

Detekcja koloru skóry z wykorzystaniem sieci neuronowych.

Krzysztof Odrzywołek,
Michał Pietruszka, Rafał Włodarz

Copyright © 2015
Krzysztof Odrzywołek, Michał Pietruszka, Rafał Włodarz

PUBLISHED BY AGH

First printing, January 2015

Contents

1	Wstęp	5
1.1	Opis projektu	5
2	Realizacja projektu	7
2.1	Dobór topologii sieci	7
2.2	Implementacja sieci neuronowej w MATLAB	8
2.3	Realizacja algorytmu w środowisku ISE	10
3	Efekt działania	11
3.1	Efekt działania	11
4	Podsumowanie	15
4.1	Wnioski	15
5	Dodatek	17
5.1	Opis plików	17

1 — Wstęp

1.1 Opis projektu

Celem projektu było zrealizowanie algorytmu opartego na sieciach neuronowych którego zadanie polegało na rozpoznawaniu skóry na podstawie obrazu z kamery w czasie rzeczywistym. Realizacja tego zagadnienia miała na celu pokazanie słuszności zastosowań tego typu algorytmów do rozpoznawania skóry oraz poprawności ich działania.

2 — Realizacja projektu

Projekt została podzielony na kilka etapów w których realizacja każdego jest zależna od powodzenia poprzedniego. Najważniejszymi etapami okazały się: teoretyczne przygotowanie do zagadnienia wraz z doborem topologii sieci, implementacja algorytmu w środowisku MATLAB wraz z algorytmem uczenia sieci oraz ostateczna jego realizacja na układzie fpga.

2.1 Dobór topologii sieci

Wybór odpowiedniej topologii sieci wymagał znajomości teoretycznej zagadnień związanych z algorytmami opartymi na sieciach neuronowych, wykorzystania własnego doświadczenia związanego z tymi zagadnieniami oraz realizacji algorytmów na układach fpga. Ze względu na niewystarczającą wiedzę w tym zakresie, początkowa faza projektu opierała się na poszukiwaniu informacji o zagadnieniach zbliżonych do tematu projektu. Okazało się, iż bardzo zbliżony projekt został już zrealizowany i opisany w <wstawić bibliogarrii>. Artykuł przedstawia dokładne etapy realizacji projektu którego celem było rozpoznawanie twarzy. Zawierał on również dokładny opis topologii sieci oraz sposób przygotowania danych do przetwarzania.

Informacje te okazały się bardzo pomocne, pozwoliły one rozpoczęć doświadczalny dobór topologii sieci, która pozwoli na poprawne przetwarzanie danych wejściowych. Podstawą do przeprowadzania takich eksperymentów było zdobycie bazy (została ona załączona wraz z plikami projektu), która umożliwi naukę sieci. Baza posiada zbiór danych przygotowany w formie 4 wartości odpowiadających poszczególnym wartośćom RGB oraz oczekiwanej wartością wyjściu z sieci, którego wartości zamykają się w zbiorze 0,1. Świadczy ono o tym czy na podstawie określonego wejścia powinna zostać wykryta skóra.

Liczna baza danych wypełniona takimi rekordami pozwoliła na przeprowadzanie pierwszych doświadczeń. Na podstawie zdobytych informacji dowiedzieliśmy się, iż jakość wykrywania pikseli przedstawiających skórę znacznie wzrasta jeśli na wejście sieci oprócz sygnałów RGB podawane zostaną również CbCr(wartości sygnałów z modelu przestrzeni kolorów YCbCr) oraz SV(wartości sygnałów z przestrzeni barw HSV). Aby otrzymać pełny wektor danych wejściowych zaimplementowano algorytm w postaci skryptu w programie MATLAB, który na podstawie wartości RGB wyznaczał pozostałe parametry. Tak przygotowane dane wejściowe podawano na wejście wbudowanej sieci neuronowej w środowisku MATLAB, a następnie na podstawie przykładowego zdjęcia określano czy sieć poprawnie rozpoznaje skórę na przedstawionym obrazie. Ze względu na łatwą konfigurację topologii sieci sposób ten okazał umożliwił szybkie

przetestowanie wielu wariantów oraz wybranie najlepszego w interesującym nas zakresie.

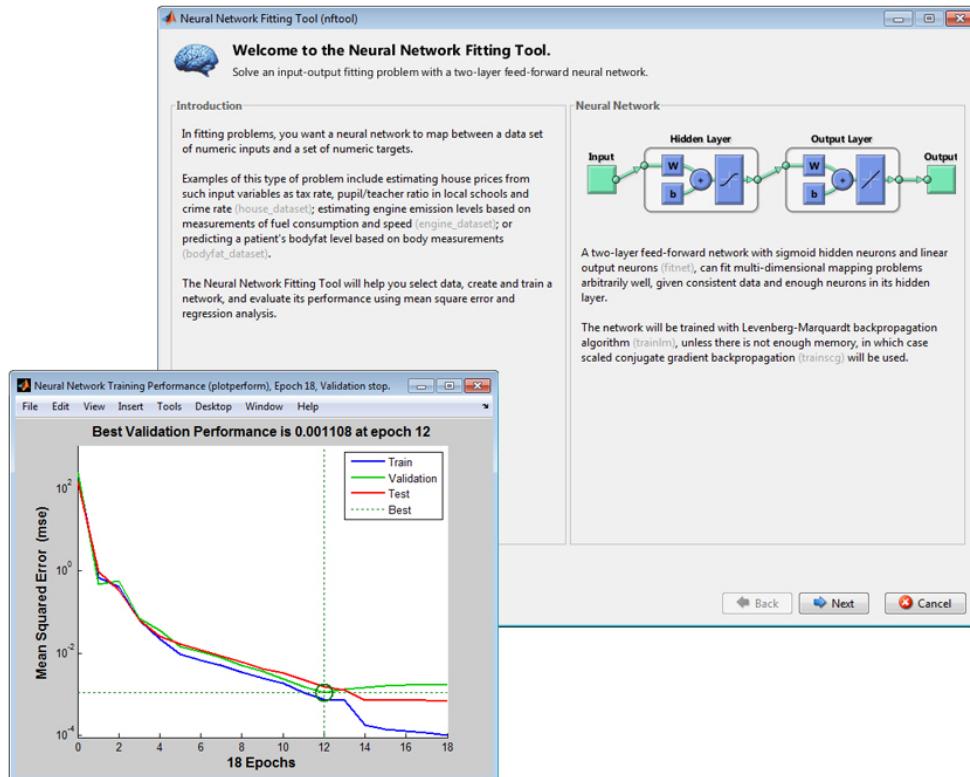


Figure 2.1: Narzędzie środowiska MATLAB służące do testów sieci neuronowych

Najlepsze rezultaty uzyskaliśmy dla sieci posiadającej 7 wejść, 13 neuronów w warstwie ukrytej oraz jednym w warstwie wyjściowej. Niestety narzędzie to uniemożliwia konfigurowanie funkcji wyjścia z neuronu. Na podstawie doświadczeń, zaimplementowanych skryptów oraz zdobytej wiedzy udało się rozpocząć prace nad kolejnym etapem.

2.2 Implementacja sieci neuronowej w MATLAB

W tej fazie skoncentrowano się na implementacji sieci spełniającej nasze wymagania. Ze względu na prostotę działania wybraliśmy sieci propagation o strukturze wybranej w poprzednim etapie czyli. 7 wejść, 13 neuronów w warstwie ukrytej oraz 1 wyjściu. Model takiej sieci zakłada prosty przepływ informacji między kolejnymi warstwami, neurony w warstwie wejściowej przekazują wartość na wejścia neuronów w warstwie ukrytej. Następnie jest ona przemnażana przez wartość wagi przypisaną do wejścia na którym się pojawiła. Wartości te są sumowane w obrębie każdego z neuronów a następnie przekazywane na wyjście, które na otrzymanej wartości wywołuje funkcje przejścia. Jednoznacznie określona wartość wyjścia przekazywana jest do następnej warstwy (przepływ danych przez sieć przedstawiono na rysunku 2.2).

W tym przypadku stosowanie sieci o bardziej złożonych strukturach wydaje się zbędny i sprzeczny z intuicją. Sieci te doskonale rozpoznają cechy wartości podanych na wejście co powoduje poprawne zachowanie dla sygnałów zbliżonych z którymi sieć nie miała wcześniejszej styczności.

Również ważną cechą sieci jest dobór funkcja przejścia dla neuronów. W tym przypadku została zastosowana funkcja $\tanh(x)$ ze względu na jej charakterystykę, która akcentuje wartości brzegowe, które w naszym przypadku odpowiadają wykryciu skóry przez sieć.

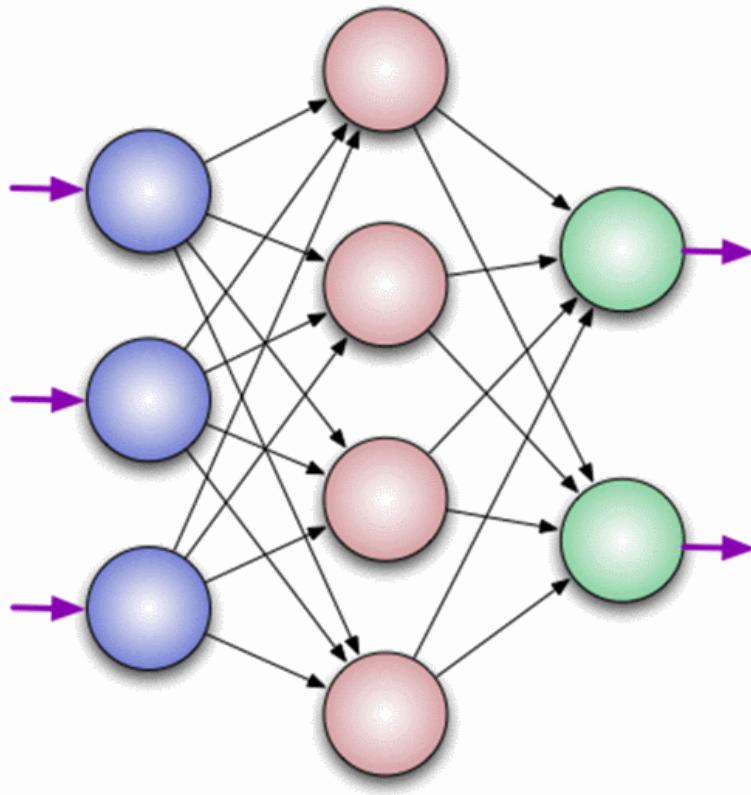


Figure 2.2: Sieć neuronowa typu propagation.

Po implementacji sieci przystąpiono do adaptacji algorytmu uczenia. W sieciach typu propagation najczęściej stosowany jest algorytm backpropagation, który cechuje się prostą lecz również długim czasem uczenia sieci oraz żmudnymi obliczeniami. Polega on na podawaniu na wejście sieci wektor odpowiednich wartości, a następnie wyznaczaniu różnicy wyjścia sieci z wartością oczekiwana. Różnica ta jest przetwarzana wstecznie przez sieć wraz z wyznaczaniem gradientów błędów poszczególnych neuronów. Na ich podstawie korygowane są wagi każdego z wejść neuronów co powoduje uczenie sieci.

Wszystkie skrypty utworzone na tym etapie oraz obszerna baza danych umożliwiły rozpoczęcie prac nad uczeniem sieci. Wywołanie odpowiednich skryptów rozpoczęło proces uczenia sieci, którego czas trwania zależał od ilości wykonania powyżej opisanej operacji dla każdego z wierszy. Optymalne wyniki otrzymywane były dla około 100 iteracji. Warto również wspomnieć, iż początkowo baza zawierała dane posortowane ze względu na wektor oczekiwanej wyjścia co zdecydowanie utrudniało proces uczenia. Rozwiązano ten problem przez przemieszanie wierszy w bazie.

Na rysunku 2.3 przedstawiono przykładowe przetworzenie małego obrazka na którym testowano sieć po zakończeniu uczenia. Na jego podstawie wstępnie oceniano powodzenie procesu uczenia sieci. Jak widać na rysunku udało się osiągnąć zadowalającą jakość. Warto zauważyć, iż zdjęcie testowe zostało wykonane przez autorów projektu więc wartości RGB poszczególnych pikseli są niezależne względem tych umieszczonej w bazie.

W kolejnej fazie zmodyfikowano system reprezentacji liczb na których bazuje działanie sieci neuronowych. Zastosowano zmienne typu `fixed_point`, które pozwalają na dokładne określenie ilości bitów przeznaczonych na reprezentacje poszczególnej liczby. Do konwersji z typu `double` do typu `fixed_point` została stworzona funkcja w środowisku MATLAB. Jako



Figure 2.3: Przykładowy wynik testu działania sieci po uczeniu.

argument przyjmuje ona zmienną typu double, a następnie na podstawie sztywno określonej konfiguracji funkcji tworzy zmienną typu fixed_point, która reprezentuje wartość argumentu lub wartość najbliższą, która jest możliwa do zaprezentowania. Zabieg ten miał na celu dokładne odwzorowanie działania sieci na układzie fpga. Czas przetwarzania obrazu testowego przez sieć, a w szczególności sam proces uczenia wydłużył się kilkakrotnie ze względu na liczne konwersje.

Po przeprowadzaniu ponownego procesu uczenia oraz sprawdzeniu, że modyfikacje te nie zaburzyły w znaczny sposób działania sieci zakończono proces tworzenia odwzorowania sieci w środowisku MATLAB.

Podczas realizacji tej fazy zorientowano się, iż właściwa implementacja na układzie fpga nie wymaga implementowania procesu uczenia w znacznym stopniu upraszcza implementację. Uzyskane dotychczas informacje na temat topologii sieci, funkcji przejścia, wartości poszczególnych wag dla każdego z neuronów oraz optymalnej ilości bitów przeznaczonych do reprezentacji tych wartości pozwoliły na przystąpienie do ostatniego etapu.

2.3 Realizacja algorytmu w środowisku ISE

3 — Efekt działania

3.1 Efekt działania

W tym rozdziale zostały zebrane zdjęcia zrobione podczas testowania algorytmu. Niestety z powodu braku odpowiedniego sprzętu zdjęcia te zostały zrobione w słabej jakości.



Figure 3.1: Struktura katalogów

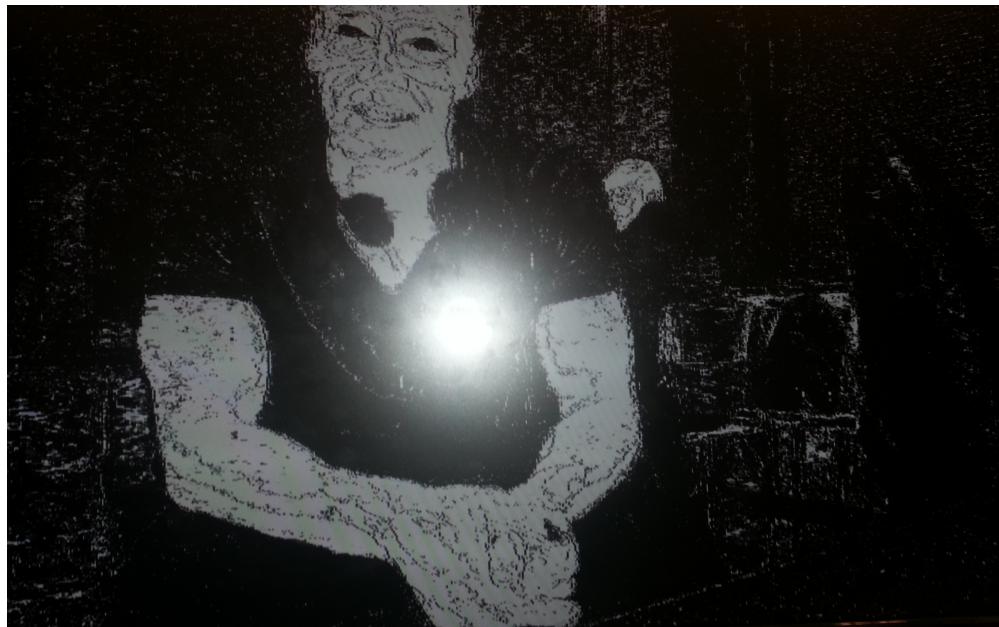


Figure 3.2: Struktura katalogów



Figure 3.3: Struktura katalogów

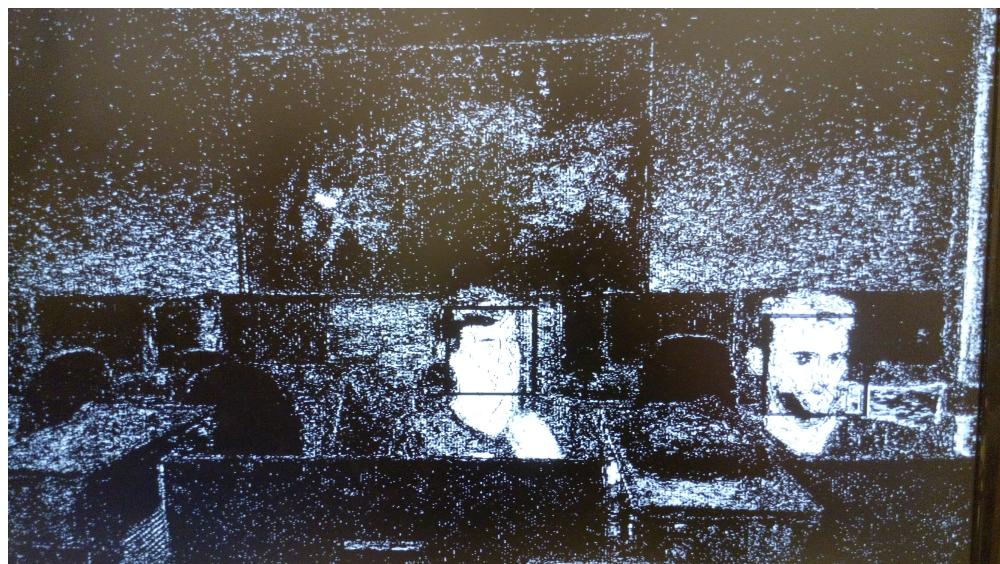


Figure 3.4: Struktura katalogów

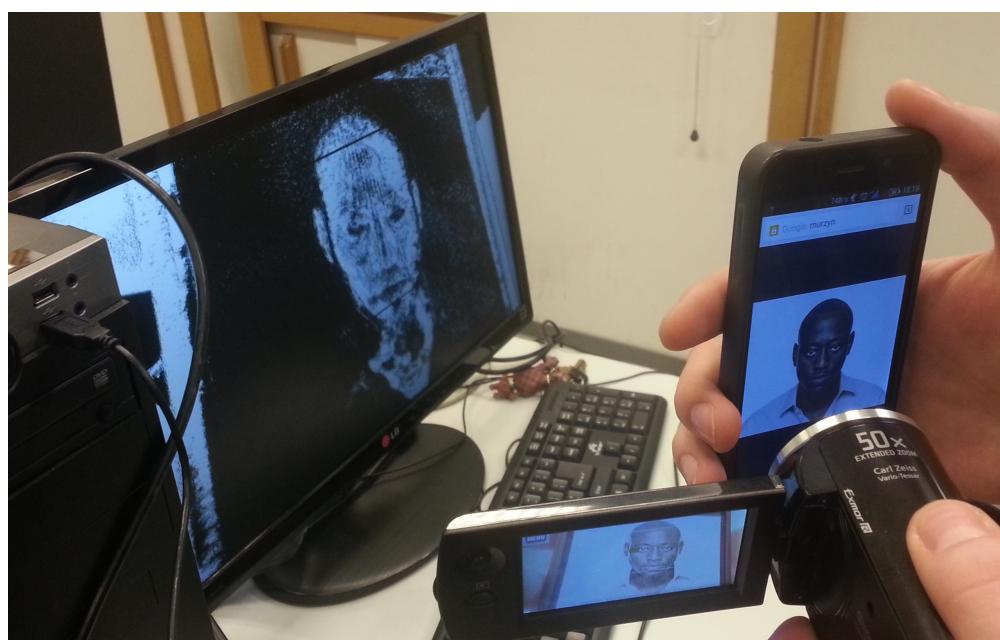


Figure 3.5: Struktura katalogów

4 — Podsumowanie

4.1 Wnioski

Udało się zrealizować wszystkie założenia projektu. Nie mniej jednak warto mieć na uwadze, iż udało się udowodnić jedynie poprawność działania algorytmów opartych na sieciach neuronowych. Jakość obrazu oraz dokładność rozpoznawania może zostać poprawiona przy użyciu filtrów medianowych oraz nie wyklucza się, iż zmiana topologii może spowodować lepsze rezultaty. Zmiany te powinny zostać wprowadzone w pierwszym etapie dalszych prac nad projektem. Z powodu nie wystarczającej ilości czasu kolejne prace nad udoskonaleniem algorytmu nie zostały zrealizowane.

5 — Dodatek

5.1 Opis plików

Wszystkie pliki utworzone podczas pracy nad projektem zostały wysłane razem ze sprawozdaniem oraz są dostępne pod adresem <http://github.com/Kyhu/wsw-neuro>, gdzie mogą być w dalszym ciągu aktualizowane. Poniżej krótko opisano strukturę projektu.

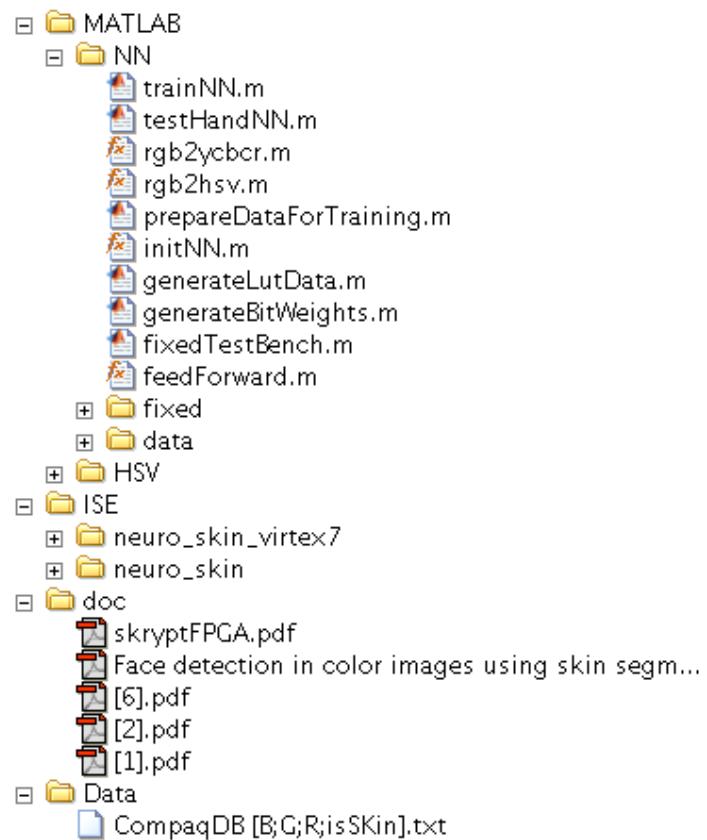


Figure 5.1: Struktura katalogów

- MATLAB - folder zawierający wszystkie utworzone pliki środowiska MATLAB.
 - NN
 - trainNN.m - skrypt trenujący model sieci neuronowej.
 - testHandNN.m - skrypt testujący sieć na przykładowym obrazie.
 - generateLutData.m - skrypt generujący dane dla LUTa w module języka verilog.
 - generateBitWeights.m - skrypt generujący linijki kodu veriloga odpowiadające za przechowywanie wartości wag sieci neuronowych.
 - fixedTestBench - test bench dla skryptów feedForward oraz fix_feedforward.
 - feedForward.m - funkcja symulująca moduł neural_networks.v na liczbach zmiennoprzecinkowych
 - fix_feedForward.m - funkcja symulująca moduł neura_networks.v na liczbach stałoprzecinkowych
 - data - folder zawierający zapisane modele sieci Neuronowych (wagi) oraz testowe obrazy.
 - HSV - folder zawierający skrypty wykorzystane do testowania modułu rgb2 hsv zaimplementowanego w verilogu.
- ISE - folder zawierający projekty realizowane w środowisku ISE.
 - neuro_skin - projekt finalny realizowany na kartę Spartan-6.
 - neuro_skin_virtex7 - projekt finalny przerzucony na tor wizyjny karty virtex7 (z projektu pbas)
- doc - folder zawierający wykorzystane artykuły naukowe. Artykuł dostarczony przez prowadzącego jest opisany pełnym tytułem, pozostałe są oznaczone odnosząc się do jego bibliografii.
- Data - folder zawierający bazę danych wykorzystaną do trenowania sieci neuronowej.
(Compaq Database - https://archive.ics.uci.edu/ml/machine-learning-databases/00229/Skin_NonSkin.txt)

Bibliography

- [1] Xilinx Inc. Spartan-6 FPGA Configurable Logic Block - User Guide. 2010.
- [2] Xilinx Inc. Spartan-6 FPGA DSP48A1 Slice User Guide. 2010.
- [3] Xilinx Inc. Spartan-6 FPGA GTP Transceivers User Guide. 2010.
- [4] Xilinx Inc. Spartan-6 FPGA Integrated Endpoint Block for PCI Express User Guide. 2010.
- [5] Xilinx Inc. Spartan-6 FPGA Integrated Endpoint Block for PCI Express User Guide (AXI). 2010.
- [6] Xilinx Inc. Spartan-6 FPGA Memory Controller User Guide. 2010.
- [7] Xilinx Inc. Spartan-6 FPGA Block RAM Resources User Guide. 2011.
- [8] Xilinx Inc. Spartan-6 FPGA Clocking Resources User Guide. 2011.
- [9] Xilinx Inc. Spartan-6 FPGA SelectIO Resources User Guide. 2014.