

Systemy cyfrowe i komputerowe

Dokumentacja projektu “exe_unit_w1”

Karol Ambroziński
Nr. albumu: 318488

Spis treści

1	Wejścia, wyjścia, parametry i zakresy ich wartości	2
1.1	Parametry	2
1.2	Wejścia	2
1.3	Wyjścia	2
2	Realizowane funkcje i ich argumenty	2
2.1	Podmoduł <i>mod1</i> :	2
2.2	Podmoduł <i>mod2</i> :	3
2.3	Podmoduł <i>mod3</i> :	3
2.4	Podmoduł <i>mod4</i> :	3
3	Schemat blokowy struktury jednostki	4
4	Sygnały zaimplementowanych flag i ich wartości	4
5	Przykład użycia modułu	5
5.1	Przykład nr. 1	5
5.2	Przykład nr. 2	5
5.3	Przykład nr. 3	6
6	Lista plików	6
7	Raport z syntezy logicznej	7
8	Synteza projektu i komendy	9

1 Wejścia, wyjścia, parametry i zakresy ich wartości

1.1 Parametry

- *m* - określa wielkość w bitach główne wejścia i wyjścia danych,
- *n* - określa ilość operacji.

1.2 Wejścia

- *i_oper* - *n*-bitowe wejście określające wykonywaną operację,
- *i_argA* - *m*-bitowe wejście; kodowanie ZNAK-MODUŁ,
- *i_argB* - *m*-bitowe wejście; kodowanie ZNAK-MODUŁ,
- *i_clk* - 1 bitowe wejście zegarowe,
- *i_rsn* - 1 bitowe wejście resetu synchronicznego.

1.3 Wyjścia

- *o_status* - *m*-bitowe wyjście,
- *o_result* - 4 bitowe wyjście; kodowanie ZNAK-MODUŁ lub U2.

2 Realizowane funkcje i ich argumenty

Układ realizuje 4 operacje (4 podmoduły):

2.1 Podmoduł *mod1*:

Odejmowanie argumentów ($A - B$); jeśli operacja nie może zostać wykonana, jednostka zgłasza błąd, a wyjście jest niezdefiniowane. Kod modułu *i_oper*: 00_2 .

Wejścia

- *i_argA* - *m*-bitowe wejście,
- *i_argB* - *m*-bitowe wejście,

Wyjścia

- *o_result* - *m*-bitowe wyjście,
- *o_status* - 4 bitowe wyjście.

2.2 Podmoduł *mod2*:

Porównanie argumentów ($A < B$); jeśli warunek jest spełniony to wynikiem jest liczba 1_{10} , w przeciwnym wypadku wynikiem jest 0_{10} . Kod modułu *i_oper*: 01_2 .

Wejścia

- *i_argA* - m-bitowe wejście,
- *i_argB* - m-bitowe wejście,

Wyjścia

- *o_result* - m-bitowe wyjście,
- *o_status* - 4 bitowe wyjście.

2.3 Podmoduł *mod3*:

Ustawienie bitu w argumencie A na wartość 0_2 ; numer bitu jest określony w argumencie B; zgłoszenie błędu jeśli wartość B jest ujemna lub przekracza liczbę bitów argumentu A. Kod modułu *i_oper*: 10_2 .

Wejścia

- *i_argA* - m-bitowe wejście,
- *i_argB* - m-bitowe wejście,

Wyjścia

- *o_result* - m-bitowe wyjście,
- *o_status* - 4 bitowe wyjście.

2.4 Podmoduł *mod4*:

Konwersja argumentu A z kodu ZNAK-MODUŁ na U2; jeśli konwersja nie może zostać wykonana - zgłaszany jest błąd, a wynik jest nieokreślony. Kod modułu *i_oper*: 11_2 .

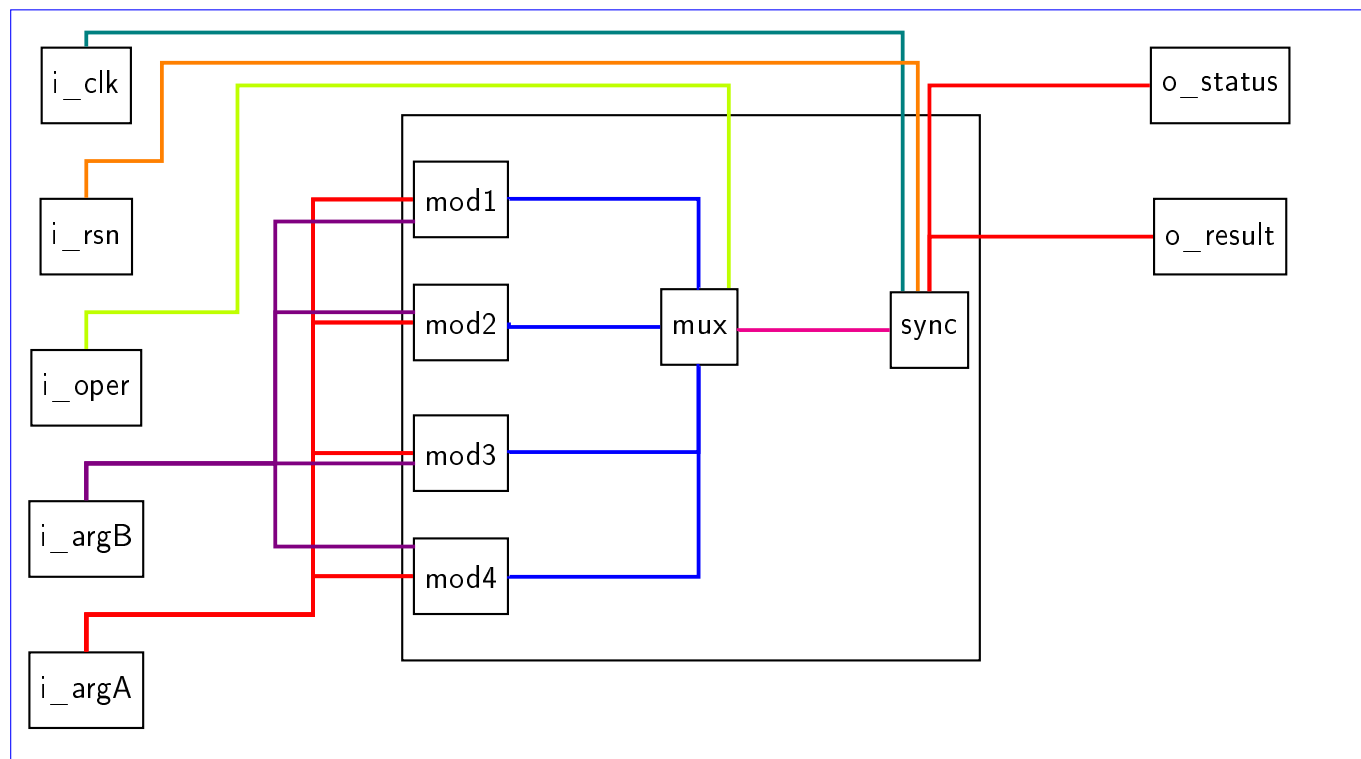
Wejścia

- *i_argA* - m-bitowe wejście,

Wyjścia

- *o_result* - m-bitowe wyjście,
- *o_status* - 4 bitowe wyjście.

3 Schemat blokowy struktury jednostki



Ilustracja nr. 1: Schemat blokowy *exe_unit_w1*

4 Sygnały zaimplementowanych flag i ich wartości

Zaimplementowane flagi **o_status**; każdy bit jest przypisany do danej flagi:

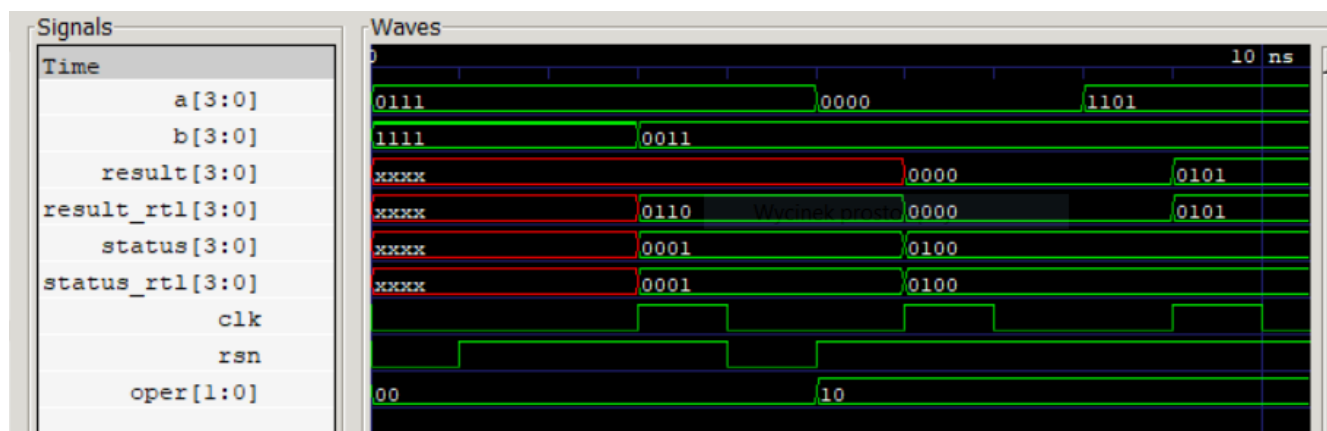
- **ERROR** - operacja nie została wykonana - **o_status** = 0001₂,
- **NEG** - wynik jest liczbą ujemną; **o_status** = 0010₂,
- **EVEN** - w wyniku jest parzysta liczba jedynek; **o_status** = 0100₂,
- **ONES** - wszystkie bity o_result ustawione; **o_status** = 1000₂.

Jeśli wynik jest nieokreślony (flaga **ERROR**) to pozostałe bity nie są ustawiane (warunki pozostałych flag nie są sprawdzane); jednocześnie może być ustawione wiele flag.

5 Przykład użycia modułu

5.1 Przykład nr. 1

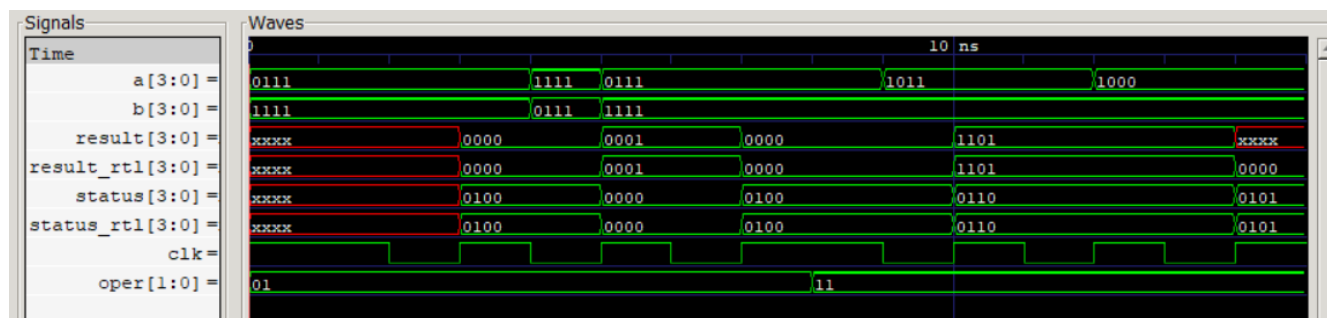
Na ilustracji nr. 2 przedstawiono wykonanie dwóch operacji: odejmowanie liczb A i B oraz zmianę bitu w argumencie A oznaczonego indeksem B (moduły: **mod1** i **mod3**). Przy pierwszej operacji na początku wynik jest niezdefiniowany i flaga błędu ustawiona na 1; przepełnienie wartości. W kolejnej operacji w argumencie B został zmieniony bit na indeksie B: B równe jest 3_{10} , więc bit nr. 3_{10} (liczony od 0_{10}) w A został zmieniony na 0_2 . Przy operacji ustawiania bitu poprzez B widać ustawienie flagi 0100_2 która oznacza że wynik posiada parzystą liczbę jedynek (co też jest widoczne na wyjściu **o_result**).



Ilustracja nr. 2: Widok wykonania testbenchu RTL i oryginalnych plików w GTKWave

5.2 Przykład nr. 2

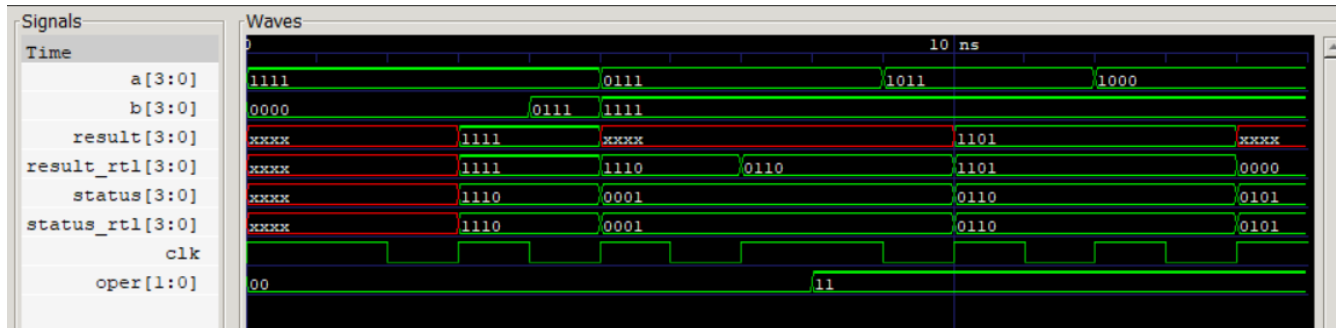
Na ilustracji nr. 3 wykonywane są dwie operacje: porównanie liczb A i B oraz konwersja liczby w kodowaniu ZNAK-MODUŁ na U2 (moduły: **mod2** i **mod4**). Na początku liczba A jest większa od liczby B (liczba A jest dodatnia, a liczba B jest ujemna) więc na wyjściu jest 0_{ZM} . Przy następnych wartościach A i B (A mniejsze od B) wyjście równe jest 1_{10} . Następnie zmieniana jest operacja (ZM na U2) gdzie wejście A równe jest 1011_{ZM} które później konwertowane jest na 1101_{U2} . Po tej konwersji na wejście podane jest ujemne zero (1000_{ZM}) które dla **mod4** jest błędem, więc flaga **o_status** ustawiana jest na 0001_2 , a wyjście jest niezdefiniowane.



Ilustracja nr. 3: Widok wykonania testbenchu RTL i oryginalnych plików w GTKWave

5.3 Przykład nr. 3

Na ilustracji nr. 4 widać wykonywanie operacji odejmowania liczb A - B oraz zamianę liczby A z kodowaniem ZNAK-MODUŁ na U2 (moduły: **mod1** i **mod4**). Na początku A ma wartość 1111_{ZM} , a B 0000_{ZM} , więc ich różnica równa jest początkowemu A; widać tutaj ustawienie wszystkich flag **o_status**, oprócz bitu **ERROR**. Następnie zmieniają się wartości A i B, których różnica powoduje przepełnienie, więc błąd (wyjście - niezdefiniowane, a flaga **ERROR** ustawiona na 1₂). W kolejnym takcie zegara, zmianie ulega wejście A oraz operacja na zamianę kodowanie z ZNAK-MODUŁ na U2: wartość 1011_{ZM} zamieniana jest na 1101_{U2} . Później zmieniana jest wartość A na ujemne zero (1000_{ZM}); błąd - wyjście niezdefiniowane.



Ilustracja nr. 4: Widok wykonania testbenchu RTL i oryginalnych plików w GTKWave

6 Lista plików

- **exe_unit_w1.sv** - plik zawierający główny moduł,
- **otherModules.sv** - plik zawierający wszystkie podmoduły głównego modułu (**mod1**, **mod2**, **mod3** i **mod4**),
- **exe_unit_w1_rtl.sv** - plik główny modułu po syntezie,
- **synth.log** - plik raportu Yosysa po syntezie.

7 Raport z syntezy logicznej

Podmoduł <i>mod1</i>	
Number of wires:	88
Number of wire bits:	115
Number of public wires:	12
Number of public wire bits:	39
Number of memories:	0
Number of memory bits:	0
Number of processes:	0
Number of cells:	84
\$_AND_	33
\$_NOT_	16
\$_OR_	24
\$_XOR_	11
Estimated number of transistors:	506

Podmoduł <i>mod2</i>	
Number of wires:	48
Number of wire bits:	103
Number of public wires:	11
Number of public wire bits:	66
Number of memories:	0
Number of memory bits:	0
Number of processes:	0
Number of cells:	39
\$_AND_	14
\$_NOT_	8
\$_OR_	15
\$_XOR_	2
Estimated number of transistors:	214

Podmoduł <i>mod3</i>	
Number of wires:	30
Number of wire bits:	79
Number of public wires:	8
Number of public wire bits:	57
Number of memories:	0
Number of memory bits:	0
Number of processes:	0
Number of cells:	30
\$_AND_	13
\$_NOT_	7
\$_OR_	7
\$_XOR_	3
Estimated number of transistors:	170

Podmoduł <i>mod4</i>	
Number of wires:	17
Number of wire bits:	32
Number of public wires:	6
Number of public wire bits:	21
Number of memories:	0
Number of memory bits:	0
Number of processes:	0
Number of cells:	16
\$_AND_	7
\$_NOT_	4
\$_OR_	4
\$_XOR_	1
Estimated number of transistors:	86

Moduł główny: <i>exe_unit_w1</i> (całość z <i>MUX</i>)	
Number of wires:	254
Number of wire bits:	443
Number of public wires:	54
Number of public wire bits:	243
Number of memories:	0
Number of memory bits:	0
Number of processes:	0
Number of cells:	239
<i>\$_AND_</i>	103
<i>\$_NOT_</i>	37
<i>\$_OR_</i>	74
<i>\$_SDFF_PN0_</i>	8
<i>\$_XOR_</i>	17
Estimated number of transistors:	1340+

8 Synteza projektu i komendy

Założenia systemowe: Linux (testowano na Ubuntu (*makefile*) i Windows, (gdzie komendy z *makefile* są wpisywane ręcznie).

Założenia dotyczące plików:

- *makefile*, znajdujący się w folderze **WORK**,
- *run.js*, znajdujący się w folderze **WORK**),
- *exe_unit_w1.sv*, znajdujący się w folderze **MODEL**,
- *otherModules.sv*, znajdujący się w folderze **MODEL**,
- *testbench.sv*, znajdujący się w folderze **MODEL**,

znajdują się w jednym folderze. Aby zsyntezować projekt, wykonać testbench i wyświetlić przebiegi wystarczy użyć komendy:

```
$ ./makefile
```

Testowano przy użyciu: Icarus Verilog w wersji 11 i 12 oraz Yosys w wersji 0.23.