Tento článek je původním rukopisem textu publikovaného v časopise DPS Elektronika A-Z:
J. Šťastný. Simulace číslicových obvodů na hradlové úrovni: dobrá praxe, DPS Elektronika od A do Z, pp. 14 - 18, březen/duben 2016

Simulace číslicových obvodů na hradlové úrovni: dobrá praxe

Jakub Šťastný
ASICentrum, s.r.o.
Katedra teorie obvodů FEL ČVUT Praha

1 Úvod

V předchozích příspěvcích [1] a [2] byl čtenář seznámen s obecnými vlastnostmi, výhodami a nevýhodami simulace číslicového obvodu na hradlové úrovni (*back-annotated gate level simulation, timing simulation*) a technickými detaily modelování obvodu na této úrovni abstrakce. Tento článek sérii o simulacích na hradlové úrovni uzavírá a shrnuje několik doporučení, jejichž důsledné dodržování by mělo provádění takových simulací zjednodušit – pohlédneme zde na přípravu simulací z pohledu pracovních postupů. I nyní je text nezávislý na návrhové platformě a prezentované poznatky jsou aplikovatelné jak při návrhu zákaznických integrovaných obvodů, tak při návrhu číslicových systémů na FPGA obvodech.

2 Před spuštěním simulace

Provádění simulace na hradlové úrovni je náročné na čas a nervy návrháře i když vše funguje jak má. Aby byla zátěž návrháře spojená s prováděním simulací co nejmenší, doporučujeme před spuštěním simulace udělat některé přípravné práce:

1. **Důkladně prostudovat žurnály** (logy) generované nástroji pro implementaci číslicového systému (z kompilátoru VHDL pro simulaci, ze syntezátoru a z nástrojů pro rozmístění a propojení). Před zahájením simulací na hradlové úrovni je třeba porozumět všem varováním poskytovaným těmito nástroji a odstranit všechny hlášené problémy v návrhu (např. neúplné citlivostní seznamy a nezamýšlené hladinové klopné obvody, více viz [3]). Dále je doporučené spustit statickou kontrolu kódu (LINT, viz [1]) a vyřešit všechny objevené problémy.
2. **Provést statickou časovou analýzu** navrhovaného obvodu, vyřešit všechny reportované problémy s časováním – ať už úpravou *constraints*, nebo opravou příslušné části návrhu.
3. **Identifikovat všechny synchronizátory** v navrhovaném systému a vypnout příslušné kontroly časových parametrů (*timing checks*) na klopných obvodech určených k izolaci metastability.
4. **Vybrat množinu testů**, které budou na hradlové úrovni spouštěny.
5. **Zvolit pracovní podmínky** simulovaného obvodu.

2.1 Jak na resynchronizátory

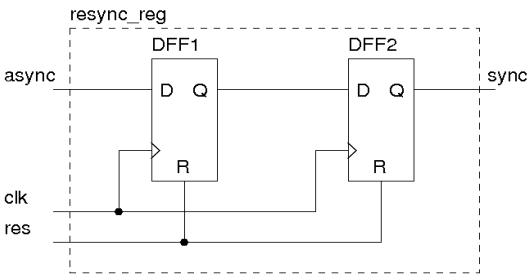
Resynchronizátor (viz obrázek 1) slouží pro synchronizaci asynchronního signálu do lokální hodinové domény a tak z principu věci na jeho vstupu (klopný obvod *DFF1*, vstup *D*, v obrázku 1) běžně dochází k porušení předstihu či přesahu (*setup time violation*, *hold time violation,* více viz [4] a [5], kapitola 11). Protože modely klopných obvodů dodržení předstihu a přesahu během simulace kontrolují, byl by výstup *Q* registru *DFF1* nastaven za těchto podmínek do stavu *X* po příští hodinový cyklus. To může mít nežádoucí dopad na další běh simulace – může dojít k zaplavení simulovaného

návrhu stavem *X* a způsobit nefunkčnost obvodu v simulaci. Generování stavu *X* je zde zbytečně pesimistickým opatřením. Proto je běžně kontrola předstihu a přesahu na *DFF1* v hradlových simulacích vypínána. Ale pozor: bavíme se pouze o *DFF1*, na *DFF2* k porušení předstihu a přesahu *nesmí* docházet a není důvod kontroly vypínat.

Dobrou praxí je před spuštěním simulace na hradlové úrovni připravit seznam všech resynchronizátorů v návrhu – to by měla být práce návrhářů RTL kódu. Potom je třeba pro každý resynchronizátor identifikovat registr *DFF1* v netlistu a vypnout na něm příslušné kontroly časových parametrů. Tento úkol může být poněkud obtížný, pokud návrháři obvodu nedodržují jednoduché konvence: základním pravidlem je, že resynchronizátor má být implementován jako samostatná entita se jménem obsahujícím např. *resync* v názvu. Tak je pak možné resynchronizátory jednoduše vyhledat v návrhu například příkazem *grep* nebo hledáním řetězce v *Total Commanderu* a současně je sníženo riziko záměny resynchronizátoru s běžným posuvným registrem se vstupem synchronním k hodinám, apod.

Vypnutí kontroly předstihu a přesahu na klopném obvodu *DFF1* je specifické podle použitého simulátoru, prostudujte si proto dokumentaci k Vašemu nástroji. Univerzálním postupem nezávislým na simulátoru je ruční editace záznamu pro předstih a přesah v *SDF* souboru (více viz [6], kapitola 7.2.4).

Vypínání kontrol dodržení časových parametrů klopných obvodů v návrhu je delikátní operace, která může při nesprávném provedení vést i k přehlédnutí vážných chyb v návrhu. Dobrou praxí je provést review všech vypnutých časových kontrol zkušeným návrhářem, který na projektu nepracuje a má tak „čerstvé oči“ - a vše pečlivě zdokumentovat. A pozor: klopný obvod na kterém byla vypnuta kontrola předstihu a přesahu při porušení časových parametrů svůj výstup ponechá v předchozím stavu (viz např. [7], strana 36). Ve skutečnosti by ale mohl klopit, simulace tedy nemusí odpovídat reálnému chování návrhu v FPGA obvodu. Částečným řešením může být lepší modelování resynchronizátoru na RTL úrovni, kde tento jev lze podchytit, ale to už je nad rámec tohoto textu – viz například [6], kapitola 7.7.



Obrázek 1: RTL schéma synchronizátoru. Soubor synchronizer.eps.

2.2 Výběr testů

V [1] jsme se dotkli potřeby správně vybrat sadu testů pro verifikaci obvodu na hradlové úrovni, protože simulace na hradlové úrovni může běžet významně pomaleji než na RTL úrovni a je celkově náročnější na provádění. Obecně je třeba (viz např. shrnutí v [8]) spouštět takovou skupinu testů, aby byly jak vyzkoušeny všechny jeho hlavní funkce, tak i bloky (funkční a strukturní pokrytí). Dále množství spouštěných testů závisí na tom, jak velkou důvěru vkládáme ve výsledky statické časové analýzy a jak moc je navrhovaný systém synchronní. Uveďme tedy základní pravidla pro výběr skupiny testů pro simulaci na hradlové úrovni:

1. **Start systému**: Všechny testy, které verifikují start systému po náběhu napájecího napětí je vhodné spustit i na hradlové úrovni. Start systému je kritickou operací a chyba během startu může být pro navrhovaný produkt fatální, proto je třeba verifikaci věnovat zvláštní pozornost.
2. **Generování resetů**: V simulacích na hradlové úrovni by měly být aktivovány všechny způsoby, kterými lze obvod

inicializovat. Blok/bloky návrhu zodpovědné za generování resetu pro celý systém jsou kritické a chyba zde může být smrtící pro celý navrhovaný systém.

3. **Generování hodin**: Pokud číslicový systém obsahuje případně bloky pro přepínání, hradlování, či generování hodin, je doporučené na hradlové úrovni spustit verifikační testy, které ověří správnou funkci těchto bloků. I zde – stejně jako při generování resetu – může chyba napáchat velké škody a proto se vyplatí bloky pro generování hodin verifikovat zvlášť důkladně. Zřejmě čím více je v obvodu hodinových domén, tím důkladnější simulace na hradlové úrovni bude třeba provádět.
4. **Strukturní pokrytí**: každý blok v obvodu musí být pokryt alespoň jedním testem spuštěným na hradlové úrovni.
5. **Funkční pokrytí**: Simulace na hradlové úrovni by měly aktivovat všechny módy navrženého systému a všechny způsoby jak mezi nimi přepínat (například u systémů s podporou pro práci v módu se sníženým příkonem – *sleep mode* – vstup a výstup ze *sleep mode*). Dále je doporučené aktivovat v simulacích hlavní funkce celého návrhu.
6. **Pokrytí časových cest**: Simulace na hradlové úrovni by měly pokrýt asynchronní cesty v obvodu (je třeba „procvičit“ všechny přechody mezi všemi hodinovými doménami v návrhu) a dále by měly demonstrovat, že výjimky definované pro statickou časovou analýzu (*timing exceptions in the STA*) nejsou chybně definované. Množství prováděných simulací na hradlové úrovni je také ovlivněno časovým modelem návrhu zachyceným v *constraints*. Pokud jsou *constraints* nekompletní, či nemají „plnou důvěru“ návrhářů, bude lepší provádět hradlové simulace důkladněji. Podobně, skládá-li se obvod z více hodinových domén a časový model je složitý, lze také doporučit provádění většího množství simulací na hradlové úrovni.

Pro usnadnění výběru testů a minimalizace rizika, že část obvodu zůstane spouštěnými simulacemi nepokrytá, autor z vlastní praxe čtenáři doporučuje vytvořit praktickou tabulku například v Excelu – matici pokrytí obsahující na jedné ose jednotlivé důležité funkce systému, na druhé ose vybrané testy. Ta bude názorně ukazovat, který test pokrývá kterou funkci či aktivuje jaký režim návrhu. Závěrem poznamenejme, že ideální stav je v každém případě dosažení maximálního pokrytí – tedy pokud můžete spouštět všechny testy, které spouštíte na RTL úrovni i na hradlové úrovni, činte tak.

2.3 Výběr PVT podmínek

Skutečná rychlost obvodových prvků v číslicovém systému je závislá na třech hlavních faktorech: napájecím napětí (V – *voltage*), teplotě (T – *temperature*) a výrobním procesu (P – *process* – tento parametr zohledňuje variace rychlosti způsobené odchylkami ve výrobním procesu). Protože simulace na hradlové úrovni simuluje reálná zpoždění v obvodu, je třeba zvolit několik vhodných trojic PVT podmínek, pro které budou vypočtena zpoždění logických prvků a spojů, jež se nakonec použijí pro simulaci, více viz [2]. Výběr PVT podmínek pro generování časového modelu obvodu pro simulaci na hradlové úrovni je dán zejména extrémy skutečných fyzikálních podmínek za kterých bude navrhovaný číslicový obvod pracovat v konečném produktu. Rozumné je simulace spouštět alespoň ve dvou rozích:

- **s nejpomalejšími obvodovými prvky** pro kontrolu dodržení předstihu (*setup check*) na sekvenčních prvcích v obvodu. Hodinová frekvence, na které obvod při simulaci běží, by měla být stanovena jako maximální jakou dokáže hodinový zdroj v aplikaci generovat dále zvýšená o rezervu např. 10%. Chceme zde, aby byl celý návrh provozován za mezních podmínek.
- **s nejrychlejšími obvodovými prvky** pro ověření dodržení přesahu (*hold check*) na všech prvcích v obvodu.

Všechny testy spouštěné na hradlové úrovni jsou tedy spouštěny alespoň dvakrát – v rychlém i pomalém rohu; Xilinx dokonce ve svých materiálech doporučuje spouštět všechny simulace čtyřikrát; více detailů viz dokument [7], strana 49

a 50.

3 Verifikační prostředí

Provádění simulací na hradlové úrovni často vyžaduje úpravy verifikačního prostředí použitého pro simulace na RTL úrovni. Úpravy mohou být jak strukturní (často na rozhraní pro *top level* entitu návrhu – tzv. DUT, *Design Under Test*), tak funkční (nejčastěji kvůli problémům s propagačním časem obvodem).

3.1 Rozhraní DUT

RTL syntéza je proces, při kterém je abstraktní RTL popis převeden do konkrétního schématu užívajícího konkrétní buňky v příslušné technologii. Jeden z výsledků syntézy je také odstranění veškeré abstrakce z rozhraní číslicového systému. Přibližme si to na příkladu kódu v obrázku 2.

RTL kód – definice entity	Netlist – interface bloku

`LIBRARY IEEE;`
`USE IEEE.std_logic_1164.ALL;`
`USE IEEE.numeric_std.ALL;`
`ENTITY dut IS`
`  GENERIC (`
`  N : natural := 16`
`  );`
`  PORT (`
`  clk : IN std_logic;`
`  res_n : IN std_logic;`
`  op_a : IN unsigned (N-1 DOWNTO 0);`
`  op_b : IN natural RANGE 0 TO 255;`
`  result : OUT std_logic_vector (N-1 DOWNTO 0)`
`  );`
`END ENTITY dut;`
	`module dut (`
	`  clk, res_n, op_a, op_b, result`
	`);`
	
	`  input clk;`
	`  input res_n;`
	`  input [15 : 0] op_a;`
	`  input [ 7 : 0] op_b;`
	`  output [15 : 0] result;`

Obrázek 2: Příklad definice rozhraní bloku v RTL kódu v jazyce VHDL a implementace skutečného rozhraní bloku v netlistu na hradlové úrovni v jazyce Verilog.

V levém sloupci je uveden příklad definice rozhraní bloku na RTL úrovni v jazyce VHDL. Všechny použité konstrukce jsou bez problémů syntetizovatelné. V pravém sloupci je rozhraní bloku po syntéze v netlistu v jazyce Verilog. Všimněte si, že

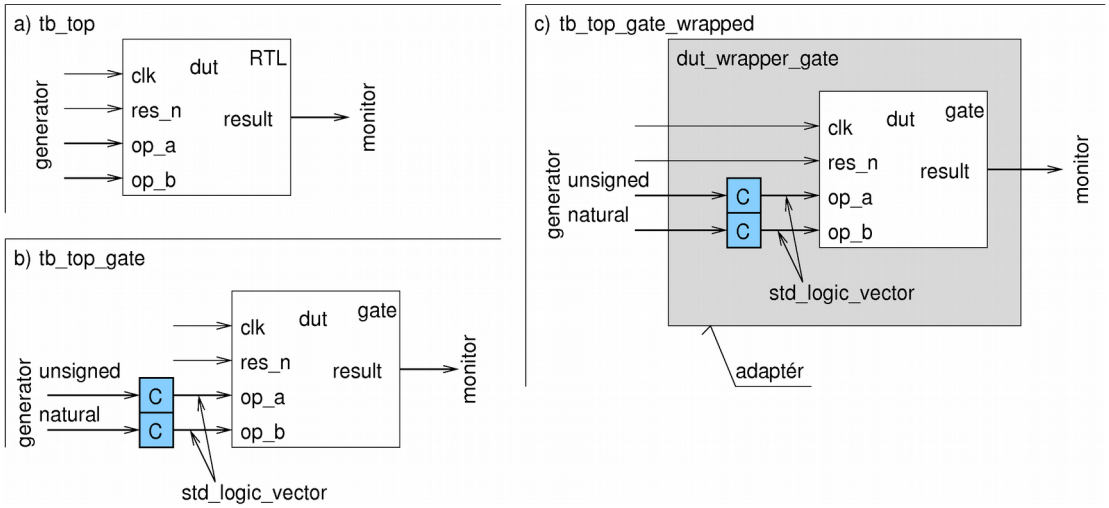
- syntéza odstranila klauzuli *GENERIC* a vstupní i výstupní sběrnice nyní mají pevnou šířku – za *N* bylo všude dosazeno 16.
- vstup *op_a* je před syntézou typu *unsigned*, po syntéze je vstup jen jednoduchá sběrnice (verilogový ekvivalent VHDL definice *op_a: IN std_logic_vector*) o šířce 16 bitů. Na rozhraní bloku *dut* jsou implementovány vstupy a výstupy jen typu *std_logic_vector*, vše na ně bylo převedeno. Stejnou transformací prošel výstup *result*, podobně vstup *op_a* byl převeden z přirozeného čísla v rozsahu 0-255 na osmibitovou sběrnici.

Blok *dut* je ve verifikačním prostředí instancován, jak je znázorněno na obrázku 3a. Změna rozhraní bloku po syntéze nutí návrháře používat různé verze bloku *tb_top* (implementace nejvyšší úrovně hierarchie ve verifikačním prostředí) pro simulace na RTL a hradlové úrovni; změna datových typů vstupů a výstupů bloku vede na nutnost konvertovat interní signály na jiné typy a zpátky, viz obrázek 3b. Doporučit lze následující:

- učitě se nemá smysl vyhýbat použití konstrukce *GENERIC* na rozhraní *dut*. Generické parametry jsou velmi užitečné a jejich použití obvykle stojí za jisté nepohodlí při implementaci nejvyšší úrovně hierarchie ve verifikačním prostředí.
- Na druhou stranu, jiné datové typy, než *std_logic_vector* a *std_logic* na *dut* rozhraní návrhu je dle názoru autora lépe nepoužívat. Jejich použití si vynucuje krom změny rozhraní také implementaci konverzního kódu pro převod signálů mezi jednotlivými typy. Tím vznikají větší odlišnosti mezi verifikačními prostředími pro RTL a hradlové simulace a zvyšuje se riziko zavlečení chyby. Také stoupá pracnost implementace. Jiná situace nastává u vnitřních rozhraní v návrhu, kde naopak abstraktnější datové typy mohou být velmi užitečné.

Změnám na nejvyšší úrovni verifikačního prostředí se s přechodem od RTL k hradlovým simulacím tedy nevyhneme. To je mrzuté, protože blok *tb_top* může být dosti složitý a udržovat jeho dvě téměř identické kopie může být jednak časově náročné a dále to zvyšuje riziko zavlečení chyb do návrhu. Lze proto doporučit implementaci znázorněnou na obrázku 3c, kde je mezi *dut* a *tb_top* vložena další úroveň hierarchie – blok adaptéru *dut_wrapper*. Jeho jedinou funkcí je úprava signálů z rozhraní bloku na RTL úrovni na rozhraní netlistu. Dosáhne se tak dobré dekompozice návrhu, vše co souvisí s přizpůsobením verifikačního prostředí požadavkům rozhraní *dut* je lokalizované v adaptéru (*wrapper*). Ten má dvě verze – jednu pro RTL kód, druhou pro hradlovou implementaci entity. Není pak třeba udržovat dvě kopie bloku *tb_top*.

Používáte-li prostředí Xilinx Vivado, lze dvě verze bloku pro RTL a hradlovou simulaci dosti snadno udržovat pomocí tzv. simulačních setů (*simulations sets*, viz [7], strana 15). Podpora v jiných simulátorech bude odlišná, zde odkážeme na dokumentaci k Vašemu nástroji; pomoci může také použití VHDL konstrukce *CONFIGURATION* (viz např. [9]).

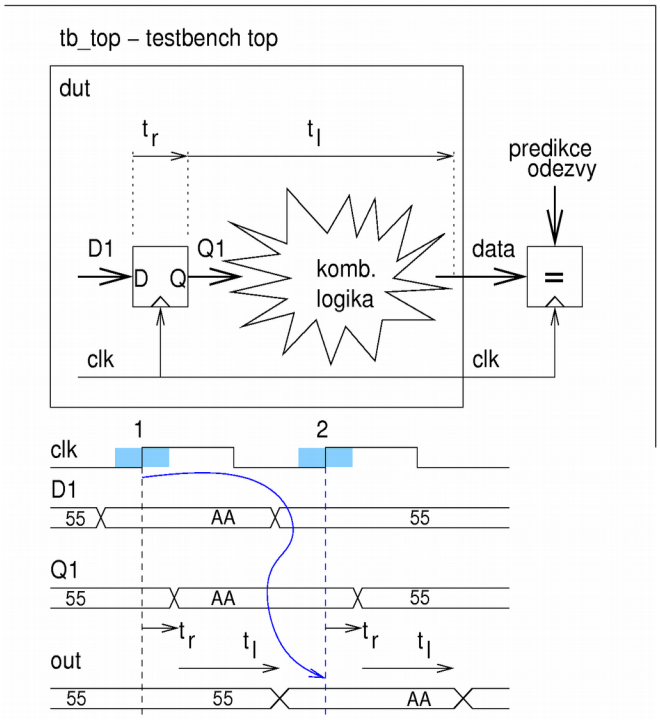


Obrázek 3: Různé možnosti, jak nainstancovat dut *na nejvyšší úrovni verifikačního prostředí (v bloku* tb_top*). a – verifikace RTL verze* dut*. b – verifikace implementace* dut *na hradlové úrovni; pro vstupy* op_a *a* op_b *je třeba implementovat konverze a jsou potřeba dvě verze bloku* tb_top*. c – implementace s pomocí adaptéru/wrapperu, všechny změny spojené s přechodem na hradlové simulace ovlivňují jen jednoduchý blok adaptéru, místo dvou verzí složitého bloku* tb_top *udržujeme dvě verze jednoduchého adaptéru (*dut_wrapper_rtl *a* dut_wrapper_gate*). Soubor tb_top_rtl.eps.*

3.2 Kontrola odezev obvodu

Jednou ze základních funkcí verifikačního prostředí je kontrola správnosti odezev číslicového obvodu na předkládané stimuly. Zatímco na RTL úrovni obvod reaguje okamžitě a vše se děje s „nekonečně rychlou“ odezvou, na hradlové úrovni jsou simulována reálná zpoždění obvodových prvků a obvod tak bude reagovat jako ve skutečnosti, tedy později – viz obrázek 4 s příkladem časování. Dále pokud jsou některé výstupy *dut* generované přímo z kombinační logiky,

můžeme na nich pozorovat přítomnost hazardních stavů (tzv. *glitches*). Z autorovy zkušenosti lze doporučit před implementací automatické kontroly odezev obvodu rozdělit primární výstupy *dut* do těchto skupin (například ve formě tabulky jako je na obrázku 5):

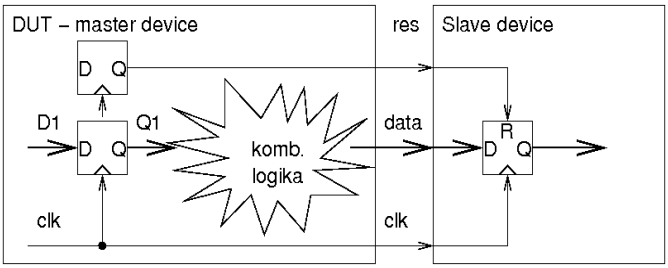


Obrázek 4: Časování výstupů synchronního číslicového systému. Soubor gate_delays_check.eps

1. **Výstupy synchronní s hodinami**, které jsou vně navrhovaného obvodu dále zpracovány s hranou hodin a nevadí na nich přítomnost hazardních stavů. Příklad je například výstup *data* v obrázku 6. U těchto výstupů je třeba externě jen zajistit dodržení předstihu a přesahu (viz modré intervaly v obrázku 4), stačí jejich správné hodnoty kontrolovat jen synchronně s hodinami – tedy musí být stabilní po dobu předstihu a přesahu kolem vzorkovací hrany hodin.
2. **Výstupy, které jsou vně navrhovaného obvodu zpracovány asynchronně** a hazardní stavy na nich být nesmí. Přítomnost hazardních stavů na těchto výstupech způsobí vážné narušení funkce obvodů připojených k výstupům návrhu. Příkladem je výstup *res* v obrázku 6, zákmity na tomto výstupu by způsobily neočekávané resetování podřízeného systému. Přítomnost správné hodnoty na takovém výstupu je třeba monitorovat nepřetržitě, aby byly detekovány případné hazardní stavy.
3. **Hodinové signály generované navrhovaným obvodem** (viz výstup *clk* v obrázku 6) jsou samostatnou kategorií. U hodinových signálů je důležité nejen hlídat, aby byly prosty hazardních stavů, ale také je třeba monitorovat dodržení jejich periody a střídy. Je tedy typicky třeba implementovat ve verifikačním prostředí monitor jejich periody a střídy a současně hlídat zda na nich nejsou hazardní stavy.

Port	Směr	Šířka	Vzorkovací hodiny	Poznámka
clk	out	1	hodinový signál	Minimální frekvence na výstupu 9 MHz, maximální 11 MHz, typická 10 MHz dle oscilátoru. Střída signálu musí být z rozsahu 40% - 60%.
data	out	16	náběžná hrana *clk*	Musí být stabilní s předstihem 5 ns a přesahem 1 ns.
reset	out	1	asynchronní signál	Nesmí se vyskytnout hazardní stavy, signál musí být generovaný přímo z klopného obvodu.
				atd.

Obrázek 5: Příklad formy popisu vstupů a výstupů DUT.



Obrázek 6: Příklad použití různých druhů výstupních signálů vně návrhu. Soubor gate_delays.eps

4 Závěr

Článek shrnul několik doporučení jak postupovat během přípravy simulací na hradlové úrovni a při implementaci verifikačního prostředí, která mohou návrháři ušetřit část problémů se simulacemi na hradlové úrovni. Je zřejmé, že příprava těchto simulací může vést až k částečnému přepracování verifikačního prostředí; autor proto doporučuje na potřebu provádění simulací na hradlové úrovni myslet od samého začátku návrhu a vše implementovat tak, aby potřeba pozdějších změn byla minimalizována. Pomoci k tomu může také provádění zkušebních (tzv. *pipecleaning*) simulací za běhu projektu, kdy RTL kód sice není ani kompletní ani odladěný, ale přesto lze už objevit závažnější problémy a postupně je odladit dříve, než ve stresu před koncem projektu. Kromě problémů s verifikačním prostředím také není vyloučeno, že objevíte závažné problémy v RTL kódu, které by se jinak odhalily až při finální implementaci.

Problematika simulací na hradlové úrovni je značně rozsáhlá a bylo třeba ji vzhledem k omezenému rozsahu textu značně zjednodušit. Čtenáři zde proto doporučujeme studium další literatury, jako inspirace může sloužit seznam použité literatury níže.

5 Použitá literatura

[1] ŠŤASTNÝ, Jakub. Simulace číslicových obvodů na hradlové úrovni. *DPS Elektronika od A do Z,* květen/červen 2015, s 8-11.

[2] ŠŤASTNÝ, Jakub. Simulace číslicových obvodů: model návrhu. *DPS Elektronika od A do Z*, leden/únor 2016, s 6-

12.

[3] ŠŤASTNÝ, Jakub. Simulace číslicových obvodů: triky i úskalí simulace. *DPS Elektronika od A do Z*, březen/duben 2015, s 20-23

[4] GINOSAR, R. Metastability and Synchronizers: A Tutorial. *IEEE Design & Test of Computers*, Vol. 28, No. 5, pp. 23-35. Dostupné též z: http://webee.technion.ac.il/~ran/papers/Metastability-and-Synchronizers.IEEEDToct2011.pdf

[5] ŠŤASTNÝ, Jakub. *FPGA prakticky.* Praha : BEN-technická literatura, 2011. 200 s.

[6] CUMMINGS, Clifford. Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog. In: SNUG 2008 Boston. [vid. 13. prosince 2015] Dostupné z http://www.sunburst-design.com/papers/CummingsSNUG2008Boston_CDC.pdf

[7] XILINX. Vivado Design Suite User Guide Logic Simulation, UG900, verze 2014.2 (4. července 2014)

[8] *Gate Level Simulations : A Necessary Evil - Part 1,2, and 3* [online] [vid. 13. prosince 2015] Dostupné z http://whatisverification.blogspot.cz/2011/06/gate-level-simulations-necessary-evil.html

[9] DOULOS [online]. *Configurations.* [vid. 13. prosince 2015] Dostupné z: https://www.doulos.com/knowhow/vhdl_designers_guide/configurations_part_1/