Návrh obvodů založených na programovatelných hradlových polích

V předchozím článku (Automatizace č. 1/2008, str. 9) jsme se zabývali koncepcí a vlastnostmi programovatelných hradlových polí, jejich aplikačními možnostmi a výběrem vhodného obvodu pro konkrétní aplikaci. Tento příspěvek popisuje základní kroky návrhu programovatelného obvodu. Text je doplněn názorným příkladem jednoduchého návrhu na programovatelném hradlovém poli.

POUŽITÉ ZKRATKY

DUV	*Design Under Verification*
FPGA	*Field Programmable Gate Array*
GAL	*Generic Array Logic*
HDL	*Hardware Description Language*
PAL	*Programmable Array Logic*
RTL	*Register Transfer Level*
SDF	*Standard Delay File*
SRAM	*Static Random Access Memory*
VHDL	*Very high speed integrated circuit Hardware Description Language*

Úvod

V historických dobách programovatelných obvodů byl běžný návrh obvodu bez pomoci počítače. U jednoduchých obvodů typu PAL či při použití paměti PROM pro realizaci logické funkce to ani nebyl problém. Spolu se zvětšováním a rostoucí komplexností součástek a s neustále klesajícím časem na návrh se však plně ruční realizace stávala stále méně vhodnou. Objevily se tak nástroje, které umožnily celý proces automatizovat. Přestože návrhové nástroje pro integrované obvody jsou extrémně drahým softwarem, aplikace pro návrh obvodů FPGA jsou významně levnější. Někteří výrobci programovatelných hradlových polí dokonce nabízejí zjednodušené verze svého softwaru zdarma [1 až 3]. Aplikace mají mnoho omezení, které nejsou příliš vhodné pro použití v průmyslové sféře, ale pro malé obvody a domácí kutění zcela postačují. Návrhový cyklus se skládá ze specifikace požadavků, zachycení návrhu, simulace, verifikace a implementace (*obr. 1*). Kroky se často opakují se změnou požadavků specifikace nebo opravami chyb v návrhu. Proto hovoříme o cyklu.

Specifikace funkce

Prvním krokem návrhu musí být vždy specifikace funkce. Na jejím správném vytvoření závisí úspěch celé práce. Specifikační dokument musí být:

- *úplný* – musí popisovat všechny požadované aspekty chování budoucího obvodu, nevynechat nic důležitého;
- *bezchybný a bezesporný* – jednotlivé požadavky na funkce systému si nesmí vzájemně odporovat;
- *schválený* – všemi stranami participujícími na projektu.

Pro demonstraci jednotlivých kroků návrhu použijeme velmi jednoduchý digitální obvod. Ten postupně popíšeme v jazyce VHDL, zverifikujeme a provedeme jeho implementaci. Nejprve tedy musíme specifikovat požadavky kladené na jeho chování a vlastnosti. Vytvoříme funkční blok pro měření časového intervalu od resetu systému do výskytu události na vstupním signálu. Jeho funkci můžeme popsat v následujících bodech:

1. Obvod pracuje s náběžnou hranou hodin (aktivní hrana), požadovaná hodinová frekvence je 50 MHz.
2. Asynchronní reset (aktivní v log. 1) vynuluje stav čítače, spustí měření času (počtu hodinových pulzů). Uvolnění asynchronního resetu je již externě synchronizováno s aktivní hranou hodin.
3. S každým hodinovým pulzem se stav čítače zvýší o jedničku.
4. Rozsah měření času (čítání) je 0 až 255 hodinových pulzů.
5. Pokud čítaná hodnota dosáhne 255, měření času se automaticky zastaví (čítač nepřeteče).
6. Stav čítače je přímo přístupný na výstupní sběrnici.
7. Indikace vstupní události zastaví čítání času až do příštího resetu. Vstupní signál indikující událost je synchronní proti hodinám, událost je indikována aktivní hodnotou (log. 1) na vstupu.

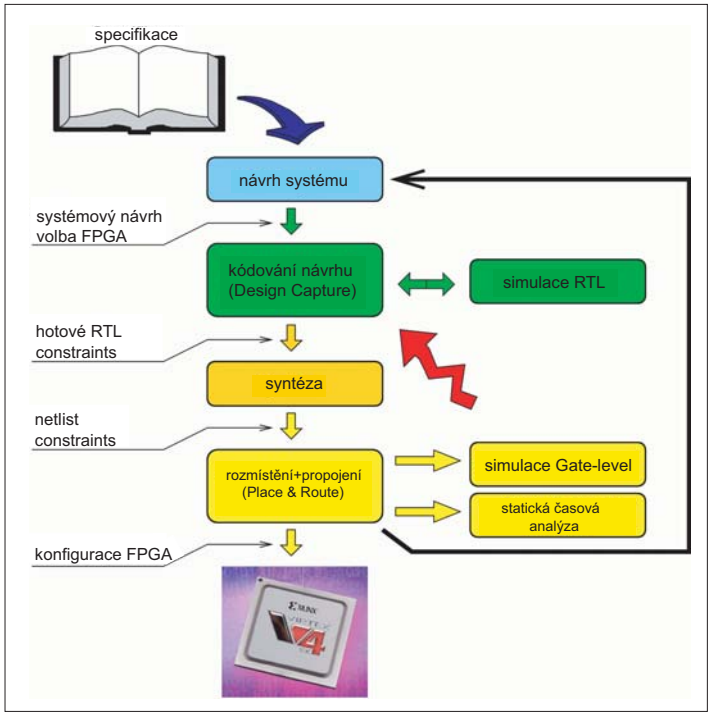
Zachycení návrhu

Zachycení návrhu (*design capture, design entry*) je proces, při němž přenášíme svou představu o funkci a struktuře budoucího obvodu a o požadavcích specifikace do počítačem zpracovatelné formy popisu požadované digitální funkce. Práci lze provádět na různých úrovních abstrakce:

- *Hradlové/schematické*
  Navrhujeme přímo kreslením schématu budoucího obvodu. Výhodou tohoto postupu je jeho srozumitelnost a zachycení skutečné podoby návrhu – co máme ve schématu je to, co se realizuje. Nevýhody nicméně převyšují nad výhodami. Kreslené schéma je obvykle specifické pro zvolený obvod, protože často používáme struktury, které jsou k dispozici jen na příslušném obvodu FPGA. Konverze návrhu do jiného obvodu FPGA, nebo do zákaznického integrovaného obvodu, znamená překreslení schématu. Vlastní proces kreslení je pomalý a únavný, protože pracujeme na nízké úrovni abstrakce – kreslíme obvod hradlo po hradle. Chceme-li například realizovat stavový automat, musíme nejprve zminimalizovat jeho přechodovou a výstupní funkci, a pak nakreslit schéma. Děláme tak zbytečně práci, kterou by za nás zvládl udělat počítač. Navíc schematický návrh téměř neumožňuje používat generické konstrukce, navrhovat parametrizovatelné bloky (například čítač je stále osmibitový, není možné ho jednoduše rozšířit, je nutné přikreslit další hradla). To významně omezuje znuvupoužitelnost vytvořených logických bloků. Zbytečně komplikované a časově náročné jsou i opravy chyb v navrženém obvodu. Snadno se můžeme dostat do situace, kdy je nutné kompletní překreslení. Uložené schéma je obvykle nepřenosné mezi různými vývojovými prostředími (uložené v proprietárním formátu). Schematický návrh obvodů FPGA se pro jeho nevýhody dnes už téměř nepoužívá.
- *Meziregistrové přenosy, tzv. RTL*
  Digitální synchronní obvody se skládají ze dvou základních typů logiky: registrů a kombinačních funkcí. Na úrovni RTL zachytíme digitální obvod tak, že jednotlivé struktury popíšeme pomocí těchto dvou typů logiky a doplníme informací o propojení jednotlivých logických bloků (odkud, kam a přes jaké kombinační logické funkce jsou přelévána data mezi registry). Popis obvodu je realizován v textové podobě pomocí zápisu ve speciálním programovacím jazyku (HDL). Nejčastěji se můžeme setkat s jazyky VHDL a Verilog. Použití úrovně RTL má nesporné výhody: získáme technologicky nezávislý popis obvodu na relativně vysoké úrovni abstrakce. Jazyky HDL mají běžně podporu pro vytváření vysoce konfigurovatelných (generických) struktur. Například čítač může mít obecnou šířku a při jeho použití jen nastavíme odpovídající konstantu v kódu. To umožňuje znovu použít jednou navržené bloky. Vysoká úroveň abstrakce významně zjednodušuje přenos návrhu mezi různými technologiemi a zrychluje jak vlastní návrh, tak pozdější opravy. Popis RTL je dnes standardním prostředkem pro zachycení návrhu. Jedinou nevýhodou je nevhodnost pro ryze asynchronní návrh, to ale není při práci s hradlovými poli ome-

Tab. 1 Zdrojový kód demonstračního obvodu

```
--import knihoven
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
--deklarace interface bloku
ENTITY evnt_cnt IS
  PORT (
    res  : IN  std_logic; --reset - vstupni signal
    clk  : IN  std_logic; --hodiny - vstupni signal
    evnt : IN  std_logic; --indikace udalosti, vstupni sig-
nal
    cnt  : OUT std_logic_vector (7 DOWNTO 0) --citac,
vystupni sbernice (8 bit)
  );
END ENTITY evnt_cnt;
--vlastni implementace bloku - co je unvitr krabicky
s interface def. nahore
ARCHITECTURE rtl OF evnt_cnt IS
  --interni signaly
  SIGNAL cnt_d          : std_logic_vector (7 DOWNTO 0); --
citac, vstup registru
  SIGNAL cnt_q          : std_logic_vector (7 DOWNTO 0); --
citac, vystup registru
  SIGNAL cnt_allowed_d : std_logic; --je povolene citani?
vstup registru
  SIGNAL cnt_allowed_q : std_logic; --je povolene citani?
vystup registru
BEGIN
  --je povolene citani? Pristi stav registru
  cnt_allowed_d <= '0' WHEN evnt  = '1'         ELSE --ne
kdyz je udalost
                   '0' WHEN cnt_q = "11111110" ELSE --ne
kdyz bude citac 255
                   cnt_allowed_q; --jinak zachovej soucas-
ny stav
  --citac - pristi stav citace udalosti.
  cnt_d         <= cnt_q + 1 WHEN cnt_allowed_q = '1' ELSE
--+1, pokud je povoleno citani
                   cnt_q; --jinak udrzuj aktualni stav
  --popis registru v obvodu, reaguji na hodiny a reset
  registers : PROCESS (clk, res)
  BEGIN
    IF res = '1' THEN --je aktivni asynchronni reset?
      cnt_q         <= "00000000";
      cnt_allowed_q <= '1';
    ELSIF clk = '1' AND clk'EVENT THEN --hodiny jsou
v jednicce po zmene?
      --tedy, byla nabezna hrana na hodinach?
      cnt_q         <= cnt_d;
      cnt_allowed_q <= cnt_allowed_d;
    END IF;
  END PROCESS registers;
  cnt <= cnt_q; --vystupni signal - prirazeni
END ARCHITECTURE rtl;
```



Obr. 1 Model návrhového procesu

zující. Poznamenejme nicméně, že jazyky HDL nejsou omezeny jen na práci na úrovni RTL. Můžeme stejně dobře popisovat chování obvodu na behaviorální (vhodné pro různé pomocné bloky pro simulaci, obvykle nelze užít pro návrh), strukturní (hierarchie bloků) i hradlové úrovni (pak vlastně získáme textovou podobu schématu – tzv. netlist). Ne všechny konstrukce, které jazyk HDL umožňuje, jsou použitelné pro návrh logiky. Množina konstrukcí, které můžeme použít, je označována jako „syntetizovatelná podmnožina jazyka“ (*synthesizable subset*).

- *Algoritmické* Neustále se zkracující délka návrhového cyklu spolu s rostoucí složitostí navrhovaných systémů nutí návrháře používat stále vyšší úrovně abstrakce. Jestliže použití úrovně RTL umožnilo významně zrychlit návrh, jeho ladění i proces implementace, stále nebyla odstraněna základní a časově náročná operace: konverze algoritmu do architektury, kterou potom návrhář popíše na úrovni RTL. Architektura na úrovni RTL je navrhována vždy s přihlédnutím ke skutečnému časování obvodu a použitému paralelismu. Systém je třeba navrhnout s odpovídajícím počtem výkonných výpočetních jednotek, řídicích bloků, sběrnic apod. Problém nastává v okamžiku, kdy v pozdějších fázích návrhu zjistíme, že navržená architektura nesplňuje naše očekávání, například pro nízký výpočetní výkon. Změna ve struktuře systému se pak prakticky rovná kompletnímu přepsání příslušné části. Právě odstranění informace o paralelismu a časování ze zdrojového kódu je přínosem algoritmické syntézy. Funkce bloku je popsána v některém z jazyků na „vyšší úrovni“, například v prostředí ANSI C, Handel-C, SystemC nebo System Verilogu. Příslušný syntézní nástroj pak dostane informaci o počtu funkčních jednotek a časování ve formě jednoduchých omezení (například povolíme použití nejvýše dvou násobiček a dvou sčítaček) a na základě předloženého algoritmu vygeneruje RTL kód výsledného systému (datových cest i řídicích bloků). Každá změna v architektuře je triviální. Zjistíme-li, že výpočetní výkon systému je příliš nízký, stačí jen znovu spustit syntézní proces s jiným počtem aritmetických jednotek. Rychlost celého procesu umožňuje vyzkoušet celou řadu alternativ architektury, tzv. analýza „*what-if*“ („*co kdyby?*“) a najít nejvhodnější kompromis mezi plochou a rychlostí obvodu. Zjednodušuje se i verifikace. Použitelnost algoritmické syntézy nicméně omezuje fakt, že výsledek není zatím tak dobrý (velikost, spotřeba, finální vyladění), jak by ho navrhl člověk. Proto se používá zejména u velkých digitálních obvodů pro číslicové zpracování signálu, kde výhoda rychlého návrhu významně převyšuje snížení kvality.

Pro větší projekty je často vhodné používat specializované nástroje pro návrh, např. HDLDesigner [4], které umožňují skloubit grafický i textový pohled na systém, navrhovat stavové automaty kreslením diagramu přechodů apod. Práce je pak blízká užívání vizuálních nástrojů pro programování.

Vraťme se zpět k našemu příkladu. Na základě požadavků uvedených ve specifikaci můžeme vytvořit kód (*tab. 1*). Pro zápis byla použita syntetizovatelná podmnožina jazyka VHDL.

Verifikace

Verifikace (*verification*) představuje další významný krok v procesu návrhu obvodu. V rámci verifikačního procesu ověřujeme prostřednictvím simulací funkční správnost navrhovaného obvodu a dodržení časových parametrů použitých prvků. Aby byla verifikace úspěšná, musí jí předcházet důkladné plánování. Na základě požadavků ve specifikaci je třeba sestavit seznam simulací, které budeme provádět, a pro každou simulaci navrhnout stimuly, jež vybudí odpovídající chování obvodu. K tomu musíme specifikovat i očekávané odezvy obvodu pro kontrolu správné funkce. Dalším krokem je návrh verifikačního prostředí (*testbench*). Prostředí je systém napsaný v jazyce HDL, který

instancuje verifikovaný obvod (DUV), aplikuje požadované stimuly a v ideálním případě i sám ověří správnost odezvy obvodu. Implementace automatické kontroly odezev obvodu sice komplikuje psaní verifikačního prostředí, nicméně umožňuje za všech podmínek objektivní kontrolu správné funkce systému nezávisle například na únavě návrháře. Verifikujeme-li i jen jednodušší obvod s řádově desítkami požadavků ve specifikaci, automatická kontrola se stává zcela nezbytnou i z časových důvodů. Simulaci můžeme provádět na různých úrovních abstrakce:

- *Simulace RTL kódu* (*RTL simulation*)
  Simulujeme návrh zapsaný v kódu HDL na úrovni abstrakce RTL. Prováděná simulace nerespektuje reálná zpoždění na jednotlivých prvcích logiky, ale je velmi rychlá. Simulace RTL je vhodná pro ověření správné funkce popisu RTL HDL. Můžeme pomocí ní ověřit, zda je kód funkčně správně a obvod reaguje, jak jsme si přáli. Nejsme ovšem schopni ověřit korektnost interního časování (zpoždění na kombinačních prvcích apod.).
- *Simulace na hradlové úrovni s reálnými zpožděními* (*back-annotated gate level simulation*)
  Simulujeme *netlist* obvodu po rozmístění a propojení, tedy finální schéma obvodu implementovaného ve zvolené technologii. Spolu s *netlistem* simulátor potřebuje soubor SDF [5 a 6] s informacemi o reálném zpoždění a požadovaném časování na jednotlivých spojích a prvcích obvodu. Dalším nezbytným vstupem pro simulaci na hradlové úrovni jsou modely skutečných obvodových prvků načtené z technologické knihovny poskytnuté výrobcem obvodu FPGA. Ty zajišťují modelování reálného funkčního chování bloků na FPGA, dále simulují zpoždění šíření signálu podle informací ze souboru SDF, a nakonec kontrolují dodržování správných časových parametrů jednotlivých bloků (například dodržení předstihu a přesahu u klopných obvodů). Při porušení předepsaného časování je během simulace generováno automaticky chybové hlášení. Proces nastavení časových parametrů v instancích modelů jednotlivých buněk obvodu je proveden ihned po načtení *netlistu* a souboru SDF a nazývá se zpětná anotace (*back annotation*).
- *Simulace na hradlové úrovni o syntéze* (*post-synthesis netlist simulation*)
  Simulujeme netlist po syntéze před rozmístěním a propojením. Vzhledem k charakteru obvodů FPGA a syntézního procesu nelze po syntéze spolehlivě odhadnout zpoždění kombinační logiky a spojů v obvodu, a tak nejsme schopni simulovat reálné chování. Proto se postsyntézní simulace obvykle nedělají. Jejich použití se u obvodů FPGA téměř výlučně omezuje jen na ověření správné funkce syntézního nástroje, pokud máme podezření na chybu v návrhovém softwaru, a ne v návrhu.

Roli důkladných a důsledných simulací v návrhovém procesu nelze podceňovat. V každém systému jsou chyby a praxe ukazuje, že nesimulované bloky budou s jistotou chybně navržené. Ani v případě dokonalého návrhu RTL nelze vyloučit chyby v návrhových nástrojích, jež mohou mít na finální produkt devastující dopad.

S verifikací je nutné počítat i během plánování projektu. Lze říci, že čas strávený návrhem a verifikací je u většího obvodu v poměru zhruba 3 : 7; přitom je běžný i významně horší poměr (ještě více času na verifikaci). Ani odložení simulací ve prospěch testování ve finální aplikaci („v železe“) není ideální postup. Obecně platí, že náklady na odstranění chyby během tvorby specifikace, kódu RTL, verifikace a ladění ve finální aplikaci jsou v poměru zhruba 1 : 10 : 100 : 1 000.

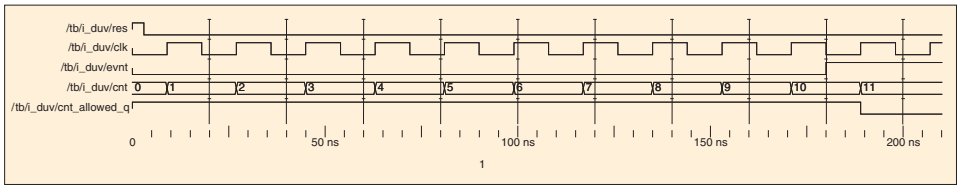
Volba vhodných stimulů ani dokonalé provedení verifikace nicméně nezaručuje správnou funkci navrhovaného obvodu. Stavový prostor digitálního integrovaného obvodu je obrovský a pokrýt simulacemi všechny relevantní kombinace funkčních režimů, konfigurace obvodu a časování lze skutečně jen u velmi jednoduchých obvodů. Mnoho scénářů tak typicky nesimulujeme a správnou funkci obvodu můžeme zajistit jen dobrým návrhem a volbou zkušeného návrháře.

Součástí ukázkového příkladu je i jednoduché verifikační prostředí. Bude jen generovat stimuly, automatické ověření funkce nebude z důvodů jednoduchosti implementováno. Abychom otestovali požadavky specifikace, přivedeme na vstup obvodu stimuly podle *tab. 2*. Hodinovou frekvenci zvolíme jako *fclk* = 55 MHz. Specifikace vyžaduje, aby obvod fungoval na 50 MHz, tedy byla připočtena rezerva 10 %. Výsledný kód lze nalézt v *tab. 3*. Pro simulaci příkladu použijeme volně dostupný simulátor ModelSim XE Starter firmy Mentor Graphics. Po jeho instalaci ze stránek [3], kompilaci, načtení kódu do simulátoru a spuštění simulace získáme průběhy na *obr. 2*. Po kontrole proti *tab. 2* můžeme konstatovat, že na úrovni RTL obvod funguje podle očekávání. Podobně můžeme provést simulace na hradlové úrovni. V časových průbězích pak budeme pozorovat reálná zpoždění na buňkách obvodu. Při simulaci větších obvodů nicméně rychle narazíme na úmyslné omezení funkčních možností volně šiřitelného simulátoru (ModelSim XE). Pro profesionální práci je nezbytné vlastnit plnou licenci simulátoru (např. ModelSim SE/PE). Simulace pak poběží zhruba 30× rychleji (jak na RTL, tak na hradlové úrovni).

Implementace

Dalším krokem je konverze popisu RTL HDL do konfiguračních dat pro programovatelné hradlové pole. Proces implementace se skládá z několika kroků:

- *Syntéza* (*synthesis*)
  Obecně se jedná o proces konverze popisu systému na vyšší úrovni abstrakce na nižší úroveň abstrakce. V případě syntézy RTL (dnes stále nejběžnější) hovoříme o konverzi kódu RTL HDL zapsaného pomocí syntetizovatelné podmnožiny jazyka do seznamu logických prvků a jejich vzájemného propojení (netlist). Přitom jsou detekovány ve zdrojovém kódu registry a kombinační logika, zakódovány symbolické stavy stavových automatů, rozpoznány specielní struktury (např. násobičky, sčítačky). Dále dochází k odstranění nepoužité logiky a dalším optimalizacím návrhu. Kromě kódu HDL je proces syntézy řízen ještě technologickou knihovnou a tzv. *constraints* (omezení). Technologická knihovna obsahuje popisy funkcí, časování a další důležité parametry všech prvků dostupných ve zvoleném typu obvodu FPGA. Knihovnu dodává výrobce programovatelného hradlového pole. *Constraints* naproti tomu dodává návrhář obvodu a jejich prostřednictvím definuje různé klíčové parametry, např. očekávanou maximální hodinovou frekvenci, na které má obvod pracovat,



Obr. 2 Simulace funkce demonstračního obvodu

Tab. 2 Stimuly a očekávaná chování obvodu během verifikační simulace

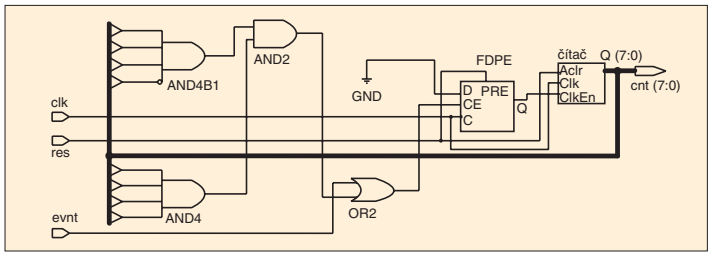
počet period hodin od startu simulace	stimuly	pokryté požadavky specifikace	očekávaná odezva
0	generování resetu	2, 6	vynulování výstupu
1–9	nic	3, 6	čítání
10	generování události	6, 8	zastavení čítání
11–15	nic	6, 8	zastavení čítání
15	generování resetu	2, 6	vynulování výstupu
15–272	nic	4, 5, 6	dočítání do 255, zastavení

Tab. 3 Zdrojový kód verifikačního prostředí

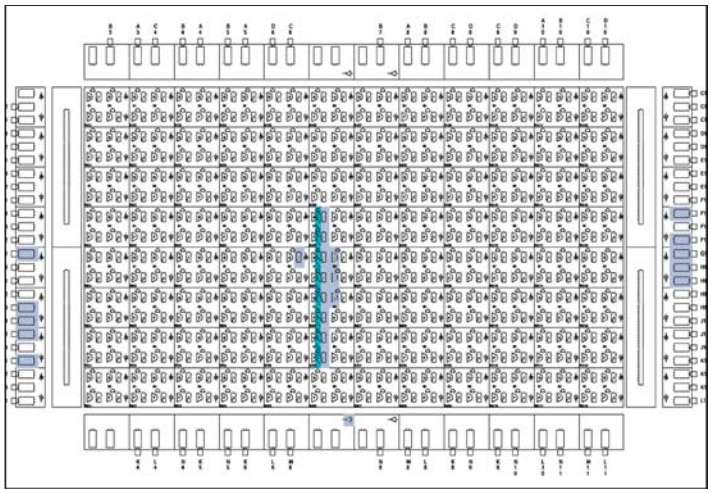
```
--import knihoven
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
--interface bloku je prazdny, zadne vstupy ani vystupy
ENTITY tb IS
END ENTITY;
--vlastni implementace bloku - co je unvitr krabicky
s interface def. nahore
ARCHITECTURE beh OF tb IS
  --vnitrni signaly
  SIGNAL res_i  : std_logic; --reset pro testovany obvod
  SIGNAL clk_i  : std_logic; --hodiny pro testovany obvod
  SIGNAL evnt_i : std_logic; --indikace udalosti obvodu
  SIGNAL cnt_i  : std_logic_vector (7 DOWNTO 0); --citana
hodnota
  CONSTANT clk_per : time := 18 ns; --perioda generovanych
hodin
  --budeme chtit pouzivat blok evnt_cnt s timto interfacem
  COMPONENT evnt_cnt IS
    PORT (
      res  : IN  std_logic;
      clk  : IN  std_logic;
      evnt : IN  std_logic;
      cnt  : OUT std_logic_vector (7 DOWNTO 0)
    );
  END COMPONENT evnt_cnt;
BEGIN
  clk_generator : PROCESS --generator volne bezicich hodin
  BEGIN
    clk_i <= '0';
    WAIT FOR clk_per/2;
    clk_i <= '1';
    WAIT FOR clk_per/2; --perioda je 18 ns = 55 MHz
  END PROCESS clk_generator;
  evnt_res_generator : PROCESS --generator stimulu
  BEGIN
    res_i  <= '1'; --reset obvodu
    evnt_i <= '0'; --zadna udalost
    WAIT FOR 3.5 ns; --uvolni asynchronne reset
    res_i  <= '0'; --bez resetu
    evnt_i <= '0'; --zadna udalost
    WAIT FOR 10*clk_per - 3.5 ns; --cekej 10 period hodin
    res_i  <= '0'; --bez resetu
    evnt_i <= '1'; --generuj udalost
    WAIT FOR 5*clk_per; --cekej 5 period hodin
    res_i  <= '0'; --bez resetu
    evnt_i <= '0'; --zadna udalost
    WAIT FOR 4*clk_per; --cekej 4 periody hodin
    res_i  <= '1'; --reset obvodu
    evnt_i <= '0'; --zadna udalost
    WAIT FOR 3.5 ns; --uvolni reset
    res_i  <= '0';
    evnt_i <= '0'; --zadna udalost
    WAIT FOR 257 * clk_per - 3.5 ns; --nech dojet citac na
maximalni hodnotu
    ASSERT FALSE --ukonci simulaci
    REPORT "Konec simulace"
    SEVERITY FAILURE;
  END PROCESS evnt_res_generator;
  --instance DUV - verifikovaneho bloku
  i_duv : evnt_cnt
    PORT MAP (
      res  => res_i,
      clk  => clk_i,
      evnt => evnt_i,
      cnt  => cnt_i
    );
END ARCHITECTURE beh;
```

nebo mapování vstupů a výstupů návrhu na skutečné piny obvodu FPGA.

- *Mapování na technologii* (*mapping*)
  Obecně se jedná o konverzi technologicky nezávislého netlistu do buněk dané technologie. U FPGA ovšem toto často provádí už vlastní syntézní nástroj. Další funkcí mapování je seskupování LUT, registrů a dalších logických prvků do řezů obvodu a doplnění LUT použitých jen pro



Obr. 3 Schéma RTL obvodu



Obr. 4 Blokové schéma programovatelného hradlového pole Spartan xc2s15

propojení bloků kdekoliv, kde je to nezbytné (tzv. *route-thru LUT*). Současně jsou prováděny další jednodušší optimalizace obvodu, odstranění pří- padné zbývající nepoužité logiky, replikace registrů apod. [7].

- *Rozmístění a propojení* (*place and route*)
  Schéma obvodu je namapováno („zobrazeno“) do matice obvodu FPGA. V prvním kroku – rozmístění – jsou jednotlivé logické prvky umístěny do matice obvodu, jejich pozice je zafixována. V druhém kroku – propojení – jsou postupně probírány jednotlivé spoje mezi prvky v obvodu a jsou identifikovány vhodné propojovací struktury (kanály, spoje, přepínací prvky) v matici hradlového pole a postupně zafixovány jednotlivé spoje. Nástroje pro rozmístění a propojení jsou dodávány výrobci obvodů FPGA, protože pro svou správnou funkci potřebují detailní znalost struktury obvodu.

Pro účinnou optimalizaci struktury obvodu na rychlost je třeba, aby nástroj syntézy znal zpoždění v jednotlivých kombinačních větvích obvodu co nejpřesněji. To je u moderních obvodů FPGA čím dál tím složitější úkol, protože v submikronových technologiích výroby obvodů už jasně dominuje zpoždění na spojích v obvodu ve srovnání se zpožděním na logických prvcích. Zpoždění na spojích jsou obtížně predikovatelná – závisí nejen na schématu obvodu, ale i na velikosti FPGA a zaplnění jeho propojovacích kanálů.

Proto se setkáváme s tzv. *fyzickou syntézou* (*physical synthesis*). Obvod je syntetizován, je proveden odhad budoucího rozmístění a propojení a na jeho základě jsou vypočteny zpřesněné odhady zpoždění jednotlivých kombinačních cest. Dále jsou analyzována jednotlivá zpoždění a opětovně jsou syntetizovány příliš pomalé části obvodu. Celý proces je opakován, dokud není dosaženo požadovaných parametrů nebo do okamžiku, kdy je situace vyhodnocena jako bezvýchodná. Algoritmy fyzické syntézy jsou implementovány například v nástroji Precision RTL Physical [7]. Algoritmy fyzické syntézy má smysl používat, pokud používáme novější hradlová pole (Xilinx Virtex III, IV, V apod.), pro návrh větších

Tab. 4 Velikost a rychlost demonstračního obvodu

počet použitých registrů	9
počet použitých čtyřvstupových LUT	4
celkem obsazených řezů	7
minimální perioda hodin	5,7 ns
maximální hodinová frekvence	175 MHz

obvodů, případně pokud chceme využít obvod FPGA „do poslední buňky“. U starších produktových řad nebudou přínosy fyzické syntézy nijak oslňující.

Nyní již můžeme přikročit k experimentální implementaci našeho čítače – jeho syntéze, rozmístění a propojení do zvoleného programovatelného obvodu. Pro naše pokusy použijeme softwarový balík Xilinx ISE WebPack [1]. Pro implementaci použijeme obvod Spartan 2 15-6cs144. Finální parametry rozmístěného a propojeného obvodu spolu se schématem po syntéze lze nalézt v *tab. 4* a *obr. 3*. Na *obr. 4* lze nalézt graficky znázorněné zaplnění obvodu FPGA.

Konfigurace FPGA

Posledním krokem návrhového procesu je konfigurace programovatelného hradlového pole. V našem případě předpokládáme použití SRAM FPGA firmy Xilinx. Zde jen stačí propojit počítač s vývojovou deskou a pomocí modulu Impact z programového balíku ISE nakonfigurovat FPGA.

Technické parametry návrhu

Abychom byli schopni proces návrhu účinně řídit, potřebujeme mít zpětnou vazbu pomocí vhodně zvolených metrik, které kvantitativně udávají kvalitu a „stav rozpracovanosti“ celé práce na obvodu. Z hlediska technických parametrů návrhu nás obvykle zajímají odpovědi na tyto otázky.

- *Jaká je velikost výsledného návrhu?*
  Během návrhu je užitečné sledovat velikost výsledného obvodu a zejména zaplnění obvodu FPGA. Praktická zkušenost říká, že není dobré zaplnit FPGA „až na doraz“ jednak z důvodů možného problému při opravách chyb v systému (obvod se tím může zvětšit) a dále z důvodů potenciálních problémů s časováním. V téměř zaplněném obvodu je *router* často nucen propojovat logické prvky „oklikami“ díky vysokému zahlcení kanálů FPGA už propojenými spoji. To způsobuje nepříjemné prodlužování jednotlivých spojů a zvyšování zpoždění – snižování maximální hodinové frekvence. Je rozumné nechat asi 10 % obvodu volného. Podobně nemá smysl zaplnit FPGA jen z malé části. Je pak vhodnější zvolit menší a levnější obvod.
- *Jaká je maximální přípustná systémová hodinová frekvence?*
  Maximální hodinová frekvence systému je u mnoha návrhů kritický parametr. Maximální dosažitelnou hodinovou frekvenci můžeme zjistit pomocí tzv. statické časové analýzy (STA, Static Timing Analysis). Nástroj STA (např. modul *trace* v balíku Xilinx ISE) provede analýzu kritických cest v obvodu bez potřeby simulace (proto statická analýza), najde nejdelší cesty a na jejich základě je schopen určit spolehlivý odhad maximální hodinové frekvence.
- *Jaká je přibližně spotřeba navrhovaného obvodu?*
  Především při návrhu větších systémů pracujících na vysokých hodinových frekvencích je nezbytné věnovat pozornost také tepelnému výkonu disipovanému obvodem FPGA a na jeho základě navrhnout vhodnou techniku chlazení obvodu. Výrobci obvodů FPGA poskytují i nástroje pro odhad ztraceného tepelného výkonu.

Kvalitativní parametry návrhového procesu

Monitorování kvality návrhového procesu (a zejména verifikace správné funkce) je v řadě případů zanedbáváno. Kvalitu vlastního procesu návrhu je přitom nutno sledovat stejně důkladně jako vlastnosti výsledného obvodu, nechceme-li se ve finální aplikaci dočkat ošklivých překvapení. Z „manažerského“ pohledu na věc nás často budou zajímat odpovědi na tyto otázky:

- *Kolik z požadavků specifikace je skutečně pokryto simulacemi?*
  Principiálně neexistuje žádná univerzální a snadno použitelná metoda, jak to zjistit. Jedno z mála dobře fungujících řešení je disciplína při návrhu testovacích simulací. Návrhář by si v ideálním případě měl udržovat např. v tabulkovém kalkulátoru matici incidence simulací a požadavků a pečlivě dbát na to, aby každý z požadavků byl skutečně simulacemi otestován (tedy aby obvod byl buzen stimuly, které vyvolají danou situaci a aby byly korektně kontrolovány jeho odezvy).
- *Jaká část kódu HDL obvodu je vykonána simulacemi?*
  Myšlenka za touto otázkou je velmi jednoduchá. Pokud část návrhu (nějaký podblok obvodu) není simulacemi vybuzena a část kódu HDL se nikdy nevykoná, nemáme ani šanci zjistit, zda v něm není chyba. Naopak to samozřejmě neplatí. I v kódu, který je vykonán, mohou být chyby, pokud jej korektně nezverifikujeme. Moderní simulátory umožňují automaticky sledovat a reportovat vykonávání jednotlivých řádků kódu VHDL. Příslušný proces je nazýván „pokrytím kódu RTL simulacemi“ (*code coverage*). Podpora pro *code coverage* nicméně není ve volně šiřitelné verzi simulátoru ModelSim XE, ale jen v placených plných verzích.
- *Jaká část skutečného obvodu (jeho struktury) je „rozhýbána“ simulacemi?*
  Stejně jako v předchozím případě se snažíme zjistit, zda je celý obvod aktivován použitými stimuly. Zde ale aplikujeme přístup na simulace na hradlové úrovni a sledujeme logickou aktivitu na jednotlivých spojích ve finálním obvodu. I pro tento typ měření mají digitální simulátory integrovanou podporu, proces nazýváme „pokrytí netlistu simulacemi“ (*toggle coverage*). I v tomto případě je nutné vlastnit plnou licenci simulátoru.

Závěr

Příspěvek ve stručnosti rozebral základní kroky a postup návrhu obvodu na programovatelném hradlovém poli. Jednotlivé kroky byly demonstrovány na ukázkovém příkladu návrhu obvodu pro měření časového intervalu. Pro čtenáře, kteří by chtěli celý postup vyzkoušet „na vlastní kůži“, jsme připravili balíček [12]. Jedná se o ukázkový příklad se všemi kódy, skripty a dalšími poznámkami k práci s příslušnými návrhovými nástroji.

Ing. Jakub Šťastný, Ph.D.
katedra teorie obvodů
FEL ČVUT v Praze
ASICentrum, s. r. o.

LITERATURA

[1] Xilinx, ISE WebPack, *http://www.xilinx.com/ise/logic_design_prod/webpack.htm* [18. 2. 2008]

[2] Altera, Quartus II, *http://www.altera.com/products/software* [18. 2. 2008]

[3] Xilinx, ModelSim Xilinx Edition-III Starter, *http://www.xilinx.com/ise/verification/mxe_details.html* [18. 2. 2008]

[4] Mentor Graphics, HDL Designer, *http://www.mentor.com/products/fpga_pld/hdl_design/hdl_designer_series* [18. 2. 2008]

[5] Open Verilog International, Standard Delay Format Specification, version 3.0, 1995.

[6] RUČKAY, L., *Z05-3: Návrh prostředí pro verifikaci FPGA bloků*. Praha : ČVUT FEL, Katedra teorie obvodů, 2005. *http://amber.feld.cvut.cz/fpga/publikace.html*

[7] Mentor Graphics, Precision Physical Synthesis, *http://www.mentor.com/products/fpga_pld/synthesis/precision_synthesis*

[8] ŠŤASTNÝ, J., BÍLÝ, P., An FPGA Microcontroller Design. *Elektrorevue*, 2006, č. 30. *http://www.elektrorevue.cz/clanky/06030/english.htm* [18. 2. 2008]

[9] MAXFIELD, C., *Design Warrior's guide to FPGA*. Elsevier, 2004, ISBN-13: 9780750676045

[10] LASS, S., Improving Time to Design Closure with ISE Software. *XCell Journal*, Fourth Quarter, 2005, pp. 6–8.

[11] ŠŤASTNÝ, J., *Syntéza na vyšší úrovni. http://amber.feld.cvut.cz/fpga/prednasky/hll_synteza/hlls.html* [18. 2. 2008]

[12] Ukázkový příklad návrhu, dostupné ke stažení z adresy *http://amber.feld.cvut.cz/fpga/prednasky/FPGA_navrh/fpga_navrh.html*

[13] ŠŤASTNÝ, J., Programovatelná hradlová pole. *Automatizace*, 51 (2008), č. 1, s. 9–13.