Wykonały: Karolina Kopczak
Aleksandra Piechota

Kierunek: Elektronika i Telekomunikacja

Przedmiot: Systemy dedykowane
w układach programowalnych

CELE I ZAŁOŻENIA:

Szybka odwrotność pierwiastka kwadratowego:

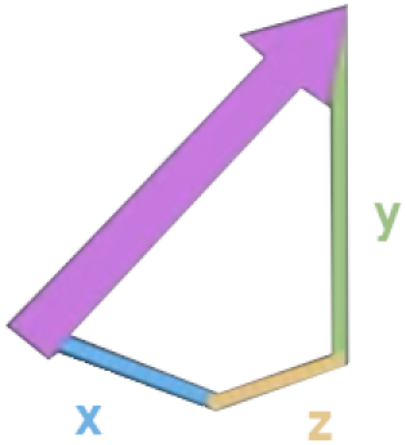
Jest to metoda obliczania $x^{-1/2}$, odnosząca się do przekształceń z 32-bitowej liczby zmiennoprzecinkowej w standardzie IEEE 754. Największą zaletą tego algorytmu jest uniknięcie kosztownych obliczeniowo operacji zmiennoprzecinkowych na korzyść operacji na liczbach całkowitych.

Cel:

Algorytm szybkiej odwrotności pierwiastka kwadratowego pomaga przyspieszyć obliczenia procesora.

Po co?

W celach normalizacji np. wektorów.



Długość wektora wyraża się wzorem $\sqrt{x^2+y^2+z^2}$. Takie równanie procesor policzy bardzo szybko. Jednakże jeżeli chcielibyśmy unormowaną formę wektora (żeby jego długość wynosiła 1), to współrzędne wektora również powinny być odpowiednio unormowane. W takim wypadku wartości każdego z nich wynosiłyby:

$$x_1 = x \cdot \frac{1}{\sqrt{x^2+y^2+z^2}} \qquad y_1 = y \cdot \frac{1}{\sqrt{x^2+y^2+z^2}} \qquad z_1 = z \cdot \frac{1}{\sqrt{x^2+y^2+z^2}}$$

A takie działanie nasz procesor wykona w znacznie dłuższym czasie. Stąd pomysł na ten algorytm - żeby przyspieszyć to działanie.

Kod

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

        x2 = number * 0.5F;
        y  = number;
        i  = * ( long * ) &y;                       // evil floating point bit level hacking
        i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
        y  = * ( float * ) &i;
        y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//      y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

        return y;
}
```

Krótkie wytłumaczenie skąd takie liczby

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

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

$$(\mathrm{M}_\Gamma + 2^{23} * \mathrm{E}_\Gamma) = \frac{3}{2}2^{23}(127 - \mu) - \frac{1}{2}(\mathrm{M}_y + 2^{23} * \mathrm{E}_y)$$

```
= 0x5f3759df - ( i >> 1 );
```

Materiały:
https://pl.wikipedia.org/wiki/Szybka_odwrotno%C5%9B%C4%87_pierwiastka_kwadratowego

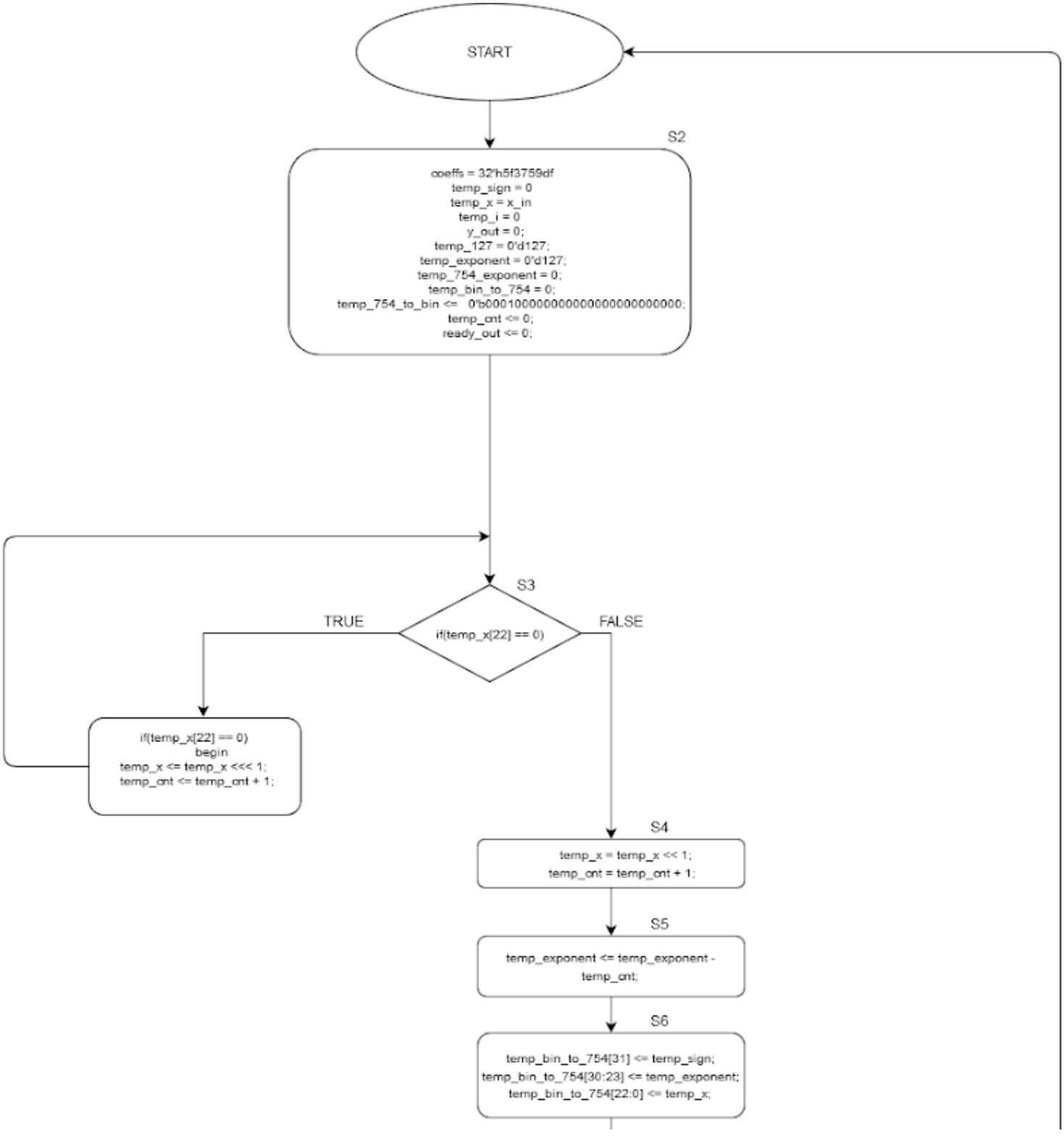
https://docs.google.com/presentation/d/10omR9Pbtn0QZo0_31juCz5SavMAyupZeR1-mI2m6Fcc/edit#slide=id.p

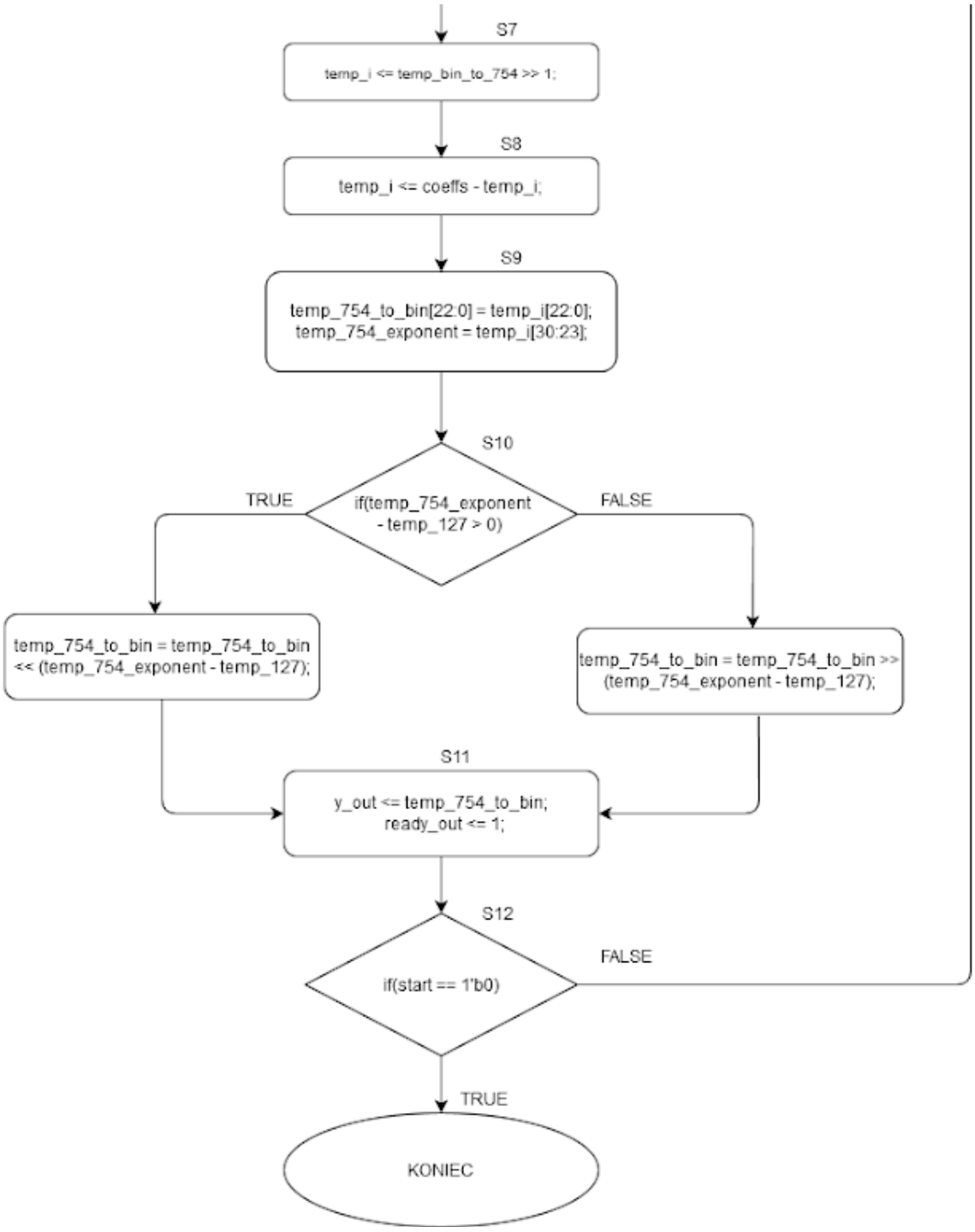
https://www.wykop.pl/link/5880113/algorytm-z-quake-iii-obliczanie-odwrotnosci-pierwiastka/

WYKONANIE:

Diagram stanów wykonanych w pliku rtl znajduje się poniżej.

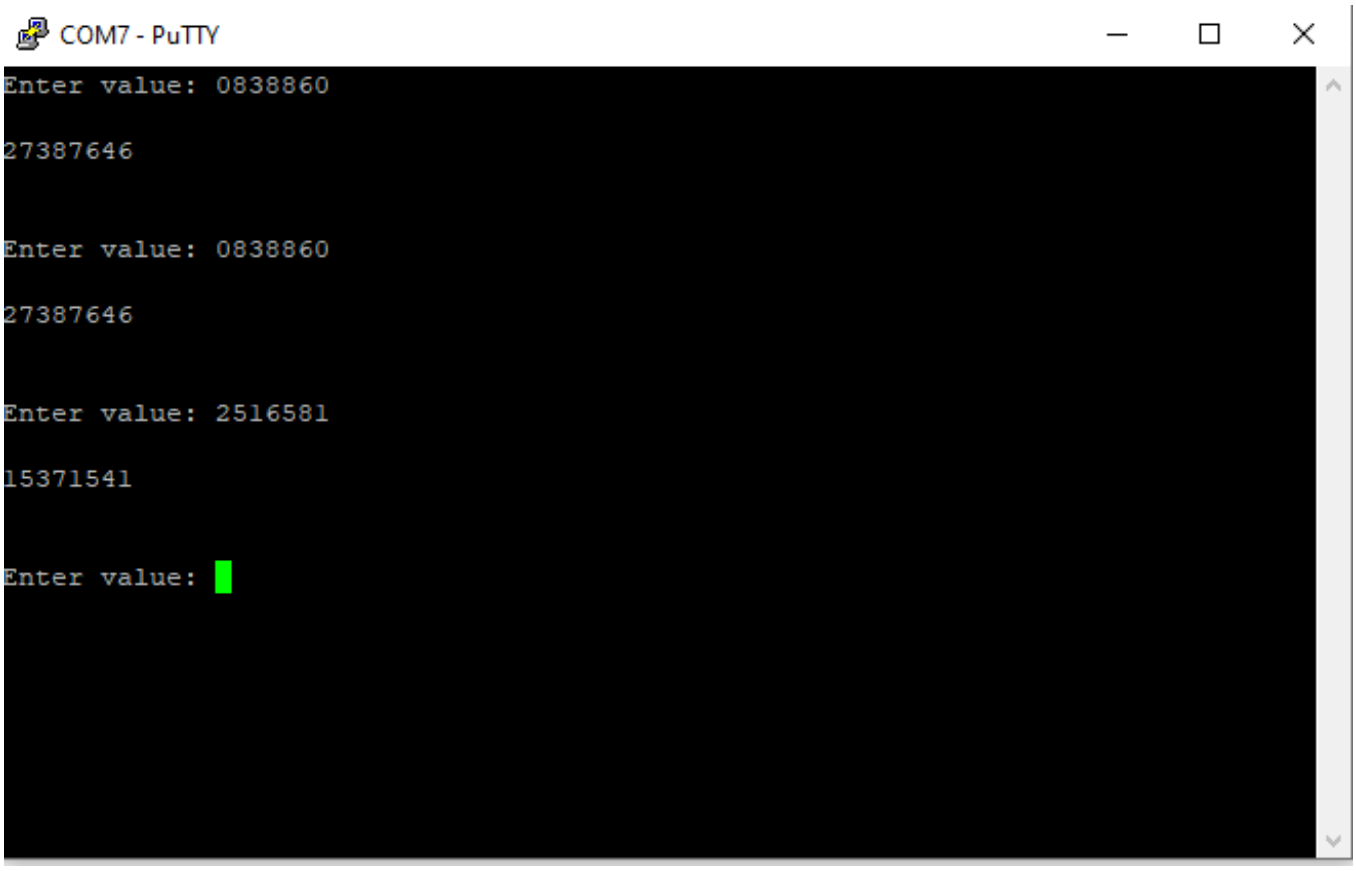
Na początku wpisywane są dane w naturalnym kodzie binarnym. Następuje następnie konwersja na standard IEEE 754 - w takim formacie następuje obliczenie szybkiej odwrotności pierwiastka. Wykorzystane zostały podstawowe zależności liczb binarnych, takich jak przesunięcie o 1 bit w prawo - dzielenie. W kodzie zostały umieszczone odpowiednie komentarze.

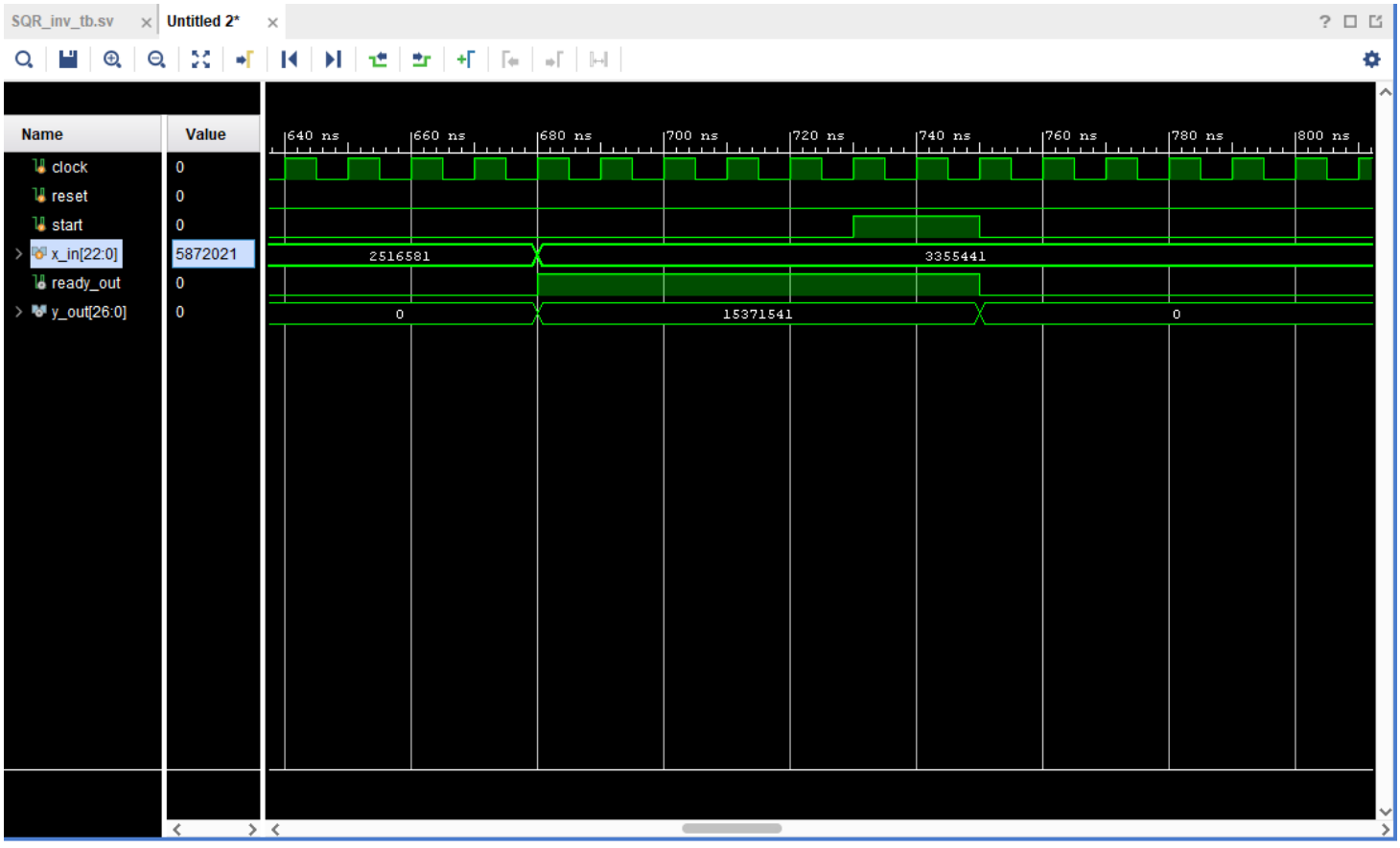




Po weryfikacji napisane zostały kody na płytkę oraz wgrane. Porównanie wartości z płytki i z symulacji znajdują się poniżej.

Wpisujemy dane w formacie unsigned Decimal - aby nie wpisywać ciągu 23 zer i jedynek. - A więc liczba 0.1 jest reprezentowana jako 838860, itd.





Po wpisaniu 2516581 powinniśmy otrzymać 15371541. Tak też widzimy w terminalu.

2516581 -> w systemie dziesiętnym odpowiada $\frac{2516581}{2^{23}} = 0.2999998331$

$\frac{1}{\sqrt{0.2999998331}} = 1.82574236621$

A więc otrzymana liczba w systemie decymalnym:

15371541 -> w systemie dziesiętnym odpowiada $\frac{15371541}{2^{23}} = 1.83243048191$