ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1991

Seria: INFORMATYKA z. 15 Nr kol. 1044

Włodzimierz WRONA

Instytut Transportu
Politechniki Śląskiej

METODA OPISU I SYSTEM MODELOWANIA UKŁADÓW VLSI W JĘZYKACH OBIEKTOWO ZORIENTOWANYCH

Streszczenie. Specyfikacja układu VLSI jest integralną częścią procesu projektowania układów cyfrowych. Narzuca zgodną wewnętrzną reprezentację danych, wykorzystywaną przez inne narzędzia środowiska projektowego. W procesie specyfikacji układów cyfrowych można wykorzystać język programowania wysokiego poziomu. Szczególnie adekwatne do tego celu wydają się być języki obiektowo zorientowane. Ich wykorzystanie w procesie programowania wymaga zastosowania nowej metodologii projektowania systemów, różnej od metod stosowanych w tradycyjnych językach programowania, takich jak Pascal, czy C.

1. Wstęp

Szybki rozwój technologii produkcji układów scalonych, wyrażający się możliwością umieszczenia na płytce krzemowej ponad miliona tranzystorów, zmusza projektanta do znalezienia nowych metod projektowania wspomaganego komputerem (CAD). Celem wykorzystania programów CAD jest [1,7,37,54] :

- skrócenie czasu projektowania i wytwarzania układów VLSI, jak również systemów budowanych z tych układów,
- zmniejszenie kosztu wyrobu,
- zminimalizowanie błędów w projekcie,
- ułatwienie zmian projektu,
- skrócenie czasu weryfikacji i testowania,
- pełna dokumentacja projektu oraz możliwość wyświetlenia schematu projektu na ekranie monitora.

Kontynuacja tradycyjnych metod CAD wymagałaby nakładów rzędu setek osobo-lat dla opracowania i wdrożenia nowych złożonych elementów cyfrowych. Program systemów komputerowych V-tej generacji stawia przed narzędziami projektowania układów VLSI wymóg, by projektant-użytkownik był w stanie zaprojektować w ciągu miesiąca układ masek dla kostki VLSI z milionem tranzystorów [7,16,49] . Element scalony powinien być z kolei dostępny w przeciągu trzech miesięcy. Pokonanie bariery złożoności układów VLSI wymaga radykalnych zmian w metodach opisu, projektowania i wytwarzania układów scalonych [7,16,36,37] .

Jednym z rozwiązań może być stworzenie zintegrowanego środowiska automatycznego projektowania i weryfikacji układów cyfrowych w postaci kompilatora krzemowego. Kompilacja krzemowa jest procesem automatycznego projektowania układów VLSI, obejmującym wszystkie etapy projektowania (od specyfikacji projektu do masek układu). Z powstaniem kompilatora krzemowego wiąże się możliwość szybkiego zaspokojenia potrzeb na układy dedykowane. Stwarza to zarazem możliwość ochrony projektów przed kopiowaniem.

Specyfikacja układu VLSI jest integralną częścią procesu kompilacji krzemowej. Narzuca zgodną wewnętrzną reprezentację danych, wykorzystywaną przez inne narzędzia środowiska kompilatora. W procesie specyfikacji układów cyfrowych można wykorzystać język programowania wysokiego poziomu, taki jak ADA, Modula-2, Simula-67, Smalltalk czy Loglan [2,3,4,6,17,23, 39,45,49] . Wobec nieustannego rozwoju technologii i wzrostu złożoności systemów cyfrowych projektant jest zmuszony do operowania na coraz to wyższym poziomie abstrakcji. Dorobek języków programowania w znacznej mierze może być wykorzystany w zakresie projektowania układu scalonego w języku opisu sprzętu, szczególnie w zakresie reprezentacji danych.

W ostatnich latach obserwuje się intensywny rozwój nowej klasy języków programowania, tzw. języków obiektowo zorientowanych [9,12,13,27,29,42, 55] . Ich wykorzystanie w procesie programowania wymaga zastosowania nowej metodologii projektowania systemów, różnej od metod stosowanych w tradycyjnych językach, takich jak Algol, C, czy PASCAL [5,8,25,44] . Można pokazać, że języki obiektowo zorientowane są wygodnym narzędziem specyfikacji układów VLSI.

2. Projektowanie układów VLSI

Układy cyfrowe można klasyfikować według stopnia ich integracji. Dla układów VLSI, czyli układów o bardzo dużym stopniu scalenia, przyjmuje się, że minimalny wymiar obszarów występujących na powierzchni struktury półprzewodnikowej jest równy 3 mikrometry lub mniejszy, a liczba prostych elementów logicznych przekracza 10 tys. [37] . Przy tak dużym stopniu scalenia zmniejszenie kosztu wyrobu zależy przede wszystkim od zmniejszenia kosztów opracowania projektu, szczególnie przy małej wielkości produkcji (koszt jednego układu scalonego = A/B + C, gdzie A - koszt opracowania, B - wielkość produkcji, C - koszt produkcji jednego układu scalonego). Rozwiązanie problemu złożoności układów VLSI staje się możliwe przez zautomatyzowanie procesu ich projektowania.

2.1. Kompilacja krzemowa

Kompilacja krzemowa jest nową gałęzią automatycznego projektowania zintegrowanych układów cyfrowych. Reprezentuje proces translacji specyfikacji projektu układu w maski obwodu zintegrowanego. Na przestrzeni ostatnich

kilku lat zmieniły się podstawowe technologie wytwarzania układów cyfrowych. W kompilatorze krzemowym wykorzystuje się standardowe komórki do projektowania wyłożenia układu (ang. layout), tj.: tablice bramek (PLA, SLA), pamięci ROM; umożliwia to zautomatyzowanie procesu projektowania, a przez to zmniejszenie czasu realizacji prototypu oraz zwiększenie niezawodności jego wykonania.

Można wymienić kilka powodów powstania kompilacji krzemowej [16] :

- technologia wytwarzania układów zintegrowanych obejmuje możliwości integracji układów o wysokiej złożoności w jednej kostce krzemowej 100 tys. - 1 mln tranzystorów.
- powstanie nowych kompilatorów i baz danych, przydatnych w procesie projektowania,
- powstanie szybkich algorytmów rozwiązujących problemy rozmieszczenia elementów i połączeń, minimalizacji funkcji logicznych, syntezy symulacji, generowania testów (umożliwia to połączenie algorytmów w jeden proces automatycznego projektowania),
- nowe stacje projektowania (ang. workstations) oferują inżynierom duże możliwości obliczeniowe przy coraz mniejszej cenie: 32-bitowe procesory, kontrolery pamięci wirtualnej, megabajty pamięci dynamicznej, setki megabajtów pamięci na twardych dyskach, wysokiej rozdzielczości grafika.

Projektowanie układów VLSI w kompilatorze krzemowym umożliwia realizację dowolnych funkcji w pojedynczej kostce krzemowej. Zamiast użycia standardowych układów projektant może zaproponować układy zintegrowane, specyficzne dla danego zastosowania ASIC (ang. Application Specific Integrated Circuit). Często również realizuje się wiele funkcji w pojedynczej kostce scalonej wraz z umieszczeniem jej razem ze standardowymi układami (dopasowanie do otoczenia, itd.). Funkcje te mogą również składać się z części standardowych. Ponadto wytwórcy elementów standardowych starają się rozszerzyć własności tych elementów o nowe dodatkowe cechy. Każda odmiana elementu standardowego zawiera nowe cechy zintegrowane z układem podstawowym. Prowadzi to do powstania nowej klasy układów zintegrowanych SASIC (ang. Standard Application Specific Integrated Circuit).

2.2. Zagadnienie specyfikacji sprzętu elementem środowiska automatycznego projektowania układów VLSI

Etap specyfikacji projektu jest integralną częścią projektowania układów cyfrowych. Model reprezentujący projekt układu cyfrowego może być typu: behawioralnego, strukturalnego lub fizykalnego. Każdy z wymienionych typów specyfikacji posiada następujące poziomy abstrakcji [53] :

- poziom architektoniczny (ang. architectural level) czyli sprecyzowanie architektury systemu, tj. zdefiniowanie bloków operacyjnych, rodzaju operacji logicznych i arytmetycznych,

- poziom algorytmiczny (ang. algorythmic level), czyli zdefiniowanie algorytmu działania układu, podobnie jak w przypadku języków programowania,
- poziom bloków funkcjonalnych (ang. functional block level), czyli rozbicie systemu na elementarne bloki funkcjonalne precyzujące zależności funkcjonalne między poszczególnymi blokami lub rejestrami,
- poziom układowy (ang. circuit level), czyli schemat elektryczny układu połączony z rozmieszczeniem (ang. placement) i wyłożeniem jego warstw na płaszczyźnie półprzewodnika.

Proces projektowania układu rozumiany jest jako ciąg transformacji pomiędzy wyszczególnionymi modelami [36] . Każda transformacja projektu połączona jest z procesem weryfikacji projektu oraz badaniem zgodności jego modeli.

Specyfikacja układu cyfrowego wymaga zgodnej wewnętrznej reprezentacji akceptowanej i wykorzystywanej przez różne narzędzia środowiska (translator języka opisu, symulatory). Powinna jednocześnie gwarantować efektywną reprezentację w bazie danych projektowych. W systemie CAD narzędziem specyfikacji układu jest język opisu sprzętu.

2.3. Wprowadzenie do problematyki języków opisu i projektowania sprzętu

Historia języków opisu sprzętu ma ponad trzydzieści lat. W okresie tym były one wykorzystywane najczęściej w środowisku akademickim. Rzadko korzystał z nich projektant - inżynier. Sytuacja ta zmienia się jednakże stopniowo wraz z rozwojem technologii wytwarzania układów cyfrowych. Środowisko projektanta sprzętu musi ulec przeobrażeniu, by mógł on sprostać rosnącej złożoności układów VLSI [11,32] . Zadaniem języków opisu sprzętu jest [54] :

- wspieranie systematycznego - hierarchicznego projektowania sprzętu,
- umożliwienie opisu projektu oraz jego sprawdzenie w całym procesie jego powstania,
- ułatwienie wymiany informacji pomiędzy projektantami oraz całymi zespołami projektowymi,
- ułatwienie dokumentowania projektu,
- umożliwienie gromadzenia doświadczeń projektantów (najlepsze rozwiązania wchodzą w skład biblioteki projektów),
- zapewnienie możliwości wykorzystania w projektowanym układzie ostatnich osiągnięć w dziedzinie technologii,
- umożliwienie wygenerowania nowych wersji projektów, realizowanych w nowych technologiach na podstawie rozwiązań projektowych, przechowywanych w bibliotece projektów.

Przykładami wybranych języków opisu i projektowania sprzętu są [11,38, 46] :

- CONLAN (ang. a Consensus HDL) - rodzina języków opisu sprzętu opartych na języku podstawowym BCL [41] ,

- CAP (ang. Concurrent Algorithmic Programming Language), łączący metody algorytmicznego programowania z sieciami Petrii [43] ,
- KARL (Kaiserslautern Register transfer Language) - narzędzie do specyfikacji projektu wejściem graficznym (ABL - język "obrazów blokowych") [38] ,
- ISPS (ang. Instruction Set Processor Specifications) - język projektowania i symulacji architektury układów [26] ,
- OSM (Opis Struktur Mikroprogramowanych) [24,31,33,34] - przeznaczony do opisu i symulacji układów mikroprogramowanych oraz
- VHDL (ang. VHSIC HDL) [51,52] .

Języki te są reprezentantami pewnej klasy języków opisu sprzętu, które różnią się zarówno sposobem zapisu jednostki projektowej, metodą projektowania, jak również sposobem realizacji.

Traktując VHDL jako standard w dziedzinie języków opisu sprzętu można następująco przedstawić podstawowe jego cechy [1,19,20,21,22,28,46,54] :

- umożliwia hierarchizację opisu,
- projekt układu może być reprezentowany na poziomie abstrakcji: systemowym, architektonicznym, przesłań międzyrejestrowych i logicznym,
- umożliwia tworzenie równoważnych funkcjonalnych modeli,
- umożliwia reprezentację dynamiki układu cyfrowego oraz naturalną reprezentację współbieżnych operacji w sprzęcie,
- pozwala na rozdzielenie sterowania i danych,
- umożliwia dekompozycję złożonego zadania na podzadania oraz
- zapewnia niezależność od technologii, metodologii projektowania czy narzędzia wspomagania projektowania.

Aktualna definicja VHDL-u nie zawiera mechanizmów językowych dla reprezentacji sprzętu układu cyfrowego na poziomach poniżej poziomu bramkowego.

Można jednakże pokazać, że obiektowo zorientowane środowisko programowe może tworzyć podstawę języka opisu sprzętu o nowych własnościach oraz że w środowisku tym można reprezentować układy na dowolnym poziomie abstrakcji, dla dowolnego typu opisu.

3. Programowanie obiektowo zorientowane

Języki obiektowo zorientowane umożliwiają projektowanie elementów składowych systemu w postaci obiektów. Typowy język obiektowo zorientowany, taki jak SMALLTALK, zapewnia projektantom środowisko zawierające "tylko obiekty", Często jest on wskazywany jako "ekstremalny" przykład języka obiektowo zorientowanego. Do tej klasy języków zaliczyć można również języki takie, jak: Neon, ExperCommon LISP, Flavours, Actor, Garden, a także FOOPS [9,13,14] . Można jednakże wskazać na szereg języków konwencjonalnych, rozszerzonych jedynie o cechy umożliwiające realizację obiektowo

zorientowanego programowania, takie jak Simula, Loglan, Objective-C, C++ i Objective Pascal [9,12,13,14,30,42]. Języki tego typu, poza możliwością tworzenia klas, posiadają również podprogramy będące elementarnymi blokami programu.

3.1. Cechy języków obiektowo zorientowanych

Język programowania może być przyporządkowany do grupy języków obiektowo zorientowanych, jeżeli charakteryzuje się odpowiednim zestawem atrybutów. Można następująco przedstawić cechy i mechanizmy języka obiektowo zorientowanego:

- zamykanie stanu i zachowania obiektu w przestrzeni, czyli ograniczenie deklaracji atrybutów obiektu do definicji wzorca (ang. encapsulation),
- posiadanie klas,
- możliwość konkretyzacji obiektów (ang. instantiation),
- mechanizm dziedziczenia (ang. inheritance),
- możliwość tworzenia podklas,
- polimorfizm,
- dynamiczne łączenie (ang. dynamic binding),
- mechanizm chowania informacji (ang. information hiding),
- programowe "odśmiecanie" pamięci (ang. garbage collection).

Rzeczywisty obiekt w systemie tworzy jednostkę "zamkniętą", charakteryzowaną poprzez jej stan i zachowanie, przy czym stan wewnętrzny jednostki może być zmieniany za pomocą operacji reprezentujących jej zachowanie.

Klasa jest wzorcem dla obiektów. W klasie deklarowana jest informacja w postaci zbioru metod (procedur) oraz atrybutów (zmiennych będących stanem obiektu). Obiekt jest konkretyzacją klasy. Każdy konkretyzowany z klasy obiekt posiada swój własny indywidualny zbiór wartości zmiennych.

Budowa problemowo zorientowanych systemów może opierać się na mechanizmie dziedziczenia (konkatenacji), umożliwiającym przekazywanie tych samych atrybutów i metod jednej klasy obiektów do innych klas, bez ich kopiowania.

Zdolność języka, która pozwala na poprawną reakcję różnego rodzaju obiektów, na to samo żądanie (wywołanie procedury) w swój własny indywidualny sposób, nazywa się polimorfizmem. Wywołanie metody o tej samej nazwie dla różnych obiektów spowoduje różną jej realizację. W zależności od klasy obiektu, który ją wywołał (powoduje uruchomienie metody związanej tylko z tym wybranym obiektem). Przyporządkowanie metody do obiektu jest realizowane w trakcie wykonania programu. Jest to tzw. dynamiczne łączenie metody z wybranym obiektem.

Projektując klasy obiektów możemy pewne atrybuty tych klas zabezpieczyć przed dostępem z innych obiektów w czasie realizacji programu, Do tego celu służy mechanizm chowania informacji. Usuwanie obiektów systemu,

nie biorących udziału w dalszej części programu, może odbywać się na drodze programowej lub automatycznie poza programem, w środowisku systemu.

3.2. Pojęcie obiektu

Obiekt jest jednostką programową istniejącą w czasie i przestrzeni (pamięci), której zachowanie jest charakteryzowane zarówno przez jego operacje wewnętrzne (może być uaktywniany przez działanie innych obiektów), jak również przez jego oddziaływanie na inne obiekty. W rzeczywistym systemie obiektowo zorientowanym realizacja programu ma miejsce w samych obiektach. Dlatego obiekt jest traktowany jako aktywna inteligentna jednostka odpowiedzialna za swoje własne programowe zachowanie. Pojęcie obiektu odgrywa zasadniczą, centralną rolę w środowisku obiektowo zorientowanym. Obiekt jest jednostką, która:

- jest modułem utworzonym z wzorca (klasy lub podklasy),
- posiada stan,
- jest charakteryzowana przez działanie na innych obiektach,
- dopuszcza zmianę stanu obiektu przez inne obiekty za pomocą "własnych" metod,
- jest wskazywana i dostępna przez nazwę,
- umożliwia ograniczenie dostępu do własnych atrybutów i operacji z innych obiektów (ograniczenie "widoczności"),
- jest reprezentowana w środowisku innych obiektów przez swoją specyfikację, przy czym implementacja tej jednostki jest w tym środowisku niewidoczna.

Obiekt jest modułem, który powstał z wzorca, tj. klasy zawierającej wszystkie informacje potrzebne do jego konstrukcji. Klasa może być wzorcem dla wielu obiektów posiadających te same cechy. W języku Simula czy SMALLTALK obiekt jest podobny do rekordu występującego w języku Pascal. Jest on jednakże bardziej rozbudowany, szczególnie o zbiór procedur (metod).

Obiekt posiada swój stan w postaci zbioru zmiennych (ang. instance variables). Zmienne mogą być typu podstawowego, np. integer lub typu innego obiektu. Każdy obiekt posiada swój własny zbiór zmiennych. Zarówno zmienne tymczasowe (w metodach), jak i zmienne obiektu mogą być zmieniane przez uaktywnienie metod. W przeciwieństwie do zmiennych tymczasowych wartości zmiennych obiektu nie są usuwane z pamięci komputera, lecz pozostają w niej nawet po zakończeniu realizacji metody (są w dalszym ciągu dostępne).

W skład klasy wchodzą, oprócz jej atrybutów, metody, będące podobnie jak w językach strukturalnych procedurami. Wszystkie obiekty danej klasy posiadają identyczny zadeklarowany w tej klasie zbiór metod. Metody zawarte w klasie obiektu stanowią o jego zachowaniu. Dostęp do nich jest możliwy dzięki mechanizmom języka, tj. zdalny dostęp w języku Simula czy ADA lub nazwę metody w języku SMALLTALK.

Obiekt jest traktowany zawsze jako całość poprzez użycie jego nazwy (jest przez nią wskazywany). Elementy i własności obiektu lub klasy obiektów widziane z "zewnątrz" są jego specyfikacją. Specyfikacja obiektu jest dostępna ("widziana") przez obiekty całego projektowanego systemu. Część wewnętrzna obiektu jest jego implementacją i jest ukryta przed pozostałymi obiektami systemu. Dla wybranej specyfikacji obiektu może istnieć wiele implementacji reprezentujących jego zachowanie, przy czym zmiana części implementacyjnej nie powoduje zmian innych elementów systemu. Umożliwia to wygodną dekompozycję procesu projektowania do postaci lokalnych modułów systemu.

Własności języków obiektowo zorientowanych wskazują na to, że można budować na ich podstawie dowolne systemy modelowania, a m.in. systemy modelowania układów VLSI.

4. Sformułowanie języka opisu układów VLSI. Przykłady zastosowań

W punkcie tym przedstawia się metodę projektowania układów cyfrowych VLSI w środowisku obiektowo zorientowanym. Podstawowym elementem takiego środowiska jest język programowania o cechach typowego języka obiektowo zorientowanego. Język ten rozszerzony jest dodatkowo o elementy języka opisu sprzętu, przy zachowaniu jednak wszystkich swoich obiektowo zorientowanych cech. Rozszerzenie to tworzy system modelowania układów cyfrowych w nowym obiektowo zorientowanym środowisku projektowym.

Wybór obiektowo zorientowanego środowiska projektowego narzuca związaną z tym środowiskiem metodologię projektowania oraz postać struktury danych. Nowy element systemu deklarowany jest jako klasa. Natomiast wykorzystanie tego elementu jako składnika w deklaracji innego elementu określane jest jako konkretyzacja zdefiniowanej już klasy.

Wyspecyfikowany w systemie modelowania projekt układu VLSI jest zarazem jego modelem funkcjonalnym, strukturalnym czy fizykalnym. Dysponując modelem funkcjonalnym układu, wskazane byłoby zdefiniowanie narzędzi umożliwiających jego symulację. Środowisko obiektowo zorientowane pozwala na naturalne przedstawienie procesów współbieżnych (ponieważ sam obiekt może być procesem), a tym samym umożliwia implementację narzędzi symulacyjnych.

Do celów badawczych jako przedstawiciel języków obiektowo zorientowanych został wybrany język programowania Loglan.

4.1. Propozycja systemu modelowania układów VLSI opartego na wybranym języku obiektowozorientowanym (OOHDL)

System modelowania układów VLSI jest narzędziem wspomagającym użytkownika w procesie projektowania układów. W systemie modelowania układ VLSI może być specyfikowany na dowolnym poziomie abstrakcji. Reprezentacja wybranego poziomu opisu układu zawiera określony zestaw atrybutów, zależny

jedynie od typu układu oraz od stopnia szczegółowości jego opisu. System modelowania powinien posiadać wszystkie możliwe mechanizmy ułatwiające "konwersację" z użytkownikiem. Budowa systemu modelowania układów w języku obiektowo zorientowanym może przebiegać w następujących fazach [5,39]

- identyfikacji obiektów systemu i ich atrybutów,
- identyfikacji operacji dla obiektów systemu,
- ustalenia relacji pomiędzy obiektami systemu,
- ustalenia dostępu do atrybutów obiektów systemu oraz
- implementacji zadeklarowanych w poprzednich fazach obiektów.

Fazy realizacji środowiska do opisu i modelowania układów VLSI można przedstawić w kategoriach języka programowania Loglan, posiadającego atrybuty języka obiektowozorientowanego.

4.1.1. Identyfikacja obiektów systemu i ich atrybutów

Pierwszym krokiem budowy systemu modelowania układów cyfrowych jest znalezienie zbioru klas elementów, wchodzących w skład układów cyfrowych lub służących do ich budowy oraz przyporządkowanie tym klasom określonych atrybutów.

Rozpatrując klasy układów cyfrowych można przyjąć, że mimo występujących między nimi różnic posiadają one pewne wspólne atrybuty i operacje. Znaczy to, że dla wszystkich układów VLSI można zadeklarować wspólną klasę, tzw. superklasę, której własności są przekazywane innym klasom układów. Umożliwia to mechanizm prefiksowania. Tą klasą może być:

class integrated-circuit, która definiuje podstawowe atrybuty układów VLSI, tj.: nazwę, funkcję, wejścia, wyjścia, strukturę oraz rozmieszczenie i wyłożenie na płaszczyźnie krzemu.

Poziom logiczny układu VLSI, czyli jego struktura, może być reprezentowany jako model połączonych ze sobą podukładów. Tworzenie podukładów, węzłów i sieci połączeń jest możliwe w przypadku zdefiniowania następujących klas:

- class wire, która wskazuje poziom sygnału w sieci połączeń; zakłada się, że sieć połączeń jest typu Manhattan (Rys. 1),
- class node, reprezentująca końcówki i węzły układu,
- class ic structure, która deklaruje zbiór składników układu wraz z ich połączeniami.

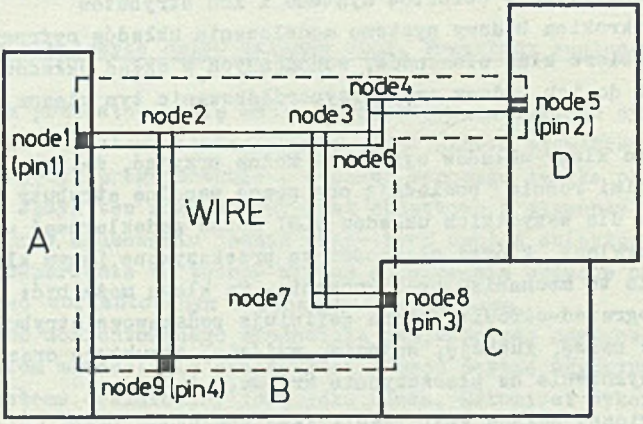
Opis fizykalny klasy układów VLSI oparty jest na elementach geometrycznych wyłożenia. Należy zdeklarować następujące klasy umożliwiające specyfikację poziomu układowego:

- class point, która określa współrzędne punktu na płaszczyźnie,
- class rectangle, definiującą położenie prostokąta na płaszczyźnie poprzez określenie współrzędnych prawego górnego rogu oraz jego długości boków; klasa ta może reprezentować rozmieszczenie układu na płaszczyźnie krzemu,

- class box, która definiuje prostokąt jw., uwzględniając atrybut reprezentowanej warstwy ("D" - dyfuzja, "P"-polikrzem, "M"-warstwa metalizacji, "I"-warstwa implantacji, "C"-warstwa kontaktowa); klasa ta reprezentuje warstwy układu i jego wewnętrzne połączenia (ścieżki),
- class layers, definiująca zbiór warstw układu.

Projektując maski układu użytkownik powinien dysponować mechanizmem transformacji rozmieszczenia wyłożenia układu. Jednym z elementów tego mechanizmu może być klasa:

class transformation, która określa translację, rotację i odbicie obiektów geometrycznych na płaszczyźnie. Translacja przenosi obiekt z punktu (0,0) do wybranego punktu na płaszczyźnie. Rotacja obraca wybrany obiekt względem punktu o określony kąt obrotu. Odbicie następuje względem osi OX lub OY.



Rys. 1. Przykładowa sieć połączeń
Fig. 1. A concept of wire class

4.1.2. Identyfikacja operacji dla obiektów systemu

Następnym etapem projektu powinno być określenie zbioru operacji (procedur) opisujących działanie zdefiniowanych wcześniej obiektów. Operacje mogą być wykonywane zarówno przez obiekty, jak również na obiektach.

Klasa integrated circuit, jako podstawowa klasa systemu modelowania, służąca do definiowania układów VLSI, powinna zawierać metody generowania sprzętu, struktury, rozmieszczenia i wyłożenia układu. Powinna również posiadać procedurę transformacji wyłożenia warstw układu na płaszczyźnie krzemu. Rozpatrując również przykładowo klasę node można założyć, że powinny w niej występować następujące procedury:

- procedure connect, tworząca połączenia pomiędzy węzłami układu w postaci ścieżek logicznych i geometrycznych,

- procedure set delay, która wyznacza opóźnienie ścieżki łączącej,
- procedure set value, ustalająca wartość logiczną w sieci połączeń węzła,
- procedure value of, która określa wartość logiczną w sieci połączeń węzła,
- procedure process run, która uaktywnia układy będące współprogramami, a których wejścia są połączone z danym węzłem tylko w przypadku zmiany wartości logicznej węzła.

Operacje zdefiniowane w pozostałych klasach systemu modelowania zostały szerzej przedstawione w punkcie 4.1.5., dotyczącym implementacji systemu.

4.1.3. Relacje pomiędzy obiektami systemu

Ustalenie relacji pomiędzy obiektami systemu polega na znalezieniu statycznej zależności pomiędzy nimi. Można to przedstawić jako określenie "widoczności" obiektów w relacji do innych obiektów.

System modelowania układów VLSI zawiera elementy specyfikacji układu na dowolnym poziomie abstrakcji. Relacje występujące pomiędzy obiektami systemu modelowania odwzorowują hierarchię tworzenia układów VLSI. Klasa podstawowa integrated circuit może generować obiekt sprzęgu układu jako zbiór wejść i wyjść, strukturę układu związaną z danym sprzęgiem, a także jego model fizyczny dla danej struktury. Wyłożenie układu jest zbiorem warstw w postaci prostokątów umieszczonych na płaszczyźnie krzemu, z określonymi dla tych warstw atrybutami. Model reprezentujący opis funkcjonalny układu może uczestniczyć w procesie jego symulacji.

4.1.4. Ustalenie dostępu do atrybutów obiektów

Zdefiniowane obiekty systemu komunikują się pomiędzy sobą przez wywołanie określonych funkcji i procedur, umożliwiających przesyłanie wartości stanu "na zewnątrz" obiektu oraz wprowadzanie informacji o wartości nowego stanu "do wnętrza" obiektu.

W trakcie specyfikacji poszczególnych elementów systemu, uwzględniając założenia z poprzedniego punktu, ustala się możliwość dostępu do każdego obiektu i klasy obiektów. Można również, korzystając ze zdefiniowanych mechanizmów języka, zabezpieczyć się przed niepoprawnym niedozwolonym dostępem do "wnętrza" obiektu.

4.1.5. Faza implementacji obiektów

W trakcie realizacji fazy implementacji może wystąpić iteracyjnie sekwencja poszczególnych faz. Ostatecznie otrzymujemy zbiór klas obiektów systemów modelowania wraz z określonymi operacjami oraz zbiór relacji występujących między obiektami. Definiując nowy wzór obiektów systemu modelowania deklarujemy nowe klasy języka. Klasy te mogą składać się z obiektów zadeklarowanych wcześniej jako modułów bibliotecznych lub być projektowane bezpośrednio przez użytkownika. Mogą również dziedziczyć atrybuty zdefiniowanych już klas.

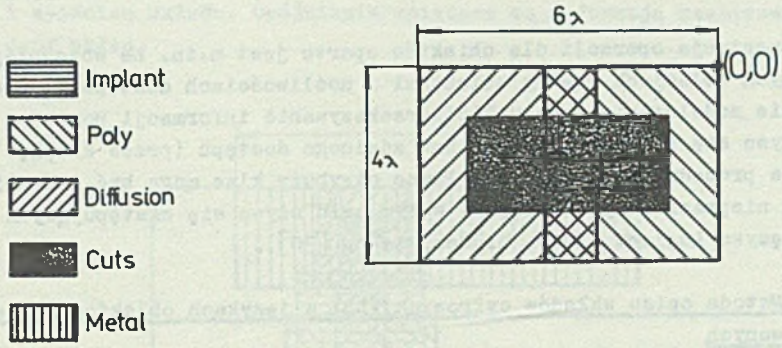
Klasa VLSI_DESIGN_TOOLS jako implementacja systemu modelowania układów VLSI w języku programowania Loglan dziedziczy wartości klasy simulation. Klasa simulation stanowi integralną część środowiska języka Loglan umożliwiającą modelowanie procesów w postaci współprogramów (ang. coroutine). Klasa integrated circuit, jako klasa podstawowa układów VLSI prefiksowana jest współprogramem simprocess z klasy simulation (Rys. 2). Umożliwia to przedstawienie projektowanych układów VLSI jako procesów (pseudo)współbieżnych.

```
unit integrated_circuit: simprocess class (t: transformation, name: string):
    unit virtual functional_description procedure (AI, AO: arrayof node;
                                                   output delay: real):
    end functional_description;
    (*procedury generujące sprzęg, strukturę, rozmieszczenie
     i wyłożenie *)
    unit transform: procedure:
        (* treść procedury transform *)
    end transform;
    var inputs, outputs: node;
        structure: ic_structure;
        placement: rectangle;
        layout: layers;
    begin
        writeln (name, "is created");
        inner;
        call transform;
        return;
        (* simulation part *)
    end integrated_circuit;
```

Rys. 2. Podstawowa klasa układów cyfrowych
Fig. 2. A basic class of integrated circuit

Elementami klasy integrated_circuit są: deklaracja funkcji w postaci procedury wirtualnej functional_description, deklaracja sprzęgu oraz deklaracje struktury, rozmieszczenia i wyłożenia na płaszczyźnie krzemu. Sprzęg projektowanego układu określa zbiór jego wejść i wyjść. Wejścia i wyjścia sprzęgu są węzłami układu (obiekty klasy node). Struktura układu może być obiektem klasy ic_structure, deklarującej zbiór składników układu i ich połączeń. W przypadku definiowania układu VLSI jako jednostki podstawowej reprezentowanej przez funkcję i wyłożenie warstw atrybut struktury układu jest równy none (nie jest konkretyzowany). Wyłożenie układu jest obiektem klasy layers, deklarującej zbiór poszczególnych warstw układu.

W klasie VLSI_DESIGN_TOOLS można zadeklarować również podstawowe klasy sprzęgowe, ułatwiające projektantowi definiowanie bardziej złożonych projektów. Przykładem takiej jednostki projektowej może być klasa butting_ contact (Rys. 3).



b)

```
unit butting_contact: class(t: transformation);
    var placement: rectangle,
        diff, poly, cont: box;
    begin
        placement: = new rectangle (6,4,0,0);
        diff: = new box (3.5,4,2.5,0,'D');
        poly: = new box (3.5,4,0,0,'P');
        cont: = new box (1,2,1,1,'C');
    end butting_contact;
```

Rys. 3. Klasa butting_contact: a) topografia, b) specyfikacja

Fig. 3. The butting_contact class: a) layout, b) textual representation

Poszczególne klasy systemu modelowania mają określone operacje w postaci funkcji i procedur języka Loglan. Deklaracja procedury wirtualnej functional_description w klasie integrated_circuit pozwala na użycie jej nazwy w części symulacyjnej tej klasy. Procedura ta reprezentuje opis funkcjonalny wybranego układu i jest wywoływana w trakcie procesu symulacji, czyli w trakcie realizacji programu. W klasie integrated_circuit zadeklarowana jest również procedura transform, która w rzeczywistości wywołuje procedurę transformacji w klasie layers. Oznacza to, że wszystkie projektowane układy w systemie modelowania mogą korzystać z jednej procedury transformacji.

W klasie node, reprezentującej węzły i końcówki układu, zdefiniowano procedurę connect, łączącą węzły w łańcuch węzłów. Jeżeli dodatkowo węzły są końcówkami układu, są one łączone również w łańcuch końcówek. Procedura connect tworzy nową ścieżkę pomiędzy węzłami, dlatego można przyjąć, że w trakcie procesu symulacji znane jest opóźnienie sygnału przechodzącego przez tę ścieżkę. Opóźnienie to jest zależne od parametrów elektrycznych ścieżki.

Implementacja operacji dla obiektów oparta jest m.in. na wcześniej zdefiniowanych relacjach między obiektami i możliwościach dostępu do nich. W systemie modelowania układu VLSI przekazywanie informacji pomiędzy obiektami odbywa się za pomocą mechanizmu zdalnego dostępu (przez kropkę) przez wywołanie procedury lub funkcji. Pewne atrybuty klas mogą być zabezpieczane przed nieprawidłowym dostępem. W tym celu używa się następujących dyrektyw języka Loglan: close, hidden, taken [30] .

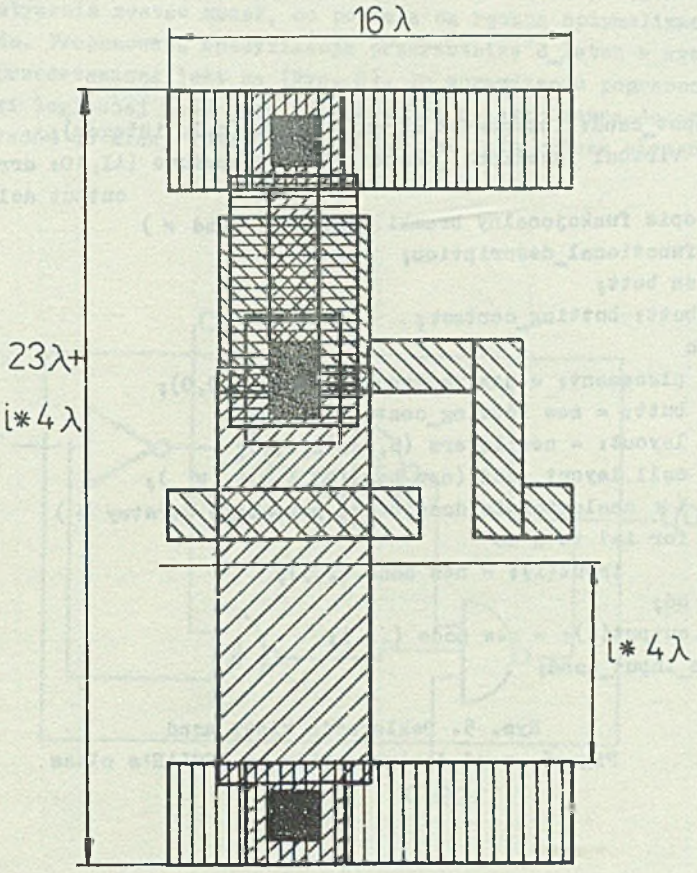
4.2. Metoda opisu układów cyfrowych VLSI w językach obiektowozorientowanych

Projektant modelując układy cyfrowe w nowym środowisku projektowym dysponuje nowo utworzonymi klasami systemu modelowania, a także może wykorzystywać mechanizmy języka obiektowo zorientowanego, na którego bazie system modelowania powstał. Znaczy to, że w trakcie definiowania nowego układu cyfrowego postępuje tak, jakby dobudowywał nowy element do istniejącego środowiska systemu. Może odbywać się to również w fazach obiektowozorientowanej metodologii projektowania. Nowy układ może być dołączony do istniejącego środowiska, a następnie wykorzystywany w nowych projektach jako składnik lub punkt wyjściowy dla innych projektów. Zdefiniowane układy cyfrowe mogą tworzyć bibliotekę, z której projektant może korzystać w trakcie deklarowania nowego układu. Biblioteka klas układów jest podobna do katalogu, przy czym poza opisem układu mogą być również przechowywane w niej elementy służące do jego symulacji.

Projektując układ cyfrowy jako nową klasę w obiektowo zorientowanym środowisku można skoncentrować się na projekcie tej klasy, wykorzystując środowisko jedynie jako narzędzie opisu. Jest to jedna z cech języka obiektowo zorientowanego, która umożliwia deklarację klasy bez zmiany pozostałych elementów środowiska. Deklaracja klasy nowego układu cyfrowego zależy od przyjętego poziomu opisu dla tego układu. Dla opisu behawioralnego układ reprezentowany jest poprzez funkcje wyjść zależne od wejść. W przypadku opisu strukturalnego projekt układu obejmuje wyszczególnienie składników układu oraz ich połączeń. Forma reprezentacji geometrii układu obejmuje zarówno rozmieszczenie składników, jak również wyłożenie jego warstw na płaszczyźnie krzemu. Zależy ona od struktury i technologii, według której układ ma zostać wykonany.

Z zadeklarowanej dowolnej klasy układu cyfrowego można konkretyzować dowolną ilość modułów (obiektów) tych układów. W trakcie konkretyzacji

układu generowane są wszystkie elementy jego specyfikacji. Rozmieszczenie elementów składowych klasy układu odnosi się do punktu początkowego (0,0). Rzeczywiste rozmieszczenie elementów klasy następuje w czasie generacji modułu w zależności od określonej dla niego transformacji. Parametry elektryczne poszczególnych warstw, które zależne są od wymiarów geometrycznych warstw, są wykorzystywane do określenia opóźnień sygnałów pomiędzy wejściem i wyjściem układu. Opóźnienia związane są z funkcją realizowaną przez dany układ.



Rys. 4. Topografia bramki logicznej nand
Fig. 4. Layout of nand gate object

Tworząc bibliotekę podstawowych układów znajdujemy ich opis funkcjonalny oraz wyłożenie na płaszczyźnie krzemu. Wzorując się na zakładanym wyłożeniu warstw bramki logicznej nand (Rys. 4) i traktując ją jako jeden z elementów podstawowych systemu, można wyspecyfikować jej opis w postaci

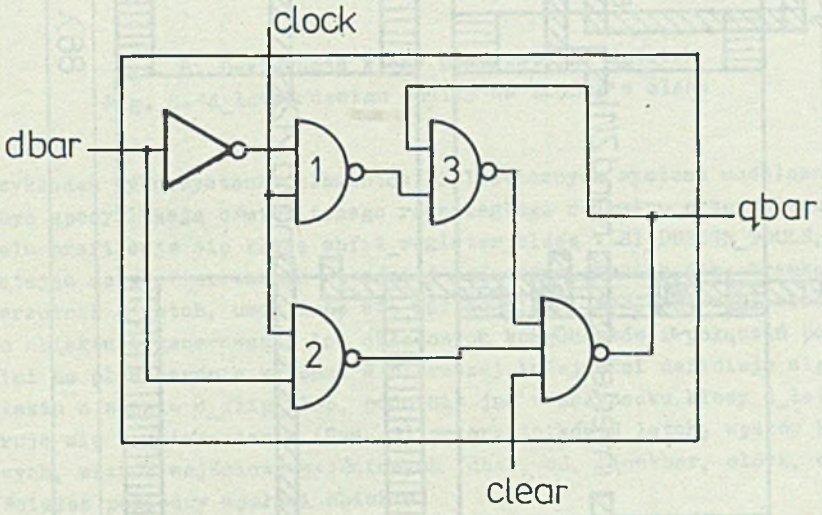
klasy n_input_nand (Rys. 5). Klasa n_input_nand jest prefiksowana klasa integrated_circuit i dziedziczy wszystkie atrybuty tej klasy.

Deklaracje wejść i wyjść typu node w klasie n_input_nand mają istotne znaczenie w procesie symulacji obiektów. Typ node może reprezentować prosty węzeł grafu połączeń lub końcówkę będącą wejściem lub wyjściem układu. Przy generacji węzła wprowadza się informację o rozmieszczeniu go na powierzchni krzemu. Przy generacji końcówki obiektu wprowadza się dodatkowo, jako parametr formalny klasy node, zmienną wskazującą na układ, który ją wygenerował.

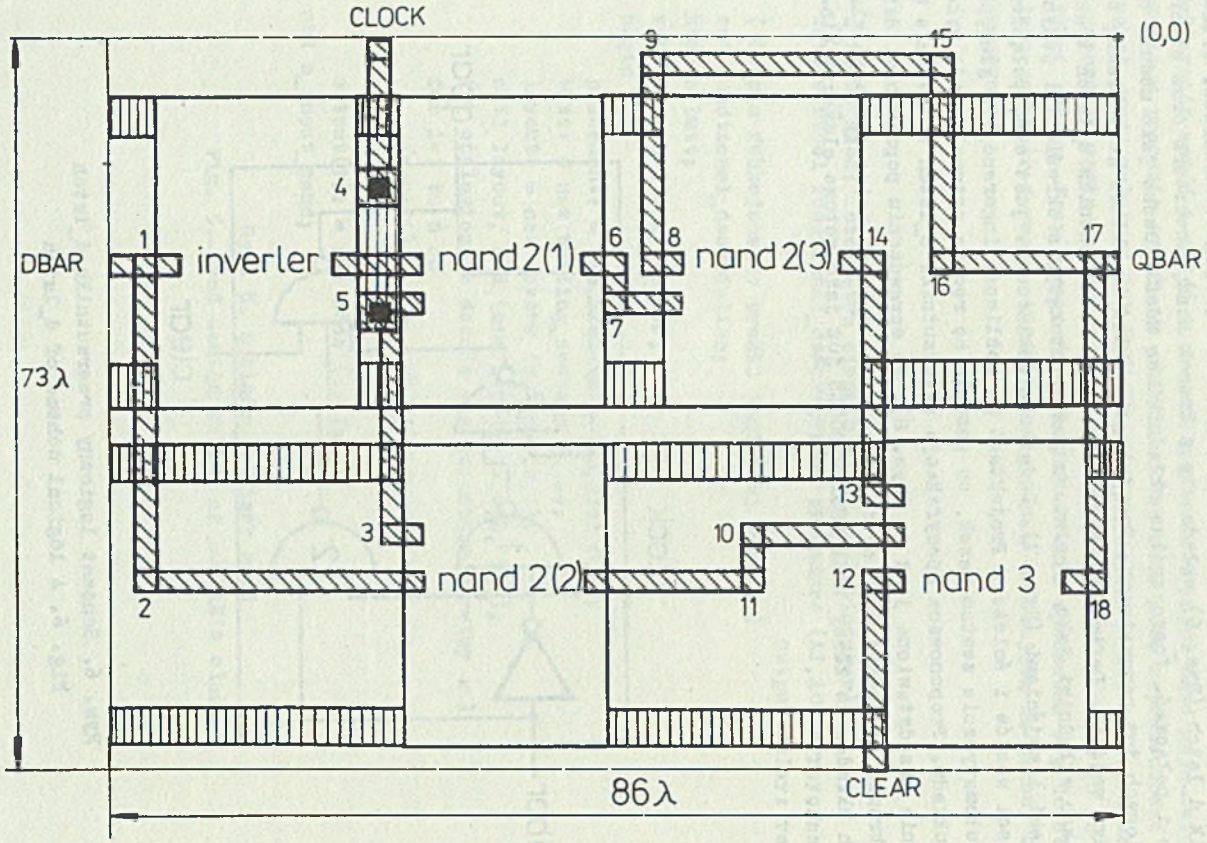
```
unit n_input_nand: integrated_circuit class (n,1: integer);
    unit virtual functional_description: procedure (AI,AO: arrayof node;
                                                    output delay: real);
    (* opis funkcjonalny bramki logicznej nand *)
    end functional_description;
    hidden butt;
    var butt: butting_contact;
    begin
        placement: = new rectangle (20,23+1,0,0);
        butt: = new butting_contact (...);
        layout: = new layers (8,4+n,2,2,1);
        call layout, add (new box (20,4,0,0, M  );
        (* analogicznie dodaje się pozostałe warstwy *)
        for i=1 to n do
            input(i): = new node (...);
        od;
        output(1): = new node (...);
    end n_input_nand;
```

Rys. 5. Deklaracja klasy nand
Fig. 5. nand design entity as LOGLAN's class

Projekt układu cyfrowego może obejmować specyfikację strukturalną. Opis funkcjonalny układu jest wtedy wypadkową funkcji jego składników. Przerzutnik d_latch (Rys. 6) składa się z bramek nand, inwertera oraz zbioru węzłów i połączeń. Poszczególne składniki są zdefiniowane jako moduły o określonych już parametrach. Przyjęto, że układ inwertera jest bramką nand o jednym wejściu. Definiując wyłożenie układu przerzutnika d_latch traktujemy jego składniki jako wypełnione już prostokąty, z określonymi jedynie wejściami i wyjściami (Rys. 7). Dodatkowo deklarowana jest sieć połączeń w postaci węzłów i ścieżek. Projektant ma możliwość ingerencji w generowany automatycznie zestaw masek, co pozwala na ręczną optymalizację wyłożenia układu. Proponowana specyfikacja przerzutnika d_latch w systemie modelowania przedstawiona jest na (Rys. 8). Po sprawdzeniu poprawności działania bramki logicznej nand oraz przerzutnika d_latch można dołączyć klasy tych układów do klasy VLSI_DESIGN_TOOLS już jako gotowe elementy biblioteczne.



Rys. 6. Schemat logiczny przerzutnika d_latch
Fig. 6. A logical scheme of d_latch

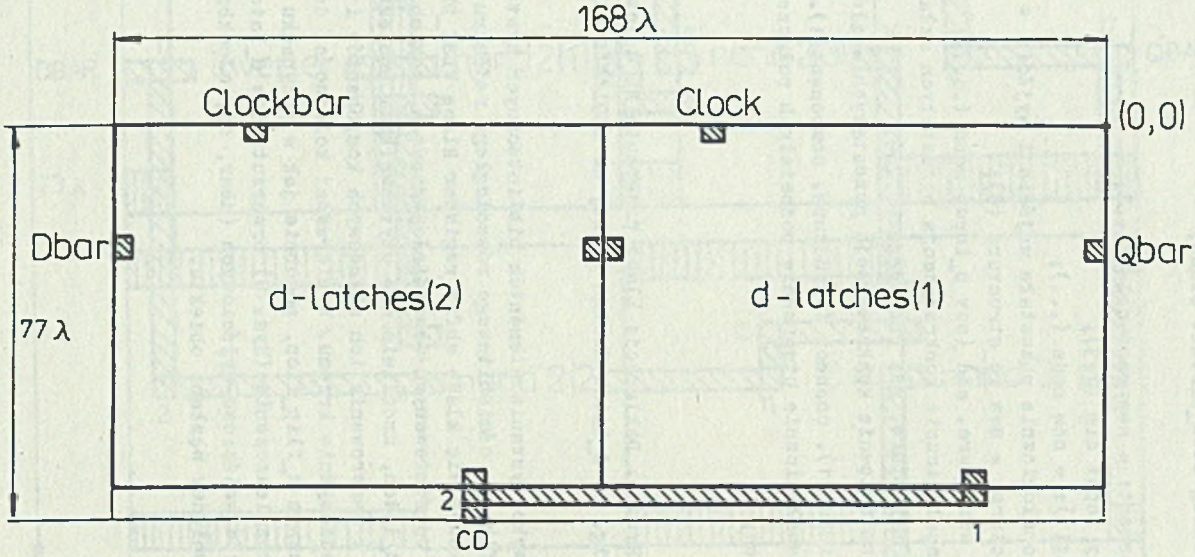


Rys. 7. Floor plan przerzutnika d_latch
Fig. 7. Floorplan of d_latch

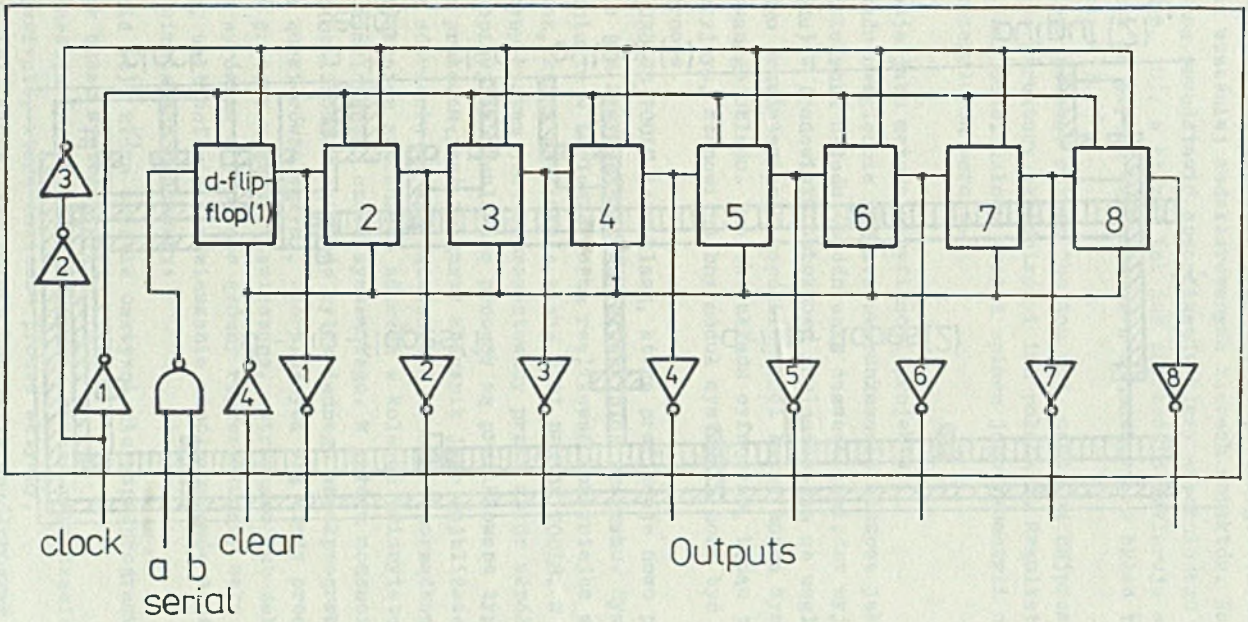
```
unit d_latch: integrated_circuit class;
    begin
        placement: = new rectangle (...);
        array input dim (1:3);
        input(1): = new node (...);
        ( * analogicznie pozostałe wejścia i wyjścia * )
        structure: = new ic_structure (5);
        call structure. add (new n_input_nand (...));
        ( * analogicznie konkretyzacja pozostałych składników * )
        call structure. set (new node (...));
        (  analogicznie konkretyzacja pozostałych węzłów  )
        callinput(1). connect (structure. component(1). input(1));
        ( * analogicznie utworzenie pozostałych połączeń * )
    end d_latch;
```

Rys. 8. Deklaracja klasy przerzutnika d_latch
Fig. 8. d_latch design entity as LOGLAN'S class

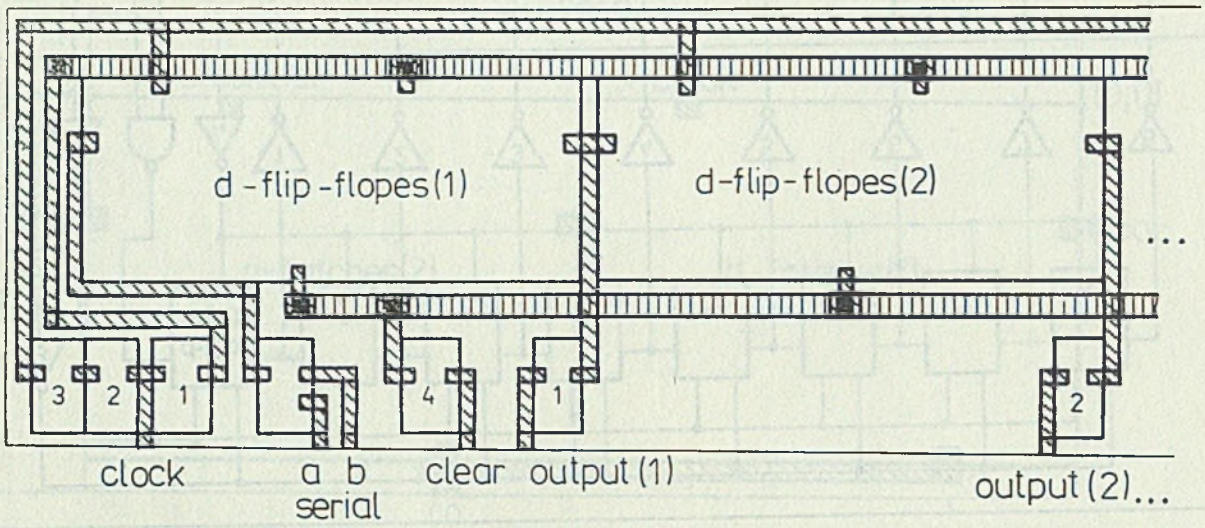
Przykładem wykorzystania elementów bibliotecznych systemu modelowania może być specyfikacją ośmiobitowego równoległego rejestru przesuwnego. W tym celu prefikauje się klasę shfit_register klasą VLSI_DESIGN_TOOLS, gdzie korzystając ze zdefiniowanych wcześniej jednostek, takich jak: bramka nand i przerzutnik d_latch, umożliwia się użytkownikowi projektowanie struktury nowego obiektu i generowanie ich składowych komponentów i połączeń pomiędzy nimi na płaszczyźnie krzemu. W pierwszej kolejności definiuje się klasę obiektu o nazwie d_flip_flop, podobnie jak w przypadku klasy d_latch, deklaruje się rozmieszczenie (Rys. 9) przerzutników d_latch, węzłów kontaktowych, węzłów wejściowo-wyjściowych (dbar, cd, clockbar, clock, qbar) oraz ścieżek pomiędzy węzłami obiektu.



Rys. 9. Floor plan przerzutnika d_flip_flop
Fig. 9. Floorplan of d_flip_flop



Rys. 10. Schemat logiczny ośmiobitowego równoległego rejestru przesuwnego
Fig. 10. A logical scheme of eight_bit_parallel_shift_register



Rys. 11. Floor plan rejestru przesuwnego

Fig. 11. Floorplan of eight_bit_parallel_shift_ _register

Jak widać, schemat postępowania przy tworzeniu nowego obiektu jest ustalony, bazuje na wcześniej zadeklarowanych klasach obiektów. Na potwierdzenie tego można przedstawić specyfikację klasy wspomnianego już rejestru przesuwnego (Rys. 10), w której tak jak poprzednio deklaruje się rozmieszczenie obiektu na płytce krzemu wraz z wchodzącymi w skład tego obiektu elementami (Rys. 11).

Projekt układu powstał metodą "bottom up", czyli od najprostszych elementów układu do struktury składników i ich połączeń. Rezultatem procesu projektowania jest model układu wraz z opisem jego geometrii w postaci masek dla poszczególnych warstw.

4.3. Symulacja jako metoda weryfikacji projektu

Projekt układu uwzględnia również uwarunkowania czasowe jakie występują w trakcie działania układu (opóźnienia czasowe pomiędzy wyjściami i wejściami układu).W środowisku obiektowo zorientowanym ze względu na możliwość wygodnego przedstawiania współbieżności łatwo można symulować działanie projektowanego układu. Projekt układu cyfrowego, będąc zarazem jego modelem symulacyjnym, stanowi osobny moduł systemu i może być traktowany jako odrębny proces.

Klasa VLSI_DESIGN_TOOLS jest klasą, która prefiksuje nowo projektowaną klasę układu, a tym samym prefiksuje główny blok programu. Symulację wybranej klasy obiektu w Loglanie można realizować korzystając z własności klasy simulation, która prefiksuje klasę VLSI_DESIGN_TOOLS. W klasie simulation modelowany system jest reprezentowany przez zbiór współprogramów. Współprogramy opisujące symulowane procesy są prefiksowane typem simprocess. Z każdym procesem jest związany znacznik jego najbliższego zdarzenia. Wszystkie zdarzenia są uporządkowane w czasie. Uporządkowanie to wiąże się z umieszczeniem znaczników zdarzeń w kolejce priorytetowej reprezentującej oś symulowanego czasu systemowego. W każdym momencie wykonywania programu realizowane są akcje tylko jednego współprogramu (działanie w systemie quasi-równoległym), który nazywa się wtedy procesem aktywnym. Pozostałe procesy mogą być zawieszone, wstrzymane lub zakończone. Wśród procesów wyróżnia się proces główny reprezentujący cały program. Do wstrzymania, uaktywniania i zawieszania współprogramów służą zdefiniowane w klasie simulation procedury:

- procedure hold (t), która opóźnia uaktywnienie współprogramu aktywnego o t jednostek czasu systemowego,
- procedure schedule (p,t), która uaktywnia proces p po czasie t,
- procedure passivate, która zawiesza proces aktywny,
- procedure run (p), która natychmiast uaktywnia współprogram p,
- procedure canacel (p), która likwiduje znacznik zdarzenia dla współprogramu p.

Klasa simulation prefiksowana jest typem priorytqueue, który umożliwia niedeterministyczne działanie programu symulacyjnego. W przypadku gdy wiele procesów zaplanowanych jest na tę samą chwilę czasu, wybrany jest proces, którego znacznik zdarzenia jest najmniejszym elementem kolejki priorytetowej.

Konkretyzowane obiekty, realizujące złożone funkcje logiczne, mogą składać się z podstawowych bramek logicznych lub innych bardziej złożonych układów cyfrowych. W przypadku zdefiniowania procedury wirtualnej, opisującej zachowanie obiektu, możliwa jest również symulacja obiektu na wyższym poziomie abstrakcji. Przykładem opisu funkcjonalnego układu cyfrowego jest specyfikacja bramki logicznej nand (Rys. 12).

```
unit virtual functional_description: procedure (AI,AO: arrayof node;
                                                cutput delay: real);
begin
    for i: = 1 to upper AI do
        if value_of (AI(i) = 'L' then
            call set_value (AO(1), 'H');
            delay: = tnl;
            return
        fi
    od:
    call set_value (AO(1), 'L');
    delay: = tnh;
end functional_description;
```

Rys. 12. Opis funkcjonalny bramki nand
Fig. 12. Functional specification of nand

Współprogram simprocess z klasy simulation prefiksuje klasę integrated_circuit. Każdy z projektowanych układów, będąc prefikcowany tą klasą, dziedziczy tym samym własności współprogramu. Każdy układ może być oddzielnym współprogramem lub składać się z współprogramów elementów podstawowych. Uaktywnianie współprogramu reprezentującego dany układ cyfrowy powoduje odczytanie wartości sygnału na wejściach układu, a następnie obliczenie jego funkcji. Zmiana wartości sygnału na wyjściu układu powoduje uaktywnienie wszystkich współprogramów reprezentujących układy połączone z tym wyjściem (procedure process_run). Uaktywnienie współprogramów następ-

puje po czasie opóźnienia wprowadzonym przez ścieżki łączące końcówki układu. Czas opóźnienia wyliczany jest dla każdej ścieżki osobno i dodawany do czasu symulacji. Podczas procesu symulacji układu pamiętane są wartości sygnałów wejściowych: bieżąca i ostatnia. Symulator pamięta historię ostatniej zmiany sygnału. Wartości tych sygnałów mogą być rejestrowane po każdej ich zmianie. W tym celu używa się procedury wirtualnej o nazwie print_all_pins, umożliwiającej rejestrowanie wybranych wartości sygnałów określonych w tekście programu symulacyjnego.

W przykładzie proces symulacji n-bitowego równoległego rejestru przesuwnego przebieg na poziomie bramek logicznych. Zakłada się, że czasy opóźnień są stałe, zależne od przyjętego wyłożenia warstw bramki. Po przetestowaniu nowego układu można umieścić go jako element biblioteczny w klasie VLSI_DESIGN_TOOLS.

5. Podsumowanie

W pracy pokazano możliwość reprezentacji języka opisu sprzętu w środowisku obiektowo zorientowanym. Przedstawiono również zalety wykorzystania własności języka obiektowo zorientowanego do opisu i projektowania układów VLSI, takich jak: abstrakcyjne typy danych, dziedziczenie, polimorfizm.

Wybrany język programowania Loglan umożliwił projektowanie systemu modelowania układu VLSI według nowej obiektowo zorientowanej metodologii. Przyjęta metoda realizacji systemu wykazuje następujące zalety:

- umożliwia odzwierciedlenie natury rzeczywistego systemu, jako że struktura systemu oparta jest na obiektach,
- poprzez chowanie danych i abstrakcję danych zwiększana jest niezawodność i umożliwia się oddzielenie proceduralnej i reprezentacyjnej specyfikacji od implementacji,
- dziedziczenie atrybutów i operacji obiektów pozwala na redukcję kodu programu (prostszy i łatwiejszy zapis programu przy mniejszej liczbie linii, krótszy czas projektowania),
- możliwość generacji obiektów klasy, zajmującej jedno oddzielne miejsce w programie, ułatwia zarządzanie danymi programu (zmniejsza się liczbę błędów programu),
- występuje naturalna współbieżność obiektów (obiekty zapisywane są jako współprogramy),

- łatwa gospodarka zasobami umożliwia prowadzenie dalszych operacji na obiektach takich, jak symulacja czy weryfikacja formalna (istnienie zbioru danych w postaci gotowych układów),
- wpływ zmian elementu projektu na całość projektowanego systemu jest ograniczony (giętkość i podatność na zmiany, zmniejszenie przypadkowych i złośliwych błędów w trakcie implementacji systemu) oraz
- programowanie w jednorodnym środowisku programowym podnosi poziom percepcji programisty (program jest blokiem całego systemu).

Zastosowanie języka obiektowo zorientowanego do projektowania klasy VLSI_DESIGN_TOOLS wydłużyło jednakże czas realizacji procedur (metod) programu. Zaistniała również konieczność poznania większości klas bibliotecznych przed przystąpieniem do procesu projektowania.

Podkreślić należy, że podstawową cechą obiektowo zorientowanego modelowania układów VLSI jest możliwość zmiany dowolnego elementu układu bez zmiany pozostałych jego atrybutów. W dekompozycji proceduralnej każda zmiana opisu zmiennej globalnej systemu powoduje zmianę wszystkich związanych z nią modułów. W dekompozycji obiektowo zorientowanej efekt zmiany obiektu jest bardziej lokalny. Pozwala to na rozbudowę systemu modelowania o nowe elementy, tj. algorytmy syntezy i weryfikacji, bez konieczności modyfikowania pozostałych klas systemu. System więc nie jest systemem zamkniętym; może być wzbogacany o nowe narzędzia ułatwiające użytkownikowi projektowanie układów.

Ostatnie badania w dziedzinie projektowania układów cyfrowych w systemie CAD [6,10,15,16,18,35,40,47,48] wykazują, że zaznacza się wyraźna tendencja definiowania obiektowo zorientowanych narzędzi programowych, używanych w procesie projektowania układów. Potwierdza to zatem celowość prowadzenia dalszych badań w tej dziedzinie.

6. LITERATURA

[1] Aylor J., Waxman R., Scarratt C.: "VHDL-Feature Description and Analys", IEEE Design and Test, April 1986.

[2] Barbacci M.: "Structural and behavioural description of digital systems, New computer architectures", J. Tiberghien (ed.), Academic Press, 1984.

[3] Barbacci M. et al.: "ADA as a Hardware Description Language: An Initial Report", Proc. of int. Symp. on Computer Hardware Description Languages and Their Applications, C.J. Koomen and T. Moto-oka (eds.), Tokyo, 1985.

[4] Baudet G. et al.: "The Relationship Between HDLs and Programming Languages", VLSI and Software Engng. Workshop, Port Chester, NY, 1982.

[5] Booch G.: "Object-Oriented Development", IEEE Transactions on Software Engineering, Vol.SE-12, No.2, Feb. 1986.

[6] Buchanan I.: "Modelling and Verification in Structured Integrated Circuit", Ph.D. thesis, Univ. of Edinburgh, July, 1980.

[7] Buckley J.: "High-level Hardware Description Languages: A new Computer - Aided Design Tool", British Telecomunications Engineering, Vol. 2, Jan. 1984.

[8] Buzzard G., Mudge T.: "Object-based computing and the ADA programming language", Computer, Vol. 18, No.3, p.12, 1985.

[9] Byte, special issue on Object-oriented Programming, August 1986.

[10] Caviglia R., et al.: "An object-oriented design framework implementing EDIF construct".

[11] Computer: Vol. 7, No.12, 1974; Vol. 10, No.6, 1977; Vol. 18, No.2, 1985; specjalne wydanie periodyku IEEE poświęcone językom opisu sprzętu.

[12] Cox B.: "Message/Object Programming: An Evaluationary Change in Programming Technology", IEEE SOFTWARE, Jan. 1984.

[13] Cox B.: "Object-Oriented Programming. An evolutionary approach", Addison_Wesley, 1986.

[14] Cox B.: "Object-Oriented Pre-Compiler - Programming Smalltalk - 80 Methods in C Language", Sigplan Notices, Vol. 18, Jan. 1983.

[15] Demers L., et al.: "An object-oriented integration of VLSI CAD Tools", 24th ACM/IEEE Design Automation Conference, Miami Beach, June 28-July, 1987.

[16] Gajski D., Dutt N., Paurgle B.: "Silicon Compilation (Tutorial)", IEEE Custom Integrated Circuits Conference, 1986.

[17] Girczyc E., Buhr R., Knight J.: "Applicability of a Subset of ADA as an Algorithmic Hardware Description Language for Graph-Based Hardware Compilation", IEEE Transactions on Computer-Aided Design, Vol. 4, No.2, April, 1985.

[18] Girczyc E., Ly T.: "STEM; An IC Design Environment Based on the Smalltak Model View Controller Construct", 24th ACM/IEEE Design Automation Conference, Miami Beach, June 28-July 1, 1987.

[19] Gizdoń H., Pawlak A., Wrona W.: "Hardware Description Languages - Introduction to VHDL", raport publikowany w GMD w RFN (W-40%).

[20] Gizdoń H., Pawlak A., Wrona W.: "VHDL - Ada języków opisu i projektowania sprzętu". Praca przygotowywana do druku, Informatyka (W-40%).

[21] Gizdoń H., Pawlak A., Wrona W.: "Język opisu sprzętu VHDL - podstawowe mechanizmy (cz. I i II)", praca przygotowywana do druku, Informatyka (W-40%).

[22] Gizdoń H., Pawlak A., Wrona W.: "Wykorzystanie VHDL-u do opisu i weryfikacji projektów układów cyfrowych", praca przygotowywana do druku, Informatyka (W-30%).

[23] Ghosh S.: "ADA as a Hardware Description Language and a Distributed Simulation Environment", Proc. ICCAD-85, Santa Clara, 1985.

[24] Gutowski R., Szturmowicz M.: "Opis języka OSM i instrukcja obsługi systemu 3SM", raport IPI, PAN, Warszawa, 1985.

[25] Guttag J., et al.: "The Design of Data Type Specification (Current Trend in Programming Methodology)", Vol. 4, Englewood Cliffs, NJ: Prentice-Hall, p.200, 1978.

[26] Hines J.: "Where VHDL fits within the CAD environment", 24th ACM/IEEE Design Automation Conference, Miami Beach, June 28-July 1, 1987.

[27] Horowitz E.: "Fundamentals of Programming Languages", Springer-Verlag 1984.

[28] IEEE Design and Test, April, 1986, specjalne wydanie poświęcone językowi VHDL.

[29] Kreczmar A.: "Object-Oriented Languages", Proc. Autumn School on Computer Science, Mrągowo, 1986.

[30] Kreczmar A., Salwicki A.: "Report on the Loglan 82 programming language", PWN Warszawa-Łódź, 1984.

[31] Marczyński R., et al.: "PINLAN - Language for Digital Integrated Circuit Description", Prace IPI PAN, Nr 453, Warszawa 1981.

[32] Marczyński R., Bąkowski P.: "What do the Computer Hardware Description Languages Describe?", Proc. Int. Symp. on Computer Hardware Description Languages and their Applications, Palo Alto, 1979.

[33] Marczyński R., et al.: "OSM - Microprogrammed Hardware Structure Description Language", Proc. Int. Symp. on CHDLs, New York, 1975.

[34] Marczyński R., et al.: "3SM Computer Simulation System and Some Problems of Microprogrammed Computer Simulation", EUROMICRO Conf. Nice, 1975.

[35] Marshall R., Buchanan I.: "Scale - a language for VLSI design", Edynburg, 1984.

[36] Mead C., Conway L.: "Introduction to VLSI systems", Addison-Wesley, 1980.

[37] Muroga S.: "Projektowanie układów VLSI". WNT, Warszawa 1986.

[38] Pawlak A.: "A tutorial Guide to Modern Hardware Description and Design Languages", Proc. 11-th EUROMICRO Symp., Brussels, 1985.

[39] Pawlak A., Wrona W.: "Modern Object-Oriented Programming Language as a HDL", Proc. of 8-th CHDL, Amsterdam, 1987.

[40] Piguet C., et al.: "Automatic generation of CMOS Layout Cells in a Hardware Description Language", EUROMICRO Conf., Bruxellas, 1985.

[41] Piloty R., et al.: "CONLAN Report". Lecture Notes in Computer Science, No.151, Springer Verlag, 1983.

[42] Oktaba H., Ratajczak W.: "Simula 67", PWN, 1980.

[43] Ramming F.: "DACAPO Language Reference Manual", Report Univ. of Dortmund, 1984.

[44] Rentsch T.: "Object-oriented programming", SIGPLAN Notices, Vol. 17, No.9, p.51, Sept. 1982.

[45] Robinson P., Dion J.: "Programming Languages for Hardware Description", Proc. 20-th Design Automation Conference, Miami Beach, 1983.

[46] Shadad M. et al.: "VHSIC Hardware Description Language Overview", Proc. ACM 84 Annual Conference, The Fifth Generation Challenge, 1984.

[47] Smit J. et al.: "The MoDL Hardware Design System", 8-th CHDL, Amsterdam, 1987.

[48] Spaanenburg L., et al.: "Separation of hierarchies in VLSI design", Proc. of 8-th CHDL, Amsterdam, 1987.

[49] Suzuki N.: "The Fifth Generation Kernel Language as a CHDL", keynote, 6-th Int. Symp. on CHDLs, Pittsburgh, 1983.

[50] Touati H.: "Is Ada an object-oriented programming language?", Sigplan Notices, V. 22, May 1987.

[51] VHDL Users's Manual, vol. 1-Tutorial, 1 August 1985, IR-MD-065-1.

[52] VHDL Language Reference Manual, Version 7.2, 1 August 1985, IR-MD-045-2.

[53] Walker R., Thomas D.: "A Model of Design Representation and Synthesis", IEEE Design Automation Conference, 1985.

[54] Waxman R.: "Hardware Design Language for Computer Design and Test", Computer, April, 1986.

[55] Yokote Y., Tokoro M.: "The design and implementation of concurrent Smalltalk", OOPSLA 86 Proceedings.

Recenzent: Prof.dr hab Andrzej Grzywak

Wpłynęło do Redakcji 20.09.1988

OBJECT-ORIENTED PROGRAMMING FOR VLSI CAD SYSTEM DEVELOPMENT

S u m m a r y

Up to now, the software development problems have been adressed using structured programming metodology with languages such as Pascal and C. This paper deals with use of object-oriented programming metodology for development of VLSI CAD applications. It analyses also the suitability of object-oriented programming, in particular, of LOGLAN language for specification of VLSI integrated circuit.

МЕТОД ОПИСАНИЯ И СИСТЕМА МОДЕЛИРОВАНИЯ СИСТЕМ В ЯЗЫКАХ ОБЪЕКТНО ОРИЕНТИРОВАННЫХ

Р е з ю м е

Спецификация системы является неотъемлемой частью процесса проектирования цифровых систем. Дает она согласованное внутреннее представление данных, используемое другими инструментами проектной среды. В процессе спецификации цифровых систем можно использовать язык программирования высокого уровня. Особенно адекватными для этой цели могут быть языки объектно ориентированы. Их использование в процессе программирования требует применения новой методологии проектирования систем, отличающейся от методов применяемых в традиционных языках программирования, таких как ПАСКАЛЬ или Си.