ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1993

Seria: INFORMATYKA z. 22 Nr kol. 1195

Stefan SENCZYNA

PRZEGLĄD METOD REALIZACJI FORMUŁY PROCESU W WYBRANYCH JĘZYKACH OPISU SPRZĘTU CYFROWEGO

Streszczenie. Zadaniem języka opisu sprzętu jest pełnienie funkcji sprzęgu między koncepcjami formułowanymi przez projektanta a narzędziami programowymi dla wyrażenia opisu projektów oraz dalszego przetwarzania - symulacji, syntezy. We wstępnych punktach opracowania został przedstawiony ogólnie proces projektowania układów cyfrowych. Problem wyrażenia formułami językowymi zagadnienia współbieżnego funkcjonowania elementów układu cyfrowego został przedstawiony za pomocą wybranych języków opisu sprzętu na płaszczyźnie instrukcji procesu. Wybrane zostały języki: ISPS, SFL, VHDL oraz język ADA - który jest językiem programowania zawierającym instrukcję procesu.

A REVIEW OF METHODS OF REALIZATION THE PROCESS INSTRUCTION IN CHOOSEN HARDWARE DESCRIPTION LANGUAGE

Summary. The VLSI project usually request to use CAD environment for designing. Hardware description language is interface between designer's concept and CAD environment. The first sections describes process of designing in the domain of modeling and representation of project. For analizys the problem of process instruction and parallel procesing it has been chossen some exampls of HDL's, like ISPS, SFL, VHDL and programing language ADA witch include process instruction. Additionaly, the example of NAND describe in VHDL has been shown.

REVUE DES MÉTHODES D'IMPLEMENTATION DE LA FORMULE DU PROCESSUS DANS LES LANGAGES D'INSCRIPTION DU MATÉRIEL CHOISIS

Résumé. La technologie de la construction des semi-conducteurs VLSI exige l'utilisation de l'environnement de CAO. Les langages d'inscription du matériel sont implémentés comme l'interface entre le constructeur et l'ordinateur avec l'environnement de CAO. Dans les premières sections de cet article, il est présenté le processus de construction des semi-conducteurs, partagé en simulation de parallélisme de fonctionnement des modules et représentation du projet. L'analyse des problèmes attachés à simulation du parallélisme de fonctionnement des éléments numériques, est présenté par quelques exemples élaborés dans les langages d'inscription du matériel (ISPS, SFL, VHDL) et dans langage ADA incluant des instructions du processus. L'article est terminé par présentation l'inscription de la porte NAND en VHDL.

1. Ogólne zasady formułowania opisu behawioralnego projektu układu cyfrowego

1.1. Proces projektowania układów cyfrowych wspomagany komputerowo

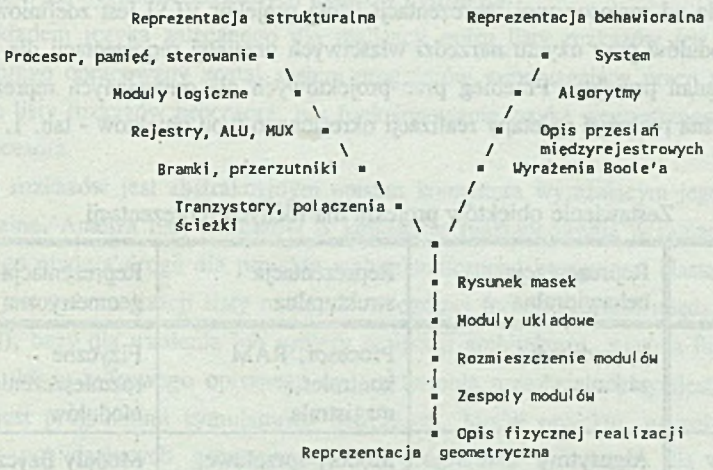
Projektowanie układów cyfrowych jest dziedziną, gdzie można zaobserwować oddziaływanie właściowości realizacji technicznej projektu na sam proces projektowania. Rozwijanie projektów systemów komputerowych daje narzędzia dla implementacji algorytmów programów wspomagania projektowania,, umożliwiając efektywne wykorzystanie technologii półprzewodnikowych o coraz większej złożoności. Ta sytuacja stawia projektantów przed konstrukcjami o coraz większym stopniu komplikacji.

W skład uogólnionej postaci procesu projektowego wchodzi informacyjne sprzężenie zwrotne między etapem formułowania elementów koncepcji struktury całości a etapem analizy efektów płynących z zastosowania określonych rozwiązań. Sprzężenie jest realizowane drogą formułowania wniosków prowadzących do weryfikacji składowych koncepcji etapu początkowego i dalszego obiegu informacji poprzez etap analizy.

Etap koncepcyjny jest opracowywany przez projektanta, systemy programowego wspomagania są włączane do pracy na etapie analizy. Zastosowanie wspomagania kompu-

terowego opiera się na wykorzystaniu formuły językowej dla realizacji opisu projektu układu cyfrowego, który pełni funkcję interfejsu między projektantem a systemem programów wspomagających.

Kompleksowy proces projektowania obejmuje całą ścieżkę dojścia od założeń do projektu technologicznego - rysunku masek układu VLSI. Programowe systemy wspomagania projektowania VLSI, których wynikiem funkcjonowania jest rysunek masek, przyjęto nazywać kompilatorami krzemowymi [15]. W dziedzinie projektowania VLSI są możliwe trzy koncepcje reprezentacji projektu: strukturalna, behawioralna, geometryczna, ich model przedstawia diagram Y Gajskiego-Kuhna [15] - rys. 1.



Rys. 1. Model Y reprezentacji projektowania VLSI
Fig. 1. Levels of design in tripartite representation

Reprezentacja behawioralna wykorzystuje model czarnej skrzynki dla opisu całości i elementów projektu, w postaci zbioru funkcji uzależnień wyjść od wejść protokołów interfejsów między modułami, ograniczeń nakładanych na projekt. Protokół interfejsu jest specyfikacją uzależnień czasowych i dopuszczalnych sekwencji sygnałów interfejsu. Ograniczenia wynikają z planowanej realizacji technologicznej projektu. Reprezentacja strukturalna, podobnie jak behawioralna wykorzystuje model czarnej skrzynki, projekt jest dzielony na moduły stanowiące zwarte jednostki od strony architektonicznej (procesor, pamięć).

Na kolejnych etapach projektu korzysta się z modułów o ustalonej architekturze, a potrzeba uzyskania zgodności funkcjonalnej między modułami wymusza zmiany w schematach modułów. W przypadku stosowania modelu behawioralnego korekcie ulegają funkcje opisujące moduł.

Wymienione reprezentacje stosują podobne koncepcje realizacji procesu projektowego, tak że możliwe było opracowanie jednego narzędzia wspierającego projektowanie strukturalne i behawioralne - języka VHDL.

Reprezentacja geometryczna ściśle odzwierciedla proces produkcyjny elementów VLSI, należy ją jednak rozpatrywać jako etap końcowy projektu, rozpoczynający się od punktu opracowania zweryfikowanego schematu logicznego projektowanego układu. Ten etap w przypadku elementów VLSI wymaga stosowania programowych środków wspomagania projektowania - nazywanych zwykle kompilatorami krzemowymi, których zadaniem jest generacja masek technologicznych dla projektu wyrażonego językiem opisu sprzętu.

Niezależnie od zastosowanej reprezentacji istotą projektu VLSI jest zdefiniowanie potrzebnych modułów przy użyciu narzędzi właściwych przejętej reprezentacji dla realizacji założonych zadań projektu. Przebieg prac projektowych dla omawianych reprezenatacji, rys. 1.1, można rozłożyć na etapy realizacji określonego typu obiektów - tab. 1.

Tabela 1

Zestawienie obiektów projektu dla różnych reprezentacji

Poziomy projektu	Reprezentacja behawioralna	Reprezentacja strukturalna	Reprezentacja geometryczna
Systemowy	Specyfikacja zadania	Procesor, RAM kontroler, magistrala	Fizyczne rozmieszczenie modułów
Algorytmów	Algorytmy operacji na danych	Moduły sprzętowe, struktury danych	Moduły fizyczne, połączenia
Mikro-architektury	Przesyły międzyrejestrowe, grafy stanów	Mikropamięć sekwenser, ALU, MUX, rej.	Rozplanowanie schematu modułów
Logiki	Wyrażenia Boole'a	Przerzutniki, bramki	Komórki, rozmieszczenie
Obwodów	Funkcje czasowe przesyłów	Tranzystory, połączenia, styki	Rysunek masek

Języki opisu sprzętu (od HDL - hardware description language) są rozwinięciem języków programowania dla realizacji specjalistycznego zagadnienia, projektowania układow cyfrowych. Znaczącym przykładem z najnowszych opracowań jest język opisu VHDL, wzorowany na języku ADA.

Informację zawartą w projekcie układu cyfrowego można uogólnić do dwóch kategorii: opisu konstrukcyjnego - architektury oraz opisu funkcjonalnego - behawioru. Nie ma konieczności wyróżniania ostrych granic między architekturą i behawiorem, oba opisy stanowią funkcjonalną całość i w szczególnych przypadkach mogą być stosowane wymiennie. Opis behawioralny jest tym bardziej elastyczny, im mniej szczegółowy jest opis architektury, przy czym dla realizacji funkcjonalnego opisu konieczny jest opis struktury o odpowiednio dobranym poziomie abstrakcji. Od strony procesu projektowego i realizacji wspomagania (kompilator języka, symulator) można przedstawić języki opisu zorientowane na specjalistyczną symulację listy rozkazów procesora oraz na symulację przebiegów czasowych sygnałów logicznych układów cyfrowych.

Przykładem języka zalecanego dla realizacji opisu listy rozkazów jest ISPS [1], na bazie którego opracowany został system programów wspomagający prace projektowe na poziomie listy rozkazów procesora, np: funkcjonowanie języka wewnętrznego projektowanego procesora.

Lista rozkazów jest abstrakcyjnym opisem komputera wyrażającym jego właściwości funkcjonalne. Analiza listy rozkazów w kontekście funkcjonowania w ramach języka wewnętrznego otwiera drogę dla projektu architektonicznego komputera. Następnym etapem projektu jest opis realizacji listy rozkazów w ogólnej formie przesłań międzyrejestrowych (RTL [7]), bazy dla ustalenia lub syntezy realizacji architektury. Analiza funkcjonowania projektu układu cyfrowego opisanego ogólną formułą przesłań międzyrejestrowych realizowana jest programami symulatorów logicznych. Model projektu, uzupełniony opisem sygnałów pobudzających przetwarzany jest programem symulatora dla zobrazowania funkcjonowawnia w formie przebiegu wybranych sygnałów logicznych w dziedzinie czasu. Jakość efektów pracy symulatora wynika z cech zastosowanego języka opisu, który powinien umożliwić wielopoziomowy opis projektu - od funkcjonalnego podziału na moduły, opisu przesłań międzyrejestrowych do poziomu bramek, przerzutników i sygnałów logicznych.

Przykładem takiego języka jest VHDL [2]-[6], posiadający bogaty zestaw konstrukcji językowych umożliwiających opis listy rozkazów, chociaż w formie bardziej skomplikowanej niż ISPS [1].

W ramach wspomagania komputerowego zweryfikowany projekt stanowi dane dla realizacji etapu syntezy, końcowy produkt może być szczegółowym projektem architektury z zastosowaniem wybranej bazy elementowej lub dalej projektem masek i testów do produkcji elementu VLSI. Etap syntezy, w zależności od zastosowanego języka, może być realizowany w ramach dwóch skrajnych koncepcji: pierwsza to automatyczna konwersja [11],[7] opisu behawioralnego w opis struktury oraz druga - wyrażenie opisu projektu architektury językiem opisu i transformacja tego opisu w realizację sprzętowa. Automaty-

czna konwersja oznacza konieczność przyjęcia określonej formuły algorytmów generacji i zastosowanie algorytmów optymalizacji generowanej struktury logicznej.

Praktycznie pełny opis koncepcji architektury umożliwia język VHDL, ale bez konieczności jej wyczerpującego definiowania w celu uzyskania poprawnej implementacji opisu projektu w tym języku.

1.2. Syntetyczne ujęcie metod projektowania

Projektowanie układów cyfrowych podporządkowane jest metodzie stosowanej w ogólnie rozumianym projektowaniu. Koncepcja układu rozkładana jest na moduły nazywane czarnymi skrzynkami [9], które następnie opisywane są na poziomie ich zewnętrznego funkcjonowania stosując ujednoliconą postać opisu dla danego etapu projektu (np. procesor, pamięć; licznik, multiplekser, rejestr). W praktycznej realizacji projektowanie przebiega od poziomu założeń ogólnych poprzez etapy opracowywania kolejnych poziomów projektu struktury, na których prowadzona jest ciągła weryfikacja założeń tak, by uwzględnić pojawiające się ograniczenia w realizacji potrzebnych funkcji.

Metoda oparta na modelu czarnej skrzynki wymaga określenia opisu struktury połączeń wejść/wyjść między składowymi modułami projektowanego układu. Połączenia te są częścią architektury układu cyfrowego [8]. Na tle takiego podejścia prosty język opisu może być zdefiniowany w postaci list połączeń i modułów funkcjonalnych.

Rozwijanie realizacji projektu może przebiegać według wariantu od globalnej specyfikacji funkcjonalnej do szczegółowago opracowania architektury lub od ustalonych elementów architektury do struktury realizującej coraz bardziej złożone funkcje oraz według wariantu pośredniego, w którym elementy składowe architektury są konfrontowane z założeniami globalnymi, weryfikowane i zestawiane w większe jednostki.

Wymienione warianty [8] można nazwać: zstępującym, wstępującym i łączącym, ważną różnicą między dwoma pierwszymi jest stosowanie behawioralnego opisu projektu i elementów architektury jako metody realizacji projektu w wariancie zstępującym, gdy wariant wstępujący ma na celu kreowanie coraz bardziej złożonej struktury poprzez zestawianie zdefiniowanych elementów architektury. Wariant łączący jest kompromisem pomiędzy wymogami stawianymi przez opis behawioralny a właściwościami zastosowanych modułów architektonicznych. Na tym tle opis behawioralny jest narzędziem projektowania zstępującego, co uwidocznia się w językach opisu [1],[2], gdzie strona funkcjonalna projektu jest nadrzędna wobec stosowanych definicji elementów architektury.

Ogólnie koncepcja opisu behawioralnego jest predystynowana do ujmowania zjawisk zachodzących w układach cyfrowych według formuły: "co i według jakiego algorytmu należy wykonać" niż "jakimi środkami dysponujemy i co można zrealizować".

Wymienioną klasyfikację wariantów projektowania można odnieść do realizacji projektu w płaszczyźnie styku układ cyfrowy - język opisu i przedstawić następujące ramowe podejścia: język opisu jest narzędziem do wykonania opisu już istniejącego projektu układu cyfrowego, założenia dotyczące funkcjonowania układu są rozwijane w realizację architektury stosując formuły językowe; lub projekt jest konsekwentnie opracowywany wyrażeniami opisu językowego w celu odzwierciedlenia wymaganych założeń i uzyskania schematu konkretyzującego potrzebne elementy architektury jako następnego etapu prac.

Przedstawiona interpretacja metod zstępującej i wstępującej wynika z uniwersalnego charakteru tych pojęć, w odniesieniu do zagadnienia realizacji opisu językowego istniejącego projektu praktycznie jest powtarzany tok całego procesu projektowania, np. transformacji schematu ideowego w formuły wybranego języka opisu sprzętu.

W ujęciu kompleksowym system programów wspomagania, skojarzony z językiem opisu, umożliwia realizację opisu projektu układu cyfrowego od sformułowania ramowej koncepcji poprzez konkretyzację składników behawioralnych aż do uzyskania pełnego opisu struktury architektonicznej. Realizacja projektu, począwszy już od etapu formułowania koncepcji ogólnej, może być prowadzona językiem opisu z wykorzystaniem zaleceń dotyczących kodowania w językach programowania. Ogólnie są to: struktulizacja opisu - stosowanie zwartych konstrukcji semantycznych i podział na moduły; stosowanie mechanizmów organizacji współbieżnego przetwarzania dla opisu funkcjonalnych aspektów układu cyfrowego; rozwinięte deklaracje typów danych.

Realizacja procesu projektowego z zastosowaniem języka behawioralnego, jako narzędzia opisu ustaliła tok postępowania projektowego według właściwości programów wspomagania projektowania, głównego nurtu ich specjalizacji, np: projektowanie elementów VLSI. Funkcje projektowanego układu są konsekwentnie określane konstrukcjami językowymi, podporządkowując opis elementów architektury opisowi funkcjonalnemu. Od strony systemu wspomagania program symulatora dokonuje analizy opisu, umożliwiając ciągłą weryfikację rozwijanego projektu, aż do osiągnięcia założonych wymagań i przygotowania danych dla etapu syntezy.

1.3. Aspekt behawioralny i strtukturalny opisu projektu układu cyfrowego

Schemat ideowy projektu układu cyfrowego jest zasadniczo podstawowym narzędziem pracy konstruktora. Szerokie zastosowanie języków behawioralnych i systemów wspomagania wymusza stosowanie nowych technik formułowania koncepcji i opisu projektu. Jednakże w tych systemach, po przekroczeniu odpowiedniej mocy obliczeniowej komputerów, pojawiła się metoda wspomagania prac nad formułowaniem opisu projektu z zastoso-

waniem równoległego opisu językowo-graficznego [12] z możliwością konwersji w obu kierunkach. Daje to wzbogacenie narzędzi wspomagania projektowania, ale nie wpływa w istotny sposób na dalsze przetwarzanie.

Praca nad schematem ideowym może być ustalona na różnych poziomach abstrakcji, poziomem jednoznacznie określającym architekturę jest schemat struktury połączeń między elementami podstawowymi, opisanymi funkcjami przejścia, np. bramki, przerzutniki. Uogólnianie koncepcji prowadzi do stosowania na coraz wyższych poziomach abstrakcji symboli modułów - czarnych skrzynek, elmentów struktury, jednostek opisu behawioralnego. Pewnym uporządkowaniem zagadnienia opisu konstrukcji układu cyfrowego jest wprowadzenie definicji [9] rozgraniczających aspekt behawioralny i strukturalny opisu.

W tym celu można przedstawić wiele następujących określeń: element prymitywny jest to obiekt urządzeniowy, który został opisany listą relacji wejście - wyjście (we/wy). Opis relacji we/wy wyznacza zestaw wejść modułu oraz rodzajów wyjść modułu, ten opis jest również nazywany modelem behawioralnym. Elementem złożonym jest obiekt urządzeniowy opisany w formie listy połączeń między elementami składowymi, którymi mogą być elementy prymitywne i złożone. Opis w formie listy połączeń jest nazywany modelem strukturalnym. Połączenie jest kanałem komunikacyjnym przenoszącym informację między dwoma elementami. Opis hierarchiczny jest wielopoziomowym opisem projektu urządzenia cyfrowego, który zawiera elementy złożone. Pierwotny opis projektu zawiera tylko elementy podstawowe.

Pełny opis projektu urządzenia cyfrowego składa się z: listy relacji we/wy i definicji elementów podstawowych, albo jest opisem hierarchicznym, który może być transformowany do postaci podstawowej odpowiednim zbiorem formuł.

Opis projektu jest poprawny tylko, gdy spełnia warunek, że jest to opis pełny i dla każdej pary elementów podstawowych c1, c2 opis podstawowy zawiera dopuszczalne połączenia między wejściami i wyjściami c1 i c2.

Poziom opisu projektu określa, jaka część opisu jest podana w formie modelu behawioralnego, a jaka w formie strukturalnego. Wysoki poziom realizacji opisu projektu charakteryzuje się prostą strukturą architektoniczną i złożonym opisem behawioralnym. Zwiększenie szczegółowości opisu architektonicznego jest osiągane na niższych poziomach opisu projektu. Projekt urządzenia cyfrowego może być realizowany z zastosowaniem mieszanych poziomów opisu.

2. Realizacja mechanizmu opisu procesu w wybranych behawioralnych językach opisu sprzętu

2.1. Strukturalna współbieżność w funkcjonowaniu układów cyfrowych

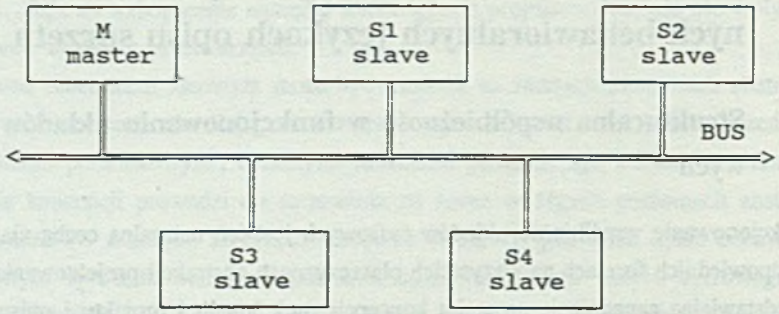
Funkcjonowanie współbieżne układów cyfrowych jest ich naturalną cechą ujawniającą się w odpowiednich formach na wszystkich płaszczyznach abstrakcji projektowania.

Przedstawiając zagadnienie na styku koncepcji funkcjonalnej projektu i opisu językowego w aspekcie stosowania odpowiednich formuł językowych dla wyrażenia strukturalnej współbieżności i ilustracji można wybrać etap zestawiania projektu z modułów roboczych i ustalania zależności funkcjonalnych między nimi. Od strony konstrukcyjnej w odniesieniu do układów komputerowych problem współbieżności można przedstawić przykładem modelu organizacji współpracy bloków funkcjonalnych magistrali komputera w konwencji master-slave.

Analizowany przykład modelu master-slave opisuje funkcjonowanie struktury architektonicznej według rys. 1 do realizacji zadania przesyłu danych między modułami układu cyfrowego. Ogólny opis behawioralny modelu można przedstawić za pomocą struktury składającej się z dwóch typów modułów oraz magistrali. Magistrala od strony architektonicznej jest zbiorem sygnałów logicznych lub linii przesyłowych, które można podzielić na dwie grupy: linie danych i linie sterujące. Oba typy modułów, nazwane master i slave, posiadają porty wejście - wyjście łączące moduły z magistralą, a w skład struktury wewnętrznej wchodzi automat sekwencyjny oraz bufory danych. Linie sterujące oraz linie danych magistrali są podłączone poprzez port wejście-wyjście modułu odpowiednio z automatem sekwencyjnym i buforem danych.

Do wykonania opisu behawioralnego modelu master-slave nie jest konieczne precyzowanie operacji na danych i związanej z tym odpowiedniej struktury architektonicznej modułów. Ustalenie opisu modelu w aspekcie sterowania modułami dla przesyłu danych jest poziomem wyjściowym dla konkretyzowania dalszych szczegółów funkcji przetwarzania danych i wewnętrznej struktury architektonicznej modułów.

Opis behawioralny przesyłu danych obejmuje jednostkę nazwaną cyklem magistralowym. Otwarcie cyklu magistralowego wymusza moduł master, odpowiednia sekwencja sygnałów sterujących wybiera moduł slave, który podejmnie współpracę w realizacji cyklu. Następnie realizowana jest operacja wymiany danych między zaktywizowanymi modułami, a zakończenie cyklu może być wsparte sygnałem potwierdzenia wystawionym przez moduł slave.



Rys. 2. Układ master-slave
Fig. 2. The master-slave architecture

W tej konstrukcji tylko moduł master rozpoczyna i kontroluje przebieg cyklu magistralowego, moduł slave podporządkowuje się wymuszeniom podawanym przez magistralę. Współbieżne funkcjonowanie wynika z aktywizacji co najmniej dwóch modułów, które jednocześnie wykonują swoje wewnętrzne operacje według ustalonej sekwencji zdarzeń na magistrali.

Przedstawiona konstrukcja może być dalej rozwijana do realizacji określonego zadania np. cyklu magistrali asynchronicznej, lub po uproszczeniu cyklu magistrali synchronicznej, układów interfejsów, przerzutników. Zwiększenie precyzji opisu wymaga wprowadzenia odpowiednich elementów architektonicznych np. ilości linii sterujących i danych, rejestrów, buforów, dekoderów. Od strony języka opisu wyrażenie funkcjonowania niezależnych modułów architektonicznych jest wsparte konstrukcją semantyczną z zastosowaniem instrukcji procesu. W dalszych przykładach zostaną przedstawione konstrukcja językowa - instrukcja procesu lub deklaracja etapu, zadania, do realizacji opisu współbieżnego funkcjonowania układów cyfrowych. Na bazie języka programowania ADA [10] został przedstawiony sposób zastosowania instrukcji zadania dla realizacji prostego kanału komunikacyjnego CONSUMER - PRODUCER, co jest pewnym analogiem konstrukcji master - slave.

2.2. Wybrane przykłady języków opisu

Sposoby implementacji instrukcji procesu zostaną przedstawione w wybranych przykładach różniących się od siebie języków opisu sprzętu (HDL [8][2]) z włączeniem omówienia deklaracji zadania w języku programowania ADA, który jest podstawą koncepcyjną dla języka VHDL. Pod uwagę zostały wzięte: ISPS[1], SFL[7], VHDL[2]-[6], [12].

W przypadku języka ISPS jego dziedziną zastosowań jest wspomaganie szerokiego zakresu aplikacji na jednym ustalonym poziomie, w szczególności funkcjonowania listy rozkazów, wspomagania projektowania na różnych poziomach, np. sieci logicznych. Język SFL w założeniach zaprojektowany został do realizacji opisu behawioralnego równoległego funkcjonowania układów na poziomie przesłań międzyrejestrowych, przy czym jego głównym zastosowaniem mają być projekty architektur wieloprocesorowych, wektorowych procesorów.

Następna pozycja uwzględniona w porównaniu, język VHDL, jest protegowana do realizacji opisu na swobodnie dobranym poziomie abstrakcji projektu: od listy rozkazów procesora, struktury w formie przesłań międzyrejestrowych, sieci logiczne złożone z bramek i przerzutników do opisu parametrów technologicznych rozmiarów geometrycznych masek. Koncepcja tego języka jest w sensie definicji struktur odwzorowaniem języka ADA i stanowi realizację jego rozwoju w kierunku formuły obiektowo zorientowanej. Konstrukcje językowe VHDL-a umożliwiają realizację szczegółowego opisu struktury architektonicznej, jednocześnie opis behawioralny może być wyrażany w formie odrębnych modułów semantycznych. Dla symulatora opis behawioralny i jego podstawowy reprezentant - instrukcja procesu jest strukturą "nośną" sterując przebiegiem symulacji.

2.3. Deklaracja zadania w języku ADA

Język programowania ADA [10] opracowany został według koncepcji programowania współbieżnego. Do zestawu struktur programowych wprowadzona została deklaracja zadania oraz związane z nią typy danych i reguły konstruowania treści programu.

Program współbieżny układany jest metodą organizacji ciągu instrukcji w zadania, przyjmując regułę niezdeterminowanej kolejności realizacji zadań. Od strony komputera sekwencyjnego i odpowiednio programu sekwencyjnego przebieg realizacji zadań jest również sekwencyjny, a ich kolejność przetwarzania jest ustalana wewnętrznymi mechanizmami kompilatora z zastosowaniem jawnych mechanizmów synchronizacji. Stąd pojęcie współbieżności bezpośrednio dotyczy logicznej struktury programu i jest narzędziem umożliwiającym programiście wyrażanie czasowej równoległości czynności zwartymi

instrukcjami programu. Problem tym samym zostaje rozdzielony na dwie płaszczyzny: struktur programowych, rozwinięcia w instrukcje języka algorytmu oraz implementacji komputerowej języka. Ma to na celu implementację języka na komputerach wieloprocesorowych bez konieczności naruszania struktur programowych i danych dostępnych programiście.

Wyodrębnienie definicji deklaracji zadania ma na celu ujawnienie mechanizmów współbieżnego funkcjonowania w czasie elementów algorytmu progamu, który to mechanizm stosowany jest w językach opisu sprzętu do projektowania układów cyfrowych.

Deklaracje zadań języka ADA [10] zawarte są w części deklaracyjnej obejmującej jej jednostki programowej - nazywanej jednostką macierzystą. Wykonanie jednostki macierzystej określa początek i koniec wykonywania zadania. Dla wszystkich zadeklarowanych zadań przyjmuje się ich wzajemną równoległą pracę z treścią jednostki macierzystej. Czynności zadań są wykonywane w kolejności instrukcji zadania z uwzględnieniem stosowanych instrukcji synchronizacji. Zakończenie pracy jednostki macierzystej następuje z zakończeniem wykonywania instrukcji wszystkich zadań i instrukcji własnych. Zadania jednostki macierzystej nazwane zadaniami siostrzanymi dzielą się na bierne - zapewniające innym zadaniom pewne usługi i czynne, które mogą korzystać z tych usług. Wzajemna synchronizacja i komunikacja między zadaniami jest realizowana mechanizmem spotkania, realizującego udostępnianie usług zadaniom czynnym przez zadania bierne. Forma deklaracji zadania ilustrująca koncepcję współdziałania zadań może być przedstawiona przykładem treści zadań czynnych PRODUCER (1) i CONSUMER (2) połączonych buforem oraz biernego zadania SIMPLE_BUFFER (3) [10].

```
(1)  task PRODUCER;
     task body PRODUCER is
       C1 : CHARACTER;
     begin
       -- petla
       -- wytworzenie znaku C1
       SIMPLE_BUFFER.WRITE(C1);
       -- koniec petli
     end PRODUCER;
```

```
(2)  task CONSUMER;
     task body CONSUMER is
       C2 : CHARACTER;
     begin
       -- petla
       SIMPLE_BUFFER.READ(C2);
       -- użycie znaku C2
       -- koniec petli
     end CONSUMER;
```

```
(3)  task SIMPLE_BUFFER is
       entry WRITE(CH:in CHARACTER);
       entry  READ(CH:in CHARACTER);
     end SIMPLE_BUFFER;

     task body SIMPLE_BUFFER is
       CHAR:CHARACTER;
     begin
       loop
         accept WRITE(CH:in CHARACTER) do
```

```
        CHAR := CH;
      end WRITE;
      accept READ(CH:in CHARACTER) do
        CH := CHAR;
      end READ;
      exit when CHAR = ASCII.EOT;
    end loop;
end SIMPLE_BUFFER;
```

W przykładzie zadanie bierne (3) służy jako prosty kanał komunikacyjny, do którego inne zadania wpisują znaki lub odczytują je z niego.

Nazwy usług w treści zadania specyfikowane są słowem kluczowym entry, a słowem accept ustalany jest punkt spotkania. Metoda komunikowania między zadaniami, nazwana spotkaniem, realizuje synchronizację, gdy działanie dwóch zadań zbiega się w określonym odcinku czasu, po którym wykonują swoją pracę dalej niezależnie. Zadanie, które pierwsze osiągnie punkt spotkania, czeka na partnera, gdy oba - wywołujące i przyjmujące osiągnęły instrukcje realizujące spotkanie, następuje wykonanie działań określonych instrukcją accept. Po zakończeniu spotkania podejmowana jest dalsza sekwencyjna egzekucja instrukcji w obu zadaniach.

Jeżeli kilka zadań wywoła to samo wejście, to zadanie przyjmujące wywołanie spotka się z tym, które nadeszło pierwsze. Zadanie przyjmujące zawiera zazwyczaj pętlę, w której jest powtarzana instrukcja accept i obsługa wejść wykonywana jest w kolejności ich nadchodzenia.

W przedstawionym przykładzie PRODUCER - CONSUMER ujawnia się pewna właściwość semantyczna języka ADA [10], reguła budowania usługi wywoływanej przez zadania. Nazwa ta jest składana z nazwy zadania i nazwy usługi.

Odnośnie do treści algorytmu przedstawionych zadań (1), (2) należy zaznaczyć, że do poprawnej realizacji usług muszą one być wywoływane na przemian, gdyż bufor zadania SIMPLE_BUFFER (3) jest jednoznakowy. Obok synchronizacji przez spotkanie w języku ADA stosowane są również zmienne współdzielone.

Przedstawiona instrukcja programu - zadanie umożliwia bezpośrednie wyrażanie opracowywanych wielowątkowych algorytmów specjalizowanymi instrukcjami i znalazła również swój odpowiednik w językach opisu sprzętu (np. proces w języku VHDL).

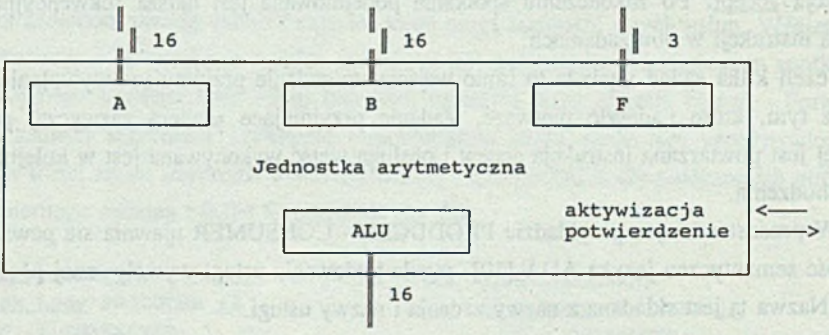
2.4. Instrukcja procesu języka ISPS

Głównym przeznaczeniem języka ISPS [1], według jego założeń koncepcyjnych, jest realizacja opisu projektów układów komputerowych ze wskazaniem na konstrukcje procesorów.

Struktury języka specjalizowane są dla prostego i elastycznego opisu funkcjonowania komputera w płaszczyźnie listy rozkazów. System programów wspomagania, zbudowany na bazie języka ISPS, realizuje szeroki zakres zadań projektowania architektur komputerów: kompilator języka opisu, symulator funkcjonalny, analizator właściwości języka wewnętrznego projektowanej jednostki komputerowej, w tym realizacja kompilatora dla opracowanego języka wewnętrznego.

Wspomaganie realizacji odmiennych aspektów projektu wymaga stosowania zróżnicowanych struktur danych np. generator asemblera wymaga specyfikacji mnemoniki instrukcji, dla etapu syntezy technologicznej realizacji stosowany jest opis właściwości fizycznych elementów konstrukcyjnych. Dla całości systemu programów wspólna jest baza danych Global Data Base [1] posiadająca pośredni format, do postaci którego kompilowany jest opis projektu wyrażony językiem ISPS.

Instrukcja procesu stosowana w języku ISPS została zilustrowana przykładem deklaracji (2) [1] jednostki arytmetyczno - logicznej ALU (1) [1].



Rys. 3 Architektura jednostki ALU
Fig. 3 The architecture of ALU unit

```
(2) proces ALU ( A<0:15>, B<0:15>, F\Function<3:0> )<0:15>
        := Begin
          Decode F =>
                  Begin
                  0\Add  := Alu = A + B,
                  1\Sub  := Alu = A - B,
```

```
                2\Not.A := Alu = Not A,
                3\Not.B := Alu = Not B,
                4\Or    := Alu = A Or B,
                ....
                14\Zero := Alu = 0,
                15\Ones := Alu = "FFFF
                End
        End
```

Ta deklaracja może być zastosowana w organizacji współbieżnego funkcjonowania modułów architektonicznych projektu procesora.

Zastosowanie (3) jednostki ALU w programie może być następujące:

```
(3)  ... Next ALU(Y,Z,5) Next
                ... Next X = ALU; Zcc Egl 0 .
```

Należy zauważyć, że użycie wyjścia ALU nie zawsze musi wystąpić jednoznacznie po zakończeniu realizacji procesu ALU, przetwarzanie którego jest niezależne i czas jego realizacji może przekroczyć punkt użycia nośnika ALU w innych współbieżnych instrukcjach. W takich przypadkach powinien być zastosowany jawny mechanizm synchronizacji między wywołaniem procesu a wykorzystaniem wyników jego funkcjonowania.

Prostą metodą jest użycie zmiennej synchronizującej. Dla synchronizacji przetwarzania jednostki ALU jej zastosowanie (4)(5) jest następujące:

(4) wstawienie zmiennej synchronizującej do deklaracji procesu ALU (2):

```
process ALU ( ... )
        := Begin
          ...
          ALU.done = 1
          End.
```

(5) zastosowanie synchronizacji:

```
ALU.done = 0 Next
 ... ALU(Y,Z,5) Next
 ... Next
 Wait(ALU.done) Next
 X = ALU; Zcc = ALU Egl 0 Next ...
```

Zastosowanie zmiennej synchronizującej nie jest optymalne, gdyż w przypadku konwersji opisu behawioralnego w implementację układową jej obecność może spowodować zniekształcenie koncepcji projektu. Uniknięcie dodatkowych komplikacji od zmiennej synchronizującej umożliwia zastosowanie predeklarowanej procedury IS.RUNING (6):

```
(6) ALU(Y,Z,5) Next
        ... Next
    Wait (Not IS.RUNING(ALU)) Next ...
```

Innym problemem dotyczącym synchronizacji jest niedopuszczenie, tam gdzie to mogłoby spowodować błędy, do nałożenia się wywołań jednostek. Wprowadzenie arbitrażu wywołań, czyli przyjęcie ponownego wywołania jednostki po zakończeniu jej bieżącego funkcjonowania, umożliwia konstrukcja (7) za słowem kluczowym "critical".

```
(7) Critical ALU(A,B,F)
        := Begin ... End.
```

Przedstawione przykłady dotyczą realizacji koncepcji dla wyrażania behawioralnego opisu funkcjonowania układów cyfrowych, ISPS przyjmuje domyślnie realizację współbieżną wszystkich instrukcji, o ile nie są rozdzielone słowem kluczowym Next wyznaczającym jawną sekwencję ich realizacji.

2.5. Realizacja etapu w języku SFL

W dziedzinie zintegrowanych systemów programowych do projektowania urządzeń cyfrowych interesującym przykładem jest zestaw programów przetwarzania opisu projektu w behawioralnym języku SFL [7].

Funkcje realizowane przez programy systemu SFL ogólnie są następujące:

- opis projektu, język SFL,
- symulator funkcjonalny SFLSIM,
- syntezator realizacji urządzeniowej SFLEXP.

Koncepcja języka SFL jest rozwinięciem formuły RTL (register transfer level) do projektowania złożonych architektur procesorów o równoległym przetwarzaniu. Do wyrażenia opisu funkcjonalnego urządzeń cyfrowych realizujących równoległe czy potokowe przetwarzanie język SFL, w zamierzeniach jego autorów, ma dostarczać struktury językowe w prosty sposób opisujące ich projeky.

Propozycja struktury języka SFL jest analogiczna do języków programowania, np. język C, z odpowiednim ukierunkowaniem obiektowym. Konstrukcje programowe języka pozwalają na wyrażanie opisów: definicja niezależnych jednostek sterujących, definicja cyklu maszynowego, opis komunikacji między zdefiniowanymi jednostkami sterującymi. Te funkcje w zamierzeniach mają być podstawą do wyrażania opisu procesorów równoległego przetwarzania.

W skład struktury języka behawioralnego wchodzą definicje jednostek urządzeniowych, np. rejestrów, pamięci, portów, magistrali, układy kombinacyjne. W języku SFL wprowadzono jawną definicję układu kombinacyjnego, aby umożliwić bezpośrednią optymalizację w zakresie urządzeniowym.

Całość projektu dzielona jest na aktywne i pasywne obiekty. Elementy urządzeniowe, takie jak port wejście - wyjście, rejestry, arytmometr są obiektami pasywnymi, operacje na obiektach pasywnych są rozumiane jako obiekty aktywne. Genaralnie urządzenie cyfrowe składa się z wielu modułów opisanych jako jednosteki architektoniczne lub jednostek funkcjonalnych związanych z realizacją określonych operacji, np. cyklu instrukcji. Treść zwartego opisu funkcjonalnego, stanowiącego obiekt aktywny, jednostkę sterującą lub jednostkę architektoniczną jest wyrażana konstrukcją semantyczną języka nazwaną etapem (1) [7].

Wymagania funkcjonalne stawiane wobec projektowanego urządzenia cyfrowego są wyrażane w języku SFL za pomocą zadań. Przeznaczeniem konstrukcji zadania jest organizacja przetwarzania opisu funkcjonalnego zawartego w etapach, opis zadania jest częścią treści etapu. Mechanizmy języka umożliwiają przełączanie sterowania między zadaniami opisanymi w różnych etapach, organizując w potrzebnych przypadkach aktywizację więcej niż jednego zadania dla np. modelowania przetwarzania potokowego, wieloprocesorowego.

(1) format konstrukcji językowej nazywanej etapem :

```
STAGE : stage name (type) [permissible task names]
(state name)
  condition part : execution part ;
  condition part : execution part ;
        :
(state name)
  condition part : execution part ;
  condition part : execution part ;
        :
```

pole "stage name" jest nazwą własną etapu,
pole "type" jest informacją dla kompilatora,

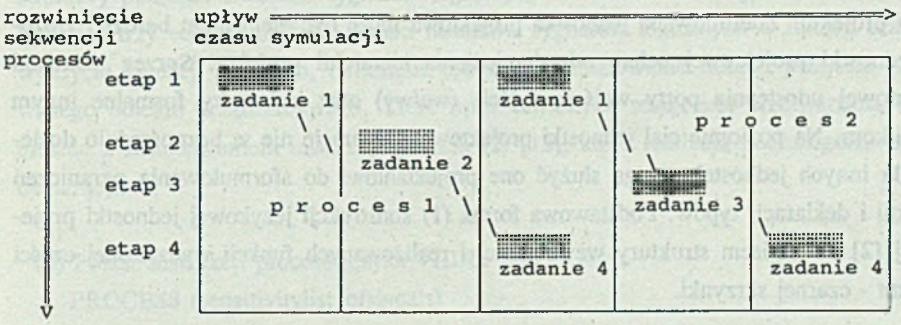
pole "permissible task names" jest listą zadań
zadeklarowanych w dalszej treści etapu,
pole "condition part" jest wyrażeniem logicznym zawierającym
następujące składniki :
- nazwa rejestru,
- nazwa obwodu kombinacyjnego,
- [nazwa zadania],
- nazwa sygnału logicznego,
pole "execution part" może zawierać następujące wyrażenia :
a) nazwa rejestru <== nazwa obwodu kombinacyjnego
(...(...(nazwa rejestru)...)...);
b) nazwa linii sygnałowej;
c) => (state name);
d) => [nazwa zadania];
e) => nazwa etapu [nazwa zadania];
f) =>=> nazwa etapu [nazwa zadania];
g) [end].
gdzie :
a) jest operacją przesyłu danych między rejestrami,
b) aktywizacja linii sygnałowej, gdy wyrażenie
"condition part" jest prawdziwe,
c) przekazanie sterowania do innego stanu (state name),
d) , e) aktywizacja zadania wewnątrz lub na zewnątrz
etapu,
f) aktywizacja zadania współbieżnego,
g) zakończenie zadania.

W skład systemu wspomagania projektowania z zastosowaniem języka SFL wchodzą programy SFLSIM (symulator), SFLEXP (syntezator realizacji sprzętowej). Podstawowe funkcje wspomagania prac konstrukcyjnych realizuje symulator pracujący w trybie interakcyjnym. Organizacja pracy symulatora jest ustalana przez projektanta za pomocą instrukcji procesu (2) [7].

(2) Gramatyka konstrukcji językowej procesu (w uproszczeniu)
w notacji BNF:
<sfl> ::= <common-info> {<module>}
CEND

```
<module>      ::= <module-info> {<def-fac>} {<def-beh>}
                  END;
<module-info> ::= NAME:<name>;
                  PURPOSE:<name> {,<name>};
                  LEVEL:SFL;
                  PROCESS:<name> {,<name>};
                  VERSION:<name>.<name>;
                  COMMENT:<string>;
```

Z poziomu symulatora proces jest jawnie określoną sekwencją zadań, których opis jest zawarty w treści etapów. W skład różnych procesów mogą wchodzić te same zadania, które są rozróżniane przez symulator odpowiednim mechanizmem pamiętania historii przebiegu procesów. Organizację przetwarzania procesów przez symulator SFLSIM schematycznie przedstawia rysunek 3.



Rys. 4. Współbieżne przetwarzanie procesów
Fig. 4. Task activation flow and process

2.6. Instrukcja procesu w języku VHDL

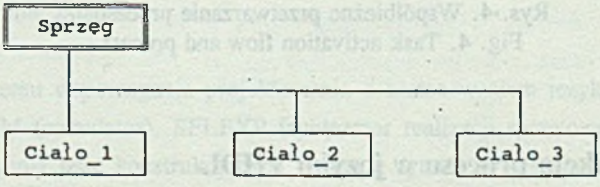
Przeznaczeniem języka VHDL [2]-[6] [12], w rozumieniu jego projektantów, jest pełnienie roli języka bazowego dla komputerowych systemów projektowania układów VLSI.

Zadaniem tego języka jest kompleksowe objęcie problemów projektowania, od poziomu systemowego do realizacji bramkowej i technologicznej. Te zadania [2] można przedstawić następująco:

- wspieranie metody systematycznego projektowania, realizując wielopoziomowy opis projektu w ujęciu hierarchicznym,

- umożliwienie gromadzenia dokumentacji dobrze opracowanych projektów w formie biblioteki,
- skuteczne stosowanie w projektowaniu osiągnięć technologicznych,
- wspieranie organizacji pracy zespołowej nad projektem,
- realizowanie wersji projektów w nowych technologiach na podstawie rozwiązań projektowych przechowywanych w bibliotece.

Organizacja systemów CAD z zastosowaniem języka VHDL oparta jest na koncepcji otwartego środowiska, które składa się z sześciu elementów: analizatora (kompilatora), analizatora odwrotnego (rekompilatora), programu zarządzającego biblioteką projektów, programu upraszczającego, symulatora i biblioteki projektów. Narzędzia w podstawowym środowisku języka VHDL oraz ewentualne nowe narzędzia dołączone do systemu komunikują się ze sobą przez dane zawarte w bibliotece projektów. Mechanizmem formalnym, służącym w języku VHDL do modelowania struktury układów cyfrowych, jest jednostka projektowa. W skład jednostki projektowej (rys. 4) wchodzi sprzęg (interfejs) oraz jedno lub kilka alternatywnych ciał (body), całość składa się na formułę opisu określonego fragmentu projektu. Zdefiniowana jednostka projektowa może być elementem bardziej złożonej jednostki projektowej celem odzwierciedlenia hierarchii projektu. Sprzęg jednostki projektowej udostępnia porty wejścia-wyjścia (we/wy) oraz parametry formalne innym jednostkom. Na poziomie ciał jednostki projektowej deklaracje nie są bezpośrednio dostępne dla innych jednostek, mogą służyć one projektantowi do sformułowania ograniczeń projektu i deklaracji typów. Podstawowa forma (1) konstrukcji językowej jednostki projektowej [2] jest opisem struktury wewnętrznej i realizowanych funkcji wydzielonej części projektu - czarnej skrzynki.



Rys. 5. Struktura jednostki projektowej w języku VHDL
Fig. 5. The structure of body project in VHDL

(1) Podstawowa forma semantyczna jednostki projektowej

```
-- package use statements
ENTITY devicename IS
```

```
    -- generic parameters
    -- port declarations
  END devicename;
  ARCHITECTURE full OF devicename IS
    -- internal input delay signals
  BEGIN
    -- error checking processes
    -- input delay processes
    -- device operation process
  END full;
```

Definicja sprzęgu, deklaracja ENTITY [12], zawiera dwie części: listę parametrów formalnych (generic) służącą do przekazywania wartości o charakterze technologicznym np. dane o opóźnieniach czasowych oraz listę portów we/wy, stanowiących elementy struktury połączeń i nośniki sygnałów logicznych. W skład deklaracji ARCHITEKTURE wchodzą trzy sekcje: analiza wartości nośników sygnałów logicznych w funkcji czasu dla wykrycia sytuacji błędnych, odliczanie opóźnień, modelowanie funkcji przejścia realizowanego obiektu urządzeniowego. Treść przedstawionych sekcji jest przetwarzana współbieżnie z zastosowaniem instrukcji procesu (2) [12], które realizują poszczególne zadania deklaracji.

(2) Postać instrukcji procesu języka VHDL.

```
  PROCESS (sensitivitylist_ofsignals)
    -- local variables
    -- local signals
    BEGIN
      -- a bunch of statements to execute when
      -- signal in sensitivity list changes value
    END PROCESS;
```

Nagłówek instrukcji procesu otwiera lista sygnałów czułości, zmiana stanu dowolnego z nich aktywizuje przetwarzanie instrukcji zawartych między nawiasami BEGIN .... END. Do operacji kontroli błędów w treści instrukcji procesu umieszczana jest instrukcja ASSERT, która wykonuje raport błędów, gdy związane z nią wyrażenie logiczne jest fałszywe.

Proste operacje przypisywania wartości i określenia opóźnienia nośników sygnałów logicznych są opisywane z zastosowaniem współbieżnej instrukcji (3). Ze względu na swoje funkcje ta instrukcja jest przetwarzana analogicznie do instrukcji procesu.

(3) outputsignal < = expression AFTER timedelayexpression

Opis behawioralny elementów projektu zawierający wiele instrukcji współbieżnego przypisania może prowadzić do komplikacji w sytuacjach stosowania więcej niż jednego przypisania wartości nośnikowi sygnału logicznego (deklarowanego jako port we/wy), np. opisując operacje na magistrali. Zastosowanie instrukcji procesu w formacie (4) [12] dla rozwiązania tego zagadnienia umożliwia wyrażenie skomplikowanych funkcji urządzeń w bardziej czytelnej postaci.

```
(4) -- device operation
    PROCESS (delayinputs)
      BEGIN
      -- operation logic
      -- output assignements
      END PROCESS;
```

Argumenty z listy delayinputs przenoszą wartości nośników sygnałów między instrukcjami procesu zawartymi w całościowym opisie projektu, dobór sygnałów na liście wynika z rodzaju operacji synchronicznej lub asynchronicznej wykonywanej przez opisywany obiekt urządzeniowy. Urządzenie, które posiada wejście zegarowe "clock" i funkcjonuje synchronicznie względem tego wejścia, jest opisane deklaracją (5):

(5) PROCESS (clock).

W przypadku gdy urządzenie posiada dwa wejścia asynchroniczne "preset", "clear", stosowana jest deklaracja (6):

(6) PROCESS (clock,preset,clear).

3. Przykład opisu bramki NAND w języku VHDL

Według materiałów źródłowych [12].

```
USE std.std_logic.ALL;
USE std.STD_ttl.ALL;
ENTITY nand_indelay IS
   GENERIC (a_i01, a_i10, b_i01, b_i10, y_o01, y_o10, a_min,
         b_min : TIME);
   PORT    (a, b : IN t_wlogic; y : OUT t_wlogic);
END nand_indelay;
ARCHITEKTURE behawioral OF nand_delay IS
    SIGNAL adelay, bdelay : t_wlogic;
BEGIN
    -- spike detection
    PROCESS (a)
      VARIABLE alastev : TIME := 0 ns;
      BEGIN
      ASSERT (NOW = 0 ns) OR ((NOW - alastev) >= a_min)
       REPORT "Spike detection on a" SEVERITY warning;
      alastev := NOW;
    END PROCESS;
    PROCESS (b)
      VARIABLE blastev : TIME := 0 ns;
      BEGIN
      ASSERT (NOW = 0 ns) OR ((NOW - blastev) >= b_min)
       REPORT "Spike detection on b" SEVERITY warning;
      blastev := NOW;
    END PROCESS;
    -- input delay processing
    adelay <= a AFTER f_delay(a,a_i01,a_110);
    bdelay <= b AFTER f_delay(b,b_i01,b_i10);
    -- gate function
    y <= adelay NAND bdelay
      AFTER f_delay(adelay NAND bdelay, y_o01, y_o10);
END behavioral;
```

LITERATURA

[1] Barbacci M. R.: "Instruction set processor specifications (ISPS): the notation and its applications" IEEE. Transasactions On Computers, vol C-30, no .1. January 1981.

[2] Gizdoń H., Pawlak A., Wrona W.: "VHDL - ADA języków opisu i projektowania sprzętu" Informatyka nr 10. 1988.

[3] Gizdoń H., Pawlak A., Wrona W.: "Język opisu sprzętu VHDL - podstawowe mechanizmy (1)" Informatyka nr 11-12. 1988.

[4] Gizdoń H., Pawlak A., Wrona W.: "Język opisu sprzętu VHDL - podstawowe mechanizmy (2)" Informatyka nr 1. 1989.

[5] Gizdoń H., Pawlak A., Wrona W.: "Zastosowanie języka VHDL do opisu i weryfikacji projektów układów cyfrowych" Informatyka nr 3. 1989.

[6] Nash J.D, Saunders L.F.: "VHDL critique" IEEE Design & Test, April 1986.

[7] Nakamura Yukihiro: "An integrated logic design environment based on behavioral description" IEEE Transasactions On Computers-Aided Design, vol. CAD-6, no. 3. May 1987.

[8] Thorp T.L, Peeling N.E.: "The role of HDLs in the design process" VLSI'87, C.H.Sequin (editor), Elsevier Science Publishers B.V. (North-Holand) IFIP, 1988.

[9] Weinbaum D., Shapiro E.: "Hardware description and simulation using Concurrent Prolog" Computer Hardware Description Languages and their Applications, M.R Barbacci and C.J Koomen (eds.), Elsevier Science Publishers B.V. (North-Holand) IFIP, 1988.

[10] Pyle Ian C.: "ADA" Wydawnictwo Naukowo-Techniczne, Warszawa 1986.

[11] Camposano R., Rosenstiel W.: "Synthesizing circuits from behavioral descriptions" IEEE Transasactions On Computers-Aided Design, vol. 8, no. 2. February 1989.

[12] Gunn L.: "DAC 1990 focuses on digital, analog simulation",
Gunn L.: "A plethora of VHDL products parade at DAC",
Coelho D.R: "Follow simple rules to create VHDL models" Penton publication, Electronic Design June 14, 1990.

[13] Bell C. G., Newell A.: "Computer Structures: Reading and examples".

[14] Derwey A., Gadient A.: "VHDL Motivation" IEEE design and Test of Computers, April 1988.

[15] "Silicon Compilation".

Recenzent: Prof. dr inż. Romuald Marczyński

Wpłynęło do Redakcji 17 września 1991 r.

Abstract

The main tool in CAD system for VLSI design is hardware description language (HDL) and is crucial that a languages has the semantics to contain the appropiate information from which the design transformation process start.

Black-box modeling of compnents is used when the interactions between a collection of components have to be checked by simulation. The process of preparing the model of VLSI circuit can be described as bottom-up, top-down or meet-in-the-middle which is a combination of the previous two.

Levels of design in the tripartite representation was shown on fig. 1. This levels are structural, behavioral and physical. The descriptions (specifications) of behaviour and structure are made with HDL's. In order to concern on particurial aspect of modeling VLSI circuit with HDL's the process instruction has been analyzed. The task of process instruction in black-box modeling is to give the ability to compouse the box model using behaviour or structural form.

From HDL's was arbitrary choosen examples: SFL, VHDL, ISPS and ADA like programing language being fundamental for VHDL.

The SFL and ISPS are introduced as behavioral style of designing. The VHDL can be considered as tool for structural designing and suported behavioral style as well.