DOKUMENTACJA PROJEKTOWA 2 - TESTBENCH

Agnieszka Klępka
310712
01158709@pw.edu.pl

PLATFORMA SPRZĘTOWA:

Synteza i symulacja układu została przeprowadzona na systemie macOS Catalina wersja 10.15.2

SPOSÓB URUCHAMIANIA:

Należy uruchomić terminal w katalogu /WORK i wpisać następujące polecenia do przeprowadzenia następujących działań:

1. Samodzielna synteza: make rtl
2. Test działania: make sim
3. Wyświetlanie przebiegów sygnałów w programie gtkwave: make wave

Komendy te zapisane są w pliku makefile.

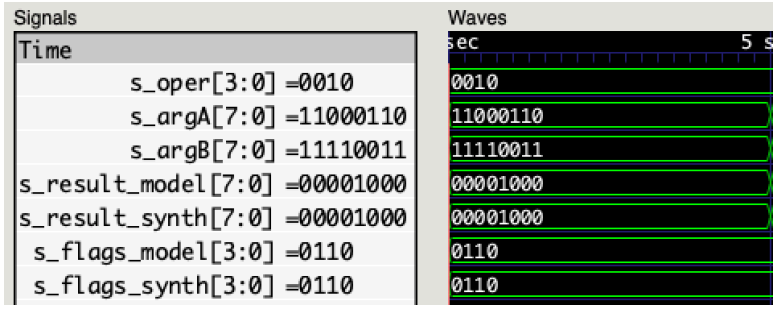
FLAGI

BF0 - znacznik informujący, że w wyniku jest tylko jedno zero

BF1 - znacznik informujący, że w wyniku jest tylko jedna jedynka

PF - znacznik uzupełnienia do parzystej liczby jedynek

NF - znacznik uzupełnienia do nieparzystej liczby jedynek

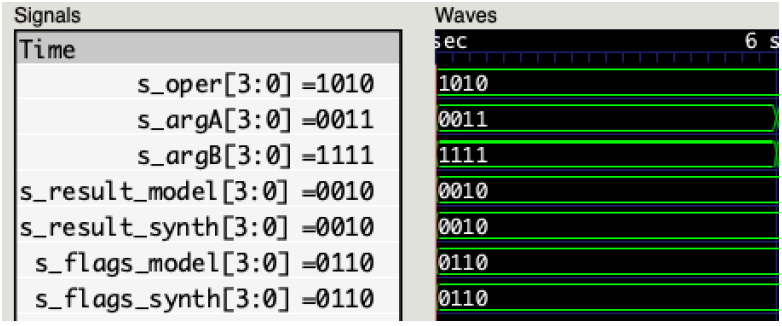


BF0 = 0 ->więcej niż jedno 0 w wyniku

BF1 = 1 ->w wyniku znajduje się jedna 1

PF = 1 -> uzupełnienie do parzystej liczby (nieparzysta liczba 1)

NF = 0 -> nieparzysta liczba 1



BF0 = 0 ->więcej niż jedno 0 w wyniku

BF1 = 1 ->w wyniku znajduje się jedna 1

PF = 1 -> uzupełnienie do parzystej liczby (nieparzysta liczba 1)

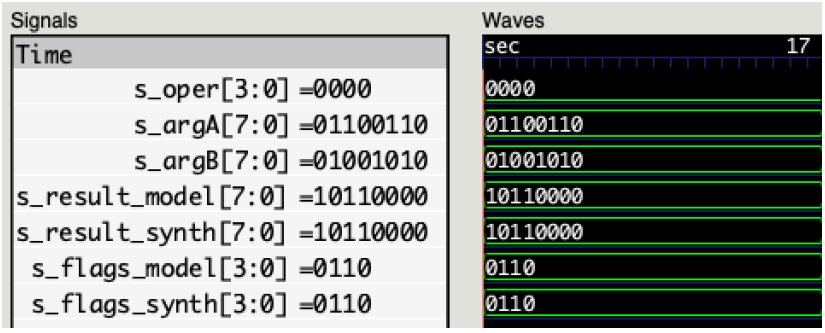
NF = 0 -> nieparzysta liczba 1

		i_argA	i_argB	o_result	o_result sim	Flagi	Flagi sim
1	**Dodawanie**	.01100110	.01001010	10110000	10110000	.0110	.0110
2		.00000001	10100011	10100100	10100100	.0110	.0110
3		.00001001	10110001	10111010	10111010	.0110	.0110
4		11001101	.00001000	11010101	11010101	.0101	.0101
1	**OR**	.00000000	.00111000	.00111000	.00111000	.0110	.0110
2		10000110	.01011100	11011110	11011110	.1001	.1001
3		11001110	11000111	11001111	11001111	.1010	.1010
4		11000110	11110011	11110111	11110111	.0101	.0101
1	**NOR**	11000110	11110011	.00001000	.00001000	.0110	.0110
2		11000011	.01011111	.00100000	.00100000	.0110	.0110
3		.01000111	10001001	.00110000	.00110000	.1001	.1001
4		.01101110	10001111	.00010000	.00010000	.0110	.0110
1	**<<**	.00000000	.00111000	.00000000	.00000000	.1001	.1001
2		10000110	.01011100	.00000000	.00000000	.1001	.1001
3		11000110	11110011	.00000000	.00000000	.1001	.1001
4		11000011	.01011111	.00000000	.00000000	.1001	.1001
1	**<<<**	11000110	11110011	.00000000	.00000000	.1001	.1001
2		10000110	.01011100	.00000000	.00000000	.1001	.1001
3		11001110	11000111	.00000000	.00000000	.1001	.1001
4		11000011	.01011111	.00000000	.00000000	.1001	.1001

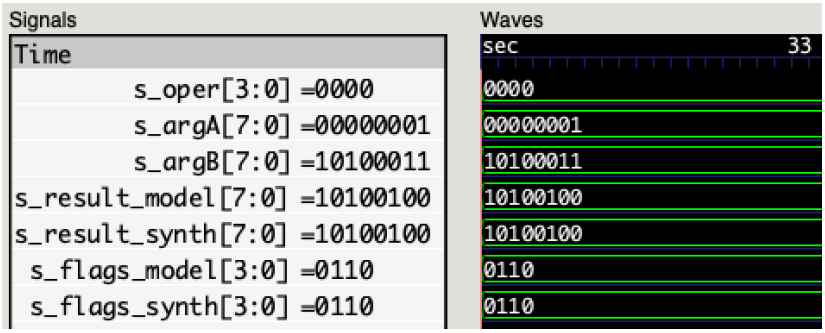
		i_argA	i_argB	o_result	o_result sim	Flagi	Flagi sim
1	U2 -> Gray	.00010011		.00011010	.00011010	.0110	.0110
2		10110100		11101110	11101110	.1001	.1001
3		.01000011		.01100010	.01100010	.0110	.0110
4		10010010		11011011	11011011	.1010	.1010
1	U1 -> U2	10011010	10110111	.01001101	.01001101	.1010	.1010
2		.00000110	10001101	.00000110	.00000110	.1001	.1001
3		11110010	11000100	.01111001	.01111001	.0101	.0101
4		.01001001	.01011010	.01001001	.01001001	.0101	.0101
1	CRC4 - check	.0110	.1100	.0000	.0000	.1001	.1001
2		.1110	.0111	.0000	.0000	.1001	.1001
3		.0111	.1001	.0000	.0000	.1001	.1001
4		.1110	.1111	.0000	.0000	.1001	.1001
1	4 CRC	.0011	.1111	.0010	.0010	.0110	.0110
2		.1110	.0101	.0011	.0011	.1010	.1010
3		.1100	.1001	.0001	.0001	.0101	.0101
4		.0111	.1001	.0010	.0010	.0110	.0110
1	Zliczanie 1	.00000000	.00111000	.00000011	.00000011	.1010	.1010
2		11001110	11000111	.00001010	.00001010	.1001	.1001
3		11000011	.01011111	.00001010	.00001010	.1001	.1001
4		.01011101	.10010001	.00001000	.00001000	.0110	.0110
1	Dekoder termometrowy	11111000		.00000111	.00000111	.0101	.0101
2		.00010010		.00000100	.00000100	.0110	.0110
3		.00001000		.00000011	.00000011	.1010	.1010
4		10100000		.00000111	.00000111	.0101	.0101
1	Dekoder priorytetowy	.01000100		.00000110	.00000110	.1001	.1001
2		10110101		.00001000	.00001000	.0110	.0110
3		10001000		.00000101	.00000101	.1010	.1010
4		10101001		.00001000	.00001000	.0110	.0110

WERYFIKACJA REALIZACJI JEDNOSTKI:

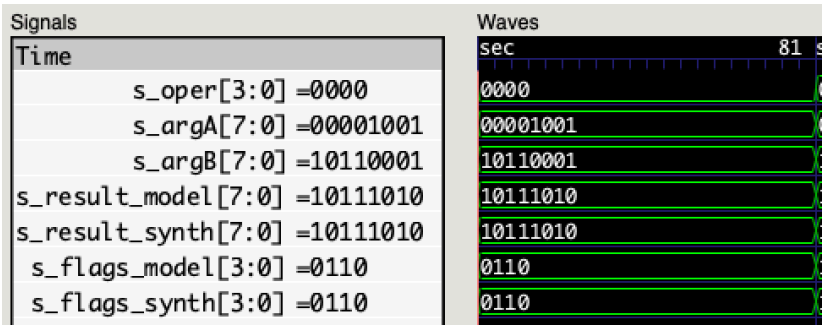
Dodawanie i_oper = 0000



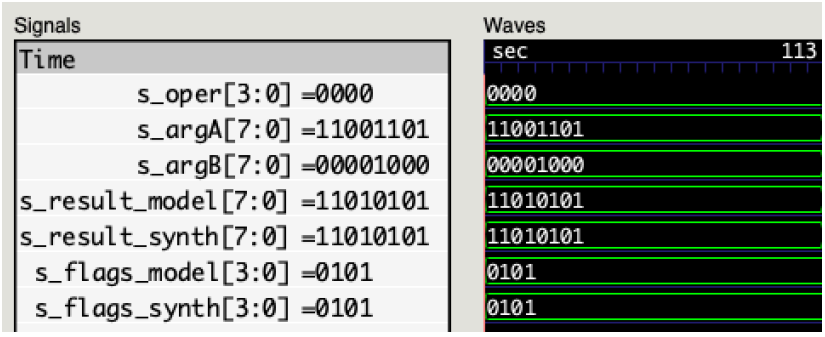
Sygnał 1



Sygnał 2



Sygnał 3

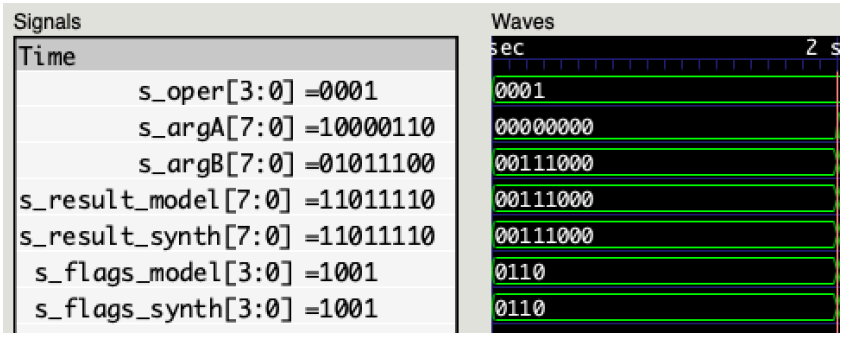


Sygnał 4

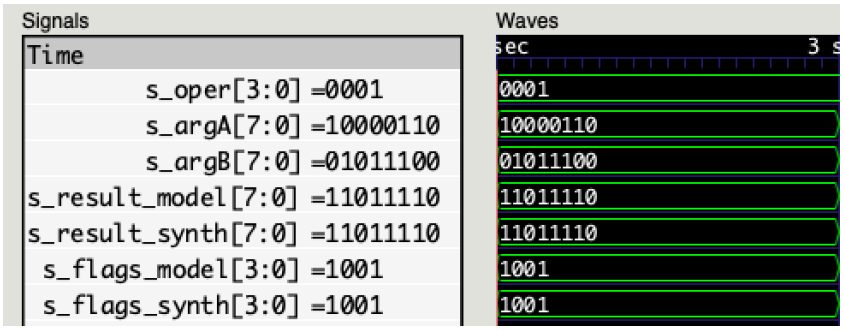
Operacja dodawania przebiega poprawnie, jak jest to zaprezentowane na powyższych przebiegach.

Przykładowo dla sygnału 4 na wejściu mamy dwie wartości 205 (11001101) i 8 (00001000) a na wyjściu jest podawana wartość 213 (11010101).

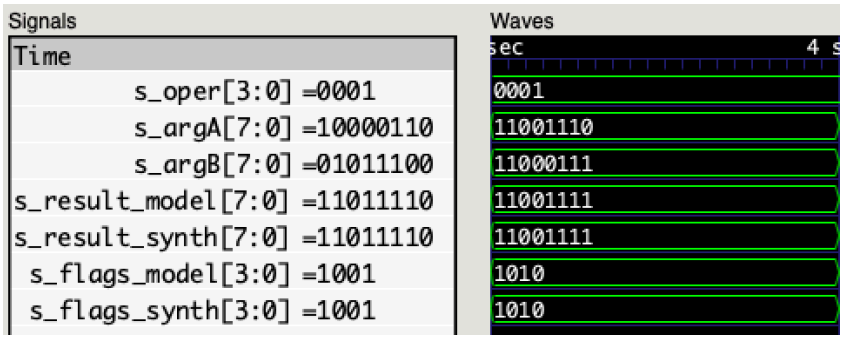
Suma logiczna argumentów i_oper = 0001



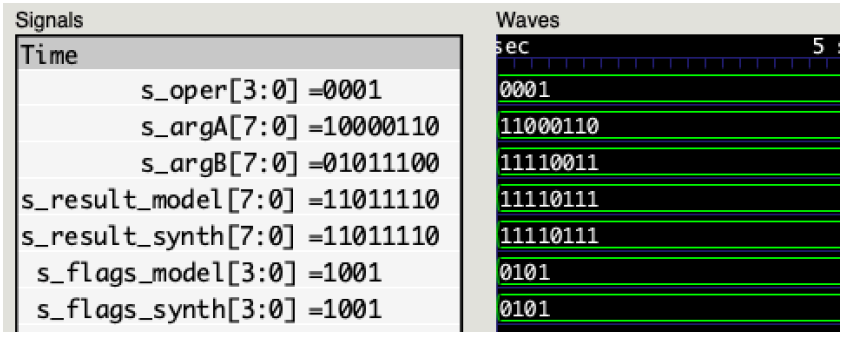
Sygnał 1



Sygnał 2



Sygnał 3



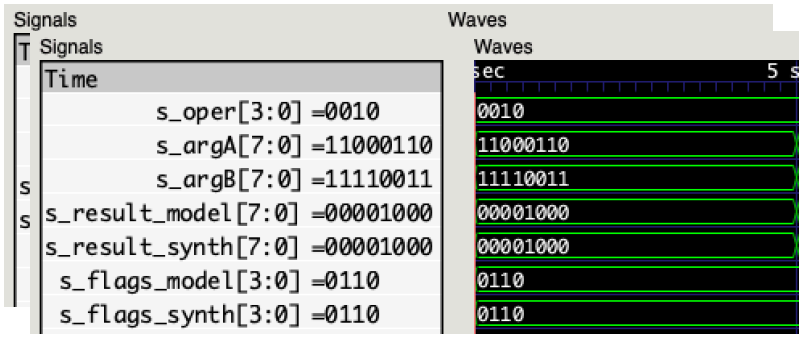
Sygnał 4

Operacja OR przebiega poprawnie, jak jest to zaprezentowane na powyższych przebiegach.

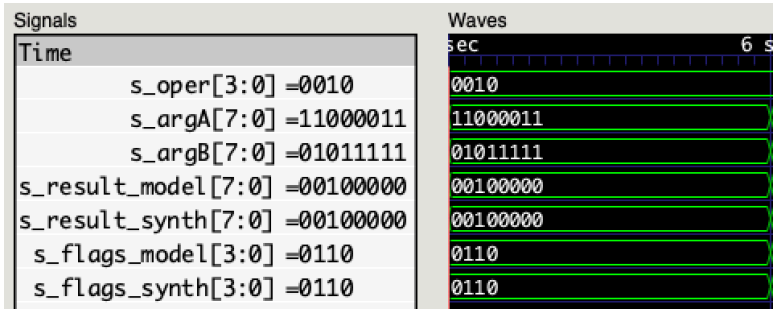
Przykładowo dla sygnału 4 na wejściu mamy dwie wartości (11000110) i (11110011) a na wyjściu jest podawana wartość (11110111), co zgadza się z teorią tej operacji.

(1 v 1 -> 1, 1 v 0 -> 1, 0 v 1 -> 1, 0 v 0 -> 0)

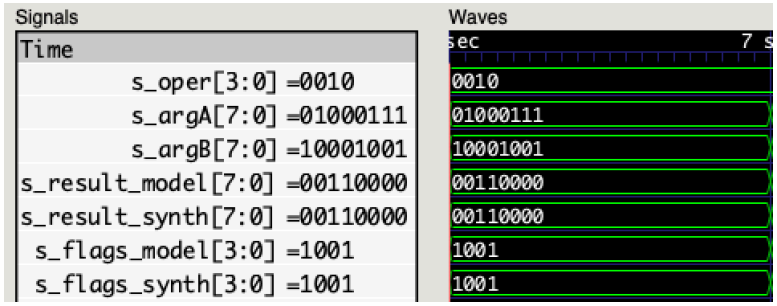
Zanegowana suma logiczna argumentów i_oper = 0010



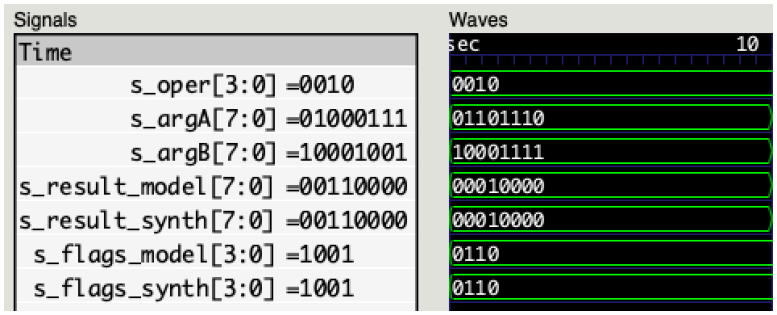
Sygnał 1



Sygnał 2



Sygnał 3



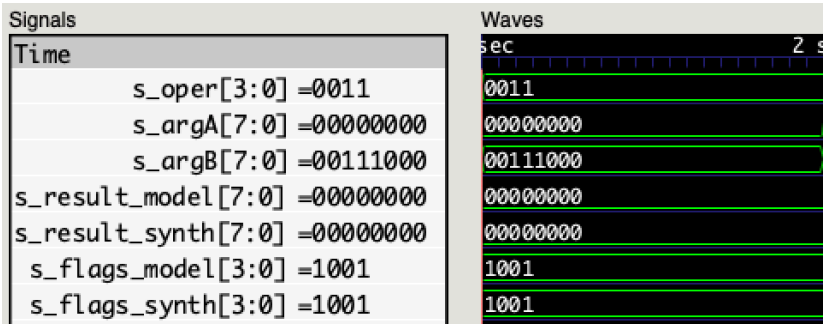
Sygnał 4

Operacja NOR przebiega poprawnie, jak jest to zaprezentowane na powyższych przebiegach.

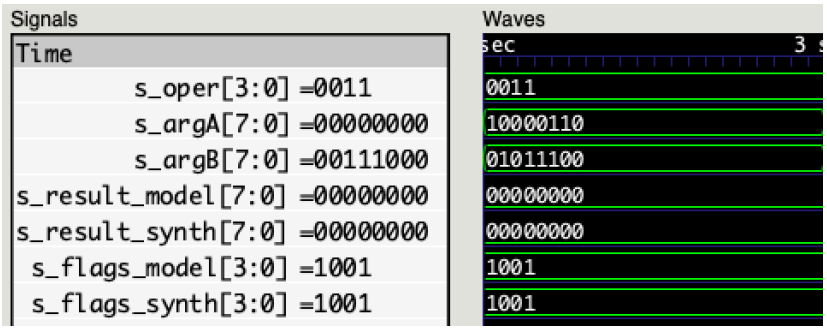
Przykładowo dla sygnału 4 na wejściu mamy dwie wartości (01101110) i (10001111) a na wyjściu jest podawana wartość (00010000), co zgadza się z teorią tej operacji.

(1 v 1 -> 0, 1 v 0 -> 0, 0 v 1 -> 0, 0 v 0 -> 1)

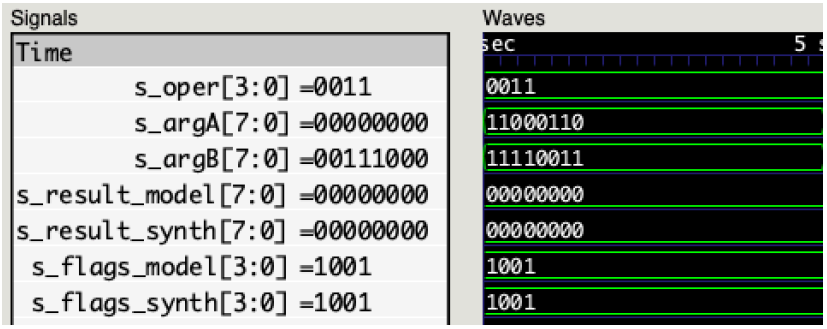
Logiczne przesunięcie argumentów w lewo i_oper = 0011



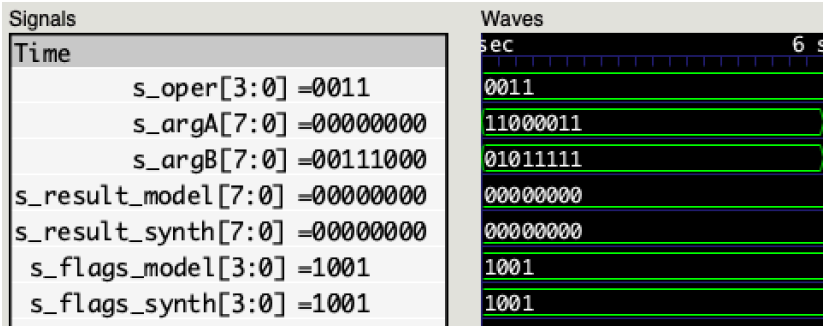
Sygnał 1



Sygnał 2



Sygnał 3

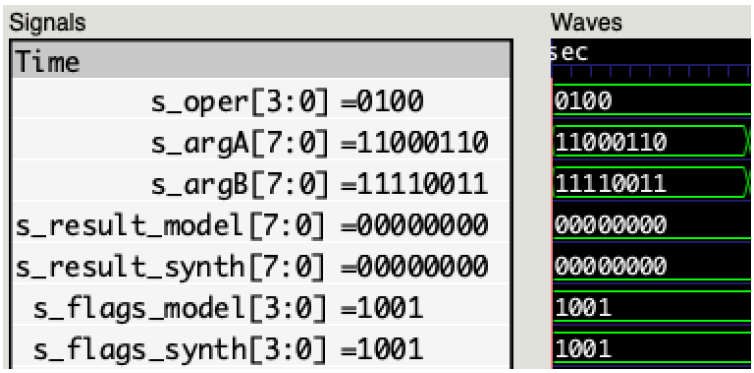


Sygnał 4

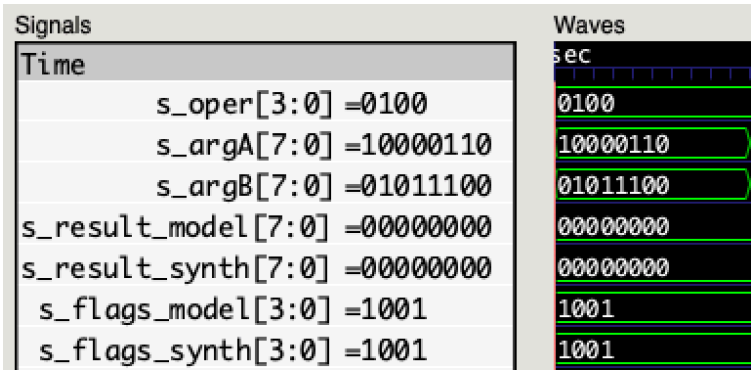
Operacja logicznego przesunięcia przebiega poprawnie, jak jest to zaprezentowane na powyższych przebiegach.

Dla liczb o dużej wartości, wynikiem przesunięcia będzie zawsze 0.

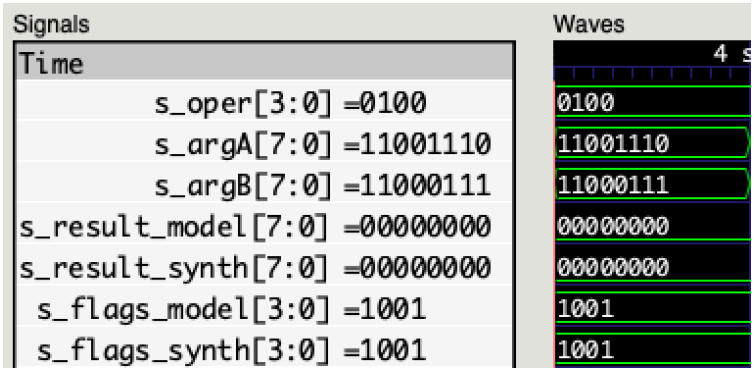
Arytmetyczne przesunięcie argumentów w lewo i_oper = 0100



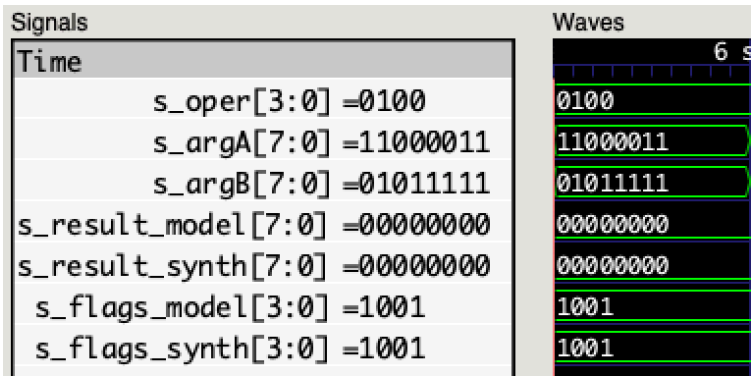
Sygnał 1



Sygnał 2



Sygnał 3

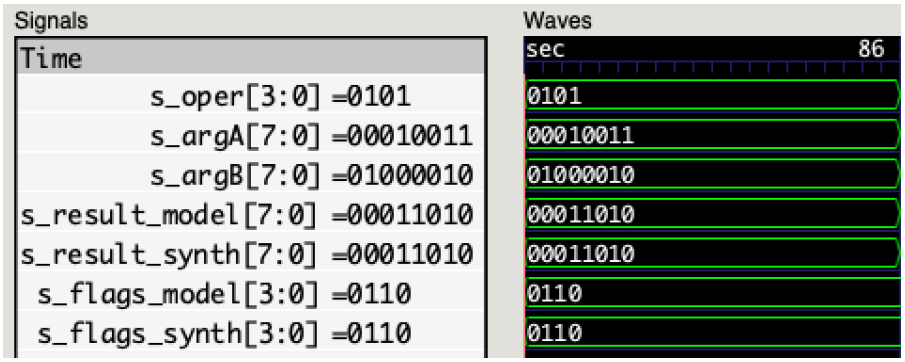


Sygnał 4

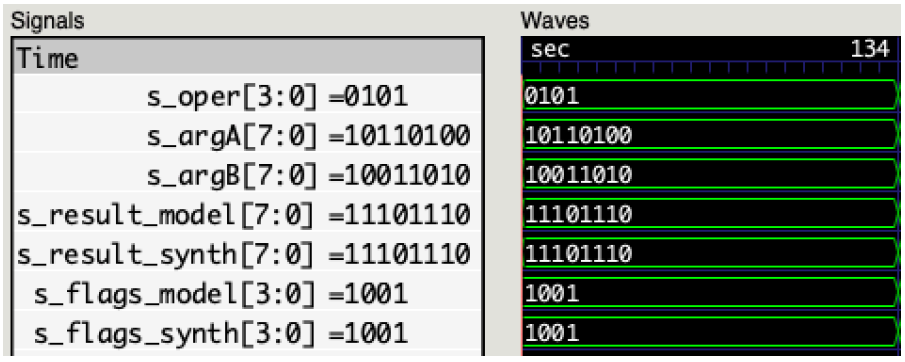
Operacja arytmetycznego przesunięcia przebiega poprawnie, jak jest to zaprezentowane na powyższych przebiegach.

Dla liczb o dużej wartości, wynikiem przesunięcia będzie zawsze 0,

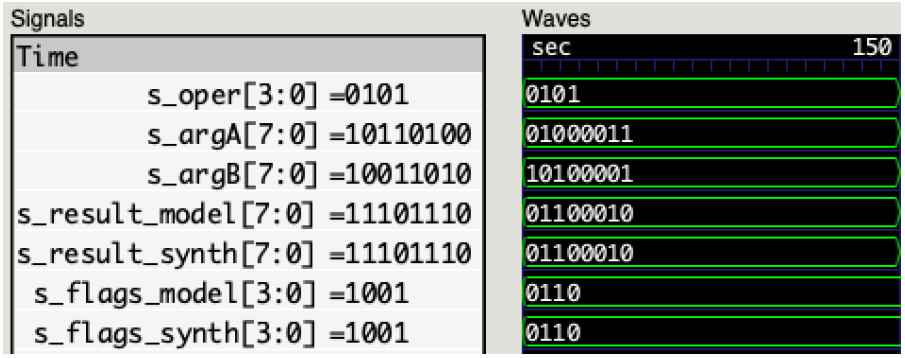
Konwersja danej wejściowej z kodu U2 na kod GRAY i_oper = 0101



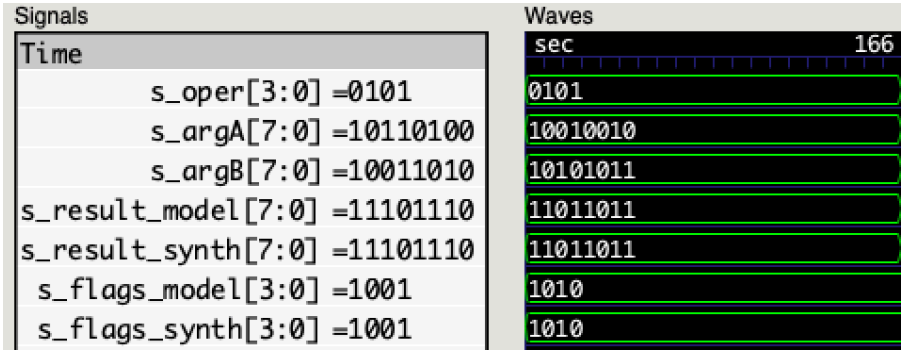
Sygnał 1



Sygnał 2



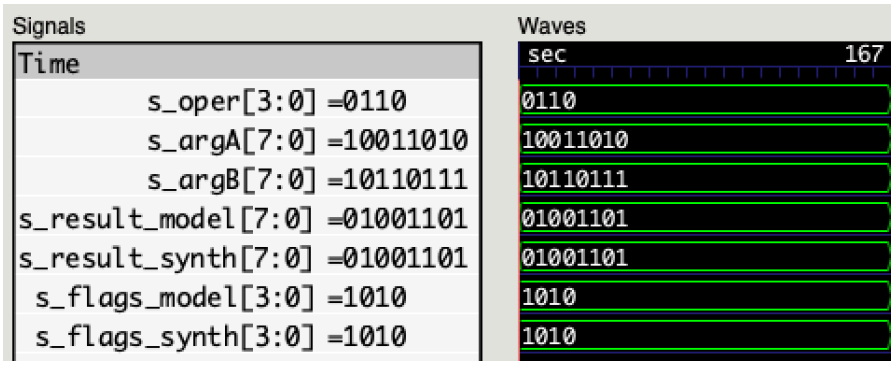
Sygnał 3



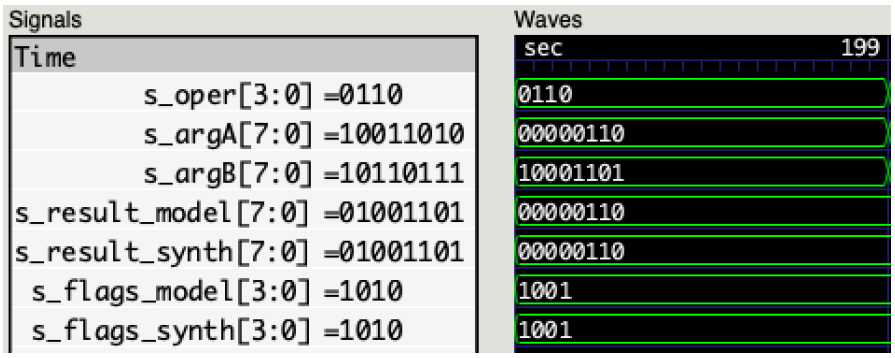
Sygnał 4

Z powyższych symulacji można wywnioskować, że operacja działa prawidłowo.

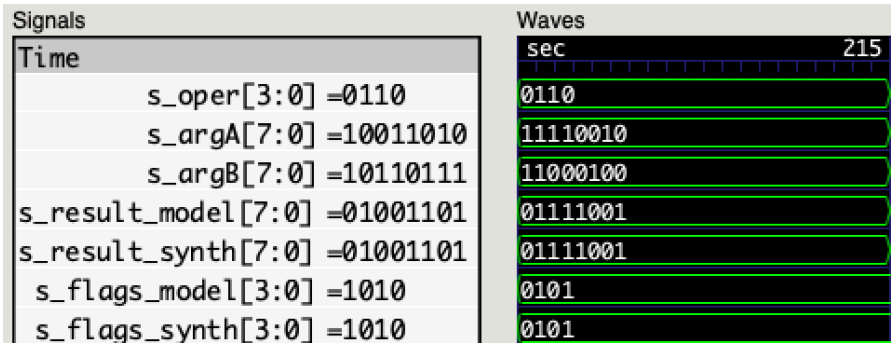
Konwersję danej wejściowej z kodu U1 na kod U2 i_oper = 0110



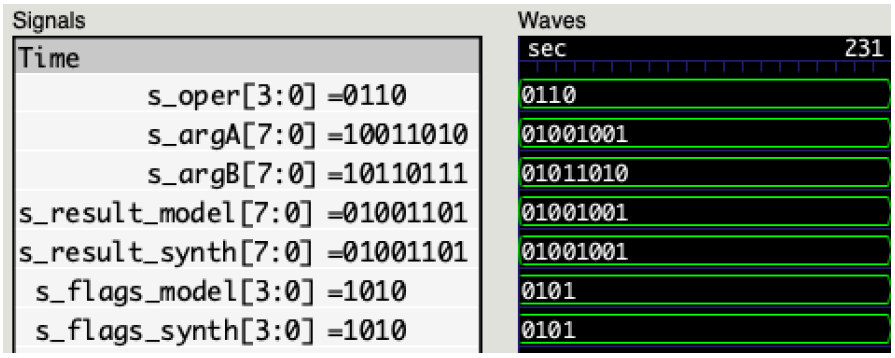
Sygnał 1



Sygnał 2



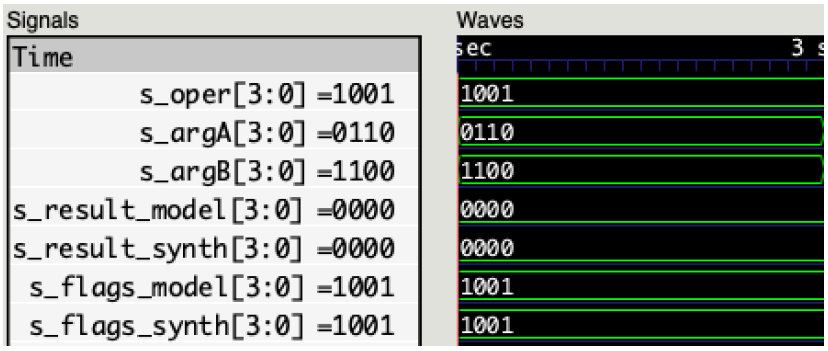
Sygnał 3



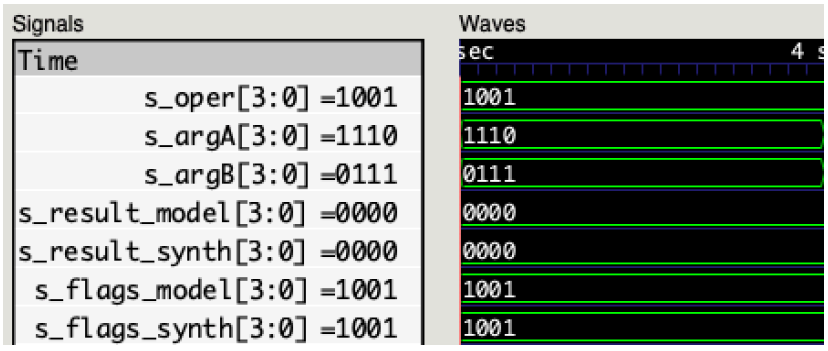
Sygnał 4

Z powyższych symulacji można wywnioskować, że operacja działa prawidłowo.

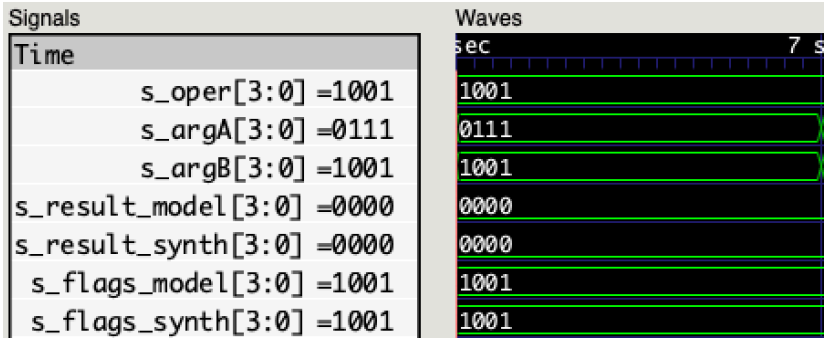
Sprawdzenie zgodności kodu CRC-4 i_oper = 1001



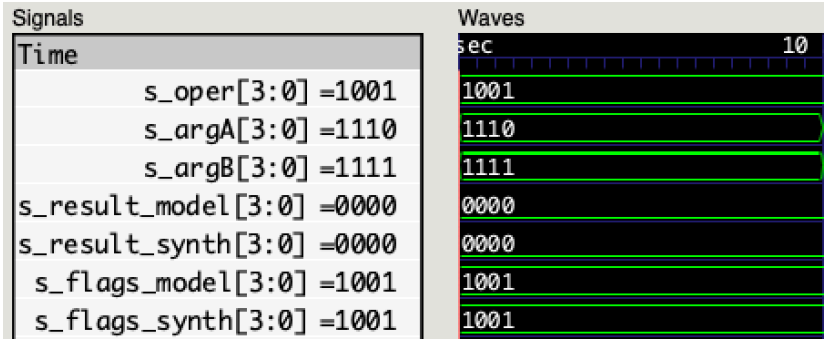
Sygnał 1



Sygnał 2



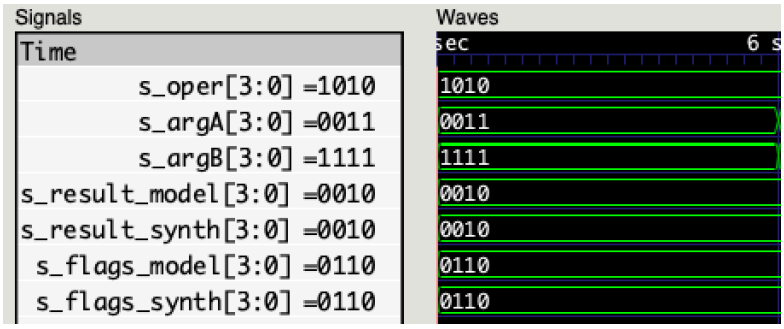
Sygnał 3



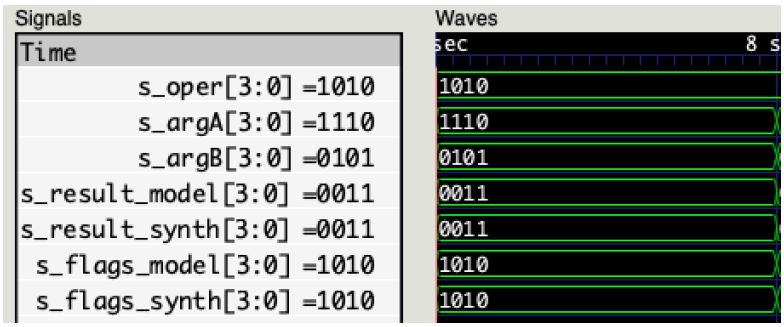
Sygnał 4

Wyznaczenie kodu CRC-4 i_oper = 1010

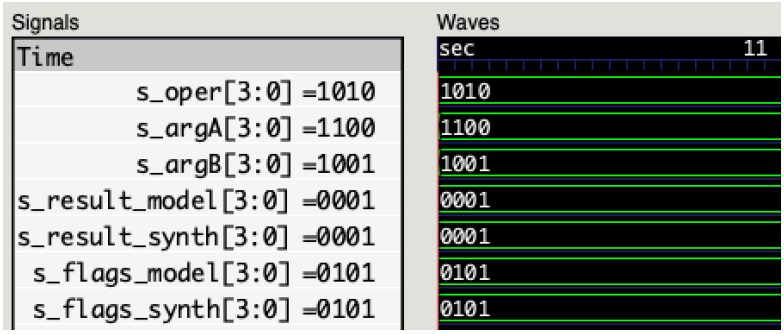
Symulacja tej operacji została przeprowadzona dla liczb 4-bitowych.



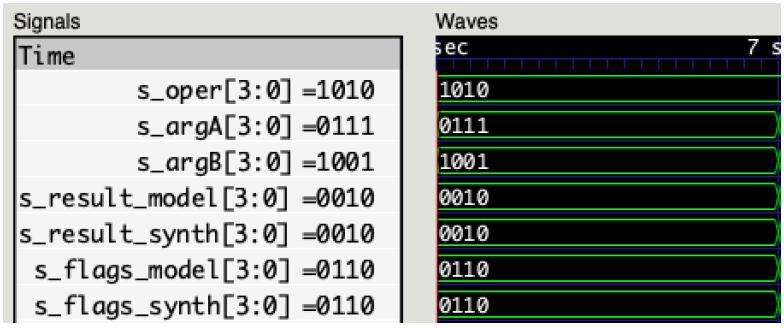
Sygnał 1



Sygnał 2



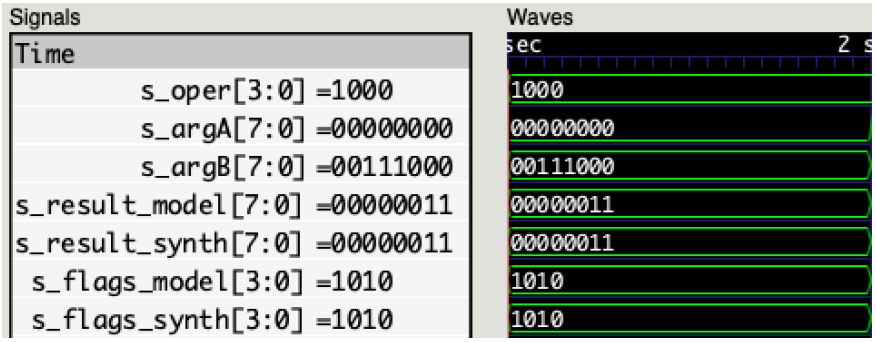
Sygnał 3



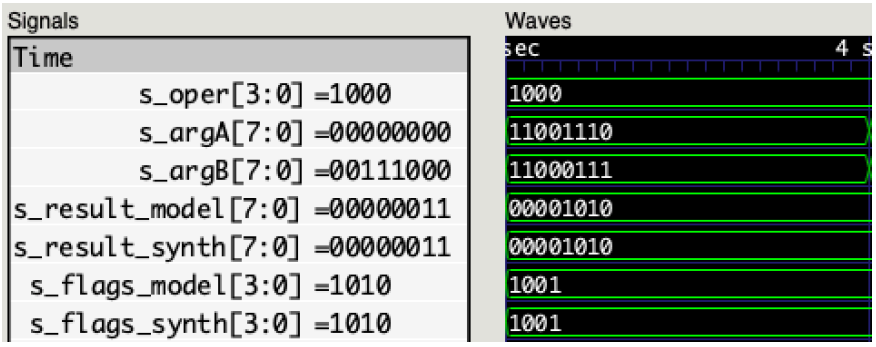
Sygnał 4

Symulacja wyznaczania kodu CRC4 przebiega poprawnie dla powyższych przebiegów (sprawdzone wg algorytmu)

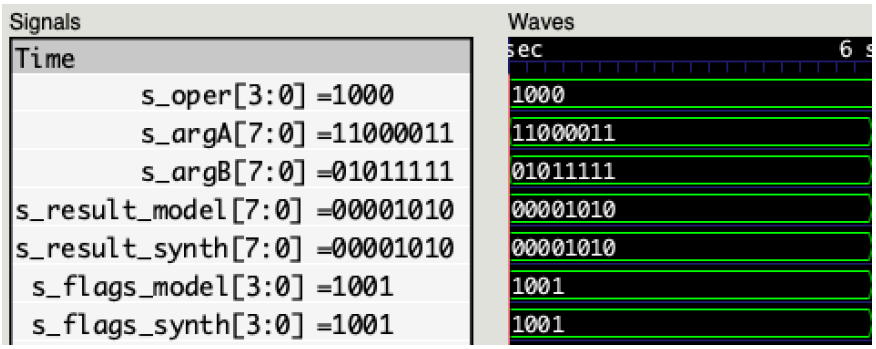
Zliczanie sumarycznej liczby jedynek w obu argumentach wejściowych i_oper = 1000



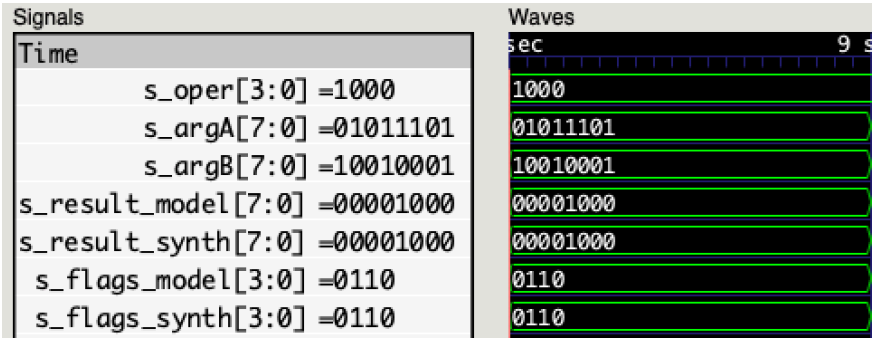
Sygnał 1



Sygnał 2



Sygnał 3

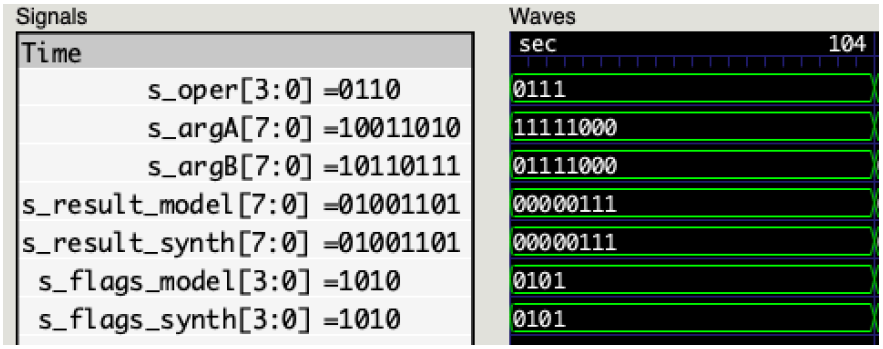


Sygnał 4

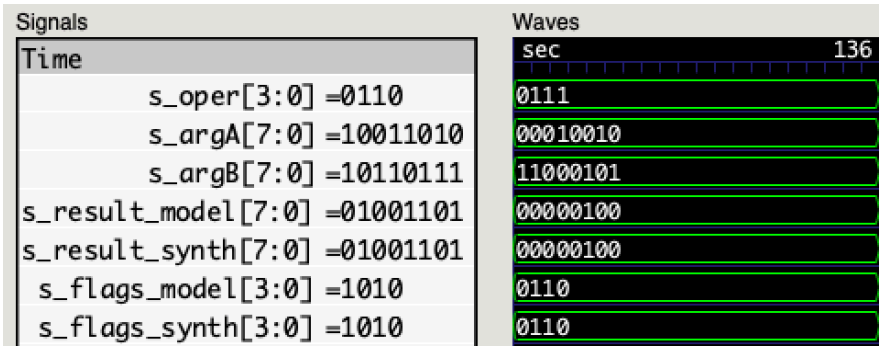
Jak wynika z powyższych przebiegów, sumaryczne zliczanie jedynek działa poprawnie.

Przykładowo dla sygnału 4, ilość jedynek z obu danych wejściowych to 8, co zgadza się z wynikiem (00001000).

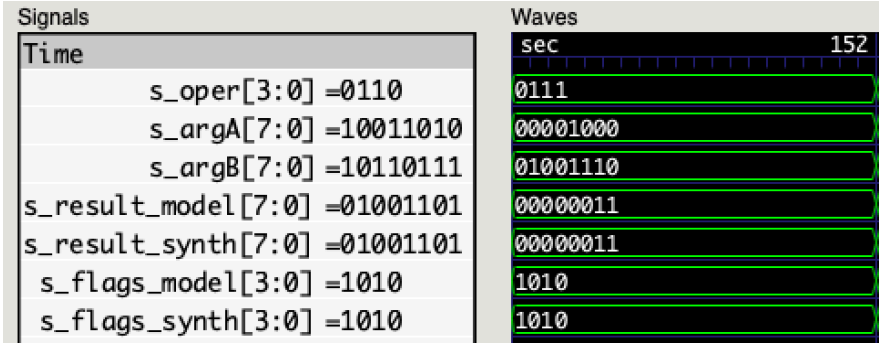
Dekoder termometrowy i_oper = 0111



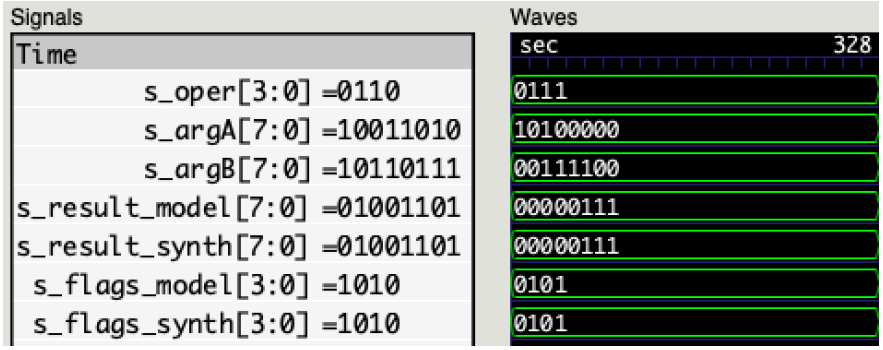
Sygnał 1



Sygnał 2



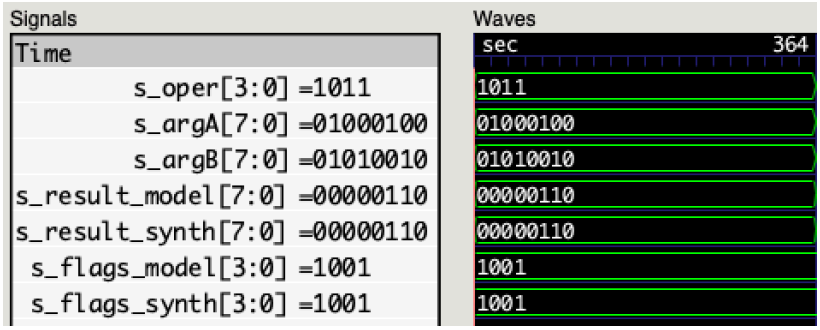
Sygnał 3



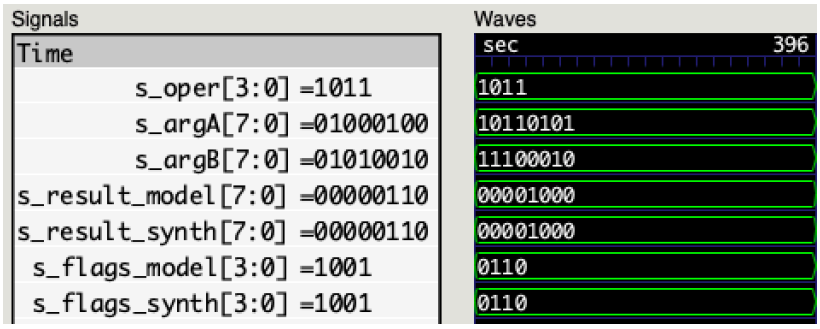
Sygnał 4

Operacja konwersji na termometrowy przebiega poprawnie, jak jest to zaprezentowane na powyższych przebiegach.

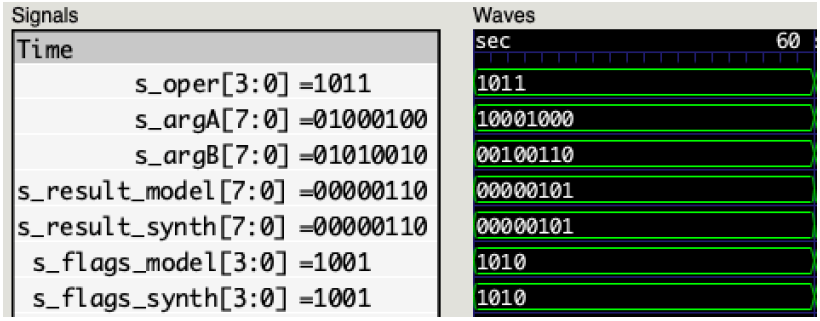
Dekoder priorytetowy i_oper = 1011



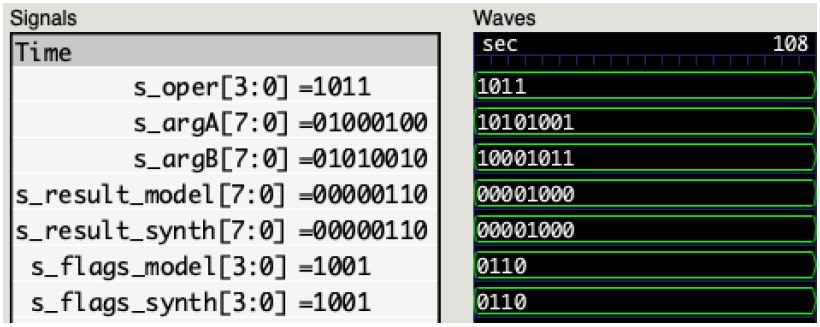
Sygnał 1



Sygnał 2



Sygnał 3



Sygnał 4

Operacja dekodera priorytetowego przebiega poprawnie, jak jest to zaprezentowane na powyższych przebiegach.

W końcowych etapach prac pojawił się błąd w symulacji.

W jednostce nie działa do końca poprawnie flaga BF1 oraz BF0. Dla części sygnałów przyjmują prawidłową wartość, jednak nie zawsze zgadza się z teorią.