Rozdział 2

EFEKTYWNE METODY WYZNACZANIA REDUKTÓW OPARTE NA UKŁADACH FPGA

Mateusz Choromański*

Streszczenie Redukcja atrybutów jest istotnym zagadnieniem w dziedzinie eksploracji danych, którego istotność wzrasta wraz z powiększającymi się danymi, które muszą być poddane analizie. Wyznaczanie minimalnych reduktów, czyli minimalnego zestawu atrybutów, znacznie przyśpiesza analizę danych, jednak wraz z ich wzrostem: ilościowym i objętościowym, pojawiła się także potrzeba maksymalnego przyśpieszenia wyliczania wspomnianych reduktów. Zagadnienie akceleracji obliczeń służących zredukowaniu liczby istotnych atrybutów stało się obiektem licznych badań i eksperymentów, ze względu na ogromną liczbę obliczeń oraz innych operacji, którym poddawane są dane. W ich wyniku powstawały coraz to wydajniejsze algorytmy. Oprócz opracowywania nowych rozwiązań, podejmowane są próby uzyskania dodatkowych przyśpieszeń z użyciem przetwarzania wielowątkowego i obliczeń równoległych. Znaczne przyśpieszenia są uzyskiwane także dzięki wykorzystaniu nowoczesnych rozwiązań sprzętowych. Do najpowszechniejszych należą współczesne cyfrowe układy programowalne z rodziny FPGA. Niniejszy rozdział jest przeglądem najefektywniejszych metod przyśpieszających wyliczanie minimalnego zestawu atrybutów koniecznych do prawidłowej analizy danych. Zostały w nim przedstawione autorskie podejścia sprzętowe wykorzystujące programowalny układ cyfrowy FPGA.

Słowa kluczowe: zbiory przybliżone, redukcja atrybutów, implementacja sprzętowa, układy FPGA

Wprowadzenie

Redukcja atrybutów (Pawlak, 1991) jest bardziej istotnym zadaniem wstępnego przetwarzania danych niż kiedykolwiek wcześniej w historii. W erze Big Data redukcja

* Wydział Informatyki, Politechnika Białostocka, Wiejska 45A, 15-351 Białystok, m.choromanski@pb.edu.pl
DOI 10.24427/978-83-66391-58-1_2

zbioru danych, nawet o jedną cechę/atrybut, może znacząco wpłynąć na rozmiar danych, a co za tym idzie na czas potrzebny do ich przetworzenia. Pomimo faktu, że redukcja jak największej liczby atrybutów dużego zbioru danych może być sama w sobie kosztowna, korzyści płynące z dalszego przetwarzania znacznie mniejszego zbioru mogą być nieporównywalnie większe.

Zadanie redukcji atrybutów było szeroko badane z teoretycznego i praktycznego punktu widzenia (np. Chen, Zhao, Zhang, Yang i Zhang (2012); Degang, Changzhong i Qinghua (2007); X. Hu i Cercone (1995); Kryszkiewicz (1998, 2001); Skowron i Rauszer (1992); Stepaniuk (2008); R. Swiniarski (2001); R. W. Swiniarski i Skowron (2003); Thi i Giang (2013); X. Zhang, Mei, Chen i Li (2013)) w teorii zbiorów przybliżonych (Pawlak, 1991; Pawlak i Skowron, 2007), gdzie jest postrzegane jako narzędzie matematyczne do radzenia sobie z niespójnymi danymi. Pomimo istnienia bogatej literatury, redukcja atrybutów jest nadal rozwojowym tematem; nieustannie odkrywane są nowe obszary jej zastosowania oraz nowe metody jej doskonalenia (np. Czolombitko i Stepaniuk (2016); Dong, Sun i Yang (2016); Y. Jiang i Yu (2016); Jing, Yunliang i Yong (2017); Li i Yang (2016); Liu, Hua i Chen (2017); Teng i in. (2016)).

Poniższe podrozdziały zawierają przegląd podejść znajdowania reduktów opartych na rozwiązaniach sprzętowych.

Programowe podejścia do wyznaczania reduktów

W teorii zbiorów przybliżonych opracowano różnorodne algorytmy służące do znajdowania reduktów systemu informacyjnego lub tablicy decyzyjnej. Można je podzielić na następujące ogólne grupy: podejście oparte na macierzy rozróżnialności (np. Degang i in. (2007); X. Hu i Cercone (1995); Kryszkiewicz (1998); Skowron i Rauszer (1992); C. Wang, He, Chen i Hu (2014); Ye i Chen (2002); W.-X. Zhang, Mi i Wu (2003)), podejście oparte na obszarze pozytywnym (np. Grzymala-Busse (1991); Q. Hu, Yu, Liu i Wu (2008); Jia, Liao, Tang i Shang (2013); Pawlak (1991); Qian, Liang, Pedrycz i Dang (2010); Xie, Shen, Liu i Xu (2013); Yao i Zhao (2008)) oraz podejście oparte na entropii informacyjnej (np. Q. Hu, Yu, Xie i Liu (2006); Liang, Mi, Wei i Wang (2013); Liang i Xu (2002); Ślęzak (2002); Wei, Liang, Qian, Wang i Dang (2010); Wei, Liang, Wang i Qian (2013)).

Algorytmy pierwszej grupy wykorzystują strukturę pomocniczą, zwaną macierzą rozróżnialności, która jest konstruowana na podstawie tabeli danych. Każda komórka macierzy pokazuje atrybuty, które są różne dla danej pary obiektów. Każdy minimalny podzbiór atrybutów zawierający co najmniej jeden atrybut z każdej komórki jest reduktem.

Druga grupa algorytmów działa na pozytywnym obszarze tablicy decyzyjnej. Obszar ten zawiera obiekty spójne dla danego podzbioru atrybutów. Minimalnym podzbiorem atrybutów, który zachowuje spójność danych jest redukt.

Ostatnia grupa algorytmów ocenia podzbiór atrybutów za pomocą miary entropii informacyjnej. Entropia informacyjna pokazuje stopień rozróżnialności danego podzbioru atrybutów. Ta o najwyższej jakości w ramach danego kryterium jest wybierana jako redukt.

Ważną gałęzią redukcji atrybutów jest znajdowanie minimalnych reduktów. Znalezienie takiego reduktu pozwala na skonstruowanie najmniejszej reprezentacji danych (pod względem liczby atrybutów), zachowując pierwotny poziom rozpoznawalności obiektów. Znalezienie minimalnego reduktu jest jednak bardziej złożonym zadaniem niż znalezienie jakiegokolwiek reduktu, a mianowicie zostało pokazane, że jest to problem NP-trudny (Skowron i Rauszer, 1992).

Jednym z ogólnych podejść jest ograniczenie problemu znalezienia wszystkich reduktów do tych, których wielkość jest najmniejsza. Rozwiązanie jest raczej proste, ponieważ może polegać na filtrowaniu wszystkich reduktów lub kontrolowaniu procesu ich wytwarzania, tak aby przynajmniej niektóre z nich, które zostały wcześniej rozpoznane jako nieminimalne nie były generowane. Wadą tego podejścia jest jego złożoność, która jest zbieżna z podejściem znajdowania wszystkich reduktów.

Inne, bardziej ogólne podejście opiera się na dobrze znanej strategii wyszukiwania wszerz. Pusty zestaw atrybutów, alternatywnie zbiór składający się z podstawowych atrybutów (rdzeń reduktów), jest za każdym razem iteracyjnie zwiększany o jeden atrybut. Jeśli dany podzbiór nie jest reduktem, to poprzednio dodany atrybut jest usuwany, a do podzbioru dodawany jest inny, dotychczas nieużywany. Takie podejście gwarantuje sprawdzenie tylko podzbiorów o liczności nie wyższej niż liczba minimalnych reduktów. Jednak rozwiązanie może być czasochłonne, jeśli liczba wszystkich atrybutów opisujących dane jest stosunkowo duża.

Modyfikacja powyższego podejścia polega na obliczeniu częstotliwości występowania każdego z atrybutów, która określa kolejność, w jakiej atrybuty mają być dodawane do podzbioru. Mianowicie, atrybut najczęściej występujący w macierzy rozróżnialności jest uznawany za pierwszy.

Można również znaleźć bardziej konkretne podejścia do znalezienia jednego lub wszystkich minimalnych reduktów.

Algorytm genetyczny został zaadaptowany w pracy Wroblewski (1995) w celu znalezienia minimalnego reduktu. Każdy podzbiór zestawu atrybutów jest indywidualny i reprezentowany przez ciąg bitów, gdzie „1” („0”) oznacza, że atrybut występuje (nie występuje) w podzbiorze. Funkcja dopasowania jest definiowana na podstawie liczności podzbioru i liczby wierszy w macierzy rozróżnialności, które są objęte podzbiorem atrybutów. Najlepszy osobnik z ostatniego pokolenia jest zwracany jako minimalny redukt. Podejście to może być szybkie tylko wtedy, gdy kryterium zatrzymania jest łatwe do osiągnięcia (np. mała liczba pokoleń) i nie gwarantuje, że znaleziony redukt będzie zawsze minimalny.

Podobne, ale prostsze rozwiązanie do tego z Wroblewski (1995) zostało umieszczona w X. Wang, Yang, Peng i Teng (2005). Autorzy postawili problem znalezienia minimalnych reduktów w ramach optymalizacji roju cząstek. Dzięki temu nie są potrzebne złożone operatory, takie jak krzyżowanie czy mutacje, a potrzebne są tylko prymitywne i proste operatory matematyczne. Zostało eksperymentalnie zweryfikowane, że propozycja jest mniej kosztowna obliczeniowo pod względem czasu wykonywania i użytej pamięci w porównaniu z podejściem używającym algorytmów genetycznych.

Aby znaleźć minimalny redukt, zbiór danych w pracy Bakar, Sulaiman, Othman i Selamat (2002) przekształca się w model programowania binarnych liczb całkowitych (BIP), a redukcję atrybutów uważa się za problem spełnialności. Aby rozwiązać problem za pomocą modelu BIP, zastosowano algorytm rozgałęzienia i powiązania. Weryfikacja eksperymentalna wykazała, że podejście to znacznie zmniejsza liczbę reguł, które można wygenerować na podstawie uzyskanych reduktów.

W pracy Jensen, Shen i Tuson (2005) problem znalezienia minimalnych reduktów zbioru przybliżonego jest przeformułowany w ramach propozycji spełnialności (SAT). Klauzule cech w koniunkcyjnej postaci normalnej (CNF) są generowane ze zbioru danych. Klauzule są spełnione, jeśli po przypisaniu wartości prawdziwych do ich wszystkich zmiennych formuła jest prawdziwa w zbiorze danych. Zadanie polega na znalezieniu najmniejszej liczby takich cech, aby spełnić formułę CNF. W tym podejściu problem SAT rozwiązano za pomocą algorytmu Davisa-Logemanna-Lovelanda (DPLL). Podejście zostało porównane eksperymentalnie z podejściem opartym na ocenie redukcji obszaru pozytywnego. Proponowane rozwiązanie jest bardziej czasochłonne, ale w przeciwieństwie do odniesienia zawsze zwraca minimalne redukty.

Technika optymalizacji roju cząstek (PSO) została zaadaptowana w badaniach X. Wang, Yang, Teng, Xia i Jensen (2007) w celu znalezienia minimalnych reduktów zbioru przybliżonego. Cząstki z PSO odpowiadają atrybutom. Pozycja cząstki to binarny ciąg bitów o długości równej całkowitej wielkości podzbioru atrybutów. Każdy bit ciągu określa, czy atrybut został wybrany (1), czy nie (0). Każda pozycja odpowiada podzbiorowi atrybutów. Znajdowanie minimalnych reduktów odbywa się zgodnie ze standardowym algorytmem PSO. Podejście jest stosunkowo szybkie i zawsze gwarantuje znalezienie minimalnych reduktów.

Do znalezienia minimalnych reduktów Xu, Liu i Zhou (2008) używają modyfikację metody redukcji atrybutów opartą na obszarze pozytywnym. Proces redukcji atrybutów jest kontrolowany przez próg, czyli minimalną liczbę atrybutów potrzebnych do rozróżnienia wszystkich obiektów w zbiorze danych.

W pracy Su i Guo (2017) użyto kombinacji teorii zbiorów przybliżonych i algorytmu stada, aby znaleźć minimalny redukt. Rdzeń reduktów obliczony na podstawie macierzy rozróżnialności jest iteracyjnie zwiększany o jeden atrybut, aż do znalezienia minimalnego reduktu. Zestawy atrybutów innych niż podstawowe, lecz tej samej liczności są kodowane jako liczby całkowite będące indywiduami w algorytmie

stada. Zależność atrybutów jest stosowana do obliczania wartości dopasowania podzbiorów atrybutów.

Sprzętowe podejście do wyznaczania reduktów

Wsparcie sprzętowe przetwarzania danych oparte na zbiorach przybliżonych ma długą historię i jest ściśle związane z początkami teorii zbiorów przybliżonych. Pomysł przykładowego procesora, który generuje reguły klasyfikacji z tablic decyzyjnych opisał Pawlak (2004). W proponowanym rozwiązaniu decyzję podejmuje się po sekwencji obliczeń czynników opisujących jakość decyzji (siła, pewność, pokrycie). Konstrukcja procesora pozwala na użycie arytmometru cyfrowego lub analogowego. Inne wczesne badania obejmują konstrukcję procesora opartego na sieciach komórkowych, przedstawioną przez Lewis, Perkowski i Jozwiak (1999), a także koncepcję urządzenia zdolnego do zminimalizowania dużych funkcji logicznych tworzonych na podstawie macierzy rozróżnialności, opracowanego przez Kanasugi i Yokoyama (2001).

Jeden z pierwszych rzeczywistych systemów wspierających przetwarzanie danych ze zbiorów przybliżonych został opisany przez Kanasugi i Matsumoto (2007). Autorzy przedstawili projekt i implementację wstępnie ustawionego procesora i pokazali ogromne przyśpieszenie w obliczeniach: proponowany procesor był dziesięciokrotnie szybszy od komputera PC, mimo że częstotliwość taktowania była mniejsza o około 70 razy. Sun, Qi i Zhang (2011) przedstawili implementację metod ze zbiorów przybliżonych dla badań technologii diagnostyki usterek w oparciu o układ FPGA. Ci sami autorzy w (Sun, Wang, Lu i He, 2013) pokazali sprzętową implementację dyskretyzacji atrybutów ciągłych. W badaniach D.-L. Jiang, Zhao, Wang, Li i Yang (2013) zbiory przybliżone zostały użyte do zgrupowania danych do eksploracji. Wszystkie te rozwiązania wykazały duże przyśpieszenie w porównaniu z implementacjami programowymi i udowodniły, że implementacje oparte na FPGA są obecnie jednym z najważniejszych problemów badawczych.

Najbardziej zaawansowane i najnowsze badania związane z problemem generowania reduktów opisane są w pracy K. Tiwari i Kothari (2015, 2016); K. S. Tiwari i Kothari (2014). Badania na temat sprzętowych implementacji metod zbiorów przybliżonych przedstawiono w pracy Grześ, Kopczyński i Stepaniuk (2013). Urządzenie generujące superreduktы zostało opisane w pracy Kopczynski, Grzes i Stepaniuk (2014). Żadne z opisanych rozwiązań nie pozwala na obliczenie wszystkich reduktów.

Celem tego rozdziału jest zaprezentowanie efektywnych metod wyznaczania minimalnego reduktu ze wsparciem sprzętowym w postaci układu z rodziny FPGA.

Na wstępie, na podstawie analizy istniejących metod i algorytmów wybrano rozwiązania wykorzystujące metodę znajdowania minimalnych reduktów za pomocą

wyszukiwania wszerz. W porównaniu z innymi nie są one skomplikowane, ponieważ wykonywane są tylko podstawowe operacje takie, jak generowanie kandydatów na redukt (czyli kombinacji atrybutów), obliczenie macierzy rozróżnialności i sprawdzenie kandydata na redukt w macierzy rozróżnialności.

Następnie zostały zdefiniowane algorytmy przeszukiwania wszerz oparte na poszukiwaniu „ślepym", oraz z wykorzystaniem częstotliwości występowania atrybutów (podrozdział 2.1.1). Przedstawione zostały architektury FPGA wybranych podejść (podrozdział 2.1.3), implementacja oraz jej efekty są weryfikowane pod kątem podejścia sprzętowego (podrozdział 2.2).

Na koniec omówiono ważne różnice między tymi dwiema strategiami.

2.1 Znajdowanie minimalnych reduktów metodą przeszukiwania wszerz

W niniejszym rozdziale zostały przedstawione algorytmiczne podejścia do znajdowania minimalnych reduktów przy użyciu strategii „ślepego" wyszukiwania wszerz oraz w oparciu o częstotliwości występowania atrybutów, jak również możliwości sprzętowej implementacji prowadzącej do zmniejszenia czasu wykonania algorytmu.

Definicje podstawowych pojęć, używanych w pracy, tj. tablica decyzyjna, macierz rozróżnialności, redukt, Czytelnik może odnaleźć np. w Choromański, Grześ i Hońko (2020).

2.1.1 Proponowane algorytmy wyznaczania reduktów

Wersja „ślepa" strategii opartej na wyszukiwaniu wszerz (patrz algorytm 2.1) (Choromański i in., 2020) rozpoczyna się od obliczenia macierzy rozróżnialności oraz rdzenia, tj. podzbioru atrybutów, który jest zawarty w każdym redukcie. Następnie wszystkie zestawienia o najmniejszej liczności (każde zestawienie składa się z podstawowych atrybutów i jednego dodatkowego atrybutu) są sprawdzane pod kątem bycia reduktem. Przeszukiwanie jest przerywane po znalezieniu wymaganej liczby reduktów lub po sprawdzeniu wszystkich zestawień i znalezieniu co najmniej jednego reduktu. Jeśli nie zostanie znaleziony żaden redukt, wszystkie zestawienia o liczności większej o jeden (każde zestawienie jest konstruowane na podstawie zestawienia z poprzedniego poziomu liczności przez dodanie kolejnego atrybutu) są sprawdzane w ten sam sposób. Zestawienia konstruowane są w kolejności alfanumerycznej, dzięki czemu każde nowe zestawienie nie jest powtórzeniem żadnego wygenerowanego wcześniej.

Algorytm 2.1 wykorzystuje następujące funkcje:

Algorytm 2.1 *GenerateMinReducts*

INPUT: $DT = (U, A \cup \{d\})$ – tablica decyzyjna; k – liczba minimalnych reduktów do odnalezienia (0 – wszystkie redukty);
OUTPUT: MR – zestaw znalezionych pierwszych k minimalnych reduktów;

```
1:  MR := ∅;
2:  DM := computeDiscMatrix(DT);
3:  C := computeCore(DM);
4:  𝒮 := ∅;
5:  while MR = ∅ do
6:      if 𝒮 = ∅ then
7:          𝒮 = {C}
8:      else 𝒮 := computeAllCombs(A, 𝒮)
9:      end if
10:     for S ∈ 𝒮 do
11:         if isReduct(DM, S) then
12:             MR = MR ∪ {S};
13:             if |MR| = k then
14:                 break
15:             end if
16:         end if
17:     end for
18: end while
```

1. $computeDiscMatrix(DT)$ oblicza macierz rozróżnialności dla tabeli decyzyjnej DT,
2. $computeCore(DM)$ oblicza rdzeń dla macierzy rozróżnialności DM,
3. $computeAllCombs(A, \mathscr{S})$ oblicza dla każdego zestawienia atrybutów $S \in \mathscr{S}$ wszystkie zestawienia (przez dodanie jednego atrybutu), które nie zostały do tej pory sprawdzone,
4. $isReduct(DM, S)$ sprawdza, czy zestawienie atrybutów S jest reduktem w nawiązaniu do macierzy rozróżnialności DM.

Opis działania algorytmu 2.1 w układzie FPGA został przedstawiony w rozdziale 2.1.3.3.

Wersja strategii opartej na wyszukiwaniu wszerz w oparciu o częstotliwości występowania atrybutów (patrz algorytm 2.2), (Choromański i in., 2020) zaczyna się od tych samych obliczeń co w przypadku metody „ślepej", tj. od wyliczenia macierzy rozróżnialności wraz z rdzeniem. Następnie dla każdego atrybutu, który nie jest zawarty w rdzeniu, obliczana jest jego częstotliwość występowania w macierzy rozróżnialności. Zestawienie o najmniejszej liczności (czyli podstawowe atrybuty plus jeden atrybut) są konstruowane zaczynając od najczęstszego atrybutu i kończąc po użyciu wszystkich lub najczęstszych atrybutów l (l zdefiniowanych przez użytkownika). Zestawienia są weryfikowane jako redukty w taki sam sposób, jak w przy-

padku wersji „ślepej”. Zestawienia o wyższym poziomie liczności są konstruowane na podstawie tych z poprzedniego poziomu przez dodanie jednego atrybutu wybranego zgodnie z kolejnością częstości występowania. Należy podkreślić, że taki sposób konstruowania zestawień atrybutów nie gwarantuje, że powstają tylko unikalne zestawienia, dlatego każde nowe zestawienie jest najpierw porównywana z innymi o tym samym poziomie liczności, które zostały do tej pory wygenerowane. W porównaniu z algorytmem 2.1, dodatkową funkcją wykorzystywaną przez algorytm 2.2 jest $computeAttrFreq(DM,B)$), który oblicza dla każdego atrybutu z zbioru B częstotliwość jego występowania w macierzy rozróżnialności DM.

Algorytm 2.2 *GenerateMinReductsFreq*

INPUT: $DT = (U, A \cup \{d\})$ – tablica decyzyjna; k – liczba minimalnych reduktów do odnalezienia (0 – wszystkie redukty); l – numer pierwszego, najczęściej występującego atrybutu (0 – wszystkie redukty);
OUTPUT: MR – zestaw pierwszych k odnalezionych minimalnych reduktów;

```
1:  MR := ∅;
2:  DM := computeDiscMatrix(DT);
3:  C := computeCore(DM);
4:  𝒮 := C;
5:  while MR = ∅ do
6:      𝒮' = ∅;
7:      for S ∈ 𝒮 do
8:          order := sort(computeAttrFreq(DM, A \ S));
9:          if l = 0 then
10:             l' = |order|;
11:         else l' = l;
12:         end if
13:         i := 0;
14:         while i < l' do
15:             S' = S ∪ order[i];
16:             if S' ∈ 𝒮' then
17:                 continue;
18:             end if
19:             if isReduct(DM, S') then
20:                 MR = MR ∪ {S'};
21:                 if |MR| = k then
22:                     break;
23:                 end if
24:             end if
25:             i := i + 1;
26:             𝒮' := 𝒮' ∪ {S'};
27:         end while
28:     end for
29:     𝒮 := 𝒮';
30: end while
```

Należy zauważyć, że w związku z wprowadzeniem parametru l, algorytm 2.2 jest w rzeczywistości połączeniem strategii przeszukiwania wszerz i wgłąb. Mianowicie,

- $l = 0$ – strategia przeszukiwania wszerz,
- $l = 1$ – strategia poszukiwania wgłąb,
- $l > 1$ – ograniczona strategia wyszukiwania wszerz, rozszerzona strategia poszukiwania wgłąb.

Opis działania algorytmu 2.2 w układzie FPGA został przedstawiony w rozdziale 2.1.3.3.

Algorytmy przeszukujące wszerz zaimplementowane na zwykłym komputerze klasy PC posiadają złożoność obliczeniową $O(n^2)$, gdzie n – liczność zbioru obiektów A, natomiast pamięciowa wynosi $O(2^m)$, gdzie m – liczba atrybutów warunkowych. W przypadku implementacji z użyciem układów kombinacyjnych na układzie FPGA, operacje mogą być wykonywane od razu, dzięki czemu może być osiągnięta złożoność nawet równa $O(1)$. Rzeczywista złożoność obliczeniowa jest jednak trudna do przewidzenia, ponieważ wszystko zależy od czasu propagacji sygnału do poszczególnych bloków wewnątrz FPGA, natomiast osiągnięcie takiego wyniku może być niemożliwe ze względu na fizyczne ograniczenie wynikające ze skończonej liczby elementów logicznych zawartych w układzie FPGA. W tym wypadku liczba wykorzystywanych elementów logicznych jest proporcjonalna do n^2.

2.1.2 Układy programowalne z rodziny FPGA

2.1.2.1 Definicja

FPGA (ang. *Field Programmable Gate Array*) jest to rodzaj programowalnego układu logicznego (ang. *Programmable Logic Device* – PLD) charakteryzujący się architekturą złożona z matrycy komórek logicznych połączonych za pomocą linii poprowadzonych w kanałach połączeniowych (Skahill, 2001). Na obrzeżach matrycy komórek logicznych znajdują się elementy wejścia/wyjścia, mogące być skonfigurowane jednokierunkowo (jako wejście lub wyjście) oraz dwukierunkowo. Współczesne układy FPGA mogą zawierać ponad pół miliona bloków logicznych oraz być zintegrowane z procesorem ARM tworząc tym samym układ SoC (ang. *System-on-Chip*). Najmniejszą jednostką w układzie FPGA jest posiadający cztery wejścia oraz 3 wyjścia element logiczny. Dzięki zastosowaniu tablicy przeglądowej (LUT) możliwe jest zaimplementowanie w układzie praktycznie każdej funkcji logicznej.

2.1.2.2 Języki opisu sprzętu

Języki opisu sprzętu (HDL) jest to rodzina wyspecjalizowanych języków komputerowych wykorzystywana do opisu struktury i działania układów cyfrowych. Pierwsi przedstawiciele tej rodziny powstali w latach 60. XX wieku, kiedy pojawiło się zapotrzebowanie na uproszczenie procesu projektowania układów cyfrowych oraz zautomatyzowania niektórych jego etapów. Oprócz samej syntezy układów, HDL umożliwiają również optymalizację układu, minimalizację funkcji czy tworzenie platform testowych do symulacji i tym samym sprawdzenia zaprojektowanego układu. Projektowanie układów cyfrowych z użyciem HDL można wykonać na dwa sposoby:

- *behawioralny* – opisywane są zależności pomiędzy danymi z wejścia oraz wyjścia układu, sama realizacja układu zostaje zrealizowana przez kompilator,
- *funkcjonalny* – tworzone są bloki funkcjonalne (np. sumatory, bloki zawierające funkcje logiczne) oraz opisywane są zależności między nimi.

Przykładami języków opisu sprzętu są:

- VHDL (ang. *Very High Speed Integrated Circuits Hardware Description Language*) – jeden z najpowszechniejszych języków z rodziny HDL. Pierwotnie stworzony na potrzeby dokumentacji układów ASIC. Język rozwijany jest do chwili obecnej, aktualnie najnowszą wersją jest IEEE Std 1076-2008 zwana potocznie VHDL 2008,
- Verilog – drugi najpopularniejszy język opisu sprzętu, składniowo bardzo przypominający język C; od roku 1990 język ten jest otwartym standardem; język ten podobnie jak VHDL jest stale rozwijany, najnowsza aktualna wersja to IEEE Std 1364-2001; z Veriloga wyewoluował inny język z rodziny HDL – SystemVerilog,
- AHDL (ang. *Altera Hardware Description Language*) – język opisu sprzętu stworzony przez firmę Altera (aktualnie jest ona częścią firmy Intel); język ten przypomina Veriloga, jednak może być wykorzystywany tylko w przypadku programowania układów CPLD i FPGA produkcji Altery/Intela,
- SystemC – HDL pozwalający modelować układy z poziomu systemu; składniowo praktycznie nie różni się od języka C++.

2.1.3 Sprzętowa implementacja algorytmów wyznaczania reduktów

W poniższych podrozdziałach opisano używany sprzęt, zestaw danych i zaimplementowane bloki służące do wyliczania minimalnych reduktów z użyciem układu FPGA.

2.1.3.1 Wykorzystany sprzęt

Projekt został zrealizowany na dwóch różnych urządzeniach. Pierwsze z nich to Intel Arria V SoC (5ASTFD5K3F40I3), określany dalej jako „Arria", będącym 28 nm FPGA z 462 tysiącami elementów logicznych i zintegrowanym z dwurdzeniowym procesorem ARM Cortex-A9 o taktowaniu 1,05 GHz. W naszym rozwiązaniu jednak nie zastosowano zintegrowanego procesora ARM; zamiast niego zastosowano procesor typu softcore Nios II. Został on wybrany ze względu na łatwiejsze debugowanie na poziomach rejestrów i ALU. Nios II był taktowany zegarem o częstotliwości 50MHz, co dawało większą szansę zaobserwowania różnic między rozwiązaniami. Oprogramowanie użyte do kompilacji, syntezy i generowania plików konfiguracyjnych to Quartus Prime 17.1.0 Build 590 25.10.2017 SJ Standard Edition. Kod C został skompilowany za pomocą Eclipse Mars.2 Release (4.5.2) Build 20160218-0600 z kompilatorem GCC w wersji 4.8.3.
Drugim urządzeniem był Xilinx Zynq Ultrascale+ MPSoC (XCZU9EG-2FFVB1156) określany dalej jako „Zynq US+", który jest 16 nm FPGA z 600 tysiącami elementów logicznych i zintegrowanym z czterordzeniowym procesorem ARM Cortex-A53 (taktowany na 1,5 GHz), dwurdzeniowym procesorem czasu rzeczywistego Cortex-R5 (taktowany z częstotliwością 600 MHz) i procesorem graficznym Mali-400MP2 (taktowany z częstotliwością 667 MHz). Oprogramowanie używane dla Zynq US+ to Vivado Design Edition 2018.3 (64-bit) SW Build 2405991, IP Build: 2404404. Kod C został skompilowany za pomocą Xilinx SDK, który jest oparty na Eclipse Wersja: 4.6.1.v20160907-1200, identyfikator kompilacji: M20160907-1200 z kompilatorem GCC w wersji 7.3.1.

2.1.3.2 Użyte dane oraz ich struktura

W tym projekcie zostały wykorzystane dane dotyczące dzieci z cukrzycą insulinozależną typu 1 (Stepaniuk, 1999). Struktura danych determinuje strukturę niektórych bloków w implementacji sprzętowej. Jednakże ogólna koncepcja działania pozostaje niezależna od typu, rodzaju i struktury danych. Dane wykorzystywane w badaniach obejmują 107 obiektów. Każdy z nich ma 12 atrybutów, które w reprezentacji binarnej tworzą łącznie 16-bitowe słowo:

- płeć - 1 bit,
- wiek w którym zdiagnozowano chorobę - 2 bity,
- czas trwania choroby - 2 bity,
- występowanie cukrzycy w rodzinie - 1 bit,
- typ terapii insulinowej - 1 bit,
- infekcje układu oddechowego - 1 bit,
- remisja - 1 bit,
- HbA1c - 2 bity,

- nadciśnienie - 1 bit,
- masa ciała - 2 bity,
- hipercholesterolemia - 1 bit,
- hipertriglicerydemia - 1 bit.

Istnieje również jeden 1-bitowy atrybut decyzji:

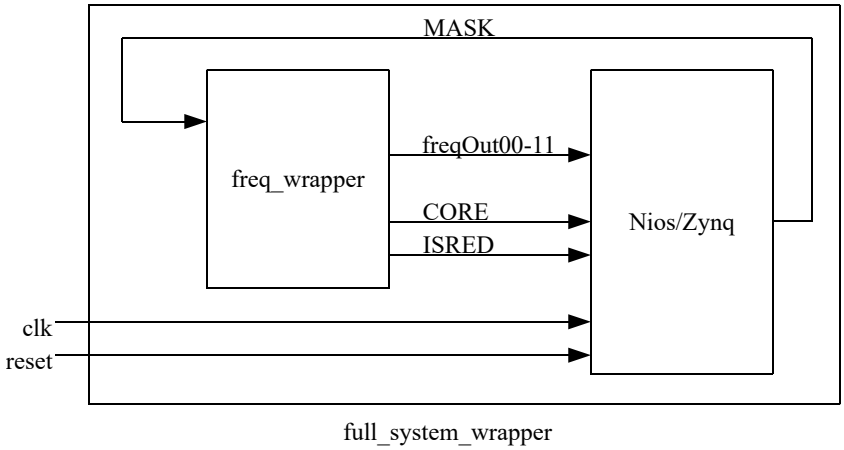
- mikroalbuminuria.

Dane są podzielone na dwa oddzielne zestawy:

- pozytywne - z 56 przypadkami, które mają mikroalbuminurię (wartość atrybutu decyzyjnego jest równa „1”),
- negatywna - z 51 przypadkami, które nie mają mikroalbuminurii (wartość atrybutu decyzyjnego jest równa „0”).

2.1.3.3 Zaimplementowane bloki sprzętowe

Architektura zaproponowanego rozwiązania została zaprezentowana na rysunku 2.1



Rysunek 2.1: Architektura rozwiązania na podstawie Choromański i in. (2019)

Full_system_wrapper jest plikiem najwyższego poziomu, który zawiera komponenty *freq_wrapper* oraz *nios/Zynq*, a także łączy je ze sobą. Blok ten można traktować jako cały system i posiada dwa porty wejściowe:

- **clk** – wejście zegarowe o częstotliwości 50 MHz dla procesora *nios* w Arrii (100 MHz w przypadku Zynq US+),
- **reset** – sygnał resetujący wszystkie komponenty.

Blok ten nie posiada portów wyjściowych. Cały system składa się z następujących elementów:

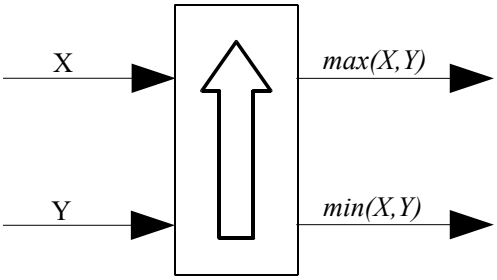
- **nios/Zynq** – instancja procesora Nios II/f (procesora Zynq w przypadku Zynq US+) używany do kontrolowania wykonania algorytmu. Głównym celem tego bloku jest znalezienie reduktu za pomocą aplikacji napisanej w C. Wykorzystuje pozostałe bloki systemu podczas wykonywania programu w celu zwiększenia szybkości obliczeń: odbiera rdzeń reduktów obliczony przez FPGA i zgodnie z nim wysyła możliwych kandydatów na redukt, dopóki sygnał ISRED zmieni wartość na „0”. Działanie tego bloku zostało przedstawione w liniach 1.6–1.15 w algorytmie 2.1 oraz w liniach 2.6–2.23 w algorytmie 2.2. Ten blok ma cztery porty wejściowe:
  - **clk** – wejście zegara o częstotliwości 50 MHz (100 MHz dla Zynq US+),
  - **reset** – sygnał do resetowania procesora,
  - **freqOut00…freqOut11** – 16-bitowe słowo z częstotliwością każdego atrybutu,
  - **CORE** – dostarcza rdzeń obliczony w komponencie *freq_wrapper*,
  - **ISRED** – sygnał wskazujący, że sygnał MASK jest reduktem.

  Ten blok ma jeden port wyjściowy:
  - **MASK** – wysyła do *freq_wrapper* kandydata na redukt na podstawie obliczonego rdzenia (*CORE*) podanego na wejściu.Ten sygnał jest również używany w innych komponentach.

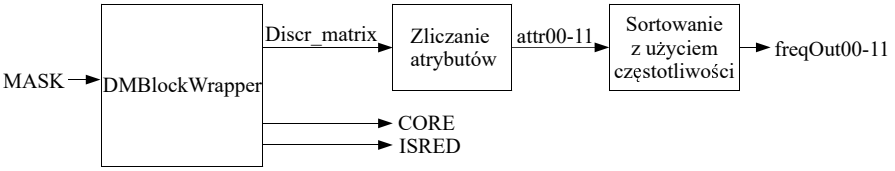
- **freq_wrapper** – komponent, w którym częstości atrybutów są obliczane i sortowane za pomocą algorytmu bitonicznego (przedstawionego na rysunku 2.2) od najczęściej do najrzadziej występujących. W algorytmie 2.2 działanie tego bloku jest przedstawione jako funkcja *sort()* znajdująca się w linii 2.8. Każdy wynik jest reprezentowany przez 16-bitowy wektor, gdzie pierwsze 4 bity zawierają numer atrybutu, a pozostałe 12 bitów to liczba wystąpień. *freq_wrapper* zawiera komponent *DMBlockWrapper* odpowiedzialny za obliczanie rdzenia i reduktu. Ten blok ma jeden port wejściowy:
  - **MASK** – kandydat na redukt dostarczony z komponentu *nios/Zynq*.

  Ten blok ma czternaście portów wyjściowych:
  - **freqOut00…freqOut11** – 16-bitowe słowo z częstotliwością każdego atrybutu,
  - **CORE** – obliczony rdzeń dostarczony z *DMBlockWrapper*,

- **ISRED** – pojedynczy bit informujący, że kandydat przekazany z komponentu *nios/Zynq* jest reduktem.



Rysunek 2.2: Schemat sortowania bitonicznego



Rysunek 2.3: Schemat bloku freq_wrapper na podstawie Choromański i in. (2020)

Architekturę bloku *freq_wrapper* przedstawiono na rysunku 2.3. Jak wspomniano wcześniej, ten komponent składa się z jednego bloku *DMBlockWrapper* opisanego poniżej:

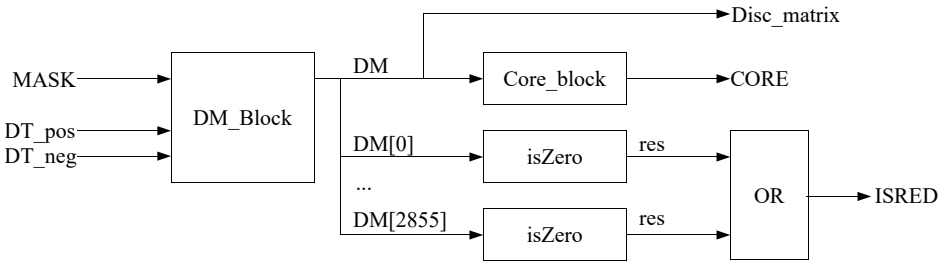
- **DMBlockWrapper** – wrapper dla komponentów używanych do obliczania rdzenia i reduktu. Składa się z bloków *DM_block*, *Core_block*, 2856 wygenerowanych komponentów *isZero* i łączy je ze sobą. Na podstawie wyliczonych danych poszukiwany jest redukt, co zostało przedstawione jako funkcja *isReduct()* linii 1.10 algorytmu 2.1 oraz linii 2.14 algorytmu 2.2. Gdy zostanie znaleziony redukt, wysyła on do *freq_wrapper* 1-bitowy sygnał '0'. Komponent *DMBlockWrapper* zawiera również pozytywne i negatywne zbiory danych opisane w 2.1.3.2, które są przekazywane do modułu *DM_block* z użyciem sygnałów *DT_pos* i *DT_neg*. Ten blok ma jeden port wejściowy:

  - **MASK** – kandydat na redukt dostarczony z komponentu *nios/Zynq*.

  Ten blok posiada trzy porty wyjściowe:

  - **Disc_matrix** - tablica zawierająca obliczoną macierz rozróżnialności,

- **CORE** – wysyła obliczony rdzeń do *freq_wrapper* gdzie następnie jest przekazywany do komponentu *nios/Zynq*,
- **ISRED** – wysyła 1-bitowy sygnał, który wskazuje, że wektor podany na wejściu MASK jest reduktem (lub nie) do bloku *freq_wrapper* gdzie następnie jest przekazywany do komponentu *nios/Zynq*.



Rysunek 2.4: Schemat bloku DMBlockWrapper na podstawie Choromański i in. (2020)

Połączenie między komponentami wewnątrz wrappera pokazano na rysunku 2.4. Bloki, które tworzą komponent *DMBlockWrapper*, są opisane poniżej:

- **DM_block** – komponent tworzący macierz rozróżnialności na podstawie danych z wygenerowanych 2856 modułów *DM_comp*. Działanie tego bloku jest przedstawione jako funkcja *computeDiscMatrix()*, która znajduje się w algorytmie 2.1 w linii 1.3 oraz w algorytmie 2.2 w linii 2.3. Ten blok ma trzy porty wejściowe:
  - **MASK** – kandydat na redukt dostarczony z komponentu *nios/Zynq*,
  - **DT_pos** – dostarcza 56-elementową tablicę zawierającą 16-bitowe wyrazy binarne złożone z atrybutów pochodzące z pozytywnego zestawu danych z komponentu *DMBlockWrapper*,
  - **DT_neg** – - dostarcza 51-elementową tablicę zawierającą 16-bitowe wyrazy binarne złożone z atrybutów pochodzące z negatywnego zestawu danych z komponentu *DMBlockWrapper*.

  Ten blok ma jeden port wyjściowy:

  - **DM** – wysyła tablicę złożoną z 2856 12-bitowych słów (wygenerowana macierz rozróżnialności) do komponentu *DMBlockWrapper*.

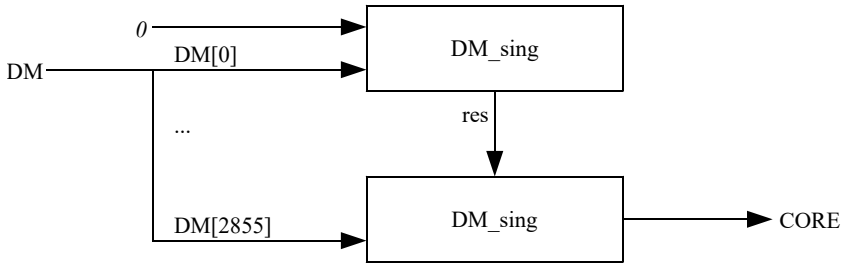
- **Core_block** – komponent, w którym obliczany jest rdzeń. Ten blok składa się z 2856 wygenerowanych komponentów *DM_sing*, które tworzą kaskadę elementów. Połączenia między blokami pokazano na rysunku 2.5. Działanie tego bloku jest przedstawione jako funkcja *computeCore()*, która znajduje się w algoryt-

mie 2.1 w linii 1.4 oraz w algorytmie 2.2 w linii 2.4. Ten blok ma jeden port wejściowy:

- **DM** – 2856-elementowa tablica 12-bitowych słów, która zawiera całą macierz rozróżnialności dostarczoną z *DMBlockWrapper*.

Ten blok ma jeden port wyjściowy:

- **CORE** – wysyła do *DMBlockWrapper* rdzeń obliczony na podstawie dostarczonej macierzy rozróżnialności.



Rysunek 2.5: Schemat połączeń wewnątrz Core_block na podstawie Choromański i in. (2020)

- **isZero** – komponent wyszukujący ’0’ w danym 12-bitowym słowie wykonując operację OR na każdym pojedynczym bicie z podanego wektora. wynik jest przekazywany do *DM_sing*, gdzie następuje sprawdzenie, czy nowy wpis macierzy rozróżnialności powinien być równy „0”, czy nie. Ten blok ma jeden port wejściowy:
  - **inval** - 12-bitowe słowo będące wynikiem porównania dwóch wartości z pozytywnego i negatywnego zbioru danych.

  Ten blok ma jeden port wyjściowy:
  - **res** - pojedynczy bit, który ma wartość ’0’ tylko wtedy, gdy sygnał podany na wejściu jest równy ’0’.

Jak wspomniano wcześniej, *DM_block* składa się z wielu generowanych komponentów *DM_comp* które opisano poniżej:

- **DM_comp** – komponent porównujący wartości danych z pozytywnego oraz negatywnego zestawu danych, określoną przez bit decyzyjny (dodatnie wartości są przypisywane osobom z mikroalbuminurią). Porównania dokonuje się przez wykonanie operacji XOR danych pozytywnych z negatywnymi, a następnie dokonywana jest koniunkcja wyniku z daną maską. Blok *DM_comp* używa komponentu *isZero* do wyszukiwania dowolnego „0” w wyrazie powstałym w wyniku

poprzednich operacji. Wynikiem tych operacji jest pojedynczy wpis do macierzy rozróżnialności. Jeśli blok *isZero* nie znajdzie „0”, wartość wprowadzona do macierzy rozróżnialności jest równa „0”. Ten blok ma trzy porty wejściowe:

- **MASK** - 12-bitowe słowo używane do maskowania atrybutów,
- **DT_pos** – wpis tablicy decyzyjnej ze zbioru danych klasy pozytywnej (słowo 16-bitowe) z komponentu *DMBlockWrapper*,
- **DT_neg** – wpis tablicy decyzyjnej ze zbioru danych klasy negatywnej (słowo 16-bitowe) ze składnika *DMBlockWrapper*.

Ten blok ma jeden port wyjściowy:

- **DM_entry** – wysyła obliczony wpis macierzy rozróżnialności (słowo 12-bitowe) do komponentu *DM_block*.

Jak pokazano na rysunku 2.5, *Core_block* składa się z wielu generowanych komponentów *DM_sing* opisanych poniżej:

- DM_sing –komponent, w którym każdy wpis macierzy rozróżnialności jest sprawdzany, czy jest singletonem (słowo binarne posiada tylko jedną wartość ‘1’), czy nie. Przyjmuje dwie wartości: poprzednią wartość z kaskady i wartość z macierzy rozróżnialności, a następnie sprawdza, czy podana wartość z macierzy rozróżnialności jest singletonem, czy nie. Jeśli wartość z macierzy nim jest, to wykonywana jest operacja OR z wartością kaskadową i wynik jest wypychany na wyjście. W przeciwnym razie wypychana na wyjście jest wartość kaskady. Ten blok ma dwa porty wejściowe:

  - **prev** – dostarcza 12-bitowe słowo z kaskady,
  - **DM** – zapewnia 12-bitowy wpis z macierzy rozróżnialności.

  Ten blok ma jeden port wyjściowy:

  - **res** – wysyła 12-bitowy wynik operacji OR lub wartość kaskadową do *Core_block*.

W systemie znajdują się również drobne bloki, które służą jako kontenery logiczne:

- **rMux** – moduł, w którym wszystkie atrybuty są liczone na potrzeby sortowania,
- **sort** – moduł używany do algorytmu sortowania bitonicznego i odpowiedzialny za porównanie dwóch podanych wartości, schemat tego bloku przedstawia rysunku 2.2,
- **isSingleton** – moduł sprawdzający czy wartość podana na wejściu jest singletonem czy nie.

2.1.3.4 Zużyte zasoby sprzętowe

Projekt dla układu Arria został zbudowany przy użyciu Quartus Prime firmy Altera. Raporty z kompilacji informuje o zużyciu następującej liczby zasobów FPGA:

- wykorzystanie elementów logicznych (w ALM): 15,492 / 176,160 (9 %),
- suma rejestrów: 2763,
- suma pinów: 1/876 (<1 %),
- zużyta pamięć: 8 452 928/23 367 680 (36 %),
- łączna liczba bloków DSP: 3/1090 (<1 %).

Wykorzystanie zasobów FPGA jest niskie. Projekt wymagał mniej niż 10 % elementów logicznych (ALM). Należy zauważyć, że projekt wykorzystuje procesor softcore, który potrzebuje pewnych zasobów (około 3553 ALM zostało wykorzystanych do implementacji procesora Nios II), dlatego zużycie elementów logicznych można zmniejszyć po przejściu na zintegrowany procesor ARM.

Projekt dla układu Zynq US+ został zbudowany przy użyciu Vivado firmy Xilinx. Raporty kompilacyjne informują o wykorzystaniu następujących zasobów FPGA:

- wykorzystanie LUT: 26 930/274 080 (9,83 %),
- LUTRAM: 70/144 000 (0,05 %),
- liczba rejestrów: 3 161 / 548 160 (0,58 %),
- liczba buforów globalnych (BUFG): 1/404 (0,25 %).

Wykorzystanie zasobów FPGA jest bardzo niskie. Projekt przygotowany dla Zynq US+ wymagał mniej niż 10 % tablic przeglądowych (LUT) i mniej niż 1 % dostępnych rejestrów.

2.1.3.5 Implementacja oprogramowania sterującego

Wszystkie komponenty sprzętowe do obliczania macierzy rozróżnialności i rdzenia oraz do sprawdzania kandydatów pod kątem bycia reduktem, zostały napisane w języku VHDL.

Implementację sprzętową wspiera aplikacja napisana w języku C, która na podstawie danych dostarczonych przez sprzęt określa najbardziej prawdopodobnego kandydata do miana reduktu.

Macierz rozróżnialności jest obliczana w *DM_block* który składa się z wielu komparatorów, porównujących wartości dwóch obiektów z tablic decyzyjnych. Tablica decyzyjna następnie jest przekazywana z komponentu *DMBlockWrapper* w którym zadeklarowano zestaw danych zarówno negatywnych, jak i pozytywnych. Wyniki obliczeń są przekazywane z powrotem do *DMBlockWrapper*, gdzie są używane do obliczenia rdzenia dla podanych danych.

Rdzeń jest obliczany w *Core_block*, który składa się z kaskady bloków *isSingleton*. *Core_block* jest układem kombinacyjnym i nie potrzebuje sygnału zegarowego,

więc czas obliczeń zależy tylko od czasu propagacji sygnałów do poszczególnych bloków logicznych wewnątrz układu FPGA.

Obliczony rdzeń jest również używany jako pierwsza maska, która zeruje już użyte atrybuty w macierzy rozróżnialności. Po uprzednim wyzerowaniu, komponent *freq_wrapper* oblicza częstotliwość występowania każdego atrybutu, a następnie sortuje te dane od najczęściej do najrzadziej występujących. Posortowane dane są wysyłane z obliczonym rdzeniem i flagą ISRED (która informuje, czy aktualny zestaw atrybutów przesyłany jako maska jest reduktem, czy też nie) do instancji procesora Nios/Zynq.

W procesorze Nios/Zynq, aplikacja napisana w C na podstawie częstotliwości występowania atrybutów, wybiera najbardziej prawdopodobnego kandydata do bycia reduktem i odsyła go do komponentu *freq_wrapper* jako maskę. Maska, która została wysłana z aplikacji C jest sprawdzana pod kątem reduktu przez FPGA, po czym ponownie wykonywane jest zerowanie i obliczanie częstotliwości.

Nowe częstotliwości ze znacznikiem ISRED są odsyłane z powrotem do procesora Nios/Zynq, który wpisuje kolejnego kandydata. Proces jest powtarzany, aż zostaną znalezione wszystkie minimalne redukty.

2.2 Eksperymenty

W tym podrozdziale opisano badania eksperymentalne dotyczące znalezienia minimalnych reduktów za pomocą podejścia sprzętowego przy użyciu implementacji wybranych algorytmów wyszukiwania wszerz.

Na potrzeby badań eksperymentalnych w języku C napisano aplikację do obliczania rdzenia i minimalnych reduktów. Działa ona w dwóch trybach: samodzielnym i ze wsparciem systemu sprzętowego opisanego w podrozdziale 2.1.3.5. Główną różnicą między tymi dwoma trybami jest przeniesienie logiki odpowiedzialnej za obliczenia rdzenia i reduktów do FPGA (Choromański i in., 2020). W rzeczywistości tryb samodzielny można krótko opisać jako programową symulację algorytmu zaimplementowanego sprzętowo. Aby uzyskać jak najdokładniejszy wynik czasowy, każdy eksperyment wykonano 10, 100, 1000 i 10000 razy. Wszystkie przeprowadzono na zbiorze danych opisanym w podrozdziale 2.1.3.2.

Początkowo testy były wykonywane przy użyciu aplikacji C w trybie samodzielnym, uruchomionej na procesorze Nios II pracującym z zegarem o częstotliwości 50 MHz. Celem tych testów było zbadanie średniego czasu potrzebnego do obliczenia rdzenia i reduktów za pomocą „czystej" aplikacji C przed uruchomieniem jej ze wsparciem układu FPGA.

Celem pierwszego eksperymentu jest sprawdzenie, ile czasu zajmują obliczenia podczas korzystania ze strategii „ślepego" przeszukiwania wszerz. Tabela 2.1 przed-

stawia czas wielokrotnego wyliczania wszystkich minimalnych reduktów dla danego zbioru danych oraz czas potrzebny do pojedynczego wykonania obliczeń.

Tabela 2.1: Wyniki obliczania rdzenia i reduktu z użyciem oprogramowania w C, uruchomionego na procesorze Nios II - BBFS

Liczba wykonań	**Czas [s]**	**Czas [s] potrzebny na 1 wykonanie**
10	0.657	0.0657
100	6.572	0.06572
1000	65.728	0.065728
10000	657.272	0.0657272

Drugi eksperyment sprawdza czas potrzebny do obliczeń przy użyciu strategii wyszukiwania wszerz partej na częstotliwościach występowania atrybutów. Wyniki przedstawione w tabeli 2.2 w porównaniu z poprzednim testem pokazują, że strategia oparta na częstotliwościach jest około 1,38 razy szybsza, a tym samym lepsza.

Tabela 2.2: Wyniki obliczania rdzenia i reduktu z użyciem oprogramowania w C, uruchomionego na procesorze Nios II - FBFS

Liczba wykonań	**Czas [s]**	**Czas [s] potrzebny na 1 wykonanie**
10	0.732	0.0732
100	7.327	0.0732
1000	73.275	0.073275
10000	732.749	0.0732749

Po ustaleniu czasów odniesienia uruchomiono aplikację ze wsparciem stworzonego systemu sprzętowego. Kolejne eksperymenty zostały przeprowadzone dokładnie tak samo, jak dwa poprzednie, ale tym razem obliczenia zostały wykonane przez układ FPGA.

W trzecim eksperymencie zastosowano strategię „ślepego" przeszukiwania wszerz. Wyniki przedstawione w tabeli 2.3 pokazują czasy wielokrotnego i uśrednione czasy pojedynczego wykonania obliczeń.

Celem czwartego eksperymentu jest sprawdzenie, ile czasu zajmuje obliczenie rdzenia i reduktu przy zastosowaniu strategii wyszukiwania szerokości opartej na częstotliwościach. Wyniki przedstawione w tabeli 2.4 pokazują, że ta strategia daje prawie takie same wartości, jak strategia „ślepego" przeszukiwania wszerz.

Tabela 2.3: Wyniki obliczania rdzenia i reduktu przy użyciu FPGA - BBFS

Liczba wykonań	**Czas [s]**	**Czas [s] potrzebny na 1 wykonanie**
10	0.006	0.0006
100	0.059	0.00059
1000	0.590	0.000590
10000	5.898	0.0005898

Tabela 2.4: Wyniki obliczania rdzenia i reduktu przy użyciu FPGA - FBFS

Liczba wykonań	**Czas [s]**	**Czas [s] potrzebny na 1 wykonanie**
10	0.005	0.0005
100	0.048	0.00048
1000	0.482	0.000482
10000	4.817	0.0004817

Wyniki przedstawione w tabelach 2.3 i 2.4 pokazują, że jeśli do rozwiązania problemu zostanie użyty układ kombinacyjny, to nie ma znaczenia, jaki algorytm został użyty, ponieważ wszystko zależy głównie od czasu propagacji sygnału.

W celach testowych aplikacja C w trybie autonomicznym została uruchomiona na komputerze PC z procesorem Intel Core i7-770 @ 3,60 GHz i 32 GB RAM.

Tabela 2.5: Wyniki obliczania rdzenia i reduktu przy użyciu komputera PC - BBFS

Liczba wykonań	**Czas [s]**	**Czas [s] potrzebny na 1 wykonanie**
10	0	0
100	0.015622	0.00015622
1000	0.177181	0.000177181
10000	1.696734	0.0001696734

Jak widać w tabelach 2.5 i 2.6, wyniki są inne niż w przypadku FPGA oraz, że BBFS była nieco szybsza niż FBFS.

Uzyskane wyniki są również około 3,5 razy (w przypadku BBFS) i około 2,7 razy (w przypadku FBFS) lepsze niż wyniki z FPGA.

Tabela 2.6: Wyniki obliczania rdzenia i reduktu przy użyciu komputera PC - FBFS

Liczba wykonań	**Czas [s]**	**Czas [s] potrzebny na 1 wykonanie**
10	0	0
100	0.015652	0.00015652
1000	0.187467	0.000187467
10000	1.880553	0.0001880553

Należy jednak zauważyć, że zegar komputera PC jest $\frac{clk_{PC}}{clk_{FPGA}} = \frac{3600MHz}{50MHz} = 72$ razy szybszy niż źródło zegara FPGA, więc jeśli weźmiemy tę różnicę, wyniki są znacznie lepsze - współczynnik przyśpieszenia wynosi około 21 dla BBFS i około 28 dla FBFS.

Dla celów testowych, ta sama konfiguracja sprzętowa została uruchomiona również na drugim urządzeniu konkurencyjnej firmy - Zynq US+ (Choromański i in., 2019). Kolejne eksperymenty przebiegały identycznie do poprzednich.

Na początku zestawiono obliczenia z użyciem BBFS (tabela 2.7). Porównując wynik z czasami uzyskanymi z Arrii można stwierdzić, że czas realizacji jest znacznie krótszy (około 15,5 razy).

Następnie sprawdzono ile czasu zajmuje obliczenie rdzenia i reduktów przy użyciu FBFS. Wyniki przedstawione w tabeli 2.7 pokazują, że ta strategia daje prawie 1,75 razy wyższe wartości niż BBFS i około 7,2 razy lepsze wyniki niż w przypadku Arrii.

Tabela 2.7: Czas obliczania reduktu [ms] z użyciem Zynq US+

Liczba wykonań	**BBFS**	**FBFS**
10	0.38	0.67
100	3.80	6.69
1000	38.13	66.94
10000	380.10	665.80

Przedstawione wyniki pokazują, że użycie tego samego projektu sprzętu na dwóch różnych urządzeniach może prowadzić do poprawy wydajności. Opisane eksperymenty dowiodły, że możliwe jest osiągnięcie 15,5 razy lepszych czasów (w przypadku BBFS) i 7,2 razy lepszych (w przypadku FBFS). Należy zauważyć, że zegar używany w Zynq jest 2 razy szybszy niż zegar używany w Nios, więc jeśli weź-

miemy tę różnicę pod uwagę, to współczynnik przyśpieszenia wynosi około 7,75 dla BBFS i 3,6 dla FBFS.

Podsumowanie

Zdefiniowano i przeanalizowano dwa algorytmy realizujące strategie przeszukiwania wszerz, oparte na podejściu „ślepym" oraz na częstotliwościach występowania atrybutów.

Zaprezentowano dwie architektury FPGA służące do znajdowania minimalnych reduktów. Na podstawie badań eksperymentalnych przedstawionych w niniejszym rozdziale można sformułować wniosek, że stosowanie układów programowalnych do wspomagania obliczeń reduktów determinuje ogromne korzyści w postaci zredukowania czasu wykonania algorytmu.

W porównaniu do komputera PC, obliczenia robione na układzie FPGA są o wiele wydajniejsze. Największą akceleracje można jednak zauważyć w przypadku FBFS. Dzieje się tak, ponieważ większość bloków w implementacji sprzętowej jest kombinacyjnych, dzięki czemu wyniki uzyskiwane są niemal natychmiast (tylko czas propagacji sygnału opóźnia obliczenia).

Ogromną zaletą FPGA jest możliwość wykorzystania sprzętowego bloku sortowania, dzięki czemu FBFS jest znacznie bardziej wydajny niż BBFS.

Bibliografia

Bakar, A. A., Sulaiman, M. N., Othman, M. i Selamat, M. H. (2002). Propositional satisfiability algorithm to find minimal reducts for data mining. *International Journal of Computer Mathematics*, *79*, 379–389.

Chen, D., Zhao, S., Zhang, L., Yang, Y. i Zhang, X. (2012). Sample pair selection for attribute reduction with rough set. *IEEE Transactions on Knowledge and Data Engineering*, *24*, 2080-2093.

Choromański, M., Grześ, T. i Hońko, P. (2019). Two FPGA devices in the problem of finding minimal reducts. W: K. Saeed, R. Chaki i V. Janev (red.), *Computer information systems and industrial management,* Springer, 410–420.

Choromański, M., Grześ, T. i Hońko, P. (2020). Breadth search strategies for finding minimal reducts: towards hardware implementation. *Neural Computing and Applications*, *32*, 14801–14816.

Czolombitko, M., i Stepaniuk, J. (2016). Attribute Reduction Based on MapReduce Model and Discernibility Measure. W: K. Saeed i W. Homenda (red.),

Computer Information Systems and Industrial Management: 15th IFIP TC8 International Conference, 9842, Springer, 55–66.

Degang, C., Changzhong, W. i Qinghua, H. (2007). A new approach to attribute reduction of consistent and inconsistent covering decision systems with covering rough sets. *Information Sciences*, *177*, 3500–3518.

Dong, Z., Sun, M. i Yang, Y. (2016). Fast algorithms of attribute reduction for covering decision systems with minimal elements in discernibility matrix. *International Journal of Machine Learning and Cybernetics*, *7*, 297–310.

Grześ, T., Kopczyński, M. i Stepaniuk, J. (2013). FPGA in rough set based core and reduct computation. W: P. Lingras, M. Wolski, C. Cornelis, S. Mitra i P. Wasilewski (red.), *Rough sets and knowledge technology,* Springer, 263–270.

Grzymala-Busse, J. (1991). An algorithm for computing a single covering. W: J. Grzymala-Busse (red.), *Managing uncertainty in expert systems,* Kluwer Academic Publishers.

Hu, Q., Yu, D., Liu, J. i Wu, C. (2008). Neighborhood rough set based heterogeneous feature subset selection. *Information Sciences*, *178*, 3577-3594.

Hu, Q., Yu, D., Xie, Z. i Liu, J. (2006). Fuzzy probabilistic approximation spaces and their information measures. *Transactions on Fuzzy Systems*, *14*, 191-201.

Hu, X., i Cercone, N. (1995). Learning in relational databases: A rough set approach. *Computational Intelligence*, *11*, 323-338.

Jensen, R., Shen, Q. i Tuson, A. (2005). Finding rough set reducts with SAT. W: *Rough sets, fuzzy sets, data mining, and granular computing, proceedings of 10th international conference,* Springer, 194–203.

Jia, X., Liao, W., Tang, Z. i Shang, L. (2013). Minimum cost attribute reduction in decision-theoretic rough set models. *Information Sciences*, *219*, 151-167.

Jiang, D.-L., Zhao, B., Wang, Y., Li, H.-W. i Yang, G.-Q. (2013). FPGA low level data mining based on cluster and rough set. *Guangxue Jingmi Gongcheng/Optics and Precision Engineering*, *21*, 233-238.

Jiang, Y., i Yu, Y. (2016). Minimal attribute reduction with rough set based on compactness discernibility information tree. *Soft Computing*, *20*, 2233–2243.

Jing, F., Yunliang, J. i Yong, L. (2017). Quick attribute reduction with generalized indiscernibility models. *Information Sciences*, *397-398*, 15 - 36.

Kanasugi, A., i Matsumoto, M. (2007). Design and implementation of rough rules generation from logical rules on FPGA Board. W: M. Kryszkiewicz, J. F. Peters, H. Rybinski i A. Skowron (red.), *Rough sets and intelligent systems paradigms,* Springer, 594–602.

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. (2014). FPGA in rough-granular computing: Reduct generation. W: *Proceedings of the 2014 IEEE/WIC/ACM In-*

ternational Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), IEEE, 364–370.

Kryszkiewicz, M. (1998). Rough set approach to incomplete information systems. *Inforamtion Sciences*, *112*, 39-49.

Kryszkiewicz, M. (2001). Comparative study of alternative type of knowledge reduction in inconsistent systems. *International Journal of Intelligent Systems*, *16*, 105-120.

Lewis, T., Perkowski, M. i Jozwiak, L. (1999). Learning in hardware: architecture and implementation of an FPGA-based rough set machine. W: *Informatics: Theory and practice for the new millennium: Proceedings of 25th euromicro conference,* 1, IEEE, 326-334.

Li, F., i Yang, J. (2016). A new approach to attribute reduction of decision information systems. W: Y. Qin, L. Jia, J. Feng, M. An i L. Diao (red.), *Proceedings of the 2015 international conference on electrical and information technologies for rail transportation: Transportation,* Springer, 557-564.

Liang, J., Mi, J., Wei, W. i Wang, F. (2013). An accelerator for attribute reduction based on perspective of objects and attributes. *Knowledge-Based Systems*, *44*, 90-100.

Liang, J., i Xu, Z. (2002). The algorithm on knowledge reduction in incomplete information systems. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *10*, 95-103.

Liu, G., Hua, Z. i Chen, Z. (2017). A general reduction algorithm for relation decision systems and its applications. *Knowledge-Based Systems*, *119*, 87-93.

Pawlak, Z. (1991). *Rough sets. theoretical aspects of reasoning about data*. Kluwer Academic, Dordrecht.

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

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

Qian, Y., Liang, J., Pedrycz, W. i Dang, C. (2010). Positive approximation: An accelerator for attribute reduction in rough set theory. *Artificial Intelligence*, *174*, 597-618.

Skahill, K. (2001). *Język VHDL.* Warszawa: Wydawnictwo Naukowo-Techniczne.

Skowron, A., i Rauszer, C. (1992). The discernibility matrices and functions in information systems. W: *Intelligent decision support,* Springer, 331–362.

Ślęzak, D. (2002). Approximate entropy reducts. *Fundamamenta Informaticae*, *53*, 365–390.

Stepaniuk, J. (1999). Rough set data mining of diabetes mellitus data. *Lecture Notes in Computer Science*, *1906*, 457—465.

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

Su, Y., i Guo, J. (2017). A novel strategy for minimum attribute reduction based on rough set theory and fish swarm algorithm. *Computational Intelligence and Neuroscience*, *2017*, 6573623:1–6573623:7.

Sun, G., Qi, X. i Zhang, Y. (2011). A FPGA-based implementation of Rough Set Theory. W: *Proceedings of the 2011 chinese control and decision conference,* 2561-2564.

Sun, G., Wang, H., Lu, J. i He, X. (2013). A FPGA-based discretization algorithm of continuous attributes in rough set. W: Y.-H. Kim i P. Yarlagadda (red.), 278-280, 1167-1173.

Swiniarski, R. (2001). Rough sets methods in feature reduction and classification. *International Journal of Applied Mathematics and Computer Science*, *11*, 565–582.

Swiniarski, R. W., i Skowron, A. (2003). Rough set methods in feature selection and recognition. *Pattern Recognition Letters*, *24*, 833-849.

Teng, S.-H., Lu, M., Yang, A.-F., Zhang, J., Nian, Y. i He, M. (2016). Efficient attribute reduction from the viewpoint of discernibility. *Information Sciences*, *326*, 297-314.

Thi, V. D., i Giang, N. L. (2013). A method for extracting knowledge from decision tables in terms of functional dependencies. *Cybernetics and Information Technologies*, *13*, 73-82.

Tiwari, K., i Kothari, A. (2015). Design and implementation of rough set coprocessor on FPGA. *International Journal of Innovative Computing, Information and Control*, *11*, 641-656.

Tiwari, K., i Kothari, A. (2016). Design of intelligent system for medical applications using rough set theory. *International Journal of Data Mining, Modelling and Management*, *8*, 279-301.

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*, 14-23.

Wang, C., He, Q., Chen, D. i Hu, Q. (2014). A novel method for attribute reduction of covering decision systems. *Information Sciences*, *254*, 181-196.

Wang, X., Yang, J., Peng, N. i Teng, X. (2005). Finding minimal rough set reducts with particle swarm optimization. W: D. Ślęzak, G. Wang, M. Szczuka, I. Düntsch i Y. Yao (red.), *Rough sets, fuzzy sets, data mining, and granular computing,* Springer, 451–460.

Wang, X., Yang, J., Teng, X., Xia, W. i Jensen, R. (2007). Feature selection based on rough sets and particle swarm optimization. *Pattern Recognition Letters*, *28*, 459–471.

Wei, W., Liang, J., Qian, Y., Wang, F. i Dang, C. (2010). Comparative study of decision performance of decision tables induced by attribute reductions. *International Journal of General Systems*, *39*, 813-838.

Wei, W., Liang, J., Wang, J. i Qian, Y. (2013). Decision-relative discernibility

matrices in the sense of entropies. *International Journal of General Systems*, *42*, 721-738.

Wroblewski, J. (1995). Finding minimal reducts using genetic algorithms. W: *Proceedings of the second annual join conf. on information sciences,* 186—189.

Xie, J., Shen, X., Liu, H. i Xu, X. (2013). Research on an incremental attribute reduction based on relative positive region. *Journal of Computational Information Systems*, *9*, 6621-6628.

Xu, N., Liu, Y. i Zhou, R. (2008). A tentative approach to minimal reducts by combining several algorithms. W: *Advanced intelligent computing theories and applications,* Springer, 118–124.

Yao, Y., i Zhao, Y. (2008). Attribute reduction in decision-theoretic rough set models. *Information Sciences*, *178*, 3356-3373.

Ye, D., i Chen, Z. (2002). A new discernibility matrix and the computation of a core. *Acta Electronica Sinica*, *30*, 1086-1088.

Zhang, W.-X., Mi, J.-S. i Wu, W.-Z. (2003). Approaches to knowledge reductions in inconsistent systems. *International Journal of Intelligent Systems*, *18*, 989-1000.

Zhang, X., Mei, C., Chen, D. i Li, J. (2013). Multi-confidence rule acquisition oriented attribute reduction of covering decision systems via combinatorial optimization. *Knowledge-Based Systems*, *50*, 187-197.