Wydział Elektrotechniki, Informatyki i Telekomunikacji
Uniwersytet Zielonogórski

ROZPRAWA DOKTORSKA

AUTOMATYCZNA DEKOMPOZYCJA SPECYFIKACJI BEHAWIORALNEJ SPRZĘTOWO-PROGRAMOWEGO MIKROSYSTEMU CYFROWEGO

mgr inż. Andrzej Stasiak

Promotor: prof. dr hab inż.M.Adamski

Zielona Góra 2006

Spis treści

Spis rysunków

Spis tabel

Spis ważniejszych symboli i skrótów

γ	-	funkcja opisująca w interpretowanej sieci Petriego wyjścia typu Mealy'ego
λ	-	funkcja opisująca w interpretowanej sieci Petriego wyjścia typu Moore'a lub funkcja wyjścia w automacie skończonym FSM
ρ	-	funkcja opisująca wejścia w interpretowanej sieci Petriego
$\aleph$	-	liczby naturalne (0, 1, 2, ...)
$\bullet p$, $p\bullet$	-	zbiór tranzycji wejściowych, wyjściowych danego miejsca p
$\bullet t$, $t\bullet$	-	zbiór miejsc wejściowych, wyjściowych danej tranzycji t
$[M\ t\rangle$	-	wystąpienie tranzycji t przy znakowaniu M
ASIC	-	układy scalone wielkiej skali integracji - ang. Application Specific Integrated Circuit
C^-	-	macierz incydencji wejściowej (poprzedników) - ang. Backward Incidence Matrix
C^+	-	macierz incydencji wyjściowej (następników) - ang. Forward Incidence Matrix
CAD	-	komputerowe wspomaganie prac projektowych - ang. Computer Aided Design
CAE	-	komputerowe wspomaganie prac inżynierskich - ang. Computer Aided Engineering
CDFG	-	graf przepływu sterowania i danych - ang. Control Data Flow Graph
CFG	-	graf przepływu sterowania - ang. Control Flow Graph
CFSM	-	automat skończony stosowany w systemie POLIS - ang. Co-Design Finite State Machine
CPLD	-	układy programowalne o strukturze hierarchicznej - ang. Complex Programmable Logic Designs
DFG	-	graf przepływu danych - ang. Data Flow Graph
F	-	relacja przepływu sieci Petriego
FPGA	-	układy programowalne o strukturze komórkowej - ang. Field Programmable Gate Arrays
FSM	-	automat skończony - ang. Finite State Machine

FSMD	-	automat skończony z ścieżką danych - ang. Finite State Machine with Datapath
HCFSM	-	automat skończony z hierarchią i równoległością - ang. Hierarchical Concurrent Finite State Machine
HDL	-	języki opisu sprzętu - Hardware Description Language
IP	-	własność intelektualna - ang. Intellectual Property
IPN	-	interpretowana sieć Petriego
K	-	funkcja pojemności miejsc
M, M_i	-	znakowanie sieci
M_0	-	znakowanie początkowe sieci
M	-	zbiór miejsc sieci Petriego
p, pi	-	miejsce w sieci Petriego
PIP	-	własność intelektualna o zmiennych parametrach (sparametryzowane moduły) - ang. Parameterized Intellectual Property
PLD	-	układy programowalne przez użytkownika - ang. Programmable Logic Devices
PN	-	znakowana sieć Petriego
PNSF, PNSF2	-	tekstowe formaty opisu sieci Petriego - ang. Petri Net Specification Format
PSM	-	maszyna stanowo-programowa - ang. Program State Machine
RTL	-	poziom przesłań międzyrejestrowych - ang. Register Transfer Level
SM	-	sieć Petriego typu automatowego
SOC	-	system w jednym układzie scalonym - ang. System on Chip
T	-	zbiór tranzycji sieci Petriego
t, ti	-	tranzycja w sieci Petriego
V	-	funkcja wagi (krotności) łuków
Verilog HDL	-	język opisu sprzętu (standard IEEE-1364)
VHDL	-	język opisu sprzętu (standard IEEE-1076)

ROZDZIAŁ PIERWSZY

1. Wstęp

W rozdziale przedstawione zostały motywacje podjęcia tematu, cele oraz zakres przeprowadzonych prac i badań w dziedzinie akceleracji sprzętowej systemów osadzonych oraz metodologii projektowania zintegrowanego sprzętowo-programowego mikrosystemu cyfrowego. Przedstawiono zagadnienia dotyczące dekompozycji systemowej. Sformułowano problem naukowo-techniczny podjęty w rozprawie oraz możliwe kierunki jego rozwiązania ze wskazaniem na nowatorstwo pracy. Omówiono stan prac zrealizowanych przez środowisko akademickie oraz najnowsze technologie przemysłu, w których wyniki badań autora mogą zostać wdrożone.

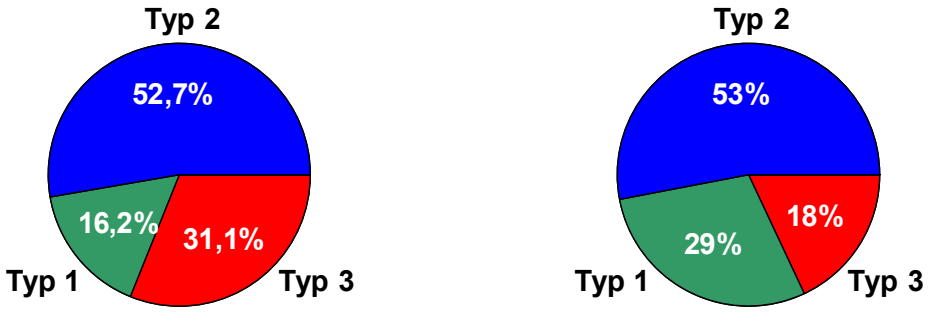
1.1. Wprowadzenie

W erze informacji i globalnej informatyzacji społeczeństwa, codzienność zastosowania i wykorzystywania maszyn cyfrowych przez człowieka przyćmiewa najbardziej optymistyczne prognozy analityków. Nie sposób wymienić dziedziny życia, gdzie urządzenia cyfrowe pośrednio bądź bezpośrednio nie wpływają na jej stan lub przebieg. Na szerokim polu zastosowań użytkowych, inżynierskich i przemysłowych, bezustannie występuje zapotrzebowanie na coraz bardziej wydajne systemy obliczeniowe. Opracowywane są wciąż nowe metody projektowania złożonych systemów cyfrowych w celu uproszczenia procesu poznawczego, korekcji błędów oraz zarządzania.

Zauważalny w ostatnich latach rozwój technologiczny na świecie wywołany zapotrzebowaniem rynku, dyskwalifikuje rozwiązania teleinformatyczne, które dostarczane są na rynek zbyt późno w stosunku do konkurencji lub z ograniczoną funkcjonalnością i wydajnością pracy. Hasło „time-to-market" dostarcza inżynierom wielu problemów dotyczących podejmowania kluczowych decyzji na etapie projektowym oraz w trakcie jego realizacji. Badania rynku [John05, Stan04] dostarczają istotnych informacji, które umożliwiają identyfikację trzech kluczowych aspektów procesu projektowego, mających znaczący wpływ na wynik

prac – sukces rynkowy: a) czas realizacji projektu, b) koszt finansowy systemu, c) funkcjonalność. Zaspokojenie wymienionych zagadnień realizacji projektu zapewnia udział produktu w walce o rynek. Badania CHAOS [Stan04] dotyczą realizacji projektów informatycznych prowadzonych na terenie Stanów Zjednoczonych (58%), Unii Europejskiej (27%) i pozostałych części świata (15%). Dostarczane wyniki stanowią jedno z wielu uznanych źródeł informacji wykorzystywanych przez zarządy firm IT oraz rządy państw w podejmowaniu decyzji w sprawach rozwoju i wdrożeń systemów teleinformatycznych. Rysunek 1.1 przedstawia wyniki badań realizacji projektów IT w roku 1994 i w trzecim kwartale roku 2004 (wyniki badań lat 2005 i 2006 nie były dostępne w czasie finalizowania części tekstowej rozprawy).

a) b)



Rysunek 1.1. Wyniki badań CHAOS, a) koniec roku 1994, b) trzeci kwartał roku 2004 [Stan04]

Legenda dla rysunku nr 1:

- Typ 1 – sukces, projekt został ukończony w ustalonym czasie, w zakresie przewidzianego budżetu, z zachowaniem pełnej funkcjonalności przewidzianej w specyfikacji,
- Typ 2 – wyzwanie, projekt zakończony, funkcjonujący, ale przekroczony budżet projektu, przekroczony czas realizacji, ograniczona funkcjonalność w stosunku do założeń specyfikacji projektu,
- Typ 3 – porzucony, projekt został zatrzymany i rozwiązany na pewnym etapie realizacji.

Problematyka realizacji projektu informatycznego dotyczy pełnego spektrum zarządzania i prowadzenia projektu, poczynając od wyboru sposobu projektowania, specyfikacji systemu, technologii, poprzez podział zadań projektowych, synchronizację prac, testów, a na weryfikacji i walidacji systemu kończąc. Wymienione zagadnienia projektowania stanowią potencjalne źródło błędów i problemów z jakimi na co dzień spotykają się projektanci systemów cyfrowych. Rozprawa w szczególności omawia i proponuje rozwiązania pozwalające na eliminację części zagrożeń występujących podczas realizacji projektu [John05, Stan04] poprzez homogeniczną reprezentację specyfikacji systemu oraz automatyzację procesu projektowego zintegrowanych sprzętowo-programowych mikrosystemów cyfrowych.

1.2. Poziomy specyfikacji systemu cyfrowego

Opracowywanie, wdrażanie i użytkowanie nowych technologii podyktowane jest zapotrzebowaniem społeczeństwa na dostęp do coraz szerszej gamy informacji cyfrowej dystrybuowanej m.in. przez środki masowego przekazu, usługodawców, miejsca pracy, a nawet członków rodziny. Narastające zapotrzebowanie konsumentów wymusza stosowanie nowoczesnych metod projektowania systemów teleinformatycznych, w tym systemów cyfrowych.

Proces projektowy dzisiejszych systemów cyfrowych znacząco różni się od metod minionych lat. Powyższe stwierdzenie dotyczy w szczególności systemów osadzonych, których metodologia projektowania oraz architektura poddana została znaczącym modyfikacjom.

Definicja 1.1. *Mikrosystem cyfrowy* jest systemem cyfrowym zrealizowanym w jednym programowalnym układzie scalonym.

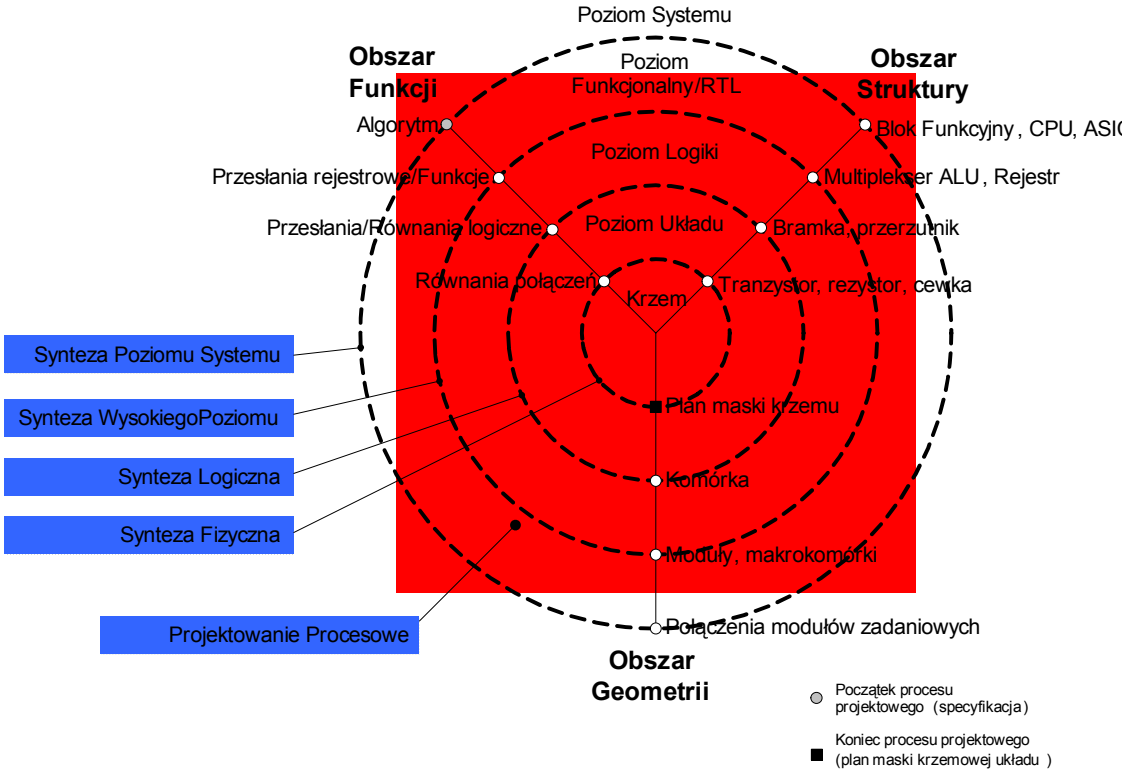
Definicja 1.2. *System osadzony* jest mikrosystemem cyfrowym stanowiącym samodzielną jednostkę zadaniową o określonym interfejsie wejścia/wyjścia, w nadrzędnym/większym systemie cyfrowym [Gajs96].

Proces projektowy cyfrowych systemów osadzonych, ze względu na złożoność dzisiejszych rozwiązań, składa się z kilku etapów – poziomów abstrakcji reprezentacji systemu – odpowiadających kolejnym krokom realizacji projektu.

Poziomy specyfikacji oraz etapy syntezy systemu cyfrowego prezentuje rysunek 1.2, który został opracowany na podstawie prac [JeMe99,ElKu98,HuJe98].

Pierwszym etapem w procesie projektowym cyfrowego systemu osadzonego jest specyfikacja algorytmiczna systemu – poziom specyfikacji systemu [JeMe99, ElKu98]. Opis zachowania całego systemu rozważany jest jako zbiór współoddziaływujących abstrakcyjnych procesów/zadań. Na tym etapie przeprowadzana jest synteza poziomu systemu[ElKu98], która dotyczy formułowania komponentów składowych architektury implementacyjnej projektowanego systemu. System może być zrealizowany za pomocą kooperujących procesorów, dedykowanych kontrolerów, układów programowalnych, procesorów DSP, i innych elementów peryferyjnych. Dobór zbioru fizycznych jednostek wykonawczych oraz przypisanie poszczególnym jednostkom zadań specyfikacji behawioralnej systemu, jest najbardziej krytyczną decyzją w procesie syntezy poziomu systemu. Kolejnym etapem w projektowaniu systemowym jest synteza wysokiego poziomu, której zadaniem jest translacja specyfikacji behawioralnej do opisu funkcjonalnego precyzującego zachowanie wyznaczonych jednostek fizycznych systemu (kod niezależny od dostawcy urządzenia). Opis zachowania systemu przedstawiony jest w wybranym języku specyfikacji funkcjonalnej, np. VHDL, Verilog, C. Kolejnym poziomem specyfikacji systemu w obszarze funkcjonalnym jest poziom przesłań międzyrejestrowych/funkcji tzw. RTL (ang. Register Transfer Level). Wśród elementów stosowanych do opisu układów na tym poziomie można wyróżnić: rejestry, liczniki, multipleksery i arytmetyczne jednostki logiczne ALU (ang. Arithmetic Logic Unit). Tego typu układy określane są jako podstawowe bloki funkcjonalne. Wszystkie elementy na tym poziomie są elementami fizycznymi o określonym rozmiarze, czasie propagacji oraz ustalonym interfejsie wejścia/wyjścia. Układ na tym poziomie opisu można przedstawić zarówno w formie tablicy prawdy, jak też w formie maszyny stanów. Następnym

poziomem specyfikacji systemu jest poziom logiki postrzegany w obszarze strukturalnym jako reprezentacja bramek (ang. Gate Level). Jest to podstawowy i najniższy poziom abstrakcji opisu stosowany w metodologii projektowania systemów cyfrowych.



Rysunek 1.2. Poziomy specyfikacji systemu cyfrowego [JeHu98]

Podstawowymi obiektami struktury układu na tym poziomie opisu są bramki oraz przerzutniki. Poprzez realizację połączeń na określonym zbiorze komponentów, projektant może budować układy kombinacyjne lub sekwencyjne. Do opisu układu na tym poziomie abstrakcji używa się równań logicznych. Poziom układu dostępny jest tylko i wyłącznie w ścieżce projektowej, której celem jest realizacja systemu jako układu typu ASIC (ang. Application Specific Integrated Circuit). Reprezentacją układu na tym poziomie jest struktura zarówno pasywnych (rezystory, kondensatory, cewki) jak też aktywnych elementów elektronicznych (tranzystory). Funkcjonalnie układ na tym poziomie można opisać przy użyciu równań różniczkowych. Podstawowym poziomem systemu cyfrowego jest poziom krzemu (ang. Silicon Level). Jest to poziom fizycznego odwzorowania funkcjonalności systemu w strukturach układu elektronicznego. Na tym poziomie, układ jest przedstawiony w postaci geometrycznych obszarów krzemu i metalu, odpowiednio połączonych między sobą.

Z punktu widzenia tematyki rozprawy, interesującym i jedynym rozważanym poziomem specyfikacji systemu jest poziom funkcjonalny/RTL, którego dotyczą zagadnienia syntezy wysokiego poziomu. Na rysunku 1.2 kolorem czerwonym oznaczono zakres przeprowadzonych badań naukowych. Zrealizowane prace dotyczą projektowania procesorowego, którego celem jest skrócenie czasu przetwarzania danych i procesu sterowania przez główną jednostkę CPU

(ang. Central Processing Unit) w mikrosystemie cyfrowym. Projektowanie procesorowe jest elementem metodologii syntezy wysokiego poziomu. Częściowo obejmuje swoimi zadaniami obszar opisu systemu na poziomie funkcjonalnym (reprezentujący opis zachowania komponentów systemu) oraz poziomie logicznym. W procesie projektowym realizowanym na poziomie procesorowym, projektant operuje pojedynczymi operazjami/instrukcjami w celu precyzyjnej alokacji mikrozadania systemu do części programowej lub sprzętowej. W zakresie projektowania procesorowego znajdują się: estymacja czasu realizacji mikroinstrukcji, koszt realizacji, czas komunikacji program-sprzęt oraz inne analizy formalne i funkcjonalne, przedstawione w rozdziale czwartym rozprawy.

W obszarze projektowania procesorowego przeprowadzono szereg badań naukowych, które sklasyfikowano jako rozwiązania [BaLa02, MiOl98]:

- RISP (ang. Reconfigurable Instruction Set Processor); procesor hybrydowy, realizujący instrukcje programowe i sprzętowe z wykorzystaniem dodatkowego bloku sprzętowego RFU (ang. Reconfigurable Functional Unit),
- ASIP (ang. Application Specific Instruction set Processor); specjalizowana sprzętowa architektura procesora budowana na podstawie docelowej specyfikacji programu.

Szczegółowa charakterystyka wymienionych rozwiązań akademickich i komercyjnych przedstawiona została w rozdziale drugim.

1.3. Realizacje fizyczne systemów cyfrowych

Powszechnie stosowanymi architekturami systemów cyfrowych są rozwiązania typu SoB (ang. System on a Board), gdzie poszczególne składowe systemu (pamięć, procesor, DSP, urządzenia we/wy, itp.) montowane są na wcześniej przygotowanej płycie PCB (ang. Printed-Circuit Board). Rezultatem technologicznej ewolucji układów scalonych jest architektura oraz metodologia projektowania układów typu SoC (ang. System on a Chip) [JeHu98]. W tym rozwiązaniu wszystkie elementy cyfrowe (i w pewnej części analogowe) projektowanego systemu zostają zintegrowane w jednym układzie scalonym typu ASIC. Końcowa realizacja fizyczna (produkcja maski) poprzedzona jest wcześniejszą weryfikacją funkcjonalną systemu zazwyczaj w reprogramowalnym układzie FPGA (ang. Field Programmable Gate Array) [Xili06,Alte06]. Główną zaletą układów SoC jest przede wszystkim znaczna redukcja poboru mocy systemu w stosunku do rozwiązań SoB oraz:

- małe gabaryty fizyczne; dzięki czemu możliwe jest projektowanie urządzeń przenośnych takich jak telefony komórkowe, PDA (ang. Personal Digital Assistant), i inne,
- zwiększona wydajność pracy poprzez zintegrowanie składowych systemu,
- nowa zintegrowana metodologia projektowania.

Do wad należy zaliczyć zamkniętą realizację końcową (krzemowa matryca – ang. Chip) układu ASIC. Zaprojektowany i wyprodukowany układ cyfrowy dedykowany jest do jednego konkretnego zadania. Ponadto, koszt produkcji jednego układu scalonego typu ASIC jest nie opłacalny. Wymagane są zamówienia rzędu tysięcy sztuk jednego układu cyfrowego [Alte06]. Dodatkowo czas

produkcji układu ASIC jest stosunkowo długi (kilka – kilkanaście miesięcy). Realizacja projektu składającego się z kilku lub kilkunastu układów cyfrowych jest dużo tańsza przy wykorzystaniu układów FPGA w porównaniu z rozwiązaniami ASIC. Z drugiej strony, koszt wyprodukowania pojedynczego układu ASIC (zamówienie masowe) w porównaniu z zakupem jednego układu FPGA jest znacząco mniejszy [Alte06]. Bardziej uniwersalnym rozwiązaniem jest architektura SOPC (ang. System On a Programmable Chip), którą wyróżnia otwarta realizacja końcowa kompletnego mikrosystemu cyfrowego, poprzez zastosowanie struktur reprogramowalnych. Ze względu na coraz większą dostępność układów FPGA, czas realizacji projektu (brak cyklu produkcyjnego nawiązującego do ASIC) oraz możliwość rekonfiguracji struktury wewnętrznej, układy FPGA masowo wykorzystywane są do realizacji cyfrowych systemów osadzonych. Cechą charakterystyczną systemów realizowanych w technologii SOPC jest możliwość integracji w jednym układzie scalonym logiki reprogramowalnej FPGA z rdzeniem mikroprocesora typu *hardcore* lub *softcore* [Xili06,Alte06].

Realizacje systemów cyfrowych z wykorzystaniem układów (architektur) SoC lub SOPC określa się nazwą mikrosystemu cyfrowego – system w jednym układzie scalonym. W projektowaniu mikrosystemów cyfrowych stosuje się metodologię klasyczną projektowania systemów cyfrowych lub metodologię projektowania zintegrowanego sprzętowo-programowych systemów cyfrowych. Szczegółowy opis wymienionych metodologii przedstawiono w rozdziale drugim.

Realizacja techniczna prac przeprowadzonych w zakresie rozprawy dotyczy implementacji opracowanych rozwiązań z wykorzystaniem architektury typu SOPC.

1.4. Motywacje podjęcia tematu prac

Powszechnie stosowany (czyt. klasyczny) proces projektowania sprzętowo-programowych mikrosystemów cyfrowych poprzedzony jest podziałem specyfikacji systemu na zbiór funkcji składowych, które w realizacji końcowej projektu wykonywane są przez mikroprocesory oraz bloki sprzętowe typu IP CORE (ang. Intellectual Property CORE). Przydział zadań/funkcji składowych systemu do realizacji programowej lub sprzętowej realizowany jest zazwyczaj w sposób manualny przez projektanta w procesie projektowym. W rezultacie, około 71% (rozdział 1.1) projektów nie spełnia założeń implementacyjnych lub odrzucanych jest w czasie realizacji [Stan04,Xili06,Alte06]. Ponadto, główną jednostką sterowania i (w części) przetwarzania wykorzystywaną w rozwiązaniach SoC i SOPC, jest mikroprocesor, który charakteryzuje się sekwencyjnym przetwarzaniem instrukcji programu. W rezultacie, przy dużych obciążeniach systemu (systemy reaktywne, systemy bezprzewodowe, telekomunikacja), sumaryczna wydajność (częstotliwość pracy) całego systemu może spadać do poziomu krytycznego, rysunek 1.2. Za przyczyny można uznac między innymi: błędnie zdefiniowaną specyfikację systemu/mikrosystemu, błędnie zaimplementowane funkcje lub procedury programu lub sprzętu, konflikty/różniec

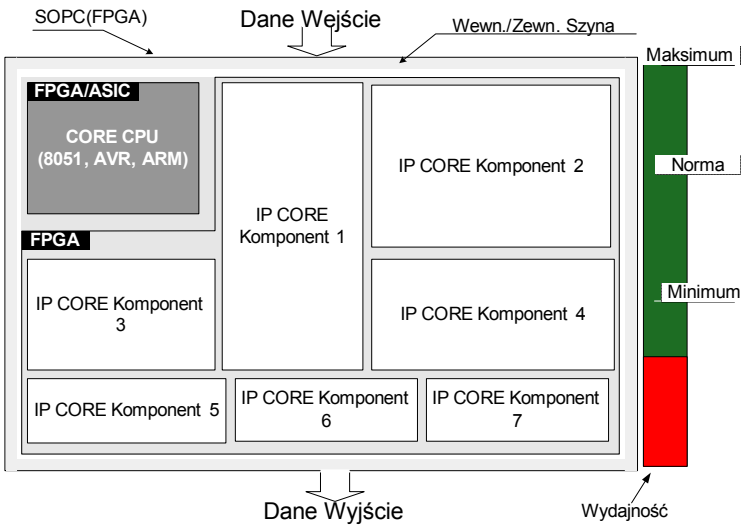
na poziomie komunikacji sprzęt-program, zbyt optymistyczne założenia projektowe, inne.

Poszukiwane są, zatem rozwiązania wspomagające projektanta na etapie specyfikacji i procesu projektowego mikrosystemu cyfrowego, jednocześnie ograniczające nakład pracy człowieka. Pomocą mogą służyć znane opracowania (akademickie i komercyjne) wspierające proces projektowy systemu i mikrosystemu cyfrowego [ErHe98, BaCh97,xili06,alte06], rozwiązując tym samym problem manualnego i często nie efektywnego ze względu na koszty czasu projektowania systemowego. Jednak, w domenie niskopoziomowej optymalizacji pracy mikroprocesora systemu SOPC, zauważalne są braki gotowych rozwiązań, a istniejące propozycje obarczone są ograniczeniami funkcjonalnymi lub metodologicznymi, które autor szeroko omawia w rozdziale trzecim rozprawy.

W rozprawie rozważaniom poddane zostały zagadnienia dotyczące akceleracji pracy mikroprocesora przetwarzającego program w mikrosystemach cyfrowych. Podjęto próbę rozwiązania problemu niskiej wydajności pracy głównej jednostki sterowania i przetwarzania CPU poprzez zdefiniowanie uniwersalnej, pod względem funkcjonalnym i implementacyjnym, architektury akceleratora sprzętowego oraz opracowanie metody projektowania nowej architektury.

Praktyka potwierdzona badaniami [Hans04], pokazuje, że w większości rozpatrywanych projektów realizacji systemów cyfrowych typu SoC lub SOPC, inżynier wspierany szeregiem narzędzi CAD jest w stanie zagospodarować, co najwyżej, 70%-90% logiki reprogramowalnej układu FPGA. Można przyjąć, że zawsze pozostaje pewien obszar niewykorzystanych zasobów sprzętowych logiki reprogramowalnej.

Nowatorskim rozwiązaniem w pracy jest opracowana sprzętowo architektura akceleratora cyfrowego oraz metoda jego projektowania zorientowana na wykorzystanie wolnej logiki programowalnej układu SOPC, jako modułu wspomagającego pracę mikroprocesora. Celem jest zwiększenie wydajności pracy głównej jednostki przetwarzania i sterowania mikrosystemu cyfrowego. W ten sposób wyodrębniona zostaje Sprzętowo-Programowa Mikrostruktura Cyfrowa SPMC (rozdział trzeci rozprawy), składająca się z jednostki CPU i części sprzętowej w postaci wolnej logiki reprogramowalnej układu SOPC. Głównym zadaniem mikrostruktury, jako głównej jednostki sterowania i przetwarzania mikrosystemu cyfrowego, jest zapewnienie nominalnej wydajności pracy systemu ze względu na czas reakcji i częstotliwość.



Rysunek 1.3 Wydajność CPU przy dużym obciążeniu systemu

Konieczność akceleracji zadań realizowanych programowo w systemach osadzonych dostrzegany jest również przez przemysł [Celo06,Cowa06,Xili06,Alte06], którego aktywność w domenie wspomagania projektowania zintegrowanego można traktować jako odpowiedź na potrzebę rozwiązania sformułowanego problemu. Rozwiązania proponowane przez rynek komercyjny skupiają się jednak tylko na jednym z dwóch poniższych zagadnień:

- Projektowanie systemowe [Celo06,Cowa06,Syst06]. Specyfikacja systemu jest wspólna dla sprzętu i oprogramowania, a złożone algorytmy mogą być asygnowane w zależności od potrzeb, do części programowej lub sprzętowej. Możliwa jest współsymulacja części sprzętowej i programowej. Proces syntezy wysokiego poziomu przeprowadzany jest w sposób automatyczny bezpośrednio z zapisu algorytmicznego systemu (poziom systemu) do realizacji programowej lub sprzętowej (HDL-RTL). Próby implementacyjne [ŚnRu03] dyskwalifikują jednak proponowane rozwiązania ze względu na nieefektywne wyniki procesu syntezy części sprzętowej.
- Wspomaganie procesu integracji komponentu sprzętowego projektanta z interfejsem dedykowanego procesora [xili06,alte06]. Brak metodologii projektowania oraz uniwersalności rozwiązań implementacyjnych akceleratora [alte06] i koprocesora [xili06] sprzętowego.

Propozycje akademickie dotyczą głównie metod optymalizacji kodu programu poprzez techniki profilowania kodu [Mart00]. Ponadto, znane są metody akceleracji przetwarzania programu poprzez przeniesienie wybranych zadań do części sprzętowej procesora RISP. Jednak, wskazane rozwiązania [BaLa02] nie są uniwersalne i mogą być stosowane tylko do wybranych architektur sprzętowych i implementacji programowych (wybrane typy mikroprocesorów).

1.5. Cele i teza pracy

Genezą podjęcia tematu są przeprowadzone badania naukowe [Stas03a, Stas03b,Stas02a,Stas02b,BaLa02] wykazujące konieczność przyspieszenie

pracy głównej jednostki CPU oraz publikacje naukowe [Hart[a1], Goud05, Scha00], które definiują problem jako krytyczny zarówno dla obecnych, jak i przyszłych zintegrowanych mikrosystemów cyfrowych. Konieczność przeprowadzenia rozważań, a w konsekwencji badań i prac naukowo-technicznych dotyczących zwiększenia wydajności przetwarzania i sterowania w zintegrowanych mikrosystemach cyfrowych, zainicjował również brak narzędzi klasy CAD (ang. Computer Aided Design) umożliwiających automatyczne przeprowadzenie procesu dekompozycji funkcjonalnej specyfikacji mikrosystemu cyfrowego oraz brak metody projektowania uniwersalnego akceleratora sprzętowego.

Rozważania prowadzone w pracy służą wykazaniu prawdziwości następującej tezy:

> **Wprowadzenie sprzętowo-programowej mikrostruktury cyfrowej realizowanej w wyniku podziału wejściowej specyfikacji funkcjonalnej na część sprzętową i programową, pozwala zwiększyć wydajność pracy mikrosystemu cyfrowego.**

Tezę pracy można uszczegółowić następująco:

> Możliwe jest dokonanie automatycznego oraz efektywnego pod względem szybkości pracy i kosztów realizacji technicznej, podziału specyfikacji funkcjonalnej mikrostruktury cyfrowej na część programową w języku ANSI C, wykonywaną przez mikroprocesor oraz część sprzętową, realizowaną przez specjalizowane układy programowalne typu FPGA, z zachowaniem pełnej funkcjonalności projektowanego mikrosystemu.

Wyróżniono trzy główne cele rozprawy doktorskiej:

1. Opracowanie metody projektowania SPMC pozwalającej na realizację mikrostruktury SPMC z zachowaniem pełnej funkcjonalności specyfikacji wejściowej.
2. Skrócenie czasu wykonywania instrukcji przetwarzania i sterowania przez główną jednostkę CPU mikrosystemu cyfrowego poprzez wykorzystanie wolnych zasobów logiki reprogramowalnej układu FPGA.
3. Opracowanie i udostępnienie oprogramowania CAD wspomagającego proces projektowy mikrostruktury SPMC.

Dowód tezy zostanie przeprowadzony na podstawie rozważań teoretycznych oraz zestawienia wyników przeprowadzonych badań eksperymentalnych dotyczących zależności zwiększenia wydajność pracy nowej mikrostruktury cyfrowej (i bezpośrednio mikrosystemu cyfrowego), względem określonej liczby wolnych bloków konfiguracyjnych logiki reprogramowalnej układu FPGA.

Za specyfikację formalną mikrosystemu cyfrowego przyjęto interpretowane, hierarchiczne, uwarunkowane czasem sieci Petriego [Petr62, Mura89]. Wybór podyktowany został właściwościami sieci Petriego, których charakter w pełni determinuje założenia modelu formalnego[ChGi93, StSk06] w projektowaniu zintegrowanym.

W pracy założono, że specyfikacja mikrostruktury cyfrowej podawana jest w postaci tekstowej na wejście opracowanego systemu CAD. Natomiast wynikiem procesu dekompozycji funkcjonalnej [JeMe98, Ster97, ErHe98] są dwa podzbiory sieci Petriego:

- część sprzętowa, zawierająca konfigurację wewnętrznego i zewnętrznego interfejsu komunikacyjnego,
- część programowa, zawierająca konfigurację wewnętrznego i zewnętrznego interfejsu komunikacyjnego.

Ponadto, opracowany algorytm dekompozycji funkcjonalnej nie może być utożsamiany z klasycznymi rozwiązaniami, w sensie rozumienia terminu projektowania zintegrowanego, takimi jak Cosyma [ErHe98], czy Vulcan [RGCM94]. Problem oraz jego rozwiązanie referowane w rozprawie dotyczą przyspieszenia przetwarzania danych i sterowania w reprogramowalnych zintegrowanych mikrosystemach cyfrowych klasy SOPC, zorientowanych na realizację sterowania przez wbudowany mikroprocesor. Zagadnienia poruszane w klasycznych metodologiach projektowania zintegrowanego dotyczą ogólniejszych problemów podziału specyfikacji formalnej na poziomie systemu [ElKu98]. W rozprawie użyto technik i koncepcji projektowania zintegrowanego tylko w celu uzyskania większej wydajności nowej a zaproponowanej w pracy mikrostruktury cyfrowej. Badania i testy przeprowadzone w pracy dotyczą opracowanej metody projektowania i zaproponowanej architektury sprzętowego akceleratora. Obszar teoretyczny i tematyczny rozprawy nie obejmuje więc metodologii projektowania na poziomie systemu, stąd praca nie zawiera rozważań dotyczących zastosowania metody SPMC do projektowania systemowego.

Wyznaczono następujące szczegółowe cele pracy:

- sformalizowanie modelu formalnego mikrostruktury cyfrowej,
- opracowanie języka zapisu tekstowego funkcjonalności mikrostruktury cyfrowej,
- opracowanie algorytmu dekompozycji/podziału funkcjonalnej mikrostruktury cyfrowej, na część realizowaną w strukturach FPGA i oprogramowanie procesora,
- wykonanie oprogramowania przeprowadzającego syntezę hierarchicznych sieci Petriego do języka C dla wybranego mikroprocesora,
- wykonanie oprogramowania przeprowadzającego syntezę hierarchicznych sieci Petriego do języka RTL-VHDL,
- zaprojektowanie i realizacja symulatora hierarchicznych sieci Petriego dla potrzeb projektowania zintegrowanego, uwzględniającego parametry czasowe procesów,
- realizacja programu do wizualizacji i graficznej edycji hierarchicznych sieci Petriego,
- zaprojektowanie i realizacja wirtualnego systemu kosymulacji sprzętowo-programowej, bazującej na symulatorze języków HDL,
- system kosymulacji sprzętowo-programowej mikrosystemu cyfrowego w układzie FPGA.

Dodatkowo uwzględnia się następujące założenia praktyczne pracy:

- uniwersalność aplikacyjna opracowanej architektury sprzętowej,
- zorientowanie na minimalizację zasobów sprzętowych akceleratora sprzętowego,
- zastosowanie formalnego modelu specyfikacji systemu,
- automatyzacja procesu projektowego.

Prace przeprowadzone w zakresie rozprawy doktorskiej, zostały w części zrealizowane w ramach współpracy naukowej pomiędzy autorem a zespołem badawczym realizującym grant KBN nr 4 T11C 006 24.

1.6. Struktura pracy

Praca została podzielona na sześć części tematycznych. Rozdział pierwszy wprowadza to tematyki rozprawy. Przedstawia genezę podjętego problemu, formułuje tezę, cele oraz sposoby dowodzenia twierdzeń rozprawy.

Rozdział drugi omawia zagadnienia teoretyczne poruszane w pracy. W pierwszej części rozdziału przedstawiono zagadnienia ogólne. Rozważaniom poddano metody projektowania mikrosystemów cyfrowych, specyfikacje systemów cyfrowych oraz techniki opisu zachowania systemu. W końcowej części skupiono się na szczegółowych zagadnieniach implementacyjnych sprzętowo-programowych architektur hybrydowych, w szczególności RISP.

W rozdziale trzecim przedstawiono charakterystykę pracy mikroprocesora osadzonego w mikrosystemach cyfrowych, wskazując na problemy opisane w rozdziale pierwszym. Sformułowano wymagania stawiane uniwersalnej, pod względem konstrukcyjnym i funkcjonalnym, architekturze akceleratora sprzętowego. Przedstawiono właściwości projektowania zintegrowanych mikrosystemów cyfrowych ze względu na wykorzystanie reprogramowalnej logiki układu FPGA. Zdefiniowano nową architekturę sprzętowo-programowej mikrostruktury cyfrowej, której zadaniem jest skrócenie czasu przetwarzania i sterowania programu wykonywanego przez główną jednostkę kontroli mikrosystemu cyfrowego. Szczegółowo omówiono budowę, interfejs i protokół komunikacyjny nowej mikrostruktury cyfrowej oraz zdefiniowano koszty komunikacji części programowej i sprzętowej.

Rozdział czwarty poświęcony został opracowanej metodzie projektowania sprzętowo-programowej mikrostruktury cyfrowej SPMC. Przedstawiono metodę SPMC oraz proponowane/możliwe punkty jej integracji z klasyczną i zintegrowaną metodologią projektowania systemów cyfrowych Zaprezentowano definicję nowego modelu formalnego mikrostruktury cyfrowej wskazując na własności, jakie powinien spełniać dobry model oraz sformułowano wymagania stawiane przed modelem formalnym specyfikującym zachowanie i charakterystykę pracy programowo-sprzętowego systemu cyfrowego. Przedstawiono nowy format zapisu elektronicznego sprzętowo-programowej mikrostruktury cyfrowej specyfikowanej hierarchicznymi, czasowymi sieciami Petriego. Ponadto, rozdział prezentuje algorytm dekompozycji funkcjonalnej SPMC odpowiedzialny za przydział zadań specyfikacji wejściowej do część programową i sprzętową. Przedstawiono opracowane metody optymalizacji wybranych algorytmów syntezy sprzętowej i programowej sieci Petriego.

Rozdział piąty ilustruje wybrane przykłady implementacji sterowania i przetwarzania, które w mikrosystemach cyfrowych wykonywane są przez wbudowany mikroprocesor. Porównano wyniki czasu realizacji przydzielonych zadań przez standardowy mikroprocesor oraz sprzętowo-programową mikrostrukturę cyfrową. Testom wydajności i jakości poddano zarówno algorytmy syntezy programowej, sprzętowej jak i algorytm dekompozycji funkcjonalnej SPMC. Zaprezentowano przykład realizacji wybranego zadania z wykorzystaniem metody SPMC obrazując wpływ wolnej logiki reprogramowalnej systemu SOPC (w układzie FPGA) na wzrost wydajności pracy mikrosystemu cyfrowego.

Przedstawiono wyniki procesu syntezy programowej i sprzętowej oraz zbiorcze rezultaty metody SPMC. Sformułowano wnioski podsumowujące przeprowadzoną rozprawą doktorską.

W ostatnim, szóstym rozdziale rozprawy, podsumowano wyniki opracowanej metody projektowania sprzętowo-programowej mikrostruktury cyfrowej oraz wpływ nowego, uniwersalnego pod względem realizacji akceleratora sprzętowego SPMC na wydajność pracy mikrostruktury cyfrowej.

W końcowej części pracy załączono dodatki, do których odwołują się wybrane części pracy.

ROZDZIAŁ DRUGI

2. Zagadnienia teoretyczne przedmiotu

W rozdziale przedstawiono zagadnienia teoretyczne i terminy, którymi autor posługuje się w dalszej części pracy. W części pierwszej rozdziału przedstawiono zagadnienia ogólne dotyczące znanych metodologii projektowania mikrosystemów cyfrowych. Rozważaniom poddano specyfikacje systemów cyfrowych oraz techniki opisu zachowania systemu. W końcowej części skupiono się na szczegółowych zagadnieniach implementacyjnych sprzętowo- programowych architektur hybrydowych, w szczególności RISP. Przedstawiono aktualne stan prac naukowych dziedziny rozprawy oraz bieżące rozwiązania komercyjne.

2.1. Metodologie projektowania systemów cyfrowych

Dziedziną informatyki, która szczególnie integruje się z codziennością życia człowieka XXI wieku, są cyfrowe systemy osadzone. Za przykład mogą posłużyć urządzenia masowego odbioru, takie jak: telefon komórkowy, PDA, urządzenia nawigacji GPS (ang. Global Positioning System) [ieee06], i inne. Koncepcje dotyczące dalszych ingerencji urządzeń cyfrowych w życie człowieka, w znaczeniu pozytywnym, skierowane są w kierunku mobilności aplikacji, miniaturyzacji, oszczędności energii, kreując w ten sposób model urządzenia prostego w użytkowaniu, bezpiecznego, a przede wszystkim produktu pożądanego. Ekspansja rozwiązań kompaktowych pod względem funkcjonalnym i zastosowania jest nieunikniona
[Inte06,Micr06].

W latach 80-90 na rynek wprowadzono reprogramowalne układy cyfrowe PLD, CPLD, FPGA[Xili06,Alte06]. Poprzez programowanie poszczególnych bloków logicznych, uzyskuje się określoną funkcjonalność układu cyfrowego. Konstrukcja dzisiejszych układów programowalnych umożliwia budowę kompletnych systemów cyfrowych (lub cyfrowo-analogowych) w postaci jednego układu scalonego. Obecnie firma Xilinx oferuje układ programowalny VirtexII Pro, który w swojej strukturze posiada wbudowanych od jednego do czterech mikroprocesorów IBM

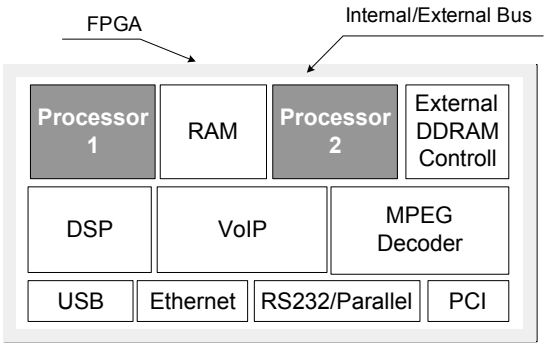
PowerPC 405, o częstotliwości zegara systemowego 400MHz, zintegrowanych z logiką programowalną (55000 komórek programowalnych) [xili06]. Firma Atmel dostarcza na rynek układ FPSLIC [Atme06], który zawiera w swojej budowie mikrokontroler typu RISC AVR pracujący z częstotliwością 25MHz zintegrowany z logiką programowalną (2300 komórek programowalnych). Również firma Altera rozwija rodziną Excalibur i Stratix [Alte06] z mikrokontrolerami typu RISC rodziny ARM. Są to układy, które wykorzystuje się do budowy kompleksowych, pod względem funkcjonalności systemów cyfrowych SOPC lub do prototypowania w metodologii projektowania systemów SoC. Tego typu układy znajdują zastosowanie w wielu dziedzinach przemysłu, miedzy innymi: telekomunikacja, przetwarzanie dźwięku i obrazu, systemy sterowania, specjalizowane systemy obliczeniowe. Pomimo technologii umożliwiającej produkowanie złożonych systemów cyfrowych w postaci jednego układu scalonego (45nm[Inte06]), metodologia projektowania zintegrowanych systemów cyfrowych w rozwiązaniach komercyjnych praktycznie nie uległa zmianie. Powszechnie stosowana jest „klasyczna" metodologia projektowania systemów cyfrowych. Rezultaty tak realizowanych projektów obrazują wyniki badań przedstawione w rozdziale pierwszym, rysunek 1.1. Modyfikacji uległy jedynie narzędzie wspomagające proces projektowy w obszarze dotyczącym kodowania opisu zachowania systemu w wybranym języku, weryfikacji funkcjonalnej, integralności programowania.

Nowatorską metodologią projektowania systemów cyfrowych, promowaną przez środowisko naukowe i akademickie, a wyłonioną w efekcie prowadzonych prac w sferze syntezy systemowej, jest metodologia zintegrowanego projektowania sprzętu i oprogramowania (ang. Hardware Software Co-design) [ElKu98, GaVa95,ÖsBe97].

2.1.1. Metodologia klasyczna projektowania układów i systemów cyfrowych

Proces projektowy złożonych systemów cyfrowych jako realizacji SoC lub SOPC, w głównej mierze polega na wykorzystaniu w procesie projektowym sprzętowych elementów bibliotecznych typu IP CORE [Xili06].

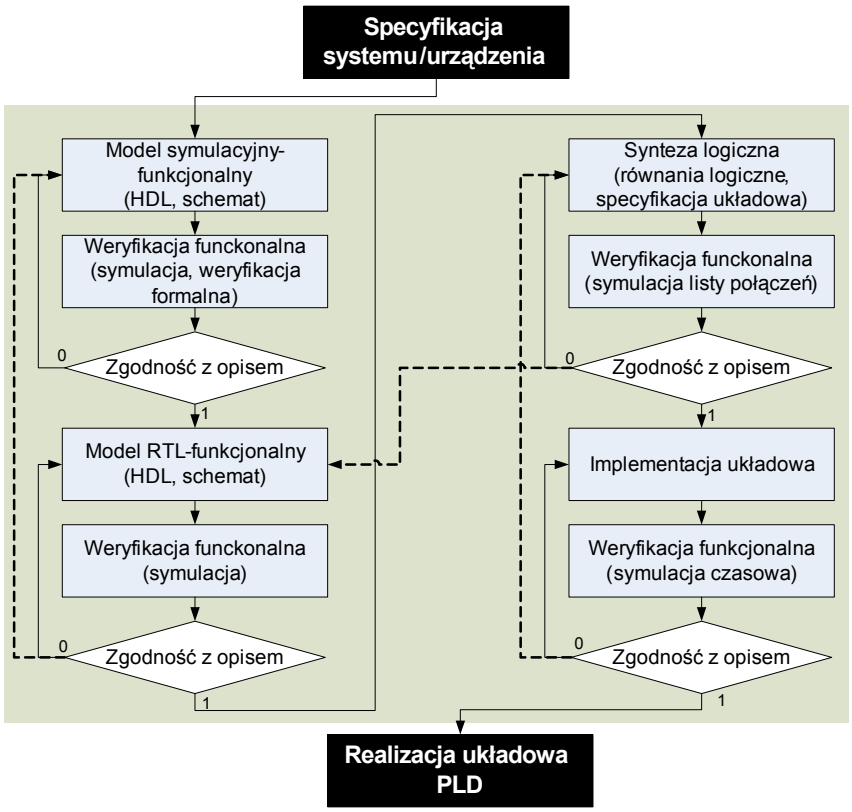
Definicja 2.1 *IP CORE (ang. Intellectual Property Core) [JeMe98] jest jednostką biblioteczną układu cyfrowego opisanego w języku HDL, w pełni odzwierciedlającą wzorcową specyfikację zachowania, która wykorzystywana jest w produkcji mikrosystemów cyfrowych opartych na technologii FPGA lub ASIC.*



Rysunek 2.1 Układ programowalny ze zintegrowanym mikroprocesorem

Rysunek 2.1 przedstawia poglądową implementację systemu cyfrowego. Wszystkie przedstawione elementy blokowe są komponentami IP CORE opisanymi zazwyczaj za pomocą jednego z języków opisu sprzętu VHDL lub Verilog.

Proces projektowy systemu cyfrowego [GaVa95, Xili06, Alte06, Atme06, JaMe98,AdBa06] przedstawia rysunek 2.2. Na etapie specyfikacji systemu definiowane są zadania systemu oraz wstępnie określana jest architektura systemu, uwzględniająca konstrukcje układów FPGA, dostępne bloki DSP oraz wbudowane komponenty funkcjonalne (układy mnożące, programowalne, pamięci, itp.).



Rysunek 2.2 Metodologia projektowania klasycznego systemów i układów PLD

Następnie rozpoczynany jest proces realizacji systemu (kodowanie funkcjonalności) poprzez wykorzystanie gotowych składowych IP CORE lub opracowanie i implementacja pozostałych zadań sprzętowych. Na etapie opisu behawioralnego, system poddawany jest weryfikacji funkcjonalnej w celu wykrycia i eliminacji błędnego zachowania komponentów systemu. Z kolei, opis abstrakcyjny każdego komponentu systemu musi zostać zamieniony przez projektanta na opis poziomu przesłań między rejestrowych (ang. Register Transfer Level). W dalszym ciągu operuje się na opisie funkcjonalnym systemu, jednak o większej szczegółowości implementacyjnej. W dalszej kolejności, podczas syntezy logicznej [Bara94] opis RTL zostaje zamieniony na równania logiczne odzwierciedlające zachowanie systemu. W procesie syntezy logicznej, na podstawie wyznaczonych funkcji logicznych, generowana jest tzw. lista połączeń (ang. netlist) podstawowych elementów całego systemu cyfrowego (ang. primitives) w wybranym formacie, np. NGD [Xili06] lub EDN(**E**lectronic **D**esign Interchange Format **N**etlist File) [ieee06]. Konieczne jest przeprowadzenie weryfikacji funkcjonalnej modelu matematycznego, otrzymanego po procesie syntezy, przez

porównanie z modelem behawioralnym. Wczesne wykrycie błędów funkcjonalnych wynikających głównie ze źle wykonanego modelu systemu na poziomie RTL, skróci czas realizacji projektu. Kolejnym krokiem w homogenicznej metodologii projektowania systemów cyfrowych jest proces implementacji (składający się z kilku etapów), w całości obsługiwany przez producenta układów FPGA. Rezultatem jest binarny strumień programujący dla układu FPGA oraz pliki symulacyjne niezbędne do przeprowadzenia ostatniego kroku – symulacji czasowej (ang. timing simulation). Ponowne porównanie wyników symulacji wzbogaconej o parametry czasowe z pierwotnym modelem funkcjonalnym uwiarygodnia weryfikację behawioralną wykonanego systemu cyfrowego. Ponadto możliwa jest analiza i korekta modelu w celu poprawy parametrów czasowych jego pracy lub zajmowanej przestrzeni programowalnej FPGA. Na tym etapie kończy się proces prototypowania systemu cyfrowego. Dalsze kroki uzależnione są od docelowej implementacji realizowanego systemu do układów SoC lub SOPC.

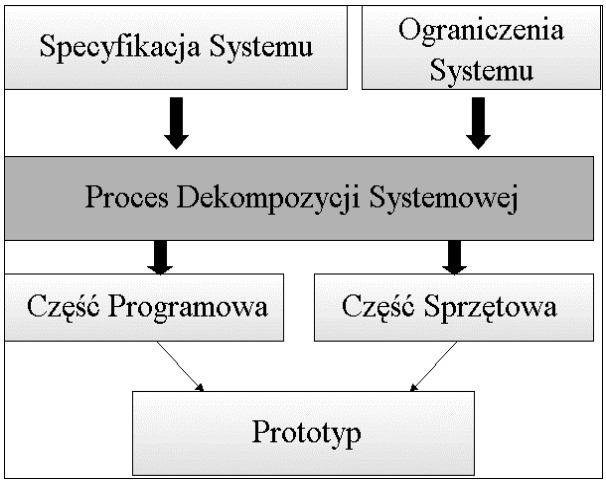
2.1.2. Zintegrowane projektowanie sprzętu i oprogramowania

Popularność systemów sprzętowo-programowych zawierających mikroprocesory, mikrokontrolery i mikrokomputery jednoukładowe jest wynikiem ich prostoty oraz uniwersalności zastosowań. Często zachodzi jednak sytuacja, w której szybkość działania zaprojektowanego prototypu urządzenia nie spełnia kryteriów użytkownika, na przykład z powodu zbyt długiego czasu potrzebnego na wykonanie programu przez mikrokontroler systemu [Stas03a]. Zachodzi wówczas konieczność zastosowania rozwiązania sprzętowego z wykorzystaniem układów cyfrowych, takich jak ASIC. Wówczas cała funkcjonalność systemu umieszczona jest w części sprzętowej, co wpływa na poprawę szybkości działania urządzenia. Minusem tego rozwiązania jest wzrost kosztów związanych z produkcją dedykowanych układów cyfrowych. Rozwiązaniem pośrednim jest podział specyfikacji określającej funkcjonalność systemu na część wykonywaną przez mikrokontroler systemu (część programowa) oraz na część odwzorowywaną w strukturze układu programowalnego (część sprzętowa). Znalezienie właściwego podziału między poszczególne części realizowane jest na podstawie kryteriów użytkownika, takich jak: czas działania systemu oraz dopuszczalny koszt (np. wyrażony w ilości bramek logicznych) części sprzętowej.

Tego typu podział, dokonywany na poziomie specyfikacji systemowej, określany jest mianem dekompozycji systemowej (ang. System Partitioning) [JeMe98, Ster97, ErHe98]. Jest to jeden z podstawowych etapów nowoczesnego podejścia do projektowania systemów cyfrowych, określanego mianem zintegrowane projektowanie systemów sprzętowo-programowych. Podczas takiego projektowania system poddawany jest podziałowi na program i sprzęt dopiero w końcowej fazie procesu projektowego, a nie jak w tradycyjnym podejściu – w fazie początkowej.

Zaletą tego podejścia jest możliwość podejmowania kluczowych decyzji, dotyczących podziału systemu na sprzęt i program w końcowym etapie projektowym, tj. w czasie, gdy projektant dysponuje znacznie dokładniejszymi informacjami na temat parametrów, możliwości i ograniczeń systemu.

Podstawowym etapem projektowania zintegrowanego jest proces dekompozycji systemowej. Dekompozycja dotyczy podziału systemu pomiędzy sprzęt i oprogramowanie, rysunek 2.3. Konieczność wykonania dekompozycji determinuje system z co najmniej dwoma komponentami funkcjonalnymi. Na podstawie specyfikacji wejściowej oraz zadanych wymagań (czasowych, finansowych, użytkowych i innych) projekt dzielony jest na moduły realizowane odpowiednio przez procesor (ogólnego przeznaczenia, DSP, inne) - część programowa (ang. Software) oraz układy cyfrowe współpracujące z procesorem (w tym układy programowalne ASIC, reprogramowalne FPGA) - część sprzętowa (ang, Hardware).



Rysunek 2.3 Metodologia projektowania zintegrowanego

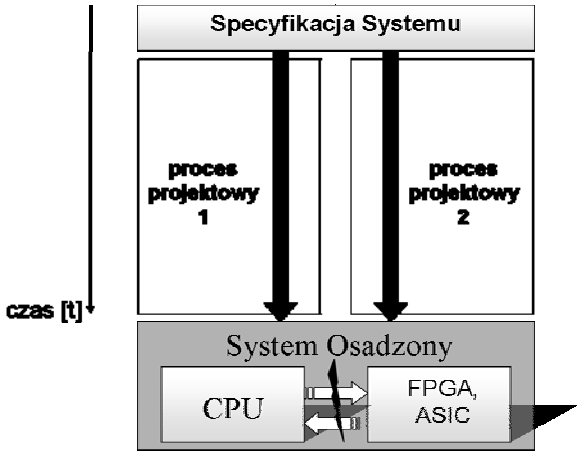
Podczas dekompozycji wykonywany jest podział na podstawie dedykowanego algorytmu podziału i przyjętych kryteriów. Wynikiem dekompozycji jest specyfikacja programowa oraz specyfikacja sprzętowa, które w połączeniu ze współbieżnymi mechanizmami komunikacji i sterowania pozwalają na funkcjonalne i zarazem optymalne odwzorowanie pracy systemu. Istota procesu dekompozycji tkwi w spójności i integralności obu części wynikowych zarówno pod względem współpracy (wymiany informacji) poszczególnych elementów, jak i ich połączeń. Komponenty zakwalifikowane do jednej z list powinny stanowić ściśle powiązany ze sobą podzbiór głównej sieci połączeń i elementów. Takie rozwiązanie minimalizuje czas potrzebny do wymiany informacji pomiędzy modułami sprzętowymi i programowymi, zwiększając w ten sposób częstotliwość pracy całego systemu.

Specyfikacja heterogeniczna

W projektowaniu heterogenicznym zintegrowanych sprzętowo-programowych systemów cyfrowych, specyfikacja systemu formalizowana jest za pośrednictwem niezależnych od siebie, sprecyzowanych i dostosowanych do tego celu języków. Podział specyfikacji funkcjonalnej systemu dokonywany jest przez projektanta w początkowej fazie projektowej, rysunek 2.4. Za przykład mogą posłużyć języki: C ANSI dedykowany do specyfikowania programu dla procesora oraz język VHDL pozwalający na opis zachowania i implementację część sprzętowej systemu.

Głównym aspektem w projektowaniu heterogenicznym jest walidacja systemu oraz opracowanie poprawnego interfejsu komunikacyjnego: wewnętrznego

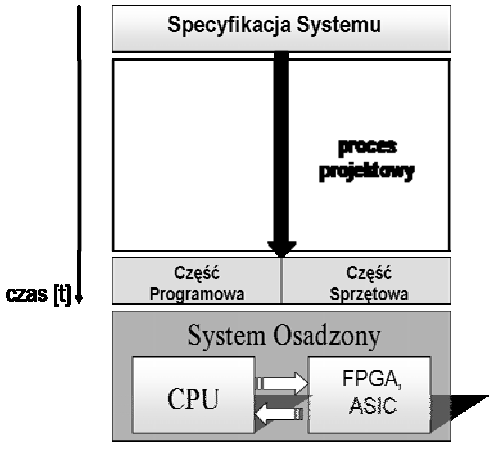
program↔sprzęt i zewnętrznego. Niezbędna jest zaprojektowanie wirtualnego systemu symulacyjnego, który umożliwiałby integrację w jądrze symulatora całego systemu cyfrowego specyfikowanego za pomocą różnych języków opisu.



Rysunek 2.4 Projektowanie heterogeniczne

Dostępne są technologie gwarantujące wsparcie dla projektowania heterogenicznego. Rozwiązania firmy Coware całkowicie wspierają projekty hybrydowe o składowych C, C++, VHDL, Verilog [ieee05a]. Standaryzowane przez konsorcjum IEEE technologie PLI oraz VHPI [ieee05b] umożliwiają na połączenie kodów C++ Verilog, C++ VHDL w dowolnym symulatorze języków HDL, np. ModelSIM firmy MetorGraphics[ment06]. Cykl projektowy kończy walidacja wyników kosymulacji programowo-sprzętowej systemu.

Wadą projektowania heterogenicznego jest brak właściwej analizy formalnej specyfikacji systemu przed podziałem na część sprzętową i programową. Trudno (bądź nie jest to możliwe) w fazie początkowej precyzyjnie określić, która część systemu powinna zostać wykonana jako implementacja sprzętowa, a która wykonywana jako program przez procesor. W rezultacie proces weryfikacji i walidacji wydłuża się, co jest efektem niepożądanym.



Rysunek 2.5 Projektowanie homogeniczne

Specyfikacja homogeniczna

W projektowaniu homogenicznym system poddawany jest podziałowi na program i sprzęt dopiero w końcowej fazie procesu projektowego (rysunek 2.5), a nie jak

w projektowaniu heterogenicznym w fazie początkowej. Do głównych zalet zalicza się możliwość podejmowania kluczowych decyzji dotyczących podziału systemu na sprzęt i program w końcowym etapie projektowym, tj. w czasie, gdy znane są dokładne informacje na temat parametrów, możliwości i ograniczeń badanego systemu. W projektowaniu homogenicznym specyfikacja systemu musi zostać poddana translacji z opisu abstrakcyjnego do formalnego modelu pośredniego systemu. Proces „tłumaczenia" może napotkać na szereg problemów dotyczących konwersji składni lub reprezentacji zapisów specyfikacji systemu w zdefiniowanym modelu pośrednim. Wiele narzędzi projektowania zintegrowanego do specyfikowania funkcjonalności systemu stosuje podzbiory wybranych języków, jednocześnie wprowadzając rozszerzenia funkcjonalne. Za przykład może posłużyć pakiet COSYMA z językiem Cx [ÖsBe97]. System Vulcan wykorzystuje rozszerzenie języka C – HardwareC [KuMi88], natomiast Lycos [MaGr98] i Castle do specyfikacji systemu stosują język C. Inne narzędzia wykorzystują języki opisu sprzętu HDL. Stosowane są również języki specjalizowane takie jak Esterel [Berry91] pakietu Polis [BaCh97] , SpecSync.

W rozprawie metodę projektowania mikrostruktury cyfrowej oparto na schemacie specyfikacji homogenicznej.

2.1.3. Style dekompozycji systemowej

W literaturze znane są systemy wspierające bądź realizujące zadania zintegrowanego projektowania sprzętu i oprogramowania. W dziale zagadnień teoretycznych pracy omówione zostaną trzy charakterystyczne implementacje metodologii projektowania zintegrowanego: a) COSYMA, b) VULCAN, c) POLIS. Trwają badania i prace nad udoskonalaniem i wprowadzaniem do przemysłu projektów wspomagających zintegrowane projektowanie sprzętu i programowania.

COSYMA

Jednym z pierwszych systemów do kosyntezy systemów sprzętowo-programowych był system Cosyma (ang. Cosynthesis of Embedded Micro Architectures) [ÖsBe97]. System przeznaczony do projektowania cyfrowych systemów osadzonych. Wynikiem ukończonego procesu dekompozycji są dwie specyfikacje, sprzętowa i programowa, implementowane w odpowiednie moduły systemu. W tym celu wykorzystywane są: kompilator języka C – do utworzenia części programowej oraz wysokiego poziomu narzędzia projektowania sprzętu – w celu syntezy logicznej i implementacji części sprzętowej. Elementarne operacje są przenoszone w jednostkach odpowiadających sprzętowym blokom funkcjonalnym. Komunikacja między procesami zrealizowana jest poprzez predefiniowane funkcje języka C, mające dostęp do abstrakcyjnych kanałów komunikacyjnych. Część sprzętowa i programowa komunikują się ze sobą poprzez wspólną pamięć (ang. shared memory).

Badania nad pakietem COSYMA przeprowadzono na Sharif Univeristy of Technology, San Antonio USA. Wyniki przyspieszenie pracy systemu cyfrowego (składającego się z procesora, pamięci i układu ASIC) zaprojektowanego z wykorzystaniem rozwiązań COSYMA wynoszą 2- 3-krotnego przyspieszenie pracy systemu cyfrowego [Guda05].

System Vulcan

Vulcan [RGCM94] jest przykładem systemu, rozpoczynającego pracę od specyfikacji czysto sprzętowej, z której niekrytyczne zadania przenoszone są do części programowej w celu obniżenia kosztów. Program jest złożeniem dwóch różnych części. Vulcan I dzieli funkcjonalny system między układy ASIC. Gdy rozdrobnienie projektu jest wystarczające, czyli projekt przedstawiony jest jako zbiór składający się z komponentów realizowalnych w układach ASIC oraz połączeń między nimi, wtedy Vulcan II dzieli system na część sprzętową i programową. Dekompozycja polega na specyfikacji całego systemu jako sprzętu, a następnie na podstawie określonych ograniczeń dokonywana jest ocena systemu oraz przenoszenie poszczególnych zadań do programu. Elementem wejściowym systemu Vulcan I są zadania oraz rozmieszczenia układów scalonych, natomiast efektem działania jest układ operacji, które są stopniem wejściowym dla narzędzi syntezy wyższego poziomu. Vulcan I łączy operacje, które będą implementowane w sprzęcie. Vulcan II dekomponuje wejścia zadań do logicznych wyrażeń blokowych. Celem architektury jest jeden procesor i jeden komponent ASIC, jedna magistrala oraz jedna globalna pamięć, przez którą komunikują się wszystkie komponenty systemu zarówno programowe, jak i sprzętowe.

Specyfikacja systemu tworzona jest w języku Hardware C, opracowanym specjalnie do syntezy wysokiego poziomu. Jest to implementacja standardu ANSI C, rozszerzonego o typy danych przeznaczone do: specyfikacji części sprzętowej, opisu ograniczeń czasowych oraz określania zależności danych. Opis zachowania podzielony jest na wątki w punktach o niedeterministycznych opóźnieniach, przy czym opóźnienie każdego wątku jest ograniczone. Wątki programowe wykonywane są pod kontrolą systemy operacyjnego. Komunikacja wątków następuje poprzez kolejki.

System POLIS

Polis [BaCh97] jest narzędziem wspierającym zintegrowane projektowanie sprzętu i oprogramowania dla reakcyjnych systemów osadzonych. Został zaprojektowany na Uniwersytecie Kalifornijskim w Berkeley.

Opis systemu specyfikowany jest w języku Esterel, który jest językiem synchronicznym, wspierającym równoległość, gdzie podstawą są zdarzenia. Cały system jest reprezentowany jako zbiór współdziałających procesów, które komunikują się ze sobą za pomocą sygnałów i zdarzeń. Na podstawie opisu zachowania systemu specyfikowanego w języku Esterel generowany jest zapis CFSM (ang. Codesign Finite State Machine). CFSM jest obiektem za pomocą, którego w pakiecie Polis jest opisywany model pośredni. Cała ścieżka projektowa opiera się na szkielecie typowego systemu wspierającego zintegrowane projektowanie sprzętu i oprogramowania. Polis używa narzędzia Ptolemy podczas kosymulacji (ang. co-simulation). W czasie tego procesu projektant może wybierać pomiędzy sprzętową a programową implementacją każdego składnika CFSM, typu zegara, szybkości zegara procesora, na którym program będzie uruchomiany oraz rodzaju systemu harmonogramowania (ang. scheduler). Dekompozycja na część sprzętową i programową nie jest automatyczna i decyzja zależy od projektanta. To on decyduje kiedy podzielić projekt na syntezę sprzętową i syntezę programową.

Synteza sprzętowa – podsieci CFSM, które zostały zakwalifikowane do tej części zostają poddane syntezie i optymalizacji przy pomocy narzędzia SIS. Każda podsieć jest reprezentowana jako specyfikacja Register-Transfer Level może być mapowana do formatów BLIF, XNF, VHDL, VERILOG. Synteza programowa – podsieci CFSM, które zostały zakwalifikowane do tej części zostają zmapowane do programowej struktury, która zawiera procedury dla każdej podsieci CFSM. Na syntezę programową składa się również prosty system czasu rzeczywistego.

System Polis generuje system operacyjny, który jest odpowiedzialny za komunikację pomiędzy modułami syntezy sprzętowej i programowej, kontroluje kolejkowanie sieci CFSM w syntezie programowej, tworzy sterowniki urządzeń, które zajmują się komunikacją pomiędzy syntezą sprzętową i programową.

2.2. Modele specyfikacji formalnej systemu

Pierwszym krokiem w projektowaniu systemowym jest opracowanie specyfikacji funkcjonalnej systemu. Natomiast pierwszy krok podczas formalizowania specyfikacji determinuje funkcjonowanie systemu. W celu zrozumienia i uszeregowania tej funkcjonalności w znaczeniu semantycznym, można użyć wielu różnych modeli koncepcyjnych. W rozdziale zostaną omówione modele formalne specyfikujące funkcjonalność systemu oraz architektury implementacyjne, które są najczęściej wykorzystywane w zintegrowanym projektowaniu sprzętu i oprogramowania.

Model

Projektowanie systemowe jest procesem implementacyjnym określonej funkcjonalności systemu poprzez wykorzystanie fizycznych komponentów. W związku z tym, cały proces projektowania systemowego musi rozpocząć się od sformalizowania pożądanej specyfikacji funkcjonalnej. Wymagane jest precyzyjny opis zachowania systemu. Najbardziej pożądaną cechą projektowania zintegrowanego jest rozważanie systemu jako kolekcji prostych składowych systemu, mniejszych części. W rozdziale zostaną zaprezentowane metody dekomponujące funkcjonalność na proste części. Zazwyczaj, cechą wyróżniającą te metody są rodzaje/typy składowych części systemu oraz reguły komponowania tych części w jeden systemy. Każdą wyróżnioną metodę nazywamy modelem [GaVa94].

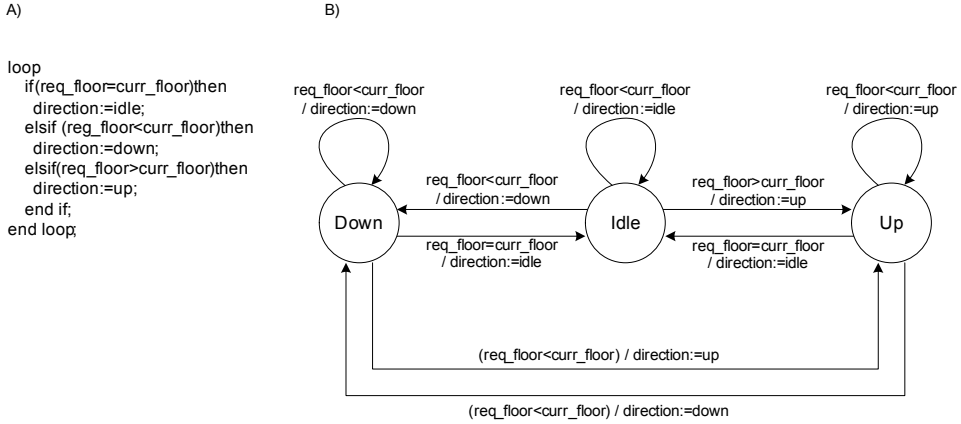
Model musi posiadać cechy:

- jednoznaczna reprezentacja opisu i jej rozumienia – model formalny,
- istnienie mechanizmów i struktur pozwalających na swobodny opis pełnej funkcjonalności systemu,
- prostota i przerzystość formuł specyfikacji,
- łatwość modyfikacji,
- wspomaganie i ułatwianie zrozumienia funkcjonowania systemu.

Model jest systemem formalnym zawierającym obiekty i reguły komponowania systemu oraz jest wykorzystywany do opisywania charakterystyki systemu.

Głównym celem modelu jest zapewnienie i dostarczenie inżynierowi abstrakcyjnego poglądu na funkcjonowanie i strukturę systemu. Rysunek 2.7

przedstawia dwa różne modele kontrolera windy. Rysunek 2.7.a) opisuje kontroler jako zbiór instrukcji programu. Natomiast rysunek 2.7.b) przedstawia kontroler jako maszynę stanów [GaVa94].



Rysunek 2.7 Specyfikacja zachowania sterownika windy: a) kod języka programowania, b) maszyna stanów FSM [GaVa94]

Jak można zauważyć, każdy model reprezentuje zbiór obiektów i interakcji pomiędzy nimi. Zaletą wykorzystywania w projektowaniu zintegrowanym wielu różnych modeli jest możliwość reprezentacji różnych widoków/poglądów na system, przez eksponowanie jego różnych charakterystyk. Rozważając przykład z rysunku 2.7, model maszyny stanów nadaje się najbardziej do reprezentacji czasowego zachowania się systemu. Umożliwia klarowne przedstawienie trybów pracy systemu oraz transakcji/zależności występujących pomiędzy nimi pobudzanymi przez zdarzenia wewnętrzne lub zewnętrzne. Natomiast algorytm programu nie prezentuje tak klarownie stanów pracy systemu, w porównaniu do maszyny stanów. Jednak znakomicie przedstawia on zależności wyjść od wejść uwarunkowanych stanem pracy systemu, przez co dobrze nadaje się do reprezentacji proceduralnej systemu.

Architektura

Różne modele wymagają i są stosowane w różnych domenach aplikacyjnych. Modelowanie systemów czasu rzeczywistego znacząco różni się od modelowania, np. systemów baz danych – gdzie pierwszy system skupia się na zachowaniu w czasie, a drugi na reprezentacji danych. Dobranie właściwego modelu umożliwiającego sformalizowanie pełnej specyfikacji systemu oraz opracowanie specyfikacji, nie kończy procesu projektowego. Pozostaje pytanie: jak formalnie system ma zostać zrealizowany ? Następnym krokiem jest transformacja modelu do wybranej architektury sprzętowej – implementacja modelu przez specyfikowanie komponentów modelu formalnego oraz połączeń między nimi.

Model opisuje jak system pracuje, natomiast architektura przedstawia w jaki sposób system zostanie zrealizowany fizycznie.

> **Definicja 2.2** *Proces projektowy jest zbiorem zadań, które transformują model do architektury.*[GaVa94]

Na początku procesu projektowego, znana jest tylko funkcjonalność systemu. Zadaniem projektanta jest opisanie funkcjonalności w wybranym języku bazując na najbardziej odpowiednim modelu. Architektura zostanie określona podczas

procesu projektowego w efekcie analiz i precyzowania funkcjonalności składowych części systemu.

Klasyfikacja modeli

W szerokiej gamie metodologii projektowania sprzętu i oprogramowania, projektant systemu może stosować wiele różnych modeli formalnych do specyfikowania funkcjonalności systemu. Można wydzielić pięć kategorii modeli formalnych:

- zorientowane na stany,
- zorientowane na aktywność,
- zorientowane na strukturę,
- zorientowane na dane,
- heterogeniczne.

W kolejnych podrozdziałach opisane zostaną wybrane modele formalne. Wskazane zostaną rozwiązania możliwe do zastosowania w zintegrowanym projektowaniu sprzętowo-programowych mikrosystemów cyfrowych.

2.2.1. Skończona hierarchiczna maszyna stanów

Skończona maszyna stanów (ang. Finite State Machine) [GaVa94,GaZh00] jest przykładem modelu zorientowanego na stany. Jest to najbardziej popularny model opisu kontrolerów logicznych. Zazwyczaj model FSM składa się ze sbioru stanów, tranzycji pomiędzy stanami oraz zbioru akcji przypisanych do tranzycji lub stanów. Formalnie, maszyna stanów jest szóstką:

$$FSM = (X, S, Y, \delta, \lambda), \qquad \textit{(Wzór 2-4)}$$

gdzie:

$X = \{x_1,\ldots,x_i\}$ - *skończony niepusty zbiór wejść;*

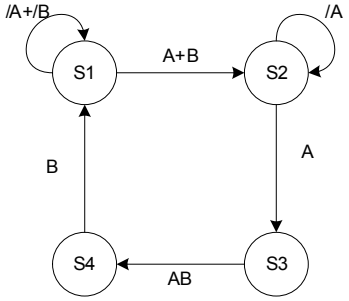
$S = \{s_1,\ldots,s_j\}$ - *skończony niepusty zbiór stanów;*

$Y = \{y_1,\ldots,y_k\}$ - *skończony niepusty zbiór wyjść;*

δ - *funkcja przejść, taka że* $\delta: D_\delta \rightarrow S$, *przy czym* $D_\delta \subset X \times S$;

λ - *funkcja wyjść, taka że* $\lambda: D_\lambda \rightarrow Y$, *przy czym* $D_\lambda \subset X \times S$ *dla automatu Mealy'ego albo* $D_\lambda \subset S$ *dla automatu Moore'a.*

Graficznie FSM przedstawiana jest za pomocą grafu stanów i przejść w postaci łuków skierowanych, rysunek 2.8. Problemy analizy i syntezy skończonych maszyn stanów (automatów sekwencyjnych) poruszane są między innymi w pracach [ZbŁu00,Poch04,Ziel03]



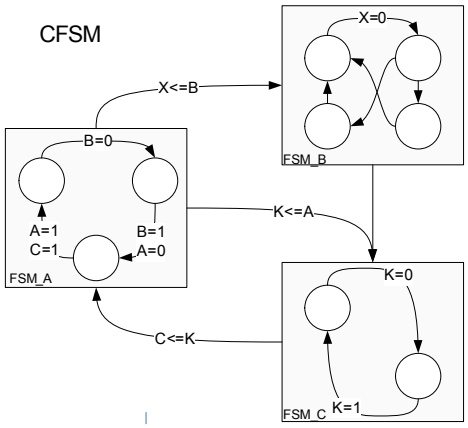
Rysunek 2.8 Przykład specyfikacji zachowania sterownika za pomocą modelu FSM

Wyróżnia się dwa typy modeli FSM: a) zorientowany na miejsca (Moore), b) zorientowany na tranzycje (Mealy); które różnią się definicją funkcji wyjściowej λ. W modelu FSM zorientowanym na tranzycje, wyjścia zależne są od stanu i wejść (λ:D->$S \wedge X$); natomiast w modelu zorientowanym na stan wyjścia zależne są tylko od stanu (λ:D->S). Z praktycznego punktu widzenia, główną różnicą pomiędzy dwoma modelami jest to, że model zorientowany na stany będzie wymagał więcej stanów do realizacji wybranego zadania, niż model zorientowany na tranzycje. W modelu zorientowanym na tranzycje można definiować wiele łuków wskazujących na jeden stan, gdzie do każdego łuku przypisuje się inną akcję.

Model FSM jest wykorzystywany głównie do specyfikacji systemów sterowania, podczas gdy model rozszerzony FSMD (ang. Finite State Machine with Data Path) może być zastosowany do opisu zachowania systemów sterowania i przetwarzania danych. Należy wyraźnie zaznaczyć, że żaden ze wskazanych modeli (FSM, FSMD) nie nadaje się do specyfikowania bardziej złożonych systemów cyfrowych z powodu braku reprezentacji równoległości i hierarchii.

Rozszerzenia FSM

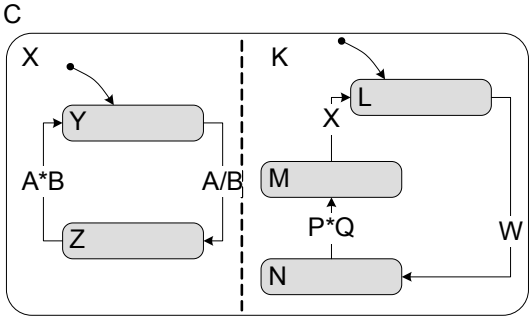
Odpowiedzią na brak wsparcia przez FSM równoległości jest model CFSM (ang. Codesign Finite State Machine). Model CFSM składa się ze zbioru pojedynczych bloków FSM pracujących współbieżnie w stosunku do siebie. Synchronizacja między modułami jest asynchroniczna, natomiast komunikacja wewnętrzna realizowana jest synchronicznie.



Rysunek 2.9 Powiązane automaty FSM

Kolejnym modelem formalnym wykorzystywanym w projektowaniu systemów cyfrowych, a rozszerzającym model FSM, jest automat hierarchiczny z równoległością HCFSM (ang. Hierarchical Concurrent Finite State Machine) [DrHa89, GaVa94]. Wprowadza on do klasycznego automatu FSM hierarchię i równoległość, pozwalając tym samym uniknąć wykładniczego wzrostu liczby stanów i przejść. Podobnie jak w przypadku automatu FSM występują w nim stany i przejścia. Te pierwsze mogą być dodatkowo dekomponowane na stany podrzędne (sekwencyjne lub równoległe). Możliwe jest także, dowolne lub na tym samym

poziomie hierarchii, stosowanie przejść. Przykładem automatu HCFSM jest diagram Statecharts [GaVa94], rysunek 2.10.



Rysunek 2.10 Model strownika reprezentowany diagramam statechart

2.2.2. Sieci Petriego

Model sieci Petriego[Petr62,Mura89,GaVa94] jest kolejnym typem modelu zorientowanego na stany, zaprojektowany na potrzeby specyfikowania zachowania systemu, który składa się ze wspólnie kooperujących współbieżnych zadań. Sieci Petriego znalazły szerokie zastosowanie w projektowaniu systemów cyfrowych szczególnie ze względu na: naturalny zapis, elastyczność w dostosowaniu sieci do opisu danych zjawisk i rozbudowany aparat matematyczny, wykorzystywany w analizie poprawności. Formalnie sieć Petriego jest czwórką [Mura89]:

$$PN = (P,T,F,M0,u) \qquad \textit{(Wzór 2-5)}$$

gdzie:

P = {p1, p2, ..., pm} jest skończonym nie pustym zbiorem miejsc;

T = {t1, t2, ..., tn} jest skończonym nie pustym zbiorem tranzycji;

$F \subseteq (P \times T) \cup (T \times P)$ jest skończonym nie pustym zbiorem łuków;

M0: $P \rightarrow \{0, 1, 2, ...\}$ jest oznaczeniem początkowym sieci;

u: $P \rightarrow K$ definiuje liczbę żetonów w każdym miejscu, $K \in N$;

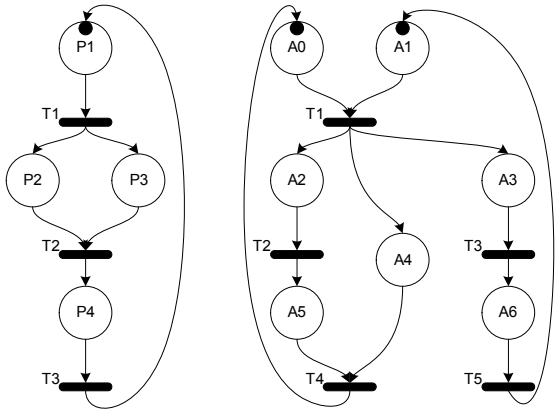
$P \cap T = \varnothing$ oraz $P \cup T \neq \varnothing$.

Model sieci Petriego zbudowany jest ze zbioru miejsc, tranzycji, łuków oraz żetonów. Sieć pracuje poprzez zmianę oznakowania sieci w wyniku przemieszczania się znaczników/żetonów pomiędzy miejscami sieci.

Zmiana oznakowania sieci uwarunkowana jest realizacją tranzycji. Tranzycja może zostać odpalona (zrealizowana) tylko wtedy gdy wszystkie jej miejsca wejściowe posiadają przynajmniej jeden żeton. Poprzez „odpalenie" tranzycji rozumie się zabrania po jednym żetonie ze wszystkich miejsc wejściowych (konsumowanie) i wstawienie po jednym żetonie do wszystkich miejsc wyjściowych tranzycji (produkowanie). W modelu sieci Petriego tranzycje połączone są z miejscami za pomocą łuków skierowanych, obiekt tranzycji znajduje się zawsze pomiędzy miejscami. Przykład reprezentacji graficznej sieci Petriego przedstawia rysunek 2.11.

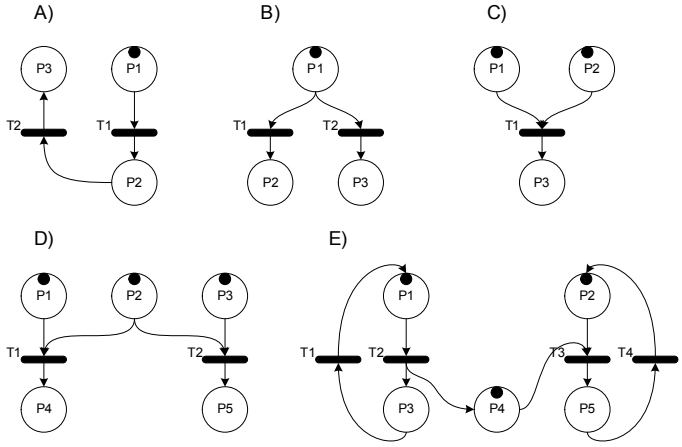
Geneza popularności oraz praktycznych zastosowań sieci Petriego wynika z naturalnych mechanizmów teorii pozwalających na efektywne modelowanie szerokiej gamy charakterystyk systemów cyfrowych: sekwencyjnych, równoległych,

asynchronicznych, synchronicznych, sprzętowo-programowych. Na rysunku 2.12 przedstawiono wybrane schematy modelowania zachowania i interakcji zdarzeń możliwych do opisania za pomocą sieci Petriego. Rysunek 2.12.a) przedstawia modelowanie sekwencji, gdzie tranzycja t1 odpalana jest po tranzycji t2. Żeton przemieszcza się poczynając od miejsca P1, przez P2 do miejsca P3.



Rysunek 2.11 Przykład sieci Petriego

Na rysunku 2.12.b) zobrazowano zdarzenie niedetermistyczne, gdzie dwie tranzycje są aktywne, ale tylko jedna z nich może zostać odpalona. Rysunek 2.12.c) przedstawia model synchronizacji, gdzie tranzycja może zostać odpalona wtedy i tylko wtedy, gdy oba miejsca wejściowe będą zawierały żeton. Na rysunku 2.12.d) przedstawiono model „walki o zasoby", gdzie dwie tranzycje „walczą" o ten sam żeton rezydujący w miejscu P2. Specyficzny model równoległości przedstawia rysunek 2.12.e), na którym dwie tranzycje t2 i t3 mogą być odpalane jednocześnie. Jest to model dwóch równoległych procesów, producenta i konsumenta; żeton w miejscu centralnym produkowany jest przez tranzycję t2 i konsumowany przez t3.



Rysunek 2.12 Schematy modelowania systemu cyfrowego za pomocą sieci Petriego [GaVa94]

Interpretowana sieć Petriego

Ogólną definicję sieci Petriego można rozszerzyć, dodając do niej różne obiekty (wejścia i wyjścia) lub nadając restrykcje na istniejące obiekty (predykaty tranzycji, restrykcje czasowe). Możliwe jest także wprowadzenie hierarchii w budowie sieci.

Ze względu na elementy modyfikujące, powstały rozszerzenia sieci Petriego, takie jak: systemy miejsc i tranzycji, sieci kolorowane, hierarchiczne, interpretowane oraz wiele innych. Każda z tych odmian jest przeznaczona i przystosowana do opisu różnych własności. Obiekty dodane do sieci Petriego w danym rozszerzeniu są łączone z obiektami innego rozszerzenia. Najbardziej interesującymi ze względu na modelowanie systemów i układów cyfrowych są interpretowane sieci Petriego.

Model formalny sieci Petriego zostaje uzupełniony o dodatkowe definicje. Interpretowaną siecią Petriego opisuje się uporządkowaną szóstką [Mura89,Adam91]:

$$PN_{IO} = (PN, X, Y, \rho, \lambda, \gamma) \qquad \textit{(Wzór 2-6)}$$

gdzie;

PN jest siecią żywą i bezpieczną;

X jest zbiorem stanów wejść;

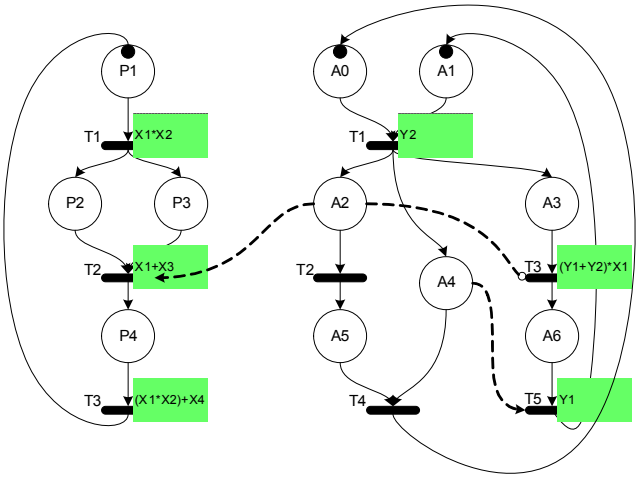
Y jest zbiorem stanów wyjść;

$\rho: T \rightarrow 2^X$ jest funkcją, która każdej tranzycji przyporządkowuje jednoznacznie podzbiór stanów wejść $X(t)$.

$\lambda: [M_0\rangle \rightarrow 2^Y$ jest funkcją, która każdemu znakowaniu sieci M przyporządkowuje jednoznacznie pewien stan wyjść $Y(M)$.

$\gamma: (M \times X) \rightarrow 2^Y$ jest funkcją, która przy znakowaniu sieci M oraz podzbiorze stanów wejść przyporządkowuje jednoznacznie podzbiór stanów wyjść $Y(M,X)$

[gore2]Graficzna reprezentacja sieci z rysunku 2.11 została rozszerzona o zdania logiczne nakładające kolejne warunki realizacji tranzycji i przedstawiona na rysunku 2.13.

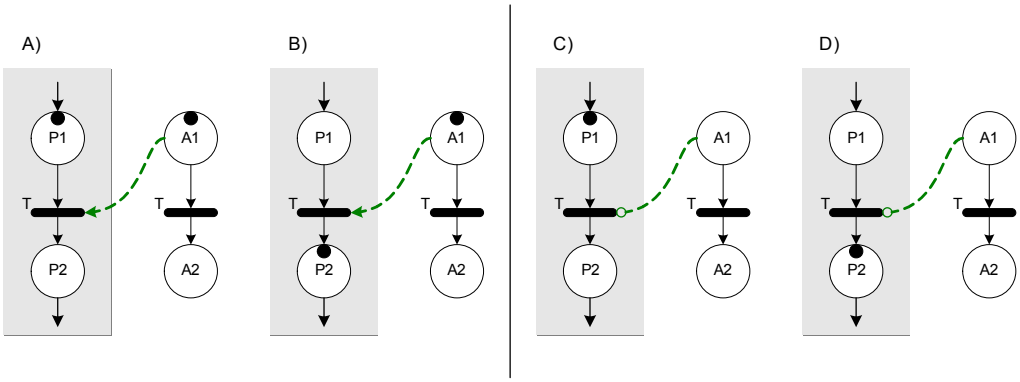


Rysunek 2.13 Interpretowana sieć Petriego

W interpretowanej sieci Petriego generowanie wyjść może być realizowane na dwa sposoby. Pierwszy dotyczy modelowania wyjść typu Moore'a, gdzie funkcja sygnału wyjściowego przyporządkowana jest do miejsca i zależy tylko od stanu

oznaczenia sieci. Drugi dotyczy modelowanie wyjść typu Mealy'ego, gdzie funkcja sygnału wyjściowego przypisana jest do tranzycji, a wartość wyjścia zależna jest od stanu lokalnego miejsca i wejścia. Dodatkowo, sygnały wejściowe mogę być wykorzystane do budowy zdań logicznych (predykaty) warunkujących realizację tranzycji, rysunek 2.13. Wówczas, tranzycja może być zrealizowana, gdy wszystkie miejsca wejściowe są oznaczone i spełniony jest predykat logiczny tranzycji.

W celu zwiększenia elastyczności modelowania systemów cyfrowych, do modelu sieci Petriego wprowadzono tzw. łuki zezwalające i zabraniające dodające kolejne uwarunkowanie realizacji tranzycji. Reprezentację graficzną prezentuje rysunek 2.14.



Rysunek 2.14 Łuki zezwalające i zabraniające sieci Petriego

W trakcie realizacji tranzycji, gdzie miejsce wejściowe połączone jest łukiem zezwalającym lub zabraniającym, tranzycja nie konsumuje znacznika miejsca. Na rysunku 2.14.a) tranzycja zostanie odpalona pod warunkiem, gdy miejsce P1 będzie przetrzymywało znacznik, natomiast w modelu 2.14.b) tranzycja będzie gotowa do realizacji, gdy miejsca P1 nie będzie przechowywało żetonu.

Analiza sieci Petriego

Bogaty aparat matematyczny sieci Petriego może być wykorzystane do analizy i walidacji szeregu interesujących własności systemu cyfrowego, takich jak: bezpieczeństwo, żywotność, hierarchia, składowe automatowe, ścieżka danych. Większość algorytmów i metod analizy sieci Petriego zostało przedstawionych w pracach [Mura89, AgWę03]. Dla potrzeb rozważań prowadzonych w pracy omówiono wybrane własności sieci Petriego.

Bezpieczeństwo sieci

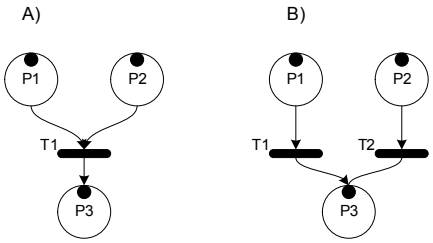
Sieć jest bezpieczna, gdy dla każdego możliwego oznakowania sieci liczba żetonów w każdym miejscu zawiera się w przedziale K={0,1} [Mura89,BaKu93,Bili96, Węgr98].

Definicja 2.2 *Niech PN będzie siecią Petriego. Sieć PN jest siecią bezpieczną wtedy i tylko wtedy, gdy*

$$\forall p \in P, K\{0,1\}. \qquad \textit{(Wzór 2-7)}$$

Z punktu widzenia urządzenia cyfrowego, niedopuszczalne jest ponowne uruchomienie procesu, który aktualnie jest realizowany i nie został jeszcze zakończony. Realizowane zadanie musi się zakończyć, tak aby można było ponownie je wykonać. Bezpieczeństwo dotyczy również problemu wysyłania na

jeden sygnał (linię) informacji z wielu źródeł, wprowadzając daną linię w stan niezdefiniowany (X). Jest to charakterystyczna właściwość, determinująca poprawność pracy systemu cyfrowego, niezbędna w realizacji złożonych systemów cyfrowych. Rysunek 2.15 przedstawia fragmenty dwóch sieci niebezpiecznych. Na rysunku 2.15.a) miejsce P3 przechowuje już jeden żeton (zakładana pojemność miejsca jest równa 1). Odpalenie tranzycji T1 wprowadza drugi żeton do P3 wprowadzając destabilizację pracy systemu.



Rysunek 2.15 Przykład niebezpiecznych sieci Petriego

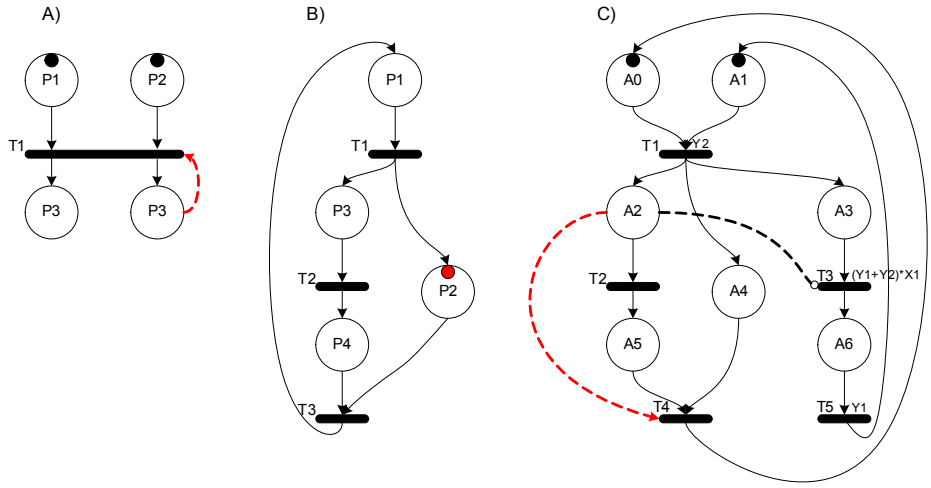
Przypadek 2.15.b) jest analogiczny pod względem bezpieczeństwa, gdyż obie tranzycje T1 i T2 skonsumują żetony z miejsc P1 i P2 oraz przekażą do swojego miejsca wyjściowego po jednym żetonie. W rezultacie miejsce P3 będzie przetrzymywał trzy żetony.

Żywotność sieci

Żywotność jest to właściwością sieci Petriego, która gwarantuje istnienie przynajmniej jednej tranzycji, która będzie mogła być zrealizowana w każdym znakowaniu sieci [Mura89,BaKu93,Bili96,Węgr98].

> **Definicja 2.3** *Niech PN będzie siecią Petriego. Sieć PN jest siecią żywą wtedy i tylko wtedy, gdy dla każdego znakowania osiągalnego* $M \in R(PN, M_0)$ *możliwe jest przygotowanie każdej tranzycji* $t \in T$ *(tzn. istnieje* $M' \in R(PN, M)$*) w wyniku odpaleń sekwencji tranzycji* σ*, takiej, że* $\delta(M, \sigma) = M'$.

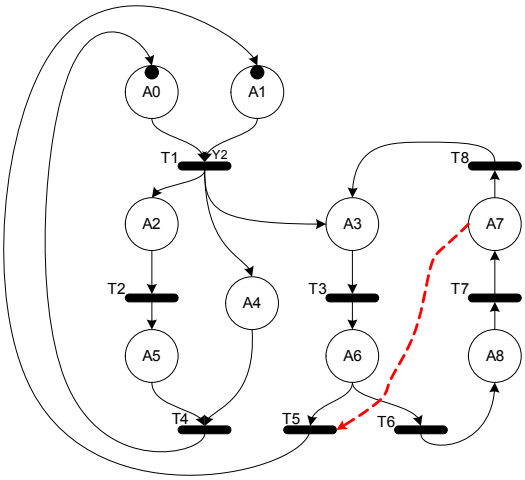
Dzięki analizie żywotności sieci, projektant ma pewność, że zrealizowany system cyfrowy nie znajdzie się w stanie globalnym, w którym przestanie reagować na bodźce zewnętrzne, bądź przejdzie do pewnego (nieprzewidzianego) trybu pracy, z którego już nie wyjdzie. Rozważając żywotność sieci rozpatruje się zagadnienia blokad i pułapek.



Rysunek 2.16 Przykłady sieci Petriego nie spełniających warunków żywotności

Rysunek 2.16 przedstawia wybrane przypadki sieci, zawierające blokady i nie spełniające warunku żywotności. Na przykładzie 2.16.a) tranzycja T1 nigdy nie zostanie zrealizowana, ponieważ jej realizacja uwarunkowane jest oznaczeniem miejsca P3, które oczekuje na znacznik właśnie od tranzycji T3. Tranzycja T3 jest zawsze martwa Na rysunku 2.16.b) miejsce P2 należy do zbioru miejsc inicjalizujących. Tranzycja T3 posiada dwa miejsca wejściowe, gdzie zawsze tylko jedno miejsce będzie posiadać znacznik (miejsce P2). Tranzycja T3 nie zostanie zrealizowana, ponieważ miejsce P4 nigdy nie zostanie oznaczone. Natomiast w sieci z rysunku 2.16.c) tranzycja T4 blokuje przepływ znacznika do miejsca A0, ponieważ realizacja T4 uwarunkowana jest łukiem zezwalającym miejsca A2. Końcowym oznakowaniem sieci będzie A5,A4,A1.

Kolejną własnością sieci Petriego jest występowanie pułapek. Dotyczy to takiej pracy sieci, gdzie zmiana znakowania tej sieci powoduje wprowadzenie znacznika do tego samego miejsca sieci lub innego miejsca, należącego do wyodrębnionego skończonego zbioru miejsc sieci. W rezultacie, znacznik(i) krążą tylko w obrębie określonego podzbioru miejsc sieci uniemożliwiając wydostanie się znacznika do innych części sieci, które stają się martwe. Przykład obrazuje rysunek 2.17.



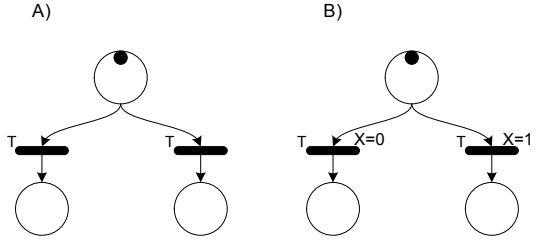
Rysunek 2.17 Przykład sieci z pułapką

Ze względu na łuk zezwalający A7->T5, żeton będzie krążyć w podzbiorze sieci o składowych: A3, A6, A8, A7. Po zakończeniu pierwszego cyklu pracy sieci, do miejsca A1 nigdy nie wpłynie znacznik (niezbędna jest inicjalizacja (restart) pracy sieci). W rezultacie, część sieci staje się nieżywa.

Determinizm sieci

Zachowanie niedeterministyczne występuje na przykład wtedy, gdy jedno miejsce jest wejściem dla dwóch tranzycji. W takim przypadku, obie tranzycje gotowe są do realizacji. Jednak powstaje pytanie: która z dwóch tranzycji powinna zostać zrealizowana jako pierwsza. Problem dotyczy wyścigu tranzycji w konsumpcji znacznika (tranzycja w trakcie realizacji usuwa znacznik z miejsca wejściowgo). Nie można określić zachowania się układu dla takiej konfiguracji sieci (rysnek 2.18.a). W celu wyeliminowania konfliktu, w sieciach interpretowanych, na

tranzycję narzuca się warunki logiczne wykluczające się wzajemnie, co gwarantuje realizację tylko jednej tranzycji (rysunek 2.18.b).



Rysunek 2.18 Determinizm sieci Petriego

~~**6.Hierarchiczna sieć Petriego**~~

~~Jest hierarchiczna HHPN[]; sieć wzbogacona o obiekty typu makro: makromiejsce, makrotranzycja. Makromiejsce jest elementem oznaczonym graficznie w sieci przez podwójną linię okręgu. Instancjonuje kolejną sieć Petriego niższego rzędy budując w ten sposób hierarchię funkcjonalną systemu. Zdeponowanie żetonu do makromiejsca aktywuje pracę zainstancjonowanej sieci poprzez wprowadzenie żetonów do wszystkich miejsc inicjalizujących. Wyróżnia się miejsca wejściowe (inicjalizujące) i miejsca końcowe podsieci aktywujące tranzycję wyższego poziomu – koniec pracy makromiejsca. Natomiast makrotranzycja instancjonuje sieci w obrębie dwóch tranzycji globalnych: wejściowej i wyjściowej; które przekazują sterowanie do instancjonowanej podsieci.~~

Sieć Petriego w modelowaniu układu i systemu cyfrowego

Przed siecią Petriego specyfikującą zachowanie się układów lub systemów cyfrowych, stawiane są wymagania wynikające z charakterystyki modelu układu cyfrowego, zdarzeń zachodzących w środowisku dyskretnym, elastyczności procesu modelowania zachowania oraz innych czynników [Adam98,Adam99]. Sieć Petriego opisująca zachowanie systemu cyfrowego powinna spełniać następujące kryteria [Adam98,Adam99,Skow00]:

- sieć jest żywa
- sieć jest bezpieczna,
- sieć jest deterministyczna,
- sieć jest interpretowana jako współbieżna maszyna stanów,
- sieć jest hierarchiczna,
- sieć przechowuje ścieżkę danych.

Główna zaletą modelu systemu cyfrowego, bazującego na sieci Petriego w porównaniu z innymi modelami zorientowanymi na stan, jest wparcie dla opisu współbieżności oraz bogaty aparat matematyczny. Do wad należy zakwalifikować utratę czytelności, powodującą trudności w zrozumieniu funkcjonalności opisanego systemu, gdy jest on poważnie rozbudowany (skomplikowany) o liczne interakcie między hierarchicznymi madułami.

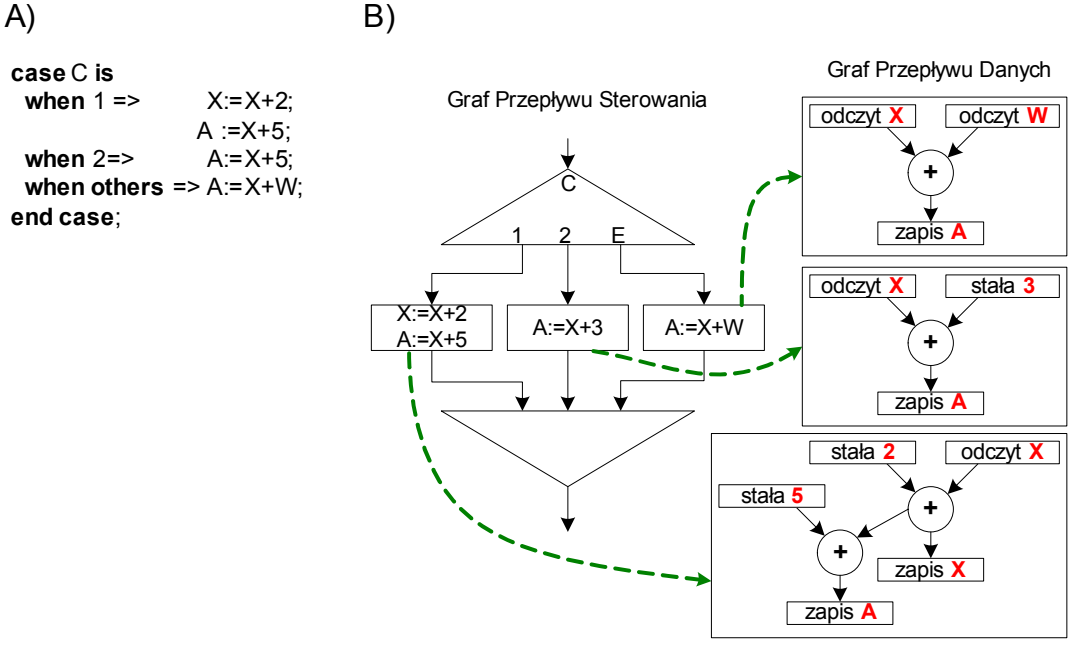
Synteza układów współbieżnych opisanych interpretowaną siecią Petriego jest zagadnieniem złożonym. Składa się na nią kilka technik, takich jak: implementacja bezpośrednia (ang. Direct Implementation), kodowanie znakowania (ang. Marking Encoding), kodowanie miejsc i dekompozycja.

Jednym z etapów implementacji sieci Petriego jest kodowanie miejsc. Jest ono przeprowadzane bezpośrednio na sieci uzyskanej podczas projektowania lub dopiero po zdekomponowaniu sieci na składowe automatowe. Istnieje kilka sposobów kodowania miejsc [Adam86, Adam90a, Adam91, Amro90, BaKu93]:

- kodowanie *jeden do jednego* (ang. *one to one*, *one hot encoding* lub *concurrent one hot*) polegające na tym, że każdemu miejscu odpowiada jeden przerzutnik,
- kodowanie polegające na dekompozycji sieci Petriego na składowe automatowe i odrębnym kodowaniu każdej ze składowych. W tym wypadku sieć Petriego musi być konserwatywna, czyli w każdym możliwym oznakowaniu liczba markerów musi być taka sama. W celu osiągnięcia konserwatywności sieci niejednokrotnie konieczne jest stosowanie dodatkowego miejsca spoczynkowego, co powoduje zwiększenie liczby przerzutników,
- kodowanie hierarchiczne polegające na utworzeniu makrosieci. Osobny wektor kodu użyty jest dla makrosieci oraz osobny wektor dla każdego makromiejsca w makrosieci Petriego. Łączny kod miejsca jest superpozycją wszystkich przypisanych mu wektorów powiązanych z makromiejscami, do których należy oraz z kodu identyfikującego rozpatrywane miejsce w makromiejscu na najniższym szczeblu hierarchii,
- kodowanie metodą heurystyczną łączące zalety kodowania *jeden do jeden* z kodowaniem hierarchicznym.

2.2.3. *Graf przepływu danych i sterowania CDFG*

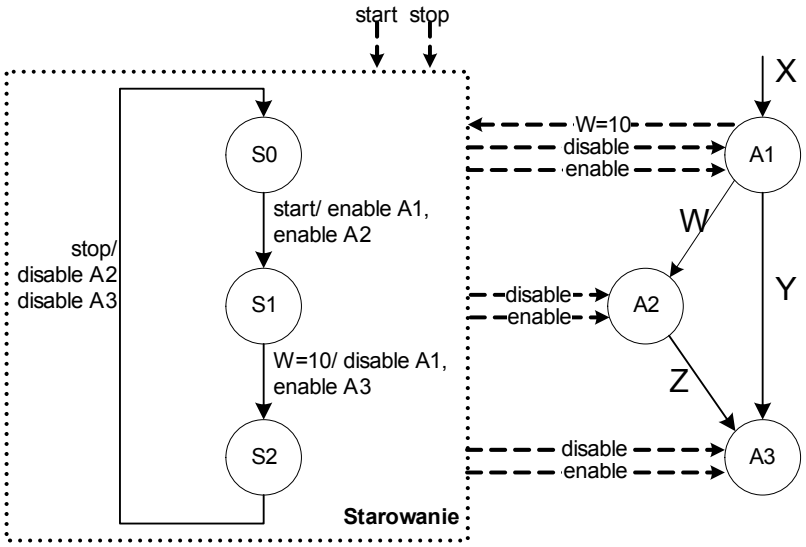
Graf przepływu danych i sterowania jest modelem heterogenicznym zaprojektowanym do połączenia własności i zalet modeli przepływu sterowania CFG (flowchart) oraz graf przepływu danych DFG. Innymi słowy, graf CDFG łączy w sobie reprezentacje przepływu danych (aktywność) oraz reprezentację sekwencji operacji. Za pomocą jednego modelu możliwe jest przedstawienie zależności danych i sekwencji sterowania, rysunek s.2.19.



Rysunek 2.19 Graf przepływu danych i sterowania [GaVa94]

Na rysunku 2.19b) przedstawiono graf CDFG reprezentujący program opisany na rysunku 2.19a). Programowe konstrukcje sterowania zostały przedstawione jako punkty kontroli w ścieżce sterowania, połączone liniami z instrukcjami przetwarzania danych. Model CDFG nie jest tylko ograniczony do reprezentacji programowych konstrukcji sterowania i przypisanych im zdarzeń. Przeciwnie, może zostać zastosowany do reprezentacji dowolnie skomplikowanej aktywności i kontroli wymaganych przez system. Model CDFG znajduje zastosowanie w projektowaniu systemów czasu rzeczywistego. Na rysunku 2.20 przedstawiono przykład modelu opisanego grafem CDFG.

Przedstawiona reprezentacja specyfikacji zachowania (rysunek 2.20) składa się z dwóch części: sterującej, przetwarzania. W chwili uruchomienia systemu, maszyna FSM przebywa w stanie S0. Po nadejściu zdarzenie START, odblokowana zostanie aktywność zadań A1 i A2, kontroler przejdzie do stanu S1. Następnie, będąc w stanie S1, w chwili wystąpienia zdarzenia W=10, zdarzenie A1 zostanie zablokowane, a A3 odblokowane – system przechodzi do stanu S3. W końcowym etapie, aktywny stan sygnału STOP blokuje aktywność zdarzeń A2 i A3, system przechodzi do stanu S0. Zmienne X,W,Y,Z reprezentują przepływ danych pomiędzy poszczególnymi aktywnościami grafu DFG.



Rysunek 2.20 Przykład systemu sterowania opisanego grafem CDFG [GaVa94]

Główną zaletą modelu CDFG jest eliminacja wad grafów DFG i CFG (jednostronność w specyfikacji odpowiednio przepływu danych i sterowania). W konsekwencji, model CDFG jest uniwersalnym rozwiązaniem spełniającym wymagania wielu obszarów projektowania systemów cyfrowych, znajdując zastosowanie w projektowaniu systemów czasu rzeczywistego i syntezy behawioralnej ASIC.

2.3. Techniki opisu zachowania systemu[gore3]

Projektowanie systemu cyfrowego zaczyna się od opracowania jego abstrakcyjnego modelu, w formie specyfikacji programowej, służącego do reprezentacji założeń projektowych. Specyfikacja ta najczęściej jest formułowana przy zastosowaniu

języków opisu sprzętu (ang. HDLs – Hardware Description Languages), języków wyższego rzędu takich jak SystemC, SystemVerilog lub opisów formalnych jak UML lub sieci Petriego. ~~Wykorzystywanie języków opisu sprzętu, w procesie projektowania układów cyfrowych nie jest nową koncepcją. Od ponad dziesięciu lat stosuje się tego typu metodę projektową, wypierającą tradycyjne metody, jak choćby projektowanie przy użyciu schematów. Do najbardziej rozpowszechnionych języków należą VHDL[IEEE93] oraz Verilog[IEEE96].~~

2.3.1. *Język* VHDL

Język VHDL (ang. Very High Speed Integrated Circuits Hardware Description Language) [Ashe96,Smit96] jest językiem opisu sprzętu, początkowo przeznaczonym do dokumentowania oraz modelowania systemów cyfrowych, od prostych układów do złożonych systemów cyfrowych, na praktycznie każdym poziomie abstrakcji opisu (od poziomu bramek do poziomu systemu). Specyfikację języka VHDL opracowano w latach 1970-80 na potrzeby Ministerstwa Obrony USA. W roku 1987 powstał pierwszy standard przemysłowy opracowany przez IEEE (ang. Institute of Electrical and Electronics Engineers), który został zmodernizowany w roku 1993, w tej chwili znany jest pod nazwą IEEE Std 1076-1993
[ieee06]. Notowany w ostatnich latach wzrost popularności tego języka, wypływa z możliwości przeprowadzenia procesu syntezy logicznej opracowanych modeli, w celu redukcji poziomu opisu do poziomu listy połączeń. Opracowana lista połączeń (specyfikacja połączeń na poziomie bloków funkcjonalnych docelowego układu), może być użyta do konfiguracji architektury programowalnych układów typu FPGA (ang. Field Programmable Gate Array), CPLD (ang. Complex Programmable Logic Device) lub ASIC (ang. Application Specific Integrated Circuit). Istniejące narzędzia syntezy, dostępne na rynku nie są w stanie przeprowadzić syntezy wszystkich konstrukcji języka, jak również nie wszystkie konstrukcje mogą być zrealizowane sprzętowo. Tak, więc do modelowania syntezowalnych systemów cyfrowych [Rush98] używa się wydzielonego podzbioru języka VHDL, zawierającego konstrukcje możliwe do fizycznej realizacji układowej (zależne od wyboru narzędzia syntezy). Opis nie powinien być zbyt abstrakcyjny, tzn. powinien być zrealizowany na poziomie przesyłań międzyrejestrowych RTL lub na poziomie bramek, gdyż wyższy poziom opisu może utrudnić, a nawet uniemożliwić prawidłowa syntezę logiczną. Poziom RTL jest to trzeci poziom „piramidy abstrakcji", przedstawiającej poziom opisu sprzętu z wykorzystaniem języków opisu. Projektowanie na tym poziomie gwarantuje powstanie specyfikacji, w której użyte konstrukcje będą mogły być poprawnie zinterpretowane przez narzędzia syntezy logicznej wysokiego poziomu. Zbiór tych wydzielonych konstrukcji z całości syntaktyki języka VHDL określonej dokumentem [ieee06], określa się mianem syntezowalnego podzbioru języka VHDL, a ich standaryzacje zapewnia konsorcjum IEEE poprzez dokument [ieee06].

2.3.3. Język SystemVerilog

Język SystemVerilog [alte06,ieee06] jest rozszerzeniem standardu języka Verilog IEEE 1364-2001. SystemVerilog rozszerza możliwości języka Verilog w opisie na poziomie RTL oraz umożliwia opis modeli na poziomie systemu, wprowadza nowe

konstrukcje w celu dokładnej weryfikacji projektu (ang. assertion), oraz rozszerza możliwości języka Verilog w pisaniu bloków testowych (ang. test-benches). Ponadto SystemVerilog oferuje bogaty zbiór elementów typowych dla języków programowania (większość zaczerpnięta z języka C) np.: klasy, abstrakcyjne typy danych i typy danych definiowane przez użytkownika,, przekazywanie wartości zmiennym poprzez referencje, semafory itp. Jednym z głównych celów opracowania specyfikacji języka SystemVerilog było umożliwienie sprawnego modelowania rozbudowanych projektów. Dlatego też oprócz wprowadzenia możliwości modelowania na poziomie systemu, SystemVerilog rozbudowuje i rozszerza możliwości języka Verilog w modelowaniu na poziomie RTL. Do standardu wprowadzono wiele typów danych i konstrukcji zaczerpniętych z innych rozwiązań (VHDL, C, C++, SystemC), które umożliwiają swobodne modelowanie na poziomie systemu jak i sprawniejsze modelowanie na niższych poziomach abstrakcji.

Standard SystemVerilog przez długi okres czasu nie był wspierany jest przez dostępne na rynku narzędzia syntezy i implementacji w segmencie EDA (ang. Electronik Design Automation). Dopiero na przełomie roku 2006 znani producenci EDA (np.Synopsys, MentoGraphics) włączyli podzbiór standardu SystemVerilog jako opis wejściowej specyfikacji zachowania systemu.

2.4. Architektury i realizacje sprzętowe hybrydowych systemów cyfrowych[gore4]

Realizacja systemów cyfrowych, w szczególności systemów osadzonych, musi spełniać wymagania dotyczące rozmiarów fizycznych całkowitej konstrukcji, kosztów realizacji, poboru mocy oraz czasu realizacji [JeMe98,GaVa95]. Zazwyczaj systemy osadzone budowane są jako układy SoC lub SOPC, gdzie mikroprocesor otoczony układami peryferyjnymi stanowi centralną jednostkę sterowania i przetwarzania danych. Implementacja pełnej funkcjonalności specyfikowanego systemu jako oprogramowania dostarcza wielu korzyści, m.in. takich jak: prosta i szybka zmiany funkcjonowania systemu, prosty proces weryfikacji błędów, krótki etap projektowy, wielokrotne użycie tych samych funkcji programu. Ze względu na właściwości „programu" – komponenty programowe są proste do modyfikacji i adaptacji w kolejnych projektach. Główną wadą takich rozwiązań jest wydajność pracy – czas wykonania instrukcji programu – dla specyfikowanego systemu jako realizacji wyłącznie programowej. Ponowne rozważenia stawianych wymagań: koszt, rozmiar fizyczny, pobór mocy; skłaniają do realizacji pełnej specyfikacji systemu jako dedykowanego układu cyfrowego typu ASIC. Produkcja masowa eliminuje problem poniesionych kosztów, pobór mocy jest minimalny. Natomiast, cykl realizacji projektu ASIC w stosunku do rozwiązania programowego jest kilkukrotnie dłuższy, co wpływa niekorzystnie na termin dostarczenia produktu na rynek.

Proponowanych jest szereg rozwiązań, których celem jest jednoczesne spełnienie postawionych wymagań: koszt, czas realizacji, pobór mocy, rozmiar fizyczny.

W celu usystematyzowania zagadnień dotyczących akceleracji obliczeniowej systemów cyfrowych i komputerowych, w pracy [Stas05,BaLa02] klasyfikacji

poddano znane działy i technologie systemów, gdzie poprzez rozwiązania inżynieryjno-naukowe, dąży się do przyspieszenia operacji przetwarzania danych i sterowania.

2.4.1. Scalone systemy ogólnego przeznaczenia

Koprocesory o stałej architekturze (ogólnego przeznaczenia) odgrywają bardzo istotną rolę w przetwarzaniu danych większości systemów komputerowych, w tym również systemów osadzonych. Są one układami z rodziny ASIC o dużym nakładzie produkcyjnym, charakteryzującymi się niskimi kosztami produkcji oraz uniwersalnością zastosowań. Wadą uniwersalnych koprocesorów arytmetycznych w konkretnych zastosowaniach może być przystosowanie ich architektury do obliczeń o bardzo dużej złożoności. Efektem tego jest znaczny nadmiar elementów scalonych w strukturze koprocesora w stosunku do bieżących wymagań obliczeniowych. Sztywna architektura uniwersalnych koprocesorów powoduje, że średnie wykorzystanie ich możliwości wynosi niewiele ponad połowę [JaZb00], np.: 55% – w procesorze Pentium, 53%–w Pentium II czy zaledwie 44% w procesorze RISC AX3000 firmy IDT. Względnie niska efektywność koprocesora jest spowodowana koniecznością zapewnienia uniwersalności struktury, w której ścieżki obliczeniowe są projektowane i optymalizowane przy spełnieniu kompromisu dla różnych warunków. W sprzęcie powszechnego użytku taki spadek mocy obliczeniowej nie jest wyraźnie dostrzegalny, jak w przypadku obliczeń w zadaniach o bardzo dużej złożoności, dla których może on mieć znaczące znaczenie, a nawet decydować o ich wykonalności.

Wśród propozycji poprawy efektywności systemów obliczeniowych przy użyciu dodatkowego sprzętu są interesujące dwa sposoby realizacji akceleratora:

- wykonanie specjalizowanego koprocesora realizującego ściśle określone zadanie lub jego najbardziej czasochłonne fragmenty. Rozwiązanie to jest kosztowne i wymaga wprawnego łączenia umiejętności inżynierskich i teoretycznych;
- zaprojektowanie rekonfigurowalnego koprocesora, który zmienia swoją architekturę sprzętową stosownie do realizowanego zadania; zaletą tego rozwiązania jest możliwość wykorzystywania tej samej wersji koprocesora w różnych aplikacjach, przy czym właściwe rozłożenie algorytmu na podzadania pomiędzy sprzęt (koprocesor) i oprogramowanie (główny procesor systemu) pozwoli znacznie zwiększyć jego szybkość w porównaniu z koprocesorami uniwersalnymi o stałej architekturze.

Za wyborem drugiego rozwiązania przemawiają bardzo obiecujące wyniki badań i udane eksperymenty przeprowadzone w wielu ośrodkach akademickich oraz naukowych-badawczych przemysłu i wojska [MiOl98,RaNa00,LeCh03,BaLa02, Schu01].

2.4.2. Specjalizowane systemy osadzone

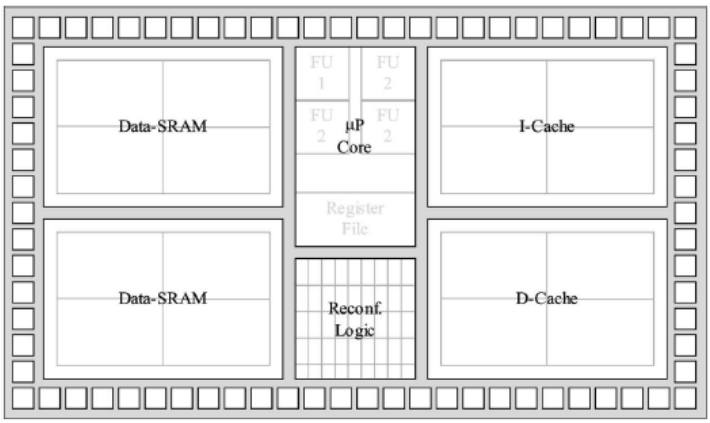
Dzisiejsze systemy osadzone składają są z wielu komponentów sprzętowych i programowych, które pozostają w ścisłej interakcji ze sobą. Określenie właściwego podziału (balansu) systemu pomiędzy obiema częściami determinuje sukces rozwiązania. Rozważając rozwiązanie programowe można łatwo zaobserwować, że komponenty programowe są prostsze do modyfikacji, niż ich odpowiedniki sprzętowe. Dzięki tej właściwości, programowe elementy systemu, przetwarzane

przez programowalne procesory, zapewniają prostotę i szybkość podczas eliminacji błędów, łatwą zmianę zastosowania danego programu, ponowne użycie funkcji lub procedur do budowy innych programów, lub w celu eliminacji czasu realizacji projektu. Jednak w porównaniu z komponentami sprzętowymi, elementy programowe są dużo wolniejsze i zużywają znacznie więcej energii podczas wykonywania tych samych funkcji. Rozwiązania wyłącznie sprzętowe używane są w systemach, gdzie czas reakcji i pobór mocy są parametrami krytycznymi. Niestety, czas projektowania takich systemów jest długi i drogi. Dodatkowo, procedury zaimplementowane z strukturach sprzętowych (ASIC) nie mogą być modyfikowane. Zadaniem projektanta systemu jest znalezienie właściwego balansu pomiędzy komponentami programowymi i sprzętowymi w celu zapewnienia wymagań pracy systemu. Interakcja pomiędzy elementami sprzętowymi i programowymi jest ścisła, szczególnie w systemach osadzonych. Procesory (ASIP – ang. Application Specific Instruction-Set Procesor) [LeCh03,DiHe03,BaLa02] o ograniczonej liście instrukcji oraz procesory ze zmienną/programowalną listą instrukcji (RISP – Reconfigurable Instruction-Set Procesor) [MiOl98,BaLa02,CaCh01] stanowią przykład konstrukcji, gdzie wymiana informacji pomiędzy częściami sprzętowymi i funkcjami programowymi – interakcja – jest bardzo ścisła. Procesory ogólnego przeznaczenia nie mają tak krytycznych zależności, co wynika z ich uniwersalnej budowy. Projektowanie systemów specjalizowanych, a w szczególności procesorów specjalizowanych (ASIP, RISP), wymaga metodologii, która uwzględnia wszelkie aspekty projektowania zaróno architektury części sprzętowej, jak i programu. Procesory ASIP można określić jako specjalizowane procesory o zdefiniowanej liście instrukcji. W procesie projektowym, na podstawie analizy programu, który będzie wykonywany przez ten procesor, ustala się listę instrukcji, jakie zostały użyte w kodzie programu. W procesorze ASIP implementowane są te bloki sprzętowe, które są niezbędne do wykonania wybranych instrukcji programu. W ten sposób budowany jest procesor zoptymalizowany pod kątem wydajności obliczeniowej o minimalnym poborze mocy. Wadą tego rozwiązania jest produkcja procesora tylko dla jednego programu i jego jednorazowe zastosowanie.

Bardziej uniwersalnym rozwiązaniem, z jednoczesnym zachowaniem wydajności i zminimalizowanym poborze mocy, są procesory typu RISP. Architektura RISP jest otwarta i podatna na procesy ewolucji pozwalając bez większego ryzyka w krótkim czasie osiągnąć gotowość rynkową.

2.4.3. Procesor reprogramowalny RISP

Procesor o rekonfigurowanym zbiorze instrukcji składa się z rdzenia mikroprocesora ogólnego przeznaczeni wzbogacony o logikę rekonfigurowaną.Jego budowa i funkcjonowanie jest zbliżona do procesorów ASIP z tą różnicą, że z miejsca specjalizowanych jednostek wykonawczych wprowadzono rekonfigurowalne bloki funkcyjne. Rekonfigurowane bloki funkcyjne zapewniają adaptację procesora do szerokiej gamy zastosowań, natomiast rdzeń procesora uniwersalność oprogramowania. Rysunek 2.27. przedstawia ogólny podgląd architektury RISP.



Rysunek 2.27 Ogólna architektura procesora RISP [BaLa02]

RISP wykonuje instrukcje tak samo jak każdy procesor ASIP, z tą różnicą, że zbiór instrukcji RISP podzielony jest na dwa podzbiory: 1) zdefiniowany zbiór instrukcji, który został zaimplementowany w rdzeniu procesora uniwersalnego, 2) zbiór instrukcji programowalnych, który realizowany przez rekonfigurację logiki programowalnej. Zbiór instrukcji rekonfigurowanych jest adekwatny do zbioru specjalizowanych instrukcji procesora ASIP, jednak z możliwością modyfikacji po procesie produkcyjnym.

Zagadnienia dotyczące procesu projektowego części sprzętowej RISP

Projektując część sprzętową procesora RISP, rozpatruje się dwa zagadnienia. Pierwsze dotyczy sposobu połączenia procesora i logiki rekonfigurowalnej. W szczególności sposobu transmisji danych z i do rekonfigurowalnego bloku zadaniowego oraz synchronizacji obu jednostek. Drugim zagadnieniem jest sam proces projektowania rekonfigurowalnego bloku zadaniowego, który obejmuje zakresem rozważania dotyczące ziarnistości, sposobu rekonfiguracji, połączeń wewnętrznych i zewnętrznych, oraz inne. W pracy [BaLa02] opracowano tabelę charakteryzującą wybrane procesory typu RISP oraz cechy charakterystyczne, dotyczące ich konstrukcji oraz metod projektowania.~~Do konstrukcji najbardziej interesujących ze względu na swą uniwersalność aplikacji, a w szczególności ze względu na proces projektowy, należą układy typu ASC (ang. Application Specific Co-processors). W procesie projektowym, nazywanym dekompozycją systemową (ang. Hardware/Software Codesign) program, który ma zostać uruchomiony na docelowym procesorze lub specyfikacja kontroli systemu cyfrowego dedykowana dla wybranego procesora, poddawane są analizom formalnym i symulacyjnym. W rezultacie, program lub specyfikacja dzielone są na dwie części: sprzętową i programową. Część sprzętowa realizuje zbiór zadań, które stanowią największe obciążenie operacyjne dla procesora. Natomiast procesor przetwarza rzadko wykonywane proste instrukcje sterowania i przetwarzania. Wynikiem procesu dekompozycji funkcjonalnej jest konfigurowalny koprocesor sprzętowy, który wspomaga procesor pracujący sekwencyjnie. W oparciu o założenia projektowania zintegrowanego opracowano wiele algorytmów dekompozycji funkcjonalnej, między innymi: Cosyma[12], Vulcan, Polis[13], Camad, Chinook, Tosca, SpecSyn. Domeną wymienionych rozwiązań są architektury o sztywnym/stałym szkielecie architektury systemu po zakończeniu procesu projektowego (czyt. produkcyjnego),~~

są to rozwiązania typu SoB (ang. System on a Board) [6] oraz SoC (ang. System on a Chip). Do grupy specjalizowanych systemów osadzonych zaliczamy również rozwiązania klasy SOPC, które w coraz szerszej skali wypierają architektury SoB i SoC w implementacjach systemów osadzonych.

Badania przeprowadzone w rozprawie obajmują opracowaną architekturę sprzętowo-programowego mikrosystemu cyfrowego dla rozwiązań SOPC oraz wybrane aspekty syntezy systemowej zaprzeniete w procesie projektowym.

Sposoby połączenia procesora z dedykowaną sprzętowo jednostką funkcjonalną

Procesory o rekonfigurowanym zbiorze instrukcji składają się z logiki reprogramowalnej zintegrowanej z procesorem. Pozycja relatywna logiki programowalnej w stosunku do procesora bezpośrednio wpływa na wydajność całego systemu oraz rodzaju oprogramowania wykorzystującego część programowalną. Korzyści czerpane z wykonywnia części kodu programu z wykorzystaniem logiki reprogramowalnej zależą od dwóch aspektów: czasu komunikacji, czasu wykonywania instrukcji. Czas niezbędny do wykonania instrukcji T*s* jest sumą czasu transferu danych z T*c* i do T*p* procesora oraz czasu przetwarzania operacji T*r*. Jeśli łączny czas Ts jest mniejszy niż czas niezbędny do wykonania danej instrukcji przez procesor ogólnego przeznaczenia, wówczas osiągany jest zysk – zwiększona wydajność pracy procesora.

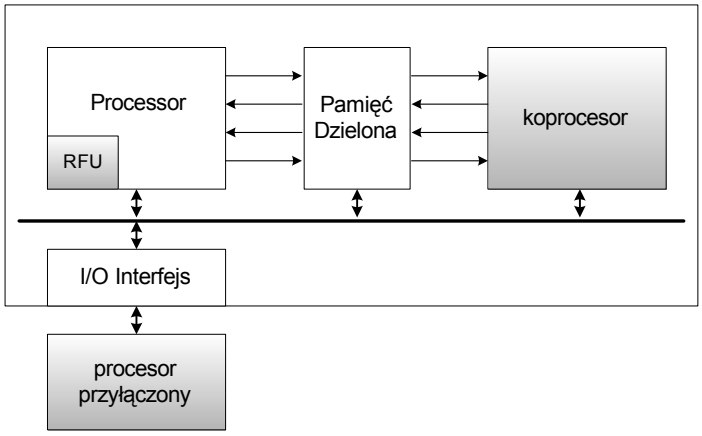
Charakteryzują się one tym, że zawierają rekonfigurowany blok funkcjonalny RFU (ang. Reconfigurable Functional Unit), który spełnia rolę akceleratora podczas przetwarzania wybranej instrukcji. Blok RFU może zostać zintegrowany z główną jednostką przetwarzania programu (zazwyczaj procesor RISC) na trzy sposoby, rysunek 2.28:

- Procesor przyłączony. Logika rekonfigurowana połączona jest z procesorem za pośrednictwem magistrali IO (np.PCI), przykład procesor PRISM [AtSi93].
- Koprocesor. Część sprzętowa umiejscowiona jest w pobliżu (w systemie osadzonym) procesora, komunikacja jest realizowana za pośrednictwem standardowego protokołu, przykład procesor Garp [HaWa97].
- Rekonfigurowana jednostka funkcjonalna RFU (ang. Reconfigurable Funcional Unit). Logika rekonfigurowana umiejscowiona jest wewnątrz procesora ogólnego przeznaczenia, dekoder instrukcji traktuje instrukcje logiki rekonfigurowanej jako standardową instrukcję bloku zadaniowego procesora, osiągana jest pełna integralność rozwiązania, przykład OneChip[CaCh01], PRISC[RaBr94].

W dwu pierwszych schematach rozwiązań połączeń (połączenia luźne) straty ponoszone przez długi cykl komunikacji rekompensowane są sprzętową akceleracją przetwarzania. Na przykład, koszt komunikacji w systemie PRISM wynosi 50 cykli zegarowych. W aplikacjach strumieniowych, gdzie przetwarzany jest ciągły strumień danych, koszt komunikacji może zostać „ukryty" poprzez potokową realizację transferu i przetwarzania danych.

Natomiast, schemat trzeci dotyczy procesorów RISP, gdzie koszty komunikacji pomiędzy procesor i logika programowalna praktycznie nie występują. W związku z czym, łatwiej osiąga się większą wydajność pracy w szerokim zakresie aplikacyjnym. Niestety, koszt produkcji jest duży ze względu na brak możliwości

wykorzystania standardowych komponentów. W jednym układzie scalonym ASIC musi zostać zaimplementowany procesor ogólnego przeznaczenia oraz przestrzeń logiki programowalnej. Ponadto, rozmiar logiki programowalnej w procesorze RISP jest ograniczony ze względu na rozmiar układu ASIC, co ma bezpośredni wpływ na możliwość osiąganych wyników (prędkość pracy).



Rysunek 2.28 Sposoby podłączenia części sprzętowej do procesora w rozwiązaniach RISP [BaLa02]

Typy instrukcji wykonywanych przez RISP

Typ połączenia RFU do procesora [BaLa02] zależny jest od charakterystyki typów instrukcji dla jakich procesor jest projektowany. Z wykorzystaniem RFU mogą być zaimplementowane dwa typy instrukcji:

- Instrukcje strumieniowe. Przetwarzania dużej ilości danych w sekwencji blokowej. Tylko mały zbiór aplikacji może wykorzystać taką architekturę, na przykład: FIR, DCT.
- Instrukcje użytkownika. Tego typu instrukcje pobierają małą porcję danych w jednym czasie (zazwyczaj bezpośrednio z rejestrów procesora) i produkują adekwatną porcję danych wyjściowych. Instrukcje mogą być użyte przez szeroką gamę aplikacji, jednak należy spodziewać się mniejszego przyspieszenie w porównaniu z instrukcjami strumieniowymi.

Typ implementowanej instrukcji determinuje obszar aplikacyjny procesora RISP. Słowo instrukcji również specyfikuje operandy przekazywane do jednostki RFU. Operandami mogą być wartości zmiennych, adresy, rejestry. Mogą reprezentować źródło lub wynik operacji. Wyróżnia się kilka typów kodowanie operandów:

- *Hardwired*. Zawartość wszystkich rejestrów przesyłana jest do RFU. Które rejestry zostaną wykorzystane przez RFU zależy od zaprogramowanej logiki. RFU ma dostęp w ten sposób do wszystkich rejestrów procesora, lecz generacja kodu operacji jest znacząco skomplikowana. Takie rozwiązanie stosowane jest w systemie Chimaera, gdzie osiem rejestrów procesora ogólnego przeznaczenia jest dostępnych przez RFU jednocześnie.
- *Fixed*. Operandy znajdują się w określonym przedziale słowa instrukcji oraz posiadają określony typ. Jest to najczęściej stosowany typ kodowania operandów.

- *Flexible*. Pozycja operandów jest konfigurowalna. Stopień konfigurowalności jest bardzo ogólny. Jeśli wykorzystywana jest tablica konfiguracji instrukcji, może być ona zastosowana do dekodowania położenie operandów.

Projektowanie logiki RFU

Projektowanie logiki konfiguracyjnej determinuje między innymi liczbę instrukcji, jaką będzie przechowywał procesor, rozmiar strumienia programującego, czas rekonfiguracji, typ instrukcji dostarczających największych zysków wydajności.

Kontroler konfiguracji RFU

Czas rekonfiguracji nie zależy tylko od ilości danych konfiguracyjnych, ale również od zastosowanych metod. W procesorze PRISC [RBS94] blok RFU konfigurowany jest przez bezpośrednie kopiowanie danych konfiguracyjnych do pamięci warstwy konfiguracji (używając standardowej operacji programowania). Jeśli to samo zadanie zostanie przeprowadzone przez dedykowany kontroler konfiguracyjny w trakcie, gdy procesor wykonuje instrukcje programu, wówczas osiągnięte zostaje przyspieszenie. Poprzez dodatkową modyfikację interfejsu programowania z komunikacji bitowej na równoległą, możne osiągnąć kolejne zyski czasu pracy procesora.

Narzędzie programistyczne procesorów RISP

Główną różnicą pomiędzy RISP a procesorami ogólnego przeznaczenia jest zmiana zbioru instrukcji procesora RISP w trakcie jego pracy, a nie na etapie projektowym. Ten fakt zmienia znaczenie sposobu generacji kodu dla RISP. Generacja kodu procesora RISP dotyczy technik inżynierii oprogramowania i projektowania sprzętu. Ten proces jest podobny do zagadnień sprzętowo-programowego projektowania zintegrowanego, gdzie aplikacja dzielona jest na część programową i sprzętową. W rozwiązaniach RISP, cała aplikacja wykonywana jest programowo ze wspomaganiem sprzętowym w realizacji wybranych instrukcji programowych aplikacji. Niezbędne jest również zaprojektowanie części sprzętowej według specyfikacji funkcjonalnej wybranej instrukcji.

2.5. Platforma SOPC

Obecnie najbardziej wyróżniającymi się układami FPGA są rozwiązania firmy Xilinx [Xili06]. Rodzina układów Virtex4 oferuje trzy platformy (kilkanaście układów FPGA) ukierunkowane na spełnienie wymagań stawianych przez zróżnicowane domeny aplikacyjne systemów osadzonych:

- FX; wysoka wydajność logiki, układy zorientowane na implementację osadzonych systemów cyfrowych niewymagających złożonych obliczeń matematycznych, możliwa implementacja procesorów typu SOFT CORE,
- SX; wysoka wydajność przetwarzania sygnału; układy dedykowane dla systemów przetwarzania sygnału DSP, ponad 512 komórek XtremeDSP pracujących z częstotliwością 500MHz,
- FX; przetwarzania osadzone; zintegrowanych do dwóch procesorów HARD CORE (PowerPC405) w jednym układzie FPGA.

Układy z rodziny Virtex4 charakteryzują się następującymi parametrami technicznymi: a) 200.000 konfigurowalnych komórek logicznych, b) maksymalna częstotliwość taktowania wynosi 500MHz, c) o połowę obniżono pobór mocy w porównaniu z wcześniejszymi generacjami FPGA, d) technologia wykonania 90nm.

Układy Virtex4 rozszerzają oraz otwierają nowe perspektywy projektowania, implementacji i realizacji osadzonych systemów cyfrowych. Alternatywą są realizacji systemów SOPC są układy z rodziny Spartan2/Spartan3 lub Virtex/Virtex2 oferujące dużo niższe koszty zakupu oraz optymalne wykorzystanie mocy w realizowanym zadaniu.

Konkurencyjnymi rozwiązaniami są układy Stratix2 i Excalibur firmy Altera [alte06]. Parametry techniczne są zbliżone do modeli firmy Xilinx. Wybór często podyktowany jest ceną układu, obsługą techniczną oraz szczegółowymi parametrami prądowymi, takimi jak: temperatura, opakowanie w sensie wyprowadzeń portów wejścia/wyjścia, typ pamięci. Ponad to Altera dostarcza technologię migracji projektu wykonanego dla układu FPGA do układu ASIC (HardCopy) bez konieczności dodatkowego cyklu produkcyjnego gwarantując powodzenie realizacji końcowej.

Alternatywne rozwiązania dostarcza firma Actel [acte06]. Układy Actel ProASIC3 posiadają pojemność od 30k do 3 milionów bramek systemowych w jednym układzie FPGA (w tym ponad 75k przerzutników do wykorzystania). Cechą wyróżniającą układy FPGA ProASIC3 spośród rozwiązań Xilinx i Altera jest warstwa programująca logikę układu. Firma Actel wykorzystuje do tego celu pamięci Flash [acte06], w przeciwieństwie do innych producentów FPGA bazujących na pamięciach SRAM (w chwili zaniku zasilania układu tracona jest konfiguracja FPGA). Ponadto jako jedyny producent FPGA dostarcza programowalne cyfrowo-analogowe układy cyfrowe Actel Fusion™ z możliwością programowalnia części cyfrowej i analogowej.

Pozostali producenci układów programowalnych: Atmel [atme06], Lucent [Luce06], Cypress [Cypr06], QuickLogic [Quic06]; nie posiadają w ofercie układów FPGA mogących w pełni konkurować z rozwiązaniami Xilinx lub Altera lub stopniowo wycofują się z branży FPGA.

2.6. Wybrane narzędzia CAD wspierające projektowanie mikrosystemów cyfrowych z uwzględnieniem sprzętowego koprocesora

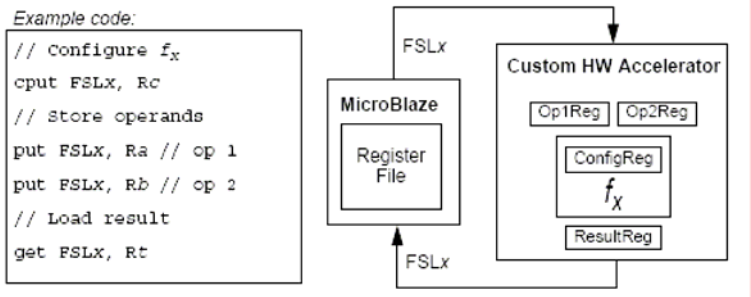
Adoptowanie nowych rozwiązań technologicznych przez sferę przemysłu wymaga czasu. Potwierdzenie wyników przez ośrodki naukowe oraz opracowanie właściwej metodologii projektowej musi zostać zaakceptowane nie tylko przez środowisku naukowe. Nadanie standardu dla danej technologii przez konsorcja międzynarodowe ma bezprecedensowe znaczenie dla masowego zastosowania metodologii w rozwiązaniach komercyjnych. Projektowanie zintegrowane, będące przedmiotem badań już przez dłuższy okres czasu, w dalszym ciągu nie znalazło implementacji w systemach EDA na szeroką skale. Nowoczesne narzędzie CAD, do których zaliczamy między innymi: Xilinx EDK [Xili06], Altera Quartus SOPC Builder

[Alte06], CoWare ConvergenSC [Cowa06], Celoxica [Celo06]; wspomagają proces projektowania zintegrowanego na poziomie systemowym ograniczając się jednak tylko do metod wspomagania procesu projektowego typu heterogenicznego. Wspierają proces kosymulacji sprzętowo-programowej oraz automatyczną generację interfejsu komunikacyjnego pomiędzy programem i modułami sprzętowymi. Brak natomiast zaawansowanych rozwiązań umożliwiających, już na etapie projektowym, kwalifikację oraz alokację dowolnych bloków/zadań funkcjonalnych systemu do zasobów sprzętowych lub programowych.

2.6.1. Rozwiązania Xilinx EDK

Platforma projektowa firmy Xilinx XPS (ang Xilinx Platform Studio) stanowi kompletne rozwiązanie wspomagające proces projektowania systemów osadzonych bazujących na implementacji dwóch procesorów: MicroBlaze (FPGA: Virtex2, Spartan2, Spartan3), PowerPC (FPGA: Virtex2Pro, Virtex4). Zintegrowane środowisko programistyczne zawiera szeroką gamę narzędzi projektowania osadzonego [Xili06b], komponenty IP CORE, biblioteki, oraz zbiór generatorów ułatwiających tworzenie platformy systemu osadzonego użytkownika.

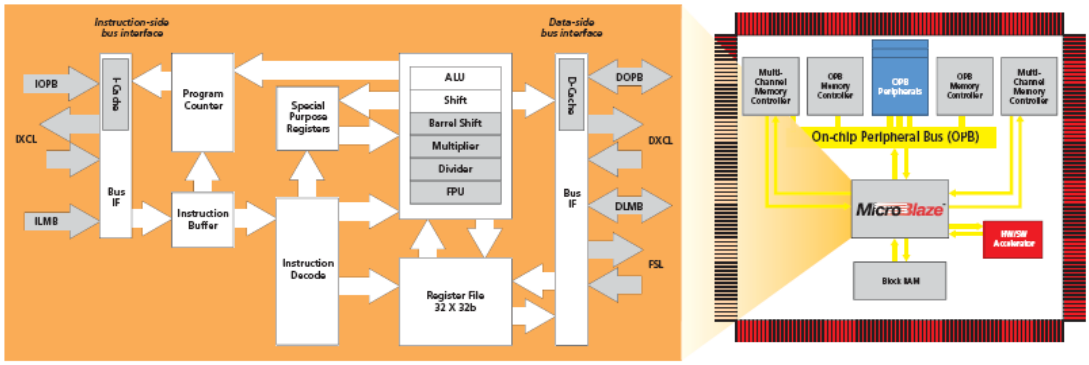
Z punktu prowadzonej rozprawy, najbardziej interesującym narzędziem jest *Configure Coprocessor Wizard*, który odpowiedzialny jest za dołączanie koprocesora sprzętowego do procesora. W rozwiązaniach firmy Xilinx, koprocesor traktowany jest jako moduł sprzętowy implementujący funkcje użytkownika z wykorzystaniem logiki FPGA. Połączenie do procesora realizowane jest poprzez interfejs FSL (ang.Fast Simplex Link) dla procesora „miękkiego" MicroBLAZE, a dla procesora twardego PowerPC poprzez kontroler AUP (ang. Auxiliary Processing Unit) [Xili06a], które realizują komunikacje za pośrednictwem bloku FIFO. Dla procesora MicroBlaze dostępnych jest osiem kanałów komunikacyjnych FSL. W związku z czym, możliwe jest podłączenie do ośmiu zewnętrznych koprocesorów, oznacza to że dopuszczalna jest implementacja maksymalnie ośmiu sprzętowo wspomaganych instrukcji programowych.



Rysunek 2.33 Wykonanie instrukcji sprzętowej w rozwiązaniach firmy Xilinx [xili06]

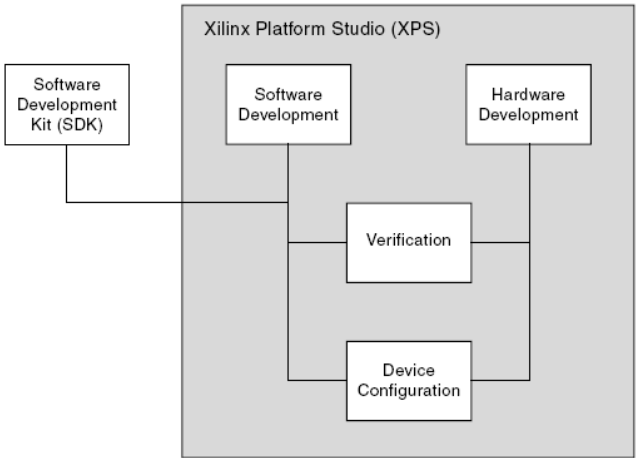
Kanał FSL ma szerokość 32-bitów, zgodny z architekturą procesora. Dodatkowy bit nr 33 identyfikuje słowo odczytu/zapisu względem jego typu: dane, kontrola. Komunikacja procesor ↔ sprzęt odbywa się poprzez wywołanie komend programu: get, put (rysunek 2.33). Instrukcja *get* jest użyta do transferu informacji z koprocesora FSLx do rejestru ogólnego przeznaczenia. Natomiast instrukcja put przesyła dane do wybranego kanału FSL. Rysunek 2.34 przedstawia sposób podłączenia koprocesora do procesora MicroBlaze.

Kanał FSL zapewnia bezpośrednią komunikację procesor ↔ koprocesor. Natomiast magistrala OPB (ang. On chip Peripherial Bus) implementowana w układzie FPGA jako komponent IP CORE zapewnia interfejs pomiędzy procesorem i pozostałymi składowymi systemu.



Rysunek 2.34 Podłączenie akceleratora sprzętowego do mikroprocesora MicoBlaze w rozwiązaniu firmy Xilinx [xili06]

Środowisko firmy Xilinx wspomaga proces projektowy systemu osadzonego w zakresie przedstawionym na rysunku 2.35.



Rysunek 2.35 Schemat środowiska projektowego Xilinx EDK [xili06]

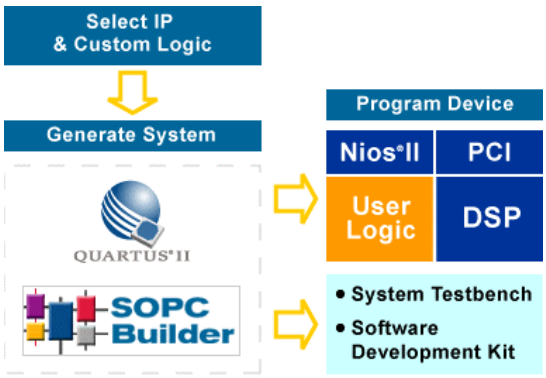
Realizacja projektu sprzętowo-programowego systemu osadzonego przebiega według schematu projektowania heterogenicznego. Wyróżnia się pięć etapów projektowych:

- tworzenie platformy sprzętowej (Hardware Development); konfiguracja procesora, tworzenie połączeń komunikacyjnych pomiędzy sprzętowymi blokami peryferyjnymi użytkownika z procesorem,
- tworzenie platformy programowej (Software Development); kolekcje sterowników programowych obsługiwanych urządzeń systemu oraz zbiór dedykowanych systemów operacyjnych gotowych do budowania aplikacji użytkownika,
- weryfikacja platformy sprzętowej (Verification); w celu weryfikacji funkcjonalnej opracowanej platformy sprzętowej, generowany jest model symulacyjny systemu oraz uruchamiany jest proces symulacji w wybranym symulatorze HDL, w trakcie symulacji procesor wykonuje swój aktualny program,

- weryfikacja programowa (Verification); podstawową techniką weryfikacji oprogramowania proponowaną przez Xilinx EDK jest uruchomienie procesora w docelowym systemie osadzonym i użycie dedykowanych narzędzi kontroli wykonywania programu, alternatywnym rozwiązaniem jest zastosowanie wirtualnego modelu symulacyjnego,
- programowanie układu (Device Configuration); po zakończeniu cyklu projektowego i weryfikacji tworzony jest binarny plik programujący układ FPGA.

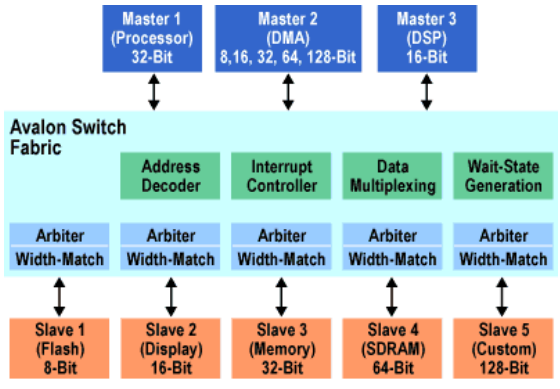
2.6.2. *Rozwiązania Altera-SOPC*

Oprogramowanie i rozwiązania Altera SOPC Builder są częścią pakietu Altera Quartus2 [Alte06], rysunek 2.36. Narzędzia SOPC Builder eliminują zadania dotyczące manualnej integracji systemu automatyzując proces łączenia komponentów IP CORE, urządzeń peryferyjnych; a przez to umożliwiając projektantowi przeniesienie całego wysiłku na realizację procesu projektowego systemu.



Rysunek 2.36 Schemat środowiska projektowego Quartus2 SOPC Builder firmy Altera [alte06]

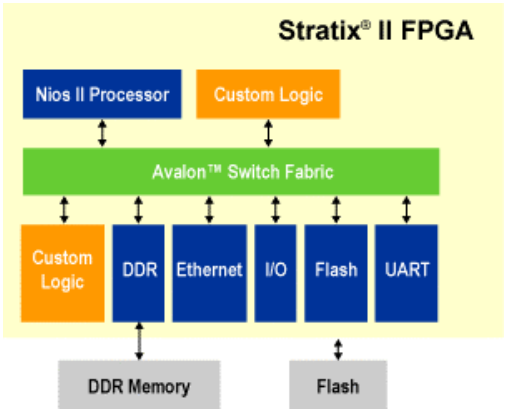
Proces projektowy w środowisku Altera SOPC przebiega analogicznie jak w Xilinx EDK. Środowisko zintegrowanego projektowania dostarcza narzędzi niezbędnych do modelowania komponentów sprzętowych, specyfikowania programu dla procesora oraz wspomaganie w procesie konfiguracji i alokacji interfejsu wewnętrznego i zewnętrznego systemu. Natomiast proces projektowy dotyczący kodowania funkcjonalności komponentów sprzętowych i programowych, wyznaczanie funkcji dla koprocesora sprzętowego; realizowany jest manualnie przez inżyniera.



Rysunek 2.37 Magistrala Altera Avalon dedykowana do realizacji komunikacji w mikrosystemie cyfrowym [alte06]

Dla systemu SOPC Builder opracowano dedykowaną strukturę przełączania o nazwie Avalon, która odpowiedzialna jest za dostarczenie wspólnego zbioru sygnałów danych i sterowania wszystkim urządzeniom systemu oraz zarządzanie procesem komunikacji (przełączanie źródeł danych, dekodowanie adresu, generowanie stanu oczekiwania, ustanawianie priorytetów przerwań), rysunek 2.37.

Ponadto, magistrala Avalon jest elementem bibliotecznym IP CORE. Niezbędna jest alokacja dodatkowych zasobów sprzętowych układu FPGA w celu implementacji jej funkcjonalności. Dodatkowo, do systemu Avalon przyłączane są wszystkie urządzenia projektowanego systemu, włączając koprocesory sprzętowe (komponenty sprzętowe użytkownika). Rysunek 2.38 prezentuje przykład realizacji systemu osadzonego z wykorzystaniem procesora NIOS.II



Rysunek 2.38 Przykład połączenia układu cyfrowego wspomagającego pracę mikroprocesora w rozwiązaniach Altera [alte06]

W rozwiązaniach firmy Altera, akceleracja sprzętowa programu wykonywanego przez procesor dotyczy tylko i wyłącznie połączenia interfejsu wejścia/wyjścia procesora i komponentu sprzętowego IP CORE realizującego wybrane zadania. Zastosowanie ogólnego interfejsu komunikacyjnego Avalon w realizacji połączenia procesor ↔ koprocesor, może generować znaczne opóźnienia we wzajemnej komunikacji (wymiany danych), np. w czasie natężonej aktywności składowych systemu w dostępie do głównej jednostki sterownia i przetwarzania CPU systemu.

2.7. Weryfikacja i kontrola pracy systemu cyfrowego w wykorzystaniem interfejsu JTAG

Najpopularniejszy obecnie interfejs wykorzystywany do testowania i programowania (konfigurowania) układów PLD i ASIC jest znany pod akronimem JTAG
[ieee06,Texa06], powstał w końcu lat '80. Prace prowadzone przez Joint Test Action Group miały pierwotnie na celu opracowanie systemu umożliwiającego testowanie złożonych modułów cyfrowych po ich zmontowaniu na płytkach drukowanych. Do tego celu opracowano specjalizowane układy logiczne interfejsów magistralowych, umożliwiających monitorowanie większości sygnałów

w module (na przykład układy logiczne skanowania brzegowego – Boundary Scan Logic – firmy Texas Instruments). Dzięki temu możliwe stało się testowanie nie tylko pojedynczych struktur półprzewodnikowych, lecz także wzajemnych połączeń pomiędzy układami oraz połączeń pomiędzy układami i elementami stanowiącymi ich otoczenie. Twórcy interfejsu JTAG założyli, że nie ma potrzeby szczegółowego testowania wewnętrznych fragmentów układów, o których poprawną pracę powinien zadbać projektant na etapie projektowania struktury logicznej. Testowanie funkcjonalne, z małymi wyjątkami, ograniczono do weryfikacji stanów logicznych w komórkach wejściowych i wyjściowych testowanych układów. Stąd właśnie BST, skrótowa nazwa najważniejszej cechy i funkcji interfejsu JTAG, która jest akronimem od Boundary Scan Testing, co należy rozumieć jako testowanie brzegowe. Duża elastyczność i łatwość stosowania interfejsu JTAG, możliwość łatwego, praktycznie nieograniczonego zwiększania jego funkcjonalności i powszechne uznanie, jakim cieszył się na rynku spowodowały, że komitet normalizacyjny IEEE przyjął w 1990 roku normę IEEE 1149.1, w której zdefiniowano jego strukturę i sposób sterowania. Wprowadzona w 1993 roku nowelizacja normy miała na celu stworzenie języka opisu urządzeń i układów cyfrowych wyposażonych w interfejs JTAG. Nosi on nazwę BSDL (ang. Boundary Scan Description Language) [Park06].

Do komercyjnych rozwiązań wspierających proces weryfikacji funkcjonalnej i kontroli pracy systemu cyfrowego pracującego wewnątrz układu FPGA, można zaliczyć oprogramowania np. Xilinx ChipScopePRO™ [xili06] lub Synplicity Identify [synp06]. Jednak cena zakupu licencji na jedno stanowisko projektowe jest wysoka, szczególnie dla małych i średnich firm. Alternatywą są prace prowadzone przez społeczność akademicką (np. [BeKu06]), w tym opracowania niniejszej rozprawy, tj. oprogramowanie JTAG-SIM omówione w dodatku piątym.

ROZDZIAŁ TRZECI

3. Sprzętowo-programowa mikrostruktura cyfrowa SPMC

W rozdziale trzecim przedstawiono wybrane, charakterystyczne cechy mikroprocesora wbudowanego w struktury mikrosystemu cyfrowego, wskazując na problemy opisane w rozdziale pierwszym. Sformułowano wymagania stawiane uniwersalnej, pod względem konstrukcyjnym i funkcjonalnym, architekturze akceleratora sprzętowego. Przedstawiono własności projektowania zintegrowanych mikrosystemów cyfrowych ze względu na wykorzystanie reprogramowalnej logiki układu FPGA. Zdefiniowano nową architekturę sprzętowo-programowej mikrostruktury cyfrowej, której zadaniem jest skrócenie czasu przetwarzania i sterowania programu wykonywanego przez główną jednostkę kontroli mikrosystemu cyfrowego. Szczegółowo omówiono budowę, interfejs i protokół komunikacyjny nowej mikrostruktury cyfrowej oraz zdefiniowano koszty czasu komunikacji wewnętrznej program↔sprzęt.

3.1. Charakterystyka pracy mikroprocesora w mikrosystemach cyfrowych

W zintegrowanych mikrosystemach cyfrowych, główna jednostka CPU (mikroprocesor) nie pełni tylko i wyłącznie roli kontrolera systemu. Zazwyczaj jest to jednostka o ograniczonej liście rozkazów RISC (ang. Reduced Instruction Set Computers) realizująca zadania ogólnego przeznaczenia. Ponadto, programy sterowania w rozbudowanych mikrosystemach cyfrowych zawierają również ścieżkę danych, czyli realizację instrukcji algebraicznych i logicznych. Do realizacji programowych zazwyczaj kwalifikowane są operacje arytmetyczne lub dedykowane procedury programowe, które nie mają swojego odpowiednika w implementacjach sprzętowych IP lub ich implementacja w strukturach FPGA jest nie opłacalna za względu na koszty. Pozostałe operacje, w tym wybrane zadania przetwarzania danych oraz instrukcje sterowania, mogą byc realizowane w części sprzętowej.

Wybór mikroprocesora oraz architektury sprzętowej mikrosystemu cyfrowego w procesie projektowania zintegrowanego, wiąże się bezpośrednio z wymaganiami budowanego mikrosystemu cyfrowego. Jednym z parametrów krytycznych w projektowaniu systemowym (zarówno dla systemów bezprzewodowych jak i rozwiązań o charakterze jednostki stacjonarnej) jest pobór mocy, który determinuje czas pracy urządzenia bez stałego dostępu do źródła zasilania oraz wywiera bezpośredni wpływ na emisję ciepła urządzenia. Podstawowymi dwoma klasami procesorów ogólnego przeznaczenia wykorzystywanymi w projektowaniu systemów osadzonych, są rozwiązania CISC (ang. Complex Instruction Set Computer) oraz RISC. Ze względu na postulat zużycie energii, który wiąże się bezpośrednio z budową mikroprocesora, zazwyczaj wykorzystuje się rozwiązania typu RISC. Możliwe są dwa typy implementacji tego samego procesora w architekturze SOPC. Wyróżnia się tak zwane „rdzenie miękkie" (ang. softcore) i „rdzenie twarde" (ang. hardcore) [JeMe99,xili06,alte06].

3.1.1. Rdzenie miękkie

Pod pojęciem rdzenia miękkiego *softcore* kryje się jednostka IP CORE modelu procesora specyfikowanego w wybranym języku opisu sprzętu HDL, który został opracowany dla wybranego układu FPGA. Zależność od rodziny lub producenta układu FPGA wynika przed wszystkim z różnych technologii wykonania struktur programowalnych FPGA. Firma Xilinx stosuje w swoich układach konfigurowalne bloki synchroniczne – przerzutniki. Dzieki temu, w naturalny sposób możliwa jest implementacja rejestrów lub komponentów synchronicznych zbudowanych z przerzutników aktywnych na zbocze (typu flip-flop) lub przerzutników typu zatrzask. Natomiast, firma Altera zamieszcza w swoich układach FPGA tylko przerzutniki typu flip-flop, brak przerzutników aktywnych na poziom logiczny sygnału strobującego. W rezultacie, wymagana jest emulacja funkcjonalności przerzutnika LATCH podczas realizacji układowej w strukturach FPGA firmy Altera. Stąd wynika np. konieczność modyfikacji modelu IPCORE mikroprocesora dla implementacji w układach firmy Xilinx i Altera.

W procesie projektowym SOPC procesor o rdzeniu miękkim implementowany jest w układzie FPGA z wykorzystaniem standardowych narzędzi CAD. Częstotliwość pracy wbudowanego mikroprocesora zależy od technologii układu FPGA oraz optymalizacji procesu implementacji pod względem obszaru i waha się w przedziale 10MHz – 200MHz (dla różnych mikroprocesorów i układów FPGA).

3.1.2. Rdzenie twarde

Rdzenie twarde stanowią nową technologię projektowania układów cyfrowych typu FPGA. W jednym układzie scalonym rdzeń mikroprocesora o strukturze ASIC zostaje zintegrowany z typową przestrzenią logiki programowalnej FPGA. Są to konstrukcje stałe, zamknięte. Nie można zmieniać architektury wbudowanego mikroprocesora typu *hardcore*. Przykładem są układy Xilinx Virtex2Pro posiadające zintegrowanych od 1 do 4 mikroprocesorów IBM PowerPC 405, o częstotliwości zegara systemowego 400MHz.

3.2. Wymagania stawiane uniwersalnej architekturze mikrosystemu cyfrowego

Cechą charakterystyczną głównej jednostki sterowania i przetwarzania (CPU), wynikającą z jej budowy, jest sekwencyjne wykonywanie instrukcje programu. Przy dużych obciążeniach systemu (systemy reaktywne, systemy bezprzewodowe, telekomunikacja), sumaryczna wydajność pracy mikrosystemu może spaść do poziomu krytycznego (poniżej minimum) z powodu długiego cyklu pracy mikroprocesora, rysunek 1.3.

W domenie akceleracji przetwarzania programu przez procesor znajdują się między innymi rozwiązania szeroko omówione w rozdziale drugim. Niestety procesory RISP i ASIP posiadają szereg cech dyskwalifikujących ich adaptację do systemów SOPC, są to m.in.:

- Ingerencja w architekturę mikroprocesora. Szereg realizacji mikrosystemu cyfrowego w układzie SOPC wykorzystuje procesory RISC w postaci komponentów IP CORE. Natomiast, brak jest metod adaptacji procesorów RISC do architektury RISP lub wręcz brak możliwości adaptacji szerokiego spektrum procesorów RISC do architektury RISP ze względu na indywidualne rozwiązania implementacyjne.
- Metodologia RISP zorientowana jest na sekwencyjną interakcję części programowej i sprzętowej. Zarówno procesory RISP oraz ASIP są w pełni sekwencyjne. Instrukcje sprzętowe wywoływane są przez procesor ogólnego przeznaczenia w sekwencji instrukcji programowych (lub sprzętowych) i wykonywane przez dedykowane bloki RFU (ang. Reconfigurable Functional Unit) lub FU (ang. Functional Unit).
- Brak uniwersalności. Większość procesorów RISP wykorzystuje własną logikę reprogramowalną – konstrukcje dedykowane. Wybrane rozwiązania RISP wykorzystują istniejące układy FPGA. W takim podejściu wystepuje konieczność alokacji z góry określonej logiki reprogramowalnej FPGA w celu realizacji RFU. W rezultacie, przestrzeń implementacyjna mikrosystemu cyfrowego zostaje zmniejszona.
- Obszar aplikacyjny. Szereg procesorów RISP dedykowanych jest do realizacji zadań multimedialnych lub strumieniowego przetwarzania danych. W systemach SOPC do tego typu operacji przeznaczone się dedykowane bloki DSP.

Drugim rozwiązaniem, należącym do grupy produktów komercyjnych, który dotyczy problemu minimalizacji czasu przetwarzania i sterownia w mikrosystemach cyfrowych, jest integracja w jednym układzie cyfrowym FPGA logiki reprogramowalnej i procesora(ów) klasy HardCORE, np. układ Xilinx Virtex2Pro z procesorem PowerPC405 (integracja procesora ASIC z logiką FPGA). Wybór architektur np. Virtex2Pro, Virtex4FX lub Excalibur; związane jest z poniesieniem poważnych kosztów realizacji projektu oraz ograniczeniami aplikacyjnymi budowanego systemu wynikającymi z braku możliwości wymiany wbudowanego procesora.

Trzecia metoda dotyczy zaprzęgnięcie do procesu projektowego klasycznych metod zintegrowanego projektowania systemowego. Tutaj jednak można zauważyć trzy poważne problemy:

- Alokacja zadań procesora. Przydział zadań dla procesora dokonywany jest na poziomie specyfikacji systemu w odcięciu od analizy niskiego poziomu pracy procesora, co niekorzystnie może wpływać na wydajność końcową pracy procesora [Guda05].
- Zapotrzebowanie na dodatkowe zasoby logiki programowalnej. Podjęcie próby opracowania komponentu sprzętowego wspomagającego pracę procesora wiąże się z koniecznością przydziału na ten cel stosownych zasobów logiki programowalnej, które to wymagania nie są znane na etapie projektowania systemowego. W rezultacie, zostaje zmniejszony obszar implementacyjny pozostałych składowych mikrosystemu cyfrowego SOPC [Xili06a,Alte06,ErHe98].
- Implementacja dodatkowych komponentów funkcjonalnych. Większość znanych systemów projektowania zintegrowanego (np. Cosyma) implementuje dodatkowe bloki funkcjonalne, tj. pamięci, kontrolery DMA, specjalizowane magistrale; realizujące komunikację program-sprzęt lub składujące współdzielone dane. Natomiast, w projektowaniu mikrosystemu cyfrowego SOPC zagospodarowanie zasobów jest jednym z krytycznych parametrów realizacji projektu.

Na podstawie powyższych rozważań, poszukiwanie rozwiązania mającego na celu zmniejszenie obciążenia procesora w mikrosystemie SOPC poprzez sprzętową akcelerację wybranych zadań, powinno być skierowane na zaspokojenie następujących wymagań:

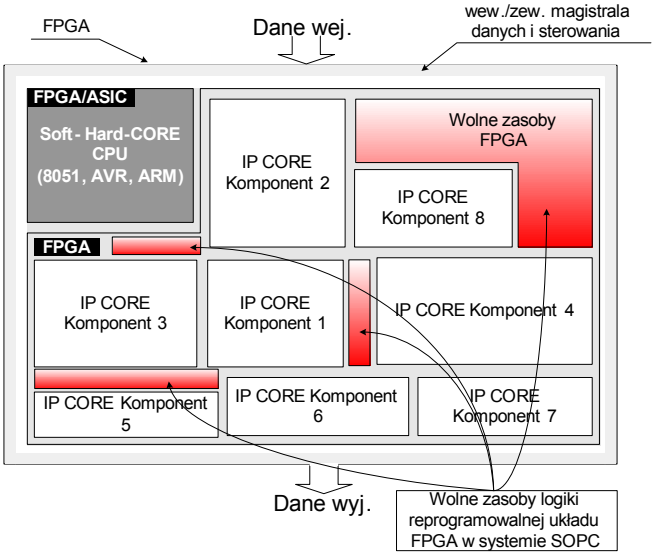
- uniwersalna architektura akceleratora sprzętowego pozwalająca na integrację dowolnego procesora RISC lub CISC w układzie SOPC,
- dostepna metoda projektowania zintegrowanego sprzętowo-programowej mikrostruktury cyfrowej (procesor + akcelerator sprzętowy),
- uniwersalność aplikacyjna opracowanego rozwiązania,
- współbieżność przetwarzania i sterowania w relacji procesor i sprzęt,
- proces analizy zadań prowadzony na niskim poziomie abstrakcji, w celu wykorzystania maksymalnych możliwości implementacyjnych procesora i części sprzętowej,
- brak nadmiarowych komponentów funkcjonalnych. Niedopuszczalna jest alokacja zasobów logiki reprogramowalnej niezbędnej do implementacji funkcjonalności kompletnego mikrosystemu cyfrowego.

3.3. Rozwiązania SPMC

W praktyce, inżynier w większości rozpatrywanych projektów, nie jest w stanie zagospodarować 100% logiki programowalnej układu FPGA w systemie zintegrowanym SOPC. Pozostaje pewien obszar niewykorzystanych zasobów sprzętowych. Średni procent wykorzystania układu FPGA w projektach realizowanych na świecie waha się w przedziale 70-90% [Xili06,Alte06], rysunek 3.1. Jednocześnie, dla każdego realizowanego projektu, obszar wolnych zasobów logiki reprogramowalnej jest różny. W rozprawie proponuje się wykorzystać wolną logikę programowalną systemu SOPC jako akcelerator

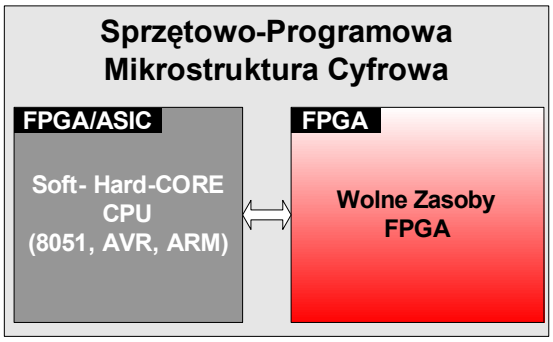
sprzętowy, tj. moduł wspomagający pracę mikroprocesora, w celu zwiększenia wydajności głównej jednostki przetwarzania i sterowania.

Definicja 3.1. *Ściśle zintegrowany sprzętowo-programowy system cyfrowy, będący składową większego mikrosystemu cyfrowego rezydującego w układzie FPGA, składający się z rdzenia procesora ogólnego przeznaczenia oraz wolnej logiki reprogramowalnej układu FPGA, jest nazywany sprzętowo-programową mikrostrukturą cyfrową SPMC.*



Rysunek 3.1 Wolne zasoby sprzętowe FPGA w realizacji systemu SOPC

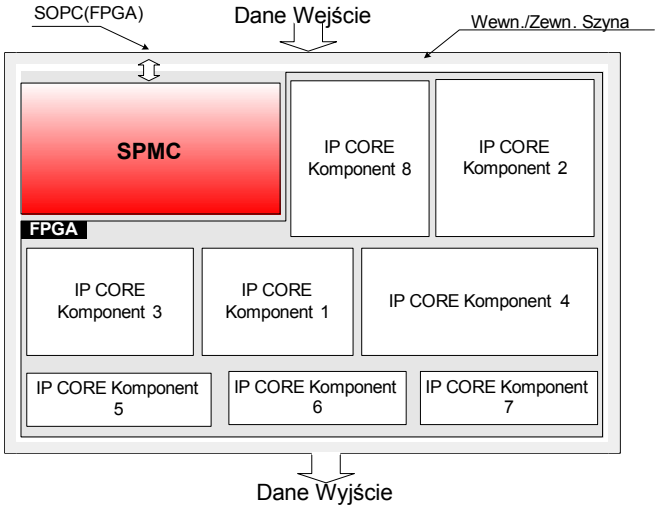
Rysunek 3.2 przedstawia główne bloki składowe mikrostruktury SPMC: *miękki* lub *twardy* mikroprocesor, plus reprogramowalna logika FPGA. Istotą opracowanej architektury jest zaprzęgnięcie niewykorzystanych zasobów sprzętowych FPGA do realizacji zadań akceleracji przetwarzania równoległego programu sekwencyjnego wykonywanego przez procesor. Proces projektowy SPMC rozpoczyna się po etapie implementacji kompletnego mikrosystemu cyfrowego do układu SOPC. Do budowy akceleratora sprzętowego SPMC wykorzystywane są tylko i wyłącznie niewykorzystane elementy logiczne FPGA procesu implementacji mikrosystemu cyfrowego.



Rysunek 3.2 Sprzętowo-Programowa Mikrostruktura Cyfrowa

W proponowanym rozwiązaniu, sprzętowo-programowa mikrostruktura cyfrowa pełni rolę głównej jednostki przetwarzania i sterowania w systemie (rysunek 3.3, 3.4). Część zadań wykonywanych do tej pory przez mikroprocesor, przejmowana

jest przez część sprzętową SPMC. Im więcej wolnych zasobów sprzętowych SOPC, tym więcej funkcji może zostać zaimplementowanych w dostępnej logice reprogramowalnej. Możliwe są dwa ekstrema realizacji SPMC: a) realizacja wyłącznie programowa, tj. cała funkcjonalność zostanie zrealizowana z wykorzystaniem procesora – brak akceleratora sprzętowego, b) realizacja wyłącznie sprzętowa, tj. brak procesora, kompletna implementacja specyfikacji zachowania w sprzęcie.



Rysunek 3.3 SPMC jako główna jednostka sterowania i przetwarzania w mikrosystemie cyfrowym

Z punktu widzenia komponentów funkcjonalnych mikrosystemu cyfrowego (bloków zadaniowych IP) oraz systemu zewnętrznego, blok SPMC postrzegany jest jako jeden komponent zadaniowy o wspólnym interfejsie wejścia/wyjścia, rysunek 3.3.

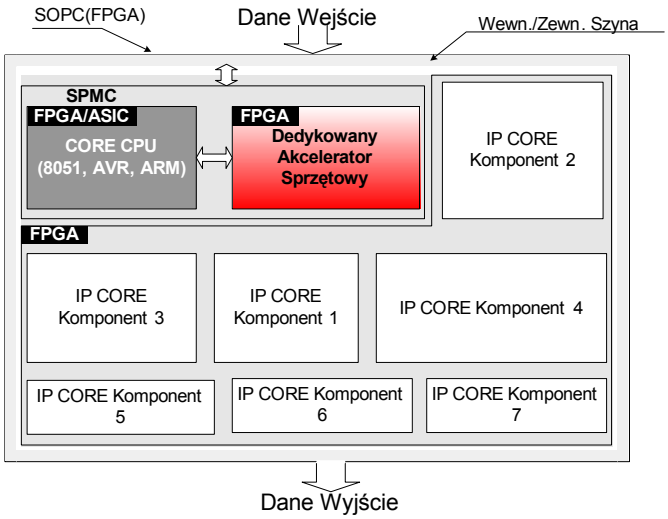
3.3.1. Implementacja ściśle zintegrowanych komponentów IP CORE w strukturach FPGA

Analiza rysunku 3.1 może nasunąć szereg pytań dotyczących sposobu implementacji wolnych zasobów FPGA oraz wpływu rozproszenia logiki programowalnej na wydajność pracy SPMC. Po zakończeniu procesu implementacji SOPC, w układzie FPGA pozostaje pewna przestrzeń niewykorzystanych zasobów CLB [xili06]. Fragmentacja wolnej logiki programowalnej układu FPGA powoduje rozproszenie implementacji układowej zadań części sprzętowej specyfikacji SPMC. Realizacja akceleratora sprzętowego w takim środowisku (rysunek 3.1) może negatywnie wpływać na wydajności pracy mikrostruktury, a wręcz ją zmniejszyć.

Problem nie dotyczy tylko i wyłącznie sposobu implementacji funkcji logicznych w układzie FPGA, ale również dystrybucji sygnałów sterujących na przestrzeni konstrukcyjnej FPGA. Proces zagospodarowania wolnych zasobów sprzętowych, pokazany na rysunku 3.1, dla architektury SPMC, realizowany z wykorzystaniem narzędzi CAD zapewniających integralność procesu implementacji wybranych bloków funkcjonalnych: Xilinx FloorPlan PACE[xili06], Altera LogicLock [alte06]. Technika implementacji SPMC składa się z pięciu kroków:

1. Implementacja w strukturach FPGA kompletnego mikrosystemu cyfrowego.
2. Oszacowanie pozostałych wolnych zasobów sprzętowych układu SOPC.
3. Zaprojektowanie części sprzętowej mikrostruktury SPMC z uwzględnieniem ograniczeń zasobów FPGA określonych w punkcie nr 2.
4. Przygotowanie instrukcji konfiguracyjnych procesu implementacji z wykorzystaniem narzędzie CAD (np. Xilinx PACE lub Altera LogicLock).
5. Ponowienie procesu implementacji mikrosystemu cyfrowego z opracowanym akceleratorem sprzętowym i konfiguracją PACE/LogicLock.

Opis przygotowywania pliku konfiguracji dla programu PACE lub LogicLock procesu implementacji dostarczany jest przez producenta oprogramowania CAD lub układu FPGA. Rezultatem przedstawionej techniki, wykorzystującej planowanie przestrzeni Xilinx-FloorPlan lub Altera-ChipEditor, jest ściśle zintegrowana realizacja sprzętowo-programowej mikrostruktury cyfrowej, która została przedstawiona na rysunku 3.4.



Rysunek 3.4 Ściśle zintegrowana sprzętowo-programowa mikrostruktura cyfrowa

3.3.2. Metody komunikacji w systemach mikroprocesorowych

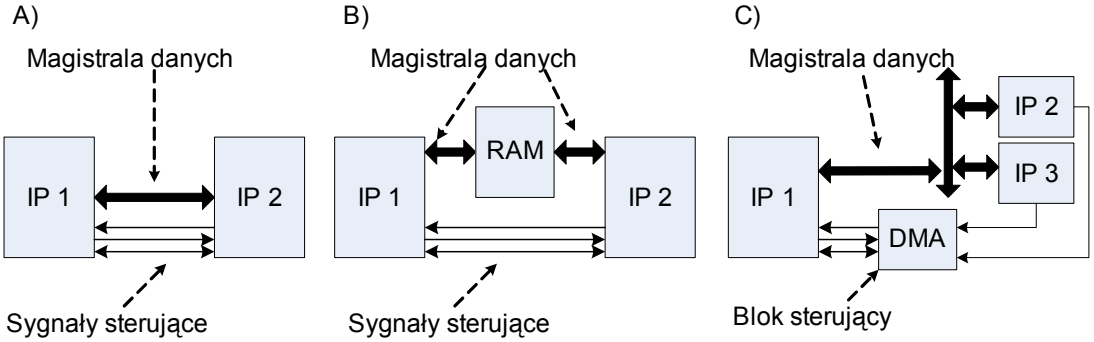
Opracowania [JeMe99, JeHu98] dostarczają szeregu rozwiązań realizacji komunikacji w systemach procesorowych lub sprzętowo-programowych. Wyróżnić można trzy podstawowe architektury komunikacji: bezpośrednia, z wykorzystaniem pamięci współdzielonej oraz DMA (ang. Direct Memory Access).

Komunikacja bezpośrednia pomiędzy dwoma obiektami systemu charakteryzuje się brakiem dodatkowych elementów funkcjonalnych niezbędnych do realizacji komunikacji. Do wad należy zaliczyć prędkość komunikacji w systemie, która nie może być większa od maksymalnej częstotliwości pracy najwolniejszego komponentu systemu, rysunek 3.5.a).

Najczęściej wykorzystywanym sposobem realizacji komunikacji w projektowaniu systemowym (Cosyma, Vulcan) jest wykorzystanie pamięci współdzielonej (rysunek 3.5.b). Zaletą jest wspólna przestrzeń pamięci danych składowych

systemu, co ułatwia proces analizy i weryfikacji funkcjonalnej. Ponadto, możliwy jest równoczesny zapis i odczyt danych z pamięci dzielonej przez oba bloki zadaniowe systemu. Wadą natomiast jest konieczność realizacji komponentu pamięci współdzielonej systemu, co wiąże się bezpośrednio z koniecznością alokacji dodatkowych zasobów sprzętowych.

Trzecim rozwiązaniem jest realizacja komunikacji za pośrednictwem kontrolera DMA [MiRo01], rysunek 3.5.c). Metoda powszechnie stosowana w systemach procesorowych i raczej nie stosowana do realizacjach komunikacji w architekturach ściśle zintegrowanych struktur sprzętowo-programowych.

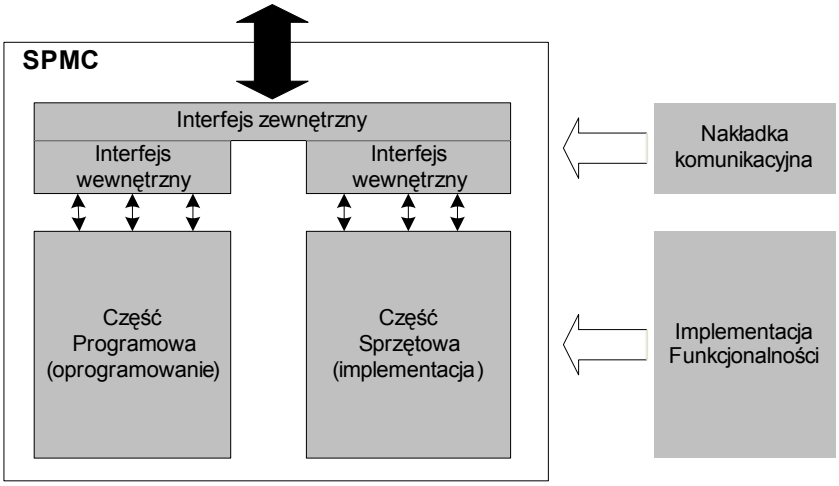


Rysunek 3.5 Metody komunikacji w systemach zintegrowanych

Ze względu na specyficzny charakter pracy, rozważaniom dotyczącym realizacji komunikacji poddane zostały tylko dwie metody: komunikacja bezpośrednia oraz komunikacja z wykorzystaniem pamięci współdzielonej. Przeprowadzone zostały badania [Stas02a] obu rozwiązań w środowisku ściśle zintegrowanego sprzętowo-programowego mikrosystemu cyfrowego. Wyniki pracy [Stas02a] (tabela 3.1) wykazały porównywalną prędkość wymiany danych w komunikacji bezpośredniej i z wykorzystaniem pamięci współdzielonej. Ze względu na ograniczone zasoby sprzętu, a zwłaszcza konieczność implementacji pamięci współdzielonej, w niniejszej pracy zdecydowano się na realizację komunikacji bezpośredniej.

3.3.3. Architektura SPMC

Strukturę wewnętrzną nowej sprzętowo-programowej mikrostruktury cyfrowej SPMC podzielono na dwa główne bloki konstrukcyjne, rysunek 3.6. Blok specyfikacji funkcjonalnej zachowania mikrostruktury zawiera dwa rozłączne opisy realizacji funkcjonalnej części sprzętowej i programowej, które definiowane są w procesie projektowym SPMC. Algorytm dekompozycji funkcjonalnej SPMC asygnuje wybrane zadania do realizacji programowej lub sprzętowej. Na etapie syntezy sprzętowej i programowej, generowany jest opis zachowania sprzętu (część sprzętowa) w języku HDL oraz kod programu (część programowa) w języku C.



Rysunek 3.6 Schemat blokowy mikrostruktury SPMC

Obie części (sprzętowa i programowa) stanowią samodzielne jednostki zadaniowe, realizujące wybrane instrukcje specyfikacji wejściowej. Drugim blokiem implementacyjnym jest wewnętrzny i zewnętrzny interfejs komunikacyjny.

Magistrala komunikacyjna w ściśle zintegrowanej sprzętowo-programowej mikrostrukturze cyfrowej (oraz w innych systemach wymiany danych) jest punktem decydującym często o powodzeniu rozwiązania, nazywanym „wąskim gardłem" systemu (ang. bottleneck). Projekt magistrali oraz protokołu komunikacyjnego w systemach zintegrowanych, musi być przeprowadzony szczególnie starannie, tak, aby nie zniweczyć zysków przetwarzania współbieżnego. Magistrala komunikacyjna program↔sprzęt w mikrostrukturze SPMC powinna charakteryzowac się następującymi cechami:

- brakiem ograniczeń dotyczących szerokości magistrali komunikacyjnej,
- zachowaniem współbieżności w realizacji komunikacji i przetwarzania przez część sprzętową,
- minimalizacją kosztów implementacyjnych systemu komunikacyjnego.

W systemach mikroprocesorowych, popularnymi rozwiązaniami dotyczącymi komunikacji w obrębie składowych systemu, są magistrale konstrukcje unormowane standardami międzynarodowymi przez m.in. ISO (ang. International Standard Organization) oraz IEEE (ang. Institute of Electrical and Electronics Engineers) w zakresie funkcjonalnym oraz elektrycznym (I2C, RS232, RS426, USB, FireWire, inne). Motywacja budowy systemów cyfrowych z wykorzystaniem tego typu rozwiązań jest oczywista: unormowany standard komunikacyjny w tym protokół transmisji, określone parametry elektryczne (łatwość integracji rozwiązań osób/firm trzecich), dostępność gotowych i sprawdzonych rozwiązań.

Znane autorowi opracowania literaturowe [JeMe99, JeHu98] dotyczące rozwiązań interfejsów i protokołów komunikacyjnych w systemach zintegrowanych, nie spałniają wymagań dotyczących systemu komunikacji w ściśle zintegrowanej architekturze SPMC.

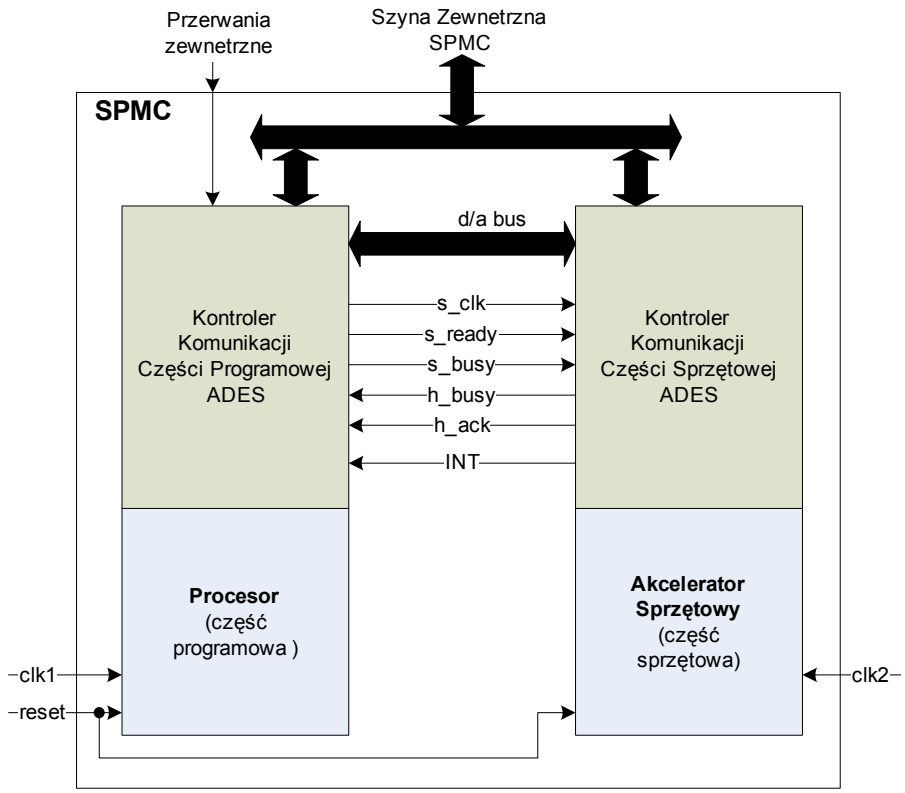
W celu zapewnienia wymiany danych i synchronizacji w hybrydowym środowisku przetwarzania i sterowania SPMC, opracowany został dedykowany interfejs komunikacyjny. Jego głównym zadaniem jest realizacja procesu wymiany danych w

komunikacji wewnętrznej SPMC, m.in.: obsługa przerwań, implementacja wewnętrznego protokołu komunikacyjnego, alokacja przestrzeni pamięci danych SPMC oraz systemu sterowania. Drugie zadanie dotyczy realizacji interfejsu komunikacyjnego mikrosystemu SPMC z otaczającym środowiskiem mikrosystemu cyfrowego. Blok komunikacji zewnętrznej udostępnia cały interfejs procesora oraz dodatkowe porty części sprzętowej. Szczegółowa architektura przedstawiona została na rysunku 3.7.

Magistrala zewnętrzna SPMC stanowi wspólny zbiór portów wejścia/wyjścia standardowego interfejsu procesora ogólnego przeznaczenia, rozszerzony o generowany na podstawie specyfikacji zachowania mikrosystemu i konfiguracji wewnętrznej, interfejs części sprzętowej. Możliwe jest zatem definiowanie zewnętrznego interfejsu wejścia/wyjścia SPMC o konfigurowalnej, czyli zmiennej liczbie sygnałów. Dzięki takiemu podejściu, uzyskano rozwiązanie uniwersalne ze względu na swobodny dostęp do głównej jednostki sterowania mikrosystemu cyfrowego.

Architektura SPMC i sposób podłączenia sprzętowego akceleratora mogą zostać zastosowane dla dowolnego procesora. Jedynymi stawianymi wymaganiami są: dostępność portów wejścia/wyjścia procesora oraz jedno wejście przerwania procesora.

Jedną z cech charakterystycznych architektury SPMC są dwa różne źródła sygnału zegarowego dla części programowej i sprzętowej, oznaczone na rysunku przez sygnały clk1 i clk2. W rozwiązaniu SPMC przyspieszenie uzyskiwane jest nie tylko przez zwielokrotnienie równoległego przetwarzania i sterowania, ale również poprzez zapewnieni maksymalnej częstotliwości pracy części sprzętowej. Synchronizacja sterowania i przetwarzania wewnętrznego i zewnętrznego dwóch bloków pracujących w różnej domenie zegarowej, zapewniona jest na poziomie specyfikacji funkcjonalnej SPMC (rozdział czwarty, sieci Petriego).



Rysunek 3.7 Architektura SPMC

W procesie komunikacji wewnętrznej SPMC wykorzystywane są następujące sygnały sterujące i magistrale danych:

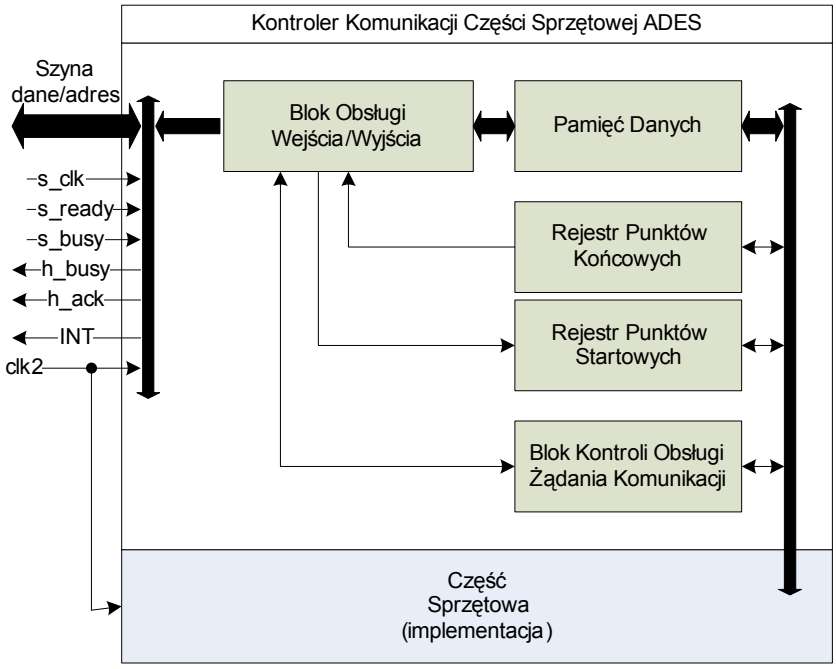
- s_clk; sygnał sterowania generowany przez program, synchronizuje przesył pakietów danych zarówno z jak i do sprzętu, aktywne zbocze narastające i opadające w zależności od typu komunikacji,
- s_ready; sygnał generowany przez procesor, aktywny podczas transmisji danych sprzęt→program, informuje o gotowości procesora (uruchomienie procedury komunikacji) na rozpoczęcie transmisji – odbiór danych, po zakończeniu sygnał ustawiany w stan nie aktywny równy '0';
- s_busy; sygnał generowany przez procesor, aktywny podczas komunikacji program→sprzęt, blokowanie wejść sygnałów przerwań procesora,
- d/a bus; szyna danych i adresowa, szerokość szyny zależna od rodzaju procesora (mechanizmy zarządzania pakietami danych i sterowania wykorzystują całkowitą liczbę portów procesora) oraz konfiguracji systemu, szyną przesyłane są pakiety danych oraz identyfikatory komunikacji (szczegółowe informacje znajdują sięw podpunkcie „Protokół Komunikacyjny SPMC"),
- h_busy; sygnał generowany przez sprzęt, identyfikuje gotowość części sprzętowej do procesu przesłania danych,
- h_ack; sygnał generowany przez sprzęt, potwierdzenie wystawienia przez sprzęt ważnego identyfikatora transmisji,
- INT; zgłoszenie żądania obsługi przerwania nadawany przez część sprzętową.

W następnych punktach pracy przedstawione zostaną szczegóły implementacyjne bloku komunikacji części programowej i sprzętowej. Natomiast bloki specyfikujące

funkcjonalność programową i sprzętową omówione zostaną w punkcie 4.1.6 i 4.1.7 rozdziału czwartego.

Blok komunikacyjny części sprzętowej

Funkcjonalność bloku komunikacyjnego części sprzętowej specyfikowana jest za pomocą języka opisu sprzętu (VHDL). Proces generacji kontrolera jest przeprowadzany w sposób automatyczny na podstawie wynikowej konfiguracji uzyskiwanej w procesie syntezy sprzętowej. Rysunek 3.8 przedstawia blokową architekturę kontrolera sprzętowego komunikacji w mikrosystemie SPMC.



Rysunek 3.8 Architekturę kontrolera sprzętowego komunikacji mikrostruktury SPMC

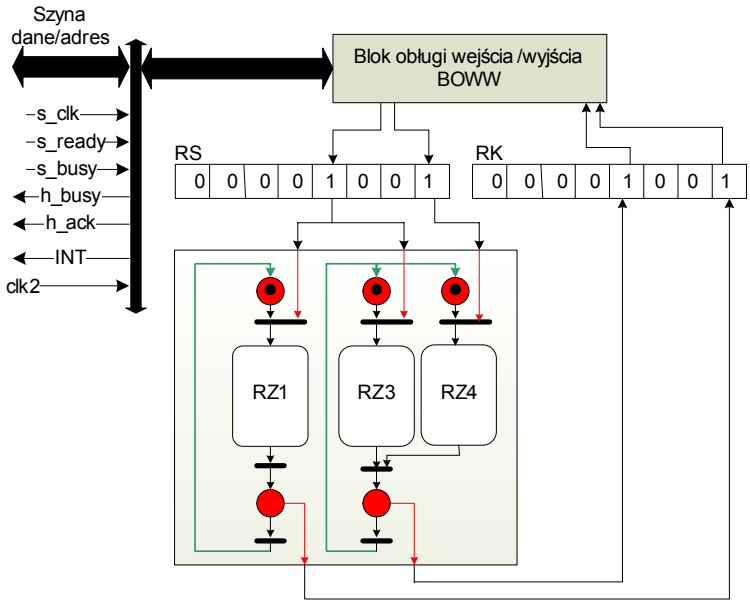
Blok obsługi wejścia/wyjścia (BOWW) jest główną jednostką sterowania części sprzętowej. Realizuje proces wysyłania i odbierania danych z procesora. Ponadto, nadzoruje start i zakończenie realizacji wybranego zadania oraz szereguje proces wysyłania danych.

Głównym założeniem opracowanej architektury jest zapewnienie współbieżności pracy części sprzętowej w stosunku do procesora oraz równoległe przetwarzanie wielu zadań w części sprzętowej. System pracuje według następującego schematu:

1. Programowe zgłoszenie żądania wykonania określonego zadania.
2. Przesłanie identyfikatora sterowania oraz danych do części sprzętowej.
3. Uruchomienie w części sprzętowej wyznaczonego zadania (start zadania); każde zadanie ma przydzielony tzw. punkt początkowy i punkt końcowy pracy oraz zdefiniowaną przestrzeń pamięci danych.
4. Zakończenie realizacji danego zadania; wyniki składowane są w pamięci wewnętrznej.
5. Wysłanie danych i identyfikatora sterowania do procesora.

Główny blok przetwarzania części sprzętowej składa się z szeregu zadań realizowanych sprzętowo, które mogą zostać: a) uruchomione w sposób niezależny od siebie, b) dwa lub więcej zadań może w dalszej części realizacji być zależnymi,

c) każde zadanie kończy się niezależnie. Przykład obrazujący sposób zlecania wykonania zadania oraz identyfikacji jego zakończenia przedstawia rysunek 3.9.



Rysunek 3.9 System zlecenia wykonania zadania części sprzętowej SPMC

Przedstawiony na rysunku 3.9 blok obsługi wejścia wyjścia BOWW wprowadza odebrane żądania przetwarzania lub sterowania do rejestru startu zadania (dla przykładu z rysunku 3.9 jest to bit 0 i 3 rejestru RS. Po czym, wybrany blok sprzętowy RZ rozpoczyna swoją pracę. Zakończenie procesu realizacji zadania RZ identyfikowane jest aktywnym bitem rejestru RK bloku BOWW (dla omawianego przykładu są to bity 0 i 3). Następnie inicjalizowany jest cykl komunikacji z procesorem. W odróżnieniu od architektur RISP i ASIP, wszystkie zadania zlecane programowalnej części sprzętowej mikrostruktury SPMC, przetwarzane są w sposób współbieżny, równolegle do pracy procesora.

Model kontrolera jest w pełni konfigurowalny. Oznacza to swobodne dopasowanie architektury kontrolera do realizowanych zadań oraz efektywne wykorzystanie układu FPGA pod względem obszaru, poprzez parametryczny zapis modelu VHDL. Możliwa jest zmiana szerokości magistrali komunikacyjnej na podstawie informacji dotyczących rodzaju procesora i typu danych (BOWW). Konfiguracji poddawana jest również wewnętrzna pamięć danych (zmiana zakresu i liczby danych) oraz rejestry punków startu RZ i końca RK. Rysunek 3.10 przedstawia przykład kodu opisującego blok komunikacyjny części sprzętowej oraz jego konfigurację.

```
1   entity accelerator is
2   generic(bus_width:integer:=7);
3   port(
4   clock          :in std_logic;
5   reset          :in std logic;
6   data           :inout std logic vector(bus width downto 0);
7   …);
8   end entity;

9   architecture n6 of accelerator is
10  constant allEntryPoints:integer:=6;
11  constant allLeavingPoints:integer:=4;
12  constant allInDataCount:integer:=5;
```

```
13  constant allOutDataCount:integer:=7+allLeavingPoints;

14  type DataRegister1 is array (0 to allInDataCount-1)of std logic vector(bus width
    downto 0);
15  signal IndataReg:DataRegister1;
16  type DataRegister2 is array (0 to allOutDataCount-1)of std_logic_vector(bus_width
    downto 0);
17  signal OutdataReg:DataRegister2;
18  signal end branches:std logic vector(allLeavingPoints-1 downto 0);
19  signal hindex:integer range 0 to allOutDataCount;
20  signal lindex:integer range 0 to allOutDataCount;
21  signal start_branches:std_logic_vector(allEntryPoints-1 downto 0);
22  signal machines done:std logic vector(allEntryPoints-1 downto 0);
23  …
24  end n6;
```

Rysunek 3.10 Fragment kodu nakładki komunikacyjnej części sprzętowej

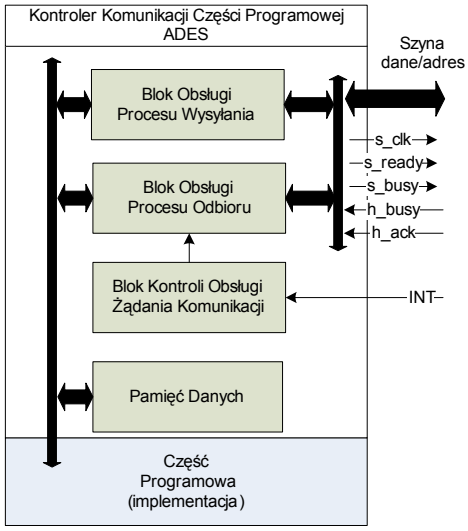
Kompletna konfiguracja dokonywana jest za pośrednictwem pięciu stałych (rysunek 3.10):

- „allEntryPoints"; linia 10, liczba punktów startowych zadań realizowanych w części sprzętowej,
- „allLeavingPoints"; linia 11, liczba punktów kończących realizację poszczególnych zadań,
- „allInDatacount"; linia 12, liczba danych wejściowych procesu synchronizacji,
- „allOutDataCount"; linia 13, liczba danych wyjściowych procesu synchronizacji,
- „bus_width"; linia 2, szerokość magistrali danych = maksymalny rozmiar danych.

Stałe te konfigurują cały moduł kontroli komunikacji wewnętrznej mikrostruktury SPMC. Część funkcjonalna i komunikacyjna realizacji sprzętowej SPMC generowane są w sposób automatyczny w procesie syntezy sprzętowej.

Blok komunikacyjny części programowej

Realizacja kontrolera komunikacji części programowej dotyczy opracowania programu kontrolera części funkcjonalnej oraz programu konfiguracyjnego procesora. Funkcje odpowiedzialne za konfigurację procesora muszą zostać przygotowywane indywidualnie dla wybranego typu procesora. Konieczność wynika z indywidualnej architektury procesorów, takich jak: konfiguracja przerwań, konfiguracja portów wejścia/wyjścia, układów licznikowych, i innych. W opracowanym systemie SPMC, funkcjonalnoś bloku kontrolera komunikacji części programowej (zbiór programowych procedur komunikacji i zarządzania), jest stała dla wszystkich typów procesorwó. Opis zachowania jest uniwersalny i został przedstawiony w języku wysokiego poziomu, ANSI C.



Rysunek 3.11 Architekturę kontrolera programowego komunikacji mikrostruktury SPMC

Reprezentacja blokowa programowej jednostki obsługi wejścia/wyjścia jest zbliżona do realizacji sprzętowej, rysunek 3.11. Różnica implementacyjna dotyczy podziału programowego bloku obsługi komunikacji na dwa niezależne komponenty funkcjonalne: blok obsługi procesu wysyłania, blok obsługi procesu odbioru.

A)

```
void print_ssp_out_pointOut7(){
  while (bit_is_set(PINB, H_BUSY)) {}

  sbi(PORTB, S_BUSY);          <- Żądanie komunikacji program->sprzęt
  PORTA = 2;
  sbi(PORTB, S_CLK);
  cbi(PORTB, S_CLK);           <- Identyfikator komunikacji
  PORTA = wy[1];
  sbi(PORTB, S_CLK);           <- Wysłanie pierwszej danej
  cbi(PORTB, S_CLK);
  PORTA = wy[2];
  sbi(PORTB, S_CLK);
  cbi(PORTB, S_CLK);
  cbi(PORTB, S_BUSY); }


int main (){
  ...
    if (isSetVect(trans, 18)) {
        print_ssp_out_pointOut7();   <- Wywołania funkcji obsługi procesu wysłania danych
        setValueVect(place, 12, 0);
  ...
}
```

B)

```
void get_ssp_in_pointIn2(){
  sbi(PORTB, S_CLK);
  cbi(PORTB, S_CLK);
  we[1] = PINA;                <- pierwsza dana
  sbi(PORTB, S_CLK);
  cbi(PORTB, S_CLK);
  we[2] = PINA;                <- druga dana
  sbi(PORTB, S_CLK);
  cbi(PORTB, S_CLK);
  cbi(PORTB, S_READY);
  setValueVect(in_flags,2, 1);<- ustawienie sygnału synchronizacji
}
  ...
INTERRUPT(SIG_INTERRUPT4) {
  uchar_t ident;
  DDRA = 0x00;
  while (!bit_is_set(PINB, H_ACK)) {}
  ident = PINA;                <- Identyfikator komunikacji
   if (ident == 0){
     get_ssp_in_pointIn0();
  }
   else if (ident == 2){
     get_ssp_in_pointIn2();     <- wywołanie funkcji procesu odbioru danych
  }
  ...
```

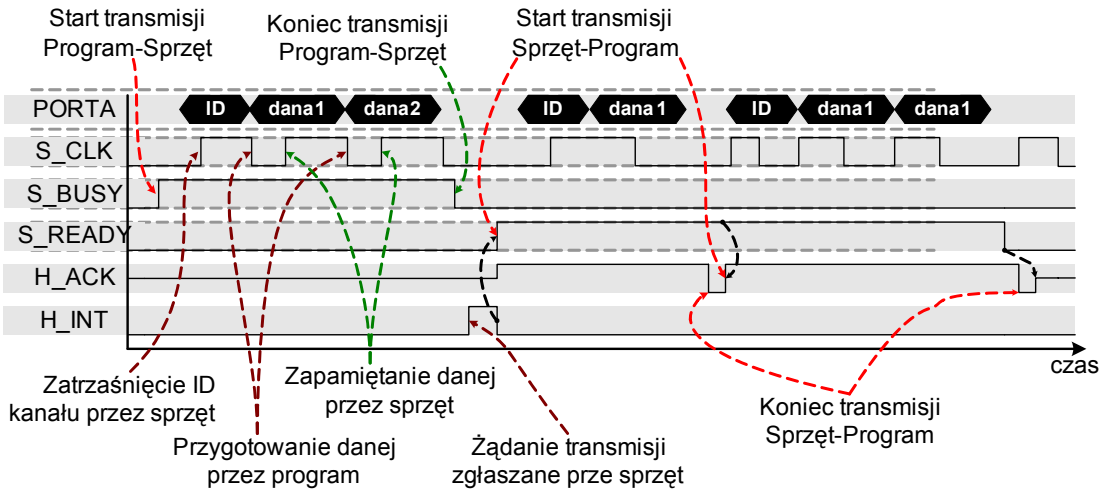
Rysunek 3.12 Fragment kod C dla mikroprocesora AVR Atmega realizującego: a) wysyłanie danych według protokołu SPMC, b) odbiór danych według protokołu SPMC

Proces wysyłania danych instancjonowany jest bezpośrednio w kodzie programu (część programowa). Odbiór jest realizowany poprzez funkcję obsługi przerwania. Przykład funkcji wysyłania danych przedstawia rysunek 3.12.a), natomiast funkcję odbioru danych przedstawiono na rysunku 3.12.b). Kod deklaracji funkcji obsługi przerwania w powyższym przykładzie opracowano dla procesora Atmel Atmega103. ~~Niezbędna jest alokacja wewnętrzna pamięci danych, która przechowuje zbiór zmiennych podlegających bezpośredniej synchronizacji z~~

~~częścią sprzętową. Zastosowanie zewnętrznej pamięci danych może ujemnie wpłynąć na całkowitą wydajność pracy procesora.~~

Protokół komunikacyjny SPMC

W rozwiązaniu SPMC szerokość magistrali komunikacyjnej opracowywana jest na etapie syntezy sprzętowej i programowej. Stały pozostaje zbiór sygnałów sterujących opisanych w dziale 3.1.2. Komunikacja przebiega według opracowanego autorskiego protokołu SPMC. Na rysunku 3.13, za pomocą wykresów czasowych, przedstawiono przykład komunikacji typu program→sprzęt oraz sprzęt→program.



Rysunek 3.13 Komunikacja wewnętrzna mikrostruktury SPMC

Protokół SPMC charakteryzuje się komunikacją pakietową z wykorzystaniem kanałów komunikacyjnych. Po nawiązaniu połączenia, pierwszą wysyłaną daną jest identyfikator komunikacji, który determinuje kanał komunikacji. Kontroler komunikacji na podstawie otrzymanego identyfikatora dekoduje kanał komunikacji, w tym liczbę i typ danych kanału, przestrzeń adresową zapisu danych oraz rozkaz uruchomienia zadania(ń) sprzętowego/programowego skojarzonego z danym kanałem. Na przykład, realizując proces komunikacji w systemie SPMC z wykorzystaniem procesora Atmel Atmega103 o magistrali 8-bitowej, można zrealizować 2^8 kanałów komunikacyjnych. Po przesłaniu identyfikatora, urządzenie nadawcze wysyła zdefiniowaną dla danego kanału liczbę danych. Transmisja kończy się po przesłaniu określonej liczby pakietów danych.

Proces transmisji program→sprzęt przebiega według protokołu:

1. Ustawienie w stan aktywny ‘1’ sygnału S_BUSY.
2. Wystawienie na szynę DATA_ADRESS identyfikatora.
3. Wygenerowanie zbocza narastającego na S_CLK, czyli przejście 0-1, i następnie 1-0.
4. Wystawienie ważnych danych na DATA_ADRESS.
5. Pwtórzenie punktu 3 i 4 w zależności od liczby przekazywanych danych.
6. Ustawienie w stan nieaktywny (‘0’)sygnału S_BUSY.

Realizacja komunikacji w kierunku program→sprzęt jest komunikacją jednokierunkową, bez potwierdzenia. Przyjmuje się za oczywiste, że część sprzętowa pracuje z kilkukrotnie większą częstotliwością pracy w porównaniu z procesorem. Ponadto blok odbioru danych części sprzętowej jest jednostką niezależną od aktualnego stanu pracy akceleratora (pod warunkiem gotowości odbioru danych). Zbytecznym okazuje się obsługa przez część programową, potwierdzania otrzymanych danych przez część sprzętową podczas transmisji danych. Rezygnacja z zaawansowanych procedur weryfikacji otrzymanych danych podyktowana jest następującymi wynikami badań przeprowadzonymi w ramach rozprawy [Stas02a]:

- Brak błędów komunikacyjnych. Wszystkie testy systemu oraz przeprowadzone eksperymenty wykazały brak błędów komunikacyjnych dla opracowanego protokołu.
- Uproszczenie protokołu komunikacyjnego. Przyspieszenie procesu komunikacyjnego poprzez eliminację zbędnych instrukcji programowych kontrolera komunikacji programowej.

Natomiast transmisja typu sprzęt→program realizowana jest według następującego protokołu:

1. Detekcja przerwania (np. INT4).
2. Ustawienie przez procesor aktywnego sygnału S_READY.
3. Dezaktywacja przez procesor sygnału S_BUSY.
4. Podanie na magistralę DATA_ADRESS identyfikatora kanału komunikacyjnego (np. 0x04).
5. Ustawienie przez część sprzętową aktywnego poziomu logicznego sygnału H_ACK.
6. Wygenerowanie przez procesor zbocza narastającego S_CLK, czyli przejścia 0-1 i następnie 1-0.
7. Podanie na magistralę DATA_ADRESS ważnych danych.
8. Wygenerowanie zbocza narastającego na S_CLK, czyli przejścia 0-1, i następnie 1-0.
9. Powtórzenie punktu 7 i 8 w zależności od liczby przekazywanych danych.
10. Ustawienie przez software w stan nieaktywny '0' sygnału S_READY.
11. Ustawienie przez część sprzętową nieaktywnego syantu sygnału H_ACK.

Koszty czasu komunikacji w architekturze SPMC

W procesie komunikacji SPMC występują dwa rodzaje komunikacji, dla których koszt czasu niezbędny do przeprowadzenia komunikacji, jest różny. Różnice wynikają jednoznacznie z protokołu komunikacyjnego oraz szerokości magistrali komunikacyjnej.

Wyznaczając koszt komunikacyjny należy uwzględnić szereg aspektów zależnych od typu procesora, ilości danych, typu danych, szerokości dostępnej magistrali oraz samego protokołu komunikacyjnego. Koszt transmisji danych w architekturze mikrostruktury SPMC pomiędzy dwoma blokami A i B rezydującymi odpowiednio: A-program, B-sprzęt, wyznacza wzór kosztów czasu komunikacji:

$$Kc = \left[\sum_{\forall LD \in D_{path}} \left(\frac{SD}{SM} \right) * Kp \right] + Kpconst \qquad \textit{(Wzór 3-1)}$$

gdzie;

Kc– suma kosztów transmisji danych z punktu A do punktu B w architekturze sprzętowo-programowej mikrostruktury cyfrowej,
D_{path}– zbiór danych wchodzących w skład ścieżki danych punktu A i B,
LD– zmienna będąca składową ścieżki danych D_{path},
SD– typ danych (szerokość),
SM– szerokość magistrali komunikacyjnej,
Kp– koszt transferu jednego pakietu danych,
Kpconst– koszt stały realizacji transferu danych (np. proces inicjalizacji komunikacji).

Dla opracowanej architektury SPMC, koszt komunikacji uzależniony jest przede wszystkim od ilości danych niezbędnych do przekazania z hipotetycznego bloku A do bloku B. Na podstawie ilości danych, typów danych i szerokości dostępnej magistrali, określana jest liczba pakietów danych, niezbędnych do przesyłania, czyli

$$\left(\frac{SD}{SM} \right) * LD.$$

Liczba pakietów obliczana jest z uwzględnieniem optymalizacji, która polega na upakowaniu w jednym pakiecie transmisji, kilku danych.

Przykład 3.1.

W procesie komunikacyjnym wysyłane są cztery dane: A(4bit), B(8bit), C(16bit), D(4bit). Ze względu na użyty procesor, dostępna jest magistrala 16bit. Liczba pakietów będzie równa 2, ponieważ:

$$\frac{4}{16} + \frac{8}{16} + \frac{16}{16} + \frac{4}{16} = 2$$

Dla systemu SPMC w dalszym ciągu prowadzone są prace eksperymentalne mające na celu implementację przedstawionej optymalizacji wysyłanych pakietów (upakowanie). Ponadto, opracowany protokół SPMC ma inną charakterystykę transmisji w konfiguracji program→sprzęt i sprzęt→program. W rezultacie przeprowadzonych badań opracowano wzór kosztów dla systemu SPMC przy realizacji komunikacji program→sprzęt, który ma postać:

$$Kc_{ps} = \left[\sum_{\forall LD \in D_{path}} \left\lceil \frac{SD}{SM} \right\rceil * Kps \right] + Kps + K_{INT} \qquad \textit{(Wzór 3-2)}$$

gdzie,

Kps– koszt transferu jednego pakietu danych w komunikacji program-sprzęt,
K_{INT}– koszt jednorazowy, czas reakcji procesora na zgłoszone żądanie obsługi przerwania zewnętrznego.

Wzór kosztów komunikacji systemu SPMC przy realizacji transmisji sprzęt→program ma postać:

$$Kc_{sp} = \left[\sum_{\forall LD \in D_{path}} \left\lceil \frac{SD}{SM} \right\rceil * Ksp \right] + Ksp \qquad \textit{(Wzór 3-3)}$$

gdzie;

Ksp– koszt transferu jednego pakietu danych w komunikacji sprzęt-program.

Koszt stały realizacji transferu danych *Kpconst* jest równy wysłaniu jednego pakietu komunikacyjnego. Wartoś kosztu *Kc* wyrazony jest jednostce czasu [t].

3.4. Podsumowanie rozwiązań SPMC

Opracowana architektura sprzętowo-programowej mikrostruktury cyfrowej jest nowatorskim rozwiązaniem w domenie akceleracji przetwarzania i sterowania mikrosystemów SOPC. Jest nową architekturą sprzętowo-programowej jednostki cyfrowej wyróżniającą się na tle znanych rozwiązań typu RISP czy ASIP:

1. SPMC zapewnia uniwersalną architekturę części sprzętowej umożliwiającą integrację dowolnego procesora RISC lub CISC. Sposób podłączenia części sprzętowej w systemie SPMC pozwala na adaptację opracowanego rozwiązania do większości dostępnych procesorów wykorzystywanych w projektowaniu mikrosystemów SOPC.
2. Opracowana metoda projektowania zintegrowanego sprzętowo-programowej mikrostruktury cyfrowej wspomaga proces realizacji układowej części sprzętowej oraz opracowanie oprogramowania dla części programowej SPMC (procesor + akcelerator), temat przedstawiono w rozdziale czwartym.
3. Architektura SPMC jest uniwersalna ze względu obszary aplikacyjne mikrosystemów cyfrowych.
4. Mikrosystruktura SPMC zapewnia współbieżność przetwarzania i sterowania w relacji mikroprocesor↔akcelerator. Część programowa zleca realizację zadań jednostce sprzętowej, jednocześnie kontynuując przetwarzanie instrukcji programowych. Ponadto akcelerator sprzętowy wykonuje wszystkie swoje zadania w sposób równoległy.
5. W rozwiązaniach SPMC, proces analizy zadań oraz wstępnej kwalifikacji instrukcji specyfikacji do części programowej lub sprzętowej, prowadzony jest na niskim poziomie abstrakcji w celu wykorzystania pełnych możliwości implementacyjnych procesora i części sprzętowej.
6. Metoda projektowania SPMC eliminuje konieczność nadmiarowej alokacji zasobów logiki reprogramowalnej, która przeznaczona jest do realizacji zadań mikrosystemu cyfrowego.
7. Dzięki realizacji komunikacji bezpośredniej w mikrostrukturze SPMC, wyeliminowano konieczność implementacji nadmiarowych komponentów funkcjonalnych (np. pamięć współdzielona, DMA).

W celu opracowania oraz implementacji tak złożonej architektury sprzętowo-programowej jednostki przetwarzania i starowania SPMC, opracowana została metoda projektowania sprzętowo-programowej mikrostruktury cyfrowej przedstawiona w rozdziale czwartym.

ROZDZIAŁ CZWARTY

4. Metoda projektowania sprzętowo-programowej mikrostruktury cyfrowej

Rozdział poświęcony został nowej metodzie automatycznego projektowania sprzętowo-programowej mikrostruktury cyfrowej SPMC. Przedstawiono metodę oraz możliwe jej punkty integracji z klasyczną i zintegrowaną metodologią projektowania systemów cyfrowych Zaprezentowano definicję nowego modelu formalnego mikrostruktury cyfrowej SPMC, wskazując na własności, jakie powinien spełniać dobry model oraz wymagania stawiane przed modelem formalnym specyfikującym zachowanie i charakterystykę pracy ściśle zintegrowanej mikrostruktury programowo-sprzętowej. Opracowano nowy format zapisu elektronicznego dla sprzętowo-programowej mikrostruktury cyfrowej specyfikowanej hierarchicznymi, czasowymi sieciami Petriego. Rozdział prezentuje również algorytm dekompozycji funkcjonalnej SPMC odpowiedzialny za podział zadań specyfikacji wejściowej na część programową i sprzętową. Przedstawiono modyfikacje i optymalizacje wybranych metod syntezy sprzętowej i programowej sieci Petriego.

4.1. Metoda projektowania sprzętowo-programowej mikrostruktury cyfrowej

Metoda projektowania SPMC oparta została na ogólnym schemacie metodologii zintegrowanego projektowania sprzętowo-programowego, przedstawionej na rysunku 2.3 rozdziału drugiego. Idee projektowania *hardware-software codesign*, ukierunkowanły autora oraz pomogły w opracowaniu pochodnej metody projektowania sprzętowo-programowego, zorientowanej na niskopoziomowy opis funkcjonalny mikrostruktury cyfrowej.

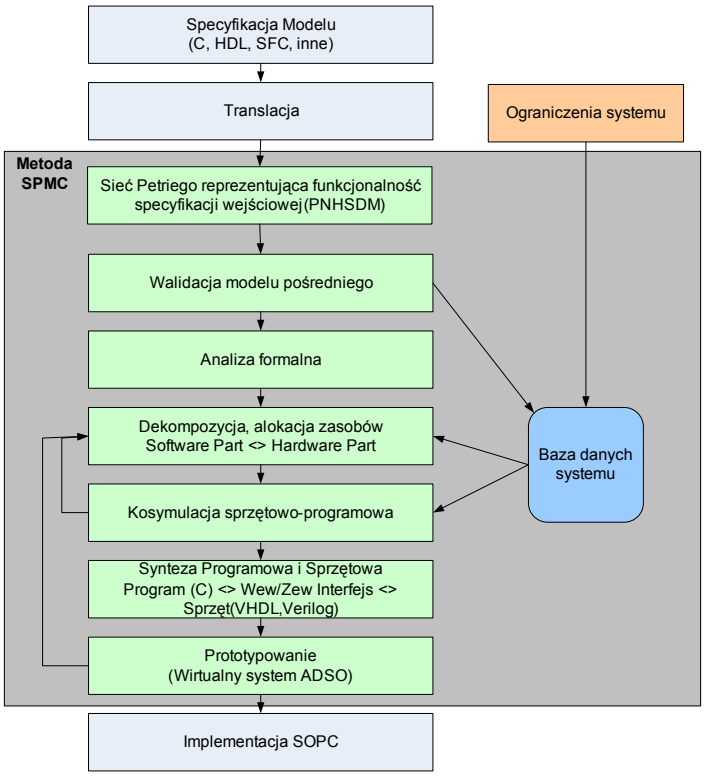
Proces projektowy dla ściśle zintegrowanej sprzętowo-programowej mikrostruktury cyfrowej, ze względu na poziom abstrakcji rozważanego problemu akceleracji przetwarzania oraz architekturę implementacyjną, prowadzony jest na

poziomie funkcjonalnym/RTL specyfikacji zachowania. Za specyfikację wejściową metody SPMC przyjmuje się sformalizowany, funkcjonalny, homogeniczny zapis przepływu danych i sterowania, składający się z prostych instrukcji programu lub wyrażeń logicznych (przypisań równoległych w języku HDL). W odniesieniu do rysunku 1.2 rozdziału pierwszego, poziom specyfikacji wejściowej metodologii SMPC zawiera się w obszarze projektowania procesorowego. Dzięki realizacji szeregu analiz formalnych i funkcjonalnych na niskim poziomie opisu zachowania, metoda projektowania SPMC pozwala na precyzyjną alokację wybranych zadań/operacji do części programowej lub sprzętowej. Ziarnistość specyfikacji funkcjonalnej ma podstawowe znaczenie w procesie pełnego wykorzystaniu wolnych, niekiedy wręcz szczątkowych zasobów logiki reprogramowalnej FPGA. Metoda SPMC zorientowana jest na realizację programową, co oznacza początkowe przypisanie wszystkich zadań specyfikacji do części programowej, a następnie przenoszenie wybranych części do realizacji sprzętowej.

Głównym zadaniem opracowanej metody jest

> *podział specyfikacji funkcjonalnej mikrostruktury cyfrowej na część programową, wykonywaną przez mikroprocesor oraz część sprzętową, realizowaną przez specjalizowane układy programowalne, z zachowaniem pełnej funkcjonalności projektowanego systemu, redukując pracochłonność realizacji projektu poprzez automatyzację oraz zwiększając wydajność pracy wynikowego mikrosystemu cyfrowego.*

Rysunek 4.1 przedstawia diagram proponowanej metody projektowania sprzętowo-programowej mikrostruktury cyfrowej SPMC.



Rysunek 4.1 Metoda projektowania SPMC

Pierwszym etapem metody projektowania sprzętowo-programowej mikrostruktury cyfrowej jest opracowanie modelu pośredniego specyfikującego funkcjonalność projektowanej mikrostruktury. Etap pierwszy SPMC poprzedzają operacje (specyfikacja, translacja) zapewniające uniwersalność opracowanych rozwiązań w zakresie swobody zapisu zachowania mikrostruktury cyfrowej. Możliwa jest integracja lub adaptacja szeregu metod translacji zapisu specyfikacji wejściowej (C, VHDL, Verilog, SystemC, inne) na model pośredni reprezentowany sieciami Petriego, np. opracowania [MiSk98a,Skow00].

Ze względu na charakter prowadzonych rozważań dotyczących akceleracji przetwarzania instrukcji programowych przez procesor, wyróżnioną specyfikacją wejściową jest opis w języku ANSI C. Wejściowy zapis sekwencyjny instrukcji programu, przygotowywany jest przez inżyniera lub np. narzędzia projektowania systemowego (rozdział 2). A następnie, z wykorzystaniem dostępnych algorytmów, program C, tj. specyfikacja wejściowa SPMC, tłumaczony jest model pośredni SPMC. Klejnymi etapami metody SPMC opisanymi w dalszej części rozprawy są: translacja i walidacja modelu pośredniego, analizy formalne i funkcjonalne, dekompozycja oraz synteza programowa i sprzętowa modelu sprzętowo-programowej mikrostruktury cyfrowej.

4.1.1. Translacja specyfikacji wejściowej do interpretowanej sieci Petriego

W pracach [Skow00,MiSk98a] przedstawiono metody translacji specyfikacji mikrostruktury cyfrowej przedstawionej w języku C, do opisu zachowania reprezentowanego sieciami Petriego. Najistotniejszą cechą metody [MiSk98a] jest zrównoleglenie programu sekwencyjnego C poprzez analizę zależności danych i sterowania zmiennych programu. Wynikiem translacji jest sieć Petriego, specyfikująca w sposób sekwencyjny i współbieżny procesy sterowania i przetwarzania programu z pełnym zachowaniem semantyki specyfikacji wejściowej. W sposób analogiczny realizowana jest translacja języków opisu sprzętu (VHDL, Verilog) do sieci Petriego. W tym przypadku optymalizacji poddawane są bloki sekwencyjne języka HDL, tj. procesy. Wynikiem translacji jest sieć Petriego specyfikująca funkcjonalność wejściową, rozszerzona o współbieżne operacje procesów.

Metoda została zaimplementowana i zweryfikowana w pracy [Skow00].

4.1.2. Model formalny sprzętowo-programowego mikrosystemu cyfrowego

Projektowanie sprzętowo-programowej mikrostruktury cyfrowej wymaga przeprowadzenia rozważań, badań oraz analiz szerokiej gamy cech, właściwości i parametrów pracy projektowanej mikrostruktury [StSk04]. Kluczowym krokiem podejmowanym już na wstępie procesu projektowego, jest opracowanie lub wybór właściwego modelu formalnego w pełni specyfikującego zachowanie dowolnego mikrosystemu cyfrowego. Ze względu na różnorodność realizowanych zadań, dobry model formalny powinien wspierać systemy pracujące współbieżnie, synchroniczne, asynchroniczne oraz mieszane, hierarchiczne, heterogeniczne; równocześnie zapewniając opis homogeniczny specyfikacji SPMC. Istniejące

modele posiadają wady, w większości natury formalnej, dyskwalifikujące ich zastosowanie lub nawet dostosowanie do specyfikacji zachowania modelu mikrostruktury cyfrowej SPMC [Skow00, GaVa95]. Dobry model musi reprezentować problem projektowy najdokładniej, jak to tylko możliwe, jednoznacznie odwzorowywać operacje projektowanego urządzenia oraz charakteryzować się przejrzystością, dostępnością narzędzi i standardem. W przeprowadzonej rozprawie za model formalny przyjęto interpretowane, czasowe, hierarchiczne sieci Petriego. Wymagania, definicje i semantyka modelu formalnego sprzętowo-programowej mikrostruktury cyfrowej SPMC zostały przedstawione w pracy [StSk06]. W kolejnych dwóch punktach omówiono wybrane definicje modelu formalnego SPMC oraz najistotniejsze, ze względu na tok prowadzonej rozprawy, zasady semantyczne modelu.

Definicja sprzętowo-programowej sieci Petriego

Nowy model formalny, opracowany dla potrzeb projektowania SPMC, opiera się na istniejących rozwiązaniach [Andr03, Skow00, GaVa95]. Nowatorstwo polega na wprowadzeniu szeregu nowych definicji rozszerzających możliwość specyfikacji heterogenicznych systemów cyfrowych.

Definicja 4.1 *Sprzętowo-programowa sieć Petriego HSPN (ang. Hardware/Software Petri Net) jest uporządkowaną siódemką:*

$$HSPN = (M, T, A, L, R, C, \mu_0) \qquad \textit{(Wzór 4-1)}$$

gdzie: M jest zbiorem *trybów*, T jest zbiorem *tranzycji*, A jest zbiorem *łuków*, L jest zbiorem *instrukcji języka programowania*, R jest zbiorem *produktów*, C jest zbiorem *warunków logicznych* i μ_0 jest *oznakowaniem początkowym,* taką że:

Definicja 4.2 *Tryb M_i jest uporządkowaną czwórką:*

$$M_i = (P_i, \lambda_i, \mu_i, \tau_i) \qquad \textit{(Wzór 4-2)}$$

gdzie: P_i jest *miejscem* sieci Petriego; λ*:* $P_i \rightarrow L \cup \{\varnothing\}$ jest funkcją przypisującą do miejsca zbiór produktów (instrukcji programistycznych) specyfikujących zachowanie (miejsce *reprezentuje* miejsce i instrukcje przypisane do niego); $\mu_i \in \{0,1\}$ jest *oznakowaniem* miejsca (miejsce może zawierać albo zero albo jeden znacznik, obecność znacznika w miejscu oznacza uruchomienie instrukcji reprezentowanych przez to miejsce); $\tau_i \in N$ jest skończonym czasem przypisanym do miejsca (reprezentuje czas wykonania instrukcji reprezentowanych przez odpowiednie miejsce). Pojedyncza instrukcja l_i ze zbioru L jest przyporządkowana do produktu r_i ze zbioru produktów R; każdy produkt r_i posiada dwa parametry $\{h,s\}$ określające jego funkcjonalność w trakcie zmiany miejsca pracy systemu; operacja przypisania nowej wartości dla produktu zależna jest od stanu logicznego parametru s:

$$\forall r_i \in P_i \; \exists s_{ij} = \begin{cases} 1, clk' \, active \Rightarrow r_i = \lambda(l_i) \\ 0, r_i = \lambda(l_i) \end{cases} \qquad \textit{(Wzór 4-3)}$$

po wykonaniu instrukcji l_i przypisanej do miejsca M_i, wartość produktu r_i może być podtrzymana lub zerowana (przypisanie wartości inicjalizującej) w zależności od parametru *h*:

$$\forall r_i \in P_i \; \exists h_{ij} = \begin{cases} 1, r_i = r_i \\ 0, r_i = r_{init} \end{cases} \qquad \textit{(Wzór 4-4)}$$

- Miejsce może być hierarchiczny. W takim przypadku miejsce Pi może być zastąpione przez HSPN. Takie miejsce określane jest nazwą: makromiejsce.
- Tranzycja może mieć przypisany warunek logiczny, nazywany strażnikiem (ang. guard) lub predykatem danej tranzycji, istnieje funkcja:

$$\chi_{Ti} : T_i \rightarrow C \cup \{\varnothing\} \qquad \textit{(Wzór 4-5)}$$

- *Łuki* łączą miejsca z tranzycjami:

$$A \subseteq M \times T \cup T \times M \qquad \textit{(Wzór 4-6)}$$

- *Oznakowanie* sieci jest zbiorem oznakowań wszystkich miejsc danej sieci, tj.:

$$\mu = \{ \mu_1, \mu_2, ..., \mu_n \} \quad dla \quad M_1, M_2, ..., M_n \in M \qquad \textit{(Wzór 4-7)}$$

Zasady semantyczne dotyczące *HSPN* są następujące:

- Oznakowanie danego miejsca M_i jest *dostępne* dla wyjściowych tranzycji jeśli miejsce jest oznakowane co najmniej przez czas przypisany do niego, tj.

$$\theta(\mu_i) \geq \tau_i \qquad \textit{(Wzór 4-8)}$$

gdzie: $\theta(x)$ jest czasem, w którym x pozostaje niezmienione.

- *Tranzycja jest przygotowana* do realizacji wtedy, gdy wszystkie jej miejsca wejściowe są oznakowane i ich indywidualne oznakowania są dostępne oraz spełnione są następujące zależności:

a) $\forall$ p$\in$ $^\bullet$t, M(p)= 1,
b) $\forall$ p$\in$ t$^\bullet$, M(p)= 0

- *Tranzycja zostaje zrealizowana* wtedy, gdy jest przygotowana do realizacji i jej warunek jest prawdziwy (*strażnik*, *predykat*). Realizacja tranzycji usuwa znaczniki ze wszystkich jej miejscach wejściowych i umieszcza znaczniki we wszystkich jej miejscachwyjściowych, w rezultacie zachodzi relacja:

a) $\forall$ p$\in$ $^\bullet$t, M(p):= 0,
b) $\forall$ p$\in$ $^\bullet$t, M(p):= 1,

Gdy instrukcja programistyczna jest przyporządkowana do miejsca, to występowanie znacznika w takim miejscu reprezentuje wykonanie przypisanej operacji. Z powodu niezerowego czasu wykonania danej operacji miejsce może być uwarunkowane czasem. W takim przypadku znacznik staje się dostępny dla wyjściowej tranzycji tylko wtedy, gdy upłynie czas przebywania w miejscu oznakowania. Jest to możliwe również wtedy, gdy rezultaty wykonania instrukcji stają się dostępne dla otoczenia.

Miejsce M_j danej sieci *HSPN* jest nazywane *hierarchicznym*, jeśli reprezentuje inny *HSPN*. Takie miejsce jest nazywane *makromiejscem* a sieć *podsiecią* i jest oznaczona przez:

$$\propto (M_j) \qquad \textit{(Wzór 4-9)}$$

Podsieć ta spełnia następujące warunki:

- Posiada ona jedno miejsce wejściowe (ang. entry mode) oznaczone jako min(Mj), które pobiera znacznik, gdy opowiadające makromiejsce odbiera znakowanie. Miejsce wejściowe ma ten sam zbiór tranzycji wejściowych co makromiejsce, tj.:

$$^{\bullet}(m_{in}(M_j)) = {}^{\bullet}M_j \qquad \textit{(Wzór 4-10)}$$

- Oznakowanie makromiejsca reprezentuje każde obowiązujące (ważne) znakowanie podsieci.
- Posiada ona jedno miejsce wyjściowe – końcowe (ang. termination mode) oznaczone jako mout(Mj), które jest jedynym oznakowanym miejscem podsieci, w przypadku ostatniego ważnego oznakowania podsieci. Miejsce końcowe ma ten sam zbiór tranzycji wyjściowych co makromiejsce, tj.:

$$(m_{out}(M_j))^{\bullet} = M_j{}^{\bullet} \qquad \textit{(Wzór 4-11)}$$

Model formalny sprzętowo-programowej mikrostruktury cyfrowej SPMC

Definicja 4.3 *Sieć Petriego dla potrzeb opisu sprzętowo-programowej mikrostruktury cyfrowej PNHSDM (ang. Petri Net for Hardware/Software Digital Microsystems) jest uporządkowaną trójką:*

$$PNHSDM = (HSPN, E, EA) \qquad \textit{(Wzór 4-12)}$$

gdzie: *HSPN* jest sprzętowo-programową siecią Petriego określoną w definicji 1, *E* jest zbiorem *wyjątków*, *EA* jest zbiorem *rozszerzonych łuków* (ang. *extended arcs*), takich że:

- *Wyjątek* jest formą tranzycji z unikalnym strażnikiem, tj. dla każdego wyjątku E_i istnieje funkcja:

$$\chi_{Ei} : E_i \to C \qquad \textit{(Wzór 4-13)}$$

- *Rozszerzone łuki* są rozszerzeniem łuków sieci *HSPN* w taki sposób, że łuki łączą miejsca z tranzycjami i wyjątkami:

$$EA \subseteq A \cup M \times E \cup E \times M \qquad \textit{(Wzór 4-14)}$$

- Pojęcia zbiorów wejściowych i wyjściowych węzłów (wierzchołków) są rozszerzeniem wyjątków.

Jeśli makromiejsce *Mj* jest wejściem wyjątku E_k, to wówczas wszystkie miejsca w jego podsieci są jawnie podłączone do tego samego wyjątku, tj.:

$$\forall m_i \in \propto(M_j) \quad M_j \in {}^{\bullet}E_k \Rightarrow m_i \in {}^{\bullet}E_k \qquad \textit{(Wzór 4-15)}$$

Zasady realizacji wyjątków różnią się od zasad realizacji tranzycji. W szczególności w celu przygotowania do realizacji wyjątku, wystarczy aby tylko jedno z jego miejsc wejściowych było oznakowane. Dodatkowo, jeśli dane miejsce ma przyporządkowany czas, to jego oznakowanie jest zawsze dostępne dla wyjątku.

Realizacja wyjątku może mieć szczególny wpływ na stan podsieci makromiejsca wywołującego obsługę wyjątku. Możliwe jest wstrzymanie (pauza) wykonywania

lub wprowadzenie w stan początkowy podsieci w chwili utraty znakowania przez makro miejsce. Stan logiczny ϱ determinuje właściwe zachowanie podsieci:

$$\forall M \; \exists \rho = \begin{cases} 1, {}^{\bullet}M_j = M_{exception}{}^{\bullet} \\ 0, {}^{\bullet}M_j = {}^{\bullet}M_{init} \end{cases} \qquad \textit{(Wzór 4-16)}$$

Zasady semantyczne dotyczące *PNHSDM* są zdefiniowane następująco:

- Wyjątek jest przygotowany do realizacji, jeśli co najmniej jeden z jego miejsc wejściowych jest oznakowany.
- Wyjątek zostaje zrealizowany, gdy jest przygotowany do realizacji i jego warunek logiczny (strażnik, predykat) jest prawdziwy. Realizacja wyjątku usuwa znaczniki ze wszystkich jego oznakowanych miejsc wejściowych (także z tych, dla których nie jest dostępne oznakowanie) i umieszcza znaczniki we wszystkich jego miejscach wyjściowych, gdy ϱ=0. Natomiast, wyjątek pozostawia (nie usuwa) znaczniki we wszystkich oznakowanych miejscach wejściowych i oznacza wszystkie swoje miejsca wyjściowe, gdy ϱ=1.
- Wyjątki są przygotowanego do realizacji przez każde oznakowanie swoich miejsc wejściowych, niezależnie od czasu w jakim są oznakowane.

Dla potrzeb opisu zachowania i dokumentacji mikrostruktury cyfrowej, specyfikowanej w oparciu o model *PNHSDM*, opracowany został format zapisu sprzętowo-programowych sieci Petriego *SPNF* (ang. *System Petri Net Format*) [StAd06] opisujących kompletny system cyfrowy w standardzie XML [OASIS].

4.1.3. Format zapisu modelu pośredniego SPNF

Prace nad ustanowieniem jednolitego standardu opisu sieci Petriego są wciąż prowadzone [ieee06]. Grupy robocze skupiają się wokół organizacji Petri Net Word [PeNe] oraz OASIS [OASIS]. Wczesne wersje robocze powstającego standardu PNML [ieee06] wykorzystano między innymi w pracach [Kuba04,Dec06]. ~~Dla celów specyfikacji zachowania i syntezy układów cyfrowych opisanych sieciami Petriego opracowano format PNSF[KoDa95], którego główną wadą jest autorski system regul utrudniający swobodną wymianę projektów/plików w środowisku akademickim i komercyjnym (CAD). Konstrukcje regulowe PNSF wprowadzają znaczące utrudnienia często uniemożliwiając wręcz tekstową edycję, analizę oraz odtworzenie zapisu tekstowego na reprezentację graficzną, złożonej sieci Petriego, przez człowieka.~~

Pierwsze prace naukowe prezentujące koncepcje PNML [BiCh03,WeKi03] zostały opublikowane w latach 2003. Prace inżynieryjno-badawcze, dotyczące tematu niniejszej rozprawy doktorskiej, rozpoczęto w roku 2002 [Stas02a,Stas03a,Stas03b, StMi03,StMi03b]. Już w początkowym etapie prac niezbędnym okazało się opracowanie autorskiego formatu zapisu elektronicznego sieci Petriego wspierającego modelu PNHSMC, którego wyniki opublikowano w pracy [StAd06].

Nowy format SPNF zapisu sieci Petriego umożliwia opis specyfikacji zachowania i dokumentacji sprzętowo-programowej mikrostruktury cyfrowej. Ponadto, specyfikacja SPNF pozwala na zapis zachowania systemu cyfrowego z uwzględnieniem aspektów takich jak: hierarchia, czas, kolorowanie, parametry

implementacji składowych systemu jako program lub realizacja sprzętowa, deklaracja interfejsu we/wy, konfiguracja makromiejsc, i inne. Format SPNF zapewnia spójność opisu zachowania dowolnego systemu cyfrowego specyfikowanego za pomocą:

- klasyczne sieci Petriego – PN,
- interpretowane sieci Petriego – IPN,
- hierarchiczne IPN – HIPN,
- sieci HIPN uwarunkowane czasem– THIPN,
- sieci THIPN implementujące algorytmy programowe, w tym ścieżki przepływu danych – DTHIPN,
- asynchroniczne sieci DTHIPN.

Ponadto, dopuszczalne jest specyfikowanie zachowania systemów synchronicznych, asynchronicznych lub mieszanych, gdzie np. sieć instancjonowana jako makromiejsce pracuje w sposób synchroniczny, natomiast makromiejsce względem pozostałych elementów głównego systemu pracuje w sposób asynchroniczny. Dodatkowo, wprowadzono szereg znaczników rozszerzających możliwości zapisu specyfikacji, dokumentacji i konfiguracji mikrostruktury cyfrowej, między innymi. *htime* (czas propagacji miejsca P jako realizacji sprzętowej), *hresources* (liczba zasobów sprzętowych niezbędnych do implementacji miejsca P), *stime* (czas wykonania funkcji miejsca P przez określoną jednostkę CPU), *sresources* (rozmiar kodu miejsca P jako realizacji programowej), *activity* (aktywność symulacyjna miejsca P). Wymienione parametry w pełni charakteryzują własności implementacyjne specyfikacji wejściowej, reprezentowanej przez model pośredni, a przez to system SPMC. Ponadto, dzięki wprowadzeniu omówionych znaczników XML zapisu SPNF, zapewniono spójność danych dotyczących konfiguracji, analizy i opisuj zachowania systemu podczas wymiany (pomiędzy projektantami), publikacji i przetwarzania projektu.

Format SPNF został opracowany w standardzie XML (ang. Extensible Markup Language). Dzięki temu zapewniono swobodny rozwój formatu oraz jego adaptację przez inne narzędzia typu CAD. Format SPNF zapewnia między innymi:

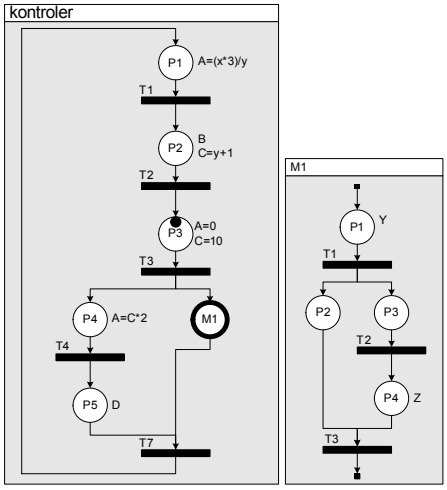
- przejrzystość opisu funkcjonalnego sieci; rozdzielenie warstwy opisu zachowania modelu od reprezentacji wizualnej i analitycznej, możliwa ręczna analiza/korekta i edycja opisu systemu przez projektanta,
- przechowywanie w jednym pliku projektowym wyników szeregu przeprowadzonych analiz sieci, takich jak: DFG, FSM, aktywność funkcjonalna, czas, analiza formalna sieci Petriego (żywotność, bezpieczeństwo, determinizm),
- łatwa wymiana danych pomiędzy różnymi systemami CAD poprzez transformację formatu XML-SPNF do innego zapisu XML z wykorzystaniem arkusza transformacji XSLT[Clar99], w tym również na format PNML [Kuba04,Dec06,Węgr03] oraz PNSF [KoDa95],
- proste i łatwe rozszerzenie własności formatu SPNF o kolejne definicje i parametry, zalety XML,
- hierarchiczny przegląd/podgląd opisu systemu w dowolnym edytorze XML,
- adaptacja wielu narzędzi przeznaczonych do analizy i przetwarzania danych przechowywanych w zapisie XML.

Pełna specyfikacja SPNF załączona została w dodatku A.

Przykład specyfikacji mikrostruktury cyfrowej z wykorzystaniem formatu SPNF

Prezentowany przykład ma na celu przedstawienie sposobu zapisu specyfikacji PNHSMC oraz konfiguracji hierarchicznych systemów sterowania i przetwarzania z wykorzystaniem formatu SPNF. Na rysunku 4.11 przedstawiona została specyfikacja zachowania sprzetowo-programowej mikrostruktury cyfrowej, która została uzupełniona o parametry pracy części programowej i sprzętowej, tj. koszty realizacji, wyniki analizy funkcjonalnej oraz wstępną konfigurację implementacyjną (wyniki syntezy programowej i sprzętowej).

Przykład przedstawia specyfikację zachowania hierarchicznego modelu kontrolera cyfrowego. Na rysunku 4.11 przedstawiono sieć Petriego, gdzie zdefiniowano jedno miejsce inicjalizujące P3. Sieć spełnia właściwości bezpieczeństwa i żywotności. Ponadto w miejscach P1, P2, P5, M1.P1, M1.P4 zadeklarowane zostały akcje sygnałów typu Moore'a.



Rysunek 4.11 Model kontrolera opisanego hierarchicznymi sieciami Petriego

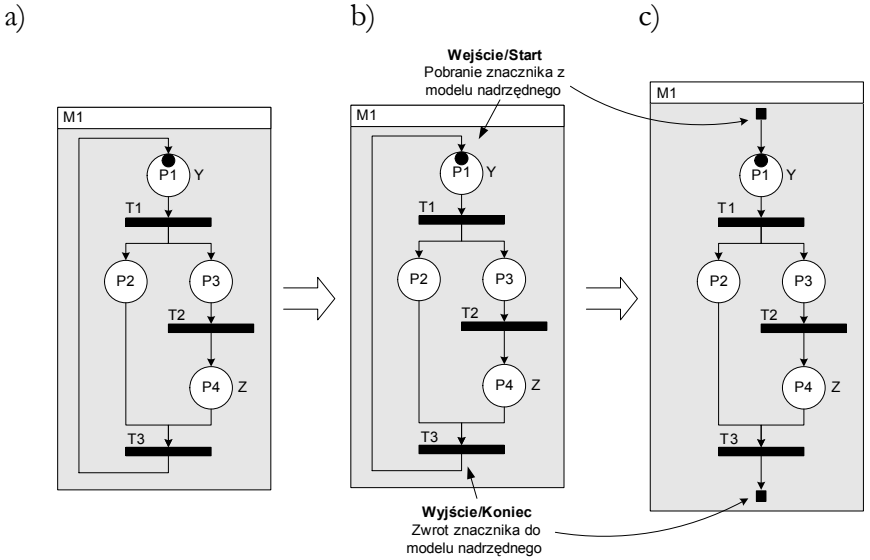
Rysunek 4.11 przedstawia reprezentację graficzną funkcjonalności kontrolera, natomiast rysunek 4.12 reprezentację tekstową w formacie SPNF.

```
<SystemDescription name=" kontroler">
    <resolution>NS</resolution>
    <clock name="clk">
        <edge>rising</edge>
    </clock>
    <model name="kontroler">
        <interface>
            <output>A</output>
            <output>B</output>
            <output>C</output>
            <output>X</output>
        </interface>
        <place name="P1">
            <produce>A=(X*3)+Y</produce>
        </place>
        <transition name="T1">
            <sensitive>P1</sensitive>
            <action>P2</action>
        </transition>
        <place name="P2">
            <produce>B</produce>
            <produce>C=Y+1</produce>
        </place>
        <transition name="T2">
            <sensitive>P2</sensitive>
            <sensitive>P4</sensitive>
            <action>P5</action>
        </transition>
        <place name="P5">
            <produce>X</produce>
        </place>
        <transition name="T7">
            <sensitive>P5</sensitive>
            <sensitive>M1</sensitive>
            <action>P1</action>
        </transition>
    </model>
    <model name="M1">
        <interface>
            <output>Y</output>
            <output>Z</output>
        </interface>
        <place name="M1_P1">
            <produce>Y</produce>
        </place>
        <transition name="M1_T1">
            <sensitive>M1_P1</sensitive>
            <action>M1_P2</action>
            <action>M1_P3</action>
        </transition>
```

```
        <action>P3</action>
    </transition>
    <place name="P3">
        <initial>1</initial>
        <produce>A=0</produce>
        <produce>C=10</produce>
    </place>
    <transition name="T3">
        <sensitive>P3</sensitive>
        <action>P4</action>
        <action>M1</action>
    </transition>
    <place name="P4">
        <produce>A=C*2</produce>
    </place>
    <place name="M1">
        <source>current</source>
    </place>
    <transition name="T4">
        <place name="M1_P2">
        </place>
        <place name="M1_P3">
        </place>
        <transition name="M1_T2">
            <sensitive>M1_P3</sensitive>
            <action>M1_P4</action>
        </transition>
        <place name="M1_P4">
            <produce>Z</produce>
        </place>
        <transition name="M1_T3">
            <sensitive>M1_P4</sensitive>
            <sensitive>M1_P2</sensitive>
            <action>M1_P4</action>
        </transition>
    </model>
</SystemDescription>
```

Rysunek 4.12 Zapis tekstowy modelu z rysunku 4.11

Miejsce M1 jest makromiejscem instancjonującym niezależny model, który może być zrealizowany i poddany weryfikacji funkcjonalnej w oddzielnym procesie projektowym. W opisie modelu „kontroler", komponent M1 rozumiany jest jako miejsce o budowie hierarchicznej, które poddawane jest analizom i symulacji jako niezależny komponent funkcjonalny. Proces adaptacji modelu do implementacji jako komponent makromiejsca systemu przedstawia rysunek 4.13.



Rysunek 4.13 Proces adaptacji modelu do implementacji jako komponent makromiejsca systemu, gdzie: a) samodzielna jednostka funkcjonalna, b) analiza wszystkich miejsc inicjalizujących sieci Petriego, c) instancjacja modelu jako komponentu – wyznaczenie punktów końcowych sieci

Opracowany format SPNF pozwala na zapis dwóch typów makromiejsc (pełny opis z załączniku A): proceduralne i współdzielone. W modelu „kontroler" specyfikacji z rysunku 4.11, miejsce M1 zadeklarowano jako makromiejsce typu „procedural macroplace". W takim rozwiązaniu, wszystkie zasoby zarówno komponentu M1 i modelu „kontroler" (zasoby rozumiane są jako: sygnały lokalne, nazwy miejsc i tranzycji) są współdzielone i traktowane jak jeden łączny opis systemu (dodatek A). Natomiast deklaracja miejsca M1 jako makromiejsca typu „shared macroplace" nie narusza wewnętrznej przestrzeni nazw

instancjonowanego komponentu, jednak wymagana jest definicja połączeń (mapowanie) sygnałów lokalnych (interfejs wejścia/wyjścia komponentu) z aktualnymi sygnałami modelu. System nadrzędny „kontroler", w przeciwieństwie do implementacji makromiejsca proceduralnego, nie ma bezpośredniego dostępu do zmiennych komponentu M1 identyfikujących miejsca, tranzycje, sygnały lokalne. Wymiana danych i sygnałów sterowania realizowana jest tylko i wyłącznie przez interfejs wejścia/wyjścia zainstancjonownej podsieci. Zasady kojarzenia sygnałów zostały szczegółowo opisane w dodatku A, znacznik: *place*, *bind*. Rysunek 4.14 przedstawia fragment kodu SPNF prezentujący instancjację modelu M1 jako miejsca typu „shared macroplace".

```
<place name="M1">
    <component>M1</component>
    <source>current</source>
    <bind>Y_aktual</sensitive>
    <bind>Z_aktual</sensitive>
</place>
</model>
```

Rysunek 4.14 Instancjacja makromiejsca typu „shared" (fragment kodu SPNF)

Przykład zapisu informacji analitycznych precyzujących własności sprzętowo-programowej mikrostruktury cyfrowej SPMC, przedstawia rysunek 4.15. Prezentowane dane pozyskiwane są w wyniku analizy funkcjonalnej i formalnej rozpatrywanej specyfikacji zachowania modelu cyfrowego. Zapis SPNF z rysunku 4.14, dla miejsca P4 został rozszerzony o parametry charakteryzujące: implementację, czas, funkcjonalność.

```
01 place name="P4">
02            <produce>A=C * 2</produce>
03            <htime>23</htime>
04            <stime>4500</stime>
05            <hresources>81</hresources>
06            <sresources>11</sresources>
07            <activity>100</activity>
08            <implement>h</implement>
09            <color>12</color>
10        </place>
11    </model>
```

Rysunek 4.15 Fragment kodu SPNF opisujący parametry pracy i własności miejsca P4 sieci Petriego z rysunku 4.11

Rozpatrując zapis XML rysunku 4.15, odczytuje się co następuje w kolejności zapisu: a) do miejsca P4 przypisana jest akcja produkująca zmienną/sygnał A, b) czas realizacji zadania miejsca P4 jako realizacji sprzętowej wynosi 23[ns], c)czas wykonania instrukcji programowej miejsca P4 przez dany procesor wynosi 4500[us], d) koszt implementacji sprzętowej jest równy 81[CLB], e) rozmiar kodu programu wynosi 11[B], f) aktywność funkcjonalna pracy zadania jest równa 100 wywołaniom, g) P4 zakwalifikowano do implementacji w sprzęcie, h) przydzielono kolor nr 12 .

4.1.4. Dekompozycja funkcjonalna SPMC

Problematyka algorytmu dekompozycji funkcjonalnej sprzętowo-programowej mikrostruktury cyfrowej, ze względu na zdefiniowaną architekturę SPMC oraz zdecydowanie mniejszy zakres obszaru poszukiwań rozwiązań, pozbawiona jest problemów występujących w metodach kosyntezy [ErHe98,BaCh97] systemów cyfrowych, tj. szeregowania, poszukiwania optymalnych architektur procesorów,

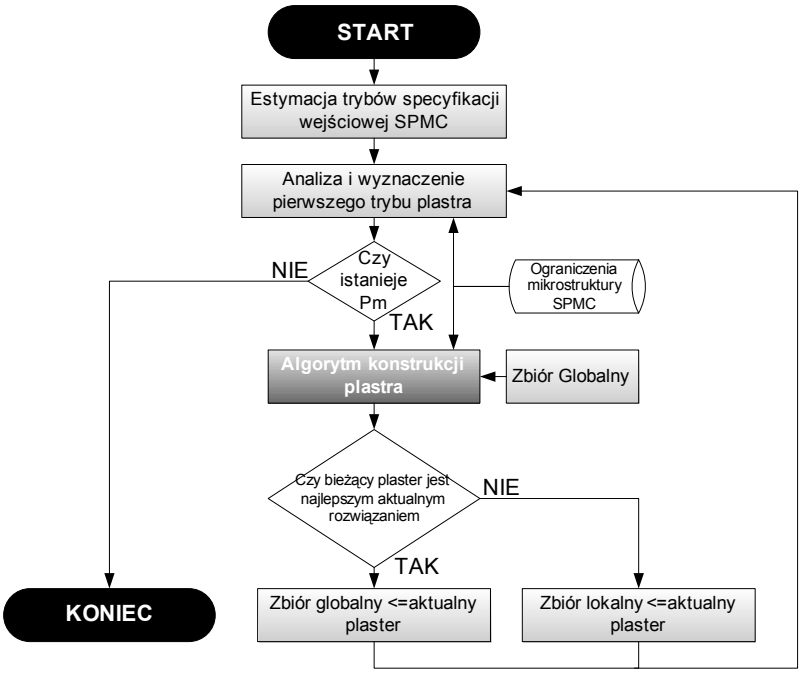
wyznaczania technologii części sprzętowej (ASIC, FPGA, CPLD), przyporządkowania zadań, poboru mocy, i inne. W implementacji algorytmu dekompozycji funkcjonalnej SPMC wskazane jest zastosowanie algorytmów wyszukiwania wyczerpującego [DAHu94] lub metod opierających się na programowaniu liniowym całkowitoliczbowym [PrPa92]. Z drugiej strony, dopuszczalne jest również zaprzęgnięcie metod heurystycznych wykorzystywanych w syntezie systemowej, takich jak: algorytmy konstrukcyjne [DaLa97,BiAu98], rafinacyjne [HoWo96,ElPe95] lub algorytmy probabilistyczne [DiJh97].

~~W procesie dekompozycji funkcjonalnej SPMC algorytm podziału dokonuje szacowania, kwalifikując poszczególne zadania do realizacji programowej lub sprzętowej.~~

Algorytm dekompozycji SPMC

Opracowany w rozprawie algorytm dekompozycji funkcjonalnej SPMC jest wynikiem studiów literaturowych oraz badań nad autorskim algorytmem dekompozycji funkcjonalnej ADSO [StMi03b][as5]. Algorytm SPMC zorientowany jest na grupowanie zadań części programowej i sprzętowej oraz eliminację ziarnistości i rozproszenia realizacji funkcjonalnej SPMC. Ze względu na definiowanie rozwiązań dekompozycji w postaci plastrów zadaniowych oraz technikę wychodzenia z lokalnych minimów (wyznaczanie kolejnych punktów startowych z poza bieżącego obszaru analizy), algorytm SPMC można zaliczyć do algorytmów heurystycznych, z grupy konstrukcyjnych. Wybrane rozwiązania algorytmu SPMC wzorowane były na opracowaniach algorytmu COSYM [DaLa97]. Pod względem koncepcyjnym, są one do siebie zbliżone, natomiast znacząco różnią się algorytmem budowy plastra zadaniowego, co wynika bezpośrednio z rozważanego problemu rozprawy oraz mikrostruktury SPMC. Za *plaster zadaniowy SPMC* uznaje się integralny/nierozłączny zbiór sąsiadujących ze sobą miejsc i tranzycji sieci Petriego zakwalifikowany do realizacji programowej lub sprzętowej. Konieczność budowy tzw. plastra zadaniowego wynika z własności pracy systemów heterogenicznych. Jednym z głównych źródeł spadku wydajności pracy systemów heterogenicznych o ścisłej integracji części programowej i sprzętowej, jest proces wewnętrznej komunikacji sprzęt-program, dotyczący synchronizacji danych i sterowania. Przeprowadzone badania [Stas02a,StMi03a] oraz rozważania literaturowe [JeMe98,JaZb00,GaVa94] wskazują na konieczność optymalizacji i minimalizacji punktów komunikacyjnych w sprzętowo-programowej mikrostrukturze cyfrowej. Algorytm budowy plastra zadaniowego dąży do minimalizacji punktów komunikacyjnych w architekturze programowo-sprzętowej mikrostruktury cyfrowej.

~~W pracy nie dokonano porównania obu rozwiązań ze względu na wyraźne różnice koncepcyjnie wynikające z poziomu abstrakcji rozpatrywanej specyfikacji systemu.~~
Diagram pracy algorytmu SPMC przedstawia rysunek 4.17.



Rysunek 4.17 Algorytm dekompozycji funkcjonalnej SPMC

Pierwszym etapem jest wyznaczenie początkowego miejsca *Pm* sieci dla pierwszego sprzętowego plastra zadaniowego. Miejsce *Pm* sieci musi spełniać następujące warunki wyboru:

1. Miejsce znajduje się w podzbiorze sieci o największym współczynniku współbieżności,
2. Miejsce ma najgorsze parametry realizacji programowej,
3. Miejsce ma najkorzystniejsze parametry realizacji sprzętowej,
4. Miejsce ma największy współczynnik aktywności funkcjonalnej,
5. Miejsce nie należy do żadnego znalezionego rozwiązania zbioru lokalnego i globalnego.
6. Miejsce ma największą liczbę miejsc przyległych do końcowej (współbieżnej) tranzycji zbiorczej.

Algorytm dekompozycji funkcjonalnej SPMC składa się z dwóch części. Pierwszą częścią jest algorytm wyznaczania punktu *Pm*, który zarządza plastrami zadaniowymi oraz wyznacza punkty startowe kolejnego plastra, rysunek 4.17. Druga część to algorytm konstrukcji sprzętowego plastra zadaniowego, który realizuje proces konstrukcji pojedynczego plastra sieci Petriego na podstawie podanego punktu startowego *Pm*.

W procesie pracy algorytmu dekompozycji SPMC, wyznaczanych jest PN plastrów zadaniowych, gdzie PN jest liczbą miejsc sieci Petriego specyfikującej zachowanie mikrostruktury cyfrowej. Wyodrębniono dwa zbiory, gdzie składowane są wyniki przeprowadzonych analiz i rozwiązań. Zbiór globalny przechowuje aktualnie najkorzystniejsze znalezione rozwiązanie, natomiast zbiór lokalny zapamiętuje wszystkie znalezione rozwiązania. W procesie wyznaczania kolejnego punktu początkowego plastra P_m, mogą brać udział miejsca sieci nie oznaczone wcześniej jako punkt P_m. Proces dekompozycji SPMC kończy pracę w chwili, gdy nie można

wyznaczyć kolejnego punktu początkowego *Pm* plastra (znaleziono wszystkie możliwe rozwiązania). Rozwiązaniem jest zbiór globalny. Szkic algorytmu przedstawia rysunek 4.18.

```
Alg.4.1
    estymacja kosztów realizacji i czasu ∀m∈M
   Pm:= wyznacz_miejsce_początkowe_klastra(Zbiór.Miejsc);
   Zbiór.Globalny:=∅;
   outher_loop: do
           Zbiór.Aktualny=uruchom_algorytm_plastra(Pm, Zbiór.Globalny);
           if Zbiór.Aktualny(zysk)>Zbiór.Globalny(zysk) then
             ZbiórGlobalny:=Zbiór.Aktualny;
             Zbiór.Lokalny:= Zbiór.Lokalny +Zbiór.Globalny;
           eles
              Zbiór.Lokalny:= Zbiór.Lokalny +ZbiórAktualny;
           end if;
           Zbiór.Miejsc:=Zbiór.Miejsc–Zbiór.Aktualny;
           Pm:= wyznacz_miejsce_początkowe_plastra(Zbiór.Miejsc);
         while(Pm≠∅)
rozwiązanie:=Zbiór.Globalny
```

Rysunek 4.18 Algorytm zarządzania plastrami zadaniowymi oraz wyznaczaniem punktów startowych

Zadanie znalezienia najbardziej korzystnego rozwiązania spoczywa na algorytmie konstrukcji plastra. Od niego zależy, który element składowy sieci specyfikującej zachowanie SPMC, zostanie przypisany do realizacji sprzętowej, a który do programowej. Niezbędne jest wyznaczenie funkcji zysku podziału, na podstawie której algorytm konstrukcji plastra będzie podejmował decyzje alokacji wybranego zadania do części sprzętowej lub programowej. Dla rozwiązań SPMC funkcja kosztów przeniesienia wybranego zadania do części sprzętowej jest równa:

$$\Delta t = \sum_{\forall M \in S} tp(M) - \sum_{\forall M \in P} tp(M) + \sum_{\forall M \in P} ts(M) + Kc_{ps} + Kc_{sp}$$

(Wzór 4-17)

gdzie:

S– zbiór miejsc specyfikacji SPMC,
P– zbiór miejsc plastra zadaniowego,
Kc_{sp}– koszt czasu transmisji sprzęt→program,
Kc_{ps}– koszt czasu transmisji program→sprzęt,
tp(M)– czas pracy miejsca specyfikacji SPMC jako realizacja programowa,
tp(M)– czas pracy miejsca plastra zadaniowego jako realizacja programowa,
ts(M)– czas pracy miejsca plastra zadaniowego jako realizacja sprzętowa,
Δt– różnica czasu po przeniesieniu miejsca do części sprzętowej.

Zysk podziału rozumiany jest jako skrócenie (przyrost ujemny) czasu przetwarzania zadanej specyfikacji przez mikrostrukturę SPMC. Zysk Δt jest różnicą sumy czasów *tp(M)* wykonania zadań miejsc *M* specyfikacji wejściowej *S* przez mikroprocesor i sumy czasów wykonania miejsc *M* sprzętowego plastra zadaniowego *P*, zwiększoną o koszty komunikacji *Kc* oraz sumy czasów wykonania miejsca *M* specyfikacji plastra *P* jako realizacji sprzętowej.

Algorytm dekompozycji funkcjonalnej SPMC dokonuje przeniesienia wybranego miejsca pracy sieci z części programowej do sprzętowej przy spełnionej nierówności:

$$\Delta t_{actual} < \Delta t_{previous} \qquad \textit{(Wzór 4-18)}$$

gdzie:

t_{actual}; czas wykonania specyfikacji SPMC po przeniesieniu miejsca *M* do części sprzętowej,

$t_{previous}$; .czas wykonania specyfikacji SPMC przed przeniesieniem miejsca M do części sprzętowej.

Proces wstępnej kwalifikacji miejsca M specyfikacji SPMC jako realizacji sprzętowej, przeprowadzany jest według następujących kryteriów:

- miejsce nie jest wykluczone z bieżącego rozwiązania (algorytm budowy plastra zadaniowego może wykluczyć miejsce z poszukiwania bieżącego rozwiązania),
- miejsce posiada największy współczynnik aktywności funkcjonalnej (wartość wyznaczana w procesie symulacji funkcjonalnej, punkt 4.1.5 rozdziału czwartego),
- miejsce w relacji zależności przepływu danych (punkt 4.1.5 rozdziału czwartego),
- miejsce położone jest najbliżej punktu startowego *Pm* plastra *P*,
- miejsce posiada najgorsze parametry realizacji programowej,
- miejsce posiada najkorzystniejsze parametry realizacji sprzętowej,
- nie przekroczono zasobów sprzętowych,
- osiągnięte zyski spełniają nierówność 4.18.

Algorytm konstrukcji plastra SPMC operuje na miejscach sieci przydzielając do wspólnej grupy wybrane zadania ze względu na zyski czasu pracy SPMC oraz korzyści implementacyjne aktualnego plastra. Nadrzędny proces dekompozycji funkcjonalnej SPMC, przekazuje do algorytmu plastra wybrane miejsce początkowe *Pm,* będące punktem startowym nowego rozwiązania. Warunkiem pracy algorytmu jest wyznaczenie miejsca sąsiadującego plastra spełniającego kryteria kwalifikacji przedstawione wyżej. Następnie badany jest koszt realizacji sprzętowej plastra wzbogaconego o nowe miejsce programowe oraz zysk czasu pracy SPMC przy wstępnej kwalifikacji miejsca *Pnext* do części sprzętowej. Brak spełnionego warunku kosztu lub zysku skutkuje wykluczeniem miejsca *Pnext* z aktualnie prowadzonych analiz. Szkic algorytmu przedstawia rysunek 4.19.

```
Alg.4.2
Zbior.Wykluczen=∅;
P_ACTUAL=P_m;
while( (P_NEXT := wyznacz.miejsce.sasiadujace(S, P_ACTUAL,Zbior.Wykluczen)) != 0) {
  If (Ograniczenia.Systemu(SPMC)>Koszt(P_ACTUAL + P_NEXT)) {
    If Δt(S, P_ACTUAL + P_NEXT)< Δt(S, P_ACTUAL) )then
      P_ACTUAL += P_NEXT;
    else
      Zbior.Wykluczen += P_NEXT;
    end if;
  } else {
    Zbior.Wykluczen += P_NEXT;
  }
}
return(P_ACTUAL);
```

Rysunek 4.19 Algorytm pracy wyznaczania plastra zadaniowego

Proces jest przerywany w chwili, gdy nie można wyznaczyć kolejnego miejsca *Pnext* sprzętowego plastra zadaniowego. Algorytm zwraca aktualne znalezione rozwiązanie do systemu nadrzędnego.

W szczególności, postępowanie pracy algorytmu wyznaczania sprzętowego plastra zadaniowego przedstawiono w trzech punktach:

1. Krok pierwszy.
   a) Dla plastra *P* znajdź sąsiada będącego w relacji sekwencji przepływu sterowania tylko z *Pm*, oznacz miejsce jeśli spełnione są warunki:
      i. miejsce nie jest wykluczone z bieżącego rozwiązania,
      ii. miejsce o najgorszych parametrach realizacji programowej,
      iii. miejsce o najkorzystniejszych parametrach realizacji sprzętowej,
      iv. nie przekroczono zasobów sprzętowych,
      iv. osiągnięte zyski spełniają nierówność 4.18,
   b) Jeśli brak spełnienia warunków 1a)v-vi dla znalezionego miejsca, odrzuć bieżące miejsce (oznakowanie wykluczające miejsce dla bieżącego rozwiązania). Przejdź do kroku nr 1a).
   c) Jeśli brak możliwych do wyznaczenia miejsc sekwencyjnych, przejdź do kroku nr 2.
   d) Jeśli znaleziono dwa lub więcej miejsc równorzędnych ze względu na punkty a)i-iv, przejdź do punktu 3.
   e) Przejdź do punktu 1a).
2. Krok drugi.
   a) Dla plastra *P* znajdź miejsce sąsiadujące *Pm*, będący w relacji współbieżności przepływu sterowania, według klucza:
      i. miejsce nie jest wykluczone z bieżącego rozwiązania,
      ii. miejsce o największym współczynniku aktywności funkcjonalnej,
      iii. miejsce w relacji zależności przepływu danych,
      iv. miejsce położone najbliżej punktu startowego Pm plastra K,
      v. miejsce o najgorszych parametrach realizacji programowej,
      vi. miejsce o najkorzystniejszych parametrach realizacji sprzętowej,
      vii. nie przekroczono zasobów sprzętowych,
      viii. osiągnięte zyski spełniają nierówność 4.18,
   b) Jeśli brak spełnienia warunków 2a)v-vi dla znalezionego miejsca, odrzuć bieżące miejsce (oznakowanie wykluczające miejsce dla bieżącego rozwiązania). Przejdź do kroku nr 1.
   c) Jeśli brak możliwych do wyznaczenia miejsc równoległych, koniec budowy plastra zadaniowego. Zwróć rozwiązanie do pętli głównej algorytmu nadrzędnego.
   d) Przejdź do kroku nr 1.
3. Realizuj przeszukiwanie wszystkie ścieżek równych miejsc do chwili znalezienie najlepszego rozwiązania spełniającego punkty1a)i-iv. W momencie niepowodzenia, wybierz rozwiązanie w sposób losowy. Przejdź do punktu 1a).

Sterowanie dalszymi krokami postępowania algorytmu dekompozycji SPMC, przejmuje algorytm nadrzędny. Rozwiązanie aktualne zostaje zapamiętane w zbiorze lokalnym. Jeżeli koszt realizacji i pracy mikrosystemu SPMC dla aktualnego plastra zadań jest mniejszy od kosztów najlepszego znalezionego

rozwiązania mikrosystemu SPMC, wówczas aktualny plaster zostaje zaakceptowany i staje się plastrem globalnym.

Przykład pracy algorytmu dekompozycji funkcjonalnej dla sprzętowo-programowej mikrostruktury cyfrowej przedstawiono w rozdziale piątym.

4.1.5. Synteza programowa modelu formalnego PNHSMC

Sieć Petriego pozwala na modelowanie zdarzeń realizowanych w sposób sekwencyjny oraz współbieżny. Podejmując próbę implementacji specyfikacji zachowania funkcjonalnego programu opisanego sieciami Petriego, należy rozstrzygnąć problem współbieżności. Każdy procesor ogólnego przeznaczenia przetwarza program w sposób sekwencyjny, więc realizacja współbieżności musi mieć charakter emulacji (symulacji równoległości).
W pracy [Andr03] przedstawiono następującą klasyfikację metod realizacji programowej sieci Petriego:

- metody jednorodne;
  - sekwencyjna,
  - dynamiczna,
- metody z podziałem na cześć sterującą i blok danych;
  - z wykorzystaniem grafu znakowań,
  - z wykorzystaniem tablic behawioralnych,
  - z wykorzystaniem macierzy incydencji,
  - z wykorzystaniem systemu wnioskującego.

Wybór metody implementacji oprogramowania dla mikroprocesorów systemów cyfrowych ma duże znacznie dla czasu pracy całego systemu. Dla celów rozprawy, opracowano nową metodę syntezy programowej sieci Petriego, która bazuje na rozwiązaniach [Misi80]. Wybór algorytmu syntezy programowej sieci Petriego uwarunkowano prostotą implementacyjną programu zapewniając w ten sposób lepszą wydajność pracy procesora (krótszy czas przetwarzania programu) w stosunku do znanych wyników przeprowadzonych testów [Andr03]. Opracowane rozwiązania SPMC optymalizują metodę jednorodnej realizacji sekwencyjnej
[Misi80,Stas03b].

Optymalizacja jednorodnego algorytmu sekwencyjnego emulacji równoległości

Cechą charakterystyczną jednorodnego algorytmu sekwencyjnego jest bez wątpienia wzrost objętości programu, wraz ze wzrostem stopnia komplikacji sieci, zwłaszcza ilości tranzycji. Powoduje to wydłużenie czasu przetwarzania programu wynikowego przez mikroprocesor. O ile problem wzrostu objętości kodu programu nie da się wyeliminować, gdyż wynika on z idei algorytmu, o tyle możliwe jest skrócenie czasu realizacji programu przez mikroprocesor. Wykonane optymalizacje dotyczą:

- ograniczenia liczby sprawdzanych warunków do aktualnie wykonywanych modelów makromiejsc w przypadku przetwarzania hierarchicznej sieci Petriego,

- ograniczenia liczby sprawdzanych warunków w przypadku przetwarzania sekwencji tranzycji.,
- zmiany sposobu wymiany danych program-sprzęt, pobierania danych wejściowych i wysyłania wyników przetwarzania.

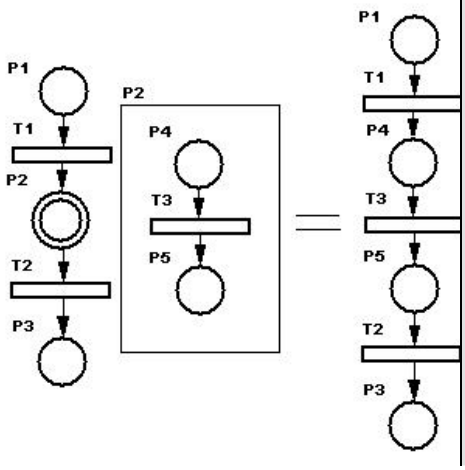
Optymalizacja przetwarzania hierarchii w sieci Petriego

W metodzie jednorodnej sekwencyjnej, konsekwencją syntezy hierarchicznej sieci Petriego jest spłaszczenie struktury modelu do jednego poziomu. W miejsce deklaracji makromiejsca wstawiany jest kod instancjonowanej sieci. Proces „spłaszczania" sieci nie modyfikuje zapisu funkcjonalnego sieci. Jednak, otrzymujemy jedną, płaską sieć Petriego, która składa się z superpozycji (złożenia) miejsc i tranzycji całej sieci. Przykład przedstawia rysunek 4.20.

W wyniku zastosowania algorytmu pierwotnego [Misi80], program wynikowy zawiera instrukcje ciągłej analizy wszystkich tranzycji sieci w celu wykrycia zmiany i właściwego odpalenia gotowej do realizacji tranzycji programowej sieci Petriego.

Czas przetwarzania tak skonstruowanego programu wzrasta wraz ze złożonością sieci.

a)



b)

```
if (isSetVect(trans, 1)) {
   setValueVect(place, 4, 1);
   setValueVect(mp, 1, 1); // aktywuje warunek
włączenia przetwarzania macromiejsca
   setValueVect(trans, 1, 0);
}
if   (isSetVect(mp,1)   {//   warunek   włączenia
przetwarzania macromiejsca
if (isSetVect (trans, 3)) {
   setValueVect (place, 5, 1);
   setValueVect (trans, 3, 0);
}}
if (isSetVect (trans, 2)) {
   setValueVect (place, 5, 0);
   setValueVect (place, 3, 1);
   setValueVect(mp, 1, 0); // deaktywuje warunek
włączenia przetwarzania macromiejsca
   setValueVect (trans, 3, 0);
}
```

Rysunek 4.20 Spłaszczenie hierarchicznej sieci Petriego

W metodzie syntezy programowej SPMC zastosowano rozwiązanie eliminujące konieczność kontroli wszystkich tranzycji sieci. Optymalizacja w przetwarzaniu sieci hierarchicznych polega na pominięciu analizy obiektów składowych makromiejsca, w chwili gdy dane makromiejsce jest nie aktywne. W rezultacie, algorytm programu analizuje zbiór tylko wybranych tranzycji sieci. Dzięki czemu, poprzez skrócenie cyklu przetwarzania programu, czas reakcji mikrostruktury SPMC został skrócony.

Optymalizacja przetwarzania sekwencji tranzycji sieci Petriego

Własnością sieci Petriego specyfikującej sekwencję zdarzeń jest tranzycja mająca jedno miejsce wejściowe i jedno miejsce wyjściowe. Grupę kolejno występujących po sobie takich tranzycji nazwano *sekwencją*. Rysunek 4.21 a) przedstawia przykład opisu sekwencji w sieci Petriego. Realizacja kolejnych tranzycji sekwencji skutkuje przeniesieniem żetonu z jednego miejsca wejściowego do miejsca wyjściowego.

Analizując rysunek 4.21.a.) zauważono, że nie jest konieczne rozpatrywanie gotowości realizacji tranzycji T2 i T3, jeśli żeton znajduje się w miejscu P1 i tranzycja T1 nie jest spełniona. Założenie jest prawdziwe pod warunkiem, że sieć dla miejsc sekwencji będzie przechowywać zawsze jeden znacznik:

$$S = \{p1, p2, ..., pn\}, M_S = 1$$

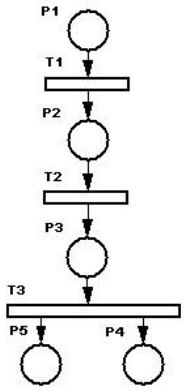
gdzie;

S– zbiór miejsc sekwencji,

Ms– suma znaczników w sekwencji.

Optymalizacja przetwarzania tranzycji w sekwencji polega na pominięciu analizy tranzycji występujących w sekwencji, które nie mogą zostać zrealizowane ze względów formalnych. W implementacji technicznej programu w języku C rozszerzeniu ulega wyrażenia warunkowe *if* o konstrukcję *else* obejmującą kolejne tranzycje sekwencji, rysunek 4.21.b).

a)



b)

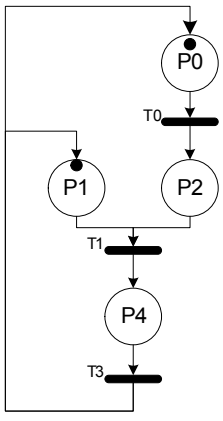
```
if (isSetVect(trans, 1)) {
   setValueVect(place, 1, 0);
   setValueVect(place, 2, 1);
   setValueVect(trans, 1, 0);
} else
       if (isSetVect (trans, 2)) {
          setValueVect (place, 2, 0);
          setValueVect (place, 3, 1);
          setValueVect (trans, 2, 0);
       } else
              if (isSetVect (trans, 3)) {
                 setValueVect (place, 3, 0);
                 setValueVect (place, 5, 1);
                 setValueVect (place, 4, 1);
   setValueVect (trans, 3, 0); }
```

Rysunek 4.21. Fragment sieci Petriego przedstawiający sekwencję tranzycji

Synteza punktów komunikacyjnych program–sprzęt

Synteza programowa SPMC sieci Petriego realizuje automatyczną generację kodu programu na podstawie wejściowej specyfikacji funkcjonalnej w formacie SPNF (sieć Petriego) oraz generuje wewnętrzny (program-sprzęt) i zewnętrzny (program-SOPC) interfejs komunikacyjny SPMC dla części programowej.

W wyniku pracy algorytmu dekompozycji SPMC, główny model mikrostruktury cyfrowej dzielony jest na część programową i sprzętową. Informacje dotyczące podziału składowane są w pliku projektu. W formacie SPNF każdy obiekt modelu, czyli miejsce i tranzycja, otrzymują znacznik *implement* z wartością *h*-implementacja sprzętowa, lub *s*-implementacja programowa. Na podstawie tak skonstruowanego zapisu, algorytm generacji programu C analizuje sieć Petriego zapisując w pliku wyjściowym właściwe funkcje odpowiadające wybranym miejscom i tranzycjom sieci. Przykład kodu C (tylko funkcjonalność, brak konfiguracji procesora) dla sprecyzowanej funkcjonalności opisanej sieciami Petriego przedstawia rysunek 4.22.

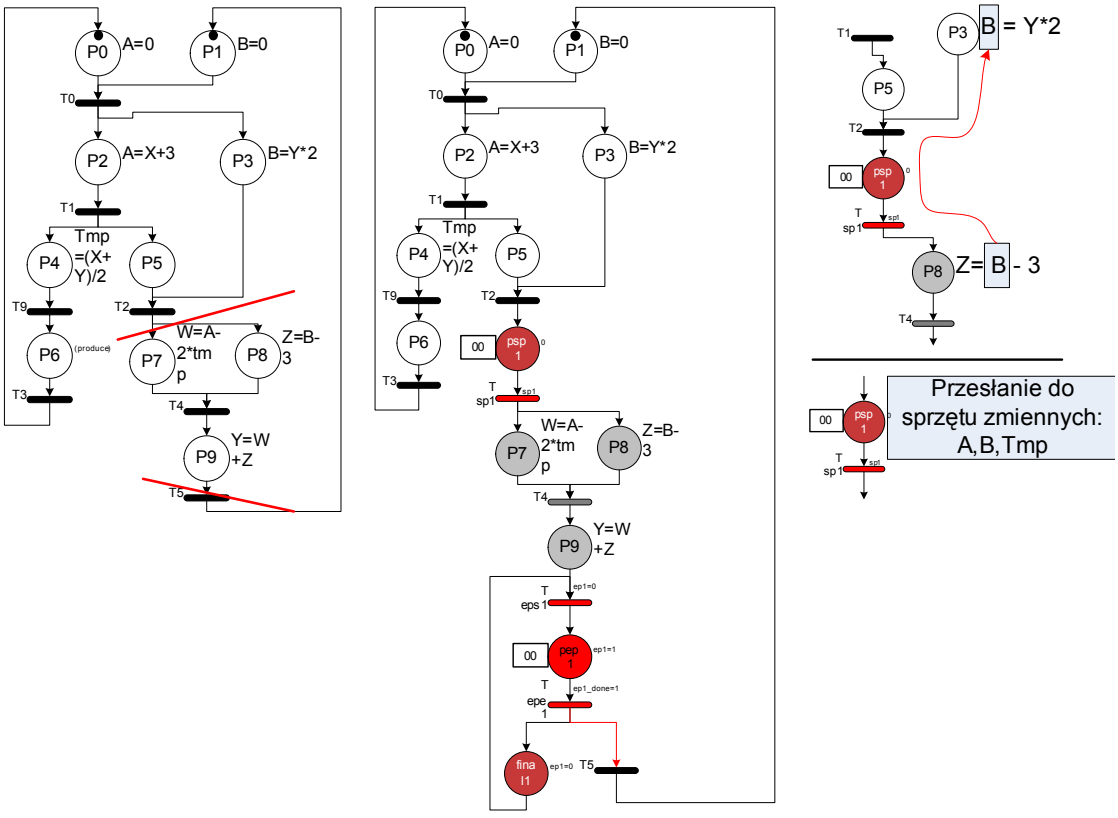



```
… //konfiguracja procesora oraz inne funkcje programu
void init_main() {
   setValueVect(place, 0, 1);
   setValueVect(place, 1, 1);
}
void place_main() {
   if (isSetVect(trans, 0)) {
      setValueVect(place, 0, 0);
      setValueVect(place, 2, 1);
      setValueVect(trans, 0, 0);
   } else
      if(isSetVect(trans, 1)) {
         setValueVect(place, 1, 0);
         setValueVect(place, 2, 0);
         setValueVect(place, 3, 1);
         setValueVect(trans, 1, 0);
      }
   if(isSetVect(trans, 2)) {
      setValueVect(place, 3, 0);
      setValueVect(place, 1, 1);
      setValueVect(place, 0, 1);
      setValueVect(trans, 2, 0); }
}
void trans_main() {
   if((isSetVect(place, 0) ))
     setValueVect(trans, 0, 1);
   else
      if((isSetVect(place, 1) && isSetVect(place, 2)))
        setValueVect(trans, 1, 1);
```

```
    if((isSetVect(place, 3) ))
      setValueVect(trans, 2, 1);
}
void main() {
      init_device();
      init_main();
      while(1) {
            place main();
            setOutput();
            trans main();
      }
}
```

Rysunek 4.22 Przykład kodu C specyfikacji funkcjonalnej reprezentowanej sieciami Petriego

Proces podziału SPMC na podstawie końcowej konfiguracji sieci mikrostruktury cyfrowej dokonuje analizy zależności ścieżki danych dla zdefiniowanych punktów podziału. Wyznaczane są zmienne, które podlegają wymianie w procesie komunikacyjnym. Przykład reprezentuje rysunek 4.23.

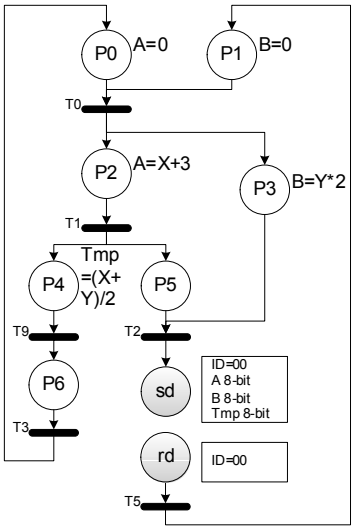


Rysunek 4.23 Podział specyfikacji na część programową i sprzętową oraz wyznaczenie zmiennych do procesu transmisji

Dla rysunku 4.23, algorytm dekompozycji funkcjonalnej SPMC przydzielił do części sprzętowej miejsca P7,P8,P9. Pozostała specyfikacja przypisana została do realizacji programowej. Punktami cięcia zostały oznaczone tranzycie T2 i T5. Analiza DFG specyfikacji SPMC, wyznacza zbiór zmiennych, które muszą zostać poddane synchronizacji w obszarze implementacyjnym programowo-sprzętowej mikrostruktury cyfrowej.

Na podstawie konfiguracji podziału rysunku 4.23, programowa sieć Petriego może zostać zobrazowana jak na rysunku 4.24.

a)



b)

```
<AESD name="projekt1">
<decompositionConfig name="spmc1">
   <point name="pointOut0">
      <sensitive>T2</sensitive>
      <data width="8" sequence="1">A</data>
      <data width="8" sequence="2">B</data>
      <data width="8" sequence="3">Tmp</data>
      <address>0</address>
      <direction>H</direction>
   </point>
   <point name="pointIn0">
      <sensitive>T5</sensitive>
      <address>0</address>
      <direction>S</direction>
   </point>
</decompositionConfig>
<SystemDescription name="test2">
…
</ADES>
```

Rysunek 4.24 Programowa sieć Petriego po procesie podziału, a) reprezentacja graficzna, b) zapis konfiguracji punktów komunikacyjnych program-sprzęt-program w formacie SPNF

Do realizacji programowej zakwalifikowana została wejściowa specyfikacja SPMC z wyłączeniem miejsc sprzętowego plastra zadaniowego. W końcowym etapie opracowywana jest konfiguracja interfejsu wejścia/wyjścia dla części programowej i sprzętowej. Wyniki konfiguracji składowane są w bloku „decompositionConfig" formatu SPNF. Kod zapisu dla przykładu z rysunku 4.26a) przedstawiono na rysunku 4.24b). Dla zdefiniowanych punktów końcowych i startu, generowana jest funkcja obsługi implementująca protokół komunikacyjny SPMC opisany w rozdziale trzecim. Przykład implementacji zapisu konfiguracji podziału w języku C dla procesora Atmel AVR Atmega103 przedstawia rysunek 4.26.

```
void get ssp in pointIn0(){
   sbi(PORTB, S_CLK);
   cbi(PORTB, S_CLK);
   cbi(PORTB, S READY);
   setValueVect(in flags,0, 1);
}
INTERRUPT(SIG INTERRUPT4) {
   uchar_t ident;
   DDRA = 0x00;
   sbi(PORTB, S READY);
   cbi(PORTB, S BUSY);
   while (!bit is set(PINB, H ACK)) {}
   ident = PINA;
    if (ident == 0){
      get_ssp_in_pointIn0();
   }
   DDRA = 0xFF;
}
```

```
void print ssp out pointOut9(){
   while (bit_is_set(PINB, H_BUSY)) {}
   sbi(PORTB, S_BUSY);
   PORTA = wy[0]; // A
   sbi(PORTB, S CLK);
   cbi(PORTB, S CLK);
   PORTA = wy[1]; //B
   sbi(PORTB, S_CLK);
   cbi(PORTB, S_CLK);
   PORTA = wy[1]; //Tmp
   sbi(PORTB, S CLK);
   cbi(PORTB, S CLK);
   cbi(PORTB, S BUSY);
}
…
   if (isSetVect(trans, 2)) {
      print ssp out pointOut0();
   }
```

Rysunek 4.26 Funkcje programu C realizujące proces komunikacji SPMC

4.1.6. Synteza sprzętowa modelu formalnego PNHSMC

Metoda syntezy sprzętowej SPMC modelu układu cyfrowego opisanego sieciami PNHSMC, wykorzystuje opracowania pracy doktorskiej P.Wolańskiego [Wola98], w szczególności metodę syntezy zorientowaną na tranzycje.

Niniejsza rozprawa rozszerza możliwości realizacji modelu opisanego sieciami Petriego modyfikując metodę [Wola98] oraz wprowadzając szereg nowych rozwiązań w procesie syntezy.

Nowatorskim rozwiązaniem dotyczącym syntezy sieci Petriego do języków opisu sprzętu, jest zastosowanie algorytmu optymalizacji zasobów sprzętowych w realizacji hierarchii strukturalnej modelu. Założeniem opracowanego algorytmu jest wsparcie dla specyfikacji modelu formalnego PNHSMC. Natomiast, nowymi elementami syntezy w stosunku do rozwiązań [Wola98], są :

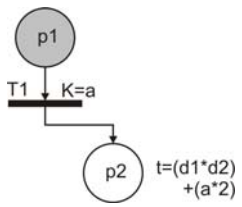
- realizacja sprzętowa instrukcji programistycznych zdefiniowanych w miejscu,
- realizacja konfiguracji podtrzymania lub braku podtrzymania stanu sygnału wyjściowego po usunięciu znacznika z miejsca,
- realizacja wyjść typu Moore'a i Mealye'go zadeklarowanych w miejscu,
- realizacja makromiejsc typu proceduralnego i współdzielonego,
- sprzętowa realizacja wyjątków,
- sprzętowa realizacja historii bez dodatkowych kosztów implementacyjnych.

W dalszej części rozdziału przedstawione zostały wybrane zagadnienia syntezy sprzętowej SPMC sieci Petriego, prezentując nowatorskie rozwiązania rozprawy.

Deklaracja oraz synteza miejsca prostego i jego produktów

Rysunek 4.27.a) przedstawia deklarację dwóch miejsc prostych P1 i P2 oraz tranzycję T1 uwarunkowaną strażnikiem. W miejscu P2 zadeklarowano instrukcję programową przypisaną do produktu 't'. Modyfikacji poddawany jest sygnał(zmienna) „t", w chwili gdy miejsce P2 przetrzymuje znacznik. Ponadto, na podstawie konfiguracji sygnału, sprecyzowanej w zapisie SPNF (rysunek 4.27b), kod XML: *produce ID="REG"*), sygnał 't' jest typu rejestrowego, tzn. zmiana stanu logicznego sygnału 't' może nastąpić wtedy i tylko wtedy, gdy miejsce posiada znacznik oraz wystąpiło aktywne zbocze sygnału zegarowego (synchronizującego). Sygnał „t" zrealizowano jako przerzutnik (lub zbiór przerzutników). Adekwatny kod VHDL przedstawia rysunek 3c).

a)



b)

```
<ADES name="example1">
  <place name="p1">
    <initial>1</initial>
  </place>
  <transition name="T1">
    <sensitive>p1</sensitive>
    <action>p2</action>
    <condition>K=a</condition>
  </transition>
  <place name="p2">
    <produce ID="REG">
t=(d1*d2)+(a*2)
    </produce>
  </place>
```

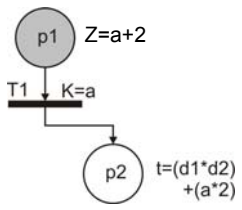
c)

```
T(1)<='1' when (P(1)='1' and K=a) else
'0';

place_p:process(clk,reset)
begin
  if reset='0' then
    P<="00";
  elsif(clk'event and clk='0')then
    if T(8)='1' or P(1)='1' and
T(1)='0' then P(1)<='1' else P(1)<='0'
end if;
    if T(1)='1' or P(2)='1' and
T(3)='0' then P(2)<='1' else P(2)<='0'
end if;
…
end process;

p2_:process(clk,reset,p(2))
begin
  if reset='0' or p(2)='0' then
    t<=0;
  elsif clk'event and clk='1' then
    if P(2)='1' then
        t<=(d1*d2)+(a*2);
…
end process;
```

Rysunek 4.27 Przykład syntezy sprzętowej dwóch miejsc sieci Petriego, a) specyfikacja fragmentu sieci, b) kod SPNF, c) kod VHDL

Analizując zapis funkcjonalny rysunku 4.27.c), można zauważyć, że sygnał 't' zostanie ustawiony w stan początkowy (wyzerowany) w chwili pojawienia się sygnału 'reset' lub w chwili utraty znacznika przez miejsce P2. Model formalny PNHSMC dopuszcza deklarację w miejscu również sygnału typu kombinacyjnego oraz konfigurację podtrzymania stanu wybranego sygnału. Kolejny przykład zobrazowany na rysunku 4.28 przedstawia zmodyfikowaną sieć Petriego z rysunku 4.27. Do miejsca P1 dodano sygnał 'Z'. W zapisie modelu pośredniego SPNF, rysunek 4.28.b), sygnał 'Z' zadeklarowano jako typ kombinacyjny. Dodatkowo, modyfikacji poddana została konfiguracja sygnału 't', dla którego wprowadzono atrybut podtrzymania stanu sygnału po utracie znacznika przez miejsce P2. Wynik syntezy przedstawia rysunek 4.28.c).

a)



b)

```
<ADES name="example1">
…..
   <place name="p1">
     <initial>1</initial>
     <produce
ID="COMB">Z=a+2</produce>
   </place>
   <transition name="T1">
     <sensi-
tive>p1</sensitive>
     <action>p2</action>
     <condi-
tion>K=a</condition>
   </transition>
   <place name="p2">
     <produce ID="REG"
HOLD='1'> t=(d1*d2)+(a*2)
     </produce>
   </place>
…….
</ADES>
```

c)

```
T(1)<='1' when (P(1)='1' and K=a) else '0';

place_p:process(clk,reset)
…
end process;

p1_:process(p(1),a)
begin
  if p(1)='0' then
    Z<=0;
  else
        Z<=a+2;
  end if;
end process;

p2_:process(clk,reset,p(2))
begin
  if reset='0' then
    t<=0;
  elsif clk'event and clk='1' then
    if P(2)='1' then
        t<=(d1*d2)+(a*2);
…
end process;
```

Rysunek 4.28 Przykład syntezy sprzętowej sygnału kombinacyjnego i rejestrowego

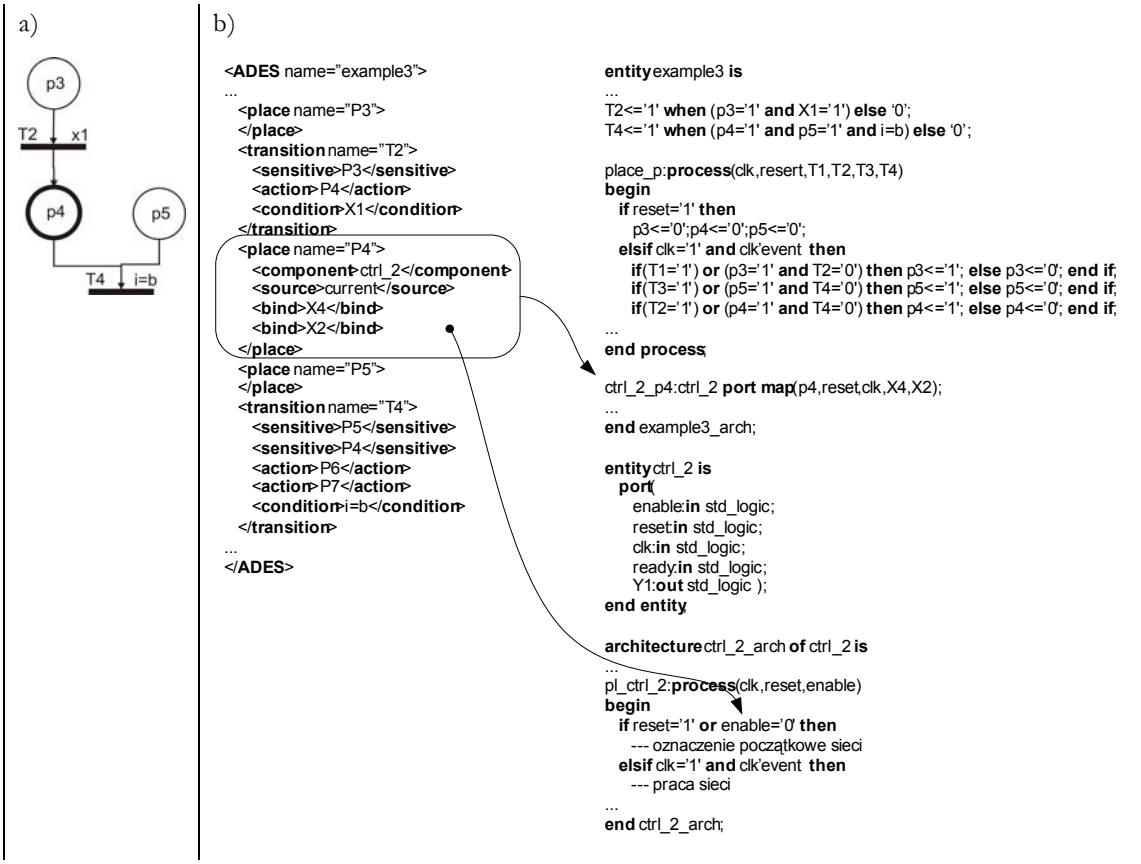
Realizacja sprzętowa sygnału Z jest implementacją układu kombinacyjnego, natomiast realizacja sygnału 't' jest implementacją układu pamiętającego (przerzutnik).

Deklaracja i synteza miejsca współdzielonego.

W modelu formalnym PNHSDM, w jednym maktomiejscu może zostać zainstancjonowany jeden komponent funkcjonalny systemu (*model*). Każdy model składowy systemu może być rozważany jako komponent oraz może zostać wielokrotnie instancjonowany (używany) poprzez deklarację kolejnego obiektu typu makro miejsce.

Z definicji PNHSMC wynika, że model deklarujący makromiejsce współdzielone, w odróżnieniu do miejsca proceduralnego, nie ma dostępu do wewnętrznej struktury instancjonowanej podsieci. Komunikacja (przepływ danych i sterowania) w obrębie instancjonowanego komponentu i systemu nadrzędnego realizowana jest wyłącznie poprzez interfejs wejścia-wyjścia komponentu, przy aktywnym stanie miejsca. Rysunek 4a) przedstawia fragment modelu opisanego sieciami Petriego, gdzie wyróżnia się dwa miejsca proste P3 i P5, tranzycje T2 i T4 oraz makromiejsce współdzielone P4 instancjonujące model Ctrl_2. Gdy miejsce P4 otrzyma znacznik wówczas aktywowana jest cała podsieć Ctrl_2 – rozpoczęcie pracy. W chwili realizacji tranzycji T2, sygnał sterujący „p4" otrzymuje stan

logiczny ‘1’, który przekazywany jest poprzez mapowanie sygnału aktualnego modelu nadrzędnego (p4) z sygnałem lokalnym podsieci Ctrl_2 (enable).



Rysunek 4.29 Przykład specyfikacji, opisu i syntezy makromiejsca współdzielonego

Przy braku aktywnego poziomu logicznego sygnału *enable*, instancjonowana podsieć CTRL_2, rysunek 4.29, zostaje wprowadzona w stan początkowy. Innymi słowy, odpalenie tranzycji T4 przykładu 4.29a, zmienia oznakowanie sieci. Żetony miejsc P4 i P5, zostaną skonsumowane przez tranzycję T4 i przekazane do wszystkich miejsc wyjściowych tranzycji. Brak aktywnego sygnału P4 sieci głównej, wymusza zakończenie pracy podsieci CTRL_2. ~~Zakładając, że u5=1; według definicji 4.12 (makromiejsca) tryb wyjściowy podsieci jest ostatnim ważnym oznakowaniem podsieci, dla którego tranzycja T4 jest gotowa do realizacji.~~

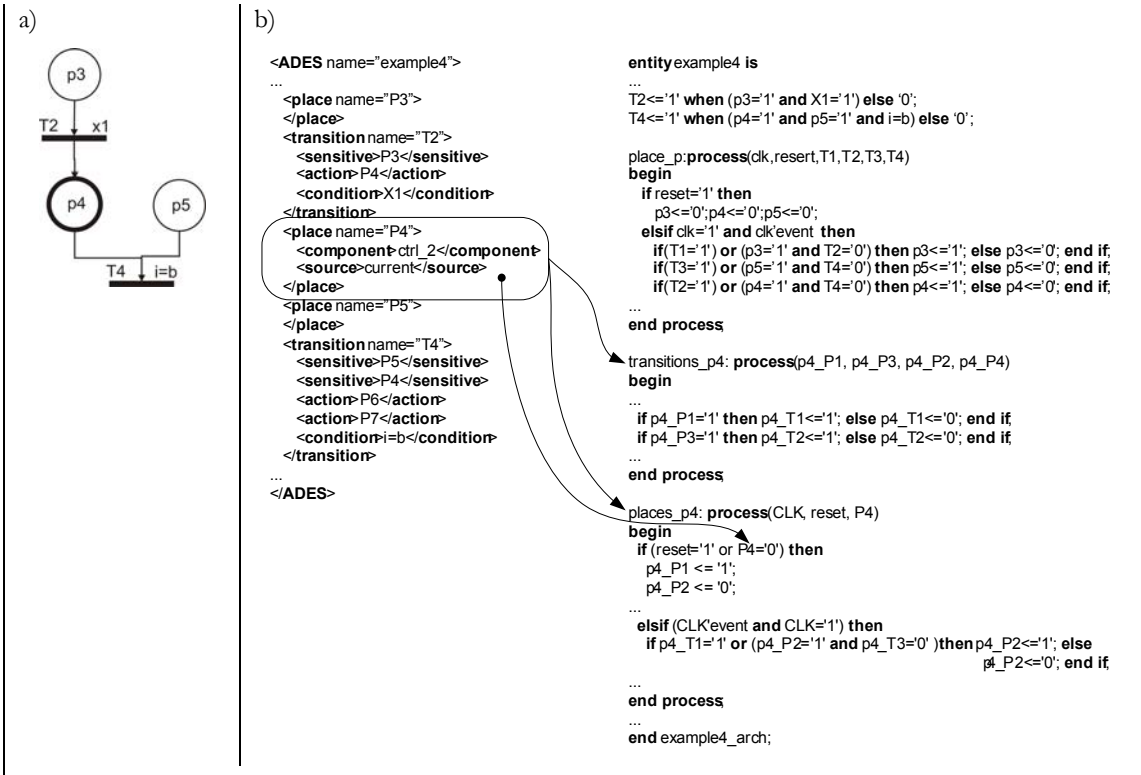
~~Istnieje szeroki wachlarz sposobów modelowania procesu sterowania pracy makromiejsca. Tranzycja, której miejscem wejściowym jest makromiejsce, może być uwarunkowana strażnikiem specyfikującym zakończenie pracy podsiec~~

Deklaracja oraz synteza miejsca proceduralnego

Lokalny interfejs wej/wyj modelu, instancjonowanego jako makromiejsce proceduralne, jest dodany do interfejsu wej/wyj głównego systemu. Przepływ danych i sterowania pomiędzy komponentem i systemem nadrzędnym realizowany jest poprzez współdzielony zbiór sygnałów komponentu i modelu. Nazwy miejsc i tranzycji są wspólne. Jeśli w systemie zadeklarowano więcej niż jedno makromiejsce typu proceduralnego, instancjonujące ten sam model, wówczas niezbędne jest wyróżnienie zbioru interfejsu, miejsc i tranzycji makromiejsca X i makromiejsca Y. Stosowane się następujące schematy zapisu SPNF:

a) nazewnictwo sygnałów:
<*nazwa_makromiejsca*>.<*nazwa_sygnału*>
b) nazewnictwo miejsc i:
<*nazwa_makromiejsca*>.<*nazwa_miejsca*>
<*nazwa_makromiejsca*>.<*nazwa_tranzycji*>

Rysunek 4.30 przedstawia fragment specyfikacji modelu instancjonującego makromiejsce proceduralne, zapis w formacie SPNF oraz wynik syntezy sprzętowej (kod VHDL). W procesie syntezy sprzętowej zapisu SPNF do VHDL, makromiejsce proceduralne P4 realizowane jest w kodzie VHDL za pomocą deklaracji procesu (etykieta „places_p4"). Sygnał lokalny „P4" aktywuje i deaktywuje pracę instancjonowanej podsieci CTRL_2 makromiejsca P4.

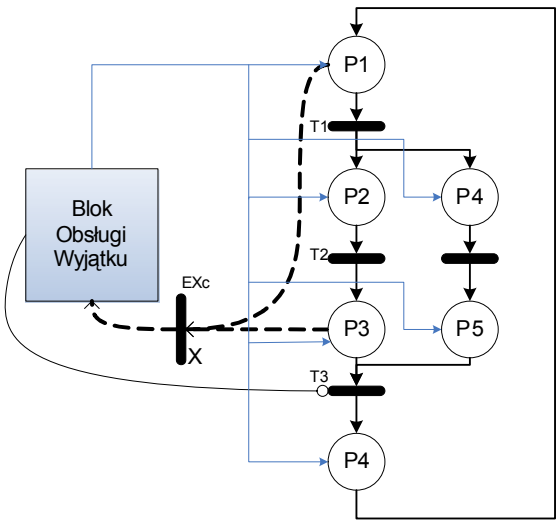


Rysunek 4.30. Przykład specyfikacji, opisu i syntezy makro miejsca proceduralnego

Synteza sprzętowa wyjątków oraz wstrzymania realizacji zadania

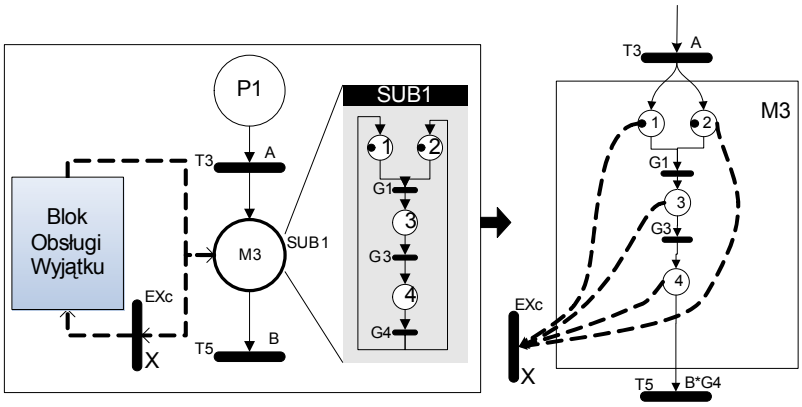
Opracowana na rzecz niniejszej rozprawy metoda syntezy sprzętowej wyjątków modelu PNHSMC jest rozwiązaniem pozwalającym na implementację pełnej funkcjonalności mikrostruktury reprezentowanej modelem PNSHMC.

Deklaracja wyjątku na poziomie wybranej sieci dotyczy tylko wybranych miejsc pracy sieci. Oznacza to połączenie konkretnych miejsc lub jednego miejca sieci $m_i \in M$ łukami wyjątku EA z tranzycją wyjątku $e_i \in E$. Uruchomienie wyjątku może zostać aktywowane tylko dla określonego miejsca pracy sieci. Dla rysunku 4.31., tranzycja wyjątku będzie aktywna w chwili, gdy miejsce P1 lub P3 będą oznakowane. Uruchomienie wyjątku usuwa znacznik ze wszystkich miejsc danej sieci. Natomiast powrót z bloku obsługi wyjątku wprowadza daną sieć w tryb inicjalizujący, czyli realizowane jest oznakowanie początkowe sieci.



Rysunek 4.31 Wyjątek zadeklarowany i wywołany przez miejsce proste sieci Petriego

Deklaracja wyjątku dla makromiejsca oznacza połączenia tranzycji wyjątku ze wszystkimi miejscami instancjonowanej sieci, co ilustruje przykład przedstawiony na rysunku 4.32. Funkcjonalnie, zdarzenie wyjątku przerywa pracę całej sieci danego poziomu, a sterowanie przekazywane jest do funkcji obsługi wyjątku.



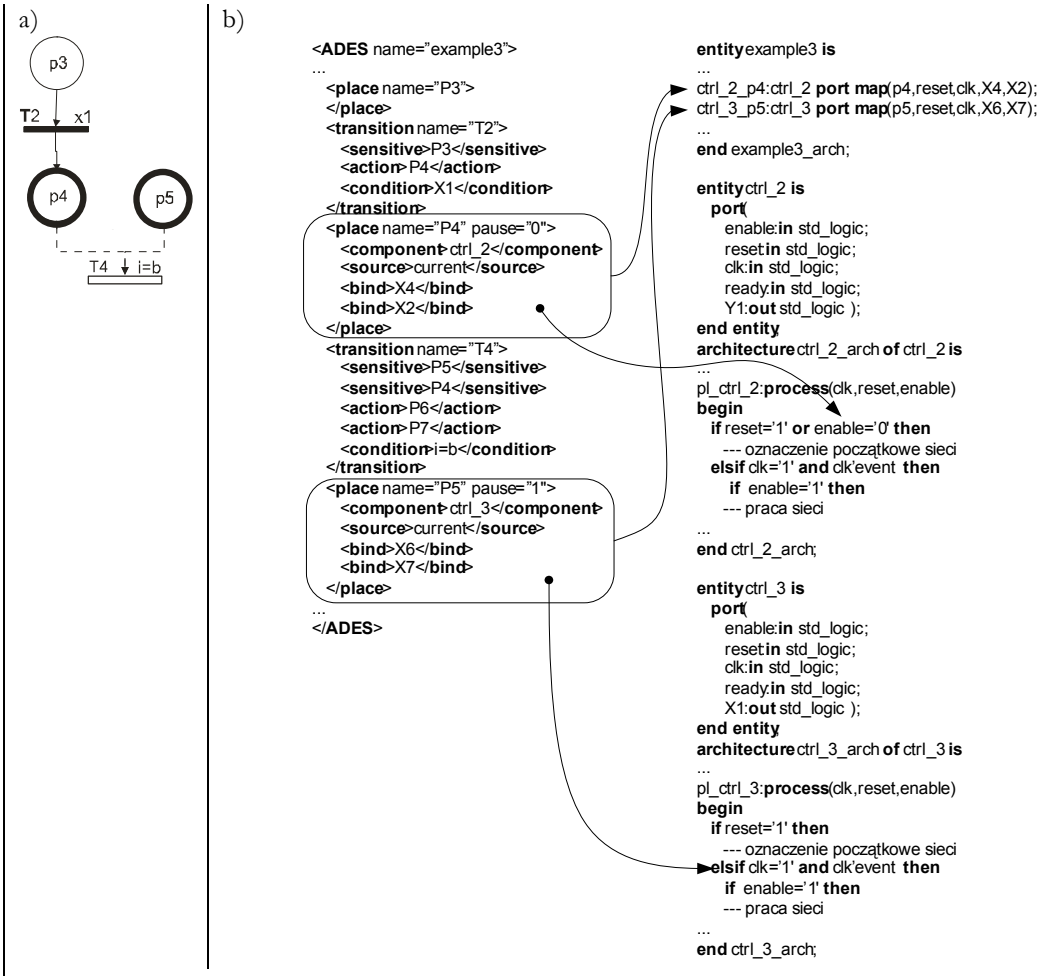
Rysunek 4.32 Wyjątek wywołany przez makromiejsce

Po wykonaniu bloku obsługi wyjątku i ponownego oznakowania makromiejsca, sieć może zachować się według dwu schematów (definicja 4.12 modelu formalnego PNSHMC), w zależności od konfiguracji danej sieci według wzoru 4-16:

a) wznowienie pracy podsieci, $\varrho=1$,

b) ustawienia stanu początkowego podsieci, $\varrho=0$.

Przykład deklaracji wyjątku w zapisie specyfikującym funkcjonalność mikrostruktury cyfrowej oraz realizacji układowej, przedstawia rysunek 4.33.



Rysunek 4.33 Przykład specyfikacji, opisu i syntezy makro miejsca proceduralnego

Optymalizacja obszaru implementacyjnego akceleratora sprzętowego

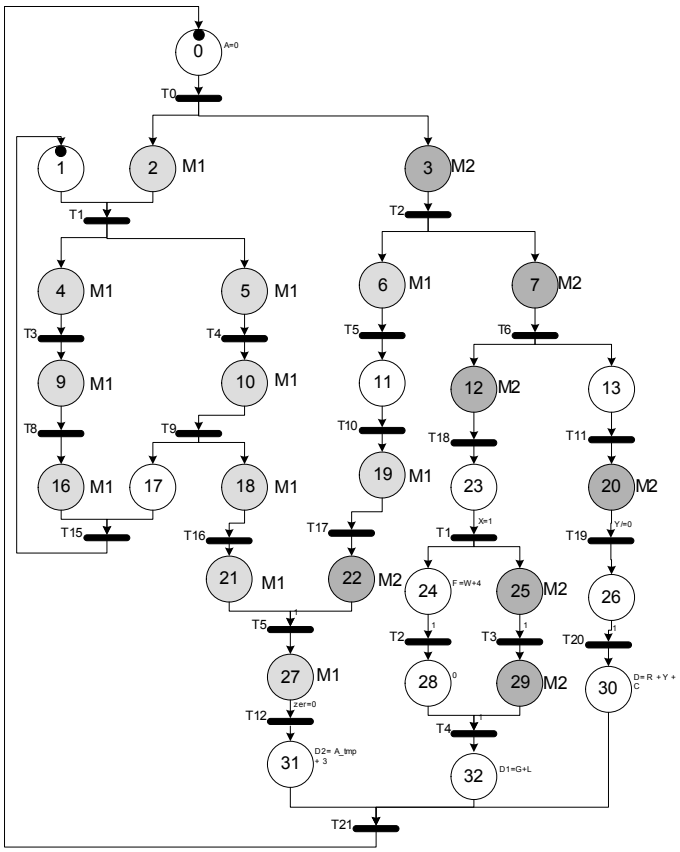
Opracowana metoda syntezy sprzętowej sieci Petriego optymalizuje obszar implementacyjny akceleratora SPMC poprzez wielokrotne wykorzystanie jednego, kilkakrotnie instancjonowanego, sprzętowego bloku funkcyjnego. Podobne techniki wykorzystywane są w syntezie wysokiego poziomu podczas procesu szeregowania ASAP, ALAP [ElKu98] operacji wykonywanych przez wspólną jednostkę operacyjną. Jednak, wskazane techniki nie zostały do tej pory zastosowany w syntezie hierarchicznych sieci Petriego.

Ze względu na charakter prowadzonych rozważań, w szczególności znaczących ograniczeń dotyczących obszaru logiki programowalnej SPMC, podjęto prace naukowo-badawcze, których celem było opracowanie techniki optymalizacji implementacyjnej hierarchicznych sieci Petrigo. Postawiono dwa główne założenia optymalizacji SPMC:

- Optymalizacja wielokrotna. Analizie poddawany jest cały sprzętowy plaster zadaniowy w celu wyznaczenia zbiorów sekwencji makromiejsc możliwych do optymalizacji.

- Wyznaczenie funkcji kosztów optymalizacji. Niezbędne jest określenie kosztu implementacji systemu kontroli i przełączania optymalizowanych miejsc. Istnieje niebezpieczeństwo zwiększenia kosztów implementacyjnych SPMC w sytuacji, gdy suma kosztów logiki przełączania (obszar) jest większy od sumy kosztów implementacji zasobów zadaniowych poddawanych optymalizacji.

Procesowi optymalizacji mogą zostać poddane jedynie makromiejsca typu współdzielonego, instancjonujące ten sam model nie posiadający deklaracji sygnałów z podtrzymaniem oraz makromiejsca nie będące w relacji współbieżności względem siebie. Sieć Petriego specyfikująca zachowanie części sprzętowej poddawana jest szczegółowej analizie formalnej. Wyznaczone zostają wszystkie możliwe sekwencje deklaracji makromiejsc spełniających postawione wcześniej wymagania. Na rysunku 4.34 przedstawiono sieć ze zdefiniowanymi makromiejscami instancjonującymi modele: M1 i M2.



Rysunek 4.34 Sieć hierarchiczna, deklaracje wielu makromiejsc instancjonujących wspólny zasób (model)

Rezultatem analizy jest sześć zbiorów miejsc sekwencji instancjonujących wspólny zasób zadaniowy:

zbiór 1→M1:2,4,9,16

zbiór 2→M1:2,5,10,18,21,27

zbiór 3→M1:6,19,27

zbiór 4→M2:3,22

zbiór 5→M2:3,7,12,25,29

zbiór 6→M2:3,7,20.

Algorytm optymalizuje zbiory zaczynając od grupy o największej liczbie makromiejsc instancjonujących wspólny model. W przypadku równości zbiorów pod względem ilości składników, kosztów realizacji sprzętowej i programowej, czasów pracy – wybór dokonywany jest losowo. Rozważając przykład z rysunku 4.32, w pierwszym kroku algorytmu, dla modelu M1 optymalizacji poddany zostanie zbiór nr 2 (miejsca 2,5,10,18,21,27), natomiast dla modelu M2 zbiór nr 5 (miejsca 3,7,12,25,29). Miejsca zakwalifikowane do optymalizacji zostają usunięte ze wszystkich zbiorów. Dla omawianego przykładu pozostaną cztery zbiory o zredukowanej liczbie miejsc sieci:

zbiór 1→M1:4,9,16

zbiór 3→M1:6,19

zbiór 4→M2:22

zbiór 6→M2:20.

Zbiory zawierające jedno miejsce nie są poddawane optymalizacji. Dla rozważanego zadania z rysunku 4.34, procesowi optymalizacji poddane zostaną zbiory: 2, 5, 1, 3. Optymalizacja SPMC w układzie FPGA realizowana jest za pomocą systemu przełączania współdzielonego komponentu zadaniowego, pomiędzy zdefiniowany zbiór miejsc specyfikacji funkcjonalnej.

Jednym z kluczowych problemów przedstawionej metody optymalizacji zasobów sprzętowych w implementacji sprzętowych sieci Petriego (w szczególności sieci hierarchicznych), jest znalezienie odpowiedzi na pytanie:

Czy wybrana sekwencja przyniesie wymierne korzyści implementacyjne?

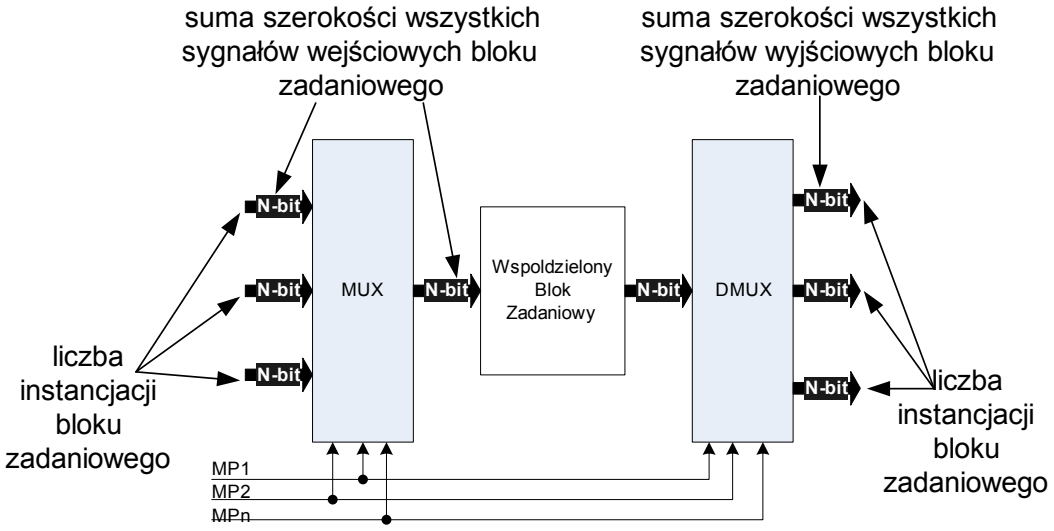
W procesie analizy i estymacji kosztów części sprzętowej, dla każdego makromiejsca określany jest koszt realizacji układowej wyrażony jako liczba konfigurowalnych bloków logicznych CLB układu reprogramowalnego. W celu zobrazowania problemu rozważono następujący przykład.

> Suma zbioru nr 2 wszystkich makromiejsc rysunku 4.34 wynosi 346[CLB]. Ile zasobów logiki reprogramowalnej zostanie zajętych przez system przełączania poszczególnych instancji? Nie jest znany koszt implementacji logiki przełączania (sterowania) współdzielonego modelu M1 w zależności od oznakowania sieci, który zależny jest od interfejsu wejścia (liczba i szerokość sygnałów wejściowych) oraz wyjść (liczba i szerokość sygnałów wyjściowych) instancjonowanego modelu. Jeśli koszt układu przełączania będzie mniejszy od kosztów implementacji wszystkich rozważanych makromiejsc, wówczas optymalizacja przyniesie zysk. W przeciwnym przypadku algorytm optymalizacji zwiększy koszt implementacji części sprzętowej ←sytuacja NIEDOPUSZCZALNA.

Nowatorskim opracowaniem rozprawy jest statyczne szacowanie kosztów implementacji cyfrowego układu przełączania optymalizowanych komponentów zadaniowych (makromiejsc). System przełączania/sterowania współdzielonego bloku zadaniowego zbudowany jest z wejściowego bloku wyboru – multiplekser, oraz wyjściowego bloku wyboru – demultiplekser. Oba bloki sterowane są wspólnym zbiorem sygnałów wyboru, rysunek 4.35. Cały system przełączania specyfikowany jest w języku opisu sprzętu VHDL. Zadanie polega na obliczeniu

ilości bramek logicznych (liczba bloków HMAP i FMAP komórki programowalnej CLB układu Xilinx FPGA) niezbędnych do implementacji multipleksera i demultipleksera.

Proces syntezy logicznej kodu VHDL realizowany jest przez narzędzia komercyjne. W pracy wykorzystano oprogramowanie Xilinx ISE 7.3 oraz MentorGraphic LeonardoSpectrum 2002, gdzie w procesie syntezy logicznej układy kombinacyjne realizowane są z wykorzystaniem bramek 2-wejściowych.



Rysunek 4.35 System przełączania współdzielonego bloku zadaniowego

Wyznaczenie kosztu implementacji multipleksera.

Blok multipleksera składa się z N liczby M-wejściowych bramkek AND oraz W-wejściowej bramki OR. Na wejście bramki AND podawany jest wektor sumy wszystkich sygnałów wejściowych bloku zadaniowego oraz zbiór sygnałów sterujących, więc:

$$A1=INcount + \log(IMPLcount) \qquad \textit{(Wzór 4-19)}$$

gdzie,

A1– liczba bitów wejścia bramki AND,
INcount– suma szerokości wszystkich sygnałów wejściowych bloku zadaniowego (makromiejsca),
IMPLcount – liczba implementacji bloku zadaniowego.

Znajdujemy liczbę dwu-wejściowych bramek AND do realizacji multipleksera. Ponieważ bramkę o N wejściach można rozłożyć na (N-1) bramek 2-wejściowych, dlatego liczba bramek jednej instancji bloku zadaniowego jest równa A1=A1-1.

Aby obliczyć całkowitą liczbę 2-wejściowych bramek AND multipleksera, należy pomnożyć liczbę bramek AND dla realizacji jednej instancji przez liczbę implementacji, więc

$$\mathbf{sumaAND}=A1*IMPLcount \qquad \textit{(Wzór 4-20)}$$

Na wejście bramki OR podawany jest wektor sumy wszystkich sygnałów wejściowych bloku zadaniowego pomnożony przez liczbę implementacji, więc

sumaOR=(INcount*IMPLcount)-1 *(Wzór 4-21)*

Koszt implementacji układowej multipleksera: Kmux=sumaAND+sumaOR, więc:

Kmux=IMPLcount[INcount+log(IMPLcount)-1]+(INcount*IMPLcount)-1

(Wzór 4-22)

Wyznaczenie kosztu implementacji demultipleksera.

Blok demultipleksera składa się z N liczby M-bitowych bramek wyjściowych AND. Pierwszym zadaniem jest wyznaczenie liczby bramek jednej instancji,

A2=OUTcount+log(IMPLcount) *(Wzór 4-23)*

gdzie,

A2– liczba bitów wejścia bramki AND,
OUTcount– suma szerokości wszystkich sygnałów wyjściowych bloku zadaniowego,
IMPLcount – liczba implementacji bloku zadaniowego.

Znajdujemy liczbę dwu-wejściowych bramek AND w realizacji demultipleksera A2=A2-1. Koszt implementacji układowej demultipleksera jest równy liczbie bramek jednej instancji pomnożonej przez liczbę implementacji:

Kdmux= (OUTcount+log(IMPLcount)-1) *IMPLcount *(Wzór 4-24)*

Wyznaczenie funkcji kosztu całkowitego

Całkowity koszt implementacji systemu przełączania i sterowania jest równy sumie realizacji układowej multipleksera i demultipleksera, czyli:

KS=Kmux+Kdmux;

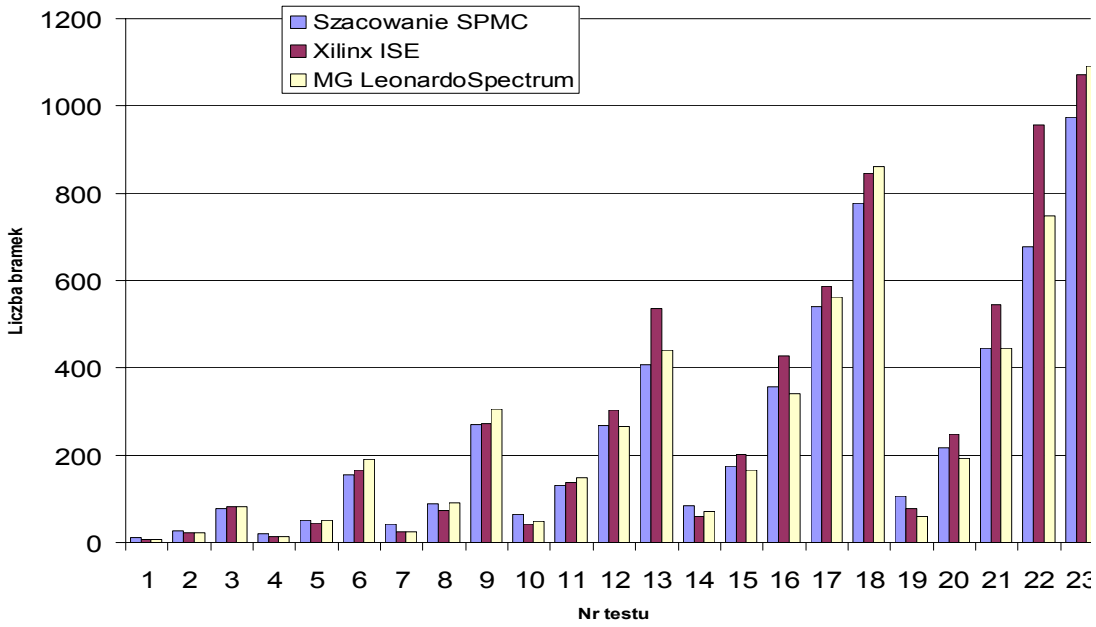
KS= IMPLcount[INcount+log(IMPLcount)-1]+(INcount*IMPLcount)-1+
+(OUTcount+log(IMPLcount)-1) *IMPLcount

KS=IMPLcount(INcount+OUTcount+2log(IMPLcount) -2)+
+(INcount*IMPLcount)-1 *(Wzór 4-25)*

gdzie,

INcount– suma szerokości wszystkich sygnałów wejściowych bloku zadaniowego,
OUTcount– suma szerokości wszystkich sygnałów wyjściowych bloku zadaniowego,
IMPLcount – liczba implementacji bloku zadaniowego.

Poprawność wyznaczonego równania zweryfikowano doświadczalnie poprzez porównanie wyników syntezy logicznej z kosztami obliczonymi na podstawie wzoru 4.25. Badaniu poddano 23 projekty o zróżnicowanej architekturze wejścia/wyjścia oraz zmiennej liczbie instancjacji bloku zadaniowego. Konfrontacji poddano narzędzia komercyjne (Xilinx ISE 7.1sp3 oraz MentorGraphics LeonardoSpectrum 2002b) realizujące proces syntezy logicznej badanego układu przełączania oraz wzór 4.25 estymacji SPMC opracowany na rzecz rozprawy. Wyniki prezentuje rysunek 4.36.



Rysunek 4.36 Wyniki konfrontacji estymacji statycznej SPMC zasobów logiki FPGA systemu przełączania z wynikami implementacji

Wyniki wyznaczonego wzoru są akceptowalne, szczególnie zważając na fakt rozbieżności syntezy logicznej produktów komercyjnych firmy Xilinx i MentorGraphics.

Nowatorski algorytm syntezy sprzętowej sieci Petriego, realizujący optymalizację implementacyjną akceleratora SPMC, posługuje się zależnością warunkującą redukcję kosztów logiki reprogramowalnej FPGA, wyrażoną nierównością:

$$KS<(IMPLcount-1)*KMP \qquad (Wzór\ 4\text{-}26)$$

gdzie,

KS– koszt całkowity systemu przełączania,
KMP– koszt implementacji jednego bloku zadaniowego (makromiejsca),
IMPLcount – liczba makromiejsc poddawanych optymalizacji.

4.1.7. Metoda SMPC w projektowaniu systemów cyfrowych

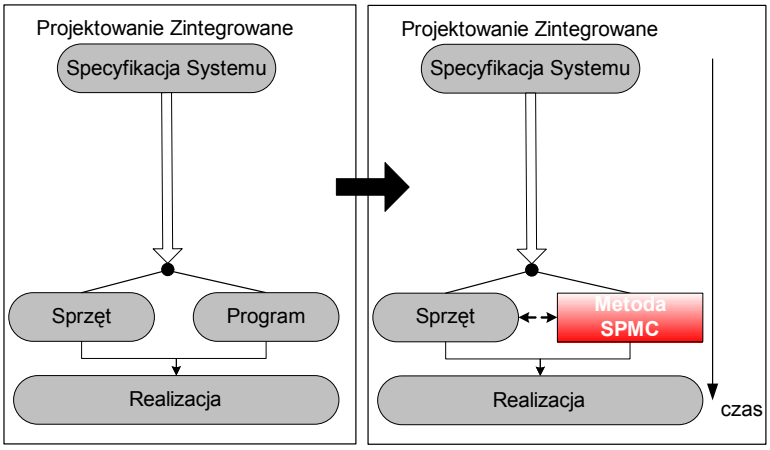
Proces projektowy ściśle zintegrowanej sprzętowo-programowej mikrostruktury cyfrowej SPMC dotyka również problemów sprzętowo-programowego projektowania zintegrowanego. Za wspólną przestrzeń badawczą można uznać algorytm dekompozycji funkcjonalnej, który zarówno w metodologii projektowania zintegrowanego oraz SMPC jest zadaniem krytycznym ze względu na wynik realizowanego projektu.

Metodologia projektowania zintegrowanego mikrosystemów cyfrowych realizuje swoje zadania na poziomie specyfikacji systemu. W procesie dekompozycji funkcjonalnej, która jest częścią procesu syntezy poziomu systemu, realizowana jest analiza funkcjonalna/formalna oraz poszukiwanie architektury realizacji fizycznej systemu cyfrowego. Na podstawie specyfikacji wejściowej oraz zadanych wymagań (czasowych, finansowych, użytkowych i innych) opis zachowania systemu dzielony jest na moduły realizowane odpowiednio przez procesory (ogólnego przeznaczenia, DSP, inne) - część programowa (ang. Software) oraz

układy cyfrowe współpracujące z procesorami (w tym układy programowalne ASIC, reprogramowalne FPGA, układy peryferyjne) - część sprzętowa (ang, Hardware). Proces dekompozycji funkcjonalnej kończy się sukcesem w momencie zaspokojenia określonych wymagań realizowanego systemu (definiowanych przez projektanta).

W metodologii zintegrowanego projektowania systemu sprzętowo-programowego, poszukiwanie końcowej architektury systemu cyfrowego oraz właściwej alokacji zadań w zakresie sprecyzowanych ograniczeń, zobligowane jest jedynie do spełnienia warunku wystarczającego realizacji projektu, nie koniecznie najlepszego. W obszarze niepoddanym eksploracji analitycznej (algorytmicznej) dekompozycji systemowej, wskazane jest zastosowanie metody niższego rzędu, zdolnej w obrębie zdefiniowanego obszaru znaleźć najlepsze rozwiązanie w pełni wykorzystując dostępne zasoby sprzętowe SOPC.

Zintegrowane projektowanie sprzętowo-programowej mikrostruktury cyfrowej SPMC może doskonale uzupełniać metodologię zintegrowanego projektowania systemów sprzętowo-programowych (ang. hardware software codesign) w obszarze projektowania procesorowego/nisko-poziomowego. Rysunek 4.37 przedstawia punkt integracyjny SPMC i projektowania zintegrowanego.



Rysunek 4.37 Metoda SPMC w zintegrowanym projektowaniu systemowym

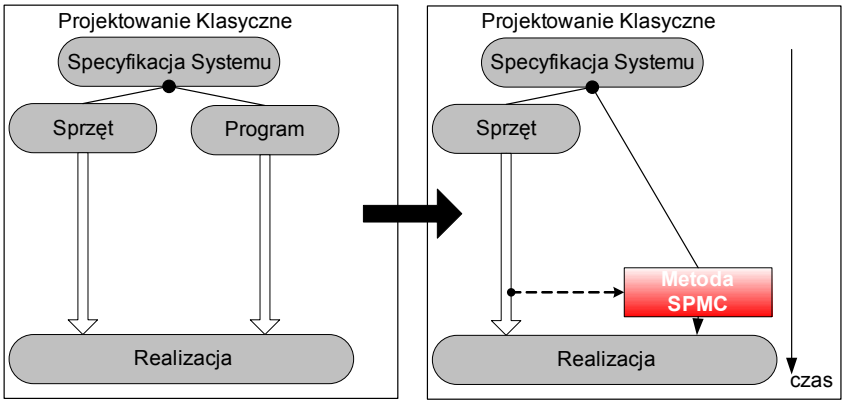
Ścieżka projektowa SPMC, w propozycji niniejszej pracy, jest kolejnym etapem w zintegrowanym projektowaniu sprzętowo-programowego systemu cyfrowego. Rozważane są dwa scenariusze realizacji projektu z wykorzystaniem metodologii SPMC:

1. Pomyślne zakończenie dekompozycji systemowej. Przeprowadzenie etapu projektowania SPMC w celu poprawy parametrów pracy systemu SOPC, np. poprzez zmniejszenie czasu odpowiedzi bloku sterowania na zgłaszane żądania obsługi.

2. Brak pomyślnego zakończenia dekompozycji systemowej ze względu na niską wydajność pracy procesora ogólnego przeznaczenia. Na wejście projektowania SPMC podawana jest ostatnia najkorzystniejsza konfiguracja przydziału zadań dla procesora (C) i logiki programowalnej (HDL). Proces translacji tłumaczy opis heterogeniczny na homogeniczny zapis modelu pośredniego SPMC. Zadaniem

metody SPMC jest szczegółowa analiza i podjęcie próby spełnienia postawionych założeń pracy CPU w mikrosystemie cyfrowym SOPC.

Proponowana metoda projektowania SPMC może znaleźć również zastosowanie w projektowaniu klasycznym. Ogólny schemat projektowania klasycznego oraz integracji metody SPMC przedstawia rysunek 4.38.



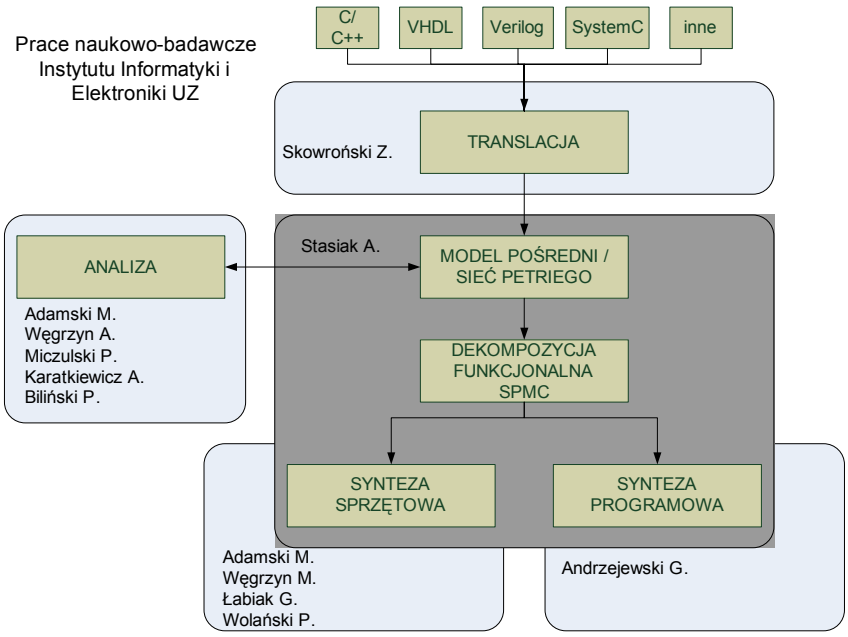
Rysunek 4.38 Metoda SPMC w projektowaniu klasycznym

Proces projektowy inicjalizowany jest wyborem architektury systemu cyfrowego oraz przydziałem zadań dla realizacji sprzętowej i programowej. W procesie projektowym obie specyfikacje (programowa i sprzętowa) realizowane są w sposób rozłączny względem siebie. Dostępne narzędzia CAD wspomagają projektanta w procesie symulacji, weryfikacji, syntezy i implementacji fizycznej specyfikacji sprzętowej i programowej. Wyniki poddawane są weryfikacji w systemie prototypowania, co w rezultacie przedkłada się na podsumowanie badań omówionych w rozdziale pierwszym (punkt 1.1). Uzupełnienie metodologii klasycznej o metodę projektowania SPMC może wspomóc rozwiązanie szeregu problemów projektowania klasycznego, dotyczących m.in.: niskiej wydajności pracy procesora, problemów z przeniesieniem wybranych zadań programowych do części sprzętowej, zapewnieniem wydajnej i taniej pod względem zasobów FPGA komunikacji typu program-sprzęt. Dodatkowym atutem jest automatyzacja całego procesu SPMC oraz osiągnięcie pewnego zysku, tzn. metoda projektowania mikrostruktury SPMC pozwala na osiągnięcie lepszych wyników realizacji w porównaniu z aktualnymi rozwiązaniami lub w najgorszym przypadku zachowanie istniejącego stanu.

4.1.8. Integracja opracowanych rozwiązań z pracami badawczymi innych grup naukowych

W grupie naukowej Instytutu Informatzki i Elektroniki Uniwersytetu Zielonogrskiego, której członkiem jest autor, prowadzone są badania w sferze specyfikacji formalnej systemu i mikrosystemu cyfrowego, syntezy programowej i sprzętowej sieci Petriego, analizy formalnej sieci Petriego oraz realizacji układowej sieci Petriego w strukturach PLD. Zagadnienia naukowe przeprowadzonej rozprawy stanowią istotny etap prac zespołu badawczego. W pracy wykorzystano rozwiązania zaczerpnięte z prac [Wola98,Andr03,Skow00] wprowadzając udoskonalenia i modyfikacje opisane szeroko w rozdziale czwartym. Tematyka

oraz opracowania o charakterze naukowo-yechnicznym niniejszej pracy zintegrowane zostały z istniejącymi opracowaniami grupy naukowej IIE. Rysunek 4.39 przedstawia umiejscowienie zrealizowanych zadań technicznych i naukowych na tle prac innych członków grupy IIE.



Rysunek 4.39 Schemat poglądowy prac prowadzonych w Zakładzie Inżynierii Komputerowej Instytutu Informatyki i Elektroniki Uniwersytetu Zielonogórskiego

Wyniki niniejszej pracy w pełni uzupełniają dotychczasowe osiągnięcia zespołu IIE zespalając dotychczas przeprowadzone prace w domeny sieci Petriego w jedną całośc. Niniejsza rozprawa wypełnia lukę w sferze prowadzonych badań naukowych IIE, wprowadzając:

- algorytm dekompozycji funkcjonalnej sieci Petriego,
- uniwersalny pod względem formalnym model pośredni sieci Petriego, przygotowany do opisu zarówno programowych jak sprzętowych sieci Petriego,
- sformalizowany formatu zapisu sieci Petriego,
- optymalizacje oraz nowe metody syntezy programowej i sprzętowej sieci Petriego,

Szczególnie istotnym pierwiastkiem, wzbogacającym naukowy dorobek IIE, jest nowa metoda projektowania sprzętowo-programowych mikrostruktur cyfrowych.

TRANSLACJA

W procesie projektowania systemowego, a w szczególności w procesie projektowania procesorowego, niezbędne jest posługiwanie się modelem formalnym specyfikacji systemu lub urządzenia. Specyfikacja wejściowa przedstawiona za pomocą dowolnego formatu elektronicznego, poddawany jest na wstępie translacji [Skow00] do formy pośredniej reprezentowanej za pomocą interpretowanych sieci Petriego. Dodatkowo, przeprowadzana jest analiza zapisu funkcjonalnego specyfikacji wejściowej pod względem zrównoleglenia zdażeń opisanych w sposób naturalny sekwencyjny. W efekcie pozwala to na przyspieszenie wykonania zdefiniowanego zbioru zadań. Ponadto, w procesie translacji [Skow00] wynikowa sieć Petriego wyposażana jest mechanizmy

zapewniające zgodność wyników symulacji funkcjonalnej danego modelu pośredniego w dowolnym symulatorze sieci Petriego.

ANALIZA

Operowanie w procesie projektowym na modelu pośrednim niepoddanym analizie formalnej i optymalizacji, można porównać do projektowania układów kombinacyjnych lub sekwencyjnych z pominięciem procesu minimalizacji funkcji logicznych i stanów maszyny FSM. Prace [Adam99,Adam86,Micz06,Węgr03] pozwalają na precyzyjną analizę formalną sieci oraz minimalizację przestrzeni stanów pracy sieci Petriego. Dzięki temu, wynikiem procesu dekompozycji funkcjonalnej SPMC będzie architektura sprzętowo-programowa pozbawiona nadmiarowych zadań (minimalizacja stanów) oraz błedów funkcjonalnych (analiza formalna) niemożliwych lub trudnych do wykrycia w procesie symulacji funkcjonalnej. Ponadto, aktualnie prowadzone prace [BuAd06] prognozują minimalizację przestrzeni stanów sieci poprzez heurystyczne wyznaczenie metody kodowania stanów sieci (minimalizacja zasobów sprzętowych i programowych).

SYNTEZA SPRZĘTOW

Praca [Węgr98] dostarczaja rozwiązań dotyczących realizacji układowej sieci Petriego poprzez implementację sieci bezpośrednio do struktur FPGA [Węgr98] lub poprzez automatyczną generację i zapis maszyny stanów w języku HDL na podstawie specyfikacji wysokiego poziomu. Natomiast, opracowania przedstawione w [Wola98] pozwalają na pośrednią implementację sieci Petriego do układów PLD poprzez syntezę sieci do języka opisu sprzętu VHDL. Wybrane zagadnienia rozprawy [Wola98] zostały wykorzystane i udoskonalone w niniejszej pracy, rozdział czwarty.

SYNTEZA PROGRAMOWA

Opracowany model formalny programowej sieci Petriego [Andr03] był podstawą do zdefiniowania modelu formalnego sprzętowo-programowej jednostki SPMC. Wprowadzono szereg modyfikacji i nowych cech modelu w porównaniu do [Andr03], które wynikają z charakteru funkcjonowania mikrostruktury sprzętowo-programowej. Dla potrzeb rozwiązań SPMC modyfikacji poddano istniejącą metodę syntezy programowej sieci Petriego (rozdział 4.1.5). Badania nad opracowaną metodą syntezy programowej sieci Petriego SPMC skonfrontowano z wynikami prac [Andr03].

W Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego prowadzone są prace nad zintegrowanym systemem projektowania systemów, mikrosystemów i układów cyfrowych specyfikowanych za pomocą sieci Petriego, który efektywnie wykorzysta przedstawione powyżej opracowania.

ROZDZIAŁ PIĄTY

5. Weryfikacja opracowanych rozwiązań oraz przeprowadzone eksperymenty

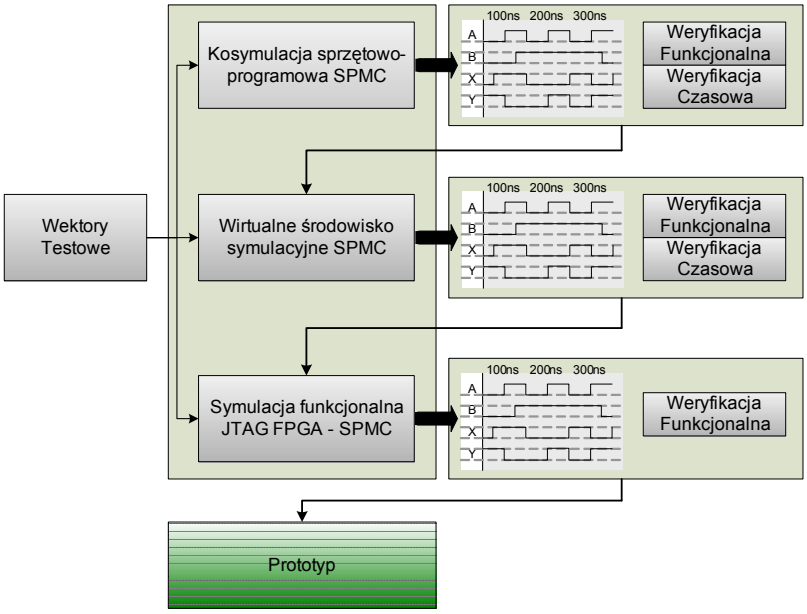
Rozdział poświęcony jest ilustracji wybranych przykładów implementacji sterowania i przetwarzania, które w mikrosystemach cyfrowych wykonywane są przez wbudowany mikroprocesor. Dokonano porównania wyników czasu przetwarzania zadań przez standardowy mikroprocesor oraz sprzętowo-programową mikrostrukturę cyfrową. Testom wydajności i jakości poddano zarówno algorytmy syntezy programowej, sprzętowej jak i algorytm dekompozycji funkcjonalnej SPMC. Zaprezentowano przykład realizacji prostego zadania z wykorzystaniem metody SPMC obrazując wpływ wolnej logiki reprogramowalnej systemu SOPC (w układzie FPGA) na wzrost wydajności pracy mikrostruktury cyfrowej. Przedstawiono wyniki procesu syntezy programowej i sprzętowej oraz zbiorcze rezultaty metody SPMC.

5.1. Weryfikacja funkcjonalna rozwiązań SPMC

Weryfikację funkcjonalną procesu projektowego SPMC, w tym dekompozycji funkcjonalnej, przeprowadzono na trzy sposoby. Pierwszy polega na weryfikacji pracy mikrostruktury cyfrowej specyfikowanej sieciami Petriego po zakończonym procesie dekompozycji funkcjonalnej SPMC. Na rzecz rozprawy wykonany został kosymulator sprzętowo-programowych sieci Petriego CoSPeN (załącznik B) realizujący proces kosymulacji z zachowaniem semantyki sieci Petriego oraz modelu formalnego mikrostruktury cyfrowej PNHSMC. Koniecznością wykonania nowego symulatora był po pierwsze brak symulatorów realizujących kosymulację sprzętowo-programowej sieci Petriego, po drugie koszty zakupu licencji [bige], po trzecie brak możliwości integracji ze wcześniej opracowanymi rozwiązaniami [Skow00].

Drugim sposobem weryfikacji pracy SPMC było uruchomienie gotowego rozwiązania w opracowanym wirtualnym środowisku projektowym.

Trzecia technika weryfikacji funkcjonalnej SPMC dotyczyła przeprowadzenia procesu symulacji funkcjonalnej SPMC z wykorzystaniem opracowanego na rzecz rozprawy symulatora JTAG-SIM (załącznik D), który pozwala na symulację mikrostruktury SPMC po implementacji w układzie FPGA. Rysunek 5.1 przedstawia zastosowane techniki w połączeniu ze wspólnym źródłem wektorów testowych.



Rysunek 5.1. Weryfikacja funkcjonalna SPMC

Wyniki trzech procesów symulacji weryfikowane były pod względem funkcjonalnym.

5.1.1. Weryfikacja funkcjonalna SPMC poprzez kosymulację sprzętowo-programowej sieci Petriego

Po zakończeniu procesu dekompozycji funkcjonalnej SPMC, niezbędna jest weryfikacja poprawności podziału, a w konsekwencji pracy opracowanego rozwiązania sprzętowo-programowej mikrostruktury cyfrowej – w szczególności zależności czasowych. Weryfikacja dokonywana jest poprzez czasową kosymulację programowo-sprzętową podzielonej sieci Petriego. W procesie kosymulacji wymagane jest podanie tego samego wektora testowego, na podstawie którego przeprowadzono walidację modelu pośredniego. Dopuszczalne jest również przeprowadzenie procesu kosymulacji ze zbiorem wektorów testowych, odpowiadających najbardziej spodziewanej rzeczywistej pracy SPMC. Celem jest określenie parametrów czasu pracy SPMC na tle szacowanego w sposób statyczny (w procesie dekompozycji funkcjonalnej) zwiększenia wydajności pracy mikrosystemu cyfrowego. W pracy, *wydajność* zdefiniowano jako czas potrzebny na wykonanie pełnego cyklu zadaniowego, czyt. wszystkich zadań specyfikacji funkcjonalnej X przetwarzanej przez mikrosystem SPMC, w porównaniu do czasu wykonania programu X przez wbudowany mikroprocesor (bez akceleracji).

Wynikiem procesu dekompozycji specyfikacji funkcjonalnej mikrostruktury cyfrowej są dwa zbiory sieci Petriego: zbiór programowy i zbiór sprzętowy. Model

pośredni zapisany w postaci elektronicznej, w formacie SPNF, przechowuje wszystkie informacje niezbędne do przeprowadzenia procesu kosymulacji w opracowanym na rzecz rozprawy symulatorze sprzętowo-programowej sieci Petriego CoSPeN. Symulator dokonuje analizy zapisu sieci uwzględniając znaczniki alokacji IMPLEMENT oraz punkty podziału sieci na część programową i sprzętową. Tranzycjom podziału przydzielany jest czas, wynikający z kosztów realizacji komunikacji program-sprzęt lub sprzęt-program, wyznaczony według wzoru 3-2, 3-3. Sprzętowo-programowa sieć Petriego charakteryzuje się parametrami opóźnień czasowych i kosztów implementacji, adekwatnie do przydziału implementacyjnego miejsca. Dodatek B prezentuje symulator CoSPeN realizujący proces kosymulacji przykładowej sprzętowo-programowej sieci Petriego.

Wyniki kosymulacji sprzętowo-programowej sieci Petriego specyfikującej mikrosystem SPMC, dostarczają wzorców pracy mikrostruktury cyfrowej niezbędnych do weryfikacji wyników symulacji funkcjonalnej SPMC uzyskanych w środowisku wirtualnym mikrosystemu SPMC.

5.1.2. Weryfikacja funkcjonalna SPMC z wykorzystaniem środowiska wirtualnego

Opracowane na rzecz rozprawy wirtualne środowisko prototypowania i weryfikacji mikrosystemu SPMC, zbudowane zostało na bazie modelu IP CORE mikroprocesora AVR Atmega103 oraz komponentu sprzętowego HDL w postaci komponentu. Symulacja czasowa układu cyfrowego odzwierciedla jego rzeczywistą pracę w układzie FPGA, zapewniając zachowanie czasowych parametrów implementacyjnych z dokładnością 98% [xili06]. W pracy wykorzystano model syntezowalnego mikroprocesora RISC AVR Atmega103 ze względu na licencję użytkowania (GPL, ang.General Public License), parametrów pracy oraz stan zaawansowania prac nad rdzeniem procesora. Testy przeprowadzono za pomocą dwu symulatorów języków opisu sprzętu: MentorGraphics ModelSim [ment06], Aldec Active-HDL [alde06]. Wykonane środowisko wirtualnego prototypowania zostało opisane w języku VHDL gwarantując niezależność od dostawcy symulatora HDL.

Przygotowanie nowej realizacji mikrosystemu SPMC do współsymulacji programowo-sprzętowej oraz weryfikacji czasowej sprowadza się do wprowadzenia plików wynikowych procesu projektowego SPMC do właściwej struktury katalogów projektu środowiska wirtualnego. Środowisko jest gotowe do natychmiastowego uruchomienia procesu symulacji wirtualnej. Prezentację procesu kosymulacji czasowej z wykorzystaniem środowiska wirtualnego przedstawiono w dodatku C.

5.1.3. Weryfikacja SPMC w rzeczywistym układzie FPGA

W ramach rozprawy opracowano oprogramowanie pozwalające na przeprowadzenie rzeczywistej weryfikacji i testy zaprojektowanego mikrosystemu SPMC już po implementacji w układzie FPGA. Wykonane oprogramowanie JTAG-SIM może współpracować z komercyjnym oprogramowaniem Xilinx ISE lub Altera Quartus2. Testy funkcjonalne mikrostruktury SPMC przeprowadzono z

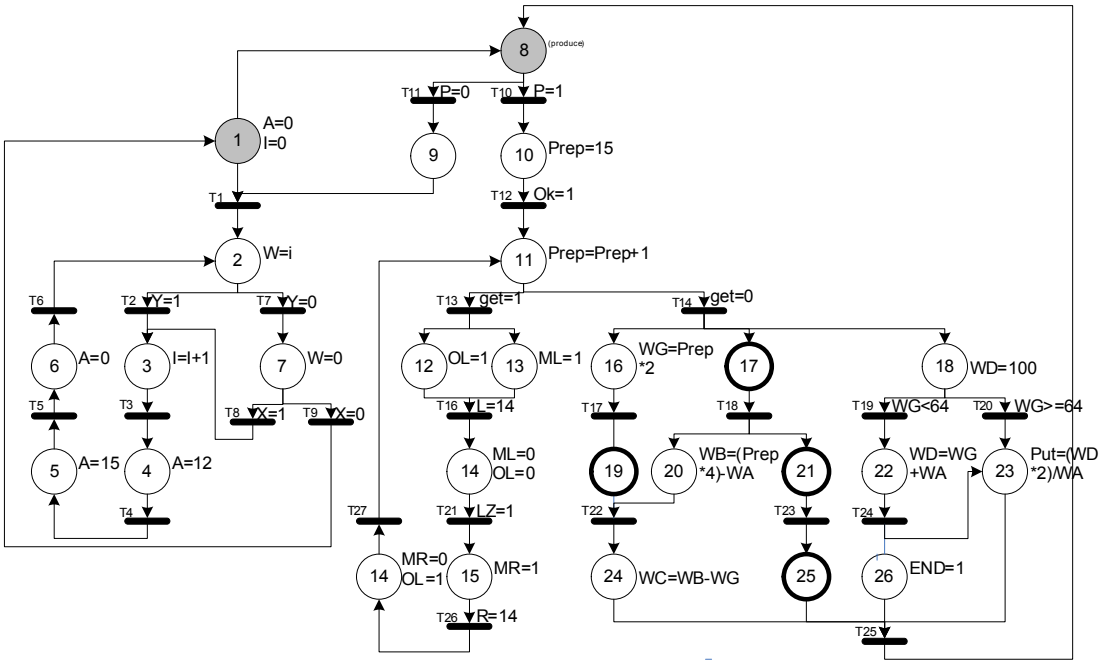
wykorzystaniem układów Spartan II oraz Spartan IIE w obudowie PQ208. W dodatku D zamieszczono możliwe konfiguracje i implementacje kontrolera TAP oraz przykład procesu symulacji.

5.2. Eksperyment przedstawiający proces dekompozycji funkcjonalnej SPMC

Eksperyment przeprowadzono z następującymi parametrami/ograniczeniami mikrostruktury SPMC:

- mikroprocesor AVR Atmega103 taktowany częstotliwością 12MHz,
- system SOPC zbudowano w układzie Xilinx SpartanIIE XC2S200E-6pq208,
- określona liczba wolnych zasobów sprzętowych Zs (zmienna dla poszczególnych testów),
- zdefiniowana specyfikacja SPMC (rysunek 5.17),
- poszukiwanie konfiguracji SPMC w celu zmniejszenia obciążenia obliczeniowego mikroprocesora mikrosystemu cyfrowego,
- dla procesora AVR Atmega103 wyznaczono współczynniki: Kps=1, Ksp=3, K_{INT}=25 (rozdział 3); gdzie KINT uzależniony jest od czasu przejścia mikroprocesora do funkcji obsługi przerwania, wartości współczynników Ksp, Kps i Kint wyrażone są liczbą cykli zegara mikroprocesora oraz mogą zostać wyznaczone w sposób doświadczalny lub analityczny.

W celu zobrazowania pracy algorytmu dekompozycji SPMC oraz analizy zysku rozumianego jako skrócenie czasu pracy mikrosystemu cyfrowego, do eksperymentu wybrano jeden z modeli testowych opracowany we współpracy z firmą Kronopol-Polska [kron06]. Rysunek 5.15 przedstawia specyfikację funkcjonalną sprzętowo-programowej mikrostruktury cyfrowej SPMC, opisaną sieciami Petriego.



Rysunek 5.17 Specyfikacja wejściowa SPMC

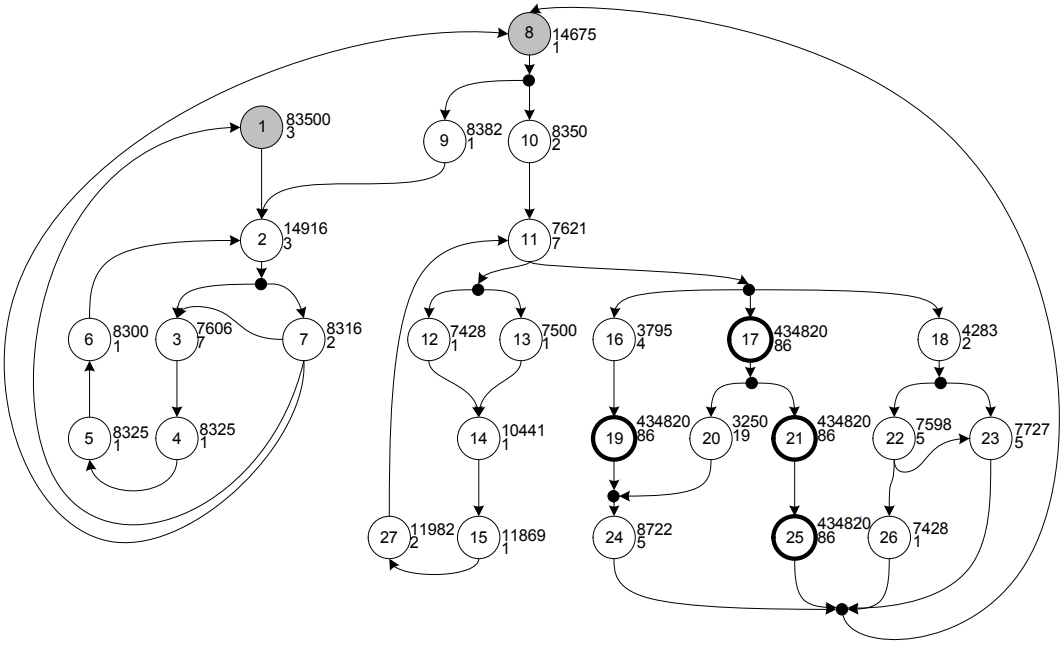
Model pośredni specyfikacji podawany jest do procesu SPMC w formacie SPNF. Następnie, przeprowadzana jest walidacja modelu za pomocą symulatora CoSPeN. W kolejnych krokach specyfikacja wejściowa poddawana jest analizie kosztów czasu realizacji programowej i sprzętowej oraz kosztów realizacji części sprzętowej w układzie FPGA i programowej jako liczby bajtów programu. Do każdego miejsca i tranzycji modelu PNHSMC wprowadzane są parametry *stime* [cykle_zagara] (liczba cykli zegara mikroprocesora niezbędne do wykonania trybu lub tranzycji), *htime* [ns] (najgorszy czas transferu sygnału dla produktu miejsca), *hresources* [CLB] (liczba komórek programowalnych układu FPGA niezbędna do implementacji miejsca/tranzycji) [Xili06]. Na rysunku 5.18 przedstawiono fragment zapisu modelu pośredniego. Do opisu obiektów miejsca i tranzycji wprowadzono informacje analizy kosztów realizacji programowej i sprzętowej. Pełny kod SPNF modelu z rysunku 5.17 przedstawia załącznik E rozprawy.

```
<place name="P2">
      <produce ID="COMB" HOLD="1">W=i</produce>
      <stime>1253</stime>
      <htime>7</htime>
      <hresources>3</hresources>
</place>
<transition name="T2">
      <sensitive>P2</sensitive>
      <action>P3</action>
      <condition>Y=1</condition>
      <stime>491</stime>
      <htime>8</htime>
      <hresources>1</hresources>
</transition>
```

Rysunek 5.18 Model pośredni specyfikacji SPMC

W celu uproszczenia analizy procesu dekompozycji, na rysunku 5.19 przedstawiono sieć opisującą specyfikację SPMC uwzględniającą tylko miejsca modelu. Standardowe obiekty reprezentujące wybrane tranzycje sieci Petriego zastąpiono czarnymi punktami. Koszty czasu pracy oraz implementacji poszczególnych tranzycji zostały odpowiednio dodane do miejsc następujących po danej tranzycji. Ponadto, dla każdego miejsca specyfikacji SPMC wyznaczono współczynnik zysku (akceleracji)

$W_{ACC} = \frac{tp(M)}{ts(M)}$ (tp(M)–czas realizacji zadania miejsca w realizacji programowej, ts(M)–czas realizacji zadania miejsca w realizacji sprzętowej), który określa wagę korzyści przeniesienia miejsca do części sprzętowej. Im większy współczynnik zysku, tym większy zysk. Na rysunku 5.19 pierwsza wartość przypisana dla miejsca określa współczynnik zysku, natomiast druga koszt implementacji miejsca w części sprzętowej. Koszt realizacji części programowej wyrażony jest liczbą instrukcji przypisanych do miejsca lub tranzycji (wzrost liczby instrukcji programu, zwiększa rozmiar kodu programu). Współczynnik zysku pośrednio wyraża również koszt realizacji programowej wybranego miejsca lub tranzycji.

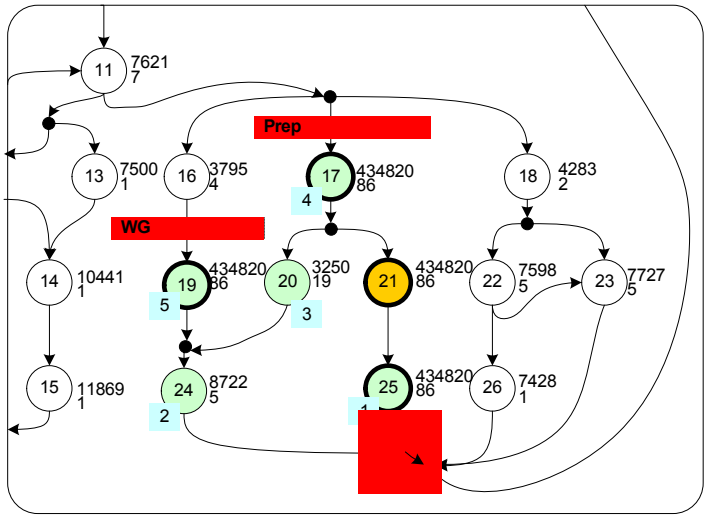


Rysunek 5.19 Uproszczona sieć specyfikacji rysunku 5.17

Postępując zgodnie z algorytmem dekompozycji funkcjonalnej, pierwszym etapem jest wyznaczenie pierwszego punktu plastra zadaniowego. Kierując się kryterium przedstawionym w rozdziale czwartym, punkt 4.1.5, algorytm wybiera pierwsze miejsce programowe do realizacji sprzętowej. Rozważany przykład eksperymentu poddawany jest dekompozycji funkcjonalnej bez analizy aktywności specyfikacji SPMC. Asygnowane zostaje miejsce nr 21, czyli Pm=21. Punkt startowy Pm przekazywany jest do algorytmu budowy plastra zadaniowego. Proces budowy plastra postępuje według algorytmu Al.4.2. Wyznaczana jest cała przestrzeń plastrów zadaniowych oraz wybierane jest najlepsze rozwiązanie. Dla celów prezentacji oraz analizy, w dalszej części podpunktu przedstawiono tylko wybrane konfiguracje podziału i rozwiązania plastra zadaniowego SPMC. Rysunki 5.20, 5.21 oraz 5.22 przedstawiają przykładowe znalezione rozwiązania podziału specyfikacji SPMC, ze wskazaniem kolejnych kroków pracy algorytmu plastra zadaniowego. W celu zobrazowania wpływu parametru określającego liczbę wolnych zasobów sprzętowych FPGA na wyniki procesu dekompozycji SPMC (w tym na zyski czasu pracy SPMC), dla prezentowanych przykładów zdefiniowano trzy różne wartości parametru Z*s*.

Rysunek 5.20 przedstawia podział specyfikacji SPMC przy wolnych zasobach Zs=486(slices). Czerwone prostokąty identyfikują miejsca przecięcia sieci na część programową i sprzętową oraz opcjonalnie nazwy zmiennych synchronizowanych(przesyłanych) z programu do sprzętu lub odwrotnie. Dla rozważanego przykładu są to zmienne: WG,Prep. Niebieskie kwadraty przypisane do miejsc sieci identyfikują kolejność kwalifikacji miejsca do plastra zadaniowego. Dla zdefiniowanych parametrów dekompozycji funkcjonalnej i specyfikacji SPMC, przedstawiono analizę zysku, tj. skrócenia czasu przetwarzania zadanej specyfikacji, po wprowadzeniu mikrostruktury SPMC.

Przykład 1



$Z_s=486-k_{INTF}=386$
$S=\{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27\}$
$K=\{17,19,20,21,24,25\}$
$Pm=21$

$T_p(S)= 25,620[ms]$
$T_{estm}=T_p(S)-T_p(K)+T_s(K)+T_{com}$

$T_p(S) - T_p(K) = 2,853[ms]$
$T_s(K) = 80[ns]$
$T_{com}=Kc_{sp}+Kc_{ps} = 2,6[us]$
$T_{estm}=2,853[ms]+80[ns]+2,6[us]=2,879[ms]$

$\Delta T=T_p(S)-T_{estm}=25,620[ms]-2,879[ms]$ $=22,741[ms]$ ←(zysk)

Rysunek 5.20 Przykład pierwszy podziału specyfikacji funkcjonalnej SPMC

Opis analizy formalnej zysku czasu wykonywania zadań specyfikacji wejściowej dla rysunków 5.20, 5.21, 5.22:

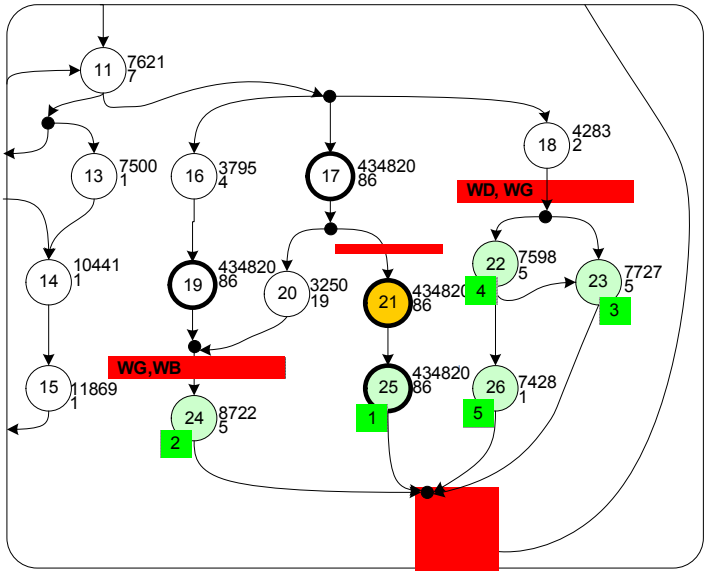
Z_s – liczba wolnych zasobów części sprzętowej, wartość wyrażona w CLB[xili06]
S – zbiór miejsc całej specyfikacji
K – zbiór miejsc plastra zadaniowego
T_{estm} – czas wykonania kompletnej specyfikacji przez SPMC
$T_p(S)$ – czas wykonania kompletnej specyfikacji przez mikroprocesor
$T_p(K)$ – czas wykonania plastra zadaniowego przez mikroprocesor
$T_s(K)$ – czas wykonania plastra zadaniowego przez część sprzętową
T_{com} – czas komunikacji program-sprzęt, sprzęt-program
ΔT – zysk czasu przetwarzania zadanej specyfikacji po wprowadzeniu SPMC do mikrosystemu cyfrowego

W konfiguracji początkowej (bez akceleratora SPMC), całkowity czas wykonania zadanej specyfikacji przez mikroprocesor mikrosystemu SOPC wynosi 25,620[ms]. Mając do dyspozycji wolną logikę reprogramowalną układu FPGA, możliwe jest skrócenie czasu wykonania zadanych instrukcji programu mikroprocesora poprzez wprowadzenie i implementację mikrostruktury SPMC. Wynikiem dekompozycji funkcjonalnej SPMC jest sprzętowy plaster zadaniowy o składowych K={17,18,20,21,24,25}. Przeniesienie wyznaczonych miejsc specyfikacji funkcjonalnej do części sprzętowej skraca czas realizacji zadań przetwarzania i sterowania przez mikroprocesor do 2,879[ms], czyli zysk jest równy 22,741[ms]. Mikrostruktura SPMC postrzegana w mikrosystemie cyfrowym jako główna jednostka sterowania i przetwarzania, wykonuje zadaną specyfikację z rysunku 5.18

w czasie 2,855[ms]. Szacowana wielokrotność zwiększenia wydajności pracy mikrosystemu cyfrowego po wprowadzeniu SPMC dla badanego przykładu wynosi 10 (10 razy).

Rysunek 5.21 przedstawia proces dekompozycji funkcjonalnej sieci z rysunku 5.19 dla Zs=288[slices].

Przykład 2



$Z_s=288-k_{INTF}=188$
$S=\{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27\}$
$K=\{21,22,23,24,25,26\}$
$Pm=21$

$T_p(S)=25,620[ms]$
$\mathbf{T_{estm}=T_p(S)-T_p(K)+T_s(K)+T_{com}}$

$T_p(S) - T_p(K) = 13,990[ms]$
$T_s(K) = 67[ns]$
$T_{com}=Kc_{sp}+Kc_{ps} = 2,916[us]$
$T_{estm}=13,990[ms]+67[ns]+2,916[us]=13,9929[ms]$

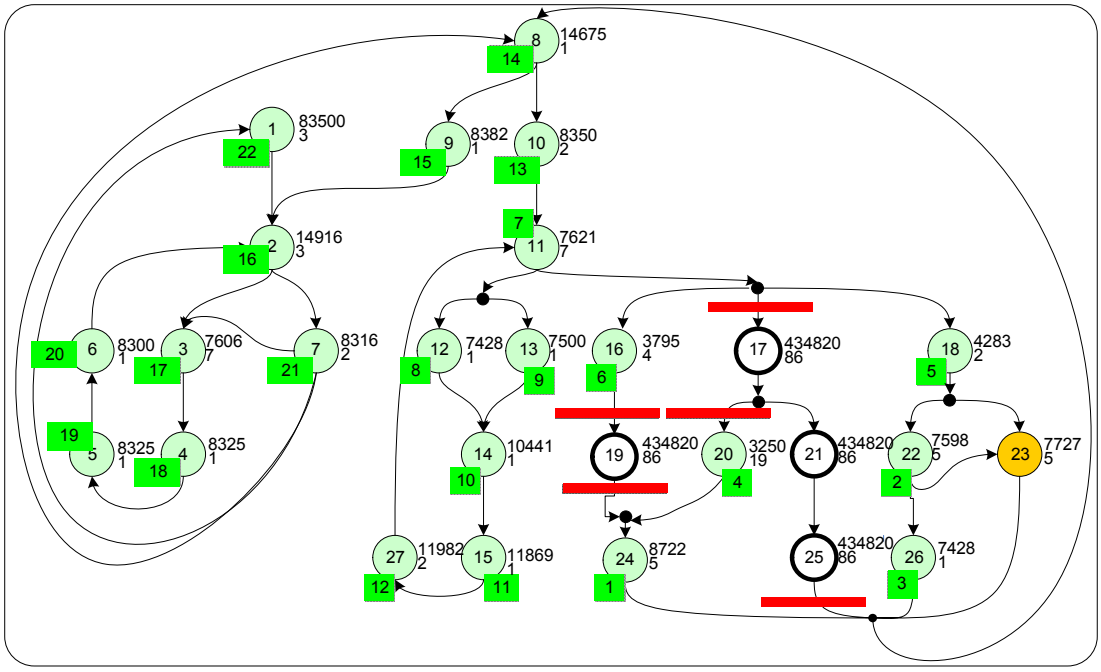
$\mathbf{\Delta T=}T_p(S)-T_{estm}=25,620[ms]-13,9929[ms]\ \mathbf{=11,627[ms]}$

Rysunek 5.21 Przykład podziału specyfikacji funkcjonalnej SPMC z Zs=288

Wyniki rysunku 5.21 przedstawiają wpływ obszaru wolnej logiki reprogramowalnej FPGA na poprawę wydajności pracy mikrosystemu cyfrowego przez wprowadzenie mikrostruktury SPMC. Ograniczenie wolnych zasobów sprzętowych zmniejsza liczbę zadań programowych przenoszonych do części sprzętowej. W rezultacie, zysk czasu ΔT po wprowadzeniu SPMC maleje. Szacowana wielokrotność zwiększenia wydajności pracy mikrosystemu cyfrowego spada do 2 (2 razy).

Rysunek 5.22 przedstawia przykład procesu dekompozycji funkcjonalnej dla Zs zdefiniowano na poziomie 100(slices).

Przykład 3



$Z_s=100-k_{INTF}=80$
$S=\{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27\}$
$K=\{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,18,20,22,23,24,26,27\}$
$Pm=23$

$T_p(S)=25,620[ms]$
$T_{estm}=T_p(S)-T_p(K)+T_s(K)+T_{com}$

$T_p(S) - T_p(K) = 23,78225[ms]$
$T_s(K) = 216[ns]$
$T_{com}=Kc_{sp}+Kc_{ps} = 4,9[us]$
$T_{estm}=23,782[ms]+216[ns]+4,9[us]=23,787[ms]$

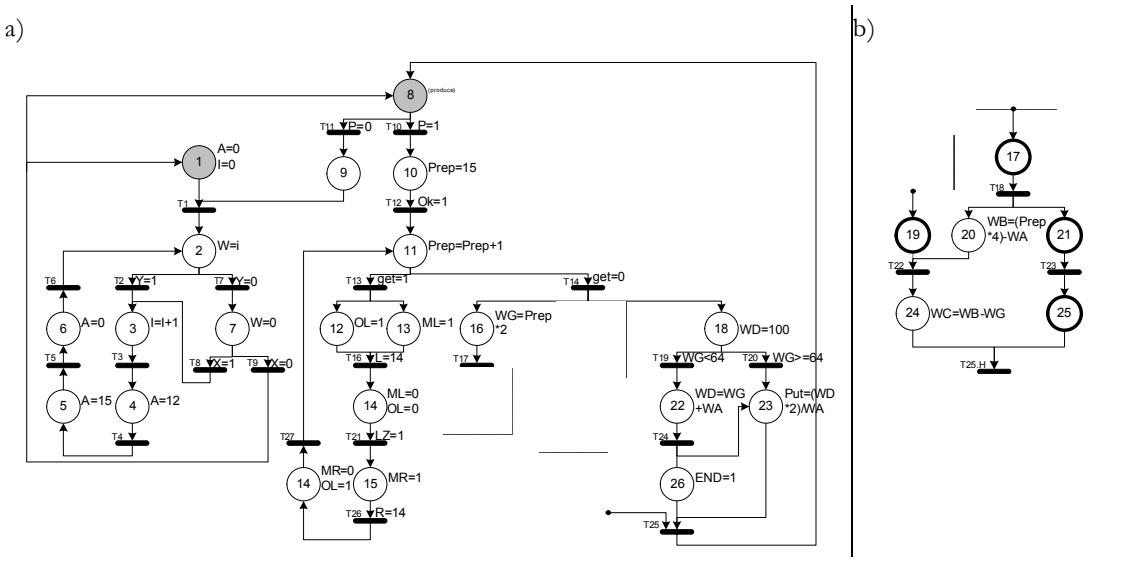
$\Delta T=T_p(S)-T_{estm}=25,620[ms]-23,787[ms]$ $=2,21[ms]$

Rysunek 5.22 Przykład podziału specyfikacji funkcjonalnej SPMC z Zs=100

Pomimo, że zysk dla przykładu z rysunku 5.22 nie jest tak znaczący w porównaniu z rozważanymi przykładami 1 i 2, to jednak skrócenie czasu realizacji programu przetwarzania i sterowania przez główną jednostkę sterowania i przetwarzania w mikrosystemie cyfrowym jest zawsze korzystne. Przeniesienie do wolnej logiki reprogramowalnej nawet jednej instrukcji programowej, zachowując wzór 4-18, spełnia zależność 6-1.

Po etapie dekompozycji funkcjonalnej dla rozważanego modelu przeprowadzano proces kosymulacji programowo-sprzętowej w środowisku CosPeN, z uwzględnieniem szacowanych czasów. Weryfikacji poddawane są zależności czasowe części programowej i sprzętowej oraz poprawność funkcjonalna z modelem wzorcowym (przed podziałem).

Kolejnym etapem jest synteza sprzętowa i programowa części programowej i sprzętowej zdekomponowanej sieci Petriego. Rozważając podział z rysunku 5.20, programowa sieć Petriego reprezentowana jest przez zbiór miejsc jak na rysunku 5.23.a), natomiast sieć sprzętowa przedstawiona została na rysunku 5.23.b).



Rysunek 5.23 Specyfikacja SPMC po procesie dekompozycji funkcjonalnej ADES: a) część programowa, b) część sprzętowa

Wynikiem metody SPMC jest gotowa ściśle zintegrowana sprzętowo-programowa mikrostruktura cyfrowa SPMC opracowana na podstawie zadanej specyfikacji wejściowej. Dla mikroprocesora Atmega103 generowany jest kod C, natomiast część sprzętowa przedstawiona jest za pomocą języka VHDL. Ostatnim krokiem metody SPMC jest weryfikacja funkcjonalna z wykorzystaniem środowiska wirtualnego oraz opcjonalnie symulacja JTAG-SIM. Dodatek E przedstawia kompletną ścieżkę projektową SPMC.

5.3. Wyniki syntezy programowej sieci Petriego metodą SPMC

Weryfikacja funkcjonalna oraz pomiar czasów reakcji programu opracowanego drogą automatycznej syntezy programowej sieci Petriego SPMC, zweryfikowano w środowisku wirtualnym oprogramowania ADES. Pomiar czasów reakcji i wydajności pracy przeprowadzono z wykorzystaniem środowiska ModelSIM 6.0 firmy MentorGraphics [ment06]. Testy funkcjonalne zostały przeprowadzone dla mikrokontrolera klasy RISC, Atmel AVR Atmega103 [Atme06] taktowanego zegarem o częstotliwości 12MHz.

Przygotowano pięć grup testowych sieci Petriego w formacie SPNF o różnym współczynniku współbieżności i hierarchii. Następnie kod XML poddano procesowi automatycznej generacji programu C. Kompilację przeprowadzono z wykorzystaniem oprogramowania WinAVR [wina06] z wyłączeniem dodatkowych technik optymalizacji kodu wynikowego. Testy dotyczą czasu wykonania pełnego cyklu decyzyjnego. Miejsca sieci nie posiadają zdefiniowanych akcji (są puste) oraz tranzycje nie mają przypisanych warunków logicznych realizacji (brak interpretacji).

Konfigurację poszczególnych sieci testowych oraz wyniki prezentuje tabela 5.1.

Tabela 5.1 Konfiguracja zbiorów sieci testowych.

Nazwa testu	Lm	Lt	Wr	wh	t[ms]	to[ms]

Test1	19	17	2	0	8,8	7,38
Test2	34	28	2	1	18,5	16,2
Test3	53	45	4	1	30,38	27,7
Test4	113	96	10	1	62,11	56,27
Test5	203	172	13	1	121,44	111,6

Objaśnienia tabeli:

LM– liczba miejsc sieci
LT– liczba tranzycji sieci
WH– współczynnik hierarchii sieci
WR– współczynnik współbieżności
T– maksymalny czas wykonania pełnego cyklu programu bez optymalizacji przetwarzania hierarchii i tranzycji
TO– maksymalny czasy wykonania pełnego cyklu programu z optymalizacją przetwarzania hierarchii i tranzycji

Wynikom tabeli 3.1 konfrontacji poddane zostały wyniki syntezy programowej sieci Petriego zaczerpnięte z pracy [Andr03], która charakteryzuje się zbliżonym modelem formalnym sieci Petriego. W pracy [Andr03] estymację czasu pracy przeprowadzono z wykorzystaniem narzędzi profilowania kodu programu, który dostarcza idealnych wyników pomiarowych. W referowanej pracy doktorskiej, pomiary przeprowadzono z wykorzystaniem czasowego modelu HDL mikrokontrolera AVR, gdzie wyniki przetwarzania programu przez procesor są zbliżone do wartości rzeczywistych. Testy [Andr03] wykonano dla mikrokontrolera CISC 8051 (zegar systemowy równy 12MHz), natomiast w niniejszej rozprawie wykorzystano mikroprocesor RISC. Różnice konstrukcyjne mikroprocesorów CISC 8051 i RISC AVR Atmega103 nie są brane pod uwagę (są pomijalne) ze względu na brak powiązania ewentualnych rozbieżności czasu wykonania tej samej instrukcji programowej przez CISC i RISC (Atmega103 nie wpiera potoku przetwarzania danych).

Tabela 5.2 przedstawia porównanie czasów pracy cyklu decyzyjnego obu rozwiązań z uwzględnieniem ilości miejsc, tranzycji, współczynnika równoległości oraz hierarchii. Ze względu na brak dostępu do sieci testowych pracy[Andr03], skonfrontowano najbardziej zbliżone konfiguracje sieci.

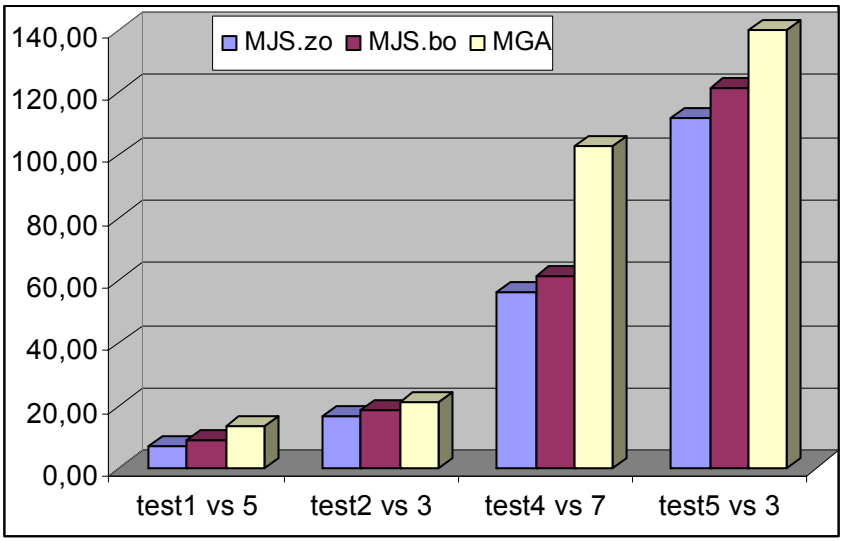
Tabela 5.2 Zestawienie czasów wykonania pełnego cyklu decyzyjnego programu przez mikroprocesor dla wybranych sieci Petriego.

Synteza programowa sieci Petriego metodą SPMC							Porównanie		Synteza programowa sieci Petriego metodą G.A.[]					
tk	lm	Lt	wr	wh	T	to	⇔		ta	lm	lt	wr	wh	Tmax
1	19	17	2	0	8,8	7,38	✓	×	5	21	20	2	0	13,6
2	34	28	2	1	18,5	16,6	✓	×	6	31	29	3	0	21
4	113	96	10	1	62,1	56,27	✓	×	7	101	83	10	0	103
5	203	172	13	1	121,4	111,6	✓	×	3	256	256	0	0	140

Objaśnienia tabeli:

LM– liczba miejsc sieci
LT– liczba tranzycji sieci
WH– współczynnik hierarchii sieci
WR– współczynnik współbieżności
T– maksymalny czas wykonania pełnego cyklu programu bez optymalizacji przetwarzania hierarchii i tranzycji
TO– maksymalny czasy wykonania pełnego cyklu programu z optymalizacją przetwarzania hierarchii i tranzycji
TMAX– maksymalny czas wykonania pełnego cyklu decyzyjnego programu opracowanego metodą G.A.[Andr03]

Wyniki prezentowane w tabeli przedstawiono ponownie na rysunku 5.14. Porównaniu poddano trzy metody syntezy programowej sieci Petriego: a) klasyczna metoda jednorodnej realizacji sekwencyjnej [Misi80], b) metoda jednorodnej realizacji sekwencyjnej SPMC, c) metoda [Andr03].



Rysunek 5.14 Zestawienie graficzne czasów przetwarzania cyklu decyzyjnego przez mikroprocesor programu opracowanego metodą: a) MJS.bo (klasyczna metoda jednorodnej realizacji sekwencyjnej [Misi80]), b) MJS.zo (metoda jednorodnej realizacji sekwencyjnej SPMC), c) metoda G.Andrzejewskiego

Porównanie dotyczy tylko i wyłącznie czasu wykonania przez mikroprocesor cyklu decyzyjnego sieci Petriego. Wadą metody jednorodnej realizacji sekwencyjnej sieci Petriego jest na pewno rozmiar kodu programu, który w porównaniu z wynikami [Andr03] jest średnio dwukrotnie większy. Jednak głównym celem opracowanej metody SPMC syntezy programowej sieci Petriego, jednak była minimalizacja czasu przetwarzania programu przez mikroprocesor. Rozprawa dotyczy akceleracji pracy procesora. Koszt objętości programu jest istotny, jednak analizując dane konstrukcyjne szeregu architektur mikroprocesorów [atme06, xili06, inte06], pamięć programu w większości przypadków jest wielokrotnie większa od pamięci danych. W związku z czym, zasadna jest optymalizacja czasu pracy procesora kosztem rozmiaru kodu programu. Jednocześnie, prowadzone są dalsze prace nad syntezą programową SPMC, skupiające się na optymalizacji rozmiaru kodu programu zachowując lub zwiększając aktualny poziom skrócenia czasu przetwarzania i sterowania.

5.4. Wyniki syntezy sprzętowej sieci Petriego metodą SPMC

Badania przeprowadzono dla języka opisu sprzętu VHDL. Za względu na poziom abstrakcji modelowanych zadań części sprzętowej, użyte rozwiązania opisu sprzętu SPMC (sterowania i przetwarzania danych) mogą swobodnie zostać zrealizowane z wykorzystaniem innych języków HDL, np. Verilog.

Metoda syntezy sprzętowej SPMC modelu układu cyfrowego opisanego sieciami PNHSMC, wykorzystuje opracowania pracy doktorskiej [Wola98], w szczególności metodę syntezy zorientowaną na tranzycje.

Realizacja sprzętowa sieci Petriego, ze względu na stany pracy sieci, wymaga implementacji pamięci stanu aktualnego sieci – reprezentacja części sterującej układu cyfrowego[ElKu98]. Dotychczasowe prace [Adam86,Kuba04] przedstawiają kilka metod kodowania stanów. Celem jest minimalizacja kosztów implementacyjnych bloku sterowania części sprzętowej poprzez ograniczenie liczby elementów pamiętających. Jednak wraz ze zmniejszeniem liczby wykorzystanych przerzutników, wzrastają funkcje stanu następnego dla bloku sterowania, co skutkuje zwiększeniem czasu reakcji projektowanego sprzętu. Ze względu na charakter prowadzonej rozprawy, do realizacji sprzętowej sieci Petriego SPMC zastosowano kodowanie „one-hot", które dostarcza satysfakcynujących parametrów czasowych [Kuba04], lecz charakteryzuje się stosunkowo dużym zapotrzebowaniem na zasoby sprzętowe w postaci przerzutników. Aktualnie prowadzone prace [BuAd06] w przyszłości mogą zostać zastosowane w metodzie syntezy sprzętowej sieci Petriego SPMC poprawiając współczynnik alokacji wolnej logiki sprzętowej. Ponadto, w niniejszej rozprawie (rozdział czwarty) przedstawiono dodatkowe metody optymalizacji realizacji sprzętowej interpretowanych, hierarchicznych sieci Petriego.

Badania syntezy sprzętowej sieci Petriego metodą SPMC przeprowadzono na wydzielonych trzech zbiorach testów: a)sieci płaskie, b)sieci hierarchiczne, c)sieci mieszane; charakteryzujące typ specyfikowanego zachowania oraz strukturę behawioralną. W grupie modeli testowych znajdują się specyfikacje sterowania i przetwarzania wykorzystywane w przemyśle (Kronopol [kron06], Zielonogórska Elektrociepłownia [ziel06]) oraz przykłady literaturowe ([Andr03, ElKu98, Adam91]). Tabela 5.3 przedstawia zestawienie parametrów wybranych modeli testowych.

Tabela 5.3 Konfiguracja sieci testowych poddanych syntezie sprzętowej.

Nr	Nazwa testu [źródło]	LM	LT	WH	LWI	WR	LE/LY
1	Reaktor[Węgr98]	16	13	0	0	3	
2	linkAdapter [Wola98,]	31	37	1	0	2	12/12
3	Parallel Controller [Adam91]	16	13	0	0	4	10/8
4	Test1(optyma1)	55	41	1	11	5	13/11
5	Test2(optyma3)	86	66	3	18	20	
6	Test3(nowy)	59	43	2	3	7	
7	Test4(nowy2)	63	46	2	7	13	

Opis tabeli:

Nr– numer testu
Nazwa testu– nazwa testu zaczerpnięta z literatury lub nazwa własna
LM– liczba miejsc sieci
LT– liczba tranzycji sieci
WH– współczynnik hierarchii sieci
WR– współczynnik współbieżności
LWI– liczba współdzielonych instancjacji
LE/LY– liczba wejść/liczba wyjść

Wyniki implementacji oraz pomiary czasów pracy sprzętowej realizacji sieci Petriego opisanych językiem VHDL (rezultat syntezy sprzętowej SPMC) zostały przeprowadzone z wykorzystaniem komercyjnego środowiska syntezy logicznej i implementacji układów cyfrowych Xilinx ISE 7.3.

Konfrontacji poddano wyniki implementacji układowej sieci Petriego opracowanej metodą [Wola98] (zorientowanie na tranzycje) oraz metodą SPMC z uwzględnieniem algorytmów optymalizacji omówionych w rozdziale czartym. Tabela 5.4 przedstawia zestawienie wyników obszaru implementacji i parametrów czasowych dla wybranych modeli testowych.

Tabela 5.4 Wyniki algorytmu SPMC syntezy sprzętowej sieci Petriego.

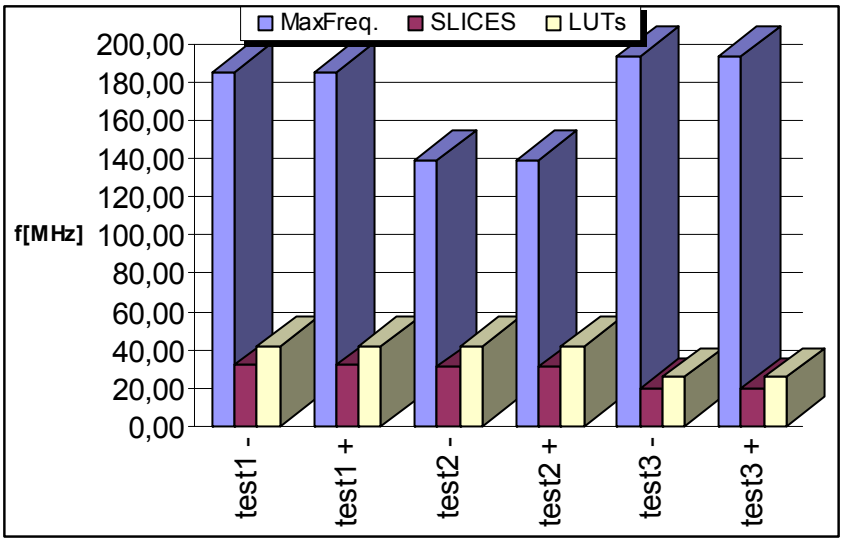
Test	Optym.	SLICES	CLB FF	4-input LUT	SLICES%	IOB	MaxFaq[MHz]
1	☐	32	25	42	2	24	184.980
	☒	32	25	42	2	24	184.980
2	☐	31	29	42	2	17	139.334
	☒	31	29	42	2	17	139.334
3	☐	20	19	26	1	16	193.349
	☒	20	19	26	1	16	193.349
4	☐	56	63	59	4	27	218.818
	☒	39	44	54	3	27	148.434
5	☐	119	150	147	9	29	202.758
	☒	75	103	114	6	29	148.434
6	☐	260	169	477	21	30	107.664
	☒	99	59	180	8	30	107.664
7	☐	608	391	1116	50	66	103.767
	☒	203	117	371	16	66	103.767

Opis kolumn tabeli:

Test– numer testu odpowiadający tabeli nr 5.3
Optym.– oznaczenie włączonej optymalizacji zasobów sprzętowych w procesie syntezy sprzętowej SPMC, ☒(optymalizacja włączona), ☐(optymalizacja wyłączona)
SLICES– liczba wykorzystanych bloków konfiguracyjnych układu FPGA
CLB FF– liczba wykorzystanych przerzutników komórek konfiguracyjnych FPGA
4-input LUT– liczba wykorzystanych generatorów funkcji
SLICES %– procentowa alokacja układu FPGA
IOB– liczba wykorzystanych zasobów portów wejścia-wyjścia
MaxFreq– maksymalna częstotliwość pracy części sprzętowej

Synteza sprzętowa SPMC płaskich sieci Petriego

Rysunek 5.15 przedstawia graficzną reprezentację wyników syntezy sprzętowej płaskich sieci Petriego metodą SPMC w optymalizacją i bez optymalizacji. Badaniu poddano sieci płaskie z tabeli 5.4.

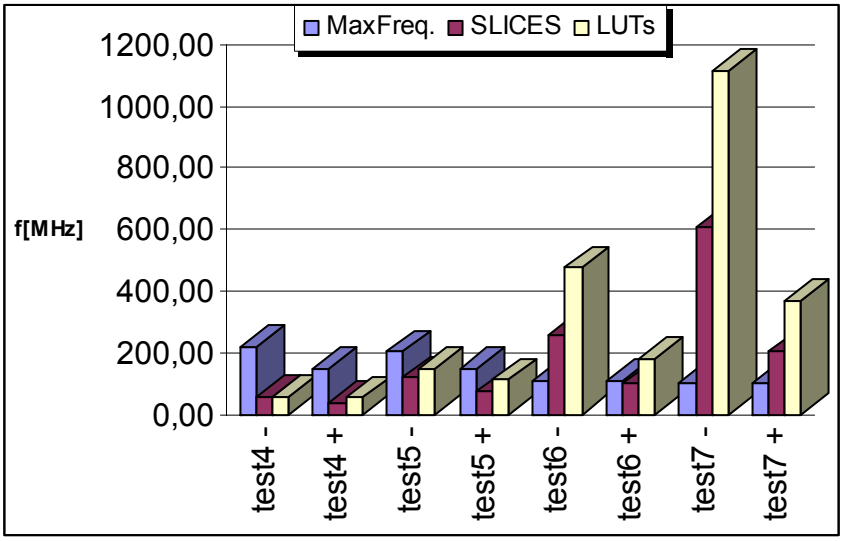


Rysunek 5.15 Wpływ optymalizacji SPMC na koszt implementacji oraz maksymalną częstotliwość pracy części sprzętowej dla sieci płaskich, gdzie „test-" – optymalizacja wyłączona, „test+" – optymalizacja włączona

Algorytm optymalizacji SPMC syntezy sprzętowej mikrostruktury cyfrowej nie wpływa na wynik końcowy syntezy sieci płaskich. Sieć pozbawiona hierarchii oraz sekwencji makromiejsc poddawana jest procesowi syntezy według klasycznej metody [Wola98], czyli algorytm syntezy sprzętowej SPMC nie wykorzystuje opracowanych w rozprawie metod optymalizacji.

Synteza sprzętowa hierarchicznych sieci Petriego metodą SPMC

Wyniki syntezy złożonych sieci hierarchicznych przedstawia rysunek 5.16. Rezultaty procesu syntezy sprzętowej sieci Petriego przeprowadzonej z uwzględnieniem technik optymalizacji SPMC, potwierdzają rozważania przeprowadzone w rozdziale czwartym, punkt 4.1.7. Wzrost liczby instancjacji jednego komponentu w syntezie sprzętowej SPMC nie powoduje lawinowego wzrostu zajętości obszaru logiki układu reprogramowalnego w porównaniu z opracowaniami [Wola98], wyniki test6+ i test6- oraz test7+ i test7-. Natomiast, konsekwencją realizacji systemu przełączania jest obniżenie maksymalnej częstotliwości pracy części sprzętowej SPMC. Testy 4 i 5 instancjonują odpowiednio 11 i 18 współdzielonych komponentów sieci o małej złożoności (3 miejsca, 4 tranzycje). Obserwowany spadek częstotliwości pracy układu cyfrowego jest stosunkowo duży w odniesieniu do uzyskanych korzyści minimalizacji obszaru FPGA.



Rysunek 5.16 Wpływ optymalizacji SPMC na koszt implementacji oraz maksymalną częstotliwość pracy części sprzętowej dla sieci hierarchicznych, gdzie „test-" – optymalizacja SPMC jest wyłączona, „test+" –optymalizacja SPMC jest włączona

Opcja optymalizacji części sprzętowej w procesie dekompozycji funkcjonalnej SPMC, jest automatycznie załączana lub wyłączana przez algorytm syntezy SPMC, w celu minimalizacji obszaru implementacji części sprzętowej FPGA. Dzieki temu, możliwe jest wiekszenie liczby alokacji zadań programowych w realizacji sprzętowej.

~~*1.3.1.Weryfikacja funkcjonalna syntezy sprzętowej SPMC*~~

~~1.4.Akceleracja~~

~~Ze względu na charakter prowadzonych prac w zakresie modelowania zadań programowych i sprzętowych oraz braku w literaturze modeli sieci o wystarczającej złożoności funkcjonalnej i strukturalnej (w tym brak dostępu do źródeł modeli testowych wykorzystywanych w pracach[][]), niezbędnym okazało się opracowanie zbioru trzech typów testów.~~

ROZDZIAŁ SZÓSTY

6. Rezultaty metody SPMC oraz podsumowanie

Opracowana metoda projektowania SPMC, ze względu na poziom abstrakcji rozważanych problemów, pozwala na precyzyjną alokację wybranych zadań/operacji specyfikacji wejściowej SPMC do części programowej lub sprzętowej. Dodatkowo, zintegrowane projektowanie sprzętowo-programowej mikrostruktury cyfrowej SPMC może doskonale uzupełniać metodologię zintegrowanego projektowania systemów sprzętowo-programowych (ang. hardware software codesign) w obszarze projektowania procesorowego/nisko-poziomowego. Jednoczesnie, uzupełnienie metodologii klasycznej o metodę projektowania SPMC, może wspomóc rozwiązanie szeregu problemów projektowania klasycznego, dotyczących m.in.: niskiej wydajności pracy procesora, problemów z realizacją części sprzętowej w reprogramowalnej logice FPGA oraz zapewnieniem wydajnej i taniej pod względem zasobów FPGA komunikacji typu program-sprzęt. Dodatkowym atutem jest automatyzacja całego procesu SPMC oraz gwarancja osiągnięcia zysku, tj. metoda projektowania SPMC pozwala na osiągnięcie lepszych wyników realizacji w porównaniu z aktualnymi lub w najgorszym przypadku braku poprawy (realizacja wejściowa bez zmian), wzór 6.1. W procesie dekompozycji funkcjonalnej metoda SPMC posiada dwa ekstrema:

1. *ep*; cała wejściowa specyfikacja funkcjonalna SPMC jako realizacja programowa, brak części sprzętowej.
2. *es*; cała wejściowa specyfikacja funkcjonalna SPMC jako realizacja sprzętowa, brak części programowej, komponent mikroprocesora usuwany z mikrosystemu cyfrowego.

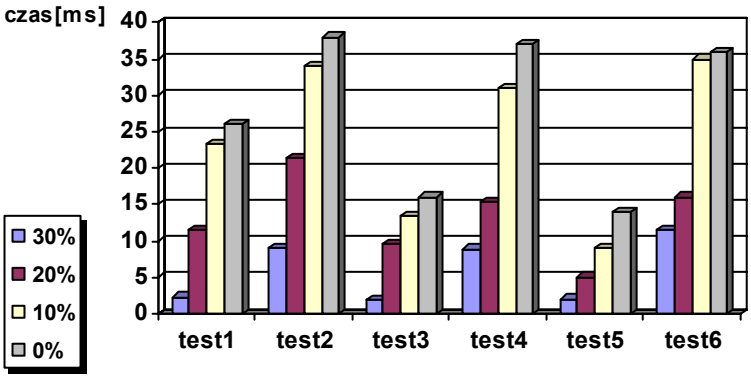
Metoda SPMC w projektowaniu klasycznym i zintegrowanym dokonuje szeregu analiz formalnych i funkcjonalnych specyfikacji wejściowej, którą jest kod programu mikroprocesora (opracowany przez narzędzia lub projektanta w procesie projektowym). Więc, wprowadzenie mikrostruktury SPMC do mikrosystemu cyfrowego zwiększa jego wydajność, ze względu na zależność:

$$t(ep) > t(SPMC) > t(es) \qquad \text{(Wzór 6-1)}$$

gdzie:

T(ep)– czas przetwarzania zadanej specyfikacji tylko przez mikroprocesor
t(SPMC)– czas przetwarzania zadanej specyfikacji przez mikrostrukturę SPMC
t(es)– czas przetwarzania zadanej specyfikacji tylko przez część sprzętową

Przedmiotem rozprawy było zwiększenie wydajności pracy wbudowanej głównej jednostki sterowania i przetwarzania w celu przyspieszenia pracy mikrosystemu cyfrowego, rysunek 1.3. Teoria techniki cyfrowej dowodzi, że określone zadania realizowane przez specjalizowane układy cyfrowe, zawsze wykonywane są znacznie szybciej w porównaniu z przetwarzaniem takiej samej instrukcji przez standardowy procesor. Czas propagacji przykładowej funkcji iloczynu logicznego dwóch literałów jednobitowych w realizacji układowej wynosi (dla układu FPGA firmy Xilinx Spartan2E) 9[ns] (z uwzględnieniem czasu propagacji w blokach wejścia-wyjścia). Natomiast wykonanie tego samego zadania przez procesor AVR Atmega103 zajmuje około 100[us]. W związku z powyższym, wydajność części sprzętowej przyjmuje się za wartość stałą w relacji do wydajności wbudowanego CPU mikrosystemu cyfrowego. Rozważając problem podjęty w rozprawie, a przedstawiony w rozdziale pierwszym, zwiększając wydajność pracy głównej jednostki CPU mikrosystemu cyfrowego poprzez wprowadzenie do systemu sprzętowo-programowej mikrostruktury cyfrowej SPMC, wzrasta wydajność pracy całego mikrosystemu cyfrowego. Przeprowadzone badania rozprawy wykazały wzrost wydajności pracy SPMC, który zależny jest od liczby wolnych zasobów sprzętowych układu FPGA implementującego system SOPC. W grupie modeli testowych znajdują się specyfikacje sterowania i przetwarzania wykorzystywane w przemyśle (Kronopol [kron06], Zielonogórska Elektrociepłownia [ziel06]) oraz przykłady literaturowe ([Andr03, ElKu98, Adam91]) wkomponowane w rozbudowane syntetyczne modele testowe (rozdział piąty). Zbiorczy wykres prezentujący zależności wydajność pracy SPMC, a przez to potencjalnie całego mikrosystemu cyfrowego, względem dostępnych wolnych zasobów logiki programowalnej FPGA, przedstawia rysunek 6.1.

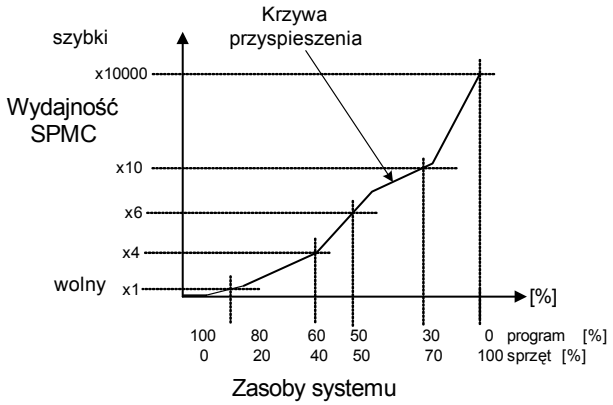


Rysunek 6.1 Zależność wzrostu wydajności SPMC względem zdefiniowanej wolnej logiki reprogramowalnej układu FPGA

Oś X rysunku 6.1 reprezentuje numer testu, natomiast oś Y czas przetwarzania zadanej specyfikacji wejściowej przez mikrostrukturę SPMC. Dla każdego z testów przeprowadzono badania dotyczące wpływu liczby (obszaru) wolnej logiki reprogramowalnej, na czas wykonania specyfikacji programu przez jednostkę SPMC. Na podstawie doniesień [xili06] (punkt 1.1 rozprawy), zdefiniowano

przedział (wartość Zs) od 30% do 1% zasobów sprzętowych pozostających do zagospodarowania po procesie implementacji mikrosystemu cyfrowego w układzie FPGA. Testy przeprowadzono dla układu Xilinx Spartan2E o łącznej liczbie bloków konfiguracyjnych 1728 [xili06]. Analiza rysunku 6.1 pozwala zaobserwować znaczące skrócenie czasu wykonania określonego zadania przy wzroście obszaru wolnej logiki rekonfigurowalnej FPGA.

Dodatkowo, wyniki akceleracji procesora ogólnego przeznaczenia mikrosystemu cyfrowego przy wykorzystaniu rozwiązań SPMC, mogą być zaniżone, ze względu na starszą wersję systemu komunikacji program↔sprzęt wykorzystywaną podczas testów. Na rysunku 6.2 przedstawiono krzywą wzrostu wydajności pracy SPMC, która jest średnią wypadkową wyników testów przeprowadzonych w ramach rozprawy.



Rysunek 6.2 Wpływ przydziału zadań specyfikacji do części programowej i sprzętowej

Analiza rysunków 6.1 oraz 6.2 potwierdza korzystny wpływ wprowadzenia architektury SPMC do mikrosystemu cyfrowego, czego wymiernym efektem jest zwiększenie jego wydajność pracy. Rezultatem przeprowadzonej rozprawy jest automatyczny oraz efektywny pod względem szybkości pracy i kosztów realizacji technicznej, podział specyfikacji funkcjonalnej mikrostruktury cyfrowej na część programową w języku ANSI C, wykonywaną przez mikroprocesor ogólnego przeznaczenia oraz część sprzętową, realizowaną przez specjalizowane układy programowalne typu FPGA, z zachowaniem pełnej funkcjonalności projektowanego mikrosystemu.

Dodatkowo, zasadność przeprowadzonych prac w domenie akceleracji sterowania i przetwarzania na poziomie procesorowym (instrukcje, pojedyncze zadania), które zapoczątkowano przez autora referatu już w roku 2002 [Stas02a,Stas02b], potwierdzają prace szeregu producentów [cibl06,mimo06] raportowane w [xcel06] przez jednego z czołowych światowych producentów układów FPGA [xili06]. Przeprowadzona rozprawa może zostać zakwalifikowana do narzędzi klasy ESL (ang. Electronic System Level design), które z coraz większym pożądaniem wprowadzane są do procesu projektowania systemów typu SOPC i SoC.

Podsumowanie

W ramach przeprowadzonych pracy, opracowana została *nowa metoda projektowania zintegrowanej sprzętowo-programowej mikrostruktury cyfrowej* (rozdział czwarty)

wspomagająca proces realizacji układowej części sprzętowej oraz opracowanie oprogramowania dla części programowej SPMC (procesor + akcelerator), z zachowaniem pełnej funkcjonalności specyfikacji wejściowej. Przedstawiona w pracy metoda projektowania SPMC oparta została na znanych rozwiązaniach syntezy systemowej. Posiłkując się doświadczeniem wzorcowych opracowań, takich jak Cosyma czy Vulcan; opracowano metodę, która zapewnia współbieżność sterowania i przetwarzania danych w domenie sprzętowo-programowej mikrostruktury cyfrowej. Wskazane w rozprawie prace naukowe charakteryzują się realizacją sekwencyjnej ścieżki przepływu danych i sterowania w obrębie zintegrowanych, hybrydowych architektur cyfrowych.

Sformalizowano nowy model formalny mikrostruktury cyfrowej SPMC (rozdział 4.1.2) oraz język SPNF pozwalający na zapisu funkcjonalności mikrostruktury w postaci elektronicznej, z wykorzystaniem standardu XML (rozdział czwarty, 4.1.3.). Zaproponowany w pracy model sprzętowo-programowych sieci Petriego PNHSDM, rozszerza istaniejące opracowania definiując cechy modelu charakterystyczne opisowi systemu heterogenicznego:

- implementacja ścieżki danych,
- deklaracja zadań sprzętowych i programowych,
- deklaracja i implementacja historii w realizacji układowej (pauza),
- deklaracja i implementacja wyjątków w realizacji układowej i programowej,
- deklaracja oraz realzacja układowa programowych i sprzętowych strukturalnych typów sieci Petriego (makromiejsca proceduralne i współdzielone).

Opracowano algorytm dekompozycji funkcjonalnej SPMC (rozdział 4.1.4), pozwalający na efektywny pod względem przydzielanych zasobów i czasu realizacji zadań, podział specyfikacji wejściowej na część programową i sprzętową. Zorientowanie pracy algorytmu na budowę plastry zadaniowych pozwala na optymalizację kosztów czasu komunikacji w domenie program-sprzęt.

Opracowano architekturę sprzętowo-programowej mikrostruktury cyfrowej SPMC (rozdział trzeci), która jest nowatorskim rozwiązaniem w domenie akceleracji przetwarzania i sterowania mikrosystemów SOPC. To nowa zintegrowana architektura sprzętowo-programowej jednostki cyfrowej wyróżniająca się na tle znanych rozwiązań typu RISP czy ASIP. Skrócenie czasu wykonywania instrukcji przetwarzania i sterowania przez główną jednostkę CPU mikrosystemu cyfrowego uzyskano poprzez wykorzystanie *wolnych (nie zagospodarowanych) zasobów logiki reprogramowalnej układu FPGA* w układzie SOPC, jako akceleratora sprzętowego oraz poprzez *zrównoleglenie wykonywanywanych zadań przetwarzania i sterowania w strukturach sprzętowych*. SPMC zapewnia uniwersalną architekturę części sprzętowej pozwalającą na integrację opracowanego interfejsu komunikacyjnego z dowolnym procesorem ogólnego przeznaczenia. Ponadto, architektura SPMC eliminuje konieczność implementacji nadmiarowych komponentów funkcjonalnych (np. pamięć współdzielona, zaawansowane systemu komunikacji, w tym DMA) w obszarze wolnej logiki reprogramowalnej FPGA, która przeznaczona jest do realizacji zadań mikrosystemu cyfrowego.

Na rzecz rozprawy *dokonano szeregu modyfikacji/optymalizacji oraz wprowadzono nowe, autorskie rozwiązania do istniejących algorytmów syntezy sprzętowej i programowej sieci Petriego*, które potwierdzono badaniami teoretycznymi oraz testami funkcjonalnymi

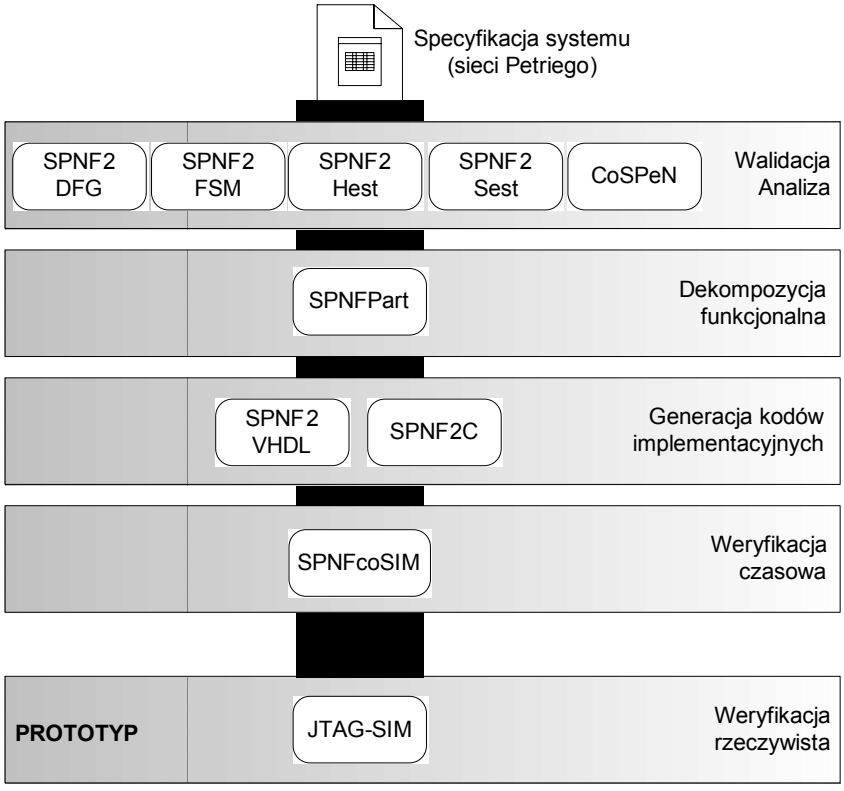
prezentowanymi w rozdziale piątym. Opracowano również nowy algorytm statycznego szacowania zasobów w realizacji sprzętowego systemu przełączania zadań współdzielonych specyfikacji.

Na potrzeby przeprowadzonej rozprawy oraz dalszych prac naukowych, *wykonano szereg projektów inżynierskich wspomagających etapy projektowania i analizy formalnej mikrostruktury SPMC,* są to między innymi:

- symulator hierarchicznych sieci Petriego dla potrzeb projektowania zintegrowanego, uwzględniający parametry czasowe oraz przydział implementacyjny (programowy lub sprzętowy) zadań mikrostruktury,
- program do wizualizacji i graficznej edycji hierarchicznych sieci Petriego,
- wirtualny system kosymulacji sprzętowo-programowej, bazujący na symulatorze języków HDL,
- system kosymulacji sprzętowo-programowej mikrosystemu cyfrowego w układzie FPGA wykorzystujący interfejs JTAG.

Wszystkie modele zbioru testowego poddano weryfikacji funkcjonalnej w symulatorze CoSPeN oraz w środowisku wirtualnym ADES. Wybrane rozwiązania SPMC zaimplementowano do układu FPGA oraz poddano symulacji funkcjonalnej JTAG-SIM.

W ramach prac naukowo-technicznych z zakresu przeprowadzonej rozprawy, opracowano zbiór narzędzi operujących na zapisie specyfikacji zachowania modelu w formacie SPNF. Dostępne oprogramowanie stanowi kompletne zaplecze programistyczne w procesie projektowym systemów programowych, sprzętowych, i sprzętowo-programowych. Organizację środowiska projektowe przedstawia diagram na rysunku 6.3.



Rysunek 6.3 Diagram oprogramowania SPMC

Opis składowych środowiska SPMC:

SPNF2DFG – wyznaczenia grafu przepływu danych

SPNF2FSM – wyznaczenie składowych automatowych systemu

SPNF2VHDL – synteza opisu SPNF do języka VHDL z wieloma optymalizacjami

SPNF2C – synteza opisu SPNF do języka C

CoSPeN – edytor graficzny, programowy symulator i kosymulator sieci Petriego reprezentowanych w formacie SPNF

SPNFcoSIM – wirtualne środowisko prototypowania oraz sprzętowej weryfikacji funkcjonalnej i czasowej

SPNF2HSestm – estymacja zasobów składowych systemu opisanego w formacie SPNF

SPNFPart – algorytm dekompozycj funkcjonalnej SPMC na część programową i sprzętową

7. Literatura

[acte06] www.actel.com

[Adam86] M.Adamski, *Heurystyczna metoda strukturalnego kodowania miejsc sieci Petriego*, Wyższa Szkoła Inżynierska w Zielonej Górze, Zeszyty Naukowe nr 78, Elektrotechnika nr 12, Zielona Góra, 1986, str. 113-125

[Adam90a] M.Adamski, *Projektowanie układów cyfrowych systematyczną metodą strukturalną*, Monografie, Wydawnictwo WSI w Zielonej Górze, Zielona Góra 1990

[Adam91] M.Adamski, *Parallel Controller Implementation using standard PLD Software*, in W. R. Moore, W. Luk (Eds.), FPGA, Abingdon EE&CS Books, Abingdon, England, 1991, pp. 296-304

[Adam98] M.Adamski, *Metodologia projektowania reprogramowalnych sterowników logicznych z wykorzystaniem elementów CPLD i FPGA*, Materiały I Krajowej Konferencji Naukowej - Reprogramowalne Układy Cyfrowe, Szczecin, 12-13 marzec, 1998, str. 15-22

[Adam99] M.Adamski, *Specyfikacja, analiza i synteza reprogramowalnych sterowników logicznych*, Materiały II Krajowej Konferencji Naukowej – Reprogramowalne Układy Cyfrowe, Szczecin, 14-16 marzec, 1999, str. 11-20

[AdBa06] M.Adamski, A.Barkalov, *Architectural and sequential synthesis of digital devices*, University of Zielona Góra Press, Zielona Góra, Poland, 2006, ISBN 83-7481-039-4

[alde06] www.aldec.com

[Alte06] www.altera.com

[Amro90] A.Amroun, *A Petri Net Methodology for the Design of Parallel Controllers*, PhD.Thesis, University of Bristol, Bristol, 1990

[Andr03] G.Andrzejewski, *Programowy model interpretowanej sieci Petriego dla potrzeb projektowania mikrosystemów cyfrowych*, Oficyna Wydawnicza Uniwersytetu Zielonogórskiego, 2003

[Ashe96] P.J.Ashenden, *The Designer's Guide to VHDL*, Morgan Kaufmann Publishers, Inc., 1996

[Atme06] www.atmel.com

[AtSi93] P.M.Athanas and H.F.Silverman, *Processor Reconfiguration through Instruction-Set Metamorphosis*, Computer, pp. 11–18, Mar. 1993

[BaCh97] F.Balarin, M.Chiodo, P.Giusto, H.Hsieh, L.Lavagno, C.Passerone, A.Sangiovanni-Vincentelli, E.Sentovich, K.Suzuki, B.Tabbara, *Hardware/Software Co-Design of Embedded Systems - The Polis Approach*, Kluwer Academic Publishers, 1997, ISBN 0-7923-9936-6

[BaKu93] Z.Banaszak, J.Kuś, M.Adamski, *Sieci Petriego. Modelowanie, sterowanie i synteza procesów dyskretnych*., Wydawnictwo Naukowe Wyższej Szkoły Inżynierskiej w Zielonej Górze, Zielona Góra 1993

[BaLa02] F.Barat, R.Lauwereins, G.Deconinck, *Reconfigurable Instruction Set Processors from a Hardware/Software Perspective*, IEEE Transactions on software engineering, vol. 28, no. 9, 2002

[BaPi05] T.Barć , P.Pieczonka, A.Stasiak, *Optymalizacja algorytmu programowej syntezy sieci Petriego*, Materiały KKE'2005, wyd. Politechnika Koszalińska, Kołobrzeg 2005, Polska

[Bara94] S.Baranov, *Logic synthesis for control automata*, Kluwer Academic Publishers, Netherlands, 1994, ISBN 0-7923-9458-5

[BeKu06] R.Bermbach, M.Kupfer, *Development of a debug module for FGPA-based micro-controller*, Proced. of IFAC workshop, PDeS2006, Brno 2006, ISBN 80-214-3130-X

[Berry91] G.Berry, *Hardware implementation of pure Esterel*, Proceedings of the ACM Workshop on Formal Methods in VLSI Design, January 1991

[BiAu98] L.Bianco, M.Auguin, A.Pegatoquet, *A path analysis based partitioning for time constrained embedded systems*, Proc. of the Int. Workshop on Hardware/Software Codesign, 1998

[BiCh03] J.Bilington, S.Christensen, K.von Hee at al, *The Petri net Markup Language: concepts, technology and tools*, Lecture Notes In Computer Scence, 2679/2003:483-505, 2003

[bige] http://www.bigeneric.com/

[Bili96] K.Biliński, *Application of Petri Nets in parallel controllers design*, PhD. Thesis, University of Bristol, Bristol, 1996

[BuAd06] P.Bubacz, M.Adamski, *Heuristic algorithm for an effective state encoding for reconfigurable matrix-based logic controller design*, Proced. of IFAC workshop, PDeS2006, Brno 2006, ISBN 80-214-3130-X

[CaCh01] J.E.Carrillo, P.Chow, *The effect of reconfigurable units in superscalar processors*, FPGA01, ACM 2001, Monterey, USA

[Celo06] http://www.celoxica.com/

[ChGi93] M.Chiodo, P.Giusto, H.Hsieh, A.Jurecska, L.Lavagno, A.Sangiovanni-Vincentelli, *A formal model specification for hardware/software codesign*, In Proceeding of International Workshop on Hardware-Software Codesign, 1993

[cibl06] www.criticalblue.com

[Clar99] J.Clark, *XSL Transformation (XSLT)*, version 1.0 W3C,1999, www.w3.org/TR/xslt

[Cowa06] www.coware.com

[Cypr06] www.cypress.com

[DAHu94] H.D'Ambrosio, X.Hu, *Configuration-level hardware/software partitioning for real-time systems*, Proc. of the Int. Workshop on Hardware/Software Codesign, 1994

[DaLa97] B.P.Dave, G.Lakshminarayana, N.K.Jha, *COSYN: Hardware-Software Co-synthesis of Embedded Systems*, Proc. Of the Design automation Conference, 1997, pp.703-708

[Dec06] G.Dec, *Rozmyty system ekspertowy jako sprzętowy układ sterowania i kontroli*, Rozprawa doktorska, Politechnika Warszawska, Wydział Elektroniki I Technik Informacyjnych, Warszawa, 2006

[DiHe03] G.Dittmann, A.Herkersdorf, *Multi-Layer Intermediate Representation for ASIP Design and Critical Path Optimization*, Electrical Engineering, IBM Research Report RZ 3484, 02.2003

[DiJh97] R.P.Dick, N.K.Jha, *MOGAC: A multiobjective genetic algorithm for the cosynthesis of hardware-software embedded systems*, Proc. of the Int. Conference on Com-

puter Aided Design, 1997

[DrHa89] D.Drusinsky, D.Harel, *Using Statecharts for hardware description and synthesis*, IEEE Transactions on Computer Aided Design, 1989

[ElKu98] P.Eles, K.Kuchciński, Z.Pend, *System Synthesis with VHDL*, Kluwer Akademic Publishers, Londyn 1998, ISBN 0-7923-8082-7

[ElPe95] P.Eles, Z.Peng, K.Kuchciński, A.Doboli, *System level hardware/software partitioning based on simulated annealing and tabu serach*, Design Automation for Embedded Systems, vol.2, no.1, 1995

[ErHe98] R.Ernst, J.Henkel, Th.Benner, *The Cosyma environment for Hardware/Software cosynthesis of small embedded systems*, Technische Universitat Braunschweig

[GaVa94] D.D.Gajski, F.Vahid, S.Narayan, J.Gong, *Specification and Design of embedded Systems*, Prentice-Hall, New Jersey 1994, U.S.A., ISBN 0-13-150731-1

[GaVa95] D.D.Gajski, F.Vahid, *Specification and Design of Embedded Hardware-Software Systems* , IEEE Design & Test of Computers, vol. 12, no 1, Spring 1995, pp. 53-67

[GaZh00] D.D.Gajski, J.Zhu, R.Domer, A.Gerstlauer, S.Zhao, *Specc: Specification language and methodology*, Kluwer Academik Publishers, Boston, USA, 2000, ISBN 0-7923-7822-9

[Goud05] J.Goud, *Faster and More Flexible Embedded Systems*, XCell Jurnal, 3th/2005, Xilinx 2005

[Guda05] M.Gudarzi, *Co-Synthesis Algorithms:HW/SW Partitioning, Depatment of Computer Enginiering*, Sharif University of Technology, 2001, USA, http://ce.sharif.edu/~ce226/

[Hans04] L.Hansen, *Design Tools Performance that Lowers Your Costs*, Xcell Journal, Xilinx Inc. 2004

[Hart] R.Hartenstein, *Reconfigurable Computing: the Roadmap to a new business model – and its impact on SoC design*, University of Kaiserlauten, Germany

[HaWa97] J.R.Hauser, J.Wawrzynek, *Garp: A MIPS Processor with a Reconfigurable Coprocessor*, IEEE Symposium on {FPGA}s for Custom Computing Machines, IEEE Computer Society Press, Los Alamitos, 1997

[HoWo96] J.Hou, W.Wolf, *Process partitioning for distributed embedded systems*, Proc. of the Int. Workshop on Hardware/Software Codesign, 1996

[ieee05a] 1364-2005 IEEE Standard for Verilog Hardware Description Language, IEEE Standard No:1364-2005 ISBN:0-7381-4850-4, 2005

[ieee05b] IEEE VuSpec: Electronic Design Automation, IEEE Product No:SE144 ISBN:0-7381-4609-9, 2005

[ieee06] www.ieee.com

[Inte06] www.intel.com
www.intel.com/technology/silicon/new_45nm_silicon.htm

[JaZb00] K.Jasiński, P.Zbysiński, *Rekonfigurowany koprocesor systemowy: uniwersalna platforma obliczeniowa*, Przegląd Telekomunikacyjny, nr 10/2000

[JeHu98] J.Jess, J.Hurk, *System Level Hardware/Software Co-design*, Kluwer Akademic Publishers, 1998, ISNB 0-7923-8084-3

[JeMe99] A.A.Jerraya, J.Mermet *System-Level Synthesis*, Kluwer Academic Publishers, 1999, Dordrecht/Boston/ London, Published in cooperation with NATO Scientific Affair Division

[John05] J.Johnson, *Poznać prawdziwy obraz rzeczy*, II Krajowa Konferencja Jakości

Systemów Informatycznych, Warszawa 2005, Polska

[KoDa95] T.Kozlowski, E. L.Dagless, J.M.Saul, M.Adamski and J.Szajna, *Parallel controller synthesis using Petri nets*, In: IEE Proc. - Comput. Digit. Tech., vol. 142, No. 4 , 1995

[kron06] www.kronopol.com.pl

[Kuba04] H.Kubatowa, *Some Aspects of Digital Design*, PhD, Czech Technical University in Prague, Faculty of Electrical engineering, Prague 2004

[KuMi88] D.Ku, G. De Micheli, *Hardware C - a language for hardware design*, Tech. Rep. CSL-TR-88-362, Computer Systems Laboratory, Stanford University, August 1988

[LeCh03] J.Lee, K.Choi, N.D.Dutt, *Automatic Instruction set design through efficient instruction encoding for ASIP*, ACES 2003, University of California, Irvine 2003

[Luce06] www.lucent.com

[MaGr98] J.Madsen, J.Grode, P.V.Knudsen, *Hardware/Software Partitioning using the LYCOS System*, in StWo97, pp. 283-305

[Mart00] P.Martyniuk, *Zagadnienia profilowania programów w języku C w rozwiązywaniu problemu Harware-Software Co-design*, RUC'2000, Szczecin 2000, ISBN 83-87362-21-2

[ment06] www.mentor.com

[Micr06] www.microsoft.com

[Micz06] P.Miczulski, *Metody weryfikacji i syntezy współbieżnych sterowników cyfrowych z wykorzystaniem hierarchicznych sieci Petriego oraz diagramów decyzyjnych*, Politechnika Szczecińska, Wydział Informatyki, 2006

[mimo06] www.mimosys.com

[MiOl98] T.Miyamori, K.Olukotun, *REMARC: Reconfigurable Multimedia Array Coprocessor*, 1998, Computer Systems Laboratory, Stanford University

[MiRo01] P.Mishra, F.Rousseau, N.Dutt, A.Nicolau, *Coprocessor Codesign for Programmable Architectures*, System Level Synthesis Group Laboratoire TIMA 46 av. Felix Viallet 38031 Grenoble cedex FRANCE http://tima-cmp.imag.fr/Homepages/cosmos/SLS.html, Technical Report #01-13 Dept. of Information and Computer Science University of California, Irvine, CA 92697, USA, April 2001

[Misi80] P.Misiurewicz, *Zagadnienia projektowania cyfrowych układów sterowania binarnego*, materiały konferencyjne VIII Krajowej Konferencji Automatyki, Szczecin, 1980, str. 664-670

[MiSk98a] J.Mirkowski, Z.Skowroński, *Translation of C and VHDL specifications into interpreted Petri Nets for Hardware/Software Co-design*, Mixed Design of Integrated Circuits and Systems, pp. 163-168, Kluwer Academic Publishers, 1998, ISBN 0-7923-8116-7

[Mura89] T.Murata, *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, Vol.77, nr 4, Kwiecień 1989

[OASIS] Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/home/index.php

[ÖsBe97] A.Österling, Th.Benner, R.Ernst, D.Herrmann, Th.Scholz, W.YeJ.Staunstrup, W.Wolf, *Hardware/Software Co-Design: Principles and Practice*, Kluwer Academic Publishers 1997

[Park06] K.Parker, *BSDL Version 0.0 Parser Specifications*, Hewlett-Packard Company

Report, www.eda.org/vug_bbs/bsdl.parser

[PeNe] Petri Nets World (PNW), www.informatik.uni-hamburg.de/TGI/PetriNets/

[Petr62] C.A.Petri, *Kommunikation mit Automaten*, PhD Thesis, Schriften des IIM Nr 3, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962

[Poch04] B.Pochopień, *Podstawy techniki cyfrowej*, Wyższa Szkoła Biznesu w Dąbrowie Górniczej, 2004, ISBN 83-88936-15-8

[Poch05] B.Pochopień, *Arytmetyka w systemach cyfrowych*, Oficyna wydawnicza EXIT, Warszawa, 2005, ISBN 83-87674-78-8

[PrPa92] S.Prakash, A.Parker, *SOS: Synthesis of application-specific heterogeneous multiprocessor systems*, Journal of Parallel and Distributed Comp. vol.16, 1992

[Quic06] www.quicklogic.com

[RaBr94] R.Razdan, K.Brace, M.D.Smith, *PRISC Software Acceleration Techniques*, ICCD'94, p.145-149, 1994

[RaNa00] S.Ramanathan, S.K.Nandy, V.Visvanathan, *Reconfigurable filter coprocessor architecture for DSP applications*, Jurnal of VLSI Signal Processing 26, p.333-359, 2000 Kluwer Academic Publishers

[RBS94] R.Razdan, K.S.Brace, M.D.Smith, PRISC Software Acceleration Techniques, IEEE Computer Society, Pages 145 – 149, Washington, USA, 1994, 0-8186-6565-3

[RGCM94] K.Rajesh, Gupta, N.Claudionor Coelho Jr., and Giovanni De Micheli., *Program implementation schemes for hardware-software systems*, Computer, pages 48-55, 1994

[Scha00] S.Scharfenberg, *Automated IP Integration: Gaining Cycle Time Advantage*, TechOnLine Publication, 2000

[Schu01] E.Schueler, *Reconfigurable coprocessors create flexibility in DSP apps*, EE Times, Nov 15 2001, http://www.commsdesign.com/story/OEG20011115S0059

[Skow00] Z.Skowroński, *Translacja specyfikacji funkcjonalnej układów cyfrowych na sieć Petriego dla potrzeb syntezy systemowej*, PhD, Politechnika Szczecińska, Szczecin 2000

[Smit96] D.J.Smith, *HDL Chip Design*, Doone Publications, 1996

[ŚnRu03] P.Śniatała, R.Rudnicki, P.Sydow, *Implementacja sprzętowa deskryptora aktywności ruchu standardu MPEG-7*, KKE03, Wydawnictwo Uczelniane Politechniki Koszalińskiej, Koszalin 2004

[StAd06] A.Stasiak, M.Adamski, *System Petri net specification* , Programmable Devices and Embedded Systems - PDeS 2006 : proceedings of IFAC workshop. Brno, Czechy, 2006 .- Brno, 2006

[Stan04] www.standishgroup.com, www.standishgroup.com/sample_research/PDFpages/q3-spotlight.pdf

[Stas02a] A.Stasiak, *Metody synchronicznej wymiany danych w systemach osadzonych dla potrzeb dekompozycji systemowej*, W: Materiały XVIII Sympozjum Koła Zainteresowań Cybernetycznych. Warszawa, Polska, 2002 .- Warszawa : BEL Studio Sp. z o.o., 2002, s. 98--107 .- ISBN: 83-88442-39-2

[Stas02b] A. Stasiak, M. Michalczak, *Eksperymentalny system CAD dla potrzeb zintegrowanego projektowania układów mikroprocesorowych*, Praca magisterska, Uniwersytet Zielonogórski, Wydział Elektrotechniki Informatyki i Telekomunikacji, Zielona Góra, lipiec 2002

[Stas03a] A.Stasiak, *System dekompozycji funkcjonalnej w projektowaniu zintegrowanym*, OWD'2003, Istebna, PPEE i BSE, ISBN 83-915991-5-9, Polska 2003

[Stas03b] A.Stasiak, *System zrównoleglenia pracy mikrokontrolera sekwencyjnego dla sprzętowo-programowej architektury systemu osadzonego*, Materiały KKE'2003, wyd. Politechnika Koszalińska, Kołobrzeg 2003, Polska, str.375-380, ISBN 83-7365-030-X

[Stas05] A.Stasiak, *Klasyfikacja systemów wspomagających proces przetwarzania i starowania*, KNWS'05, Oficyna Wydawnicza Uniwersytetu Zielonogórskiego, Złotniki Lubańskie 2005, ISBN 83-89712-62-8

[Ster97] A.Sterling, *COSYMA Guidance,* Technische Universitat Braunschweig, 1997

[StMi03] A.Stasiak, M.Michalczak, Z.Skowroński, *Realizacja Wirtualnych Połączeń Programowo-Sprzętowych w Dekompozycji Systemowej z Zastosowaniem Reprogramowalnych Struktur Logicznych*, RUC'03, Pracowania Poligraficzna Politechniki Szczecińskiej, Szczecin 2003, ISBN 83-87362-56-5

[StMi03b] A.Stasiak, M.Michalczak, Z.Skowroński, *Algorytm dekompozycji systemowej dla potrzeb zintegrowanego projektowania*, RUC'03, Pracowania Poligraficzna Politechniki Szczecińskiej, Szczecin 2003, ISBN 83-87362-56-5

[StSk04] A.Stasiak, Z.Skowroński, *The intermediate model for hardware/software Microsystems based on Petri nets*, DESDes'04, Wydawnictwo Uniwersytetu Zielonogórskiego, Dychów 2004, ISBN 83-89712-16-4

[StSk06] A.Stasiak, Z.Skowronski, *Model formalny sprzętowo-programowego systemu cyftowego*, RUC2006, Szczecin 2006, przyjęty do publikacji

[synp06] www.synplicity.com

[Syst06] www.systemc.org

[Texa06] Texas Instruments Inc., "IEEE Std 1149.1 (JTAG) Testability Primer", http://www-s.ti.com/sc/psheets/ssya002c/ssya002c.pdf

[Węgr03] A.Węgrzyn, *Symboliczna analiza układów sterowania binarnego z wykorzystaniem wybranych metod analizy sieci Petriego*, UZ, 2003

[Węgr98] M.Węgrzyn, *Hierarchiczna implementacja kontrolerów cyfrowych z wykorzystaniem FPGA*, Rozprawa doktorska, Politechnika Warszawska, Warszawa, 1998

[WeKi03] M.Weber, E.Kindler, *The Petri net markup language*, Lecture Notes In Computer Scence, 2472/2003:124-144, 2003

[wina06] winavr.sourceforge.net/

[Wola98] P.Wolański, *Modelowanie układów cyfrowych na poziomie RTL z wykorzystaniem sieci Petriego i podzbioru języka VHDL*, Rozprawa doktorska, Warszawa 1998

[xcel06] www.xilinx.com/publications/xcellonline/xcell_58/index.htm

[Xili06] www.xilinx.com,
www.xilinx.com/company/success/appsig.htm

[Xili06a] www.xilinx.com/ise/embedded/mb_ref_guide.pdf

[Xili06b] www.xilinx.com/ise/embedded/est_rm.pdf

[ZbŁu00] B.Zbierzchowski, T.Łuba, *Komputerowe projektowanie układów cyfrowych*, Wydawnictwa Komunikacji i Łączności, Politechnika Warszawska, Polaska, 2000, ISBN 83-206-1364-7

[Ziel03] C.Zieliński, *Podstawy projektowania układów cyfrowych*, PWN, Warszawa, 2003

[ziel06] www.ec.zgora.pl

Dodatek A

Opis znaczników formatu SPNF.

LP	Znacznik	Opis
0	`<ADES name="project"> </ADES>`	Znacznik główny – projektu ADES interpretowany jako ROOT w standardzie XML. Znacznik wymagany: **SystemDescription**(1) Znacznik opcjonalny: gfx(), decompositionConfig(), DFG(), inne
1	`<SystemDescription name="system_name"> </SystemDescription>`	Otwiera i zamyka opis funkcjonalności całego systemu. Znacznik może wystąpić tylko raz. Znacznik wymagany: **model**(8) Znacznik opcjonalny: resolution(2), clock(3)
2	`<resolution></resolution>`	Definicja jednostki czasu dla całego systemu. Znacznik „resolution" występuje tylko w bloku „SystemDescription", tylko raz. Znacznik może przyjmować wartości: `PS` (-12 sekundy), `NS` (-9 S), `US` (-6 S), `MS` (-3 S), `S`. Na podstawie podanej jednostki przeliczane są wszystkie wartości znaczników „waitfor"(31). Jeśli znacznik „resolution"nie występuje, przyjmowana jest domyślna jednostka: `NS`. Znacznik: brak.
3	`<clock name="clock_name"> </clock>`	Definicja zegara systemowego, jego nazwy. Może być tylko jeden zegar w systemie. Podanie nazwy oznacza, że system jest synchroniczny. Jeśli nazwa zegara ma wartość „" (jest pusta), wówczas system traktowany jest jako automat asynchroniczny – znacznik edge(4) jest pomijany. Znacznik wymagany: **edge**(4)
4	`<edge></edge>`	Deklaracja aktywnego zbocza zegara systemowego. Może przyjmować wartości tekstowe: `rising` (narastające) lub `falling` (opadające). Znacznik „edge" może występować tylko raz w deklaracji obiektu „clock"(3) Znacznik: brak.
5	`<predicat name="predicat_name"> </predicat>`	Definicja warunku logicznego. Obiekt może być deklarowany w dowolnym miejscu w modelu (na poziomie miejsc i tranzycji). Zadeklarowana nazwa warunku może zostać użyta wyłącznie w znaczniku „transition"->"condition". Nazwa warunku musi być unikatowa. Liczba predykatów nie jest ograniczona. Znacznik: brak.
6	`<include file="file_name"> </include>`	Załączenie modeli rezydujących w innym pliku o nazwie "file_name". Znacznik może zostać użyty na poziomie deklaracji modeli. Jeśli nie zadeklarowano znacznika „part"(7), wówczas do aktualnego systemu załączane są wszystkie modele rezydujące w systemie/pliku „file_name". Znaczniki opcjonalne: part(7)
7	`<part>model_name</part>`	Wskazuje nazwę modelu „model_name" z pliku „file_name" znacznika „include"(6), który ma zostać w całości załączony do bieżącego systemu. Znacznik: brak.
8	`<model name="model_name"> </model>`	Deklaracja opisu zachowania poszczególnego modelu (komponentu) systemu. Model musi posiadać zdefiniowany interfejs wejścia/wyjścia(9). Model musi posiadać unikatową nazwę. Można definiować dowolną liczbę bloków typu „model" w systemie. Jeden z obiektów „model" powinien być nadrzędny (nie jest to wymagane), wówczas jego interfejs(9) jest interfejsem we/wy systemu. Jeśli w opisie systemu jest kilka modeli równorzędnych wówczas zbiór ich interfejsów(9) we/wy jest interfejsem systemu. Nazwy zmiennych lokalnych są współdzielone pomiędzy modelami równorzędnymi. Znaczniki wymagane: **interface**(9), **transition**(13), **place**(18) Znaczniki opcjonalne: predicat(5)

9	<interface></interface>	Deklaracja portów wejścia/wyjścia modelu. Dla obiektu „model" może występować tylko jedna deklaracja „interface". Deklaracja wejść/wyjść realizowana jest poprzez znaczniki: input(8), output(9). Znaczniki wymagane: **output**(11) Znaczniki opcjonalne: input(10)
10	<input name="input_name"> </input>	Deklaracja portu wejściowego. Znacznik definiuje tylko jedną nazwę portu. Liczba znaczników „input" nie jest ograniczona. Znaczniki opcjonalne: wide(12)
11	<output name="output_name"> </output>	Deklaracja portu wyjściowego. Znacznikach definiuje jedną nazwę portu. Liczba znaczników „output" nie jest ograniczona. Znaczniki opcjonalne: wide(12)
12	<wide></wide>	Deklaracja szerokości magistrali wejściowej lub wyjściowej. Znacznik przypisany tylko dla obiektów „input", „output". Znacznik przyjmuje wartość jako liczbę naturalną zawsze w orientacji $X_{n-1}...X_0$.
13	<transition name="trans_name"> </tansition>	Blok definiujący tranzycję. Liczba tranzycji w modelu nie jest ograniczona. Tranzycja musi posiadać unikatową nazwę. Tranzycja aktywowana jest, gdy wszystkie miejsca wskazane w znaczniku „sensitive"(14) są aktywne (wykonały się) - wówczas tranzycja jest spełniona (odpalona), zaznacza/uruchamia wszystkie miejsca wskazane w znacznikach „action"(16). Dla tranzycji może zostać przypisany dodatkowy warunek logiczny *odpalenia* za pomocą znacznika „condition"(17) (interpretowane sieci Petriego). Tranzycja może być uwarunkowana czasem, znacznik „waitfor"(); Znaczniki wymagane: **sensitive**(14), **action**(16) Znaczniki opcjonalne: permit(15), condition(17), htime(26), stime(27), sresource(29), hresource(28), activity(30), implement(31), waitfor(32), color(33)
14	<sensitive></sensitive>	Deklaracja miejsc(a), które *odpala(ją)* tranzycje. (tranzycja jest „czuła" na aktywność tych miejsc). Znacznik pobiera jedną nazwę miejsca. Liczba znaczników „sensitive" w tranzycji nie jest ograniczona. (strzałka wejściowa do tranzycji) Znaczniki: brak
15	<permit></permit>	Deklaracja zezwalająca lub nie zezwalająca na spełnienie tranzycji w zależności od stanu miejsca (czy w miejscu jest marker, czy nie). Wartością znacznika jest nazwa miejsca oraz opcjonalnie nazwa modelu do którego dane miejsce należy. Przykład: <permit>P8=1</permit> (warunek „permit" tranzycji będzie tak długo spełniony, dopóki w miejscu P8 będzie marker, tranzycja nie zabiera markera z P8) <permit>P8=0</permit> (warunek „permit" tranzycji będzie tak długo spełniony dopóki w miejscu P8 NIE będzie markera, tranzycja nie zabiera markera z P8) <permit>counter.P8=0</permit> (warunek „permit" tranzycji będzie tak długo spełniony dopóki w miejscu P8 modelu <counter> NIE będzie markera, tranzycja nie zabiera markera z P8). Model „counter" jest równorzędny do rozpatrywanego modelu. Znaczniki: brak
16	<action></action>	Deklaracja miejsc(a), które zostaną uruchomione (zaznaczone) po *odpaleniu* tranzycji. Znacznik pobiera jedną nazwę miejsca. Liczba znaczników „action" w tranzycji nie jest ograniczona. (strzałka wyjściowa do tranzycji) Znaczniki: brak
17	<condition></condition>	Deklaracja warunku logicznego *odpalenia* tranzycji. Warunek składa się z nazw sygnałów portów wejściowych i nazw sygnałów wewnętrznych. Liczba znaczników „condition" w tranzycji nie jest ograniczona. Przyjmuje nazwy zadeklarowanych predykatów. Przykłady formatu zapisu: <condition>Y=1</condition> <condition>Y=(A+B) or (C*D)</condition> <condition>Y+(2*X)=(A+B) or (C*D)</condition> <condition>Y/=2</condition>

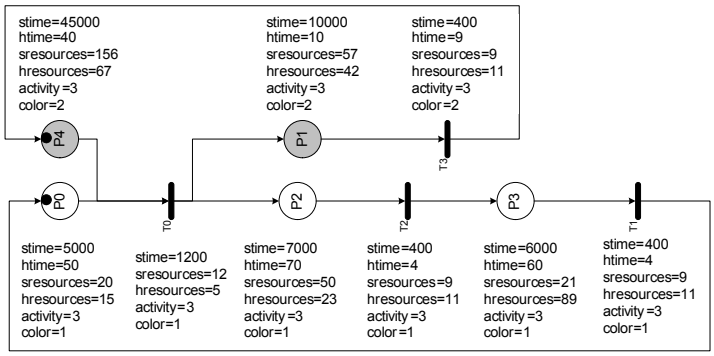
		Znaczniki: brak
18	<place name="place_name"> </place>	Deklaracja miejsca w modelu. Liczba miejsc w modelu nie jest ograniczona. Miejsce może być definiowane na trzy sposoby: 1. *simple place* Miejsce musi posiadać unikatową swoją nazwę. Miejsce aktywowane jest tylko i wyłącznie przez tranzycje. Miejsce może być inicjujące - początkowy stan modelu. Wówczas pomiędzy znacznikami obiektu „place" znajduje się znacznik „initial"(19). Liczba miejsc inicjujących w modelu nie jest ograniczona. Znaczniki wymagane: brak Znaczniki opcjonalne initial(17), produce(23) htime(26), stime(27), sresource(29), hresource(28), activity(30), implement(31), waitfor(32), color(33) 2.*macroplace (info)* *Macroplace* (makro-miejsce) instancjonuje model(8) zadeklarowany i opisany w systemie (na poziomie opisu modeli). Każdy model może zostać użyty jako *macroplace*. Za pomocą makro-miejsc konstruuje się hierarchię strukturalną systemu. Makro-miejsce nie może instancjonować modelu, w którym zostało zadeklarowane. Każde makromiejsce jest aktywowane i deaktywowane przez nadrzędny model (tranzycje), który instancjonuje dane makromiejsce. Obowiązują dwie zasady: a)zakończenie pracy makromiejsca nie jest zależne od stanu pracy danego makromiejsca, b)aktywacja makromiejsca polega na przekazaniu znacznika do wszystkich miejsc inicjalizujących „initial" instancjonowanego modelu, c) zakończenie pracy makromiejsca zależne jest od tranzycji „zabierającej" marker z makromiejsca (w modelu nadrzędnym). Makro-miejsce może być opisane w bieżącym lub zewnętrznym pliku XML – znacznik „source"(20) wskazuje lokalizacje. Wybrany model (rozumiany jako komponent) może być wielokrotnie instancjonowany (używany) poprzez deklaracje kolejnego obiektu typu „macroplace". 2.1."*shared macroplace*" Instancjonowanie obiektu *shared macroplace* dokonuje się poprzez podanie nazwy modelu(8) (używanego jako komponent) w znaczniku „component" oraz połączenie sygnałów lokalnych komponentu z aktualnymi systemu, znacznik „bind"(21). Każde *macroplace* typu *shared* musi posiadać swą unikalną nazwę. Przepływ danych pomiędzy komponentem i modelem realizowany jest tylko i wyłącznie przez interfejs komponentu, przy aktywnym stanie miejsca. Wewnętrzna struktura makro miejsca nie jest dostępna. Znaczniki wymagane: **component**(22), **bind**(21), **source**(20) Znaczniki opcjonalne: htime(26), stime(27), sresource(29), hresource(28), activity(30), implement(31), waitfor(32), color(33) 2.2.*procedural macroplace* Instancjonowanie modelu(8) jako *macroplace* typu *procedural* realizowane jest poprzez podanie nazwy wybranego modelu (traktowanego jako komponent) w znaczniku component(22) obiektu „place". Deklaracja „procedural macroplace" nie przyjmuje znacznika bind(21) – w ten sposób rozróżniemy makromiejsca typu *shared* i *procedural*. Lokalny interfejs wej/wyj modelu (instancjonowanego jako komponent) zostaje automatycznie dodany do interfejsu wej/wyj systemu. Przepływ danych pomiędzy komponentem i modelem realizowany jest za pomocą sygnałów wewnętrznych komponentu i modelu, które stanowią wspólny zbiór. Również nazwy miejsc i tranzycji są wspólne (widoczne zarówno w komponencie i modelu) Jeśli w systemie zadeklarowano więcej niż jedno makro-miejsce typu *procedural* instancjonujące ten sam model, wówczas niezbędne jest wyróżnienie zbioru interfejsu, miejsc i tranzycji makro miejsca X i makro miejsca Y. Stosuje się następujące schematy: a) nazewnictwo sygnałów zainstancjonowanego makromiejsca *procedural* jest ustalane i rozpoznawane w systemie wyższego rzędnym

		według schematu: *<nazwa_makto-miejsca>.<nazwa_sygnału>* b) nazewnictwo miejsc i tranzycji jest ustalane i rozpoznawane w systemie wyższego rzędu analogicznie do nazw sygnałów: *<nazwa_makro-miejsca>.<nazwa_miejsca>* *<nazwa_makro-miejsca>.<nazwa_tranzycji>* Znaczniki wymagane: **component**(22), **source**(20) Znaczniki opcjonalne: pause(), htime(26), stime(27), sresource(29), hresource(28), activity(30), implement(31), waitfor(32), color(33)
19	`<initial></initial>`	Deklaracja miejsca początkowego (inicjującego, startowego). Może przyjąć wartość: 0 (miejsce nie jest inicjujące), 1 (miejsce inicjujące). Tylko jeden znacznik „initial" jest dozwolony w obiekcie „place". Znacznik „initial" może być pominięty w opisie „place" bądź posiadać wartość 0, wówczas miejsce nie jest inicjujące. Znaczniki: brak
20	`<pause></pause>`	Deklaracja funkcjonalności makro miejsca. Jeśli znacznik pause=0, wówczas utracenie znacznika przez makro miejsce wprowadza sieć oraz sygnały/zmienne sieci w stan początkowy (zeruje sieć). W przeciwnym razie, pause='1', utracenie znacznika przez makro miejsce powoduje wstrzymanie pracy instancjonowanej sieci (blokada pracy, wstrzymanie). Stan instancjonowanej sieci jest podtrzymywany do chwili globalnego zerowania systemu lub ponownego otrzymania znacznika przez nadrzędne makro miejsce, który to znacznik„odblokowuje" sieć wznawiając jej pracę. Znaczniki: brak
21	`<source></source>`	Deklaracja lokalizacji opisu funkcjonalnego makro-miejsca. Znacznik „source" może przyjmować wartości: *current* (model w aktualnym pliku XML) lub „<bezwzględna ścieżka do pliku>" (opis znajduje się w zewnętrznym pliku XML). Znacznik „source" może zostać podany tylko raz w opisie obiektu „place". Znaczniki: brak
22	`<bind></bind>`	Łączenie sygnałów lokalnych obiektu „component" (użytego modelu) z aktualnymi sygnałami modelu nadrzędnego. Łączenie poprzez kolejność: Każdy kolejny znacznik „bind" łączy podaną nazwę sygnału aktualnego z kolejną/odpowiednią nazwą sygnału lokalnego (zadeklarowanego w bloku „interface" modelu) komponentu. Znacznik „bind" pobiera nazwę sygnału wejściowego, wyjściowego lub sygnału wewnętrznego modelu, gdzie zadeklarowano „place" typu „macroplace". Składnia: <bind>x1</bind> Liczba znaczników „bind" w obiekcie „place" zależy od ilości deklaracji wej/wyj w bloku „interface" danego komponentu (instancjonowanego modelu). Znaczniki: brak
23	`<component></component>`	Instancjacja poprzez nazwę modelu jako makro-miejsca typu „shared". Nazwa podana w znaczniku „component" musi być przypisana do jednego modelu w systemie. Znaczniki: brak
24	`<produce ID="produce_type" HOLD="1/0">` `</produce>`	Ustawienie odpowiedniej wartości sygnału wyjściowego lub wewnętrznego w chwili uzyskania znacznika przez miejsce(18). Znacznik „produce" przyjmuje zapis: A=0; A=123; B=((A+C)*2)/4; itp. Liczba znaczników „produce" w miejscu nie jest ograniczona. Znacznik posiada dwa atrubuty: • ID: o ID="REG"; (registered) aktualizacja wartości sygnału/zmiennej dozwolona jest tylko przy aktywnym zboczu zegara systemowego, podczas gdy miejsce przetrzymuje znacznik, o ID="COMB"; (combinational) aktualizacja wartości sygnału/zmiennej dozwolona jest podczas, gdy miejsce przetrzymuje znacznik (zegar nie jest istotny), tzw. przypisanie ciągłe.

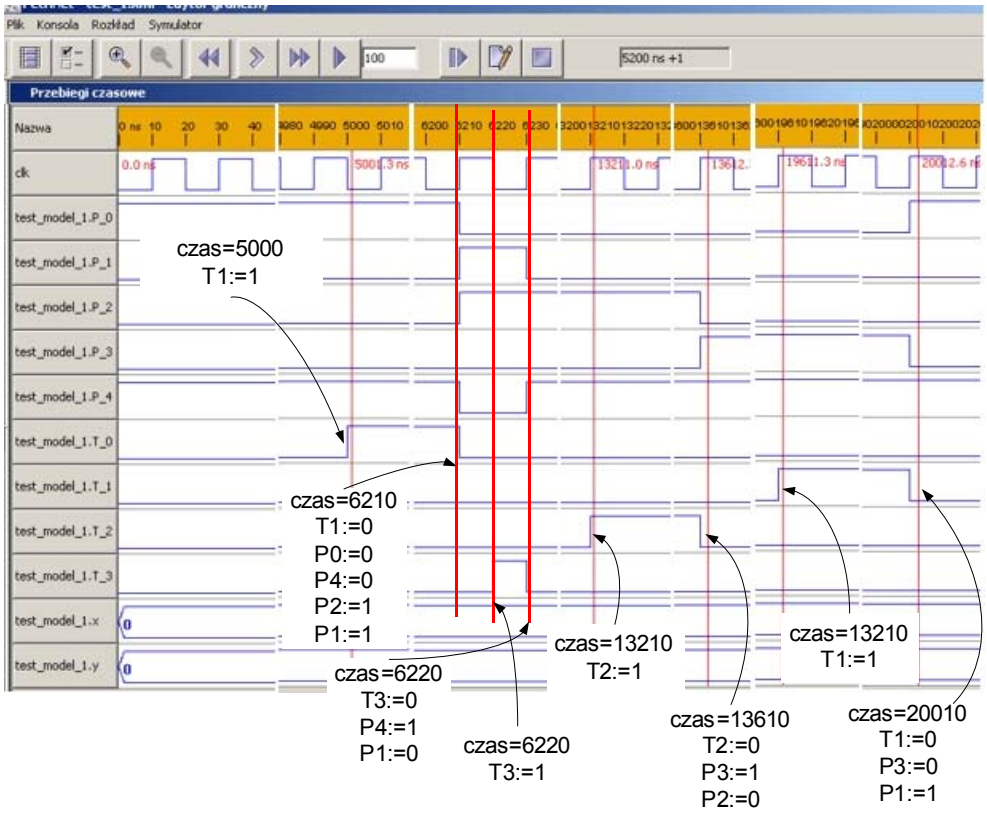
		W przypadku, gdy nie podano żadnego atrybutu, domyślnie przyjmowany jest atrybut REG. • HOLD; definiuje, czy dana przypisywana do zmiennej w znaczniku „produce" powinna być utrzymana/podtrzymana po wyjściu z miejsca „place", czy zerowana: o HOLD=1; podtrzymuj wartość zmiennej, o HOLD=0; 0 lub brak atrybutu = zeruj wartość zmiennej W przypadku, gdy nie zdefiniowano atrybutu „hold", przyjmuje się wartość domyślną „hold=0". Znacznik: brak
25	<htime></htime>	Wartość czasu wykonania danej tranzycji/miejsca/makroM jako implementacji sprzętowej (VHDL). Wartością znacznika „htime" jest czas podany jako liczba naturalna. Systemowa jednostka czasu. Znaczniki: brak
26	<stime></stime>	Wartość czasu wykonania danej tranzycji/miejsca/makroM jako implementacji programowej (C). Wartością znacznika „stime" jest czas podany jako liczba naturalna. Systemowa jednostka czasu. Znaczniki: brak
27	<hresource></hresource>	Wartość liczbowa, wyrażona w [CLB], zasobów sprzętowych niezbędnych do realizacji danej tranzycji/miejsca/makroM. Wartością znacznika jest liczba CLB dla wybranego układu FPGA. Znaczniki: brak
28	<sresource></sresource>	Wartość liczbowa, wyrażona w [kB], rozmiaru kodu C niezbędnego do realizacji tranzycji/miejsca/makroM dla wybranego mikrokontrolera (AVR, 8051). Znaczniki: brak
29	<activity></activity>	Aktywność tranzycji/miejsca/makroM wyznaczona w sposób dynamiczny podczas symulacji funkcjonalnej systemu. Znacznik przyjmuje liczbę naturalną. Znaczniki: brak
30	<implement></implement>	Deklaracja realizacji sprzętowej lub programowej miejsca bądź tranzycji. Możliwe wartości parametru (not case sensitive): S ; implementacja programowa H ; implementacja sprzętowa Jeśli znacznik jest pusty, oznacza że model/komponent nie został poddany analizom SH.
31	<waitfor></waitfor>	Deklaracja opóźnienia wykonania tranzycji/miejsca/makroM po spełnieniu wszystkich warunków tranzycji lub wykonaniu wszystkich zadań przez miejsce. Znacznik przyjmuje wartość liczbową czasu jako liczbę naturalną, systemowa jednostka czasu. Jest to parametr czasowy systemu/komponentu wykorzystywany podczas symulacji funkcjonalnej sieci. Wartość znacznika ulega zmianie w przypadku, gdy znacznik „implement"(31) nie jest pusty. Możliwe są dwa przypadki automatycznej aktualizacji/napisania aktualnej wartości znacznik „waitfor": 1. gdy „implement" =S => „waitfor" = „stime" 2. gdy „implement" =H => „waitfor" = „htime" Natomiast, gdy „implement"=' ', to „waitfor"="waitfor". Znaczniki: brak
32	<color></color>	Wartość przypisanego koloru do tranzycji/miejsca/makroM, wyznaczonego podczas analizy całego modelu ze względu na składowe automatowe. Podawana jest liczba naturalna. Znaczniki: brak

Dodatek B

Dla każdego miejsca sieci z rysunku B.1, przydzielono w sposób syntetyczny koszt czasu realizacji zadania dla części programowej i sprzętowej.



Rysunek B.1 Specyfikacja modelu po procesie analiz SPMC

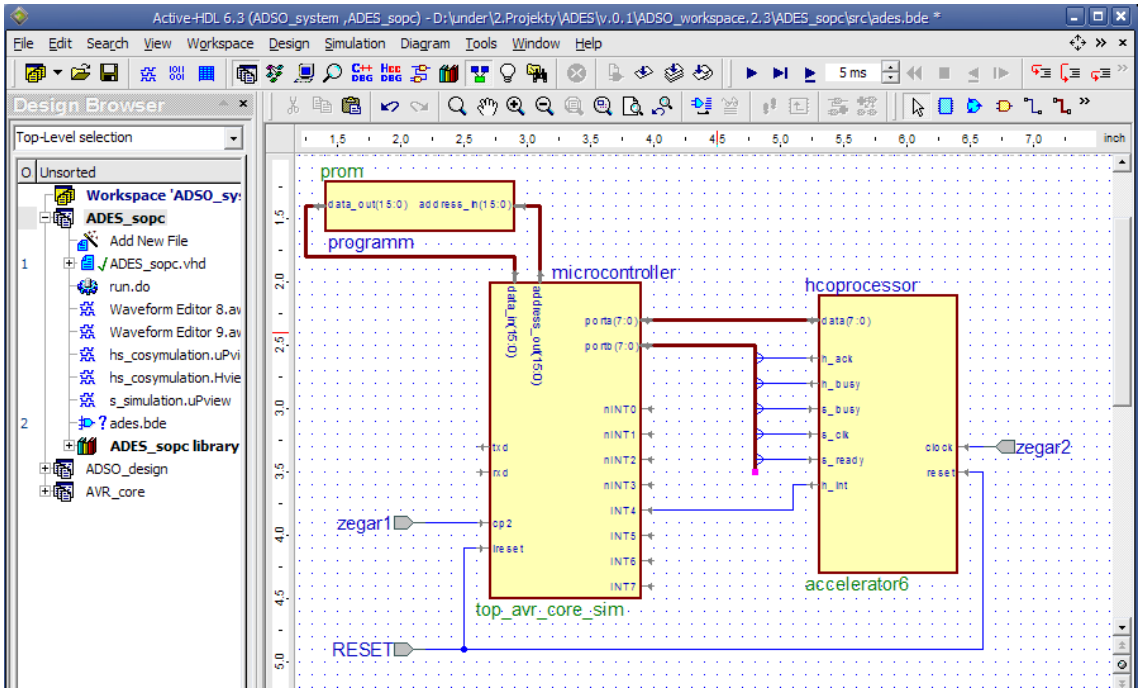


Rysunek B.2 Przykład procesu kosymulacji w opracowanym symulatorze CoSPeN

Na rysunku B.2 przedstawiono wyniki pracy symulatora CoSPeN. Cykl pracy badanego modelu SPMC wynosi 20010 [ns]. Natomiast minimalny czas cyklu pracy tej samej specyfikacji wykonywanej przez mikroprocesor wynosi 75400[ns]. Wydajność procesu przetwarzania zadanej przykładowej specyfikacji, realizowanej przez mikrostrukturę SPMC, wzrosła 3,7-krotnie.

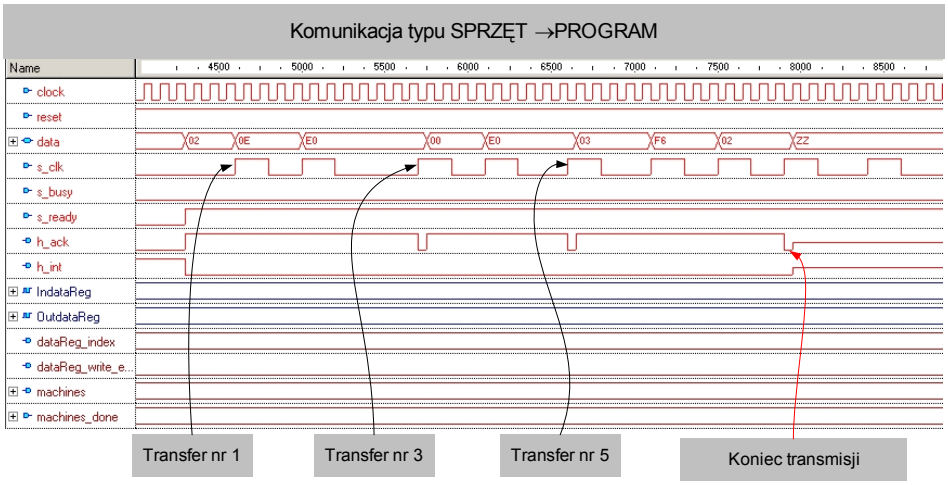
Dodatek C

Budowę blokową środowiska wirtualnej symulacji SPMC przedstawia rysunek C.1. Natomiast wyniki przykładowej symulacji, w transmisji program↔sprzęt, przedstawiają rysunki C.2 i C.3. Pierwszym przykładem jest transmisja typu program→sprzęt. Po nawiązaniu połączenia program-sprzęt, w szeregu wysyłanych jest sześć identyfikatorów kanału komunikacyjnego oraz właściwe pakiety danych.



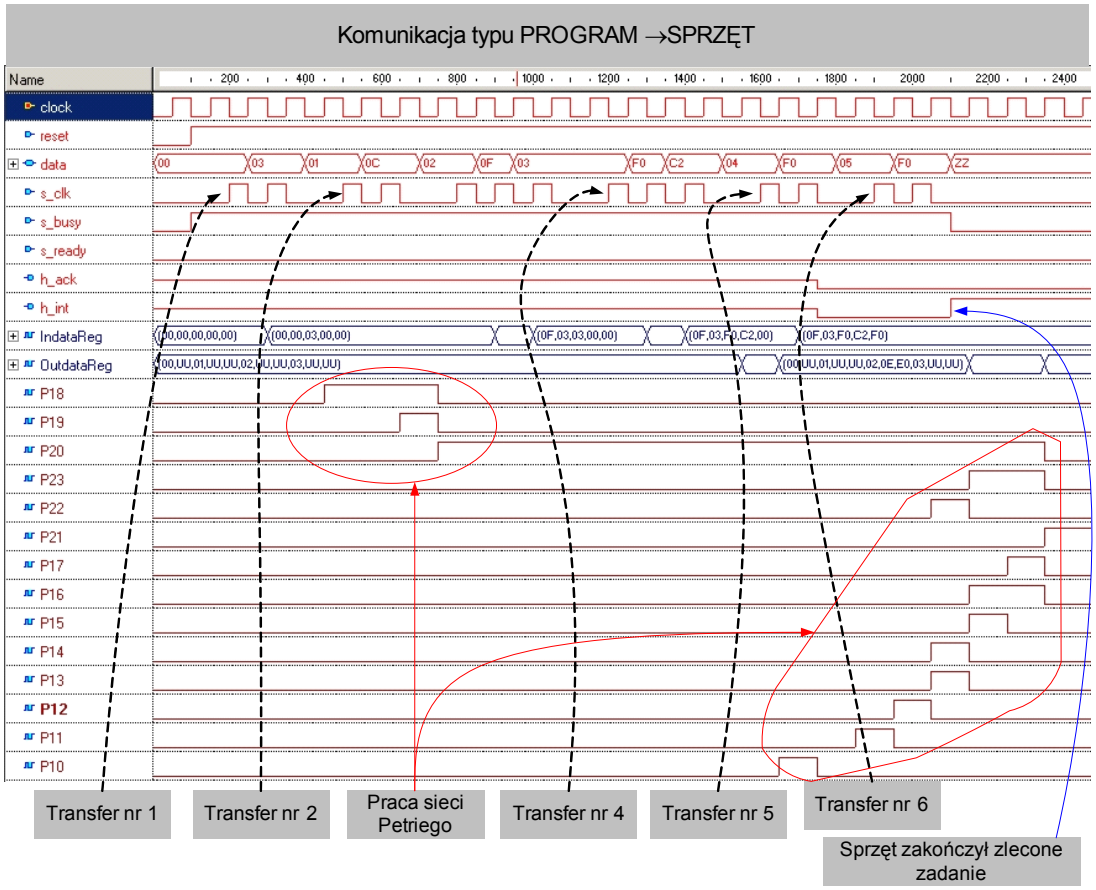
Rysunek C.1 Środowisko wirtualnej symulacji mikrostruktury SPMC

Proces transmisji sprzęt→program wymaga większych interwałów czasu sygnału *s_clk* ze względu na konieczność potwierdzenia przez mikroprocesor otrzymanych danych.



Rysunek C.2 Walidacja programowo-sprzętowa w środowisku wirtualnym – przykład komunikacji typu sprzęt-program

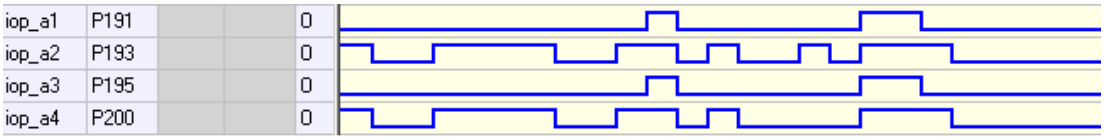
Każdy komunikator kanału skojarzony jest z jednym zleceniem wykonania zadanie przez część sprzętową.



Rysunek C.3 Współsymulacja programowo-sprzętowa w środowisku wirtualnym – przykład komunikacji typu program-sprzęt

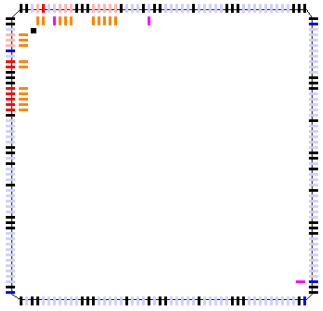
Dodatek D

Wyniki w programie JTAG-SIM prezentowane są na dwa sposoby: a) za pomocą wykresów przebiegów czasowych (ang. waveform), rysunek D.1, b) poglądowy obiekt obudowy układu FPGA z wyprowadzeniami (ang. chip view), rysunek D.2.



Rysunek D.1 Graficzna prezentacja wyników testu SPMC w układzie FPGA

Wykresy przebiegów służą głównie do przeprowadzania testów, w których uzyskane dane będą w późniejszym czasie analizowane bądź porównywane z innymi wynikami symulacji funkcjonalnej. Doskonałym przykładem takich testów może być przeprowadzanie symulacji funkcjonalnej w sprzęcie za pomocą instrukcji INTEST z wykorzystaniem skryptu testującego (ang. testbench). Taki test można przeprowadzać dla kolejnych wersji rozwiązań SPMC, a następnie porównać zmiany różnych wersji implementacyjnych SPMC.

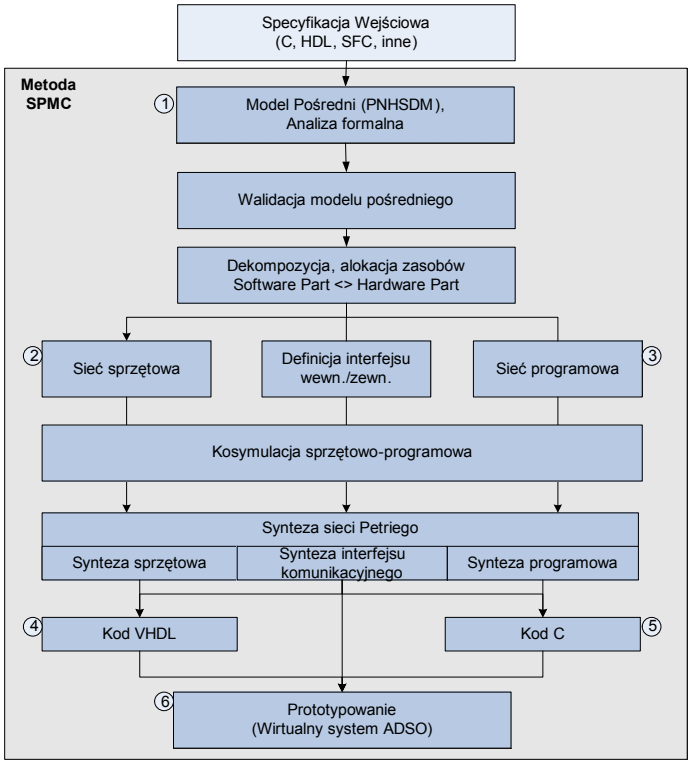


Rysunek D.2 Graficzna forma prezentacji procesu weryfikacji funkcjonalnej –widok portów wejścia/wyjścia układu FPGA

Widok układu w procesie symulacji JTAG–SIM pozwala na szybką prezentację wszystkich odczytanych wartości wyprowadzeń. Ponieważ w tym widoku nie jest zapamiętywana historia poprzednich wartości, widok ten należy używać do weryfikacji ogólnej.

Dodatek E

Proces projektowy sprzętowo-programowej mikrostruktury cyfrowej składa się szeregu etapów wchodzących w skład opracowanej w rozprawie metody SPMC. Cały proces projektowy może być w całości zautomatyzowany. Wszystkie zadania, z wyłączeniem opracowania specyfikacji wejściowej oraz weryfikacji funkcjonalnej (które muszą zostać przeprowadzone przez człowieka), realizowane są przez oprogramowanie przedstawione w rozdziale szóstym rozprawy, rysunek 6.3. W niniejszym załączniku, z wykorzystaniem prostego przykładu, przedstawiono kompletną ścieżkę projektową metody SPMC. Rysunek E1 przedstawia graficzny diagram etapów projektowania SPMC. Numerami oznaczono punkty, których charakterystykę szczegółowo opisano w dalszej części załącznika.

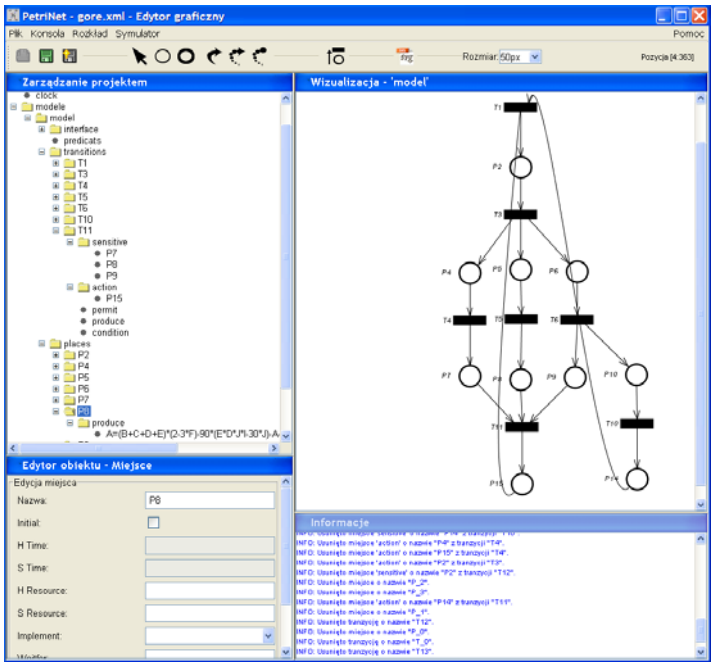


Rys. E1 Etapy projektowania SPMC

AD 1. Specyfikacja wejściowe procesu projektowego, nawiązując opisu z rozdziału czwartego, może zostać podana w postaci kodu np. C, HDL, SFC, SystemC, inne. W procesie translacji specyfikacja wejściowa poddawana jest zamianie na model pośredni – interpretowane, hierarchiczne sieci Petriego. Dodatkowo, możliwe jest opracowanie i wprowadzenie do środowiska projektowego SPMC specyfikacji systemu, opisanego bezpośrednio sieciami Petriego. W tym celu można wykorzystać opracowany w ramach rozprawy edytor sieci Petriego CoSPen, rysunek E.2 a). Format SPNF został opracowany w standardzie XML, dzięki czemu zapewniono integrację opracowanych w pracy programów z innymi narzędziami CAD (edytory, kompilatory) poprzez arkusz XSTL. Model pośredni w postacie elektronicznej reprezentowany jest jako zapis tekstowy w formacie SPNF.

Rysunek E.2 a) przedstawia specyfikację wejściową w postaci graficznej sieci Petriego, natomiast rysunek E.2 b) reprezentację tej specyfikacji w formacie SPNF.

a) reprezentacja graficzna



b) zapis tekstowy SPNC

```
<ADES name="Projekt">
<SystemDescription name="Siec">
    <resolution>NS</resolution>
    <clock name="">
        <edge>rising</edge>
    </clock>
<model name="model">
<interface>
    <input name="A"></input>
    <input name="B"></input>
    <input name="C"></input>
    <input name="D"></input>
    <input name="E"></input>
    <output name="AA"></output>
    <output name="BB"></output>
    <output name="CC"></output>
    <output name="DD"></output>
    <output name="EE"></output>
</interface>
<place name="P2">
    <initial>0</initial>
    <produce>E=D*2-20;</produce>
    <produce>E=2*E;</produce>
    <produce>E=A+B;</produce>
</place>
<place name="P4">
    <initial>0</initial>
</place>
<place name="P5">
    <initial>0</initial>
</place>
<place name="P6">
    <initial>0</initial>
</place>
<place name="P7">
    <initial>0</initial>
</place>
<place name="P8">
    <initial>0</initial>
    <produce>A=(B+C+D+E)*(2-3*F)-90*(E*D*J*I-
30*J)-A-20+30/4*32*(45-2*(B*7)-34565)-
3333;</produce>
</place>
<place name="P9">
    <initial>0</initial>
</place>
<place name="P10">
    <initial>0</initial>
</place>
<place name="P14">
    <initial>0</initial>
</place>
<place name="P15">
    <initial>0</initial>
</place>
<transition name="T1">
    <sensitive>P15</sensitive>
    <sensitive>P14</sensitive>
    <action>P2</action>
</transition>
<transition name="T3">
    <sensitive>P2</sensitive>
    <action>P4</action>
    <action>P5</action>
    <action>P6</action>
</transition>
<transition name="T4">
    <sensitive>P4</sensitive>
    <action>P7</action>
</transition>
<transition name="T5">
    <sensitive>P5</sensitive>
    <action>P8</action>
    <condition>A=(B+C+D+E)*(2-3*F)-90*(E*D*J*I-
30*J)-A-20+30/4*32*(45-2*(B*7)-34565)-
3333</condition>
</transition>
<transition name="T6">
    <sensitive>P6</sensitive>
    <action>P9</action>
    <action>P10</action>
</transition>
<transition name="T10">
    <sensitive>P10</sensitive>
    <action>P14</action>
</transition>
    <transition name="T11">
    <sensitive>P7</sensitive>
    <sensitive>P8</sensitive>
    <sensitive>P9</sensitive>
    <action>P15</action>
</transition>
</model>
</SystemDescription>
```

Rys. E2 Specyfikacja wejściowa: a) reprezentacja graficzna, b) zapis tekstowy SPNF

Model pośredni (zapis SPNF w standardzie XML) służy jako obiekt, na którym dokonywane są wszelkie analizy formalne i funkcjonalne. Celem jest eliminacja błędów projektowych oraz przygotowanie danych pośrednich niezbędnych w procesie dekompozycji funkcjonalnej SPMC.

Przykład analizy programowej przedstawia rysunek E3. Specyfikacja wejściowa w zapisie pośrednim SPNF poddawana jest syntezie programowej oraz kompilacji kodu wynikowego C dla wybranego procesora. Celem jest oszacowanie kosztów realizacji programowej (czas wykonania, rozmiar kodu) wszystkich komponentów specyfikacji wejściowej, tj. miejsc (ang. place) i tranzycji (ang. transition) sieci Petriego. Równorzędne operacje przeprowadzane są w procesie analiz realizacji sprzętowej.

```
Wartosc znacznika <stime> dla miejsca P2 jest rowna 1025.
Wartosc znacznika <stime> dla miejsca P8 jest rowna 5206.
Wartosc znacznika <stime> dla miejsca P9 jest rowna 624.
Wartosc znacznika <stime> dla tranzycji T5 jest rowna 1150.
Wartosc znacznika <stime> dla tranzycji T10 jest rowna 484.
```

Rys. E3 Przykład wyników analizy programowej sieci Petriego

Wyniki procesów analiz składowane są w dedykowanych znacznikach specyfikacji SPNF (np. <stime> dla parametru czasu wykonania części programu). W ten sposób zapewniono konsolidację danych, opisu i konfiguracji w jednym pliku projektu, jednocześnie nie zakłócając przejrzystości i czytelności wszystkich informacji pozyskanych w procesie przetwarzania/analiz, rysunek E4.

```
<ADES name="Projekt">
<SystemDescription
name="Siec">
  <model name="controler">
  </model>
</SystemDescription>
<dfg>
</dfg>
<decompositionConf
name="Siec">
</decompositionConf>
<gfx>
</gfx>
</ADES>
```

ADES – deklaracja projektu

SystemDescription – specyfikacja zachowania systemu

dfg – graf przepływu danych

decompositionConfig – konfiguracja podziału sieci ,wynik procesu dekompozycji funkcjonalnej

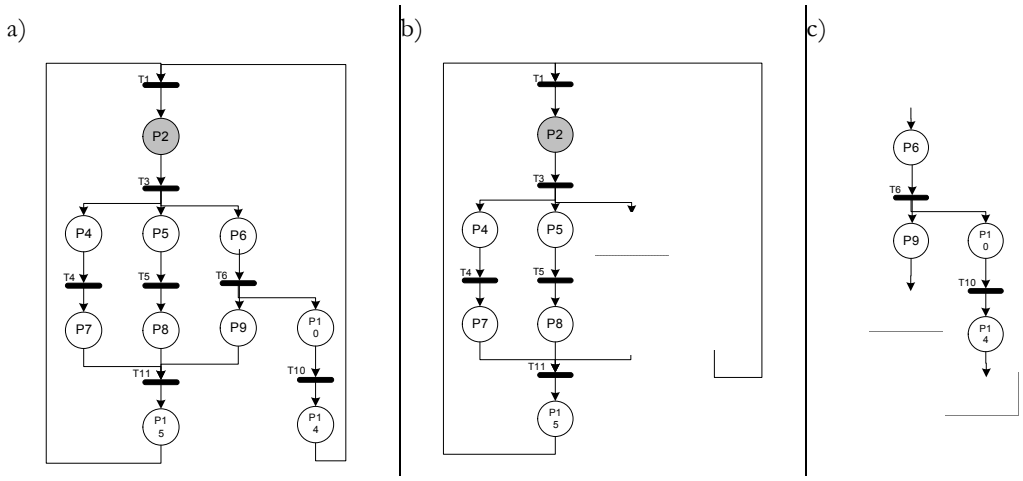
gfx – konfiguracja reprezentacji graficznej

Dane opisujące właściwości miejsc i tranzycji, np. parametry: <stime>, <htime>, <activity>, <color>; składowane są w deklaracji miejsc i tranzycji.

Rys. E4 Główne znaczniki zapewniające konsolidację danych w projekcie SPMC

AD 2 i 3.

W procesie dekompozycji funkcjonalnej, specyfikacja wejściowa poddawana jest podziałowi na część programową i sprzętową. Rezultatem algorytmu dekompozycji są dwie sieci Petriego: sieć programowa, sieć sprzętowa. Na rysunku E5, w sposób graficzny przedstawiono przykładowy podział zadanej specyfikacji wejściowej E5 a). Oznaczone kolorem czerwonym prostokąty są punktami komunikacyjnymi (punktami podziału specyfikacji wejściowej) w obszarze wewnetrznej komunikacji program <-> sprzęt mikrostruktury cyfrowej. Przeprowadzona w procesie projektowym analiza przepływu danych, dostarcza informacji niezbędnych do konfiguracji systemu komunikacji/synchronizacji program<>sprzęt w zakresie wymiany przetwarzanych danych.



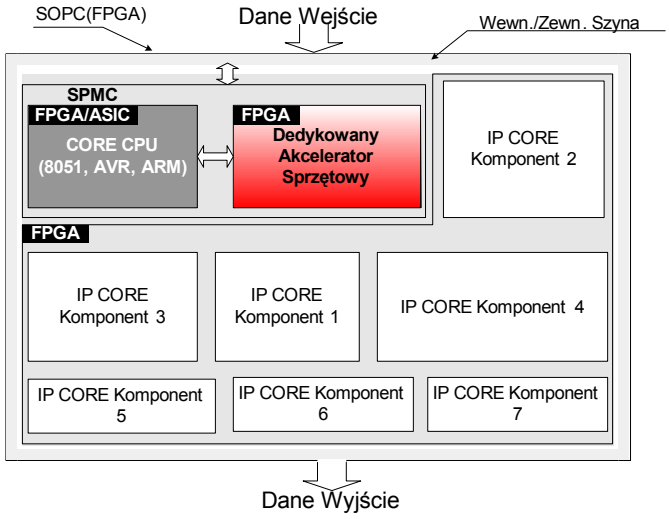
Rysunek E5. Specyfikacja wejściowa SPMC po procesie dekompozycji funkcjonalnej: a) specyfikacja wejściowa b) część programowa, c) część sprzętowa

AD 4 i 5

W dalszej części procesu projektowego SPMC, sieć programowa i sprzętowa poddawane są procesowi syntezy SPMC sieci Petriego. Na podstawie specyfikacji funkcjonalnej w postaci sieci Petriego, generowany jest kod ANSI C dla części programowej, oraz kod VHDL dla części sprzętowej.

AD 6

Wyniki procesu syntezy programowej i sprzętowej sieci Petriego gotowe są do bezpośredniej (z wyłączeniem dodatkowych operacji przetwarzania komputerowego) implementacji w sprzętowo-programowej mikrostrukturze cyfrowej, rysunek E6.



Rys. E6 Realizacja fizyczna mikrostruktury cyfrowej

Sposób fizycznego dostarczenia wyników procesu SPMC do rzeczywistej jednostki zadaniowej mikrostruktury cyfrowej, może zostać przeprowadzona na wiele sposobów, w zależności od dostępnych konstrukcji i interfejsów cyfrowych systemu: JTAG, ROM, EEPROM, PCI, inne.