Návrh specifických struktur na programovatelných hradlových polích

V předchozích dílech volného seriálu jsme postupně seznámili čtenáře s vlastnostmi hradlových polí, nástroji pro návrh a základními jazykovými konstrukcemi jazyka VHDL. V novém díle se zaměříme na správné postupy při návrhu některých specifických logických funkcí, jmenovitě ošetření asynchronních signálů a správnou inicializaci obvodu. V příspěvku jsou uvedena a zdůvodněna základní pravidla, jejichž dodržování vede k návrhu obvodů s bezpečným a deterministickým chováním. Jejich dodržování doporučujeme zejména začátečníkům a méně zkušeným návrhářům. Ušetří si tak mnoho starostí s „podivně" se chovajícími obvody.

POUŽITÉ ZKRATKY

ASIC	*Application Specific Integrated Circuit*
DCM	*Digital Clock Manager*
FIFO	*First In First Out*
LUT	*Look Up Table*
RTL	*Register Transfer Level*
STA	*Statical Timing Analysis*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
WE	*Write Enable*

Úvod

Přestože moderní software pro návrh obvodů na FPGA za návrháře vyřeší mnohé problémy, je automatizován pouze návrh plně synchronních struktur. Správné zpracování asynchronních signálů (například externích vstupů do obvodu) je tak možné zajistit pouze korektním návrhem. Navíc chyby v implementaci struktur pro ošetření asynchronních signálů můžou způsobit nepředvídatelná selhání obvodu a náhodný charakter takových funkčních poruch způsobuje, že se jejich příčina velmi špatně hledá. V tomto příspěvku prezentujeme několik rad, obvodových řešení a elementárních technik, které by mohly začínajícím návrhářům ulehčit jak práci s návrhem, tak s jeho následným laděním. Jejich důslednou aplikací lze předejít zmíněným problémům.

Asynchronní a synchronní signály

Synchronní obvod je obvod, ve kterém se mění stavy všech registrů současně. K jejich změně dochází v důsledku změny speciálního synchronizačního (hodinového) signálu. Předpokládáme také, že vstup každého registru se ustálí do příchodu další změny příslušného hodinového signálu. Synchronním signálem pak rozumíme signál, který se mění jen v důsledku změny hodinového signálu a opět dojde k jeho ustálení do příchodu další změny hodinového signálu. Asynchronním signálem je signál, který se může měnit kdykoliv, a nerespektuje tak pravidlo ustálení do příchodu nejbližší další události na hodinovém signálu. Hovoříme-li o asynchronním či synchronním signálu, udáváme vždy, vůči kterému hodinovému signálu je signál asynchronní/synchronní. Charakteristickými příklady asynchronních signálů jsou signály přivedené na vstupy FPGA obvodu („z vnějšího světa") s neznámým vztahem vůči interním hodinám obvodu, signály buzené z registrů buzených jiným hodinovým signálem s proměnným fázovým posuvem k hodinovému signálu registru, do kterého je signál zaveden, asynchronní reset obvodu a mnohé další. Dále se se zpracováním asynchronních signálů běžně setkáme například při používání a implementaci nejrůznějších komunikačních rozhraní (SPI, UART, I2C atd.).

Hodinové domény

Hodinová doména je část obvodu, jejíž registry jsou buzeny jen jedním hodinovým signálem. O tzv. mezidoménovém přechodu (*clock domain crossing*) pak hovoříme tehdy, když signál buzený z registru v jedné doméně zavedeme na vstup registru v jiné hodinové doméně.

I v případě hodinových domén rozeznáváme domény vzájemně synchronní (signály buzené z jedné domény se stihnou vždy ustálit před příchodem nejbližší události na hodinovém signálu v druhé doméně) a asynchronní, kde jsou signály buzené z jedné domény asynchronní vůči hodinám v druhé doméně. V dalším textu budeme pro zjednodušení předpokládat, že všechny hodinové domény jsou vzájemně asynchronní. V případě synchronních domén lze často implementaci mezidoménových přechodů významně zjednodušit. Pro začátečníky je nicméně lepší uvažovat vždy plně asynchronní případ. Vyvarují se tak nepříjemných chyb. Signál buzený z libovolné hodinové domény zde proto pro zjednodušení výkladu vždy pokládáme za asynchronní vůči všem ostatním hodinovým doménám.

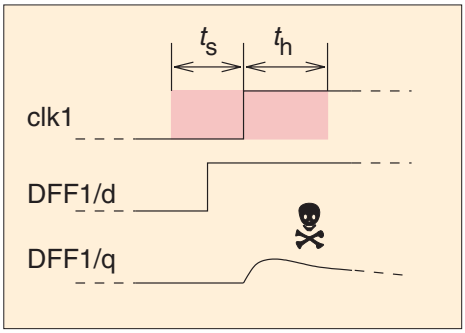
U složitějších obvodů se lze běžně setkat s více hodinovými doménami a mnoha přechody. Charakteristickým příkladem je opět návrh obvodu se sériovým rozhraním typu SPI či I2C. Jednou doménou je zde rozhraní SPI, které pracuje s hodinami sériové linky, druhou doménou vlastní obvod, který používá své systémové hodiny. Spoje mezi oběma bloky (například adresní a datová sběrnice, řídicí signály pro zápis a čtení z rozhraní) přechází hranice hodinových domén. Jiným příkladem je použití více hodinových signálů pro jednotlivé bloky obvodu a jejich vypínání v době, kdy příslušné bloky nepotřebujeme. Dosáhneme tak snížení dynamické spotřeby obvodu. Každou skupinu bloků na jednom hodinovém signálu potom považujeme za jednu hodinovou doménu. V tomto případě jsou dílčí hodinové domény často vzájemně synchronní.

Metastabilita

Zjednodušeně můžeme říci, že metastabilitou registru nazýváme neschopnost jeho výstupu se ustálit na definované logické úrovní v přesně definovaném čase [1], obvykle za jednu hodinovou periodu. U výstupu registru v metastabilním stavu nelze předpovědět chování, podle použité technologie se může metastabilní chování projevovat jen jako krátký zákmit, delší výskyt nedefinované logické úrovně mezi log 1 a 0, oscilacemi či zvětšeným zpožděním registru.

Příčinou metastabilního chování registru je porušení jeho časových parametrů – předstihu (*setup time* – t_s), přesahu (*hold time* – t_h) či zotavení po resetu (*reset recovery time* – t_{rr}). Vznik metastabilního chování demonstruje *obr. 1*. K popsanému jevu dochází nejčastěji, přivedeme-li asynchronní signál na vstup registru – ať už přímo či přes další kombinační logické prvky.

Na závažnost dopadu případného metastabilního chování registru na funkci obvodu má vliv více faktorů, zejména to jsou strmost hrany původního asynchronního signálu, strmost hrany hodinového signálu, který budí příslušný registr, napájecí napětí, teplota obvodu, rychlost logických prvků v obvodu



Obr. 1 Vznik metastability na výstupu klopného obvodu

a jejich vlastní implementace i šum na napájecím napětí. Významný vliv má také hodinová frekvence obvodu – čím vyšší, tím vyšší je riziko negativního dopadu metastabilního chování na funkci obvodu. Zajímavé analýzy metastabilního chování registrů u často používaných FPGA obvodů lze nalézt ve zdrojích [2, 3, 4 a 5]. Výsledky jsou zde uvedeny spolu se statistickými modely pro výpočet pravděpodobnosti vzniku metastabilního chování.

Toto chování může mít mnoho nepříjemných důsledků [1]. Nedefinovaná logická úroveň může způsobit otevření obou tranzistorů v totemové struktuře CMOS hradel, a způsobovat tak proudové špičky na napájení obvodu. Je-li metastabilní výstup registru zaveden do vstupů více logických členů, může každý z nich „vidět“ na vstupu jinou logickou hodnotu. Nedojde-li k odeznění tohoto přechodového děje do příchodu aktivní hrany hodin, obvod může přejít i do nedefinovaného stavu, protože jsou hodnoty v různých registrech v návrhu ovlivněny rozdílně detekovanou logickou úrovní na výstupu metastabilního registru. Přitom za normální situace bez metastability by byly všechny registry ovlivněny stejně.

Jeden z možných nebezpečných projevů metastabilního chování je „zamrznutí obvodu“. K tomu může dojít například tehdy, zavedeme-li asynchronní signál přímo do logiky stavového automatu a metastabilní chování se objeví na výstupu stavového registru. Pak automat může přejít do celkem náhodného či nedefinovaného stavu a „zaseknout se“. Z výše uvedeného vyplývá, že metastabilita je parazitním jevem. Snahou návrháře je udělat návrh proti ní imunní, tj. s dostatečně nízkou pravděpodobností selhání obvodu v důsledku metastabilního chování.

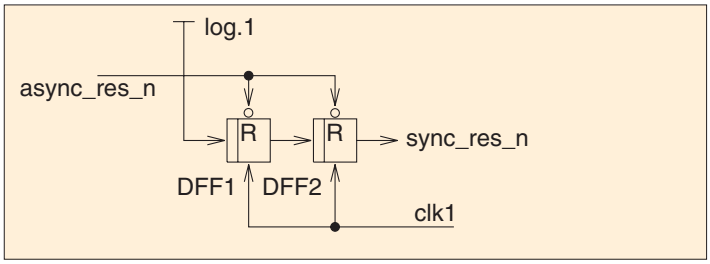
Vidíme tedy, že i prosté použití signálu „z vnějšího světa“ připojeného na pin FPGA obvodu vyžaduje zvláštní ošetření, je-li externí signál zcela asynchronní proti hodinám obvodu, který ho zpracovává. Dále si ukážeme několik základních technik pro potlačení metastabilního chování.

Systémový návrh obvodu

Ve fázi prvotního návrhu systému je důležité aplikovat několik jednoduchých pravidel.

Rozdělte obvod do menších a relativně samostatných bloků podle implementovaných funkcí. Oddělte také řídicí logiku a datovou cestu do samostatných bloků. Pro jednotlivé bloky specifikujte jejich rozhraní i funkce a vytvořte si základní představu o tom, jak budou bloky propojeny na nejvyšší úrovni (tzv. *top level*) návrhu. Návrh pak bude lépe zvládnutelný, případné opravy chyb budou znamenat menší zásahy. Zjednodušíte tak i případnou týmovou práci.

Výstupní signály bloku vždy generujte z výstupu registru. Pokud vstupní signál do bloku nevede přímo z registru, vložte registr i na vstup. Omezíte tak kombinační cesty obvodem vždy jen na jeden blok. To se pozitivně projeví na maximální hodnotě pracovní hodinové frekvence obvodu. Navíc výstupní signál z bloku nebude vykazovat „hazardní“ chování, bude synchronní s hodinami a bude se měnit jen v přesně definovaných časových okamžicích. Tím ulehčíte znovupoužití jednotlivých částí obvodu i integraci s bloky, které třeba navrhli vaši kolegové. Konečně má tato praktika i pozitivní dopad na rychlost běhu syntézy – nástroj pro syntézu bude řešit kratší kombinační cesty. Programovatelná hradlová pole jsou architekturami bohatými na registry, a tak jimi většinou není třeba šetřit.



Obr. 2 Obvodové řešení pro resynchronizaci asynchronního resetu

Inicializace obvodu

Používejte vždy globální reset. Globálním resetem zde rozumíme resetovací (nulovací) signál pro všechny registry v FPGA obvodu. Přestože například u FPGA obvodů s konfigurací uloženou v paměti SRAM jsou registry v matici obvodu automaticky vynulovány během konfigurace obvodu, je v drtivé většině aplikací vhodné mít možnost navržený obvod nastavit do počátečního stavu i jinak, než časově náročnou rekonfigurací FPGA. FPGA má implementované obvody pro rozvod globálního resetu po matici obvodu, a tak použití globálního resetu většinou nepřináší dodatečné zvětšení návrhu. Jsou-li přesto v návrhu bloky neinicializované globálním resetovacím signálem, je vhodné zajistit jejich inicializaci jiným dodatečným signálem.

Synchronizujte asynchronní reset. Je-li asynchronní reset uvolněn příliš blízko aktivní hrany hodin a současně by s hranou hodin mělo dojít ke změně výstupu registru, může dojít k nedodržení časového parametru „doby zotavení po resetu“ (*reset recovery time* – trr). Důsledkem může být opět vznik metastability [6] na výstupu registru. Proto je nezbytné asynchronní reset vždy synchronizovat, a zajistit tak jeho mírné zpoždění po aktivní hraně hodin. K tomu slouží tzv. synchronizátor resetu [6] – *obr. 2*. I zde užíváme konceptu izolace metastability Náběžnou hranu na resetovacím signálu (přechod z log 1 do log 0) synchronizovat nemusíme, metastabilní chování se může objevit jen s uvolněním resetu – s koncem aktivní hladiny na resetovacím signálu (přechod z log 0 do log 1). Je-li asynchronní reset async_r_n uvolněn dost daleko od hrany hodin, nic závažného se neděje. Na výstupu obvodu pozorujeme resetovací pulz o délce dvou hodinových pulzů, změny výstupního signálu následují aktivní hranu hodin po době dané zpožděním registru. Je-li asynchronní reset uvolněn příliš blízko aktivní hrany hodin, může vzniknout metastabilita na DFF1, nikoliv na DFF2 (jeho vstup i výstup jsou v okamžiku uvolnění resetu na stejné hodnotě). Opět spoléháme na to, že do příchodu náběžné hrany hodinového signálu se ustálí výstup DFF1 natolik, že na DFF2 už se metastabilní stav nevyskytne. Uvolnění synchronního resetu sync_res_n i v tomto případě nastane až po aktivní hraně hodin, čímž zabráníme u dalších registrů buzených stejným hodinovým signálem vzniku metastabilního chování.

V případě použití více hodinových domén je třeba buď reset přesynchronizovat do každé domény zvlášť, nebo zajistit zastavení příslušných hodinových signálů aktivací resetu a jejich opětovné rozběhnutí po jeho uvolnění.

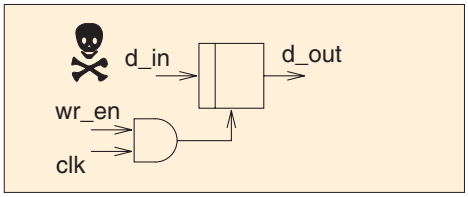
Hodinový signál

Navrhujte plně synchronní obvody. Nepoužívejte nikdy asynchronní sekvenční logické obvody a vyhněte se kombinačním zpětným vazbám. Na programovatelném hradlovém poli nelze zajistit bezhazardní implementaci logických funkcí a přesné zachování obvodové struktury nutné pro správnou funkci navrhovaných asynchronních struktur. I když bude výsledný obvod náhodou fungovat, jeho časování bude nepredikovatelné a chování pravděpodobně nestabilní se změnou teploty či napájecího napětí. Toto pravidlo při návrhu obvodů na programovatelných hradlových polích nikdy neporušujte!

Definujte kolik a jakých hodinových signálů budete používat. Pro každý signál specifikujte také aktivní hranu a definujte vzájemné vztahy mezi jednotlivými hodinovými signály. Vzájemným vztahem může být například to, že hodinový signál clk2 vzniká ze signálu clk1 vydělením čtyřmi pomocí fázového závěsu apod.

Pečlivě rozdělte obvod na jednotlivé hodinové domény, používáte-li více hodinových signálů. Roztřiďte také jednotlivé bloky podle příslušnosti k hodinovým doménám. To znamená podle toho, na které hodinové signály reagují registry v bloku. Speciální pozornost je nutné věnovat signálům mezi bloky, které jsou synchronní k jiným hodinám, než jaké

jsou hodiny použité cílovým blokem. Předem si zjistěte, kolik globálních hodinových signálů umožňuje použít vaše FPGA a udržujte počet hodinových domén menší, než je počet dostupných rozvodů. Méně zkušeným návrhářům doporučujeme použít jen jeden hodinový signál. Dále vezměte při návrhu desky s plošnými spoji v potaz fakt, že pro hodinové signály jsou na pouzdře FPGA obvodu často rezervované speciální piny.



Obr. 3 Zakázaný způsob řízení zápisu do registru

V jednom VHDL modulu používejte vždy jen jeden hodinový signál. Jednotlivé bloky návrhu implementujte vždy jako plně synchronní, snažte se v jednom bloku užívat jen jeden hodinový signál. Pokud je to jen trochu možné, používejte všude registry aktivní na stejnou hranu hodin.

Nehradlujte hodinové signály. Přestože hradlování hodin je velmi silná technika při návrhu zákaznických integrovaných obvodů, na programovatelných hradlových polích nikdy pro řízení zápisu do registrů nepoužívejte konstrukci uvedenou na *obr. 3*. Nedokážete totiž dost dobře ovlivnit zpoždění signálu wr_en (*write enable* – povolení zápisu) a při dostatečně rychlých hodinách se může stát, že místo pěkného pulzu bude na výstupu hradla jen krátká špička. To pak povede na špatný zápis do registru a metastabilní chování na jeho výstupu. Navíc globální hodinový rozvod nelze jednoduše připojit na vstup LUT pro realizaci hradla AND, takže hodinový signál bude propojen pomocí normálních spojů. Fázový posuv mezi hranou hodin na globálním hodinovém rozvodu (signál clk na *obr. 3*) a hranou hodin na vstupu registru za hradlem bude nepredikovatelný. Potřebujete-li vypínat některým blokům v návrhu hodiny (dělá se to obvykle pro snížení spotřeby), použijte k tomu specializované bloky pro spínání hodin (DCM), např. [7]. Ty jsou na většině FPGA obvodů přítomné. Podobně jsou na FPGA dostupné i další dedikované bloky pro dělení a násobení hodinové frekvence či multiplexování hodinových signálů.

Asynchronní jednobitové signály

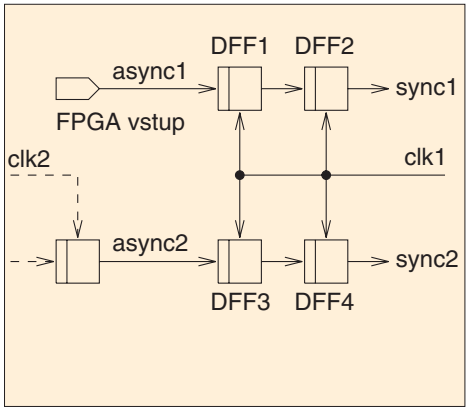
Abychom se při práci s asynchronními signály vyhnuli metastabilitě, používáme synchronizační buňku, která ji izoluje a další logice poskytne už vyčištěný signál (*obr. 4*). Synchronizační buňka je tvořená dvojicí registrů – zde DFF1 a 2, případně DFF3 a 4 v *obr. 4*. První registr (DFF1 a DFF3) přímo vzorkuje asynchronní signál a na jeho výstupu bude docházet k výskytu metastabilního chování. Pokud ale dojde k metastabilnímu chování, hodnota je vzorkována druhým registrem (DFF2 a DFF4 na *obr. 4*) až za jednu celou hodinovou periodu a mezitím dojde k alespoň částečnému ustálení výstupu prvního registru. Druhý registr tak navzorkuje hodnotu už bližší správné ustálené logické hodnotě, a jeho výstup se proto ustálí dříve.

Při používání synchronizační buňky je třeba brát v potaz její nepredikovatelné zpoždění, mohou to být typicky jeden, dva, ale i tři hodinové cykly. Samotná izolace metastability je statistickým procesem a spoléháme na to, že pravděpodobnost vysoce metastabilního chování druhého registru je mizivá.

Druhou podmínkou úspěšného použití resynchronizační buňky z *obr. 4* je dostatečně pomalá změna asynchronního signálu async1 proti clk1, případně $f_{clk2} \leq f_{clk1}$. Při nesplnění této podmínky mohou krátké pulzy na vstupu jednoduše zůstat nezaznamenány.

Pokud potřebujeme zachytit krátké pulzy, můžeme použít řešení naznačené na *obr. 5*. Asynchronní signál async_p připojíme na hodinový vstup registru DFFC (registr reaguje na náběžnou hranu hodin) a na jeho datový vstup přivedeme trvalou logickou jedničku. S náběžnou hranou na signálu async_p dojde k zapsání jedničky do registru. Protože se jedná o událost asynchronní proti clk1, je nutné výstup registru dále převzorkovat a synchronizovat do cílové hodinové domény. Zde – po úspěšném zpracování události – je možné aktivní úrovní na signálu ack vyresetovat celý detektor pulzů, a povolit tak další detekci. Protože signál ack přivádíme přímo na asynchronní resetovací vstup záchytného registru, je nutné ho generovat z registru. Jen tak se s jistotou vyhneme přítomnosti logických hazardů na signálu které by způsobovaly parazitní reset záchytného registru.

Nevýhodou navrženého obvodu je nemožnost detekovat pulzy ve sledu kratším, než je čtyřnásobek periody hodin clk1. V nejhorším případě totiž potřebujeme 3 periody hodin na převzorkování asynchronního signálu a 1 periodu na převzorkování signálu ack.



Obr. 4 Technika resynchronizace jednobitového signálu

Konečně v případě kdy $f_{clk2} > f_{clk1}$ je často výhodné užití korespondenčního režimu (*handshake*).

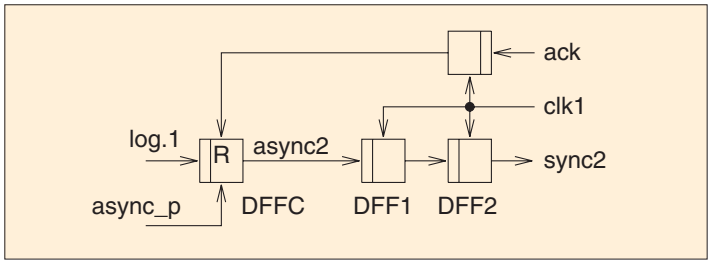
Závěrem našeho stručného přehledu přes techniky synchronizace jednobitového signálu je třeba čtenáře upozornit na to, že na signálech async1 ani async2 na *obr. 4* nesmějí být zákmity. Proto signál procházející přes hranice hodinových domén musí vždy vést z registru do registru, nesmí být buzen přímo kombinační logikou, která může potenciálně generovat logické hazardy. Přítomnost i jednoduchého statického hazardu může mít katastrofální dopad na funkci obvodu. V obecném případě může být tento hazardní stav zachycen a interpretován jako korektní změna logické úrovně (*obr. 6*). Vidíme, že při troše smůly můžeme krátký zákmit na signálu async1 přenést na výstup synchronizační buňky jako pulz o délce celé periody hodinového signálu. Existují i řešení, kterým zákmity nevadí, ale ta jsou již nad rámec tohoto příspěvku.

Vícebitové signály

Významně složitější situace nastává, převádíme-li z jedné hodinové domény do druhé vícebitový (*N*-bitový) signál. Intuitivně bychom řekli, že v dané situaci použijeme *N* synchronizačních buněk, na každý bit jednu. Takové řešení ale nemůže fungovat korektně. Ani na programovatelném hradlovém poli ani na zákaznickém integrovaném obvodu totiž nelze zaručit, že budou zpoždění t0-7 na všech bitech sběrnice zcela totožná (*obr. 7*). Představme si, že se hodnota osmi bitového signálu async_d0 – 7 na obr. 7 mění z 01111111 na 10000000. V závislosti na skutečných zpožděních na vodičích sběrnice můžeme v cílové doméně při „šikovném“ vzorkování navzorkovat v podstatě libovolnou hodnotu mezi 0 a 255.

V případě, že $f_{clk2} \leq f_{clk1}$ je zde řešením použít jednobitový signál, který indikuje platnost dat po změně (viz časové průběhy na *obr. 8*). Signál je přesynchronizován do cílové hodinové domény (registry DFF1 a DFF2 na *obr. 7*) a použit k povolení zápisu do registrů DFFA0 – 7. Popsané řešení se nazývá *recirculation synchronizer* [1]. Na zdrojové straně je ovšem nutné zajistit stabilitu sběrnice async_d po dobu aktivního signálu async_dr a ještě minimálně tři periody hodin clk1 po jeho deaktivaci aby nedošlo ke vzniku metastability na registrech DFFA0-7.

Problém nicméně nastává, bude-li $f_{clk2} > f_{clk1}$. Pak je do systému nezbytné zavést zpětnou vazbu pro zpomalení zdrojové (vysílající) strany. Jedno z vhodných řešení je použití korespondenčního režimu (*handshake*). Komunikace v korespondenčním režimu zajistí, že cílový systém data zpracuje právě jednou, žádná data neztratí ani nezduplikuje. Ideové schéma propojení zdrojové a cílové hodinové domény spolu s časovými průběhy řídících signálů lze nalézt na *obr. 9* [8]. Pořadí



Obr. 5 Obvod pro resynchronizaci a detekci krátkých pulzů

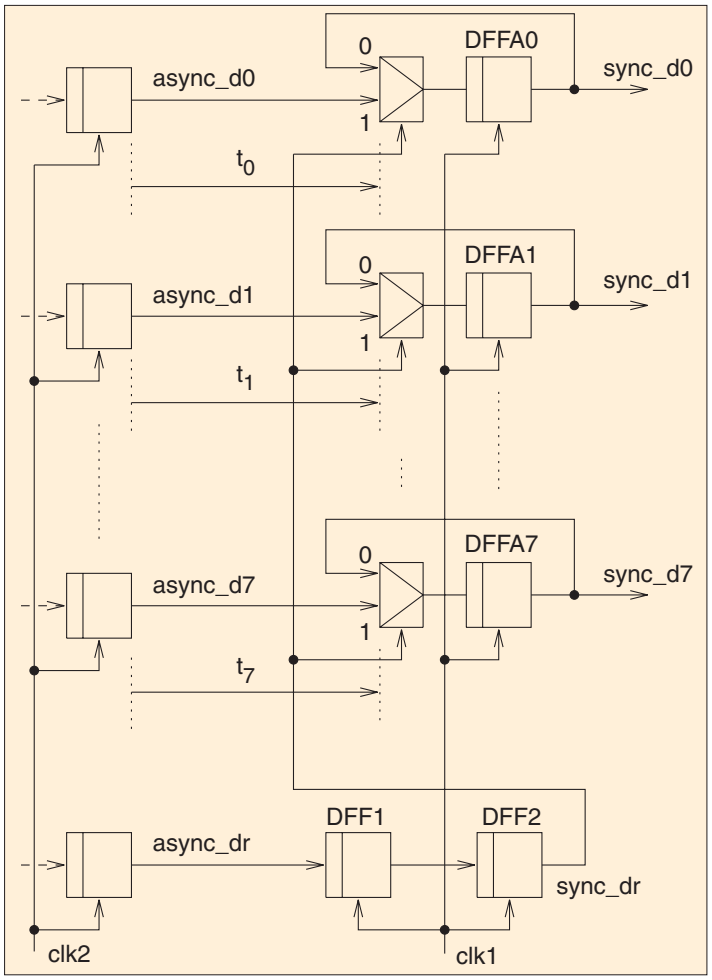
jednotlivých událostí a jejich návaznost jsou na *obr. 9* vyznačeny očíslovanými šipkami:

1. Zdrojová strana vystaví data a aktivuje signál req (*request*) indikující požadavek na zpracování dat.
2. Cílová strana detekuje aktivní signál req, zpracuje data (to trvá vyznačenou dobu t_z) a aktivuje v odpověď signál ack (*acknowledge*) potvrzující příjem dat.
3. Zdrojová strana detekuje aktivní signál ack a deaktivuje signál req; indikuje tak, že zaznamenala úplné dokončení zpracování dat.
4. Cílová strana detekuje deaktivaci signálu req a deaktivuje ack; signalizuje tak, že je připravena přijmout nová data.
5. Zdrojová strana detekuje deaktivaci signálu ack, vystaví nová data a pokračuje bodem 1.

Oba signály req i ack musí být vždy přesynchronizovány do příslušné hodinové domény pomocí resynchronizační buňky.

Nespornými výhodami korespondenčního režimu jsou sladění rychlostí zdrojové a cílové strany a spolehlivost protokolu. Nevýhodou je vyšší režie, která přenos dat poněkud zpomaluje. Existují i další varianty uvedeného komunikačního schématu [8], ale pro začínající návrháře je nejvhodnější použijte už existující implementaci – např. automaticky generované makro pro FPGA Xilinx [9]. Návrh fronty s asynchronními hodinami pro čtení a zápis je problematický a není pravděpodobné, že by ho začínající návrhář dokázal úspěšně zpracovat bez chyb.

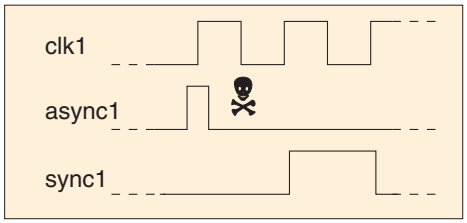
Za určitých podmínek (např. právě při návrhu fronty) je časová režie spojená s resynchronizací řídicího signálu nepřípustná. V takovém případě je možné se uchýlit k řešení s použitím *N* synchronizačních buněk, které jsme v úvodu kapitoly zmínili. Je ovšem nutné dodržet jednu dodatečnou podmínku – na *N*-bitové sběrnici se nesmí nikdy měnit víc jak 1 bit. Více informací je uvedeno v [10].



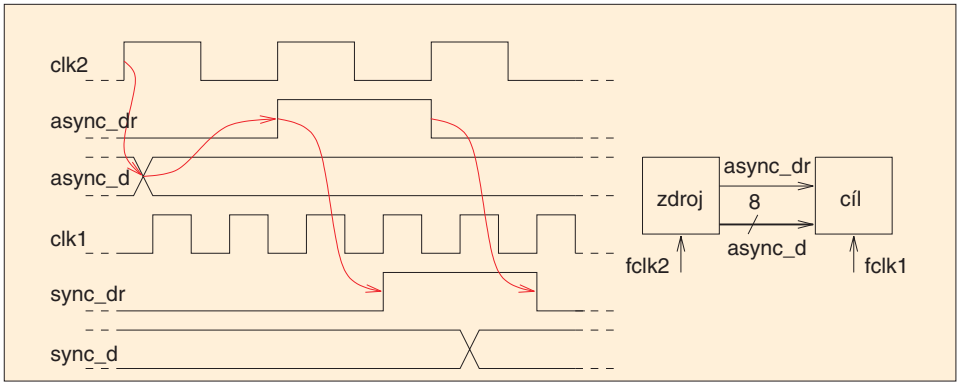
Obr. 7 Nejjednodušší technika synchronizace vícebitové sběrnice na rozhraní hodinových domén

Závěr

S asynchronními signály se v technické praxi setkáváme relativně často. Mohou to být např. externí vstupy do obvodu nebo glo-



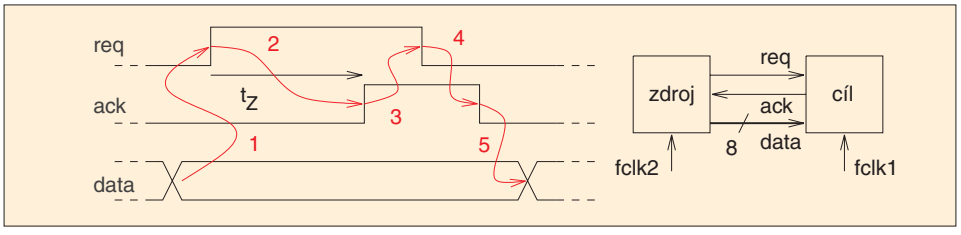
Obr. 6 Hazardní stav na asynchronním signálu nikdy nesmí nastat pokud používáme jednoduchý synchronizér ze dvou registrů (obr. 4)



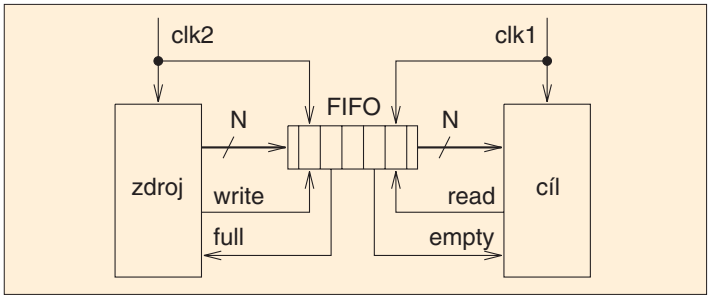
Obr. 8 Časování signálů z obvodu na obr. 7

zentovaná alternativa pro vysokou spolehlivost celé komunikace.

Jinou technikou, jak zajistit korektní předávání dat mezi hodinovými doménami, je použití fronty – FIFO paměti (*obr. 10*). Logický obvod ve zdrojové hodinové doméně bude do fronty ukládat data tak dlouho, dokud fronta nebude signalizovat úplné zaplnění signálem full. Cílový obvod bude číst a zpracovávat data jen dokud signál empty signalizuje, že fronta není prázdná. Frontu ovšem sami v žádném případě nenavrhujte,



Obr. 9 Plně vázaný korespondenční protokol



Obr. 10 Použití fronty pro komunikaci mezi dvěma hodinovými doménami

bální reset. Jejich nesprávné zavedení do logiky může způsobit vážné funkční defekty. Pro problémy způsobené nesprávným ošetřením asynchronních signálů je charakteristické, že se objevují z vnějšího pohledu náhodně a jejich výskyt závisí například na teplotě či napájecím napětí.

Digitální simulace používá abstraktní modely a sama o sobě pro svou správnost vyžaduje dodržení mnoha pravidel, např. právě korektního dodržení předstihu a přesahu klopných obvodů. Dodržení předstihu a přesahu pro plně synchronní obvody kontroluje statická časová analýza (STA), o které jsme se již zmínili v předchozích dílech seriálu. Ani simulací ani pomocí STA obecně nelze zkontrolovat správnost implementace synchronizace asynchronních signálů. Existují specializované nástroje, které podobnou kontrolu dokáží udělat samy, nicméně se domníváme, že jsou pro většinu čtenářů nedostupné a příliš luxusní. Kontrolu správné synchronizace lze pak provést jen ručně. Chybám je nejlepší předcházet už pečlivým plánováním ve fázi systémového návrhu a důkladnou revizí zdrojových kódů na konci návrhu.

V některých jednodušších případech a pro nízké hodinové kmitočty dokonce nemusí být ani ošetření metastabilního chování (izolace metastability) nezbytné. V případě pochybností ovšem doporučujeme popsané techniky používat. Moderní FPGA obvody poskytují dostatek registrů, a tak téměř nemá smysl jimi šetřit a případně riskovat selhávající aplikaci. Návrhářům doporučujeme další studium literatury. Problematika otevřená v tomto příspěvku je velmi široká. Tento text byl věnován jen omezené skupině nejdůležitějších řešení.

Práce byla podpořena výzkumným záměrem MSM6840770012 – Transdisciplinární výzkum v biomedicínském inženýrství 2.

Ing. Jakub Šťastný, Ph.D.
České Vysoké Učení Technické v Praze, katedra teorie obvodů, FPGA Laboratoř ASICentrum, s. r. o.

LITERATURA

[1] VERMA, S., DABARE, A. S., Understanding Clock Domain Crossing Issues. *EDA DesignLine*, 12/24/2007. Dostupné na *www.edadesignline.com/howto/205201913* (zkontrolováno 31. 8. 2008)

[2] Xilinx. Metastable Recovery in Virtex-II Pro FPGAs, Application Note 94.

[3] ALFKE, P., *Metastable Delay in Virtex FPGAs (Xilinx User Community Forums – PLD Blog)*. Dostupné na *http://forums.xilinx.com/xlnx/blog/article?message.uid=7996* (zkontrolováno 31. 8. 2008)

[4] Altera. Metastability in Altera Devices. Application Note 42.

[5] Actel. Metastability Characterization Report for Actel Flash FPGAs. Dostupné na *www.actel.com/documents/Flash_Metastability_HBs.pdf* (zkontrolováno 31. 8. 2008)

[6] CUMMINGS, C. E., MILLS, D., GOLSON, S., Asynchronous & Synchronous Reset Design Techniques – Part Deux. Synopsys User Group, SNUG Boston 2003. Dostupné na *http://amber.feld.cvut.cz/fpga/prednasky/FPGA_navrh/fpga_navrh.html* (zkontrolováno 31. 8. 2008)

[7] Xilinx. Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs, Application Note 462.

[8] STEIN, M., Crossing the abyss: asynchronous signals in a synchronous world. *EDN Magazine*, July 2003. pg. 59–69. Dostupné na *www.edn.com/article/CA310388.html* (zkontrolováno 31. 8. 2008)

[9] FISCHABER, T., OGDEN, J., Never Design Another FIFO. Xcell Journal Third Quarter, 2005. Dostupné na *www.xilinx.com/publications/xcellonline/xcell_54/xc_fifo54.htm* (zkontrolováno 31. 8. 2008)

[10] SARWARY, S., VERMA, S., Critical Clock Domain Crossing Bugs. *EDN Magazine*, April 2008. pg. 55–60. Dostupné na *www.edn.com/article/CA6544743.html* (zkontrolováno 31. 8. 2008)

[11] CREWS, M., YUENYONGSGOO, Y., Practical design for transferring signals between clock domains. *EDN Magazine*, February 2003. pg. 65–71. Dostupné na *www.edn.com/article/CA276202.html* (zkontrolováno 31. 8. 2008)