

# Systemy cyfrowe i komputerowe

-

## Projekt zespołowy - ALU 4, ALU 27, ALU Z

Adam Szajgin s319821  
Maciej Gójski sXXXXXX  
Kacper Kęsy sXXXXXX

Politechnika Warszawska

22 stycznia 2024

### 1. Opis funkcjonalny układu

1. Układ ALU
2. Porty układu:

W skład portów układu wchodzi:

Nazwa	WE/WY	Funkcja portu
<b>i_op</b>	WE	n-bitowe wejście określające kod operacji
<b>i_arg_A</b>	WE	m-bitowe wejście wektora binarnego A
<b>i_arg_B</b>	WE	m-bitowe wejście wektora binarnego B
<b>i_clk</b>	WE	Wejście sygnału zegarowego
<b>i_reset</b>	WE	Wejście resetu synchronicznego wyzwalanego stanem niskim
<b>o_result</b>	WY	Wyjście synchroniczne zwracające wynik operacji na wektorach wejściowych
<b>o_status</b>	WY	4-bitowe wyjście synchroniczne dostarczające informacje o statusie wyniku operacji

#### **i\_reset:**

Odpowiada za zerowanie wszystkich rejestrów wyjściowych. Proces resetowania jest zsynchronizowany z sygnałem zegarowym. Aktywacja tego wejścia występuje w stanie niskim.

Natomiast port **o\_status** składa się z bitów mających poniższe znaczenie:

Nazwa	Znaczenie bitu
<b>bit ERROR</b>	Sygnalizacja o tym, iż wynik został określony niepoprawnie
<b>bit EVEN_1</b>	Sygnalizuje parzystą liczbę jedynek w wyniku. Ustawiany na 0, gdy jest sygnalizowany błąd operacji
<b>bit ONES</b>	Sygnalizuje, że wszystkie bity wyniku są ustawione na 1. Ustawiany na 0, gdy jest sygnalizowany błąd
<b>bit OVERFLOW</b>	Sygnalizuje, że nastąpiło przepełnienie i wynik operacji wykracza poza szerokość wektora wejściowego

## 2. Lista realizowanych operacji

Realizowane operacje:

Na dwanaście operacji wykonywalnych przez jednostkę składają się:

Część układu sync\_arith\_unit\_4:

-  $A - 2 * B$ , dla **i\_op = 1'b00**

Odejmowanie dwukrotności liczby B od A

-  $A < B$ , dla **i\_op = 1'b01**

Sprawdzenie czy liczba A jest mniejsza od liczby B. Gdy warunek jest spełniony, układ ma wystawić na wyjściu liczbę większą od zera, w przeciwnym wypadku ma to być liczba 0.

-  $(A + B)[B] = 0$ , dla **i\_op = 1'b10**

Wynikiem operacji ma być liczba będąca sumą A i B z bitem o index B ustawionym na wartość 0. Jeżeli liczba B jest mniejsza od zera lub większa od szerokości wektora A, układ ma zgłaszać błąd.

-  $U2(A) \Rightarrow ZM(A)$ , dla **i\_op = 1'b11**

Zamiana liczby A zapisanej w kodzie U2 na zapis w kodzie ZM. Jeżeli nie można dokonać poprawnej konwersji, należy zgłosić błąd, a wyjście układu powinno pozostać nieokreślone.

---

Część układu sync\_arith\_unit\_27:

-  $A \gg \sim B$ , dla **i\_op = 2'b00**

Przesunięcie wektora A o  $\sim B$  bitów w prawo. Jeżeli  $\sim B$  bitów jest mniejsze od zera, układ ma zgłosić błąd, a wartość wyjściowa ma pozostać nieokreślona.

-  $A + B$ , dla **i\_op = 2'b01**

Dodawanie A i B.

-  $A + -B$

$\frac{A}{B}$ , dla **i\_op = 2'b10**

Dzielenie liczby A przez B. Jeżeli liczba B jest zerem, należy zgłosić błąd, a wyjście ma być nieokreślone.


-  $U2(A) \Rightarrow ZM(A)$ , dla **i\_op = 2'b11**

Zamiana liczby A z kodu U2 na kod ZM. Jeżeli nie można dokonać poprawnej konwersji, należy zgłosić błąd, a wyjście układu powinno pozostać nieokreślone.

---

Część układu nr sync\_arith\_unit\_XX:

## 3. Schemat blokowy realizowanego modułu



schemat.png

Rys. 1. Schemat ALU

## 4. Synteza logiczna

Do syntezy jednostki arytmetyczno logicznej używaliśmy yosys'a. Pobraliśmy program z oficjalnego GitHuba projektu Yosys i używaliśmy zgodnie z dokumentacją. By przeprowadzić poprawną syntezę odpalaliśmy Yosys'a z dowolnego miejsca w drzewie plików. Do tego użyliśmy w kolejności komend:

- Wczytanie pliku do oprogramowania:

```
$read_verilog -sv alu.sv
```

- Skopiowanie jednostki sync\_4 do sync\_4\_synthesis:

```
$copy sync_arith_unit_4 sync_arith_unit_4_synthesis
```

- Ustalenie hierarchii, gdzie jednostka sync\_4\_synthesis jest najwyższym poziomem:

```
$hierarchy -top sync_arith_unit_4_synthesis
```

- Synteza projektu:

```
$synth
```

- Optymalizacja i czyszczenie projektu:

```
$opt_clean
```

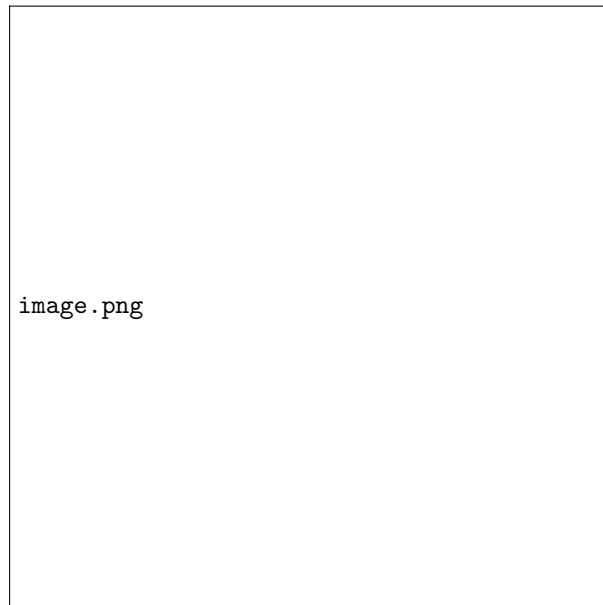
- Zapisanie projektu w formie pliku Verilog bez atrybutów:

```
$write_verilog -noattr alu_synthesis.sv
```

- Wyświetlenie statystyk dotyczących projektu:

```
$stat
```

W wyniku powyższej syntezy uzyskiwaliśmy wiele linijek raportu. Jednak załączanie całego wydruku tutaj, zajęłoby ok. 10 stron, dlatego też załączamy jedynie wydruk statystyk po syntezie mojej jednostki ALU:



Rys. 2. Wydruk statystyk jednostki po syntezie

## 5. Przeprowadzone symulacje - testy

By kompilować plik testbench.sv używaliśmy komendy: `$iverilog -g2005-sv testbench.sv -o test_compiled`. Następnie egzekwowaliśmy skompilowany plik komendą: `$vvp test_compiled`. Na wskutek czego uzyskiwaliśmy przebieg testbenchu który odpalaliśmy w gtkwave komendą: `gtkwave signals.vcd`

Każdą z operacji testowaliśmy osobno w ramach projektów indywidualnych, dlatego nie załączamy tego w poniższej dokumentacji. Jednak by sprawdzić rzeczywście działanie naszej jednostki i z nią komunikacji przeprowadziliśmy pojedyncze testy po połączeniu wszystkich operacji w jedno ALU oraz po dodaniu modułów komunikacji APB i UART.

1.