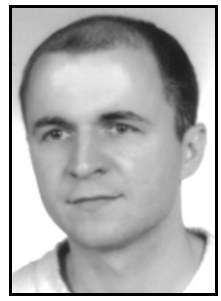
Mirosław MOŚCICKI
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY, WYDZIAŁ INFORMATYKI

Metoda generowania równań boolowskich dla podprogramów języka VHDL

Mgr inż. Mirosław MOŚCICKI

Ukończył studia na Wydziale Informatyki Politechniki Szczecińskiej w roku 2000. Jest asystentem w Katedrze Technik Programowania Wydziału Informatyki PS. Jego zainteresowania naukowe to programowanie, ze szczególnym uwzględnieniem tworzenia kompilatorów języków opisu sprzętu oraz testowanie oprogramowania.



e-mail: mmoscicki@wi.ps.pl

Streszczenie

W artykule zaprezentowano metodę generowania równań boolowskich dla podprogramów języka VHDL. W pierwszej części artykułu zostały przedstawione problemy pojawiające się podczas generowania równań boolowskich ze źródeł napisanych w języku VHDL. W części drugiej zaprezentowano metodę umożliwiającą generowanie równań boolowskich dla procedur oraz funkcji. W części trzeciej dokonano porównania działania kompilatora VHDL2Bool z innymi istniejącymi narzędziami.

Słowa kluczowe: język VHDL, podprogramy, równania boolowskie.

Boolean equations generation method for subprograms in VHDL language

Abstract

A method of boolean equation generation for subprograms of the VHDL language is presented int the paper. The first part of the paper presents subprograms in VHDL language: procedure and function. This part also presents problems of the boolean equation generation for procedure and function with sources written in the VHDL language. The second part presents the main method. This method consists of two phases and 11 steps. Steps 1 to 10 prepare source code for translation. The main goal of the first 10 steps is to change all variables and signals names: step 1 – order subprograms parameters, step 2 – find all subprograms names, step 3 – check formal and actual subprogram parameters, step 4 - order actual parameters, step 5 – create new return variable, step 6 – compute all variables length, step 7 – prepare subprogram source code, step 8 – compute arithmetic expressions, step 9 – prepare local variables names, step 10 – prepare subprogram source code for boolean equations generation. Step 11 translates source code for boolean equations. There are 15 algorithms described in all steps. Each step is illustrated by an example. The method use lexical, semantic and syntactic analyser results. Steps 5,6,7,9,10 and 11 are novelty. As an example of practical application of the method some results of the boolean equations generation are shown in the third part. In the third part the comparison of the method with existing industrial compilers there is presented.

Keywords: VHDL language, subprograms, boolean equations.

1. Wstęp

Język VHDL służy do opisu struktury i funkcjonowania różnych systemów cyfrowych. Urządzenia takie jak telewizja cyfrowa, telefony komórkowe nie mogłyby powstawać bez zaawansowanych układów cyfrowych, z których są zbudowane. Język VHDL spełnia podobną funkcję w dziedzinie projektowania sprzętu, jak język C++ w dziedzinie języków programowania. Układ cyfrowy zaprojektowany w języku VHDL doskonale nadaje się do symulacji oraz do syntezy. Na Wydziale Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego w Szczecinie realizowany jest kompilator języka VHDL generujący na podstawie źródła VHDL zbiór równań boolowskich. Równania boolowskie są doskonałym materiałem wyjściowym do dalszej pracy, ponieważ jest to forma matematyczna. Na ich podstawie można stworzyć układy cyfrowe realizujące określone zadania, lub poddać je minimalizacji. Równania boolowskie można odwzorować na układy FPGA[1].

Rozwiązując bardziej złożony problem wyodrębnia się zwykle pewne jego części, dla których formułuje się oddzielnie rozwiązania. Wyodrębnione części programu lub układu cyfrowego to podprogramy. W języku VHDL możemy korzystać z dwóch rodzajów podprogramów: procedur oraz funkcji.

Procedurą nazywamy algorytm z przyporządkowanym mu identyfikatorem, za pośrednictwem którego można się do tego algorytmu odwołać i spowodować jego wykonanie dla określonych argumentów [2]. Algorytm ten jest na ogół zapisany w postaci sparametryzowanej, tzn. przy użyciu pomocniczych nazw, zwanych parametrami formalnymi. Bezpośrednio przed rozpoczęciem przykładu procedury parametry formalne są zastępowane parametrami aktualnymi.

Funkcje podobnie jak procedury można traktować jako sekwencje deklaracji i instrukcji, które mogą być wielokrotnie wywoływane z różnych miejsc programu [3]. Wywołanie funkcji jest wyrażeniem, dlatego po zakończeniu obliczeń funkcja zwraca pojedynczą wartość, która może być typu złożonego [4]. Definicja funkcji w języku VHDL składa się z dwóch części [5]:

- deklaracji funkcji, która zawiera nazwę funkcji, listę parametrów formalnych oraz typ wartości zwracany przez funkcję,
- ciała funkcji, które może zawierać deklaracje zmiennych lokalnych oraz instrukcje które są wykonywane sekwencyjnie.

Niniejszy artykuł przedstawia metodę generowania równań boolowskich, ze źródła napisanego w języku VHDL, dla procedur oraz funkcji.

2. Metoda generowania równań boolowskich

Metoda generowania równań boolowskich dla podprogramów jest ściśle uzależniona od specyfikacji przyjętej przy realizacji kompilatora VHDL2BOOL. Głównym czynnikiem mającym zasadniczy wpływ na kształt opisanej metody są dwa założenia specyfikacji. Po pierwsze brak instrukcji skoku w wynikowych równaniach. Po drugie w wygenerowanych równaniach nie może występować wielokrotne przypisane do tej samej zmiennej.

Metoda generowania równań boolowskich dla podprogramów została podzielona na kilka etapów:

1) Uporządkowanie argumentów przekazywanych do podprogramu
2) Znalezienie podprogramu o szukanej nazwie
3) Sprawdzenie zgodności parametrów formalnych z aktualnymi oraz zwracanego typu (dla funkcji)
4) Uporządkowanie parametrów aktualnych
5) Tworzenie zmiennej do której zostanie przypisany zwracany wynik (tylko dla funkcji)
6) Obliczenie długości zadeklarowanych zmiennych
7) Przygotowanie źródła ciała funkcji
8) Obliczenie wartości wyrażeń arytmetycznych
9) Przygotowanie zmiennych lokalnych
10) Końcowe przygotowanie ciała podprogramu do przekładu na równania boolowskie
11) Wygenerowanie równań boolowskich

Poniżej znajduje się szczegółowe omówienie każdego z tych etapów.

Etap 1: Uporządkowanie argumentów przekazywanych do podprogramu

W języku VHDL tak jak w większości innych języków programowania argumentami przykazywanymi do funkcji podczas jej wywołania mogą być wyrażenia złożone np.

Przykład 1:

```
P4(x, y, a+b);
```

Przed przystąpieniem do wyszukiwania odpowiedniej funkcji należy sprawdzić poprawność takiego wyrażenia oraz typ zmiennej będącej wynikiem operacji i zastąpić je jedną utworzoną w tym celu zmienną tymczasową. Na tym etapie nie są jeszcze generowane żadne równania boolowskie.

Etap 2: Znalezienie podprogramów o szukanej nazwie

W języku VHDL funkcje mogą być przeciążone, dlatego w drugim etapie który należy wykonać po napotkaniu wywołania funkcji jest znalezienie wszystkich podprogramów o zadanej nazwie. Są one zapamiętywane i w kolejnym etapie następuje wybór tylko jednego pasującego podprogramu.

Przykład 2:

```
function Func_4 (constant A,B: in integer) return integer is
function Func_4 (constant A,B: in bit_vector(0 to 7)) return bit_vector is

x<=func_4(a,b);
y<=func_4(aa,bb);
```

Etap 3: Sprawdzenie zgodności parametrów formalnych z aktualnymi oraz zwracanego typu (dla funkcji)

Jeśli znajdziemy podprogram o szukanej nazwie, to musimy następnie sprawdzić, czy parametry aktualne podane przy jego wywołaniu są zgodne z parametrami formalnymi. Zgodność parametrów musi być sprawdzana pod względem typu danych, kierunku przepływu danych oraz ich rozmiaru. W przypadku programu z przykładu 2 dwie funkcje posiadają takie same nazwy w pierwszym wywołaniu zostanie dopasowana pierwsza zdefiniowana funkcja, natomiast w drugim wywołaniu druga funkcja.

Przykład 3:

```
function Func_4 (constant A,B: in integer) return integer is
function Func_4 (constant A,B: in bit_vector(0 to 7)) return bit_vector is

z_d<=func_4(a,b&c);
z_db<=func_4(ab,bb);
```

Etap 4: Uporządkowanie parametrów aktualnych

W języku VHDL podczas wywołania funkcji parametry aktualne mogą być podane przez pozycje (w takiej kolejności w jakiej występują parametry formalne) lub przez nazwę.

Przykład 4:

```
Z<=func_4(x,y);
Z<=func_4(b =>y,a=>x);
```

Parametr aktualny przekazywany do funkcji może być również elementem tablicy. Wtedy będzie się składał z kilku leksemów:

Przykład 5:

```
Z<=func_4(tab1[5] , tab1[10]);
```

W takim przypadku należy wygenerować równania, które umożliwią zastąpienie kilku leksemów jednym.

Przykład 6:

```
Tmp1<= tab1[5];
Tm21<= tab1[10];
Z<=func_4(Tmp1 ,Tmp2 );
```

Na tym etapie parametry aktualne są sortowane w kolejności występowania parametrów formalnych.

Etap 5: Tworzenie zmiennej do której zostanie przypisany zwracany wynik (tylko dla funkcji)

Etap 5 dotyczy tylko funkcji, a konkretnie argumentu zwracanego przez funkcję. Dla zwracanego argumentu tworzona jest zmienna tymczasowa, która będzie wykorzystywana podczas procesu generowania równań boolowskich dla ciała funkcji.

Przykład 7:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity test is
   port ( clk : in STD_LOGIC;    a,b : in integer range -100 to 1000;
                                 z_d: out integer range -100 to 1000
   );
end test;

architecture arch of test is
function Func_4 (constant A,B: in integer) return integer is
  variable counter, max: integer;
  begin
    counter:= b;
    if a=100 then
            return b;
    else
            return counter+3;
    end if;
  end Func_4;
Begin
z_d<=func_4(a,b);
end arch;
```

Funkcja z przykładu 7 zostanie zmodyfikowana i przyjmie postać jak w przykładzie 8:

Przykład 8:

```
function Func_4 (constant A,B: in integer) return integer is
  variable counter, max: integer;
  begin
    counter:= b;
    if a=100 then
                          tmpRet <= b;
                           return tmpRet;
            else
                          tmpRet <= counter + 3;
return tmpRet;
            end if;
  end Func_4;
```

Etap 6: Obliczenie długości zadeklarowanych zmiennych

W programie napisanym w języku VHDL długość zmiennych zadeklarowanych wewnątrz podprogramu może być uzależniona od wartości konkretnego argumentu przekazywanego podczas wywoływania podprogramu.

Przykład 9:

```
 function Func_4 (a,b: in bit_vector) return bit_vector is
  variable c : bit_vector(a'range);
```

Dla przykładu 9 długości poszczególnych zmiennych muszą zostać dokładnie obliczone.

Etap 7: Przygotowanie źródła ciała funkcji

Na tym etapie analizowane są wszystkie leksemy wchodzące w skład definicji ciała podprogramu. Jeżeli jakiś leksem jest parametrem formalnym to jego wartość jest zastępowana przez wartość parametru aktualnego lub wartość leksemu tymczasowego utworzonego w celu zastąpienia parametru aktualnego. Jeśli kompilowanym podprogramem jest funkcja to również wartość zwracana przez funkcję jest zastępowana przez zmienną tymczasową. Etap ten jest jednym z etapów końcowych przygotowania podprogramu do kompilacji.

Przykład 10:

```
c <= a or b;
return c;
```

Program z przykładu 10 po modyfikacji przyjmie postać jak na przykładzie 10

Przykład 11:

```
c <= a or b;
zmienna_tymczasowa <= c;
return zmienna_tymczasowa;
```

Etap 8: Obliczenie wartości wyrażeń arytmetycznych

W etapie 8 obliczane są wszystkie możliwe do wyliczenia wartości. Etap ten najlepiej wyjaśni konkretny przykład. Jeśli w podprogramie pojawia się następująca deklaracja stałej (przykład 12) to możemy wyliczyć jaką wartość będzie miała stała p. Właśnie taki proces jest wykonywany w etapie 8.

Przykład 12:

```
constant p : integer := 1 + 10;
```

Etap 9: Przygotowanie zmiennych lokalnych

Specyfikacja przyjęta podczas tworzenia kompilatora VHDL2BOOL zakłada, że nazwy zmiennych do których występuje przypisanie równań nie mogą się powtarzać. Ponieważ w programie napisanym w języku VHDL możemy wielokrotnie wywoływać te same podprogramy, a w ciele funkcji może wystąpić przypisanie wyrażenia do zmiennej lokalnej, to przy każdym wywołaniu podprogramu musimy zastąpić oryginalne deklaracje nazw zmiennych lokalnych zmiennymi tymczasowymi.

Przykład 13:

```
procedure Func_4 (A,B: in integer; signal c: out integer) is
  variable z:integer ;
  begin
            …
end procedure Func_4;
```

Przykład 13 po modyfikacji przyjmie postać jak przykład 14.

Przykład 14:

```
procedure Func_4 (A,B: in integer; signal c: out integer) is
  variable tmp1_z:integer ;
  begin
            …
end procedure Func_4;
```

Etap 10: Końcowe przygotowanie ciała podprogramu do przekładu na równania boolowskie

W kroku 10 każda zmienna lokalna występująca w podprogramie zostaje zastąpiona unikalną zmienną tymczasową. Dzięki temu zostanie spełniony jeden z warunków specyfikacji mówiący o tym, że w równaniach nie może występować wielokrotne przypisanie do tej samej zmiennej. W etapie 10 powstaje ostateczny kształt definicji ciała funkcji.

Przykład 15:

```
procedure Func_4 (A,B: in integer; signal c: out integer) is
  variable z:integer ;
  begin
            z:=a+5;
            c<=(a + z);
end procedure Func_4;
```

Procedura z przykładu 15 po modyfikacji przyjmie postać pokazaną na przykładzie 16:

Przykład 16:

```
procedure Func_4 (A,B: in integer; signal c: out integer) is
  variable tmp1_z:integer ;
  begin
            tmp1_z:=a+5;
            c<=(a + tmp1_z);
end procedure Func_4;
```

Etap 11: Wygenerowanie równań boolowskich

Po wykonaniu wszystkich 10 etapów niniejszej metody. Podprogram jest gotowy do wygenerowania na jego podstawie odpowiednich równań boolowskich. Zarówno nazwy zmiennych będących argumentami formalnymi jak i zmiennych oraz stałych zadeklarowanych wewnątrz podprogramu posiadają unikalne nazwy. W przypadku funkcji nazwa zmiennej do której zostaje przypisany wynik zwracany przez nią jest również nazwą unikalną. W 11 etapie następuje ostateczny proces generowania równań boolowskich. Bazuje on na dziesiątkach algorytmów wykorzystywanych w całym kompilatorze VDHL2BOOL i opracowanych przez różne osoby biorące udział w realizacji całego kompilatora. W pierwszym etapie podprogram jest analizowany pod względem występowania pętli. Jeśli one występują to podprogram jest konwertowany do postaci w której pętle nie występują. Dopiero po tym etapie następuje właściwy proces generowania równań boolowskich.

3. Wnioski

Przedstawiona metoda generowania równań boolowskich dla podprogramów języka VHDL została przetestowana na kilkuset prostych przykładach i po weryfikacji stwierdzono, że działa prawidłowo. Jej działanie przetestowano również na przykładzie dwóch profesjonalnych układów realizujących szyfrowanie metodą DES. Przetestowane zostały dwa rodzaje układów szyfrujących: układ zoptymalizowany pod względem wielkości oraz pod względem szybkości szyfrowania. Podczas testowania zwrócono szczególną uwagę na czas kompilacji oraz zużycie pamięci operacyjnej komputera. Wyniki kompilacji porównano z komercyjnym pakietem Xilinx. Na podstawie uzyskanych wyników można stwierdzić, że zastosowane w kompilatorze VHDL2BOOL metody i algorytmy generowania równań boolowskich dość dobrze sprawdzają się podczas syntezy sprzętu dla profesjonalnych układów stworzonych przy pomocy języka opisu sprzętu jakim jest język VHDL. W przypadku układu szyfrującego algorytmem DES zoptymalizowanego pod względem prędkości szyfrowania czas potrzebny na syntezę w przypadku kompilatora VHDL2BOOL jest około 25% mniejszy niż pakietu Xilinx. Dla układu zoptymalizowanego pod względem wielkości czas potrzebny na syntezę w przypadku obu kompilatorów jest jednakowy. W przypadku wykorzystania pamięci operacyjnej komputera na którym odbywa się synteza przewaga kompilatora VHDL2BOOL nad Xilinx jest jeszcze większa. Zapotrzebowanie na pamięć dla VHDL2BOOL jest prawie o 50% mniejsze niż dla pakietu Xilinx.

4. Literatura

[1] J. Sołdek: Miejsce układów reprogramowalnych w informatyce, Materiały I Krajowej Konferencji Naukowej. Reprogramowalne układy cyfrowe.
[2] W. Wrona: VHDL – język opisu i projektowania układów cyfrowych, Pracowania Komputerowa Jacka Skalmierskiego, 2000.
[3] K. Kołek: Język opisu sprzętu VHDL, Akademia Górniczo-Hutnicza, Kraków, 1999.
[4] VHDL'93 IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993.
[5] S. Yalamanchili: VHDL Starters Guide, Prentice Hall, Inc., 1998.

Artykuł recenzowany