Tento článek je původním rukopisem textu publikovaného v časopise DPS Elektronika A-Z:
J. Šťastný, M. Skiba. Precision RTL a konverze ASIC obvodu na FPGA platformu, DPS Plošné spoje od A do Z, no 2, pp. 4-6, 2010.

Bez souhlasu autorů tohoto materiálu a redakce časopisu DPS a uvedení zdroje není povolena jakákoli další publikace, přetištění nebo distribuce tohoto materiálu nebo jeho části. Další podmínky použití jsou uvedeny na internetové stránce *http://minimizedlogic.sweb.cz/*.

PrecisionRTL a konverze ASIC obvodu na FPGA platformu

Ing. Jakub Šťastný, Ph.D., Ing. Michal Skiba
ASICentrum s.r.o., Novodvorská 994, Praha 4,142 00, Česká republika

Úvod

V poslední dekádě pozorujeme prudký rozvoj mobilních aplikací. Ruku v ruce s novými možnostmi, které mobilní systémy nabízejí a s rostoucí komplexitou SoC (Systems On Chip, systémů na čipu) používaných v mobilních terminálech ovšem přicházejí i nové výzvy. Z nich nejpalčivější je nutnost dosáhnout co nejkratšího času návrhu jak číslicového systému, tak finální aplikace. I přes rostoucí složitost navrhovaných zařízení je třeba u nových projektů dobu návrhu udržet stejnou, ne-li menší, než dříve.

Součástí aplikace je často i velké množství software nezbytného pro běh SoC systému – ať už je to firmware, nebo zákaznická aplikace. A zde je skryt kámen úrazu – chceme-li navrhnout a vyladit aplikační software, potřebujeme mít k dispozici vlastní SoC systém. Přitom softwarové aplikace jsou komplexní a jejich návrh časově náročný. S vývojem proto není možné čekat na ukončení návrhu SoC systému, je třeba pracovat paralelně s návrhem čipu, ať už je „programování“ prováděno přímo na pracovišti navrhujícím SoC systém, nebo třetí stranou. Navíc návrhem software paralelně s SoC čipem získáme možnost ladit číslicový systém a případně včas upravit architekturu podle požadavků programátorů.

Popsanou situaci lze řešit několika způsoby:

- **simulací SoC v PC** – nejjednodušší, nejsou potřeba dodatečné nástroje, ale simulace je příliš pomalá pro praktické užití.
- **použitím hardwarové akcelerace, nebo emulačního systému** – elegantní řešení, které ale vyžaduje speciální nástroje a hardwarovou podporu s poměrně vysokou pořizovací cenou.
- **FPGA prototypováním** – implementací prototypu systému v FPGA obvodu. Dodatečné specializované bloky pak umožňují propojení prototypu přímo s PC a implementaci tzv. emulátoru.

Pro implementaci emulátoru lze použít proprietární desku s FPGA obvodem a dalšími podpůrnými obvody stejně dobře jako standardní FPGA kit. Zvídavého čtenáře můžeme odkázat na článek o koncepcích systémové emulace [1].

Hradlování hodin

Druhou velkou výzvu představuje mobilita spolu s omezenou kapacitou baterií v mobilním zařízení. Ta dále omezuje množství a složitost funkcí navrhovaného systému. Proto je trendem implementovat do číslicových systémů v maximální možné míře prvky pro snížení spotřeby energie.

Hradlování hodin je nejčastější technika používaná při návrhu zákaznických integrovaných obvodů s nízkou spotřebou. Jedním z největších spotřebičů energie v integrovaném obvodu je hodinový strom. Periodické nabíjení a vybíjení kapacit v hodinovém stromu spotřebovává až 50% z celkové spotřeby čipu [2]. Hradlování hodin brání šíření změn hodinového signálu po hodinovém rozvodu v době, kdy má být příslušná část obvodu neaktivni. To vede ke snížení spotřeby elektrické energie. Princip hradlování hodin je zachycen na obr. 1a.

Návrh zákaznického obvodu

Uvažujme nyní následující situaci zobrazenou na obr. 2: přidáním hradlovacího členu *and_1* opozdíme hodinový signál označený jako *clk_r2*; signál povolující hodiny je pojmenován *gate*. Základním předpokladem pro správnou činnost synchronního návrhu je, aby aktivní hrana hodin dorazila do všech klopných obvodů v jeden okamžik. Tohoto ideálního stavu nikdy nedosáhneme. Rozdíl mezi časy, kdy náběžná hrana dorazí do různých klopných obvodů, nazýváme *skew*. Jedním z úkolů implementačních nástrojů je omezení *skew* během procesu nazývaného *vyvážení hodinového stromu*. Nástroj pro syntézu hodinového stromu přidá budič *BUF*, čímž opozdí i signály *clk_r1* a *clk_r3*, a

tak dojde k vyvážení hodinového stromu. Použitý obvod – viz obr. 2 – je tvořen třemi sériově zapojenými klopnými obvody typu D (posuvný registr), přičemž po dobu jednoho hodinového taktu byla na vstup *d1* nastavena logická „1“, která se pak s každým taktem hodin dále šíří obvodem. Na obrázku 3 jsou příklady časových průběhů u korektně fungující implementace.

Návrh na FPGA

Problémy nicméně nastávají při konverzi návrhu zákaznického integrovaného obvodu do FPGA emulátoru. Na FPGA je hradlování hodin velmi nebezpečná praktika, hodinový rozvod FPGA obvodu je pevně dán a není možné jednoduše kompenzovat zpoždění hradlovacího obvodu vložením budiče do nehradlované části hodinového stromu. Navíc problémy vzniklé hradlováním hodin na FPGA nejsou viditelné na RTL úrovni a nemusí se vyskytnout po každé iteraci implementace; můžeme tak získat obvod, který nefunguje jen za určitých provozních podmínek (teplota, napájecí napětí). Chyby způsobené touto nevhodnou praktikou návrhu pak patří k těm nejzáludnějším.

Na obrázku 4 jsou vidět opět časové průběhy signálů v posuvném registru; hradlovací prvek zpožďuje hodiny *clk_r2*. Registr *reg_q1* pracuje správně, registr *reg_q2* dostává hodiny zpožděné hradlovacím členem *and_1*, a proto se, na první pohled paradoxně, objeví na jeho výstupu logická jednička o takt dříve. Mezi registry *reg_q2* a *reg_q3* pak v našem případě problém není.

Z výše uvedeného jasně vyplývá, že hradlování hodin na FPGA nelze použít.

Využití clock enable na FPGA

Odstranění hradel v hodinovém rozvodu u již existujícího návrhu zákaznického integrovaného obvodu může být provedeno následujícími postupy:

- **ruční konverzí**, převedením hradlování hodin na tzv. clock enable – registr s recirkulací, viz obrázek 1b. Ruční konverze již existujícího návrhu zákaznického obvodu je velmi zdlouhavá.
- **automatickou konverzí během syntézy pro FPGA**. Pokud by syntezátor dokázal sám rozpoznat nebezpečné konstrukce a převést je na schéma v obrázku 1b, podstatně by to zkrátilo dobu potřebnou k návrhu obvodu.

Naše zkušenosti

V minulosti jsme při převodu ASIC kódů na FPGA platformu řešili každý případ hradlování hodin samostatně. Ve většině případů je užito hradlování hodin pouze jako metody pro snížení spotřeby obvodu, užití tedy nemá vliv na logickou funkci systému. V takovém případě lze hradlovací člen jednoduše vyjmout a nahradit ho přímým propojením. Problémem samozřejmě je být si jist, že hradlovací člen lze opravdu odstranit. V případě, že hradlovací člen má vliv na logickou funkci obvodu, je zapotřebí vhodně přepsat RTL kódy tak, aby byla zachována logická funkce obvodu. Opět vyvstával problém jak ověřit, že přepsaný kód je co do funkce ekvivalentní. Změna RTL kódu stála několik dnů práce vyplněné průzkumem ne vždy známého návrhu a diskusemi s jeho autory.

V současné době plně využíváme automatickou konverzi prováděnou syntézním nástrojem, v našem případě PrecisionRTL od firmy Mentor Graphics, [3]. Ta veškeré problémy vyřeší za nás, jedinou starostí návrháře je zkontrolovat, zda konverze proběhla na všech místech, kde bylo na ASIC obvodu užito hradlování hodin. O tom, že PrecisionRTL provedl transformaci hradlovaných hodin, nás informuje hlášením *Gated clock net:clk_g converted into clock net:clk_int and complex enable logic.*

Pro bezchybnou funkci syntézního nástroje je třeba správně nastavit *constraints*. Každý hodinový signál který vstupuje do hradlovacího členu by měl být definován příkazem *create_clock*, více viz [4]. Syntéza je ovšem v naprosté většině případů schopna rozpoznat i hodinové signály, které nebyly takto nadefinovány.

Závěr

Automatická konverze hradlování hodin během FPGA syntézy může v případě implementace FPGA prototypu většího SoC systému ušetřit značné množství času. Použití PrecisionRTL má nicméně i další výhody; zejména podporu pro syntézu do FPGA obvodů všech výrobců. To dále šetří čas v případě, že je nutné navrhovat současně pro FPGA více výrobců (např. v rámci jednoho projektu pro Xilinx i Alteru, nebo různé projekty pro Xilinx, různé pro Alteru), protože používáme stále stejné uživatelské rozhraní. Co je ale důležitější – používáme stále stejný syntezátor; mezi jednotlivými nástroji totiž existují rozdíly v implementaci některých konstrukcí, ne vždy nutně musí stejný RTL kód vést na funkčně (!) stejný výsledek při použití jiného syntézního nástroje. Ani integrace s nástroji pro rozmístění a propojení, které jsou vždy proprietární pro výrobce FPGA obvodů, se není třeba bát; naše zkušenost získaná více jak pěti lety používání na zhruba desítce projektů ukazuje, že integrace je bezproblémová.

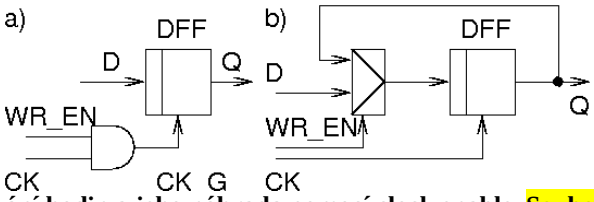
Poděkování

Práce na tomto příspěvku byla podpořena ARTEMIS Joint Undertaking, grantová dohoda n° 100029.

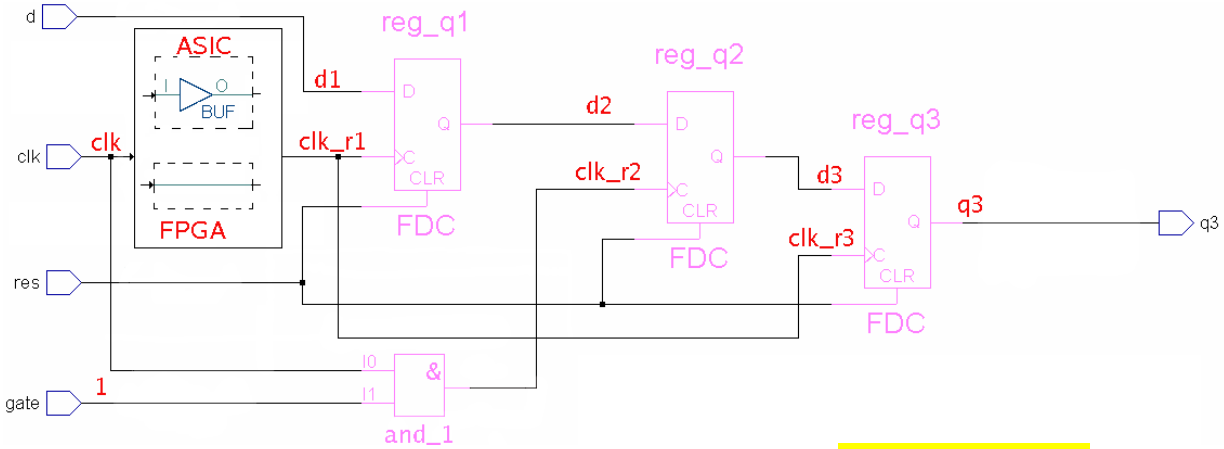
Použitá literatura

[1] J. Jager. FPGA-based prototyping of ASIC, ASSP, and SoC designs. PLD Design Line, 21.10.2009
http://www.eetimes.com/design/embedded/4015246/FPGA-based-rapid-prototyping-of-ASIC-ASSP-and-SoC-designs
[2] J. Shabel. Analyzing Clock Trees, SNUG Boston 2005.

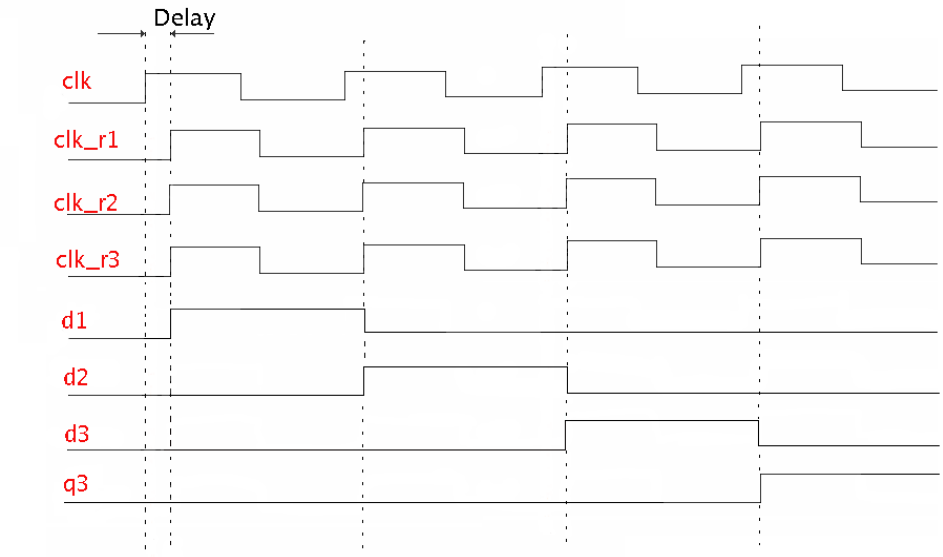
[3] PrecisionRTL, http://www.mentor.com/products/fpga/synthesis/precision_rtl/
[4] Mentor Graphics. Precision Synthesis Reference Manual.



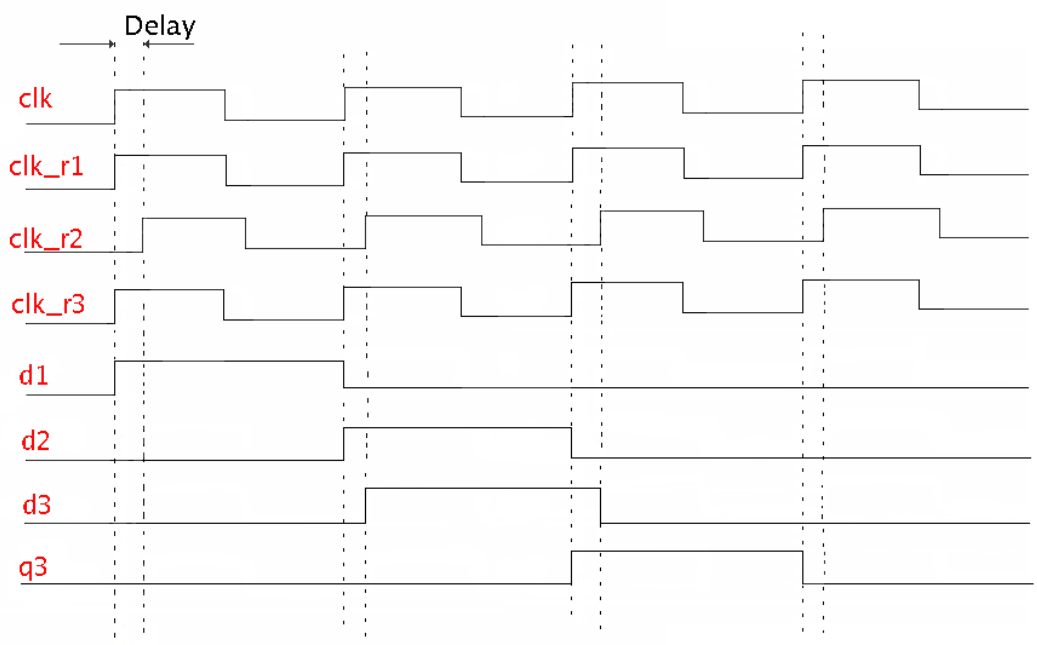
Obrázek 1 Hradlování hodin a jeho náhrada pomocí clock enable. Soubor ck_gating.png



Obrázek 2 Posuvný registr s hradlováním hodin. Soubor asic_fpga1.png



Obrázek 3 Časové průběhy důležitých signálů, implementace pro ASIC. Soubor wave_asic.png



Obrázek 4 Časové průběhy důležitých signálů, FPGA implementace.