Warszawa, 12.12.2023

Projekt jednostki wykonawczej **exe_unit_w38** operującej na liczbach w kodzie U2.

Alicja Misterka

Cel projektu

Celem projektu jest zaimplementowanie jednostki wykonawczej **exe_unit_w38** realizującej operacja arytmetyczne, logiczne i inne na liczbach całkowitych zapisanych w kodzie U2.
W skład realizacji projektu wchodzi:

1. implementacja modelu **exe_unit_w38** w języku *SystemVerilog*.
2. synteza logiczna **exe_unit_w38** przy użyciu programu *YOSYS*.
3. realizacja modułu **testbench** i weryfikacja poprawności działania **exe_unit_w38** na drodze symulacji logicznej w *ICARUS VERILOG* zarówno modelu (przez syntezą logiczną) jak i moduły uzyskanego po syntezie logicznej; wyniki symulacji układów powinny być identyczne.
4. napisanie specyfikacji zrealizowanego układu wraz z raportem zawierającym wyniki realizacji projektu.

Wstęp

Przedmiotem pracy jest jednostka **exe_unit_w38** inicjująca różnorodne operacje na danych wejściowych. Moduł odpowiedzialny jest za przeprowadzanie operacji arytmetycznych, porównań oraz konwersji kodów liczbowych i został zaprogramowany z zapewnieniem poprawności wyników zachowując odpowiednią syntezę i weryfikacji na drodze symulacji logicznej.

Jednostka **exe_unit_w38** obejmuje realizację czterech operacji na danych wejściowych zapisanych na m-bitach:

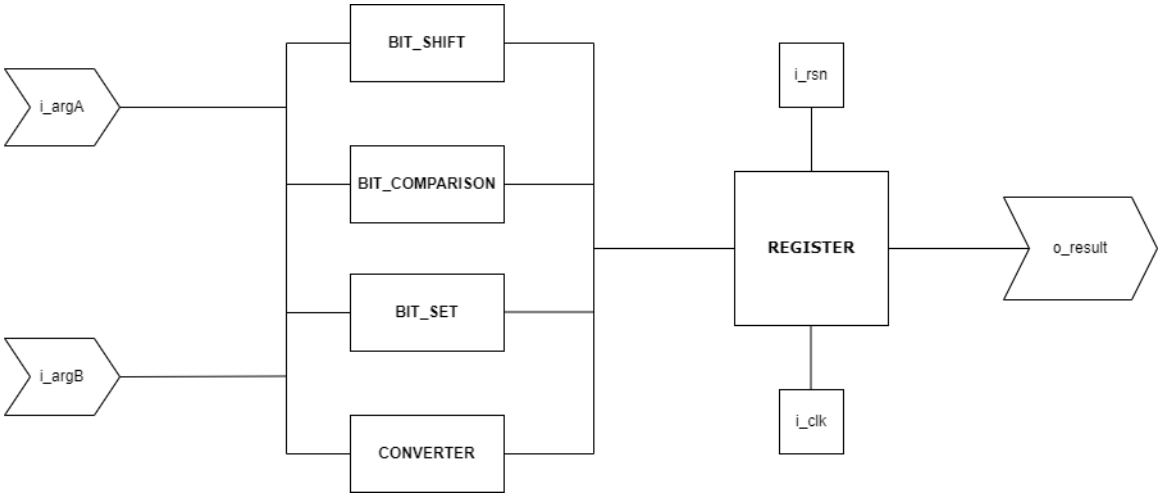
1. operacja przesunięcia bitów argumentu A o B bitów w lewo (B>0) przy zachowaniu znaku argumentu; jeśli liczba B jest ujemna, zostaje zgłoszony błąd i wynik operacji nie jest określony (*bit_shift.v*).
2. operacja porównywania argumentów A i B, gdzie warunkiem jest A<=B a wynik jest liczbą większą od zera, w przeciwnym wypadku wynik wynosi zero (*comparator.v*).
3. operacja ustawiania bitu w argumencie A na wartość 1, gdzie numer bitu jest określony w argumencie B; jeżeli wartość B jest ujemna lub przekracza liczbę bitów A - zostaje zgłoszony błąd (*bit_set.v*).
4. operacja konwersji liczby z kodu U2 na kod ZNAK-MODUŁ; jeśli konwersja nie może zostać poprawnie wykonana, zostaje zgłoszony błąd a wyniki konwersji jest nieokreślony (*u2_to_sm.v*).

Wszystkie opisane operacje jednostka **exe_unit_w38** wykonuje na poziome m-bitów zarówno na wejściu jak i na wyjściu. Spełnia przy tym wymogi precyzyjności operacji i poprawność w swoim działaniu:

1. wejście sterujące (kombinacyjne) określające rodzaj operacji do wykonania na argumentach A i B: **i_oper**, n-bitowe.
2. wejście argumentu A: **i_argA**, m-bitowe.
3. wejście argumentu B: **i_argB**, m-bitowe.
4. wejście zegara (aktywne zbocze narastające): **i_clk**.
    a. operacja nie została wykonana; wartość **o_result** jest nieokreślona: **ERROR**.
    b. w wyniku jest parzysta liczba jedynek: **EVEN**.
    c. wszystkie bity wyniku są ustawione na 1: **ONES**.
    d. w wyniku operacji nastąpiło przepełnienie (brak możliwości zapisania wyniku na zadanej liczbe bitów): **OVERFLOW**.
5. wejście resetu synchronicznego (wartość logiczna 0 ustawia wszystkie rejestry modułu **exe_unit_w38** na wartość 0): **i_rsn**.
6. wyjście synchroniczne (z rejestru) wyniku operacji, zmiana wyjścia następuje na zboczu narastającym zegara **i_clk**: **o_result**.

Realizacja

1. *Wstępny schemat projektu*



2. *Tworzenie modułów*

Aby właściwie zainicjalizować jednostkę exe_unit_w38, stworzyłam cztery oddzielne moduły, każdy posiadający indywidualną syntezę, pliki i dane wejściowe/wyjściowe. Dzięki podjęciu takiej strategii, podczas projektu uniknęła nieczytelnego, skomplikowanego kodu oraz umożliwił mi to opracowywanie systematycznych postępów w przepływie pracy.

Proces implementacji każdego modułu wyglądał następująco:
1. Napisanie surowego kodu w języku *Verilog* (**.v*), a następnie sprawdzenie błędów kompilacji za pomocą debugera.
2. Wykorzystanie skryptu *YOSYS* (**_run.ys*) do syntezowania i generowania programu **_synth.v*.
3. Stworzenie kodu testowego (**_tb.v*) i skompilowanie go do pliku **.vvp*.
4. Uruchomienie skryptu przy użyciu polecenia '*vvp*', generując plik **.vcd* do symulacji w programie *GTKWave*.

Spójność w podejściu implementacyjnym dla wszystkich czterech modułów ułatwiła pracę nad kodem, zapewniając bardziej uporządkowany i metodyczny proces rozwoju.

3. *Klasyfikacja plików znajdujących się w projekcie:*

Każdy moduł zawiera indywidualny plik typu:

nazwa.sv - napisany kod źródłowy modułu w języku *system verilog.*
nazwa_run.ys - plik syntezy *YOSYS.*
nazwa_synth.sv - plik utworzony w wyniku syntezy *YOSYS.*
nazwa_tb.sv - plik typu *testbench* dla kodu przed syntezą
nazwa.vpp - wynik kompilacji *nazwa_tb.sv*
nazwa.vcd - plik wykonywalny umożliwiający podgląd sygnałów przed syntezą
nazwa_tb_synth.sv - plik typu *testbench* dla kodu po syntezie
nazwa_synth.vpp - wynik komplikacji *nazwa_tb_synth.sv*
nazwa_synth.vcd - plik wykonywalny, umożliwiający podgląd sygnałów przed syntezą.

4. *Moduł "bit_set"*

Moduł *'bit_set'* ma na celu ustawienia określonego bitu na wektorze wejściowym '*i_argA'* na podstawie wartości podanej w '*i_argB'*. Można go zainstancjonować z dowolnym parametrem *'M'* określającym ilość bitów dla wyjść i wejść.

Parametry, wejścia oraz wyjścia:
M - szerokość wektorów wejściowych 'i_argA' i 'o_y'. Domyślna wartość to 8.

i_argA - wektor wejściowy o szerokości M, w którym określony bit zostanie ustawiony na 1.
i_argB - wektor wejściowy o szerokości M, reprezentujący indeks bitu do ustawienia na 1.

ERROR - sygnał wyjściowy wskazujący na stan błędu. Jest ustawiany na 1, gdy określony indeks bitu wychodzi poza zakres, a 0 w przeciwnym przypadku.
o_y - wektor wyjściowy o szerokości 'M', wynik ustawienia określonego bitu na 1 w 'i_argA'.

Plik typu testbench pokazał, że kod został wykonany poprawnie - zarówno po syntezie jak i przed, wyniki są ze sobą zgodne.

5. *Moduł "bit_shift"*

Moduł *'bit_shif't* ma na celu przesuwania bitów w lewo w wektorze wejściowym *'i_argA'* o zadaną liczbę pozycji określoną w wektorze wejściowym *'i_argB'*. Można go zainstancjonować z dowolnym parametrem *'M'* określającym ilość bitów dla wyjść i wejść.

Parametry, wejścia oraz wyjścia:
M - szerokość wektorów wejściowych i wyjściowych. Domyślna wartość to 8.

i_argA - wektor wejściowy o szerokości *M*, który będzie przesuwany w lewo.
i_argB - wektor wejściowy o szerokości *M*, określający liczbę pozycji do przesunięcia w lewo.

o_y - wektor wyjściowy o szerokości *M*, wynik przesunięcia w lewo.
ERROR - sygnał wyjściowy wskazujący na stan błędu. Jest ustawiany na 1, gdy liczba pozycji do przesunięcia jest ujemna, a 0 w przeciwnym przypadku.

Plik typu testbench pokazał, że kod został wykonany poprawnie - zarówno po syntezie jak i przed, wyniki są ze sobą zgodne.

6. Moduł "comparator"

Moduł *'comparator'* ma na celu porównywanie dwóch wektorów wejściowych *'i_argA'* oraz *'i_argB'*. Wynik porównywania jest dostarczany na wyjściu o_y, gdzie 0 oznacza, że *'i_argA'* jest większe od' *i_argB'*. Można go zainstancjonować z dowolnym parametrem M określającym ilość bitów dla wyjść i wejść.

Parametry, wejścia oraz wyjścia.
M - Szerokość wektorów wejściowych *'i_argA'* i *'i_argB'*. Domyślna wartość to 8.

i_argA - wektor wejściowy o szerokości M, do porównania.
i_argB - wektor wejściowy o szerokości M, do porównania.

o_y - wynik porównania, gdzie 0 oznacza, że *'i_argA'* jest większe niż *'i_argB'*, a 1 w przeciwnym przypadku.

Moduł typu testbench pokazał, że kod został wykonany poprawnie - zarówno po syntezie jak i przed, wyniki są ze sobą zgodne.

7. Moduł "u2_to_sm"

Moduł *'u2_to_sm'* ma na celu przekształcenie wektora '*u2'* w kodzie U2 na kod ZNAK-MODUŁ Wynik przekształcenia jest dostarczany na wyjściu *'sm'*, a sygnał *'ERROR'* sygnalizuje, czy wystąpił błąd podczas operacji. Można go zainstancjonować z dowolnym parametrem *'M'* określającym ilość bitów dla wyjść i wejść.

Parametry, wejścia oraz wyjścia:

M - szerokość wektorów wejściowego 'u2' oraz wyjściowego 'sm'. Domyślna wartość to 8.

u2 - wektor wejściowy o szerokości M, w kodzie U2 do przekształcenia.

sm - wektor wyjściowy o szerokości M, wynik przekształcenia kodu U2 na kod ZNAK-MODUŁ.
ERROR - sygnał wyjściowy wskazujący na stan błędu. Jest ustawiany na 1, gdy wystąpi przepełnienie (overflow), a 0 w przeciwnym przypadku.

Moduł typu testbench pokazał, że kod został wykonany poprawnie - zarówno po syntezie jak i przed, wyniki są ze sobą zgodne.

8. Implementacja jednostki "exe_unit_w38" w języku SystemVerilog

Jednostka o nazwie '*exe_unit_w38*' stanowi jednostkę wykonawczą obsługującą różne operacje na danych wejściowych.Moduł korzysta z modułów pomocniczych - '*bit_set*', *'bit_shift'*, *'comparator'* i *'u2_to_sm'*, aby realizować określone operacje na wektorach. Można go zainstancjonować z dowolnym parametrem *'M'* określającym ilość bitów dla wyjść i wejść.

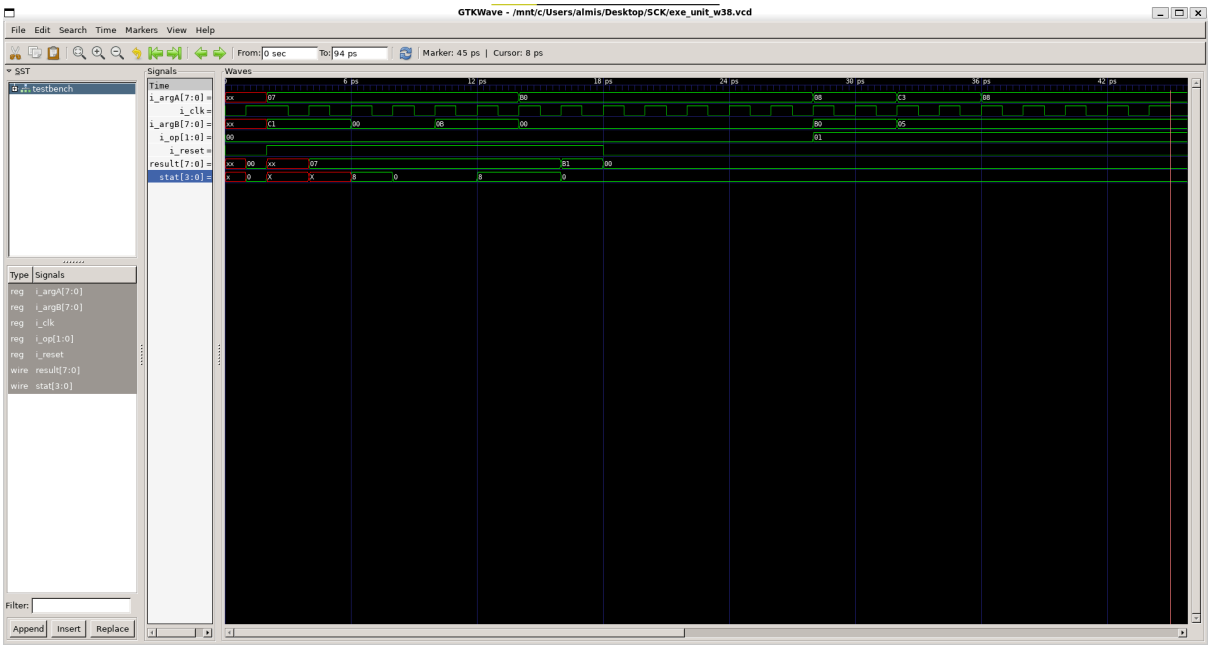
Parametry, wejścia i wyjścia:
M - szerokość wektorów wejściowych i wyjściowych. Domyślna wartość to 8.

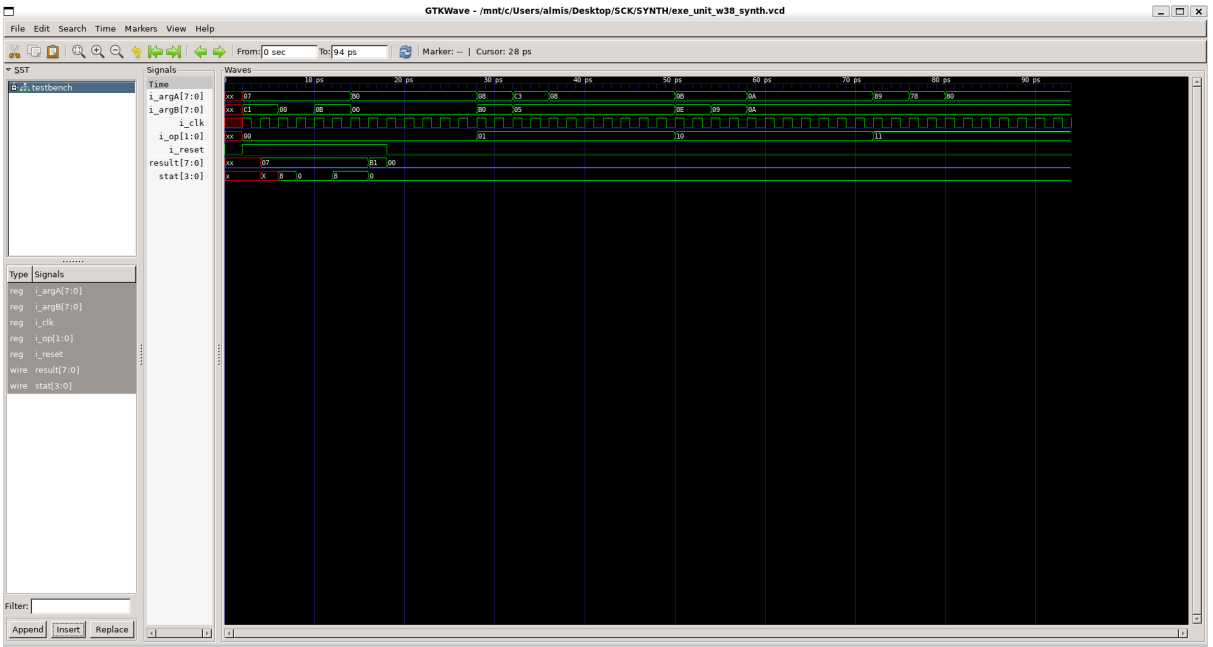
i_op - wektor wejściowy o szerokości M, określający rodzaj operacji do wykonania.
i_argA - wektor wejściowy o szerokości M, reprezentujący jedno z operandów operacji.
i_argB - wektor wejściowy o szerokości M, reprezentujący drugi operand operacji.
i_reset - sygnał wejściowy, inicjujący reset modułu.
i_clk - sygnał wejściowy zegara.

o_result - wektor wyjściowy o szerokości M, zawierający wynik wykonanej operacji.
o_stat - wektor wyjściowy o szerokości 4 bitów, zawierający informacje statystyczne:
- *Bit 3 (`OVERFLOW)*: Informacja o przepełnieniu (1, jeśli wystąpiło, 0 w przeciwnym przypadku).
- *Bit 2 (`ERROR)*: Informacja o błędzie operacji (1, jeśli wystąpił, 0 w przeciwnym przypadku).
- *Bit 1 (`ONES):* Informacja o tym, czy wynik jest liczbą nieparzystą (1, jeśli tak, 0 w przeciwnym przypadku).
- *Bit 0 (`EVEN)*: Informacja o tym, czy wynik jest liczbą parzystą (1, jeśli tak, 0 w przeciwnym przypadku).

9. *Wyniki przed i Po syntezie*

Synteza w *Yosys* umożliwia nam konwertować opis sprzętu w języku HDL na struktury bramkowe dla *FPGA* - dlatego tak ważne jest, by sprawdzić czy wyniki plików typu testbench są zgodne ze sobą zarówno po syntezie jak i przed nią. Tylko to zagwarantuje nam dobrą poprawne działania *FPGA*.





Jak widać w załącznikach, za pomocą gtkwave jesteśmy w stanie powiedzieć, że jednostka ALU exe_unit_w38 została zsyntetyzowana poprawnie.