A VHDL kódtól az FPGA-ba való ágyazásig

From the VHDL Code to the Implementation to FPGA-s

KIREI Botond Sándor
Kolozsvár

Abstract

The purpose of the VHDL hardvare describing language is to descibe the behaviour of digital circuits. This technology is suitable for designing, simulating and implementing digital circuits and systems. In order to make the design of the digital circuit successful, we have to be familiar with the design flow, which drives us to the digital circuit's netlist. This paper is briefly presenting the possibilities and the syntax of VHDL, and the design flow. In spite of the wide possibilities of the VHDL any VHDL code cannot be implemented to FPGAs, the limits are contoured by the synthesizing software, the size of the FPGA, or the required speed.

Bevezetés

A VHDL hardver leíró nyelv célja, hogy leírja a digitális áramkörök viselkedését. Ez a technológia alkalmas áramkörök és rendszerek tervezésére, szimulálására, illetve fizikai kivitelezésére. Viszont, hogy az áramkörök tervezése sikeres legyen, nagyon fontos, hogy ismerjük azokat a folyamatokat, melyek során a VHDL kódtól elindulva eljutunk az áramkör kapcsolási rajzához. A dolgozat röviden bemutatja a VHDL nyelv szintaxisát, illetve lehetősegeit, és a tervezés folyamatát. A VHDL nyelv szeleskörű lehetőségei ellenére nem minden VHDL kód ágyazható FPGA-ba. Határt szab az áramkör szintetizálását végző szoftver, az FPGA mérete vagy akár az általunk kívánt sebesség.

A VHDL mint leíró nyelv

Mindennapi életünkben számos elektronikai eszköz vesz körül minket, amelyek nem születhettek volna meg a VHDL nyelv nélkül.

Már a '70-és években megjelent az igény a digitális eszközök viselkedésének egyszerű írására, szimulációjára és szintézisére. A '80-as években megjelentek a VLSI (Very Large Scale Integration) integrált áramkörök. Az integrált áramkörökbe egyre több logikai kapu, regiszter vagy ROM fért el, ezért az áramkör által elvégzett feladatok bonyolultabbá váltak. A klasszikus tervezési eljárások elavultak (egy egyszerű szorzó áramkör sematikus úton való megtervezése több mint 30 oldalt vesz igénybe, míg egy szorzó VHDL kódja 6 sor), ezért szükség volt egy alternatív lehetőségre, amely leegyszerűsíti és meggyorsítja a tervezést. Ugyanakkor, megnövekedtet az igény a magasabb szintű tervezésre. A fejlesztők a digitális eszközök viselkedését leíró nyelvben látták ezt az alternatívát.

A VHDL 1981-ben jelent meg, de csak 1987-ben vált nemzetközi szabvánnyá. 1993-ban megjelent a VHDL bővített szabványa, amit 1999-ben és 2001-ben átjavítottak. Mint a nyelv neve is mutatja (VHSIC Hardware Description Language) alkalmas különböző digitális áramkörök, rendszerek és eszközök viselkedésének a leírására.

Szeretném kihangsúlyozni, hogy a VHDL nyelv nem egy klasszikus értelemben vett programozási nyelv, mint a Pascal vagy a Java, hanem egy eszkö,z amivel digitális rendszereket írunk le. Ezt a nyelvet elsősorban áramkörök szimulálására és szintézisére használjuk, nem pedig bonyolult algoritmusok megvalósítására (annak ellenére, hogy a nyelv szintaxisa alkalmas volna erre a feladatra is).

A VHDL nyelvnek voltak előfutárai, ilyen például az ABEL. Ugyanakkor „testvérei" is jelen vannak az ipari tervezésben, pl. a VERILOG leírási nyelv. A VHDL egyszerű szintaxisa, könnyű megérthetősége és kiterjedt lehetőségei miatt didaktikai szempontból nagyon jól bevált.

Röviden a VHDL szintaxisáról

Mivel a VHDL áramkörök vagy rendszerek leírására készült, többszintű absztraktizálást engedélyez. Az absztraktizálás megengedi, hogy egy adott szintű tervezés során, az adott szinten nem lényeges részletek elrejthetővé váljanak. Például így, egy digitális rendszer megtervezhetővé, leellenőrizhetővé, szimulálhatóvá válik anélkül, hogy az adott tervezőcsoport az idejét a rendszer megvalósításával töltené. Egy másik tervezési szinten a rendszer szerkezetét lehet megtervezni, finomítani, majd egy más szinten a rendszer tömbjeinek a megvalósítására lehet koncentrálni.

Innen következik, hogy a VHDL nyelv alapjában véve két féle leírást engedélyez: *strukturált* leírást, amely tömbök vagy áramkörelemek összekapcsolását írja le, és *viselkedési* leírást, amely tömbök működését írja le. A kétféle leírás együtt létezhet egy VHDL kódban.

Amikor kívülről vizsgálunk meg egy áramkört, mi nem látunk mást, mint bemenő és kijövő adatokat. Ennek a leírására használjuk az entitást. Az entitásban határozzuk meg, hogy a mi áramkörünknek milyen I/Ó portjai vannak. Az entitás leírásának a szintaxisa a következő:

```
entity Entitás_neve is
   port (
   <port_deklaraciok>
   );
end Entitás_neve;
```

Ami az áramkörön belül van, azt az áramkör arhitektúrájának nevezzük. Az adott entitásokhoz egy vagy több arhitektúrát is hozzárendelhetünk. Az arhitektúra szintaxisa a következő:

```
architecture Arhitektúra_neve of Entitás_neve is
   <deklaraciok>
begin
   <utasitasok>
end Arhitektúra_neve;
```

Az arhitektúra leírása során két fő modellel találkozunk. Az első a *viselkedés modell*, ahol egy logikai függvénnyel írjuk le az áramkör viselkedését. Egy egyszerű példa:

```
architecture Viselkedés_modell of Entitás_neve is

begin
      E<= (A AND B) OR (C AND D);
end Viselkedés_modell;
```

A második a *gerjesztés modell* (process), aminek a használatával lehetőségünk nyílik az áramkör viselkedésének a szekvenciális leírására. A gerjesztés modell akkor kerül végrehajtásra, amikor egy jel megváltozik a szenzitivitási listában (process (<szenzitivitási lista>)). A parancsok egymás után hajtódnak végre, akárcsak egy klasszikus program futtatásakor.

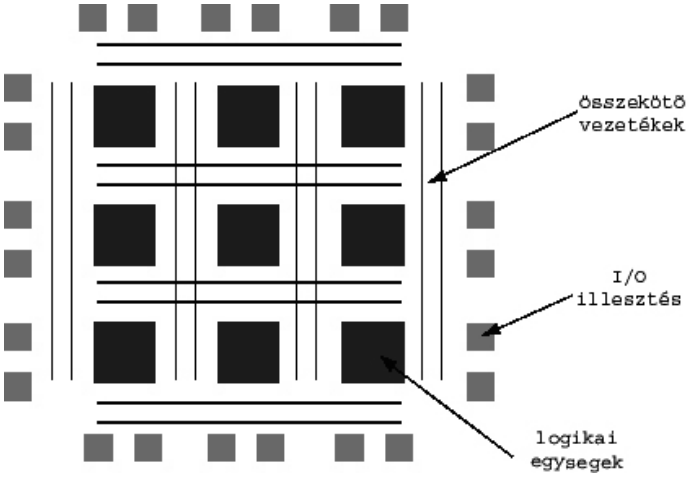
```
architecture gerjesztés_model of szamlaló is
begin
      process (clk)
      variable state:std_logic_vector(7 downto 0);
      begin
            if clk'event and clk='1' then
                  state:=state+'1';
                  szam<=state;
            end if;
      end process;
end gerjesztés_model;
```

Nagyon fontos tudni még a VHDL nyelvről, hogy az arhitektúrában leírt parancsok és gerjesztés modellek párhuzamos módon hajtódnak végre. Az FPGA-ba ágyazott algoritmusok igazi előnye, hogy az utasítások külön szálakon hajtódnak végre, ami igazán gyors adatfeldolgozást tesz lehetővé.

Az FPGA mint áramkör

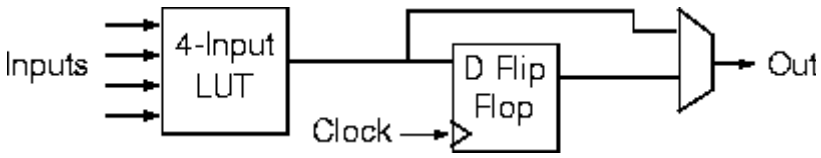
Az ipar fejlődése során egyre bonyolultabb rendszerekre volt szükség. Ezek a rendszerek egyre több logikai kaput, regisztert vagy ROM-ot tartalmaztak. Ekkor született meg Ron Cline[1] ötlete: adjunk a mérnökök keze alá egy olyan eszközt, amit kedvük szerint gyúrhatnak, kedvük szerint konfigurálhatnak. Carnough óta tudjuk, hogy bármilyen logikai függvény megvalósítható ÉS kapukból és VAGY kapukból. Ron Cline két programozható síkot képzelt el: az egyik síkban ÉS kapuk, a másik síkban VAGY kapuk vannak. A síkok összekapcsolásával szinte bármilyen ÉS-VAGY kombináció megvalósítható. Így született meg a SPLA, vagyis a Simple Programmable Logic Array. Ezek az áramkörök azonban csak kombinatorikus áramkörök megvalósítására alkalmasak.

Az FPGA arhitektúráját 1985-ben egy Freeman nevű úriember találta ki. A SPLA arhitektúrájától annyiban különbözik, hogy egyszerű logikai kapuk helyett logikai blokkokat (egységeket) használ. A 1. ábra az FPGA arhitektúrájának egy leegyszerűsített modellje. Az I/O illesztések kapcsolják össze az FPGA-ban található összekötő vezetekéket a külvilággal. Ezek az illesztések teszik lehetővé, hogy az FPGA chip-ek 10-15 gyártási technológiával kompatibilisak. Az összekötő vezetékek segítségével kapcsoljuk össze a logikai egységeket vagy blokkokat.



1. ábra
Az FPGA arhitektúrájának egyszerűsített modellje

Mint már említettük, egy logikai egység nem egy egyszerű logikai kapu, hanem sokkal bonyolultabb arhitektúra. Ezeket a logikai egységeket használjuk a különböző logikai függvények megvalósításához. A 2. ábrán láthatjuk egy logikai egység egyszerűsített arhitektúráját.



2. ábra
A logikai egység egyszerűsített modellje

A LUT vagyis Look Up Table működése hasonlít egy memória működéséhez, és segítségével megvalósítható bármilyen négyváltozós bináris függvény. A D bistabilt a szinkron áramkörök esetében latch-nek, a szekvenciális automaták esetében pedig tárolóelemként használjuk. Megemlítendő, hogy a szekvenciális automaták elkészítése elképzelhetetlen tárlóelemek nélkül. Ezért az FPGA nemcsak kombinatorikus áramkörök, hanem szekvenciális automaták elkészítésére is alkalmas.

[1] Ron Cline a Signetics mérnöke, amit később a Xilinx felvásárolt

A VHDL kód FPGA-ba való ágyazása

Az a tervezési folyamat, amely során a VHDL kódtól eljutunk egy áramköri rajzhoz, amelyet be lehet ágyazni az FPGA chipbe, három lépésből áll.

Leírás

Az első lépés során meghatározzuk, hogy a tervezett áramkör tulajdonképp milyen feladatot kell ellásson. Ennek a lépésnek a végeredménye egy táblázat, amelyben az áramköri kapcsolás van tárolva (milyen logikai kapuk, bistabilok vagy ROMok vannak az áramkörbe és ezek az alkotó elemek hogyan vannak összekötve).

Két dolgot tehetünk. Vagy megépítjük az áramkört logikai kapukból, regiszterekből és ROMokból, vagy leírjuk az áramkört VHDL segítségével. Amennyiben VHDL segítségével írjuk le az áramkört, akkor az áramköri kapcsolást egy szintézis során kapjuk meg. Ezt a szintézist elvégzik helyettünk az erre a célra kifejlesztett szoftverek.

Ellenőrzés

Ebben a lépésben elvégezzük a szimulációkat, és kijavítjuk az esetleges hibákat. A szimulátor is egy szoftver, amely segítségével megkereshetjük a szemantikai hibákat, így egyszerűbbé válik a kód (vagy kapcsolási rajz) kijavítása. Nagyon fontos tudni, hogy az a VHDL kód ami a szimulátoron kifogástalanul viselkedik, sokszor nem implementálható az FPGA-ba.

Implementáció, vagy a chipbe való ágyazás

Ebben a lépésben a leírás eredményeképpen kapott áramköri kapcsolást megvalósítjuk az FPGA-ban található logikai egységek segítségével. Ez egy bonyolult folyamat, mert nemcsak a logikai kapukat kell összekötni, hanem az I/O portokat is meg kell határozni, majd az I/O portokat hozzá kell rendelni az áramköri kapcsolás megfelelő be- és kimeneteleihez, és végül be kell tölteni az adott FPGA-ba. Ezek a lépések már nem platform függetlenek, például figyelembe kell venni az FPGA-ban található logikai blokkok számát. De nem kell megijedni, mert ezt is számítógép végzi el helyettünk.

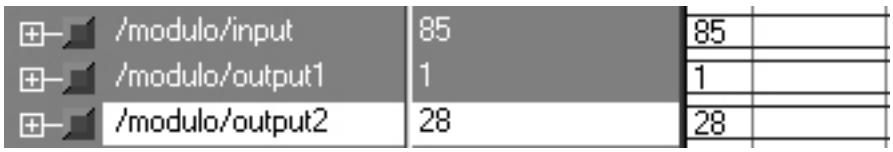
Minden VHDL kód FPGA-ba ágyazható?

A kérdésre a válasz: nem. Csak olyan VHDL kódod implementálhatunk, ami szintetizálható, vagyis biztosan létre lehet hozni olyan áramkört, ami elvégzi a VHDL kód által leírt feladatot.

Egy egyszerű példa:

```
entity moduló is
    Port ( input : in std_logic_vector(7 downto 0);
           output1, output2 : out std_logic_vector(7 downto 0));
end modulo;
architecture Behavioral of moduló is
begin
process (input)
variable szam1,szam2:integer range 0 tó 255;
begin
    szam1:=conv_integer(input);
    szam2:=szam1;
    szam1:=szam1 mod 3;
    szam2:=szam2/3;
    output1<=conv_std_logic_vector(szam1,8);
    output2<=conv_std_logic_vector(szam2,8);
end process;
end Behavioral;
```

A VHDL nyelv megengedi a moduló és az osztás használatát. A fenti program egy olyan digitális eszközt ír le, ami beolvas egy számot, kiszámítja hárommal való osztás hányadosát és maradékát, majd az eredményeket a kimenetre irányítja. Ezt a kódot le lehet szimulálni. A 3. ábrán láthatjuk a szimulálás eredményét.



3. ábra
A szimuláció eredménye

A problémák akkor kezdődnek, amikor megpróbáljuk a kódot szintetizálni. Az általam használt szintetizátor nem enged osztani csak 2 hatványaival. Ahhoz, hogy egy bármilyen számmal való osztást megvalósítsunk, egy szekvenciális automatát (state machine) kell használjunk, ami elbonyolítja a kódot. Valószínű, hogy a komolyabb szintetizáló programok meg tudnak birkózni a problémával.

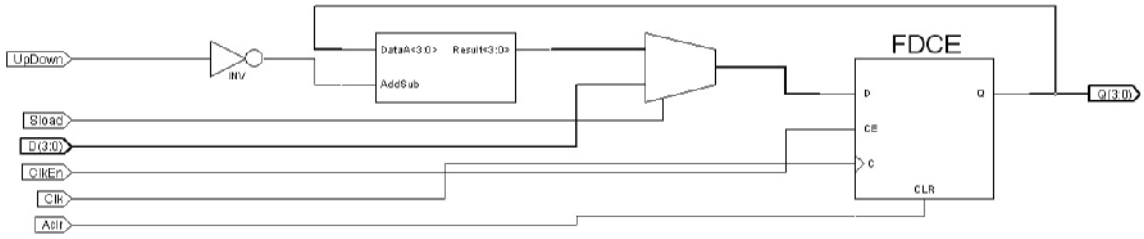
Amikor VHDL kód segítségével tervezünk egy áramkört, mindig arra kell gondoljunk, hogy a kód számára kell léteznie egy megfelelő áramkörnek. Az ilyen típusú tervezésnek van egy pár szabálya, amit jó ha betartunk, különben a szintetizátor nem tudja a kódot „lefordítani" fizikai áramkörbe, vagy pedig hibás áramkört készít.

A következőkben bemutatunk egy VHDL kódot, ami egy tipikus számlálót ír le, majd megvizsgáljuk a szintetizátor által előállított áramkör elemeit.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity szamlalo is
 port ( CLK: in STD_LOGIC;
 RESET: in STD_LOGIC;
 CE, LOAD, DIR: in STD_LOGIC;
 DIN: in STD_LOGIC_VECTOR(3 downto 0);
 COUNT: inout STD_LOGIC_VECTOR(3 downto 0)
 );
end szamlalo;
architecture Behavioral of szamlalo is
begin
process (CLK, RESET)
begin
   if RESET='1' then
      COUNT <= "0000";
   elsif CLK='1' and CLK'event then
      if CE='1' then
         if LOAD='1' then
          COUNT <= DIN;
         else
            if DIR='1' then
               COUNT <= COUNT + 1;
            else
               COUNT <= COUNT - 1;
            end if;
         end if;
      end if;
   end if;
end process;
end Behavioral;
```

Láthatjuk, hogy a számláló viselkedését a gerjesztés modellel írjuk le. Általában a gerjesztés modellek egy clock-ot engednek meg, amit az FPGA-ban tálalható logikai cellák felépítésével magyarázunk. A logikai cellák belső felépítésében találhatunk egy D bistabilt, amelynek a következő pinjei vannak: CLK, RESET, DATA, Q. A kódban a RESET jel mint egy aszinkron jel szerepel, és arra számítunk, hogy ez a jel a bistabil RESET jeléhez lesz kötve. A CLK jel a kódból mint szinkron jel jelenik meg, és valószínűleg a bistabil CLK jelére lesz majd kötve. Az elsif ágon belül található utasítások kombinatorikus áramkörökkel lesznek megoldva, de a kimeneten csak akkor jelennek meg, amikor a bistabil vált.

Most nézzük meg, hogy a szintetizátor milyen áramkört javasol. A 4. ábrán láthatjuk a szintetizátor által előállított áramkört.



4. ábra
A szintetizátor által előállított áramkör kapcsolási rajza

Mint azt láthatjuk, elég jól meg lehet saccolni a szintetizátor által előállított struktúrát, de ehhez természetesen szükséges az FPGA belső felépítésének az ismerete.

Összefoglalás

Amikor egy implementálható kódot akarunk írni, figyelembe kell vegyük az FPGA belső felépítését, mert különben nem leszünk eredményesek. Érdemes megtanulni egy pár VHDL programozási sémát, mert így elérhetjük, hogy gyorsan és könnyen tervezzünk áramköröket.

Sok algoritmus van, amely párhuzamos számítást igényel. Amennyiben ezeket az algoritmusokat sikerül FPGA-ba ágyazni, látványos sebességet érhetünk el. Az, amit a számítógép több ezer művelet alatt végez el, az FPGA chipekben csak pillanatok műve...

Bibliográfia

[1] Hosszú, G., F. Kovács, L. Varga, V. Gajodi (1998): "VHDL based circuit synthesis using language transformations", XI. Polish National Conference, Application of Microprocessors in Automatic Control and Measurement, Warsaw, October 13-14, 1998 pp. 42-48.
[2] J. Birkner: "A very-high-speed field-programmable gate array using metal-tometal antifuse programmable elements," Microelectronics Journal, v. 23, pp. 561-568
[3] Ulrich Heinkel: "The VHDL Reference", John Wiley And Sons, 2000
[4] Karen Parnell, Nick Mehta: "Programmable Logic Design Quick Start Handbook", Xilinx Coorporation, June 2003
[5] Sorin Hintea: "Tehnici de Proiectear cu Arii Logice", UTPress, 2003