Architektura Komputerów 2

Liczby zdenormalizowane

czwartek nieparzysty, 18:55

Michał Sieroń
256 259

Paweł Różański
252 772

prowadzący
dr inż. Piotr Patronik

Informatyka Techniczna, Wydział Elektroniki

8 czerwca 2021

Spis treści

1 Wstęp

Zadanie projektowe polegało na analizie zawartości artykułu i implementacji przedstawionych układów w języku *Verilog*. Z powodu ograniczonego czasu na wykonanie projektu możliwe było zaimplemenetowanie jedynie układów *A1* oraz *M* przedstawionych w artykule [1].

Następnie zaimplementowano odpowiadajace im układy zgodne ze standardem *IEEE-754* [**?**]. Dokonano również porównania błędów wynikajacych z użycia danej reprezentacji liczby zmiennoprzecinkowej.

2 Opis koncepcji zapisu i formatu

Koncepcja formatu liczb zmiennoporzecinkowych, zaproponowanego w artykule [1] wzięła się z obserwacji, że standard *IEEE-754* nie został stworzony z myślą o systemach wbudowanych. Eliminacja logiki normalizującej powinna obniżyć koszt produkcji układu, a wpływ na precyzję obliczeń nie powinien mieć znaczenia w docelowych zastosowaniach. Normalizacja liczby jest sktukiem używania ukrytej jedynki w liczbach znormalizowanych. Sprawia to, że pojawia się wyjątek, który trzeba obsłużyć w sprzęcie. Proponowany format pozbywa się ukrytej jedynki, kosztem jednego z bitów mantysy. Konsekwencją tego jest zmniejszona precyzja liczb.

3 Opis metody

Wszystkie układy zostały zaimplementowane w języku opisu sprzętu *Verilog*. Jednak druga część projektu, która je ze sobą łączy i porównuje z wartością referencyjną, została napisana w języku *Python*. Dla zadanej ilości przypadków testowych generowaliśmy tyle samo par liczb typu `float`. W języku *Python*, typ `float` odpowiada liczbie zmiennoprzecinkowej o podwójnej precyzji. Wobec tego konieczne było przekonwertowanie wygenerowanych liczb na liczbę zmiennoprzecinkową o pojedynczej precyzji. W tym celu napisaliśmy funkcję `py2float` w języku *C*, która zamienia wartość typu `double` na `float`. Tak otrzymane wartości były następnie zamieniane na ich szesnastkową reprezentację. W tym celu musieliśmy uprzednio otrzymać bajtową reprezentację danej liczby, która z kolei była zamieniana na liczbę typu `int`, z której w końcu mogliśmy otrzymać reprezentację w systemie szesnastkowym.

Konwersję z liczby znormalizowanej na zdenormalizowaną zaimplementowaliśmy w tym samym skrypcie. W ten sam sposób co wcześniej, otrzymywaliśmy zapis szesnastkowy liczby lecz denormalizacja liczby wymaga operacji bitowych. Przez fakt użycia *Pythona* konieczne było do tego zamienienie liczby na listę bitów (w tym przypadku liczb całkowitych typu `int` o wartościach 0 lub 1). Tak otrzymana lista była następnie odpowiednio dzielona na części znaku, wykładnika i mantysy. Mantysa była przesuwana o jedną pozycję w prawo, a wykładnik zwiększany o jeden. Tak zmodyfikowaną reprezentację bitową zamienialiśmy z powrotem na zapis szesnastkowy.

Wykorzystując wygenerowane pary liczb tworzyliśmy pliki *Veriloga* wykorzystujące napisane przez nas moduły opisane w sekcji 4. Utworzone pliki były następnie uruchamiane, a wyniki zapisywane w pliku *csv* do dalszego przetwarzania.

4 Opis implementacji

Implementacja układu dodawania liczb zmiennoprzecinkowych została wykonana na podstawie opisu oraz modelu układu z artykułu [1]. Według nazewnictwa z artykułu układ dodawania, który odwzorowaliśmy, to *A1*. Jest to najprostsza wersja dodawania dwóch liczb zdenormalizowanych. Występuje wiele podobieństw względem zwykłej implementacji układu dodawania, a główną różnicą jest brak bloku normalizowania wyników po obliczeniu sumy. Implementację układu zaczęliśmy od dokładnego przeczytania opisu przedstawionego w artykule, a następnie stworzeniu podstawowych bloków do obliczeń zawartych na diagramie. Były to między innymi bloki dodawania dwóch liczb w systemie U2, czy też blok zamiany wartości. Wyznaczenie znaku w przypadku liczb zdenormalizowanych jest trudniejsze, ponieważ nie możemy go otrzymać w ten sam, bezpośredni sposób jak w liczbach znormalizowanych. Aby sobie z tym poradzić, autorzy artykułu wprowadzili specjalny bit (w artykule [1] nazwany sticky bit), który decydował czy przeniesienie wejściowe do sumatora powinno być ustawione na 1, czy na 0.

Kolejną implementacją, którą wykonaliśmy było stworzenie układu mnożącego dwie liczby zmiennoprzecinkowe pojedynczej precyzji, który tak samo jak układ dodawania pochodzi z artykułu [1]. Autorzy nadali mu nazwę *M*. Jest to pierwsza zaproponowana przez autorów wersja implementacji układu mnożenia liczb zdenormalizowanych. Podobnie jak układ dodawania, największą jego różnicą jest brak układu normalizacji wyniku. Implemetancja układu mnożenia zdenormalizowanych liczb zmiennoprzecinkowych została wykonana w oparciu o diagram i opis układu *M* w artykule. W architekturze, która została użyta w układzie mnożenia, autorzy zakładają, że wynik mnożenia dwóch liczb zawsze zakończy się przepełnieniem, wobec tego wykładnik jest zawsze zwiększany jeden, a mantysa przesuwana o jedna pozycję w prawo.

5 Różnice między metodami

Wynika to z faktu, że gdy porównujemy tylko wykładniki może się zdażyć ze operand z wyższego wykładnika będzie mniejszy niż drugi.

6 Narzędzia

Narzędziami, które zostały użyte przez znas są między innymi 3 języki programowania. Najbardziej korzystaliśmy z języka Verilog, który posłużył nam

do łatwiejszej implementacji układów mnożenia i dodawania. Języka C użyliśmy do implementacji układu porówania wyników a Pythona do przeprowadzenia wszystkich potrzebny testów wraz w wykonaniem wykresów obrazujących wyniki. Przy implementacji z języku Verilog używaliśmy kompilatora iVerilog. A do sprawdzenia poprawności wyników poszczególnych bloków i debugowania programu GTKWave. Diagramy wykonane do lepszego zobrazowania wyglądu układów zostały wykonane w Draw.io.

7 Opis sposobu

8 Wyniki pomiarów

9 Wnioski

Literatura

[1] S. Gonzalez-Navarro, J. Hormigo. Normalizing or not normalizing? an open question for floating-point arithmetic in embedded systems. 2017.