Arch                 : B2304
DPU Target Version       : v1.4.1
DPU Fregency             : 300 MHz
Ram Usage                : Low
DepthwiseConv            : Enabled
DepthwiseConv+Relu6      : Enabled
Conv+Leakyrelu           : Enabled
Conv+Relu6               : Enabled
Channel Augmentation     : Enabled
Average Pool             : Enabled
```

Rys. 4. Lista parametrów konfiguracji DPU

Wraz z dostępem do zbioru modeli Vitis AI również dostępna jest baza przykładów, która zawarta jest w katalogu domowym (rys 5). Każdy przykład składa się ze skompilowanej wersji aplikacji gotowej do uruchomienia. Udostępnione są również pliki źródłowe zawierające kody, które można zmodyfikować a na-

stępnie skompilować przy użyciu gotowego skryptu ‘build.sh’ (rys. 6). Większość przykładów można uruchomić według typowego schematu, tj. wywołanie pliku wykonywalnego, następnie podanie danych wejściowych - może być to obraz statyczny lub plik video, w dalszej kolejności specyfikuje się typ użytego modelu. Przykładowo uruchomienie aplikacji ‘facedetect’ wymaga polecenia *./test_video_facedetect densebox_640_360 0*, gdzie specyfikujemy rodzaj modelu (densebox) oraz w tym przypadku identyfikator urządzenia fizycznego pobierania obrazu (kamera USB). W efekcie uruchomienia widoczne będzie okno z rezultatem przetwarzania, podobne do rys. 7.

```
root@u96v2-sbc-base-2020-2:~/Vitis-AI/demo/Vitis-AI-Library/samples# ls
3Dsegmentation       facequality5pt       multitask     refinedet     yolov2
classification       hourglass            openpose      reid          yolov3
covid19segmentation  lanedetect           platedetect   retinaface    yolov4
facedetect           medicaldetection     platenum      segmentation
facefeature          medicalsegcell       pointpillars  ssd
facelandmark         medicalsegmentation  posedetect    tfssd
```

Rys. 5. Lista jednego ze zbiorów przykładów Vitis AI v1.3

```
lsot@u96v2-sbc-base-2020-2:~/Vitis-AI/demo/Vitis-AI-Library/samples/facedetect#
build.sh                      test_accuracy_facedetect_mt
gstshark_2021-01-26_05:47:39  test_jpeg_facedetect
images                        test_jpeg_facedetect.cpp
process_result.hpp            test_performance_facedetect
readme                        test_performance_facedetect.cpp
sample_facedetect.jpg         test_performance_facedetect.list
test_accuracy_facedetect      test_video_facedetect
test_accuracy_facedetect.cpp  test_video_facedetect.cpp
```

Rys. 6. Lista plików źródłowych oraz plików wykonywalnych aplikacji „facedetect”

Rys. 7. Rezultat wykrywania twarzy algorytmem densebox wykonywanym na sprzętowym DPU w FPGA

Wśród innych ciekawych udostępnionych aplikacji do testów oraz modyfikacji można wymienić: rozpoznawanie tablic rejestracyjnych (platedetect i platenum), wykrywanie linii na drodze (linedetect), a także dostępne w katalogu VART detekcja pozy osoby (pose_detection), wykrywanie pojazdów w ruchu ulicznym (adas_detection), klasyfikacja obrazów statycznych (resnet50).

4. KOMPILACJA MODELU

Vitis AI zawiera zbiór narzędzi umożliwiających tworzenie implementacji DPU dla konkretnego skompilowanego we własnym zakresie modelu ML. W tym przypadku należy na stacji roboczej PC przygotować środowisko Vitis AI przy użyciu platformy Docker [13]. Xilinx udostępnia dwie wersje środowiska, jedna wykorzystuje do treningu, kwantyzacji i kompilacji jednostkę centralną CPU druga wykorzystuje zainstalowane GPU. Wersja dla CPU jest udostępniana w postaci gotowego kontenera dla Dockera. W niniejszej pracy używano wersji GPU, którą należało skompilować we własnym zakresie. Wersja ta wykorzystuje akceleracje GPU przy treningu sieci, co jest procesem czasochłonnym i nawet przy wykorzystaniu silnej jednostki wymaga od kilku do kilkunastu godzin ciągłej pracy algorytmu. Warto dodać, że wspierane są tylko karty graficzne zawierające wybrane układy Nvidia GeForce. Całe środowisko zostało przygotowane przy użyciu narzędzi dostępnych w Ubuntu 20.04. Udostępniane przez Xilinx oprogramowanie można również uruchomić w środowisku MS Windows, ale jest ono natywne dla platform linuksowych.

W wersji 1.3 Vitis AI dostępne są 92 prekompilowane modele dla układów FPGA serii UZ7EV i UZ3EG, dostępnych m.in. na płytach deweloperskich Xilinx ZCU102 i ZCU104 oraz Avnet Ultra96v2. Archiwum z wybranym modelem należy pobrać posługując się adresem wymienionym w pliku model.yaml specyfikującym konkretny model. Każdy model z Vitis AI posiada tak sformułowany specyficzny dla siebie opis. Na rys. 8 pokazano opis modelu *densebox* przygotowanego dla procedury detekcji twarzy, omawianej w punkcie 2. Jak widać z opisu model ten został przygotowany przy użyciu frameworka Caffe.

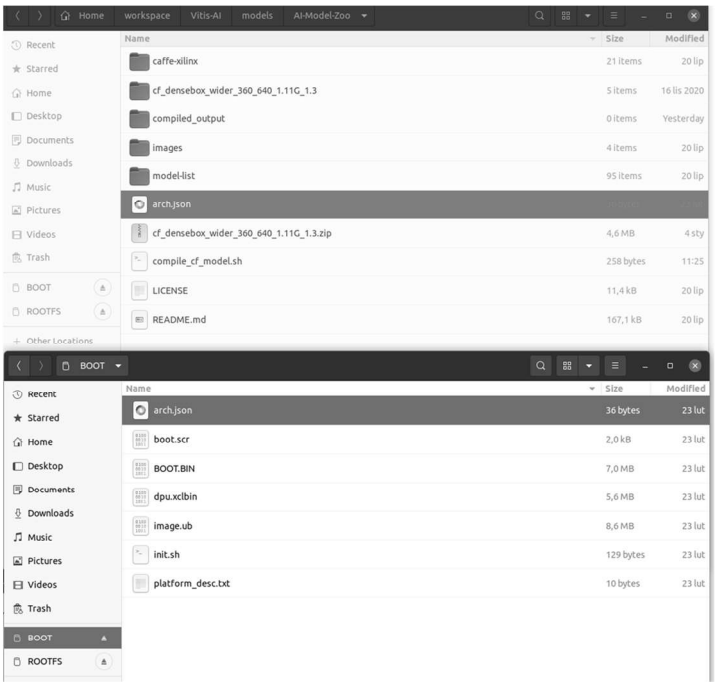
Możliwe jest skompilowanie modelu dla własnej platformy. Procedura w tym przypadku będzie dotyczyła płyty Avnet Ultra96v2 (Zynq Ultrascale UZ3EG). Kompilacja ma między innymi na celu wygenerowanie kodu wykonywalnego dla DPU reprezentującego konkretny model ML wraz ze skwantyzowanymi wagami. Jednym z wymagań jest szczegółowe wyspecyfikowanie platformy sprzętowej FPGA, która zawarta jest w specjalnie spreparowanym pliku *arch.json*. Plik taki można uzyskać kopiując go z partycji rozruchowej obrazu Petalinux przygotowanego w środowisku deweloperskim Xilinx (rys. 9). Cała procedura przygotowywania obrazu wymaga specjalistycznej wiedzy i jest czasochłonna. Gotowe obrazy są jednak udostępniane przez Xilinx bądź Avnet, zależnie od producenta platformy sprzętowej.

```
usr@usr-B460MDS3HV2:~/workspace/Vitis-AI/models/AI-Model-Zoo$ cat model-list/cf_densebox_wider_360_640_1.11G_1.3/model.yaml

# Copyright 2019 Xilinx Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.


description: face detection model.
input size: 360*640
float ops: 1.11G
task: face detection
framework: caffe
prune: 'no'
version: 1.3
files:
- name: cf_densebox_wider_360_640_1.11G_1.3
  type: float & quantized
  board: GPU
  download link: https://www.xilinx.com/bin/public/openDownload?filename=cf_densebox_wider_360_640_1.11G_1.3.zip
  checksum: 80b618cbb5bf35c842820be7f2dd3e12
```

Rys. 8. Fragment informacji z model.yaml zawierający typ modelu oraz URLe pobierania wybranej wersji



Rys. 9. Widok zawartości folderu AI-Model-Zoo (góra) oraz partycji BOOT obrazu systemu (dół)

W celu zautomatyzowania procedury kompilacji rekomenduje się opracowanie prostego skryptu zawierającego uogólnioną dla dowolnego modelu procedurę (rys. 10). Szczegóły opisane są w instrukcji użytkownika Vitis AI [10].

```
model_name=$1
modelzoo_name=$2
vai_c_caffe \
--prototxt ./${modelzoo_name}/quantized/deploy.prototxt \
--caffemodel ./${modelzoo_name}/quantized/deploy.caffemodel \
--arch ./arch.json \
--output_dir ./compiled_output/${model_name} \
--net_name ${model_name}
```

Rys. 10. Listing skryptu Bash uruchamiającego procedurę kompilacji modelu ML z Vitis AI

Jak widać z listingu procedura kompilacji odczytuje specyfikę modelu (deploy.prototx i deploy.caffemodel) i została przeprowadzana dla konkretnej platformy (specyfikacja w *arch.json*). Przykładowo skompilowano model densebox dla platformy UZ3EG. Przebieg kompilacji ilustruje rys 11.

```
(vitis-ai-caffe) Vitis-AI /workspace/models/AI-Model-Zoo > source ./compile_cf_model.sh densebox_640_360 cf_densebox_wider_360_640_1.11G_1.3
**************************************************
* VITIS_AI Compilation - Xilinx Inc.
**************************************************
[INFO] Namespace(inputs_shape=None, layout='NCHW', model_files=['./cf_densebox_wider_360_640_1.11G_1.3/quantized/deploy.caffemodel'], model_type='caffe', out_
filename='./compiled_output/densebox_640_360/densebox_640_360_org.xmodel', proto='./cf_densebox_wider_360_640_1.11G_1.3/quantized/deploy.prototxt')
[INFO] caffe model: cf_densebox_wider_360_640_1.11G_1.3/quantized/deploy.caffemodel
[INFO] caffe model: cf_densebox_wider_360_640_1.11G_1.3/quantized/deploy.prototxt
[INFO] parse raw model     :100%|████████████████████████████████████████| 43/43 [00:00<00:00, 676.83it/s]
[INFO] infer shape (NCHW)  :100%|████████████████████████████████████████| 43/43 [00:00<00:00, 53677.10it/s]
[INFO] infer shape (NHWC)  :100%|████████████████████████████████████████| 43/43 [00:00<00:00, 36479.59it/s]
[INFO] generate xmodel     :100%|████████████████████████████████████████| 43/43 [00:00<00:00, 9372.99it/s]
[INFO] generate xmodel: /workspace/models/AI-Model-Zoo/compiled_output/densebox_640_360/densebox_640_360_org.xmodel
[UNILOG][INFO] The compiler log will be dumped at "/tmp/vitis-ai-user/log/xcompiler-20210729-093234-102"
[UNILOG][INFO] Compile mode: dpu
[UNILOG][INFO] Debug mode: function
[UNILOG][INFO] Target architecture: DPUCZDX8G_ISA0_B2304_MAX_BG2
[UNILOG][INFO] Graph name: deploy, with op num: 95
[UNILOG][INFO] Begin to compile...
[UNILOG][INFO] Total device subgraph number 4, DPU subgraph number 1
[UNILOG][INFO] Compile done.
[UNILOG][INFO] The meta json is saved to "/workspace/models/AI-Model-Zoo/./compiled_output/densebox_640_360/meta.json"
[UNILOG][INFO] The compiled xmodel is saved to "/workspace/models/AI-Model-Zoo/./compiled_output/densebox_640_360/densebox_640_360.xmodel"
[UNILOG][INFO] The compiled xmodel's md5sum is ca6d431aeb66726f0d2704a51a32e5eb, and been saved to "/workspace/models/AI-Model-Zoo/./compiled_output/densebox_
640_360/md5sum.txt"
(vitis-ai-caffe) Vitis-AI /workspace/models/AI-Model-Zoo >
```

Rys. 11. Przebieg procedury kompilacji modelu densebox w środowisku Vitis AI

Rezultaty w postaci skomplikowanych modeli (plik z rozszerzeniem xmodel) rezydują w utworzonym wcześniej katalogu *compiled_output* (rys. 12). Uzyskany plik z opisem modelu można wykorzystać w procedurze uczenia maszynowego uruchamianej bezpośrednio na platformie FPGA, jak pokazano w punkcie 2. Plik *.xmodel* należy skopiować do zasobów lokalnych płyty FPGA, np. do lokalnego systemu plików Petalinux, do katalogu w którym rezydują opisy innych modeli.

```
(vitis-ai-caffe) Vitis-AI /workspace/models/AI-Model-Zoo > tree compiled_output/
compiled_output/
└── densebox_640_360
    ├── densebox_640_360_org.xmodel
    ├── densebox_640_360.xmodel
    ├── md5sum.txt
    └── meta.json
```

Rys. 12. Rezultat kompilacji modelu densebox_640_360

PODSUMOWANIE

W artykule omówiono jedno z możliwych podejść do implementacji sprzętowego toru przetwarzania algorytmów uczenia maszynowego w środowisku FPGA. Przy wykorzystaniu gotowych narzędzi takich jak Vitis AI oraz dedykowanej kompatybilnej platformy sprzętowej możliwe jest szybkie prototypowanie algorytmów ML bez konieczności programowania struktur sprzętowych na poziomie RTL. Dzięki temu odpowiednio przeszkoleni programiści są w stanie bez znajomości tajników implementacyjnych na poziomie sprzętowym wdrażać rozwiązania ML wykorzystujące dedykowany koprocesor DPU.

LITERATURA

[1] STMicroelectronic, nota katalogowa DS12110 Rev 8, dostęp online: https://www.st.com/resource/en/datasheet/stm32h743vi.pdf (dostęp na dzień 30.07.2021 r.).
[2] EMBC, wykaz danych pochodzących z benchmarku dostępny online: https://www.eembc.org/coremark/scores.php (dostęp na dzień 30.07.2021 r.).
[3] Apfeldorfer R., STM32 Artificial Intelligence Solutions, MDG/MCD/AI Solutions, styczeń 2021, dostęp online: www.st.com. (dostęp na dzień 30.07.2021 r.).
[4] Fallahlalehzari F., FPGA vs GPU for Machine Learning Applications: Which one is better?, ALDEC, dostęp online: https://www.aldec.com, (dostęp na dzień 30.07.2021 r.).
[5] Ovtcharov, Kalin, Accelerating deep convolutional neural networks using specialized hardware, Microsoft Research Whitepaper 2.11, 2015.
[6] Khona C., Key Attributes of an Intelligent IIoT Edge Platform, Xilinx White Paper: All Programmable Devices WP493 (v1.0) September 6, 2017.
[7] Fu Y., Wu E., Sirasao A., Attia S., Khan K., Wittig R., Deep Learning with INT8 Optimization on Xilinx Devices, Xilinx WP486 (v1.0.1) April 24, 2017.
[8] Xilinx, PetaLinux Tools Documentation Reference Guide, UG1144 (v2020.1) July 24, 2020.
[9] Xilinx DS962 (v1.3.1) Alveo U200 and U250 Data Center Accelerator Cards Data Sheet, May 5, 2020.
[10] Xilinx Vitis AI User Guide UG1414 (v1.3) February 3, 2021.
[11] Zhang X., Wang Y., Shi W., pcamp: Performance comparison of machine learning packages on the edges. In {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18), 2018.
[12] Xilinx, Zynq DPU v3.2 Product Guide PG338 (v3.2) July 7, 2020.
[13] Dokumentacja platformy Docker, https://docs.docker.com/ (dostęp na dzień 30.07.2021 r.).