DOKUMENTACJA

**Autor:** Konrad Krupski

**Numer indeksu:** 310729

**Kontakt:**

Email: [01158726@pw.edu.pl](mailto: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 2](#_Toc86581529)

[Wejścia 2](#_Toc86581530)

[Wyjścia 2](#_Toc86581531)

[Sygnały pomocnicze 2](#_Toc86581532)

[Sygnały wewnętrzne 2](#_Toc86581533)

[Schemat blokowy 5](#_Toc86581534)

[Lista realizowanych funkcji bezpośrednio w module exe\_unit 5](#_Toc86581535)

[Moduły instancjonowane 6](#_Toc86581536)

[count\_zeros 6](#_Toc86581537)

[gray\_coder2 7](#_Toc86581538)

[one\_hot 8](#_Toc86581539)

[crc\_coder 9](#_Toc86581540)

[Crc3\_eval 11](#_Toc86581541)

[Sm\_to\_u2 13](#_Toc86581542)

[Adder 14](#_Toc86581543)

[Specyfikacje sygnałów określających flagi 14](#_Toc86581544)

[Lista plików 17](#_Toc86581545)

# 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 nieparzystą 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

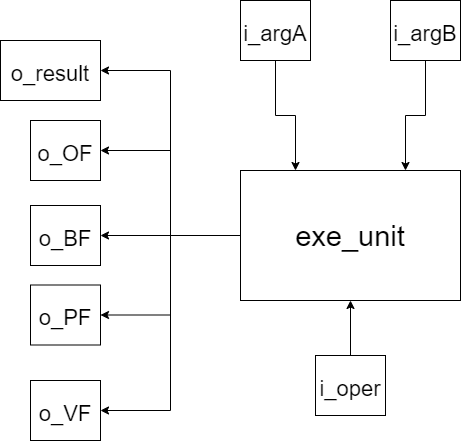
Przechowuje informacje o przepełnieniu wynikającą z zliczania zer. Wartość jest przypisywana na podstawie operacji wykonanych przez moduł count\_zeros

Typ: logic unsigned

Ilość bitów: 4

Zakres: [0,1]

## Schemat blokowy



Rysunek 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**.

Text

Description automatically generated

Rysunek 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**.

Text

Description automatically generated

Rysunek Pętla zliczająca modułu count\_zeros

Ostatecznie ilość zer jest zwracana na wyjście **o\_zeros.**

****

Rysunek 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.



Rysunek 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.



Rysunek 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.

A screenshot of a computer

Description automatically generated with medium confidence

Rysunek Analiza kodu one-hot

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



Rysunek 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.



Rysunek Przypisanie wartości zerowych w module crc 4 bitowym

Następnie wartości zapisywane są do zmiennych tymczasowych



Rysunek 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.



Rysunek Intrukcja warunkowa sprawdzająca czy wartości bitu jest 1

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



Rysunek Wykonanie operacji przesunięcia bitowego

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



Rysunek 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.



Rysunek Przypisanie wartości zerowych od zmiennych

Następnie wartości zapisywane są do zmiennych tymczasowych



Rysunek 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.



Rysunek Wykonanie operacji XOR na zmiennych

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



Rysunek Operacja przesunięcia bitowego

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



Rysunek 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.

Graphical user interface

Description automatically generated with medium confidence

Rysunek 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.

Text

Description automatically generated

Rysunek 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.



Rysunek 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.