# DOKUMENTACJA

**Autor:** Konrad Krupski

**Numer indeksu:** 310729

**Kontakt:**

Email: 01158726@pw.edu.pl

MS Teams: Konrad Krupski (STUD)

**Data:** 13.10.2021

**Platforma sprzętowa**

Projekt został wykonany na systemie Linux Ubuntu 20.04.3 64-bit.

**Sposób uruchamiania**

W celu samodzielnej konfiguracji potrzebne będą programy **yosys** oraz edytor tekstu.

Należy uruchomić terminal w katalogu, gdzie znajduje się plik opisu modelu exe_unit.sv. Następnie wpisujemy polecenia:

yosys

read_verilog -sv exe_unit.sv

synth

opt_clean

write_verilog -noattr exe_unit_rtl.sv

exit

# Contents

# Lista wejść i wyjść

Wartości zmiennych zostały w dokumentacji zostały podawana dla długości 4 bitów. Niemniej jednak w układzie został zastosowany parametr do ustalenia dokładnej długości, więc zależnie od potrzeb może ona się różnić.

## Wejścia

**i_argA** – wejście układu przechowujące liczbę zapisaną w systemie U2. Zakres wartości to [-7,7] w systemie dziesiętnym.

**i_argB** - wejście układu przechowujące liczbę zapisaną w systemie U2. Zakres wartości to [-7,7] w systemie dziesiętnym.

**i_oper** – czterobitowy sygnał sterujący układu zapisany w systemie binarnym. Zakres wartości to [0, 15].

## Wyjścia

**o_result** – wyjście układu przechowujące wynik operacji. Wartość wyniku jest określana na 4 bitach i jej zakres wynosi [-7,7]. Zapisana jest w kodzie U2.

**o_OF** – wyście układu przyjmujące wartość 0 lub 1 w systemie binarnym. Wartość 1 oznacza, że o_result przyjmuje wartości samych jedynek ‘1.  Wartość 0 oznacza, że o_result nie składa się z samych jedynek.

**o_BF** - wyście układu przyjmujące wartość 0 lub 1 w systemie binarnym. Wartość 1 oznacza, że wyjście o_result posiada jedną wartość 1. Wartość 0 oznacza, że wyjście o_result może mieć więcej bądź wcale wartości 1.

**o_VF** – wyjście układu przyjmuje wartość 0 lub 1 w systemie binarnym. Wartość 1 oznacza, że nastąpiło przepełnienie a wartość 0, że przepełnienie nie nastąpiło.

**o_PF** – wyjście układu przyjmuje wartość 1 lub 0 w systemie binarnym. Wartość 1 oznacza parzystą ilość jedynek na wyjściu a wartość 0 oznacza nie parzystą ilość jedynek.

## Sygnały pomocnicze

**counter**

Opis

zmienna typu int wykorzystywana do zliczania wartości przy sygnalizacji flag.

Typ: int

Ilość bitów: 32

## Sygnały wewnętrzne

**Nkb_code**

Opis

Przechowuje wartość obliczoną przez moduł one_hot.sv

Typ: logic signed

Ilość bitów: 4

Zakres: [-7, 7]

**Gray_code**

Opis

**Przechowuje wartość obliczoną przez moduł one_hot.sv**

Typ: logic signed

Ilość bitów: 4

Zakres: [-7, 7]

**Number_of_zeros**

Opis

Przechowuje wartość obliczoną przez moduł cout_zeros.sv

Typ: logic signed

Ilość bitów: 4

Zakres: [-7, 7]

**Crc_code**

Opis

Przechowuje wartość obliczoną przez moduł crc_coder.sv

Typ: logic signed

Ilość bitów: 4

Zakres: [-7, 7]

**Crc_3**

Opis

Przechowuje wartość obliczoną przez moduł crc_coder.sv

Typ: logic signed

Ilość bitów: 4

Zakres: [-7, 7]

**u2_code**

Opis

Przechowuje wartość obliczoną przez moduł sm_to_u2.sv

Typ: logic signed

Ilość bitów: 4

Zakres: [-7, 7]

**sum**

Opis

Przechowuje wartość obliczoną przez moduł adder.sv

Typ: logic signed

Ilość bitów: 4

Zakres: [-7, 7]

**overflow**

Opis

Przechowuje informacje o przepełnieniu wynikającą z operacji dodawania. Wartość jest przypisywana na podstawie operacji wykonanych przez moduł adder

Typ: logic unsigned

Ilość bitów: 4

Zakres: [0,1]

**Overflow_2**

Opis

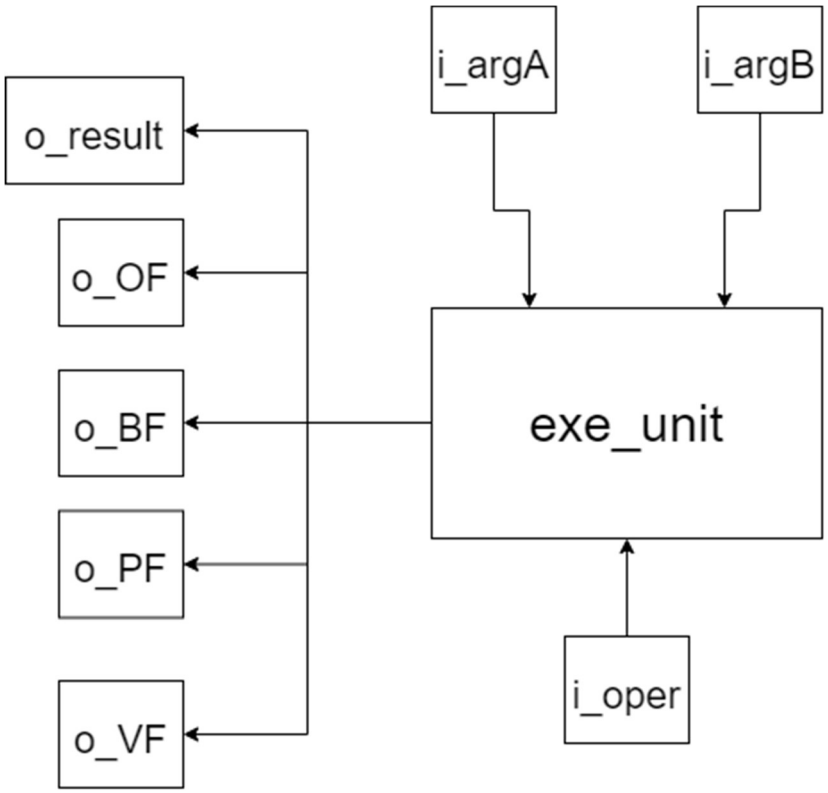
Przechowuje informacje o przepełnieniu wynikającą z zliczania zer. Wartość jest przypisywana na podstawie operacji wykonanych przez moduł count_zeros

Typ: logic unsigned

Ilość bitów: 4

Zakres: [0,1]

## Schemat blokowy




*Rysunek 1 Schemat blokowy*

# Lista realizowanych funkcji bezpośrednio w module exe_unit

**Funkcja XOR**

Funkcja realizuje operację NOR na dwóch liczbach, w kodzie U2, podanych jako argumenty tej funkcji. Liczba pierwsza: **i_a** oraz liczba druga: **i_b.** Wynik jest zwracany w postaci liczby 4-bitowej zapisanej w kodzie U2.

W celu wywołania funkcji na wejście **i_oper** należy podać kod 0001 zapisany w postaci binarnej.

**Funkcja NAND**

Funkcja realizuje operację NAND na dwóch liczbach, w kodzie U2, podanych jako argumenty tej funkcji. Liczba pierwsza: **i_a** oraz liczba druga: **i_b.** Wynik jest zwracany w postaci liczby 4-bitowej zapisanej w kodzie U2.

W celu wywołania funkcji na wejście **i_oper** należy podać kod 0010 zapisany w postaci binarnej.

**Funkcja arytmetycznego przesunięcia bitowego w prawo**

Funkcja realizuje operację przesunięcia arytmetycznego w prawo liczby i_argA  o liczbę i_argB.

W celu wywołania funkcji na wejście **i_oper** należy podać kod 0100 zapisany w postaci binarnej.

**Funkcja logicznego przesunięcia bitowego w lewo**

Funkcja realizuje operację przesunięcia logicznego w lewo liczby i_argA o liczbę i_argB.

W celu wywołania funkcji na wejście **i_oper** należy podać kod 0010 zapisany w postaci binarnej.

# Moduły instancjonowane

## count_zeros

Kod do podania na wejście: 0110

Moduł realizuje operację zliczania zer w jego argumentach wejściowych zapisanych w postaci binarnej 4-bitowej.

**Wejścia**

- i_a
  - typ: logic signed
  - ilośc bitów: 4
  - zakres: [-7, 7]
- i_b
  - typ: logic signed
  - ilośc bitów: 4
  - zakres: [-7, 7]

**Wyjścia**

- o_zeros
  - typ: logic signed
  - ilośc bitów: 4
  - zakres: [-7, 7]
- o_carry
  - typ: logic unsigned
  - ilośc bitów: 1
  - zakres: [0,1]

**Parametry:**

LEN – określa długość sygnału wejściowego

Na początku zostają wczytane zmienne podane na wejściach modułu: i_a, i_b. Zapisane są w postaci binarnej 4 bitowej. Następnie inicjowana jest zmienna pomocnicza **i** oraz **zeros**.

```
input logic signed [LEN-1:0] i_a, i_b;
output logic signed [LEN-1:0] o_zeros;

// Zmienna pomocnicza
integer i;
logic signed [LEN-1:0]zeros;
```

*Rysunek 2 Zmienne modułu count_zeros*

Zmienna i wykorzystywana jest przy iteracji pętli, natomiast zmienna **zeros** jest licznikiem.

W kolejnym kroku wartość zmiennej **zeros** ustawiana jest na 0 oraz rozpoczyna się pętla, która iteruje po każdym znaku w wejściach **i_a** oraz **i_b**. Za pomocą instrukcji warunkowej znajdywane są zera oraz ich ilość zapisywana jest w **zeros**.

```
zeros = '0;
for(i=0; i<LEN; i=i+1)begin

    if(i_a[i]==0)begin
    zeros = zeros + 4'b0001;
    end

    if(i_b[i]==0)begin
    zeros = zeros + 4'b0001;
    end
```

*Rysunek 3 Pętla zliczająca modułu count_zeros*

Ostatecznie ilość zer jest zwracana na wyjście **o_zeros.**

```
{o_carry, o_zeros} = zeros;
```

*Rysunek 4 Wyjście modułu count_zeros*

Dodatkowo zwracana jest informacja o przepełnieniu, jeżeli liczba zer nie może pomieścić się w liczbie bitów wyjścia.

## gray_coder2

Kod do podania na wejście: 0101

Moduł realizuje operacje zamiany kodu binarnego na kod Graya. Posiada wejście i_data oraz wyjście o_gray.

Wejścia

- i_data
  - liczba_bitów: 4
  - zakres: [-7, 7]

Wyjścia

- o_gray
  - liczba_bitów: 4

- zakres: [-7, 7]

**Parametry:**

LEN – określa długość sygnału wejściowego

Operacja zamiany realizowana jest na podstawie algorytmu, gdzie liczba przesuwana jest bitowo w lewo o 1 a następnie wykonywana jest operacja XOR z wektorem przesuniętym.

```
o_gray  = i_data ^ (i_data >> 1);
```

*Rysunek 5 Operacja zamiany kodu graya na nkb*

Ostatecznie wynik podawany jest na wyjście.

Jeżeli liczba podana na wejście jest ujemna to zwracana jest wartość ‘1 jako kod błędu.

## one_hot

Kod do podania na wejście: 0111

Moduł realizuje dekodowanie kodu one hot.

Wejścia:

- i_onehot
  - liczba bitów: 4
  - zakres: [-7,7]

Wyjścia

- o_nkb
  - liczba bitów: 4
  - zakres: [-7, 7]

**Sygnały pomocnicze**

**s_was_1**

Opis

sygnał przyjmujący wartość logiczną 0 lub 1, użyty został do wykrycia 1 w kodzie one-hot.

Typ: logic unsigned

Ilość bitów: 1

Zakres: [0,1]

**Parametry:**

LEN – określa długość sygnału wejściowego

BITS – określa długość sygnału wyjściowego

Na początku moduł ustawia wartości o_nkb oraz s_was1 na ‘0 w celu uniknięcia błędu oraz możliwości realizacji układu kombinacyjnego.

```
{o_nkb, s_was1}  = '0;
```

*Rysunek 6 Przypisanie wartości zerowych do zmiennych wejściowych*

Następnie zaczyna się pętla, gdzie analizowany jest kod onehot. Za pomocą instrukcji warunkowej sprawdzane jest miejsce, gdzie znajduje się jedynka i potem jest ono zapisywane na wyjście.

```
for (int i=0; i < LEN; i++)
begin : LOOP
    if (i_onehot[i] == 1'b1)
    // jesli wczesnij byla jedynka
    // znaczy ze bledne dane wejsciowe
        if (s_was1) o_nkb  = '1;
        else
        begin
            s_was1 = '1;
            o_nkb  = i;
        end
end : LOOP
```

*Rysunek 7 Analiza kodu one-hot*

W przypadku nieznalezienia jedynki lub znalezienia więcej niż jedna kod błędu w postaci ‘1 przypisywany jest do wyjścia.

```
if (!s_was1) o_nkb = '1;
```

*Rysunek 8 Sprawdzenie warunku kodu one-hot*

## crc_coder

Kod do podania na wejście: 1000

Moduł realizuje kodowanie wejścia do kodu crc 4 bitowego.

Wejścia:

- i_data
  - typ: logic signed
  - liczba bitów: 4
  - zakres: [-7,7]
- i_poly
  - typ: logic unsigned
  - liczba bitów: 5
  - przyjmuje wartość stałą: 10011

Wyjścia

- o_crc

- typ: logic unsigned
- liczba bitów: 8
- zakres: [-7, 7]

**Sygnały pomocnicze**

**poly_tmp**

Opis

Wielomian wykorzystywany do kodowania wejścia modułu

Typ: logic signed

Ilość bitów: 8

Zakres: przyjmuje wartość wejścia i_poly z dopisanymi czterema zerami.

**crc_tmp**

Opis

Zmienna tymczasowa służąca do przechowania wartości i_data powiększonej o 4 zera z prawej strony

Typ: logic unsigned

Ilość bitów: 8

Zakres: przyjmuje wartość wejścia i_data z dopisanymi czterema zerami.

**Parametry:**

WCODE – określa długość sygnału wejściowego

WPOLY – określa długość wielomianu

Na początku ustawiane są wartości zmienny na 0’ w celu uniknięcia błędów.

```
{o_crc, crc_tmp, poly_tmp}  = '0;
```

*Rysunek 9 Przypisanie wartości zerowych w module crc 4 bitowym*

Następnie wartości zapisywane są do zmiennych tymczasowych

```
crc_tmp     = {i_data, 4'b0000};
poly_tmp    = {i_poly, {(WCODE-1){1'b0}}};
```

*Rysunek 10 Utworzenie zmiennych pomocniczych*

Następnie zaczyna się pętla for, która iteruje po zmiennej i_data w celu znalezienia jedynki, gdy ta wartość jest znaleziona to następuje wykonanie operacji XOR  z zmienną poly_tmp.

```
if (i_data[i] == 1'b1)
        crc_tmp = crc_tmp ^ poly_tmp;
```

*Rysunek 11 Intrukcja warunkowa sprawdzająca czy wartości bitu jest 1*

Jeżeli jedynka nie zostanie odnaleziona to następuje [przesunięcie bitowe w prawo o 1.

```
poly_tmp = poly_tmp >> 1;
```

*Rysunek 12 Wykonanie operacji przesunięcia bitowego*

Ostatecznie kod crc-4 bitowy jest zapisywany do zmiennej wyjściowej.

```
o_crc = crc_tmp[WPOLY-2:0];
```

*Rysunek 13 Zwrócenie wartości na wyjscie modułu*

## Crc3_eval

Kod do podania na wejście: 1001

Moduł realizuje sprawdzenie kodu crc-3 bitowego.

Wejścia:

- i_data
  - typ: logic signed
  - liczba bitów: 4
  - zakres: [-7,7]
- i_crc
  - typ: logic unsigned
  - liczba bitów: 3
  - zakres: [0, 7]
- i_poly
  - typ: logic unsigned
  - liczba bitów: 4
  - przyjmuje wartość stałą: 1011

Wyjścia

- o_crc
  - typ: logic signed
  - liczba bitów: 3
  - zakres: [-3, 3]

**Sygnały pomocnicze**

**poly_tmp**

Opis

Wielomian wykorzystywany do kodowania wejścia modułu

Typ: logic signed

Ilość bitów: 7

Zakres: przyjmuje wartość wejścia i_poly z dopisanymi trzema zerami.

**crc_tmp**

Opis

Zmienna tymczasowa służąca do przechowania wartości i_data powiększonej o 4 zera z prawej strony

Typ: logic unsigned

Ilość bitów: 7

Zakres: przyjmuje wartość wejścia i_data z dopisanymi trzema zerami.

**Parametry:**

WCODE – określa długość sygnału wejściowego

WPOLY – określa długość wielomianu

Na początku ustawiane są wartości zmienny na 0’ w celu uniknięcia błędów.

```
{o_crc, crc_tmp, poly_tmp}  = '0;
```

*Rysunek 14 Przypisanie wartości zerowych od zmiennych*

Następnie wartości zapisywane są do zmiennych tymczasowych

```
crc_tmp     = {i_data, i_crc};
poly_tmp    = {i_poly, {(WCODE-1){1'b0}}};
```

*Rysunek 15 Przypisanie wartości do zmiennych pomocniczych modułu crc-3*

Następnie zaczyna się pętla for, która iteruje po zmiennej i_data w celu znalezienia jedynki, gdy ta wartość jest znaleziona to następuje wykonanie operacji XOR  z zmienną poly_tmp.

```
if (i_data[i] == 1'b1)
        crc_tmp = crc_tmp ^ poly_tmp;
```

*Rysunek 16 Wykonanie operacji XOR na zmiennych*

Jeżeli jedynka nie zostanie odnaleziona to następuje [przesunięcie bitowe w prawo o 1.

```
poly_tmp = poly_tmp >> 1;
```

*Rysunek 17 Operacja przesunięcia bitowego*

Ostatecznie kod crc-3 bitowy jest zapisywany do zmiennej wyjściowej.

```
o_crc = crc_tmp[WPOLY-2:0];
```

*Rysunek 18 Przypisanie wartości obliczonej na wyjście modułu*

## Sm_to_u2

Kod do podania na wejście: 1011

Wejścia:

- i_a
  - typ: logic signed
  - liczba bitów: 4
  - zakres: [-7,7]

Wyjścia

- o_u2code
  - typ: logic signed
  - liczba bitów: 4
  - zakres [-7, 7]

Parametry

LEN – przechowuje informację o liczbie bitów sygnałów wejściowych.

**Sygnały pomocnicze**

tmp – sygnał przechowujący wartość i_a w celu wykonywania na niej operacji.

**tmp**

Opis

sygnał przechowujący wartość i_a w celu wykonywania na niej operacji

Typ: logic signed

Ilość bitów: 4

Zakres: [-7, 7]

Moduł jest odpowiedzialny z konwersje kodu znak_moduł na kod U2. Na początku sprawdzane jest czy liczba podana na wejście jest wartością dodatnią. Jeżeli jest to wartość dodatnia to sygnał zwracany jest na wyjście, ponieważ te wartości niezależnie od kodowania są takie same.

```
if (i_a[LEN-1] == 0) begin
    o_u2code = i_a;
```

*Rysunek 19 Sprawdzenie czy najstarszy bit jest wartością 0*

Jeżeli wartość podana na wejście jest ujemna to moduł przeprowadza zamianę. Wejście i_a jest negowane i zapisywane do zmienne tmp, następnie dodawana jest wartość 1 i wartość najstarszego bitu jest zmienna na 1. Wartość uzyskana jest zwracana na wyjście.

```
tmp = ~(i_a);
tmp = tmp + 1'b1;
tmp[LEN-1] = 1;
o_u2code = tmp;
```

*Rysunek 20 Konwersja kodu znak moduł na u2*

### Adder

Kod do podania na wejście: 0000

Wejścia:

- i_a
  - typ: logic signed
  - liczba bitów: 4
  - zakres: [-7,7]
- i_b
  - typ: logic signed
  - liczba bitów: 4
  - zakres: [-7,7]

Wyjścia

- o_u2code
  - typ: logic signed
  - liczba bitów: 4
  - zakres [-7, 7]
- o_carry
  - typ: logic unsigned
  - liczba bitów: 4
  - zakres: [-7,7]

**Parametry**

LEN – przechowuje informację o liczbie bitów sygnałów wejściowych.

Moduł jest odpowiedzialny za realizacje operacji dodawania na dwóch liczbach wejściowych i_a, i_b. Dodatkowo posiada on wyjście o_carry, które informuje o wystąpieniu przeniesienia. Informacja ta wykorzystana jest to określenia wartości flagi VF.

```
{o_carry, o_sum} = i_a + i_b;
```

*Rysunek 21 Przypisanie wyników do wyjśc modułu*

## Specyfikacje sygnałów określających flagi

**Flaga OF**

Flaga, która jest odpowiedzialna za informację dotyczącą czy wyjście o_result składa się z samych jedynek. Wartość flagi przyjmuje wartość 1, jeżeli na wyjściu pojawiła się wartość ‘1 lub 0 jeżeli pojawiła się inna wartość.

Przykład 1

Na wejście podajemy sygnały i_argA = 1011. Następnie wykonujemy operację sprawdzenia kodu onehot, czyli podajemy sygnał sterujący i_oper = 0111. Wykonywana jest odpowiednia operacja i zostaje zwrócona wartość ‘1, gdyż mamy więcej niż jedną jedynkę na wejściu. Dlatego flaga OF przyjmie wartość 1.

Przykład 2

Na wejście podajemy sygnały i_argA = 0001. Następnie wykonujemy operację onehot, czyli podajemy sygnał sterujący i_oper = 0111. Wykonywana jest odpowiednia funkcja i wynik o_result = 0000 nie składa się z samych jedynek, dlatego flaga OF przyjmie wartość 0.

**Flaga BF**

Flaga, która jest odpowiedzialna za informację czy w wartości wyjściowej znajduje się tylko jedna jedynka. Wartość flagi przyjmuje wartość 1, jeżeli warunek jest spełniony lub wartość 0 jeżeli w wartości wyjściowej jest więcej lub wcale wartości 1.

Przykład 1

Na wejście podajemy sygnały i_argA = 0000 oraz i_argB = 0001. Następnie wykonujemy operację dodawania, czyli podajemy sygnał sterujący i_oper = 0000. Suma tych liczb daje nam wynik o_result = 0001. W tym momencie jest wywoływana funkcja sprawdzająca warunek flagi OF. Sygnał wyjściowy zawiera tylko jedną jedynkę, dlatego flaga przyjmuje wartość 1 i podawana jest na wyście układu.

Przykład 2

Na wejście podajemy sygnały i_argA = 0010 oraz i_argB = 0001. Następnie wykonujemy operację dodawania, czyli podajemy sygnał sterujący i_oper = 0000. Suma tych liczb daje nam wynik o_result = 0011. W tym momencie jest wywoływana funkcja sprawdzająca warunek flagi OF. Sygnał wyjściowy zawiera tylko więcej niż jedną jedynkę, dlatego flaga przyjmuje wartość 0 i podawana jest na wyście układu.

Przykład 3

Na wejście podajemy sygnały i_argA = 0000 oraz i_argB = 0000. Następnie wykonujemy operację dodawania, czyli podajemy sygnał sterujący i_oper = 0000. Suma tych liczb daje nam wynik o_result = 0000. W tym momencie jest wywoływana funkcja sprawdzająca warunek flagi OF. Sygnał wyjściowy nie zawiera jedynki, dlatego flaga przyjmuje wartość 0 i podawana jest na wyście układu.

**Flaga PF**

Flaga, która jest służy jako znacznik uzupełnienia do parzystej liczby jedynek. Ma ona sprawdzić czy w sygnale wyjściowym o_result znajduje się parzysta liczba jedynek, jeżeli tak to przyjmuje wartość 0, jeżeli nie to przyjmuje wartość 0. Wartość 0 jest również wartością domyślną.

Przykład 1

Na wejście podajemy sygnały i_argA = 0010 oraz i_argB = 0011. Następnie wykonujemy operację NOR, czyli podajemy sygnał sterujący i_oper = 0001. Jako wynik operacji uzyskujemy o_result = 0100. W takim przypadku flaga przyjmie wartość 1, ponieważ do parzystej liczby jedynek brakuje jednej.

Przykład 2

Na wejście podajemy sygnały i_argA = 0010 oraz i_argB = 0011. Następnie wykonujemy operację NOR, czyli podajemy sygnał sterujący i_oper = 0001. Jako wynik operacji uzyskujemy o_result = 0110. W takim przypadku flaga przyjmie wartość 0, ponieważ na wyjściu mamy parzystą liczbę jedynek.

**Flaga VF**

Flaga, która jest odpowiedzialna za informacje czy liczba podana na wyjściu jest podawana z przepełnieniem. Flaga przyjmuje wartość 1, jeżeli nastąpiło przepełnienie lub 0 jeżeli przepełnienia nie było.

Przykład 1

Na wejście podajemy sygnały i_argA = 0000 oraz i_argB = 0000. Następnie wykonujemy operację zliczania zer, czyli podajemy sygnał sterujący i_oper = 0110. Jako wynik operacji uzyskujemy o_result = 1000. W takim przypadku nastąpiło przepełnienie więc flaga przyjmie wartość 1.

Przykład 2

Na wejście podajemy sygnały i_argA = 0001 oraz i_argB = 0011. Następnie wykonujemy operację zliczania zer, czyli podajemy sygnał sterujący i_oper = 0110. Jako wynik operacji uzyskujemy o_result = 0101. W takim przypadku nie nastąpiło przepełnienie więc flaga przyjmie wartość 0.

## Lista plików

**/MODEL/exe_unit.sv**

W pliku znajdują się funkcje, które są realizowane przez model exe_unit. exe_unit.sv jest plikem opisu modelu, służy do użycia w programie yosys w celu dokonania syntezy.

Instancjonowane moduły:

- Crc_coder
- Count_zeros
- Crc3_eval
- Gray_coder
- One_hot
- Adder
- Exe_unit

**/MODEL/crc_coder.sv**

Plik zawierający moduł służący do kodowania CRC 4 bitowego.

**/MODEL/count_zeros.sc**

Plik zawierający moduł służący do zliczania ilości zer w jego wejściach.

**/MODEL/crc3_eval.sv**

Plik zawierający moduł służący do dekodowania kodu CRC 3 bitowego.

**/MODEL/gray_coder.sv**

Plik zawierający moduł służący do zamiany liczby zapisane w kodzie U2 na liczbę zapisaną w kodzie Graya.

**/MODEL/one_hot.sv**

Plik zawierający model służący do dekodowania kodu one_hot na kod NKB.

**/MODEL/adder.sv**

Plik zawierający model służący do wykonywania operacji dodawania.

**/RTL/exe_unit_rlt.sv**

Plik zawierający model exe_unit po syntezie logicznej. Zawiera on dokładne opisy każdego bloku logicznego.

**/RTL/ crc_coder _rlt.sv**

Plik zawierający model crc_coder po syntezie logicznej. Zawiera on dokładne opisy każdego bloku logicznego.

**/RTL//crc3_eval _rlt.sv**

Plik zawierający model crc3_eval po syntezie logicznej. Zawiera on dokładne opisy każdego bloku logicznego.

**/RTL/ gray_coder _rlt.sv**

Plik zawierający model gray_coder po syntezie logicznej. Zawiera on dokładne opisy każdego bloku logicznego.

**/RTL/ one_hot _rlt.sv**

Plik zawierający model **o**ne_hot po syntezie logicznej. Zawiera on dokładne opisy każdego bloku logicznego.

**/RTL/ adder.sv**

Plik zawierający model adder po syntezie logicznej. Zawiera on dokładne opisy każdego bloku logicznego.