VHDL ОПИСАНИЯ НА РЕГИСТРОВО И ПОВЕДЕНЧЕСКО НИВО ПРИ РАЗРАБОТКА НА ПРОЦЕСОРНИ МОДУЛИ СЪС SYNOPSYS

инж. Димитър А. Кавалов, Технически университет - София, ECAD - Лаборатория

RTL and Behavioral Descriptions in Synthesising Processor Modules Using SYNOPSYS' VHDL

Dimitar A. Kavalov M.Sc., Technical University - Sofia, ECAD - Laboratory

Abstract

This article concerns the possibility of using Registry Transfer Level (RTL) methodology for synthesis and analysis of small and middle scale digital designs as arithmetic-logic units, registers and counters as well as small processors and interface modules.
A comparison analysis between behavioral and register transfer level methodologies for VHDL descriptions is carried out. Conclusions for the grade of the utilization of each methodology are made.
VHDL possibilities when using registry transfer level methodology in synthesising already developed designs, described in languages based on direct registry transfer of information (data), is discussed in the paper.
Special attention is paid to the behavioral and RTL methodologies for synthesis and analysis and the conclusions of their practical application.
A particular example of designing an arithmetic-logic unit using both methodologies is presented.

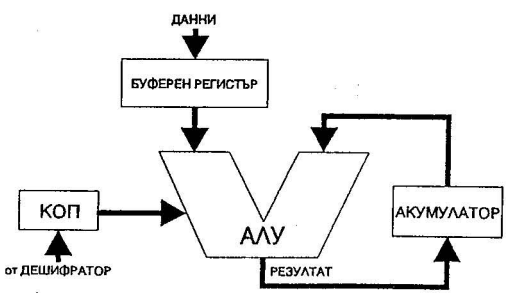
В настоящата обзорна статия се разглеждат проблемите за използването на *регистровия* и *поведенчески* стил на описание на VHDL.
В литературата се разглежда често проблема за възможностите на различни езици за схемно описание, какъвто е и VHDL за синтезиране на схеми на различно ниво.
При класификацията на схемните описания с VHDL могат да се разграничат следните четири стила (нива) на описание:

Поведенческо ниво
Регистрово ниво
Логическо ниво (ниво гейт)
Топологично ниво

Фиг. 1

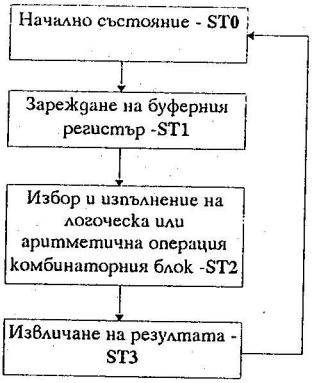
Тези нива на описание са известни още като абстракции на представянето. По-конкретно, най-ниското ниво на абстракция, топологичното, дава информация и описва схемното разположение върху силициевата подложка, както и може да описва подробно времеви съотношения или специфични аналогови ефекти. По-горното логическо ниво описва връзките между включените логически гейтове и регистри. Описанието съдържа информация за функцията, архитектурата, технологията и времевите съотношения. На регистрово ниво описанието е точно определено, като се дефинира всеки регистър в проекта със съответните му логически връзки. Описанието съдържа информация за архитектурата, но не и за технологията. Абсолютните времеви закъснения не се определят. Най-високото, поведенческо, ниво се

използва за описание на функцията на проекта, без да се навлиза детайлно в архитектурата на регистровите връзки. Поведенческото описание може да съдържа в себе си времеви съотношения до степен, която изисква представянето на дадената функция.
На практика за проектирането на цифрови схеми със съвременни средства се използват *регистровото* и *поведенческото* ниво на описание. Повечето от предлаганите днес средства за синтез на VHDL описания изискват кода да бъде написан на *регистрово* ниво или на смесено *регистрово-поведенческо* ниво. При *регистровото* ниво проектантът запазва контрол върху архитектурата на регистрите и състоянията в своя проект. Поведенческите средства за синтез автоматично генерират архитектурата на регистрите от описанието на VHDL. Само някои от днешните програми за синтез могат да използват чисто поведенческо VHDL описание и то не за директно описание на схеми, а за прилагане на определени алгоритми (DSP алгоритми), както и за синтезиране на тестови установки.
Всички литературни източници са единни в това, че *поведенческото* ниво на описание е много по-абстрактно и съдържа много по-малко подробности за производството на дадена схема, докато, *регистровото* ниво на описание е по-малко абстрактно и съдържа повече детайли за производството на схемата. Проблемът с описание на регистрово ниво може да се разгледа и в аспекта на транслирането на описания на проекти от езици за проектиране, в основата (CDL), на които е заложено регистровото предаване на сигнали към VHDL описания.
Една от очевидните разлики между двата стила на описание се състои в следното: ако целият модел, които се изследва, се управлява от явен тактов сигнал и съдържа регистър за текущото състоянието, то това е *регистров* модел.
Като пример може да се разгледа модел на 4-битово Аритметико-Логическо Устройство (АЛУ) като съставна част от процесорен модул. Блоковата схема на АЛУ -то е представена на фиг. 2. Това е примерен опростен модел, затова не се разглеждат входове и изходи за пренос.



Фиг.2 Блокова схема на Аритметико-логически модул (АЛУ)

Въз основа на тази блокова схема може да се направи и алгоритъм за работата на устройството като се отчитат състоянията, през които преминава схемата по време на работата си. Фактически това е описание на машина на състоянията, които се определят от регистрите (*буферен регистър* и *акумулатор*), свързани с комбинаторната логика на самия аритметико-логически блок. На фиг. 3 е представен опростен вид на алгоритъма с преходите между състоянията. Състоянията са отбелязани със **ST0...ST3**.



Фиг.3

Аритметико-логическият блок съдържа комбинационна логика, която в конкретния пример е следната:

Код на операцията (КОП)	Операция
'000'	Зареждане на Акум.
'001'	Събиране (Акум.+Данни)
'010'	Изваждане (Акум.- Данни)
'011'	Акум. + 1
'100'	Нулева оперция
'101'	Изчистване на Акум.
'110'	Акум. AND Данни
'111'	NOT Акум.

Моделите в примера са идеализирани, така че не се отчитат никакви закъснения за извършваните операции в Аритметико-логическия блок и в регистрите.

```
library Synopsys;
  use Synopsys.bv_arithmetic.all;
entity ALU is
    Port (Clk, Rst : In   bit;
          DATA : In   bit_vector (3 downto 0);
          OPCODE : In   bit_vector (2 downto 0);
          RESULT : Inout  bit_vector (3 downto 0));
end ALU;
architecture rtl_process of ALU is
 type state_val is (st0,st1,st2,st3);
 signal S, S_in:  state_val;
 signal Din, Dout, Acc: bit_vector (3 downto 0);
 begin
  Comb: process
 begin
   wait until Clk='1';
        if  Rst = '1' then  s <= st0;
        else  case s is
           when st0 => Din <= DATA;
                  s <= st1;
            when st1 => Case OPCODE  is
                        when "000" => Dout <= Din;
                        when "001" => Dout <= Acc + Din;
                        when "010" => Dout <= Acc + Din;
                        when "011" => Dout <= Acc + "0001";
                        when "100" => Null;
                        when "101" => Dout <= "0000";
                        when "110" => Dout <= Acc and Din;
                        when "111" => Dout <= not Acc;
                      end case;
                  s <= st2;
          when st2 => Acc <= Dout;
                  s <= st3;
          when st3 => s <= st0;
         end case;
       end if;
    end process Comb;
  RESULT <= Acc;
end rtl_process;
```

Фиг. 4

Средата за проектиране, в която са описани и синтезирани различните модели е една от най-мощните съвременни системи за автоматизирано проектиране - SYNOPSYS.

При проектирането на регистрово ниво, проектантът е този, който решава колко логически състояния да има и колко да бъдат операциите между тях. При синтезирането, VHDL сигналите се преобразуват в хардуерни регистри, комбинационна логика и се оптимизират. Не се генерират нови състояния, които да се комбинират и оптимизират, както по отношение на ограничения по площ, така и по време.

Представеното на фиг.4 VHDL описание е фактически код за синтезиране на модел на машина на състоянията с регистри за данни и комбинационна логика. Преминаването между отделните състояния е обединено в един процес. Сигналът **S** представлява регистъра на състоянията, а сигналите **Din**, **Dout** и **Acc** са регистрите за данни. Процесът е установен

да превключва по нарастващ фронт на тактовия сигнал.
Изборът на това колко регистъра и логически операции да се използват, а също така и разпределението времевите съотношения, зависи от конкретната задача.
От проведената симулация се вижда, че моделът се държи сравнително добре, сменя състоянието си на всеки такт така, че всяка операция на комбинационната логика се извършва на всеки четвърти такт или респективно на всеки четири такта се завършва един цикъл на работа на устройството. Това забавя до известна степен работата на схемата и я прави зависима от тактовия сигнал и времето на активните данни, което е характерно за устройствата съдържащи памет.

```
library Synopsys;
  use Synopsys.bv_arithmetic.all;
entity ALU is
    Port (Clk, Rst : In   bit;
          DATA : In   bit_vector (3 downto 0);
        OPCODE : In   bit_vector (2 downto 0);
           RESULT : Inout  bit_vector (3 downto 0));
end ALU;
architecture rtl_process_s of ALU is
 type state_val is (st0,st1,st2,st3);
 signal s, s_in:  state_val;
 signal Din, Dout, Acc: bit_vector (3 downto 0);
 begin
   reg: process
     begin
      wait until Clk='1';
        if Rst = '1' then    s <= st0;
        else                 s <= s_in;
      end if;
        Din <= DATA;
        Acc <= Dout ;
  end process reg;
Comb:process (Din, Dout, s)
 begin
          case s is
           when st0 => s_in <= st1;
           when st1 => Case OPCODE is
                 when "000" => Dout <= Din;
                 when "001" => Dout <= Acc + Din;
                 when "010" => Dout <= Acc + Din;
                 when "011" => Dout <= Acc + "0001";
                 when "100" => Null;
                 when "101" => Dout <= "0000";
                 when "110" => Dout <= Acc nand Din;
                 when "111" => Dout <= not Acc;
                 end case;
                 s_in <= st2;
                 when st2 => s_in<= st3;
                 when st3 => s_in <= st0;
       end case;
     end process Comb;
  RESULT <= Acc;
end rtl_process_s;
```

Фиг. 5

Един друг подход за прилагане на RTL методологията е представен на фиг. 5. При този подход може да се забележи, че регистрите и комбинационната логика са представени с различни процеси. В процесът „**reg**“ се „изработват“ буферният регистър и акумулаторът, докато в процеса „**Comb**“ се съдържат преходите между състоянията и комбинационната логика. Този тип модели могат да се нарекат регистрово ориентирани. Лесно се вижда каква операция се извършва над даден регистър, а по-трудно -- операциите над устройството като цяло, тъй като информацията е разпределена по целия модел. Вижда се също, че някои от състоянията са само преходни.Те могат да се изключат и така модела ще остане само с две състояния - **ST0** и **ST1**. Сигналът **S** свързва двата процеса. Може да се очаква, че симулацията на така построения модел ще е по тежка поради повечето сигнали, които са включени.
Предимство на разгледания модел е, че проектантът има достъп до данните за всеки отделен регистър, но и той страда от недостатъците на предишния разгледан модел по отношение на времевите съотношения.
Съществуват и други методи за описание на чисто регистрово ниво, но те не са обект на тази статия.

Поведенческото ниво на описание е интересно от гледна точка на това, че се описва алгоритъм на поведение в чист вид, без проектантът да се интересува от регистри, тактови сигнали и други схемотехнически ограничения. Едно поведенческо описание на разглежданата схема е представено на фиг. 6:

```
library Synopsys;
  use Synopsys.bv_arithmetic.all;
entity ALU is
    Port (    ACC : In   bit_vector (3 downto 0);
              DATA : In   bit_vector (3 downto 0);
            OPCODE : In   bit_vector (2 downto 0);
            RESULT : Out  bit_vector (3 downto 0) );
end ALU;
architecture BEHAVIORAL of ALU is
  begin
  Main:Process(Acc,Opcode,Data)
        begin
          Case Opcode is
                when "000" => Result <= DATA;
                when "001" => Result <= Acc + data ;
                when "010" => Result <= Acc + data;
                when "011" => Result <= Acc + "0001";
                when "100" => Null;
                when "101" => Result <= "0000";
                when "110" => Result <= acc nand data;
                when "111" => Result <= not acc;
          end case;
        end process main;
end BEHAVIORAL;
-----
library Synopsys;
  use Synopsys.bv_arithmetic.all;
entity ACCUMULATOR is
    Port (    CLK : In   bit;
          RESULT : In   bit_vector (3 downto 0);
             ACC : Out  bit_vector (3 downto 0) );
end ACCUMULATOR;
architecture BEHAVIORAL of ACCUMULATOR is
  begin
  Main: process (clk)
        begin
          if clk = '1' then
                Acc <= result;
          end if;
  end process Main;
end BEHAVIORAL;
```

Фиг.6

На практика това не е чист поведенчески модел защото все още той се управлява от явен тактов сигнал (Clk). Това, което липсва от предишните по-горе описаните модели, е явният регистъра на състоянието. „Състоянието" се определя от естествената последователност на процеса, която по подразбиране определя текущото състояние на модела. В основата на това описание е комбинационната схема на АЛУ -то, която се среща и във всички описания по-горе. Освен него има и модел на акумулаторния регистър, който фактически е описан на RTL ниво. Симулацията на този модел е в известна степен по-добра отколкото на предишните два, най-вече поради липсата на явни регистри на състоянието. По този начин на всеки нарастващ фронт на тактовия сигнал се включва комбинационната схема (извършва се логическа или аритметична операция) и резултатът се съхранява в акумулатора.

Един идеален поведенчески описан модел не би трябвало да съдържа в себе си тактове, машини на състоянието и регистри. Не е нужно сигналите и променливите да бъдат отнесени към регистри или комбинационна логика. Моделът просто изчислява резултата веднага щом има някакви данни на входа му. Не е необходимо проектантът да се интересува от логическите структури и времевите модели. Това значително улеснява работата от една страна, но от друга я прави по-трудна, защото моделите стават значително по-абстрактни.

В заключение може да се каже че и двата стила на описание имат своите положителни страни и недостатъци.

Регистровото ниво на описание е по-трудоемко, но за това по-ясно отразява модели на устройства, които лесно се трансформират в машина на състоянието. При този тип описание проблемите с избор на

регистри, сигнали , състояния се оставят в ръцете на проектанта и това, което той е решил се отразява пряко върху синтезираната схема. Почти всички инструменти за синтез поддържат различните разновидности на RTL описанията.

Поведенческото описание от своя страна е много удобно за алгоритми на по-сложни модели на завършени системи, въпреки по-голямата степен на абстрактност, с която е свързано. Средствата за синтез, които поддържат поведенчески модели могат да създават и комбинират логически състояния и логически операции за преминаване между тях, да определят времеви ограничения при превключване на състоянията. Това прави поведенческото описание по-лесно за използване при синтез. SYNOPSYS, като една от водещите фирми в бранша, притежава много мощен пакет за синтез на поведенчески описани модели. С помощта на тези средства са анализирани и синтезирани гореизложените примери. За съжаление, чисто поведенчески описан модел не може да се синтезира във физически реализуема схема.

По отношение на използването на RTL за описание на схеми, описани преди това на регистрови езици като CDL, това е напълно възможно, но е по добре такива системи да се опишат и синтезират на поведенческо ниво. Така се спестяват значителни усилия на проектанта при транслацията.

Литература.

1. Saunders Lary, Aspects of RTL ana Behavioral Modeling - Part 1, 2 http://www.vhdl.org, 1998.
2. Holmes C., Willoughb M., VHDL Language Course, Rutherford Appleton Laboratory, 1997.
3. FPGA Express, VHDL Reference Manual, SYNOPSYS, Inc., 1997.
4. Behavioral Compiler Methodology Guide, SYNOPSYS, Inc., 1997.
5. Армстронг, Дж., Моделирование цифровых систем на языке VHDL, „МИР“, Москва, 1992.