|  |  |
| --- | --- |
| Akademia Górniczo-Hutnicza  im. Stanisława Staszica  w Krakowie.  Wydział Elektrotechniki, Automatyki, Informatyki  i Elektroniki | |
| SDUP 2020  IV ROK / SW / EiT | |
| Temat projektu:  **„Wyszukiwanie wartości minimum i maksimum dla wielomianu o zadanych współczynnikach w zadanym zakresie argumentów funkcji”** | |
| Grupa:  1. Rafał Łaskawski    2. Rafał Jakubiec  3. Krzysztof Jaje | |
| Ocena sprawozdania: | |
| Uwagi prowadzącego ćwiczenia: | Informacje dodatkowe: |

1. Założenia projektowe

Zadaniem było przygotowanie procesora sprzętowego który realizuje wyszukiwanie wartości minimum i maksimum dla wielomianu o zadanych współczynnikach o wartościach od -10 do 10 w zakresie argumentów funkcji od -100 do 100. Do obliczania wykorzystano metodę Hornera. Dzięki niej redukuje się ilość niezbędnych obliczeń do minimum.

1. Sposób komunikacji CPU

CPU Zynq ARM komunikuje się za pomocą magistrali AXI z modułem IP za pomocą dziesięciu czterobajtowych rejestrów. Przestrzeń adresowa wygląda następująco:

//Horner processor base addres redefinition

**#define** HORNER\_BASE\_ADDR XPAR\_HORNER\_IP\_0\_S00\_AXI\_BASEADDR

//Horner processor registers' offset redefinition

**#define** CONTROL\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG0\_OFFSET

**#define** ARG\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG1\_OFFSET

**#define** CN0\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG2\_OFFSET

**#define** CN1\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG3\_OFFSET

**#define** CN2\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG4\_OFFSET

**#define** CN3\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG5\_OFFSET

**#define** CN4\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG6\_OFFSET

**#define** CN5\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG7\_OFFSET

**#define** STATUS\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG8\_OFFSET

**#define** RESULT\_REG\_OFFSET HORNER\_IP\_S00\_AXI\_SLV\_REG9\_OFFSET

Gdzie:

**#define** XPAR\_HORNER\_IP\_0\_S00\_AXI\_BASEADDR 0x43C00000

**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG0\_OFFSET 0

**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG1\_OFFSET 4

**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG2\_OFFSET 8

**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG3\_OFFSET 12

**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG4\_OFFSET 16

**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG5\_OFFSET 20

**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG6\_OFFSET 24

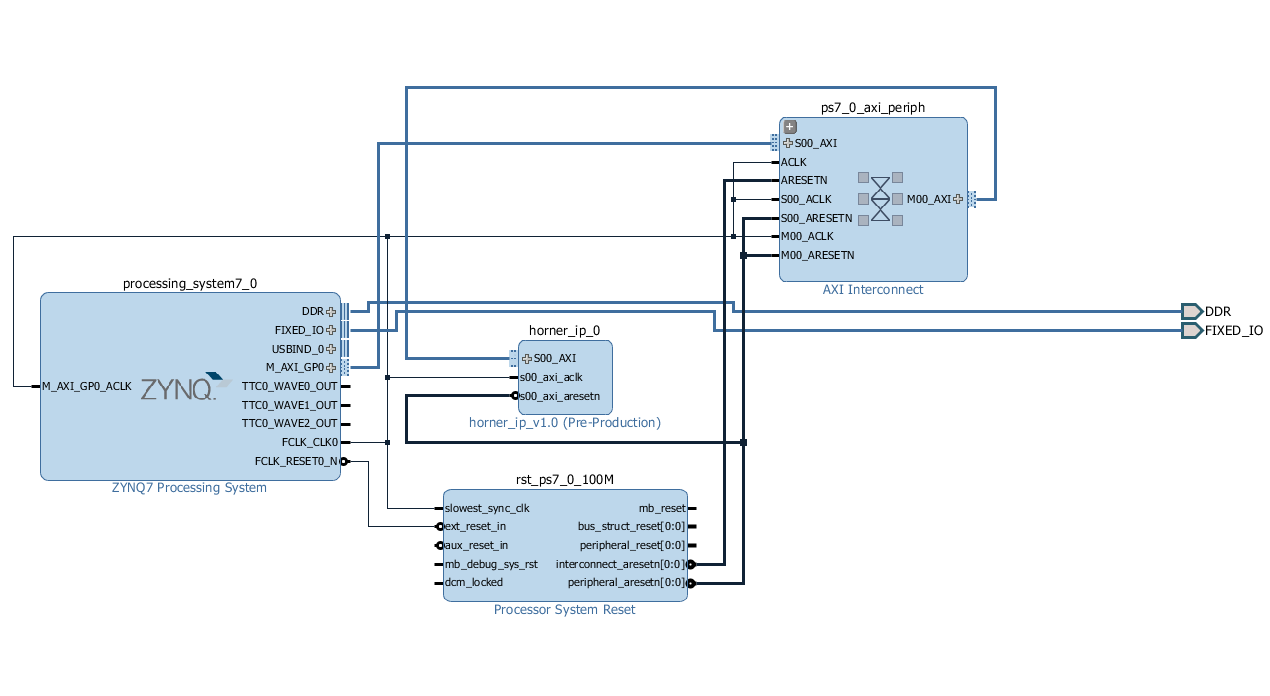
**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG7\_OFFSET 28

**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG8\_OFFSET 32

**#define** HORNER\_IP\_S00\_AXI\_SLV\_REG9\_OFFSET 36

1. Schemat blokowy.

Połączenia procesora ARM z modułem IP (Rys. 1) za pomocą magistrali AXI wygenerowane automatycznie przez Vivado. Automatycznie zostały dodane moduły AXI interconnect oraz Processor System Reset:



Rys. 1 Schemat blokowy.

1. Zasada działania algorytmu

Zasada działania polega na wstępnym obliczeniu wartości minimalnej oraz maksymalnej w zakresie od -100 do 100 dla argumentów typu integer (krok 1). Następnie na podstawie otrzymanych wartości minimum i maximum integer szukane są wartości bardziej precyzyjne z krokiem 0,01 w promieniu o wartości 1 od wyszukanej wartości minimum i maximum integer (krok 2).

#define CALCULATE\_HORNER\_IN\_FPGA (1)

int32\_t calc\_poly\_rough\_int**(**const int8\_t **\*** constants**,** const int32\_t argument**)**

**{**

#if CALCULATE\_HORNER\_IN\_FPGA

**return** calculateHornerValInFPGA**((**s32**)**argument**,** **(**s32**)**constants**[**0**],** **(**s32**)**constants**[**1**],** **(**s32**)**constants**[**2**],** **(**s32**)**constants**[**3**],** **(**s32**)**constants**[**4**],** **(**s32**)**constants**[**5**]);**

#else

//Horner's method

int32\_t retVal **=** **(**int32\_t**)**constants**[**0**];**

**for(**uint8\_t a **=** 1**;** a **<** **(**DEGREE\_OF\_A\_POLYNOMINAL **+** 1**);** a**++)**

**{**

retVal **=** retVal **\*** argument **+** **(**int32\_t**)**constants**[**a**];**

**}**

**return** retVal**;**

#endif

**}**

float calc\_poly\_fine\_float**(**const int8\_t **\*** constants**,** const float argument**)**

**{**

//Horner's method

float retVal **=** **(**float**)**constants**[**0**];**

**for(**uint8\_t a **=** 1**;** a **<** **(**DEGREE\_OF\_A\_POLYNOMINAL **+** 1**);** a**++)**

**{**

retVal **=** retVal **\*** argument **+** **(**float**)**constants**[**a**];**

**}**

**return** retVal**;**

**}**

void calculate\_min\_max**(**const int8\_t **\*** polynominal\_constants**,** float**\*** min**,** float **\*** max**)**

**{**

const uint16\_t values\_array\_size **=** 2**\***RANGE **+** 1**;**

int32\_t values**[**values\_array\_size**];**

int32\_t minVal **=** INT\_MAX**;**

int32\_t maxVal **=** INT\_MIN**;**

int32\_t arg\_at\_min **=** 0U**;**

int32\_t arg\_at\_max **=** 0U**;**

//calculate rough integer values for polynominal arguments and find integer min and max

**for(**int a **=** 0**;** a **<** values\_array\_size**;** a**++)**

**{**

int32\_t arg **=** a **-** RANGE**;**

values**[**a**]** **=** calc\_poly\_rough\_int**(**polynominal\_constants**,** arg**);**

//xil\_printf("retVal: %d at argument: %d\r\n", values[a], arg);

//get min and max

**if(**values**[**a**]** **<** minVal**)**

**{**

arg\_at\_min **=** arg**;** //new min argument

minVal **=** values**[**a**];** //new min value

**}**

**if(**values**[**a**]** **>** maxVal**)**

**{**

arg\_at\_max **=** arg**;** //new max argument

maxVal **=** values**[**a**];** //new max value

**}**

**}**

//printf("min arg int: %d max arg int: %d\n", arg\_at\_min, arg\_at\_max);

//calculate fine values for minimum and maximum, step 0.01, check values in +/- 1 scope for given minimum and maximum arguments

const uint16\_t values\_fine\_array\_size **=** 201U**;** //+/- 100 steps and zero

float values\_fine**[**values\_fine\_array\_size**];**

float minVal\_fine **=** **(**float**)**minVal**;**

float maxVal\_fine **=** **(**float**)**maxVal**;**

float min\_fine\_arg **=** 0.0f**;**

float max\_fine\_arg **=** 0.0f**;**

float arg **=** 0.0f**;**

//minimum

arg **=** **(**float**)**arg\_at\_min **-** FINE\_RANGE**;**

**for(**int a **=** 0**;** a **<** values\_fine\_array\_size**;** a**++)**

**{**

values\_fine**[**a**]** **=** calc\_poly\_fine\_float**(**polynominal\_constants**,** arg**);**

//get fine min

**if(**values\_fine**[**a**]** **<** minVal\_fine**)**

**{**

minVal\_fine **=** values\_fine**[**a**];**

min\_fine\_arg **=** arg**;**

**}**

arg **+=** 0.01f**;**

**}**

//maximum

arg **=** **(**float**)**arg\_at\_max **-** FINE\_RANGE**;**

**for(**int a **=** 0**;** a **<** values\_fine\_array\_size**;** a**++)**

**{**

values\_fine**[**a**]** **=** calc\_poly\_fine\_float**(**polynominal\_constants**,** arg**);**

//get fine max

**if(**values\_fine**[**a**]** **>** maxVal\_fine**)**

**{**

maxVal\_fine **=** values\_fine**[**a**];**

max\_fine\_arg **=** arg**;**

**}**

arg **+=** 0.01f**;**

**}**

UNUSED**(**min\_fine\_arg**);**

UNUSED**(**max\_fine\_arg**);**

//printf("min arg fine: %3.2f max arg fine: %3.2f\n", min\_fine\_arg, max\_fine\_arg);

**\***min **=** minVal\_fine**;**

**\***max **=** maxVal\_fine**;**

//\*min = (float) minVal;

//\*max = (float) maxVal;

**}**

Sposób przekazywania argumentu i współczynników wielomianu do modułu IP w celu dokonania obliczeń:

int32\_t calculateHornerValInFPGA**(**s32 arg**,** s32 cn0**,** s32 cn1**,** s32 cn2**,** s32 cn3**,** s32 cn4**,** s32 cn5**)**

**{**

int32\_t retVal **=** 0**;**

//Send data to data register of Horner processor

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** ARG\_REG\_OFFSET**,** arg**);**

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CN0\_REG\_OFFSET**,** cn0**);**

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CN1\_REG\_OFFSET**,** cn1**);**

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CN2\_REG\_OFFSET**,** cn2**);**

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CN3\_REG\_OFFSET**,** cn3**);**

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CN4\_REG\_OFFSET**,** cn4**);**

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CN5\_REG\_OFFSET**,** cn5**);**

//Start Horner processor - pulse start bit in control register

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CONTROL\_REG\_OFFSET**,** CONTROL\_REG\_START\_MASK**);**

//Wait for ready bit in status register

**while(** **(**HORNER\_IP\_mReadReg**(**HORNER\_BASE\_ADDR**,** STATUS\_REG\_OFFSET**)** **&** STATUS\_REG\_READY\_MASK**)** **==** 0**);**

//finish reading

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CONTROL\_REG\_OFFSET**,** 0**);**

//Start Horner processor - pulse start bit in control register

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CONTROL\_REG\_OFFSET**,** CONTROL\_REG\_START\_MASK**);**

//Wait for ready bit in status register

**while(** **(**HORNER\_IP\_mReadReg**(**HORNER\_BASE\_ADDR**,** STATUS\_REG\_OFFSET**)** **&** STATUS\_REG\_READY\_MASK**)** **==** 0**);**

//Get results

retVal **=** HORNER\_IP\_mReadReg**(**HORNER\_BASE\_ADDR**,** RESULT\_REG\_OFFSET**);**

//finish reading

HORNER\_IP\_mWriteReg**(**HORNER\_BASE\_ADDR**,** CONTROL\_REG\_OFFSET**,** 0**);**

**return** retVal**;**

**}**

Maszyna stanów w module IP obliczająca wartość wielomianu metodą Hornera:

**always** **@** **(posedge** clock**)**

**begin**

**if(**reset**==**1'b1**)**

**begin**

state **<=** S4**;**

**end**

**else**

**begin**

**case(**state**)**

S1**:** **begin**

**if(**start **==** 1'b1**)** state **<=** S2**;** **else** state **<=** S1**;**

**end**

S2**:** **begin**

//assingment to variables

arg **<=** arg\_in**;**

cn0 **<=** CN0**;**

cn1 **<=** CN1**;**

cn2 **<=** CN2**;**

cn3 **<=** CN3**;**

cn4 **<=** CN4**;**

cn5 **<=** CN5**;**

out **<=** 0**;**

//calculations

out **=** cn0**;**

out **=** **(**out **\*** arg**)** **+** cn1**;**

out **=** **(**out **\*** arg**)** **+** cn2**;**

out **=** **(**out **\*** arg**)** **+** cn3**;**

out **=** **(**out **\*** arg**)** **+** cn4**;**

out **=** **(**out **\*** arg**)** **+** cn5**;**

//save result to register

result\_out **<=** out**;**

ready\_out **<=** 1**;**

state **<=** S3**;**

**end**

S3**:** **begin**

**if(**start **==** 1'b0**)** state **<=** S4**;** **else** state **<=** S3**;**

**end**

S4**:** **begin**

ready\_out **<=** 0**;**

state **<=** S1**;**

**end**

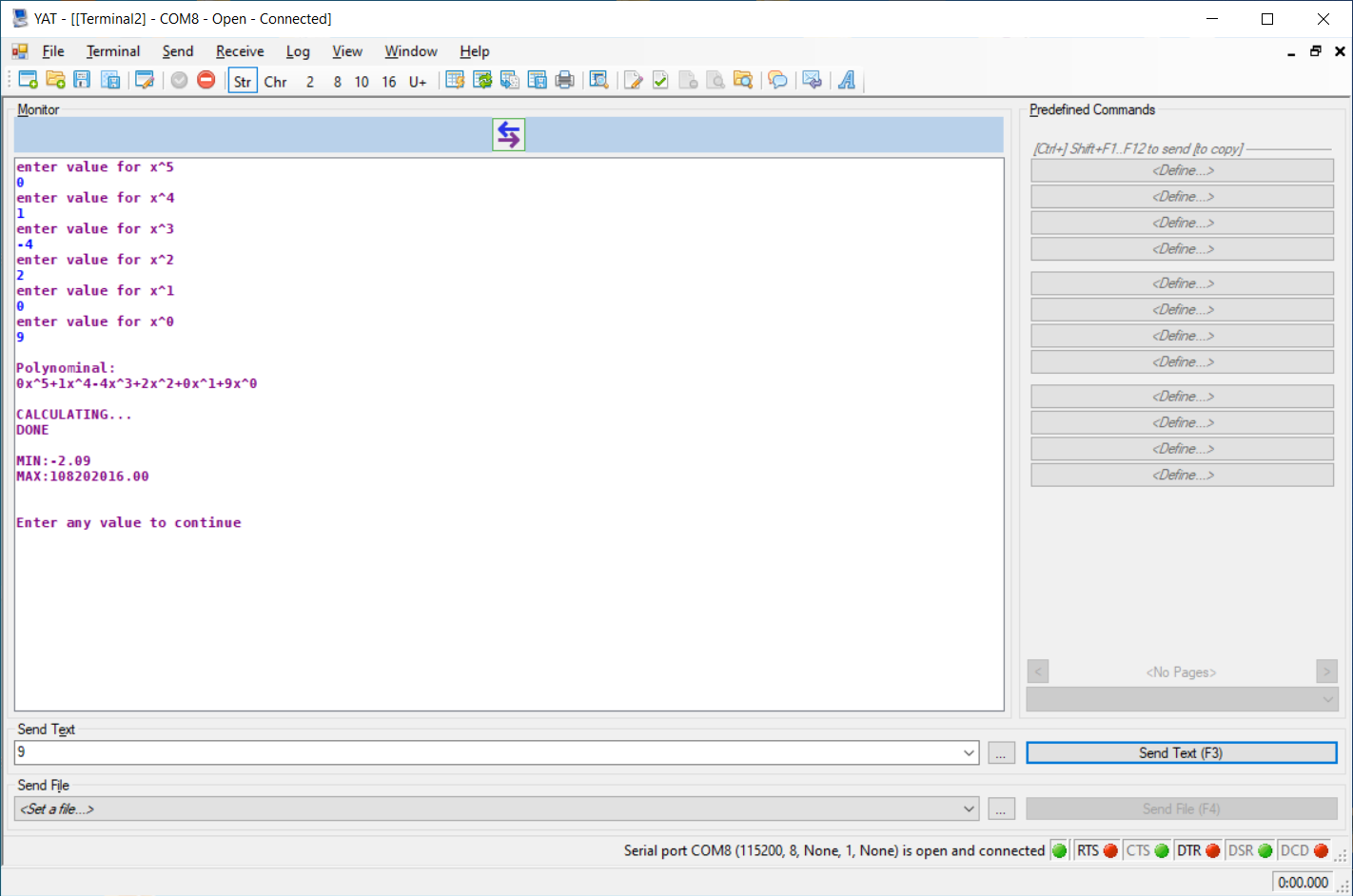
**endcase;**

**end;**

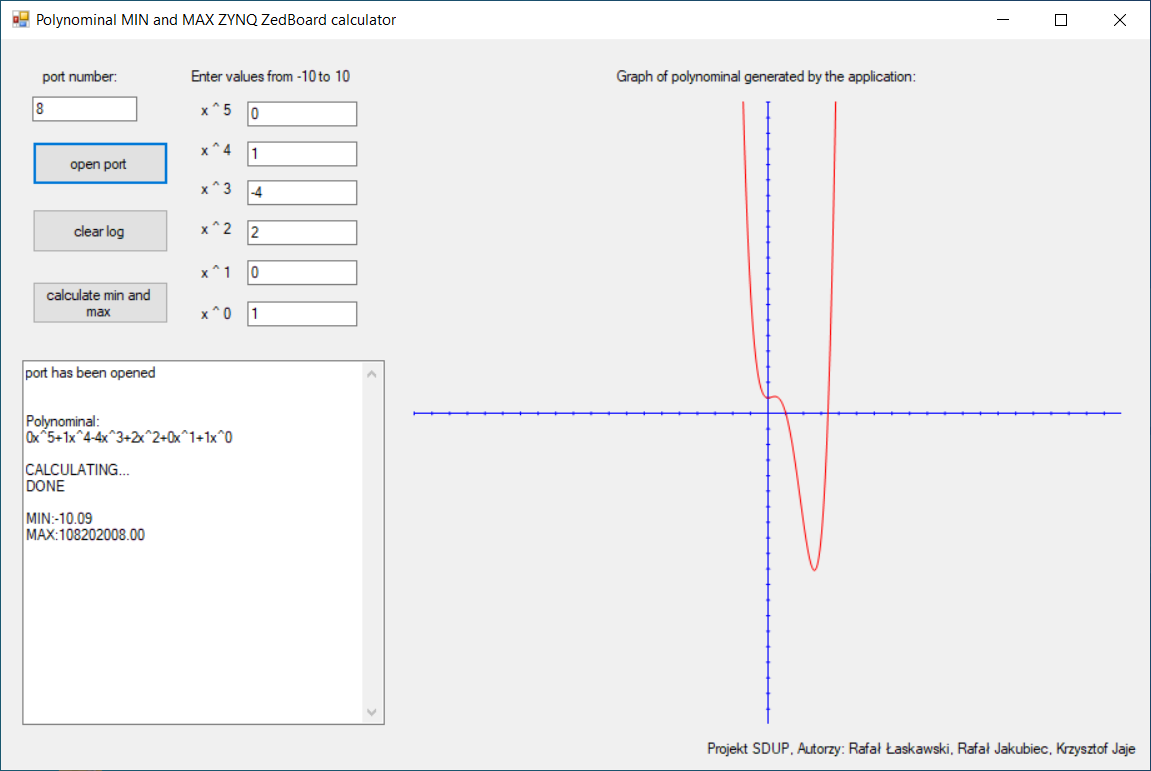
**end;**

1. Komunikacja UART.

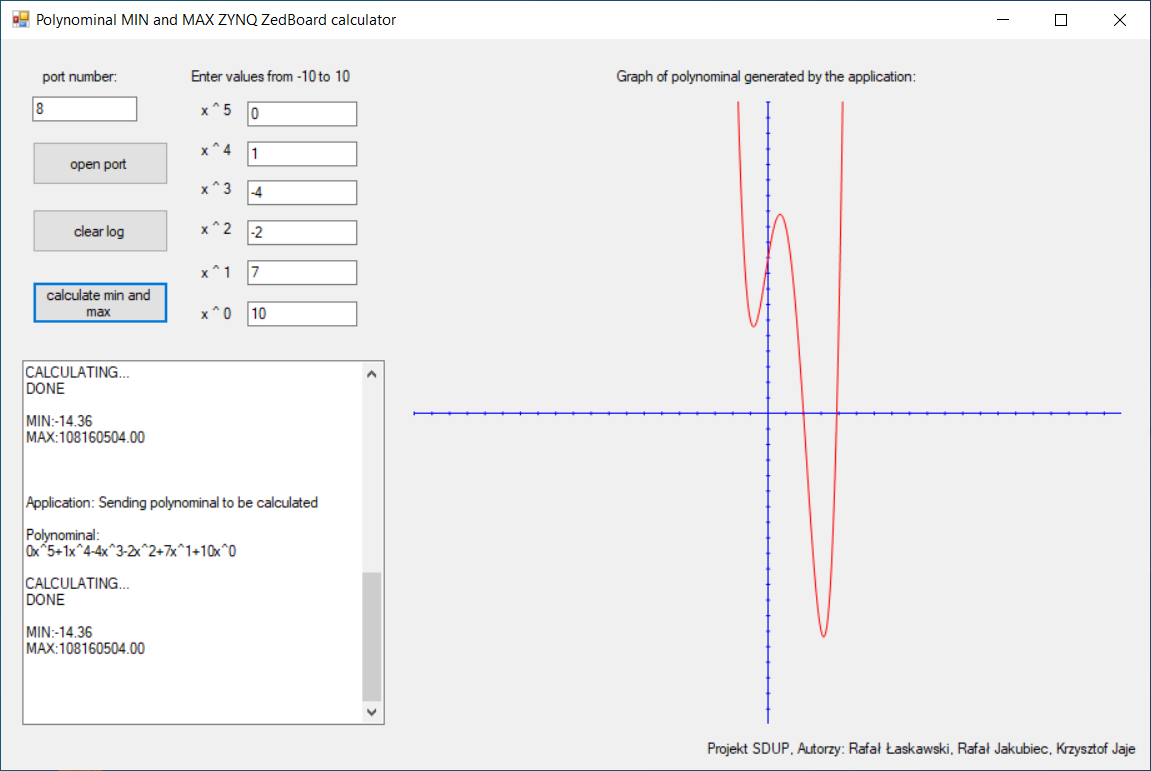
Wprowadzanie współczynników wielomianu oraz przekazywanie wyników obliczeń jest realizowane za pomocą magistrali UART. Możliwa jest komunikacja za pomocą konsoli (Rys. 2). Proszeni jesteśmy o podanie argumentów funkcji po czym otrzymujemy wyniki. Możliwe jest również użycie specjalnie przygotowanego interfejsu (Rys. 3,4) bardziej przyjaznego użytkownikowi napisanego w frameworku Windows Forms w języku C# w którym odpowiednie łańcuchy tekstowe są parsowane i dane (współczynniki wielomianu) są wprowadzane do obliczeń w sposób automatyczny z pól tekstowych. Ponadto w programie rysowany jest wykres wielomianu w celu wizualizacji aktualnie przetwarzanego wielomianu.



Rys.2 Komunikacja za pomocą konsoli.



Rys 3. Przygotowany interfejs.



Rys 4. Przygotowany interfejs.

1. Testbench

Dane testowe wprowadzone do algorytmu w celu testu poprawności działania oraz rezultat w postaci przebiegów czasowych:

//expected result\_out = 7 because only x^0 is different than zero and equals 7

**assign** arg\_in **=** 10**;**

**assign** CN0 **=** 0**;**

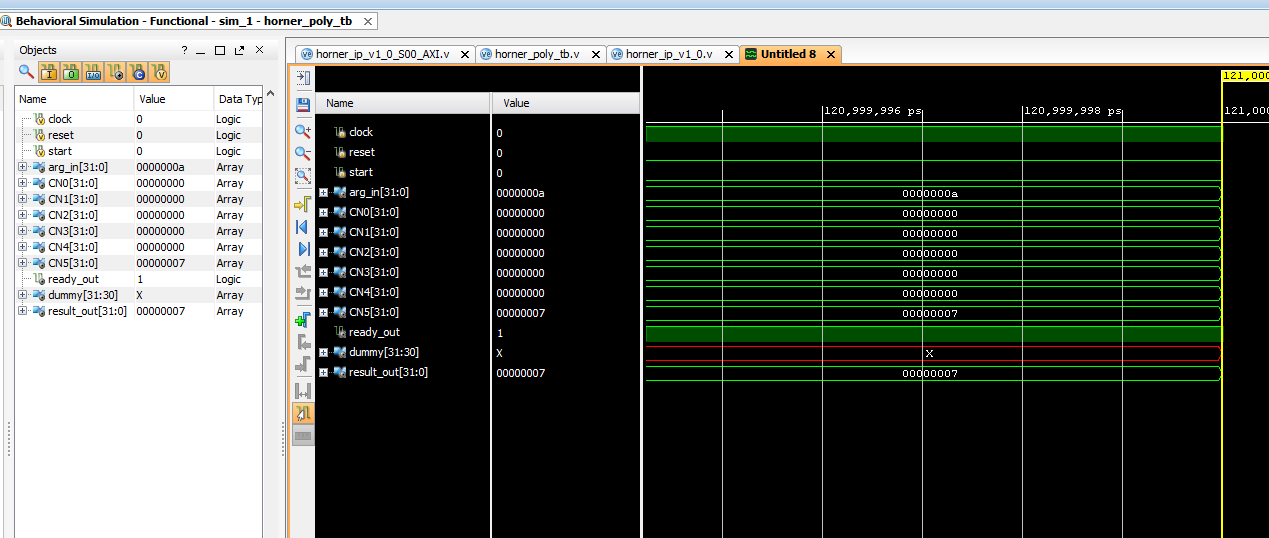
**assign** CN1 **=** 0**;**

**assign** CN2 **=** 0**;**

**assign** CN3 **=** 0**;**

**assign** CN4 **=** 0**;**

**assign** CN5 **=** 7**;**



Rys.5 Testbench.

Przy takim zestawie danych widać było od razu czy algorytm działa poprawnie. Gdy obliczenia były prowadzone w błędny sposób, zwracabe były bardzo duże losowe liczby.