Generyczny układ do mnożenia macierzy zespolonych KxN i NxM w czasie KxM taktów zegara w myhdl

Piotr Radecki

1. **Cel**

Celem jest stworzenie algorytmu do szybkiego mnożenia macierzy zespolonych. Założenia są takie,że macierze są przechowywane w dedykowanych blokach pamięci z jednym portem do zapisu i jednym do odczytu. Odczyt i zapis mogą odbywać się jedynie sekwencyjnie. Liczba zespolona jest reprezentowana w taki sposób, że na jej starszych bitach znajduje się część rzeczywista a na młodszych urojona.

1. **Opis algorytmu**

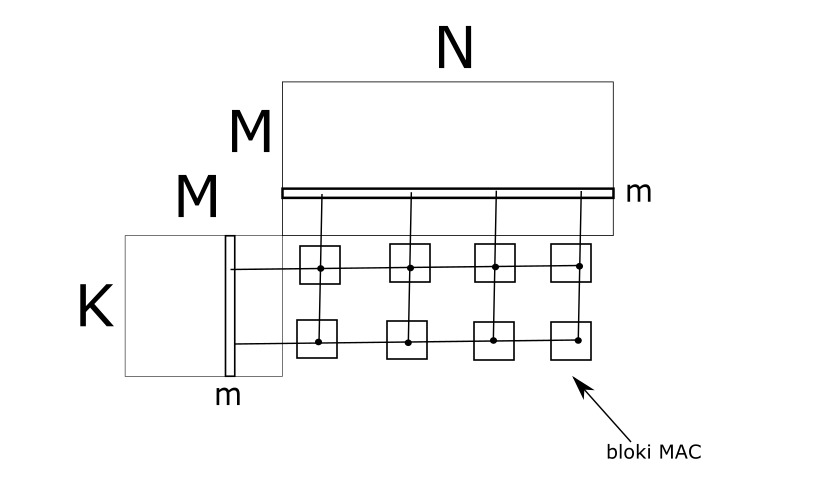
Zakładamy, że mnożymy macierze KxM-macierz X i MxN- macierz Y. Algorytm jest następujący:

1. Czekaj aż dane wejściowe będą gotowe
2. Pobierz z pamięci m-tą kolumnę macierzy X i m-ty rząd macierzy Y
3. Dokonaj mnożenia w jednym takcie zegara mnożenia tych dwóch wektorów za pomocą KxN bloków MAC.
4. Powtarzaj punkty 2 i 3 aż do przejścia przez całą macierz
5. Zapisz otrzymaną macierz Z do pamięci
6. Zakomunikuj gotowość

Czas wykonywania takiego algorytmu to:

[(max(K,N)+1)\*M + K\*N ] taktów + kilka taktów na kontrolę układu

Poniżej bardzo poglądowy wygląd układu:

fig. 1

1. **Matrix multiplier**
   1. **Lista interfejsów:**

|  |  |  |
| --- | --- | --- |
| Nazwa | Kierunek | Szerokość(wartość dla generic) |
| DAT\_WIDTH | generic | :=32 |
| K | generic | :=2 |
| M | generic | :=2 |
| N | generic | :=2 |
| ACCU\_WIDTH | generic | := 80 |
| clk | in | 1 |
| rst | in | 1 |
| input\_vld\_i | in | 1 |
| input\_rdy\_o | out | 1 |
| output\_vld\_o | out | 1 |
| output\_rdy\_i | in | 1 |
| re\_o | out | 1 |
| x\_raddr\_o | out | ceil(log2(K\*M)+1) |
| x\_dat\_i | in | DAT\_WIDTH |
| y\_raddr\_o | out | ceil(log2(M\*N)+1) |
| y\_dat\_i | in | DAT\_WIDTH |
| z\_we\_o | out | 1 |
| z\_waddr\_o | out | ceil(log2(K\*N)+1) |
| z\_dat\_o | out | DAT\_WIDTH |
|  |  |  |
|  |  |  |
|  |  |  |

1. **Szczegóły**

Dane w założeniu to 16bit signed integer- część rzeczywista i to samo urojona.

Bloki MAC są wyposażone są w 40bitowe akumulatory. Pozwala to mnożyć macierze o głębokości 255, przy 256 jak łatwo pokazać mogłoby się już zrobić przepełnienie.

Kontrola odbywa się przez interfejs axi(valid i ready).

Cały układ jest kontrolowany przez dwie maszyny stanów- główną i pomocniczą(calc). Poniżej opis.

Main FSM:

IDLE- jeśli dane na wejściu gotowe idź do CALC

CALC- Aktywuj pomocniczą maszynę stanów, gdy obliczenia skończone idź do WRITE DATA

WRITE DATA- Zapisz dane do pamięci na wyjściu, kiedy koniec przejdź do done

DONE- Wystaw valid,jeśli kontroler jest gotowy na przyjęcie danych wróc do IDLE

Calc FSM:

IDLE- jeśli main FSM wyśle sygnał przejdź do READ

READ- pobierz m-tą kolumnę i rząd, przejdź do MULT

MULT- pomnóż, jeśli już cała macierz idź do DONE, jeśli nie- wróć do READ

DONE- wystaw flagę końca i wróć do IDLE

Komentarze:

Vld\_i jest ustawiany przez kontroler jeśli pełne dane są wysłane do pamięci X i Y a także jest miejsce w pamięci Z.

Rdy\_o jest ustawiane przez układ gdy ten jest gotowy na przyjęcie danych, tj. pełne mnożenie zostało wykonane i dane poszły do pamięci Z.

Akumulatory: jeśli mnożymy załóżmy liczby 32 bitowe to bezpieczny akumulator będzie miał jakieś 80 bitów. Tak zrobię w ogóle architektura będzie 16 bitów na i i 16 bitów na q, mnożenie daje bo 32bit i jeszcze zostaje 8 na akumulację.

Maksymalny wynik mnożenia: -2^31, maksymalny wynik w akumulatorze to -2^39. Daje nam to możliwość mnożenia macierzy o wspólnym wymiaerze maksymalnie 255.

Generalnie układ powinien mnożyć w (max(N,K) + 1)\*M + K\*N + kilka taktów na kontrolę.

Czyli zegar 10ns, K=7,N=11, M=30 to powinno zająć 12\*30\*10ns + 770ns

Czyli 4.4us od momentu załadowania macierzy.

Ładowanie macierzy to oczywiście dodatkowe max(N,K)\*M taktów czyli w tymprzypadku jakieś 3.3us

Przykładowo pomnożenie macierzy 8x255 i 255x8 powinno zająć:

(9\*255+64)\*10ns ~ 24us

U mnie na komputerze(2.5GHz) liczy się w numpy ~10us

A na przykład takiej 100x10 i 10x100:

(101\*10 + 10000)\*10ns = 110us, numpy osiąga taki sam wynik.

Widać, że ogólnie największe straty to czasy zapisu i odczytu z pamięci. Komputer nas tu dogania bo to proces sekwnecyjny