Zastosowanie arytmetyki ułamkowej w reprogramowalnych jednostkach przetwarzających systemów jednoukładowych

Oleg Maslennikow[1], Piotr Ratuszniak[1], Anatolij Sergiyenko[2], Piotr Pawłowski[1]
[1]Wydział Elektroniki i Informatyki
Politechnika Koszalińska
ul. Śniadeckich 2, 75-453 Koszalin , Polska
[2]National Technical University of Ukraine
Pr. Peremogy 37, 03056 Kiev, Ukraine

1. Wprowadzenie

W niniejszej pracy przedstawiono wyniki badań przeprowadzonych przez 7-osobowy zespół badawczy na Wydziale Elektroniki i Informatyki Politechniki Koszalińskiej w latach 2006-2008 w ramach projektu badawczego MNiSW N515 002 32/0176. Tematyka projektu dotyczy projektowania strukturalnego i logicznego jednostek przetwarzających systemów jednoukładowych SoC (*ang.* SoC – system-on-chip), a jego podstawowym celem była weryfikacja zalet, jakie może dać realizacja w obszarach reprogramowalnych systemów SoC jednostek przetwarzających działających w arytmetyce ułamkowej (*ang.* RFA – ration fraction arithmetic).

Podstawową tezą projektu było wykazanie, że zastosowanie arytmetyki FRA w jednostkach przetwarzających systemów SoC przeznaczonych do realizacji wybranych algorytmów algebry liniowej i innych algorytmów regularnych wymagających wykonania operacji dzielenia, pozwala:

- zmniejszyć ich złożoność sprzętową (w porównaniu do odpowiednich jednostek przetwarzających działających w arytmetyce stałoprzecinkowej, przy zachowaniu zadanej dokładności obliczeń), dzięki dwukrotnemu zmniejszeniu złożoności sprzętowej kombinacyjnych bloków mnożenia i zamiany bloków dzielenia na bloki mnożenia;
- zwiększyć wydajność jednostek przetwarzających, dzięki zmniejszeniu czasu opóźnienia bloków mnożenia i szczególnie dzielenia, możliwości wykorzystania wbudowanych bloków mnożenia lub bloków DSP (w przypadku wykorzystania platform reprogramowalnych, np. Xilinx Virtex IIPro, Virtex 4, Altera Stratix II lub nowszych) oraz możliwości realizacji, w tym samym obszarze reprogramowalnym SoC, większej liczby bloków operacyjnych;
  lub
  zmniejszyć moc pobieraną przez jednostkę przetwarzającą, szczególnie przez jednostkę równoległą, dzięki zmniejszeniu jej złożoności sprzętowej lub

zmniejszeniu częstotliwości zegara systemowego (do wartości zapewniającej jednostce pożądaną wydajność).

Aktualność przeprowadzonych badań potwierdzają obecnie coraz bardziej rozpowszechnione tendencje w projektowaniu systemów SoC, których podstawowym celem jest radykalne zmniejszenie czasu projektowania systemu SoC:

a) wykorzystanie gotowych projektów IP-core (*ang*. Intellectual Property Core) do realizacji poszczególnych modułów (podukładów) systemu na różnych poziomach projektowania, od poziomu „soft-core" (tj. opisu modułu w języku opisu sprzętu, np. w VHDL – *ang*. Very high speed integrated circuit Hardware Description Language) do poziomu topografii modułu w określonej technologii krzemowej, tj. „hard-core" oraz (co za tym idzie),
b) realizacja w układzie scalonym osobnych obszarów („wysp") technologicznych różniących się np. typem technologii krzemowej (bramki, układy reprogramowalne, pamięć RAM/ROM), napięciem zasilania i częstotliwością zegara.

Realizacja w układzie scalonym z SoC obszarów reprogramowalnych w celu implementacji w nich jednostek przetwarzających: zwiększa funkcjonalność i niezawodność systemu oraz umożliwia jego modernizację w przyszłości; pozwala zwiększyć wydajność systemu i zmniejszyć pobór mocy poprzez dopasowanie architektury jednostki do realizowanego algorytmu i organizację równoległego przetwarzania danych. Ponadto, w przypadku małoseryjnej produkcji systemu SoC, dąży się do realizacji całego systemu w oparciu o inne, znacznie tańsze niż ASIC platformy sprzętowe, np. nowoczesne układy i platformy reprogramowalne [1], jak np. układy Virtex IIPro, Virtex 4 firmy Xilinx, które poza ogromnym polem komórek z architekturą FPGA (*ang*. Field Programmable Gate Array) i bloków pamięci operacyjnej (do kilkuset 18Kb bloków pamięci w jednym układzie), zawierają rdzenie 32-bitowych mikroprocesorów RISC PowerPC 405D bez jednostek zmienno-przecinkowych (nawet do 4 rdzeni PowerPC w jednym układzie), dużą liczbę 18-bitowych stałoprzecinkowych bloków mnożących, a nawet bloków DSP (tzw. DSP48slice, do 512 w jednym układzie, gdzie 1 blok DSP zawiera jeden 18-bitowy blok mnożący z 48-bitowym akumulatorem, kilkoma rejestrami i multiplekserami). Podobną architekturę mają inne układy FPGA innych firm-producentów, np. układy Stratix II, Stratix III firmy Altera [2].

Niestety, zarówno wyniki badań członków zespołu, jak i analiza publikacji ostatnich kilku lat dotyczących w/w tematyki, np. [3-12] świadczą o dość niskiej efektywności wykorzystania zasobów układów reprogramowalnych przy implementacji w nich zarówno zmiennoprzecinkowych jednostek przetwarzających, jak i stałoprzecinkowych jednostek przetwarzających z możliwością wykonania operacji dzielenia. Mianowicie, realizacja zmiennoprzecinkowych bloków operacyjnych oraz wielofunkcyjnych ALU wymaga wykorzystania dużej liczby komórek CLB (*ang*. Configurable Logic Block) układu FPGA, przy czym ta liczba drastycznie (nawet 10-krotnie) wzrasta w przypadku realizacji bloków dzielenia lub bloków ALU

z możliwością wykonania w/w operacji (ponieważ w tym przypadku nie da się wykorzystać wbudowane 18-bitowe bloki mnożące, wchodzące do składu bloków DSP). Pewną alternatywą tradycyjnemu sposobowi realizacji sprzętowej operacji x/y jest obliczenie wartości $1/y$ z następnym wymnożeniem wyniku tej operacji przez x. Jednak metoda obliczenia wartości $1/y$ jest metodą iteracyjną, w której liczba iteracji zależy od wymaganej dokładności wyniku (ponadto wymagane jest wykonanie dwóch operacji mnożenia i jednego dodawania w każdej iteracji). Więc mimo możliwości wykorzystania wbudowanych bloków mnożących, czas trwania operacji dzielenia pary liczb zmiennoprzecinkowych przedstawionych w standardowych formatach IEEE 754 *single* (32-bitowych), a tym bardziej *double* (64-bitowych) jest kilkadziesiąt razy dłuższy od czasu wykonania mnożenia tychże liczb. Analogicznie, operacja pierwiastkowania może również być wykonana w sposób klasyczny przez specjalistyczny układ kombinacyjny (lub potokowy, również bez możliwości wykorzystania wbudowanych bloków mnożenia) lub w sposób iteracyjny (wymagający wykonania jednej operacji dzielenia i jednego dodawania w każdej iteracji), co również powoduję kilkudziesięciokrotne wydłużenie czasu jej wykonania w stosunku do operacji mnożenia. Jednak jeśli operacja pierwiastkowania jest zazwyczaj bardzo rzadko stosowana w algorytmach algebry liniowej i prawie nigdy w algorytmach cyfrowego przetwarzania sygnałów [12-14], to operacja dzielenia jest charakterystyczna dla większości algorytmów algebry liniowej (np. eliminacji Gaussa, redukcji wstecznej, Gaussa-Jordana, Cholesky'ego-Banachewicza, Householdera, Givensa, gradientów sprzężonych, itd.) oraz jest dość często spotykana w algorytmach zaawansowanego przetwarzania sygnałów (np. w algorytmach Durbina, Levinsona, itd.). Przy tym wspólną cechą w/w algorytmów zawierających operacje dzielenia jest to, że operacja ta powinna być wykonana tylko jeden raz w każdym kroku algorytmu, tj. zwykle jeden raz na $O(N)$ operacji mnożenia z dodawaniem (gdzie N jest rozmiarem tablicy danych wejściowych), przy czym bez wyniku operacji dzielenia nie da się zacząć kolejny krok algorytmu. Z jednej strony to oznacza, że operacja dzielenia powinna być wykonana jak najszybciej, a z drugiej – że wprowadzenie potokowego trybu obliczeń nie pomoże je przyspieszyć, ponieważ operacja ta jest zwykle wykonywana na pojedynczej parze danych.

Wykonane przez członków zespołu badawczego wstępne badania [10, 11, 15, 16] potwierdziły możliwość efektywnego zastosowania arytmetyki ułamkowej [17 - 20] w jednostkach przetwarzających systemów SoC realizujących algorytmy, w których wymagania dot. dokładności przeprowadzanych obliczeń nie jest zbyt wysokie, np. nie wymaga się stosowanie formatu zmiennoprzecinkowego o podwójnej precyzji (64-bitowego formatu IEEE 754 *Double*) lub większej oraz wymagane jest wykonanie operacji dzielenia. Przykładami takich algorytmów mogą być np. (poza wymienionymi wyżej) algorytmy filtracji rekursywnej i adaptacyjnej, rekursywnej estymacji i przetwarzania Fouriera oraz większość algorytmów rozwiązywania układów równań liniowych. W związku z tym, w przeprowadzonych

badaniach położono nacisk na opracowanie i optymalizację (pod względem złożoności sprzętowej oraz wydajności) bloków operacyjnych i jednostek przetwarzających RFA, przeznaczonych do realizacji w nowoczesnych układach FPGA.

2. Arytmetyka ułamkowa a stałoprzecinkowa

W trakcie badań nad arytmetyką ułamkową oraz jednostkami przetwarzającymi RFA, zostało ustalone m.in., że w blokach operacyjnych działających w tej arytmetyce każda liczba l powinna być przedstawiona w postaci ułamku wymiernego a/b, gdzie a i b są n-bitowe liczby całkowite. Przy tym dokładność takiej $2n$-bitowej liczby l jest na pewno nie mniejsza, a zwykle jest większa od dokładności zapisu tejże liczby w $2n$-bitowym formacie stałoprzecinkowym. Na przykład, liczba π z dokładnością siedmiu cyfr dziesiętnych może być przedstawiona ułamkiem $l = 355/113$, zawierającym razem (w liczniku i mianowniku) tylko 6 cyfr dziesiętnych. Format ułamkowy pozwala na dokładniejsze przedstawienie liczb, w porównaniu do formatu stało-przecinkowego, jeszcze i dlatego, że liczby racjonalne zwykle są przedstawione w formacie stałoprzecinkowym z pewnymi błędami zaokrągleń, których wielkość zależy od liczby bitów w reprezentacji liczby. Dzięki temu, np. (1/9)*9 zawsze będzie równe 1 w arytmetyce ułamkowej, ale nie zawsze będzie równe 1 w arytmetyce stałoprzecinkowej lub nawet zmiennoprzecinkowej. Różnica liczby 1/9 i jej reprezentacji 16-bitowej w formacie stałoprzecinkowym wynosi ok. 0,0000187, a (1/9)*9 w tym przypadku będzie równe 0,9998932. Ponadto, wykonanie działań arytmetycznych na niedokładnie przedstawionych argumentach zwykle powoduje powstanie jeszcze bardziej niedokładnego wyniku. Na przykład, po wykonaniu operacji mnożenia pary liczb n-bitowych, dokładny $2n$-bitowy wynik mnożenia zostaje (w formacie stałoprzecinkowym) zaokrąglony do n-bitowego. W arytmetyce ułamkowej też po wykonaniu operacji mnożenia dwóch ułamków n-bitowych a/b i c/d powstaje ułamek $2n$-bitowy $a \cdot c / (b \cdot d)$, tj. liczba bitów w otrzymywanych wynikach ciągle wzrasta (ta sama sytuacja powstaję przy wykonaniu operacji dzielenia, a nawet dodawania, która wymaga wykonania 3 operacji mnożenia). Badania członków zespołu jednak wykazały, że w arytmetyce ułamkowej błędy zaokrągleń w RFA można dość skutecznie zredukować poprzez wykonanie operacji normalizacji wyniku mnożenia (tj. przesunięcie w lewo licznika i mianownika o jednakową liczbę bitów tak, aby co najmniej jeden z nich miał najbardziej znaczący bit tuż za bitem znaku) i tylko zatem jego zaokrąglenie do ułamku n-bitowego.

Badania dotyczące porównania dokładności obliczeń przeprowadzonych na liczbach stałoprzecinkowych i ułamkach wykonane zostały na przykładzie wielokrotnego obliczenia wyrażenia matematycznego (1) często wykorzystywanego w zadaniach cyfrowego przetwarzania sygnałów (m.in. w generatorach funkcji sin$x$) i algebry liniowej

(m.in. w metodzie QR obliczenia wartości własnych macierzy, lub w rozwiązaniu problemu najmniejszych kwadratów w oparciu o metodę obrotów Givensa):

$$x_{i+1} = x_i \cdot \cos\beta + y_i \cdot \sin\beta \quad \text{oraz} \quad y_{i+1} = y_i \cdot \cos\beta - x_i \cdot \sin\beta,$$

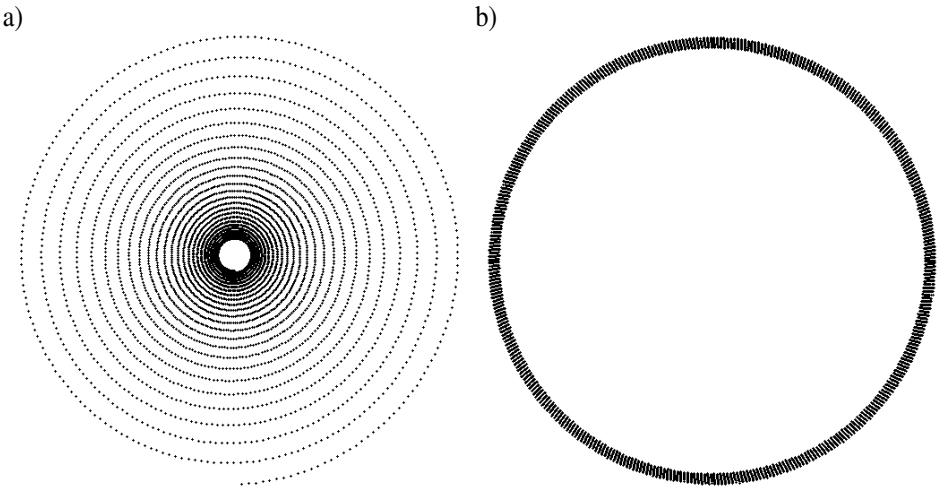
gdzie i=1,2,…,N. (1)

Stwierdzono, że głównymi przyczynami powstania błędów przy obliczeniu wyrażenia (1) są: niedokładne przedstawienie argumentów $\sin\beta$ i $\cos\beta$ (powodujące, że $(\sin\beta)^2+(\cos\beta)^2\neq1$) oraz zaokrąglenie wyników pośrednich (np. po wykonaniu operacji mnożenia). Przy czym stwierdzono, że błąd obliczeń rośnie wraz ze wzrostem liczby iteracji N. W celu oszacowania błędów obliczeń opracowano specjalny program komputerowy, który oblicza współrzędne $N = 200$ punktów (x_i, y_i) według wzoru (1). Obliczenia były przeprowadzone na 32-bitowych liczbach stałoprzecinkowych, a współczynniki $\sin\beta$ i $\cos\beta$ były reprezentowane jako 32-bitowe liczby stało-przecinkowe równe odpowiednio $\sin(2\pi/N)$ oraz $\cos(2\pi/N)$. W przypadku obliczeń bezbłędnych te punkty powinny formować koło o zadanym promieniu R. Wykonanych zostało $K = 5000$ iteracji wzoru (1), tj. powinno było powstać 5000/200=25 jednakowych, nakładających się na siebie kół o promieniu R. Wyniki obliczeń przeprowadzonych na liczbach stałoprzecinkowych reprezentuje rys. 1a. Jak widać, wartość promienia R z każdą iteracją się zmniejsza i dąży do zera (błąd jednej iteracji wynosi ok. $3{,}5\cdot10^{-4}$). W drugim przypadku obliczenia były prowadzone na modelu VHDL jednostki przetwarzającej RFA, tj. na prawdziwych ułamkach, przy czym wartości $\sin\beta$ i $\cos\beta$ były formowane z wykorzystaniem tzw. trójek Pitagorejskich, tj. liczb całkowitych odpowiadających długościom przyprostokątnych a, b i przeciwprosto-kątnej c w trójkącie prostokątnym. Przykładami takich trójek są np. liczby {3,4,5}, {12,5,13}, itd. Liczby $\{a, b, c\}$ były formowane ze wzorów (2):

$$a = 2mn,\ b = m^2-n^2,\ c = m^2+n^2,\ m > n, \quad (2)$$

gdzie m i n – liczby naturalne, wzajemno pierwsze, jedna z nich jest parzysta, druga – nieparzysta oraz $m > n$ za pomocą specjalnego programu komputerowego, który otrzymuje jako dane wejściowe wartość kąta β, dopuszczalną wartość błędu δ_{max} oraz maksymalną wartość zmiennej m (m_{max}), a oblicza wszystkie możliwe trójki $\{a, b, c\}$ z zadanego przedziału (m = 2,..., m_{max} , n = 1,..., m-1) oraz odpowiednie wartości błędów $\delta < \delta_{max}$. Jeśli dla zadanego zbioru danych wejściowych nie uda się znaleźć rozwiązań, należy zwiększyć wartość m_{max} i uruchomić program ponownie. Na przykład, dla zadanych wartości β=10°, m_{max} = 1000, $\delta_{max} = 1{,}5\cdot10^{-3}$ trójka {92,525,533} określa kąt β z błędem 0,1%, natomiast trójka {1120, 6351, 6449} – ten sam kąt z błędem 0,002%. Wyniki działania opracowanego modelu VHDL po wykonaniu $K = 5000$ iteracji wzoru (1), w przypadku obliczeń przeprowadzonych na 16-bitowych ułamkach, reprezentuje rys. 1b. Jak widać, dokładność obliczeń znacznie wzrosła (ściślej mówiąc, błąd δ dla tej samej wartości kąta β został zmniejszony ok. 2^5=32 razy, w przypadku przeprowadzenia

działań na ułamkach). Przy tym badania wykazały, że ten stosunek prawie nie zależy od wartości kąta β.



Rys. 1. Graficzna reprezentacja wyników wielokrotnego obliczenia wyrażenia (1) w arytmetyce stałoprzecinkowej (a) i ułamkowej (b)

Ponadto, w trakcie badań nad arytmetyką ułamkową RFA, zostało ustalono m.in., że

- jeśli liczba $x = a/b$ jest ułamkiem, np. $x = -1/9$, to w RFA liczba ta jest przedstawiona następująco: $a = -1$ i $b = 9$; jeśli x jest liczbą całkowitą, np. 7, to $a = 7$ i $b = 1$; jeśli $x = 0$, to $a = 0$ i $b = 11.....1$; jeśli liczba x jest liczbą rzeczywistą, np. 8.512, to $a = 8512$ i $b = 1000$; przy tym w celu zwiększenia dokładności obliczeń warto wykonywać opisaną wyżej normalizację zarówno danych wejściowych, jak i wyników pośrednich;
- w dowolnej liczbie ułamkowej *a/b* tylko licznik może być przedstawiony jako liczba ze znakiem (najlepiej w kodzie uzupełnieniowym do dwóch UD, ponieważ wbudowane bloki mnożące działają na liczbach w kodzie UD, lub na liczbach bez znaku); mianownik może być przechowywany jako liczba całkowita bez znaku;
- do przechowania *n*-bitowej liczby zespolonej $x = a/b$ potrzebne są dwa *n*-bitowych liczniki i jeden *n*-bitowy mianownik, tj. 1,5 razy mniej bitów, niż dla przechowania tej samej liczby w formacie stałoprzecinkowym (przy zachowaniu tej samej precyzji);
- operacja porównania dwóch liczb *a/b* oraz *c/d* sprowadza się do obliczenia znaku wyrażenia ($ad - cb$) i następnie obliczenia sumy *modulo* 2 trzech zmiennych jednobitowych, reprezentujących znak tego wyrażenia, znak *b* i znak *d*; stąd wynika, że operacja ta może być realizowana w dowolnej jednostce ALU RFA

(zawierającej blok mnożenia i sumator) prawie bez dodatkowych nakładów sprzętowych;

- mnożenie dwóch ułamków *a/b* i *c/d* w RFA sprowadza się do wykonania dwóch operacji mnożenia liczb *n*-bitowych, zamiast jednego mnożenia dwóch liczb *2n*-bitowych w formacie stałoprzecinkowym. Dlatego złożoność sprzętowa kombinacyjnego bloku mnożącego RFA i jego opóźnienie są około 2 razy mniejsze w porównaniu do kombinacyjnego bloku mnożącego dwie *2n*-bitowe liczby w formacie stałoprzecinkowym. Porównanie złożoności sprzętowej bloku dzielenia RFA z blokiem dzielenia dwóch *2n*-bitowych liczb w formacie stałoprzecinkowym wypada jeszcze lepiej, ponieważ dzielenie w RFA sprowadza się do wykonania dwóch operacji mnożenia liczb *n*-bitowych.

3. Projekty bloków operacyjnych i jednostek arytmetyczno-logicznych RFA

Zachęcające wyniki badań dokładności obliczeń w arytmetyce ułamkowej stymulowały dalszą pracę zespołu badawczego nad opracowaniem projektów bloków operacyjnych i jednostek arytmetyczno-logicznych ALU przeznaczonych do realizacji w układach reprogramowalnych i porównaniem ich parametrów z podobnymi jednostkami stało- i zmiennoprzecinkowymi. Jako przykład na rys. 2a przedstawiono schemat jednostki ALU przeznaczonej do wykonywania podstawowych operacji arytmetycznych w arytmetyce ułamkowej na liczbach *n*-bitowych, tj. dodawania (odejmowania) P = X+Y, mnożenia P = A·X, dzielenia P = A/X, mnożenia z dodawaniem (odejmowaniem) P = A·X+Y oraz dzielenia z dodawaniem (odejmowaniem) P = A/X+Y ułamków *n*-bitowych. Wyżej wymienione operacje w arytmetyce ułamkowej są przedstawione odpowiednio wzorami (3) – (7),

$$\frac{P_n}{P_d} = \frac{X_n}{X_d} + \frac{Y_n}{Y_d} = \frac{X_n \cdot Y_d + X_d \cdot Y_n}{X_d \cdot Y_d} \tag{3}$$

$$\frac{P_n}{P_d} = \frac{X_n}{X_d} \cdot \frac{Y_n}{Y_d} = \frac{X_n \cdot Y_n}{X_d \cdot Y_d} \tag{4}$$

$$\frac{P_n}{P_d} = \frac{X_n}{X_d} / \frac{Y_n}{Y_d} = \frac{X_n \cdot Y_d}{X_d \cdot Y_n} \tag{5}$$

$$\frac{P_n}{P_d} = \frac{A_n}{A_d} \cdot \frac{X_n}{X_d} + \frac{Y_n}{Y_d} = \frac{(A_n \cdot X_n) \cdot Y_d + (A_d \cdot X_d) \cdot Y_n}{(A_d \cdot X_d) \cdot Y_d} \tag{6}$$

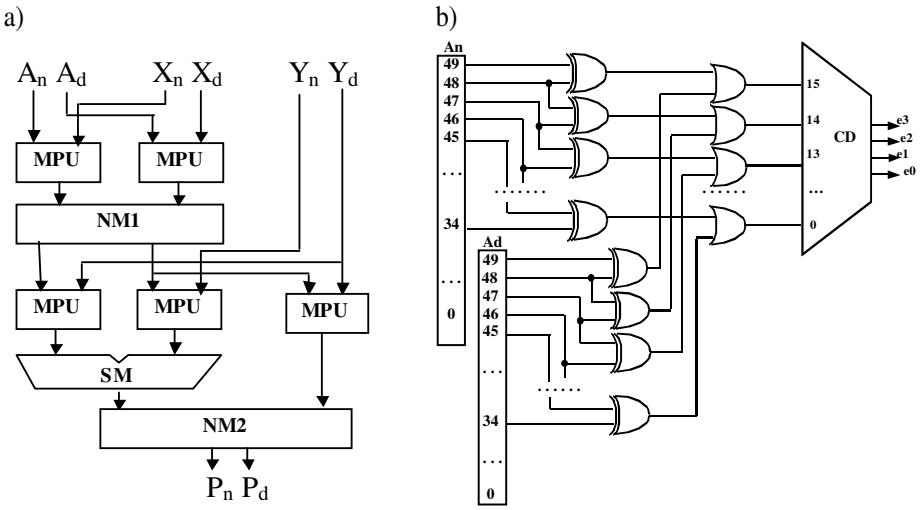
$$\frac{P_n}{P_d} = \frac{A_n}{A_d} / \frac{X_n}{X_d} + \frac{Y_n}{Y_d} = \frac{(A_n \cdot X_d) \cdot Y_d + (A_d \cdot X_n) \cdot Y_n}{(A_d \cdot X_n) \cdot Y_d} \tag{7}$$

gdzie A_n, X_n, Y_n i P_n reprezentują n-bitowe liczniki, a A_d, X_d, Y_d i P_d – n-bitowe mianowniki odpowiednio argumentów i wyniku operacji. Jednostka zawiera pięć n-bitowych bloków mnożących (MPU), jeden n-bitowy sumator (SM) i dwa bloki normalizacji wyniku (NM1 i NM2). Należy zaznaczyć, że w przypadku wykonania operacji dzielenia lub dzielenia z dodawaniem wartości X_n i X_d podawane na wejścia ALU należy zamienić miejscami a w przypadku operacji mnożenia lub dzielenia przyjąć Y = 0, tj. $Y_n = 0$ i $Y_d = 1$.

W trakcie badań zaprezentowanego projektu ALU RFA stwierdzono, że przy jego implementacji w układach FPGA Vitex4, bloki MPU i SM są realizowane w oparciu o wbudowane blokach DSP (DSP48Slice), natomiast bloki NM1 i NM2 – na blokach CLB (Slice) układu FPGA. Stwierdzono również, że wymagana dokładność obliczeń osiąga się poprzez wybór odpowiedniej liczby bitów n w reprezentacji ułamków i konstruowanie odpowiednich bloków mnożących. Na przykład w układach reprogramowalnych rodzin Virtex IIPro i Virtex 4, n –bitowe bloki mnożące RFA (w przypadku $n > 18$) trzeba realizować w oparciu o kilka wbudowanych 18-bitowych bloków mnożenia. Jako przykład, rys. 3 reprezentuje sposób otrzymania 35-bitowego bloku mnożącego MPU z czterech wbudowanych 18-bitowych bloków mnożących i sumatorów lub z czterech bloków DSP układu FPGA Virtex4. Zakłada się ze 48-bitowe wyjście tego bloku powinno być podłączone do wejścia bloku normalizacji NM, i tam wynik zostaje skrócony do 35 bitów. Na rys. 2b natomiast przedstawiono przykładową realizację fragmentu bloku normalizacji określającego liczbę bitów, na którą trzeba wykonać przesunięcie 48-bitowego licznika i 48-bitowego mianownika po wykonaniu operacji mnożenia.

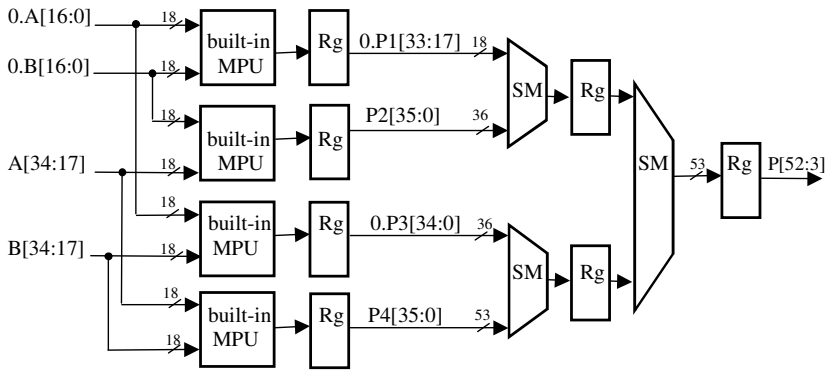
Opracowane ALU zostało wykorzystane w jednostce przetwarzającej przeznaczonej do rozwiązania układów równań liniowych w oparciu o metodę gradientów sprzężonych [15]. Zbieżność tej iteracyjnej metody zależy nie tylko od określonych warunków, które ma spełniać macierz współczynników układu, lecz w dużym stopniu zależy od dokładności prowadzonych obliczeń, dlatego zwykle ona jest realizowana w arytmetyce zmiennoprzecinkowej z m-bitową mantysą, gdzie wartość m zależy od rozmiaru macierzy danych wejściowych N zgodnie z wykresem przedstawionym na rys. 4. W związku z tym w oparciu o ALU z rys. 2a autorzy zaprojektowali 35-bitową jednostkę przetwarzającą RFA, która została zrealizowana w układzie Xilinx VirtexIIPro (XC2VP4). Zaprojektowana jednostka korzysta z 18-bitowych wbudowanych bloków mnożących oraz wbudowanych 18Kb bloków pamięci dwuportowej. Poza tym, w celu zwiększenia maksymalnej częstotliwości działania, zaprojektowana jednostka działa w trybie potokowym 9-stopniowym. Porównanie pierwiastków układu równań liniowych otrzymanych przez jednostkę RFA oraz przez odpowiedni program numeryczny realizowany na PC na 32-bitowych liczbach zmiennoprzecinkowych świadczy, że dla metody gradientów sprzężonych dokładność obliczeń jest w obu przypadkach

porównywalna. To oznacza, że zaprojektowana 35-bitowa jednostka RFA może pewnie realizować metodę gradientów sprzężonych na macierzach stopnia $N < 3000$.



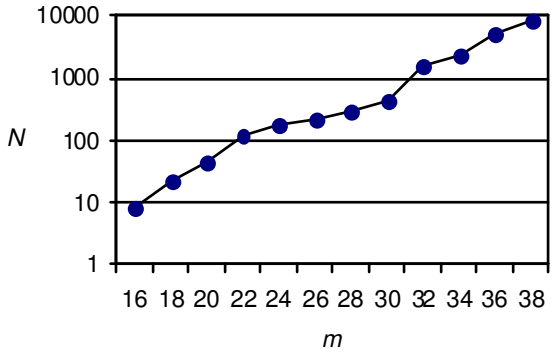
Rys. 2. Projekt jednostki RFA (a) oraz struktura wewnętrzna fragmentu bloku normalizacji NM1 (b)

Ciekawie wygląda porównanie nakładów sprzętowych opracowanej 35-bitowej jednostki RFA ze znanymi ALU zmiennoprzecinkowymi [4] oraz [6], wykonującymi mniejszy zbiór operacji, chociaż z większą dokładnością. Analiza danych z tab. 1 świadczy o ponad trzykrotnej przewadze jednostki RFA pod względem liczby wykorzystanych bloków CLB przy porównywalnej maksymalnej częstotliwości zegara systemowego. Ponadto, opracowana jednostka ma tylko 9 stopni w potoku, tj. może efektywniej działać na krótkich tablicach danych wejściowych.



Rys. 3. Struktura 35-bitowego bloku mnożącego złożonego z czterech 18-bitowych wbudowanych bloków mnożenia (lub z 4 bloków DSP48slice)

W ramach przeprowadzonych badań członkowie zespołu opracowali również projekty innych bloków operacyjnych RFA wykonujących w arytmetyce ułamkowej m.in. operację porównania, mnożenia z akumulacją wyniku, obliczenia pierwiastka kwadratowego oraz bloki konwersji ułamku *a/b* w liczbę stało- lub zmiennoprzecinkową *x* i na odwrót i inne. Przy opracowaniu struktur wewnętrznych w/w bloków operacyjnych dążono do uzyskania jednakowej liczby stopni w potoku w każdym z nich oraz porównywalnej maksymalnej częstotliwości działania (co miało na celu łatwiejsze tworzenie w ich oparciu wielofunkcyjnych jednostek ALU).



Rys. 4. Zależność szerokości *m* mantysy liczb zmiennoprzecinkowych od rozmiaru *N* macierzy danych wejściowych gwarantująca zbieżność metody gradientów sprzężonych

Tab. 1. Wyniki implementacji proponowanego ALU ze znanymi w układzie FPGA Virtex IIPro

Parametry ALU	**ALU ułamkowe**	**ALU [4]**	**ALU [6]***
Liczba wykorzystanych bloków CLB (slice), Liczba wykorzystanych bloków mnożących	1005 20	4625 9	2825 9
Liczba stopni w potoku	9	34	13
Maksymalna częstotliwość działania, MHz	138	120	140

* brak operacji dzielenia

W trakcie optymalizacji bloku pierwiastkowania stwierdzono, że operację pierwiastkowania w RFA najlepiej wykonywać w sposób iteracyjny w oparciu o metodę

Newtona, w której każda iteracja wymaga wykonania tylko jednej operacji dzielenia, jednego dodawania i jednego przesunięcia o 1 bit w prawo. Takie rozwiązanie powoduje, że dodanie operacji pierwiastkowania do listy operacji wykonywanych przez jednostkę ALU RFA, już zawierającej blok mnożenia i sumator, prawie nie zwiększa jej złożoności sprzętowej, ponieważ dzielenie dwóch liczb *a/b* oraz *c/d* w RFA sprowadza się do wykonania 2 operacji mnożenia (więc pierwiastkowanie da się przeprowadzić w oparciu o wbudowane w układy FPGA bloki DSP). Tym nie mniej, blok pierwiastkowania powinien zawierać układ pamięci stałej ROM o niewielkiej pojemności, w którym będą przechowywane początkowe (przybliżone) wartości wyniku. Wykonawcom projektu udało się zmniejszyć pojemność ROM w bloku pierwiastkowania do 16 komórek, podając na wejścia adresowe ROM 4 starsze bity z wyjścia układu normalizacji wyniku (fragment takiego układu przedstawiono na rys. 2b). Wszystkie opracowane bloki operacyjne zostały ostatecznie przedstawione w postaci syntezowalnych modeli VHDL i są sparametryzowane, przy czym głównym parametrem każdego modelu jest szerokość danych wejściowych i wyników. Większość opracowanych bloków ma opóźnienie potoku 4 takty zegarowe (bloki dodawania-odejmowania oraz mnożenia z akumulacją wyniku mają 5 stopni w potoku, tj. opóźnienie 5 taktów), a następnie wyniki pojawiają się na wyjściach bloku co takt. Wyjątkiem jest blok pierwiastkowania, w którym wykonywano od 4 iteracji algorytmu Newtona w przypadku 35-bitowych ułamków do 6 iteracji dla ułamków dłuższych. Jednak, mimo że opóźnienie potoku w tym bloku wynosi 48 taktów zegarowych, dzięki równoległemu przetwarzaniu 12 danych wejściowych, wyniki pojawiają się na wyjściu bloku co 4 takty, a częstotliwość jego działania jest porównywalna z częstotliwością działania bloków pozostałych.

W oparciu o projekty bloków operacyjnych FRA członkowie zespołu opracowali program-generator IP-core formujący na wyjściu syntezowalne modele (opisy) VHDL wybranych przez użytkownika bloków [21]. Wszystkie modele VHDL w/w bloków operacyjnych mogą być wygenerowane dla danych wejściowych i wyjściowych o szerokości 24 i 32 bitów oraz 18 i 35 bitów (co uwzględnia możliwości wbudowanych bloków mnożących i bloków DSP układów FPGA firmy Xilinx). Wprowadzono również możliwość wyboru liczby stopni potoku dla projektowanego bloku operacyjnego, co ma duży wpływ na maksymalną częstotliwość jego działania, tj. na jego wydajność, i dlatego jest szczególnie ważne przy wykorzystywaniu kilku bloków w jednej jednostce przetwarzającej.

Sprawdzenie poprawności działania opracowanych bloków operacyjnych RFA odbywało się najpierw poprzez weryfikację ich modeli VHDL w symulatorze ActiveHDL firmy Aldec. Poprawność i dokładność otrzymywanych wyników obliczeń była weryfikowana poprzez porównanie wyników otrzymywanych z modeli bloków RFA z wynikami z modeli funkcjonalnych VHDL analogicznych bloków zmiennoprzecinkowych, w których wszystkie wykorzystane zmienne miały typ *Real*. Kolejne badania miały na celu porównanie złożoności sprzętowej i maksymalnej

częstotliwości działania zaprojektowanych bloków operacyjnych RFA z podobnymi blokami działającymi na liczbach stało- i/lub zmiennoprzecinkowych, których modele były tworzone w środowisku Coregen wchodzącym do składu oprogramowania Xilinx Foundation ISE, mianowicie przez moduły Multiplier 10.0, Divider 1.0, Adder/Subtractor 7.0 tego środowiska. Synteza logiczna wszystkich bloków była przeprowadzona w środowisku Foundation ISE v.9.2 z konfiguracją w układach Xilinx FPGA Virtex4 (xc4vSX35-12). Niestety, w/w moduły Multiplier 10.0, Divider 1.0, Adder/Subtractor 7.0 nie pozwalają w szerokim zakresie określać parametry projektowanych bloków, np. moduł Divider 1.0 nie pozwala projektować stałoprzecinkowe bloki dzielenia dla danych o szerokości większej niż 32 bity, model bloku mnożenia z akumulacją wyniku można wygenerować dla danych maksimum 18-bitowych, itd. Z tego powodu w tab. 2 przedstawiono parametry tylko tych bloków stało- i zmiennoprzecinkowych, których modele udało się za pomocą w/w modułów wygenerować. Na podstawie otrzymanych wyników, z których podstawowe przedstawiono w tab. 2, można wywnioskować, że tezy autorów o mniejszej złożoności sprzętowej (nawet dwukrotnie) bloków mnożenia RFA w stosunku do odpowiednich bloków stałoprzecinkowych oraz o większej wydajności bloków RFA zostały potwierdzone. Np. 16-bitowe bloki mnożenia RFA są realizowane w oparciu o dwa bloki DSP (DSP48Slice) zamiast 4 takich bloków w 32-bitowym bloku stałoprzecinkowym. Większa liczba wykorzystanych bloków Slice w blokach RFA (122 zamiast 17) prawie nie wpływa na porównanie złożoności sprzętowej rozpatrywanych bloków dlatego, że złożoność sprzętowa 122 takich bloków jest ok. 4 razy mniejsza od złożoności sprzętowej jednego bloku DSP. Poza tym, te 122 bloki Slice są wykorzystane do realizacji bloku normalizacji wyniku NM (co korzystnie wpływa na dokładność obliczeń), a stałoprzecinkowy blok mnożący nie ma takiego bloku. Ponadto 16-bitowe bloki mnożące RFA działają z maksymalną częstotliwością 404 MHz zamiast 264 MHz w przypadku 32-bitowych bloków stałoprzecinkowych (przy zachowaniu jednakowej liczby stopni w potoku), tj. mają o ok. 55% większą wydajność. W przypadku 35-bitowych ułamków i 64-bitowych liczb stałoprzecinkowych przewagą w złożoności sprzętowej bloku mnożącego RFA maleje do 25%, ale z kolei przewaga w wydajności bloku RFA okazuje się ponad 2-krotna (176 MHz przeciw 88 MHz). Poza tym, 35-bitowy blok mnożący RFA ma większą dokładność obliczeń, niż odpowiedni 64-bitowy blok stałoprzecinkowy, który produkuje wynik 64-bitowy. Jeśli natomiast użytkownik modułu Multiplier 10.0 wybierze opcję „pełny wynik”(tj. 128-bitowy w tym przypadku), wówczas liczba wykorzystanych bloków DSP wzrośnie z 10 do 16, co już oznacza prawie dwukrotną przewagę w złożoności sprzętowej bloku mnożącego RFA. Porównanie 16-bitowych bloków dzielenia RFA z 32-bitowymi blokami stałoprzecinkowymi jeszcze bardziej podkreśla przewagę tych pierwszych, zarówno pod względem wykorzystanych bloków CLB (Slice), jak i wydajności. Na uwagę zasługuje fakt, że w blok stałoprzecinkowy nie wykorzystuje bloki DSP i dlatego cechuje się

dziesięciokrotnie większą liczbą bloków Slice oraz 36-stopniowym potokiem. Prawdopodobnie z tego powodu specjaliści firmy Xilinx w projekcie 32-bitowego bloku dzielenia zmiennoprzecinkowego zmienili sposób wykonania tej operacji na iteracyjny, co spowodowało możliwość wykorzystania bloków DSP oraz skrócenie liczby stopni w potoku do 10. Lecz 35-bitowy blok dzielenia RFA (który zapewnia porównywalną dokładność obliczeń z 32-bitowym zmiennoprzecinkowym blokiem dzielenia) cechuje się jednak mniejszą złożonością sprzętową (ok. 20%), ok. 50% większą wydajnością oraz 2,5 razy mniejszą liczbą stopni w potoku.

Tab. 2. Podstawowe parametry wybranych bloków operacyjnych RFA, stało- i zmienno-przecinkowych po ich implementacji w układzie Xilinx Virtex 4 (xc4vSX35-12)

Rodzaj bloku	RFA								
	ułamki 16-bitowe			ułamki 18-bitowe			ułamki 35-bitowe		
	Slice +DSP	MHz	Stopnie potoku	Slice +DSP	MHz	Stopnie potoku	Slice +DSP	MHz	Stopnie potoku
X*Y	122+2	404	4	188+2	220	4	402+8	176	4
X/Y	122+2	404	4	188+2	220	4	402+8	176	4
X+Y	134+3	247	5	200+3	241	5	372+12	166	5

Rodzaj bloku	Xilinx CoreGen (Multiplier 10.0, Divider 1.0, Adder/Subtractor 7.0)					
	liczby 32-bitowe stałoprzecinkowe			(liczby 32-bitowe zmiennoprzecinkowe)* (liczby 64-bitowe stałoprzecinkowe)**		
	Slice +DSP	MHz	Stopnie potoku	Slice +DSP	MHz	Stopnie potoku
X*Y	17+4	264	4	(51+10)**	88**	4**
X/Y	1229+0	254	36	(251+10)*	117*	10*
X+Y	96+0	416	5	(176+0)**	383**	5**

Operacja dodawania (odejmowania) w arytmetyce ułamkowej jest najbardziej skomplikowaną wśród w/w operacji arytmetycznych, ponieważ wymaga wykonania 3 operacji mnożenia i jednego dodawania(odejmowania). Z tego powodu złożoność sprzętowa 16-bitowego sumatora RFA jest znacznie większą od złożoności sprzętowej 32-bitowego sumatora stałoprzecinkowego (m.in. dlatego, że wykorzystuję 3 bloki DSP), a jego wydajność jest ok. 2 razy mniejsza. Lecz trudno sobie wyobrazić jednostkę

przetwarzającą (lub ALU), która powinna wykonywać wyłącznie operację dodawania. Jeśli natomiast do zbioru operacji wykonywanych przez jednostkę RFA dołączyć jeszcze przynajmniej jedną operację arytmetyczną - nawet tylko mnożenie, wówczas znajdujące się w jednostce wbudowane bloki mnożące lub DSP da się wykorzystać w obu operacjach: dodawania i mnożenia. Więc sumaryczna liczba wykorzystanych bloków DSP się nie zwiększy (lub prawie się nie zwiększy).

Kolejne badania opracowanych bloków operacyjnych RFA miały na celu porównanie podstawowych parametrów w/w bloków przy ich implementacji w układach FPGA różnych rodzin, np. Xilinx Virtex 4 (xc4vSX35-12) i Virtex5 (xc5vLX50-3) oraz różnych firm, np. Altera Stratix II (Eps215F672C5). Narzędziem do implementacji bloków RFA w układzie StratixII było środowisko Altera Quartus II 7.2 SP2. Otrzymane wyniki świadczą o tym, że w układach FPGA Xilinx Virtex4 oraz Altera StratixII wykorzystanych zostaje identyczna liczba wbudowanych 18-bitowych bloków mnożących zawartych w blokach DSP (blok DSP48Slice w rodzinie Virtex4 zawiera jeden 18-bitowy bloki mnożenia, blok DSP w rodzinie StratixII - 2 takie bloki), natomiast dla układu Virtex4 uzyskano wyższe częstotliwości pracy (o około 20-25%), niż dla układu StratixII.

Kolejnym zadaniem zespołu było opracowanie projektów potokowych i równoległych jednostek przetwarzających przeznaczonych do realizacji podstawowych algorytmów algebry liniowej. W ramach wykonania tego zadania wykorzystano wieloletnie doświadczenie kierownika zespołu w dziedzinie metod projektowania architektur procesorów systolicznych (*ang*. systolic array) oraz jednoukładowych macierzy procesorowych (*ang*. VLSI array processor) przeznaczonych do realizacji algorytmów cyfrowego przetwarzania sygnałów i algebry liniowej [12]. Jeszcze kilka lat temu architektury te mogły być realizowane fizycznie wyłącznie w postaci wyspecjalizowanych układów scalonych ASIC. Mimo to, że macierze procesorowe mają największą wartość współczynnika *wydajność/złożoność_sprzętowa* wśród wszystkich znanych typów architektur równoległych, wcześniej nie uzyskały one dużego rozpowszechnienia z powodu bardzo wysokiego stopnia specjalizacji (co uniemożliwia produkcję masową układu ASIC, niekorzystnie wpływając na jego cenę). Obecnie, dzięki rozwojowi układów reprogramowalnych, przewyższającemu ponad dwukrotnie wartości podane w prawie G. Moore'a, pojawia się potencjalna możliwość wykorzystania macierzy procesorowych jako równoległych jednostek przetwarzających (akceleratorów) systemów specjalistycznych przeznaczonych do realizacji w układach FPGA.

Niestety, już pierwsze próby bezpośredniej implementacji, przez zespół badawczy, projektów architektur macierzy procesorowych w układach FPGA wiodących producentów (Xilinx i Altera) wykazały zarówno niską efektywność wykorzystania zasobów sprzętowych w/w układów FPGA, jak i niską (w stosunku do maksymalnie możliwej) częstotliwość działania zaimplementowanego w układzie FPGA akceleratora. Przyczyn powstania opisanej sytuacji jest kilka. Po pierwsze, projektowanie logiczne

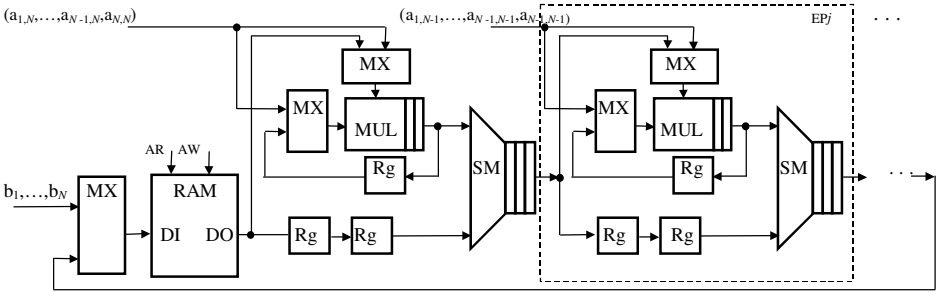
układów i systemów cyfrowych obecnie odbywa się z wykorzystaniem języków opisu sprzętu, takich jak np. VHDL, przy czym rożni projektanci mogą opisywać tę samą architekturę układu cyfrowego na różne sposoby (tym bardziej, że istnieją różne style opisu architektury: behawioralny, strukturalny i opis ścieżki danych). Po drugie, w celu zwiększenia częstotliwości działania zaprojektowanej architektury przynajmniej do 10%-15% maksymalnej częstotliwości działania układu FPGA, niezbędnym jest wprowadzenie w projekcie potokowego trybu obliczeń. Po trzecie, układy FPGA różnych firm, a nawet różnych rodzin posiadają zupełnie różne zasoby sprzętowe. To wszystko oznacza, że jakość projektu logicznego systemu cyfrowego (tj. głównie złożoność sprzętowa oraz maksymalna częstotliwość działania systemu) zależy od doświadczenia i wiedzy projektanta przynajmniej na temat projektowania architektur potokowych, budowy układów reprogramowalnych różnych rodzin oraz osobliwości zachowania oprogramowania do syntezy logicznej. Częściowym rozwiązaniem przedstawionych tu problemów jest wykorzystanie w projekcie systemu cyfrowego gotowych projektów jego podzespołów, tj. komponentów IP-core wybieranych z odpowiednich bibliotek. Jednak, mimo istnienia na rynku bibliotek IP-core, ich wykorzystanie również niesie ze sobą szereg problemów: różne bloki mogą mieć różne interfejsy, wydajność, formaty danych wejściowych i wyników, częstotliwość pracy, itd., przez co nie mogą być połączone ze sobą bezpośrednio.

Badania przeprowadzone przez członków zespołu wykazały dodatkowo, że szczególnie w przypadku projektów równoległych i potokowych jednostek przetwarzających, na jakość projektu logicznego jednostki mają wpływ nie tylko wyżej wymienione czynniki, lecz również projekt architektury jednostki, a często nawet konkretne algorytmy, które jednostka ma realizować. Innymi słowy, budowa i zasoby sprzętowe układu reprogramowalnego FPGA, w którym jednostka ma być implementowana, warto brać pod uwagę już na etapie określenia zbioru algorytmów, które jednostka ma realizować, a tym bardziej na etapie projektowania strukturalnego. Czasami algorytmy wejściowe należy modyfikować, aby uzyskać taką architekturę jednostki, której projekt logiczny (projekt RTL) przy implementacji w układzie FPGA będzie efektywnie wykorzystywał jego zasoby i miał częstotliwość bliską do maksymalnej częstotliwości działania układu FPGA.

Tak np. dla algorytmu redukcji wstecznej (*ang.* back substitution), najpierw dopracowano znaną architekturę macierzy procesorowej [12], a następnie struktury wewnętrzne poszczególnych elementów przetwarzających (EP) wprowadzając opracowane wyżej bloki operacyjne RFA. Uproszczona budowa nowej równoległej jednostki przetwarzającej przedstawiona jest na rys. 5, gdzie RAM oznacza blok pamięci dwuportowej z wejściami adresowymi zapisu AW i odczytu AR danych (uproszczona dlatego, że magistrale danych jeszcze nie są podzielone na magistrale liczników i mianowników, brakuje bloków sterowania, ponadto zakłada się, że blok mnożenia RFA ma 2 stopni w potoku, a blok dodawania RFA – 3 stopnie, co reprezentowane jest na rys.

5 prostokątami dołączonymi do w/w bloków). To oznacza, że czas wykonania każdej operacji mnożenia z odejmowaniem lub dzielenia wynosi 5 taktów zegarowych, ale równocześnie w każdym EP jednostki przetwarzane są 5 elementów macierzy **A**. W opracowanej potokowej architekturze jednostki liczba L elementów przetwarzających może być dowolna, lecz mniejsza od N oraz N/L – powinno być liczbą naturalną. W tym przypadku czas wykonania metody podstawienia będzie wynosił ok. $N^2/(2L)$ taktów, co odpowiada wysokiemu, bliskiemu 100% stopniu obciążenia (wykorzystania) poszczególnych EP. Niestety, członkom zespołu nie udało się porównać parametry zaprojektowanej jednostki ze znanymi z powodu braku podobnych projektów. Pod tym względem prawie wszystkie wykonane przez zespół projekty równoległych jednostek przetwarzających są projektami „pionierskimi".

Projekt opracowanej potokowej jednostki przetwarzającej został zaimplementowany w układzie FPGA Virtex 4. Dla najmniejszej liczby stopni potoku dla bloku mnożącego (tj. 2) i sumatora – 3, maksymalna częstotliwość działania jednostki przetwarzającej wyniosła około 90MHz; po podniesieniu liczby stopni potoku w bloku mnożącym do 5 i w sumatorze do 6 (poprzez włączenie rejestrów wewnątrz bloków DSP) maksymalna częstotliwość była równa 132MHz. Przy zwiększeniu liczby stopni potoku w sumatorze do 8 częstotliwość działania jednostki wyniosła około 173MHz, tj. około 33% od maksymalnej częstotliwości działania układu Virtex4. Dalsze zwiększenie częstotliwości dla systemu o takim stopniu złożoności jest już raczej niemożliwe.



Rys. 5. Architektura równoległej jednostki przetwarzającej realizującej metodę redukcji wstecznej (z uwzględnieniem 5-stopniowego potokowego trybu obliczeń)

Kolejnym projektem był projekt jednostki potokowej przeznaczonej do rozkładu LU macierzy trój-diagonalnej Jacobi'ego A(N,N) w oparciu o wzory (8) (które reprezentują najprostszą wersję algorytmu eliminacji Gaussa). W postaci macierzowej rozkład LU przedstawiono za pomocą równości (9), a wstępny projekt architektury

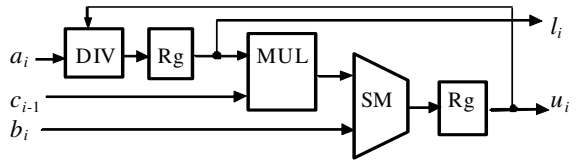
jednostki S_1 - na rys. 6 wraz z ilustracją jej działania, gdzie MUL, DIV, SM i Rg oznaczają odpowiednio blok mnożenia, blok dzielenia sumator i rejestr.

$$u_1 = b_1$$
$$l_i = a_i / u_{i-1} \qquad i=2,3,...,N \tag{8}$$
$$u_i = b_i - l_i * c_{i-1} \qquad i=2,3,...,N$$

$$\begin{bmatrix} b_1 & c_1 & & & & & \\ a_2 & b_2 & c_2 & & & & \\ & a_3 & b_3 & c_3 & & & \\ & & a_4 & b_4 & c_4 & & \\ & & & . & . & . & \\ & & & & a_{N-1} & b_{N-1} & c_{N-1} \\ & & & & & a_N & b_N \end{bmatrix} = \begin{bmatrix} 1 & & & & & & \\ l_2 & 1 & & & & & \\ & l_3 & 1 & & & & \\ & & l_4 & 1 & & & \\ & & & ... & ... & & \\ & & & & & l_N & 1 \end{bmatrix} * \begin{bmatrix} u_1 & c_1 & & & & & \\ & u_2 & c_2 & & & & \\ & & u_3 & c_3 & & & \\ & & & ... & ... & & \\ & & & & & u_{N-1} & c_{N-1} \\ & & & & & & u_N \end{bmatrix} \tag{9}$$

Następnie do eliminacji Gaussa została wprowadzona strategia lokalnego wyboru elementu wiodącego, aby jednostka przetwarzająca mogła wykonywać rozkład LU macierzy pasmowej Hessenberga, a nawet macierzy pasmowej (przykładowe struktury macierzy obu typów przedstawiono odpowiednio na rys. 7a i rys. 7b).

T A K T Y											
…	11	10	9	8	7	6	5	4	3	2	1
...		a_6		a_5		a_4		a_3		a_2	
…	c_5		c_4		c_3		c_2		c_1		c_0=0
…	b_6		b_5		b_4		b_3		b_2		b_1
WYNIKI											
...	u_6	l_6	u_5	l_5	u_4	l_4	u_3	l_3	u_2	l_2	u_1



Rys. 6. Architektura jednostki S_1 przeznaczonej do realizacji eliminacji Gaussa na macierzy Jacobi'ego wraz z ilustracją kolejności przeprowadzenia obliczeń

a) b)

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1h} & & \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots & a_{2,h+1} & \\ & a_{32} & a_{33} & a_{34} & a_{35} & \dots & a_{3,N} \\ & & a_{43} & a_{44} & a_{45} & \dots & a_{4,N} \\ & & & a_{54} & a_{55} & \dots & a_{5,N} \\ & & & & \dots & \dots & \dots \\ & & & & & a_{N,N-1} & a_{N,N} \end{bmatrix} \quad A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1h} & & & \\ a_{21} & a_{22} & a_{23} & \dots & a_{2,h+1} & & \\ \dots & a_{32} & a_{33} & a_{34} & \dots & \dots & \\ a_{f1} & \dots & \dots & \dots & \dots & \dots & a_{N-h+1,N} \\ & a_{f+1,2} & \dots & \dots & \dots & \dots & \dots \\ & & \dots & \dots & \dots & \dots & a_{N-1,N} \\ & & & a_{N,N-f+1} & \dots & a_{N,N-1} & a_{N,N} \end{bmatrix}$$

Rys. 7. Postacie ogólne macierzy: pasmowej (b) oraz pasmowej Hessenberga (a)

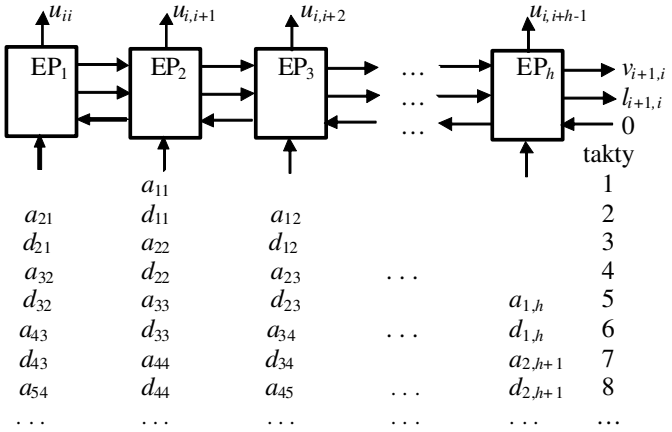
W przypadku macierzy pasmowej Hessenberga, w której szerokość pasma wynosi (h+1), metoda eliminacji Gaussa z lokalnym wyborem elementu wiodącego może być przedstawiona w postaci poniższego fragmentu programu (10):

```
for i = 1  to N-1 do
  begin
    if abs(a(i, i)) < abs(a(i+1,i))
        then  /*zamiana wierszy*/
            for k = i to h+1 do
                begin
                  u(i, k)=a(i+1, k); a(i+1,k)=a(i, k);
                end
       else  /*brak zamiany wierszy*/
            for k = i to h+1 do u(i, k)=a(i, k);
    /*eliminacja*/
    l(i+1, i)= a(i+1,i)/u(i, i);
    for k = i+1 to h+1 do a(j, k) = a(j, k) - l(j, i)*u(i, k);
                                        (10)
  end;
```

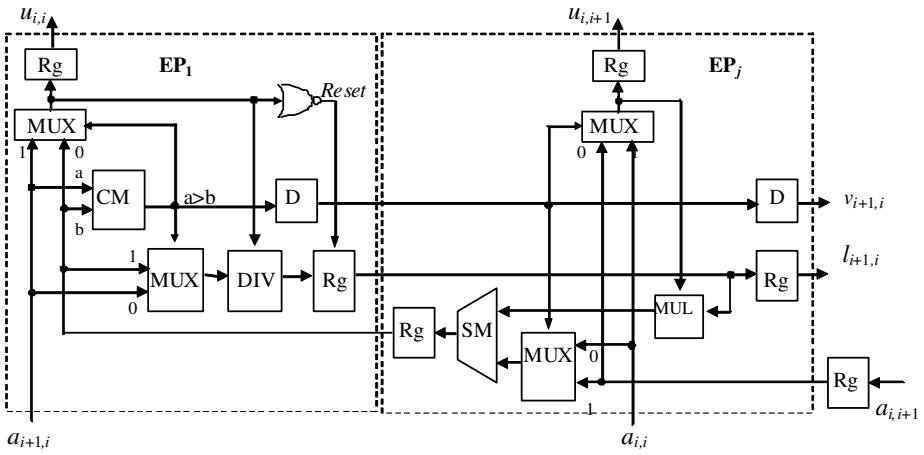
Zgodnie z opisaną w pracy [12] metodą odwzorowania, z programu (10) otrzymana została architektura równoległej jednostki przetwarzającej S_2, która jest przedstawiona na rys. 8. Jednostka zawiera h elementów przetwarzających EP dwóch różnych typów: EP_1, który realizuje operacje porównania wartości bezwzględnych elementów $a(i, i)$ z $a(i+1,i)$ i dzielenia oraz EP_j (j=2,…,h), które realizują operacje zamiany wierzy oraz mnożenia z odejmowaniem. Struktury wewnętrzne obu typów EP przedstawiono odpowiednio na rys. 9a i rys.9b, gdzie CM oznacza komparator, D – przerzutnik typu D, a MUX – multiplekser dwuwejściowy. Architektura S_2 cechuje się czasem realizacji metody eliminacji Gaussa równym ok. 2N taktów, co odpowiada ok. 50%-wemu stopniu obciążenia (wykorzystania) poszczególnych EP równoległej jednostki. Jednak analiza architektury wykazała, że jednostka może równocześnie przetwarzać dwie macierze danych wejściowych, np. **A** i **D** o tych samych rozmiarach N

i h, przy niezmiennym czasie wykonania rozkładu LU. W tym przypadku stopień obciążenia EP jednostki S_2 jest bliski 100%. Sposób podawania elementów macierzy **A** i **D** na wejścia jednostki, w przypadku ich równoległego przetwarzania przedstawiono w dolnej części rys. 8.

W celu realizacji architektury S_2 w arytmetyce ułamkowej opracowano nowy blok operacyjny wykonujący porównanie wartości bezwzględnych dwóch ułamków $|x|>|y|$ (tj. wartości bezwzględnych dwóch sąsiednich elementów tej samej kolumny macierzy $|a_{ii}|$ i $|a_{i+1,i}|$) i następnie dzielenia mniejszego elementu przez większy. W zależności od znaków liczników x_n, y_n i mianowników x_d, y_d , sprawdzenie warunku



Rys. 8. Architektura macierzy procesorowej S_2 przeznaczonej do realizacji metody eliminacji Gaussa na macierzy pasmowej Hessenberga

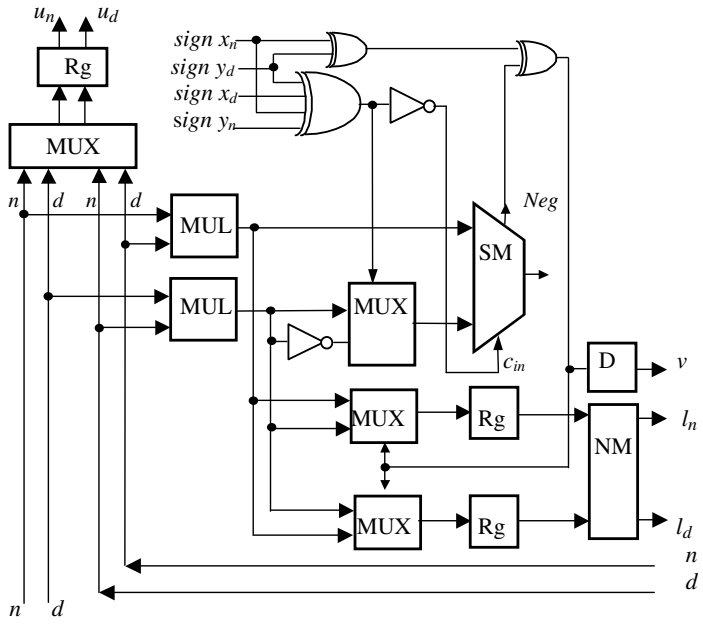


Rys. 9. Architektura wewnętrzna EP_1 (a) oraz EP_j (b) równoległej jednostki przetwarzającej S_2

$$\left|\frac{x_n}{x_d}\right| > \left|\frac{y_n}{y_d}\right| \tag{11}$$

sprowadza się do obliczenia znaku (funkcja *sign*) sumy lub różnicy iloczynów x_n*y_d oraz x_d*y_n . Więc podstawową operacją w procedurze porównania ułamków jest mnożenie z dodawaniem (odejmowaniem). Członkom zespołu badawczego udało się opracować oryginalną strukturę tego bloku operacyjnego wykorzystując obecny w EP_1 blok dzielenia (przecież dzielenie *x*/*y* w arytmetyce ułamkowej sprowadza się do obliczenia wartości licznika x_n*y_d i mianownika x_d*y_n wyniku) oraz wprowadzając tylko jeden dodatkowy sumator, dwa dwu-wejściowe multipleksery i blok normalizacji NM. Szczegółowa struktura wewnętrzna EP_1 działającego w RFA przedstawiono na rys. 10, gdzie *n* i *d* oznaczają odpowiednio licznik i mianownik odpowiedniej zmiennej, *Neg* oznacza znacznik ujemnego wyniku, a c_{in} – przeniesienie wejściowe sumatora SM.

Należy zaznaczyć, że w tej jednostce każdy blok mnożenia MUL ma 2 stopni w potoku, oraz tyle samo stopni ma blok normalizacji wyniku. To oznacza, że czas realizacji każdego wierzchołka grafu algorytmu wynosi 5 taktów zegarowych, ale równocześnie w każdym EP jednostki mogą być przetwarzane elementy pięciu różnych macierzy wejściowych **A**. W tym przypadku czas wykonania metody eliminacji Gaussa z lokalnym wyborem elementu wiodącego będzie wynosił ok. 2*N* taktów, co również odpowiada wysokiemu ok. 67% stopniu obciążenia poszczególnych EP równoległej jednostki. Projekt opracowanej potokowej jednostki przetwarzającej również został zaimplementowany w układzie FPGA Virtex 4 (4vlx15sf363-12). Dla 35-bitowych danych wejściowych oraz liczby stopni potoku w bloku mnożącym i sumatorze RFA równym odpowiednio 3 i 5, maksymalna częstotliwość działania jednostki przetwarzającej wyniosła około 149,5MHz, przy czym po podniesieniu liczby stopni potoku w bloku mnożącym do 5 i w sumatorze do 7 maksymalna częstotliwość wzrosła tylko do 151MHz. Z tego powodu dalsze zwiększeniu liczby stopni potoku nie miało sensu. Lecz te parametry częstotliwości zespół badawczy również rozpatruje jako ważne osiągnięcie. Ponadto wyniki implementacji równoległej jednostki świadczą o tym, że dla 35-bitowych danych wejściowych EP_1 zajmuję około 8 bloków DSP w/w układu FPGA oraz ok. 7% liczby CLB. Natomiast drugi i każdy następny EP architektury S_2 zajmuje 20 bloków DSP oraz 15% liczby CLB, więc w największym układzie FPGA rodziny Virtex4 (zawierającym 512 bloków DSP) może być umieszczona jednostka zawierająca nawet 25 takich EP.



Rys. 10. Struktura wewnętrzna EP_1 jednostki przetwarzającej S_2 dostosowana do działania w arytmetyce ułamkowej

Zgodnie z harmonogramem badań projektu, wykonawcy skupili uwagę na bardziej zaawansowanych algorytmach algebry liniowej, przeznaczonych do rozwiązywania układów równań liniowych i problemu najmniejszych kwadratów oraz do odnalezienia wartości własnych macierzy. Są to między innymi: algorytm Cholesky'ego-Banachewicza (wykonującego rozkład LL^T macierzy symetrycznych dodatnio określonych) oraz algorytmy odbić Householdera i obrotów Givensa (wykonujące rozkład QR macierzy prostokątnych). Mimo dużej różnicy w sposobie przekształcenia danych wejściowych oraz złożoności obliczeniowej tych algorytmów, ich wspólną cechą, która wyróżnia ich od rozpatrywanych wcześniej, jest pojawienie się operacji pierwiastkowania (oprócz mnożenia z dodawaniem i dzielenia). Przy czym w algorytmie Cholesky'ego (którego złożoność obliczeniowa wynosi $O(N^3/6)$, a liczba kroków jest równa N dla macierzy wejściowej stopnia N) operacja pierwiastkowania jest wykonywana tylko 1 raz na początku każdego kroku, a następne obliczenia są uzależnione od jej wyniku. Taka cecha algorytmu praktycznie uniemożliwia wprowadzenie potokowego trybu obliczeń w równoległych jednostkach przetwarzających oraz znacznie wydłuża czas realizacji algorytmu w jednostkach jednoprocesorowych potokowych. Z tego powodu zespół badawczy większą uwagę poświęcił opracowaniu projektu logicznemu jednostki wykonującej tylko jeden algorytm

z tej grupy – algorytm Cholesky'ego. Jako podstawę została wybrana opracowana przez kierownika zespołu architektura macierzy procesorowej, w której współczynnik obciążenia EP jest bliski 66%, tj. jest najlepszym wśród znanych architektur realizujących metodę Cholesky'ego. Niestety, bezpośrednia realizacja w układach FPGA tej macierzy procesorowej nie miała sensu ze względu na niefortunny podział operacji arytmetycznych w poszczególnych krokach algorytmu, co ilustruje tab. 3 dla przypadku macierzy A(4,4). Jak było już wyżej zaznaczono, operacja pierwiastkowania jest wykonywana tylko 1 raz na początku każdego kroku algorytmu, a następne obliczenia są uzależnione od jej wyniku. Z tego powodu przeanalizowano tok obliczeń w algorytmie Cholesky'ego i podjęto próbę zgrupowania wszystkich operacji pierwiastkowania i następnie przesunięcia ich na ostatni *N*-ty krok algorytmu.

Tab. 3. Kolejność obliczeń w klasycznym algorytmie Cholesky'ego-Banachewicza (dla macierzy A(4,4))

Krok 1		Krok 2	Krok 3	Krok 4
$l_{11}=\sqrt{a_{11}}$	$a^{*}_{22}=a_{22}-l_{21}*l_{21}$	$l_{22}=\sqrt{a^{*}_{22}}$	$l_{33}=\sqrt{a^{**}_{33}}$	$l_{44}=\sqrt{a^{***}_{44}}$
$l_{21}=a_{21}/l_{11}$	$a^{*}_{32}=a_{32}-l_{31}*l_{21}$	$l_{32}=a^{*}_{32}/l_{22}$	$l_{43}=a^{**}_{43}/l_{33}$	
$l_{31}=a_{31}/l_{11}$	$a^{*}_{42}=a_{42}-l_{41}*l_{21}$	$l_{42}=a^{*}_{42}/l_{22}$	$a^{***}_{44}=a^{**}_{44}-l_{43}*l_{43}$	
$l_{41}=a_{41}/l_{11}$	$a^{*}_{33}=a_{33}-l_{31}*l_{31}$	$a^{**}_{33}=a^{*}_{33}-l_{32}*l_{32}$		
	$a^{*}_{43}=a_{43}-l_{41}*l_{31}$	$a^{**}_{43}=a^{*}_{43}-l_{42}*l_{32}$		
	$a^{*}_{44}=a_{44}-l_{41}*l_{41}$	$a^{**}_{44}=a^{*}_{44}-l_{42}*l_{42}$		

Zgodnie ze wzorami z tab. 3, przykładowo nowa wartość elementu a_{22} jest równa $a^{*}_{22}=a_{22}-a_{21}\cdot a_{21}/(l_{11}\cdot l_{11})$, tj. nie ma potrzeby obliczenia wartości l_{11} (poprzez wykonanie operacji pierwiastkowania), można natomiast wykorzystać daną a_{11} zamiast mnożenia ($l_{11}\cdot l_{11}$). Analizując dalej dane z tab. 10 można wywnioskować, że w analogiczny sposób można modyfikować wszystkie pozostałe elementy macierzy: $a^{*}_{32},\ldots,a^{*}_{44}$. Obliczenie pierwiastka i wykonanie operacji dzielenia, a następnie mnożenie wyników pierwszych dwóch operacji powodują tylko zwiększenie błędów zaokrągleń. W związku z tym, wykonawcy projektu opracowali nową wersję algorytmu Cholesky'ego, która jest przedstawiona w tab. 4 (dla przypadku macierzy A(4,4)). Główną ideą jest wprowadzenie zmiennej pomocniczej m_{ji} zamiast elementów l_{ji} macierzy wynikowej, tak że np. $m_{21}=a_{21}/a_{11}$, i następnie modyfikacji elementów macierzy A w oparciu o zmienne m_{ji}.

Tab. 4. Zmodyfikowany algorytm Cholesky'ego dostosowany do potokowego trybu obliczeń

Krok 1	Krok 2	Krok 3	Krok 4
$m_{21} = a_{21} / a_{11}$	$m_{32} = a^{*}_{32} / a^{*}_{22}$	$m_{43} = a^{**}_{43} / a^{**}_{33}$	$l_{11} = \sqrt{a_{11}} \quad l_{22} = \sqrt{a^{*}_{22}}$
$m_{31} = a_{31} / a_{11}$	$m_{42} = a^{*}_{42} / a^{*}_{22}$	$a^{***}_{44} = a^{**}_{44} - a^{**}_{43} * m_{43}$	$l_{33} = \sqrt{a^{**}_{33}} \quad l_{44} = \sqrt{a^{***}_{44}}$
$m_{41} = a_{41} / a_{11}$	$a^{**}_{33} = a^{*}_{33} - a^{*}_{32} * m_{32}$		$l_{21} = m_{21} * l_{11}$
$a^{*}_{22} = a_{22} - a_{21} * m_{21}$	$a^{**}_{43} = a^{*}_{43} - a^{*}_{42} * m_{32}$		$l_{31} = m_{31} * l_{11}$
$a^{*}_{32} = a_{32} - a_{31} * m_{21}$	$a^{**}_{44} = a^{*}_{44} - a^{*}_{42} * m_{42}$		$l_{41} = m_{41} * l_{11}$
$a^{*}_{42} = a_{42} - a_{41} * m_{21}$			$l_{32} = m_{32} * l_{22}$
$a^{*}_{33} = a_{33} - a_{31} * m_{31}$			$l_{42} = m_{42} * l_{22}$
$a^{*}_{43} = a_{43} - a_{41} * m_{31}$			$l_{43} = m_{43} * l_{33}$
$a^{*}_{44} = a_{44} - a_{41} * m_{41}$			

Analiza zmodyfikowanego algorytmu Cholesky'ego świadczy o tym, że jest on dobrze dostosowany do realizacji w jednostkach potokowych, zarówno jednoprocesorowych, jak i równoległych, ponieważ wszystkie operacje arytmetyczne każdego z typów są zgrupowane (brak pojedynczych operacji pierwiastkowania, dzielenia a nawet mnożenia z dodawaniem, z wyjątkiem kroku przedostatniego). Ponadto, wynik każdej operacji nie jest natychmiast wykorzystywany w dalszych obliczeniach, a błędy zaokrągleń są mniejsze dzięki wyeliminowaniu zbędnych obliczeń (o czym już była mowa wyżej). Ceną za w/w korzystne cechy nowego algorytmu jest zwiększenie jego złożoności obliczeniowej o ok. $N^2/2$ operacji mnożenia, które w przypadku $N>50$ jest już niezauważalne (w porównaniu do złożoności oryginalnego algorytmu - $O(N^3/6)$ operacji). Zmodyfikowana wersja algorytmu Cholesky'ego wraz z architekturami potokowej i równoległej jednostek przetwarzających do jego realizacji została opublikowana m.in. w pracach [12, 22].

Po modyfikacji zarówno algorytmu Cholesky'ego, jak i architektury równoległej jednostki przetwarzającej przeznaczonej do jego realizacji zawierającej dowolną liczbę h EP (taka, że $p=N/h$ była liczba całkowita), zespół badawczy skupił się na projektowaniu logicznym jej elementów przetwarzających, z uwzględnieniem potokowego trybu obliczeń i osobliwości arytmetyki RFA. Analiza danych z tab. 4 świadczy o tym, że wszystkie EP, oprócz ostatniego powinny wykonywać operacje dzielenia i mnożenia z dodawaniem. Stąd wynika, że jako jednostka arytmetyczno-logiczna (ALU) wewnątrz każdego EPi (i=1,...,h-1) może być wykorzystana opracowana wcześniej 9-stopniowa potokowa jednostka ALU RFA (przedstawiona na rys. 2a), która już była wcześniej wykorzystana do rozwiązania układów równań liniowych w oparciu o metodę gradientów sprzężonych. Do w/w jednostki ALU RFA należy tylko dodać bloki pamięci typu RAM lub (jeszcze lepiej) kolejki FIFO oraz multipleksery. Struktura całego elementu EPi przedstawiono na rys. 11a, gdzie Rg oznacza rejestr dla przechowania zmiennej a_{ii}, a FIFO_1 i FIFO_2 – kolejki FIFO o długości początkowej

(*N*-*i*) komórek przeznaczone do przechowania odpowiednio elementów *i*-tej kolumny macierzy oraz elementów m_{ji}.

Natomiast ostatni EP powinien wykonywać operacje mnożenia i pierwiastkowania, przy czym zgodnie z wcześniejszymi ustaleniami, ostatnią operację w arytmetyce ułamkowej najlepiej wykonywać w oparciu o iteracyjny algorytm Newtona (wystarczy 4 iteracji w przypadku 35-bitowych ułamków). Zgodnie z tym algorytmem obliczenie wartości $P = \mathrm{SQRT}(X)$ odbywa się w oparciu o początkowe przybliżenie do pierwiastka P^1 oraz kilkakrotne obliczenie poniższego wyrażenia (12)

$$P^{k+1} = (P^k + X / P^k) / 2, \tag{12}$$

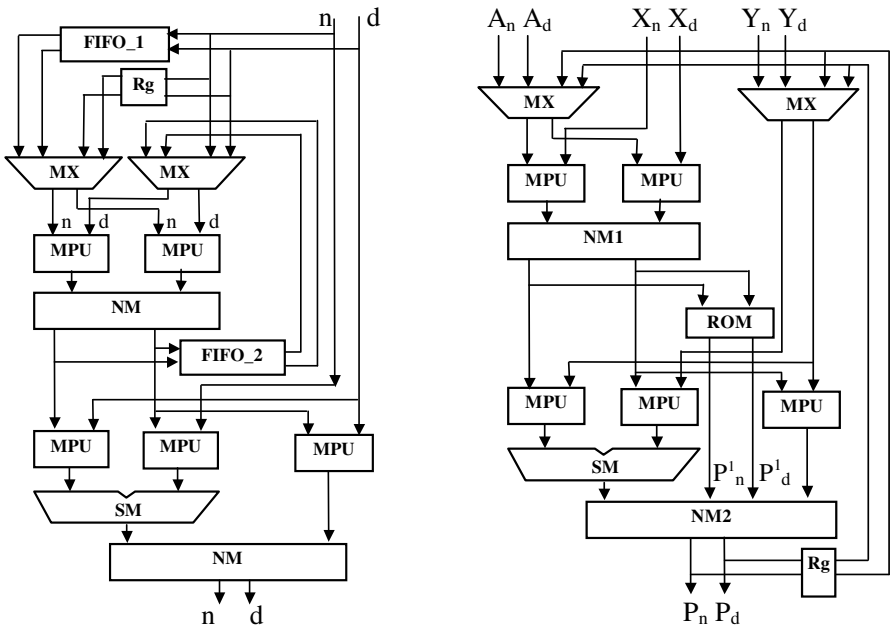
gdzie k oznacza numer iteracji, $k = 2,\ldots, 5$. Analiza wyrażenia (12) świadczy o tym, że do obliczenia pierwiastka kwadratowego wystarczy wykonać cztery operacje dzielenia z dodawaniem. Więc wyżej wymienione ALU RFA, wykorzystane w EP*i*, może również być wykorzystane jako ALU w ostatnim EP, jeśli wprowadzić do niego układ pamięci stałej ROM do przechowywania wartości P^1 oraz dodatkowe multipleksery służące do podawania wyników pośrednich P^k z wyjścia ALU na jego wejście. Ostatecznie blok ALU ostatniego elementu przetwarzającego EP*h* architektury S_5 przedstawiono na rys. 11b. W tym ALU licznik P_n^1 i mianownik P_d^1 są otrzymywane z bloku ROM, przy czym adres podawany na blok ROM jest formowany przez układ normalizacji NM1, jako różnica w liczbie bitów znaku znajdujących się w liczniku a mianowniku ułamka X przed pierwszym bitem znaczącym (odpowiedni fragment tego układu przedstawiono na rys. 2b). W ten sposób odczytywane z ROM wartości mają P_n^1, P_d^1 mają tylko 2-3 prawidłowe bity ostatecznego wyniku P, ale objętość bloku ROM wynosi tylko 16 komórek, a po wykonaniu 4 iteracji, dzięki wysokiej (kwadratowej) zbieżności metody Newtona, ostateczny wynik P będzie miał dokładność co najmniej 34 bitów.

Należy zaznaczyć, że ALU przedstawione na rys. 11b może być wykorzystane zarówno w elementach przetwarzających równoległych jednostek, jak i ALU samodzielnej, jednoprocesorowej, potokowej jednostki przetwarzającej realizującej praktycznie każdy z wszystkich wymienionych w niniejszej pracy algorytmów (poza chyba tylko algorytmami Givensa i QR) – należy tylko dołączyć do tego ALU blok pamięci RAM, 2 kolejki FIFO (jak to jest pokazane na rys. 11a) i zapewnić odpowiednie sterowanie.

Tab. 5 reprezentuje wyniki implementacji w układzie Virtex4 (xc4vSX35-12) modelu VHDL 35-bitowego 9-stopniowego ALU RFA przedstawionego na rys. 11a oraz wyniki implementacji w tym samym układzie 32-bitowego ALU zmiennoprzecinkowego złożonego z bloków operacyjnych, których opis został wygenerowany z modułów środowiska Xilinx Coregen: Multiplier 10.0, Divider 1.0 oraz Adder/Subtractor 7.0 (opis modułów jest dostępny pod adresem http://www.xilinx.com Xilinx Floating-point Operators v2.0 – DS335, Xilinx, January 18, 2006). Mimo to, że drugie ALU nie jest

zoptymalizowane, szczególnie pod względem liczby wykorzystanych bloków CLB, analiza danych z tab. 5 potwierdza tezę wykonawców projektu o mniejszej złożoności sprzętowej oraz wyższej wydajności ALU RFA w stosunku do ALU zmiennoprzecinkowego przy (jak będzie później pokazano) zachowaniu porównywalnej dokładności obliczeń.

a) b)



Rys. 11. Struktury wewnętrzna EP*i* (a) oraz EP*h* (b) równoległej jednostki przetwarzającej realizującej zmodyfikowany algorytm Cholesky'ego w arytmetyce RFA

Należy zaznaczyć, że podczas opracowania tego projektu członkowie zespołu badawczego wykorzystali oprogramowanie Xilinx PlanAhead Design Analysis Tools w celu zwiększenia maksymalnej częstotliwości działania jednostki. Ponadto, w trakcie implementacji opracowanych jednostek zawierających kolejki FIFO stwierdzono, że implementacja kolejki jako „Shift register" jest najszybsza (maks. częstotliwość jest bliska 550MHz), ale zajmuje więcej komórek slice układu FPGA. Natomiast implementacja jako „Dynamic Shift register" z wykorzystaniem prymitywów SRL16, SRL16E, SRLC16E zajmuje mniej komórek, jednak kolejka pracuje z częstotliwością około 350MHz, przy czym częstotliwość ta jest praktycznie nie zależy od długości kolejki (mniejszej od 256). W związku z tym do realizacji kolejek FIFO wybrano drugie rozwiązanie (gdyż nie jest to „wąskie gardło" jeżeli chodzi o częstotliwość całego układu).

Tab. 5. Wyniki porównania implementacji proponowanego ALU RFA z rys.11a z podobnym ALU złożonym z modułów IP-core firmy Xilinx w układzie FPGA Virtex 4

Parametry ALU z rys. 11a	**ALU RFA** (35-bitowe)	**ALU** złożone z modułów IP-core Xilinx 32-bitowe zmiennoprzecinkowe
Liczba wykorzystanych bloków CLB (slice), Liczba wykorzystanych bloków mnożących	811 20	7311 9
Liczba stopni w potoku przy wykonaniu: mnożenia z odejmowaniem dzielenia pierwiastkowania	 9 9 45	 12+10 56 56
Maksymalna częstotliwość działania, MHz	158(175*)	120

* po optymalizacji wykonanej w programie Xilinx PlanAhead Design Analysis Tools

4. Analiza dokładności obliczeń jednostek przetwarzających RFA

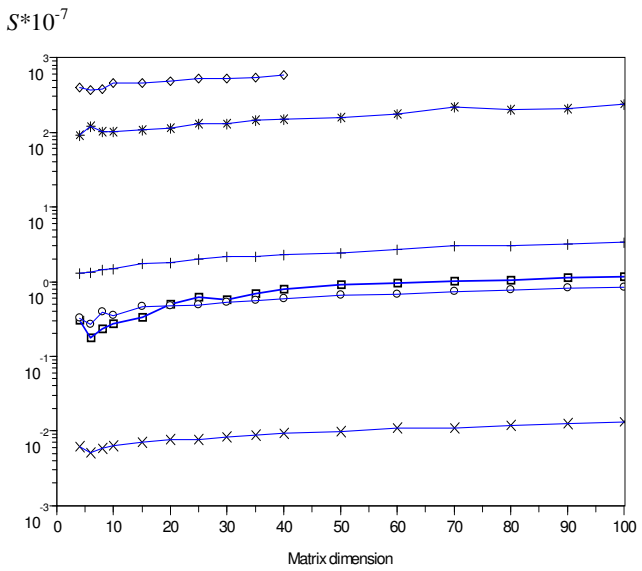
Jak już było wcześniej wspomniano, sprawdzenie poprawności działania opracowanych bloków operacyjnych RFA odbywało się najpierw poprzez weryfikację ich modeli VHDL w symulatorze ActiveHDL firmy Aldec. Poprawność i dokładność otrzymywanych wyników obliczeń była weryfikowana poprzez porównanie wyników otrzymywanych z modeli bloków RFA z wynikami z modeli funkcjonalnych VHDL analogicznych zmiennoprzecinkowych jednostek przetwarzających, w których wszystkie wykorzystane zmienne miały typ *Real*. Zgodnie ze standardem VHDL dane typu *Real* reprezentowane są w komputerze z maksymalną możliwą dokładnością, jaką może zapewnić sprzęt komputerowy (na którym przeprowadzane jest modelowanie). Ponieważ autorzy pracowali na komputerze typu PC z procesorem Intel Pentium 4, faktycznie otrzymane wyniki ze wszystkich modeli bloków ALU i jednostek przetwarzających były porównywane z wynikami zmiennoprzecinkowymi o precyzji *Double Extended* (80-bitowych) zgodnie ze standardem IEEE 754. Z tego powodu autorzy uważali wyniki produkowane przez te modele wynikami „bezbłędnymi". Wykonane porównania świadczą o tym, że dla ułamków 35-bitowych *a/b* błąd obliczeń wynosi $\delta \approx 5*10^{-9} \approx 2^{-28}$, dla ułamków 32-bitowych – $\delta \approx 5*10^{-8}$, dla ułamków 24-bitowych – $\delta \approx 5*10^{-5}$. Następnie wykonano było porównanie dokładności wyników rozkładu LL^T otrzymywanych z modelu ALU RFA z dokładnością wyników dwóch modeli ALU - działających na liczbach zmiennoprzecinkowych 32-bitowych i 64-bitowych (zgodnych z formatami IEEE 754 *Single* i *Double*) i opracowanych przez autorów w języku VHDL specjalnie do celów testowania. Macierz wejściowa (która powinna być macierzą symetryczną i dodatnio określoną) była tworzona w ten sposób, że elementy leżące na jej głównej przekątnej były generowane losowo z przedziału od 0,75 do 1,25 , natomiast

wartość elementów poniżej przekątnej była brana z w/w przedziału, którego granice były pomniejszone o odległość od przekątnej. Błąd był liczony jako odchylenie standardowe znormalizowane, tj. pierwiastek z sumy kwadratów różnic elementów macierzy wynikowej *L* dzielonych przez liczbę elementów macierzy *L* oraz wartość jednego z elementów macierzy, tj. zgodnie wzorem (20).

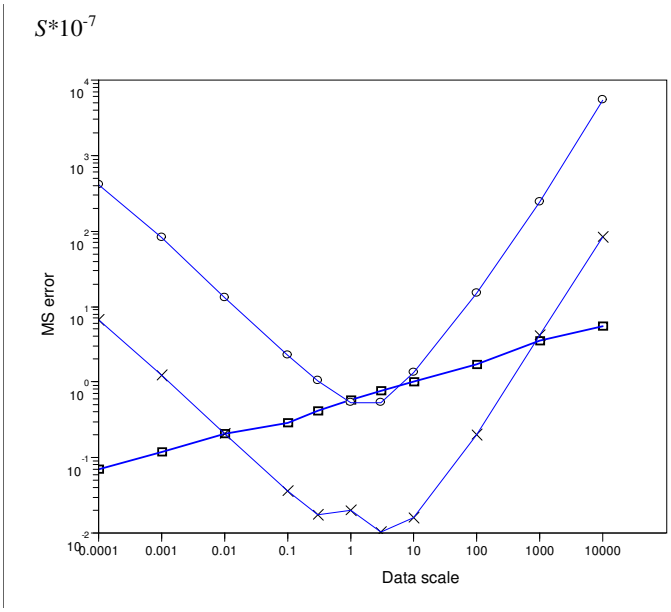
$$S = \sqrt{\frac{2 \cdot \sum_{i=1}^{N} \sum_{j=1}^{i} (l_{i,j}^{*} - l_{i,j})^2}{N \cdot (N+1) * l_{N,N}}} \tag{13}$$

Wyniki porównania świadczą o tym, że dla 32-bitowych danych zmienno-przecinkowych błąd obliczeń wynosi $\delta \approx 3*10^{-7}$, dla 64-bitowych danych zmiennoprzecinkowych $\delta \approx 7*10^{-16}$. Wynik ogólny jest następujący: dokładność obliczeń prowadzonych liczbach zmiennoprzecinkowych 32-bitowych jest porównywalna z dokładnością obliczeń przeprowadzonych na ułamkach 32- bitowych. Ten wynik potwierdzają również wykresy z rys. 12, otrzymane w pakiecie „Scilab" dla algorytmu Cholesky'ego, realizowanego na dobrze uwarunkowanych macierzach o różnych rozmiarach $N<100$, których elementy były przedstawione jako ułamki 16-bitowe (◊), 18-bitowe (*), 24-bitowe (+), 26-bitowe (o) i 32-bitowe (×) oraz jako liczby 32-bitowe zmiennoprzecinkowe (□).

Należy jednak zaznaczyć, że odróżnieniu od formatu zmiennoprzecinkowego, błędy obliczeń w RFA w dość dużym stopniu zależą od wielkości danych wejściowych. Jeśli scala wielkości danych wejściowych jest taka, że średnia ich wartość wynosi 1, to obliczenia w RFA mają największą dokładność. Np. 26-bitowe ułamki dają dokładność jednakową z 32-bitowymi danymi zmiennoprzecinkowymi (z 24-bitową mantysa i 8-bitowym wykładnikiem). Natomiast przy przetwarzaniu danych, których średnia wartość jest większa od jedynki w *M* razy, dokładność w arytmetyce RFA spada również w *M* razy, natomiast w arytmetyce zmienno-przecinkowej spada tylko ok. $\sqrt[4]{M}$ razy. To potwierdza wykres przedstawiony na rys. 12, gdzie (o) odpowiada 26-bitowym ułamkom, (×) - 32-bitowym ułamkom oraz (□) - liczbom 32-bitowym zmienno-przecinkowym. Ostatecznie można oszacować, że błąd obliczeń w RFA w przypadku algorytmu Cholesky'ego wynosi $2^{-n} MN^{1/3}$, gdzie *n* - liczba bitów reprezentujących licznik i mianownik ułamku, *N* – rozmiar macierzy danych wejściowych, a *M* - średnia wartość jej elementów.



Rys. 12. Porównanie błędów obliczeń dla ułamków 16, 18,24, 26 i 32-bitowych oraz na liczbach 32-bitowych zmiennoprzecinkowych dla algorytmu Cholesky'ego



Rys. 13. Zależność błędów obliczeń w algorytmie Cholesky'ego dla danych wejściowych różnej wielkości

W przypadku algorytmu redukcji wstecznej, badania dokładności obliczeń były prowadzone zgodnie ze wzorem (14)

$$S = \sqrt{\frac{\sum_{i=1}^{N} (x_i^* - x_i)^2}{N \cdot x_N}} . \tag{14}$$

Wyniki obliczenia wyrażenia (14) dla ułamków 32-bitowych i 32-bitowych danych zmiennoprzecinkowych przedstawiono w poniższej tablicy. Należy zaznaczyć, że zarówno elementy trójkątnych górnych macierzy danych wejściowych, jak i elementy wektorów wyrazów wolnych były generowane losowo. Analiza danych z tab. 6 świadczy o tym, że wraz z wzrostem rozmiaru macierzy błędy arytmetyki ułamkowej zbliżają się do błędów arytmetyki zmiennoprzecinkowej (dla danych 32-bitowych typu Single).

Tab. 6. Porównanie błędów obliczeń przeprowadzonych na 35-bitowych ułamkach oraz 32-bitowych danych zmiennoprzecinkowych

Rozmiar macierzy	**10**	**20**	**30**	**50**	**80**	**150**	**200**	**250**
Błąd dla RFA ($*10^{-7}$)	6,71	16,63	13,29	23,28	19,60	33,42	41,01	60,12
Błąd dla SINGLE ($*10^{-7}$)	1,66	4,33	4,51	11,59	14,55	29,64	38,27	54,09
Stosunek błędów RFA/SINGLE	4,04	3,84	2,95	2,01	1,35	1,13	1,07	1,11

Więc teza autorów o tym, że potokowe i równoległe jednostki przetwarzające działające w arytmetyce ułamkowej mogą być efektywnie stosowane (w przypadku ich realizacji w nowoczesnych układach FPGA) do realizacji w czasie rzeczywistym algorytmów algebry liniowej i cyfrowego przetwarzania sygnałów, w których wymagania dotyczące dokładności przeprowadzanych obliczeń uniemożliwiają stosowanie arytmetyki stało-przecinkowej (tj. zaleca się stosowanie arytmetyki zmiennoprzecinkowej o pojedynczej dokładności) została potwierdzona. Przy tym jednostki RFA cechują się mniejszą złożonością sprzętową i wyższą wydajnością w porównaniu do podobnych jednostek zmiennoprzecinkowych przy zachowaniu porównywalnej dokładności obliczeń.

5. Podsumowanie

W wyniku przeprowadzonych badań uzyskano następujące rezultaty:

1. Przeprowadzono dogłębna analizę budowy bloków DSP układów FPGA Virtex 4, Virtex 5 firmy Xilinx i Stratix II, Stratix III firmy Altera wraz z analizą zachowania oprogramowania Xilinx Foundation ISE oraz Altera Quartus podczas wykonania procedury syntezy logicznej jednostek arytmetyczno-logicznych (ALU).

Stwierdzono, że wykorzystanie wbudowanych w układy FPGA rodzin VirtexIIPro i Virtex4 rdzeni 32-bitowych stałoprzecinkowych procesorów RISC PowerPC 405 ma sens wyłącznie w celu realizacji algorytmów lub fragmentów algorytmów nieregularnych, gdzie zarówno równoległe, jak i potokowe przetwarzanie danych nie jest możliwe. Przeprowadzono również analizę publikacji naukowych z ostatnich lat poświęconych tematyce realizacji w nowoczesnych układach FPGA w/w jednostek (głównie ALU zmiennoprzecinkowych) oraz analiza zmienno-przecinkowych bloków operacyjnych proponowanych przez firmę Xilinx, których projekty (modele IP-core) są generowane przez oprogramowanie Xilinx Coregen.

2. Opracowano format przedstawienia danych w jednostkach przetwarzających działających arytmetyce RFA oraz procedurę normalizacji ułamków. Zbadano dokładność obliczeń przeprowadzanych na liczbach stałoprzecinkowych i ułamkach (na przykładzie generatora funkcji sin$x$) oraz określono źródła powstania błędów.
3. Opracowano struktury i syntezowalne modele VHDL bloków operacyjnych oraz kilku wielofunkcyjnych układów arytmetycznych (ALU) wykonujących w arytmetyce ułamkowej podstawowe operacje arytmetyczne: dodawania(odejmowania), mnożenia, mnożenia z akumulacją wyniku, dzielenia, pierwiastkowania, konwersji liczb binarnych (stało- i zmiennoprzecinkowych) z i do RFA. Opracowane projekty, przy implementacji w układach FPGA Firm Xilinx i Altera, cechują się znacznie mniejszą złożonością sprzętową i/lub znacznie większą wydajnością w porównaniu do analogicznych stało- i zmienno-przecinkowych bloków operacyjnych i ALU. W oparciu o w/w projekty opracowano program-generator IP-core modeli VHDL w/w układów arytmetycznych.
4. Stwierdzono, że w przypadku projektów równoległych i potokowych jednostek przetwarzających, na jakość projektu logicznego jednostki ma wpływ projekt architektury jednostki, a często nawet konkretne algorytmy, które jednostka ma realizować. Innymi słowy, budowa i zasoby sprzętowe układu reprogramowalnego FPGA, w którym jednostka ma być implementowana, warto brać pod uwagę już na etapie określenia zbioru algorytmów, które jednostka ma realizować, a tym bardziej na etapie projektowania strukturalnego. Czasami algorytmy wejściowe należy modyfikować (co zostało pokazane na przykładzie oryginalnej modyfikacji algorytmu Cholesky'ego-Banachewicza), aby uzyskać taką architekturę jednostki, której projekt logiczny (projekt RTL) przy implementacji w układzie FPGA efektywnie wykorzystywałby jego zasoby i miałby częstotliwość bliską do maksymalnej częstotliwości działania układu FPGA.
5. Dopracowano znaną metodę P. Quintona odwzorowania algorytmów regularnych w architektury równoległych jednostek przetwarzających systemów czasu rzeczywistego. Dopracowanie polegało m.in. na opracowaniu oryginalnej koncepcji lokalnego sterowania poszczególnymi elementami przetwarzającymi (EP)

równoległej jednostki, dostosowanej do jej realizacji w układach reprogramowalnych.

6. W oparciu o nowe metody opracowano i zbadano projekty VHDL potokowych i liniowej: gradientów sprzężonych, redukcji wstecznej, eliminacji Gaussa z wyborem elementu głównego, Gaussa-Jordana, Cholesky'ego, i in., które cechują się niewielkimi nakładami sprzętowymi (pozwalają na umieszczenie nawet do 25 elementów przetwarzających (EP) w jednym układzie FPGA), wysoką wydajnością i stopniem obciążenia EP oraz małą liczbą stopni w potoku.
7. Przeprowadzono implementację fizyczną potokowej jednostki przetwarzającej realizującej algorytm Cholesky'ego w płytkę prototypową ADS-XLX-V4FX-EVL12-G firmy AVNet. W celu implementacji w/w potokowej jednostki opracowany został projekt i model VHDL bloku sterowania.
8. Przeprowadzono sprawdzenie poprawności działania i dokładności obliczeń opracowanych bloków operacyjnych, jednostek ALU oraz potokowych jednostek przetwarzających. Dokładność otrzymywanych wyników obliczeń była weryfikowana poprzez porównanie wyników otrzymywanych z modeli bloków RFA z wynikami z modeli funkcjonalnych VHDL analogicznych zmiennoprzecinkowych jednostek przetwarzających, w których wszystkie wykorzystane zmienne miały typ *Real*, tj. maksymalną możliwą dokładność, jaką może zapewnić sprzęt komputerowy (na którym przeprowadzane jest modelowanie). Ponadto, dla dwóch algorytmów (Cholesky'ego i redukcji wstecznej) wykonano było porównanie dokładności wyników otrzymywanych z modelu ALU RFA z dokładnością wyników dwóch modeli ALU - działających na liczbach zmiennoprzecinkowych 32-bitowych i 64-bitowych (zgodnych z formatami IEEE 754 *Single* i *Double*) i opracowanych w języku VHDL specjalnie do celów testowania. Wynik ogólny jest następujący: dokładność obliczeń prowadzonych na liczbach zmiennoprzecinkowych 32-bitowych jest porównywalna z dokładnością obliczeń przeprowadzonych na ułamkach 32-35 bitowych.

Osiągnięte wyniki świadczą o tym, że zarówno cele, jak i główne tezy projektu zostały osiągnięte. W oparciu o otrzymane wyniki można stwierdzić, że potokowe i równoległe jednostki przetwarzające działające w arytmetyce ułamkowej mogą być efektywnie stosowane (w przypadku ich realizacji w nowoczesnych układach FPGA) do realizacji w czasie rzeczywistym algorytmów algebry liniowej i cyfrowego przetwarzania sygnałów, w których wymagania dotyczące dokładności przeprowadzanych obliczeń uniemożliwiają stosowanie arytmetyki stałoprzecinkowej (tj. zaleca się stosowanie arytmetyki zmiennoprzecinkowej o pojedynczej dokładności). Przy tym jednostki RFA cechują się mniejszą złożonością sprzętową i wyższą wydajnością w porównaniu do podobnych jednostek zmiennoprzecinkowych przy zachowaniu porównywalnej dokładności obliczeń.

Bibliografia

[1] E. Goetting. Introducing the new Virtex 4 FPGA Family. *Xcell jornal, Xilinx*, 2005, pp. 6-9.

[2] Altera FPGA Device Documentation. Informacja dostępna pod adresem WWW http://www.altera.com/literature/lit-index.html.

[3] K. Underwood. FPGAs vs CPUs: Trends in Peak Floating Point Performance**.**

[4] K. D. Underwood, K. S. Hemmert. Closing the Gap: CPU and FPGA Trends in sustained Floating Point BLAS Performance. *Proc. IEEE Symp. Field Programmable Custom Computing Machines*, *FCCM 2004*.

[5] M. Abramovici, C. Stroud, M. Emmert. Using Embedded FPGAs for SoC Yield Improvement. *Proc. Int.Conf. DAC'2002*, 2002, pp. 713-720.

[6] Dou, Y., Vassiliadis, S., Kuzmanov, G.K, Gaydadjiev, G.N.:64-bit Floating point FPGA Matrix Multiplication. ACM/SIGDA 13-th Int. Symp. on Field Programmable Gate Arrays, Feb., 2005, FPGA-2005, (2005), 86-95

[7] R. Scrofano, L. Zhuo, V. Pasana „Area-Efficient Aritmetic Expression Evaluation Rusing Deeply Pineplined Floating-Point Cores", *IEEE Trans. on VLSI Systems,* Vol.16, No2, 2008

[8] M. Beauchamp, S. Hauck, K. Underwood, K. Hemmert „Architectural Modifications to Enhance the Floating-Point Performance on FPGAs", *IEEE Trans. on VLSI Systems,* Vol.16, No2, 2008.

[9] N. Battson. Designing with the Virtex 4 XtremeDSP Slice. *Xcell Journal, Xilinx*, 2005, pp. 28-31.

[10] A. Sergyienko, O. Maslennikov, "Implementation of Givens QR Decomposition in FPGA". *Lecture Notes in Computer Science, Springer,* 2002, Vol.2328, pp. 453-459.

[11] O. Maslennikow, Ju. Shevtshenko, A. Sergyienko. Configurable microprocessor array for DSP application. *Lecture Notes in Computer Science, Springer,* 2004, Vol. 3019, s. 36–41.

[12] O. Maslennikow. Podstawy teorii zautomatyzowanego projektowania reprogramowalnych równoległych jednostek przetwarzających dla jednoukładowych systemów czasu rzeczywistego. (Monografia habilitacyjna). *Wyd. Uczelniane Politechniki Koszalińskiej*, Koszalin, 2004, 273 s.

[13] L. V. Fausett. Numerical Methods: Algorithms and Applications, *Prentice Hall*, 2003.

[14] T. Zieliński. Cyfrowe przetwarzanie sygnałów. Od teorii do zastosowań. WKŁ, Warszawa, 2006.

[15] O. Maslennikow, V. Lepekha, A. Sergyienko A. FPGA Implementation of the Conjugate Gradient Method. *Lecture Notes in Computer Science, Springer,* 2006, Vol.3911, pp. 526-533.

[16] Sergiyenko A., Maslennikow O. Structural Designing of IIR-Filters. *Proc. 7-th Int. Conf. Computer-Aided Design of Discrete Devices, CAD DD'2007,* Mińsk, Białoruś, 2007 (*in Russian*)
[17] B. K. P. Horn. Rational Arithmetic for Minicomputers. *Software – Practice and Experience,* Vol. 8, 1978, pp. 171-176.
[18] P. Kornerup, D. W. Matula. Finite-precision rational arithmetic: an arithmetic unit. *IEEE Transactions on Computers*, C-32, 1983, pp. 378-388.
[19] Maslennikow O., Maslennikowa N., Pawłowski P., Khadzhynov W., Sergiyenko A. Realizacja w FPGA jednostek operacyjnych działających w arytmetyce ułamkowej. *Elektronika*, nr 11, 2007, s. 34 – 36.
[20] Maslennikow O., Ratuszniak P., Khadzhynov W., Pawłowski P., Berezowski R., Sergiyenko A. Osobliwości stosowania arytmetyki ułamkowej w nowoczesnych układach FPGA. *Elektronika*, nr 11, 2008r, 200-2003.
[21] Maslennikow O., Ratuszniak P., Sergiyenko A. Generator opisów VHDL bloków operacyjnych działających w arytmetyce ułamkowej. *Pomiary, Automatyka, Kontrola,* nr 8, 2008 r., s.514-516.
[22] Maslennikow O., Lepekha V., Sergyienko A., Wyrzykowski R. Cholesky LL^T - algorithm Implementation in FPGA-based Processor. *Lecture Notes in Computer Science, Springer,* 2008, Vol.4967, pp. 137-147.

Application of the fractional arithmetic in the reprogrammable processing units of the single-chip systems