Projekt i implementacja procesora do kalkulatora prostego operującego na liczbach stałoprzecinkowych

Aleksy Mięki

31.05.2023

Spis treści

Rozdział 1

Wprowadzenie

Projekt o nazwie "Projekt i implementacja procesora do kalkulatora prostego" stanowi praktyczną realizację zagadnień z dziedziny architektury procesorów i cyfrowego przetwarzania informacji. W ramach zadania zaprojektowany i zaimplementowany zostanie elementarny procesor, zdolny do wykonywania podstawowych operacji arytmetycznych: dodawania, odejmowania, mnożenia i dzielenia. Rdzeń systemu powstanie w oparciu o język opisu sprzętu VHDL, powszechnie stosowany w cyfrowym projektowaniu układów scalonych. Projekt ten umożliwi nabycie praktycznych umiejętności z zakresu korzystania z VHDL, a także pogłębi zrozumienie mechanizmów działania procesora na najniższym poziomie. Niniejszy raport obejmuje szczegółowy opis procesu projektowania, implementacji, testowania oraz wnioski i obserwacje poczynione podczas realizacji projektu.

Rozdział 2

Cel Projektu

Celem projektu było stworzenie operującej jednostki cpu[1] do użycia w kalkulatorze prostym, która umożliwiać będzie wykonywanie określonych wcześniej operacji operacji a mianowicie: dodawanie, odejmowanie, dzielenie, mnożenie, wprowadzenie danych, uzyskanie wyniku. Do spełnienia tych wymagań należało utworzyć: jednostkę arytmetyczno logiczną, jednostkę kontrolną, rejestry a następnie połączyć pojedyncze komponenty w odpowiedni sposób. Projekt zakładał zastosowanie języka opisu sprzętu VHDL do stworzenia modelu zachowania procesora. Ta implementacja powinna być precyzyjna i wydajna. Kolejnym celem jest przeprowadzenie dokładnych symulacji i testów, aby zapewnić poprawność działania procesora. Możliwość łatwego wprowadzania danych i odczytu wyników jest kluczowa. Celem jest również dostarczenie kompleksowej dokumentacji technicznej, która zawierałaby szczegółowy opis projektu, procesu implementacji, wyników testów oraz wszelkich problemów napotkanych podczas realizacji projektu.

[1] z and. central processing unit

Rozdział 3

Projekt i Implementacja

Rozwój tego projektu rozpoczął się od projektowania procesora na wysokim poziomie za pomocą VHDL. Zacząłem od definiowania architektury procesora, co wiązało się z tworzeniem szczegółowych projektów wszystkich komponentów, takich jak jednostka arytmetyczno-logiczna (ALU), rejestry i jednostka sterująca.

3.1 Jednostka arytmetyczno logiczna (ALU)

ALU to kluczowy element procesora, odpowiedzialny za wykonywanie operacji arytmetycznych. W tym projekcie ALU została zaprojektowana do wykonywania działań takich jak dodawanie, odejmowanie, mnożenie i dzielenie. Wykorzystanie reprezentacji stałoprzecinkowej było niezbędne do wykonania tych operacji na liczbach rzeczywistych. Operacje w ALU były sterowane przez wejścia add_enable, sub_enable, mul_enable, div_enable, które decydowały, jaką operację wykonać.

```
entity ArithmeticUnit is
  port(
    clk          : in std_logic;
    reset        : in std_logic;
    operand1     : in std_logic_vector(15 downto 0);
    operand2     : in std_logic_vector(15 downto 0);
    add_enable   : in std_logic;
    sub_enable   : in std_logic;
    mul_enable   : in std_logic;
    div_enable   : in std_logic;
    eq_enable    : in std_logic;
    reset_enable: in std_logic;
    result       : out std_logic_vector(15 downto 0)
  );
end ArithmeticUnit;
```

Do tego wykonywanie operacji arytmetycznych jest uwarunkowane wysokim zboczem zegara oraz ustawionym stanem wysokim argumentem eq_enable, opcja reset ustawia wszystkie wartości na stan niski. Wyniki operacji zapisywane są w argumencie result, który następnie kierowany jest do poświęconego na wynik rejestru, operand1 oraz operand2 również kierowane są z dwóch osobnych rejestrów. Wyniki operacji jak i ich operandy zapisywane są za pomocą std_logic_vector, są to liczby 32 bitowe umożliwiające

zapis licz stałoprzecinkowych o wielkości 32 bitów, gdzie 16 bitów poświęconych jest na część całkowitą liczby a najmłodsze 16 bitów zajmuje część ułamkowa liczby.
Operacje realizowane były na zasadzie konwersji liczb z postaci binarnej na liczbę stałoprzecinkową ze znakiem, co umożliwił w bardzo łatwy sposó język VHDL gdzie możemy taką konwersję wykonać za pomocą jednej komendy, liczby przedstawione w ten sposób możemy następnie po prostu dodać, odjąć od siebie, przemnożyć czy podzielić. Do tego sprawdzane było wystąpienie nadmiaru lub niedomiaru co w przypadku wykrycia objawiało się ustawieniem odpowiadającej zdarzeniu flagi w jednostce arytmetycznej. W przypadku dzielenia przez zero wynik był ustawiany na wartość 0.

3.2 Rejestry

Rejestry były używane do przechowywania operandów dla operacji arytmetycznych oraz wyników tych operacji. Dla tego projektu zaimplementowano rejestr przechowujący liczby 32 bitowe, dwa przechowujące operandy dla jednostki arytmetyczno logicznej, natomiast jeden do przechowywania wyniku operacji. Liczby przechowywane były za pomocą 32 bitowego std_logic_vector. Liczba przechowywana w rejestrze jest cały czas dostępna dla jednostki ALU, w wypadku ustawienia stanu zmiennej load na wysoki danta_in wprowadzane jest do argumentu data_out, pole reset ustawia zmienną data_out na zero. Sygnał load był również synchronizowany za pomocą systemowego zegara.

```
entity DataRegister is
  port(
    clk      : in std_logic;
    reset    : in std_logic;
    load     : in std_logic;
    data_in  : in std_logic_vector(31 downto 0);
    data_out : out std_logic_vector(31 downto 0)
  );
end DataRegister;

architecture Behavioral of DataRegister is
  signal temp : std_logic_vector(31 downto 0);
begin
  process(clk, reset)
  begin
    if reset = '1' then
      temp <= (others => '0');
    elsif rising_edge(clk) then
      if load = '1' then
        temp <= data_in;
      end if;
    end if;
  end process;

  data_out <= temp;

end Behavioral;
```

3.3 Jednostka Sterująca

Jednostka sterująca była odpowiedzialna za zarządzanie operacjami procesora. Sterowała działaniem ALU, dostarczając odpowiednie instrukcje w zależności od wykonywanej operacji oraz wprowadzonego opcodu, który wprowadzany jest na zasadzie 9 bitowego wektora, który następnie dzielony jest na pojedyncze 3 bitowe kody operacji które odpowiadają ustawieniu w stan wysoki odpowiedniego pola.

```
elsif rising_edge(clk) then
  internal_opcode <= opcode_sequence(2 + 3*counter downto 3*counter);
  case internal_opcode is
    when "000" =>
      add_enable <= '1';
      sub_enable <= '0';
      mul_enable <= '0';
      div_enable <= '0';
      eq_enable <= '0';
    when "001" =>
      add_enable <= '0';
      sub_enable <= '1';
      mul_enable <= '0';
      div_enable <= '0';
      eq_enable <= '0';
    when "010" =>
      add_enable <= '0';
      sub_enable <= '0';
      mul_enable <= '1';
      div_enable <= '0';
      eq_enable <= '0';
    when "011" =>
      add_enable <= '0';
      sub_enable <= '0';
      mul_enable <= '0';
      div_enable <= '1';
      eq_enable <= '0';
    when "100" =>
      add_enable <= '0';
```

Etap implementacji obejmował również dokładne testowanie, aby zapewnić funkcjonalność i niezawodność każdego komponentu oraz procesora jako całości. Wykonano to poprzez tworzenie testbenches w VHDL, które symulowały zachowanie procesora przy różnych wejściach i warunkach.

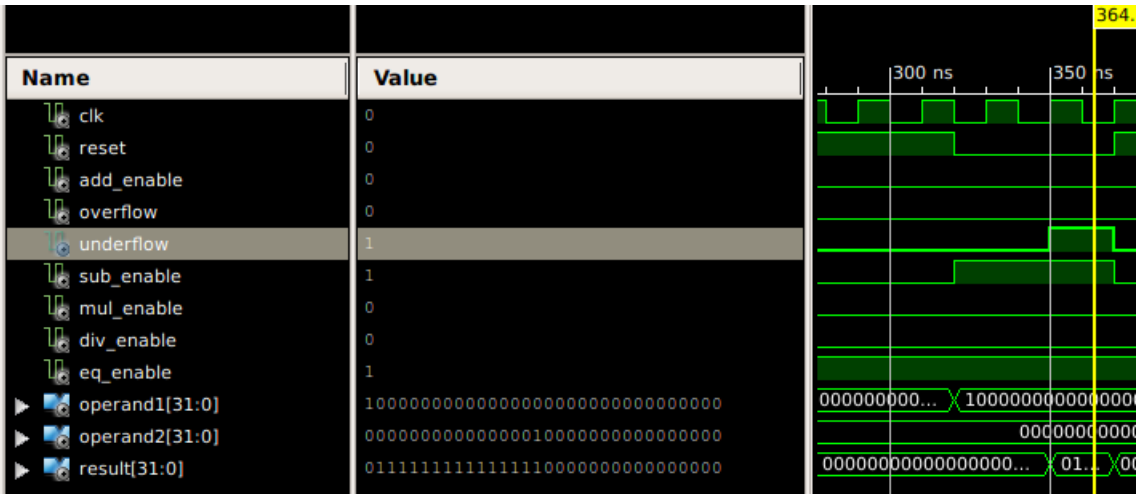
Rozdział 4

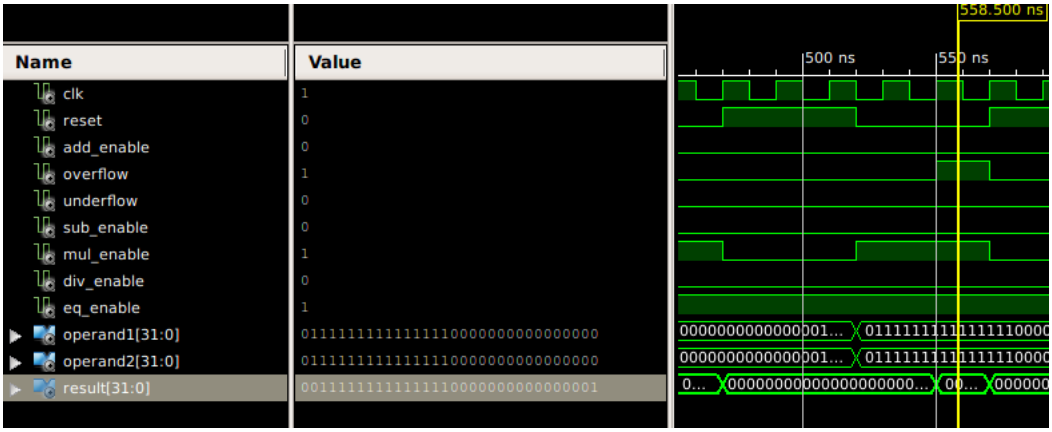
Testowanie i Weryfikacja

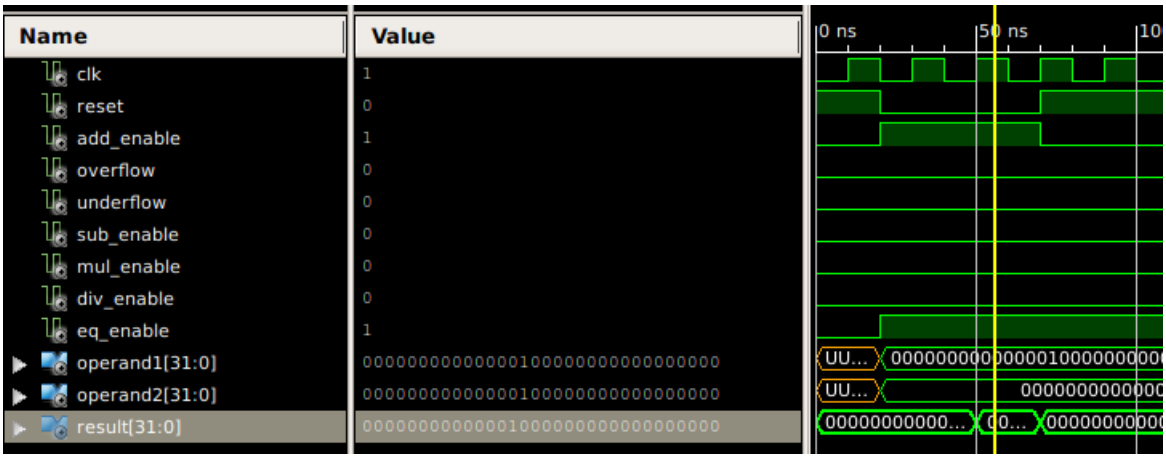
4.1 Jednostka Arytmetyczno Logiczna

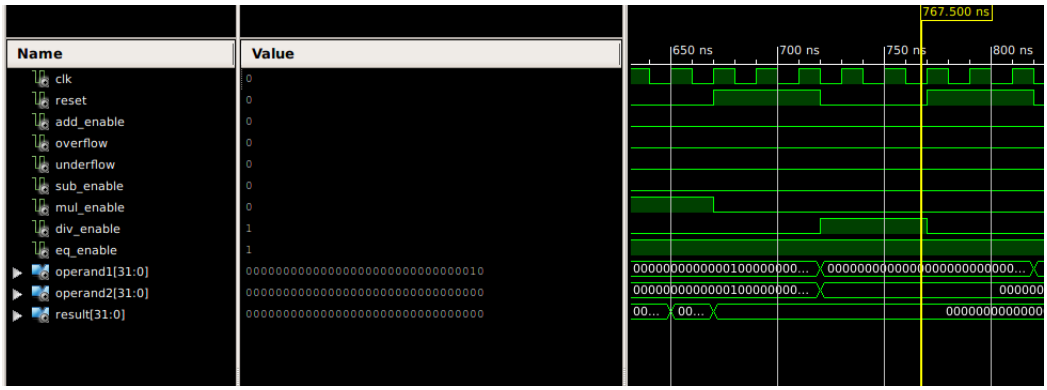
Do testowania posłużyłem się wbudowanym do narzędzia XILINX symulatorem ISIM służącym do przeprowadzania symulacji. Przetestowano działanie każdego z komponentów z osobna. Jako pierwszą możemy zobaczyć testy przeprowadzone na jednostce arytmetyczno logicznej. Testowane było poprawne działanie każdych z zaimplementowanych operacji, wczytywanie danych jak i ich otrzymywanie, przypadki graniczne, powstanie nadmiaru.

Na zamieszczonych poniżej zdjęciach możemy dostrzec kolejno operacje odejmowania z powstaniem niedomiaru, mnożenia z powstaniem nadmiaru, dodawania, dzielenia przez 0.





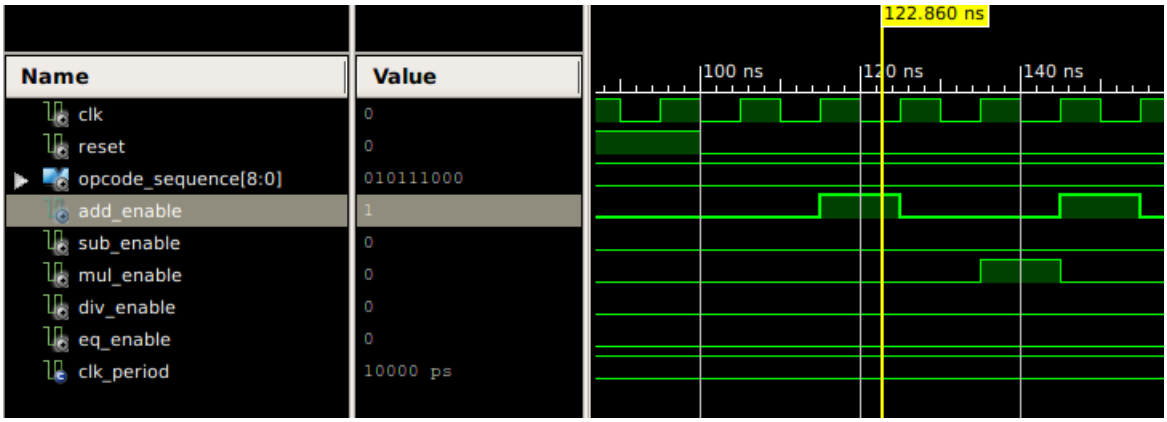




4.2 Jednostka Sterująca

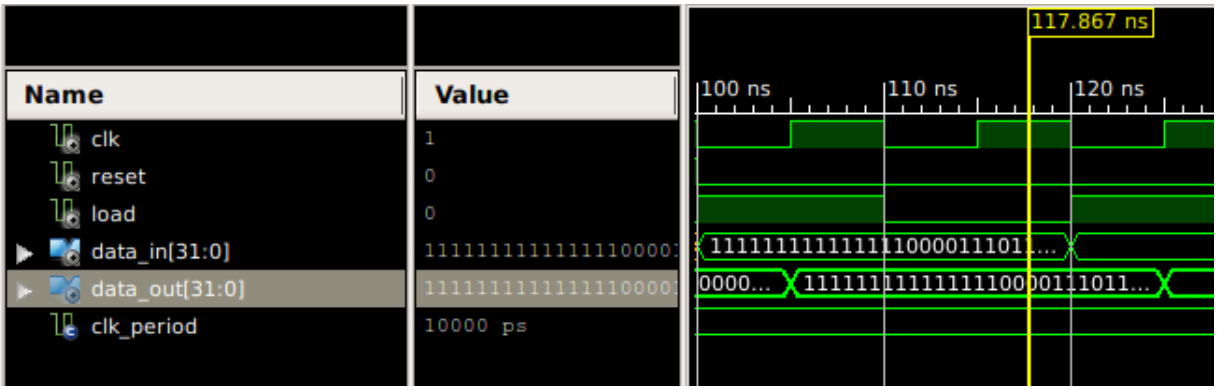
Testy dla jednostki sterującej opierały się na wprowadzeniu opCodu a następnie sprawdzenie czy jest on poprawnie dzielony na poprawne pod-operacje. Z zaprezentowanego niżej zdjęcia możemy zobaczyć iż operacje są dzielone oraz odczytywanie poprawnie, kod

powinien dodać(000), następnie sygnał powinien być zerowy z powodu wprowadzenia błędnego kodu operacji(111) a następnie powinien pojawić się sygnał mnożenia(010), wszystkie te sygnały działają poprawnie.



4.3 Rejestry

Jako ostatnie testowane były rejestry, w których testowane było poprawne ładowanie danych



Rozdział 5

Wyniki

Wnioskując po poprzednim rozdziale możemy stwierdzić, iż pojedyncze komponenty działają poprawnie obsługując wyjątki oraz przypadki skrajne.

Rozdział 6

Obliczenia na liczbach zmiennoprzecinkowych wykorzystując operacje stałoprzecinkowe

Projekt jako jeden ze swoich elementów przyjmował zastosowanie funkcji obliczających wartości liczb stałoprzecinkowych do obliczeń na liczbach o zmiennym przecinku. Reprezentacja liczb zmiennoprzecinkowych w standardzie IEEE-754 dla liczby 32 bitowej jest przedstawiana w następujący sposób:

- bit znaku - 1 bit
- wykładnik - 8 bitów
- mantysa - 23 bity

Do wykonywania operacji zmiennoprzecinkowych będziemy musieli rozszerzyć nasz zestaw funkcji o porównywanie, przesunięcia oraz porównywanie.

6.1 Dodawanie

Dodawanie liczb zmiennoprzecinkowych może wymagać następujących operacji:

- Porównanie Wykładników: Musimy porównać wykładniki obu liczb, aby zdecydować, która liczba będzie przesuwana. Wykorzystamy do tego operację porównania
- Przesunięcie Mantys: Musimy przesunąć mantysę mniejszej liczby, aby wykładniki obu liczb były zgodne. Liczba operacji przesunięcia zależy od różnicy między wykładnikami
- Dodawanie Mantys: Gdy wykładniki są ze sobą zgodne możemy dodać ze sobą mantysy, jest to operacja stałoprzecinkowa

- Normalizacja: Po dodaniu mantys wynik może wymagać normalizacji, co zazwyczaj wymaga dodatkowego przesunięcia i ewentualnej operacji dodawania lub odejmowania na wykładniku.
- Zaokrąglenie: Zaokrąglenie może wymagać operacji stałoprzecinkowych, takich jak dodawanie i porównanie.

6.2 Odejmowanie

Proces odejmowania jest podobny do dodawania, z tą różnicą, że zamiast dodawania mantys obydwu liczb będziemy je odejmować

6.3 Mnożenie

Do mnożenia potrzebować będziemy następujących operacji:

- Dodanie Wykładników: Jeśli wykładniki są równe możemy je ze sobą dodać, jeśli nie skalujemy wpierw liczbę co możemy zrobić za pomocą mnożenia oraz dodawania.
- Mnożenie Mantys: Mnożymy mantysy razem.
- Normalizacja: Jeżeli mantysa wyniku jest większa niż 2, musimy przesunąć wynik o jedno miejsce w prawo i zwiększyć wykładnik. Jeżeli mantysa wyniku jest mniejsza niż 1, musimy przesunąć wynik o jedno miejsce w lewo i zmniejszyć wykładnik, aż najważniejsza cyfra mantysy będzie wynosić 1.
- Zaokrąglanie: W przypadku mnożenia mogą wystąpić bity poza mantysą wynikową. Te bity są używane do zaokrąglenia wyniku.

6.4 Dzielenie

Dzielenie jest podobne do mnożenia z tą różnicą, że wykładniki będziemy ze sobą odejmować, natomiast mantysy ze sobą dzielić.

Rozdział 7

Wnioski

Cel projektu udało się zrealizować a mianowicie zaimplementować procesor dla kalkulatora do obliczeń stałoprzecinkowych, dużą częścią projektu było jego testowanie przez które udało się dopatrzeć wielu nieścisłości w obliczeniach stałoprzecinkowych. Sam schemat procesora umieszczony został poniżej, wszystkie wejścia możemy swobodnie modulować tak samo z wyjściami, które można podłączyć pod odpowiednio skonfigurowany wyświetlacz za pomocą którego moglibyśmy utworzyć interfejs dla użytkownika. Niestety nie udało się wykonać testów integracyjnych całego schematu a jedynie testy jednostkowe, które napisane zostały w osobnym załączonym do projektu pliku.