Rozdział 1

SPRZĘTOWO WSPOMAGANE WYZNACZANIE RDZENIA W STRUKTURACH FPGA

Maciej Kopczyński*

Streszczenie W rozdziale zaprezentowano kilka wybranych sprzętowych metod obliczania rdzenia bazujących na układach programowalnych FPGA wspieranych jednostkami obliczeniowymi CPU typu softcore. Stworzone zostały moduły sprzętowe, które należą do układów kombinacyjnych i sekwencyjnych. Zaproponowane architektury zostały przetestowane w rzeczywistym układzie FPGA na dwóch różnych zbiorach danych o różnych licznościach. Zostały również wykonane badania porównawcze w tożsamej implementacji programowej uruchamianej na komputerze PC. Otrzymane wyniki w sposób jednoznaczny wskazują na znaczące zmniejszenie czasu realizacji obliczeń dla modułów sprzętowych w porównaniu z funkcjonalnie taką samą implementacją programową. W rozdziale tym przytoczone są niezbędne definicje z zakresu zbiorów przybliżonych, pokazane są główne założenia architektury poszczególnych rozwiązań sprzętowych wraz z przykładem ich działania, jak również jest opisana charakterystyka zbiorów danych przed, jak i po przekształceniach niezbędnych do realizacji obliczeń przez moduły wykonawcze w strukturach FPGA.

Słowa kluczowe: zbiory przybliżone, rdzeń, FPGA, implementacja sprzętowa

Wprowadzenie

Jednym z największych problemów związanych z tworzeniem systemów wspomagania decyzji jest czas przetwarzania danych wejściowych, ze szczególnym uwzględnieniem dużych zbiorów danych (ang. *Big data*). *Big data* to termin odnoszący się do dużych, różnorodnych i zmiennych zbiorów danych, których przetwarzanie i analiza są trudne, ale jednocześnie cenne, ponieważ mogą prowadzić do zdobywania nowej wiedzy. W praktycznych zastosowaniach pojęcie dużego zbioru danych jest względne i oznacza sytuację, w której zbiór nie może być przetworzony przy użyciu prostych i powszechnie dostępnych metod (Marz i Warren, 2015). W zależności

* Wydział Informatyki, Politechnika Białostocka, Wiejska 45A, 15-351 Białystok, m.kopczynski@pb.edu.pl

DOI 10.24427/978-83-66391-58-1_1

od celu wybranej metody i złożoności algorytmu może to oznaczać rozmiar danych liczony w megabajtach, gigabajtach lub terabajtach.

Jednym z przykładów tak ogromnych zbiorów danych są dane odczytywane z czujników, np. z linii produkcyjnej. W każdej części takiej linii znajduje się bardzo wiele takich czujników, które cały czas dostarczają dane, co tworzy ogromny strumień danych, które muszą zostać przetworzone, np. w celu predykcji przyszłych awarii, a tym samym unikania przestojów linii z możliwie dużym wyprzedzeniem.

Powstało wiele metod przetwarzania takich zbiorów danych i wydobywania z nich wiedzy, jak również minimalizowania ilości danych, które muszą być przetwarzane. Jedną z nich jest teoria zbiorów przybliżonych (ang. *Rough sets*), pozwalająca między innymi na stosunkowo łatwe ograniczenie liczby atrybutów (kolumn) w wejściowych zbiorach danych. Główne założenia dotyczące metod zbiorów przybliżonych opisane są przez Pawlak i Skowron (2007). Pod kątem ograniczania liczby atrybutów, w metodach zbiorów przybliżonych zdefiniowane są dwa ważne pojęcia: redukt (ang. *Reduct*) oraz rdzeń (ang. *Core*). Redukty to zbiory atrybutów, które zawierają kluczowe informacje niezbędne do prawidłowej klasyfikacji danych. Rdzenie to zbiory atrybutów, których nie można w żadnych okolicznościach usunąć z początkowego zbioru danych. Innymi słowy, atrybuty istniejące w rdzeniu muszą również istnieć we wszystkich reduktach.

Teoria zbiorów przybliżonych, opracowana w latach osiemdziesiątych XX wieku przez prof. Z. Pawlaka, jest użytecznym narzędziem do analizy danych. Dlatego w naukowych i komercyjnych narzędziach przetwarzania danych zaimplementowano wiele algorytmów zbiorów przybliżonych. Jednak wraz ze wzrostem rozmiaru danych pojawia się problem z wydajnością i rosnącym czasem ich przetwarzania.

Jednym z rozwiązań jest opracowanie algorytmów o mniejszej złożoności obliczeniowej. Kolejnym jest przetwarzanie równoległe dużych zbiorów danych. Algorytmy do obliczania rdzenia i reduktów oparte na modelu programowania rozproszonego MapReduce można znaleźć w pracy Czołombitko i Stepaniuk (2015).

Innym podejściem do przetwarzania dużych zbiorów danych są sprzętowe implementacje istniejących algorytmów. Tego typu podejście można zrealizować za pomocą układów FPGA (ang. *Field Programmable Gate Arrays*). Jest to grupa cyfrowych układów scalonych, których funkcjonalność może być zmieniona w każdym momencie i jest oparta głównie na definiowanych przez inżyniera funkcjach boolowskich. Dlatego też mogą być one używane do wspomagania obliczeń w zakresie zbiorów przybliżonych, ze szczególnym uwzględnieniem równoległego przetwarzania danych. Należy jednak zaznaczyć, że wykorzystanie układów FPGA w zakresie implementacji algorytmów programowych stwarza trudność w aspekcie dostosowania tych algorytmów do charakteru przetwarzanych danych oraz ich analizy w celu identyfikacji tych elementów, które mogą być realizowane równolegle w strukturach programowalnych. Innymi słowy, implementacja sprzętowa algorytmu programowego nie zapewnia elastyczności tożsamej z funkcjonalnym odpowiednikiem rozwiązania zrealizowanego w języku zarówno wysokiego, jak i niskiego poziomu.

Obecnie istnieje kilka implementacji sprzętowych wybranych metod zbiorów przybliżonych. Idea przykładowego procesora generującego reguły decyzyjne z tablic decyzyjnych została opisana w pracy Pawlak (2004). Lewis, Perkowski i Jozwiak (1999) przedstawili architekturę teoretycznego procesora bazującego na sieciach komórkowych opisanych przez Muraszkiewicz i Rybinski (1993). Kanasugi i Yokoyama (2001) opracowali koncepcję urządzenia sprzętowego zdolnego do minimalizacji dużych funkcji logicznych utworzonych na podstawie macierzy rozróżnialności. Bardziej szczegółowe podsumowanie istniejących sprzętowych implementacji metod zbioru przybliżonego można znaleźć w opracowaniu Kopczynski i Stepaniuk (2013) i w pracy Tiwari i Kothari (2014). Wyniki poprzednich badań autora dotyczą idei procesora dla metod zbiorów przybliżonych (Stepaniuk, Kopczynski i Grzes, 2013), obliczania reduktu wspomaganego sprzętowo (Kopczynski, Grzes i Stepaniuk, 2014a), obliczania rdzenia przy użyciu rozwiązania opartego na FPGA (Kopczynski, Grzes i Stepaniuk, 2014b), jednostki sprzętowej do przetwarzania dużych zbiorów danych (Kopczynski, Grzes i Stepaniuk, 2015) czy też dwuetapowego algorytmu znajdowania reduktów (Grzes i Kopczynski, 2019).

W podrozdziale 1.1 przedstawiono najważniejsze definicje związane z operacjami obliczania rdzenia, jak również omówiono wykorzystywane algorytmy. Ponadto, w podrozdziale zawarto przykład działania algorytmu oraz opisano dane wykorzystywane w badaniach eksperymentalnych. W podrozdziale 1.2 skupiono się na opisie architektury sprzętowej opracowanych modułów wspomagających obliczanie rdzenia, jak również zawarto przykład działania takiego modułu zaprezentowany na poziomie danych binarnych. W podrozdziale 1.3 opisano rzeczywiste środowisko testowe oraz przedstawiono wyniki badań eksperymentalnych. W *Podsumowaniu* zaprezentowano najważniejsze wnioski oraz określono przyszłe możliwości optymalizacji modułów sprzętowych w kontekście badań naukowych.

1.1 Podstawowe definicje

1.1.1 Rdzeń w ujęciu zbiorów przybliżonych

Niech $DT = (U, A \cup \{d\})$ będzie tablicą decyzyjną, gdzie U jest zbiorem obiektów, A jest zbiorem atrybutów warunkowych, a d jest atrybutem decyzyjnym. W tablicy decyzyjnej niektóre atrybuty warunkowe ze zbioru A mogą być usunięte (innymi słowy: są zbędne), jednak ich usunięcie nie może pogorszyć jakości klasyfikacji. Zbiór $C \subseteq A$ wszystkich niezbędnych atrybutów warunkowych nazywany jest rdzeniem. Żaden z jego elementów nie może zostać usunięty bez wpływu na moc klasyfikacji wszystkich atrybutów warunkowych. Aby obliczyć rdzeń, można użyć macierzy rozróżnialności $[DM(x,y)]_{x,y \in U}$, gdzie $DM(x,y) = \{a \in A : a(x) \neq a(y) \text{ and } d(x) \neq d(y)\}$.

Rdzeń to zbiór wszystkich jednoelementowych komórek macierzy rozróżnialności, tj. $CORE = \bigcup_{x,y \in U, cardinality(DM(x,y))=1} DM(x,y)$. Znacznie bardziej szczegółowy opis definicji rdzenia można znaleźć w artykule (Pawlak i Skowron, 2007) lub w książce (Stepaniuk, 2008).

1.1.2 Podstawowy algorytm obliczania rdzenia

Jeden z najpopularniejszych algorytmów obliczania rdzenia wykorzystujący macierz rozróżnialności jest przedstawiony w postaci pseudokodu w tym podrozdziale. Największym jednak ograniczeniem tego algorytmu dla dużych zbiorów danych jest potrzeba przechowywania macierzy rozróżnialności *DM* jako dwuwymiarowej tablicy o rozmiarze $|U|x|U|$. Poszczególne wersje sprzętowe algorytmów obliczania rdzenia wykorzystują elementy algorytmu podstawowego, jednakże algorytmy te zostały w znacznym stopniu zmodyfikowane i dostosowane do rozwiązania sprzętowego.

Algorytm 1.1 Algorytm obliczania rdzenia oparty na definicji

INPUT: macierz rozróżnialności *DM*
OUTPUT: rdzeń *C*

```
1: C ← ∅
2: for x ∈ U do
3:     for y ∈ U do
4:         if |DM(x,y)| = 1 then
5:             C ← C ∪ DM(x,y)
6:         end if
7:     end for
8: end for
```

Dane wejściowe do algorytmu opartego na definicji stanowi macierz rozróżnialności *DM*, a wyjście to rdzeń *C*. Rdzeń jest inicjalizowany jako pusty zbiór w linii 1. Dwie pętle w liniach 2 i 3 iterują po wszystkich obiektach (oznaczonych jako U) w macierzy dostrzegalności. Instrukcja warunkowa w linii 4 sprawdza, czy komórka macierzy zawiera tylko jeden atrybut. Jeśli tak, to ten atrybut jest dodawany do rdzenia *C*.

1.1.3 Sprzętowy algorytm obliczania rdzenia dla dużych zbiorów danych

Główna koncepcja algorytmu CORE-HIDM (***CORE H**ardware **I**ndirect **D**iscernibility **M**atrix*) jest oparta na właściwości singletonu, czyli komórki z macierzy rozróżnialności składającej się tylko z jedynego atrybutu, co jest bezpośrednio związane z definicją przytoczoną na początku tego podrozdziału. Jak zostało wcześniej wspomniane, rozwiązania bazującego na definicji nie można uruchomić na sprzęcie dla większych zbiorów danych z powodu ograniczeń zasobów układu FPGA. Nie jest możliwe przechowywanie dużego zestawu danych w module sprzętowym. Dlatego też zaproponowany został algorytm dedykowany do implementacji sprzętowej - CORE-HIDM. Jest on również bazą do opracowania kilku jego wariantów dedykowanych dla różnych optymalizacji w zakresie przetwarzania równoległego. Jednym z takich wariantów sprzętowych zaprezentowanych w tym rozdziale jest implementacja opisana jako CORE-PHIDM (***CORE P**arallel **H**ardware **I**ndirect **D**iscernibility **M**atrix*). Szczegóły dotyczące tego rozwiązania podane zostaną w kolejnych podrozdziałach poświęconych omówieniu architektury modułów wykonawczych. Główną ideą algorytmu CORE-HIDM jest podzielenie całego zbioru danych na równe części przechowywane w dwóch niezależnych jednostkach pamięci RAM układu FPGA. Części te są następnie przetwarzane przez jednostkę obliczeniową (algorytm CORE-HIDM).

Dane wejściowe do algorytmu CORE-HIDM stanowi tablica decyzyjna DT, a wyjście rdzeń C. W pierwszym kroku rdzeń C jest inicjalizowany jako pusty zbiór. Dwie pętle w liniach 2 i 4 są odpowiedzialne za wybór fragmentu wejściowej tablicy decyzyjnej. Tabela decyzyjna DT jest podzielona na m części, z których każda ma rozmiar n obiektów. Linie 3 i 5 odpowiadają za załadowanie wybranych fragmentów zbioru danych do pamięci RAM zaimplementowanych w układzie FPGA. Dwie pętle w liniach 6 i 7 pobierają do porównania kolejne obiekty z fragmentu tablicy decyzyjnej. Linia 8 porównuje wartość atrybutu decyzyjnego dwóch obiektów x i y. Jeśli obiekty należą do różnych klas decyzyjnych, to wykonywana jest pozostała część algorytmu. Zmienna *count* odpowiedzialna za przechowywanie liczby różnic wartości atrybutów warunkowych pomiędzy obiektami x i y jest ustawiana na wartość 0 w linii 9. Pętla w linii 10 iteruje po zbiorze atrybutów warunkowych A. Wartości atrybutu warunkowego a są porównywane między obiektami x i y w linii 11. W przypadku różnicy wartości, zmienna *count* jest zwiększana, a atrybut a jest zapisywany w zmiennej *candidate*. Po zakończeniu pętli, atrybut w zmiennej *candidate* jest dodawany do rdzenia C, jeśli zmienna *count* jest równa 1 i atrybut a nie znajduje się w rdzeniu (wiersze od 16 do 18).

Algorytm 1.2 Algorytm CORE-HIDM

INPUT: tablica decyzyjna $DT = (U, A \cup \{d\})$, dwie liczby naturalne $n, m > 0$
OUTPUT: rdzeń C

```
1:  C ← ∅
2:  for cnt_1 ← 0 to m − 1 do
3:      RAM1 ← {x ∈ U : x_{cnt_1·n} to x_{(cnt_1+1)·n−1}}
4:      for cnt_2 ← cnt_1 to m − 1 do
5:          RAM2 ← {x ∈ U : x_{cnt_2·n} to x_{(cnt_2+1)·n−1}}
6:          for x ∈ RAM1 do
7:              for y ∈ RAM2 do
8:                  if d(x) ≠ d(y) then
9:                      count ← 0
10:                     for a ∈ A do
11:                         if a(x) ≠ a(y) then
12:                             count ← count + 1
13:                             candidate ← a
14:                         end if
15:                     end for
16:                     if count = 1 and candidate ∉ C then
17:                         C ← C ∪ {candidate}
18:                     end if
19:                 end if
20:             end for
21:         end for
22:     end for
23: end for
```

1.1.4 Przykład obliczania rdzenia na bazie definicji

W tabeli 1.1 przedstawiono przykładową tablicę decyzyjną składającą się z 4 atrybutów warunkowych i jednego atrybutu decyzyjnego. Zbiór danych opisuje decyzje związane z aktywnością fizyczną (ang. *activity*) na podstawie 4 opisowych cech pogody: warunki (ang. *outlook*), temperatura (ang. *temp*), wilgotność (ang. *humidity*) i wiatr (ang. *windy*).

Przed obliczeniem rdzenia należy najpierw stworzyć macierz rozróżnialności. Używając definicji przedstawionych w podrozdziale 1.1.1 została obliczona macierz rozróżnialności *DM* dla tablicy decyzyjnej pokazanej w tabeli 1.1. Macierz jest symetryczna względem swojej przekątnej, więc obliczany jest tylko dolny trójkąt. Dodatkowo, aby ograniczyć rozmiar tabeli, obliczono macierz rozróżnialności *DM* tylko dla 6 pierwszych obiektów ze zbioru danych. Macierz wynikowa pokazana jest w tabeli 1.2.

Tabela 1.1: Przykładowy zbiór danych składający się z 4 atrybutów warunkowych i 1 atrybutu decyzyjnego

ID	outlook	temp	humidity	windy	activity
1	sunnny	high	high	no	no
2	sunnny	high	high	yes	no
3	cloudy	high	high	no	yes
4	sunnny	low	high	no	yes
5	sunnny	high	normal	no	yes
6	sunnny	high	normal	yes	no
7	cloudy	high	normal	yes	yes
8	sunnny	low	high	no	no
9	sunnny	high	normal	no	yes
10	sunnny	low	normal	no	yes
11	sunnny	low	normal	yes	yes
12	cloudy	low	high	yes	yes

Zgodnie z algorytmem obliczania rdzenia opisanym w podrozdziale 1.1.2, poniżej przedstawiono przykład wyznaczania rdzenia na podstawie macierzy rozróżnialności *DM* pokazanej w tabeli 1.2. Na początku rdzeń C jest pusty. Dwie główne pętle wykonują pierwszą iterację przechodząc przez wszystkie komórki macierzy rozróżnialności. Dla każdej z nich sprawdzany jest warunek dodania atrybutu do rdzenia. Pierwszy taki przypadek występuje dla komórki na przecięciu obiektów 1 i 3. Komórka ta zawiera tylko jeden atrybut $\{o\}$. Ten atrybut nie istnieje w rdzeniu, więc jest do niego dodawany, czyli rdzeń $C = \{o\}$. Po wykonaniu pozostałych kroków algorytmu, końcowy rdzeń to $C = \{o,t,h,w\}$, co odpowiada $C = \{outlook, temp, humidity, windy\}$.

Tabela 1.2: Macierz rozróżnialności dla pierwszych 6 obiektów ze zbioru przykładowego

ID	**1**	**2**	**3**	**4**	**5**	**6**
1	$\emptyset$					
2	$\emptyset$	$\emptyset$				
3	$\{o\}$	$\{o,w\}$	$\emptyset$			
4	$\{t\}$	$\{t,w\}$	$\emptyset$	$\emptyset$		
5	$\{h\}$	$\{h,w\}$	$\emptyset$	$\emptyset$	$\emptyset$	
6	$\emptyset$	$\emptyset$	$\{o,h,w\}$	$\{t,h,w\}$	$\{w\}$	$\emptyset$

1.1.5 Dane do badań eksperymentalnych

Przeprowadzone eksperymenty wykorzystywały dwa zbiory danych: Poker Hand (autorstwa Roberta Cattrala i Franza Oppachera) oraz zbiór danych o dzieciach z cukrzycą insulinozależną typu 1 (autorstwa Jarosława Stepaniuka).

Pierwszy zestaw danych pozyskano z repozytorium UCI Machine Learning (Lichman, 2013). Każdy z 1 000 000 rekordów jest przykładem zestawu składającego się z pięciu kart do gry dobranych ze standardowej talii 52 kart. Każda karta jest opisana za pomocą dwóch atrybutów (kolor i ranga), co daje łącznie 10 atrybutów warunkowych. Atrybut decyzyjny opisuje "układ pokerowy".

Cukrzyca insulinozależna jest przewlekłą chorobą charakteryzującą się niezdolnością do wytwarzania wystarczającej ilości insuliny do wydajnego przetwarzania węglowodanów, tłuszczu i białek. Leczenie wymaga wstrzyknięć insuliny. Dwanaście atrybutów warunkowych obejmujących wyniki badań fizycznych i laboratoryjnych oraz jeden atrybut decyzyjny (mikroalbuminuria) opisuje bazę danych wykorzystywaną w eksperymentach. Zbiór danych składa się ze 107 przypadków. Baza danych jest zawarta jest w artykule Stepaniuk (2000).

Baza Poker Hand została wykorzystana do stworzenia mniejszych zbiorów danych przez wybranie określonej liczby wierszy oryginalnego zbioru danych z zachowaniem rozkładu klas decyzyjnych. Baza danych dotycząca cukrzycy została wykorzystana do wygenerowania większych zbiorów przez powielenie wierszy z oryginalnego zbioru danych.

Utworzone zbiory danych musiały zostać przekształcone do wersji binarnej. Wartości liczbowe zostały zdyskretyzowane, a wartość każdego atrybutu zakodowana przy użyciu czterech bitów dla obu zbiorów danych. Każdy obiekt został opisany na 44 bitach w przypadku zbioru Poker Hand i 52 bitach w przypadku bazy cukrzyków. Opisy obiektów musiały zostać rozszerzone do 64-bitowych słów, wypełniając nieużywane atrybuty binarnym "0" celem uproszczenia architektury przetwarzania danych po stronie układu FPGA.

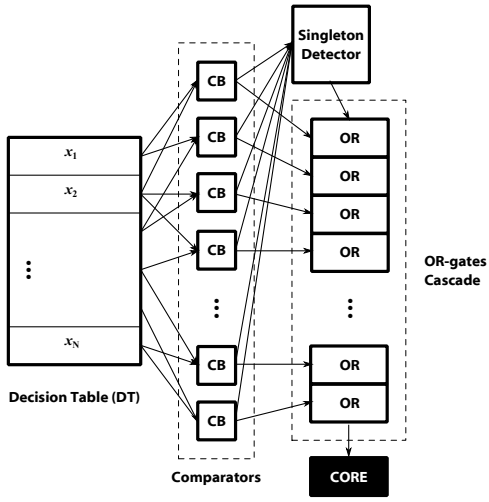
1.2 Implementacje sprzętowe

1.2.1 Obliczanie rdzenia jako układ kombinacyjny - "OR-cascade"

Pierwszym przedstawionym układem implementacji sprzętowej algorytmu obliczania rdzenia jest układ kombinacyjny bazujący bezpośrednio na definicji i metodzie obliczania rdzenia przedstawionej w podrozdziale 1.1.2.

Architekturę pierwszego rozwiązania pokazano na rysunku 1.1. Wejściem do tego bloku jest pełna tablica decyzyjna. Moduł obliczeniowy składa się z trzech bloków funkcjonalnych:

1. **Comparators CB** - blok komparatorów, które obliczają wpisy macierzy rozróżnialności. Każdy komparator ma dwa wejścia, które są podłączone do dwóch obiektów tablicy decyzyjnej.
2. **OR-gates Cascade** - blok bramek OR połączonych kaskadowo. Każda bramka wyznacza logiczną operację OR na dwóch wartościach: jednej z poprzedniej bramki w kaskadzie i drugiej z komparatora. Wynik przesyłany jest do następnej bramki OR i ostatecznie do rejestru **CORE**, który przechowuje wynik obliczeń. Wybrana bramka OR może zostać zablokowana przez wyjście z **Singleton Detector**.
3. **Singleton Detector** - blok sprawdzający, czy wpis w macierzy rozróżnialności jest singletonem, tj. składa się tylko z jednej logicznej "1". Wyjścia tego bloku są połączone z kaskadą bramek OR.

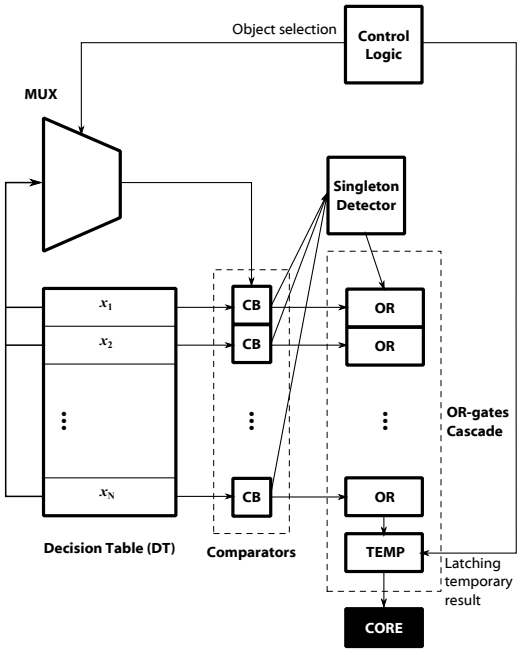


Rysunek 1.1: Diagram blokowy modułu obliczania rdzenia "OR-cascade" w wariancie kombinacyjnym

Komórki macierzy rozróżnialności są obliczane przez komparatory bardzo szybko, głównie ze względu na prostotę architektury każdego komparatora z bloku CB, jak również kombinacyjny charakter układu wykonawczego. Następnie wszystkie wejścia trafiają do kaskady bramek OR. Czas niezbędny do obliczenia wyniku zależy od wielkości tablicy decyzyjnej. Ostatnia bramka w kaskadzie przechowuje wynik obliczeń w rejestrze **CORE**.

1.2.2 Obliczanie rdzenia jako układ sekwencyjny - "Mixed"

Głównym ograniczeniem rozwiązania opartego na układzie kombinacyjnym jest mały rozmiar tablicy decyzyjnej, która może być przetwarzana. Im większy rozmiar danych, tym więcej zasobów układu FPGA jest wykorzystywane. Efektywnie, tego rodzaju rozwiązanie może przetwarzać zbiory o licznościach setek obiektów. Ze względu na ograniczenie zasobów struktury FPGA, zaproponowany został wariant modułu sprzętowego obliczania rdzenia oparty o układ sekwencyjny, który jest znacznie bardziej optymalny pod względem zużycia zasobów. Architekturę tego rozwiązania pokazano na rysunku 1.2.



Rysunek 1.2: Diagram blokowy modułu obliczania rdzenia "Mixed" w wariancie sekwencyjnym

Główne różnice w stosunku do poprzedniego rozwiązania to:

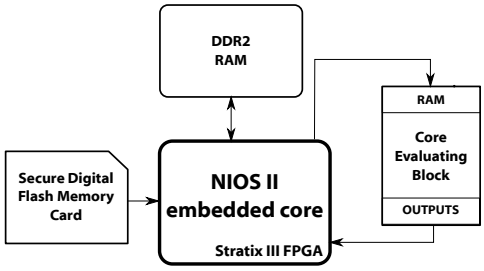
1. Liczba komparatorów **CB** jest równa liczbie obiektów w tablicy decyzyjnej, a nie liczbie elementów w macierzy rozróżnialności.
2. Liczba bramek OR w kaskadzie jest równa liczbie obiektów w tablicy decyzyjnej, a nie liczbie elementów w macierzy rozróżnialności.
3. Multiplekser **MUX** w każdym cyklu zegara taktującego wybiera kolejny obiekt z tablicy decyzyjnej i umieszcza go w komparatorach.

4. Blok **Control Blok** zlicza obiekty w tablicy decyzyjnej i zatrzaskuje wartości w rejestrze tymczasowym **TEMP**.

Ten wariant sprzętowy jest znacznie mniejszy pod względem wykorzystania zasobów niż poprzedni, ale wymaga więcej czasu na obliczenie końcowego wyniku. Liczba cykli potrzebnych do zakończenia obliczeń jest równa liczbie obiektów w tabeli decyzyjnej.

1.2.3 Obliczanie rdzenia dla dużych zbiorów danych - "CORE-HIDM"

Pomimo zmniejszenia liczby wykorzystywanych zasobów układu FPGA, układ "Mixed" nie może zostać w prosty sposób zwielokrotniony, co pozwoliłoby na przetwarzanie większej liczby obiektów. Głównym problemem są ograniczenia w wielkości bloków pamięciowych w strukturze FPGA. Z tego względu opracowana została implementacja sprzętowa wspomagana procesorem typu softcore oraz zewnętrzną pamięcią RAM. Ogólną architekturę tego rozwiązania pokazano na rysunku 1.3.



Rysunek 1.3: Ogólny diagram blokowy modułu obliczania rdzenia dla dużych zbiorów danych

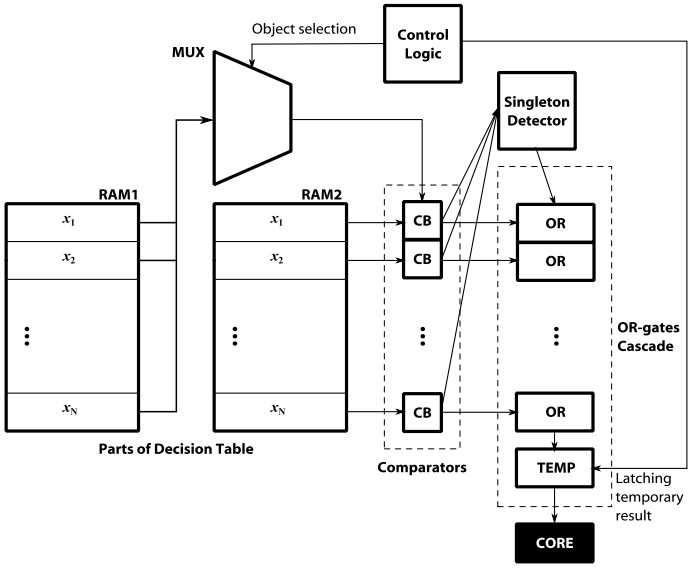
Procesor typu softcore NIOS II odpowiada za następujące operacje:

1. Kontroluje proces dzielenia na częśco dużej wejściowej tablicy decyzyjnej.
2. Nadzoruje podstawowe bloki funkcjonalne modułów sprzętowego obliczania rdzenia.
3. Przeładowuje dane pomiędzy wewnętrzną i zewnętrzną pamięcią RAM.
4. Przetwarza wyniki zwrócone przez sprzętowe moduły obliczania rdzenia.
5. Wykonuje podstawowe operacje na zbiorach zapisanych w rejestrach.

Wybrany procesor to NIOS II. Jest to jednostka typu softcore dostarczona przez firmę Altera dla jej układów typu FPGA. Jest to w pełni funkcjonalny, 32-bitowy procesor typu RISC ze wsparciem dla rozwiązań zwiększających moc obliczeniową

(np. potoki wielostopniowe, dynamiczne przewidywanie rozgałęzień, oddzielne pamięci podręczne dla instrukcji i dla danych, jednostki MMU, MPU i inne). Pamięć DDR2 przechowuje dużą tablicę decyzyjną. Karta SD to tymczasowe rozwiązanie do przenoszenia danych z komputera PC do rozwiązania opartego na FPGA. Dane z karty SD są w całości kopiowane do pamięci DDR2 na początku procesu obliczeniowego. Każda kolejna część danych gotowych do przetworzenia jest przechowywana we wbudowanych pamięciach układu FPGA (MLAB, M9k i M144k). Bloki MLAB to synchroniczne, dwuportowe pamięci z konfigurowalną organizacją 32x20 lub 64x10. Dwuportowe pamięci mogą być odczytywane i zapisywane jednocześnie, co przyśpiesza wykonywane operacje. Bloki M9k i M144k są również synchronicznymi, dwuportowymi blokami pamięci z wieloma możliwymi konfigurowalnymi organizacjami. Cechą charakterystyczną tych bloków jest możliwość przygotowania pamięci zdolnej do przechowywania niemal każdego rodzaju obiektów (opisanych jako słowa bitowe).

Szczegółowa architektura modułu sprzętowego obliczania rdzenia "CORE-HIDM" pokazana jest na rysunku 1.4.



Rysunek 1.4: Szczegółowy diagram blokowy modułu obliczania rdzenia "CORE-HIDM"

Należy zaznaczyć, że główne założenia modułu sprzętowego "CORE-HIDM" są rozwinięciem układu "Mixed", przez co wcześniej omówione elementy składowe występujące w obydwu rozwiązaniach nie będą ponownie omawiane. Wejścia modułu to:

- **RAM1** i **RAM2** - pamięci przechowujące fragmenty tablicy decyzyjnej,

- **Attribute Mask Register (AMR)** - rejestr przechowujący wartość odpowiadającą atrybutom warunkowymdo przetworzenia,
- **clock** - sygnał zegarowy układu,
- **reset** - sygnał resetujący do ustawienia początkowych wartości rejestrów wewnętrznych w module sprzętowym.

Wyjścia bloku CORE-HIDM to wartość obliczonego rdzenia w rejestrze *CORE* i sygnał *ready* informujący o zakończeniu obliczeń wykonanych przez blok.

Celem przetwarzania dużych zbiorów danych do rozwiązania dodano dwa bloki szybkich statycznych pamięci RAM, które zostały utworzone jako instancje dedykowanych bloków FPGA (MLAB, M9k i M144k). Obie pamięci danych używane w module (RAM1 i RAM2) przechowują fragmenty wejściowej tablicy decyzyjnej. Na początku algorytmu zawierają tę samą część tabeli decyzyjnej. Gdy obiekty z pamięci RAM2 zostaną porównane ze wszystkimi obiektami z pamięci RAM1, to pamięć RAM2 jest przeładowywana następną częścią tablicy decyzyjnej, aż do momentu, gdy w tablicy decyzyjnej nie będzie żadnych nieporównywanych elementów. Następnie do pamięci RAM1 i RAM2 ładowany jest drugi fragment zbioru danych i cały proces jest kontynuowany do momentu przetworzenia całego zbioru wejściowego.

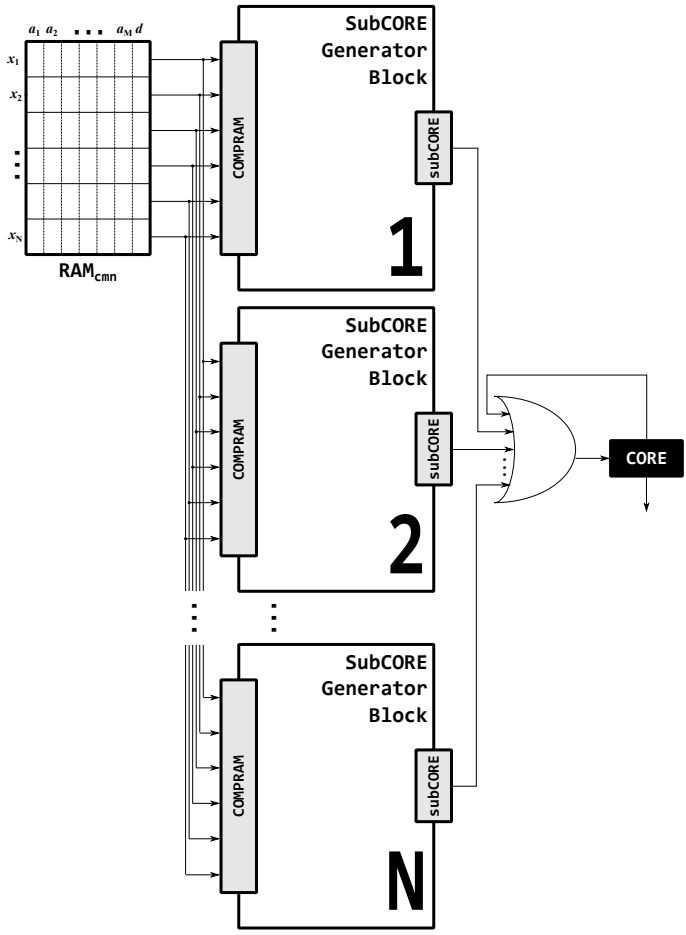
1.2.4 Obliczanie rdzenia dla dużych zbiorów danych z przetwarzaniem równoległym - "CORE-PHIDM"

Kolejnym rozwinięciem rozwiązania wspomagającego w sposób sprzętowy obliczanie rdzenia i dedykowanym dla dużych zbiorów danych jest układ "CORE-PHIDM". W tym przypadku zwielokrotniony został blok sprzętowy obliczania rdzenia, który przestawiony został na rysunku 1.4. Kluczowe jednak było wprowadzenie zmian związanych z organizacją pamięci na dane oraz synchronizacją pracy poszczególnych modułów sprzętowych. Architektura modułu obliczania rdzenia "CORE-PHIDM" pokazana jest na rysunku 1.5

System składa się z następujących bloków:

1. RAM_{cmn} - fragment tablicy decyzyjnej, która jest porównywana z częściami tablicy decyzyjnej przechowywanymi w powielonych blokach *SubCORE*.
2. *SubCORE* - blok, który dokonuje porównania dwóch części tablicy decyzyjnej i oblicza fragment rdzenia. Jego działanie jest tożsame do bloku CORE-HIDM.

Pamięć RAM_{cmn} jest wypełniana fragmentami tablicy decyzyjnej. Każdy blok *SubCORE* zawiera również kolejne fragmenty tablicy decyzyjnej. *SubCORE* oblicza częściową wartość rdzenia, która ostatecznie jest łączona w wynikowy rdzeń *C*. Powielenie bloków *SubCORE* umożliwia zrównoleglenie prowadzonych obliczeń. Całość procesu obliczeniowego jest nadzorowana przez procesor NIOS II.



Rysunek 1.5: Diagram blokowy modułu obliczania rdzenia “CORE-PHIDM”

1.2.5 Przykład działania dla modułu ”CORE-HIDM”

Przykład działania wyjaśnia sposób przetwarzania danych przez układ CORE-HIDM będący główną sprzętową jednostką obliczeniową dla generowania rdzenia. Dla przejrzystości pokazano tylko najważniejsze operacje związane z przepływem danych. Jednostka sprzętowa wymaga przekształcenia zbioru danych do wersji binarnej. Tablica decyzyjna z tabeli 1.1 po transformacji binarnej jest zaprezentowana w tabeli 1.3.

Wartości na wejściach modułu CORE-HIDM to:

- RAM_1 = 00111 01111 10110 10101 1001101011 11010 00101 10011 10001 11001 11100,

Tabela 1.3: Przykładowy zbinaryzowany zbiór danych składający się z 4 atrybutów warunkowych i 1 atrybutu decyzyjnego

ID	outlook	temp	humidity	windy	activity
1	1	1	1	0	0
2	1	1	1	1	0
3	0	1	1	0	1
4	1	0	1	0	1
5	1	1	0	0	1
6	1	1	0	1	0
7	0	1	0	1	1
8	1	0	1	0	0
9	1	1	0	0	1
10	1	0	0	0	1
11	1	0	0	1	1
12	0	0	1	1	1

- RAM_2 = 00111 01111 10110 10101 1001101011 11010 00101 10011 10001 11001 11100,

co odpowiada danym w przykładowej tablicy decyzyjnej po binaryzacji. Najbardziej znaczący bit (MSB) opisuje atrybut decyzyjny *activity*, zaś najmniej znaczący bit (LSB) odpowiada za atrybut warunkowy *outlook*.

Muliplexer MUX w pierwszym cyklu zegara wybiera z pamięci RAM_1 obiekt ID_1, który reprezentowany jest przez słowo 00111. Obiekt ten jest porównywany ze wszystkimi pozostałymi obiektami w pamięci RAM_2 przez komparatory CB. Obliczona wartość odpowiada pierwszej kolumnie macierzy rozróżnialności. Słowo binarne utworzone przez komparatory to 0000 0000 0001 0010 0100 0000 1101 0000 0100 0110 1110 1011. Każda z 4-bitowych podsekwencji jest przekazywana do bramek OR. Jednocześnie całe słowo utworzone przez komparatory jest kierowane do bloku SD, który wykrywa 4-bitowe podciągi zawierające tylko jedną logiczną "1". Wynik utworzony przez SD to 001110001000. Każdy bit utworzonej wartości steruje pojedynczą bramką OR w kaskadzie. MSB jest podłączony do pierwszej bramki OR. Wartości wejść i wyjść każdej z bramek to:

- $IN_{1CB} = 0000$; $IN_{1PREV} = 0000$; $IN_{1SD} = 0$; $OUT_1 = 0000$,
- $IN_{2CB} = 0000$; $IN_{2PREV} = 0000$; $IN_{2SD} = 0$; $OUT_2 = 0000$,
- $IN_{3CB} = 0001$; $IN_{3PREV} = 0000$; $IN_{3SD} = 1$; $OUT_3 = 0001$,
- $IN_{4CB} = 0010$; $IN_{4PREV} = 0001$; $IN_{4SD} = 1$; $OUT_4 = 0011$,
- $IN_{5CB} = 0100$; $IN_{5PREV} = 0011$; $IN_{5SD} = 1$; $OUT_5 = 0111$,
- $IN_{6CB} = 0000$; $IN_{6PREV} = 0111$; $IN_{6SD} = 0$; $OUT_6 = 0111$,
- $IN_{7CB} = 1101$; $IN_{7PREV} = 0111$; $IN_{7SD} = 0$; $OUT_7 = 0111$,
- $IN_{8CB} = 0000$; $IN_{8PREV} = 0111$; $IN_{8SD} = 0$; $OUT_8 = 0111$,
- $IN_{9CB} = 0100$; $IN_{9PREV} = 0111$; $IN_{9SD} = 1$; $OUT_9 = 0111$,

- $IN_{10CB} = 0110$; $IN_{10PREV} = 0111$; $IN_{10SD} = 0$; $OUT_{10} = 0111$,
- $IN_{11CB} = 1110$; $IN_{11PREV} = 0111$; $IN_{11SD} = 0$; $OUT_{11} = 0111$,
- $IN_{12CB} = 1011$; $IN_{12PREV} = 0111$; $IN_{12SD} = 0$; $OUT_{12} = 0111$.

Wejście IN_{CB} odpowiada wartości obliczonej przez dany komparator z bloku *CB*. Wejście IN_{PREV} jest połączone z wyjściem OUT poprzedniej bramki OR. Wyjście ostatniej bramki jest przekazywane do rejestru pośredniego *TEMP*.

W kolejnych cyklach zegarowych moduł CORE-HIDM przetwarza pozostałe obiekty w RAM_1 porównując je z pozostałą częścią zbioru danych w RAM_2. Ostatecznie utworzony rdzeń to $C = 1111$, co odpowiada rdzeniowi $C = \{outlook, temp, humidity, windy\}$.

1.3 Wyniki eksperymentalne

1.3.1 Środowisko testowe

Dla wszystkich opisanych w tym rozdziale rozwiązań sprzętowych zostały opracowane tożsame funkcjonalnie implementacje programowe w języku C. Wyniki dotyczące czasu działania wersji programowej uzyskano przy użyciu komputera PC wyposażonego w 8 GB pamięci RAM i 4-rdzeniowy procesor Intel Core i7 3632QM o maksymalnej częstotliwości taktowania 3,2 GHz w trybie Turbo w systemie operacyjnym Windows 10. Kod źródłowy aplikacji został skompilowany przy użyciu kompilatora GNU GCC w wersji 9.2.

Do projektowania, kompilacji, syntezy i weryfikacji symulacyjnej implementacji sprzętowych w języku VHDL wykorzystano środowisko Quartus II 13.1. Zsyntetyzowane bloki sprzętowe zostały uruchomione na płycie deweloperskiej TeraSIC DE- 3 wyposażonej w układ FPGA typu Stratix III EP3SL150F1152C2N. W układzie tym dostępnych jest 113 600 bloków LE (ang. *Logical Elements*). Źródłem zegara układu FPGA działającego z częstotliwością 50 MHz był oscylator kwarcowy na płycie rozwojowej.

Procesor typu softcore NIOS II, jak również większość elementów systemu wbudowanego, zostały utworzone za pomocą narzędzia Qsys 13.1. Oprogramowanie dla NIOS II zostało zaimplementowane w języku C przy użyciu NIOS II Software Build Tools for Eclipse IDE.

Pomiary czasowe dla krótkich odcinków czasu uzyskano za pomocą oscyloskopu LeCroy waveSurfer 104MXs-B (pasmo 1 GHz, próbkowanie 10 GS/s). Do pomiarów dłuższych czasów działania używano sprzętowych jednostek pomiaru czasu tworzonych wewnątrz układu FPGA.

Należy zauważyć, że zegar komputera PC jest $\frac{clk_{PC}}{clk_{FPGA}} = 64$ razy szybszy niż źródło zegara płyty deweloperskiej.

Wszystkie obliczenia zostały przeprowadzone przy użyciu zestawów danych opisanych w podrozdziale 1.1.5. Dane zostały wstępnie przetworzone na komputerze PC pod kątem transformacji binarnej i ewentualnej dyskretyzacji.

1.3.2 Wyniki dla małych zbiorów danych

W poniższym podrozdziale przedstawiono wyniki czasowe oraz zajętość zasobów FPGA dla układów w wariancie "OR-cascade" oraz "Mixed". Ze względu na ograniczenia zasobów układu FPGA, rozwiązania te nie mogły przetworzyć zbiorów danych większych niż 110 obiektów, dlatego też ograniczone są do jednego zbioru danych, czyli bazy cukrzyków.

Tabela 1.4 przedstawia wyniki czasowe otrzymane dla implementacji sprzętowej w wersji "OR-cascade" (t_H) oraz tożsamej implementacji programowej (t_S).

Tabela 1.4: Porównanie czasu obliczania rdzenia (moduł "OR-cascade")

Obiekty —	**Program -** t_S [μs]	**Sprzęt -** t_H [μs]	$\frac{t_S}{t_H}$ —
15	112.28	0.0084	13 366
30	420.22	0.0104	40 405
45	983.42	0.0216	45 528
60	1 736.67	0.0310	56 021
90	4 074.80	0.0584	69 773
107	5 990.00	0.0683	87 701

Tabela 1.5 przedstawia wyniki czasowe otrzymane dla implementacji sprzętowej w wersji "Mixed" (t_H) oraz tożsamej implementacji programowej (t_S).

Tabela 1.5: Porównanie czasu obliczania rdzenia (moduł "Mixed")

Obiekty —	**Program -** t_S [μs]	**Sprzęt -** t_H [μs]	$\frac{t_S}{t_H}$ —
15	112.28	0.57	196
30	420.22	1.17	359
45	983.42	1.77	555
60	1 736.67	2.40	723
90	4 074.80	3.58	1 138
107	5 990.00	4.26	1 406

Tabela 1.6 przedstawia wykorzystanie zasobów układu FPGA wyrażone jako liczba zajętych bloków LE uzyskana w procesie syntezy.

Tabela 1.6: Zajętość struktury układu FPGA wyrażona w LE

Obiekty	"Or-cascade"	"Mixed"
15	1 374	1 252
30	5 675	1 979
45	12 666	3 418
60	22 863	3 541
90	52 339	5 335
107	74 916	7 171

Przedstawione wyniki dla układów w wariancie "OR-cascade" oraz "Mixed" wskazują na duży wzrost szybkości przetwarzania danych. Czas wykonania dla modułu sprzętowego w porównaniu z implementacją programową jest co najmniej o 1 rząd wielkości krótszy dla jednostki sprzętowej w wariancie sekwencyjnym "Mixed", co pokazano w tabeli 1.5 w kolumnach $\frac{t_S}{t_H}$. Należy podkreślić, że przyśpieszenie rośnie wraz ze wzrostem rozmiaru przetwarzanych danych. Dla kombinacyjnych wersji jednostek sprzętowych czasy te są o 2 do 3 rzędów wielkości krótsze, jednak zużywają one znacznie więcej zasobów niż rozwiązania sekwencyjne. W przypadku praktycznych rozwiązań preferowane są jednostki sekwencyjne, głównie ze względu na fakt, że zajmują stosunkowo niewielkie zasoby układu FPGA. Należy podkreślić, że nawet implementacja sekwencyjna z 64-krotnie wolniejszym zegarem taktującym układ FPGA jest znacznie szybsza, niż jej programowy odpowiednik uruchamiany na komputerze PC.

1.3.3 Wyniki dla dużych zbiorów danych

W poniższym podrozdziale przedstawiono wyniki czasowe oraz zajętość zasobów FPGA dla układów w wariancie "CORE-HIDM" i "CORE-PHIDM". Ze względu na ich sekwencyjny charakter oraz wykorzystanie zewnętrznych zasobów pamięciowych, rozwiązania te mogą przetwarzać zbiory danych liczące miliony obiektów.

Tabela 1.7 przedstawia czas obliczenia rdzenia dla rozwiązania sprzętowego (t_H) i programowego (t_S) dla wariantu "CORE-HIDM", który również jest zgodny z modułem "CORE-PHIDM" w przypadku pojedynczej instancji bloku *SubCORE*. Badania przeprowadzone zostały dla dwóch zbiorów danych o różnych licznościach. Wykorzystane skróty w liczbie obiektów to: $k = 10^3$, $M = 10^6$.

Tabela 1.7: Porównanie czasu obliczania rdzenia (moduł “CORE-HIDM” oraz “CORE-PHIDM” z 1 instancją *SubCORE*)

Obiekty —	**Sprzęt -** t_H [s]	**Program -** t_S [s]	$\frac{t_S}{t_H}$ —
Zbiór Poker Hand			
1k	0.003	0.033	10.875
2.5k	0.013	0.143	11.119
5k	0.055	0.603	10.951
10k	0.207	2.410	11.623
25k	1.225	14.721	12.015
50k	4.710	58.726	12.469
100k	21.737	237.942	10.946
250k	130.947	1 515.449	11.573
500k	506.225	6 092.916	12.036
1M	1 850.523	24 313.094	13.138
Zbiór cukrzyków			
1k	0.003	0.018	5.911
2.5k	0.013	0.078	6.044
5k	0.055	0.328	5.953
10k	0.207	1.31	6.318
25k	1.225	8.002	6.531
50k	4.710	34.216	7.265
100k	21.737	135.309	6.225
250k	130.947	861.781	6.581
500k	506.225	3 464.821	6.844
1M	1 850.523	13 825.976	7.471

Tabele 1.8 oraz 1.9 przedstawiają czas obliczenia rdzenia dla rozwiązania sprzętowego (t_H) i programowego (t_S) dla wariantu “CORE-PHIDM” odpowiednio w przypadku podwójnej oraz poczwórnej instancji bloku *SubCORE*.

Wykorzystanie zasobów FPGA jest stałe dla danej konfiguracji modułu sprzętowego i jest niezależne od rozmiaru danych wejściowych, ponieważ zbiory danych są podzielone na równe fragmenty, które są przetwarzane przez moduł w danej konfiguracji. Wykorzystanie zasobów układu FPGA w zależności od modułu:

- 21 562 LE dla “CORE-HIDM” oraz “CORE-PHIDM” z 1 instancją modułu *SubCORE*,
- 33 234 LE dla “CORE-PHIDM” z 2 instancjami modułu *SubCORE*,
- 45 668 LE dla “CORE-PHIDM” z 4 instancjami modułu *SubCORE*.

Powyższe liczby obejmują również zasoby zużywane przez procesor NIOS II.

Przedstawione wyniki czasowe wskazują na duży wzrost szybkości przetwarzania danych dla wszystkich prezentowanych rozwiązań. Czas obliczenia rdzenia dla

Tabela 1.8: Porównanie czasu obliczania rdzenia (moduł "CORE-PHIDM" z 2 instancjami *SubCORE*)

Obiekty —	**Sprzęt -** t_H [s]	**Program -** t_S [s]	$\frac{t_S}{t_H}$ —
Zbiór Poker Hand			
1k	0.002	0.033	17.770
2.5k	0.008	0.143	18.169
5k	0.0340	0.603	17.894
10k	0.127	2.410	18.991
25k	0.750	14.721	19.632
50k	2.882	58.726	20.375
100k	13.303	237.942	17.886
250k	80.139	1 515.449	18.910
500k	309.807	6 092.916	19.667
1M	1 132.511	24 313.094	21.468
Zbiór cukrzyków			
1k	0.002	0.018	9.659
2.5k	0.008	0.078	9.876
5k	0.034	0.328	9.727
10k	0.127	1.31	10.323
25k	0.750	8.002	10.672
50k	2.882	34.216	11.871
100k	13.303	135.309	10.171
250k	80.139	861.781	10.754
500k	309.807	3 464.821	11.184
1M	1 132.511	13 825.976	12.208

modułu sprzętowego w porównaniu z funkcjonalnie odpowiadającą implementacją programową wynosi od 5 (1 instancja modułu *SubCORE*) do 37 (4 instancje modułu *SubCORE*) razy szybciej. Jeśli uwzględni się różnicę częstotliwości zegara między komputerem PC a układem FPGA, wyniki te są jeszcze lepsze i średni współczynnik przyśpieszenia wynosi od 378 (1 instancja) do 2 376 (4 instancje). Współczynnik przyśpieszenia jest praktycznie stały dla danej konfiguracji modułu *SubCORE* i jest podobny dla wszystkich rozmiarów przetwarzanych zbiorów danych.

Czasy przetwarzania wariantu sprzętowego dla danego algorytmu w odniesieniu do obu zbiorów danych są takie same, ponieważ szerokość bitowa pojedynczego obiektu ze zbioru danych nie ma znaczenia dla czasu obliczeń przy założeniu, że słowo bitowe mieści się określonych granicach pamięci. Przetwarzanie porcji danych zajmuje taki sam czas, ponieważ moduł sprzętowy zawsze wykonuje ten sam rodzaj operacji. Dotyczy to wszystkich konfiguracji przedstawionych w tym podrozdziale modułów sprzętowego obliczania rdzenia.

Tabela 1.9: Porównanie czasu obliczania rdzenia (moduł "CORE-PHIDM" z 4 instancjami *SubCORE*)

Obiekty —	**Sprzęt - t_H** [s]	**Program - t_S** [s]	$\frac{t_S}{t_H}$ —
Zbiór Poker Hand			
1k	0.001	0.033	30.741
2.5k	0.005	0.143	31.432
5k	0.019	0.603	30.957
10k	0.073	2.410	32.855
25k	0.433	14.721	33.963
50k	1.666	58.726	35.249
100k	7.690	237.942	30.944
250k	46.323	1 515.449	32.715
500k	179.079	6 092.916	34.024
1M	654.631	24 313.094	37.140
Zbiór cukrzyków			
1k	0.001	0.018	16.710
2.5k	0.005	0.078	17.086
5k	0.019	0.328	16.828
10k	0.073	1.31	17.859
25k	0.433	8.002	18.462
50k	1.666	34.216	20.537
100k	7.690	135.309	17.596
250k	46.323	861.781	18.604
500k	179.079	3 464.821	19.348
1M	654.631	13 825.976	21.120

Należy zauważyć, że średni współczynnik przyśpieszenia związany z liczbą instancji modułu *SubCORE* dla "CORE-PHIDM" nie jest liniowy i wynosi:

- 1.634 w przypadku 2 instancji modułu *SubCORE*,
- 2.827 w przypadku 4 instancji modułu *SubCORE*.

Zmniejszający się współczynnik przyśpieszenia jest związany z narzutem obliczeniowym procesora NIOS II niezbędnym do przenoszenia danych binarnych z głównej pamięci RAM do pamięci RAM_n każdego z modułów *SubCORE*.

Podsumowanie

Implementacje sprzętowe algorytmów obliczania rdzenia dają duże przyśpieszenie w porównaniu z rozwiązaniami programowymi. Tego rodzaju podejście może być

jednym z kluczowych kierunków tworzenia skalowalnych systemów decyzyjnych w rozwiązaniach wymagających działania w czasie rzeczywistym.

Sprzętowe jednostki obliczające rdzenie nie zostały zoptymalizowane pod kątem wydajności. Czas przetwarzania można znacznie skrócić zwiększając częstotliwość zegara taktującego układu FPGA i dodatkowo reagując na dwa zbocza sygnału zegarowego. Przedstawione rozwiązania sprzętowe, w szczególności w wariancie sekwencyjnym, są łatwo skalowalne. Powielenie bloków obliczeniowych znacznie poprawiło szybkość przetwarzania. Należy zauważyć, że 4 instancje modułów w ostatnim przedstawionym wariancie architektury zajmują tylko około 50% średniej wielkości układu FPGA. Współcześnie dostępne największe na rynku układy posiadają ponad 10-krotnie większą liczbę bloków LE.

Dalsze badania będą koncentrować się na weryfikacji różnych rozmiarów modułów obliczeniowych oraz ich optymalizacji pod kątem wydajności. Prace badawcze będą się również skupiały na optymalizacji transferu danych między tablicą decyzyjną, a jednostkami obliczeniowymi. W chwili obecnej, czas niezbędny na transfer danych jest pomijany w wynikach empirycznych (kopiowanie danych z karty SD do pamięci RAM), jednak w przypadkach rzeczywistych nie można go pominąć. Istnieją jednak interfejsy transmisji danych oferujące przepływności na poziomie dziesiątek Gb/s (np. standard JESD204B zapewniający transmisje na poziomie 12,5 Gb/s, czy też USB 3.2 Gen 2x2 pozwalający na transfer danych z prędkością 20 Gb/s), które są znacznie szybsze, niż czas niezbędny do analizy danych przez opisane w tym rozdziale algorytmy sprzętowe.

W kolejnych badaniach należy również wziąć pod uwagę rodzaj przetwarzanych danych. Przedstawione obecnie rozwiązania są odpowiednie dla spójnych zbiorów danych. Zaprojektowane moduły nie obsługują poprawnie zbiorów danych z brakującymi wartościami.

Bibliografia

Czołombitko, M., i Stepaniuk, J. (2015). Generating core based on discernibility measure and MapReduce. W: M. Kryszkiewicz, S. Bandyopadhyay, H. Rybinski i S. K. Pal (red.), *Pattern recognition and machine intelligence - proceedings of 6th international conference,* 9124, Springer, 367-376.

Grzes, T., i Kopczynski, M. (2019). Hardware implementation on field programmable gate array of two-stage algorithm for rough set reduct generation. *Lecture Notes in Computer Science*, *11499*, Springer, 495-506.

Kanasugi, A., i Yokoyama, A. (2001). A basic design for rough set processor. W: *Proceedings of the 15th annual conference of japanese society for artificial intelligence,* 406–412.

Kopczynski, M., Grzes, T. i Stepaniuk, J. (2014a). FPGA in rough-granular compu-

ting: Reduct generation. W: *Proceedings of the 2014 IEEE/WCI/ACM International Joint Conferences on Web Intelligence,* 2, IEEE Computer Society, 364-370.

Kopczynski, M., Grzes, T. i Stepaniuk, J. (2014b). Generating core in rough set theory: Design and implementation on FPGA. *Lecture Notes in Computer Science*, *8537*, Springer, 209-216.

Kopczynski, M., Grzes, T. i Stepaniuk, J. (2015). Computation of cores in big datasets: An fpga approach. *Lecture Notes in Computer Science*, *9436*, Springer, 153-163.

Kopczynski, M., i Stepaniuk, J. (2013). Hardware implementations of rough set methods in programmable logic devices. W: A. Skowron i Z. Suraj (red.), *Rough sets and intelligent systems – professor zdzislaw pawlak in memoriam, intelligent systems reference library* 43, Springer, 309-321.

Lewis, T., Perkowski, M. i Jozwiak, L. (1999). Learning in hardware: Architecture and implementation of an fpga-based rough set machine. W: *Proceedings of the euromicro, 25th euromicro conference,* 1, 1326-1334.

Lichman, M. (2013). *Uci machine learning repository*. http://archive.ics.uci.edu/ml: Irvine, CA: University of California, School of Information and Computer Science.

Marz, N., i Warren, J. (2015). *Big data: Principles and best practices of scalable realtime data systems*. Greenwich: Manning Publications Co.

Muraszkiewicz, M., i Rybinski, H. (1993). Towards a parallel rough sets computer. W: W. Ziarko (red.), *Rough sets, fuzzy sets and knowledge discovery, proceedings of the international workshop on rough sets and knowledge discovery,* Springer, 434-443.

Pawlak, Z. (2004). Elementary rough set granules: Toward a rough set processor. W: S. K. Pal, L. Polkowski i A. Skowron (red.), *Rough-neurocomputing: Techniques for computing with words,* Springer-Verlag, 5-14.

Pawlak, Z., i Skowron, A. (2007). Rudiments of rough sets. *Information Sciences*, *177*, IOS Press, 3-27.

Stepaniuk, J. (2000). Knowledge discovery by application of rough set models. W: *Rough set methods and applications. new developments in knowledge discovery,information systems,* 56, Springer, 137-233.

Stepaniuk, J. (2008). *Rough-granular computing in knowledge discovery and data mining*. Berlin: Springer.

Stepaniuk, J., Kopczynski, M. i Grzes, T. (2013). The first step toward processor for rough set methods. *Fundamenta Informaticae*, *127*, IOS Press, 429-443.

Tiwari, K. S., i Kothari, A. G. (2014). Design and implementation of rough set algorithms on FPGA: A survey. *International Journal of Advanced Research in Artificial Intelligence*, *3*, The Science and Information Organization, 14-23.