Projekt Indywidualny, Systemy Cyfrowe i Komputerowe

Projekt wykonawczy modułu **sync_arith_unit_12** operującej na liczbach w kodzie **ZM**

Wykonująca: Anna Dzieżyk
Opiekun: Bartosz Dec

Semestr zima 2023/2024

Spis treści

1 Cel i opis projektu

Celem projektu jest implementacja synchronicznej jednostki arytmetyczno-logicznej **sync_arith_unit_12** realizującej operacje arytmetyczne, logiczne i inne zapisane na liczbach całkowitych zapisanych w kodzie **ZM**.

W skład realizacji projektu wchodzi:

- Model układu cyfrowego opisany za pomocą języka opisu sprzętu *SystemVerilog*
- Synteza układu przy użyciu narzędzia open-source *yosys*
- Realizacja modułu(-ów) testowych *testbench* w celu weryfikacji poprawności działania zaimplementowanego modelu wraz z zamiejszczonymi w niej raportami zawierającymi wyniki działania przed i po syntezie, jak również dane statystyczne dotyczące syntezy (dostarczone przez program *yosys*).

2 Realizowane przez układ operacje

Implementowany model ma zostać zaprojektowany tak, aby realizował poniżej zapisane operacje na dwóch *m-bitowych* wektorch wejściowych **A** i **B**:

- $A >> \sim B$

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

- $A > \sim B$

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

- $\sim$ A[B] = 1

  Wynikiem operacji jest ustawienie bitu w liczbie $\sim A$ o index B (licząc od bitu najmniej znaczącego) ustawionym na wartość 1. Jeżeli liczba B jest mniejsza od 0 lub większa od szerokości wektora A, układ ma zgłaszać błąd.

- *ZM(A)=> U2(A)*

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

3 Porty układu

Układ ma mieć określone porty wejściowe i wyjściowe.

- **i_op** - n-bitowe wejście określające kod operacji
- **i_arg_A** - m-bitowe wejście argumentu A
- **i_arg_B** - m-bitowe wejście argumentu B
- **i_clk** - wejście zegarowe układu
- **i_reset** - wejście resetu synchronicznego wyzwalanego stanem niskim
- **o_result** - wyjście synchroniczne z układu

Dodatkowo, jednostka arytmetyczna posiada 4-bitowe wyjście synchroniczne **o_status** informujące określonymi bitami o statusie powiązanym z wynikiem operacji:

- **bit ERROR** - sygnalizacja o tym, iż wynik operacji został określony niepoprawnie.
- **bit NOT_EVEN_0** - sygnalizuje nieparzystą liczbę zer w wyniku. Bit ten ma być ustawiany na 0 zawsze, gdy jest sygnalizowany błąd operacji.
- **bit ZEROS** - sygnalizuje, że wszystkie bity wyniku ustawione są na 0. Bit ten ma być ustawiony na 0 zawsze, gdy sygnalizowany jest błąd operacji.
- **bit OVERFLOW** - bit ten sygnalizuje, że nastąpiło przepełnienie i wynik operacji wykracza poza szerokość wektora wyjściowego.

4 Sprawozdanie z syntezy i realizacji jednostki sync_arith_unit_12

Realizacja projektu polegała na zaprojektowaniu czterech małych modułów, zsyntezowaniu ich, po czym przetestowaniu ich samodzielnej pracy w module testbenchu.

Następnie po potwierdzeniu poprawności pracy w.w. modułów, dołączyłam ich instancje do pliku z modułem ALU. Moduł ALU łączy działanie wszystkich czterech modułów. Jego funkcjonaność polega na przyjęciu wartości[1]:

- arumentów A, B,
- dwubitowego operandu wyboru operacji,
- sterowanego resetu synchronicznego,
- zegara.

Wtedy wykonywane jest działanie, a następnie generowane są wartości wyjść: wyniku i statusu operacji. Do tego modułu też napisałam testbench sprawdzający działanie jednostki arytmetyczno-logicznej. Wszystkie testy - i te jednostkowe i testy całości wykazały poprawne zachowanie układu i oczekiwane wyniki.

4.1 Moduł przesunięcia bitowego ($A >> \sim B$)

Działanie tego modułu zostało opisane w podpunkcie 1. punktu 2. [2]

Lista portów modułu

- **i_arg_A, i_arg_B** - wejścia zapewniające operandy, na których będzie wykonywana operacja
- **o_result** - wynik operacji
- **o_error** - flaga przyjmująca wartość 1, gdy wykrywany jest błąd operacji

[1] Lista portów układu znajduje się w punkcie 3. Listy portów do poszczególnych modułów znajdą się w ich opisach poniżej.

[2] Działanie każdego "małego" modułu opisane jest w punkcie 2., dlatego w pozostałych raportach z syntez małych modułów nie będzie opisów ich działania

Symulacja i synteza

Synteza modułu przesunięcia przebiegła bez zatrzasków, w pliku synth.log nie znajdujemy też błędów. Po syntezie, działanie obydwu plików z modułami przed syntezą ("przesuniecie.sv") i po ("przesuniecie_rtl.sv") zostało sprawdzone w module "testbench". Wyniki przebiegów zegarowych tej symulacji możemy obserwować w programie GTKWave, co obrazuje poniższy rysunek.

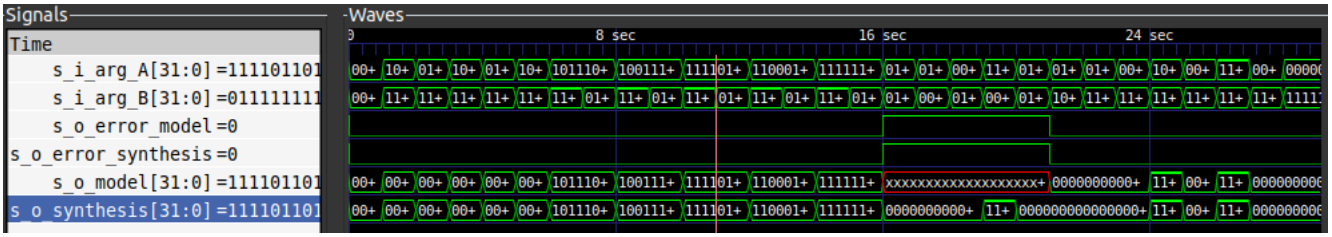

Figure 1: *Przebiegi sygnałów użytych w testbenchu, pokazane w programie GTKWave*

Stwierdzenie poprawności zachowania modułu "przesunięcie.sv" umożliwiła konktretna konstrukcja testbenchu. Pierwsze 5 wartości operandów A oraz B, czyli przebieg do szóstej sekundy, miał obrazować sytuację "idealną", która miała spełniać wymogi:

- A wylosowane z przedziału liczb możliwych do zapisania na m-bitach,
- B wylosowane z przedziału liczb od 0 do maksymalnej liczby bitów A.

Więc: flaga błędu nie jest podniesiona, wyniki modułów przed syntezą i po są identyczne.

Przypadek ten powtarza się dla przesunięć przyjmujących wartości "dodatnie zero" oraz "ujemne zero", występujących w kodowaniu liczb ZM (od 6. do 16. sekundy), dla przesunięcia większego od 32 bitów oraz dla szczególnych przypadków, gdzie operandy A i B są zerami "dodatnimi" i "ujemnymi".

Jedyna sytuacja, w której podnoszona jest flaga błędu to od 16. do 21. sekundy, wtedy też w module przed syntezą wynik jest niezdefiniowany.

4.2 Moduł porównania liczb ($A > \sim B$)

Lista portów modułu

- **i_arg_A, i_arg_B** - wejścia zapewniające operandy, na których będzie wykonywana operacja
- **o_result** - wynik operacji

Symulacja i synteza

Synteza modułu porównania przebiegła bez zatrzasków, w pliku synth.log nie znajdujemy też błędów. Po syntezie, działanie obydwu plików z modułami przed syntezą ("porownanie.sv") i po ("porownanie_rtl.sv") zostało sprawdzone w module "testbench". Wyniki przebiegów zegarowych tej symulacji możemy obserwować w programie GTKWave, co obrazuje poniższy rysunek.

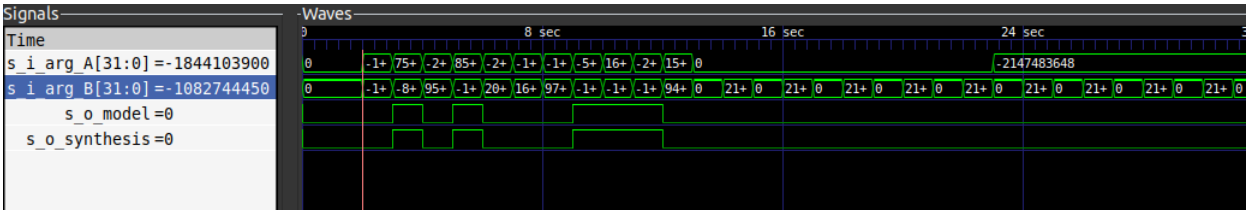

Figure 2: *Przebiegi sygnałów użytych w testbenchu, pokazane w programie GTKWave*

Więc: do 10. sekundy obserówjemy porównywanie losowych wartości A i B, następnie aż do końca symulacji A i B przyjmują wartości zer "dodatnich" i "ujemnych" mając dawać wynik porównania równy 0. Po wyrywkowym sprawdzeniu pierwszej części testbenchu oraz drugiej, można stwierdzić, że moduł działa prawidłowo.

4.3 Moduł ustawiania wartości 1 na zadanym bicie (~ A[B] = 1)

Lista portów modułu

- **i_arg_A, i_arg_B** - wejścia zapewniające operandy, na których będzie wykonywana operacja

- **o_result** - wynik operacji
- **o_error** - flaga przyjmująca wartość 1, gdy wykrywany jest błąd operacji

Symulacja i synteza

Synteza modułu ustawienia bitu na 1 przebiegła bez zatrzasków, w pliku synth.log nie znajdujemy też błędów. Po syntezie, działanie obydwu plików z modułami przed syntezą ("ustawienie.sv") i po ("ustawienie_rtl.sv") zostało sprawdzone w module "testbench". Wyniki przebiegów zegarowych tej symulacji możemy obserwować w programie GTKWave, co obrazuje poniższy rysunek.

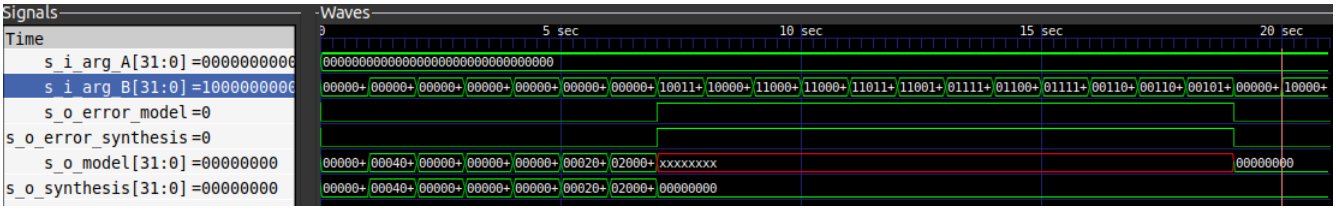

Figure 3: *Przebiegi sygnałów użytych w testbenchu, pokazane w programie GTKWave*

Budowa testbencha umożliwia na poprawne wykonanie operacji ustawienia bitu od 1. do 7. sekundy. Następnie sprawdzane jest podnoszenie flagi error dla B mniejszego od zera i większego od zakresu bitów A. Ostatnie dwie sekundy symulacji są poświęcone na sprawdzenie poprawności zachowania modułu dla dwóch różnych reprezentacji zera w argumencie B - dla nich nie powinna się pojawiać flaga error. Na podstawie powyższych faktów można uznać, że moduł ustawiania bitu na 1 działa poprawnie.

4.4 Moduł konwersji zapisu liczby z ZM na U2 ($ZM(A) => U2(A)$)

Lista portów modułu

- **i_arg_A** - wejście zapewniające operand, na którym będzie wykonywana operacja
- **o_result** - wynik operacji

- **o_error** - flaga przyjmująca wartość 1, gdy wykrywany jest błąd operacji

Symulacja i synteza

Synteza modułu konwersji przebiegła bez zatrzasków, w pliku synth.log nie znajdujemy też błędów. Po syntezie, działanie obydwu plików z modułami przed syntezą ("konwersja.sv") i po ("konwersja_rtl.sv") zostało sprawdzone w module "testbench". Wyniki przebiegów zegarowych tej symulacji możemy obserwować w programie GTKWave, co obrazuje poniższy rysunek.

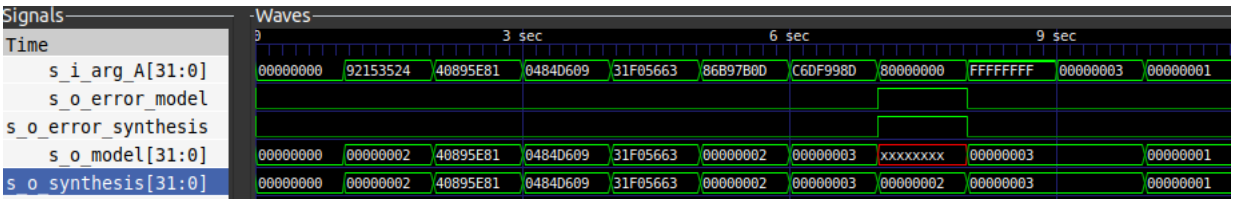

Figure 4: *Przebiegi sygnałów użytych w testbenchu, pokazane w programie GTKWave*

Testbench "konwersja_tb.sv" napisany jest w taki sposób, by pierwsze 7 sekund moduł konwertował liczby będące w zakresie konwersji, DUT wykonało poprawnie wszystkie te operacjem flaga error nie jest podniesiona. Następnie sprawdzany jest przypadek "ujemnego" zera, którego nie da się poprawnie przekonwertować, DUT ma wtedy podnieść **o_error**, wynik ma pozostać niezdefiniowany. Ostatnie 3 testy sprawdzają zachowanie modułu dla "dodatniego" zera oraz maksymalnej i minimalnej wartości z zakresu konwersji.

4.5 Moduł ALU

Moduł ALU połączył 4 moduły w całość oraz otrzymał kilka nowych sygnałów, nieużywanych w "małych modułach" **Lista portów modułu**

- **i_arg_A, i_arg_B** - wejście zapewniające operandy, na których będzie wykonywana operacja
- **o_result** - wynik operacji
- **o_error** - flaga przyjmująca wartość 1, gdy wykrywany jest błąd operacji. Każdej operacji, w której możliwe jest wykrycie błędu, odpowiada wewnętrzny oddzielnie nazwany i zadeklarowany sygnał błędu.

- **o_status** - synchroniczne wyjście z ALU pozwalające sprawdzić: czy wynik posiada parzystą liczbę zer, czy wynik składa się z samych zer, czy wynik jest błędny oraz czy nastąpiło przepełnienie[3] Wyjście to ma szerokość czterech bitów, ALU sprawdza zaistnienie w.w. sytuacji przy wykonywaniu operacji i podnosi odpowiednie flagi.

- **i_clk** - wejście sygnału zegarowego.

- **i_reset** - wejście resetu synchronicznego, aktywowanego narastającym zboczem zegara.

- **i_op** - wejście ustalające, jaka operacja zostanie wykonana.

Symulacja i synteza

Synteza "sync_arith_unit_12.sv" przebiegła bez zatrzasków, w pliku synth.log nie znajdujemy też błędów. Po syntezie, działanie obydwu plików z modułami przed syntezą ("sync_arith_unit_12.sv") i po ("sync_arith_unit_12_rtl.sv") zostało sprawdzone w module "testbench". Wyniki fragmentu przebiegów zegarowych tej symulacji możemy obserwować w programie GTKWave, co obrazuje poniższy rysunek.

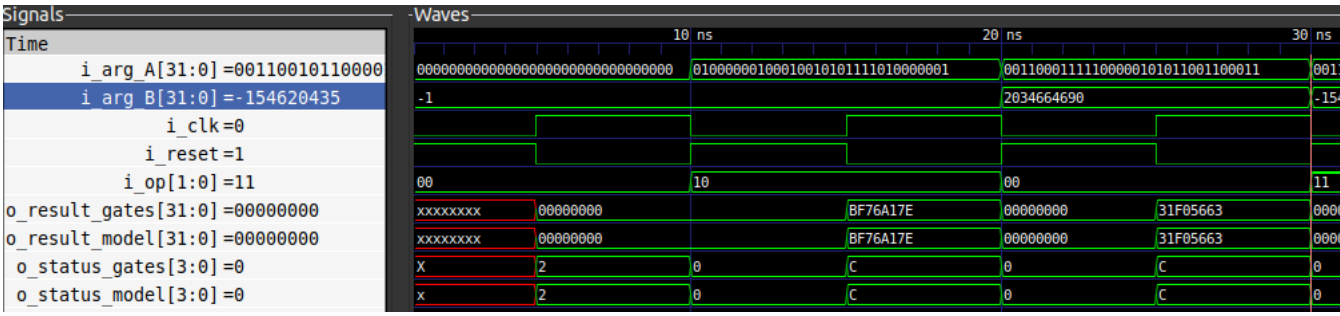

Figure 5: *Przebiegi sygnałów użytych w testbenchu, pokazane w programie GTKWave*

Dane modelu (ALU przed syntezą) i bramek (ALU po syntezie) są zgodne wszędzie tam, gdzie wynik nie pozostaje niezdefiniowany. Wynik pozostaje niezdefiniowany tylko w sytuacjach, gdy jednostka ma wykonać przesunięcie bitowe (kod 11), a sygnał B reprezentuje wartość ujemną, czyli dokładnie tak, jak się można było spodziewać. Sygnał resetu został ustawiony przeciwnie do

[3]Przepełnienie nie jest możliwe do pojawienia się w żadnym z "małych" modułów.

zegara, by umożliwić zwalnianie danych z wyjścia przed otrzymaniem wyniku następnej operacji. Biorąc pod uwagę analizę przebiegu sygnałów, działanie jednostki jest poprawne.

5 Schemat blokowy realizowanej jednostki sync_arith_unit_12

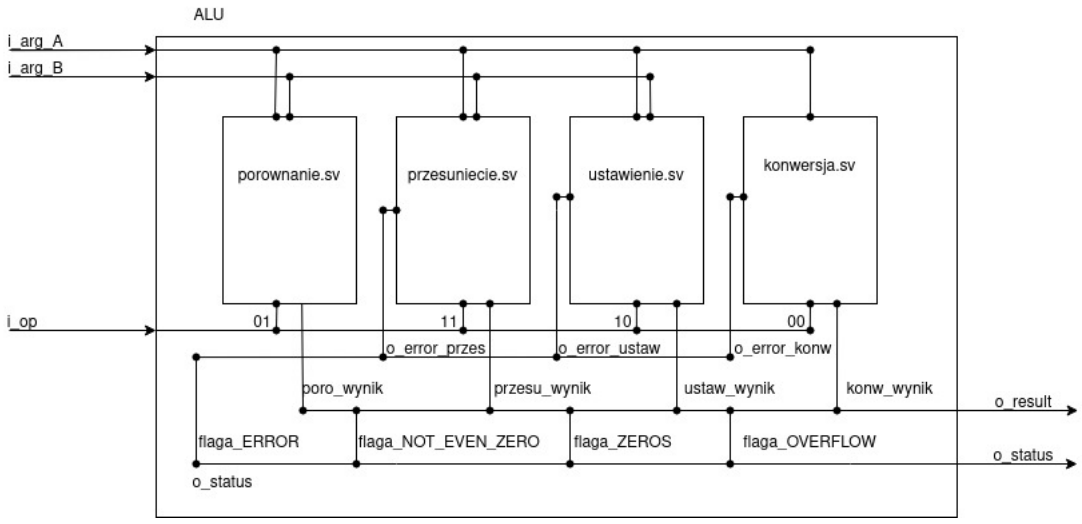

Figure 6: *Schemat blokowy realizowanej jednostki wraz z nazwanymi w niej sygnałami*