Systemy cyfrowe i komputerowe

-

Projekt zespołowy - ALU 4, ALU 27, ALU 23

Adam Szajgin s319821
Maciej Gójski s325007
Kacper Kęsy s325018

Politechnika Warszawska

25 stycznia 2024

1. Opis funkcjonalny układu

1. Układ:

   Układ kontrolera APB został zaprojektowany z myślą o efektywnym oczekiwaniu na sygnał informujący o zakończeniu operacji w jednostce arytmetyczno-logicznej (ALU). W chwili, gdy ALU zakończy operację i potwierdzi poprawność danych, kontroler APB przechodzi do fazy przygotowania do przesyłania informacji do kontrolera UART poprzez magistralę danych PWDATA.

   Po odebraniu danych przez kontroler komunikacji, układ APB przechodzi w tryb odczytu, monitorując magistralę PRDATA. Oczekuje on na sygnał z kontrolera komunikacji informujący o zakończeniu transferu danych i gotowości do przyjęcia kolejnych danych.

   W przypadku pomyślnego przesłania danych, kontroler APB informuje zewnętrzny układ, że ALU może otrzymać nowe dane i inicjuje nową operację poprzez sygnał o_waiting. Dodatkowo, emituje sygnał o_transfer_done, sygnalizując zakończenie procedury przesyłania danych.

   W sytuacji wystąpienia błędu, układ jedynie ustawia sygnał o_waiting, informując oczekujący system o konieczności ponownego przesłania danych. W ten sposób układ kontrolera APB efektywnie obsługuje proces przekazywania informacji pomiędzy ALU a kontrolerem komunikacji.

2. Porty układu:

   W skład portów układu ALU wchodzą:

Nazwa	WE/WY	Funkcja portu
i_op	WE	n-bitowe wejście określające kod operacji
i_arg_A	WE	m-bitowe wejście wektora binarnego A
i_arg_B	WE	m-bitowe wejście wektora binarnego B
i_clk	WE	Wejście sygnału zegarowego
i_reset	WE	Wejście resetu synchronicznego wyzwalanego stanem niskim
o_result	WY	Wyjście synchroniczne zwracające wynik operacji na wektorach wejściowych
o_status	WY	4-bitowe wyjście synchroniczne dostarczające informacje o statusie wyniku operacji
o_op_rdy	WY	wyjście 1-bitowe, sygnalizujące APB, że ma dla niego prawidłowe dane do wysłania
o_alu_error	WY	1-bitowe wyjście informujące, czy w ALU wystąpił błąd.

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

3. APB linie ze standardu:

Kontroler APB posiada następujące linie ze standardu:

Nazwa	WE/WY	Funkcja portu
PCLK	WE	zegar
PRESET	WE	reset synchroniczny
PADDR	WY	2 bitową magistralę adresową
PSELx	WY	2 bitowa magistrala adresowa pojedynczy bit dla każdego urządzenia
PENABLE	WY	sygnalizacja dla modułu podrzędnego, że może przyjąć dane,
PWRITE	WY	Sygnalizacja kierunku obsługi magistrali PWDATA i PRDATA
PWDATA	WY	m-bitowa magistrala danych z kontrolera do urządzeń podrzędnych
PREADY	WE	sygnalizacja z urządzenia podrzędnego
PRDATA	WE	1-bitowa magistrala danych z urządzeń podrzędnych do kontrolera

Dodatkowo, układ kontrolera APB posiada następujące porty do komunikacji z ALU i systemem zewnętrznym:

Nazwa	Znaczenie bitu
i_data	m-bitowe wejście danych z ALU, o szerokości dopasowanej do wyjścia ALU
i_data_ready	1-bitowe wejście, po którym ALU informuje o tym, że dane są gotowe do odbioru i wysłania, (jeżeli nastąpił błąd obliczeń w ALU, nie rozpoczynamy komunikacji, nie wysyłamy danych)
i_alu_error	1-bitowe wejście sprawdzające, czy w ALU wystąpił błąd. Jeżeli wystąpił, transfer nie rozpoczyna się
i_protocol_sel	2 bitowe wejście wyboru protokołu komunikacji, do którego ma zostać wysłany rozkaz przesłania danych, wybierany z testbench'a
o_waiting	1-bitowe wyjście sygnalizujące poza cały moduł projektu, że jest on gotowy na przyjęcie nowych danych do ALU i wysłanie, ustawiany w stan wysoki, gdy układ może przyjąć dane, odbierany w testbench'u
o_transfer_done	1 bitowe wyjście ustawiane razem z o_waiting tylko, jeżeli nastąpił transfer poprawnych danych z ALU

2. Lista realizowanych operacji ALU

Realizowane operacje:

Na dwanaście operacji wykonywalnych przez jednostkę składają się:

Część układu sync_arith_unit_27:

- $A \gg\sim B$, dla **i_op = 4'b0000**
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 = 4'b0001**
Dodawanie A i B.

- $\frac{A}{B}$, dla **i_op = 4'b0010**
Dzielenie liczby A przez B. Jeżeli liczba B jest zerem, należy zgłosić błąd, a wyjście ma być nieokreślone.

- $ZM(A) \Rightarrow U2(A)$, dla **i_op = 4'b0011**
Zamiana liczby A z kodu ZM na kod U2.

Część układu sync_arith_unit_4:

- $A - 2 * B$, dla **i_op = 4'b0100**
Odejmowanie dwukrotności liczby B od A

- $A < B$, dla **i_op = 4'b0101**
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 = 4'b0110**
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 = 4'b0011**
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 nr sync_arith_unit_23:

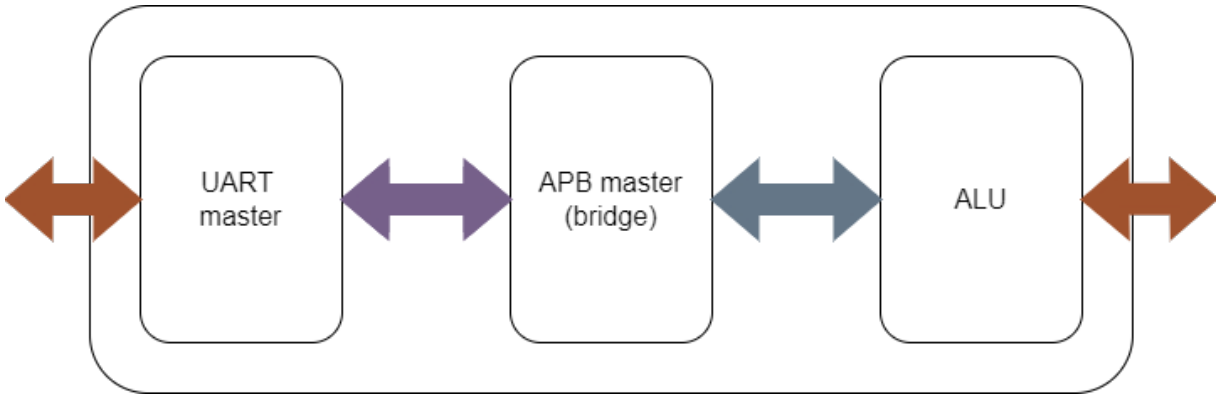
- $\sim A \gg B$, dla **i_op = 4'b1000**
Przesunięcie zaprzeczonego wektora A o B bitów w prawo. Jeżeli B bitów jest mniejsze od zera, układ ma zgłosić błąd, a wartość wyjściowa ma pozostać nieokreślona.

- $\sim A \geqslant B$, dla **i_op = 4'b1001**
Sprawdzenie czy zaprzeczona liczba A jest większa bądź równa liczbie B. Gdy warunek jest spełniony, układ ma wystawić na wyjściu liczbę większą od zera, w przeciwnym wypadku ma to być liczba 0.

- A/B, dla **i_op = 4'b1010**
Dzielenie liczby A przez B. Jeżeli liczba B jest zerem, należy zgłosić błąd, a wyjście ma być nieokreślone.

- $ABS(B)$, dla **i_op = 4'b1011**
Wartość absolutna liczby B. Jeżeli nie można dokonać poprawnej konwersji należy zgłosić błąd, a wyjście układu ma pozostać nieokreślone.

3. Schemat blokoway realizowanego modułu



Rys. 1. Schemat blokowy układu, UART-APB-ALU

4. Synteza logiczna

Do syntezy używaliśmy yosys'a. Pobraliśmy program z oficjalnego GitHuba projektu Yosys i używaliśmy zgodnie z dokumentacją. By przeprowadzić popraną syntezę odpalaliśmy Yosys'a z dowolnego miejsca w drzewie plików. Stworzyliśmy plik do łatwej automatycznej syntezy, tak by od razu wykonywały się wszystkie z góry zadane polecenia w CLI:

```
# reading the file
read_verilog -sv APB.sv

# copying read file into a temp one
copy APB APBrtl

# adjusting hierarchy
hierarchy -top APBrtl

# synthesize
synth

# chhosing gates
abc -g AND,OR,XOR

# optymalization
opt_clean

# saving the file
write_verilog -noattr APBrtl.sv

# printing stats
stat
```

Rys. 2. plik run.ys - do ułatwienia przeprowadzania syntezy przy pomocy yosysa

W wyniku powyższej syntezy uzyskiwaliśmy wiele linijek raportu. Jednak załączanie wysztkich wydruków tutaj, zajęłoby ok. 30 stron, dlatego też załączamy jedynie wydruki statystyk:

— ALU:

```
=== synth_new_alu ===

   Number of wires:                272
   Number of wire bits:            429
   Number of public wires:          37
   Number of public wire bits:     194
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                273
     $_ANDNOT_                      60
     $_AND_                          6
     $_AOI3_                        16
     $_AOI4_                        38
     $_DFF_P_                       13
     $_MUX_                         25
     $_NAND_                        16
     $_NOR_                          2
     $_NOT_                         30
     $_OAI3_                        16
     $_OAI4_                        11
     $_ORNOT_                       18
     $_OR_                           3
     $_XNOR_                         1
     $_XOR_                          6
```

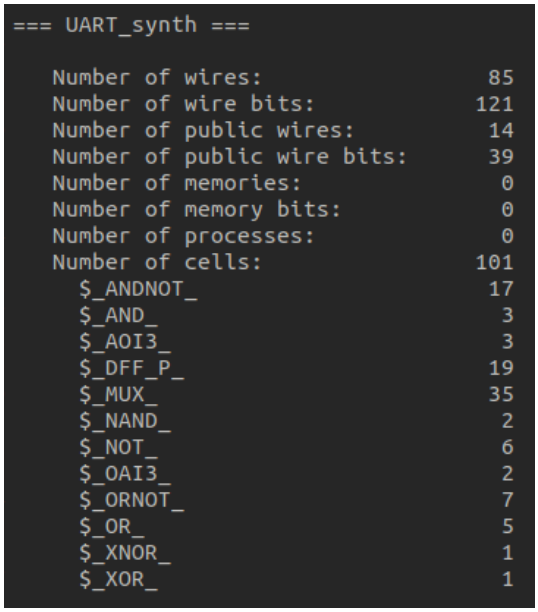
Rys. 3. wynik statystyk po syntezie ALU

— APB:

```
7. Printing statistics.

=== APBrtl ===

   Number of wires:                48
   Number of wire bits:            65
   Number of public wires:         19
   Number of public wire bits:     36
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                46
     $_AND_                        27
     $_DFF_P_                       3
     $_NOT_                         7
     $_OR_                          9
```

Rys. 4. wynik statystyk po syntezie APB

— UART:



Rys. 5. wynik statystyk po syntezie UART

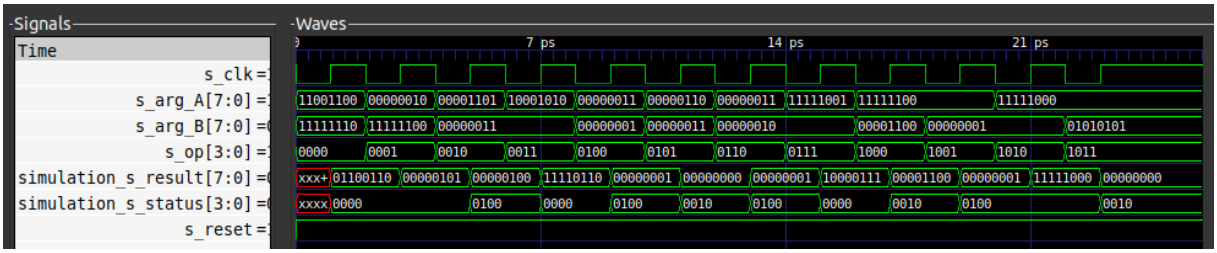
5. Kompilacja i przeprowadzone symulacje - testy

By kompliować pliki, również jak do syntezy przygotowaliśmy plik automatyzujący część poleceń. Dzięki temu mogliśmy łatwo i szybko kompilować pliki, syntezować je, oraz uzyskiwać wymagane przebiegi. Oto ten plik makefile:

```
NAME           = APB
MODEL_FILES = ${NAME}.sv
RTL_FILES   = ${NAME}rtl.sv
TB_FILES    = APBtb.sv

rtl:
	yosys -s run.ys | tee synth.log

sim: compile
	./${NAME}.iveri.run

compile: clear
	iverilog -g2012 ${MODEL_FILES} ${RTL_FILES} ${TB_FILES} -o ${NAME}.iveri.run | tee ${NAME}.iveri.log

signals:
	gtkwave signals.vcd &

clear:
	if [ -f ${NAME}.iveri.run ] ; \
	    then rm ${NAME}.iveri.run ; fi
```

Rys. 6. plik makefile - stworzony do ułatwienia operacji na plikach

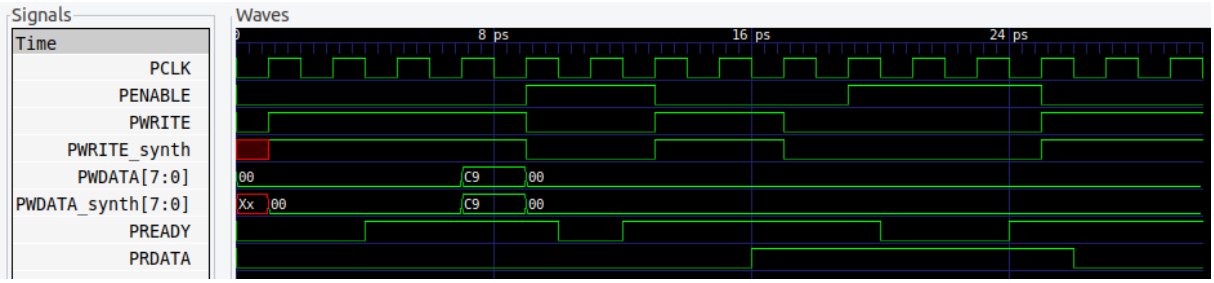
Każdą z operacji testowaliśmy wielokrotnie osobno w ramach projektów indywidualnych, dlatego nie załączamy w poniższej dokumentacji każdego z testó. Jednak by sprawdzić rzeczywiście działanie naszej jednostki ALU przeprowadziliśmy testy jednostkowe na wspólnej jednostce i zaprezentowaliśmy poniżej:



Rys. 7. Przebiegi/sygnały symulujące działanie ALU na 12 operacjach

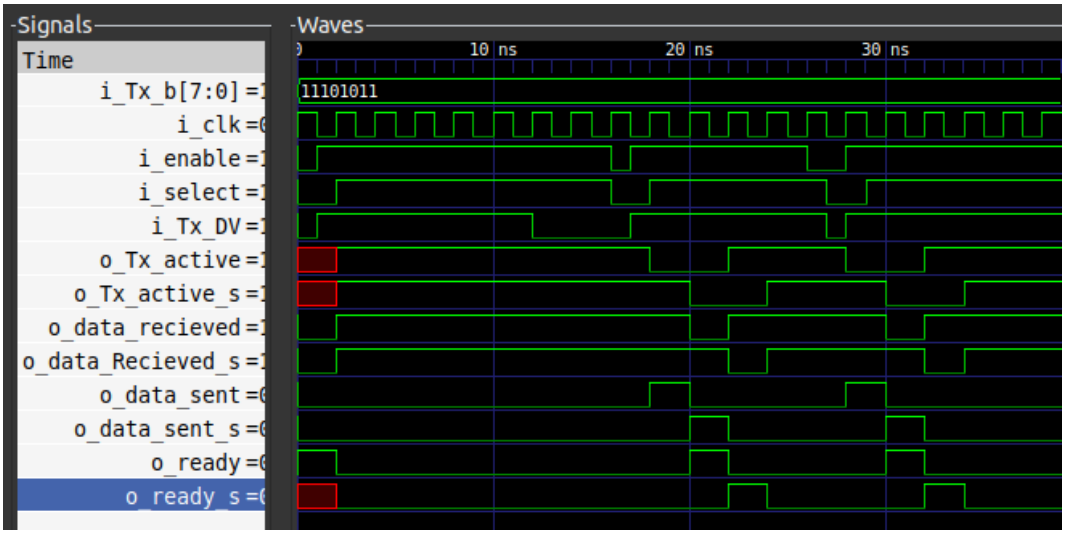
Jak widać, każda z operacji naszej jednostki ALU działa, a wynik przez nią zwracany zgadza się z wynikiem oczekiwanym.

Dodatkowo przeprowadzilimy wyizolowaną symulację działania APB master, po kompilacji i stworzeniu przykładowych testów uzyskaliśmy następujące przebiegi:



Rys. 8. Przebiegi/sygnały symulujące działanie APB dla testowych danych

Wyniki z przeprowadzonej symulacji działania UART dla przykładowych testów po kompilacji:



Rys. 9. Przebiegi/sygnały symulujące działanie UART dla testowych danych