

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Systemów Elektronicznych

Praca dyplomowa magisterska

na kierunku Elektronika
w specjalności Mikrosystemy i Systemy Elektroniczne

Implementacja sztucznych sieci neuronowych w systemach
SoC i MPSoC z wykorzystaniem akceleratorów, realizowanych
w technice HLS

Krzysztof Wasilewski

Numer albumu 265956

promotor
dr inż. Wojciech Zabołotny

WARSZAWA 2020

Implementacja sztucznych sieci neuronowych w systemach SoC i MPSoC z wykorzystaniem akceleratorów, realizowanych w technice HLS

Streszczenie

Sztuczne Sieci Neuronowe są w dzisiejszych czasach wykorzystywane w wielu zastosowaniach. Duża złożoność algorytmów Sztucznej Inteligencji wymaga dużej mocy obliczeniowej oraz odpowiednich metod optymalizacji i właściwego wyboru sprzętu. W przypadku obliczeń, które można zrównoleglić, rozsądnym wyborem są układy FPGA (ang. *Field Programmable Gate Array*).

Głównym celem pracy było zaprojektowanie i implementacja sztucznej sieci neuronowej przy wykorzystaniu systemu SoC (ang. *System on Chip*) z układem FPGA i techniki HLS (ang. *High Level Synthesis*). Użycie techniki HLS pozwala na projektowanie przy wykorzystaniu języka C, C++ lub System C i umożliwia korzystanie z wielu bibliotek zaimplementowanych w języku C i C++, co znacznie przyspiesza pracę nad projektem.

Założeniem pracy było stworzenie akceleratora, umożliwiającego osiągnięcie wzrostu wydajności algorytmu detekcji obiektów znajdujących się na obrazie w czasie rzeczywistym. W ostatnich latach można zaobserwować wielki postęp w dziedzinie komputerowego rozpoznawania obrazów (ang. *Computer Vision*), jednak większość dostępnych implementacji jest przeznaczona do uruchomienia na komputerze PC.

Słowa kluczowe: Sztuczne Sieci Neuronowe, HLS, komputerowe rozpoznawanie obrazów, FPGA

Artificial Neural Networks implementation in SoC and MPSoC systems using accelerators synthesized by HLS method

Abstract

Nowadays, Artificial Neural Networks (ANN) are used in many applications. High complexity of Artificial Intelligence algorithms requires high computing power, appropriate optimization methods and efficient hardware. In the case of computation that is easy to parallelize it is reasonable to use FPGA systems.

The main aim of the thesis was to design and implement an Artificial Neural Network algorithm using SoC (System on Chip) with FPGA system and HLS (High-Level Synthesis) method. HLS method allows to design a project using C, C++ or System C language and use lots of C libraries, which makes working on the project faster.

Assumption was made that the created accelerator will allow to achieve efficiency improvement in the real-time object detection algorithm. Recently, it is seen that huge improvement was made in the field of Computer Vision, but most of the available implementations are made to run on PC.

Keywords: Artificial Neural Networks, HLS, Computer Vision, FPGA



.....
miejscowość i data

.....
imię i nazwisko studenta
.....
numer albumu
.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płyce kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wstęp	11
1.1. Wprowadzenie	11
1.2. Wstęp teoretyczny	11
1.2.1. Model neuronu	12
1.2.2. Funkcja aktywacji	13
1.2.3. Perceptron wielowarstwowy	15
1.2.4. Sieci Głębokie	15
1.2.5. Proces uczenia sieci	16
1.2.6. Wsteczna propagacja błędu	17
1.2.7. Hiperparametry	17
1.2.8. Przeuczenie sieci	17
1.2.9. Splotowe Sieci Neuronowe	17
1.3. Implementacja Sztucznych Sieci Neuronowych w układach FPGA	17
1.3.1. Reprezentacja liczb zmiennoprzecinkowych	17
1.3.2. Uczenie Sztucznej Sieci Neuronowej w układach FPGA	17
1.4. Zastosowania Sztucznych Sieci Neuronowych	17
2. Cel i zakres pracy	19
2.1. Motywacja	19
2.2. FPGA kontra GPU	19
3. Wybór sprzętu	21
3.1. Z-turn Board	21
3.1.1. Interfejsy komunikacji	22
3.2. Kamera	22
3.3. Opcje bootowania systemu	25
4. Implementacja	27
4.1. Budowa systemu	27
4.1.1. Schemat blokowy systemu	27
4.2. Wybór narzędzi	27
4.3. Projekt systemu w środowisku Vivado	28
4.3.1. Pamięć Block RAM	28
4.4. Wykorzystanie metody HLS	29
4.5. Petalinux	30
4.6. Uczenie Sztucznej Sieci Neuronowej	30
4.7. Zbiór danych wejściowych	30
4.8. Opracowanie modelu ANN	30
4.8.1. Implementacja modelu przy użyciu narzędzia Vivado HLS	32
4.9. Testowanie systemu	32

5. Wyniki i wnioski	35
6. Podsumowanie	37
Bibliografia	39
Wykaz symboli i skrótów	40
Spis rysunków	40
Spis tabel	41

1. Wstęp

1.1. Wprowadzenie

W ostatnich latach można zaobserwować gwałtowny rozwój w dziedzinie Uczenia Maszynowego (ang. *Machine Learning*) i Sztucznej Inteligencji (ang. *Artificial Intelligence*). Coraz więcej urządzeń staje się systemami działającymi autonomicznie, bez potrzeby ingerencji człowieka. Jednym z algorytmów, który powstał już dość dawno, są Sztuczne Sieci Neuronowe (ang. *Artificial Neural Networks*). Temat Sieci Neuronowych ma długą historię rozwoju, sięgającą początku lat 40. XX wieku, jednak w ostatnich latach można zaobserwować znaczny postęp w tej dziedzinie [1]. Rozwój technologii umożliwił zastosowanie algorytmów AI w wielu aplikacjach. Działalność naukowa w kierunku Sztucznych Sieci Neuronowych spowodowała powstanie nowych modeli i architektur.

Większość algorytmów wykorzystujących Sztuczną Inteligencję wymaga dużej mocy obliczeniowej i wyboru odpowiedniego sprzętu. Często powtarzaną operacją matematyczną w przypadku algorytmu Sztucznej Sieci Neuronowej jest mnożenie macierzy. Działanie to można w łatwy sposób zrównoleglić, implementując sieć w układzie FPGA i tym samym zwiększyć efektywność algorytmu.

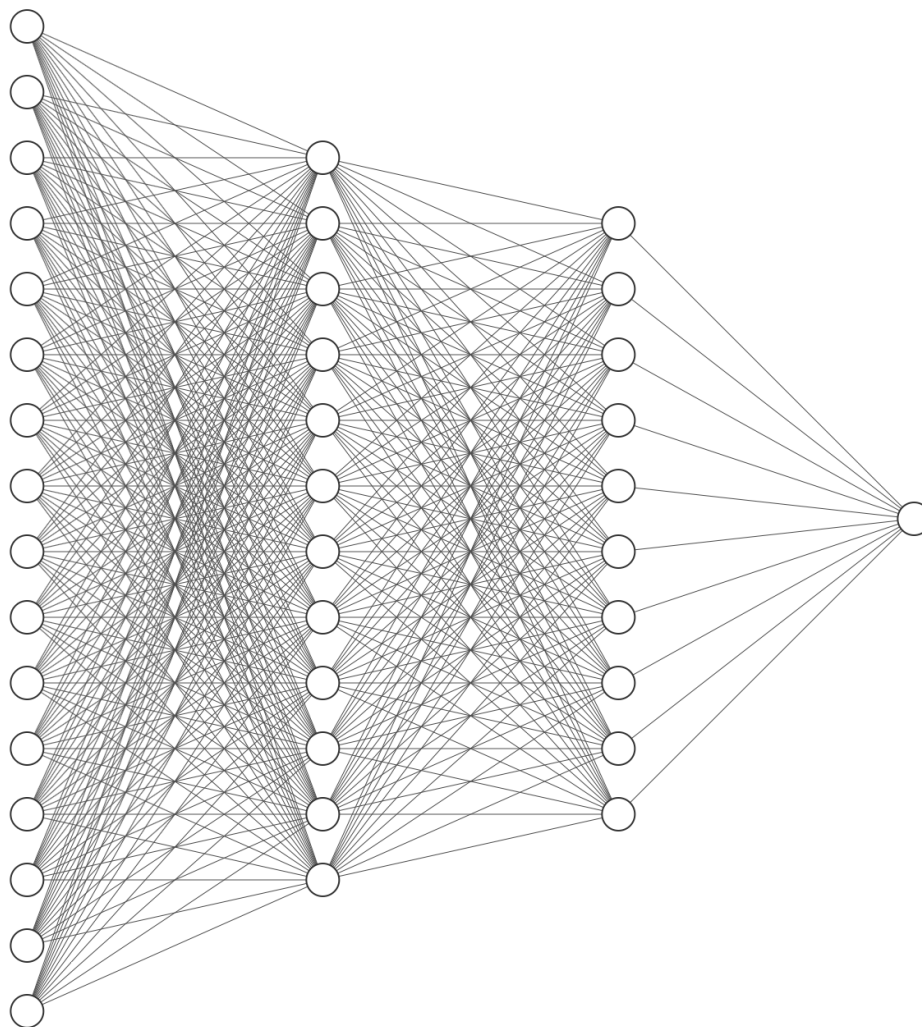
Praca nad implementacją algorytmów Sztucznej Inteligencji w większości przypadków zaczyna się od stworzenia i uruchomienia modelu. Do tego zadania często wykorzystywane są wysokopoziomowe biblioteki takie jak *Keras* lub *Theano*, które w znacznym stopniu przyspieszają proces tworzenia oprogramowania oraz ułatwiają wprowadzanie zmian w modelu sieci. Rozwój i dopracowywanie algorytmu Sztucznej Inteligencji wymaga wielu iteracji uruchamiania kodu z różnymi parametrami i właściwościami sieci. Aby w pełni wykorzystać potencjał Sztucznych Sieci Neuronowych, należy dobrać odpowiednią metody projektowania i testowania modeli.

1.2. Wstęp teoretyczny

Sieć neuronowa jest algorytmem przetwarzającym dane, inspirowanym działaniem mózgu. ANN składa się z wielu elementów przetwarzających informacje – neuronów. Model przetwarzania informacji jest inspirowany ludzkim mózgiem jednak w stosunku do komórki nerwowej, Neuron w Sztucznej Sieci Neuronowej jest bardzo uproszczony [2]. Pomimo to ANN umożliwiają rozwiązywanie złożonych problemów w wielu dziedzinach z dużą dokładnością.

Pomiędzy neuronami prowadzone są połączenia, z których każde ma swoją wagę, która zmienia wartość w trakcie uczenia sieci. Warstwę sieci, w której wszystkie wyjścia każdego z neuronów są podłączone do wszystkich neuronów w warstwie następnej, nazywa się warstwą w pełni połączoną (ang. FC — *Fully Connected*). Sieć posiadająca wszystkie warstwy w pełni połączone jest zwana siecią w pełni połączoną.

W znacznej większości sieci neuronowe budowane są w ten sposób, że dane są propagowane w kierunku od warstwy wejściowej do wyjściowej. Takie sieci nazywamy sieciami jednokierunkowymi (ang. *feedforward*). Przykładową sieć neuronową w pełni połączoną typu *feedforward* przedstawiono na Rys. 1.1.



Rysunek 1.1. Schemat w pełni połączonej jednokierunkowej Sieci Neuronowej

1.2.1. Model neuronu

Sztuczne Sieci Neuronowe to algorytm wzorowany działaniem ludzkiego mózgu i znajdujących się w nim neuronów. Model matematyczny (Rys. 1.2) pojedynczego neuronu składa się z następujących elementów[3]:

- wektora wejściowego:

$$x = (x_1, x_2, \dots, x_j)^T$$

- wektora wag przypisanych do każdego z wejść

$$w = (w_{k1}, w_{k2}, \dots, w_{kj})^T$$

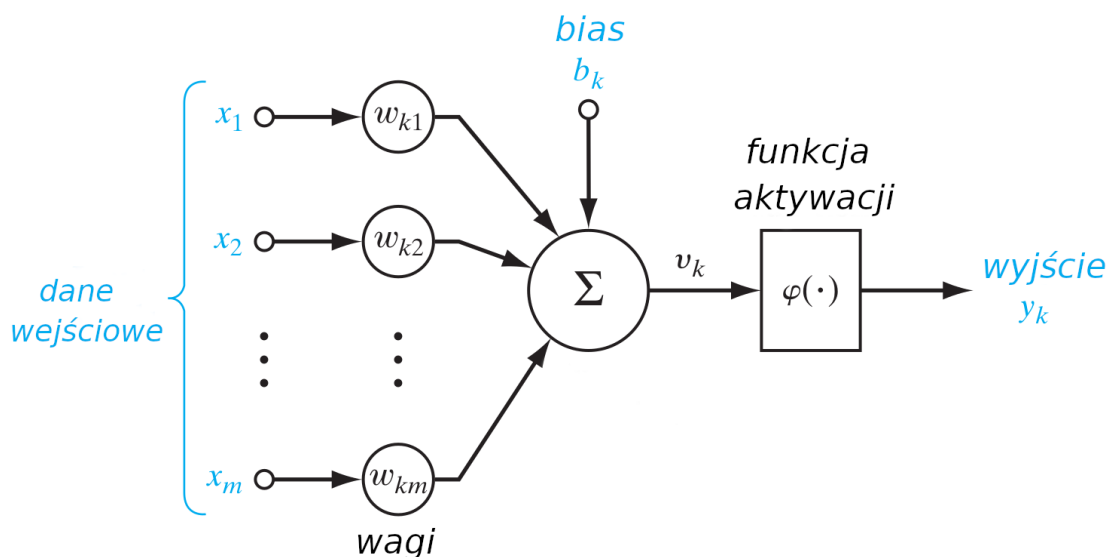
- wag obciążających (bias) b
- funkcji aktywacji $\phi(u)$
- wyjścia neuronu y .

Wagi obciążające (bias) umożliwiają kontrolowanie progu aktywacji neuronu, niezależnie od wartości wejściowych. W praktyce oznacza to przesuwanie wykresu funkcji aktywacji w kierunku w prawą lub w lewą stronę. Wyjście neuronu można policzyć, stosując następujące wzory:

$$u_k = \sum_{j=1}^N w_{kj} x_j$$

$$y_k = \phi(u_k + b_k)$$

Najprostsza Sieć Neuronowa składająca się z jednego neuronu, nazywana jest *Perceptronem*. Algorytm ten można wykorzystać do zadań klasyfikacji binarnej.



Rysunek 1.2. Model neuronu

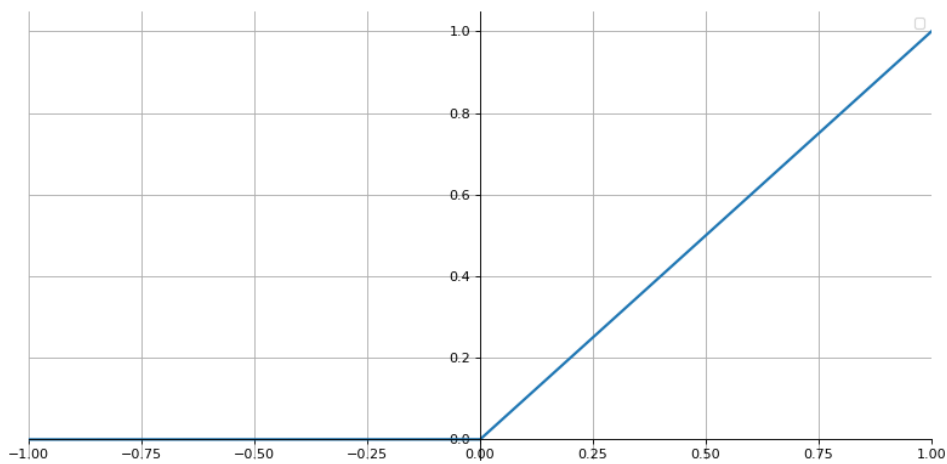
1.2.2. Funkcja aktywacji

Na działanie algorytmu znaczny wpływ może mieć dobór odpowiedniej funkcji aktywacji. Wśród najczęściej stosowanych funkcji aktywacji wyróżnia się następujące funkcje:

- funkcja ReLu (ang. *Rectified Linear Units*):

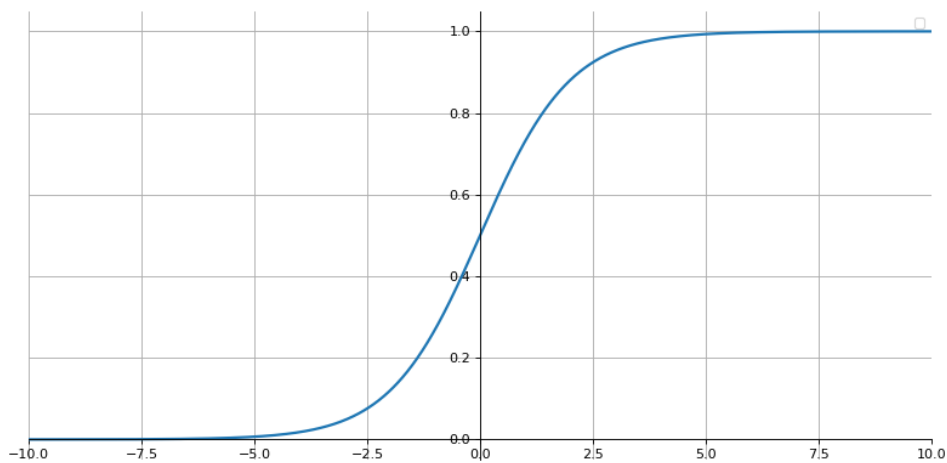
$$\phi(x) = \max(0, x)$$

- funkcja sigmoid:



Rysunek 1.3. Wykres funkcji ReLU

$$\phi(x) = \frac{a}{a + e^{-bx}}$$



Rysunek 1.4. Wykres funkcji sigmoid

- funkcja softmax (liczona dla każdego neuronu warstwy wyjściowej, $j = 1, \dots, N$):

$$\phi(x_j) = \frac{e^{x_j}}{\sum_{k=1}^N e^{x_k}}$$

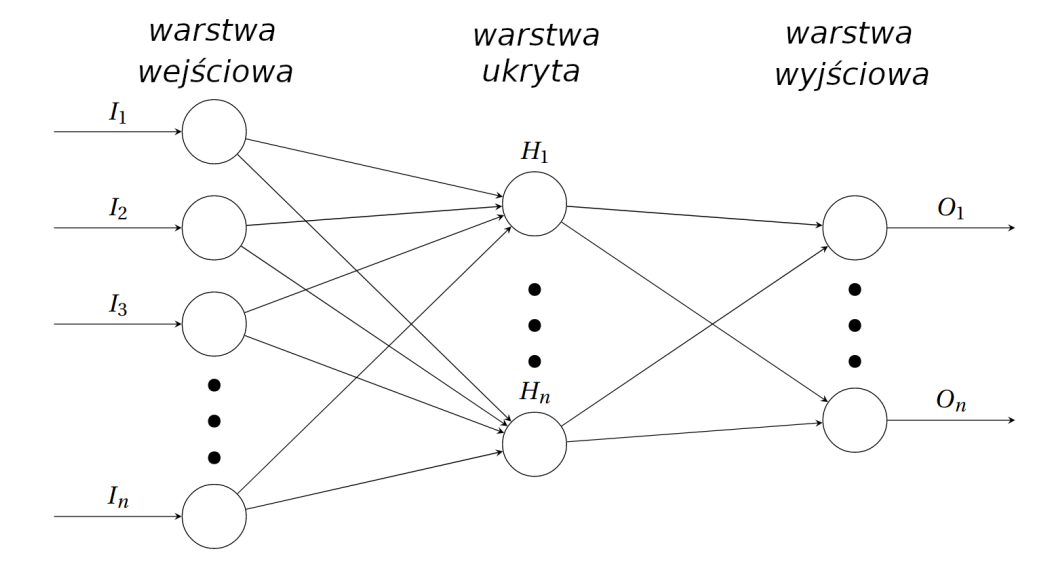
W warstwach ukrytych często używaną funkcją aktywacji jest funkcja ReLU oraz sig-

moid. Funkcję sigmoid stosuje się również w warstwach wyjściowych przy zadaniach klasyfikacji binarnej. Funkcja softmax jest używana w problemach klasyfikacji wieloklasowej (np. rozpoznawanie cyfr). Stosując funkcję softmax w warstwie wyjściowej, można traktować wektor wartości wyjść sieci jako rozkład prawdopodobieństwa wyboru danego rozwiązania.

1.2.3. Perceptron wielowarstwowy

Jednym z pierwszych modeli Sztucznych Sieci Neuronowych był Perceptron Wielowarstwowy (ang. MLP — *Multi-Layer Perceptron*), składający się z warstwy neuronów wejściowej, ukrytych i wyjściowej (Rys.1.5). Wyjście neuronów w danej warstwie staje się wejściem neuronów warstwy następnej.

Sieć Neuronowa typu MLP najczęściej posiada warstwy w pełni połączone (ang. FC – *Fully Connected*) lub inaczej Gęste (ang. *Dense*). W warstwie FC każde z wyjść jest podłączone do wszystkich wejść neuronów w warstwie następnej.



Rysunek 1.5. Model Perceptronu Wielowarstwowego

1.2.4. Sieci Głębokie

Rozwój algorytmów AI doprowadził do powstania Głębokich Sieci Neuronowych (ang. DNN — *Deep Neural Networks*), czyli takich, które posiadają wiele warstw ukrytych [4]. Algorytm Ucznia Głębokiego (ang. *Deep Learning*) umożliwia rozwiązywanie skomplikowanych problemów takich jak rozpoznawanie i klasyfikację obiektów na obrazie (Rys.1.6). Seria warstw ukrytych (ang. *hidden layers*) umożliwia ekstrakcję cech obiektów. Kolejne warstwy umożliwiają wykrywanie krawędzi, potem konturów, a na końcu całych kształtów i obiektów.



Rysunek 1.6. Model Głębokiego Uczenia sieci

1.2.5. Proces uczenia sieci

Uczenie Sztucznej Sieci Neuronowej można podzielić na nadzorowane (ang. *supervised learning*) i nienadzorowane (ang. *unsupervised learning*) oraz wersję pośrednią – uczenie częściowo nadzorowane (ang. *semi-supervised learning*). W praktyce najczęściej wykorzystywanym rodzajem uczenia jest *supervised learning*, jednak uczenie nienadzorowane jest ciekawym tematem badań i rozwoju algorytmów Sztucznych Sieci Neuronowych [5]. Proces uczenia nadzorowanego sieci polega na zwiększaniu dokładności ANN w rozwiązywaniu danego problemu. Odbywa się to poprzez iteracyjne korygowanie wartości wag i biasów tak, aby zminimalizować błąd pomiędzy rozwiązaniem wzorcowym i otrzymanym. Jeden z algorytmów zmiany wag podczas uczenia jest reguła delty (ang. *delta rule*):

$$\Delta w_{kj} = \eta e_k(n) x_j(n)$$

$$e_k(n) = d_k(n) - y_k(n)$$

gdzie $d_k(n)$ jest wzorcowym rozwiązaniem, a $y_k(n)$ rozwiązaniem otrzymanym. Wartość η oznacza współczynnik szybkości uczenia (ang. *learning rate*). Gdy jest wystarczająco mały, algorytm uczenia osiąga rozwiązanie optymalne, jednak większy współczynnik η może przyspieszyć ten proces. Wpływ na szybkość uczenia ma również wielkość mini-serii (ang. *mini-batch*) uczących.

1.2.6. Wsteczna propagacja błędu

Architekturę sieci MLP często stosuje się wraz z algorytmem wstecznej propagacji błędu (ang. *Backpropagation*), która umożliwia proces uczenia sieci. Poprzez obliczenie błędu w neuronach warstwy wyjściowej i propagacji wstecz błędu przez całą sieć pozwala dostosować wartość wagi każdego z połączeń w taki sposób, aby zminimalizować wartość funkcji kosztu.

1.2.7. Hiperparametry

W procesie uczenia Sztucznej Sieci Neuronowej bardzo istotne jest odpowiednie ustawienie parametrów sieci, zwanych hiperparametrami (ang. *hyperparameters*). Są to parametry sieci, ustawiane przez użytkownika w celu otrzymania modelu, dającego najlepsze rozwiązanie. Na wynik uczenia wpływ mają m.in.:

- liczba i rodzaj warstw sieci
- liczba neuronów w każdej z warstw
- współczynnik szybkości uczenia η
- wielkość miniserii.

1.2.8. Przeuczenie sieci

Przy używaniu Sztucznych Sieci Neuronowych w zadaniach klasyfikacji często spotykanym problemem jest przeuczenie sieci (ang. *overfitting*). Przeuczona sieć jest zbyt mocno dopasowana do danych uczących i przy walidacji poprawność klasyfikacji jest znacznie mniejsza. Jednym ze sposobów na polepszenie zdolności do generalizacji trenowanego modelu sieci jest zastosowanie *Dropoutu*, czyli losowego usuwania pewnej ustalonej liczby neuronów. Technika ta, pomimo swojej prostoty, daje bardzo dobre wyniki w przeciwdziałaniu przeuczeniu sieci.

1.2.9. Splotowe Sieci Neuronowe

Splotowe Sieci Neuronowe (ang. *CNN – Convolutional Neural Networks*) są typem sieci MLP, które znakomicie nadają się do rozpoznawania kształtów i z powodzeniem stosowane w wielu zagadnieniach klasyfikacji obrazów. Charakteryzują się dużą odpornością na rotację, skalowanie, zniekształcanie i inne zakłócenia[6]. Sieci te są uczone w trybie nadzorowanym.

1.3. Implementacja Sztucznych Sieci Neuronowych w układach FPGA

1.3.1. Reprezentacja liczb zmiennoprzecinkowych

1.3.2. Uczenie Sztucznej Sieci Neuronowej w układach FPGA

1.4. Zastosowania Sztucznych Sieci Neuronowych

2. Cel i zakres pracy

Celem pracy było zaprojektowanie i implementacja sztucznej sieci neuronowej przy wykorzystaniu systemu SoC (ang. System on Chip) i techniki HLS. Użycie metody HLS pozwala na projektowanie przy wykorzystaniu języka C, C++ lub System C, co przyspiesza pracę nad projektem. Dodatkowo HLS umożliwia korzystanie z wielu bibliotek, które pozwalają wygodnie używać funkcji, które są wykorzystywane w implementacji Sztucznych Sieci Neuronowych. Efektem pracy powinno być stworzenie akceleratora, umożliwiającego osiągnięcie wzrostu wydajności w stosunku do rozwiązań software'owych.

2.1. Motywacja

Sztuczne Sieci Neuronowe są związane z dużą ilością obliczeń, które mogą być wykonywane równolegle. Pozwala to osiągnąć krótszy czas wykonania programu, co ma duże znaczenie dla zastosowań w systemach działających w czasie rzeczywistym np. w branży *Automotive*. Aby osiągnąć przyspieszenie obliczeń, stosuje się różne metody. Jednym z najpopularniejszych obecnie sposobów na zwiększenie wydajności algorytmów AI jest wykorzystanie kart graficznych GPU (ang. *Graphics Processing Unit*). Metodą najbardziej przyspieszającą obliczenia, lecz wymagającą najdłuższego czasu projektowania i najbardziej kosztowną, jest zastosowanie specjalizowanych układów ASIC (ang. *Application-Specific Integrated Circuit*). Opcją pośrednią pomiędzy powyższymi dwoma rozwiązaniami jest zastosowanie układów FPGA. To podejście umożliwia osiągnięcie znacznego przyspieszenia wykonywania obliczeń i nie powoduje wielkiego wzrostu kosztów. Dodatkowo zastosowanie metody HLS ułatwia i minimalizuje czas tworzenia sprzętowej implementacji modelu ANN oraz wprowadzanie zmian w projekcie.

2.2. FPGA kontra GPU

W większości przypadków Sztuczne Sieci Neuronowe są projektowane, uczone i uruchamiane na procesorze z akceleratorem obliczeń w postaci GPU. Jednak układy graficzne nie zawsze są dostępne w systemach wbudowanych.

Sposobem na przyspieszenie obliczeń związanych z implementacją algorytmów ANN jest zastosowanie układów FPGA. Dodatkowym atutem po stronie układów FPGA jest mniejsze zużycie mocy.

3. Wybór sprzętu

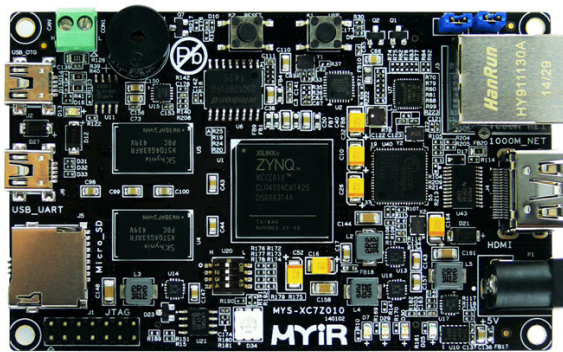
Przy wyborze płytki wzięto pod uwagę posiadane interfejsy oraz cenę sprzętu i dostępność dokumentacji. Podjęto decyzję o zastosowaniu układu FPGA firmy *Xilinx* z racji na duże wsparcie techniczne. W celu optymalizacji kosztów systemu wybrano podstawowe dwa układy z rodziny *Zynq-7000 SoC Family* XC7Z010 oraz XC7Z020. Aby mieć pewność, że parametry układu będą wystarczające, wybrano układ XC7Z020. Zgodnie z porównaniem zawartym w Tabeli 3.1, płytka Z-turn Board MYS-7Z020-C-S ma cenę sporo mniejszą od innych płytek z układem XC7Z020. Dokumentacja techniczna dotycząca płytki Z-turn Board jest dostępna na stronie producenta. Wadą płytki jest małe zaangażowanie społeczności w projekty z jej wykorzystaniem, jednak tak konkurencyjna cena zadecydowała o wyborze.

Tabela 3.1. Porównanie cen płytek z układami Zynq firmy Xilinx

Nazwa płytki	Układ SoC	Cena
Z-turn Board MYS-7Z010-C-S	XC7Z010-1CLG400C	99\$ ¹
Z-turn Board MYS-7Z020-C-S	XC7Z020-1CLG400C	119\$ ¹
Zybo Z7-10 Development Board	XC7Z010-1CLG400C	199\$ ²
Zybo Z7-20 Development Board	XC7Z020-1CLG400C	299\$ ²
ZedBoard Zynq-7000	XC7Z020-CLG484-1	449\$ ³

3.1. Z-turn Board

Z-turn Board (Rys. 3.1 jest komputerem jednopłytkowym (ang. SBC – *Single Board Computer*), opartym o układ SoC Xilinx Zynq-7020 (XC7Z020-1CLG400C), zawierającym dwurdzeniowy procesor ARM Cortex-A9 i układ FPGA Artix 7. Producentem płytki jest firma MYIR Tech Limited (ang. *Make Your Ideas Real*), dostarczająca sprzęt bazujący na procesorach ARM oraz oprogramowanie do swoich produktów[7].



Rysunek 3.1. Płytko Z-turn-Board 7020

¹ <http://www.myirtech.com/list.asp?id=502>

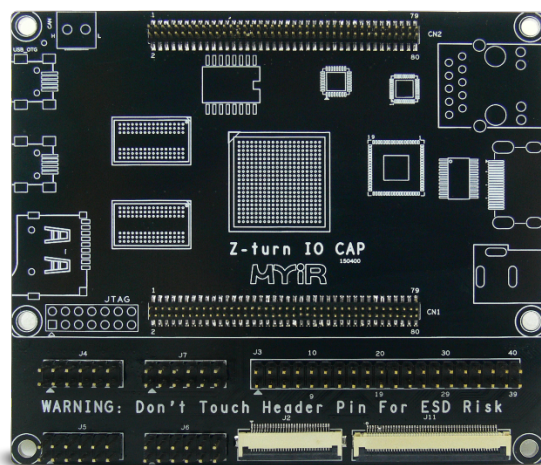
² <https://store.digilentinc.com/zybo-z7-zynq-7000-arm-fpga-soc-development-board/>

³ <https://store.digilentinc.com/zedboard-zynq-7000-arm-fpga-soc-development-board/>

Biorąc pod uwagę parametry, płytkę charakteryzuje się wysokim stosunkiem ceny do jakości, podstawowa wersja kosztuje 99\$. Dla porównania płytkę Zybo Z7-20 kosztuje 199\$. Zestawienie cen płytek zawierających układ Zynq XC7Z010 oraz XC7Z020 znajduje się w Tabeli 3.1.

3.1.1. Interfejsy komunikacji

Płytkę Z-turn posiada interfejsy UART oraz Ethernet, które zostały wykorzystane do komunikacji komputera PC z systemem przy użyciu portu szeregowego i protokołu SSH (ang. *Secure Shell*). Istnieje również możliwość podłączenia wyświetlacza bezpośrednio do płytki przy użyciu portu HDMI oraz innych peryferiów przy użyciu portu USB. Dodatkowo producent oferuje płytkę rozszerzeniową Z-turn IO-Cape (Rys. 3.2), która zawiera porty do podłączenia kamery przez protokół DVP (ang. *Digital Video Port*) oraz wyświetlacza LCD.



Rysunek 3.2. Płytkę rozszerzeniową Z-turn IO Cape

3.2. Kamera

Aby przetestować działanie systemu w czasie rzeczywistym, wykorzystano zewnętrzny moduł kamery. Przy wyborze sprzętu istotna była cena modułu, interfejs komunikacji oraz kompatybilność z SBC Z-turn Board. W przypadku wykorzystania kamery w czasie rzeczywistym bardzo ważne jest niskie opóźnienie w wysyłaniu kolejnych ramek obrazu i duża szybkość transferu obrazu.

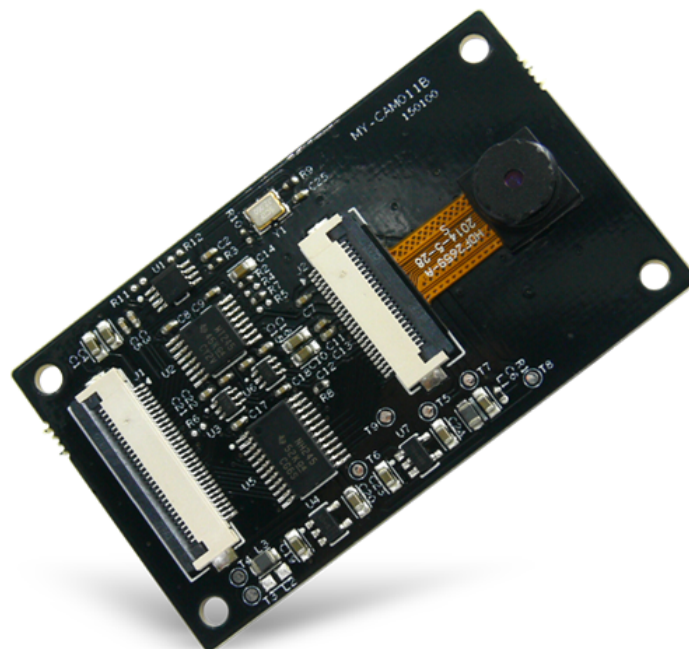
Producent płytki Z-turn Board oferuje kilka modułów kamer. Wśród nich znajdują się dwa moduły, które zostały przetestowane: MY-CAM002U USB Digital Camera Module (Rys.3.6) oraz MY-CAM011B BUS Camera Module. (Rys. 3.3). W Tabeli 3.2 przedstawiono porównanie testowanych modułów kamer.

⁴ <http://www.myirtech.com/list.asp?id=534>

⁵ <http://www.myirtech.com/list.asp?id=462>

Tabela 3.2. Porównanie testowanych modułów kamer

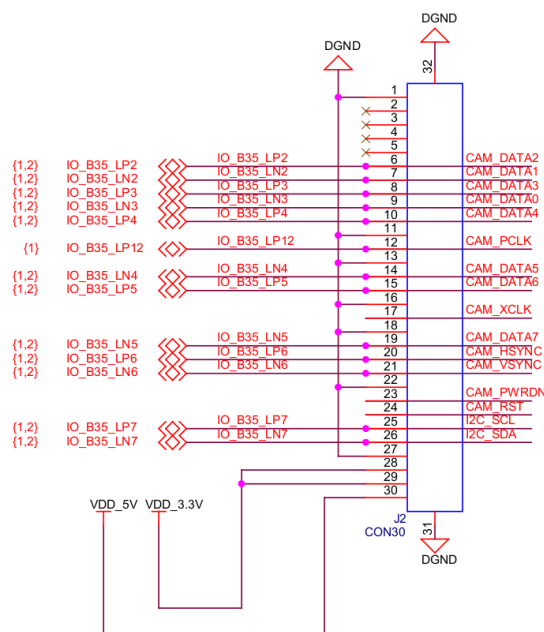
	MY-CAM011B	MY-CAM002U
Maksymalna rozdzielczość	1600x1200 pikseli	1280x800 pikseli
Pobór mocy w stanie aktywnym	224 mW	110 mW
Format wyjścia	8/10-bit RAW RGB YUV422/YCbCr422 RGB565/555 GRB422	10-bit RAW RGB
Maksymalny transfer obrazu	15 fps (1600x1200) 30 fps (800x600) 30 fps (1280x720) 24 fps (1366x768)	30 fps (1280x800) 60 fps (640x480) 30 fps (1280x720)
Cena modułu	25\$ ⁴	19\$ ⁵

**Rysunek 3.3.** Moduł kamery MY-CAM011B BUS Camera Module

Kamera MY-CAM011B zapewnia większą rozdzielczość maksymalną i posiada interfejs równoległy DVP. Skutkuje to mniejszymi opóźnieniami w wysyłaniu kolejnych ramek obrazu niż w przypadku kamery MY-CAM002U, podłączanej przez USB.

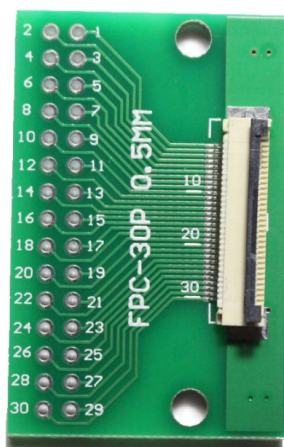
Pomimo znajdującego się na płytce IO-Cape portu interfejsu DVP, moduł kamery MY-CAM011B niestety okazał się niekompatybilny z płytką Z-turn Board. Na schemacie płytki rozszerzeniowej IO-Cape widać (Rys. 3.5), że sygnał zegara (CAM_XCLK) nie jest dołączony do żadnego portu płytki Z-Turn Board. Piny CAM_PWRDN i CAM_RST również nie są dołączone do złącza J2 od strony płytki Z-turn Board.

Dołączenie samego sygnału zegara do konektora FPC byłoby problematyczne, więc



Rysunek 3.4. Schemat portu DVP na płytce rozszerzeniowej IO-Cape

zdecydowano się na zastosowanie adaptera⁶ złącza FPC/FFC o rastrze 0,5 mm na otwory DIP raster 2,54 mm. Dzięki temu możliwe było podłączenie modułu kamery przy użyciu przewodów do złącza J3 płytki rozszerzeniowej IO-Cape [8].



Rysunek 3.5. Adapter złącza FPC/FFC o rastrze 0,5 mm na otwory DIP

⁶ <https://kamami.pl/zlaczka-ffc-fpc-zif/579387-adapter-zlaczka-fpcffc-05mm-30-pin-na-dip.html>

Po podłączeniu kamery, zainstalowaniu odpowiednich sterowników i *device-tree* oraz przeskanowaniu magistrali I^2C przy użyciu narzędzia *i2c-tools*, moduł kamery zwraca poprawnie swój adres. Jednak sterownik sensora kamery *ov2659.c* nie rozpoznaje podłączonego urządzenia. Z uwagi na ograniczony czas projektu zdecydowano się na użycie modułu kamery MY-CAM002U USB Digital Camera Module (Rys.3.6). Moduł umożliwia rejestrowanie obrazu w 30 klatkach na sekundę przy rozdzielczości 1280x640 pikseli i jest tańszy od modułu MY-CAM011B. Główną zaletą tego modułu jest łatwość podłączenia sprzętu zarówno do płytki Z-turn Board, jak i do komputera PC, co w znacznym stopniu przyspieszyło debugowanie. Kamera po podłączeniu do komputera PC działała, bez wprowadzania jakichkolwiek zmian w systemie, zarówno w dystrybucji Ubuntu, jak i w systemie Windows. Po dodaniu odpowiednich sterowników do jądra i uruchomieniu systemu Petalinux na płytce Z-turn Board, w systemie pojawiły się pliki urządzenia pod nazwą */dev/video0* oraz */dev/video1*, co umożliwiło odebranie obrazu przy pomocy interfejsu V4L2 (ang. *Video for Linux 2*) oraz OpenCV.



Rysunek 3.6. Moduł kamery MY-CAM002U USB Digital Camera Module

3.3. Opcje bootowania systemu

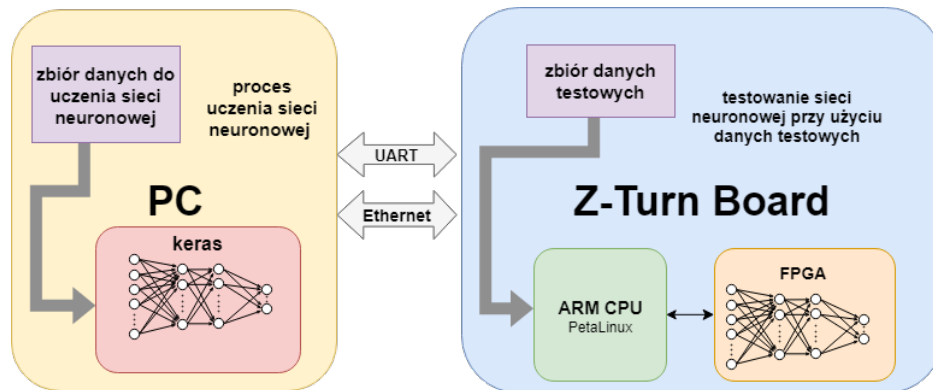
4. Implementacja

4.1. Budowa systemu

4.1.1. Schemat blokowy systemu

System składa się z dwóch głównych części (Rys. 4.1):

- aplikacja wykorzystująca pakiet keras, uruchamiana na komputerze PC
- część uruchamiana na płycie Z-Turn Board



Rysunek 4.1. Schemat blokowy systemu

4.2. Wybór narzędzi

Użycie w pracy płytki z układem Zynq determinuje użycie narzędzi wspieranych przez firmę Xilinx. Zdecydowano się na użycie najnowszej (w momencie rozpoczęcia projektu) wersji oprogramowania 2019.2. Producent zaleca[9] instalację programu Vivado na jednym z wspieranych systemów operacyjnych:

- Microsoft Windows 7 SP1 Professional (64-bit), English/Japanese
- Microsoft Windows 10.0 1809 Update; 10.0 1903 Update (64-bit), English/Japanese
- Red Hat Enterprise Workstation/Server 7.4, 7.5, and 7.6 (64-bit)
- SUSE Linux Enterprise 12.4 (64-bit)
- CentOS 7.4, 7.5, and 7.6 (64-bit)
- Ubuntu Linux 16.04.5 LTS; 16.04.6 LTS; 18.04.1 LTS; 18.04.02 LTS (64-bit)
- Amazon Linux 2 LTS (64-bit).

W projekcie wykorzystywane było również narzędzie Petalinux, które wymaga zainstalowania na maszynie z systemem operacyjnym Linux. Zgodnie z dokumentacją[10] jest to jedna z trzech dystrybucji:

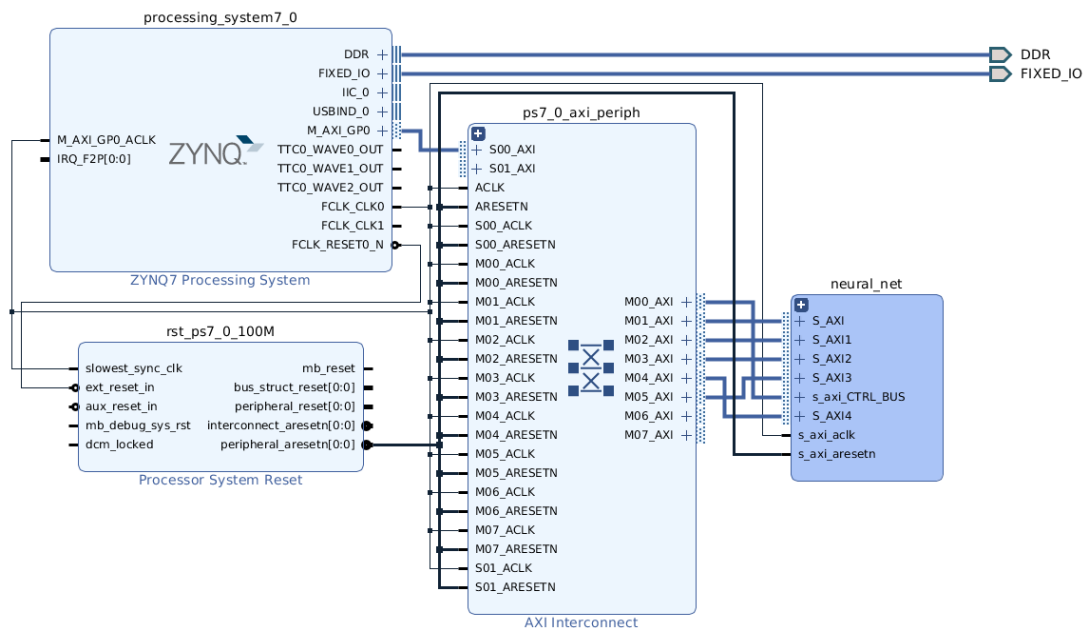
- Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6 (64-bit)
- CentOS Workstation/Server 7.4, 7.5, 7.6 (64-bit)
- Ubuntu Linux Workstation/Server 16.04.5, 16.04.6, 18.04.1, 18.04.02 (64-bit)

4. Implementacja

Aby zapewnić poprawne działanie narzędzi oraz z racji na sporą popularność i duże wsparcie społeczności wybrano dystrybucję Ubuntu 18.04.02 LTS. Przy instalacji Petalinuxa warto również zwrócić uwagę, że zalecane jest aż 100 GB wolnego miejsca na dysku twardym.

4.3. Projekt systemu w środowisku Vivado

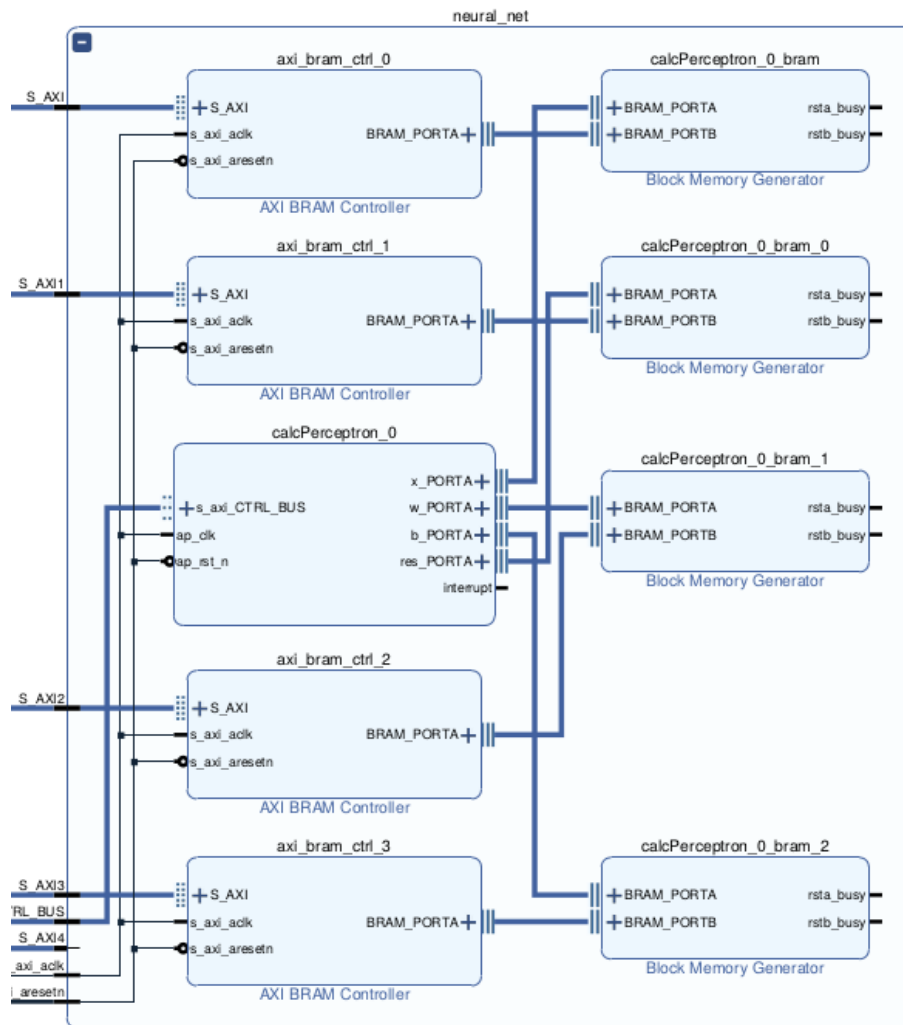
Zdecydowano się na zastosowanie bloków pamięci BRAM do komunikacji pomiędzy blokiem IP implementowanym z użyciem HLS a procesorem. Zrzut ekranu przedstawiający schemat systemu w środowisku Vivado umieszczono na Rys. 4.2. Znajduje się na nim m.in. blok IP *ZYNQ7 Processing System*, przedstawiający procesor, oraz blok *AXI Interconnect*, umożliwiający podłączenie peryferiów do procesora. Po podwójnym kliknięciu na blok procesora pojawia się okno, w którym można ustawić odpowiednią konfigurację układu.



Rysunek 4.2. Schemat systemu przedstawiony w narzędziu Vivado

4.3.1. Pamięć Block RAM

Zdecydowano się na zastosowanie bloków pamięci BRAM do komunikacji pomiędzy blokiem IP implementowanym z użyciem HLS a procesorem. Zaletą pamięci BRAM jest szybkość działania jednak ilość pamięci jaką można wykorzystać jest mocno ograniczona. Dzięki temu możliwy jest dostęp z poziomu sterownika w systemie Linux do danych przetworzonych przez blok HLS. Pamięć alokowana jest przy użyciu bloku *Block Memory Generator* i podłączana do procesora dzięki blokom *AXI BRAM Controller* (Rys. 4.3).



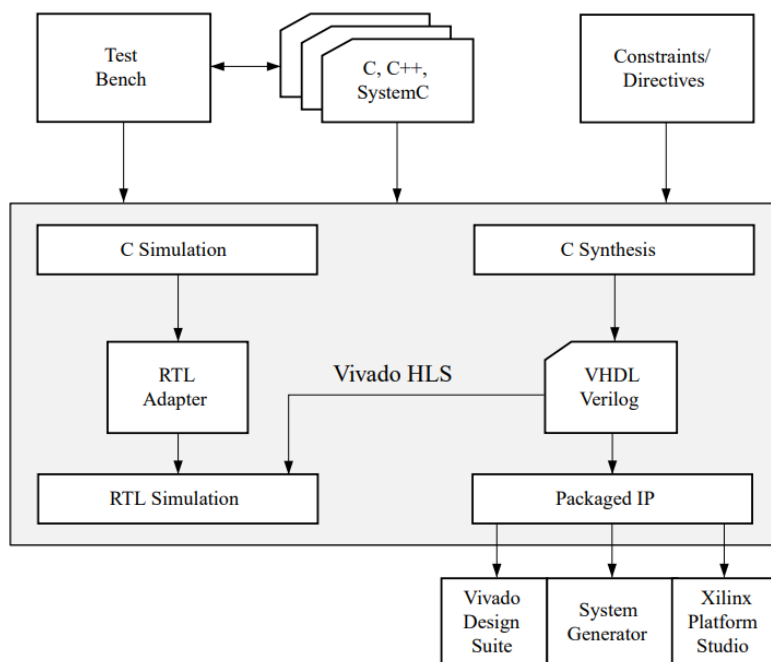
Rysunek 4.3. Szczegółowy schemat części systemu *neural_net*

4.4. Wykorzystanie metody HLS

Przy użyciu metody HLS możliwe jest stworzenie własnego bloku IP (ang. Intellectual Property), który następnie jest umieszczany w katalogu IP i można go wielokrotnie wykorzystać w projekcie RTL (ang. Register Transfer Level). Do projektu z użyciem HLS (Rys.4.4) potrzebny jest plik z algorytmem w języku C/C++ lub System C, plik testowy napisany w języku C (ang. *test bench*) oraz plik z opisem ograniczeń sprzętowych (ang. constraints). Kolejne etapy projektu z wykorzystaniem metody HLS [11]:

1. Kompilacja, wykonanie (symulacja) i debugowanie algorytmu napisanego w języku C
2. Synteza algorytmu w języku C w implementację RTL
3. Wygenerowanie raportu i analiza projektu
4. Zweryfikowanie implementacji RTL
5. Spakowanie implementacji RTL w blok IP

Zastosowanie syntezy wysokiego poziomu umożliwia przeniesienie algorytmu napi-



Rysunek 4.4. Proces projektowania przy użyciu metody HLS

sanego w języku C/C++ lub System C na implementację w układzie FPGA. Dodatkową zaletą metody HLS jest dostępność bibliotek do przetwarzania obrazów oraz ułatwiających implementację operacji matematycznych.

Przy tworzeniu nowego projektu w Vivado HLS (Rys. 4.5) trzeba podać urządzenie, na którym uruchamiany będzie blok IP. Jeśli płytką nie jest widoczna na liście, należy dodać pliki opisujące ją w odpowiednim katalogu, gdzie zostało zainstalowane narzędzie Vivado HLS.

4.5. Petalinux

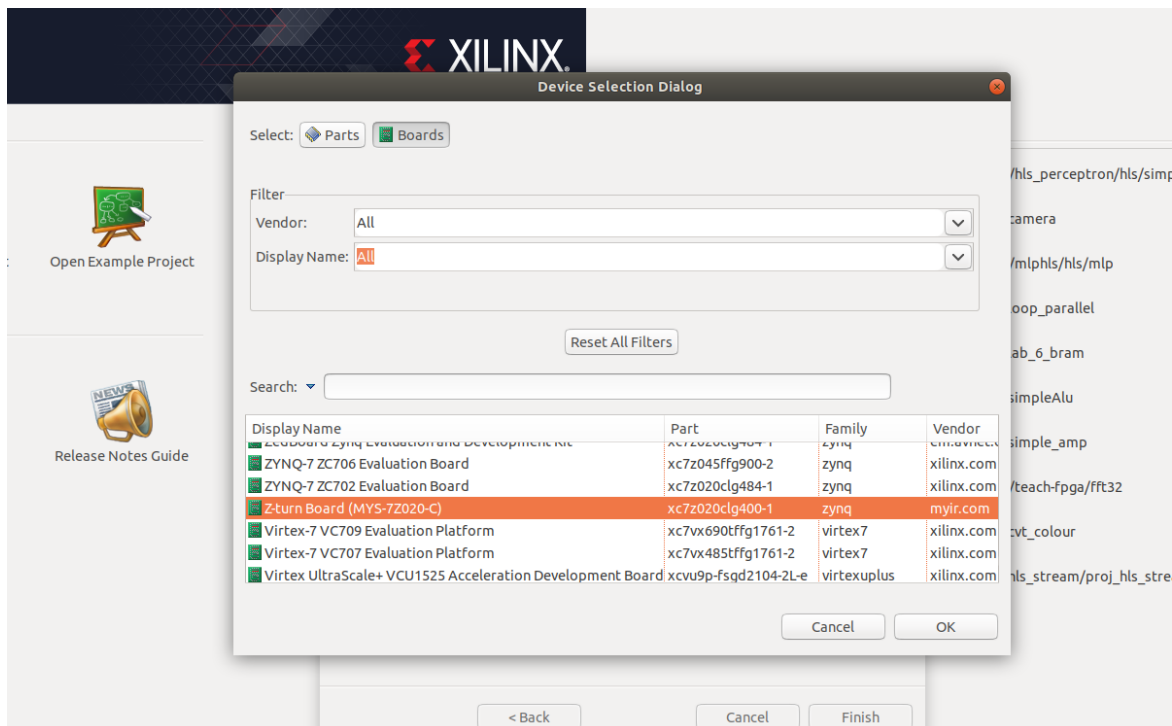
4.6. Uczenie Sztucznej Sieci Neuronowej

4.7. Zbiór danych wejściowych

W procesie uczenia oraz testowania poprawności działania modelu sztucznej sieci neuronowej wykorzystano zbiór odręcznie pisanych cyfr MNIST (ang. THE MNIST DATABASE of handwritten digits)[12]. Jest to baza 60000 obrazów do przeznaczonych do uczenia sieci oraz 10000 do walidacji. Każdy obraz przedstawia jedną cyfrę w formacie 28x28 pikseli. Fragment zbioru MNIST przedstawiono na Rys. 4.6.

4.8. Opracowanie modelu ANN

Przy projektowaniu algorytmu ANN bardzo ważnym aspektem jest odpowiednie dopasowanie modelu do danych wejściowych. Najczęściej odbywa się to poprzez wielokrotne



Rysunek 4.5. Wybór płytki podczas tworzenia projektu w Vivado HLS

testowanie systemu dla różnych parametrów sieci. W tej pracy początkowym wyborem była architektura MLP. Dane wejściowe w postaci obrazów 28x28 pikseli są konwertowane na tablicę 784 wartości typu *float*. Fragment kodu przedstawia Listing 1. Model zawiera następujące warstwy:

- warstwę wejściową (784 wejścia)
- warstwę ukrytą (16 neuronów)
- warstwę wyjściową (10 neuronów).

Listing 1. Implementacja modelu ANN MLP z jedną warstwą ukrytą

```

55 model = Sequential()
56 model.add(Flatten())
57 model.add(Dense(16, use_bias=True, activation='sigmoid'))
58 model.add(Dense(num_classes, use_bias=True, activation='sigmoid'))

```

Podczas testowania modelu użyto zbioru MNIST, który zaimportowano, wykorzystując funkcję z pakietu keras. Dokonano uczenia sieci neuronowej przy użyciu zbioru 60000 obrazów i testowania modelu, podając na wejście sieci 10000 obrazów. Osiągnięto dokładność na poziomie 93,6%. Wprowadzenie mian w postaci dodania większej ilości neuronów w warstwie ukrytej nie powodowały zwiększenia dokładności, więc podjęto decyzję o wykorzystaniu tego modelu w dalszej części projektu.



Rysunek 4.6. Fragment zbioru odręcznie pisanych cyfr MNIST

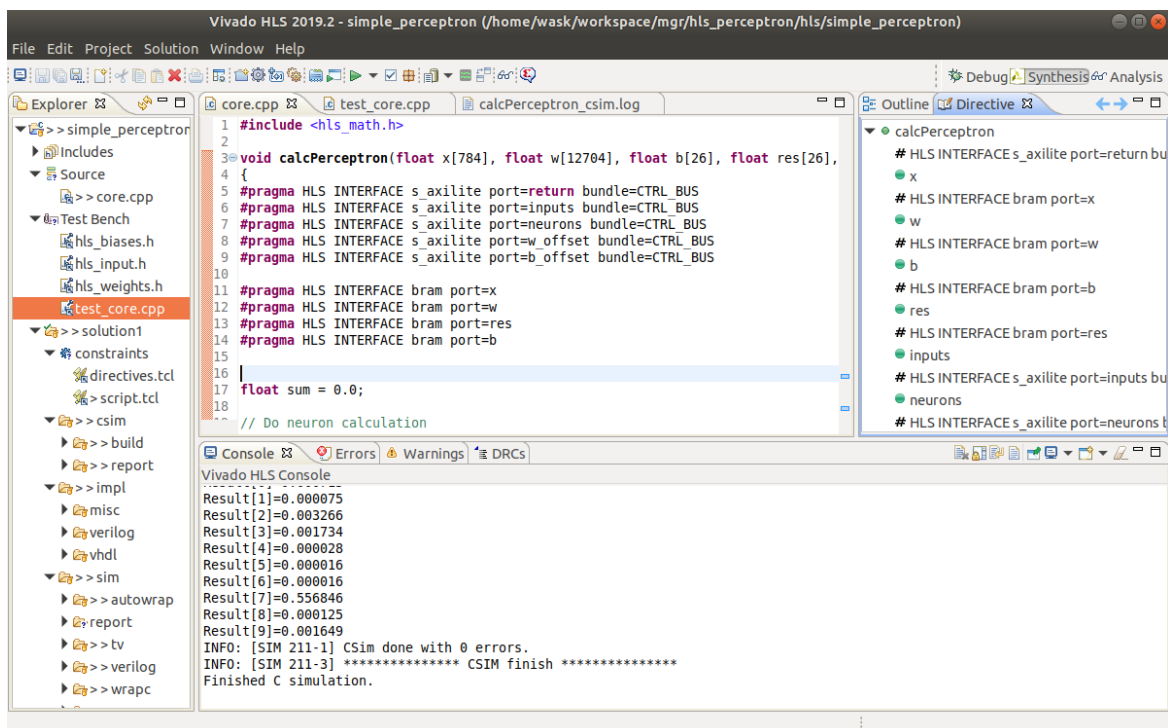
4.8.1. Implementacja modelu przy użyciu narzędzia Vivado HLS

Korzystając z narzędzia Vivado HLS, napisano program w języku C++ implementujący zaprojektowany wcześniej model sieci. W procesie uczenia ustalono wartości wag i biasów. Dwa główne pliki projektu w narzędziu Vivado HLS to `core.cpp`, zawierający implementację algorytmu ANN oraz `test_core.cpp`, który służy do przetestowania algorytmu po przeprowadzeniu syntezy (ang. test bench). Pliki zawierające wartości wejściowe, a także wagi i biasy są importowane do programu `test_core.cpp`.

Pierwszym etapem jest Symulacja C, która jest wstępną weryfikacją poprawności algorytmu. Do przeprowadzenia symulacji wykorzystano dane wyeksportowane przy użyciu skryptu `keras2fpga.py` i zapisane w plikach `hls_biases.h`, `hls_weights.h` oraz `hls_input.h`. Wynikiem symulacji jest plik `.log`, w którym można znaleźć informacje o tym, jak przebiegało wywołanie testowanych funkcji. Symulację w języku C wykonuje się dużo szybciej niż późniejszą symulację RTL, więc stosuje się ją jako pierwszy etap przed podjęciem dalszych kroków, które zajmują więcej czasu.

Następnie wykonywana jest synteza oraz kosymulacja, umożliwiająca weryfikację poprawności syntezy. Ponadto narzędzie generuje raport, który przedstawia informacje na temat zużycia zasobów i opóźnień czasowych. Po prawidłowym przeprowadzeniu kosymulacji należy użyć opcji *Export RTL*, co umożliwia dodanie nowego bloku IP do projektu w narzędziu Vivado.

4.9. Testowanie systemu



Rysunek 4.7. Wynik poprawnie przeprowadzonej symulacji w Vivado HLS

5. Wyniki i wnioski

6. Podsumowanie

Bibliografia

- [1] D. Kriesel, *A Brief Introduction to Neural Networks*. 2007. adr.: <http://www.dkriesel.com>.
- [2] R. Tadeusiewicz, *Sieci neuronowe*. 1993.
- [3] A. Omondi i J. Rajapakse, "FPGA Implementations of Neural Networks", 2006.
- [4] I. Goodfellow, Y. Bengio i A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [5] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, 2015, t. 2018.
- [6] S. S. Haykin i in., *Neural networks and learning machines/Simon Haykin*. 2009.
- [7] MYIR, *Company Profile*, Dostęp zdalny (22.09.2020): <http://www.myirtech.com/company.asp>, 2020.
- [8] *Z-turn IO Cape Schematic*, 2015.
- [9] Xilinx, *Vivado Design Suite User Guide (UG973 (v2019.2))*, 2019.
- [10] *PetaLinux Tools Documentation Reference Guide (UG1144 (v2019.2))*, 2019.
- [11] *Vivado Design Suite User Guide High-Level Synthesis (UG871 (v2019.2))*, 2020.
- [12] Y. LeCun i C. Cortes, "MNIST handwritten digit database", 2010. adr.: <http://yann.lecun.com/exdb/mnist/>.

Wykaz symboli i skrótów

ANN – ang. *Artificial Neural Network*
ASIC – ang. *Application-Specific Integrated Circuit*
DVP – ang. *Digital Video Port*
FPGA – ang. *Field Programmable Gate Array*
GPU – ang. *Graphics Processing Unit*
HLS – ang. *High Level Synthesis*
MPSoC – ang. *Multi-Processor System-on-Chip*
SBC – ang. *Single Board Computer*
SoC – ang. *System on Chip*
SSH – ang. *Secure Shell*
V4L2 – ang. *Video for Linux 2*

Spis rysunków

1.1. Schemat w pełni połączonej jednokierunkowej Sieci Neuronowej	12
1.2. Model neuronu	13
1.3. Wykres funkcji ReLU	14
1.4. Wykres funkcji sigmoid	14
1.5. Model Perceptronu Wielowarstwowego	15
1.6. Model Głębokiego Uczenia sieci	16
3.1. Płytki Z-turn-Board 7020	21
3.2. Płytki rozszerzeniowa Z-turn IO Cape	22
3.3. Moduł kamery MY-CAM011B BUS Camera Module	23
3.4. Schemat portu DVP na płycie rozszerzeniowej IO-Cape	24
3.5. Adapter złącza FPC/FFC o rastrze 0,5 mm na otwory DIP	24
3.6. Moduł kamery MY-CAM002U USB Digital Camera Module	25
4.1. Schemat blokowy systemu	27
4.2. Schemat systemu przedstawiony w narzędziu Vivado	28
4.3. Szczegółowy schemat części systemu <i>neural_net</i>	29
4.4. Proces projektowania przy użyciu metody HLS	30
4.5. Wybór płytki podczas tworzenia projektu w Vivado HLS	31
4.6. Fragment zbioru odręcznie pisanych cyfr MNIST	32
4.7. Wynik poprawnie przeprowadzonej symulacji w Vivado HLS	33

Spis tabel

3.1. Porównanie cen płytek z układami Zynq firmy Xilinx	21
-------------------------------------------------------------------	----

3.2. Porównanie testowanych modułów kamer	23
-----------------------------------------------------	----