ARITH_UNIT_26

Projekt indywidualny SCK

Projekt wykonawczy modułu arith_unit_26 operującej na liczbach w kodzie ZM

Semestr 2023 Z

Illia Kovalenko 330104

19.12.2023

1. Opis ogólny projektu

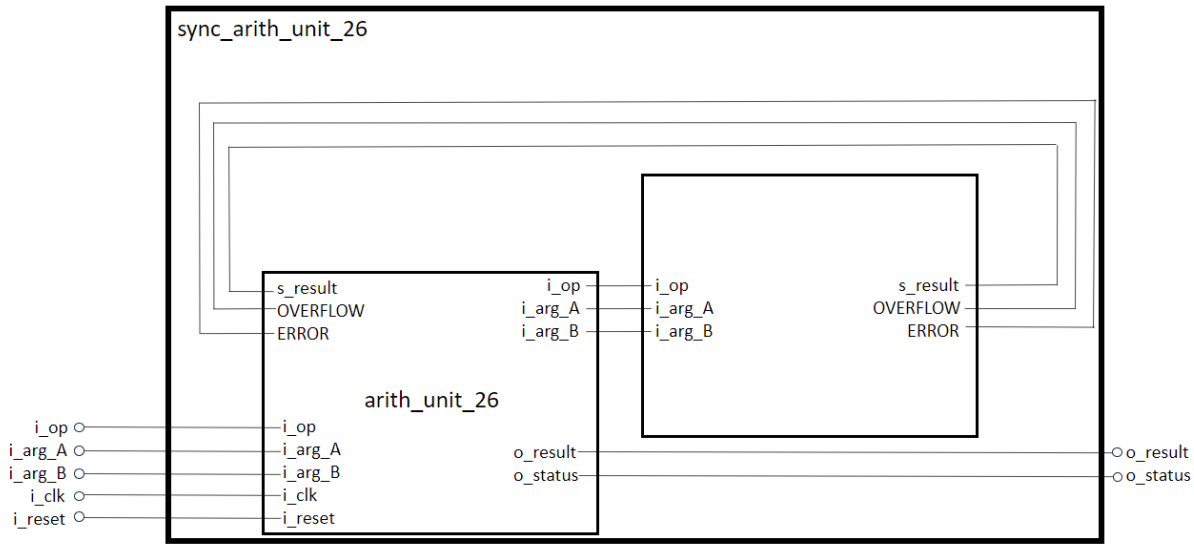
Celem projektu jest implementacja synchronicznej jednostki arytmetyczno-logicznej sync_arith_unit_26 realizującej operacje arytmetyczne, logiczne i inne na liczbach całkowitych A i B w kodzie znak-moduł.

Jednostka ALU sync_arith_unit_26 posiada 2 wektory wejściowych i_arg_A oraz i_arg_B na których wykonuje 4 podstawowe operacje (tabela 4. Operacje do wykonania), których liczba bitów zależy od parametru WIDTH_M, wejście wyboru i_op, zegara i_clk oraz resetu i_reset. Również jednostka ma wyjście synchroniczne wyniku operacji o_result oraz wyjście statusu o _status, które informuje o poprawności wyniku, liczbie jedynek czy przepełnieniu (tabela 5.wyjście o_status). Wszystkie opisane wyżej wejścia i wyjścia posiadają swoje własne rejestry oprócz wejścia resetu, ale wszystkie działają w sposób synchroniczny. Aby wykonać operację wybraną poprzez wejście wyboru i_op, musimy poświęcić 2 takty zegara (jeden do zapisu danych do rejestrów wejściowych, a drugi do rejestru wyniku oraz statusu operacji).

2. Wejścia i wyjścia układu

Nazwa	Typ	Szerokość	Opis
i_op	WE	N-bitów	Kodu operacji
i_arg_A	WE	M-bitów	Argumentu A
i_arg_B	WE	M-bitów	Argumentu B
i_clk	WE	1	Wejście zegara
i_reset	WE	1	Reset
o_result	WY	M-bitów	Wynik operacji
o_status	WY	4	Status operacji

3. Uproszczony schemat blokowy



4. Operacje do wykonania

Operacja	i_op
A << B	00
A > B	01
A / ~ B	10
ZM(A) => U2(A)	11

1) A << B

Przesunięcie wektora A o B bitów w lewo. Jeżeli B jest mniejsze od zera, układ zgłasza błąd, a wartość wyjściowa zostaje nieokreślona.

2) A > B

Sprawdzenie czy liczba A jest większa od liczby B. Gdy warunek jest spełniony, układ wystawia na wyjściu liczbę większą od zera – A jeżeli A jest dodatnie i ~A jeżeli A jest ujemne, w przeciwnym wypadku to jest liczba 0.

3) A / (~B)

Dzielenie liczby A przez ~B (zmieniamy każdy bit liczby B). Jeżeli ~B jest zerem, zgłaszamy błąd, a wyjście jest nieokreślone.

4) ZM(A) => U2

Zamiana liczby A z kodu znak-moduł na kod U2. Jeżeli nie można dokonać poprawnej konwersji zgłaszamy błąd, a wyjście układu pozostaje nieokreślone, a w szczególnym przypadku przepełnienia zmieniamy status bita OVERFLOW na 1.

5. Wyjście o_status

Nazwa	Pozycja	Opis	Przykład
ERROR	3	Wynik operacji został określony niepoprawnie/błędny wynik)	-
NOT_EVEN_1	2	Nieparzysta liczba jedynek w wyniku	0001
ONES	1	Wszystkie bity wyniku są jedynkami	1111
OVERFLOW	0	Nastąpiło przepełnienie	-

6. Lista plików projektu

Nazwa	Opis
arith_unit_26.sv	Źródłowy kod ALU
testbench.sv	Moduł testbench (testowy) do symulacji układu
makefile	Plik instrukcji, kompiluje kod verilog i robi symulację

run.ys	Skrypt sterujący syntezą logiczną w Yosys
arith_unit_26_rtl.sv	Źródłowy kod po syntezie logicznej Yosys
synth.log	Raport syntezy logicznej Yosys
arith_unit_26.iveri.log	Raport symulacji Icarus Verilog
arith_unit_26.iveri.run	Kod źródłowy skompilowany przez Icarus Verilog
signals.vcd	Dane sygnałów po symulacji dla programu GTKWave

7. Raporty

Raport z syntezy logicznej Yosys

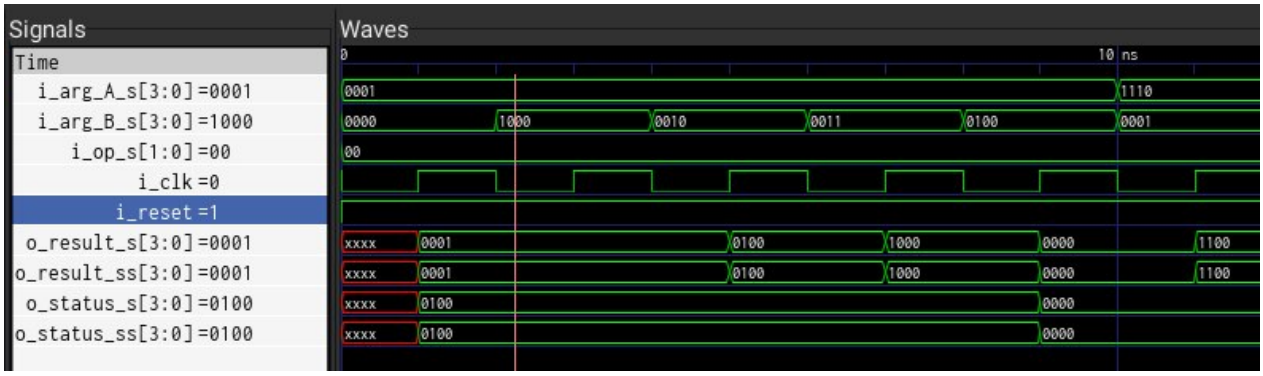
```
Number of wires:                173
Number of wire bits:            220
Number of public wires:          13
Number of public wire bits:      60
Number of memories:               0
Number of memory bits:            0
Number of processes:              0
Number of cells:                176
  $_AND_                         85
  $_DFF_PN0_                      8
  $_NOT_                         21
  $_OR_                          56
  $_XOR_                          6

Estimated number of transistors:        960+
```

Dla rozłożenia układu na podstawowe składowe było zrobione mapowanie tylko na AND, OR, XOR, NOT. Również jeszcze bardziej można rozłożyć układ mniejszą ilością bramek, np. AND, OR i NOT.

Wyniki symulacji

i_op = 2’b00



1) 0001 << 0 = 0001 (dobry wynik); (NOT_EVEN_1 = 1) (nieparzysta liczba jedynek);

2) 0001 << 0 = 0001 (dobry wynik); (NOT_EVEN_1 = 1);

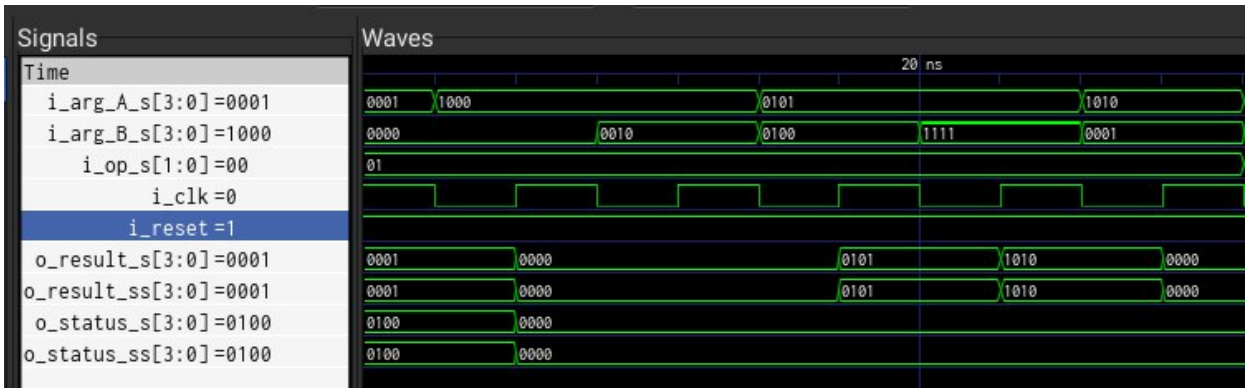
3) 0001 << 2 = 0100 (dobry wynik); (NOT_EVEN_1 = 1);

4) 0001 << 3 = 1000 (dobry wynik); (NOT_EVEN_1 = 1);

5) 0001 << 4 = 0000 (dobry wynik); (“znika” 1) (NOT_EVEN_1 = 0) (nie ma jedynek);

6) 1110 << 1 = 1100 (dobry wynik); (“znika” 1) (NOT_EVEN_1 = 0) (parzysta liczba jedynek).

i_op = 2’b01



1) 0001 > 0000 oraz wynik 0001 (dobry wynik) (NOT_EVEN_1 = 1) (nieparzysta liczba jedynek);

2) 1000 > 0000 oraz wynik 0000 (dobry wynik) (podwójna reprezentacja zera);

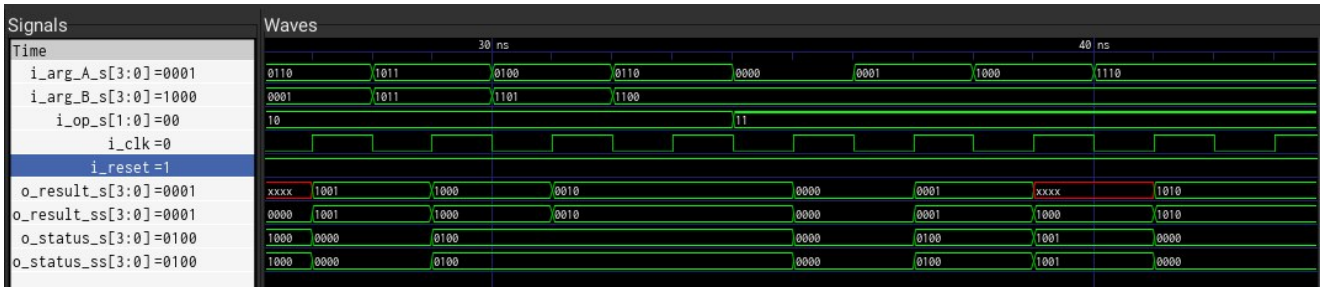
3) 1000 < 0010 oraz wynik 0000 (dobry wynik);

4) 0101 > 0100 oraz wynik 0101 (dobry wynik) (NOT_EVEN_1 = 0) (parzysta liczba jedynek);

5) 0101 > 1111 oraz wynik 0101 (dobry wynik);

6) 1010 < 0001 oraz wynik 0000(dobry wynik);

i_op = 2'b10



1) 6 / 0 = (nieokreślone) wynik xxxx (dobry wynik) (ERROR = 1, dlatego, że występuje błąd):

2)6 / -6 = -1 (dobry wynik)

3) -3 / 4 = 0 (dobry wynik) (część całkowita od dzielenia jest wynikiem) (NOT_EVEN_1 = 1) (nieparzysta liczba jedynek);

4) 4 / 2 = 2 (dobry wynik) (NOT_EVEN_1 = 1) (nieparzysta liczba jedynek);

5) 6 / 3 = 2 (dobry wynik) (NOT_EVEN_1 = 1) (nieparzysta liczba jedynek);

i_op = 2'b11

1) 0000(ZM) => 0000(U2) 0 (dobry wynik);
2) 0001(ZM) => 0001(U2) 1 (dobry wynik) (NOT_EVEN_1 = 1) (nieparzysta liczba jedynek);
3) 1000(ZM) x=>x xxxx U2 (dobry wynik) (występuje błąd związany z przepełnieniem) ERROR = OVERFLOW = 1;

4) 1110(ZM) => 1010(U2) -6 (dobry wynik).