

Obsah

A. ÚVOD.....	3
A.1. STRUČNÝ POPIS VLASTNOSTÍ NAVRŽENÉHO SYSTÉMU.....	3
A.2. POUŽITÉ NÁVRHOVÉ PROSTŘEDÍ	3
A.3. CÍLOVÝ OBVOD HARDWAROVÉ IMPLEMENTACE.....	3
A.4. POUŽITÁ SYMBOLIKA A KONVENCE.....	3
B. ARCHITEKTURA SYSTÉMU.....	5
B.1. ORGANIZACE PAMĚŤOVÉHO PROSTORU.....	5
B.2. ROZHRAŇÍ PRO HOSTITELSKOU PLATFORMU	6
<i>B.2.1. Časování a funkce signálů rozhraní.....</i>	<i>6</i>
B.3. FUNKČNÍ A POMOCNÉ VNĚJŠÍ SIGNÁLY	8
B.4. VNITŘNÍ USPOŘÁDÁNÍ, KOMUNIKAČNÍ SBĚRNICE.....	9
B.5. HLAVNÍ REGISTRY SYSTÉMU	11
<i>B.5.1. Registr a dekodér adresy bloku.....</i>	<i>11</i>
<i>B.5.2. Hlavní řídicí registr.....</i>	<i>11</i>
C. POPIS JEDNOTLIVÝCH BLOKŮ.....	14
<i>C.1.1. Vnitřní uspořádání společné pro všechny bloky</i>	<i>14</i>
C.2. BLOKY PRO GENEROVÁNÍ STIMULAČNÍCH SIGNÁLŮ	15
<i>C.2.1. Generátor obecné sekvence bitů</i>	<i>17</i>
<i>C.2.2. Generátor šířkově modulovaného signálu</i>	<i>20</i>
<i>C.2.3. Modul osmibitového čítače</i>	<i>22</i>
<i>C.2.4. Modul paměťového generátoru.....</i>	<i>25</i>
<i>C.2.5. Sigma-delta generátor</i>	<i>28</i>
C.3. MĚŘICÍ BLOKY	35
<i>C.3.1. Modul měření obecného jednobitového signálu.....</i>	<i>37</i>
<i>C.3.2. Modul měření obecného osmibitového signálu.....</i>	<i>38</i>
<i>C.3.3. Modul měření PDM signálu.....</i>	<i>39</i>
<i>C.3.4. Modul měření PWM signálu</i>	<i>40</i>
<i>C.3.5. Měřicí blok sigma-delta</i>	<i>43</i>
C.4. POMOCNÉ BLOKY	47
<i>C.4.1. Blok paměťového úložiště</i>	<i>47</i>
D. PROPOJENÍ A SPOLUPRÁCE BLOKŮ.....	50
D.1. UNIVERZÁLNÍ PROPOJOVACÍ MATICE	50
<i>D.1.1. Zajištění spolupráce mezi více systémy.....</i>	<i>52</i>
E. POZNÁMKY K REALIZACI.....	54
E.1. ČASOVÉ NÁROKY	54
E.2. SOFTWARE PRO ŘÍDÍCÍ PC.....	55
F. ZÁVĚR	56
G. SEZNAM POUŽITÉ LITERATURY	57
H. PŘÍLOHA A	59

Použité typografické konvence

Kurzívou jsou psána jména signálů, registrů či bitů v registru, pinů nebo podobné záležitosti související se zdrojovým kódem příp. přímo s hardwarem projektu.

Běžně používaná anglická slova jsou v některých případech připsána bez uvozovek do závorek () za český výraz, protože často bývají natolik zažitým označením, že český překlad je spíše matoucí.

Podtržení v textu má jediný význam a tím je zdůraznění myšlenky či pojmu.

Podobně použití **tučného písma** má za úkol pouze vizuálně zpřehlednit vysvětlovanou pasáž a nemá žádnou zvláštní funkci.

Neproporcionálním písmem jsou provedeny výpisy zdrojového kódu jazyka HDL (Verilogu) příp. i odkazy na zdrojový kód (jména modulů apod.)

Některé použité zkratky

CS.....chip select (signál aktivace čipu)

ABaddress bus (adresová/adresní sběrnice)

DBdata bus (datová sběrnice)

Clkclock (hodinový signál)

IRQ.....interrupt request (požadavek přerušení)

RAM.....random access memory (paměť pro čtení i zápis)

BCD.....binary coded decimal (binárně kódované dekadické číslo)

PWMpulse-width modulation (modulace změnou šířky impulsu)

PDMpulse density modulation (modulace změnou střední hodnoty)

MSB.....most significant bit (nejhornější bit slova)

LSB.....least significant bit (nejspodnější bit slova)

PLL.....phase lock loop (obvod fázového závěsu)

Ostatní pojmy a způsob jejich psaní

Bit (b)binary digit (základní jednotka informace, má pouze dva možné stavy, 0 a 1)

Byte (B)8 bitů (256 možných stavů, tedy 0 až 255)

Signál.....elektrická reprezentace bitu (stavu 0 odpovídá úroveň signálu L, stavu 1 pak úroveň H)

Bitstream.....sekvence bitů (po jednobitové sběrnici)

Bytestream ..sekvence bytů (po osmibitové sběrnici)

Real-timereálný čas, většinou použito ve spojení „režim reálného času“

Části anglické dokumentace

Tabulky a obrázky, které byly vytvořeny pro anglickou dokumentaci tohoto projektu, jsou převzaty beze změn. Jejich obsah a význam je raději podrobně vysvětlen v textu, než aby byly překládány do češtiny.

A. ÚVOD

Zadání této diplomové práce pochází z firmy AMI Semiconductor Czech, s.r.o.

Tento projekt není založen na žádném již existujícím systému, aspoň pokud je autorovi známo; řešení dílčích problémů (jako např. systém sběrnic apod.) jsou samozřejmě založena na obecně známých postupech, většinou však do značné míry upravených pro konkrétní potřeby návrhu.

A.1. Stručný popis vlastností navrženého systému

- obsahuje jednobitové i vícebitové měřicí (vstupní) a stimulační (výstupní) bloky
- otevřená architektura, možnost snadného rozšíření a přidání více bloků
- univerzální i specializované vnitřní bloky pro uspokojení široké škály požadavků
- hostitelský interface navržen a ověřen pro komunikaci s PC přes USB rozhraní
- real-time mód pro měření bez nutnosti přímé účasti PC
- možnost spolupráce více cílových modulů včetně práce v real-time módu

A.2. Použité návrhové prostředí

Byl použit návrhový software ISE Webpack™ (verze 6.0 s postupným přechodem až na verzi 7.1i) firmy Xilinx, který je zdarma dostupný z www.xilinx.com. Celý návrh je popsán v HDL jazyku Verilog a je kompletně k dispozici na přiloženém CD. Simulace byly prováděny pomocí simulačního prostředí ModelSim XE-III (verze 5.8 až 6.0). Ověřování funkčnosti finální implementace bylo uskutečněno na logickém analyzátoru firmy AMI Semiconductor Czech, s.r.o.

A.3. Cílový obvod hardwarové implementace




Vytvořený systém byl implementován do FPGA hradlového pole Spartan-3 XC3S400 firmy Xilinx. Testovací deska byla zakoupena a zapůjčena firmou AMI Semiconductor Czech, s.r.o. a zůstává jejím majetkem. Rovněž tak software pro hostitelské PC umožňující komunikaci s touto deskou byl vyvinut jako součást autorova kontraktu u této firmy a není součástí této diplomové práce.

Celý návrh byl vytvářen tak, aby byl nezávislý na cílovém obvodu; hned na začátku (po diskusi s digitálními designery zmíněné firmy) však bylo rozhodnuto, že fyzická implementace zřejmě zůstane v obvodu typu FPGA kvůli možnosti budoucího rozšíření a přidání dalších vnitřních bloků, pokud se ukáže, že na některé specializované požadavky stávající návrh nestačí.

A.4. Použitá symbolika a konvence

Pro návaznost a orientaci ve zdrojovém kódu uvádím u každého bloku jeho název („jméno entity“), který odpovídá označení příslušného modulu zdrojového kódu. U bloků, u kterých to má smysl, uvádím rovněž „složitost“ v počtu řezů (slices) použitého obvodu FPGA (Spartan3 XC3S400). Tyto řezy jsou do jisté míry základní stavební jednotkou těchto obvodů a obsahují kombinační i sekvenční logické prvky. Celkem jich v FPGA obvodech bývají tisíce, konkrétně zmíněný XC3S400 jich má 3584. Uvedená čísla nemusejí být definitivní, záleží na úrovni a druhu optimalizace (rychlost vs. plocha) kompilačních a syntetizujících programů, které k designu používáme.

V tabulkách paměťových prostorů je použita následující symbolika:

	...paměťové místo přístupné pro zápis i čtení
	...nepoužito
	...paměť pouze pro čtení

Názvy skalárních (jednobitových, jednoduchých) veličin (signálů, registrů, proměnných) ve zdrojovém kódu vždy začínají malým písmenkem, kdežto pro vektorové (vícebitové) veličiny (sběrnice, byty paměti) je striktně dodržováno psaní se začátečním velkým písmenem. Tak bude např. *iRQ* oproti *HiData*.

Adresy jsou psány hexadecimálně, adresa registru v bloku tedy může nabývat hodnot 0, 1, ..., 9, A, B, C, ..., F (což odpovídá dekadickému 0 až 15).

B. ARCHITEKTURA SYSTÉMU

Základní stavební bloky systému lze rozdělit do následujících kategorií:

- Měřicí (vstupní) bloky
- Bloky generující stimulační signály (výstupní bloky)
- Propojovací matice
- Moduly přístupu k blokové paměti RAM
- Blok rozhraní s řídicí platformou (USB kontrolér)

Poznámka: V dalším textu je výraz „blok“ volně zaměňován s výrazem „modul“ a oba označují totéž. Pokud je řeč o bloku paměti RAM, což je fyzická struktura obvodů FPGA, „RAM“ je vždy zdůrazněno, aby nedošlo k záměně.

Všechny bloky jsou vnitřně propojeny sběrnicemi s blokem rozhraní pro vnější ovládání (poslední v seznamu), takže jejich paměťový prostor je kompletně přístupný řídicímu počítači.

Navíc existují konfigurovatelné vazby tzv. okamžitých akcí, kterými mohou měřicí bloky přímo (a okamžitě) ovlivňovat stimulační generátory. To je základem pro tzv. real-time mód práce systému, kdy měření probíhá kontinuálně (s nepřerušovaným hodinovým signálem) bez nutnosti účasti PC (i když tato v principu není nijak omežována).

Na samotné piny použitého FPGA jsou vyvedeny výstupy stimulačních bloků, vstupy měřicích bloků, signály hostitelského rozhraní a volitelně lze některé piny přepnout do funkce signálů pro spolupráci s jinými systémy stejného typu. Dále byly implementovány některé pomocné signály jako výstupy pro informační LED diody příp. tlačítko pro asynchronní reset systému.

B.1. Organizace paměťového prostoru

Datová sběrnice (vnitřní i vnější) má šířku 8 bitů. Paměťový prostor je rozdělen na bloky o velikosti 16 bytů, což je dáno šířkou vnější adresové sběrnice *HiAddr* (4 bity); praxe ukázala, že to je pro většinu případů optimální velikost. Adresa bloku (*AddrHiReg*) má šířku 8 bitů, celkem je tedy k dispozici paměťový prostor pro 256 bloků ($\times 16 \text{ B} = 4096 \text{ B} = 4 \text{ KB} = 32 \text{ Kb}$). Názorný přehled podává tab. 1.

	AddrHiReg	00 (0)	01 (1)	02 (2)	03 (3)	04 (4)	...	FF (255)
HiAddr	0 (0)	System Address Register	System Control Register	Block 1	Block 2	Block 3	...	Block N
	1 (1)		System Control Block					
	2 (2)							
	...							
	F (15)							

tab. 1. Organizace paměťového prostoru

První dva bloky jsou rezervovány pro systémové registry. Na adrese 00 je systémový adresní registr *AddrHiReg*. Tento blok je poněkud zvláštní tím, jakým způsobem se k němu přistupuje – je to totiž ten samý registr, který určuje adresu bloku. Aby byl přístupný, musí být adresa bloku (tedy jeho obsah) nastavena na 0. Pokud do něj zapíšeme cokoliv jiného, ihned budeme přeměrováni do jiného místa paměti. Protokol přístupu k paměti této skutečnosti přímo využívá. Ještě zbývá dodat, že tento registr má velikost 8 bitů, avšak je

zrcadlem do všech 16 bytů bloku – je to jediný registr, který nebere ohled na hodnotu adresní sběrnice *HiAddr*.

Druhý blok s adresou 01 obsahuje hlavní systémové registry – prozatím jen systémový řídicí registr (adresa registru 0). Zbývající místo je rezervováno pro budoucí rozšíření.

B.2. Rozhraní pro hostitelskou platformu

Toto rozhraní se skládá ze 17 signálů, kterými lze zajistit komunikaci s potvrzováním („handshake“) s kontrolérem vyšší vrstvy. Jejich seznam spolu s krátkým popisem je v tab. 2. Jejich význam lze podle jména dobře odhadnout, většina z nich je běžně používaná v aplikacích podobného typu.

jméno signálu	počet ve sběrnici	směr toku dat	popis
hiClk	1	vstupní	hodinový signál
hiCS	1	vstupní	signál "chip select", aktivní v úrovni L
hiRdWr	1	vstupní	read (čtení) v úrovni L / write (zápis) v úrovni H
hiBusy	1	výstupní	signál "zaneprázdněn"
hiIRQ	1	výstupní	signál požadavku přerušení
HiAddr	4	vstupní	sběrnice adresy registru
HiData	8	obousměrné	datová sběrnice

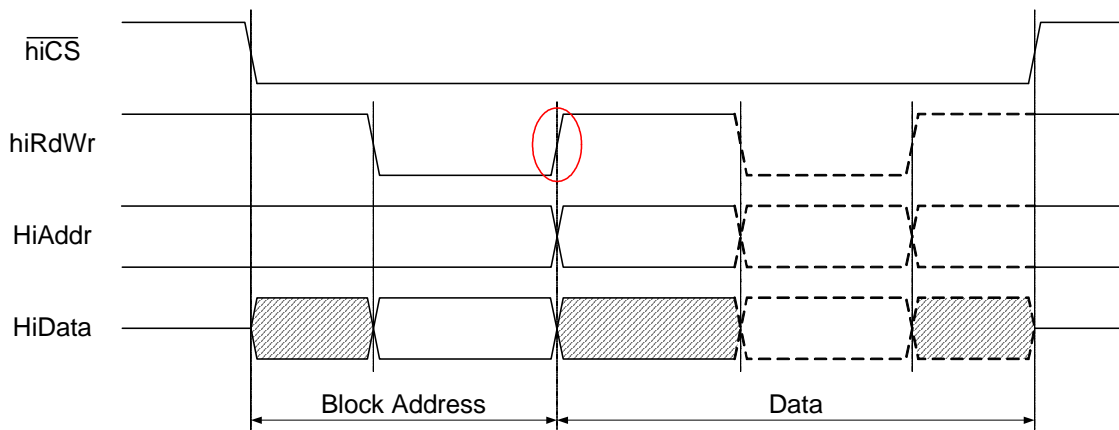
tab. 2. Signály hostitelského rozhraní

Konkrétní podoba tohoto rozhraní (šlo hlavně o počet bitů adresní a datové sběrnice) byla kvůli kompatibilitě přizpůsobena testovací desce, která byla použita k praktické realizaci (více viz kap. E). Toto rozhraní je popsáno v literatuře [2] firmy Opal Kelly, Inc., od které byly tyto moduly zakoupeny.

Signál *hiBusy* je aktuálně nepoužitý, protože všechny transakce jsou vyřízeny během jednoho hodinového taktu.

B.2.1. Časování a funkce signálů rozhraní

Signál hodinového vstupu *hiClk* je použit jako hlavní a jediný hodinový signál pro celý systém. Signály rozhraní jsou synchronní s těmito hodinami (všechna data jsou zachycována při vzestupné hraně *hiClk*), avšak v praxi je toto možné ignorovat pokud je zajištěno, že nedojde k více než jedné změně jakéhokoliv signálu během jedné periody *hiClk* (což bylo při realizaci využito – kontrolér vyšší vrstvy byl mnohem pomalejší a pravděpodobnost, že by stačil „škubnout“ s kterýmkoliv signálem vícekrát během jednoho taktu hodin, byla nulová). Tím je možné vnímat celé rozhraní jako asynchronní a na vstup *hiClk* přivést nezávislý hodinový puls, např. z konfigurovatelného PLL. Pokud je vnější řízení synchronní s jiným hodinovým signálem, pak jedinou podmínkou správné funkce je, že tento puls nebude rychlejší než *hiClk*.



obr. 1. Časování a funkce signálů hostitelského rozhraní

Funkce ostatních signálů vyplývá z obr. 1. Jak již bylo naznačeno, přes datovou sběrnici je nutné nejprve přenést a zapsat adresu bloku, ke kterému chceme přistupovat. Přístupový protokol je tedy navržen tak, že první data, která se zapisují, jsou poslána do systémového adresního registru $AddrHiReg$ – a to bez ohledu na obsah vnější adresní sběrnice $HiAddr$. Vnitřně je to zajištěno tím, že pokaždé, když deaktivujeme signál $hiCS$, bit $dataTransfer$ je vynulován (zmiňuji se o tom proto, že hodnota tohoto bitu je vyvedena jako jeden z pomocných informačních signálů pro LED). Hned potom, jakmile se přepneme do módu čtení, je datová sběrnice přepojena na přenos dat. V obr. 1 je tento okamžik označen kolečkem. Doba zápisu může být libovolně dlouhá, adresa bloku je uměle držena na hodnotě 00, dokud nepřijde nástupná hrana $hiRdWr$ (je tedy ošetřeno nepříznivé chování popisované v kap. B.1)

Na stejném obrázku je po aktivaci bloku signál $hiRdWr$ nastaven nejprve na čtení (úroveň H). To je celkem zbytečné, protože v takovém okamžiku z datové sběrnice nepřečteme nic jiného, než obsah $AddrHiReg$, který musíme stejně při dalším zápisu přepsat.

Postup při vnějším přístupu k rozhraní

1. Rozhraní se aktivuje nastavením $hiCS$ na úroveň L.
2. Pokud tak nebylo učiněno ihned v kroku 1, signálem $hiRdWr$ se přepneme do módu zápisu.
3. Na datovou sběrnici $HiData$ zapíšeme hodnotu adresy bloku, ke kterému chceme přistupovat.
4. Signál $hiRdWr$ nastavíme na úroveň H, čímž se přepneme do módu čtení. V této chvíli je datová sběrnice přepnuta na přenos dat zvoleného bloku.
5. V této chvíli máme přístup k 16 registrům zvoleného bloku. Jednotlivé registry adresujeme čtyřmi linkami vnější adresové sběrnice. Zápis nebo čtení je určen signálem $hiRdWr$.
6. Komunikaci ukončíme vrácením $hiCS$ do úrovně H.

V každém okamžiku komunikace je třeba dodržet podmínky pro manipulaci s obousměrnou datovou sběrnici – při čtení musí vnější zařízení vypnout svoje drivery (výstupní zesilovače). Viz též tab. 3.

CS	hiRdWr	HiData
1	X	High-Z (stav vysoké impedance)
0	1	Input (vstup)
0	0	Output (výstup)

tab. 3. Chování datové sběrnice

Při návrhu přístupového protokolu byl kladen důraz na to, aby doba zápisu nebo čtení nehrála žádnou roli. Není totiž jisté, jak rychle bude vnější zařízení schopno k rozhraní přistupovat. Bylo proto zajištěno, že zápis i čtení může trvat libovolný počet hodinových cyklů a stále je zapisováno resp. čteno ze stejného paměťového místa. Žádné sekvence bytů tedy nejsou požadovány ani podporovány.

Přesné časování (timing) signálů, tzn. doby předstihu a přesahu dat vzhledem k hodinovému signálu apod. jsou dány použitým obvodem FPGA a lze je nalézt v příslušné specifikaci, např. v [1].

V příloze A jsou uvedeny změřené průběhy signálů hostitelského rozhraní, včetně obsahu registru *AddrHiReg* (který byl pro tento účel dočasně vyveden na piny obvodu) pro různé typy přístupů.

B.3. Funkční a pomocné vnější signály

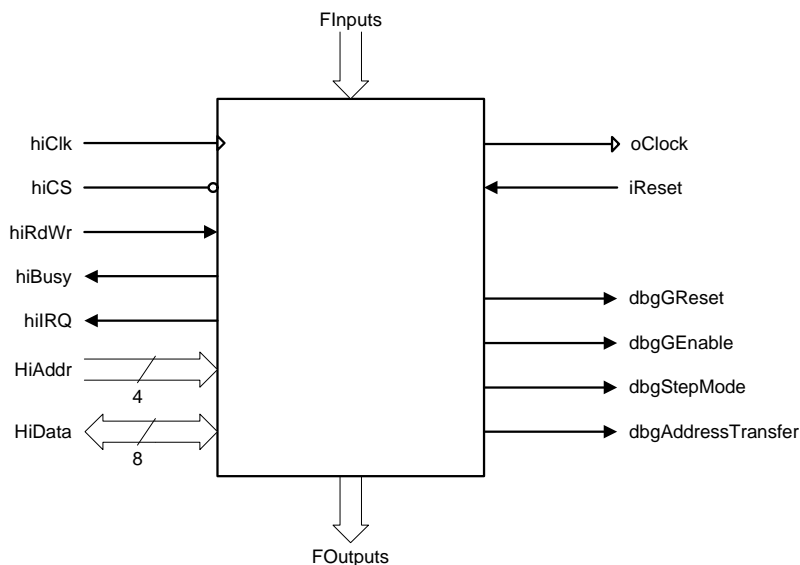
Schéma vstupů a výstupů celého systému je na obr. 2. Kromě hostitelského rozhraní jsou zde již zmiňované pomocné signály pro LED diody a některé další; velmi důležitý je výstup hodin *oClock*, který by měl být použit jako hlavní hodinový signál pro testovaný „slave“ obvod (zapojení, které je testováno naším obvodem). Tento výstup totiž neprodukuje souvislý hodinový signál (pokud nejsme v real-time módu), ale pouze jeden impuls; další se vygeneruje, až k tomu dá souhlas řídicí počítač. Je zajištěno, že tento výstupní signál je bez jakýchkoliv parazitních impulsů (glitch-free), k jeho vytvoření je totiž použita DDR komponenta IO buňky FPGA. Vygenerovaný impuls má tedy přesně stejnou délku, jako vstupní hodinový signál (*hiClk*) a v real-time módu je de facto jeho kopií.

Vstup *iReset* je navržen pro tlačítko asynchronního resetu (aktivní v úrovni L, takže musí spínat na GND).

Funkční vstupy a výstupy jsou označeny *FInputs* a *FOutputs*. Jsou to vyvedené pracovní signály jednotlivých funkčních bloků; mohou být sdíleny se signály pro spolupráci více obvodů. Vždy je však zaručen směr komunikace (vstup vs. výstup).

Pomocné informační signály určené pro připojení k LED diodám jsou aktivní v úrovni H a jsou většinou napojeny na výstupy podobně pojmenovaných bitů systémového řídicího registru. Výjimkou je *dbgAddressTransfer*, který v aktivním stavu indikuje přepnutí datové sběrnice na příjem adresy bloku. Před přivedením na LED diody (pokud jsou zapojeny tak, jak mají být, tedy na napájecí napětí) je tedy třeba tyto signály invertovat.

U realizovaného vzorku byly pro informační účely vyvedeny i další signály na výstupy osazené LED diodami, konkrétně *hiCS* a *hiIRQ* (*hiCS* bez inverze).



obr. 2. Blokové schéma vstupů a výstupů systému

jméno signálu	směr toku dat	popis
oClock	výstupní	výstupní hodinový signál
iReset	vstupní	vstup asynchronního resetu
FInputs	vstupní	vstupy vnitřních bloků
FOutputs	výstupní	výstupy vnitřních bloků
dbgGReset	výstupní	informační výstup pro stav globálního resetu
dbgGEnable	výstupní	informační výstup pro stav globální aktivace
dbgStepMode	výstupní	indikace "step"-módu
dbgAddressTransfer	výstupní	indikace stavu datové sběrnice

tab. 4. Pracovní vstupy a výstupy systému

B.4. Vnitřní uspořádání, komunikační sběrnice

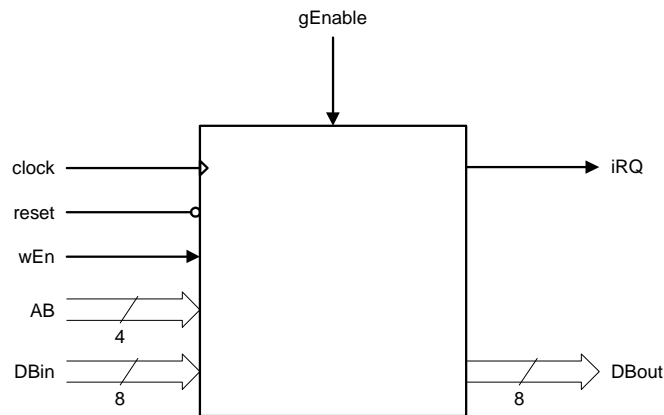
Aby byla zajištěna možnost komunikace řídicího počítače s vnitřními bloky systému, je každý tento blok napojen na vnitřní komunikační sběrnice. Vnitřní datová sběrnice (8b) je totožná s vnější (jsou propojeny 1:1), avšak je uvnitř rozdělena podle směru na *DBin* (z hlediska bloku vstupní) a *DBout* (z hlediska bloku výstupní). Jednak to vede k mírnému zjednodušení při návrh bloků, zejména však je to z toho důvodu, že ne všechna FPGA mají vnitřní třístavové zesilovače (buffery) – použitý Spartan3 byl právě jedním těchto výjimek. Pokud je tomu tak, potom syntezátor HDL kódu provede nahrazení takových struktur kombinačními obvody, nad čímž však designer ztratí kontrolu (zejména pak vznikne nebezpečí, že dva zesilovače budou sepnuty současně). Popsaným rozdělením sběrnice se tomuto elegantně vyhneme. Všechny výstupní datové sběrnice jsou nakonec přivedeny do velkého multiplexoru, který na základě adresy bloku vybere tu správnou.

Dále je ke každému bloku přivedena adresní sběrnice *AB*, která je opět totožná s vnější *HiAddr*. Tato slouží k výběru registru v rámci jednoho bloku.

Zápis obsahu vstupní datové sběrnice do registru na adrese určené obsahem *AB* se aktivuje signálem *wEn* („write enable“, aktivní v úrovni H) a je samozřejmě synchronní. Čtení je aktivní stále a umístění dat na *DBout* je řízeno asynchronně podle *AB* (tj. ihned po změně *AB* je *DBout* přepojena na novou adresu).

Reset je asynchronní a je aktivní v úrovni L. *Clock* je signál hodin a je totožný s vnějším *hiClk*.

Popsané uspořádání sběrnic a řídicích signálů shrnuje obr. 3.



obr. 3. Způsob připojení vnitřních bloků na komunikační sběrnice

Signál globální aktivace

Signál *gEnable* mají všechny funkční bloky (některé pasivní, např. blok datového úložiště, jej nemá). Tímto signálem se společně a současně aktivují všechny bloky, které byly předtím nakonfigurovány do stavu „enabled“ (aktivní) – tj. bit stejného jména v jejich řídicím registru byl nastaven do 1. Po nastavení *gEnable* se všechny takové bloky „rozběhnou“ a začnou provádět svoji funkci. U každého bloku je ověřeno, že ihned po následující vzestupné hraně hodin jsou na výstupu stimulačních bloků platná data, u měřicích bloků jsou touto hranou vzorkovány první vstupní bity (pokud není nastaven bit *delayStart*, jak bude uvedeno dále).

Jestliže v některém okamžiku signál *gEnable* vynulujeme, dojde k zmrazení funkce bloků – jejich výstupy i vnitřní stavové registry zůstanou neměnné na poslední hodnotě. Po opětovné aktivaci *gEnable* budou ve své funkci pokračovat.

Stav zmrazení však nenastane, pokud blok nebyl alespoň jednou po své konfiguraci do „enabled“ stavu aktivován. Jinak je pořád ve stavu „disabled“, tj. neaktivní.

Signál požadavku přerušení

Systém přerušení v tomto projektu nemá s klasickým mikroprocesorovým přerušením (*IRQ*) mnoho společného. Každý blok má výstupní port *iRQ*, kterým signalizuje, že vyžaduje nějaký druh obsluhy ze strany řídicího počítače. Všechny signály *iRQ* od všech bloků jsou potom pospojovány OR hradlem a přivedeny na výstup *hiIRQ* nejvrchnější entity systému. Pokud tedy kterýkoliv z bloků tento signál nastaví, řídicí kontrolér to sice ihned zjistí, avšak tím to končí. Pro řídicí počítač pak neexistuje jiná možnost, než přečíst registry každého bloku a zjistit, který z nich *iRQ* nastavil. Potom, po vynulování příslušného bitu v řídicím registru každého takového bloku, bude i globální *hiIRQ* vynulován.

Tento přístup byl zvolen jako kompromis mezi univerzálností a složitostí celého systému a zdá se být vyhovující. Signál *hiIRQ* pak je využit ještě k jedné funkci, o které bude řeč v následujícím textu.

B.5. Hlavní registry systému

B.5.1. Registr a dekodér adresy bloku

Jméno entity: TopAR

Blok obsahující registr adresy bloku má pevně stanovenou adresu 0 (a je touto adresou aktivován, jako každý jiný blok) a neslouží k ničemu jinému, než k uložení adresy bloku při komunikaci s řídicím počítačem. Na obsah adresní sběrnice se nebere zřetel, dá se tedy říci, že byte adresy bloku je zrcadlen do všech registrů tohoto bloku. Přístup do tohoto bloku byl popsán výše.

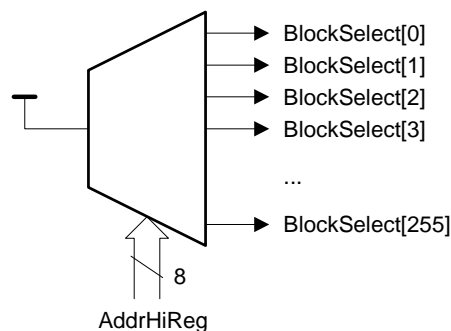
Tento blok nemá vstup pro signál *gEnable* ani výstup *iRQ*, zato má osmibitový výstup, který je uvnitř přímo připojen na registr *AddrHiReg*, a který je pak veden do dekodéru adresy bloku

Dekodér adresy bloku

Jméno entity: TopADec

Dekodér adresy bloku je pouze pomocným modulem pro TopAR a není připojen na žádnou z popsaných komunikačních sběrnic. Je kompletně asynchronní a není ničím jiným, než klasickým adresním dekodérem, tj. velkým demultiplexorem se vstupem konstantně připojeným na aktivní úroveň a řízeným adresou bloku – v našem případě přivedenou z TopAR (obr. 4).

Zmiňuji se o něm proto, že jeho výstupy *BlockSelect[255:0]* jsou přes hradlo AND s negovaným signálem *hiRdWr* přímo napojeny na porty *wEn* jednotlivých bloků. Adresa bloku je osmibitová, takže tento blok má celkem 256 výstupů. Každý výstup je jednoduchou logickou funkcí osmi vstupních proměnných, což by pro všechny obsazené adresy představovalo poměrně dost rozsáhlou kombinační logiku. Ve finální implementaci však dojde k odstranění nepotřebné logiky pro nepoužité adresy, takže náročnost tohoto modulu poklesne.



obr. 4. Dekodér adresy bloku a jeho umístění v systému

Demultiplexor podobného typu je použit pro směrování výstupů vnitřních datových sběrnic *DBout* na *HiData*. Tento je ovšem osminásobný, předchozí poznámky však pro něj platí stejně.

B.5.2. Hlavní řídicí registr

Jméno entity: TopCR

Složitost: 8 slices

Hlavní řídicí registr je jediným registrem hlavního (systémového) řídicího bloku (viz tab. 5). Obsahuje pouze čtyři bity, které však mají globální význam.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0					iRQdis	gEnable	stepMode	gReset
1								
...								
E								
F								

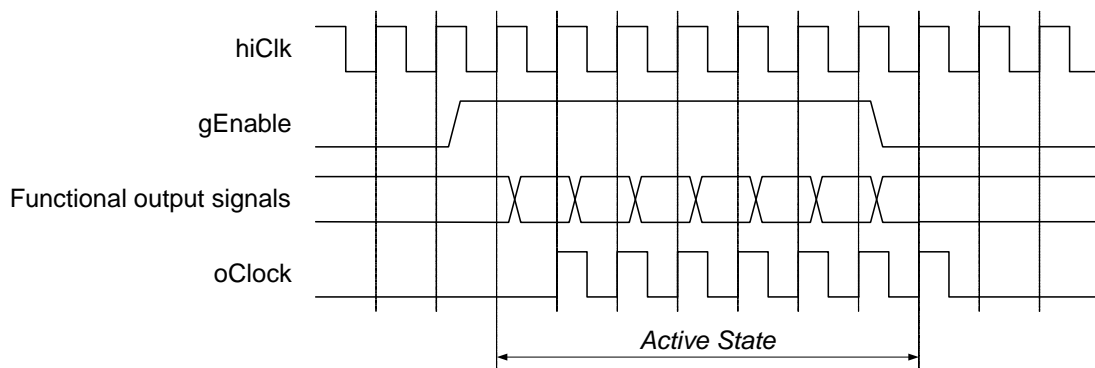
tab. 5. Registry systémového řídicího bloku

Bit *gEnable*

Tento bit je přímo napojen na signál *gEnable* („global enable“), který je zaveden do všech funkčních bloků. Pro detaily k jeho funkci viz kap. B.4, odst. „Signál globální aktivace“.

Tento bit má ještě jednu důležitou funkci: řídí signál *oClock*. Je totiž přiveden na první datový vstup DDR komponenty výstupní buňky signálu *oClock*; na druhém vstupu je konstantně nula. DDR pracuje takovým způsobem, že při vzestupné hraně hodin se na výstup zapíše hodnota prvního datového vstupu, při sestupné hraně pak hodnota na druhém vstupu (což se využívá pro vysokorychlostní komunikaci, pro náš účel je to však také velmi dobře použitelné). Jak již bylo řečeno, tato cesta byla zvolena kvůli čistotě a přesnosti generovaného hodinového signálu. Tímto způsobem je tedy tento signál de facto hradlován tak, že při neaktivním *gEnable* se žádné hodiny negenerují. Pokud tedy *oClock* skutečně použijeme jako řídicí hodinový signál pro testovaný digitální systém, máme jistotu, že i v krokovém (step) módu (viz dále) budou oba systémy správně spolupracovat.

Pozorný čtenář si jistě všiml malé nepřesnosti ve výkladu. Pokud bychom vše provedli tak, jak bylo popsáno, bude na *oClock* vygenerován impuls, avšak bloky v systému se budou teprve „probouzet“ – i ony jsou aktivovány signálem *gEnable* a po vzestupné hraně hodin teprve začnou generovat první bity – které však budou venku vzorkované až při další hraně hodin. Signál *gEnable* je tedy na DDR veden přes registr (flip-flop), čímž je zajištěno požadované zpoždění. Názorné vysvětlení podává obr. 5. Pro naprostou úplnost je třeba ještě dodat, že tento registr je asynchronně resetován signálem *iReset*.



obr. 5. K vysvětlení funkce hodinového výstupu *oClock*

Globální reset

Na vstup *reset* všech bloků kromě bloků nejvyšší úrovně (což je hlavní řídicí registr, registr adresy bloku a registry řídicího rozhraní) je přiveden signál globálního resetu. Bitem *gReset* lze čistě softwarově tento signál nastavit a držet jej neomezeně dlouhou dobu aktivní. Bloky nejvyšší úrovně jsou resetovány pouze vnějším signálem *iReset*; tento signál současně

nastaví i globální reset (globální reset je tedy logickým součinem signálů *iReset* a hodnoty bitu *gReset*).

„Real-time“ vs. „step“ mód

Pod pojmem real-time mód (práce v reálném čase) se rozumí způsob práce systému bez nutnosti zásahu řídicího počítače; v tomto módu se předpokládá práce s nepřerušovaným hodinovým signálem (který je, jak již bylo řečeno, zrcadlen na výstup *oClock*). Tento mód se nastaví vynulováním bitu *stepMode* (a zahájí se aktivací *gEnable*). Všechny bloky potom vykonávají svoji činnost synchronně se svým hodinovým vstupem a, lapidárně řečeno, na nic nečekají, maximálně se „ozvou“ svým *iRQ*.

Funkce „step“ (krokového) módu je možná až překvapivě jednoduchá, ale stoprocentně účinná. Nastavíme-li příslušný bit (*stepMode*), bude v každé periodě hodin bit *gEnable* vynulován. Tak je zajištěno, že každý blok vykoná právě jednu „instrukci“ (sekvenční operaci) a potom zůstane zastaven. Je na řídicím počítači, jak se zachová: Má možnost zkontrolovat řídicí registry všech bloků, případně přečíst vůbec všechny registry všech bloků, nebo jen zkontrolovat *hiIRQ*. Důležité je, že má dostatek času pro vykonání jakékoliv obsluhy. Pro opětovnou aktivaci je třeba znovu nastavit *gEnable*, provede se další instrukce, atd.

Je ošetřeno, že *gEnable* bude v tomto módu po jedné periodě hodin deaktivován i v případě delšího kontinuálního zápisu na jeho adresu (což je při jednoduchém a pomalém kontroléru vyšší vrstvy nevyhnutelné). Abychom jej mohli znovu zapsat, musíme nejprve zrušit zápis do tohoto bloku (po dobu alespoň jedné hodinové periody) a teprve následujícím zápisem můžeme bit *gEnable* znovu nastavit. Zápis je deaktivován buď přepnutím na čtení nebo změnou adresy bloku (k čemuž je však nutné ukončit komunikaci, viz výše).

Mód řízený přerušením

Tento mód je něčím mezi step a real-time módem a aktivuje se nastavením bitu *iRQdis* hlavního řídicího registru. Systém v tomto režimu setrvává v real-time módu činnosti, dokud některý s bloků nesignalizuje svým *iRQ* signálem požadavek na přerušení. Jakmile je tedy detekována aktivní úroveň *hiIRQ*, blok řídicího registru vynuluje bit *gEnable*, a činnost systému je zastavena. Pro pokračování je třeba stejně jako v předchozích případech znovu nastavit *gEnable*.

Pokud je zároveň nastaven i bit *stepMode*, pak je aktivován režim krokového módu.

C. POPIS JEDNOTLIVÝCH BLOKŮ

Požadavky na konkrétní funkce bloků byly zadány firmou AMI Semiconductor Czech, s.r.o. a podle nich byly v dalším textu popisované bloky vytvořeny.

C.1.1. Vnitřní uspořádání společné pro všechny bloky

Každý vnitřní blok má k dispozici paměťový prostor o velikosti 16B (adresa 0 až F), do kterého se z vnějšku přistupuje přes popisované sběrnice. S tímto prostorem lze nakládat v podstatě libovolně; pro usnadnění však byla zavedena jistá pravidla, kterých se budeme držet.

Registr na adrese 0 je označován jako řídící registr bloku (control register, srov. hlavní řídící registr). Jeho obsahem se nastavují hlavní parametry funkce bloku a obecně se blok od bloku liší, v tab. 6 jsou však vyznačeny bity, které jsou pro všechny bloky společné (jejich umístění je rovněž neměnné)

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0		iNserv	iReq					enabled

tab. 6. Uspořádání řídicího registru bloku (společné pro všechny bloky)

Bit *enabled* a funkční stavy bloků

O bitu *enabled* již byla řeč dříve. Jeho nastavením do 1 je blok aktivován a čeká na signál globální aktivace *gEnable*, který spustí činnost bloku. Pokud je tento bit vynulován, blok bude ve stavu „disabled“ i když bude *gEnable* aktivní.

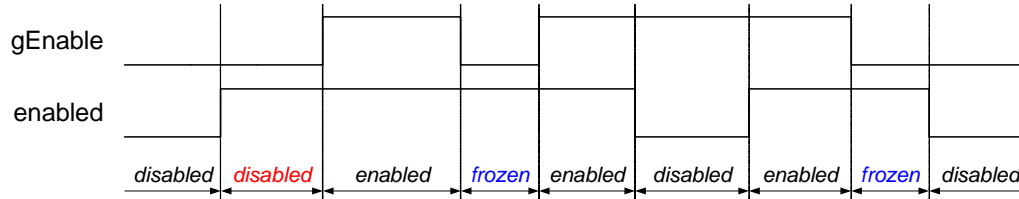
Stav „disabled“ znamená to, že výstupní signály bloku jsou neměnné a pevně určené stavem ostatních konfiguračních registrů. Pokud je bit *enabled* nastaven, blok se v tomto stavu nemůže nacházet. Výjimkou je počáteční nastavení tohoto bitu před příchodem aktivní hrany *gEnable*, kdy je blok ve stavu totožném s „disabled“. Další výjimku může vytvořit požadavek signálu okamžité akce, který může blok dočasně uvést do stavu „disabled“. Více později.

Stav „enabled“ je takový stav, kdy je bit *enabled* nastaven, s výjimkami popsány v předchozím odstavci. V tomto stavu je blok aktivní, tj. jeho výstupy závisí na nakonfigurované funkci a též na časovém okamžiku. Zvláštním stavem je stav „frozen“ (zastaveno), kdy výstup bloku je sice konstantní, ale jeho stav nelze předem určit, závisí totiž na okamžiku, kdy k zastavení došlo. V tomto stavu se blok nachází, pokud je „enabled“, ale signál *gEnable*, který předtím blok alespoň jednou aktivoval, je (dočasně) neaktivní. U měřicích bloků tento stav způsobí přerušování měření.

Pokud je signál *gEnable* aktivní a bit *enabled* je změněn z 0 na 1, pak dojde k okamžité aktivaci bloku (bez stavu „frozen“). Takovému přístupu je však dobré se vyhnout už proto, že většinou nemáme kontrolu nad přesným okamžikem nastavení tohoto bitu a rozhodně jej nenastavíme pro všechny bloky současně. U stimulačních bloků je nicméně garantováno, že se „rozběhnou“ ihned po následující hraně hodin. U měřicích bloků toto zaručeno není, první vzorek, který přijmou, nemusí být ten, který byl na vstupu při první hraně hodin.

Stejná pravidla platí i v případě, že během činnosti bloku změníme některé jeho konfigurační registry (např. hodnotu délky periody u bloku PWM). V takovém případě není vůbec zaručen okamžik, kdy tyto nové hodnoty vstoupí v platnost, a bude se to lišit blok od bloku a případ od případu (např. u již zmíněného bloku PWM to bude až po skončení aktuálně zpracovávané periody) a v některých případech k tomu nemusí dojít vůbec (např. u BPG

generátoru, pokud je nastaven do módu jednorázového impulsu). Je pak nutné nejprve vynulovat bit *enabled* a pak jej znovu nastavit.



obr. 6. Funkční stavy bloků v závislosti na signálu *gEnable* a nastavení bitu *enabled*

Do obsahu vnitřních registrů se popsané stavy promítnou jednoduchým a celkem očekávatelným způsobem, a sice tak, že vynulování bitu *enabled* vynuluje vnitřní pracovní registry (resp. přednastaví je na hodnoty zadané konfigurací), v měřicích blocích se vynulují registry naměřených hodnot.

Z popsaného plyne, že bit *enabled* je dobré nastavit jako poslední před samotnou globální aktivací systému. Ostatní bity v řídicím registru lze změnit současně s ním a i v tom případě je garantována správná funkce (i když to třeba znamená dost podstatný zásah do konfigurace bloku). Což si trochu protiřečí s předchozími doporučeními, ovšem mechanismus zápisu neumožňuje zapisovat jednotlivé bity zvlášť, takže tyto případy musely být ošetřeny.

Bity spojené s přerušením (IRQ)

Jsou to bity *iReq* (interrupt request) a *iNserv* (interrupt not serviced). Bit *iReq* je přímo spojen s výstupem *iRQ* každého bloku a je blokem nastaven v případě, že je vyžadován nějaký druh obsluhy (konkrétně je to popsáno u každého bloku). Tento bit je možné jak číst tak i zapisovat, takže řídicí počítač, jakmile obsluhu provede, by měl tento bit vynulovat (aby byl vynulován i globální *hiIRQ*). Hodnota tohoto bitu není zaručena, pokud by došlo k současnému pokusu o změnu tohoto bitu blokem i vnějším zásahem; předpokládá se totiž, že řídicí rozhraní nejprve zastaví činnost systému vynulováním *gEnable* a teprve potom bude kontrolovat a obsluhovat přerušení.

Bit *iNserv* je pomocný a má praktický význam pouze v real-time módu. Je nastaven tehdy, když blok požaduje už druhé nebo další přerušení v řadě a předchozí ještě nebylo obslouženo (bit *iReq* nebyl vynulován). Tímto způsobem lze zjistit, zda si systém během real-time činnosti vystačil sám nebo dospěl k situaci, se kterou si nevěděl rady – což by pak znamenalo, že je třeba použít jiný mód, např. mód řízený přerušením.

Hodnoty registrů po resetu

Reset asynchronně nastaví všechny registry bez výjimky (srov. též kap. B.5.2 oddíl Globální reset) buď do 0 nebo do 1. V naprosté většině je to 0, výjimek je velmi poskrovnu a jsou v dalším textu zdůrazněny.

C.2. Bloky pro generování stimulačních signálů

Bylo implementováno celkem pět bloků generujících stimulační signály (zkráceně stimulačních bloků) s různým stupněm univerzálnosti. Kromě výše uvedených vlastností společných pro všechny bloky mají stimulační bloky ještě další společné rysy. Těmi hlavními jsou vstupy pro signály okamžitých akcí.

Signály okamžitých akcí

Tyto vstupy mají největší význam v real-time módu a jejich aktivací dosáhneme u stimulačního bloku provedení určité okamžité akce. Jsou určeny k propojení s výstupy

komparátorů měřicích bloků. Filosofie real-time módu je taková, že na základě změřené hodnoty se budou určitým způsobem měnit generované signály a to ihned a bez jakéhokoli vnějšího zásahu.

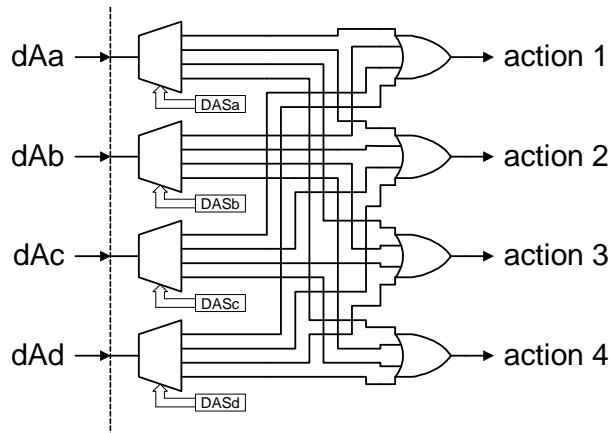
Typ akce je plně v režii stimulačního bloku a je konfigurovatelný. Není pevně určen ani počet akcí, které blok implementuje (jejich počet je však omezen, čtyři jsou pro jeden blok maximum), ne všechny totiž mají pro ten který blok smysl. Následující seznam shrnuje veškeré použité akce; je uveden název (který souhlasí s vnitřním signálem této akce) a vysvětlení:

- `dAdisable`: Dočasně deaktivuje blok, tj. uvede jej do stavu „disabled“.
- `dAfreeze`: Zastaví činnost bloku (má stejný účinek, jako vynulování `gEnable` pro jeden blok).
- `dArestart`: Restartuje činnost bloku.
- `dAchange...:` Způsobí natažení a přepnutí do záložní konfigurační sady bloku. Konkrétní název je doplněn zkratkou konfiguračního registru, který bude vyměněn, tj. např. `dAchangePatt` (change pattern, změní se registr bitového pole), `dAchangeDir` (change direction, změna směru čítání) apod.
- `dAinc...:` Zvýšení generované veličiny o jedničku. Název je opět doplněn o zkratku veličiny, která bude inkrementována, např. `dAchangeW` (change width, zvýšení šířky impulsu).
- `dAdec...:` Snížení generované veličiny o jedničku. Platí stejná pravidla, jako v předchozím odstavci.
- `dAmult2`: Zvýšení generované veličiny na dvojnásobek.
- `dAmult2`: Snížení generované veličiny na polovinu.
- jiné: Některé bloky implementují speciální akce, které budou vysvětleny při popisu funkce těchto bloků.

Těchto akcí není mnoho a k univerzální funkci systému to nestačí. Je to však jakýsi kompromis mezi únosnou složitostí systému a touhou splnit všechny možné i hypotetické požadavky.

Každý stimulační blok má čtyři vstupy pro signály okamžitých akcí, tyto jsou pojmenovány `dAa`, `dAb`, `dAc` a `dAd`. Tyto signály jsou aktivní v úrovni H. Přiřazení konkrétní akce je konfigurovatelné a může být provedeno i tak, že stejná akce je spuštěna dvěma různými vstupy. Vnitřně je toho dosaženo pomocí kombinační sítě zobrazené na obr. 7.

Možná se to zdá jako samozřejmost, ale přesto bude dobré zdůraznit, že tyto akce mění a přepisují konfigurační registry bloku, nebyly tedy zřizovány žádné pomocné pracovní registry. Čtením těchto konfiguračních registrů tak vždy dostaneme správnou aktuální hodnotu.



obr. 7. Konfigurovatelnost vstupů signálů okamžitých akcí

K vlastní konfiguraci je vyhrazen jeden registr paměti bloku obsahující čtyři dvoubitová konfigurační slova *DASa* až *DASd* (direct action selector), viz tab. 7. Každé toto slovo přiřadí příslušnému vstupu akci číslo 1, 2, 3 nebo 4 (tab. 8). V dalším textu u popisu jednotlivých bloků se toto pořadí dodržuje. Pokud je více vstupů připojeno na stejnou akci, jejich účinek se sčítá, tj. pokud aspoň jeden je aktivní, akce je provedena. Dále každý blok obsahuje kontrolní registr, který lze pouze číst a zjistit z něj okamžitý stav jak vstupů tak přiřazených signálů akcí.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
x	DASd		DASc		DASb		DASa	
y	dAchangePatt	dArestart	dAdisable	dAfreeze	dAd	dAc	dAb	dAa

tab. 7. Registry okamžitých akcí

DASx	action
00	action 1
01	action 2
10	action 3
11	action 4

tab. 8. Přiřazení vstupů *dAx* signálům okamžitých akcí

Je zaručeno, že zvolená akce bude vykonána ihned při následující hraně hodin.

Některé akce jsou řízeny staticky aktivní úrovní (H) zvoleného vstupního signálu, některé jsou spouštěny jeho vzestupnou hranou. Např. akce typu restart je typickým příkladem akce spouštěné hranou. Toto je u každé akce pevně dáno a je uvedeno v popisu jednotlivých bloků.

C.2.1. Generátor obecné sekvence bitů

Jméno entity: BPG

Složitost: 206 slices

Šířka výstupu: 1 bit

Modul BPG (Bit Pattern Generator) je navržen pro generování konfigurovatelné příp. náhodné sekvence bitů. Má jednobitový výstup *bitStream*. Zvolená sekvence může být opakována nebo vyslána pouze jednou. Přehled konfiguračních registrů je nejlépe zobrazen na mapě paměťového prostoru tohoto bloku v tab. 9.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	bitStream	iNserv	iReq	pattSel	IFSR	repeatForever	disabledV	enabled
1	PattLen (pattern length)							
2	PattLenCount (current pattern length countdown)							
3	PrescaleF (prescale factor)							
4	PrescaleCount (current prescale countdown)							
5	DASd		DASc		DASb		DASa	
6	dAchangePatt	dArestart	dAdisable	dAfreeze	dAd	dAc	dAb	dAa
7	Pattern0 (32-bit-long bit pattern)							
8	...							
9	...							
A	end							
B	Pattern1 (32-bit-long bit pattern)							
C	...							
D	...							
E	end							
F								

tab. 9. Paměťový prostor bloku BPG

Hlavní konfigurační možnosti

Stav bitu *disabledV* určuje stav výstupu *bitStream* když je blok neaktivní.

Bit *repeatForever* nastavuje, zda se zvolená bitová sekvence bude periodicky opakovat nebo bude vyslána pouze jednou a poté výstup *bitStream* přejde a setrvá ve stavu určeném *disabledV*. Pro opětovné spuštění sekvence je pak nutné buď vynulovat a znovu nastavit bit *enabled* (což bylo již vysvětleno) nebo je možné sekvenci restartovat pomocí okamžité akce *dArestart*.

Bit *pattSel* vybírá primární (ve stavu 0) či sekundární (1) konfigurační registr pro sekvenci bitů.

Pokud nastavíme bit *IFSR* do jedničky, změníme tím funkci bloku na generování pseudo-náhodné posloupnosti. Na stav bitu *repeatForever* pak už není brán zřetel, náhodná posloupnost je nekonečná (nebo se jí aspoň blíží).

Konfigurace vlastní sekvence bitů

Sekvence bitů je čtena z 32bitových registrů *Pattern0* (primární) a *Pattern1*. Který z nich je použit, je určeno bitem *pattSel*. Délka generované posloupnosti je volitelná (je však přirozeně omezena na 32) a nastavuje se registrem *PattLen*.

Registr *PrescaleF* umožňuje rozšířit trvání jednoho generovaného bitu sekvence na více hodinových period. V základním stavu, kdy je tento registr vynulován, je vysílán jeden bit v každém hodinovém taktu. Pokud nastavíme nenulovou hodnotu, doba vysílání jednoho bitu bude rozšířena na $PrescaleF + 1$ hodinových taktů.

Funkce náhodného generátoru

Náhodná resp. pseudo-náhodná posloupnost je generovaná pomocí mechanismu LFSR (linear-feedback shift register) popsáném např. v [11]. Je založen na použití posuvného registru, přičemž při každém posunutí se na uprázdněné místo vloží hodnota určená pomocí hodnot stávajících bitů vztahem (1)

$$(1) \quad \text{BitPattern}[32] = \overline{\text{BitPattern}[31] \oplus \text{BitPattern}[21] \oplus \text{BitPattern}[0]},$$

kde *BitPattern* je název použitého 32bitového posuvného registru (který má bity 0 až 31, takže v označení Verilogu `BitPattern[31:0]`) a zápisem *BitPattern[32]* je označen nový bit, který při posunutí registru přijde na místo bitu 31. Znak \oplus je použit pro logickou funkci exkluzivního součtu (XOR); protože je výsledek navíc negován, je ve skutečnosti provedena operace negovaného exkluzivního součtu XNOR. Tímto způsobem lze dostat pseudonáhodnou sekvenci, která bude mít periodu $2^n - 1$, pro $n = 32$ tedy 4.294,967.295 bitů.

Bit *IFSR* efektivně vypíná čítač délky sekvence, čítač prescalingu (rozšíření doby vysílání jednoho bitu) však zůstává funkční.

Registry pro kontrolu stavu bloku

Tyto registry jsou je možné pouze číst (v tab. 9 psány na černém pozadí) a jejich stav reflektuje stav bloku. V registru *PrescaleCount* je aktuální stav čítače rozšíření doby vysílání bitu (prescaling); tento čítač je po vyslání každého nového bitu nastaven na hodnotu *PrescaleF* a další bit je vyslán vždy až tento čítač doběhne do 0.

Podobně funguje i *PattLenCount*, který však při doběhnutí na do nuly znovu načte zvolenou sekvenci do pracovního posuvného registru a pokud je *repeatForever* v jedničce, začne se tato sekvence vysílat od začátku.

Poslední bit *bitStream* v řídicím registru prostě jen zrcadlí stav stejnojmenného výstupu bloku.

Okamžité akce bloku BPG

(pořadové číslo koresponduje s požadovaným nastavením registrů DASx, viz tab. 8 a pozn.)

1. dAfreeze

- ovládáno staticky

Pozastavuje činnost bloku, viz výše.

2. dAdisable

- ovládáno staticky

Dočasně vypne funkci bloku. Tato akce má za následek přepnutí výstupu *bitStream* do hodnoty *disabledV*.

3. dArestart

- spouštěno hranou

Spustí zvolenou sekvenci od začátku, bez ohledu na aktuální stav. Tato akce bude fungovat i v případě nastavení pouze jednorázového vyslání sekvence (*repeatForever* vynulován) a i po jejím doběhnutí.

4. dAchangePatt

- spouštěno hranou

Způsobí výměnu konfiguračního registru sekvence. Tato akce neguje bit *pattSel* a restartuje sekvenci. Pokud v současném okamžiku zapisujeme do řídicího registru bloku (tzn. pokoušíme se bit *pattSel* přepsat), pak tato přímá akce dostane přednost. V dalším hodinovém taktu však už k přepsání dojde, protože tato akce je spouštěna hranou, tzn. bude aktivní pouze jednu hodinovou periodu.

Požadavek přerušení

Požadavek přerušení nastane při spuštění okamžité akce `dAchangePat t`. Je to proto, že tato akce změní aktivní konfiguraci z primární na sekundární a další její spuštění by přepnulo opět na primární sekvenci. Předpokládá se, že primární sekvence by mezi tím měla být nahrazena jinou (terciální, pokud chceme pokračovat v zavedené terminologii) a přerušení je mechanismem, kterým dá tento blok najevo, že toto nahrazení očekává.

To může dost dobře fungovat i v plném real-time režimu. Pokud je totiž mezi spuštěními `dAchangePat t` dostatek času, může řídicí počítač stihnout konfiguraci vyměnit. O tom, že to skutečně stihnul, že přesvědčí přečtením bitu `iNserv` (který bude v kladném případě nulový).

C.2.2. Generátor šířkově modulovaného signálu

Jméno entity: PWM

Složitost: 168 slices

Šířka výstupu: 1 bit

Funkce generátoru PWM bitstreamu je jednoduchá, samotný blok PWM až tak jednoduchý není. Výstupní signál `pwmStream` je aktivní po dobu určenou hodnotou šířky impulsu a neaktivní po zbytek periody (impuls tedy začíná hned na začátku periody), viz též obr. 13. Konkrétní stav aktivní a neaktivní úrovně je konfigurovatelný. Délka periody je určená šestnáctibitovým číslem, může tedy dosáhnout maximální hodnoty 65.536. Následující tabulka (tab. 10) opět přináší přehled všech registrů bloku.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	<code>pwmStream</code>	<code>iNserv</code>	<code>iReq</code>		<code>widthSel</code>	<code>repeatForever</code> [†]	<code>invert</code>	<code>enabled</code>
1	PeriodLen (PWM period length)							LSB
2	MSB							...
3	PeriodLenCount (current period length countdown)							LSB
4	MSB							...
5	Width0 (primary PWM width length)							LSB
6	MSB							...
7	Width1 (secondary PWM width length)							LSB
8	MSB							...
9	WidthCount (current width countdown)							LSB
A	MSB							...
B	DASd		DASc		DASb		DASa	
C	<code>dAchangeW</code>	<code>dAdecW</code>	<code>dAincW</code>	<code>dAdisable</code>	<code>dAd</code>	<code>dAc</code>	<code>dAb</code>	<code>dAa</code>
D								
E								
F								

[†] read-only when enabled active

tab. 10. Paměťový prostor bloku PWM

Konfigurační možnosti bloku

Pokud je bit `invert` vynulovaný, aktivní úroveň výstupu je H, jinak je to úroveň L.

Bit `repeatForever` má naprosto stejnou funkci jako u bloku BPG. Pokud jej nastavíme, bude se PWM bitstream generovat kontinuálně, jinak bude vygenerována pouze jedna perioda. Pro zaručení správné funkce bloku nelze tento bit zapisovat, pokud je `enabled` aktivní (zápis je tedy povolen pouze pokud `enabled` je trvale nulový nebo jej nastavujeme z nuly na jedničku nebo naopak).

Bit *widthSel* vybírá primární (ve stavu 0) či sekundární (1) konfigurační registr pro hodnotu šířky impulsu. Perioda je dána pevně jediným konfiguračním registrem.

Konfigurace PWM pulsu

Periodu generovaného PWM pulsu nastavujeme šestnáctibitovým registrem *PeriodLen*. Šířka aktivního impulsu v rámci periody je určena buď primárním registrem *Width0* nebo sekundárním *Width1* v závislosti na nastavení bitu *widthSel*.

Samotná hodnota délky impulse resp. periody je pak o jednu větší, než číslo v těchto registrech, nula tedy odpovídá šířce jedné hodinové periody, 255 potom 256násobku periody hodin apod.

Pokud nastavíme šířku impulsu větší, než délku periody, pak impuls bude trvat po celou dobu periody, tj. výstup *pwmStream* bude trvale v aktivní úrovni.

Registry kontroly stavu bloku

Z registrů *PeriodLenCount* a *WidthCount* je možné přečíst aktuální stav čítačů periody a šířky. Tyto čítače jsou na začátku periody přednastaveny na příslušné hodnoty (*PeriodLen* a *Width0* resp. *Width1*) a čítají směrem dolů. Při dosažení nuly je změněna úroveň výstupu. Hodnota čítače šířky impulsu *WidthCount* je tedy v neaktivní části pulsu nulová.

Bit *pwmStream* je opět přímo napojen na stejnojmenný výstupní port a lze jej samozřejmě pouze číst.

Okamžité akce bloku PWM

(pořadové číslo koresponduje s požadovaným nastavením registrů DASx, viz tab. 8 a pozn.)

Zde je nutné poznamenat, že žádná z okamžitých akcí nemění délku periody a celá perioda je vždy kompletně dokončena. Z toho plyne, že účinek signálů okamžitých akcí se zde nemusí projevit a většinou ani neprojevívá ihned v dalším hodinovém taktu. To je sice výjimka z uvedených pravidel, ale lépe to odpovídá charakteru činnosti bloku.

1. dAdisable

- ovládáno staticky

Dočasně vypne funkci bloku. Tato akce má za následek přepnutí výstupu *pwmStream* do neaktivní úrovně (L pokud je *invert* nulový, H pokud je *invert* 1). Tato akce představuje výjimku z právě popsání chování, protože účinkuje okamžitě.

2. dAincW

- spouštěno hranou

Způsobí zvětšení šířky impulsu o jedničku. Pokud již nelze šířku zvětšit, k žádné akci nedojde a hodnota příslušného registru zůstane na svém maximu (65.535). Změny se projeví v následující periodě PWM bistreamu.

3. dAdecW

- spouštěno hranou

Způsobí zmenšení šířky impulsu o jedničku. Pokud již nelze šířku zmenšit, k žádné akci nedojde a hodnota příslušného registru zůstane na svém minimu (0). Změny se projeví v následující periodě PWM bistreamu.

4. dAchangeW

- spouštěno hranou

Dojde k výměně konfiguračního registru šířky. Tato akce neguje bit *widthSel*. Pokud v současném okamžiku zapisujeme do řídicího registru bloku (tzn. pokoušíme se bit *widthSel* přepsat), pak tato přímá akce dostane přednost. V dalším hodinovém taktu však už k přepsání dojde, protože tato akce spouštěna hranou, tzn. bude aktivní pouze jednu hodinovou periodu. Změny se projeví v následující periodě PWM bistreamu.

Požadavek přerušení

Požadavek může nastat ve třech případech. Prvním je spuštění okamžité akce *dAchangeW* a to ze stejných důvodů jako u bloku BPG (viz výše). Zbývající dva případy mohou nastat při provedení akcí *dAincW* a *dAdecW* a to tehdy, když je dosaženo limitů šestnáctibitového čísla, tedy 0 při snižování a 65.535 při zvyšování velikosti šířky impulsu. Přerušení tedy nastane, když je požadováno snížení z 1 na 0 nebo zvýšení z 65.534 na 65.535.

C.2.3. Modul osmibitového čítače

Jméno entity: Counter

Složitost: 170 slices

Šířka výstupu: 8 bitů

Tento modul je do značné míry univerzálním osmibitovým čítačem. Oproti předchozímu bloku je celkem jednoduchý (i jeho paměťový prostor je zcela prázdný, viz tab. 11), ale jeho funkčnost je poměrně složitá. Umožňuje konfigurovat začátek, konec i diferenci čítání, může pracovat jako jednosměrný nahoru či dolů nebo směr čítání střídat. V základní konfiguraci je čítačem binárním, lze jej však nastavit též do módu čítače v Grayově nebo BCD kódu.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	intE	iNserv	iReq	bCDadjust	outGray	bidir	direction	enabled
1	StartByte (start number)							
2	Delta (increment/decrement)							
3	EndByte (end number)							
4	Count (current counter count)							
5	PrescaleF (prescale factor)							
6	PrescaleCount (current prescale countdown)							
7	DASd		DASc		DASb		DASa	
8	dAchangeDir	dArestart	dAdisable	dAfreeze	dAd	dAc	dAb	dAa
9	Byte (current output byte)							
A								
B								
C								
D								
E								
F								

tab. 11. Paměťový prostor bloku Counter

Konfigurační možnosti bloku

Bit *direction* určuje směr čítání, 0 znamená čítání směrem nahoru (tedy zvyšování čísla), 1 čítání směrem dolů.

Bitem *bidir* je určeno chování čítače při dosažení konce čítání. Nula v tomto bitu má za následek skok zpět na počáteční číslo a zachování směru čítání, jednička způsobí změnu směru čítání. To se projeví též negováním bitu *direction*. Pokud do tohoto bitu ve stejném okamžiku zapisujeme, dostane zapisovaná hodnota přednost. Zde je však další důvod a

potvrzení toho, že bychom se měli vyvarovat zápisu do konfiguračních registrů bloků, které jsou spuštěné: Jestliže totiž budeme trvale zapisovat jedničku do bitu *bidir* a např. jedičku do *direction* a čítač v dalším taktu dosáhne konce (= začátku) čítání, na základě *bidir* se změní směr čítání (a další hodnota čítače bude o diferenci větší), ale bit *direction* nebude moci být změněn, takže v následujícím taktu se bude opět čítat dolů. Výsledkem bude, že výstup čítače bude oscilovat mezi dvěma hodnotami, jednou bude startovní byte a druhou hodnota o diferenci větší.

Bits *outGray* a *bCDadjust* nastavují režim čítání v Grayově nebo BCD kódu. Pokud budou nastaveny současně, na výstupu budou nesmyslné hodnoty.

Bit *intE* zakazuje (1) či povoluje (0) generování požadavku přerušení.

Konfigurace hodnot čítače

Počáteční byte je určen registrem *StartByte*, koncový byte pak registrem *EndByte*. *Delta* určuje diferenci čítání, tedy hodnotu, která se k aktuální bude přičítat nebo odečítat. Konec čítání je detekován při shodě aktuální hodnoty s *EndByte* pokud je směr čítání nahoru nebo *StartByte* při čítání dolů. Je tedy vždy třeba zadat takovou hodnotu, kterou čítač musí projít – jestliže zadáme např. *StartByte* i *Delta* sudé, ale *EndByte* liché, pak konec čítání nikdy nenastane. Aktuální hodnotu čítače lze přečíst z registru *Count* (ne *Byte*!). Všechny tyto hodnoty jsou vždy binární, i v módech Grayova resp. BCD kódu.

Registr *PrescaleF* funguje stejně jako v bloku BPG a rozšiřuje dobu trvání jedné hodnoty na výstupu čítače na více hodinových taktů. Doba vysílání jednoho bytu je tedy *PrescaleF* + 1 hodinových taktů. To s sebou nese zajímavou možnost rozšíření, kdy můžeme použít dva tyto osmibitové čítače a vytvořit z nich jeden šestnáctibitový. Oba nakonfigurujeme stejně, pouze u čítače horních 8 bitů (MSB) nastavíme *PrescaleF* na 255.

Obsah registrů *StartByte* (při *direction* nulovém) nebo *EndByte* (pro *direction* v jedničce) s případnou konverzí na Grayův kód určuje výstup bloku ve stavu „disabled“.

Registry kontroly stavu bloku

Tyto registry jsou celkem čtyři, dva z nich již byly popsány dříve. Zbývá tedy *Byte*, který je přímým zobrazením výstupu bloku a *Count*, což je stav vnitřního binárního čítače. Jak bude ještě dále vysvětleno, tyto registry spolu nemusí souhlasit.

Mód čítání v Grayově kódu

Výstup v Grayově kódu docílíme nastavením bitu *outGray* v řídicím registru. Vnitřně se práce čítače nezmění, vnitřní čítač je totiž vždy binární. Výstup v Grayově kódu je docílen kombinační logikou těsně před výstupem bloku. Grayův kód je z binárního převáděn podle známého vztahu (2)

$$(2) \quad GrayByte = (Count \gg 1) \oplus Count ,$$

přičemž *Count* označuje binární hodnotu, *GrayByte* výsledný byte v Grayově kódu, \oplus je použito pro označení operace exkluzivního součtu (XOR) a symbolem \gg je naznačen bitový posuv vpravo.

Startovní a koncový byte čítače, jak již bylo zdůrazněno, jsou vždy v binárním kódu. Zpětný převod požadovaného Grayova výstupu nelze popsat jednoduchým vzorečkem a je třeba jej provést např. podle algoritmu uvedeného v [13], pro osmibitové číslo jej lze zapsat následující sekvencí operací (3):

$$\begin{aligned} & Count = GrayByte \oplus (GrayByte \gg 4) \\ (3) \quad & Count = Count \oplus (Count \gg 2) \\ & Count = Count \oplus (Count \gg 1) \end{aligned}$$

Po provedení všech tří operací pak *Count* obsahuje binární číslo, které koresponduje s číslem v Grayově kódu *GrayByte*. Toto číslo lze pak zapsat do registrů *StartByte* a *EndByte*. Diference *Delta*, aby měl tento režim smysl, by měla být 1; pokud není, chování čítače lze určit podle uvedeného popisu.

Mód BCD

V tomto módu jsou veškerá binární čísla převáděna na kód BCD. To znamená, že i vstupní hodnoty *StartByte*, *EndByte* a *Delta* musí být platnými BCD kombinacemi. Pokud tomu tak není, výstupem budou sice platná BCD čísla, ovšem bez návaznosti na tyto nakonfigurované hodnoty.

Činnost tohoto módu je zaručena pouze pro diference jejichž obě dekadická čísla jsou menší nebo rovna 6 (tzn. 36 je v pořádku, 37 už ne, 64 je v pořádku, 67 ne).

Pokud zvolíme Grayův i BCD mód současně, bude získaná BCD kombinace převedena podle (2) na Grayův kód, což nedává smysl.

Okamžité akce bloku

(pořadové číslo koresponduje s požadovaným nastavením registrů DASx, viz tab. 8 a pozn.)

1. dAfreeze

- ovládáno staticky

Pozastavuje činnost bloku.

2. dAdisable

- ovládáno staticky

Dočasně vypne funkci bloku. Na výstupu pak bude trvale jedna z hodnot *StartByte* nebo *EndByte* s případnou konverzí na Grayův kód (*bCDadjust* se zde neprojeví).

3. dArestart

- spouštěno hranou

Vrátí čítač do stavu určeného buď registrem *StartByte* nebo *EndByte* podle aktuálního nastavení *direction*.

4. dAchangeDir

- spouštěno hranou

Způsobí změnu směru čítání. Tato akce neguje bit *direction*, čítač však není restartován (pokud vyžadujeme současný restart čítače, není nic jednoduššího než patřičný vstup přímé akce nakonfigurovat tak, aby spouštěl jak *dAchangeDir* tak i *dArestart*). Pokud v současném okamžiku zapisujeme do řídicího registru bloku (tzn. pokoušíme se bit *direction* přepsat), pak tato přímá akce dostane přednost. V dalším hodinovém taktu však už k přepsání dojde, protože tato akce spouštěna hranou, tzn. bude aktivní pouze jednu hodinovou periodu.

Požadavek přerušení

Požadavek přerušení je generován při dosažení konce čítání ale pouze tehdy, je-li bit *intE* nastaven.

C.2.4. Modul paměťového generátoru

Jméno entity: MemBPG

Složitost: 154 slices

Šířka výstupu: 8 bitů

Funkce tohoto modulu je velmi podobná funkci bloku BPG, hlavní rozdíl je v tom, že výstup je osmibitový. Tento blok využívá blokovou paměť RAM, kterou mají všechny vyšší modely obvodů FPGA. Její celková velikost se liší obvod od obvodu, obvody řady Spartan však mají tyto paměťové bloky stejně velké (18Kb), různí se však jejich počet. V použitém obvodu Spartan3 XC3S400 je těchto bloků 16.

Pomocný blok přístupu k blokové paměti FPGA

Jméno entity: XC3S_RAMB_2_PORT

Bloková paměť RAM obvodů FPGA firmy Xilinx je synchronní, dvouportová a konfigurovatelná na různou velikost slova. Blok XC3S_RAMB_2_PORT tedy neslouží k ničemu jinému než ke zpřístupnění jednoho bloku paměti pro šířku dat 8 bitů. V této konfiguraci je však nutné počítat s jedním bitem navíc, který je určen pro paritu (v tomto systému není využit). Z celkové velikosti bloku paměti 18Kb pak lze snadno spočítat, že adresa bude dána jedenáctibitovým číslem, tedy 0 až 800 (2048 B).

Tento blok byl navržen podle doporučení firmy Xilinx [14]. V tab. 12 je přehled a popis vstupů a výstupů bloku.

jméno signálu	šířka sběrnice	směr toku dat	popis
clock	1	vstupní	hodinový signál (pro oba porty paměti)
reset	1	vstupní	reset (pro oba porty paměti)
enA	1	vstupní	signál aktivace portu A
enB	1	vstupní	signál aktivace portu B
wEnA	1	vstupní	signál zápisu přes port A
wEnB	1	vstupní	signál zápisu přes port B
AddressA	11	vstupní	adresový vstup portu A
AddressB	11	vstupní	adresový vstup portu B
DataInA	9	vstupní	datový vstup portu A
DataInB	9	vstupní	datový vstup portu B
DataOutA	9	výstupní	datový výstup portu A
DataOutB	9	výstupní	datový výstup portu B

tab. 12. Popis vstupů a výstupů pomocného modulu blokové paměti

Je použit mód WRITE_FIRST pro oba porty RAM bloku (pro vysvětlení viz příslušnou literaturu, např. [14])

Popis bloku paměťového generátoru

Paměťový prostor bloku MemBPG popisuje tab. 13.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0		iNserv	iReq		finished	repeatForever	started	enabled
1	DisabledV (output byte when module not enabled)							
2	ByteStream (current output data)							
3	PattLen [†] (byte pattern length) LSB							
4							MSB	...
5	PattLenOffset (current pattern address offset) LSB							
6							MSB	...
7	PattStart (pattern start address) LSB							
8							MSB	...
9	PrescaleF (prescale factor)							
A	PrescaleCount (current prescale countdown)							
B	DASd		DASc		DASb		DASa	
C	dAnextBlock	dArestart	dAdisable	dAfreeze	dAd	dAc	dAb	dAa
D	RAMaddress (RAM data write address) LSB							
E							MSB	...
F	RAMDataByte (RAM data for both reading and writing)							

[†] reset value = 7FF

tab. 13. Paměťový prostor bloku MemBPG

Zápis a čtení z/do bloku paměti

Předtím, než budeme tento modul používat, je nutné naplnit RAM blok daty. K tomu slouží poslední dva registry bloku. Samotný přístup do RAM je velice jednoduchý. Do registru **RAMaddress** zadáme požadovanou adresu a k datům se dostaneme přes registr **RAMDataByte**. Signál zápisu je odvozen ze vstupu *wEn* bloku, takže čtením registru **RAMDataByte** čteme obsah paměti na zvolené adrese, zápisem tohoto registru na zvolenou adresu zapíšeme; o nic více se nemusíme starat.

Konkrétně je pomocný blok XC3S_RAMB_2_PORT použit tak, že port A je napojen na externí přístup (tedy na registry **RAMaddress** a **RAMDataByte**) a port B je pouze čten a jsou přes něj získávána data pro výstup *ByteStream* tohoto bloku. Port A je aktivován adresou registru nastavenou na F (adresa bytu **RAMDataByte**), port B je aktivován bitem *enabled* nebo jeho hodnotou zpožděnou o jeden takt hodin.

Konfigurace funkce bloku

Bit *repeatForever* má stejnou funkci, jakou u modulu BPG. Platí zde stejná pravidla.

Hodnota registru **DisabledV** určuje stav výstupu bloku v neaktivním (disabled) stavu.

Generovaná posloupnost bytů je určena obsahem bloku RAM. Adresu prvního generovaného byte určuje registr **PattStart**, délku sekvence pak nastavíme registrem **PattLen** (oba jsou jedenáctibitové, takže je třeba k nim přistupovat nadvakrát).

Registr **PattLen** je výjimečný tím, že při resetu je naplněn jedničkami (**PattStart** je vynulován). Je to z důvodu umožnění nepatrného zjednodušení v přístupu k tomuto bloku, paměť je totiž při konfiguraci FPGA kompletně vynulována, takže pokud svá data zapíšeme od adresy 0 a nastavíme *repeatForever* do nuly, můžeme blok ihned použít bez zdlouhavého zápisu k těmto jedenáctibitovým registrům.

Registr **PrescaleF** má stejnou funkci jako u předchozích bloků, umožňuje rozšířit dobu vysílání jednoho bytu na $PrescaleF + 1$ hodinových taktů.

Registry pro kontrolu stavu bloku

Pomocné bity *started* a *finished* oznamují započetí resp. ukončení vysílání nakonfigurované sekvence (bit *finished* je trvale vynulován při *repeatForever* v jedničce).

ByteStream je kontrolním registrem napojeným přímo na výstup bloku.

PattLenOffset obsahuje aktuální hodnotu čítače délky sekvence. Jeho hodnota se přičítá k *PattStart* a tvoří adresu čtení v RAM. Obsah tohoto registru je tedy před začátkem sekvence nulový, při skončení je roven *PattLen* (je však vzápětí vynulován).

V registru *PrescaleCount* je aktuální stav čítače prescalingu, jak to již bylo vícekrát použito.

Okamžité akce bloku MemBPG

(pořadové číslo koresponduje s požadovaným nastavením registrů DASx, viz tab. 8 a pozn.)

1. dAfreeze

- ovládáno staticky

Pozastavuje činnost bloku.

2. dAdisable

- ovládáno staticky

Dočasně vypne funkci bloku. Tato akce má za následek přepnutí výstupu *ByteStream* do hodnoty *DisabledV*.

3. dArestart

- spouštěno hranou

Spustí zvolenou sekvenci bytů od začátku, vynuluje tedy čítač *PattLenOffset*. Tato akce bude fungovat i v případě nastavení pouze jednorázového vyslání sekvence (*repeatForever* vynulován) a i po jejím dobehnutí.

4. dAnextBlock

- spouštěno hranou

Toto je speciální akce použitá pouze u tohoto bloku. Při jejím spuštění je obsah registru *PattStart* zvětšen o $PattLen + 1$, takže vstupní data se budou číst z dalšího úseku paměti. To umožňuje naplnit paměť několika různými sekvencemi bytů a tímto způsobem přecházet od jednoho k druhému. Pokud v současném okamžiku zapisujeme do registru *PattStart*, pak tato přímá akce dostane přednost. V dalším hodinovém taktu však už k přepsání dojde, protože tato akce je spouštěna hranou, tzn. bude aktivní pouze jednu hodinovou periodu.

Požadavek přerušení

Přerušení je požadováno po spuštění akce *dAnextBlock* pokud adresa začátku dalšího úseku paměti překročí celkovou velikost paměti. Popsané zvětšení *PattStart* na $PattLen + 1$ se samozřejmě děje na jedenácti bitech těchto registrů, takže při překročení rozsahu se dvanáctý bit ztratí a nová adresa pak ukazuje někam na začátek paměti. Tento dvanáctý bit součtu, předtím než se „zahodí“ je využit právě k nastavení *iRQ*.

C.2.5. Sigma-delta generátor

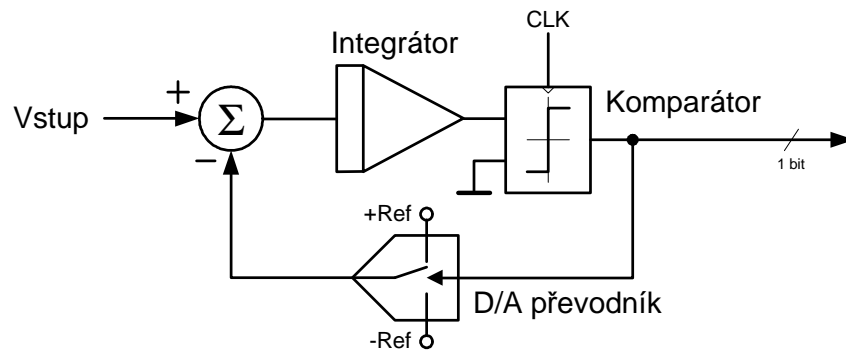
Jméno entity: SD

Složitost: 346 slices

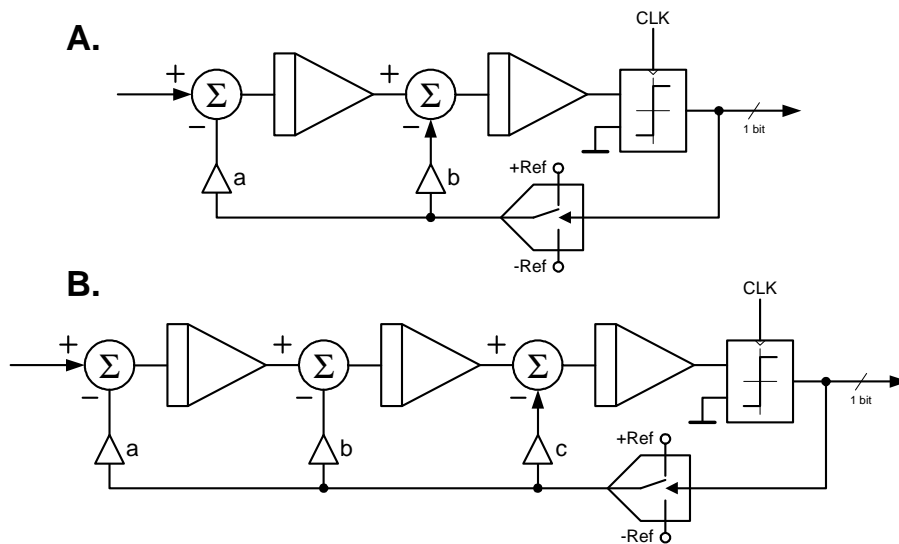
Šířka výstupu: 3 bity

Tento blok je vůbec nejsložitějším v celém systému. Obsahuje tři jednoduché modulátory sigma-delta (Σ - Δ), které mohou být použity buď samostatně nebo spojeny do modulátoru druhého či třetího řádu nebo do struktury MASH (Multistage Noise Shaping).

Návrh všech typů Σ - Δ modulátorů vycházel přímo z jejich schémat. Blokové schéma základní struktury prvního řádu je na obr. 8. Propojením dvou nebo tří jednoduchých modulátorů získáme modulátor 2. resp. 3. řádu. Důležitý je způsob takového propojení, ukázán je na obr. 9.

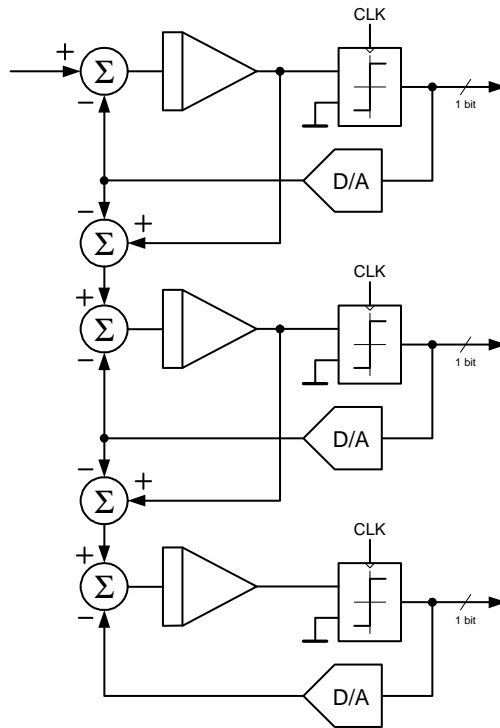


obr. 8. Sigma-delta modulátor prvního řádu



obr. 9. Sigma-delta modulátor 2. (A) a 3. (B) řádu

Dále byla implementována struktura MASH, rovněž na základě blokového schématu, které je na obr. 10.



obr. 10. Modulátor sigma-delta 3. řádu s architekturou MASH

Blok jednoduchého modulátoru sigma-delta

Jméno entity: SigmaDelta

Složitost: 46 slices

Tento pomocný blok je velmi zajímavý, proto bude rozebrán podrobněji. Označení „pomocný“ zde není zcela na místě, protože pouze s minimální úpravou může být použit i samostatně. Na následujících řádcích je vložen jen mírně upravený výpis zdrojového kódu tohoto bloku, což objasní jednak jeho funkci a jednak mechanismy jeho propojení do výše uvedených struktur. Dalším důvodem je to, že celý tento projekt byl vytvořen ve Verilogu, takže aspoň malá ukázka je na místě. Nezasvěcený čtenář může navíc získat aspoň základní představu o tom, co HDL jazyky obnáší.

```

module SigmaDelta(reset, clock, freeze, clear,
                 In, Ref, refAddSub, Integ, comp);

    /* Ports Definition */

    parameter R = 8; // reference word width
    parameter I = 8; // input word width
    parameter Q = 24; // integrator width

    input reset; // reset (async, active L)
    input clock; // clock
    input freeze; // freeze signal
    input clear; // clear integrators

    input signed [I-1:0] In; // input value

```

```

input signed [R-1:0] Ref;      // reference value
input refAddSub;              // DAC input
output signed [Q-1:0] Integ;  // integrator output
output comp;                  // comparator output

/* Internal Registers */

reg signed [Q-1:0] Integrator; // integrator register
wire signed [I:0] IntAdd;      // integrator summand

/* Executive */

assign Integ = {Q{~clear}} & Integrator;
assign comp = (Integ > 0);
assign IntAdd = refAddSub ? In - Ref : In + Ref;

always @(posedge clock or negedge reset)
  if (!reset)
    Integrator <= 0;
  else
    if (!freeze | clear) Integrator <= Integ + IntAdd;

endmodule

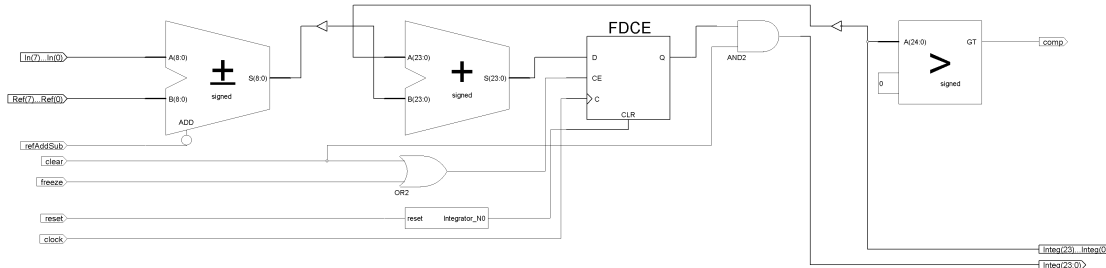
```

Syntaxe ani sémantika jazyka tu nebude vysvětlována, k tomu slouží odborná literatura (pro rychlý a velmi stručný přehled je velmi dobrý pramen [15]). Je zajímavé, že vlastní funkčnost modulu je popsána jen posledními osmi řádky modulu, vše ostatní jsou jen deklarace nebo definice. Dále je poněkud neobvyklé požití znaménkových (signed) registrů. Na samotné registry (klopné obvody) to samozřejmě nemá žádný vliv, změni to pouze reprezentaci jejich obsahu, který je pak chápán jako číslo ve dvojkovém doplňku.

Tento kód je syntetizován do struktury odpovídající obr. 8. Vstupní port je v modulu označen *In*, dále je třeba přivést hodnotu reference *Ref* pro strukturu odpovídající D/A převodníku (což je v tomto „digitálním“ případě nahrazeno přivedením této referenční hodnoty na kladný či záporný vstup sčítačky). Kromě vlastního výstupu komparátoru *comp* je vyveden ještě i výstup okamžité hodnoty integrátoru *Integ* (jedině tak budeme moci tento blok použít i pro strukturu MASH z obr. 10). Vstup *refAddSub* je vstupem D/A převodníku (resp. jeho digitální reprezentace); jednička na něm znamená odečtení reference od vstupu, nula přičtení. Pokud bude tento blok použit jako samostatný modulátor 1. řádu, je nutné tento vstup propojit s výstupem *comp* (což je ona minimální úprava, o které byla řeč).

Tento blok bude aspoň v použitém návrhové prostředí syntetizován do digitální struktury zobrazené na obr. 11. Tomuto typu zobrazení se říká RTL (register transfer level) a je jednou z vyšších úrovní popisu digitálních systémů. Detaily pro makrobloky (sčítačky a komparátor) nejsou pro přehlednost zobrazeny a z uvedeného kódu je jasné, že nejsou ani nijak detailně popisovány, v tomto ohledu byla syntetizačnímu programu ponechána tak říkajíc volná ruka. Je pak na syntetizátoru, jakou konkrétní strukturu zvolí a je jasné, že při takto obecném popisu bude mít velké možnosti optimalizace. Je však třeba na jeho postup důsledně dohlížet; v praxi se ve většině případů takovýto přístup nepoužívá, designer potřebuje mít kontrolu nad tím, co konkrétně syntéza vytvoří. Pro naše účely je to však vyhovující.

Poznámka k obr. 11: Tento obrázek byl získán pomocí programu WebPack po syntéze výše uvedeného kódu. Evidentně je v něm chyba, popsany modul totiž nemá dva výstupy *Integ*, ale pouze jeden (oba zobrazené výstupy jsou navíc propojeny, protože označují stejné signály). Z toho je vidět, že podobná schémata nejsou tímto prostředím příliš podporována, mnohem lépe si počítač „rozumí“ s textovým zadáním.



obr. 11. RTL schéma digitální struktury bloku jednoduchého sigma-delta modulátoru

Ke správné funkci tohoto bloku je však nutná ještě jedna úprava a tou je saturační aritmetika. Při překročení rozsahu čísla by totiž došlo k přetečení, což by se u čísla ve dvojkovém doplňku projevilo změnou znaménka (!) a správná funkce bloku by tak byla naprosto degradována.

První sčítačka tuto úpravu nepotřebuje, protože sběrnice na jejím výstupu je o jeden bit větší, než obě vstupní. Pro sčítačku integrátoru takovou úpravu provést nemůžeme. Proto musíme uvedený kód modifikovat následujícím způsobem: předposlední řádek

```
if (!freeze | clear) Integrator <= Integ + IntAdd;
```

nahradíme příkazem

```
if (!freeze|clear) Integrator <= SaturatingAdd(IntAdd, Integ);
```

a funkci `SaturatingAdd` vytvoříme takto:

```
function signed [Q-1:0] SaturatingAdd;
```

```
input signed [Q-1:0] A;
```

```
input signed [Q-1:0] B;
```

```
reg signed [Q-1:0] Y;
```

```
reg overflow;
```

```
begin
```

```
    Y = A + B;
```

```
    overflow = (A[Q-1] & B[Q-1] & ~Y[Q-1]) |  
              (~A[Q-1] & ~B[Q-1] & Y[Q-1]);
```

```
    SaturatingAdd = overflow ? {A[Q-1], {(Q-1){~A[Q-1]}} : Y;
```

```
end
```

Podle [16] dojde u při sčítání dvou čísel ve dvojkovém doplňku k přetečení tehdy, když nejvyšší bity obou operandů jsou shodné, nejvyšší bit výsledku je však opačný. Takový výsledek je pak nepoužitelný a musí se nahradit nejvyšším resp. nejnižším možným číslem, které nám použitá šířka slova dovoluje.

Funkční a konfigurační možnosti bloku SD

Jako obvykle uvedeme nejprve výpis paměťového prostoru tohoto bloku (tab. 14).

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0		iNserv	iReq	IntegSel		SDstruct [†]		enabled
1	Ref1 (reference for modulator 1)*							
2	Ref2 (reference for modulator 2)*							
3	Ref3 (reference for modulator 3)*							
4	In1 (input value for modulator 1)*							
5	In2 (input value for modulator 2)*							
6	In3 (input value for modulator 3)*							
7	DASd		DASc		DASb		DASa	
8	dAdiv2	dAmult2	dArestart	dAfreeze	dAd	dAc	dAb	dAa
9	SDInteg (current integrator contents)*							
A								LSB
A	MSB for modulator 1							...
B	MSB for modulator 2							...
C	MSB for modulator 3							...
D						pdm3	pdm2	pdm1
E								
F								

* signed value

† read-only when enabled active

tab. 14. Paměťový prostor bloku SD

Nejdůležitějšími dvěma bity v řídicím registru tohoto bloku jsou bity volby sigma-delta struktury *SDstruct*. Tyto bity umožňují nakonfigurovat blok do struktury odpovídající obr. 8, obr. 9 (A i B) nebo obr. 10. Výstupy *pdm1*, *pdm2* nebo *pdm3* jsou pevně napojeny na výstup pomocného bloku modulátoru 1. řádu (port *comp*). Přehled konfigurace těchto dvou bitů udává tab. 15. Tyto bity jsou uzamčeny pro zápis, pokud je bit *enabled* nastaven (tj. můžeme do nich zapisovat pouze tehdy, měníme-li zároveň bit *enabled* z 0 na 0 nebo na 1 a nebo z 1 na 0), jinak by totiž při přepojování struktury vznikly počáteční nepřesnosti, které se např. u MASH architektury projeví na celém generovaném bitstreamu.

SDstruct	Structure	Output	Register bank
00	1st-order $\Sigma\Delta$	<i>pdm1</i>	1
	1st-order $\Sigma\Delta$	<i>pdm2</i>	2
	1st-order $\Sigma\Delta$	<i>pdm3</i>	3
01	2nd-order $\Sigma\Delta$	<i>pdm2</i>	1 + 2
	1st-order $\Sigma\Delta$	<i>pdm3</i>	3
10	3rd-order $\Sigma\Delta$	<i>pdm3</i>	1 + 2 + 3
11	MASH	<i>pdm1</i> + <i>pdm2</i> + <i>pdm3</i>	1 + 2 + 3

tab. 15. Konfigurace sigma-delta struktury a její návaznost na výstupy a konfigurační registry

Jestliže zvolíme strukturu nižšího řádu, než je maximum (3), zůstane nám jeden či dva modulátory 1. řádu volné a máme je k dispozici nezávisle na ostatních. U struktur vyššího řádu jsou výstupy stále připojené na výstupy vnitřních modulátorů 1. řádu zvolené struktury, avšak smysl má pouze jeden. Výjimkou je struktura MASH, která používá všechny tři.

Pro konkrétní způsob vnitřního přepojování uvedu část zdrojového kódu, která má toto na starosti. Mimo jiné z ní plyne způsob použití registrů referencí pro jednotlivé stupně modulátorů vyšších řádů. Propojení portů modulátorů 1. řádu s vnitřními propojovacími linkami je následující:

- *In1*, *In2*, *In3* odpovídají vstupům *In* nižšího bloku modulátoru 1. řádu

- *pdm1*, *pdm2*, *pdm3* jsou napojeny na výstupy *comp*
- *Ref1*, *Ref2*, *Ref3* jsou přivedeny na vstupy referencí
- *SD1Integ*, *SD2Integ* a *SD3Integ* jsou připojeny k výstupům *Integ* těchto pomocných bloků

```

always @(SDstruct,
        In1, In2, In3, pdm1, pdm2, pdm3,
        SD1Integ, Ref1, SD2Integ, Ref2)
begin
    // 1. independent modulators
    SD1In = In1;
    SD2In = In2;
    SD3In = In3;
    sD1ref = pdm1;
    sD2ref = pdm2;
    sD3ref = pdm3;
    case (sdStruct)
        2'b01 : begin
            // 2. modulator 2 + 1 = one 2nd-order modulator
            SD2In = SD1Integ;
            sD1ref = pdm2;
        end
        2'b10 : begin
            // 3. modulator 3 + 2 + 1 = one 3rd-order modulator
            SD2In = SD1Integ;
            SD3In = SD2Integ;
            sD1ref = pdm3;
            sD2ref = pdm3;
        end
        2'b11 : begin
            // 4. 3rd-order MASH structure
            SD2In = pdm1 ? (SD1Integ - Ref1) :
                       (SD1Integ + Ref1);
            SD3In = pdm2 ? (SD2Integ - Ref2) :
                       (SD2Integ + Ref2);
        end
    endcase
end

```

Bit *enabled*, kromě svého obvyklého významu má zde ještě další funkci, a tou je synchronní nulování obsahu integrátorů všech pomocných modulátorů (přes port *reset*). Pokud jej tedy vynulujeme, budou v další hodinové periodě výstupy všech stupňů nulové. V tom se liší od provedení okamžité akce *dArestart*, která tyto bloky nuluje asynchronně přes port *clear*, takže ihned po následující hraně hodin jsou na všech výstupech již platné hodnoty.

Konfigurace vlastního sigma-delta bitstreamu

Generovaný bitstream (též PDM stream) je konfigurován registry *In1* a *Ref1* (banka 1) *In2* a *Ref2* (banka 2) a *In3* a *Ref3* (banka 3) a jejich konkrétní použití v závislosti na zvolené struktuře udává jednak uvedený zdrojový kód a jednak tab. 15. Tyto hodnoty jsou znaménkové v kódu dvojkového doplňku. Jak rovněž vyplývá z předchozího, všechny tři

reference jsou nezávislé a to i pro modulátory 2. a 3. řádu. To umožňuje navrhnout struktury z obr. 9 A i B včetně multiplikačních konstant **a**, **b**, **c** (rozdíl implementace oproti těmto obrázkům je tedy v tom, že každý stupeň má svůj vlastní D/A převodník, resp. jeho náhradu).

Tyto vstupní registry jsou osmibitové, vnitřní registry, zejména pak akumulátory, jsou větší. Je to z toho důvodu, aby ani v kaskádě tří modulátorů nedocházelo k chybám přetečení. Shrnutí je v tab. 16, názvy portů byly vysvětleny výše

	Modulator	1	2	3
Port	In	8 bit	17 bit	25 bit
	Ref	8 bit	8 bit	8 bit
	Integ	16 bit	24 bit	32 bit

tab. 16. Šířky portů jednotlivých pomocných modulátorů

Registry pro kontrolu stavu bloku

Pro tuto funkci byl zaveden jediný registr *SDInteg*, který lze nakonfigurovat tak, že zobrazuje hodnotu integrátoru modulátoru 1, 2 nebo 3 (tedy *SD1Integ*, *SD2Integ* a *SD3Integ*). Výběr integrátoru je dán dvěma bity *IntegSel* v řídicím registru a je dána tab. 17.

IntegSel	SDInteg
00	SD1Integ
01	SD2Integ
10	SD3Integ
11	not defined

tab. 17. Výběr integrátoru pro kontrolní výstup *SDInteg*

Bity *pdm1*, *pdm2*, *pdm3* zobrazují aktuální stav výstupů bloku.

Okamžité akce bloku SD

(pořadové číslo koresponduje s požadovaným nastavením registrů *DASx*, viz tab. 8 a pozn.)

1. *dAfreeze*

- ovládáno staticky

Pozastavuje činnost bloku.

3. *dArestart*

- spouštěno hranou

Nuluje integrátory. Výstupy jsou plané již při další hraně hodin.

2. *dAmult2*

- spouštěno hranou

Násobí všechny tři vstupní hodnoty (*In1*, *In2*, *In3*) dvěma. Je implementována saturační aritmetika, která omezí maximum na +127 nebo -128.

3. *dAdiv2*

- spouštěno hranou

Dělí všechny tři vstupní hodnoty (*In1*, *In2*, *In3*) dvěma – provede vlastně bitový posuv vpravo. Po osmi posuvech tedy dostaneme nulu.

Požadavek přerušení

Požadavek přerušení nastane při spuštění okamžité akce `dAmult2` pokud překročíme rozsah kteréhokoliv z měněných registrů (*In1*, *In2*, *In3*). V tom případě je číslo omezeno na maximum/minimum.

Druhá možnost pro generování přerušení je při okamžité akci `dAdiv2`, konkrétně tehdy, dělíme-li číslo, které není násobkem dvou (opět kteréhokoliv z *In1*, *In2*, *In3*). Žádné další úpravy nejsou prováděny, číslo je vždy bitově posunuto vpravo. Nakonec tedy může nastat vynulování těchto registrů.

C.3. Měřicí bloky

Většina měřicích bloků představuje duální protějšek stimulačních bloků a když je propojíme, měli by se naměřené hodnoty shodovat s hodnotami nakonfigurovanými v použitých stimulačních blocích. Měřicí bloky, stejně jako stimulační, jsou napojeny na systém sběrnic umožňujícím komunikaci s hostitelským počítačem (a ve finále tedy vyčtení změřené hodnoty).

Ve svém řídicím registru rovněž obsahují bity *enabled*, *iReq* a *iNserv* se stejným významem, jako u bloků stimulačních. Funkční stavy jsou zde poněkud zjednodušeny, ve stavu *disabled* (bit *enabled* vynulován) jsou všechny měřicí registry trvale nulové. Pokud tedy chceme restartovat měření, je dobré vynulovat bit *enabled*.

Stav zastavení (*frozen*) se projeví tím, že blok dočasně zastaví veškerou činnost, nebude tedy ani reagovat na žádný ze svých měřicích vstupů. Vše ostatní však zůstává neměnné, zejména stav signálů přímých akcí.

Tyto bloky jsou globálně aktivovány nebo pozastavovány stejným signálem *gEnable*, jako stimulační bloky.

Pokud mají měřicí bloky registry označené stejně, jako jejich vstupní signály, pak čtením těchto bitů jsme přesměrováni přímo na vstup bloku ergo na příslušný pin hardwaru systému. To má mnohem větší význam než podobné bity u stimulačních bloků, protože přes tento bit (příp. více bitů) můžeme softwarově zjistit okamžitou digitální hodnotu vstupu.

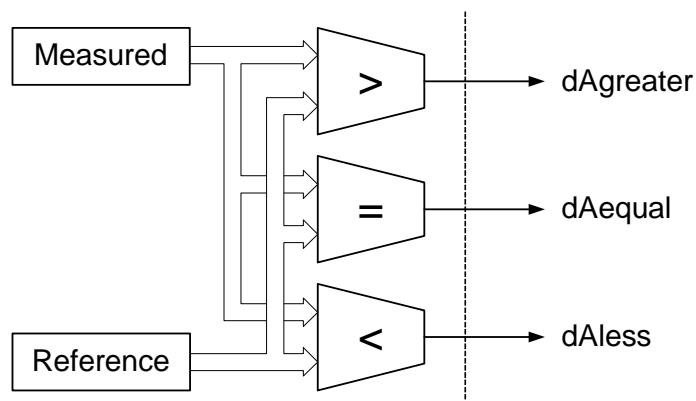
Signály okamžitých akcí

U měřicích bloků jsou tyto signály výstupní a jsou určeny k připojení na příslušné vstupy stimulačních bloků. Pro jejich existenci a činnost byla zvolena filosofie binárního porovnávání měřeného čísla se zadanou referencí. Každý měřicí blok tedy obsahuje registr pro referenční hodnotu (*CompRef*) a tři komparátory, které tuto hodnotu porovnávají s naměřenou. Výstupem jsou tři signály přímých akcí:

- `dAless`: Indikuje, že naměřená hodnota je menší než referenční
- `dAequal`: Oznamuje, že naměřená hodnota je shodná s referenční
- `dAgreater`: Indikuje, že naměřená hodnota je větší než referenční

Tento přístup je opět jen kompromisem mezi únosnou složitostí systému a jeho univerzálností, i tak však spolu s propojovací maticí skýtá velký potenciál.

Popsané vnitřní uspořádání ilustruje obr. 12.



obr. 12. Generování signálů okamžitých akcí v měřicích blocích (Reference = *CompRef*)

Platí tedy, že maximálně jeden z těchto signálů může být aktivní (v úrovni H). Stav kdy ani jeden z nich není aktivní nastává po startu měření (aktivací *gEnable*) před dokončením prvního měření (dosažení zadané periody apod.) a je charakterizován nulovým obsahem bitu *measComplete*, který rovněž většina měřicích bloků implementuje (ne však všechny).

Rozhraní pro blokovou paměť

Toto rozhraní bylo přidáno ke každému bloku proto, aby bylo možné zaznamenat výsledky měření do jedné z blokových pamětí FPGA. Je velice jednoduché a je určeno pro spolupráci s blokem paměťového úložiště MBmem (viz kap. C.4).

Rozhraní pro blokovou paměť je tvořeno pouze jedním signálem a datovou sběrnicí popsány v tab. 18. Datová sběrnice je v každém bloku napojena na registr s označením *Measured* (ve většině případů, výjimky budou zdůrazněny). Signál *mDataValid* indikuje přítomnost platných dat na této sběrnici a zároveň tedy představuje i požadavek zápisu těchto dat. Je aktivní právě jednu hodinovou periodu, pokud nejsou v dalším taktu připravena k zápisu již nová data.

jméno signálu	počet ve sběrnici	popis
mDataValid	1	indukuje platná data na sběrnici dat
MData	8, 16 nebo 24	sběrnice dat

tab. 18. Rozhraní měřicích bloků pro MBmem

Datové sběrnice měřicích bloků jsou v jednoduchých případech osmibitové, jsou však bloky, které měří s rozlišením 16 bitů a blok měřící sigma-delta má výstup dokonce 24bitový. Datový vstup bloku MBmem je pevně dán a je osmibitový. Jak tedy ukládat data, která mají větší šířku? Řešení je jednoduché, použijeme dva nebo tři MBmem bloky a širokou sběrnicí rozdělíme mezi ně. Signál požadavku zápisu však musíme rozdělit a přivést na příslušné vstupy všech použitých MBmem bloků.

Okamžitý vs. zpožděný start

Každý měřicí blok lze nakonfigurovat do módu tzv. zpožděného startu. K tomuto účelu je v řídicím registru těchto bloků vyhrazen bit *delayStart* (tab. 19). V tomto módu je vstup *gEnable* zpožděn o jednu periodu hodin *clock* (srov. též obr. 5). První vstupní vzorek je tedy měřen až při druhé vzestupné hraně hodin. Pokud je tento bit vynulován, je měřen hned první vzorek, který je na vstupu de facto již při aktivační hraně *gEnable*. Mód zpožděného startu tedy nakonec dává větší smysl, než okamžitý start.

Tento mód je nutný i pokud budeme chtít připojit některý ze stimulačních bloků zpět na měřicí blok, jinak není první měřený vzorek platný.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0		iReq	iNserv		delayStart			enabled

tab. 19. Společné bitu řídicího registru měřicích bloků

C.3.1. Modul měření obecného jednobitového signálu

Jméno entity: BP1sense

Složitost: 42 slices

Šířka vstupu: 1 bit

Tento modul je duální k bloku BPG, je však nesrovnatelně jednodušší. Jeho použití dává větší smysl při propojení s blokem paměťového úložiště MBmem. Neumí nic jiného, než ukládat vstupní proud bitů do posuvného registru a když je tento registr plný, požádá o zápis změřené hodnoty do MBmem a provede popsanou komparaci pro vytvoření signálů okamžitých akcí. Hloubka posuvného registru je pouze osm bitů, protože blok MBmem má také pouze osmibitovou vstupní datovou sběrnici. Paměťový prostor bloku BP1sense je v tab. 20. Jednoduchost tohoto bloku bude zároveň využita k vysvětlení některých společných mechanismů pro použití měřicích bloků.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	bitStream	0	0		delayStart	measComplete	invert	enabled
1						dAgreater	dAequal	dAless
2	Measured (last measured value)							
3	CompRef (comparison reference value)							
4	ShiftIn (current shift register contents)							
5						CurrMeasPeriod		
6								
7								
8								
9								
A								
B								
C								
D								
E								
F								

tab. 20. Paměťový prostor bloku BP1sense

Konfigurační možnosti

Bit *invert* dovoluje invertovat vstupní bitstream před jakoukoliv další operací.

Bit *measComplete* indikuje konec měření, tj., že změřená data jsou platná. Tímto bitem jsou zároveň aktivovány komparátory pro generování signálů okamžitých akcí. Tento bit lze jak číst tak i zapisovat a není automaticky nulován při vynulování bitu *enabled*. Je to kvůli testovacím účelům, nastavíme-li totiž tento bit externě do jedničky, aktivujeme tím komparátory a tedy i příslušný signál okamžité akce, což může být ve step-módu užitečné. Pokud nám toto chování nevyhovuje, jediné, co musíme zajistit je, že při zápisu bitu *enabled*, tedy řídicího registru bloku, vymaskujeme bit *measComplete* a docílíme tak chování běžného u stimulačních bloků.

CompRef určuje referenci komparátorů pro signály okamžitých akcí, jak již bylo vysvětleno.

Registry změřené hodnoty a registry pro kontrolu stavu bloku

Změřená hodnota se ukládá do registru *Measured* (toto označení je společné pro všechny měřicí bloky). Není to však zároveň pracovní posuvný registr vstupního bitstreamu, tím je registr *ShiftIn*, ale je vyhrazen pouze pro jediný účel – uchování naposledy změřené hodnoty. Tento registr je totiž trvale připojen na komparátory okamžitých akcí, proto musí vždy obsahovat platná naměřená data. Přepsán je pokaždé, když je dosaženo maximální hloubky pracovního posuvného registru; pak je obsah tohoto registru (*ShiftIn*) zkopírován do *Measured* a bit *measComplete* je nastaven do jedničky (pokud byl předtím nulový).

CurrMeasPeriod je čítač aktuálního posuvu pracovního registru *ShiftIn*. Měření jednoho byte je kompletní, jakmile tento čítač dosáhne 7 (naplněn jedničkami).

Na místo bitu *bitStream* je přiveden aktuální vstupní signál.

Poslední kontrolní registr, o kterém bude řeč, se nachází na adrese 1 a jsou do něj promítány aktuální hodnoty výstupů okamžitých akcí tohoto bloku.

Požadavek přerušení

Tento blok negeneruje požadavek přerušení, příslušné bity v jeho řídicím registru jsou trvale nulové.

C.3.2. Modul měření obecného osmibitového signálu

Jméno entity: BP8sense

Složitost: 25 slices

Šířka vstupu: 8 bitů

Tento blok je duální k MemBPG, a je ještě jednodušší, než předchodzí BP1sense. Nepotřebuje totiž čítač posuvu, každý vstupní byte totiž je zároveň platným bytem naměřených dat. Jako BP1sense je i tento určen primárně ke spolupráci s blokem MBmem.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0		0	0		delayStart			enabled
1						dAgreater	dAequal	dAless
2	Measured (last measured value)							
3	CompRef (comparison reference value)							
4	BPStream (current input byte)							
5								
6								
7								
8								
9								
A								
B								
C								
D								
E								
F								

tab. 21. Paměťový prostor bloku BP8sense

Konfigurační možnosti

Do registru *CompRef* se zapisuje referenční hodnota pro komparátory generující signály okamžitých akcí.

Registry změřené hodnoty a registry pro kontrolu stavu bloku

Vstupní byte je možné přečíst registrem *BPStream*, naměřená hodnota se ukládá do registru *Measured*. Mezi nimi není žádný prostředník. Hlavní náplní práce tohoto bloku je tedy rozhodování, kdy jsou vstupní data platná a kdy ne, což se děje na základě stavu bitu *enabled* a signálu *gEnable* podle již několikrát popisovaných pravidel.

Požadavek přerušení

Tento blok negeneruje požadavek přerušení, příslušné bity v jeho řídicím registru jsou trvale nulové.

C.3.3. Modul měření PDM signálu

Jméno entity: PDMSense

Složitost: 132 slices

Šířka vstupu: 1 bit

Blok PDMSense měří střední hodnotu vstupního signálu. Konfigurovatelná je délka periody PDM, která je určena 16bitovým číslem, takže může nabývat hodnot 1 až 65536. Během této doby jsou pak vlastně počítány aktivní úrovně na vstupu bloku. Vstupní signál může být vnitřně invertován, takže aktivní úroveň může být změněna z H na L. Paměťový prostor tohoto bloku je v tab. 22.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
0	pdmStream	iNserv	iReq		delayStart	measComplete	invert	enabled	
1						dAgreater	dAequal	dAless	
2	Plength (PDM period length)								LSB
3	MSB								...
4	CompRef (comparison reference value)								LSB
5	MSB								...
6	Measured (last measured value)								LSB
7	MSB								...
8	CurrMeasValue (current measured value counter)								LSB
9	MSB								...
A	CurrMeasPeriod (current measured period counter)								LSB
B	MSB								...
C									
D									
E									
F									

tab. 22. Paměťový prostor bloku PDMSense

Všechny konfigurační registry tohoto bloku (kromě řídicího) jsou šestnáctibitové, je tedy nutné číst i zapisovat je nadvakrát. Rovněž tak registry naměřených dat jsou hluboké 16 bitů, takže i komparátory generování signálů okamžitých akcí jsou šestnáctibitové (což se odrazilo na rozsáhlosti tohoto bloku, viz „složitost“). Je to však nutné, osmibitová čísla byla příliš omezující pro funkci tohoto bloku. Datová sběrnice měřených dat je tedy také šestnáctibitová. Návod, jak ji použít k propojení s blokem paměťového úložiště, byl již prozrazen výše.

Konfigurační možnosti

Hlavní a v podstatě jedinou konfigurační možností tohoto bloku je zadání délky periody přes registr *Plength*. Opět zde platí, že 0 odpovídá délce 1 hodinového taktu, 255 pak 256 hodinovým periodám a maximum je 65.535 což nastaví periodu délky 65.536 hodinových period. Dále je pak možné měnit bit *invert*, který, pokud je nastaven do 1, invertuje vstupní signál. *CompRef* určuje referenci komparátorů okamžitých akcí. Funkce bitu *measComplete* je totožná jako u předchozích měřicích modulů, opět jsou jím zároveň aktivovány komparátory pro generování signálů okamžitých akcí, pro detaily viz kap C.3.1.

Detaily k funkci bloku, registry změřené hodnoty a kontrolní registry

Změřená hodnota se po každé dokončené periodě PDM ukládá do registru *Measured* a stejná data jsou posílána i na sběrnici *MData*.

Aktuální pozici v PDM periodě udává čítač *CurrMeasPeriod*. Tento čítač čítá od nuly vzhůru a při dosažení *Plength* je nastaven bit *measComplete*, data z pracovního registru *CurrMeasValue* jsou zkopírována do *Measured* a čítač *CurrMeasPeriod* je vynulován.

CurrMeasValue tedy počítá měřenou hodnotu. Pokaždé, když je na vstupu detekována aktivní úroveň (H pro *invert* = 0, L pro *invert* = 1), je tento čítač inkrementován. Na konci periody pak obsahuje platnou změřenou hodnotu

Do bitu *pdmStream* je opět promítána aktuální hodnota vstupu. Bity *dAgreater*, *dAequal* a *dAless* pak obsahují aktuální stav výstupů okamžitých akcí.

Požadavek přerušení

Přerušení je vyžadováno po každé změřené periodě, následující perioda totiž předchozí změřenou hodnotu přepíše. Bit *iNserv* však není nastaven ani když požadované přerušení není i vícekrát obslouženo pokud je nová změřená hodnota shodná s předchozí.

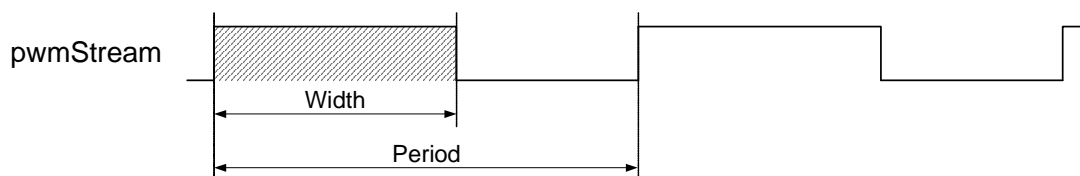
C.3.4. Modul měření PWM signálu

Jméno entity: PWMsense

Složitost: 136 slices

Šířka vstupu: 1 bit

Modul PWMsense je ve své podstatě čítačem spouštěným hranou. Vzestupná hrana vstupního signálu čítač spustí, následující sestupná hrana zastaví čítání délky impulsu PWM signálu a další vzestupná hrana ukončí měření periody. Viz též obr. 13.



obr. 13. Parametry měřené blokem PWMsense

Všechny měřicí registry tohoto bloku jsou opět šestnáctibitové. Volitelně lze na výstup kromě inverze zařadit též filtr parazitních impulsů. Mapa registrů bloku PWMsense je v tab. 23.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	glitchFilter	iNserv	iReq	measStarted	delayStart	measComplete	invert	enabled
1	in				mPorW	dAgreater	dAequal	dAless
2	CompRef (comparison reference value)							LSB
3	MSB							...
4	MeasuredWidth (last measured width)							LSB
5	MSB							...
6	MeasuredPeriod (last measured period)							LSB
7	MSB							...
8	CurrMeasWidth (current measured width counter)							LSB
9	MSB							...
A	CurrMeasPeriod (current measured period counter)							LSB
B	MSB							...
C								
D								
E								
F								

tab. 23. Paměťový prostor modulu PWMsense

Konfigurační možnosti

Rozdíl oproti předchozímu bloku PDMsense je ihned patrný, chybí zde totiž registr pro délku periody. To však neznamená že, PWMsense je jednodušší, spíše naopak. Periodu si tento blok totiž měří sám.

CompRef je šestnáctibitový registr reference pro komparaci a vytváření signálů okamžitých akcí.

Bit *invert* invertuje vstupní signál. Pokud je nulový, je jako spouštěcí hrana periody brána vzestupná hrana vstupního signálu, v opačném případě je hranice periody určena sestupnou hranou vstupu.

Bit *measComplete* oznamuje ukončení měření první periody. Více viz kap. C.3.1. Navíc však přibyl bit *measStarted* který indikuje, že již byla detekována spouštěcí hrana periody a měření tak vlastně začalo. Při další hraně periody je jeho hodnota jednoduše zkopírována do *measComplete*, se všemi důsledky.

Bitem *mPorW* určujeme, zda nás jako měřená hodnota zajímá délka periody *MeasuredPeriod* (*mPorW* v jedničce) nebo impulsu *MeasuredWidth* (při *mPorW* = 0). Zvolená hodnota se pak porovnává s referenční v komparátorech okamžitých akcí a je též přivedena na sběrnici *MData*.

Funkce filtru parazitních impulsů

Tato funkce je realizována samostatným pomocným blokem *degitcher*. Pro vysvětlení jeho funkce opět uvedu hlavní část zdrojového kódu tohoto modulu (vstupní port je nazván *in*, výstupní *out*):

```
always @(posedge clock or negedge reset)
// Blocking assignment is used to slightly simplify the code
if (!reset)
begin
temper = {{G/2{1'b0}}, {G/2{1'b1}}}; // zero degrees
compOut = 1'b0;
end
else
```

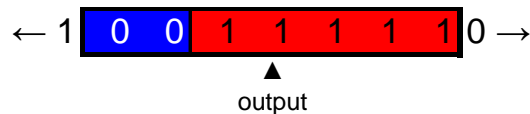
```

begin
  if (enable)
    temper = in ? (temper << 1) | 1'b1 : temper >> 1;
  else // get ready for the next measurement
    temper = {{G/2{1'b0}}, {G/2{1'b1}}}; // zero degrees
    compOut = temper[G/2];
  end

```

```
assign out = enable ? compOut : in;
```

Uvnitř tohoto bloku je tedy jakýsi posuvný „teplotní“ registr *temper* hloubky *G* (zvoleno 4), který je při resetu naplněn z poloviny jedničkami, což odpovídá řekněme nule teplotních stupňů. Přejde-li na vstup jednička, teplota je zvýšena o stupeň, tj. obsah registru je posunut vpravo a uprázdněné místo je doplněno jedničkou. Při příchodu nuly se registr „ochladí“. Výstupem celého bloku je hodnota bitu, který odpovídá nule stupňů, tj. prostředního.



obr. 14. K vysvětlení funkce filtru parazitních impulsů

Tento mechanismus je dost neobvyklý aspoň pokud je autorovi známo, ale velmi účinný. Jeho nedostatky se projeví, pokud je vstup více nestabilní než zdrávo tzn. pokud se jedničky a nuly střídají příliš často. V takovém případě však selhávají i jiné mechanismy a je vůbec otázka, jaká hodnota výstupu je správná.

Je zřejmé, že použití takového filtru zavádí do zpracování signálu zpoždění, které může nabývat největší hodnoty rovné hloubce *temper* registru. Na druhou stranu, pro signál, který neobsahuje žádné parazitní impulsy, je toto zpoždění konstantní (a rovné právě hloubce *G*). Jako parazitní impuls je vyhodnocena ta část signálu, která je konstantní pouze po dobu kratší, než *G*.

Tento filtr je ovládán bitem *glitchFilter* řídicího registru bloku *PWMsense*, jednička jej zapíná, nula jej vypíná. Pokud je vypnut, signál jím není vůbec ovlivněn.

Detaily k funkci bloku, registry změřené hodnoty a kontrolní registry

CurrMeasValue je čítač délky impulsu PWM signálu a čítá od nuly. Na konci periody je jeho hodnota přenesena do *MeasuredWidth* a je vynulován.

Registr *CurrMeasPeriod* udává aktuální hodnotu čítače délky periody PWM signálu. I tento čítač čítá od nuly vzhůru a při detekci hranice periody je nastaven *measComplete*, jeho hodnota je zkopírována do *MeasuredPeriod* a *CurrMeasPeriod* je vynulován.

Hodnoty v registrech výsledů měření tedy odpovídají přímo změřeným hodnotám a nemusejí tedy být zvětšovány o 1, jak je již pomalu zvykem při zadávání konfiguračních hodnot.

Bit *in* není prostým zrcadlem vstupního signálu, ale to výstup z bloku *deglitcher* (je tedy napojen na jeho port *out*). Vstupnímu signálu odpovídá pouze v případě, že *deglitcher* je vypnut. Bity *dAgreater*, *dAequal* a *dAless* pak obsahují aktuální stav výstupů okamžitých akcí.

Požadavek přerušení

Přerušení je vyžadováno po každé změřené periodě, následující perioda totiž předchozí hodnotu přepíše. Ani zde není bit *iNserv* nastaven ani při opakované nevyslyšené žádosti o přerušení, pokud oba měřené parametry (period i width) jsou stejné s předchozími.

C.3.5. Měřicí blok sigma-delta

Jméno entity: SDsense

Složitost: 230 slices

Šířka vstupu: 3 bity

Podle očekávání je blok SDsense přesně duální k stimulačnímu sigma-delta generátoru SD. Umožňuje jednak jednoduché měření PDM bitstreamů generovaných jednoduchými sigma-delta modulátory (potom se je jeho funkce stejná, jako funkce bloku PDMsense), dále však poskytuje možnost zpracování vstupních signálů decimačními filtry až do 3. řádu, takže je schopen zpracovat signál sigma-delta modulátoru s architekturou MASH.

Tyto decimační filtry byly navrženy a implementovány hlavně na základě požadavků a instrukcí firmy AMI Semiconductor Czech, s.r.o., jejich návrh tedy vychází především z [3] a [4], dále byl mj. použit např. pramen [8].

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	measComplete	iNserv	iReq		delayStart	integB	integA [†]	enabled
1	pdmA	pdmB	pdmC			dAgreater	dAequal	dAless
2	Plength (bitstream period length)							
3	CompRef (comparison reference value) LSB							
4	...							
5	MSB							
6	Measured (last measured value) LSB							
7	...							
8	MSB							
9	CurrMeasAcc (current measurement value accumulator) LSB							
A	...							
B	MSB							
C	CurrMeasPeriod (current measured period counter)							
D	Int2 (current 2nd-order weight coefficient)							
E	Int3 (current 3rd-order weight coefficient) LSB							
F	MSB							

[†] if integB set then integA is set and cannot be cleared

tab. 24. Paměťový prostor bloku SDsense

Možnosti konfigurace bloku

Registrem *Plength* zadáváme délku periody, přes kterou se bude decimace provádět (v rozsahu 1 až 256, nula v tomto registru tedy odpovídá délce periody 1). Bit *measComplete* indikuje, že této periody bylo již aspoň jednou dosaženo. *CompRef* udává referenci pro komparátory změřených hodnot generující signály okamžitých akcí. Co jsou to vlastně u tohoto bloku naměřené hodnoty, bude vysvětleno níže.

Konfigurace decimačních filtrů

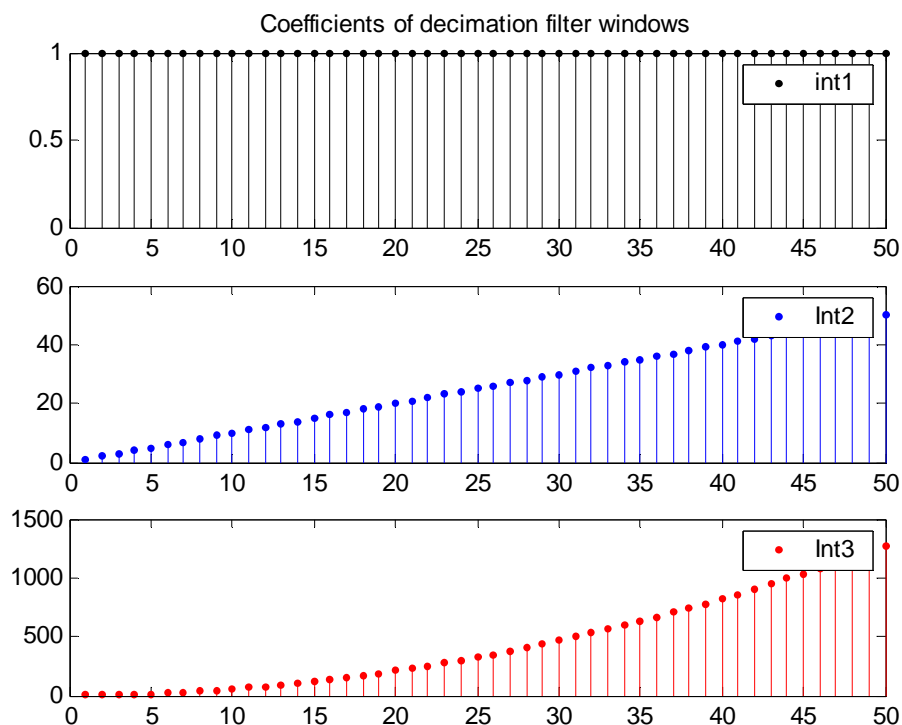
Samotná decimace probíhá tak, že během nastavené doby periody se vzorky násobí koeficientem decimačního okna a sčítají.

V základním stavu jsou všechny decimační filtry nastaveny na prosté obdélníkové váhovací okno o výšce jedna (jinými slovy všechny vzorky se násobí jedničkou a po dobu periody sčítají – jak již bylo řečeno, tak funguje i blok PDMsense).

Pokud nastavíme bit *integA*, bude filtr vstupu *pdmA* povýšen na trojúhelníkový filtr (v následujícím výňatku zdrojového kódu budou koeficienty jeho okna dané *Int2*). Jestliže nastavíme i bit *integB*, trojúhelníkovým filtrem se stane filtr na vstupu *pdmB* a filtr vstupu *pdmA* bude povýšen o další řád, jeho koeficienty pak budou dány *Int3*:

```
int1 <= 1;
Int2 <= Int2 + int1;
Int3 <= Int3 + Int2 + int1;
```

Tyto filtry však nejsou klasickými číslicovými filtry, jsou ty spíše jakási váhovací okna, kterými se měřené koeficienty násobí. Pro představu je lepší graf na obr. 15.



obr. 15. Koeficienty váhovacích oken decimačních filtrů

Při použití těchto filtrů je dále zajištěno, že vzorky decimované filtrem nižšího řádu jsou zpožděny o jeden takt hodin oproti vzorkům na vstupech s filtrem vyššího řádu. Je to z toho důvodu, že při použití MASH struktury jsou první vzorky na výstupu druhého a třetího řádu platné až po jednom nebo dvou taktích hodin od spuštění modulace a každá změna na vstupu se na těchto výstupech objeví zpožděna o stejný počet hodinových period oproti výstupu prvního řádu, viz obr. 10. Činnost decimačních filtrů shrnují tab. 25, tab. 26 a tab. 27.

clock period	0	1	2	3	4	5	6	...
pdmA samples	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	...
pdmA samples weight	1	1	1	1	1	1	1	...
pdmB samples	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	...
pdmB samples weight	1	1	1	1	1	1	1	...
pdmC samples	c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	...
pdmC samples weight	1	1	1	1	1	1	1	...

tab. 25. Váhování měřených vzorků při vypnuté integraci váhovacích koeficientů

clock period	0	1	2	3	4	5	6	...
pdmA samples		a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	...
pdmA samples weight	-	1	2	3	4	5	6	...
pdmB samples	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	...
pdmB samples weight	-	1	1	1	1	1	1	...
pdmC samples	c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	...
pdmC samples weight	-	1	1	1	1	1	1	...

tab. 26. Váhování a zpoždění vzorků při integraci koeficientů A

clock period	0	1	2	3	4	5	6	...
pdmA samples			a[0]	a[1]	a[2]	a[3]	a[4]	...
pdmA samples weight	-	-	1	3	6	10	15	...
pdmB samples		b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	...
pdmB samples weight	-	-	1	2	3	4	5	...
pdmC samples	c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	...
pdmC samples weight	-	-	1	1	1	1	1	...

tab. 27. Váhování a zpoždění vzorků při integraci koeficientů A a B

Ve „vyšších“ režimech decimace jsou tedy vzorky ze vstupů *pdmA* a/nebo *pdmB* zpožděny o jeden nebo dva takty hodin a vlastní výpočet začíná v momentě označeném dvojí čarou. Vzorky před tímto okamžikem jsou ignorovány, u MASH struktury totiž stejně nenesou žádnou informaci. Jestliže zvolíme režim podle tab. 26, jsou vstupy *pdmB* i *pdmC* váhovány obdélníkovým oknem, oba jsou však zpožděny o jeden takt hodin (a jejich první vzorek je zahozen). Na to je třeba dát pozor, pokud bychom chtěli tento režim použít pro měření MASH struktury 2. řádu a dalšího nezávislého modulátoru 1. řádu. Z předchozího též vyplývá, že doba měření první periody bude o jeden resp. dva hodinové takty delší, než všechny následující.

Striktně vzato, podle teorie těchto filtrů by navržená okna měla být otočená (tj. měla by začínat největším koeficientem a ne jedničkou). Pak by však nebylo možné navrhnout okno třetího řádu, protože jeho poslední koeficient je dán poměrně složitou rovnicí, kterou nelze v jednoduchém digitálním systému realizovat. Podle [3] a [4] je však možné s úspěchem použít i právě popsaný postup a v praxi se tak běžně děje.

Nastavení popsaným režimů se děje pomocí bitů *integA* a *integB* řídicího registru. Podle zvolené konfigurace se pak nakládá i s výsledky decimace:

- Vynulováním obou bitů nastavíme režim popsaný tab. 25. V tomto režimu se výstupy všech tří decimálních filtrů ukládají do oddělených paměťových prostor – bytů registru *Measured*.
- Nastavením *integA* do 1 a *integB* do 0 se přepneme do režimu podle tab. 26. Při tomto nastavení se výstup decimace *pdmC* ukládá do nejnižšího byte registru *Measured*, výsledky decimace bitstreamů *pdmB* a *pdmA* se sčítají a ukládají do horních dvou bytů registru *Measured*.

- Při nastavení bitu *integB* do 1 je nastaven i bit *integA* do jedničky (kombinaci *integA* = 0 a *integB* = 1 tedy nastavit nelze) a je aktivována konfigurace decimálních filtrů popsaná v tab. 27. V tomto režimu se sčítají výsledky váhování všech tří vstupů, což odpovídá měření signálů generovaných MASH sigma-delta strukturou 3. řádu a ukládají se do registru **Measured** v celé jeho velikosti.

Registry změřené hodnoty a kontrolní registry

Pro objasnění použitého způsobu ukládání naměřených hodnot bude třeba vypočítat pár jednoduchých rovnic. Maximální hodnota, kterou může dosáhnout akumulátor decimovaných vzorků při váhování jednoduchým obdélníkovým oknem je

$$(4) \quad \sum_{j=1}^P 1 = P,$$

kde P je maximální délka periody. V našem případě je $P = 2^8 = 256$.

Pro akumulátor výsledků této jednoduché decimace tedy potřebujeme registr o velikosti

$$(5) \quad \log_2 P = \log_2 2^8 = 8b.$$

Při váhování oknem trojúhelníkového typu (*Int2*), pokud bude vstup trvale v jedničce, dojdeme k maximální hodnotě akumulátoru

$$(6) \quad \sum_{j=1}^P \sum_{i=1}^j 1 = \sum_{j=1}^P j = \frac{1}{2} P(P+1),$$

potřebná velikost registru pro takový akumulátor pak bude (pro $P \geq 1$)

$$(7) \quad \log_2 \frac{1}{2} P(P+1) \leq \log_2 \frac{1}{2} P(P+P) = \log_2 P^2 = 2 \log_2 P = 2 \log_2 2^8 = 2 \cdot 8 = 16b.$$

Podobně pro váhování oknem inkrementálního typu (*Int3*) dostaneme

$$(8) \quad \sum_{k=1}^P \sum_{j=1}^k \sum_{i=1}^j 1 = \frac{1}{6} P(P+1)(P+2),$$

takže pro binární akumulátor budeme potřebovat registr o velikosti ($P \geq 1$)

$$(9) \quad \log_2 \frac{1}{6} P(P+1)(P+2) \leq \log_2 \frac{1}{6} P(P+P)(P+2P) = \log_2 P^3 = 3 \log_2 2^8 = 24b.$$

Z uvedeného tedy plyne, že pro ukládání výsledků decimace podle tab. 25 budeme potřebovat tři osmibitové akumulátory, pro decimaci podle tab. 26 bude třeba jednoho 16bitového akumulátoru a jednoho 8bitového a konečně pro režim třetího řádu podle tab. 27 budeme k ukládání výsledků potřebovat jeden 24bitový registr.

Pro registry koeficientů *Int2* a *Int3* platí stejné rovnice pouze snižené o jeden řád; pro *Int2* tak podle (4) a (5) bude nutné použít 8bitový registr, pro *Int3* podle (6) a (7) 16bitový registr. Z tab. 27 je vidět, že tomu tak skutečně je.

Registr **CurrMeasAcc** je pracovním registrem akumulátoru změřených hodnot a v každém taktu se k němu (nebo k některé jeho části) přičítají vstupní vzorky násobené příslušným koeficientem. Na konci periody je pak hodnota tohoto registru přkopírována do **Measured**, samotný registr **CurrMeasAcc** je vynulován a decimace začíná znova od začátku.

Aktuální pozici v PDM periodě udává čítač *CurrMeasPeriod*. Tento čítač čítá od nuly a při dosažení *Plength* je nastaven bit *measComplete*, atd.

Bitsy *pdmA*, *pdmB* a *pdmC* odkazují přímo na příslušné vstupy a budou tedy obsahovat jejich aktuální hodnotu. Bitsy *dAGreater*, *dAequal* a *dALess* pak obsahují aktuální stav výstupů okamžitých akcí.

Požadavek přerušení

Přerušení je vyžadováno opět po každé změřené periodě, následující perioda totiž předchozí změřenou hodnotu přepíše. Pro bit *iNserv* tu funguje stejný mechanismus, jako u několika předchozích bloků – tento bit není nastaven když požadavek přerušení není třeba i vícekrát obslužen, ale nová změřená hodnota je stále rovná minulé.

C.4. Pomocné bloky

C.4.1. Blok paměťového úložiště

Jméno entity: MBmem

Složitost: 59 slices

Modul MBmem je pomocným blokem měřicích bloků a poskytuje možnost ukládat naměřená data do bokové paměti RAM použitého FPGA. Využívá stejný blok přístupu k této paměti jako některé stimulační bloky, tedy XC3S_RAMB_2_PORT; jeho popis je v kap. C.2.4. Protože počet blokových pamětí je omezen, má tento blok celkem čtyři identická vstupní rozhraní a podle konfigurace umožní ukládat data pouze jednomu z nich. Zajímavý je také tím, že nemá vstup pro signál globální aktivace *gEnable*; je aktivován přímo signály požadavku zápisu od připojených měřicích bloků.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0		iNserv	iReq			SrcSelect		enabled
1	RAMDataIn (current RAM block input data)							
2	RAMwriteAddr (current RAM writing address) LSB							
3						MSB		...
4								
5								
6								
7								
8								
9								
A								
B								
C								
D	RAMAddress (RAM data write address) LSB							
E						MSB		...
F	RAMDataByte (RAM data for both reading and writing)							

tab. 28. Paměťový prostor bloku MBmem

Rozhraní pro příjem dat

Rozhraní vstupních signálů od měřicích bloků je definováno signály podle tab. 29. Jejich funkce vyplývá z výše popsaného, data *Inx* jsou zapsána když je signál *inxValid* aktivní (H). Tento signál tedy může být aktivní pouze jednu periodu hodin, jinak jsou vstupní data zapisována kontinuálně – což je však někdy také možné.

jméno signálu	počet ve sběrnici	poznámka
in1Valid	1	vstup bloku 1
In1	8	
in2Valid	1	vstup bloku 2
In2	8	
in1Valid	1	vstup bloku 3
In3	8	
in1Valid	1	vstup bloku 4
In4	8	

tab. 29. Signály vstupního rozhraní bloku MBmem

Konfigurace

Konfigurace se zde omezuje na dva bity, kterými jsou bity výběru vstupu **SrcSelect**. Podle jejich nastavení jsou vstupní data přijímána ze vstupu 1, 2, 3 nebo 4 (tab. 30). Ostatní vstupy jsou ignorovány.

SrcSelect	Selected Input
00	In1, in1Valid
01	In2, in2Valid
10	In3, in3Valid
11	In4, in4Valid

tab. 30. Výběr vstupu v závislosti na hodnotě konfiguračního registru *SrcSelect*

Zápis a čtení z/do bloku paměti

Přístup k paměti RAM je implementován naprosto stejným způsobem, jako u bloku MemBPG (viz kap. C.2.4) a to včetně možnosti zápisu. Dokonce jsou zachovány i adresy příslušných registrů. Registr *RAMAddress* slouží k zadání požadované adresy a k datům se dostaneme přes registr *RAMDataByte*. Signál zápisu je odvozen ze vstupu *wEn* bloku, takže čtením registru *RAMDataByte* čteme obsah paměti na zvolené adrese, zápisem tohoto registru na zvolenou adresu zapisujeme.

Konkrétně je pomocný blok XC3S_RAMB_2_PORT použit tak, že port A je napojen na externí přístup (tedy na registry *RAMAddress* a *RAMDataByte*) a přes port B jsou zapisována vstupní data. Port A je aktivován adresou registru nastavenou na F (adresa bytu *RAMDataByte*), port B je aktivován, pokud je nastaven *enabled* a zvolený vstupní signál požadavku zápisu je aktivní.

Funkce bloku a registry kontroly stavu

Vstupní data *In1*, *In2*, *In3* nebo *In4* jsou zrcadlena do registru *RAMDataIn*.

Důležitý registr *RAMwriteAddr* udává aktuální adresu zápisu dat (resp. adresu zápisu příštího byte). V každém taktu hodin, ve kterém je zvolený vstupní signál požadavku zápisu *inxValid* aktivní, jsou data zapsána na adresu *RAMwriteAddr* a tato hodnota je zvětšena o jedničku.

Data jsou zapisována kontinuálně od adresy 000 až po nejvyšší (7FF). Čtením tohoto registru tedy zjistíme, kolik dat již bylo zapsáno. Tento registr je nulován při vynulování bitu *enabled*, jinak jej nelze ovlivnit. Při dosažení konce paměti je generováno přerušení.

Požadavek přerušení

Přerušení je vyžadováno při dosažení konce paměti, tj. 7FF. Následující adresa je pak 000. Toto je pevně dáno, jelikož pevně dána je i počáteční adresa (000). Je na řídicím počítači, aby průběžně četl aktuální zapisovanou adresu a zapsaná data si správně poskládal.

D. PROPOJENÍ A SPOLUPRÁCE BLOKŮ

Máme navrženo rozhraní a protokol komunikace s hostitelským počítačem včetně potřebných registrů, je vypracován systém několika typů vnitřních komunikačních sběrnic, máme celkem širokou nabídku bloků, které můžeme na tyto sběrnice připojit. V tomto okamžiku je tedy hotový návrh skutečně použitelné architektury čipu. Má pouze jeden nedostatek a tím je spolupráce bloků v real-time režimu. I k tomu jsou však všechny bloky již dobře vybaveny, všechny umějí generovat resp. zpracovávat signály okamžitých akcí. Zbývá je pouze nějak propojit.

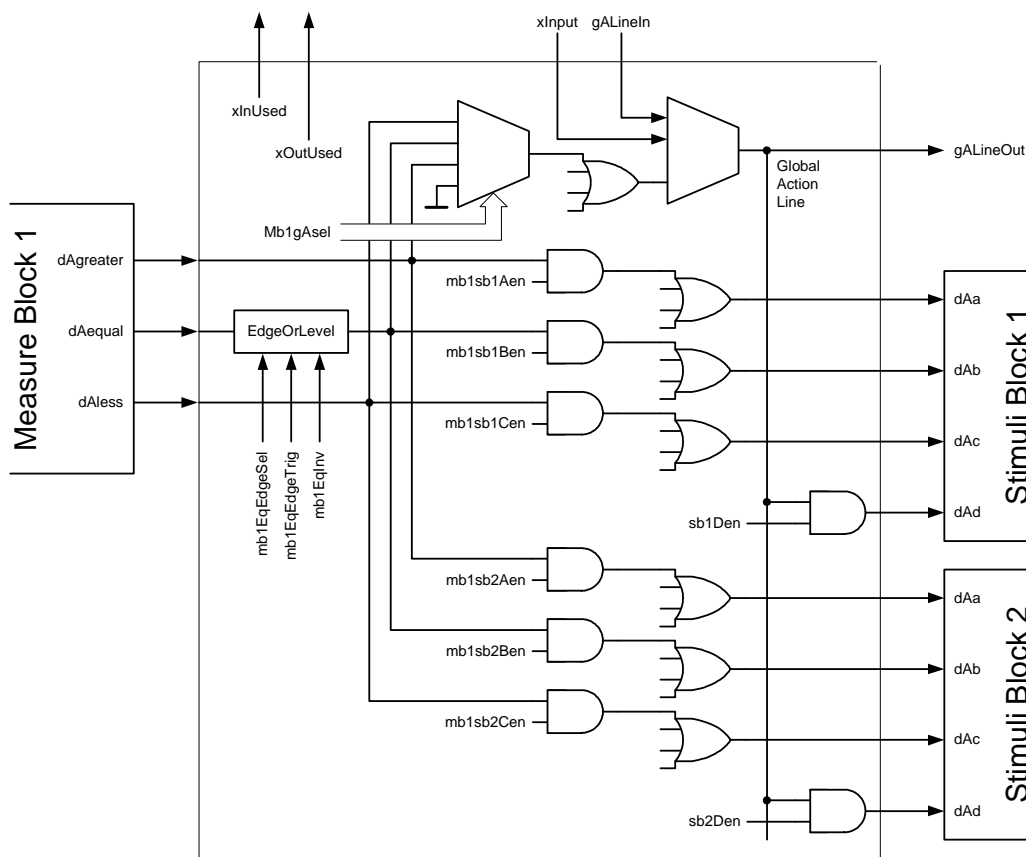
Za tímto účelem byla vyvinuta speciální propojovací matice, která umožňuje konfigurovatelné propojení bloků a navíc umí zajistit i spolupráci v real-time módu s vnějšími systémy.

D.1. Univerzální propojovací matice

Jméno entity: Interconnect

Složitost: 147 slices

Univerzální tato matice je, ale jen do jisté míry, neumí propojit libovolné množství bloků, maximálně lze spojit čtyři vstupní se čtyřmi výstupními bloky (důvod je jasný, únosná míra složitosti a náročnosti designu). I tak však skýtá poměrně velké možnosti. Zjednodušené blokové schéma je na obr. 16; pro větší přehlednost je zde vyznačeno připojení pouze jednoho měřicího bloku a dvou stimulačních.



obr. 16. Orientační blokové schéma propojovací matice Interconnect

Vyznačená jména signálů korespondují se stejnojmennými registry paměťového prostoru tohoto bloku (tab. 31). Ano, i tento blok je samozřejmě vybaven základními komunikačními sběrnici; nemá však vstup *gEnable*, výstup *iRQ* ani žádné vstupy signálů přímých akcí, které by on sám obsluhoval.

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0						xOutUsed	gAlineInDrv	xInUsed
1	Mb4gAsel		Mb3gAsel		Mb2gAsel		Mb1gAsel	
2						mb1EqEdgeSel	mb1EqEdgeTrig	mb1EqInv
3						mb2EqEdgeSel	mb2EqEdgeTrig	mb2EqInv
4						mb3EqEdgeSel	mb3EqEdgeTrig	mb3EqInv
5						mb4EqEdgeSel	mb4EqEdgeTrig	mb4EqInv
6		mb2sb1Cen	mb2sb1Ben	mb2sb1Aen	sb1Den	mb1sb1Cen	mb1sb1Ben	mb1sb1Aen
7		mb4sb1Cen	mb4sb1Ben	mb4sb1Aen		mb3sb1Cen	mb3sb1Ben	mb3sb1Aen
8		mb2sb2Cen	mb2sb2Ben	mb2sb2Aen	sb2Den	mb1sb2Cen	mb1sb2Ben	mb1sb2Aen
9		mb4sb2Cen	mb4sb2Ben	mb4sb2Aen		mb3sb2Cen	mb3sb2Ben	mb3sb2Aen
A		mb2sb3Cen	mb2sb3Ben	mb2sb3Aen	sb3Den	mb1sb3Cen	mb1sb3Ben	mb1sb3Aen
B		mb4sb3Cen	mb4sb3Ben	mb4sb3Aen		mb3sb3Cen	mb3sb3Ben	mb3sb3Aen
C		mb2sb4Cen	mb2sb4Ben	mb2sb4Aen	sb4Den	mb1sb4Cen	mb1sb4Ben	mb1sb4Aen
D		mb4sb4Cen	mb4sb4Ben	mb4sb4Aen		mb3sb4Cen	mb3sb4Ben	mb3sb4Aen
E								
F								

tab. 31. Paměťový prostor propojovací matice Interconnect

Konfigurace matice

Hlavní konfigurační bity jsou v registrech na adresách 6 až D. Tyto bity řídí propojení kteréhokoliv z měřicích bloků s kterýmkoliv z připojených stimulačních bloků podle schématu. Propojují se však vždy signál *dAgreater* měřicího bloku na vstup *dAa* stimulačního bloku, signál *dAequal* na *dAb* a signál *dAless* na *dAc* vstup stimulačního bloku. Další přepojení je možné udělat uvnitř stimulačních bloků (viz obr. 7). Pokud přivedeme více signálů do jednoho výstupu, jejich funkce se sčítá (na obr. 16 vyznačeno čtyřnásobnými OR hradly).

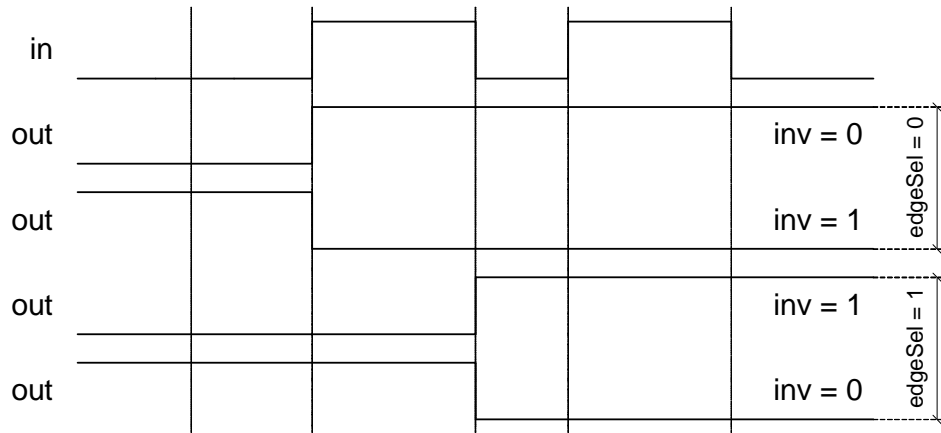
Pomocný blok zachycení hrany

Jméno entity: EdgeOrLevel

Složitost: 5 slices

Výstup *dAequal* měřicích bloků je asi nejdůležitější ze všech tří signálů okamžitých akcí, ale jeho chování je poněkud nevhodné. Při proměnné měřené hodnotě bude nastavován a resetován v závislosti na výsledku aktuálního měření, což ne vždy vyhovuje. Propojovací matice proto obsahuje pomocné bloky EdgeOrLevel, které umí aktivní hodnotu tohoto signálu zachytit a uchovat neomezeně dlouhou dobu. Jejich funkce je řízena třemi konfiguračními bity registrů propojovací matice:

- Bit *mbxEqInv* nastaví inverzi signálu *dAequal* příslušného měřicího bloku.
- Bitem *mbxEdgeDet* umožníme zachycení aktivní úrovně. Chování v tomto režimu je nejlépe zdokumentováno na obr. 17. Pokud je tento bit vynulován, je signál propouštěn staticky (pouze s volitelnou inverzí).
- Při nastaveném bitu *mbxEdgeDet* bit *mbxEdgeSel* vybírá hranu, která způsobí zachycení hodnoty.



obr. 17. Ilustrace funkce pomocného bloku *EdgeOrLevel*

Jména signálů použitých v obr. 17 se vztahují k portům bloku *EdgeOrLevel*. Na úrovni modulu *Interconnect* odpovídají *in* vstupnímu signálu *dAequal*, *out* je výstupem z bloku *EdgeOrLevel*, *inv* odpovídá *mbxEqInv* a *edgeSel* pak stavu bitu *mbxEqEdgeSel*.

V režimu zachycení aktivní úrovně je možné zachycenou hodnotu resetovat opětovným zápisem do těchto konfiguračních bitů. Na výstupu pomocného bloku *EdgeOrLevel* je potom úroveň opačná a blok je tak připraven na zachycení další změny signálu *dAequal*.

D.1.1. Zajištění spolupráce mezi více systémy

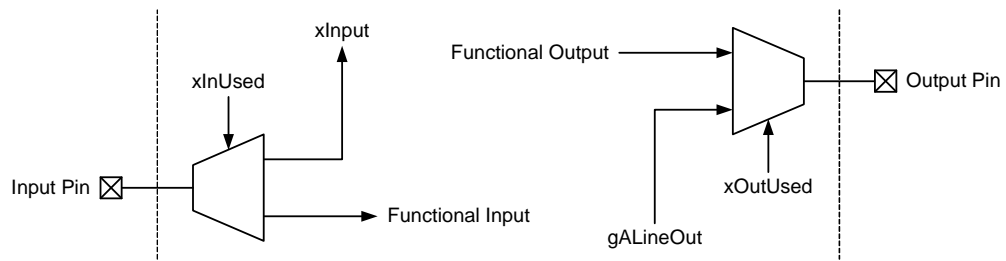
K tomuto účelu byla v matici *Interconnect* rozvedena linka GA (global action). Tato je připojena na vstupy *dAd* stimulačních bloků (toto připojení lze deaktivovat bity *sbxDen*) a jako jediná je vyvedena i ven z bloku matice. Konfigurovat lze i signál, který ji řídí, a to podle tab. 32.

<i>xInUsed</i>	<i>gALineDrv</i>	GA line driver
1	X	<i>xInput</i>
0	1	<i>gaLineIn</i>
0	0	<i>gALineHomeDrv</i>

tab. 32. Konfigurace signálu řídicího GA linku

Signál *gALineHomeDrv* odpovídá výstupům multiplexorů řízených *MbxGASel* (od všech čtyř měřicích bloků, spojených OR hradlem).

Dále v bloku propojovací matice existují registry *xInUsed* a *xOutUsed*. Předpokládané využití všech těchto portů je následující: vstup *gALineIn* bude napojen na výstup *gALineOut* jiné matice – té, se kterou se předpokládá častá spolupráce. Signály bitů *xInUsed* a *xOutUsed* budou řídit multiplexory/demultiplexory zařazené na zvolených funkčních vstupech/výstupech systému (přímo na pinech použitého hardwaru). Pokud budou nulové, budou tyto multiplexory přepnuty na cestu z pinu na zvolený funkční vstup/výstup, což je normální stav. Pokud však budou nastaveny do aktivní úrovně, přepojí signál z tohoto pinu na vstup *xInput* propojovací matice nebo na výstup *gALineOut*. Tím se z takového pinu stane prostředek k hardwarovému propojení více systémů takovým způsobem, že spolu budou moci spolupracovat i v real-time módu. Navíc bude zachován směr toku dat, takže všechny piny, které byly výstupní, zůstanou výstupní a naopak; tím se vyhneme případným dost nepříjemným konfliktům. Popsané chování ilustruje obr. 18, ale není zdaleka jediným možným, záleží jen na fantazii uživatele, jak navržené signály použije.



obr. 18. Návrh pro konfiguraci pinů systému do rozhraní spolupráce více systémů

E. POZNÁMKY K REALIZACI

Většina bloků byla syntetizována a odzkoušena na experimentální desce XEM3001v2 zakoupené od firmy Opal Kelly, Inc. Deska obsahuje především obvod FPGA řady Spartan3, (XC3S400), dále je vybavena USB portem a mikroprocesorem (s jádrem 8051), který obstarává komunikaci přes toto rozhraní. Dále je zde konfigurovatelný obvod PLL, který umožňuje generovat hodinový signál v širokém rozmezí kmitočtů, potom sériové paměti pro uložení nastavených parametrů (ne designu) a některé další pomocné obvody. Modul velmi dobře funguje i při napájení přímo z USB. Ilustrační foto je na obr. 19.



obr. 19. Použitý modul pro hardwarovou implementaci

Pro umožnění komunikace přes USB musely být do designu přidány některé bloky dodávané firmou Opal Kelly, Inc. jako součást produktu; tyto bloky pak zajistily komunikaci FPGA s použitým USB kontrolérem přes jakýsi proprietární protokol. Vlastní hostitelské rozhraní našeho systému (tak, jak bylo popsáno v části B) bylo tedy realizováno vlastně až uvnitř čipu FPGA. V budoucnosti se počítá s tím, že bude vyvinut vlastní program pro USB kontrolér, který bude umět komunikovat přes naše rozhraní bez prostředníka.

E.1. Časové nároky

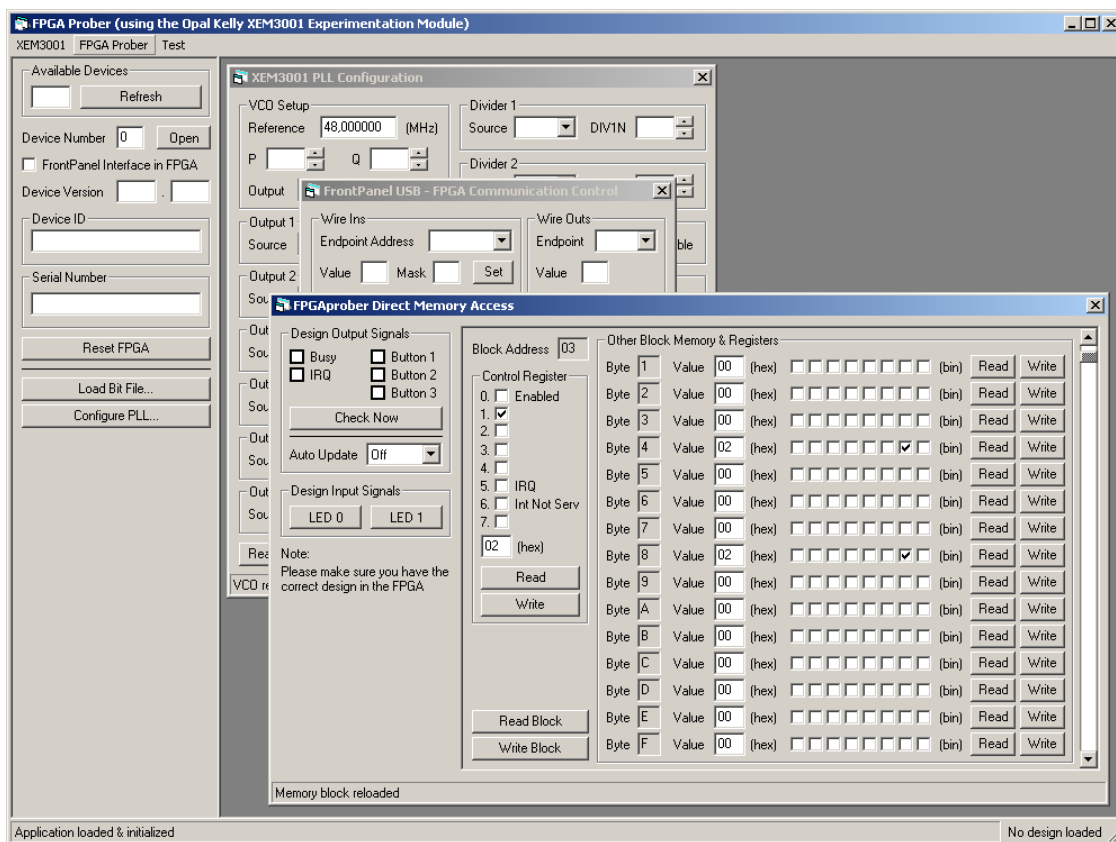
Jednotlivé bloky návrhu byly testovány samostatně s hodinovým kmitočtem 48 MHz a nebyly zaznamenány žádné problémy. Je však pravda, že některé se mohly objevit aniž by byly odhaleny. Při syntéze v prostředí WebPack jsou totiž hlášeny maximální hodnoty frekvence hodinového kmitočtu nižší, pro složitější bloky třeba jen do 10MHz. Předpokládá se, že při propojení více bloků propojovací maticí tato hodnota dále poklesne.

Konkrétní čísla zde nebudou uvedena, záleží totiž na konkrétním výběru a zapojení bloků v architektuře. Na příloženém CD je konkrétní příklad syntetizované architektury i s popisem; nic nebrání případnému zájemci, aby si poskládal vlastní návrh a při syntéze mu bude maximální hodinový kmitočet sdělen.

Je také pravda, že všechny bloky mohou fungovat i samostatně, pokud se nepředpokládá využití real-time režimu. Tím je možné omezení maximálního hodinového kmitočtu, které je dané maximálním zpožděním kombinační části, obejít a je vysoce pravděpodobné, že nejrychlejší např. paměťové bloky budou dobře fungovat i na vyšších desítkách MHz.

E.2. Software pro řídicí PC

Dále byl vytvořen software pro PC, který umí s modulem XEM3001v2 komunikovat. Děje se tak opět prostřednictvím ovladačů a knihoven dodávaných firmou Opal Kelly, Inc. Bude uveden pouze ilustrační obrázek grafického uživatelského rozhraní (GUI) tohoto programu (obr. 20).



obr. 20. Ukázka grafického rozhraní programu FPGAprober

F. ZÁVĚR

Byla navržena architektura číslicového čipu podle požadavků zadání. Navržený systém je primárně určen pro testování digitálních obvodů, je však docela dobře použitelný i pro měření nebo generování stimulačních signálů pro některé analogové obvody. Není však zaručeno, že logické výstupy budou bez parazitních impulsů (glitches); na většině výstupů je totiž více či méně jednoduchá kombinační logika z důvodu možnosti okamžité reakce na vnitřní signály.

Požadavek možnosti tzv. real-time módu byl splněn tak, jak jen to bylo možné. Nebylo proveditelné a snad ani nutné vytvářet možnosti propojení absolutně kteréhokoliv měřicího bloku s kterýmkoliv stimulačním; místo toho se předpokládá, že makrobloků skládajících se ze čtyř měřicích a čtyř stimulačních bloků a jejich propojovací matice bude použito více a konkrétní stimulační a měřicí bloky budou vybrány v takových kombinacích, jejichž spolupráce se předpokládá nejčastěji. I tak však navržený systém propojení poskytuje mechanismy, které zajistí spolupráci i přes hranice propojovacích matic.

V tomto projektu byla však značně upřednostněna co největší univerzálnost funkce měřicích a stimulačních bloků před rychlostí nebo jednoduchostí a to z několika důvodů. Tento systém je určen především pro testování synchronních digitálních systémů, kde se snížení hodinového kmitočtu na funkci většinou nijak neprojeví. Nemá nahrazovat existující systémy typu ATM (automatic test machine), které fungují s velkými paměťmi pro generování i měření signálů podle zadaných testovacích vektorů, ale vyžadují složitou obsluhu a přípravu i pro tak jednoduché úkoly, jako např. změření délky periody PWM signálu. Speciální techniky typu pipelining pro urychlení práce (a zvýšení hodinového kmitočtu) zde rovněž nebyly použity, protože velký důraz byl kladen na okamžitou reakci systému, požadavek tzv. okamžité akce je každým měřicím blokem zpracován ihned při další hraně hodinového signálu.

Největším kladem navržené architektury je však její otevřenost. Na popsany systém sběrnice lze „navěsit“ prakticky libovolné bloky, které ani nemusí souviset s funkcí či účelem systému, stačí, když budou schopny komunikace přes tyto sběrnice a budou mít kompatibilní adresovací mechanismus svého paměťového prostoru. Pak lze s minimálním úsilím přidat libovolný takový blok, současné omezení jejich maximálního počtu je dáno osmibitovou adresou bloku (což dává možnost použít v ideálním případě 254 funkčních bloků), pokud by však bylo třeba, bude možné stávající adresní registr bloku rozšířit na více bitů. Bude pak třeba poněkud změnit i protokol komunikace přes řídicí rozhraní.

Tento projekt neřeší problémy při případném zabudování navržené architektury do většího celku, je však na to dobře připraven. Prakticky jediné, co by bylo třeba v takovém případě zajistit externě, by bylo elektronické přepojování výstupních signálů našeho systému na požadovaná místa (pokud by takový požadavek ovšem vznikl). V současném stavu, kdy je používána samostatná testovací deska, není nutné tento problém řešit, propojení s případným testovaným obvodem se zajistí tím nejuniverzálnějším způsobem, dráty.

Konkrétní volba architektury tj. konkrétní provedení výběru a propojení bloků nejsou v této zprávě uváděny; není a nebylo to ani jejím cílem. K tomuto účelu se lépe hodí příložené CD.

G. SEZNAM POUŽITÉ LITERATURY

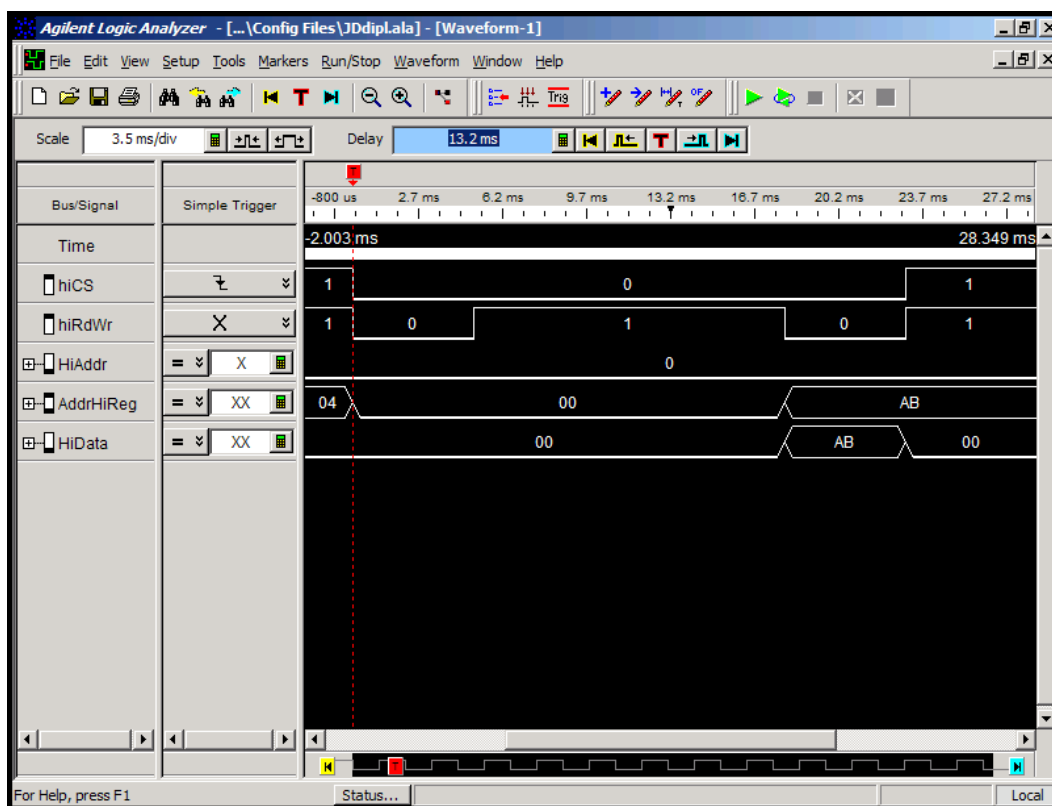
- [1] Xilinx, Inc. *Spartan-3 FPGA Family: Complete Data Sheet*. [online]. 2005. [cit. 2005-02-06].
Dostupné na WWW: <http://www.xilinx.com/bvdocs/publications/ds099.pdf>.
- [2] Opal Kelly, Inc. Champaign, IL, USA. *FrontPanel - User's Manual* [online]. 2004 [cit. 2005-01-05].
Dostupné na WWW: <http://www.opalkelly.com/library/FrontPanel-UM.pdf>.
- [3] KAMENICKÝ, Petr. *Ústní konzultace*. 23. května 2005.
- [4] HÁJEK, Rudolf. *Ústní konzultace*. Únor 2005.
- [5] DOHNAL, Jan. *Ústní konzultace*. Duben 2005.
- [6] HÁJEK, Rudolf, DOHNAL, Jan, WARCHIL, Pavel. *Ústní konzultace*. Listopad 2004.
- [7] VRBA, Kamil. *Materiály a poznámky k předmětu „Vzájemný převod analogových a digitálních signálů“*. Datum neznámé.
- [8] ROBERT, Jacques, DEVAL, Phillippe. A Second-Order High-Resolution Incremental A/D Converter with Offset and Charge Injection Compensation. *IEEE Journal of Solid-State Circuits*, 1988, vol.23, no.3, p.736-741.
- [9] KOLOUCH, Jaromír. *Programovatelné logické obvody – přednášky*. Skriptum FEKT VUT v Brně. Brno: MJ Servis, 2002, 62 s.
- [10] KOLOUCH, Jaromír. *Programovatelné logické obvody a návrh jejich aplikací v jazycích ABEL a VHDL*. Skriptum FEKT VUT v Brně. Brno: MJ Servis, 2002, 122 s.
- [11] KOLOUCH, Jaromír. *Doplněk ke skriptům Programovatelné logické obvody – přednášky*. 2004. Dostupné pouze v elektronické podobě na interní síti FEKT VUT: PLDpdn03.pdf.
- [12] MÁRKUS, János, SILVA José, TEMES Gabor C. Theory and Applications of Incremental $\Sigma\Delta$ Converters. *IEEE Transactions on Circuits and Systems—I: REGULAR PAPERS*, 2004, vol.51, no.4, p.678-690.
- [13] AVINS Jerry. *Binary / Gray code conversion*. [online]. 2005-09-28.
Dostupné na WWW: <http://www.dspguru.com/comp.dsp/tricks/alg/grayconv.htm>.
- [14] Xilinx, Inc. *Using Block RAM in Spartan-3 Generation FPGAs*. [online]. 2005-04-01. [cit. 2005-01-05]
Dostupné z WWW: <http://www.xilinx.com/bvdocs/appnotes/xapp463.pdf>.
- [15] HYDE, Daniel C. *CSCI 320 Computer Architecture Handbook on Verilog HDL*. [online]. 1997.
Dostupné na WWW: <http://www.eg.bucknell.edu/~cs320/1995-fall/manual.pdf>.
- [16] YADAV, Navindra, SCHULTE, Michael, GLOSSNER, John. Parallel Saturating Fractional Arithmetic Units. In *Ninth Great Lakes Symposium on VLSI, Ann Arbor, Michigan, 3.-4. 1999 Proceedings* [online]. 1999. [cit. 05-05-2005].
Dostupné z WWW: <http://csdl.computer.org/dl/proceedings/glsvlsi/1999/0104/00/01040214.pdf>.
- [17] SMITH, David R., FRANZON, Paul D. *Verilog Styles for Synthesis of Digital Systems*. Prentice Hall, 2000. 314 s. ISBN 0201618605.

- [18] CUMMINGS, Clifford E. *Nonblocking Assignments in Verilog Synthesis, Coding Styles That Kill!*. [online]. 2000. Dostupné z WWW: http://www.sunburst-design.com/papers/CummingsSNUG2000SJ_NBA.pdf.
- [19] Xilinx, Inc. *Language Templates*. Součást návrhového prostředí WebPack 7.1. Dostupné z WWW: <http://www.xilinx.com/webpack/index.htm>.
- [20] Xilinx, Inc. *Libraries Guide*. Součást dokumentace návrhového prostředí WebPack 7.1. 2003. Dostupné z WWW: <http://www.xilinx.com/webpack/index.htm>.
- [21] Xilinx, Inc. *Synthesis and Verification Design Guide*. Součást dokumentace návrhového prostředí WebPack 7.1. 2003. Dostupné z WWW: <http://www.xilinx.com/webpack/index.htm>.
- [22] Xilinx, Inc. *Design Consideration for HDL Implementation of Arithmetic Functions*. [online]. 2000-06-28. Dostupné z WWW: <http://www.xilinx.com/bvdocs/appnotes/xapp215.pdf>.

H. PŘÍLOHA A

Tato příloha obsahuje výsledky měření vzorku XEM3001v2 pomocí logického analyzátoru firmy AMI Semiconductor Czech, s.r.o.. Vytisknuté jsou přímo obrázky sejmuté z obrazovky přístroje. Na CD, které je součástí této práce, jsou pak stejné průběhy uloženy ve formátu .CSV, které obsahují příslušné signály navzorkované a uloženy tak, jak je logický analyzátor zachytil.

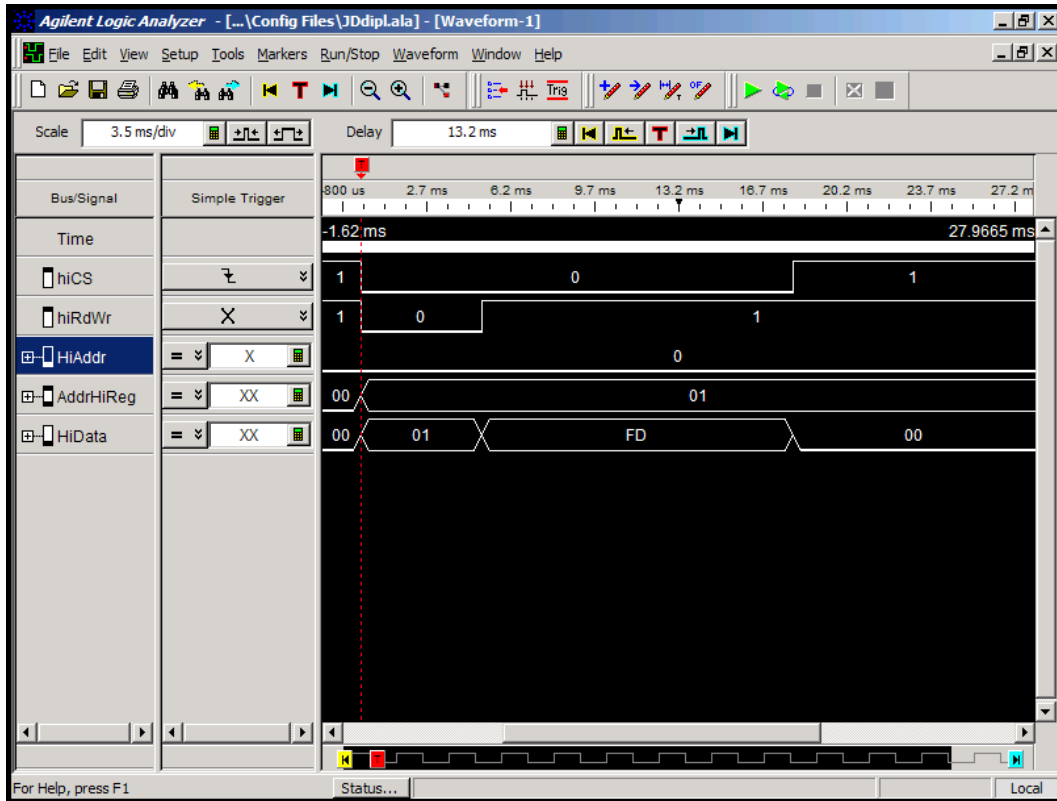
Měřené signály jsou signály hostitelského rozhraní, jak bylo popsáno v části B. Navíc jsou zde signály *AddrHiReg*, které označují výstup stejnojmenného registru a byly dočasně vyvedeny na výstup pro kontrolu funkce. Mohou dobře posloužit i k objasnění zápisu adresy bloku.



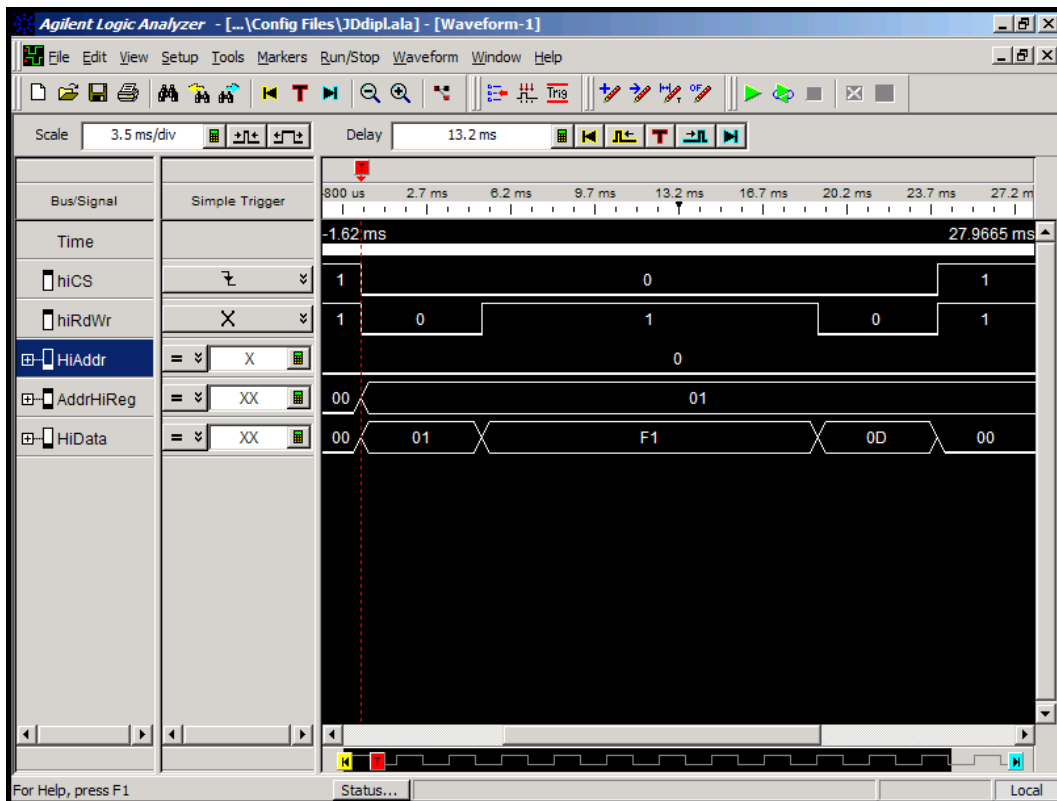
obr. 21. Zápis do hlavního registru adresy bloku

Na obr. 21 je vykresleno, co se stane, když se pokusíme zapsat do bloku na adrese 0. Tento blok totiž celý slouží jako registr adresy bloku a jeho zápisem tak vlastně přímo přepínáme cílové bloky pro zápis. Nestane se tedy nic jiného, než co popisuje obr. 1, pouze s tím rozdílem, že adresa bloku není zachycena při první, ale až při druhé vstředné hraně *hiRdWr*. Je tedy lepší tuto adresu zapsat hned jako první a ušetříme tím komunikaci.

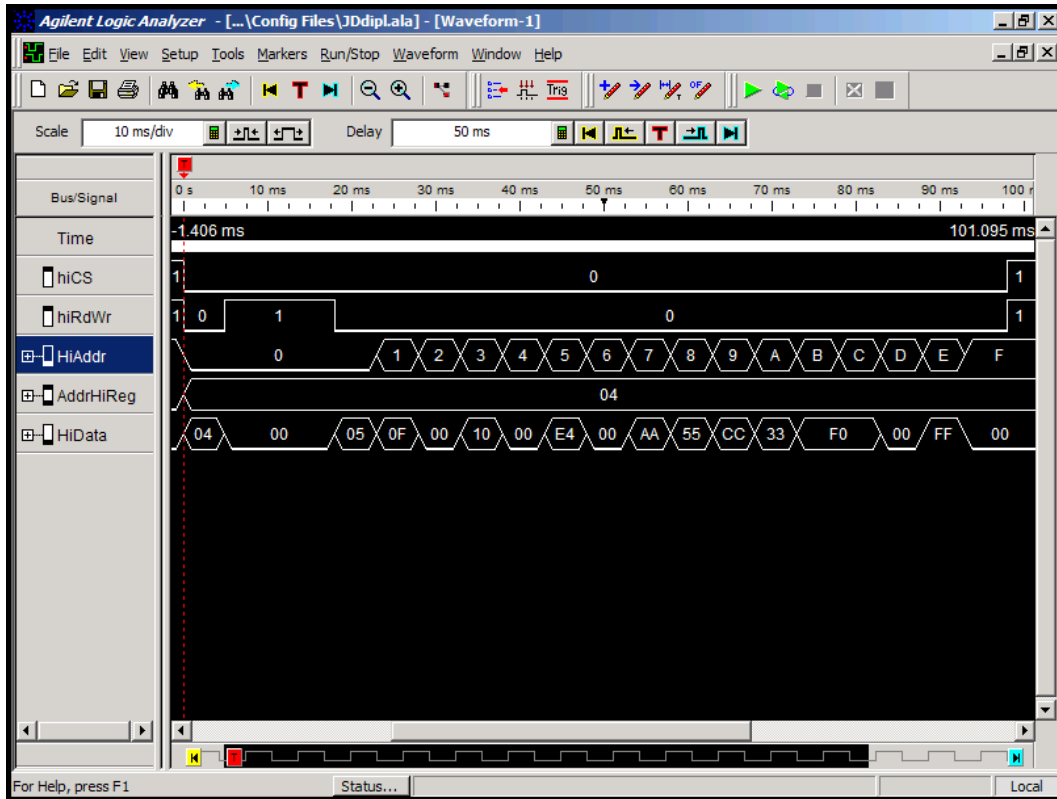
Následující obrázky jsou již platnými návody, jak číst nebo zapisovat jeden nebo všechny registry bloku.



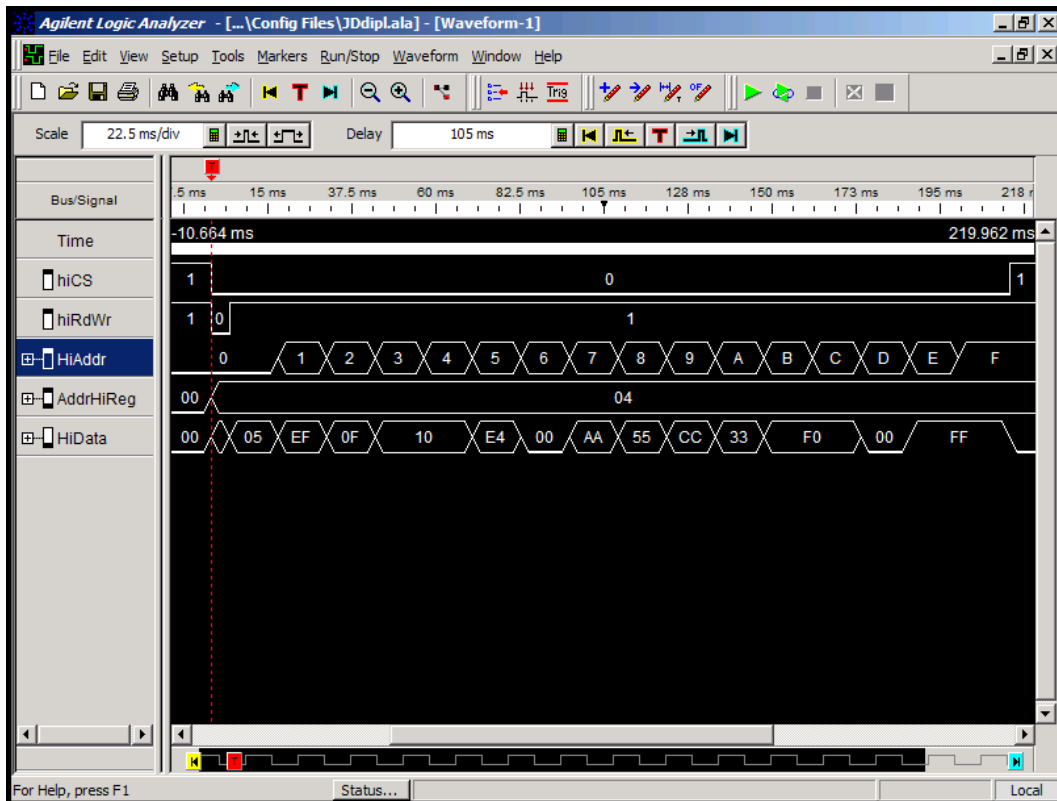
obr. 22. Čtení registru 0 bloku na adrese 1 (hlavní řídicí registr)



obr. 23. Zápis registru 0 na adrese 1 (hlavní řídicí registr)



obr. 24. Zápis všech registrů bloku (s adresou 4)



obr. 25. Čtení všech registrů bloku (na adrese 4)