PROGRAMOVATELNÁ LOGICKÁ POLE A JAZYKY HDL

Doc. Ing. Jaromír Kolouch, CSc.

Ústav radioelektroniky FEKT VUT v Brně, Purkyňova 118, kolouch@feec.vutbr.cz

Přednáška má přinést informaci o současném stavu v konstrukci číslicových systémů s využitím programovatelných logických polí (PLD, FPGA) a o jazycích HDL (zejména ABEL a VHDL) používaných jako vstupní prostředek k počítačovým návrhovým systémům. Podrobněji je pojednáno o jazyku VHDL, a jsou uvedeny příklady popisu subsystémů v tomto jazyku.

Nejběžnější dělení programovatelných obvodů v současné době je na obvody PLD (Programmable Logic Devices) a FPGA (Field Programmable Gate Arrays). Obvody PLD se dále dělí na obvody SPLD (Simple PLD) – dnes to jsou zejména obvody GAL16V8, GAL20V8 a GAL22V10 a jsou charakteristické tím, že obsahují jedno programovatelné pole, a obvody CPLD (Complex PLD – mnoho výrobců i různých typů) se strukturou odpovídající několika obvodům SPLD na společném čipu.

Obvody PLD jsou založeny na vyjadřování kombinačních logických funkcí ve tvaru součtu součinů (Sum Of Products – SOP), podobně jako známější paměti PROM. Na rozdíl od těchto pamětí, které mají pevně zapojenou součinovou část a programovatelné součtové pole, je u obvodů PLD programovatelné součinové pole a pevná součtová část (struktura PAL) nebo jsou programovatelné obě části (struktura PLA). Vyjádření logických funkcí ve tvaru SOP patří k základním znalostem v číslicové technice, takže princip implementace logických funkcí do obvodů PLD je vcelku snadno pochopitelný. V nejjednodušším případě odpovídá každé realizované logické funkci v obvodu PLD jedna makrobuňka, takže počet makrobuněk obsažených v těchto obvodech dává představu o tom, jak složité zapojení se do nich "vejde". U jednodušších aplikací není obtížné sestavit potřebný obraz konfigurace programovatelného pole ručně, i když se k tomu v praxi používají návrhové systémy. Obvody SPLD obsahují 8 až 10 makrobuněk, obvody CPLD jich obsahují až několik stovek. Obvody PLD se v současnosti vyrábějí s rozsahem až do zhruba 10 000 ekvivalentních hradel (pro GAL16V8 se udává ekvivalent 300 hradel).

Obvody FPGA obsahují pole malých programovatelných buněk, které se pro vytvoření logické funkce musí vhodně propojit. Úloha najít optimální propojení je zde mnohem složitější než u obvodů PLD a pro běžného uživatele to obvykle znamená spolehnout se na funkci návrhového systému, který toto propojení vytvoří automaticky. Rozsah logiky v obvodech FPGA se pohybuje o dva až o tři řády výše než u obvodů PLD (milióny i více ekvivalentních hradel).

Návrhové systémy jsou v současnosti nezbytným nástrojem pro práci s programovatelnými obvody. Vstupní údaje (popis vyvíjené konstrukce) je nutné zapsat ve formě, kterou je systém schopen převést na model této konstrukce. Ten je pak možno zpracovávat simulátorem k ověření jeho správnosti, časových parametrů a podobně, a dále jej syntetizérem a implementačním programem vložit (implementovat) do cílového programovatelného obvodu.

Zápis vstupních údajů bývá nejčastěji textový – pomocí jazyků HDL (Hardware Description Language) nebo grafický – editory schémat, stavových diagramů a podobně. Grafické systémy vstupu bývají obvykle nepřenositelné na jiný systém, než je ten, v němž byl popis vytvořen. Z tohoto hlediska jsou výhodnější systémy textového vstupu, které jsou ve výrazně větší míře standardizovány. K nejznámějším jazykům HDL patří jazyk ABEL a jazyk VHDL. Jazyk ABEL je poměrně jednoduchý, a jeho syntaxe vychází ze struktury obvodů PLD, pro něž je určen. Jazyk VHDL má výrazně vyšší stupeň abstrakce (a také složitosti), což dovoluje i syntézu zaměřenou na obvody FPGA.

Základní poznatky o jazyku VHDL a o jeho syntaxi:

* Jazyk VHDL je uznaným standardem, a dnes se prakticky nesetkáme s návrhovým systémem, který by tento jazyk nepodporoval.
* Konstrukce popsané v tomto jazyku je možno přenášet z jednoho návrhového systému na jiný (někdy po určitých nevelkých úpravách) a mohou být zpracovávány simulátorem.
* Návrhové systémy dovolují zvolit typ cílového obvodu, do nějž bude konstrukce implementována, až po základním zpracování popisu (simulaci a syntéze), případně tento typ změnit, pokud se ukáže, že původní typ nebyl zvolen optimálně.

Nejdůležitější syntaktická pravidla jazyka VHDL:

Popis konstrukce v jazyku VHDL se skládá ze dvou základních částí:

* z deklarace entity, kde jsou definovány brány (vstupy a výstupy signálů),
* z popisu architektury, který definuje vztah mezi vstupními a výstupními signály.

K deklaraci entity obvykle patří příslušný popis architektury. Někdy však může být účelné sestavit těchto popisů pro jednu entitu i více, a simulací nebo statickou časovou analýzou je pak možno například srovnat chování několika variant popisu architektury a vybrat pro implementaci tu, která je nejvýhodnější.

Před zápisem deklarace entity se obvykle uvádějí odkazy na knihovny a v nich na slohy (packages), ve kterých jsou definovány některé dále použité objekty nebo jejich vlastnosti (nejčastěji typy signálů a proměnných).

V jazyku VHDL rozlišujeme vyhrazená (klíčová) slova, kterým je definicí jazyka přiřazen určitý význam, a uživatelské identifikátory, kterými si uživatel označuje objekty (signály, proměnné a podobně). Existuje ještě jeden druh vyhrazených slov, která jsou definována dodatečnými standardy nebo výrobci (autory) návrhových systémů, která nejsou definována přímo standardem jazyka, a uživatel by je mohl změnit nebo užívat jinak než podle této dodatečné definice. To se však nedoporučuje, protože by to obvykle vedlo ke ztrátě některých vlastností návrhového systému.

Jazyk VHDL je necitlivý na malá a velká písmena (s několika málo výjimkami, které zde nebudou podstatné). Pro přehlednost budeme klíčová slova definovaná standardem jazyka zapisovat velkými písmeny, vyhrazená slova definovaná dodatečnými standardy písmeny malými a uživatelské identifikátory budeme značit smíšeně tak, že počáteční písmena budou velká (i u dílčích slov, z nichž může být identifikátor složen) a zbývající malá; jednopísmenové uživatelské symboly budou psány malými písmeny. Pro identifikátory můžeme používat písmena, číslice a podtržítko (to nesmí být posledním znakem a nesmí být v identifikátoru zřetězeno), první znak identifikátoru musí být písmeno. Příkazy se ukončují středníkem.

Jako příklad uvedeme popis modelu čtyřbitového komparátoru:

```
LIBRARY ieee;                 -- odkaz na knihovnu                 -- 1
USE ieee.std_logic_1164.ALL; -- odkaz na slohu v knihovně ieee    -- 2
ENTITY EqComp4 IS                                                  -- 3
  PORT(a,b:    IN  std_logic_vector(3 DOWNTO 0); -- vstupy        -- 4
       Equals: OUT std_logic);                   -- výstup        -- 5
END EqComp4;                                                       -- 6

ARCHITECTURE Dataflow OF EqComp4 IS                                -- 7
BEGIN                                                              -- 8
  Equals <= '1' WHEN (a = b) ELSE '0';                             -- 9
END Dataflow;                                                      -- 10
```

Komentáře začínají dvěma pomlčkami a končí na konci řádku. Mezery, tabulátory a znaky nového řádku představují oddělovače, které můžeme používat podle potřeby libovolně tak, aby text byl přehledný. Řádky 3, 7, 8 zde nepředstavují samostatné příkazy, takže nejsou ukončeny středníkem. Text za nimi navazující by při vynechání komentáře (čísla řádku) mohl pokračovat na stejném řádku, nový řádek je vytvořen jen pro lepší přehlednost. Chceme-li šetřit místem, můžeme toho někdy využívat, jak bude zřejmé z dalšího textu pro popis komparátoru procesem. Čísla na koncích řádků jsou v uvedeném textu zapsaná jako komentář slouží jen ke snadnějšímu odkazování na řádky, běžně se neuvádějí.

Text začíná odkazem na knihovnu `ieee` a na slohu `std_logic_1164`. V této sloze jsou definovány typy `std_logic` a `std_logic_vector`, které jsou dále přiřazeny signálům v portech (branách). Typ signálu určuje, jakých hodnot může signál nabývat a jaké operace jsou pro ně definovány. Některé typy, například typ `bit` nebo `integer`, jsou definovány přímo standardem jazyka VHDL, jiné jsou však definovány dodatečnými standardy ve slohách, na něž musíme uvést odkaz. Signály typu `std_logic` mohou nabývat hodnot `'0'`, `'1'` a některých dalších, pro nás zatím nepříliš podstatných, jako například `'Z'` – stav vysoké impedance, a podobné je to pro signály vektorového typu `std_logic_vector`. Deklarace entity začíná klíčovým slovem `ENTITY` (řádek 3), za nímž následuje její označení, zde `EqComp`. Toto označení je i na konci deklarace entity, za slovem `END`. Dále následuje deklarace bran (řádky 4 a 5) – to jsou signály, kterými je entita propojena s okolím. Signálům je v příkazu `PORT` přiřazen mód: pro vstupní signály mód `IN`, pro výstupní `OUT` a pro obousměrné (pokud by zde byly) mód `INOUT`. Signály mohou být jednobitové – skalární, nebo vícebitové – vektorové. U vektorových signálů se uvádí rozsah jejich indexů v závorkách.

V popisu architektury (jeho název můžeme zvolit, zde je to `Dataflow`) je mezi klíčovými slovy `BEGIN` a `END` uveden logický popis funkce modelu. Před slovem `BEGIN`, v tzv. deklarativním úseku popisu architektury, mohou být ještě deklarovány vnitřní signály, které jsou dostupné uvnitř modelu, ale ne navenek, a některé další objekty (např. komponenty).

V jazyku VHDL můžeme rozeznávat několik stylů popisu. Uvedený popis je příkladem stylu popisujícího tok dat (dataflow), který je charakterizován použitím tzv. souběžných přiřazovacích příkazů. Jiné druhy stylu jsou styl behaviorální, u nějž je charakteristické použití procesu, a styl strukturální, který je podobný známému netlistu u systémů pro návrh plošných spojů. Pojem stylu je jen pomůcka pro roztřídění různých způsobů zápisu a není v samotném jazyku VHDL syntakticky definován, jazyk VHDL styly nerozlišuje. Často se styl popisující tok dat pokládá za druh behaviorálního stylu.

Popisy modelů vytvářené konstruktérem se zpravidla píší behaviorálním stylem, který je pro člověka dobře srozumitelný. Pouze některé prvky, které nemají behaviorální vyjádření popisu, je nutno vkládat strukturálním stylem jako komponenty. Strukturálním stylem se také popisují hierarchické konstrukce složené z nižších bloků, které pak jsou zpravidla popsány behaviorálně, pokud je to možné. Strukturální styl je charakteristický pro některé popisy vytvářené automaticky návrhovým systémem jako mezivýsledky syntézy, které nejsou určeny pro přímou analýzu či úpravu konstruktérem.

Procesy, které jsou základem behaviorálního stylu popisu, představují mocný nástroj. Mohou být použity pro popis jak kombinačních, tak i sekvenčních subsystémů. Zde však je nutno rozlišovat pojem "sekvenční" vztažený na číslicové obvody a na druhy příkazů v jazyku VHDL: v tomto jazyku se jako sekvenční příkazy označují příkazy používané v procesech, na rozdíl od příkazů souběžných umístěných mimo proces. Souběžné příkazy si můžeme představit jako popis součástek umístěných na desce plošného spoje, u nichž jejich umístění na této desce a také jejich pořadí v zápisu textu v jazyku VHDL nemá žádný význam pro

funkci popisovaného subsystému, význam má jen jejich vzájemné propojení. Naproti tomu sekvenční příkazy mají charakter podobný příkazům programovacích jazyků pro počítač, a jejich vzájemná poloha v zápisu procesu, v němž se nacházejí, ovlivňuje výsledek. Přesněji řečeno, podobnost programovacím jazykům platí v procesech pro datové objekty zvané proměnné. Pro signály v procesu platí poněkud odlišné pravidlo, že hodnota signálu, který je zapsán na pravé straně přiřazovacího příkazu (vytváří se tak z ní nová hodnota dalšího signálu nebo proměnné), zůstává během provádění procesu neměnná, a je-li v procesu tomuto signálu přiřazena nová hodnota, projeví se to až po ukončení procesu. Je-li nová hodnota přiřazena signálu v procesu vícekrát, má signál po ukončení procesu hodnotu danou posledním přiřazením, předcházející přiřazení jsou ignorována.

Tato zdánlivě složitá pravidla jsou vynucena tím, že jazyk VHDL popisuje zapojení složené ze souběžně pracujících číslicových obvodů, zatímco počítač, který tuto simulaci provádí, nemůže pracovat jinak než sekvenčně, tedy vykonávat jeden příkaz po druhém.

Příkladem, který může osvětlit podstatu a smysl procesu, je známý algoritmus pro písemné dělení, jak se jej učí žáci ve škole. Dělení je ve své podstatě složitá kombinační logická funkce, ale rozepsáním do algoritmu, tj. do posloupnosti jednodušších úkonů, můžeme z dělence a dělitele získat podíl a zbytek vcelku nenáročnými operacemi. Tento algoritmus je tedy analogií procesu, kde vstupními a výstupními signály jsou dělenec, dělitel, podíl a zbytek, a mezivýsledky představují vnitřní signály a proměnné.

Proces představuje také jedinou možnost, jak lze v jazyku VHDL behaviorálně popsat registr řízený hranou. Podrobnosti budou uvedeny později formou příkladu. Souběžné přiřazovací příkazy můžeme ze syntaktického hlediska pokládat za zkrácenou formu zápisu procesů.

Výše uvedený popis modelu komparátoru můžeme ekvivalentně zapsat pomocí procesu (pro stručnost zapíšeme jen popis architektury, deklarace entity zůstává beze změny):

```
ARCHITECTURE Behavioral OF EqComp4 IS BEGIN                    -- 1
  PROCESS(a,b) BEGIN                                           -- 2
    Equals <= '0';                                             -- 3
    IF (a = b) THEN Equals <='1'; END IF;                      -- 4
  END PROCESS;                                                 -- 5
END Behavioral;                                                -- 6
```

Za klíčovým slovem `PROCESS` (řádek 2) následuje v závorkách seznam signálů, při jejichž změně se proces "spustí", tj. vytvoří se podle příkazů v něm uvedených nové hodnoty signálů, kterým jsou v procesu hodnoty přiřazeny (zde je to signál `Equals`). Pokud by v procesu byly použity proměnné, bylo by je nutno deklarovat v deklarativní části procesu, tj. před následujícím slovem `BEGIN`. Za tímto slovem začíná příkazová část procesu. V ní se signálu `Equals` v řádku 3 přiřadí nulová hodnota, a je-li splněna podmínka v následujícím řádku, přepíše se tato hodnota jedničkou. Pak je proces ukončen na řádku 5 a signál `Equals` nyní nabývá novou hodnotu. Podmíněný přiřazovací příkaz `IF … THEN` představuje sekvenční verzi souběžného podmíněného přiřazovacího příkazu `WHEN … ELSE`, který byl použit v předcházejícím popisu.

Začátek a konec příkazové části popisu architektury i procesu je vyznačen klíčovými slovy `BEGIN` a `END`. Každému slovu `BEGIN` musí odpovídat příslušné slovo `END`. Podobně se ohraničuje podmíněný příkaz `IF … THEN`, který musí být ukončen slovy `END IF`. Pro přehlednost je dovoleno, aby za slovem `END` následovalo označení, k čemu se vztahuje. Je vhodné si znovu uvědomit, že všechny znaky mezery, tabulátoru a nového řádku představují oddělovače, které můžeme libovolně přidávat ke stávajícím oddělovačům tak, aby text byl co nejpřehlednější (do komentáře však nemůžeme vložit znak nového řádku). Text získá na přehlednosti, jestliže se úrovně vnoření příkazů `BEGIN` a `END` vyznačí odsazením zleva.

Uvedeme nyní, jak se v jazyku VHDL popisují synchronní subsystémy – například často používané synchronní čítače. Následující text představuje popis modelu čítače čítajícího v rozsahu 1 až 6, vybaveného asynchronním naplněním číslem 1 pomocí signálu `RstNt` (tento čítač může být použit jako základ pro konstrukci elektronické kostky):

```
LIBRARY ieee;                                                    -- 1
USE ieee.std_logic_1164.ALL;                                     -- 2
USE ieee.std_logic_unsigned.ALL;                                 -- 3
ENTITY Cnt1to6 IS                                                -- 4
  PORT(Clk,RstNt: IN  std_logic;                        -- vstupy  -- 5
       Cnt1_6:    OUT std_logic_vector(2 DOWNTO 0));   -- výstup  -- 6
END Cnt1to6;                                                     -- 7

ARCHITECTURE Behavioral OF Cnt1to6 IS                            -- 8
  SIGNAL CntInt: std_logic_vector(Cnt1_6'range);                 -- 9
BEGIN                                                            -- 10
  PROCESS(Clk,RstNt) BEGIN                                       -- 11
    IF RstNt = '0' THEN CntInt <= "001";                         -- 12
    ELSIF (Clk'event AND Clk = '1') THEN                         -- 13
      CntInt <= CntInt + 1;                                      -- 14
      IF CntInt = "110" THEN CntInt <= "001"; END IF;            -- 15
    END IF;                                                      -- 16
  END PROCESS;                                                   -- 17
  Cnt1_6 <= CntInt;                                              -- 18
END Behavioral;                                                  -- 19
```

Na začátku textu je uveden odkaz na další slohu `std_logic_unsigned`, ve které je definováno tzv. přetížení operátoru sčítání tak, aby tento operátor mohl být použit pro sčítání signálu typu `std_logic` a typu `integer` (v řádku 14; v základní definici jazyka je tento operátor definován jen pro sčítání objektů typu `integer`). Označení slohy `std_logic_unsigned` odpovídá tomu, že se sčítání děje podle pravidel platných pro interpretaci objektů typu `std_logic_vector` jako celá čísla bez znaménka.

Signál s módem `OUT` má podle syntaktických pravidel vytvořen jen výstupní kanál a není dostupný uvnitř modelu. U čítače se však následující stav vytváří ze současného stavu, takže signál představující současný stav musí být uvnitř modelu dostupný. Proto je v deklarativní části popisu architektury deklarován vnitřní signál `CntInt`, pro nějž je dále zapsán popis funkce čítače, a jeho hodnota je v závěru popisu architektury přiřazena výstupnímu signálu `Cnt1_6`.

V procesu je asynchronní naplnění čítače uvedeno jako první příkaz (řádek 12), a má tedy přednost před synchronní funkcí čítače, jak je to obvyklé. Následující řádek 13 představuje behaviorální popis vzestupné hrany signálu `Clk`, a uvozuje popis synchronní funkce čítače, tj. jeho reakce na tuto aktivní hranu hodinového signálu. V řádku 14 se čítač inkrementuje, a v dalším řádku 15 je podmíněný příkaz, který čítač převádí do počátečního stavu, pokud jeho obsah byl při spuštění procesu roven číslu 6 – v tomto případě se předcházející přiřazení pro inkrementaci signálu `CntInt` ignoruje, jak bylo výše uvedeno.

Jazyk VHDL je velmi bohatý a výše uvedené příkazy jsou určeny jen k ilustraci jeho použití k popisu nejběžnějších číslicových subsystémů. Pro konstruktéra, který hodlá tento jazyk používat, však není nezbytné, aby se snažil aktivně zvládnout jazyk jako celek. Ve velké většině svých potřeb vystačí jen s omezeným výsekem z jazykových konstruktů, a v případě potřeby může najít potřebné informace v odborné literatuře. Je možné přirovnat tento jazyk například k programu Word: málokdo z něj aktivně ovládá více, než potřebuje pro svou běžnou práci, ale s použitím nápovědy nebo manuálu lze získat další informace. Je však potřebné mít přehled o možnostech jazyka a o tom, kde tyto informace vyhledat.