WYŻSZA SZKOŁA INFORMATYKI I ZARZĄDZANIA „COPERNICUS" WE WROCŁAWIU

WYDZIAŁ INFORMATYKI, ADMINISTRACJI I FIZJOTERAPII

Kierunek studiów: **Informatyka niestacjonarna**
Poziom studiów: **Studia pierwszego stopnia - inżynierskie**
Semestr: **4**
Specjalność: **Inżynieria Systemów Informatycznych**

SPRAWOZDANIE

Volodymyr Gorbachov

Fizyczna funkcja nieklonowalna (PUF) z Microblaze w Xilinx Vivado

Physical unclonable function (PUF) with Microblaze in Xilinx Vivado

Ocena pracy: .....
(ocena pracy dyplomowej, data, podpis promotora)

.....
(pieczątka uczelni)

Prowadzący:

Dr inż. Grzegorz Debita

WROCŁAW 2020

Spis treści

1. Wstęp

Nowoczesne układy FPGA zwierają wielkie tablice elementów konfigurowalnych, oferują bardzo elastyczny sposób tworzenia połączeń wewnętrznych, a ponadto realizują wszelkie działania z bardzo dużą szybkością. Opracowanie języka opisu sprzętu (HDL) ułatwiło natomiast konstruowanie i testowanie systemów cyfrowych. Ze względu na wymienione cechy układów FPGA, idea używania ich jako platformy do budowy w pełni równolegle działających układów logicznych jest wielce obiecująca. Rozwój domeny układów programowalnych dokonał się w dużym stopniu dzięki nowoczesnej technice komputerowej. Powstało wiele różnych, ciekawych i ambitnych projektów implementacyjnych bazujących na układach FPGA. Dzięki równoległości działań możliwa jest już realizacja techniczna w pełni działających neurokomputerów, co było niegdyś bardzo trudne i kosztowne.

1.1 Wprowadzenie

FPGA (Field Programmable Gate Array) są cyfrowymi układami logicznymi, które mogą zostać zaprogramowane w sposób określony przez użytkownika, aby spełniać dedykowane funkcje. Pozwala to na dostosowanie konfiguracji układu do określonych potrzeb i realizowanych zadań. Ponadto wiele układów i konfiguracji cyfrowych można zaimplementować w jednym układzie, co znacząco obniża koszty produkcji i przyspiesza realizację projektu.

1.2 Cel pracy

Celem pracy jest nauka projektowania cyfrowych układów elektronicznych z wykorzystaniem języków VHDL i System Verilog oraz implementacji w programowalnych układach FPGA. Podczas projektowania układów będziemy korzystać m.in. z gotowych elementów sprzętowych (ang. IP cores) oraz integrować układy z systemami mikroprocesorowymi w oparciu o procesory syntezowane i rzeczywiste.

1.3 Motywacja

Mikroprocesor programowy (zwany też procesorem programowym) jest procesorem w całości wykonanym na układzie FPGA przy użyciu syntezy logicznej, zwykle z języka VHDL lub Verilog. Konieczność stałego uzupełniania wiedzy jest wpisana w zawód programisty. Developer uczy się nowych rzeczy praktycznie codziennie, choćby wdrażając nową bibliotekę do projektów. Jako początkujący "PENTESTER" chciałbym użyć znania programowania FPGA i implementacji MICROBLAZE w podalszej pracy.

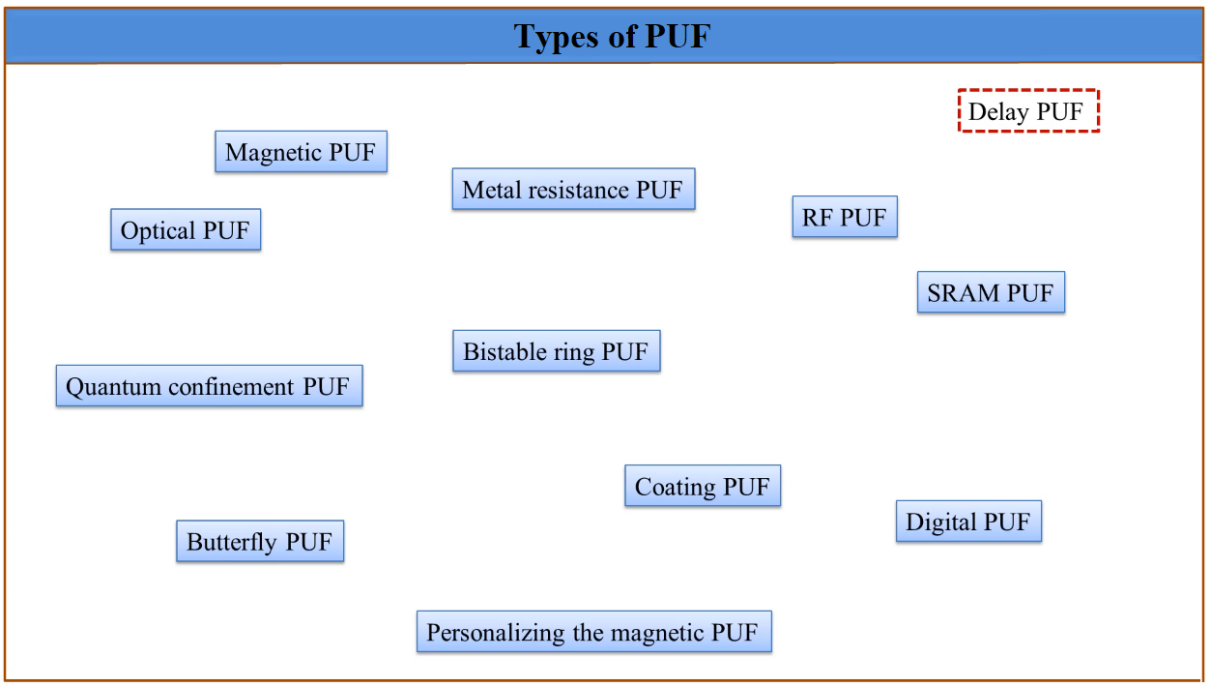
Część realizacji

2. Część realizacji

2.1 Informacje ogólne

2.1.1 Typy PUF

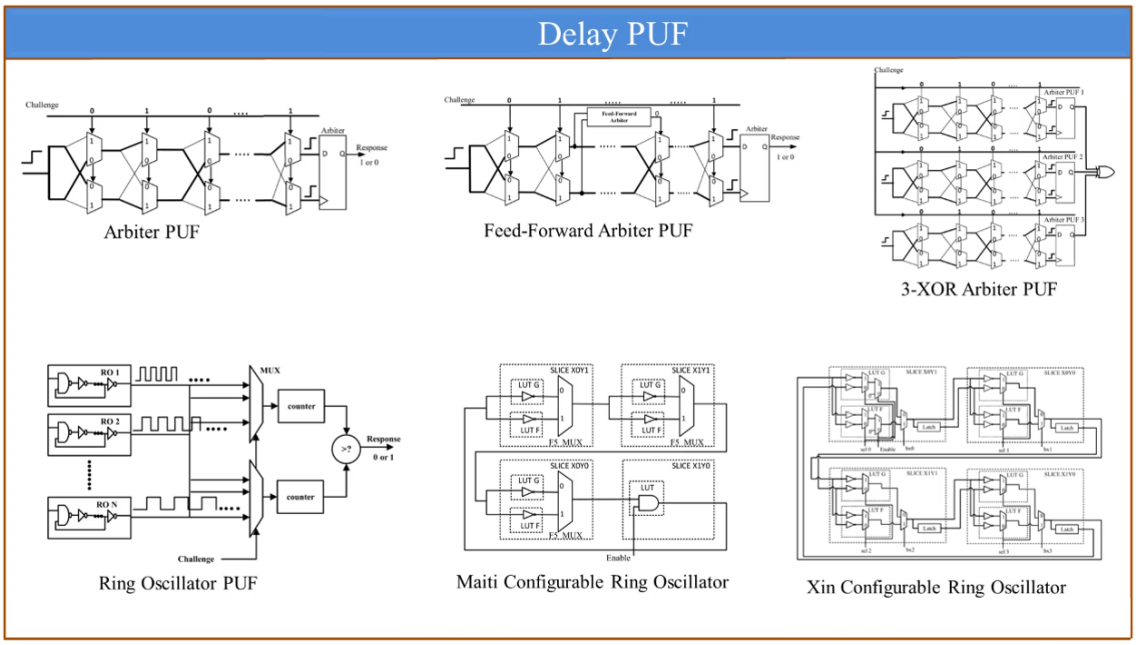
Fizyczna funkcja nieklonowalna (PUF), czasami nazywana także funkcją fizycznie niemożliwą do klonowania, jest fizyczną jednostką, która jest zawarta w strukturze fizycznej i jest łatwa do oceny, ale trudna do przewidzenia. Wszystkie PUF podlegają zmianom środowiskowym, takim jak temperatura, napięcie zasilania i zakłócenia elektromagnetyczne, które mogą wpływać na ich działanie. Dlatego prawdziwą mocą PUF, a nie tylko przypadkowa, jest jej zdolność do różnicowania się między urządzeniami, ale jednocześnie bycia tym samym w różnych warunkach środowiskowych.



Rysunek 2.1. Typy PUF

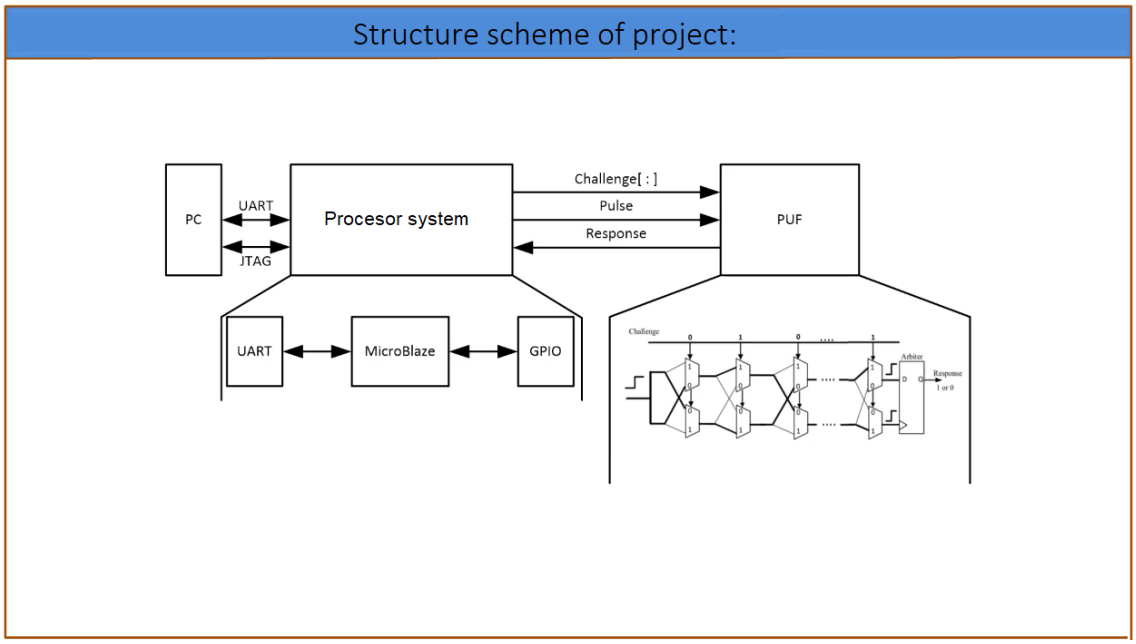
2.1.2 Typy Delay

Za pomocą logiki kombinatorycznej wykonuje się logikę PUF. W tym projekcie zaimplementowano cztery typy opóźnionego PUF, mianowicie Arbiter PUF, Xor - Arbiter PUF, Lightweight Secure PUF i Feed Forward PUF. Oparte na opóźnieniu PUF wykorzystują wbudowane charakterystyki opóźnień fizycznych komponentów, które zmieniają się między chipami.

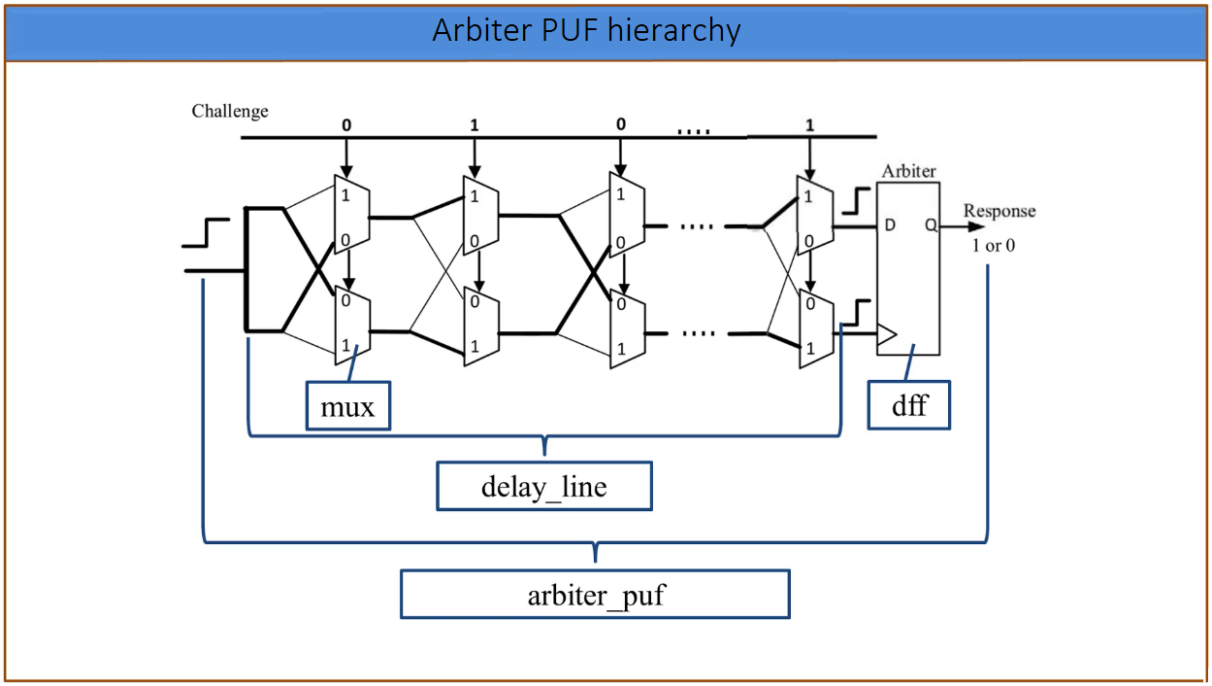


Rysunek 2.2. Typy delay PUF

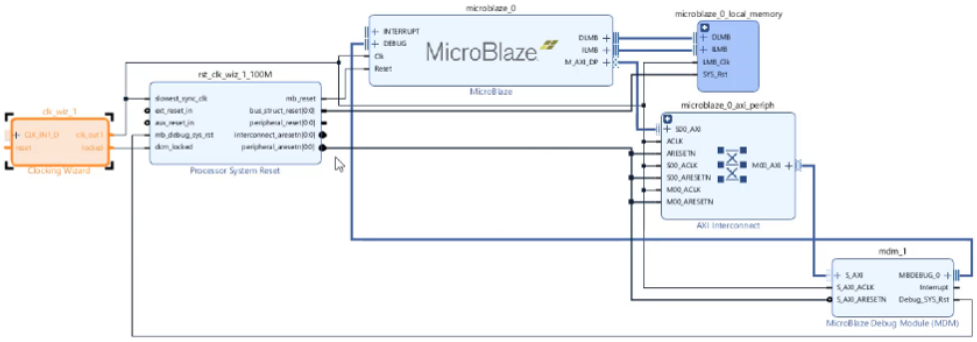
2.1.3 Schemat projektu głównego



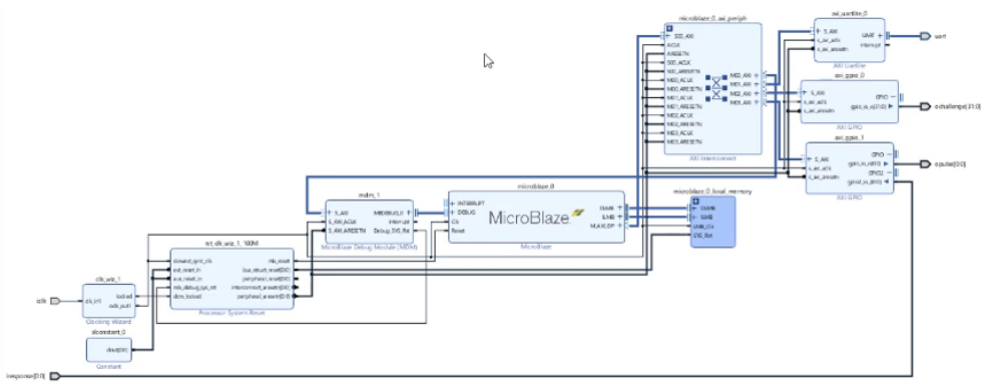
2.1.4 Schemat pierwszej części



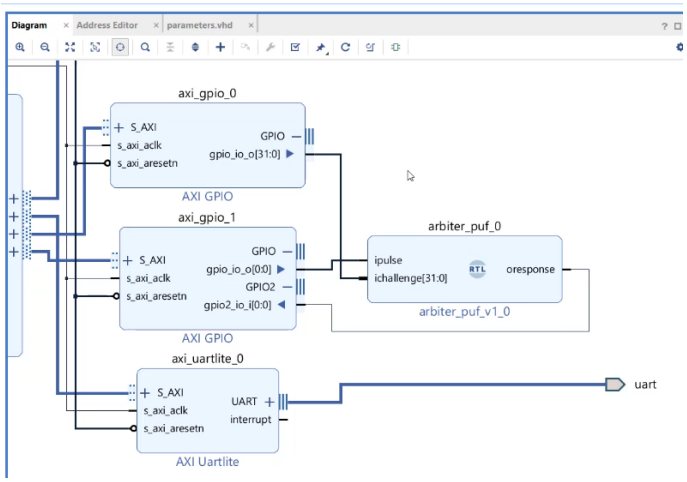
2.1.5 Schemat microblaze



2.1.6 Schemat "PROCESSOR SYSTEM"



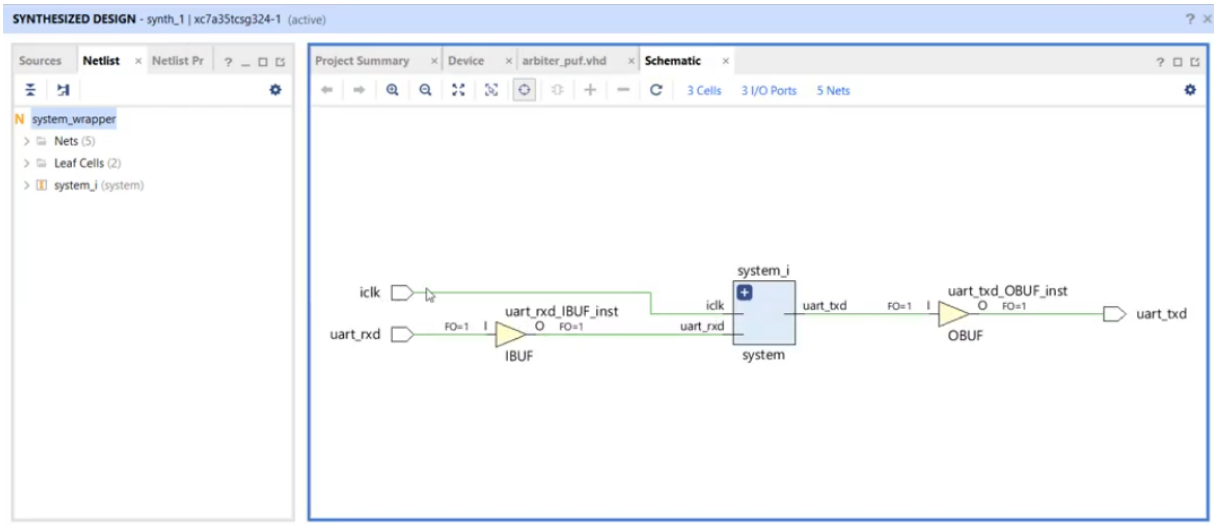
2.1.7 Schemat integracji microblaze i processor system



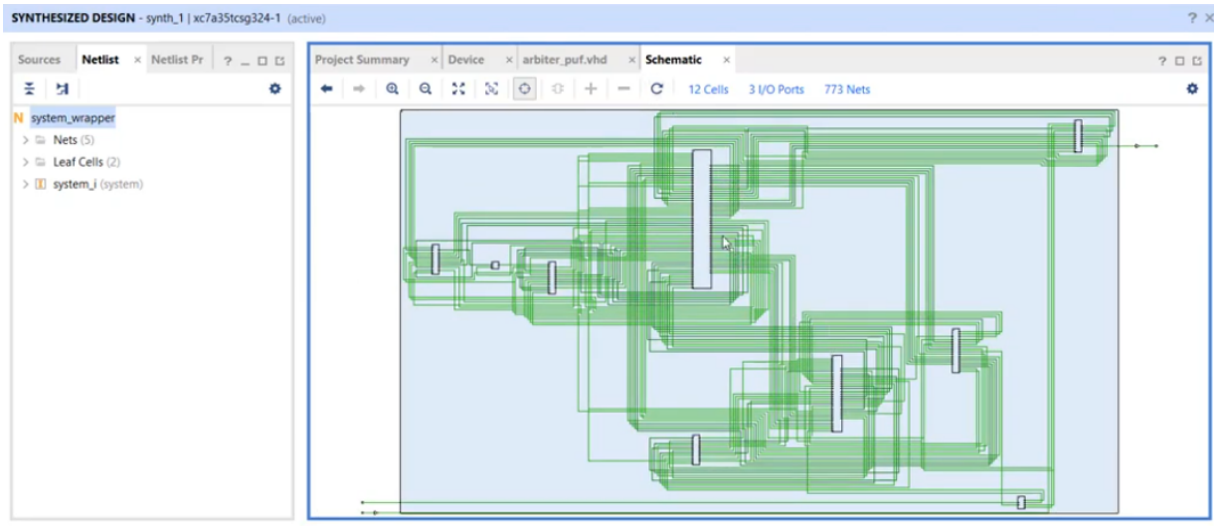
2.2 Rezultat implementacji

Po dodaniu wszystkich elementów włączamy Run Synthesis i Run Implementation. Jeśli wszystkie integracji zostały poprawnie dodane, dostajemy taki rezultat.

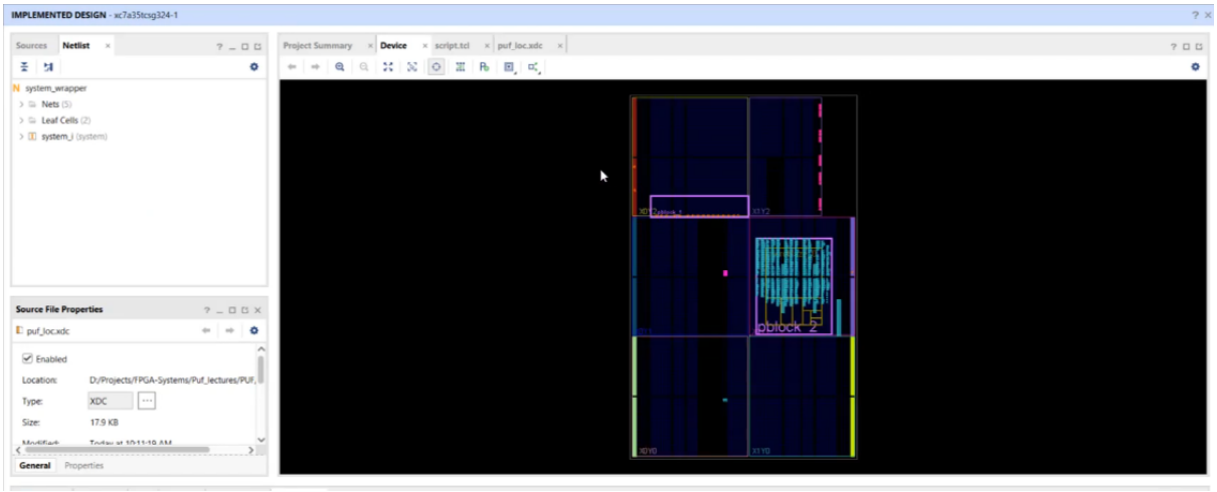
2.2.1 Synthesis



2.2.2 Schematic



2.2.3 Device

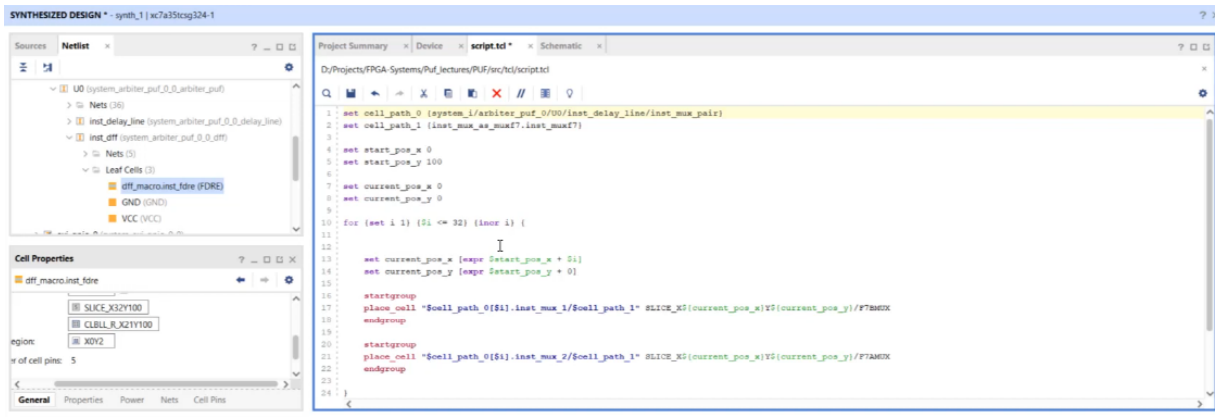

IMPLEMENTED DESIGN - xc7a35tcsg324-1
Sources
Netlist
system_wrapper
Nets (5)
Leaf Cells (2)
system_i (system)
Source File Properties
puf_loc.xdc
Enabled
Location:
Type:
XDC
Size:
17.9 KB
General
Properties
Project Summary
Device
script.tcl
puf_loc.xdc
pBlock_2

Część testowa

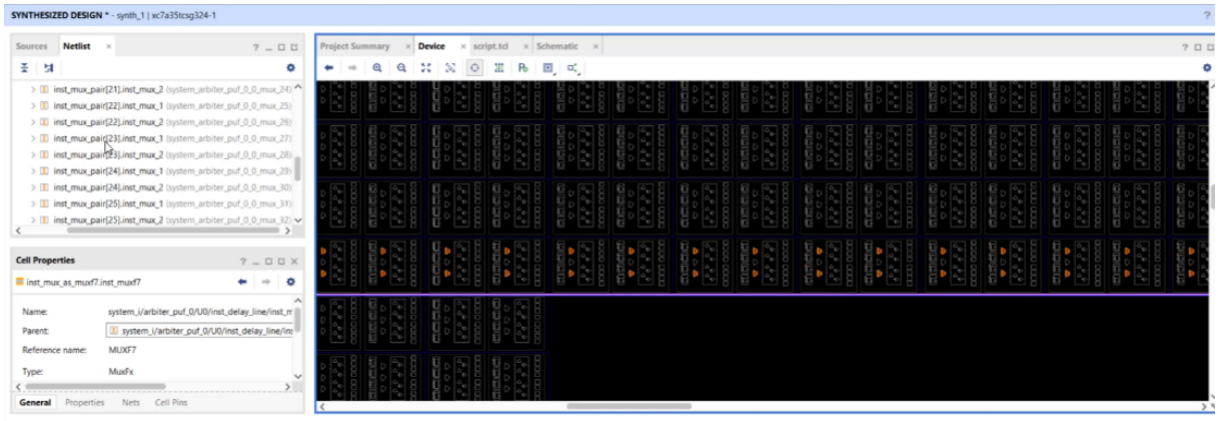
3. Część programowania w języku C

3.1 Programowanie TCL

3.1.1 Kod dla rozmieszczenie multypleksorów przez TCL



3.1.2 Rezultat TLC rozmieszczenie



3.2 Implementacja kodu C w SDK Xilinx Vivado

3.2.1 Biblioteki

```
#include "xparameters.h"
#include "xgpio.h"
#include "xil_printf.h"
#include "xuarlite.h"
```

3.2.2 Main body

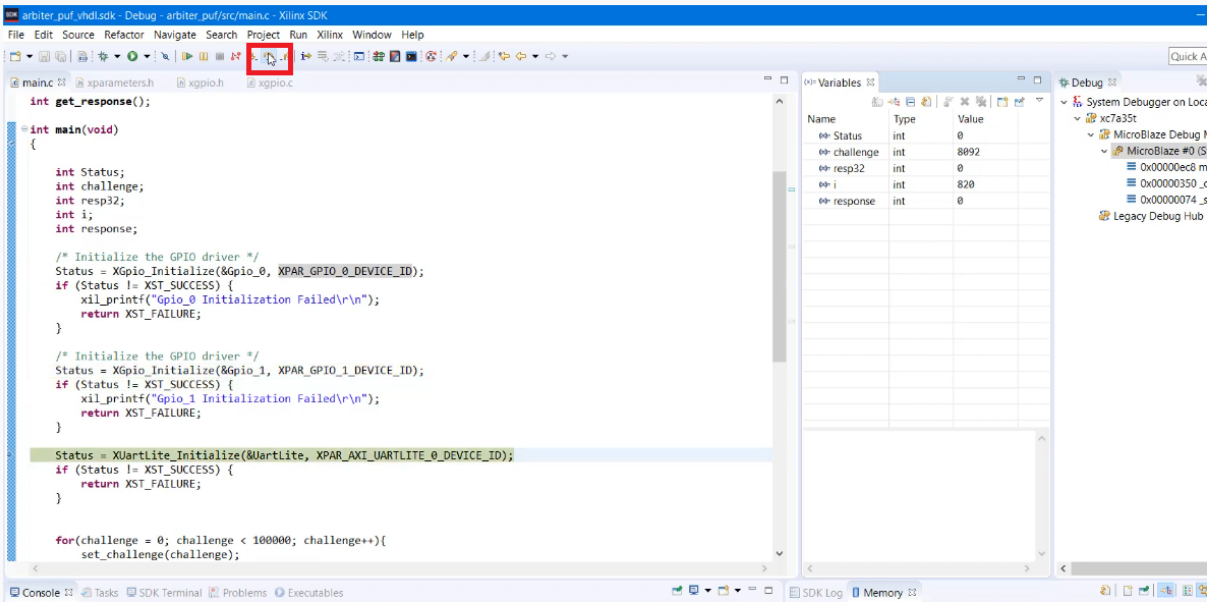
```
XGpio Gpio_0;
XGpio Gpio_1;
XUarlite Uarlite;

void set_challenge(int challenge);
void create_pulse();
int get_response();

int main(void)
{
  int Status;
  int challenge;
  int resp32;
  int i;
  int response;

  Status = XGpio_Initialize(&Gpio_0, XPAR_GPIO_0_DEVICE_ID);
  if (Status != XST_SUCCESS) {
    xil_printf("Gpio_0 Initialization Failed\r\n");
    return XST_FAILURE;
  }
  Status = XGpio_Initialize(&Gpio_1, XPAR_GPIO_1_DEVICE_ID);
  if (Status != XST_SUCCESS) {
    xil_printf("Gpio_1 Initialization Failed\r\n");
    return XST_FAILURE;
  }
  Status = XUarlite_Initialize(&Uarlite, XPAR_AXI_UARTLITE_0_DEVICE_ID
     );
  if (Status != XST_SUCCESS) {
    xil_printf("Gpio_1 Initialization Failed\r\n");
```

```
    return XST_FAILURE;
  }

  for(challenge = 0; challenge < 100000; challenge++){
    set_challenge(challenge);
    resp32 = 0;

    for(i = 0; i < 32; i++){
    create_pulse();
    response = get_response();
    resp32 = resp32 | (response << i);
    }
    if ((resp32 != 0x00000000) && (resp32 != 0xFFFFFFFF)){
      xil_printf("challenge = %d; response = %x\n\r", challenge,
         resp32);
    }

  }

  return 0;
}
```

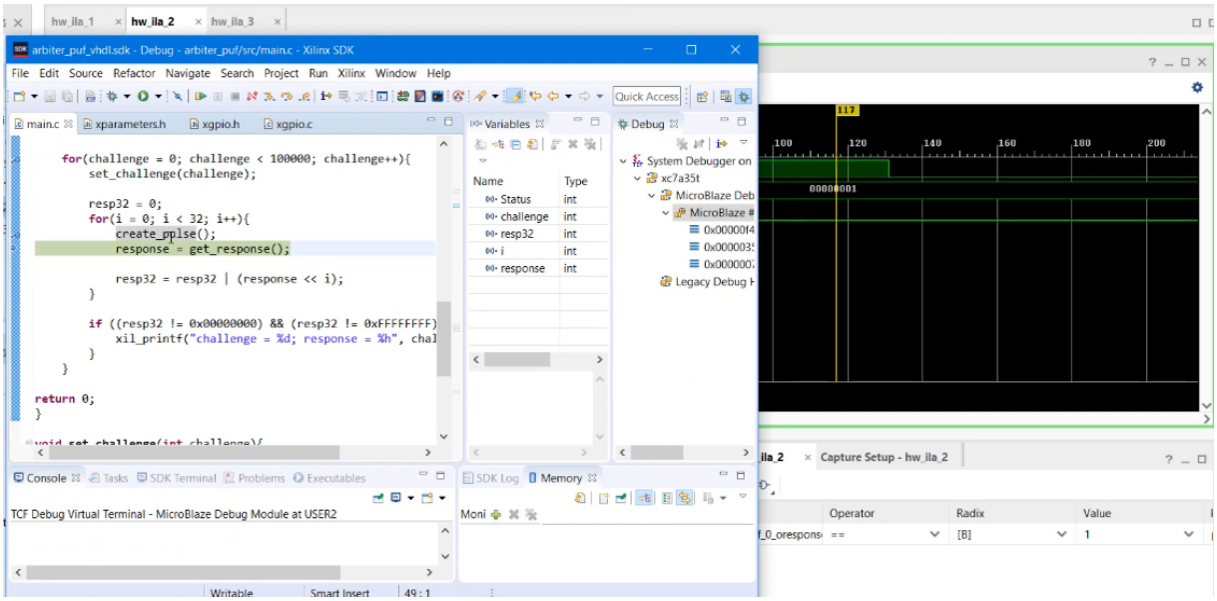
3.2.3 Odczyt i zapis impulsów

```
void set_challenge(int challenge){
  XGpio_DiscreteWrite(&Gpio_0, 1, challenge);
}
void create_pulse(){
  XGpio_DiscreteWrite(&Gpio_1, 1, 1);
  XGpio_DiscreteWrite(&Gpio_1, 1, 0);
}
int get_response(){
  int ret;
  ret = XGpio_DiscreteRead(&Gpio_1, 2);
  return ret;
}
```

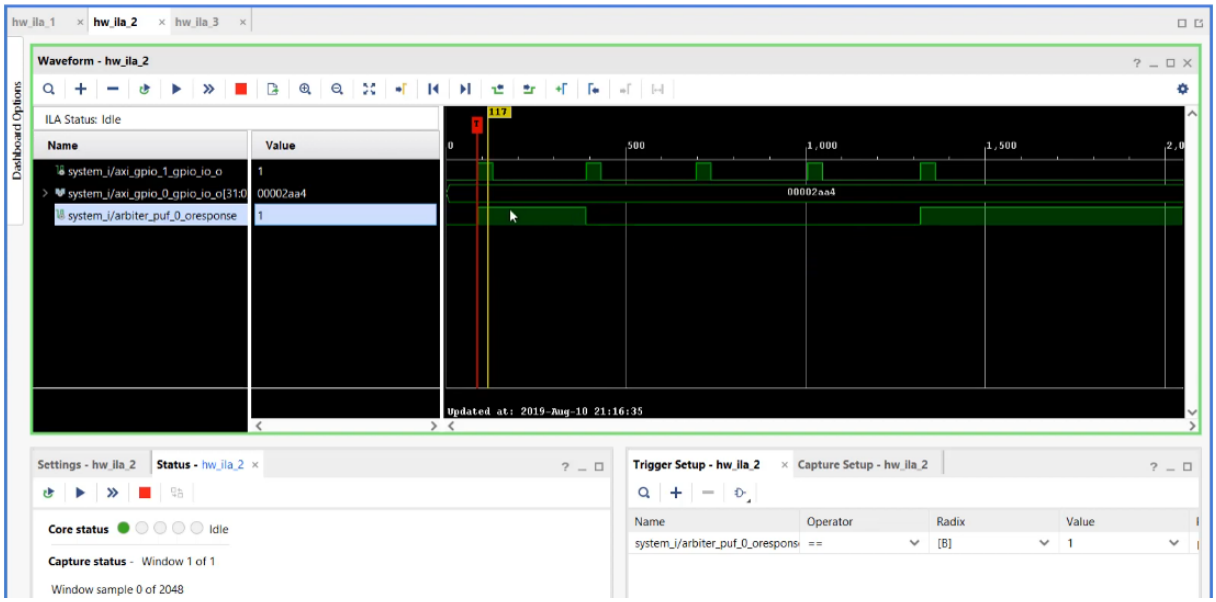
3.3 Wysyłanie impulsów



3.3.1 Odczyt i zapis impulsów



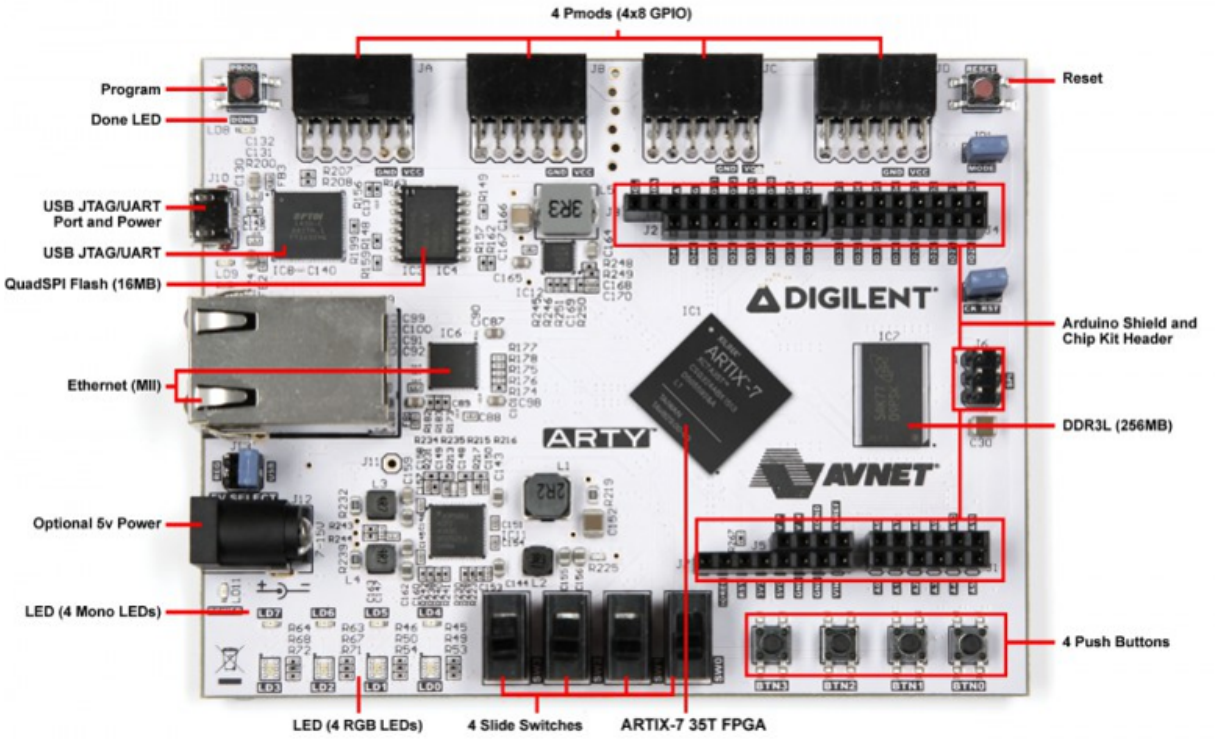
3.3.2 Informacja zwrotna od impulsu



4. Podsumowanie

4.1 Artix-7 35T Arty - zestaw dla FPGA

Podczas sprawdzenia poprawności projektu był wykorzystany Artix-7



Więcej informacji

Właściwości

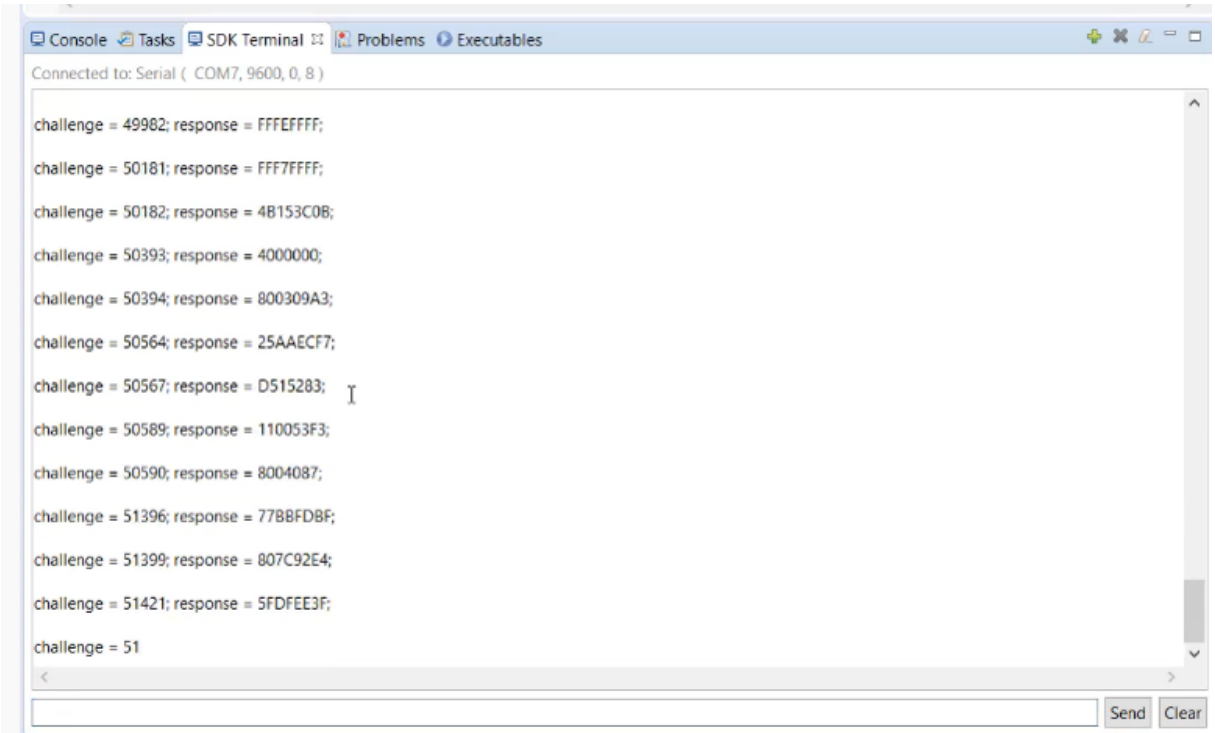
- Układ FPGA Xilinx Artix-7 XC7A35T-L1CSG324I
- Pamięć DDR3L 256 MB, szyna 16-bitowa, 667 MHz
- 16 MB pamięci Flash Quad-SPI
- Interfejs sieciowy Ethernet 10/100 Mb
- Konwerter USB - UART
- Przełączniki, przyciski, diody RGB
- Cztery złącza dla modułów Pmod (32 linie IO)
- Złącze dla shieldów Arduino (49 linii IO)

W zestawie

- Płyta ewaluacyjna Arty (AES-A7MB-7A35T-G)
- Oprogramowanie Vivado® Design Suite: Design Edition z licencją Node Locked (przypisanie do komputera) i Device Locked (przypisanie do płytki ewaluacyjnej) z aktualizacjami i wsparciem na okres jednego roku

4.1.1 Ogólny wynik

To zdjęcie przedstawia końcowy wynik naszego projektu. Jak widzimy, Artix-7 35T Arty generuje losowe cyfry - jest to faktyczna funkcja arbitra PUF - prosta struktura 32 par multiplekserów wykorzystywana w naszym codziennym życiu. Obecnie jest on wykorzystywany w cyberbezpieczeństwie, generatorach haseł i impulsach telefonicznych i zapewnia, że ta lista impulsów nie będzie klonowana ani przewidywana i wreszcie nie można go zhakować ani przechwycić - co jest naszą główną misją - aby zabezpieczyć nasze codzienne zadania.



Bibliografia

[1] W. Wrona, *Verilog. Język w projektowaniu układów cyfrowych*, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, 2009

[2] Z. Hajduk, *Wprowadzenie do języka Verilog*, Wydawnictwo BTC, 2009

[3] J. Majewski, P. Zbysiński *Układy FPGA w przykładach*, Wydawnictwo BTC, 2007

[4] Ming-Bo Lin, *Digital System Designs and Practice Using Verilog HDL and FPGAs*, John Wiley Sons, 2008

[5] B. J. LaMeres, *Introduction to Logic Circuits Logic Design with Verilog*, Springer, 2017

[6] IEEE Std 1364-2001 *IEEE Standard Verilog Hardware Description Language*

[7] IEEE Std 1076-2008 *IEEE Standard VHDL Language Reference Manual*

[8] IEEE Std 1800-2005 *IEEE Standard SystemVerilog - Unified Hardware Design, Specification and Verification Language*