AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

SYSTEMY DEDYKOWANE W UKŁADACH PROGRAMOWALNYCH

Algorytm *Fast inverse square root*

Autorzy: *Łukasz Orzeł; Jakub Świebocki*

Kierunek studiów: *Mikroelektronika w technice i medycynie*

Kraków, 2023

Spis treści

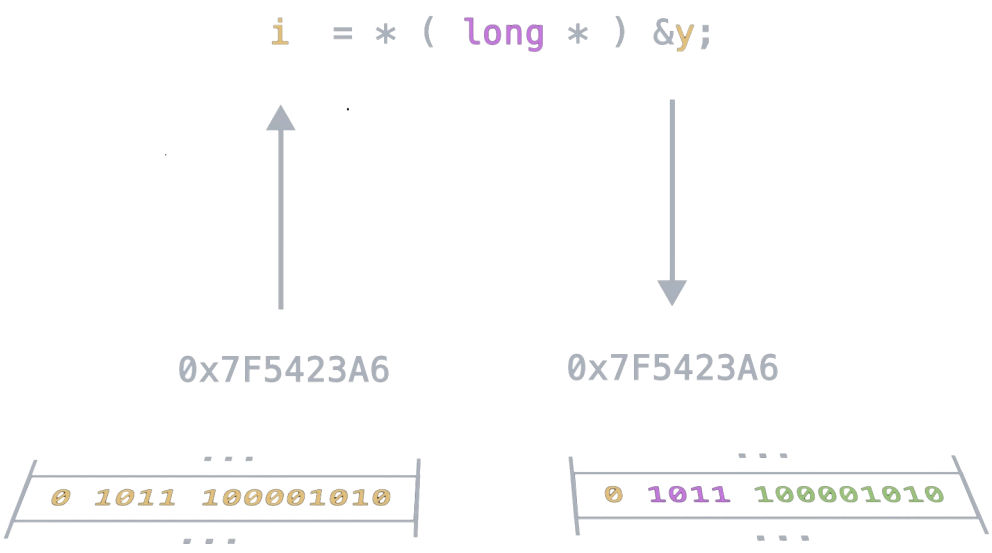
Wstęp

W niniejszym raporcie przedstawiamy wyniki projektu, którego celem było zaimplementowanie algorytmu Fast Inverse Square Root na układzie FPGA (Field-Programmable Gate Array). Jako, że algorytm Fast Inverse Square Root, ze względu na szybkość działania, był szeroko stosowany w dziedzinie grafiki komputerowej oraz obliczeń naukowych, prezentujemy również przykładowe aplikacje, które korzystają z obliczeń naszej implementacji.

Pierwszym zastosowaniem algorytmu Fast Inverse Square Root było zaimplementowanie w grze komputerowej Quake III Arena. Algorytm ten znalazł zastosowanie w przyspieszaniu obliczeń związanych z oświetleniem i renderingiem graficznym. Stąd też nazwa tego algorytmu to Quake Fast Inverse Square Root algorithm. Działa on na podstawie magii bitowej (bitwise magic) oraz manipulacji liczbami zmiennoprzecinkowymi w reprezentacji binarnej. Podstawowy schemat działania algorytmu Fast Inverse Square Root można przedstawić w kilku krokach:

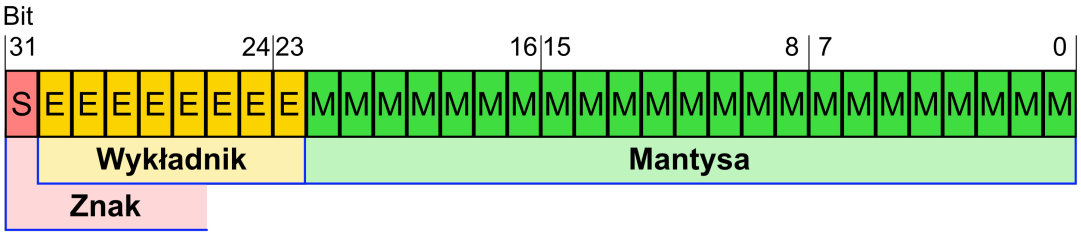
1. Wykorzystanie reprezentacji liczby zmiennoprzecinkowej i zapisanie jej ciąg bitów w zmiennej całkowitej. Tutaj jest wykorzystywany adres, gdzie przechowywana jest zmienna i kopiowana jest zawartość do drugiej zmiennej. Dzięki temu możemy pracować na znanej nam liczbie jak na danym ciągu bitów.

$$iDestination = *(long*)\ \&fSource$$



Rysunek 1: Caption

2. Zastosowanie magicznej liczby 0x5f3759df czyli pewnych operacji matematycznych i bitowych na tej reprezentacji, które mają na celu przybliżone obliczenie wartości odwrotności pierwiastka kwadratowego. Wykorzystywany jest standard IEEE754, formalnie dzielący zmienną na 3 części - znak, eksponente i mantyse.



Rysunek 2: Caption

Wprowadzając założenia pracy na logarytmach, obliczenie algorytmu staje się możliwe.

$$log(IEE754) = \frac{1}{2^{23}}(M + 2^{23} * E) + \mu - 127 \quad \begin{array}{l} M - mantysa \\ E - Wykladnik \\ \mu - stala \\ -127 - bias \end{array} \tag{1}$$

$$\frac{1}{\sqrt{y}} \; -> \; log\left(\frac{1}{\sqrt{y}}\right)$$

$$log\left(\frac{1}{\sqrt{y}}\right) \; -> \; log\left(y^{-\frac{1}{2}}\right) \; -> \; -\frac{1}{2}log\,(y)$$

Zakładając że rozwiązanie nasze to $log(\Gamma)$:

$$log(\Gamma) = -\frac{1}{2}log(y)$$

$$\frac{1}{2^{23}}(M + 2^{23} * E) + \mu - 127 = -\frac{1}{2}\left(\frac{1}{2^{23}}(M + 2^{23} * E) + \mu - 127\right)$$

$$\begin{aligned}(M_\Gamma + 2^{23} * E_\Gamma) &= \frac{3}{2}2^{23}(127 - \mu) - \frac{1}{2}\left(M_y + 2^{23} * E_y\right) \\ &= 0x5F3759DF - (iDestination >> 1)\end{aligned}$$

W ostatecznym rozrachunku stosowaliśmy logarytmy, które się znoszą, dzięki zastosowaniu ich po obu stronach równania, a dzielenie przez 2, jest wykonywane operacją sprzętową, czyli przesuwania bitów w daną stronę.

3. Ostateczne dostosowanie wyniku, aby uzyskać bardziej precyzyjną wartość odwrotności pierwiastka kwadratowego.

Algorytm wykorzystuje technikę zwana metodom Newtona-Raphsona do iteracyjnego obliczania przybliżonej wartości pierwiastka kwadratowego. Jednak kluczową innowacją algorytmu Fast Inverse Square Root jest wykorzystanie operacji bitowych, które pozwalają na przyspieszenie tego procesu.

Ważnym aspektem algorytmu jest również tzw. "magic number"(liczba magiczna), która jest używana w manipulacji bitowej i jest zależna od wartości liczby, dla której obliczamy odwrotność pierwiastka kwadratowego. To właśnie ta liczba magiczna pozwala na przyspieszenie obliczeń i uzyskanie przybliżonego wyniku.

W kolejnych sekcjach raportu szczegółowo omówimy proces implementacji algorytmu Fast Inverse Square Root na układzie FPGA oraz przedstawimy wyniki naszych badań i analizę efektywności implementacji.

1. Model behawioralny algorytmu

1.1. Implementacja modelu w języku C

Wykorzystano algorytm Quake opisany we wstępie raportu. Stworzono w tym celu funkcję *Q_rsqrt()*. Na wejściu otrzymuje dane typu *float*, a następnie implementuje opisany wcześniej algorytm. Fragment kodu dotyczący tej funkcji przedstawiono poniżej

```
int Q_rsqrt( float number ){
    long i;
    unsigned long r;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;
    i  = 0x5f3759df - ( i >> 1 );
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );
    r  = * ( long * ) &y;
    return r;
}
```

1.2. Testbench modelu behawioralnego w języku C

Testbench przyjmuje dane wprowadzane z konsoli przez użytkownika w postaci *float*. Następnie użytkownik dostaje informację zwrotną o wartości wyliczoną przez algortym Fast Inverse Square Root i przez wbudowaną funkcję *sqrt()* z biblioteki *math.h* oraz różnicę między tymi wynikami.

Poniżej umieszczono kod testujący:

```
int main(){
    while(1){
        float input;
        printf("Enter a value: ");
        scanf("%f", &input);

        if(input <= 0.0){
            printf("This value is not allowed\n");
        }
        else{
            float math_out  = 1.0 / sqrt(input);
            float fisr_out = ieee754_to_float(Q_rsqrt(input));
            float abs_val = fabs(fisr_out - math_out);
            printf("-------------------------------------\n");
            printf("Entered Value -> %.3f\n", input);
            printf("Algorithm Output -> %.8f\n", fisr_out);
            printf("Math.h Function -> %.8f\n", math_out);
            printf("Difference |FISR-MATH| -> %.8f\n",abs_val);
            printf("-------------------------------------\n");
        }
    }
}
```

W kodzie tym znajduje się również funkcja odpowiedzialna za konwersję liczby wyjściowej z algorytmu na wartość *float*. Jej działanie jest zgodne z opisem przedstawionym na Rys. 1. Poniżej umieszczono kod funkcji *ieee754_to_float()*

```
float ieee754_to_float(unsigned int value){
    return *((float*)&value);
}
```

1.3. Prezentacja wyników

Poniżej przedstawiono działanie kodu. Otrzymywane wyniki potwierdzają poprawne funkcjonowanie algorytmu Quake. Można przyjąć, że aktualnie dokładniejsze algorytmu posiadają funkcje wbudowane, jak np. sqrt(), jednakże algorytm Fast Inverse Square Root również działa z dużą precyzją oferując szybkość przetwarzania.

Kod jest zabezpieczony przed wprowadzaniem danych ujemnych oraz 0 (Rys. 1.1).

```
Enter a value: 0
This value is not allowed
Enter a value: -0.01
This value is not allowed
Enter a value: 1
-------------------------------------------
Entered Value -> 1.000
Algorithm Output -> 0.99830717
Math.h Function -> 1.00000000
Difference |FISR - MATH| -> 0.00169283
-------------------------------------------
Enter a value: 0.5
-------------------------------------------
Entered Value -> 0.500
Algorithm Output -> 1.41386008
Math.h Function -> 1.41421354
Difference |FISR - MATH| -> 0.00035346
-------------------------------------------
Enter a value: 5
-------------------------------------------
Entered Value -> 5.000
Algorithm Output -> 0.44714102
Math.h Function -> 0.44721359
Difference |FISR - MATH| -> 0.00007257
-------------------------------------------
Enter a value: 0.001
-------------------------------------------
Entered Value -> 0.001
Algorithm Output -> 31.58506393
Math.h Function -> 31.62277603
Difference |FISR - MATH| -> 0.03771210
-------------------------------------------
Enter a value: 1000
-------------------------------------------
Entered Value -> 1000.000
Algorithm Output -> 0.03156984
Math.h Function -> 0.03162277
Difference |FISR - MATH| -> 0.00005293
-------------------------------------------
```

Rysunek 1.1: Prezentacja działania kodu

2. Model syntezowalny potokowy algorytmu

2.1. Układ mnożący

2.2. Układ odejmujący

2.3. Implementacja kodu

2.4. Schemat blokowy

2.5. Testbench

3. AXI

3.1. Wrapper

4. Uruchomienie na sprzęcie

4.1. Test układu przez konsolę

5. Zastosowanie

5.1. Struktura blokowa

5.2. Komunikacja między PC, a Zedboard

5.3. Dostosowanie kodu na płytce Zedboard

5.4. Implementacja kodu z interakcją użytkownika

5.4.1. FigureMoveIn3D

5.4.2. VectorMoving

Spis tabel

Spis rysunków