Łukasz Broll 225972

Krzysztof Dombek 226093

Organizacja i architektura komputerów

**Projekt:**

Skalowanie w systemie RNS

Prowadzący: dr inż. Piotr Patronik

Grupa: wtorek/TP godz. 13.15-15.00

POLITECHNIKA WROCŁAWSKA – CZERWIEC 2017

WYDZIAŁ ELEKTORNIKI

Spis treści

# Wstęp

## teoria

RNS – alternatywna reprezentacja do pozycjonowania systemów liczbowych. Jego główna zaleta to fakt, iż arytmetyka bez przenoszenia bitu nadmiaru może być wyznaczona osobno oraz równolegle w każdym kanale aby przeprowadzić operacje takie jak: dodawanie liczb całkowitych, mnożenie czy dzielenie. Testowany w kryptografii, liniowym przetwarzaniu sygnałów tj. systemów komunikacji cyfrowej zapewnia lepszą wydajność przy mniejszym zużyciu energii. System ten stosujemy również, aby wspierać dynamicznie zmieniający się zakres, nawet dla najtrudniejszych przypadków. Gwarantuje on redukcje nadmiaru. Konwersja pomiędzy reprezentacjami czyli konwersja odwrotna między RNS a ważonym systemem binarnym, wymaga dodatkowych operacji i nie może być indywidualnie wyliczana dla każdego z kanałów RNS. Operacje te nie mają jednak znaczącego wpływu na złożoność arytmetyki RNS, tylko w przypadku bardzo złożonych i intensywnych obliczeń. Tutaj pojawia się kryptografia, w której wymagane są wielokrotności bardzo dużych liczb oraz w przypadku liniowego przetwarzania sygnałów, co wymaga iteratywnego obliczania dużej liczby operacji mnożących. W przeciwieństwie do arytmetyki zmiennoprzecinkowej, arytmetyka stałej liczby całkowitej zazwyczaj wymaga skalowania, aby zapewnić, że obliczone wyniki nie przekroczą dynamicznego zakresu. Warto dodać, iż skalowanie w przeciwieństwie do odwrotnej konwersji musi być wykonywana wielokrotnie podczas przetwarzania. Dobrym przykładem obrazujący ten problem jest iteratywna natura sygnału cyfrowego, gdzie począwszy od skończonych idąc dalej aż do nieskończonych filtrów odpowiedzi impulsowych do szybkich przekształceń Fouriera i Waveleta, zazwyczaj wymagany jest dynamiczny zakres rozszerzenia, jeżeli nie stosujemy skalowania. W tym przypadkach albo RNS wspiera dynamiczny zasięg albo stosujemy je aby po raz kolejny uniknąć przepełnienia, podobnie jak obliczanie punktu stałego. Większość algorytmów RNS korzysta z Look-Up-Tables, przez co między innymi ich implementacja wymaga komponentów pamięci takich jak RAM w celu wspierania różnych zestawów modułów. Niestety nie jest to tak efektywne, gdyż koszty sprzętu rosną wraz z dynamicznym zasięgiem oraz przepustowość przetwarzania jest ograniczona i maleje wraz z wielkością pamięci. Skalowanie jest kluczową istotną operacją arytmetyczną, która jest ciężka do zaprezentowania w systemie RNS. W naszym artykule znajdujemy efektywne rozwiązanie tego problemu, która gwarantuje zakres dla ważnych klas o dużych rozmiarach dynamicznych. Klasy te zawierają standardowy 3-modułowy zestaw, jednak jego wykładnik potęgi podwójnego modułu zwiększony jest przez oraz dowolnego zestawu, który posiada dodatkowy moduł . Zestawienie chińskiego twierdzenia o resztach oraz system radix – wyliczeniowy, używając właściwości docelowych zestawów modułów, wykorzystując skalowanie bezpośrednio w obszarze RNS. Osiągamy to poprzez działanie hierarchiczne w każdym z kanałów bez konwersji wstecznej czy przedniej. System ten jest zdecydowanie bardziej efektywny. W artykule zostały zaproponowane różne rozwiązania problemów. Przedstawiając kolejno wzbogacone 3 modułowe zestawy, które wynikają z zwiększania mocy dwóch modułów, nawet jako zmienne, rozszerzony 4 moduły zestaw, który rozszerza tradycyjne 3 modułowe zestawy o czwarty element czy narastający 3 modułowy zestaw. Każdy z algorytmów może działać efektywnie dla skalowania RNS, poprzez rozwinięcia zaproponowane w artykule oraz wszelakie implementacje rozważające dynamiczne zakresy skalowania, uzależnione od potrzeb wykorzystywania danego algorytmu.

## Podstawowe pojęcia

Definiując bazę {} elementów względnie pierwszych gdzie to moduł, dowolnej liczby całkowitej X, gdzie M to iloczyn wszystkich modułów M = może być reprezentowane w systemie RNS jako N reszt zbioru {}. Poniżej przedstawiamy krótkie wyjaśnienia poszczególnych zapisów, aby lepiej zrozumieć, co dokładnie zostało zaimplementowane w użytych przez nas wzorach:

* – reszta wynikająca z dzielenia X przez moduł , przedstawiana również jako wektor bitowy .
* Reszty dla określonego modułu są przedstawione jako
* określa resztę dla modułu po przeskalowaniu.

## Idea skalowania

Skalowanie liczb binarnych, czyli dzielenie przez potęgę 2 tj., podstawy systemu, nie może być zrealizowane wprost na liczbach dziesiętnych. Algorytm, na podstawie którego został zrealizowany projekt opiera się na resztowym systemie zapisu liczb. W procesie skalowania można wyróżnić następujące punkty:

1. Przekonwertowanie dzielnej na system **R**esidue **N**umber **S**ystem (resztowy).
2. Osobne przeskalowanie każdej z reszt R1-R4 w RNS przez dzielnik.
3. Przekonwertowanie wyniku z powrotem na system dziesiętny.

## Resztowy system zapisu liczb

Zgodnie z teoriąjest to system liczbowy służący do reprezentacji liczb całkowitych za pomocą wektora reszt z dzielenia względem ustalonego wektora wzajemnie względnie pierwszych modułów. W naszym przypadku ustalony wektor modułów wynosił: { 2n -1, 2n, 2n+1, 2n+1-1}. Chińskie twierdzenie o resztach gwarantuje, że taka reprezentacja jest jednoznaczna dla liczb całkowitych ze zbioru [0,M], gdzie M jest iloczynem wszystkich modułów.

## środowiska symulacji

### PARI/GP

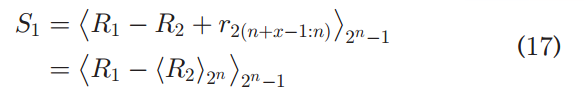
Do obliczeń czysto teoretyczno-matematycznych wykorzystaliśmy pogram Pari/GP – system komputerowy przeznaczony do szybkich obliczeń algebraicznych. Prosty interfejs konsolowy, rozbudowana dokumentacja i intuicyjna obsługa pozwoliły nam w miarę szybko zrozumieć jak program działa oraz jak korzystać z funkcji, które potrzebowaliśmy zaimplementować: tworzenie procedury, pętli, drukowanie wyników i operacje arytmetyczne. Zaimplementowany przez nas kod generował poprawne wyniki, co zostało przetestowane dla kilku różnych przykładów. Aplikacja PARI/GP została przygotowana również pod systemy z rodziny Microsoft Windows i nasz program pisaliśmy na systemie Windows 10.

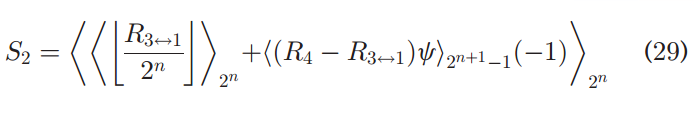
### Verilog

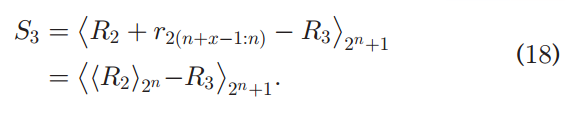
Język Veriloga wykorzystuje się do opisu sprzętu w projektowaniu oraz symulacji układów cyfrowych. Największą trudnością dla nas było zrozumienie całej idei języku opisu sprzętu. Do tej pory proste układy implementowaliśmy bezpośrednio na sprzęcie, bez komputerowego przygotowania algorytmu i symulacji. Skrócone kursy uczelniane, przykłady i dokumentacja również w tym przypadku wprowadziły nas w podstawy programowania językiem Verilog, co wystarczyło w naszym symulatorze na zadeklarowanie modułów, sygnałów, pętli, obliczeń arytmetycznych oraz modułu test\_bench. Dla potrzeb naszego projektu korzystaliśmy z internetowego środowiska kompilatora firmy Codingground dostępnego pod adresem https://www.tutorialspoint.com/compile\_verilog\_online.php

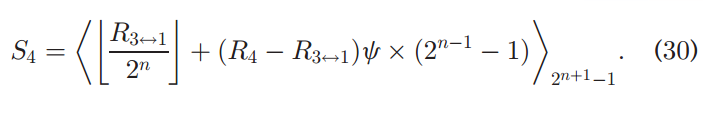
## Implementacja

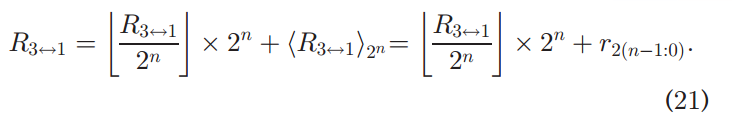
W naszej implementacji użyliśmy poniższych wyprowadzeń zastosowanych twierdzeń oraz wzorów z artykułu:

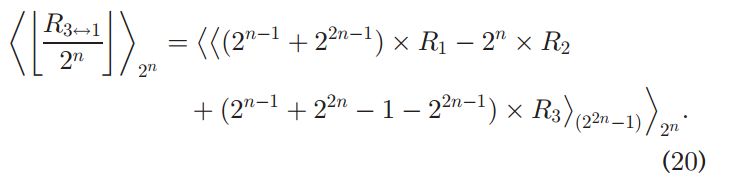


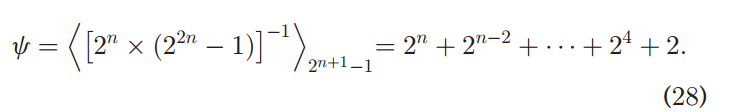












W tym miejscu warto dodać, iż parametr „widły”(wzór 28) zostaje uwzględniany tylko i wyłącznie, jeżeli podane n >= 4. W naszym przykładzie n = 8, więc korzystamy z parametru widły, aby wyliczyć składowe reszty .

Kod źródłowy implementacji w Pari/GP:

<https://pastebin.com/v7uxmfak>

Implementacja w Pari/GP jest bardzo podobna do kodu źródłowego Veroliga. W tym środowisku jednak jest o tyle prościej, że nie musimy martwić się o nadawanie danemu portu określonego rozmiaru, gdyż nie jest to konieczne. Cały kod opiera się na funkcji scalling, która tak samo przyjmuje dwa parametry – dowolnego X oraz skalar n. Trzeba w tym miejscu dodać, że nasz układ daję możliwość podania dowolnej wartości do przeskalowania. Nie trzeba samemu wyliczać reszt oraz podawać je na sztywno do naszej funkcji, gdyż program zrobi to sobie sam. Wniosek z tego taki, iż na wejście naszego układu tak naprawdę podajemy reszty, na których potem operujemy, aby uzyskać reszty wynikające ze skalowania oraz aby wyliczyć konwersję odwrotną.

Kod źródłowy implementacji w Verilogu:

<https://pastebin.com/H45PZVZw>

W naszym symulatorze podajemy dowolny X oraz skalar, który potrzebny jest do wyliczenia poszczególnych modułów, jednak tak naprawdę na wejściu naszego układu podawane są wyliczone wcześniej reszty, na których operujemy, uzyskując ostateczny wynik. Każda z reszt S1-4 rozpisana jest w osobnym module, gdzie zadeklarowane są wykorzystywane porty. Dla każdego portu określamy jaką funkcję pełni – input, output, inout – wejścia, wyjścia oraz dwukierunkowy. Korzystamy również z magistral wewnętrznych, które bezpośrednio związane są z poleceniem „wire”. Wywołując nasz program korzystamy z instancji, które wywołują poszczególne moduły. Dla każdego portu przewidzieliśmy zapas, jeżeli chodzi o rozmiar sygnału, który będzie nim wysyłany, aby umożliwić testowanie naprawdę dużych liczb. Kod mocno związany jest z implementacją w Pari/GP

Przykład – w celach lepszego zrozumienia systemu skalowania rozpisaliśmy swój przykład.

Table 1

Example of Scaling in RNS the Integer X = 2^42-1 by 2^n for the 4-Moduli A and n=8 (2^8 = 256)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Moduli | 255 | 256 | 257 | 511 |
| Original residues | 3 | 255 | 252 | 63 |
| Reverse conv. To obtain X |  | | | |
| Forward conv. – Residues of scaled result Si | S1 = 252 | S2 = 255 | S3 = 3 | S4 = 127 |
| S1 = wzor |  | | | |
| S2 = wzor |  | | | |
| S3 = wzor |  | | | |
| S4 = wzor |  | | | |

**Wnioski**

Projekt skalowania w systemie RNS opierał się na dokumencie ‘2n RNS Scalers for Extended 4-Moduli Sets’ autorstwa Leonel Sousa. Najważniejszym zadaniem na tym etapie projektu było dokładne zrozumienie treści artykułu. Naukowy język oraz fachowość nazewnictwa w języku angielskim sprawiły nam duże problemy w interpretacji treści, a bez dokładnego zrozumienia całej idei skalowania nie bylibyśmy w stanie zrealizować dalszych kroków.