Sprawozdanie z projektu OiAK

Liczby typu Posit

Maciej Myśków 272794

Kacper Kostrzewa 272855

Data oddania

23.06.2024

Spis treści

Wstęp

Liczby typu Posit stanowią alternatywę dla konwencjonalnego zapisu liczb zmiennoprzecinkowych – IEEE – 754. Nowy format Posit ma wiele podobieństw do konwencjonalnego zapisu, co czyni go łatwym w zrozumieniu oraz implementacji, czy też migracji z IEEE – 754 na Posit. Jednakże standard ten (mimo wielu zalet względem IEEE – 754) nie jest wolny od wad i potencjalnych problemów.

W projekcie przeanalizowaliśmy wszelkie różnice oraz podobieństwa w implementacji nowego standardu oraz wady i zalety jego stosowania względem poprzednika. W kolejnym kroku pokazaliśmy na przykładzie jego implementację matematyczną, co pozwoliło sprawnie przejść do implementacji programowej, dokładniej mówiąc implementacji w języku C. W końcowej fazie projektu udało się zrealizować opis syntezy wysokopoziomowej HLS – za pomocą narzędzia Bambu HLS. oraz dokonać podstawowych benchmarków otrzymanej syntezy.

Do implementacji HLS użyto wcześniej wspomnianego narzędzia Bambu HLS. Narzędzie to dostarcza bardzo wielu możliwości, jednakże my skorzystaliśmy tylko z jego niektórych aspektów. Narzędzie to pozwala na translację napisanego kodu w języku C do języka opisu układów cyfrowych – jakim jest Verilog. Bambu HLS również umożliwia dokonywanie benchmarków oraz podsumowania syntezy, przez dostarczanie informacji o przebiegu translacji, zachowywaniu się układu czy też poszczególnych elementów cyfrowych z jakich jest zbudowany nasz układ z opisu HLS.

Cele projektu

Celem projektu było zbadanie i przeanalizowanie alternatywnego formatu zapisu liczb zmiennoprzecinkowych, jakim są liczby typu Posit, oraz porównanie ich z konwencjonalnym standardem IEEE 754. Projekt miał na celu:

1. Zrozumienie różnic i podobieństw między formatem Posit a standardem IEEE 754:

    - Analiza struktury formatu, w tym wprowadzenia pola Regime w liczbach Posit.
    - Badanie, w jaki sposób Posit umożliwia reprezentację liczb.

2. Implementacja matematyczna liczby typu Posit:

    - Zdefiniowanie wzoru matematycznego i kroków konwersji liczby binarnej w formacie Posit na liczbę dziesiętną zmiennoprzecinkową.
    - Weryfikacja poprawności obliczeń za pomocą arkusza kalkulacyjnego Excel i makr napisanych w języku VisualBasic.

3. Implementacja algorytmu konwersji w języku C:

    - Napisanie algorytmu konwertującego liczby Posit na format Float IEEE-754 w języku C.
    - Testowanie poprawności algorytmu za pomocą danych testowych i porównanie wyników z wyliczeniami w Excelu.

4. Synteza wysokopoziomowa za pomocą narzędzia Bambu HLS:

    - Przeprowadzenie syntezy wysokopoziomowej (HLS) kodu C do języka opisu sprzętowego Verilog.
    - Testowanie uzyskanego układu w języku Verilog pod kątem liczby użytych przerzutników oraz czasu wykonania konwersji.

5. Analiza wyników syntezy i testów:

    - Zbadanie wpływu długości pola wykładnika na liczbę przerzutników oraz czas wykonania konwersji.
    - Wyciągnięcie wniosków na temat efektywności i skalowalności formatu Posit.

Poprzez realizację tych celów, projekt miał na celu dostarczenie praktycznej wiedzy na temat nowego formatu Posit oraz jego potencjalnych zalet i wad w kontekście rzeczywistych implementacji sprzętowych i programowych. Ostatecznym celem było ocenienie, czy format Posit może być godnym następcą standardu IEEE 754 w przyszłości.

Różnice między Posit a IEEE-754

- W przeciwieństwie do standardu IEEE 754, w liczbach typu Posit wprowadzono dodatkowe pole typu Regime, które kończy się w momencie napotkanie pierwszego przeciwnego bitu. Służy do osiągnięcia większej dokładności wyniku
- W liczbach typu posit, zakres wartości jest zwykle większy niż w IEEE 754 dzięki innej strukturze reprezentacji.
- Liczby typu posit mogą być bardziej efektywne pod względem pamięci, ponieważ mogą reprezentować szeroki zakres wartości przy mniejszej liczbie bitów w niektórych przypadkach w porównaniu do standardu IEEE 754.
- Standard IEEE 754 jest szeroko stosowany i zaimplementowany w wielu platformach i systemach obliczeniowych. Z kolei liczby typu posit, choć 4 rozwijane i badane, mogą nie być jeszcze tak powszechnie dostępne i zaimplementowane w porównaniu do IEEE 754

Charakterystyka formatu Posit

Charakterystyka

Format Posit cechuje się specjalnymi polami, które wchodzą w jego skład. Aby możliwy był zapis liczby zmiennoprzecinkowej w tym formacie, wszystkie poniżej wymienione pola muszą być uwzględnione oraz poprawnie wyliczone (długość, wartość).

- Znak (s)
- Regime (r)
- Wykładnik (es)
- Mantysa (m/f)

Jeśli chodzi o pierwsze pole czyli znak, pole to ma zawsze długość jednego bitu i może przyjąć wartość 0 lub 1. Wartość 0 jest przyjmowana dla liczb dodatnich a 1 dla ujemnych.

Kolejne pole jest polem typu Regime, służy do określenia skali liczby poprzez zakodowanie potęgi bazy w sposób bardziej efektywny i elastyczny niż tradycyjny wykładnik. Mówiąc krótko, od tego pola zależy jak dokładną reprezentację liczby otrzymamy. Jest to pole, które pozwala elastycznie wpływać na precyzją zapisu. Pole to zaczyna się zaraz po znaku a kończy na dowolnie ustalonym bicie.

Pole wykładnika służy do określania przedziału skali liczbowej, w którym znajduje się reprezentowana liczba. Oznacza to tyle, że bardziej znaczące liczby mogą mieć więcej bitów przeznaczonych na wykładnik, a mniej znaczące liczby mogą mieć więcej bitów przeznaczonych na mantysę. Długość tego pola jest zależna od skali z jaką chcemy reprezentować liczbę zmiennoprzecinkową. Pole to zaczyna się w miejscu natrafienia na bit przeciwny, idąc od pola poprzedniego – Regime. Jednakże ostatnimi czasy przyjęło się, że długość tego pola zwykle ustawiona jest sztywno na dwa bity. Wynika to ze specyfikacji systemu liczbowego, który jest używany w technice cyfrowej – system o podstawie 2 (system binarny).

Ostatnim typem pola jest pole przeznaczone na mantysę. Jest to nic innego jak część ułamkowa naszej liczby (po przecinku). Mantysa jest kodowana na pozostałych bitach, które zostają po zakodowaniu znaku, regime oraz wykładnika.

Notacja

Liczba w formacie posit bardzo często jest zapisywane w charakterystycznym dla niej formacie. Forma ten wyróżnia długość całej liczby oraz długość wykładnika liczby: **<*n, es*>**. Dla przykładu liczbę zmiennoprzecinkową 16-bitową o rozmiarze wykładnika równym dwa, zapiszemy w ten sposób: **<*16,2*>**. Taką też notacją będziemy się posługiwać w dalszej części projektu.

Implementacja matematyczna

Wzór oraz jego elementy

Do wytworzenia reprezentacji liczy typu Posit, potrzebne są obliczone i zebrane wszystkie pola omawiane w punkcie wyżej. Następnie potrzebny jest wzór aby móc te pola zamienić na wynik – liczba zmiennoprzecinkowa. Jednakże, przed przystąpieniem do konwersji należy zbadać najstarszy bit liczby, jeśli jest od jedynką, to trzeba koniecznie uzupełnić wszystkie pozostałe bity do 2. Dlatego w tym punkcie zostanie przedstawiona pełna procedura konwersji liczby binarnej w formacie Posit na liczbę dziesiętną zmiennoprzecinkową, uwzględniając wszystkie wymagane kroki:

1. Jeżeli najstarszy bit liczby wynosi 1, to trzeba uzupełnić do 2 pozostałe bity
2. Następnie pobieramy wszystkie niezbędne pola do konwersji – co zostało wytłumaczone w punkcie powyżej
3. Wprowadzamy zmienną p, która będzie informować o ilości identycznych bitów w polu Regime
4. Wprowadzamy zmienną k, która będzie przyjmować określone wartości w zależności od p
5. Wprowadzamy zmienna u, która będzie potęgą dwójkową wykładnika
6. Zamieniamy wykładnik na postać dziesiętną
7. Korzystamy ze wzoru, który będzie omówiony poniżej

Wzór jest stosunkowo prosty do skojarzenia ze zmiennymi oraz podobny do wzoru dla konwencjonalnego zapisu IEEE-754. Wzór:

$$L = (-1)^s * u^k * 2^w * m$$

Gdzie:

$s -$ $znak$

p $-$ $ilość\ identycznych\ bitów\ w\ polu\ Regime$

$u = 2^{2^{(es)}}$

$k = \begin{cases} -p, gdy\ regime\ składa\ się\ z\ p\ zer \\ p\ -\ 1, gdy\ regime\ składa\ się\ z\ p\ jedynek \end{cases}$

w $-$ $wykładnik\ dziesiętnie$

$m - mantysa(część\ ułamkowa\ liczby)$

Praktyczny przykład

Dana liczba w formacie Posit <16,2> : 1001100000011101

Konieczne uzupełnienie do 2 (poza pierwszym bitem): 1110011111100011

Dane:

s = 1 (ujemna)

p = 2

k = 2 – 1= 1

e = 1

u = 2^{2^2} = 16

m = 1,971679688 (konwersja systemu binarnego na dziesiętny ułamkowo)

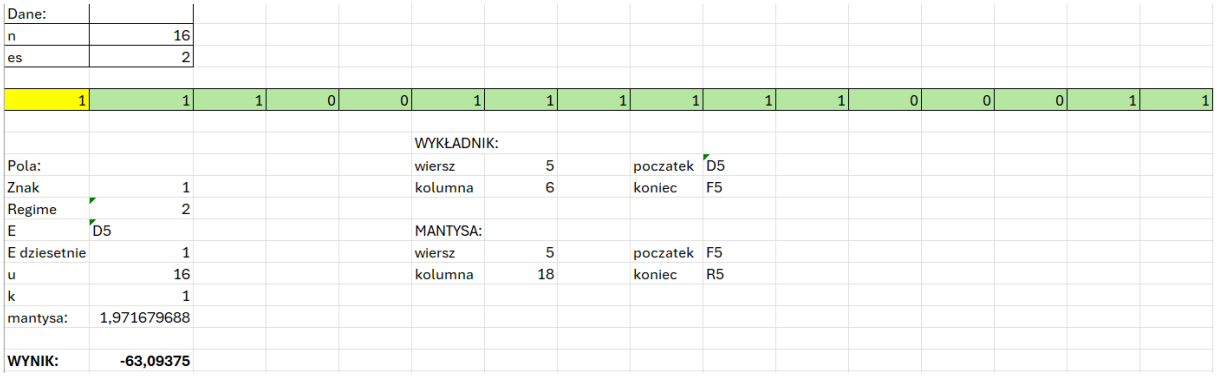
Zatem:

$$L = (-1)^1 * 16^1 * 2^1 * 1{,}971679688$$

$$L \approx -63{,}09375$$

Excel – weryfikacja poprawności obliczeń

Dla poprawności przeprowadzanych obliczeń został stworzony arkusz kalkulacyjny w oparciu o autorskie makra napisane w języku VisualBasic. Excel oraz jego kod zostanie zamieszony jako załącznik do niniejszego sprawozdania do odczytu i wglądu przez prowadzącego. Poniżej zostanie przedstawione działanie Excela dla wcześniejszego przykładu jako zrzut ekranu.



Dane:															
n	16														
es	2														
1	1	1	0	0	1	1	1	1	1	1	0	0	0	1	1
					WYKŁADNIK:										
Pola:					wiersz	5		poczatek	D5						
Znak	1				kolumna	6		koniec	F5						
Regime	2														
E	D5				MANTYSA:										
E dziesetnie	1				wiersz	5		poczatek	F5						
u	16				kolumna	18		koniec	R5						
k	1														
mantysa:	1,971679688														
WYNIK:	**-63,09375**														

Jak widać na powyższym zrzucie ekranu, dla naszej reprezentacji posit: 1001100000011101 zostały wykonane procedury i obliczenia, które przeprowadzono wcześniej. Wynik również się zgadza z otrzymanym – liczonym ręcznie. Należy pamiętać, aby do powyższego Excela podawać wartości już przekonwertowane na U2, jeśli znak liczby tego wymaga.

Implementacja algorytmu w języku C

Po poprawnej implementacji matematycznej oraz zrozumieniu całego procesu konwersji, kolejnym naturalnym krokiem było napisanie przedstawionego (w poprzednik punktach) algorytmu w języku programowania. Wybór padł na język C, przede wszystkim dlatego, że narzędzie Bambu HLS (które będzie omówione w następnych punktach) obsługuje wyłącznie język C/C++.

Program realizuje konwersje ze standardu Posit <16,es> na Float IEEE-754, gdzie długość wykładnika - es może być dowolnie zmieniana w kodzie, jednakże wyjściowym formatem jest format najbardziej spopularyzowany tj. es = 2.

Kod został podzielony na dwie główne procedury: wyodrębnianie pól i konstrukcja floata ze wcześniej wyodrębnionych pół z zachowaniem warunku konwersji na U2.

Kod napisany jest proceduralnie - w początkowej fazie pisany był obiektowo w C++, jednakże celu uniknięcia błędów przy próbie syntezy wysokopoziomowej w narzędziu Bambu HLS, zdecydowano się na podejście proceduralne w C.

Do testowania poprawności konwersji wykorzystano wektor danych testowych (liczby binarne 16-bitowe) wybrane w sposób przemyślany. Najpierw wybrano kilkanaście wariantów z mantysą, gdzie ilość jedynek jest relatywnie spora, później wybrano maksymalnego oraz minimalnego reprezentanta liczby typu Posit, a na końcu wybrano kilkanaście losowych instancji wygenerowanych przez generator napisany w osobnym programie – oczywiście wartości te można zmieniać również ręcznie na potrzeby swoich testów . Natomiast samą poprawność konwersji testowano za pomocą wcześniej omawianego Excela.

Został stworzony również kod do pomiaru czasów w zależności od długości pola wykładnika (es). Kod ten różni się nieznacznie od powyżej opisanego. Jedyną różnicą jest zastosowanie biblioteki **time** oraz lekka modyfikacja wywołania funkcji konwertującej na potrzeby dokonywania pomiarów.

Obydwie wersje kodu źródłowego zostaną załączone do niniejszego sprawozdania do wglądu prowadzącego. Natomiast listingi kodów zostaną umieszczone w ostatniej sekcji sprawozdania.

Synteza wysokopoziomowa – HLS i testy

Czym jest HLS i co otrzymano

Synteza wysokopoziomowa jest to zautomatyzowane (za pomocą odpowiedniego narzędzia) przekształcanie kodu napisanego w języku wysokopoziomowym (np. C++) na język opisu sprzętowego (HDL). Reprezentantem HDL, jest język sprzętowy Verilog i to właśnie w nim powstał kod naszego algorytmu po skonwertowaniu oraz w tym języku dokonywane były testy układu.

Jako narzędzie do wspomnianej konwersji użyto HLS Bambu. Jest to potężne narzędzie, które posiada wiele opcji. Jednakże do projektu zostały wybrane tylko wybrane funkcjonalności takie jak konwersja czy benchtesty.

Samo narzędzie jest dosyć proste w obsłudze. Potrzebujemy dostarczyć Bambu HLS plik, który zawiera opis naszego algorytmu (w języku C). Następnie potrzebujemy nasz kod skompilować i przetłumaczyć na język Verilog. Wykorzystano do tego polecenie:

$ bambu --std=c99 --top-fname=posit_na_float posit_16_1.c

Polecenie to uruchamia procez HLS, który wskazuje na funkcje, która ma być zsyntezowana. U nas w projekcie jest to funkcja *posit_na_float*, która zawiera przekazane argumenty (n, es oraz liczba posit) i odwołania do procedur pomocniczych.

Jako wynik polecenia otrzymano kompletny opis HDL Verilog oraz podstawowy listing z informacjami o skonwertowanym układzie. Z informacji tych możemy odczytać między innymi ilość i rodzaj komponentów potrzebnych do zbudowania zsyntezowanego układu, proponowane częstotliwości i wiele innych przydatnych informacji do syntezy na rzeczywistych układach.

Wyniki syntezy (kody Verilog oraz listingi) zostaną załączone do niniejszego sprawozdania do wglądu przez prowadzącego.

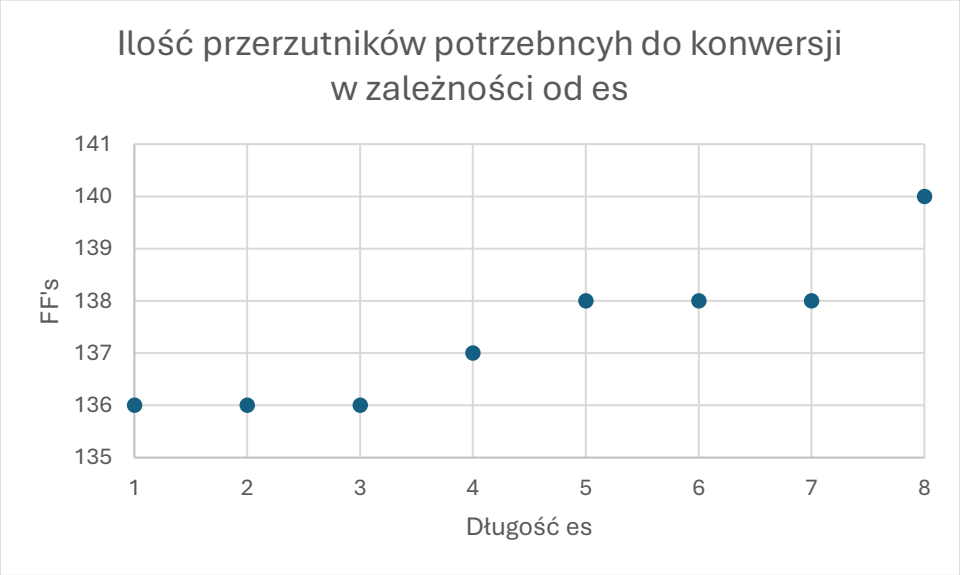
Co testowano i w jaki sposób

W projekcie zostały poddane testom dwa aspekty. Pierwszy z nich to testowanie ilość przerzutników (FF's) potrzebnych do zbudowania rzeczywistego układu konwertującego liczbę posit na floata w zależności od długości wykładnika (es) z przedziału [1, 8] dla stałej długości liczby (n = 16). Natomiast drugim testem był pomiar czasu konwersji w nawiązaniu do powyższej zależności.

Przy testowaniu syntezy układu dokonano w sumie ośmiu testów (dla każdej długości es). Każdy z testów był wykonywany osobno na tym samym wektorze danych testowych – był tylko zmieniany rozmiar wykładnika. Dla każdego testu został również wygenerowany odpowiedni plik Verilog. Końcowe rezultaty testów zebrano do tabeli oraz sporządzono odpowiedni wykres, który obrazuje ilość potrzebnych przerzutników w zależności od es. Plik Excela zostanie również dołączony do sprawozdania.

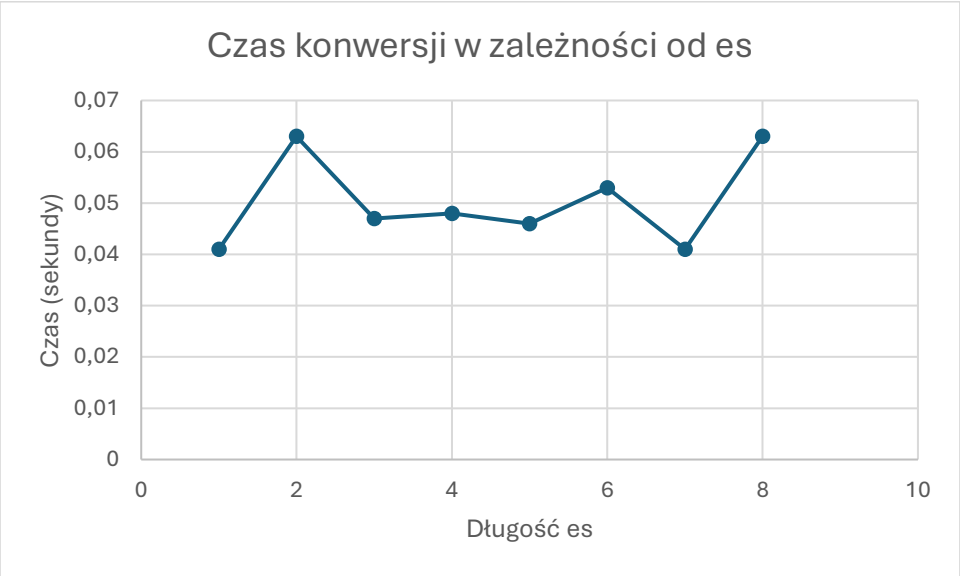
Dla testu czasów konwersji wykorzystano bibliotekę języka C time, dzięki której w odpowiedniej pętli mierzono czas konwersji dla zmieniającej się dynamicznie wartości es. Umożliwiło to otrzymanie miarodajnych wyników w relatywnie prosty sposób dla testowania złożoności czasowej algorytmu.

Wyniki eksperymentu dla HLS



WYKRES 1 – ILOŚĆ PRZERZUTNIKÓW WYKORZYSTANYCH DO KONWERSJI W ZALEŻNOŚCI OD DŁUGOŚCI ES

Wyniki eksperymentu dla czasów konwersji



WYKRES 2 – CZASY KONWERSJI W ZALEŻNOŚCI OD DŁUGOŚCI ES

Wyniki eksperymentu - skumulowane

Dla poprawnego wyciągnięcia wniosków końcowych, które pojawią się w następnym punkcie, skonstruowano wykres skumulowany, na którym nałożono ilość przerzutników w zależności od es oraz czasy konwersji dla tej samej zależności.

WYKRES 3 – SKUMULOWANY WYKRES ILOŚCI PRZERZUTNIKÓW I CZASÓW KONWERSJI

Wnioski końcowe

Po dokonaniu syntezy HLS, benchmarków syntezy oraz czasów wykonania, możemy wyciągnąć wnioski z otrzymanych danych wynikowych. Wnioski są mocno subiektywne na podstawie pracy z projektem i doświadczonych rezultatów jak i porażek wynikających na przykład z implementacji.

Do wyciągnięcia wniosków niezbędne było dogłębne zapoznanie się z wykresami w poprzednim punkcie oraz charakterystyką problematyki projektu. Punkt ten pomoże również poznać bliżej szczegółową charakterystykę badanego alternatywnego formatu zapisu liczb zmiennoprzecinkowych – formatu Posit.

Stawiana teza

Na podstawie przeprowadzonych badań postawiliśmy tezę iż **długość pola wykładnika nie wpływa znacząco na zasoby i czas**. Aby potwierdzić lub obalić tezę przenalizowaliśmy poniżej wszystkie przeprowadzone pomiary o otrzymane wyniki.

Ilość potrzebnych przerzutników do syntezy układu

Na podstawie wykresu [1] możemy stwierdzić, że ilość przerzutników w zależności od długości pola wykładnika nie wzrasta drastycznie a w pewnych momentach jest nawet stała. Stałą ilość przerzutników dostajemy kolejno dla wartości es z przedziału [1, 3] oraz [5, 7], która wynosi odpowiednio 136 i 138. Może z tego wynikać wniosek, iż poziom skomplikowania układu rozrasta się w miarę potrojenia się długości pola wykładnika.

Natomiast generalny przebieg (monotoniczność) wykresu wskazuje na dosyć łagodny przyrost FF's względem es. Sugeruje to, iż tak zbudowany układ z rzeczywistych komponentów może być łatwy i przewidywalny w rozbudowie, co może czynić go łatwo skalowalnym oraz tanim jeśli chodzi o proces produkcji.

Można również zredukować problem do minimum i użyć stwierdzenia, że zwiększanie długości wykładnika nie wpływa znacząco na zapotrzebowanie przerzutników, co również jest na plus jeśli chodzi o implementacje układów konwertujących czy też arytmetycznych bazujących na formacie Posit.

Czasy wykonania a długość wykładnika

Charakterystyka przebiegi wykresu [2] również świadczy o słabej dynamice zmian względem długości pola wykładnika. Przy większości prób dla es przebieg ten jest blisko stałej wartości.

Redukując tutaj problematykę również możemy dojść do podobnego wniosku co w poprzednik teście, mianowicie skalowanie długości pola wykładnika nie wpływa znacząco na złożoność czasową wykonywania konwersji.

Według nas jest to pewne potwierdzenie powyższego testu z przerzutnikami. Skoro poziom skomplikowania tworzonego układu konwertującego wzrasta nieznacząco względem modyfikacji długości pola wykładnika, to i złożoność czasowa wykonania się algorytmu powinna cechować się podobną tendencją w dziedzinie czasu. Jednakże jest to nasze prywatne i subiektywne (do badanej problematyki) spojrzenie na wyniki testów.

Wniosek końcowy

Na potrzeby potwierdzenia wyżej postawionej tezy stworzono również wykres skumulowany, na który nałożyliśmy czas oraz ilość przerzutników w zależności od długości pola wykładnika. Obrazuje to wykres [3]. Widzimy, że stosunek ilości przerzutników do czasu wykonania cechuje się bardzo niskim zróżnicowaniem w odniesieniu do es.

Subiektywnie uważamy, iż wykres [3] ostatecznie potwierdza stawianą tezę: długość pola wykładnika nie wpływa znacząco na zasoby i czas”. Na tym wykresie widać najdokładniej zestawiane ze sobą dwa testy oraz charakterystykę przebiegu, która w przybliżeniu jest stała. Co uważamy za potwierdzenie stawianej tezy.

Powyżej przeprowadzone badania, testy oraz ich wyniki jednoznacznie przemawiają za szerszym stosowaniem nowego standardu dla liczb zmiennoprzecinkowych – Posit. Cechuje się on wyższą precyją – na którą możemy nawet wpływać w trakcie konwersji za pomocą skalowania długości pola wykładnika a przy tym cechuje się relatywnie niskimi kosztami syntezy w kontekście komponentów oraz złożoności czasowej. Naszym zdaniem w niedalekiej przyszłości jest to godny następca formatu IEEE-754.

Kody źródłowe

KOD 1. KONWERSJA POSIT NA FLOAT

```
#include <stdio.h>

#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int N = 16;
int ES = 2; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
        0b1100101111011111,
        0b0101111011100111,
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
```

```
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int test = 0; test < ilosc_testow; ++test) {
        uint16_t posit_wartosc = testy[test];

        float ieee_float = posit_na_float(posit_wartosc); //skonwertowany
posit
        printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
        for (int i = N - 1; i >= 0; i--) {
            printf("%d", (posit_wartosc >> i) & 1);
        }
        printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
```

```
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime - wzor sprawozdania
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; // obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) - 1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 - mantysa_dlugosc);

    // do debugowania
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik, *mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) | (mantysa & ((1 << 23) - 1));
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
    bool znak = (x >> (N - 1)) & 1;

    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 : wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);

        int bias = BIAS;

        // policz wykladnik przesuniety o bias
```

```
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);

        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

KOD 2. KONWERSJA NA FLOAT – POMIAR CZASÓW WYKONANIA

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>
#include <time.h>

int N = 16;
int ES = 8; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
        0b1100101111011111,
        0b0101111011100111,
```

```
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int es = 1; es <= 8; ++es) {
        ES = es;
        clock_t start_time = clock();

        for (int test = 0; test < ilosc_testow; ++test) {
            uint16_t posit_wartosc = testy[test];
            float ieee_float = posit_na_float(posit_wartosc);
//skonwertowany posit
            printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
            for (int i = N - 1; i >= 0; i--) {
                printf("%d", (posit_wartosc >> i) & 1);
            }
```

```
            printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
        }

        clock_t end_time = clock();
        double elapsed_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;
        printf("Czas wykonania dla ES=%d: %.6f sekund\n", ES,
elapsed_time);
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime -
wzor sprawozdania
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; //
obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) -
1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 -
mantysa_dlugosc);

    // do debugowania
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik,
*mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) |
(mantysa & ((1 << 23) - 1));
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
```

```
    bool znak = (x >> (N - 1)) & 1;

    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 :
wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);

        int bias = BIAS;

        // policz wykladnik przesuniety o bias
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);

        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

KOD 3. KOD C DO BAMBU HLS DLA ES=1

```c
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int N = 16;
int ES = 1; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
        0b1100101111011111,
        0b0101111011100111,
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
```

```
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int test = 0; test < ilosc_testow; ++test) {
        uint16_t posit_wartosc = testy[test];

        float ieee_float = posit_na_float(posit_wartosc); //skonwertowany
posit
        printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
        for (int i = N - 1; i >= 0; i--) {
            printf("%d", (posit_wartosc >> i) & 1);
        }
        printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime -
wzor sprawozdania
```

```
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; //
obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) -
1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 -
mantysa_dlugosc);

    // do debugowania
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik,
*mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) |
(mantysa & ((1 << 23) - 1));
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
    bool znak = (x >> (N - 1)) & 1;

    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 :
wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);

        int bias = BIAS;

        // policz wykladnik przesuniety o bias
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);
```

```
        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

KOD 3. KOD C DO BAMBU HLS DLA ES=2

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int N = 16;
int ES = 2; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
        0b1100101111011111,
        0b0101111011100111,
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
```

```
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int test = 0; test < ilosc_testow; ++test) {
        uint16_t posit_wartosc = testy[test];

        float ieee_float = posit_na_float(posit_wartosc); //skonwertowany
posit
        printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
        for (int i = N - 1; i >= 0; i--) {
            printf("%d", (posit_wartosc >> i) & 1);
        }
        printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
```

```
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime -
wzor sprawozdania
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; //
obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) -
1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 -
mantysa_dlugosc);

    // do debugowania
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik,
*mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) |
(mantysa & ((1 << 23) - 1));
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
    bool znak = (x >> (N - 1)) & 1;

    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 :
wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);
```

```
        int bias = BIAS;

        // policz wykladnik przesuniety o bias
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);

        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

KOD 3. KOD C DO BAMBU HLS DLA ES=3

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int N = 16;
int ES = 3; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
```

```
        0b1100101111011111,
        0b0101111011100111,
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int test = 0; test < ilosc_testow; ++test) {
        uint16_t posit_wartosc = testy[test];

        float ieee_float = posit_na_float(posit_wartosc); //skonwertowany
posit
        printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
        for (int i = N - 1; i >= 0; i--) {
            printf("%d", (posit_wartosc >> i) & 1);
        }
        printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
```

```
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime -
wzor sprawozdania
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; //
obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) -
1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 -
mantysa_dlugosc);

    // do debugowania
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik,
*mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) |
(mantysa & ((1 << 23) - 1));
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
    bool znak = (x >> (N - 1)) & 1;

    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
```

```
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 :
wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);

        int bias = BIAS;

        // policz wykladnik przesuniety o bias
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);

        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

KOD 3. KOD C DO BAMBU HLS DLA ES=4

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int N = 16;
int ES = 4; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
```

```
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
        0b1100101111011111,
        0b0101111011100111,
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int test = 0; test < ilosc_testow; ++test) {
        uint16_t posit_wartosc = testy[test];
```

```
        float ieee_float = posit_na_float(posit_wartosc); //skonwertowany
posit
        printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
        for (int i = N - 1; i >= 0; i--) {
            printf("%d", (posit_wartosc >> i) & 1);
        }
        printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime -
wzor sprawozdania
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; //
obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) -
1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 -
mantysa_dlugosc);

    // do debugowania
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik,
*mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) |
(mantysa & ((1 << 23) - 1));
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
    bool znak = (x >> (N - 1)) & 1;
```

```
    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 :
wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);

        int bias = BIAS;

        // policz wykladnik przesuniety o bias
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);

        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

KOD 3. KOD C DO BAMBU HLS DLA ES=5

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int N = 16;
int ES = 5; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
```

```
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
        0b1100101111011111,
        0b0101111011100111,
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
```

```
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int test = 0; test < ilosc_testow; ++test) {
        uint16_t posit_wartosc = testy[test];

        float ieee_float = posit_na_float(posit_wartosc); //skonwertowany
posit
        printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
        for (int i = N - 1; i >= 0; i--) {
            printf("%d", (posit_wartosc >> i) & 1);
        }
        printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime -
wzor sprawozdania
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; //
obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) -
1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 -
mantysa_dlugosc);

    // do debugowania
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik,
*mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) |
(mantysa & ((1 << 23) - 1));
```

```
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
    bool znak = (x >> (N - 1)) & 1;

    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 :
wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);

        int bias = BIAS;

        // policz wykladnik przesuniety o bias
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);

        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

KOD 3. KOD C DO BAMBU HLS DLA ES=6

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int N = 16;
int ES = 6; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)
```

```
void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
        0b1100101111011111,
        0b0101111011100111,
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
```

```
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int test = 0; test < ilosc_testow; ++test) {
        uint16_t posit_wartosc = testy[test];

        float ieee_float = posit_na_float(posit_wartosc); //skonwertowany
posit
        printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
        for (int i = N - 1; i >= 0; i--) {
            printf("%d", (posit_wartosc >> i) & 1);
        }
        printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime -
wzor sprawozdania
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; //
obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) -
1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 -
mantysa_dlugosc);

    // do debugowania
```

```
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik,
*mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) |
(mantysa & ((1 << 23) - 1));
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
    bool znak = (x >> (N - 1)) & 1;

    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 :
wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);

        int bias = BIAS;

        // policz wykladnik przesuniety o bias
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);

        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

KOD 3. KOD C DO BAMBU HLS DLA ES=7

```c
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int N = 16;
int ES = 7; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
        0b1100101111011111,
        0b0101111011100111,
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
```

```
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int test = 0; test < ilosc_testow; ++test) {
        uint16_t posit_wartosc = testy[test];

        float ieee_float = posit_na_float(posit_wartosc); //skonwertowany
posit
        printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
        for (int i = N - 1; i >= 0; i--) {
            printf("%d", (posit_wartosc >> i) & 1);
        }
        printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime -
wzor sprawozdania
```

```
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; //
obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) -
1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 -
mantysa_dlugosc);

    // do debugowania
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik,
*mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) |
(mantysa & ((1 << 23) - 1));
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
    bool znak = (x >> (N - 1)) & 1;

    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 :
wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);

        int bias = BIAS;

        // policz wykladnik przesuniety o bias
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);
```

```
        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

KOD 3. KOD C DO BAMBU HLS DLA ES=8

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

int N = 16;
int ES = 8; // es bedzie zmieniany w kodzie (od 1 do 8) przy benchtestach
#define BIAS ((1 << (8 - 1)) - 1)

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa);
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa);
float posit_na_float(uint16_t x);

int main() {
    // Przykladowe wartoscii posita (wybrane recznie arbitralnie do tesow)
    uint16_t testy[] = {
        0b0100011111010101, // mantysa z większą ilością jedynek
        0b0100011000110101,
        0b0011100101001011,
        0b1011011110110101,
        0b1100111110010010,
        0b0111111000010010,
        0b0100100111110011,
        0b1101011010110100,
        0b1010100101100101,
        0b1111110111000100,
        0b0011111011011100,
        0b1000111000111000,
        0b1010011100011111,
        0b1111011011011000,
        0b0111111101101001,
        0b0001011110011110,
        0b1100101111011111,
        0b0101111011100111,
        0b1111111111111111, // maks wartość posit
        0b0000000000000001, // min wartość posit
        // inne wartocii posita (wybrane losowo za pomoca generatora w
pythonie) - sprawozdanie
        0b0110010101110000,
        0b1010110100111011,
```

```
        0b0011100111111110,
        0b0111001101101011,
        0b1001001110011111,
        0b0101111100101101,
        0b1010101011110100,
        0b1110001110110010,
        0b0011111100010101,
        0b0111110010010011,
        0b1101010101001110,
        0b1000010011000111,
        0b0101101111000101,
        0b0110111111101001,
        0b1111011001000110,
        0b0010101010101010,
        0b0110010110011001,
        0b1011101101000110,
        0b1110101001011101,
        0b1100111111110100,
        0b0100100100100100,
        0b0000000010000011,
        0b0101101100000101,
        0b1101000100100111,
        0b1111011011011110,
        0b1110000011111111,
        0b0100110010101101,
        0b0010100111101011,
        0b1010101011011011,
        0b0010011110011010,
    };

    int ilosc_testow = sizeof(testy) / sizeof(uint16_t);

    for (int test = 0; test < ilosc_testow; ++test) {
        uint16_t posit_wartosc = testy[test];

        float ieee_float = posit_na_float(posit_wartosc); //skonwertowany
posit
        printf("Test %d: Posit (es=%d): 0b", test + 1, ES);
        for (int i = N - 1; i >= 0; i--) {
            printf("%d", (posit_wartosc >> i) & 1);
        }
        printf(", IEEE 754 Float: %.10f\n", ieee_float); // %.10f dla
wiekszej precyzji - wymaga usprawnienia
    }

    return 0;
}

void wyodrebnij_pola(uint16_t wartosc, int *regime, int *wykladnik, int
*mantysa) {
    int k = 0;
```

```
    // szukamy dlugosci pola regime - do momentu natrafienia na przeciwny
bit
    while (k < (N - 1) && ((wartosc >> (N - 2 - k)) & 1)) {
        k++;
    }

    *regime = (k > 0 ? k - 1 : -k); // decydujemy jakie bedzie regime -
wzor sprawozdania
    int regime_dlugosc = k + 1;
    int wykladnik_dlugosc = ES;
    int mantysa_dlugosc = N - 1 - regime_dlugosc - wykladnik_dlugosc; //
obliczamy dl mantysy

    *wykladnik = (wartosc >> mantysa_dlugosc) & ((1 << wykladnik_dlugosc) -
1);
    *mantysa = (wartosc & ((1 << mantysa_dlugosc) - 1)) << (23 -
mantysa_dlugosc);

    // do debugowania
    // printf("Regime: %d, ES: %d, Mantysa: 0x%X\n", *regime, *wykladnik,
*mantysa);
}

// finalna konstrukcja floata z wyodrebnionych pol
float stworz_float_ieee(bool znak, int bias_wykladnik, int mantysa) {
    uint32_t ieee_int = (znak << 31) | ((bias_wykladnik & 0xFF) << 23) |
(mantysa & ((1 << 23) - 1));
    float ieee_float;
    memcpy(&ieee_float, &ieee_int, sizeof(ieee_float));
    return ieee_float;
}

float posit_na_float(uint16_t x) {
    // pobierz znak
    bool znak = (x >> (N - 1)) & 1;

    // pobierz reszte bitow - bez znaku
    uint16_t wartosc = x & ((1 << (N - 1)) - 1);

    // dopelnienie
    if (wartosc == 0) {
        return znak ? -0.0f : 0.0f;
    } else {
        // wartosc bezwzgledna
        uint16_t abs_wartosc = znak ? ((1 << (N - 1)) - 1) & ~wartosc + 1 :
wartosc;

        // wyodrebnianione pola
        int regime, wykladnik, mantysa;
        wyodrebnij_pola(abs_wartosc, &regime, &wykladnik, &mantysa);
```

```
        int bias = BIAS;

        // policz wykladnik przesuniety o bias
        int bias_wykladnik = (regime * (1 << ES)) + wykladnik + bias;

        // do  debugowania
        // printf("przesuniety wykladnik: %d\n", bias_wykladnik);

        // tworzenie floata
        float ieee_float = stworz_float_ieee(znak, bias_wykladnik,
mantysa);
        return ieee_float;
    }
}
```

Wyjścia Bambu HLS

OUTPUT 1. OUTPUT BAMBU HLS DLA ES = 1

```
**************************************************************************
*****
                    ____                    _
                   |  __ )  __ _ _ __ ___  | |_   _   _
                   |  _ \ / _` | '_ ` _ \| '_ \| | | |
                   | |_) | (_| | | | | | | |_) | |_| |
                   |____/ \__,_|_| |_| |_|_.__/ \__,_|

**************************************************************************
*****
                          High-Level Synthesis Tool

                          Politecnico di Milano - DEIB
                           System Architectures Group
**************************************************************************
*****
                 Copyright (C) 2004-2020 Politecnico di Milano
Version: PandA 0.9.7-dev - Revision
151822f6eb6b28b68ef7cde4c7c3c0add185ed9d-panda-0.9.7-dev

Target technology = FPGA

  Functions to be synthesized:
    posit_na_float


  Memory allocation information:
    BRAM bitsize: 8
    Spec may not exploit DATA bus width
    All the data have a known address
    Internal data is not externally accessible
    DATA bus bitsize: 8
    ADDRESS bus bitsize: 5
    SIZE bus bitsize: 4
    ALL pointers have been resolved
    Internally allocated memory (no private memories): 0
    Internally allocated memory: 0
  Time to perform memory allocation: 0.00 seconds


  Module allocation information for function posit_na_float:
    Number of complex operations: 0
    Number of complex operations: 0
  Time to perform module allocation: 0.07 seconds
```

```
  Scheduling Information of function posit_na_float:
    Number of control steps: 8
    Minimum slack: 4.1631999999999998
    Estimated max frequency (MHz): 171.3267543859649
  Time to perform scheduling: 0.01 seconds


  State Transition Graph Information of function posit_na_float:
    Number of states: 6
    Done port is registered
  Time to perform creation of STG: 0.00 seconds


  Easy binding information for function posit_na_float:
    Bound operations:52/65
  Time to perform easy binding: 0.00 seconds


  Storage Value Information of function posit_na_float:
    Number of storage values inserted: 10
  Time to compute storage value information: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Module binding information for function posit_na_float:
    Number of modules instantiated: 64
    Number of possible conflicts for possible false paths introduced by
resource sharing: 0
    Estimated resources area (no Muxes and address logic): 539
    Estimated area of MUX21: 20
    Total estimated area: 559
    Estimated number of DSPs: 0
    Slack computed in 0.00 seconds
    False-loop computation completed in 0.00 seconds
    Weight computation completed in 0.00 seconds
    Clique covering computation completed in 0.00 seconds
  Time to perform module binding: 0.00 seconds
```

```
  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Connection Binding Information for function posit_na_float:
    Number of allocated multiplexers (2-to-1 equivalent): 7
  Time to perform interconnection binding: 0.00 seconds

  Total number of flip-flops in function posit_na_float: 136
```

OUTPUT 2. OUTPUT BAMBU HLS DLA ES = 2

```
                    ____                  _
                   / __ )  __ _ _ __ ___ | |_   _   _
                   |  _ \ / _` | '_ ` _ \| '_ \| | | |
                   | |_) | (_| | | | | | | |_) | |_| |
                   |____/ \__,_|_| |_| |_|_.__/ \__,_|

---

                         High-Level Synthesis Tool

                         Politecnico di Milano - DEIB
                          System Architectures Group

---

                Copyright (C) 2004-2020 Politecnico di Milano

Version: PandA 0.9.7-dev - Revision
151822f6eb6b28b68ef7cde4c7c3c0add185ed9d-panda-0.9.7-dev

Target technology = FPGA

Functions to be synthesized:
posit_na_float

Memory allocation information:
BRAM bitsize: 8
Spec may not exploit DATA bus width
All the data have a known address
Internal data is not externally accessible
DATA bus bitsize: 8
ADDRESS bus bitsize: 5
SIZE bus bitsize: 4
ALL pointers have been resolved
```

```
Internally allocated memory (no private memories): 0
Internally allocated memory: 0
Time to perform memory allocation: 0.00 seconds

Module allocation information for function posit_na_float:
Number of complex operations: 0
Number of complex operations: 0
Time to perform module allocation: 0.06 seconds

Scheduling Information of function posit_na_float:
Number of control steps: 8
Minimum slack: 4.1631999999999998
Estimated max frequency (MHz): 171.3267543859649
Time to perform scheduling: 0.00 seconds

State Transition Graph Information of function posit_na_float:
Number of states: 6
Done port is registered
Time to perform creation of STG: 0.00 seconds

Easy binding information for function posit_na_float:
Bound operations:52/61
Time to perform easy binding: 0.00 seconds

Storage Value Information of function posit_na_float:
Number of storage values inserted: 10
Time to compute storage value information: 0.00 seconds

Register binding information for function posit_na_float:
Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
Time to perform register binding: 0.00 seconds

Register binding information for function posit_na_float:
Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
Time to perform register binding: 0.00 seconds

Module binding information for function posit_na_float:
Number of modules instantiated: 60
Number of possible conflicts for possible false paths introduced by
resource sharing: 0
Estimated resources area (no Muxes and address logic): 532
Estimated area of MUX21: 20
Total estimated area: 552
Estimated number of DSPs: 0
Slack computed in 0.00 seconds
False-loop computation completed in 0.00 seconds
Weight computation completed in 0.00 seconds
Clique covering computation completed in 0.00 seconds
Time to perform module binding: 0.00 seconds
```

```
Register binding information for function posit_na_float:
Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
Time to perform register binding: 0.00 seconds

Connection Binding Information for function posit_na_float:
Number of allocated multiplexers (2-to-1 equivalent): 7
Time to perform interconnection binding: 0.00 seconds

Total number of flip-flops in function posit_na_float: 136
```

OUTPUT 3. OUTPUT BAMBU HLS DLA ES = 3

```
**************************************************************************
*****
                    ____                    _
                   | __ )  __ _ _ __ ___  | |_   _   _
                   |  _ \ / _` | '_ ` _ \| '_ \| | | |
                   | |_) | (_| | | | | | | |_) | |_| |
                   |____/ \__,_|_| |_| |_|_.__/ \__,_|

**************************************************************************
*****
                         High-Level Synthesis Tool

                         Politecnico di Milano - DEIB
                          System Architectures Group
**************************************************************************
*****
                Copyright (C) 2004-2020 Politecnico di Milano
Version: PandA 0.9.7-dev - Revision
151822f6eb6b28b68ef7cde4c7c3c0add185ed9d-panda-0.9.7-dev

Target technology = FPGA

  Functions to be synthesized:
    posit_na_float


  Memory allocation information:
    BRAM bitsize: 8
    Spec may not exploit DATA bus width
    All the data have a known address
    Internal data is not externally accessible
    DATA bus bitsize: 8
    ADDRESS bus bitsize: 5
    SIZE bus bitsize: 4
    ALL pointers have been resolved
    Internally allocated memory (no private memories): 0
```

```
    Internally allocated memory: 0
  Time to perform memory allocation: 0.00 seconds


  Module allocation information for function posit_na_float:
    Number of complex operations: 0
    Number of complex operations: 0
  Time to perform module allocation: 0.06 seconds


  Scheduling Information of function posit_na_float:
    Number of control steps: 8
    Minimum slack: 4.1631999999999998
    Estimated max frequency (MHz): 171.3267543859649
  Time to perform scheduling: 0.02 seconds


  State Transition Graph Information of function posit_na_float:
    Number of states: 6
    Done port is registered
  Time to perform creation of STG: 0.00 seconds


  Easy binding information for function posit_na_float:
    Bound operations:56/65
  Time to perform easy binding: 0.00 seconds


  Storage Value Information of function posit_na_float:
    Number of storage values inserted: 10
  Time to compute storage value information: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Module binding information for function posit_na_float:
    Number of modules instantiated: 64
    Number of possible conflicts for possible false paths introduced by
resource sharing: 0
    Estimated resources area (no Muxes and address logic): 539
    Estimated area of MUX21: 20
    Total estimated area: 559
```

```
    Estimated number of DSPs: 0
    Slack computed in 0.00 seconds
    False-loop computation completed in 0.00 seconds
    Weight computation completed in 0.00 seconds
    Clique covering computation completed in 0.00 seconds
  Time to perform module binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Connection Binding Information for function posit_na_float:
    Number of allocated multiplexers (2-to-1 equivalent): 7
  Time to perform interconnection binding: 0.00 seconds

  Total number of flip-flops in function posit_na_float: 136
```

OUTPUT 4. OUTPUT BAMBU HLS DLA ES = 4

```
**************************************************************************
*****
                    ____                    _
                   | __ )  __ _ _ __ ___  | |_   _   _
                   |  _ \ / _` | '_ ` _ \| '_ \| | | |
                   | |_) | (_| | | | | | | |_) | |_| |
                   |____/ \__,_|_| |_| |_|_.__/ \__,_|

**************************************************************************
*****
                         High-Level Synthesis Tool

                         Politecnico di Milano - DEIB
                          System Architectures Group
**************************************************************************
*****
                Copyright (C) 2004-2020 Politecnico di Milano
Version: PandA 0.9.7-dev - Revision
151822f6eb6b28b68ef7cde4c7c3c0add185ed9d-panda-0.9.7-dev

Target technology = FPGA

  Functions to be synthesized:
    posit_na_float


  Memory allocation information:
```

```
    BRAM bitsize: 8
    Spec may not exploit DATA bus width
    All the data have a known address
    Internal data is not externally accessible
    DATA bus bitsize: 8
    ADDRESS bus bitsize: 5
    SIZE bus bitsize: 4
    ALL pointers have been resolved
    Internally allocated memory (no private memories): 0
    Internally allocated memory: 0
  Time to perform memory allocation: 0.00 seconds


  Module allocation information for function posit_na_float:
    Number of complex operations: 0
    Number of complex operations: 0
  Time to perform module allocation: 0.07 seconds


  Scheduling Information of function posit_na_float:
    Number of control steps: 8
    Minimum slack: 4.1631999999999998
    Estimated max frequency (MHz): 171.3267543859649
  Time to perform scheduling: 0.01 seconds


  State Transition Graph Information of function posit_na_float:
    Number of states: 6
    Done port is registered
  Time to perform creation of STG: 0.00 seconds


  Easy binding information for function posit_na_float:
    Bound operations:53/62
  Time to perform easy binding: 0.00 seconds


  Storage Value Information of function posit_na_float:
    Number of storage values inserted: 10
  Time to compute storage value information: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds
```

```
  Module binding information for function posit_na_float:
    Number of modules instantiated: 61
    Number of possible conflicts for possible false paths introduced by
resource sharing: 0
    Estimated resources area (no Muxes and address logic): 534
    Estimated area of MUX21: 20
    Total estimated area: 554
    Estimated number of DSPs: 0
    Slack computed in 0.00 seconds
    False-loop computation completed in 0.00 seconds
    Weight computation completed in 0.00 seconds
    Clique covering computation completed in 0.00 seconds
  Time to perform module binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Connection Binding Information for function posit_na_float:
    Number of allocated multiplexers (2-to-1 equivalent): 7
  Time to perform interconnection binding: 0.01 seconds

  Total number of flip-flops in function posit_na_float: 137
```

OUTPUT 5. OUTPUT BAMBU HLS DLA ES = 5

```
 ==  Bambu executed with: bambu --std=c99 --top-fname=posit_na_float
posit_16_5.c


**************************************************************************
*****
                     ____                  _
                    | __ )  __ _ _ __ ___ | |_   _   _
                    |  _ \ / _` | '_ ` _ \| '_ \| | | |
                    | |_) | (_| | | | | | | |_) | |_| |
                    |____/ \__,_|_| |_| |_|_.__/ \__,_|

**************************************************************************
*****
                          High-Level Synthesis Tool

                          Politecnico di Milano - DEIB
                           System Architectures Group
```

```
********************************************************************************
*****
                Copyright (C) 2004-2020 Politecnico di Milano
Version: PandA 0.9.7-dev - Revision
151822f6eb6b28b68ef7cde4c7c3c0add185ed9d-panda-0.9.7-dev

Target technology = FPGA

  Functions to be synthesized:
    posit_na_float


  Memory allocation information:
    BRAM bitsize: 8
    Spec may not exploit DATA bus width
    All the data have a known address
    Internal data is not externally accessible
    DATA bus bitsize: 8
    ADDRESS bus bitsize: 5
    SIZE bus bitsize: 4
    ALL pointers have been resolved
    Internally allocated memory (no private memories): 0
    Internally allocated memory: 0
  Time to perform memory allocation: 0.00 seconds


  Module allocation information for function posit_na_float:
    Number of complex operations: 0
    Number of complex operations: 0
  Time to perform module allocation: 0.07 seconds


  Scheduling Information of function posit_na_float:
    Number of control steps: 8
    Minimum slack: 4.1631999999999998
    Estimated max frequency (MHz): 171.3267543859649
  Time to perform scheduling: 0.01 seconds


  State Transition Graph Information of function posit_na_float:
    Number of states: 6
    Done port is registered
  Time to perform creation of STG: 0.01 seconds


  Easy binding information for function posit_na_float:
    Bound operations:57/66
  Time to perform easy binding: 0.00 seconds


  Storage Value Information of function posit_na_float:
    Number of storage values inserted: 10
```

```
  Time to compute storage value information: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Module binding information for function posit_na_float:
    Number of modules instantiated: 65
    Number of possible conflicts for possible false paths introduced by
resource sharing: 0
    Estimated resources area (no Muxes and address logic): 541
    Estimated area of MUX21: 20
    Total estimated area: 561
    Estimated number of DSPs: 0
    Slack computed in 0.00 seconds
    False-loop computation completed in 0.00 seconds
    Weight computation completed in 0.00 seconds
    Clique covering computation completed in 0.00 seconds
  Time to perform module binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Connection Binding Information for function posit_na_float:
    Number of allocated multiplexers (2-to-1 equivalent): 7
  Time to perform interconnection binding: 0.00 seconds

  Total number of flip-flops in function posit_na_float: 138
```

OUTPUT 6. OUTPUT BAMBU HLS DLA ES = 6

```
*******************************************************************************
*****
                     ____                    _
                    | __ )  __ _ _ __ ___  | |_   _   _
                    |  _ \ / _` | '_ ` _ \| '_ \| | | |
                    | |_) | (_| | | | | | | |_) | |_| |
                    |____/ \__,_|_| |_| |_|_.__/ \__,_|

*******************************************************************************
*****
                           High-Level Synthesis Tool

                           Politecnico di Milano - DEIB
                            System Architectures Group
*******************************************************************************
*****
                 Copyright (C) 2004-2020 Politecnico di Milano
Version: PandA 0.9.7-dev - Revision
151822f6eb6b28b68ef7cde4c7c3c0add185ed9d-panda-0.9.7-dev

Target technology = FPGA

  Functions to be synthesized:
    posit_na_float


  Memory allocation information:
    BRAM bitsize: 8
    Spec may not exploit DATA bus width
    All the data have a known address
    Internal data is not externally accessible
    DATA bus bitsize: 8
    ADDRESS bus bitsize: 5
    SIZE bus bitsize: 4
    ALL pointers have been resolved
    Internally allocated memory (no private memories): 0
    Internally allocated memory: 0
  Time to perform memory allocation: 0.00 seconds


  Module allocation information for function posit_na_float:
    Number of complex operations: 0
    Number of complex operations: 0
  Time to perform module allocation: 0.07 seconds


  Scheduling Information of function posit_na_float:
```

```
    Number of control steps: 8
    Minimum slack: 4.1631999999999998
    Estimated max frequency (MHz): 171.3267543859649
  Time to perform scheduling: 0.01 seconds


  State Transition Graph Information of function posit_na_float:
    Number of states: 6
    Done port is registered
  Time to perform creation of STG: 0.00 seconds


  Easy binding information for function posit_na_float:
    Bound operations:53/62
  Time to perform easy binding: 0.00 seconds


  Storage Value Information of function posit_na_float:
    Number of storage values inserted: 10
  Time to compute storage value information: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Module binding information for function posit_na_float:
    Number of modules instantiated: 61
    Number of possible conflicts for possible false paths introduced by
resource sharing: 0
    Estimated resources area (no Muxes and address logic): 534
    Estimated area of MUX21: 20
    Total estimated area: 554
    Estimated number of DSPs: 0
    Slack computed in 0.00 seconds
    False-loop computation completed in 0.00 seconds
    Weight computation completed in 0.00 seconds
    Clique covering computation completed in 0.00 seconds
  Time to perform module binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
```

```
  Time to perform register binding: 0.00 seconds


  Connection Binding Information for function posit_na_float:
    Number of allocated multiplexers (2-to-1 equivalent): 7
  Time to perform interconnection binding: 0.00 seconds

  Total number of flip-flops in function posit_na_float: 138
```

OUTPUT 7. OUTPUT BAMBU HLS DLA ES = 7

```
**************************************************************************
*****
                     ____                    _
                    | __ )  __ _ _ __ ___ | |_   _   _
                    |  _ \ / _` | '_ ` _ \| '_ \| | | |
                    | |_) | (_| | | | | | | |_) | |_| |
                    |____/ \__,_|_| |_| |_|_.__/ \__,_|

**************************************************************************
*****
                          High-Level Synthesis Tool

                          Politecnico di Milano - DEIB
                           System Architectures Group
**************************************************************************
*****
                 Copyright (C) 2004-2020 Politecnico di Milano
Version: PandA 0.9.7-dev - Revision
151822f6eb6b28b68ef7cde4c7c3c0add185ed9d-panda-0.9.7-dev

Target technology = FPGA

  Functions to be synthesized:
    posit_na_float


  Memory allocation information:
    BRAM bitsize: 8
    Spec may not exploit DATA bus width
    All the data have a known address
    Internal data is not externally accessible
    DATA bus bitsize: 8
    ADDRESS bus bitsize: 5
    SIZE bus bitsize: 4
    ALL pointers have been resolved
    Internally allocated memory (no private memories): 0
    Internally allocated memory: 0
  Time to perform memory allocation: 0.00 seconds
```

```
  Module allocation information for function posit_na_float:
    Number of complex operations: 0
    Number of complex operations: 0
  Time to perform module allocation: 0.06 seconds


  Scheduling Information of function posit_na_float:
    Number of control steps: 8
    Minimum slack: 4.1631999999999998
    Estimated max frequency (MHz): 171.3267543859649
  Time to perform scheduling: 0.01 seconds


  State Transition Graph Information of function posit_na_float:
    Number of states: 6
    Done port is registered
  Time to perform creation of STG: 0.00 seconds


  Easy binding information for function posit_na_float:
    Bound operations:52/63
  Time to perform easy binding: 0.01 seconds


  Storage Value Information of function posit_na_float:
    Number of storage values inserted: 10
  Time to compute storage value information: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Module binding information for function posit_na_float:
    Number of modules instantiated: 62
    Number of possible conflicts for possible false paths introduced by
resource sharing: 0
    Estimated resources area (no Muxes and address logic): 517
    Estimated area of MUX21: 20
    Total estimated area: 537
    Estimated number of DSPs: 0
    Slack computed in 0.00 seconds
```

```
    False-loop computation completed in 0.00 seconds
    Weight computation completed in 0.00 seconds
    Clique covering computation completed in 0.00 seconds
  Time to perform module binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Connection Binding Information for function posit_na_float:
    Number of allocated multiplexers (2-to-1 equivalent): 7
  Time to perform interconnection binding: 0.00 seconds

  Total number of flip-flops in function posit_na_float: 138
```

OUTPUT 8. OUTPUT BAMBU HLS DLA ES = 8

```
**************************************************************************
*****
                    ____                    _
                   | __ )  __ _ _ __ ___ | |_   _   _
                   |  _ \ / _` | '_ ` _ \| '_ \| | | |
                   | |_) | (_| | | | | | | |_) | |_| |
                   |____/ \__,_|_| |_| |_|_.__/ \__,_|

**************************************************************************
*****
                         High-Level Synthesis Tool

                         Politecnico di Milano - DEIB
                          System Architectures Group
**************************************************************************
*****
                Copyright (C) 2004-2020 Politecnico di Milano
Version: PandA 0.9.7-dev - Revision
151822f6eb6b28b68ef7cde4c7c3c0add185ed9d-panda-0.9.7-dev

Target technology = FPGA

  Functions to be synthesized:
    posit_na_float


  Memory allocation information:
    BRAM bitsize: 8
    Spec may not exploit DATA bus width
```

```
    All the data have a known address
    Internal data is not externally accessible
    DATA bus bitsize: 8
    ADDRESS bus bitsize: 5
    SIZE bus bitsize: 4
    ALL pointers have been resolved
    Internally allocated memory (no private memories): 0
    Internally allocated memory: 0
  Time to perform memory allocation: 0.00 seconds


  Module allocation information for function posit_na_float:
    Number of complex operations: 0
    Number of complex operations: 0
  Time to perform module allocation: 0.06 seconds


  Scheduling Information of function posit_na_float:
    Number of control steps: 8
    Minimum slack: 4.1631999999999998
    Estimated max frequency (MHz): 171.3267543859649
  Time to perform scheduling: 0.01 seconds


  State Transition Graph Information of function posit_na_float:
    Number of states: 6
    Done port is registered
  Time to perform creation of STG: 0.00 seconds


  Easy binding information for function posit_na_float:
    Bound operations:51/62
  Time to perform easy binding: 0.00 seconds


  Storage Value Information of function posit_na_float:
    Number of storage values inserted: 10
  Time to compute storage value information: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds
```

```
  Module binding information for function posit_na_float:
    Number of modules instantiated: 61
    Number of possible conflicts for possible false paths introduced by
resource sharing: 0
    Estimated resources area (no Muxes and address logic): 525
    Estimated area of MUX21: 20
    Total estimated area: 545
    Estimated number of DSPs: 0
    Slack computed in 0.00 seconds
    False-loop computation completed in 0.00 seconds
    Weight computation completed in 0.00 seconds
    Clique covering computation completed in 0.00 seconds
  Time to perform module binding: 0.00 seconds


  Register binding information for function posit_na_float:
    Register allocation algorithm obtains a sub-optimal result: 10
registers(LB:6)
  Time to perform register binding: 0.00 seconds


  Connection Binding Information for function posit_na_float:
    Number of allocated multiplexers (2-to-1 equivalent): 7
  Time to perform interconnection binding: 0.00 seconds

  Total number of flip-flops in function posit_na_float: 140
```

Excel – makro do arkusza

```
Function ZliczIdentyczneBityPoziomo(ciag As Range) As Long
    Dim i As Long
    Dim currentBit As String
    Dim previousBit As String

    ZliczIdentyczneBityPoziomo = 0
    previousBit = ""

    For i = 1 To ciag.Columns.Count
        currentBit = ciag.Cells(1, i).Value   ' Pobranie wartości komórki w
danym wierszu i kolumnie

        If currentBit = "" Then
            Exit For
        End If

        ' Sprawdzenie czy aktualny bit jest identyczny z poprzednim
        If currentBit = previousBit Or previousBit = "" Then
```

```
            ZliczIdentyczneBityPoziomo = ZliczIdentyczneBityPoziomo + 1
        Else
            Exit For
        End If

        previousBit = currentBit
    Next i
End Function

Function KomorkaZatrzymaniaPoziomo(ciag As Range) As String
    Dim ostatniaKomorka As Long
    Dim zliczoneBity As Long

    zliczoneBity = ZliczIdentyczneBityPoziomo(ciag)
    ostatniaKomorka = zliczoneBity + 1

    If ostatniaKomorka > 0 Then
        KomorkaZatrzymaniaPoziomo = ciag.Cells(1,
ostatniaKomorka).Address(False, False)
    Else
        KomorkaZatrzymaniaPoziomo = "Brak identycznych bitów"
    End If
End Function

Function KonwersjaBinarnaNaDziesietna(ciag As Range, przesuniecie As
Integer) As Long
    Dim liczbaDziesietna As Long
    Dim i As Long

    ' Początkowa wartość liczby dziesiętnej
    liczbaDziesietna = 0

    ' Iteracja przez komórki od pierwszej do przesunięcia + 1 (czyli
ostatniej komórki z identycznymi bitami)
    For i = 1 To przesuniecie + 1
        ' Pobranie wartości komórki
        Dim wartoscBitu As Long
        wartoscBitu = ciag.Cells(1, i).Value

        ' Dodanie wartości bitu przemnożonej przez odpowiednią potęgę
dwójki do liczby dziesiętnej
        liczbaDziesietna = liczbaDziesietna + wartoscBitu * (2 ^
(przesuniecie - (i - 1)))
    Next i

    ' Zwrócenie obliczonej liczby dziesiętnej
    KonwersjaBinarnaNaDziesietna = liczbaDziesietna
End Function

Function ObliczWartoscDziesietna(ciag As Range) As Long
    Dim przesuniecie As Long
```

```
    ' Znajdź przesunięcie, czyli liczbę identycznych bitów
    przesuniecie = ZliczIdentyczneBityPoziomo(ciag)

    ' Sprawdź, czy znaleziono identyczne bity
    If przesuniecie = 0 Then
        ObliczWartoscDziesietna = -1  ' Zwróć -1, jeśli nie znaleziono
identycznych bitów
    Else
        ' Oblicz wartość dziesiętną liczby binarnej
        ObliczWartoscDziesietna = KonwersjaBinarnaNaDziesietna(ciag,
przesuniecie - 1)
    End If
End Function

Function KonwertujBinarnaNaDziesietna(ciag As Range) As Long
    Dim liczbaDziesietna As Long
    Dim i As Long

    ' Początkowa wartość liczby dziesiętnej
    liczbaDziesietna = 0

    ' Iteracja przez komórki w podanym zakresie (jednokierunkowy zakres)
    For i = 1 To ciag.Cells.Count
        ' Pobranie wartości bitu (0 lub 1)
        Dim wartoscBitu As Long
        wartoscBitu = ciag.Cells(1, i).Value

        ' Sprawdzenie czy wartość bitu jest 0 lub 1
        If wartoscBitu <> 0 And wartoscBitu <> 1 Then
            KonwertujBinarnaNaDziesietna = -1  ' Zwróć -1 jeśli komórki
zawierają niepoprawne wartości
            Exit Function
        End If

        ' Dodanie wartości bitu przemnożonej przez odpowiednią potęgę
dwójki do liczby dziesiętnej
        liczbaDziesietna = liczbaDziesietna + wartoscBitu * (2 ^
(ciag.Cells.Count - i))
    Next i

    ' Zwrócenie obliczonej liczby dziesiętnej
    KonwertujBinarnaNaDziesietna = liczbaDziesietna
End Function

Function KonwertujBinarnaNaDziesietnaZmiennoprzecinkowa(ciag As Range) As
Double
    Dim liczbaDziesietna As Double
    Dim i As Long
    Dim czescCalkowita As Double
    Dim czescUlamkowa As Double
```

```
    ' Początkowa wartość liczby dziesiętnej
    liczbaDziesietna = 0
    czescCalkowita = 1  ' Pierwszy bit jest częścią całkowitą (domyślnie 1)
    czescUlamkowa = 0

    ' Iteracja przez komórki w podanym zakresie (jednokierunkowy zakres)
    For i = 1 To ciag.Cells.Count
        ' Pobranie wartości bitu (0 lub 1)
        Dim wartoscBitu As Long
        wartoscBitu = ciag.Cells(1, i).Value

        ' Sprawdzenie czy wartość bitu jest 0 lub 1
        If wartoscBitu <> 0 And wartoscBitu <> 1 Then
            KonwertujBinarnaNaDziesietnaZmiennoprzecinkowa = -1  ' Zwróć -1
jeśli komórki zawierają niepoprawne wartości
            Exit Function
        End If

        ' Jeśli to część ułamkowa (komórki 2 i dalej), dodaj do części
ułamkowej
        If i >= 2 Then
            czescUlamkowa = czescUlamkowa + wartoscBitu * (2 ^ (1 - i))
        End If
    Next i

    ' Obliczanie wartości dziesiętnej liczby zmiennoprzecinkowej
    liczbaDziesietna = czescCalkowita + czescUlamkowa

    ' Zwrócenie obliczonej liczby dziesiętnej
    KonwertujBinarnaNaDziesietnaZmiennoprzecinkowa = liczbaDziesietna
End Function




Sub projekt()

End Sub
```

Spis literatury

1. Artykuł bazowy: *Murillo20234040* - **Generating Posit-Based Accelerators With High-Level Synthesis**
2. Artykuł znaleziony w sieci*: Posit Working Group*- **Standard for Posit™ Arithmetic (2022).** https://posithub.org/docs/posit_standard-2.pdf
3. Prezentacja Politechniki Częstochowskiej: *dr hab. inż. Łukasz Szustak, prof. PCz* - **Binarne reprezentacje danych oraz arytmetyka systemów komputerowych.** https://icis.pcz.pl/~lszustak/students/ASK/wyklady/ASK_L02.pdf