

Funkční verifikace číslicových systémů

Zadání projektu: verifikace TIMERu (časovače)

Autor: Marcela Zachariášová, Petr Bardonek

Aktualizace: 2. 3. 2025

Úlohy k řešení

Úloha 1 — Seznamte se se specifikací TIMERu.

Timer DUT nastavuje aktivní hodnotu výstupního signálu přerušení (P_IRQ) po uplynutí uživatelem zadaného počtu cyklů. Přerušení může být vystaveno jednou nebo periodicky, závisí to na nastavení/hodnotách registrů timeru. Tyto registry jsou následující:

- **cnt_reg** – *timer count register*, uchovává hodnotu čítače, který se inkrementuje v každém hodinovém cyklu, kromě módu DISABLED (viz níže). Je přístupný pro čtení i zápis na adrese 8'h00.
- **cmp_reg** – *timer compare register*, uchovává hodnotu, s kterou se porovnává aktuální hodnota čítače v cnt_reg, a když jsou stejné, v následujícím cyklu se vystavuje přerušení. Jedinou výjimkou z tohoto pravidla je mód DISABLED, tzn. při shodě hodnot v cnt_reg a cmp_reg v módu DISABLED se přerušení v dalším cyklu nevystaví. Registr je přístupný pro čtení a zápis na adrese 8'h04.
- **ctrl_reg** – *timer control register*, uchovává mód timeru. Je přístupný pro čtení a zápis na adrese 8'h08. Módy timeru jsou specifikovány níže.

Timer může být použit i jako jednoduchý čítač cyklů, pro tento účel je přítomný 64 bitový registr `cycle_cnt`. Protože je ale rozhraní 32-bitové, jsou dostupné dvě adresy sloužící pro postupné vyčtení tohoto čítače: adresa 8'h10 (CYCLE_L, spodní 32b slovo) a 8'h14 (CYCLE_H, horní 32b slovo). Pro získání hodnoty je proto potřeba vyčíst nejdříve horní slovo CYCLE_H, pak spodní slovo CYCLE_L a pak znova horní slovo CYCLE_H. Důvod je ten, že vyčtení horního slova trvá jeden cyklus a spodního také – mezitím ale čítač stále inkrementuje! Proto je potřebné zkontrolovat, zda nedošlo k přetečení ze spodního slova do horního. Je tedy potřeba porovnat dvě vyčtené hodnoty CYCLE_H – když jsou stejné, k přetečení nedošlo a cykly je možné spočítat jako $CYCLE_H \ll 32 | CYCLE_L$. Když hodnoty CYCLE_H nejsou stejné, proces je třeba zopakovat. Adresy 8'h10 a 8'h14 jsou určeny jen pro čtení, pokus o zápis je ignorován. Tzn. odpověď RESPONSE na požadavek WRITE na tyto adresy je acknowledge (3'b001), tj. potvrzeno, ale hodnota čítače se nijak nezmění.

Rozhraní timeru je následující:

CLK:	vstupní synchronizační hodinový signál.
RST:	vstupní resetovací signál (pozn. všechny registry kromě cycle_cnt se resetují asynchronně).
REQUEST:	vstupní signál žádosti o čtení z registru (hodnota 2'b01), zápis do registru (hodnota 2'b10), žádná operace (hodnota 2'b00), rezervovaná hodnota (hodnota 2'b11).
ADDRESS:	vstupní adresový signál (zadání adresy registru, s kterým chceme pracovat).

DATA_IN: vstupní datový signál (zadání dat, které chceme zapisovat do registru).

RESPONSE: výstupní signál odpovědi na žádost (pozor, na každou žádost se odpovídá až v dalším cyklu!), idle (3'b000), potvrzení žádosti - acknowledge (3'b001), čekání - wait (3'b010), chyba - error (3'b011), nezarovnaná adresa – unaligned (3'b100), přístup mimo adresový prostor vyhrazený pro timer – out of range (3'b101).

DATA_OUT: výstupní datový signál (data z registru), vystavení dat je spárováno s odpovědí acknowledge na výstupu RESPONSE. Pozor, čtení z registrů cnt_reg, cmp_reg a ctrl_reg vrací hodnotu uloženou v registru v okamžiku čtení. Čtení z registru cycle_cnt vrací hodnotu, kterou má registr v okamžiku odpovědi.

P_IRQ: výstupní signál signalizující přerušení.

Vyhodnocení priority odpovědí RESPONSE (první věta má nejvyšší prioritu atd.!!!):

1. Na none („žádná operace“) REQUEST (2'b00) je odpověď vždy *idle* (3'b000).
2. Na rezervovanou hodnotu REQUESTu (2'b11) je odpověď vždy *error* (3'b011).
3. Adresování mimo adresový prostor timeru (>8'h14), odpověď je *out of range* (3'b101).
4. Nezarovnaná adresa (spodní dva bity adresy nejsou nulové), odpověď je *unaligned* (3'b100).
5. Jinak je odpověď *acknowledge* (3'b001). To znamená zápis a čtení do/ze zarovnaných adres v adresovém prostoru timeru je potvrzena, i když ne všechny tyto adresy jsou v timeru využity (využíváme celkem jen 5 registrů).
6. Odpověď *wait* se nikdy v timeru neuplatňuje.

Příklad okrajové situace:

cnt_reg a cmp_reg mají stejnou hodnotu F a ve stejném cyklu dochází k zápisu hodnoty 4 do cnt_reg. Stane se to, že P_IRQ se nastaví do 1 v dalším cyklu, a také se zapíše hodnota 4 do cnt_reg.

cnt_reg má hodnotu E, cmp_reg má hodnotu F a ve stejném cyklu dochází k zápisu hodnoty 4 do cnt_reg. Stane se to, že v dalším cyklu se jenom přepíše hodnota v cnt_reg, ale ke shodě registrů nedojde a proto se negeneruje přerušení.

	REQUEST					2						2					
	DATA_IN					4						4					
	ADDRESS					0						0					
CNT_REG	A	B	C	D	E	F	4	...	A	B	D	E	4	5	6	...	
CMP_REG	F	F	F	F	F	F	F	...	F	F	F	F	F	F	F	...	
P_IRQ	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	...	

Generické parametry určují šířku datového a adresového signálu, adresový prostor dedikovaný pro timer, a aktivní hodnotu resetu. Pro jednoduchost nebudeme tyto parametry měnit v rámci tohoto projektu.

Módy timeru jsou následující:

DISABLED (kód 00): timer je vypnutý, hodnota čítače v cnt_reg se nezvyšuje.

AUTO_RESTART (kód 01): když čítač timeru v cnt_reg dosáhne stanovený limit (hodnotu v cmp registru), v dalším cyklu se vystaví přerušení P_IRQ = 1 a začíná se čítat znovu od 0 (cnt_reg se nastaví na 0).

ONE_SHOT (kód 10):	když čítač timeru v <code>cnt_reg</code> dosáhne stanovený limit (hodnotu v <code>cmp_reg</code>), v dalším cyklu se vystaví přerušení <code>P_IRQ = 1</code> , <code>cnt_reg</code> se nastaví na 0 a timer se vypne (mód se přepne na <code>DISABLED</code>).
CONTINUOUS (kód 11):	když čítač timeru v <code>cnt_reg</code> dosáhne stanovený limit (hodnotu v <code>cmp_reg</code>), v dalším cyklu se vystaví přerušení <code>P_IRQ = 1</code> , ale čítač se neiniculuje na 0, ale pokračuje v inkrementování (až dokud nepřeteče).

Úloha 2 — Spuštění verifikace v simulátoru.

1. Stáhněte a nainstalujte si virtuální systém podle pokynů v souboru: **fvs-virtual.pdf**.
2. Stáhněte si zadání projektu do virtuálního systému.
3. Po přihlášení v terminálu spustíte následujícím příkazem verifikaci v nástroji Questa (přepněte se předtím do složky `uvm_fve` v zadání) v grafickém módu:

```
chmod +x start_verification.sh
```

```
./start_simulation.sh -uvm_testname timer_t_test -gui
```

Tohle nastavení by vám mělo stačit pro počáteční fázi verifikace.

Úloha 3 — Vytvořte verifikační plán a návrh testů (2b).

Na základě specifikace vytvořte verifikační plán se seznamem objektů (na aplikační úrovni, na úrovni rozhraní a na strukturní úrovni), které je potřeba verifikovat v timeru. Vytvořte i seznam testů, které by bylo vhodné pro každý objekt vytvořit (2b). Musí být součástí odevzdaného projektu (aktualizujte seznam podle skutečně implementovaných testů). Vytvořte jeden výsledný report v pdf, kde budou výsledky i z ostatních částí zadání.

Příklad:

Objekt (*Requirement*): Mód `ONE_SHOT`.

Testy:

[přímý test] Zápis hodnoty 0 do `cnt_reg`, hodnoty 5 do `cmp_reg`, hodnoty 1 do `ctrl_reg` v po sobě jedoucích cyklech.

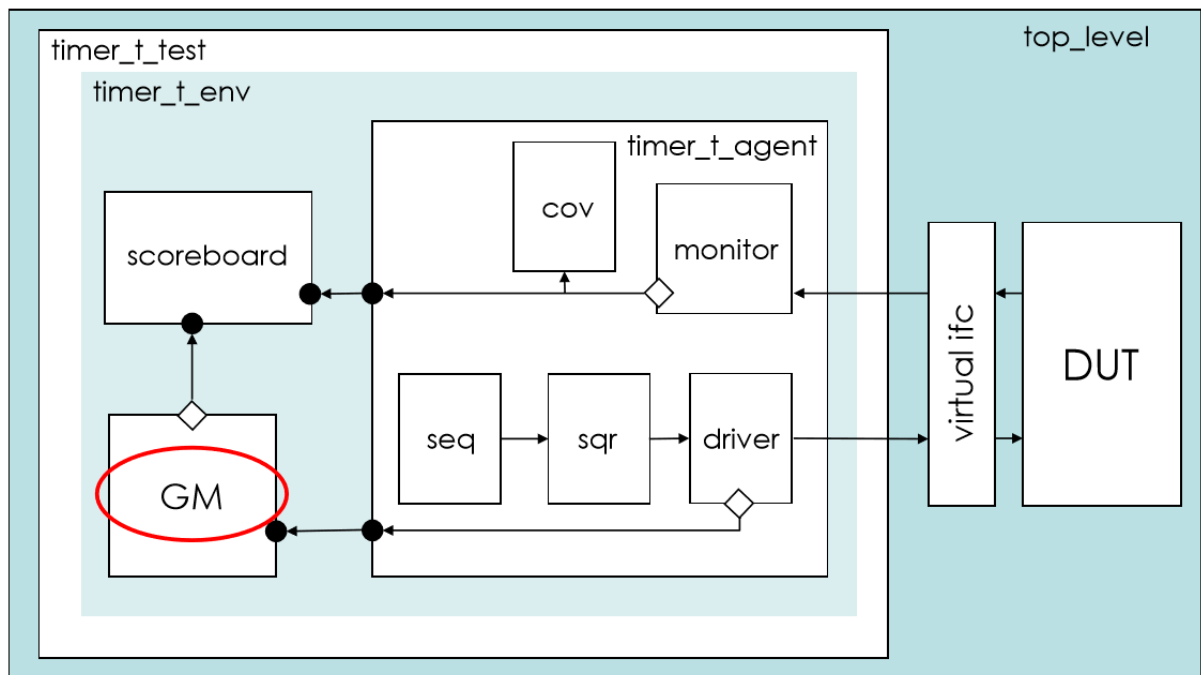
[chybový test] Žádný.

[náhodný test] Zápis náhodně generovaných hodnot do `cnt_reg` a `cmp_reg`. `ctrl_reg` má hodnotu 1.

Úloha 4 — Seznamte se s připraveným verifikačním prostředím.

Úloha 5 — Implementujte referenční model [LAB1] (3b + 2b).

Implementujte referenční model timeru (doporučení je SystemVerilog) podle instrukcí z 1. laboratoře. Ověřte jeho funkci pomocí jednoduchého předpřipraveného přímého testu `timer_t_test`. (Ověřte správnou funkčnost také bez reset sekvence!).



Referenční model by měl být implementován způsobem, že najde chybu v DUT. Řádky 85 - 90 (v timer_fvs.vhd) tuto chybu obsahují.

```

79  -----
80  -- setting of response
81  -- considering requests: read, write
82  -- address range checked during request phase
83  -- address must be aligned to 32b words
84  bus_resp_d <=
85      CP_RSP_IDLE      when ((unsigned(REQUEST) /= CP_REQ_READ) and
86                           (unsigned(REQUEST) /= CP_REQ_WRITE)) else
87      CP_RSP_ERROR     when (unsigned(REQUEST) = CP_REQ_RESERVED) else
88      CP_RSP_UNALIGNED when (ADDRESS(1 downto 0) /= "00") else
89      CP_RSP_OOR       when (unsigned(ADDRESS(ADDR_WIDTH - 1 downto TIMER_ADDR_SPACE_BITS)) /= 0) else
90      CP_RSP_ACK;

```

Je na Vás, zda chybu opravíte. Když ji opravíte, stačí v odevzdaném řešení vyznačit opravu v souboru timer_fvs.vhd. V tomto případě musí být výsledek verifikace OK. Jinak požaduji detailní popis objevené chyby/chyb. Doporučuji řešení 1, ulehčí vám to práci na dalších částech zadání. (2b)

Kontrola (a hodnocení) referenčního modelu bude v jeho funkčnosti ne v implementaci (3b).

Úloha 6 — Implementujte pseudo-náhodný test(y) [LAB2] (3b).

Implementujte pseudo-náhodný test pro timer podle instrukcí z 2. laboratoře, který testuje všechny módy timeru. Implementujte správně omezující podmínky pro generátor:

RESET: neaktivní hodnota s váhou 20, aktivní hodnota s váhou 1.

ADDRESS: rozložení vah: adresa TIMER_CNT s váhou 7, adresa TIMER_CMP s váhou 6, adresa TIMER_CTRL s váhou 5, adresa TIMER_CYCLE_L s váhou 2, adresa TIMER_CYCLE_H s váhou 2, ostatní adresy s rozloženou váhou 1.

REQUEST: rozložení vah: CP_REQ_NONE s váhou 10, CP_REQ_READ s váhou 5, CP_REQ_WRITE s váhou 5, CP_REQ_RESERVED s váhou 1.

DATA_IN: rozložení vah: hodnota 0 s váhou 10, hodnoty 1 až 20 s váhou 20, vysoké hodnoty od 21 do $2^{**DATA_WIDTH}-1$ s rozloženou váhou 1.

Kontrolována (a hodnocena) bude zejména implementace omezujících podmínek (3b).

Pro tuto úlohu se vám bude pravděpodobně hodit možnost spouštět více testů naráz v režimu příkazové řádky simulátoru (tento režim je nastaven jako výchozí). Seznam těchto testů doplníte v souboru `test_list` v adresáři `test_lib`. Tady je kompletní přehled parametrů skriptu `start_verification.sh`, které se vám můžou hodit:

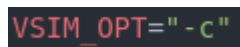
-h, -help – vypíše se nápověda, použití a popis parametrů

-c – spuštění simulátoru v režimu command line, toto nastavení je výchozí, protože je zde předpoklad, že budete chtít převážně pouštět více testů najednou.

-gui – spuštění simulátoru s grafickým uživatelským rozhraním

POZN.:

Tyto dva parametry se navzájem vylučují, vždy se vezme ten, který byl zadán jako poslední v pořadí. Defaultní nastavení můžete zvolit v souboru `start_verification.sh`, viz obr.



```
VSIM_OPT="-c"
```

-uvm_testname – výběr testu v případě, že budete spouštět pouze jeden

-uvm_tests_file – cesta a název souboru obsahující seznam testů, které chcete provést

-run_multiple_tests – spuštění více testů, pokud je nastaven tento přepínač, automaticky se spustí simulátor v režimu command line.

Příklad: `./start_simulation.sh -c uvm_tests_file path_to_tests_file -run_multiple_tests`

Úloha 7 — Implementujte scénáře pro funkční pokrytí [LAB3] (4b+0.5b+0.5b).

Implementujte scénáře pro funkční pokrytí podle instrukcí z 3. laboratoře. Doplněte sadu náhodných a přímých testů tak, abyste dosáhli vysoké pokrytí kódu a vysoké funkční pokrytí. Reflektujte váš verifikační plán, případně ho doplňte.

Výsledné dosažené pokrytí daným testem se vždy ukládá do souboru ***název_testu.ucdb*** do adresáře ***ucdb***. V případě spuštění více testů jsou pokrytí dosažená jednotlivými testy sjednocena do souboru s názvem ***final.ucdb*** v adresáři ***ucdb***. Pro zobrazení výsledného pokrytí, jak pokrytí kódu tak pokrytí funkčního, v podobě html můžete použít příkaz:

vcover report -html -annotate -details rtc_t_test.ucdb

Ten vygeneruje adresář ***covhtmlreport*** obsahující html stránky s dosaženým pokrytím.

Více testů můžete spustit i v grafickém módu, pokud budete mít ***run_multiple_tests*** nastavený defaultně a přidáte přepínač ***gui***, případně nastavíte grafický režim jako defaultní hodnotu. **SILNĚ se nedoporučuje.**

Kontrolována (a hodnocena) bude implementace funkčního pokrytí (4b). Dodatečnou úlohou je zjistit, kolik transakcí v náhodném testu, případně kolik přímých testů je potřeba pro dosažení 100% funkčního a 100% pokrytí kódu (branch + statement + condition + expression + FSM) (0.5b). Jako důkaz, že se Vám to povedlo, je potřebné doložit do výsledného reportu výpisy/screenshotsy z Questy (0.5b).

Úloha 8 — Implementujte formální tvrzení [LAB4] (3b+2b)

Implementujte formální tvrzení podle instrukcí ze 4. laboratoře. Analyzujte výsledky.

Kontrolována (a hodnocena) bude implementace formálních tvrzení (3b) + počet korektních aktivací formálních tvrzení (2b).

Úloha 9 — Vytvořte report

Vytvořte krátký report verifikace timeru, především shrňte dosažené výsledky, a uveďte, které body ve verifikačním plánu se vám podařilo úspěšně ověřit a jaké jste dosáhli pokrytí (funkční a kódu).