Streszczenie rozprawy doktorskiej

Wspomaganie decyzji oparte na sprzętowej realizacji metod zbiorów przybliżonych

Autor: mgr inż. Maciej Kopczyński

Promotor: prof. dr hab. Jarosław Stepaniuk

Promotor pomocniczy: dr inż. Tomasz Grześ

Wydział Informatyki Politechniki Białostockiej

Wprowadzenie

Rozprawa doktorska dotyczy wykorzystania teorii zbiorów przybliżonych (ang. *rough sets*) w systemach wspomagania decyzji opartych o sprzętowe implementacje algorytmów przetwarzania danych. Systemy wspomagania decyzji *DSS* (ang. *Decision Support System*) są systemami dostarczającymi informacji i wiedzy wykorzystywanymi przy podejmowaniu decyzji dotyczących procesów, zjawisk oraz w zagadnieniach klasyfikacji.

Największym problemem związanym z tworzeniem optymalnych systemów wspomagania decyzji jest czas potrzebny na przetwarzanie danych wejściowych, ze szczególnym uwzględnieniem dużych zbiorów danych, tzw. *big data*. *Big data* jest to termin odnoszący się do dużych, różnorodnych i zmiennych zbiorów danych, których przetwarzanie i analiza jest trudna, ale jednocześnie wartościowa, ponieważ może prowadzić do zdobycia nowej wiedzy [33]. Większość obecnie realizowanych operacji na danych, szczególnie z wykorzystaniem zbiorów przybliżonych, jest wykonywana za pomocą programowych implementacji algorytmów. Powszechnie stosowanymi sposobami na przyspieszenie ich działania jest wykorzystanie szybszych komputerów, próba zrównoleglenia wykonania lub zaprojektowanie nowych metod o mniejszej złożoności obliczeniowej.

Z uwagi na szybki rozwój architektur układów programowalnych otworzyła się nowa możliwość skrócenia czasu realizacji obliczeń związanych z przetwarzaniem zbiorów danych. Pojawienie się układów elektrycznie programowalnych, wśród nich *CPLD* (ang. *Complex Programmable Logic Array*) i *FPGA* (ang. *Field Programmable Gate Array*), stanowiło milowy krok w dziedzinie układów programowalnych. Układy te stają się coraz szybsze, a ich możliwości coraz większe. Możliwość przyspieszenia obliczeń związana z przeniesieniem całych algorytmów analizy zbiorów danych, bądź ich najbardziej złożonych obliczeniowo części do specjalnie zaprojektowanych jednostek sprzętowych może w znacznym stopniu skrócić czas wykonywania obliczeń.

Nie ma żadnych w pełni funkcjonalnych systemów realizujących działania związane ze zbiorami przybliżonymi w kompleksowej postaci, natomiast ta praca przedstawia kompleksowe podejście do sprzętowego wspomagania obliczeń z wykorzystaniem zbiorów przybliżonych uwzględniając układy sprzętowe zdolne do przetwarzania dużych zbiorów danych.

W streszczeniu rozprawy doktorskiej do opisu wybrane zostały trzy spośród dwunastu stworzonych układów sprzętowych:

- *ICB* (ang. *Indiscernibility Class Block*) – układ wyznaczający klasy nierozróżnialności. Był to pierwszy krok w zakresie realizacji sprzętowej jednej z metod zbiorów przybliżonych zrealizowany jako układ kombinacyjny,
- *Mixed-Core* – układ obliczający rdzeń. Był to kolejny etap na drodze do przetwarzania dużych zbiorów danych. Jest to układ sekwencyjny będący bazą do skalowania rozwiązań dla małych zbiorów danych w kierunku ich dużych odpowiedników,
- *HRG-LEM2* (ang. *Hardware Rules Generation – LEM2*) – układ obliczający reguły decyzyjne. Jest to jedna z najbardziej złożonych jednostek sprzętowych i pokazująca duże przyspieszenia w przetwarzaniu danych w porównaniu do równoważnej implementacji programowej.

Rozprawa doktorska powstała w wyniku realizacji projektu badawczego o nr 2012/07/B/ST6/01504 finansowanego ze środków *Narodowego Centrum Nauki*.

1. Informacje wstępne

1.1. Przegląd metod zbiorów przybliżonych

Zbiory przybliżone i ich teoria zostały wprowadzone na początku lat osiemdziesiątych przez prof. Zdzisława Pawlaka (1926-2006) jako metoda radzenia sobie z niepełnymi oraz sprzecznymi zbiorami informacji. Teoria zbiorów przybliżonych znalazła zastosowanie w analizie, przetwarzaniu i redukcji zbiorów danych. Jest wykorzystywana w programach medycznych, farmakologicznych, inżynierskich, bankowych, w zagadnieniach sterowania, rozpoznawania obiektów oraz w wielu innych dziedzinach [19].

W ramach tego streszczenia, ze względu na ograniczoną ilość miejsca, pokazane zostaną trzy wybrane metody sprzętowego wspomagania przetwarzania danych z wykorzystaniem zbiorów przybliżonych. Kompletna rozprawa doktorska obejmuje dużo szerszy zakres zrealizowanych operacji.

1.1.1. Operacje podstawowe

Wszystkie powstałe algorytmy przetwarzania danych z wykorzystaniem metod zbiorów przybliżonych wykorzystują w mniejszym lub większym stopniu pojęcia podstawowe.

Niech U oznacza skończony i niepusty zbiór obiektów nazwany uniwersum, zaś A określa skończony i niepusty zbiór atrybutów. Każdy atrybut $a \in A$ jest funkcją

$$a: U \rightarrow V_a$$

gdzie V_a jest zbiorem wszystkich możliwych wartości a. V_a można nazwać domeną atrybutu a. Zapis $a(x)$, gdzie $a \in A$ i $x \in U$, oznacza wartość atrybutu a dla obiektu x.

Definicja 1.1.

Para $IS = (U, A)$ określa system informacyjny *IS*. Bardzo często system *IS* przyjmuje postać tabelaryczną, która jest wówczas nazywana tablicą decyzyjną *DT*.

Definicja 1.2.

Tablica decyzyjna *DT* jest to para $(U, A \cup \{d\})$, gdzie *U* jest uniwersum, *A* jest niepustym zbiorem atrybutów warunkowych, zaś $d \notin A$ jest atrybutem decyzyjnym.

Tablica decyzyjna *DT* jest strukturą ułatwiającą zastosowanie metod zbiorów przybliżonych do analizy danych obiektów, uwzględniając ich podział na klasy [16].

Zbiór $X_v = \{x \in U: d(x) = v\}$ nazywa się klasą decyzyjną tabeli decyzyjnej.

W systemach decyzyjnych zbiór atrybutów *A* dzieli się na dwie rozłączne części – zbiór atrybutów warunkowych *B* oraz atrybut decyzyjny *d*:

$$A = B \cup \{d\}$$

Definicja 1.3.

Każdy podzbiór *B* określa binarną relację *B-nierozróżnialności* tworzącą zbiór obiektów nierozróżnialnych względem określonego zbioru atrybutów warunkowych *B*:

$$IND(B) = \{(x, y) \in U \; x \; U: \forall_{a \in B} \; a(x) = a(y)\}$$

Relacja *IND*(*B*) określa podział zbioru *U*, co określa się jako $U \; / \; IND(B)$.

Definicja 1.4.

Zbiór obiektów znajdujących się w relacji nierozróżnialności z obiektem $x \in U$ ze względu na zbiór atrybutów warunkowych B jest określany jako $I_B(x)$ i jest nazywany *klasą nierozróżnialności* lub *klasą abstrakcji*. Tak więc:

$$I_B(x) = \{y \in U: (x, y) \in IND(B)\}$$

oraz

$$U \; / \; IND(B) \; = \; \{I_B(x): x \in U\}$$

Definicja 1.5.

Para AS_B = (*U*, *IND*(*B*)) jest standardową przestrzenią aproksymacji dla systemu informacyjnego *IS* = (*U*, *A*), gdzie $B \subseteq A$.

Wykorzystując tę przestrzeń można zdefiniować dolną aproksymację.

Definicja 1.6.

Dolna aproksymacja dla dowolnego podzbioru obiektów $X \subseteq U$ jest zdefiniowana w następujący sposób:

$$LOW(AS_B, X) = \{x \in U: I_B(x) \subseteq X\}$$

W wielu przypadkach istnieje potrzeba analizy tablicy decyzyjnej z uwagi na różnice wartości występujące na poszczególnych atrybutach pomiędzy różnymi parami obiektów. Do tego celu można wykorzystać macierze rozróżnialności. Zostały po raz pierwszy wprowadzone przez A. Skowrona i C. Rauszer w [23]. Macierz ta określa atrybuty, dla których występują różnice

w wartościach dla każdej pary obiektów w *U* pod warunkiem, że obiekty te należą do różnych klas decyzyjnych.

Definicja 1.7.

Macierz rozróżnialności oznaczana jako *M*(*A*) to macierz kwadratowa o rozmiarze *n* x *n*, gdzie *n* jest równe liczbie obiektów, zaś każda komórka macierzy c_{ij} opisana jest jako:

$$c_{ij} = \{a \in A, (x, y) \in U : a(x) \neq a(y) \ \wedge d(x) \neq d(y)\}$$

Ponieważ macierz rozróżnialności jest symetryczna względem głównej przekątnej oraz główna przekątna zawiera zbiory puste, to jej pełny zapis można zrealizować za pomocą macierzy trójkątnej.

1.1.2. Redukty

Często spotykanym problemem w przypadku zbiorów danych wyrażonych za pomocą np. tablic decyzyjnych jest nadmiarowość ich opisu, przez co konieczne staje się usunięcie zbędnych atrybutów warunkowych opisujących zbiór obiektów z zachowaniem podstawowych właściwości tej tablicy. W ten sposób otrzymywana jest tablica decyzyjna zawierająca takie same aproksymacje jak tablica oryginalna, jednak opisana jest za pomocą mniejszej liczby atrybutów.

Definicja 1.8.

Niech $B \subset A$ oraz $a \in B$.

Wprowadzając poniższe zależności:

- a jest *zbędny* w *B*, jeśli $IND(B) = IND(B - \{a\})$, w przeciwnym wypadku a jest *niezbędne* w *B*,
- zbiór B jest *niezależny*, jeśli wszystkie jego atrybuty są niezbędne,

można powiedzieć, że podzbiór $B' \subset B$ jest *reduktem* (ang. *reduct*) *R*, jeśli B' jest niezależny i $IND(B') = IND(B)$ [17].

1.1.3. Rdzenie

Pojęcie rdzenia bardzo mocno wiąże się z przedstawionym w poprzednim podrozdziale pojęciem reduktu.

Definicja 1.9.

Niech $B \subset A$. *Rdzeń* (ang. *core*) *C* jest to część wspólna wszystkich możliwych reduktów. Zapisując powyższe stwierdzenie formalnie:

$$Core(B) = \bigcap Red(B)$$

gdzie $Red(B)$ jest zbiorem wszystkich możliwych reduktów [17].

Rdzeń jest zbiorem wszystkich niezbędnych atrybutów zbioru *B*. Ponieważ rdzeń jest częścią wspólną wszystkich możliwych reduktów, to każdy atrybut należący do rdzenia występuje w każdym redukcie. W tym sensie rdzeń jest zbiorem najważniejszych atrybutów warunkowych i żaden z tych atrybutów nie może być usunięty bez osłabienia możliwości klasyfikacji. Więcej informacji na temat rdzeni można znaleźć np. w [19] lub w [24].

Rdzeń może być wyliczony bezpośrednio z definicji poprzez obliczenie wszystkich możliwych reduktów, jednak w praktyce takie rozwiązanie nie jest wykorzystywane ze względu na to, że problem obliczania reduktów należy do problemów NP-trudnych. Z tego względu metody obliczania rdzenia wykorzystują inne zależności, niż bezpośrednie obliczanie wszystkich reduktów. Jeden ze sposobów polega na wykorzystaniu macierzy rozróżnialności w sposób pośredni, czyli poprzez wyliczanie pojedynczych jej wierszy. Pojedynczy wiersz macierzy wyznacza się poprzez porównanie wybranego obiektu z tablicy decyzyjnej ze wszystkimi pozostałymi obiektami, które się w niej znajdują. Algorytm został nazwany *CORE-IDM* (ang. *CORE – Indirect Discernibility Matrix*).

Pseudokod 1.1. Algorytm *CORE-IDM*

WEJŚCIE: tablica decyzyjna $DT = \{U, A \cup \{d\}\}$

WYJŚCIE: rdzeń $C \subset A$

```
1.  C := ∅
2.  for x ∈ U do
3.    for y ∈ U do
4.      if d(x) ≠ d(y) then
5.        count := 0
6.        for a ∈ A do
7.          if a(x) ≠ a(y) then
8.            count := count + 1
9.            candidate := a
10.         end if
11.       end for
12.       if count = 1 and candidate ∉ C then
13.         C := C ∪ {candidate}
14.       end if
15.     end if
16.   end for
17. end for
```

1.1.4. Reguły decyzyjne i klasyfikacja

Generowanie reguł decyzyjnych jest jednym z najważniejszych elementów w tworzeniu systemów analizy danych [21].

Definicja 1.10.

Niech $\{a_1, a_2, \ldots, a_n\} \subseteq A$. *Reguła decyzyjna* to wyrażenie o następującej postaci

$$(a_1, v_1) \wedge (a_2, v_2) \wedge \ldots \wedge (a_n, v_n) \rightarrow (d, v_d)$$

gdzie $v_i \in V_{a_i}$ (i = 1, 2, …, n), zaś $v_d \in V_d$.

Istnieje bardzo wiele metod tworzenia zbiorów reguł decyzyjnych. Jedna z grup algorytmów wykorzystuje tablice decyzyjne [16]. Przykładem takiej metody jest algorytm *LEM2* stworzony przez J. W. Grzymałę-Busse i opisany w [4]. W celu łatwiejszego zrozumienia tego algorytmu wprowadzone zostaną następujące dodatkowe oznaczenia:

- t – para (a, v),
- $[t]$ – zbiór obiektów pasujących do deskryptora $t = (a, v)$,
- $[T]$ – zbiór obiektów pasujących do zbioru deskryptorów $t \in T$, czyli $[T] = \bigcap_{t \in T}[t]$.

Poniżej zaprezentowany jest pseudokod algorytmu *HRG-LEM2* (ang. *Hardware Rules Generation – LEM2*) bazującego na algorytmie *LEM2*.

Pseudokod 1.2. Algorytm *HRG-LEM2*

WEJŚCIE: tablica decyzyjna $DT = \{U, A \cup \{d\}\}$

WYJŚCIE: globalny zbiór reguł *GR*

1. *GR* := ∅
1. **for** $v_d \in V_d$ **do**
2. *vdSet* := $\{x \in U : d(x) = v_d\}$
3. *lowerApprox* := oblicz dolną aproksymację dla zbioru *vdSet*
4. *G* := *lowerApprox*
5. *LR* := ∅
6. **while** $G \neq \emptyset$ **do**
7. *T* := ∅
8. $T_G := \{t : [t] \cap G \neq \emptyset\}$
9. **while** *T* = ∅ **or** $[T] \nsubseteq$ *lowerApprox* **do**
10. *t* := wybierz $t \in T_G$ z maksymalną wartością $|[t] \cap G|$. Jeśli istnieje więcej takich *t*, to wybierz pierwsze z minimalną wartością $|[t]|$
11. *T* := *T* ∪ $\{t\}$
12. *G* := [t] ∩ *G*
13. $T_G := \{t : [t] \cap G \neq \emptyset\} - T$
14. **end while**
15. **for** $t \in T$ **do**
16. **if** $[T - \{t\}] \subseteq lowerApprox$ **then**
17. *T* **:=** *T* **–** {*t*}
18. **end if**
19. **end for**
20. *LR* **:=** *LR* ∪ {*T*}
21. *G* := *lowerApprox* – $\bigcup_{T \in LR}[T]$
22. **end while**
23. **for** *T* ∈ *LR* **do**

24. **if** $\bigcup_{T' \in LR-\{T\}}[T']$ = *lowerApprox* **then**
25. *LR* := *LR* – {*T*}
26. **end if**
27. **end for**
28. *GR* := *GR* ∪ *LR*
29. **end for**

1.2. Sprzętowe systemy cyfrowe

Systemy cyfrowe są nieodłączoną częścią otaczającego nas świata, niezależnie od tego jaka działalność człowieka jest rozpatrywana. Jest to element tworzący również każde rozwiązanie informatyczne na najniższym jego poziomie – poziomie sprzętowym. System cyfrowy może być stworzony z wielu układów cyfrowych, jak również może być zamknięty w pojedynczym układzie. Ze względu na ich działanie dzieli się je na dwie kategorie [22]:

- układy kombinacyjne – charakteryzują się tym, że stan wyjść zależy wyłącznie od stanu wejść. Stan wyjść opisują jedynie funkcje boolowskie. W układach tych nie występuje sprzężenie zwrotne,
- układy sekwencyjne – charakteryzują się tym, że stan wyjść zależy od stanu wejść oraz od poprzedniego stanu, zwanego stanem wewnętrznym, pamiętanego w zespole rejestrów (wewnętrznej pamięci).

1.2.1. Przegląd rodzajów układów programowalnych

Programowalne układy logiczne *PLD* (ang. *Programmable Logic Devices*) są grupą układów scalonych pozwalającą realizować złożone systemy cyfrowe. Najważniejszą cechą układów *PLD* jest to, że ich funkcjonalność nie jest określana na etapie produkcji, ale definiowana przez inżyniera. Ta cecha odróżnia tą grupę układów od układów typu *ASIC* (ang. *Application Specific Integrated Circuit*), które są zaprojektowane do wykonywania zadanej na etapie projektowania i utrwalonej w procesie produkcji funkcji systemu cyfrowego.

Współczesne układy *PLD* mogą zostać podzielone na dwie kategorie [13]:

- *CPLD* (ang. *Complex Programmable Logic Devices*) – są to układy programowalne złożone z bloków funkcjonalnych połączonych wzajemnie matrycą *SW* (ang. *Switch Matrix*),
- *FPGA* (ang. *Field Programmable Gate Array*) – są to układy składające się z programowalnych bloków logicznych *CLB* (ang. *Configurable Logical Block*) rozmieszczonych w matrycy i połączonych ze sobą za pomocą traktów połączeniowych (ang. *routing channels*) oraz programowalnych matryc kluczy połączeniowych znajdujących się w miejscu krzyżowania się traktów poziomych i pionowych.

Do opisywania funkcji układów programowalnych najczęściej używa się języków opisu sprzętu *HDL* (ang. *Hardware Description Language*). Najbardziej popularne obecnie języki typu *HDL* to *VHDL* oraz *Verilog* [22].

1.2.2. Procesory typu *softcore*

Procesor typu *softcore* jest rdzeniem mikroprocesora, który może zostać w całości stworzony z wykorzystaniem syntezy logicznej. Tego typu procesory są implementowane

w wielu różnych rodzajach układów programowalnych, np. *FPGA* czy *CPLD*. Rdzenie *softcore* mogą być syntezowalne zarówno w drogich jak i tanich układach scalonych [1]. Na świecie istnieje bardzo wiele rozwiązań procesorów typu *softcore*. Do najpopularniejszych należą m.in. *Nios II* rozwijany przez *Altera* i *MicroBlaze* rozwijany przez *Xilinx*.

1.3. Obecny stan wiedzy

Do chwili obecnej powstało wiele rozwiązań, głównie programowych, wykorzystujących metody zbiorów przybliżonych. Wiele z nich zostało stworzonych jako aplikacje wykorzystywane w ramach badań naukowych, jednak istnieją też takie, które skutecznie działają w rozwiązaniach komercyjnych. Przykładami rozwiązań programowych wykorzystywanych do analizy danych są: *LERS* [30], *RSES* [29], *Rosetta* [28], pakiet *RoughSets* do systemu *R* [27] czy rozwiązania firmy *Infobright* [26].

Niewielką część, w porównaniu do rozwiązań programowych, stanowią implementacje sprzętowe. W wielu przypadkach są to pojedyncze moduły, bądź ich projekty, realizujące wybrane metody zbiorów przybliżonych. Nie ma żadnych w pełni funkcjonalnych systemów realizujących działania związane ze zbiorami przybliżonymi w kompleksowej postaci, natomiast ta praca przedstawia kompleksowe podejście do sprzętowego wspomagania obliczeń z wykorzystaniem zbiorów przybliżonych uwzględniając układy zdolne do przetwarzania dużych zbiorów danych.

Profesor Z. Pawlak w 2004 r. zaproponował architekturę przykładowego procesora *RSP* (ang. *Rough Set Processor*). Jego idea opiera się na wykorzystaniu podstawowych przybliżonych granul danych oraz zależności pomiędzy nimi [18].

Rybiński i Muraszkiewicz w 1984 r. stworzyli opis zbiorów przybliżonych z wykorzystaniem sieci komórkowych (ang. *cellular networks*). Idea sieci komórkowych opisana jest w pracy [14]. Na tej podstawie w 1994 r. opracowana została koncepcja rozwiązania sprzętowego nazwana *PRSComp* (ang. *Parallel Rough Sets COMPuter*) [15].

Lewis, Perkowski i Jóźwiak w 1999 r. opisali ideę samouczącego systemu bazującego na zbiorach przybliżonych i wykorzystującego wcześniej przedstawione sieci komórkowe. Ogólna idea rozwiązania opiera się na przeprowadzeniu uczenia systemu w warstwie oprogramowania, zaś na podstawie wyników tego procesu tworzona jest warstwa sprzętowa [10]. Częściowa implementacja opisanego powyżej systemu została zrealizowana na płycie *DEC-PERLE-1* przez Dharmadhikari, Ngo i Lewisa w 1999 r. Stworzony został kod *VHDL* opisujący obliczanie górnej oraz dolnej aproksymacji [2].

Kanasugi i Yokoyama w 2001 r. opracowali ideę systemu wspierającego w sposób sprzętowy minimalizację funkcji logicznych stworzonych na podstawie macierzy rozróżnialności. Zminimalizowane funkcje reprezentują reguły decyzyjne [6].

Kanasugi wraz z Matsumoto w 2007 r. zaimplementowali opisane powyżej rozwiązanie w układzie *FPGA*. Pozwoliło to uzyskać skrócenie czasu obliczeń o blisko 700 razy dla tego samego zbioru danych (128 obiektów opisanych na 2 032 atrybutach) uwzględniając różnicę w szybkości taktowania układu programowalnego (50 MHz) i procesora komputera PC (3 400 MHz) [5].

Sun w 2011 r. opracował koncepcję rozwiązania sprzętowego prezentując nowy algorytm wykrywania uszkodzeń bazujący na metodach zbiorów przybliżonych oraz na algorytmach ewolucyjnych. W pracy [31] zaprezentowany został model statku powietrznego o nieliniowej charakterystyce wykorzystany w symulacji działania rozwiązania.

Tiwari i Kothari w 2015 r. opracowali koprocesor wspomagający w sposób sprzętowy kilka operacji wykonywanych na zbiorach przybliżonych [32]. Rozwiązanie zostało przetestowane na danych dotyczących błędnego działania układów scalonych bardzo wielkiej skali integracji *VLSI* (ang. *Very-Large-Scale Integration*). Rozmiar przetwarzanych danych był ograniczony do bloków o rozmiarze 128 obiektów opisanych za pomocą 128 atrybutów.

Rodriguez-Diez i współautorzy w 2015 r. zaprezentowali platformę programowo-sprzętową umożliwiającą obliczanie testorów [20].

2. Opracowane implementacje sprzętowe metod zbiorów przybliżonych

W ramach prac badawczych rozpoczętych w 2011 r. opracowane zostały implementacje sprzętowe wielu różnych metod zbiorów przybliżonych. Rozwiązania pokrywają zakres od operacji podstawowych (jak np. obliczanie aproksymacji) poprzez wyznaczanie rdzeni i reduktów, a na generowaniu zbiorów reguł decyzyjnych kończąc. Część zrealizowanych sprzętowych modułów wykonawczych występuje w kilku wersjach (architekturach) wykonujących te same działania.

Większość zaproponowanych rozwiązań była projektowana do przetwarzania niewielkich zbiorów danych, jednak w kilku przypadkach zaimplementowane zostały również moduły potrafiące przetwarzać duże zbiory danych składające się z wielu milionów obiektów. W każdym przypadku przyjęte jest założenie, że dane wejściowe są przekształcone do postaci binarnej poprzez zakodowanie wszystkich wartości opisujących poszczególne atrybuty.

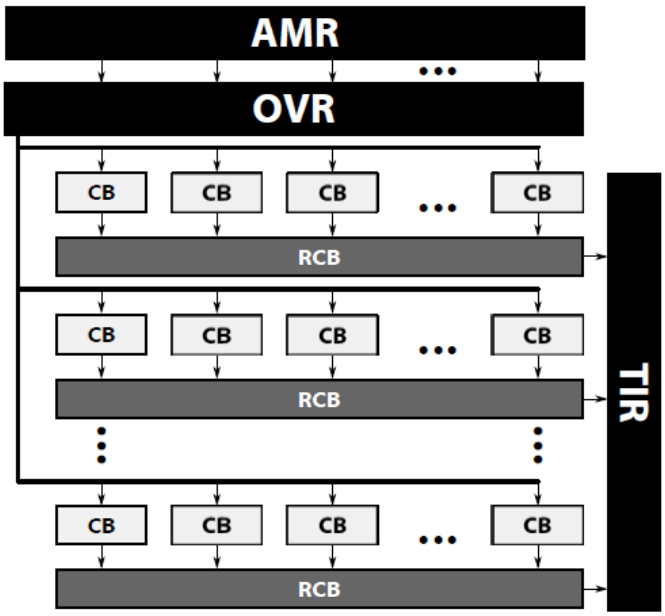
2.1. Operacje podstawowe

W ramach prac związanych ze stworzeniem układów sprzętowych wspomagających realizację operacji podstawowych wykonane zostały następujące jednostki:

- *ICB* (ang. *Indiscernibility Class Block*) – układ wyznaczający klasy nierozróżnialności,
- *LACB* (ang. *Lower Approximation Calculation Block*) – układ obliczający dolną aproksymację zadanego zbioru,
- *DMCB* (ang. *Discernibility Matrix Calculation Block*) – układ tworzący macierz rozróżnialności.

Prace naukowe zespołu badawczego, w którym pracuje autor, a dotyczące tej części rozwiązania sprzętowego to [25] oraz [21].

Jednym ze stworzonych układów jest moduł realizujący wyznaczanie klasy nierozróżnialności – *ICB* (ang. *Indiscernibility Class Block*). Zaprojektowany został on jako układ kombinacyjny, zapewniający w pełni równoległe wykonywanie obliczeń poprzez jednoczesne przetworzenie całego zbioru obiektów i atrybutów ich opisujących. Diagram blokowy modułu pozwalającego wyznaczyć klasy rozróżnialności pokazany jest na Rys. 2.1.



Rys. 2.1 Diagram blokowy modułu ICB

Moduł ma zdefiniowane następujące wejścia:

- *Attribute Mask Register* (*AMR*) – jest to wejście definiujące, które atrybuty opisujące obiekty mają zostać wykorzystane w obliczeniach. Wejście te ma szerokość równą liczbie wszystkich atrybutów warunkowych. Wartość 0 oznacza, że dany atrybut nie jest przetwarzany,
- *Object Value Register* (*OVR*) – wejście te określa wartość obiektu, dla którego obliczana jest klasa nierozróżnialności. Wejście te ma szerokość równą liczbie bitów opisujących dany obiekt.

Wyjście bloku sprzętowego stanowi *Table Indiscernibility Register* (*TIR*). Tworzy ono wektor bitowy opisujący które z obiektów w wejściowej tablicy decyzyjnej należą do klasy abstrakcji względem obiektu zawartego w *OVR*. Jego szerokość jest równa liczbie obiektów przetwarzanych jednorazowo przez moduł *ICB*. Wyjście *TIR* reprezentuje zbiór binarny.

Układ *ICB* zbudowany jest z bloków komparatorów *CB*, które realizują porównania pomiędzy poszczególnymi atrybutami wszystkich obiektów oraz obiektu wybranego do porównania zawartego w *OVR* jednocześnie uwzględniając wartość znajdującą się na wejściu *AMR*. Wyniki porównania trafiają do wewnętrznego rejestru *Row Comparator Block* (*RCB*). Jeśli rejestr *RCB* zawiera tylko wartości 0, to w wyjściu *TIR* w komórce odpowiadającej danemu obiektowi zapisywana jest wartość 1 mówiąca o przynależności tego obiektu do obliczanej klasy nierozróżnialności.

Obliczenie wszystkich klas abstrakcji wymaga podstawienia do wejścia *OVR* kolejnych słów bitowych opisujących wszystkie obiekty z tablicy decyzyjnej.

2.2. Rdzenie

W ramach prac związanych ze stworzeniem układów sprzętowych wspomagających operację obliczania rdzenia, zaimplementowanych zostało pięć wersji jednostek sprzętowych różniących się wewnętrzną architekturą oraz wielkością przetwarzanych danych:

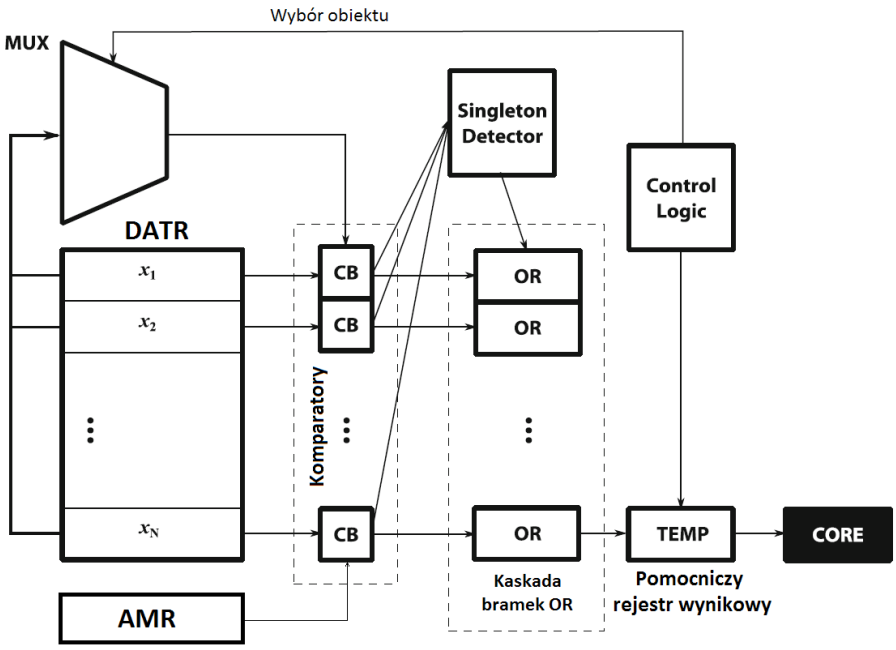
- *Cascade-Core* – blok obliczający rdzeń za pomocą kaskady bramek typu *or*,

- *Big-OR-Core* – moduł obliczający rdzeń za pomocą wielowejściowych bramek typu *or*,
- *Mixed-Core* – układ obliczający rdzeń za pomocą kaskadowo połączonych bramek typu *or*,
- *HIDM-Core* (ang. *Hardware Indirect Discerniblity Matrix Core*) – jednostka wyliczająca rdzeń dla dużych zbiorów danych oparta o kaskadowo połączone bramki typu *or*,
- *CT-Core* (ang. *Counting Tables Core*) – moduł wyliczający rdzeń dla dużych zbiorów danych oparty o algorytm tablic zliczających (ang. *counting tables*).

Każdy z modułów jako źródło danych wejściowych wykorzystuje tablicę decyzyjną.

Prace naukowe zespołu badawczego, w którym pracuje autor, a dotyczące tej części rozwiązania sprzętowego to [3], [7], [8] oraz [9].

Jednym ze zrealizowanych układów wyliczających rdzeń jest *Mixed-Core*. Blok zawiera w swojej strukturze zarówno układy kombinacyjne, jak i część sekwencyjną. Diagram blokowy modułu przedstawiony jest na Rys. 2.2.



Rys. 2.2 Schemat blokowy modułu Mixed-Core

Moduł ma zdefiniowane następujące wejścia:

- *Data Register* (*DATR*) – jest to wejście, przez które przekazywana jest cała tablica decyzyjna. Wejście te ma szerokość równą iloczynowi liczby wszystkich obiektów oraz szerokości bitowej opisu pojedynczego obiektu uwzględniającego zbiór atrybutów warunkowych oraz atrybutu decyzyjnego,
- *Attribute Mask Register* (*AMR*) – jest to wejście definiujące, które atrybuty opisujące obiekty mają zostać wykorzystane w obliczeniach. Wejście te ma szerokość równą liczbie wszystkich atrybutów warunkowych. Wartość 0 oznacza, że dany atrybut nie jest przetwarzany,
- *clock* – przekazuje sygnał zegarowy synchronizujący pracę układu,
- *reset* – wejście ustawiające stan początkowy układu.

Moduł ma zdefiniowane następujące wyjścia:

- *CORE* – wyjście opisujące obliczony rdzeń. Jego szerokość równa jest liczbie atrybutów warunkowych. Wartość logiczna 1 oznacza, że dany atrybut należy do obliczonego rdzenia,
- *ready* – sygnalizuje logiczną wartością 1 zakończenie wykonywania obliczeń.

Podstawowe elementy wykonawcze jednostki sprzętowej to:

- *Comparator Block* (*CB*) – blok komparatorów odpowiedzialny za wyliczanie różnic w wartościach atrybutów warunkowych par obiektów z uwzględnieniem przynależności badanych obiektów do różnych klas decyzyjnych. Liczba bloków jest równa liczbie obiektów w tablicy decyzyjnej. Jednostka dodatkowo uwzględnia wartość zapisaną w rejestrze *AMR* określającą atrybuty do przetworzenia,
- *Singleton Detector* (*SD*) – moduł, którego zadaniem jest wykrywanie pojedynczej wartości logicznej równej 1 na którymkolwiek bicie w przekazanym na jego wejście słowie bitowym. Blok może przetwarzać słowa bitowe o różnej długości,
- *kaskada bramek OR* – jednostka złożona z bramek *or*, których liczba jest równa liczbie wszystkich komórek macierzy rozróżnialności wyrażonej jako macierz trójkątna. Każda z bramek ma trzy zestawy wejść: jeden z nich pobiera wartość z komórki *CB* obliczającej różnice na atrybutach pomiędzy parą obiektów, drugi połączony jest z wyjściem poprzedniej bramki w kaskadzie, zaś trzeci łączy się z wyjściem bloku *SD*.
- *MUX* – multiplekser wybierający z tablicy decyzyjnej obiekty dla których obliczane będą różnice w wartościach atrybutów. Multiplekser przekazuje wybierane wartości do bloku komparatorów *CB,*
- *Control Logic* – jednostka sterująca pracą całego układu *Mixed-Core*. Odpowiada za synchronizację pracy poszczególnych elementów systemu cyfrowego. Generuje również ostateczną postać rdzenia,
- *rejestr pomocniczy TEMP* – przechowuje wyliczaną w poszczególnych cyklach zegarowych postać rdzenia. Po zakończeniu obliczeń wyliczona wartość przekazywana jest do wyjścia *CORE*.

Idea działania układu opiera się na algorytmie *CORE-IDM* przedstawionym w podrozdziale 1.1.3. Każdy takt zegara powoduje wyliczenie jednej kolumny macierzy rozróżnialności przez zbiór komparatorów *CB*. Dzięki wykorzystaniu takiego rozwiązania nie ma potrzeby obliczania i przechowywania w pamięci całej macierzy rozróżnialności. Wybór obiektu do porównania ze wszystkimi pozostałymi obiektami z tablicy decyzyjnej realizowany jest poprzez multiplekser *MUX* sterowany blokiem *Control Logic*. Wynik porównania reprezentujący jedną kolumnę macierzy rozróżnialności przekazywany jest do kaskady bramek *or* oraz do układu *SD*, który sprawdza, czy w przekazywanym słowie bitowym reprezentującym porównanie pary obiektów występuje tylko jeden bit o wartości logicznej równej 1. Po każdym takcie zegarowym odnalezione singletony zapisywane są w rejestrze *TEMP* reprezentującym wyliczany rdzeń. Zakończenie wszystkich iteracji oraz przekazanie obliczonego rdzenia do wyjścia *CORE* sygnalizowane jest wartością logiczną 1 na wyjściu *ready*. Do przetworzenia zbioru danych celem wyliczenia rdzenia wymagana liczba taktów zegara sterującego równa jest wartości kwadratu liczby obiektów w tablicy decyzyjnej.

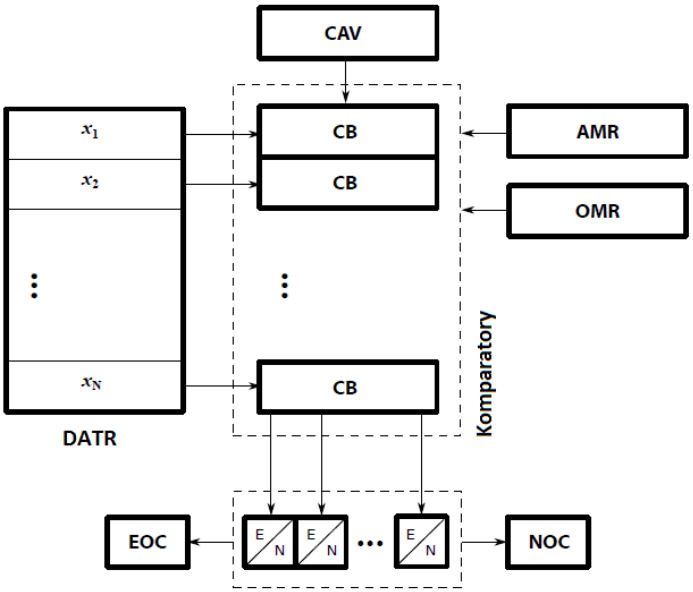
2.3. Reguły decyzyjne i klasyfikacja

W ramach prac związanych ze stworzeniem układu sprzętowego wspomagającego wyznaczanie reguł decyzyjnych zaimplementowany został moduł *HRG-LEM2* (ang. *Hardware*

Rules Generation – LEM2). Prace naukowe zespołu badawczego, w którym pracuje autor, a dotyczące tej części rozwiązania sprzętowego to [11] i [12].

Moduł *HRG-LEM2* zaprojektowany został jako blok zawierający w swojej strukturze zarówno układy kombinacyjne, jak i część sekwencyjną. Powstały układ umożliwia przetwarzanie dużych zbiorów danych liczących miliony obiektów. *HRG-LEM2* składa się z modułu sterującego oraz dwóch sprzętowych bloków wykonawczych: *avCounter* i *dtComparator*.

Rolą modułu *avCounter* jest obliczenie wartości określającej liczbę obiektów w tablicy decyzyjnej spełniających warunek występowania określonej wartości na zadanym atrybucie warunkowym oraz, równolegle, obiektów należących do zdefiniowanego zbioru. Warunek występowania wartości podany jest jako para (a, v), gdzie a to atrybut warunkowy, zaś v to jego wartość. Działanie układu odpowiada wykonaniu linii 11 w pseudokodzie 1.2 zamieszczonym w rozdziale 1.1.4. *avCounter* zaprojektowany został jako układ kombinacyjny wykonujący obliczenia w jednym cyklu zegara taktującego. Diagram blokowy modułu przedstawiony jest na Rys. 2.3.



Rys. 2.3 Schemat blokowy modułu avCounter

Moduł ma zdefiniowane następujące wejścia:

- *Data Register* (*DATR*) – jest to wejście, przez które przekazywana jest tablica decyzyjna, bądź jej część, w zależności od rozmiaru przetwarzanych danych. Wejście te ma szerokość równą iloczynowi liczby przetwarzanych obiektów oraz szerokości bitowej opisu pojedynczego obiektu uwzględniającego zbiór atrybutów warunkowych oraz atrybutu decyzyjnego,
- *Attribute Mask Register* (*AMR*) – wejście to opisuje, które atrybuty mają zostać wykorzystane w obliczeniach. Szerokość wejścia jest równa liczbie atrybutów warunkowych. Wartość 1 oznacza, że dany atrybut jest przetwarzany,
- *Object Mask Register* (*OMR*) – jest to wejście określające, które obiekty zawarte w tablicy decyzyjnej mają zostać przetworzone. Pełni rolę zbioru obiektów zakodowanego za pomocą wartości binarnych. Szerokość wejścia równa jest liczbie obiektów opisanych w *DATR*. Wartość 0 oznacza pominięcie obiektu,

- *Conditional Attributes Values* (*CAV*) – na to wejście podawane są wartości atrybutów warunkowych reprezentowane przez pary (a, v). Szerokość wejścia jest równa szerokości bitowej słowa opisującego atrybuty warunkowe pojedynczego obiektu w tablicy decyzyjnej.

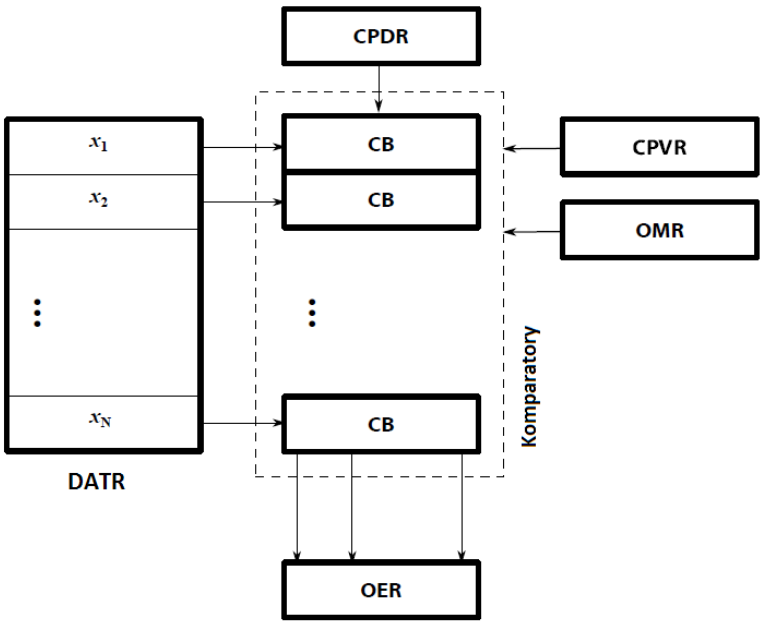
Jednostka sprzętowa zawiera następujące wyjścia:

- *Equal Objects Counter* (*EOC*) – wyjścia zwracające liczbę obiektów spełniających warunki zdefiniowane jako pary (a, v) przekazane przez wejście *CAV* w zbiorze obiektów określonym przez wejście *OMR*. Liczba wyjść *EOC* jest równa liczbie atrybutów warunkowych,
- *Not-equal Objects Counter* (*NOC*) – wyjścia zwracające liczbę obiektów spełniających warunki zdefiniowane jako pary (a, v) przekazane przez wejście *CAV* w całej tablicy decyzyjnej przekazanej przez wejście *DATR*. Liczba wyjść *NOC* jest równa liczbie atrybutów warunkowych.

Podstawowymi jednostkami budującymi moduł *avCounter* są:

- komparatory *CB* – pojedynczy komparator porównuje ze sobą obiekt pobrany z tablicy decyzyjnej oraz przekazane przez wejście *CAV* pary (a, v). Liczba komparatorów jest równa liczbie obiektów przekazanych przez wejście *DATR*,
- bloki *E/N* – ich rolą jest przetwarzanie sygnałów otrzymanych z komparatorów i generowanie na ich podstawie wartości określających liczbę obiektów spełniających warunki przekazane przez wejście *CAV*.

Drugim istotnym elementem wchodzącym w skład jednostki *HRG-LEM2* jest moduł *dtComparator*. Rolą tej jednostki jest równoległe tworzenie opisanych binarnie zbiorów obiektów spełniających pojedynczy warunek lub zestaw wielu warunków. Warunki te podane są jako para (a, v). Działanie układu odpowiada wyznaczeniu zbioru obiektów w liniach 10, 13, 17, 22 i 25 w pseudokodzie 1.2. *dtComparator* zaprojektowany został jako układ kombinacyjny wykonujący obliczenia w jednym cyklu zegara taktującego. Diagram blokowy modułu przedstawiony jest na Rys. 2.4.



Rys. 2.4 Schemat blokowy modułu dtComparator

W porównaniu do jednostki *avCounter*, moduł *dtComparator* ma zdefiniowane dodatkowe wejścia:

- *Conditional Parts Values Register* (*CPVR*) – rolą tego wejścia jest przekazanie do modułu części warunkowych reguł tworzonych przez algorytm *HRG-LEM2*. Szerokość wejścia jest równa szerokości bitowej słowa opisującego atrybuty warunkowe pojedynczego obiektu w tablicy decyzyjnej,
- *Conditional Parts Defined Register* (*CDVR*) – wejście to opisuje w sposób bitowy które części warunkowe reguły są podawane na wejściu *CPVR*. Szerokość wejścia jest równa liczbie atrybutów warunkowych.

Wyjściem jednostki *dtComparator* jest *Objects Equality Register* (*OER*). Opisuje ono zakodowany binarnie zbiór wynikowy spełniający części warunkowe reguły przekazane przez wejście *CPVR*. Szerokość wyjścia jest równa liczbie obiektów przechowywanych w tablicy decyzyjnej opisanej na wejściu *DATR*.

Jednostki sprzętowe *avCounter* i *dtComparator* wspomagają bazujący na algorytmie *HRG-LEM2* opisanym w rozdziale 1.1.4 moduł *HRG-LEM2* przy wykonywaniu operacji zliczania liczby obiektów spełniający warunki opisane parami (a, v) oraz wyznaczania zbiorów obiektów spełniających części warunkowe tworzonych reguł decyzyjnych. Ogólny sposób działania obydwu układów jest podobny. Bloki komparatorów *CB* połączone z każdym obiektem w tablicy decyzyjnej *DATR* porównują wartości opisujące obiekty z warunkami opisanymi parami (a, v). Wynik działania komparatorów z uwzględnieniem masek atrybutów oraz przekazanych zapisanych binarnie podzbiorów tablicy decyzyjnej przekazywany jest do jednostek zliczających *N/E* w przypadku *avCounter* lub bezpośrednio do wyjścia *OER* dla modułu *dtComparator*. Działanie całego algorytmu kontrolowane jest przez układ nadrzędny względem modułów *avCounter* i *dtComparator*.

3. Wyniki eksperymentalne

W ramach prowadzonych prac badawczych wszystkie stworzone rozwiązania sprzętowe zostały sprawdzone na czterech zbiorach danych o różnych licznościach obiektów. Otrzymane wyniki jednoznacznie wskazują na znaczne przyspieszenia układów sprzętowych w zakresie przetwarzania danych wejściowych w porównaniu do równoważnych implementacji programowych.

Zastosowana metodologia badawcza była skupiona głównie na porównaniu czasów niezbędnych do wykonania określonych operacji powiązanych ze zbiorami przybliżonymi na tych samych zbiorach danych przez bloki sprzętowe oraz równoważne pod kątem funkcjonalności implementacje programowe.

3.1. Wykorzystane oprogramowanie oraz sprzęt

Opis układów sprzętowych tworzony był z wykorzystaniem języka *VHDL* w środowisku *Quartus II* firmy *Altera* w wersji 13.1. Poszczególne jednostki sprzętowe testowane były w oprogramowaniu *ModelSim* stworzonym przez firmę *Mentor Graphics* dostarczanym razem z pakietem *Quartus II*. Architektura rozwiązań wykorzystujących rdzenie *softcore* oparta została o procesor *NIOS II*. Kod źródłowy opisujący działanie rdzenia *NIOS II*

pisany był w *NIOS II Sotware Build Tools* bazującym na środowisku programistycznym *Eclipse* również dostarczanym wraz z pakietem *Quartus II*. Wykorzystanym językiem programowania, zarówno dla rozwiązań uruchamianych w rdzeniu *softcore* jak i na komputerze PC, był język *C*. Kod źródłowy tworzony dla komputera PC pisany był w środowisku programistycznym *Eclipse* w wersji *Mars.1* z wtyczką *CDT* zapewniającą wsparcie dla języka *C*. Do kompilacji kodu źródłowego wykorzystany został 32-bitowy kompilator *GNU GCC* w wersji 4.8.1.

Układem *FPGA* wykorzystanym w pracach badawczych był *Stratix III* firmy *Altera* oznaczony symbolem *EP3SL150F1152C2N*. Został on umieszczony na płycie rozwojowej *DE-3* wyprodukowanej przez firmę *Terasic*.

Czasy działania poszczególnych układów sprzętowych były badane za pomocą liczników umieszczonych w strukturze *FPGA* oraz z wykorzystaniem oscyloskopu *LeCroy waveSurfer 104MXs-B*. Komputer PC wykorzystany do pomiaru czasów działania implementacji programowych wyposażony był 8 GB pamięci *RAM* oraz w 4-rdzeniowy procesor *Intel Core i7 3632QM* taktowany w trybie *Turbo* zegarem o częstotliwości 3,2 GHz.

3.2. Testowe zbiory danych

W ramach prac badawczych wykorzystywane zostały cztery zbiory danych. Pierwszy z nich nazwany *Diabetes* opisuje przypadki osób chorych na cukrzycę typu pierwszego (łac. *diabetes mellitus typi 1*) i został stworzony przez J. Stepaniuka. Drugim zbiorem danych jest *Poker Hand* utworzony przez R. Cattrala i F. Oppachera. Wymienione powyżej zbiory danych posłużyły w badaniach wszystkich opisanych w rozprawie doktorskiej rozwiązaniach za wyjątkiem układu *HMD-Cuts* (nie został opisany w streszczeniu) służącym do wyznaczania zbioru cięć niezbędnych dla operacji dyskretyzacji tablicy decyzyjnej. Zbiorem danych wykorzystanym w testowaniu *HMD-Cuts* był stworzony przez R. A. Fishera *Iris* zawierający dane dotyczące gatunków kwiatów irysa oraz utworzony przez I-Cheng Yeh'a *Concrete Slump* opisujący wytrzymałość betonu.

Dwa pierwsze z przedstawionych zbiorów zostały wykorzystane do stworzenia nowych tablic decyzyjnych składających się zarówno z mniejszej jak i większej niż nominalna liczby obiektów.

3.3. Rezultaty badań eksperymentalnych

3.3.1. Bloki realizujące operacje podstawowe

W ramach implementacji sprzętowych związanych z operacjami podstawowymi metod zbiorów przybliżonych przetestowane zostały trzy układy:

- *ICB* (ang. *Indiscernibility Class Block*) – układ wyznaczający klasy nierozróżnialności,
- *LACB* (ang. *Lower Approximation Calculation Block*) – układ obliczający dolną aproksymację zadanego zbioru,
- *DMCB* (ang. *Discernibility Matrix Calculation Block*) – układ tworzący macierz rozróżnialności.

W ramach tego streszczenia opisany są wyniki otrzymane dla bloku *ICB*. W badaniach porównawczych czasu wykonania implementacji programowej i sprzętowej wykorzystane zostały zbiory *Diabetes* i *Poker Hand* o liczności 107 obiektów. Obliczona została pojedyncza klasa nierozróżnialności dla pierwszego obiektu z uwzględnieniem wszystkich atrybutów warunkowych zgodnie z zależnościami podanymi w rozdziale 1.1.1. Wyniki czasowe pokazane są w Tab. 3.1. Kolumna t_S opisuje czas wykonania dla realizacji programowej algorytmu, zaś

t_H odnosi się do czasu wykonania algorytmu w układzie sprzętowym. Ostatnia kolumna określa współczynnik przyspieszenia C zdefiniowany jako $C = \frac{t_S}{t_H}$.

Tab. 3.1 Czas wykonania implementacji programowej oraz sprzętowej dla bloku ICB

Liczba obiektów	t_S	t_H	C
–	*[µs]*	*[µs]*	–
Zbiór *Diabetes*			
15	2,302	0,0024	959,167
30	2,579	0,0029	889,310
45	3,643	0,0032	1 138,438
60	4,263	0,0035	1 218,000
90	6,690	0,0041	1 631,707
107	12,008	0,0059	2 035,254
Zbiór *Poker Hand*			
15	1,037	0,0024	432,083
30	2,395	0,0029	825,862
45	2,678	0,0032	836,875
60	3,419	0,0035	976,857
90	4,721	0,0041	1 151,463
107	6,349	0,0059	1 076,102

Wykorzystanie zasobów układu programowalnego *FPGA* z serii *Stratix III* dla modułu *ICB* przedstawione jest w Tab. 3.2.

Tab. 3.2 Wykorzystanie zasobów struktury FPGA dla bloku ICB

Liczba obiektów	**Bloki *LE***	**Wyprowadzenia**
15	312	1 095
30	1 264	2 190
45	2 812	3 285
60	5 082	4 380
90	11 646	6 570
107	16 746	7 811

Analiza otrzymanych wyników czasowych pokazuje znaczące przyspieszenie układu sprzętowego w porównaniu do rozwiązania programowego. Dla zbioru *Diabetes* współczynnik przyspieszenia wynosi maksymalnie 2 035 razy, zaś w przypadku zbioru *Poker Hand* jest maksymalnie równy 1 076.

Czas wykonania dla układu sprzętowego jest niezależny od typu przetwarzanych danych co wynika z jego kombinacyjnego charakteru. Czas ten zależy jedynie od szerokości słowa opisującego pojedynczy obiekt, zaś w tym przypadku jego szerokość dla obydwu zbiorów jest jednakowa. Współczynnik przyspieszenia układu sprzętowego rośnie proporcjonalnie do liczby obiektów. Jest to związane z tym, że implementacja sprzętowa wykonuje porównania pomiędzy obiektami oraz wartościami ich atrybutów w sposób równoległy, zaś rozwiązanie programowe wymaga sekwencyjnego przechodzenia po zbiorze danych. Realizacja modułu *ICB* jako układu kombinacyjnego przekłada się na znaczące wykorzystanie zasobów układu *FPGA* pod względem liczby bloków *LE*. Uniemożliwia to wykorzystanie tego rodzaju rozwiązania dla większych zbiorów danych, ze względu na ograniczenia zasobów dostępnych na rynku układów programowalnych. Liczba wyprowadzeń nie stanowi istotnego problemu, gdyż większość z nich podłączona jest do bloków pamięci wewnątrz układu *FPGA*.

W przypadku implementacji programowej, czas wykonania dla zbioru *Poker Hand* jest krótszy niż dla *Diabetes* ze względu na mniejszą liczbę atrybutów.

3.3.2. Rdzenie

W ramach implementacji sprzętowych związanych z operacją wyznaczania rdzenia przetestowanych zostało pięć rozwiązań:

- *Cascade-Core* – blok obliczający rdzeń za pomocą kaskady bramek typu *or*,
- *Big-OR-Core* – moduł obliczający rdzeń za pomocą wielowejściowych bramek typu *or*,
- *Mixed-Core* – układ obliczający rdzeń za pomocą kaskadowo połączonych bramek typu *or*,
- *HIDM-Core* (ang. *Hardware Indirect Discerniblity Matrix Core*) – jednostka wyliczająca rdzeń dla dużych zbiorów danych oparta o kaskadowo połączone bramki typu *or*,
- *CT-Core* (ang. *Counting Tables Core*) – moduł wyliczający rdzeń dla dużych zbiorów danych oparty o algorytm tablic zliczających (ang. *counting tables*).

W ramach tego streszczenia opisane są wyniki otrzymane dla bloku *Mixed-Core*. W badaniach porównawczych czasu wykonania wykorzystane zostały zbiory *Diabetes* i *Poker Hand.* Zegar o częstotliwości 50 MHz taktujący układ sprzętowy został przekazany z oscylatora na płycie rozwojowej. Przy ocenie wyników końcowych należy wziąć pod uwagę fakt, że układ *FPGA* taktowany był zegarem mającym częstotliwość 64 razy mniejszą niż komputer PC (taktowanie 3,2 GHz) w związku z tym ostatecznie otrzymane wyniki współczynników przyspieszeń mogą zostać wymnożone przez 64 celem otrzymania miarodajnego wyniku.

Wyniki czasowe dla modułu *Mixed-Core* pokazane są w Tab. 3.3. Kolumna t_S opisuje czas wykonania dla programowej realizacji algorytmu rozpoczynającego działanie na tablicy decyzyjnej. Kolumna t_{H_M} odnosi się do czasów wykonania algorytmu *CORE-IDM* w układzie sprzętowym. Ostatnia kolumna określa współczynnik przyspieszenia C zdefiniowany jako $C_M = \frac{t_S}{t_{H_M}}$.

Tab. 3.3 Czas wykonania implementacji programowej oraz sprzętowej dla bloku Mixed-Core

Liczba obiektów	t_S	t_{H_M}	C_M
–	*[μs]*	*[μs]*	–
Zbiór *Diabetes*			
15	112	0,572	195
30	420	1,171	358
45	983	1,774	554
60	1 737	2,411	720
90	4 075	3,582	1 137
107	5 990	4,260	1 406
Zbiór *Poker Hand*			
15	102	0,572	178
30	347	1,171	296
45	815	1,774	459
60	1 448	2,411	601
90	3 458	3,582	965
107	4 823	4,260	1 132

Wykorzystanie zasobów układu programowalnego *FPGA* z serii *Stratix III* dla modułu *Mixed-Core* przedstawione jest w Tab. 3.4.

Tab. 3.4 Wykorzystanie zasobów struktury FPGA dla bloku Mixed-Core

Liczba obiektów	**Bloki *LE***	**Wyprowadzenia**
15	1 252	205
30	1 979	795
45	3 418	1 185
60	3 541	1 575
90	5 335	2 355
107	7 171	2 797

Analiza otrzymanych wyników czasowych pokazuje przyspieszenie układu sprzętowego wspomagającego wyliczanie rdzenia w porównaniu do rozwiązania programowego. Należy pamiętać o 64 razy wolniejszym zegarze taktującym jednostkę sprzętową w porównaniu do

zegara komputera PC. Współczynnik ten nie został uwzględniony w obliczeniach. Dla zbioru *Diabetes* współczynnik przyspieszenia wynosi maksymalnie 1 406 (89 984 po uwzględnieniu różnicy częstotliwości taktowania), zaś w przypadku zbioru *Poker Hand* współczynnik przyspieszenia jest maksymalnie równy 1 132 (72 448 po uwzględnieniu różnicy częstotliwości taktowania). W każdej sytuacji widoczny jest wzrost współczynnika przyspieszenia w powiązaniu ze wzrostem wielkości przetwarzanego zbioru danych.

Wykorzystanie zasobów struktury *FPGA* przy uwzględnieniu bloków *LE* oraz liczby wyprowadzeń jest względnie małe dla układu *Mixed-Core*. Wynika to z mniejszego stopnia skomplikowania układu *Mixed-Core* w porównaniu do układów kombinacyjnych. Biorąc pod uwagę skalowalność rozwiązania, mniejszy układ daje możliwość przetwarzania większych zbiorów danych oraz pozwala wykorzystać układ *FPGA* do wspomagania obliczeń w zakresie dodatkowych operacji na danych. Liczba wyprowadzeń nie stanowi istotnego ograniczenia, gdyż większość z nich podłączona jest do rejestrów lub bloków pamięci wewnątrz układu *FPGA*.

Różnice w czasie wykonania analizowanych zbiorów danych dla implementacji programowej jest spowodowana różną liczbą przetwarzanych atrybutów warunkowych.

3.3.3. Reguły decyzyjne i klasyfikacja

W ramach implementacji sprzętowych związanych z operacją generowania zbioru reguł decyzyjnych przetestowany został układ *HRG-LEM2* (*Hardware Rules Generation – LEM2*).

W badaniach porównawczych czasu wykonania implementacji programowej oraz odpowiadającego jej rozwiązania sprzętowego wykorzystane zostały zbiory *Diabetes* i *Poker Hand* o liczności do 1 000 000 obiektów. Dla obydwu zbiorów obliczony został zbiór reguł decyzyjnych z uwzględnieniem wszystkich atrybutów warunkowych zgodnie z algorytmem podanym w rozdziale 1.1.4. Zegar o częstotliwości 50 MHz taktujący układ sprzętowy został przekazany z oscylatora na płycie rozwojowej. Przy ocenie wyników końcowych należy wziąć pod uwagę fakt, że układ *FPGA* taktowany był zegarem mającym częstotliwość 64 razy mniejszą niż komputer PC (taktowanie 3,2 GHz) w związku z tym ostatecznie otrzymane wyniki współczynników przyspieszeń mogą zostać wymnożone przez 64 celem otrzymania miarodajnego wyniku.

Wyniki czasowe pokazane są w Tab. 3.5. Kolumna t_S opisuje czas wykonania dla programowej realizacji algorytmu, zaś kolumna t_H odnosi się do czasu wykonania algorytmu w układzie sprzętowym. Ostatnia kolumna określa współczynnik przyspieszenia C zdefiniowany jako $C = \frac{t_S}{t_H}$. Przedstawione wyniki nie uwzględniają czasu potrzebnego na obliczenie dolnych aproksymacji poszczególnych klas decyzyjnych. Z uwagi na bardzo długi czas działania implementacji programowej dla zbiorów o liczności 1 000 000 obiektów, wyniki dla tego wiersza zostały wyznaczone poprzez aproksymację funkcją wielomianową.

Wykorzystanie zasobów układu programowalnego *FPGA* z serii *Stratix III* dla modułu *HRG-LEM2* jest niezależne od rozmiaru przetwarzanych danych, gdyż blok sprzętowy przetwarza fragment danych o stałym rozmiarze. Układ sprzętowy wykorzystuje 15 679 bloków *LE* oraz 171 wyprowadzeń. Wartość ta uwzględnia zasoby niezbędne dla procesora *NIOS II* sterującego pracą stworzonej jednostki sprzętowej.

Tab. 3.5 Czas wykonania implementacji programowej oraz sprzętowej dla bloku HRG-LEM2

Liczba obiektów	t_S	t_H	C
–	*[s]*	*[s]*	–
Zbiór *Diabetes*			
1 000	4,057	0,798	5,084
2 500	18,304	1,385	13,216
5 000	187,784	3,308	56,767
10 000	1 265,410	6,537	193,577
25 000	8 101,699	12,886	628,721
50 000	62 002,117	37,008	1 675,371
100 000	264 164,271	89,297	2 958,261
250 000	1 712 942,929	339,162	5 050,520
500 000	6 935 214,680	1 065,438	6 509,260
1 000 000	27 908 410,629	3 679,876	7 584,063
Zbiór *Poker Hand*			
1 000	0,361	0,069	5,244
2 500	1,635	0,124	13,148
5 000	16,620	0,277	60,051
10 000	111,835	0,571	195,866
25 000	722,089	1,148	629,042
50 000	5 186,988	3,261	1 590,498
100 000	23 220,455	7,469	3 109,115
250 000	145 131,541	29,408	4 935,072
500 000	572 771,477	96,699	5 923,252
1 000 000	2 352 553,246	325,329	7 231,314

Analiza otrzymanych wyników czasowych pokazuje znaczące przyspieszenie układu sprzętowego wspomagającego generowanie reguł decyzyjnych w porównaniu do rozwiązania programowego. Należy pamiętać o 64 razy wolniejszym zegarze taktującym jednostkę sprzętową w porównaniu do zegara komputera PC. Współczynnik ten nie został uwzględniony w obliczeniach. Dla zbioru *Diabetes* współczynnik przyspieszenia wynosi maksymalnie 7 584 (485 376 po uwzględnieniu różnicy częstotliwości taktowania), zaś w przypadku zbioru *Poker Hand* współczynnik przyspieszenia jest maksymalnie równy 7 231 (462 804 po uwzględnieniu różnicy częstotliwości taktowania). Współczynniki przyspieszenia rosną wraz ze wzrostem liczby obiektów tworzących wejściowe zbiory danych. Jest to spowodowane faktem

wykonywania przez układy sprzętowe obliczeń na danych w sposób równoległy, jak również implementacją niektórych struktur danych reprezentujących zbiory obiektów jako wektorów binarnych. Wyniki czasowe nie uwzględniają wyliczania dolnych aproksymacji dla wszystkich klas decyzyjnych. Operacja ta wykonywana była w sposób programowy.

Wykorzystanie zasobów układu *FPGA* jest na tyle niewielkie, że w obrębie struktury pozostaje duża ilość wolnych zasobów, które mogą zostać wykorzystane do realizacji układów wspomagających analizę danych metodami zbiorów przybliżonych.

Różnice w czasie wykonania dla analizowanych zbiorów danych jest spowodowana różną liczbą przetwarzanych atrybutów warunkowych oraz charakterystyką danych rzutującą na wykonywanie pętli algorytmu.

Podsumowanie i wnioski

W ramach stworzonej pracy przedstawione zostały jednostki sprzętowe wspomagające przetwarzanie danych z wykorzystaniem metod zbiorów przybliżonych w zakresie podstawowych operacji jak i bardziej złożonych algorytmów. Głównymi rezultatami pracy są:

- przeprojektowane pod kątem wykorzystania w implementacjach sprzętowych algorytmy związane z metodami zbiorów przybliżonych, co wielu wypadkach wymagało stworzenia alternatywnych metod związanych z sekwencją przetwarzania danych,
- projekty dwunastu jednostek sprzętowych wspomagających analizę danych z użyciem metod zbiorów przybliżonych,
- implementacje wcześniej zaprojektowanych modułów sprzętowych w języku opisu sprzętu *VHDL*,
- wykonanie testów porównawczych zaimplementowanych rozwiązań sprzętowych i równoważnych implementacji programowych,
- stworzenie podstaw metod transformacji do układów sprzętowych algorytmów realizowanych do tej pory w językach wysokiego poziomu,
- implementacja systemu wspomagania decyzji *GROSHEC*, który wykorzystuje w procesie przetwarzania danych moduły sprzętowe realizujące metody zbiorów przybliżonych,
- implementacja w systemie *GROSHEC* modułu programowego służącego do badań porównawczych pod kątem czasu wykonania i realizującego operacje tożsame do rozwiązań sprzętowych.

Najważniejsze wnioski, które można sformułować na podstawie zrealizowanych prac badawczych są następujące:

1. Czasy wykonania operacji na takich samych zbiorach danych przy zastosowaniu implementacji sprzętowych algorytmów realizowanych do tej pory w sposób programowy z wykorzystaniem języków wysokiego poziomu są znacznie krótsze.
2. Układy sekwencyjne, mimo ich wolniejszego działania, są preferowane względem układów kombinacyjnych ze względu na dużo bardziej optymalne wykorzystanie struktury programowalnej oraz łatwiejszą ich adaptację do zadań wymagających przetwarzania dużych zbiorów danych.

3. Implementacje sprzętowe cechują się dużym stopniem zrównoleglenia obliczeń na większych niż w przypadku rozwiązań programowych słowach opisujących pojedyncze obiekty. Dzięki temu skrócenie czasu przetwarzania danych jest bardziej zauważalne, niż w przypadku zastosowania równoległości obliczeń w przypadku implementacji programowych.
4. Złożoność obliczeniowa rozwiązań programowych i sprzętowych jest taka sama przy uwzględnieniu faktu, że przetwarzane są duże zbiory danych wymagające dekompozycji zbioru wejściowego zarówno ze względu na atrybuty jak i obiekty. Należy jednak zauważyć, że układ sprzętowy jest w stanie prowadzić obliczenia jednocześnie na wielu obiektach i wielu atrybutach, dzięki czemu mimo tej samej złożoności obliczeniowej, będzie przetwarzał określone dane szybciej.

Praca ta jest potwierdzeniem możliwości wykorzystania sprzętowego wspomagania analizy danych z wykorzystaniem metod zbiorów przybliżonych. Otrzymane w ramach prac badawczych wyniki potwierdzają poprawność działania zaproponowanych metod. Wspomaganie sprzętowe operacji na danych, niezależnie od stosowanej metody, jest alternatywnym sposobem na zwiększanie wydajności systemów wspomagania decyzji.

Bibliografia

[1] Altera IP. (2015). Altera.

[2] Dharmadhikari, M., Ngo, V. i Lewis, T. (1999). Design of a SIMD Computer for Rough Set Theory.

[3] Grześ, T., Kopczyński, M. i Stepaniuk, J. (2013). FPGA in Rough Set Based Core and Reduct Computation. W Lecture Notes in Computer Science - Vol. 8171 : Lecture Notes in Artificial Intelligence (strony 263-270). Springer-Verlag.

[4] Grzymała-Busse, J. (2005). Rule Induction. W Data Mining and Knowledge Discovery Handbook (strony 277-294). Springer US.

[5] Kanasugi, A. i Matsumoto, M. (2007). Design and Implementation of Rough Rules Generation from Logical Rules on FPGA. W Rough Sets and Intelligent Systems Paradigms (strony 594-602). Springer Berlin Heidelberg.

[6] Kanasugi, A. i Yokoyama, A. (2001). A Basic Design for Rough Set Processor. W The 15th Annual Conference of Japanese Society for Artificial Intelligence.

[7] Kopczyński, M., Grześ, T. i Stepaniuk, J. (2014). Generating Core in Rough Set Theory: Design and Implementation on FPGA. W Rough Sets and Intelligent Systems Paradigms (strony 209-216). Springer International Publishing.

[8] Kopczyński, M., Grześ, T. i Stepaniuk, J. (2015). Computation of Cores in Big Datasets: An FPGA Approach. W Rough Sets and Knowledge Technology (strony 153-163). Springer International Publishing.

[9] Kopczyński, M., Grześ, T. i Stepaniuk, J. (2015). Core for Large Datasets : Rough Sets on FPGA. W Concurrency, Specification & Programming: 24th International Workshop: CS&P 2015 (Tom 1, strony 235-246). University of Rzeszow.

[10] Lewis, T., Perkowski, M. i Jóźwiak, L. (1999). Learning in hardware: Architecture and implementation of an FPGA-based rough set machine. 25th EUROMICRO Conference. Proceedings. Mediolan.

[11] Kopczyński, M., Grześ, T. i Stepaniuk, J. (w druku). Rough Sets Based LEM2 Rules Generation Supported by FPGA. Fundamenta Informaticae.

[12] Kopczyński, M., Grześ, T. i Stepaniuk, J. (w druku). Hardware Supported Rough Sets Based Rules Generation for Big Datasets. W Computer Information Systems, Lecture Notes in Computer Science. Springer.

[13] Łuba, T. (2003). Synteza układów cyfrowych. Wydawnictwa Komunikacji i Łączności.

[14] Muraszkiewicz, M. (1984). Sieci komórkowe do przetwarzania danych nienumerycznych. Prace IINTE nr 54.

[15] Muraszkiewicz, M. i Rybiński, H. (1994). Towards a Parallel Rough Sets Computer. W Rough Sets, Fuzzy Sets and Knowledge Discovery (strony 434-443). Springer London.

[16] Pawlak, Z. (1991). Rough Sets. Theoretical Aspects of Reasoning about Data. Dordrecht: Springer Netherlands.

[17] Pawlak, Z. (1998). Rough set elements. W Rough Sets in Knowledge Discovery 1 (strony 10-30). Physica-Verlag Heidelberg.

[18] Pawlak, Z. (2004). Elementary rough set granules: Toward a rough set processor. W Rough-Neural Computing - Techniques for Computing with Words (strony 5-13). Springer Berlin Heidelberg.

[19] Pawlak, Z. i Skowron, A. (2007). Rudiments of rough sets. Information Sciences(177), strony 3-27.

[20] Rodriguez-Diez, V., Martinez-Trinidad, J., Carrasco-Ochoa, J., Lazo-Cortes, M., Feregrino-Uribe, C. i Cumplido, R. (2015). A fast hardware software platform for computing irreducible testors. Expert Systems with Applications, strony 9612–9619.

[21] Rutkowski, L. (2009). Metody i techniki sztucznej inteligencji. Warszawa: Wydawnictwo Naukowe PWN.

[22] Skahill, K. (2006). Język VHDL. Projektowanie programowalnych układów logicznych. Wydawnictwa Naukowo-Techniczne.

[23] Skowron, A. i Rauszer, C. (1992). The discernibility matrices and functions in information systems. Intelligent Decision Support, Handbook of Applications and Advances of the Rough Sets Theory, strony 331-362.

[24] Stepaniuk, J. (2008). Rough-Granular Computing in Knowledge Discovery and Data Mining. New York: Springer.

[25] Stepaniuk, J., Kopczyński, M. i Grześ, T. (2013). The First Step Toward Processor for Rough Set Methods. Fundamenta Informaticae(127), strony 429-443.

[26] Strona internetowa firmy Infobright. (2016). Pobrano z lokalizacji https://infobright.com/

[27] Strona internetowa repozytorium CRAN dla RoughSets pod R. (2016). Pobrano z lokalizacji https://cran.r-project.org/web/packages/RoughSets/index.html

[28] Strona internetowa Rosetta. (2016). Pobrano z lokalizacji http://www.lcb.uu.se/tools/rosetta/

[29] Strona internetowa RSES. (2016). Pobrano z lokalizacji http://logic.mimuw.edu.pl/~rses/

[30] Strona internetowa twórcy LERS. (2016). Pobrano z lokalizacji http://people.eecs.ku.edu/~jerzy/LERS.html

[31] Sun, G., Qi, X. i Zhang, Y. (2011). A FPGA-based implementation of Rough Set Theory. W Chinese Control and Decision Conference (CCDC) (strony 2561 - 2564). IEEE.

[32] Tiwari, K. i Kothari, A. (2015). Design and Implementation of Rough Set Co-Processor on FPGA. International Journal of Innovative Computing, Information and Control, strony 641-656.

[33] Zikopoulos, P. i Eaton, C. (2011). Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. McGraw-Hill Osborne Media.