Teoretický úvod – laboratorní úloha číslo 5

Synchronní čítač s parametrizací kódu a s výstupem na displej

1 Synchronní a asynchronní čítače

Čítačem rozumíme obecně obvod, který (po)čítá výskyt požadovaného stavu vstupního signálu, obvykle tedy vzestupné či sestupné hrany periodického (hodinového) signálu, ale např. i počty stisků tlačítka, pulzy na datové sběrnici a další jiné podobné zdroje. Čítače mohou přičítat (vpřed) či odečítat (vzad), jejich výstupní hodnota může být v libovolném kódu, lze je doplnit např. o funkci nulování (resetu), vytvořit tzv. kruhový čítač (výstupní bit je přiveden zpět na vstup) atd.

Pro realizaci čítačů se využívají klopné obvody typu D a J-K, jeden klopný obvod vždy představuje 1 bit čítače. Z hlediska distribuce hodinového signálu rozlišujeme dva základní druhy čítačů, tzv. synchronní a asynchronní. V případě synchronního čítače jsou hodinové vstupy všech jeho klopných obvodů připojené přímo na jeden zdroj hodinového (vstupního) signálu. V tom případě tak u ideálního synchronního čítače může dojít v případě potřeby k překlopení (změně stavu) všech jeho klopných obvodů ve stejný okamžik (na vzestupnou či sestupnou hranu čítaného signálu).

U asynchronního čítače je k vnějšímu zdroji hodinového (čítaného) signálu připojen pouze první klopný obvod čítače, zatímco hodinové vstupy následujících klopných obvodů jsou vždy připojeny k výstupu předchozího klopného obvodu v řadě. Z toho vyplývá, že v případě nutnosti překlopení (změny stavu) všech klopných obvodů asynchronního čítače dochází k tomu postupně s tím, že každý klopný obvod (s výjimkou prvního) musí vyčkat na překlopení předchozího klopného obvodu, kdy se na jeho výstupu objeví požadovaná změna[1].

Čítače nemusíme vždy používat jen v základní roli pro čítání a zobrazení jejich výstupu, jako v případě stopek navrhovaných v této úloze, ale často jsou základem tzv. děliček frekvence. V nich figurují jako blok, který čítá vstupní hodinový signál, a po dosažení požadované hodnoty čítače (počtu načítaných hran) je vygenerován výstup, čímž v případě periodických signálů získáme výstupní periodický signál s frekvencí *n*-krát nižší oproti vstupnímu, kde *n* představuje počet načítaných hran (tímto způsobem využijeme čítač v laboratorní úloze č. 6 při dokončení digitálních stopek).

Z hlediska návrhu čítačů v jazyce VHDL můžeme využít jak behaviorální, tak i RTL či strukturální popis, každý z nich nabízí určité výhody či nevýhody. Možnosti využití RTL a strukturálního popisu v jazyce VHDL pro realizaci synchronního i asynchronního čítače jsme si představili na přednášce, v této laboratorní úloze využijeme pro realizaci čítačů **behaviorální popis**.

[1] Více a podrobněji jsme se čítači zabývali na přednášce předmětu věnované sekvenčním logickým obvodům a možnými způsoby návrhů čítačů v jazyce VHDL také na jedné z přednášek.

Ten je obvykle založen na využití knihovny `numeric_std` společně s konverzními funkcemi pro převod datových typů `unsigned` a `std_logic_vector`. Výstup čítače totiž obvykle požadujeme ve formátu n-bitového logického vektoru, zatímco základní myšlenka pro realizaci obecného čítače pomocí behaviorálního popisu v jazyce VHDL je založena na prostém popisu, že při výskytu požadované vlastnosti čítaného signálu (např. jeho hrany) inkrementujeme hodnotu čítače o jedničku. Avšak s datovým typem `std_logic_vector` nelze operaci +1 realizovat, je tedy nutné jej převést na datový typ `unsigned`, realizovat operaci +1 a opět tuto hodnotu převést zpět do typu `std_logic_vector` jako nový stav čítače.

2 Synchronní obvody v jazyce VHDL, detekce hrany signálu, synchronní vs. asynchronní reset obvodu

Synchronizace vstupů a výstupů obvodů je důležitá zejména pro zajištění synchronnosti vytvářených obvodů a struktur, zabránění vzniků potenciálně nebezpečných nesynchronních stavů, logických hazardů v obvodech apod. a samozřejmě za účelem vytvoření synchronních obvodů. Za tímto účelem definujeme periodický hodinový signál, ve VHDL obvykle značený `Clk` či `Clock`, na jehož vzestupné či sestupné hrany pak synchronizujeme všechny probíhající procesy a jednotlivé části kódu. Díky tomu máme pak zajištěnu časovou spolupráci a provázanost jednotlivých částí celého obvodu.

Vzhledem k tomu, že spouštění vykonávání částí kódu lze pouze v sekvenčně orientovaném návrhu, lze synchronizaci hodinovým signálem ve VHDL využít pouze v sekvenčním prostředí (`sequential domain`) architektury pomocí procesů. Protože se jedná v jazyce VHDL o velmi častý úkol, uvedeme zde trojici ekvivalentních řešení VHDL kódu pro detekci vzestupné i sestupné hrany hodinového vstupu `Clock`. Uvedený postup však lze samozřejmě obecně využít pro detekci hrany jakéhokoliv signálu (vstupu) v jazyce VHDL, nemusí se tedy jednat pouze o hodinový signál, ale např. i datový signál na komunikační sběrnici, výstup ovládacího tlačítka apod.

První dvojice řešení je založena na podmínce typu `if`:

```
process(Clock)
begin
if Clock='1' and Clock'event then
  ...
end if;
end process;
```

```
process(Clock)
begin
if rising_edge(Clock) then
  ...
end if;
end process;
```

První možnost využívá atributy v jazyce VHDL, což jsou dodatečné parametry portů, signálů či proměnných specifikované pomocí apostrofu „'“ za názvem portu/signálu/proměnné[2] a konkrétně atribut `event` slouží k detekci změny, tedy hrany (vzestupné i sestupné) daného portu/signálu/proměnné. Protože je dále v této složené podmínce definováno, že pro její platnost musí být zároveň vstup `Clock` ve stavu logické 1, `'1'`, je tato podmínka splněna jen při vzestupné hraně vstupu `Clock`. Pokud bychom tedy chtěli detekovat naopak sestupnou hranu, změnili bychom v podmínce pouze její platnost při logické 0, `'0'`.

Druhé řešení s podmínkou typu `if` využívá funkci jazyka VHDL přímo určenou pro detekci **vzestupné hrany** `rising_edge()` s názvem daného vstupu v kulatých závorkách. Pro detekci **sestupné hrany** existuje analogicky v jazyce VHDL funkce `falling_edge()`.

Poslední metodou pro detekci hrany hodinového vstupu je využití klíčového slova `wait`:

```
process
begin
wait until rising_edge(Clock);
  ...
end process;
```

Jak již bylo dříve uvedeno, při použití příkazu `wait` nesmí proces obsahovat citlivostní seznam. Dále je zde opět využita funkce `rising_edge()`, úprava pro detekci sestupné hrany vstupu `Clock` s použitím `wait` by tedy obsahovala funkci `falling_edge()`.

Jedním ze vstupů, kterým obvykle čítače obecně vybavujeme, je vstup pro jejich nulování (reset). Jak už z názvu vstupu vyplývá, jedná se o funkci, která zajistí uvedení čítače do jeho výchozího stavu, obvykle tedy do nulového stavu[3]. Z hlediska časové závislosti rozlišujeme tento reset buď jako synchronní, nebo asynchronní, obecně synchronní či asynchronní způsob resetování definujeme v případě jakéhokoliv sekvenčního logického obvodu, tedy nikoliv pouze čítače.

Synchronní reset čítače (obecně sekvenčního log. obvodu) je proveden pouze v součinnosti s hodinovým (taktovacím) signálem obvodu. Reset je tedy vykonán až v okamžiku nejbližší následující hrany hodinového signálu (vzestupné či sestupné) od okamžiku aktivace resetu, na kterou je čítání čítače prováděno. Nevýhodou tohoto způsobu resetu je tedy zpoždění vzniklé čekáním na nejbližší následující aktivační hranu hodinového signálu od okamžiku aktivace resetu, naopak výhodou tohoto řešení je, že výstup (stav) čítače je vždy synchronní, a to i po jeho resetu.

Naopak, v případě asynchronního způsobu resetování (označuje se jako tzv. nulování) je tato operace provedena okamžitě a bezprostředně po aktivaci resetu bez ohledu na stav hodinového signálu. Tím nedochází v ideálním případě k žádnému zpoždění při provedení nulování obvodu, nevýhodou však je, že výstup obvodu v tomto okamžiku přestává být synchronní, což může (ale nemusí) způsobit problémy, pokud jsou na výstup tohoto obvodu

[2] Atributy v jazyce VHDL jsme si již vysvětlili v rámci laboratorní úlohy č. 3.

[3] Nemusí to však nutně vždy platit, např. v případě čítače – odčítače naopak požadujeme, aby se takový odčítač vrátil do své výchozí hodnoty, od které se odečítá apod.

navázány jiné synchronní obvody, které se tak mohou dostat do pseudosynchronního či asynchronního stavu. Z hlediska realizace způsobu resetování obvodu v jazyce VHDL pomocí behaviorálního popisu se jedná o relativně jednoduchou úpravu – v procesu, ve kterém pomocí podmínky typu `if` detekujeme přítomnost požadované hrany hodinového signálu (viz dále), pouze vhodně upravíme a rozšíříme podmínku přidáním klíčového slova `elsif`. Zápisem pořadí detekce hodinového vstupu a resetovacího vstupu pak realizujeme synchronní či asynchronní variantu resetu, současně však také ještě musíme upravit citlivostní seznam procesu, jak můžeme vidět níže.

Asynchronní reset:

```
process(Clock,Reset)
begin
if Reset='1' then -- asynchronni reset
     stav<=(others=>'0');
elsif Clock='1' and Clock'event then
     stav<=stav+1;
end if;
end process;
```

V případě asynchronního způsobu resetování obvodu (nulování), je vstup `Reset` obsažen v citlivostním seznamu a v rámci podmínky typu `if` je nadřazen na první místo před podmínku detekce hrany hodinového signálu `Clock` tak, aby mohl být vykonán bez ohledu na jeho stav.

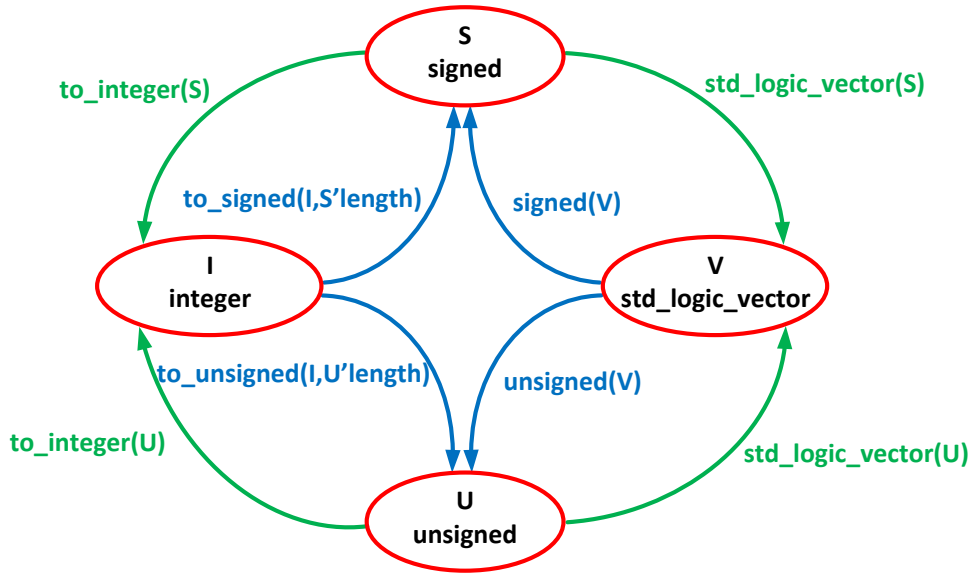
Synchronní reset:

```
process(Clock)
begin
if Clock='1' and Clock'event then
     if Reset='1' then -- synchronni reset
          stav<=(others=>'0');
     else
          stav<=stav+1;
     end if;

end if;
end process;
```

Zatímco v druhém případě, synchronního provedení resetu, je naopak podmínka pro `Reset` vnořena do podmínky detekce vzestupné hrany hodinového signálu `Clock` a z tohoto důvodu ani vstup `Reset` není obsažen v citlivostním seznamu procesu. Povšimněte si, že samotný reset je proveden uvedením výstupu (či signálu) `stav` na hodnotu samých nul, v jazyce VHDL to lze elegantně vyřešit pomocí klíčového slova `others` a daný zápis v podstatě znamená, že na všechny pozice vektoru `stav` zapíšeme hodnotu logická nula `'0'` a nemusíme tedy specifikovat počet pozic, délku vektoru.

3 Konverze datových typů, datový typ integer, porty out a buffer

Jak jsme si vysvětlili v úvodu zadání a rozboru této laboratorní úlohy, pro behaviorální popis a realizaci čítače využijeme s výhodou opět knihovnu `numeric_std` a konverzní funkce, zejména převody datových typů `std_logic_vector` a `unsigned`, v této úloze navíc i převod `integer` a `unsigned`. Obr. č. 1 ukazuje konverzní schéma pro převody datových typů (obdobně tomu bylo u laboratorní úlohy číslo 3).



Obr. č. 1: Převody mezi datovými typy integer, signed, unsigned a std_logic_vector.

V této úloze nově využijeme také datový typ `integer` (viz dále), a proto se zmíníme i o jeho konverzi na datové typy `signed/unsigned`. Datový typ `integer` v jazyce VHDL představuje základní typ pro zápis celých čísel. Při jeho konverzi na datový typ `signed/unsigned` tak musíme specifikovat, na kolik bitových pozic (řádových pozic) toto celé číslo převádíme pomocí druhého parametru ve funkcích `to_signed` a `to_unsigned`, jak je naznačeno pomocí atributu `'length` (délka vektoru) v obrázku výše.

Datový typ `integer` patří mezi základní celočíselné typy v jazyce VHDL. Je definován jako celé číslo v intervalu $-2^{32}-1$ až $+2^{32}-1$ (-2147483647 až +2147483647), v jazyce VHDL existují dále tzv. subtypy, což jsou odvozené verze základních typů pomocí zúžení původního rozsahu, pro datový typ `integer` proto existuje např. subtyp `natural`, což je `integer` v intervalu 0 až $+2^{32}-1$, či `positive`, což je `integer` v intervalu +1 až $+2^{32}-1$. Libovolné vlastní zúžení a vytvoření vlastního subtypu si můžeme v jazyce VHDL definovat sami, například vytvoříme sub-typ s názvem `small_int` v intervalu 0 až 9 pomocí:

```
subtype small_int is integer range 0 to 9;
```

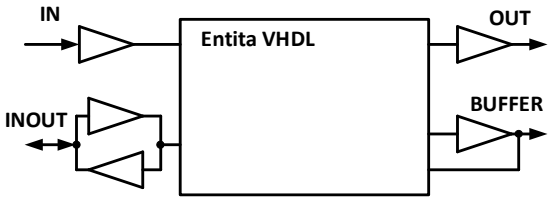
I v případě, že použijeme přímo datový typ `integer` pro deklaraci portu, signálu či proměnné a víme, že daný port, signál či proměnná může nabývat v daném obvodu jen omezeného rozsahu hodnot, je vždy vhodné uvést toto omezení rozsahu. Pokud bychom to neprovedli, bude pro daný port, signál či proměnou při syntéze automaticky alokována sběrnice v plném rozsahu typu `integer`, tedy 32 bitů, což často povede ke zbytečnému plýtvání HW

prostředky (viz přednášky). Omezení provedeme specifikací rozsahu podobně jako při definici předchozího subtypu, např. definujme signál `a`, který může nabývat rozsahu jen 0 až 99:

```
signal a : integer range 0 to 99;
```

Celočíselné typy v jazyce VHDL můžeme například použít při realizaci čítačů, kdy potřebujeme načítat potřebný počet hran vstupního signálu, nebo např. pro omezení rozsahu čítače, jak si ukážeme dále.

V krátkosti se ještě zastavme u dvojice typů (směrů) portů v jazyce VHDL, `out` a `buffer`, a rozdílu mezi nimi. V jazyce VHDL rozlišujeme 4 typy (směry) portů, `in`, `out`, `buffer` a `inout`[4]. Jak z jejich názvů a ze schematického znázornění na vyplývá, hlavním rozdílem mezi porty je možnost směru využití daného portu, zaměřme se na porty `out` a `buffer`.



Obr. č. 2: Schématické znázornění čtveřice základních portů v jazyce VHDL.

Oba porty, `out` i `buffer`, patří mezi výstupní porty, které primárně slouží pro vyvedení výstupu (výstupní hodnoty) ven z entity. Avšak zatímco pro port typu `out` je definována pouze operace zápisu hodnot pro výstup ven z entity bez možnosti jejího dalšího využití, v případě výstupu typu `buffer` lze s touto hodnotou pomocí její zpětné vazby zpět do entity i dále pracovat.

Rozdíl si můžeme demonstrovat na příkladu právě v rámci této laboratorní úlohy a návrhu čítače. Pokud bychom realizovali výstup čítače čistě jako port typu `out`, nemohli bychom s tímto portem provést operaci +1 (inkrementaci čítače), jak bychom potřebovali. V rámci entity totiž hodnotu na výstupu typu `out` neznáme, nemůžeme ji číst (je podporován pouze zápis) a ani uvnitř entity s ní tedy pracovat. Uvedený problém bychom mohli vyřešit tím, že deklarujeme např. pomocnou proměnnou (či signál), která bude obsahovat aktuální hodnotu (stav) čítače, s touto proměnnou (signálem) provedeme operaci +1 a výsledek zapíšeme na výstupní port typu `out`. Alternativně lze využít právě port typu `buffer`, který na rozdíl od typu `out` obsahuje zpětnou vazbu zpět do dané entity a tedy hodnotu portu typu `buffer` známe a můžeme s ní libovolně pracovat včetně operace +1 při využití Behaviorálního popisu čítače.

[4] Respektive, v jazyce VHDL je definováno celkem 5 typů portů, ještě kromě výše uvedených tzv. `linkage` port. Jedná se o „propojovací" port, který umožňuje obousměrnou komunikaci a propojení, avšak pouze entit a bloků v tzv. linkage módu. V jazyce VHDL není v podstatě využíván a často se tedy ani neuvádí.

4 Parametrizace VHDL kódu pomocí generic

Při návrhu logických obvodů a bloků můžeme s výhodou využít tzv. parametrizace v rámci VHDL kódu a navrhnout daný obvod s využitím parametrů. Tyto parametry jsou sdruženy v tzv. `generic` sekci deklarované na začátku entity a mohou být následně využity v rámci celého VHDL kódu v dané entitě, kde umožňují provádět změny. Můžeme tak parametrizovat např. délku vektoru, počet bitů čítače, počet rotací či posuvu registru, měnit hodnoty parametrů v různých podmínkách apod. Tyto parametry jsou tedy často typu `integer`, ale lze se setkat i s jinými datovými typy. Při deklaraci parametru určíme jeho datový typ a výchozí hodnotu, která se použije, pokud nemá parametr nastavenou hodnotu jinou.

Při použití dané entity jako komponenty v rámci jiné VHDL entity (top-level entity), kromě mapování jejích portů provedeme stejným způsobem i mapování jejích parametrů, tedy určení hodnot parametrů v rámci této top-level entity. Uveďme si pro ilustraci krátkou ukázku VHLD kódu s využitím parametrizace.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity counter is
generic (w : integer :=8);
port (Count : buffer unsigned(w-1 downto 0);
      Clock : in std_logic);
end counter;

architecture RTL of counter is
begin
process(Clock)
begin
if Clock='1' and Clock'event then
Count<=Count +1;
end if;
end process;
end RTL;
```

V rámci VHDL kódu výše jsme vytvořili jednoduchý čítač s výstupem typu `unsigned` a pomocí parametru `w` jsme parametrizovali jeho délku, výchozí délka čítače je 8 bitů (`7 downto 0`). Takto navržený čítač využijeme jako komponentu pro vytvoření jiného čítače.

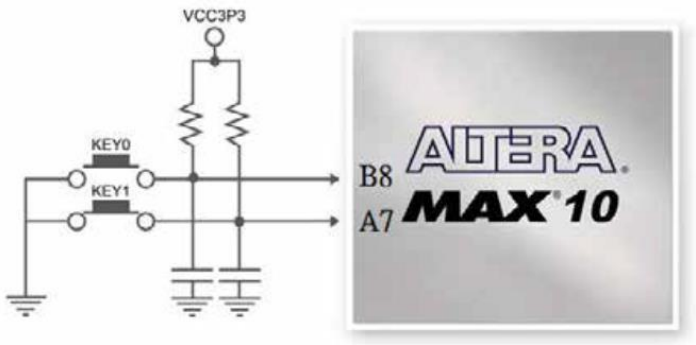
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity citac is
port (Vystup : out unsigned(15 downto 0);
      Clock : in std_logic);
end citac;

architecture Behavioral of citac is
component counter is
generic (w : integer);
port (Count : buffer unsigned(w-1 downto 0);
      Clock : in std_logic);
end component;
begin
Citac : work.counter
generic map(16)
port map(Vystup,Clock);
end Behavioral;
```

Čítač o délce 16 bitů (`15 downto 0`) jsme vytvořili pomocí komponenty předchozího čítače a při mapování parametrů v rámci `generic map` jsme nastavili hodnotu parametru `w` na 16. Pro mapování generických parametrů můžeme, stejně jako pro mapování portů, použít poziční a jmenné mapování, stejný výsledek bychom tedy získali použitím příkazu `generic map(w => 16)`. Povšimněte si dále, že mezi oběma mapováními, tedy mezi řádky `generic map` a `port map`, se nenachází středník.

V této laboratorní úloze využijeme parametrizaci pro vytvoření základního čítače, který použijeme pro čítání sekund na stopkách. Pomocí parametru omezíme rozsah čítání tohoto čítače, neboť pro čítání jednotek sekund budeme potřebovat rozsah 0–9, zatímco pro čítání desítek sekund nám bude stačit čítat v rozsahu 0–5.

5 Využití tlačítek na přípravku DE10-Lite

Na přípravku DE10-Lite se nacházejí dvě tlačítka označená KEY0 a KEY1. Tlačítka jsou zapojena v invertovaném režimu, tzn. pokud nejsou stisknuta, je na jejich výstupu hodnota logická 1, naopak při stisknutí tlačítka dojde k uzemnění a na výstupu tlačítka je tedy hodnota logická 0. Obr. č. 3 demonstruje tuto situaci.

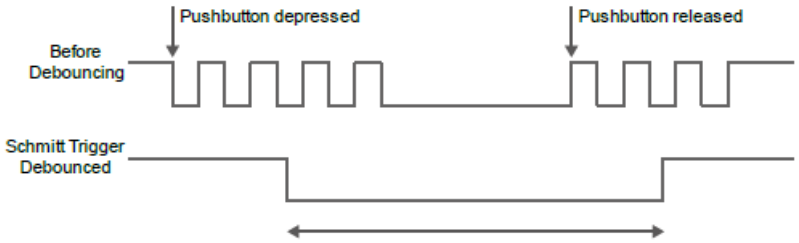


Obr. č. 3: Schéma zapojení tlačítek na přípravku DE10-Lite.

Pokud bychom tedy v jazyce VHDL chtěli detekovat okamžik stisku tlačítka, museli bychom detekovat sestupnou hranu signálu na výstupu tlačítka a nikoliv vzestupnou (tou bychom detekovali okamžik uvolnění tlačítka zpět).

Při použití tlačítek, tlačítkových spínačů a podobných mechanických prvků, je nutné počítat se vznikem tzv. bouncingu (zákmitů). Mechanické kontakty při stisku tlačítka nejsou dokonalé a nedojde tedy k jejich dokonalému a jednorázovému propojení. Pokud bychom připojili osciloskop k výstupu běžného tlačítka, zjistili bychom, že v okamžiku stisknutí i uvolnění tlačítka dochází k několikanásobnému spojení a rozpojení kontaktů, tzv. zákmitům, a pokud budeme např. v jazyce VHDL výstup tlačítka využívat v obvodech pracujících s takty v řádu alespoň stovek kHz, nezískali bychom tak pouze jeden přechod (hranu) signálu, ale několikanásobný zákmit mezi hodnotami logická 1 a logická 0 než se výstup definitivně na jedné z hodnot ustálí.

Pro odstranění bouncingu se využívají tzv. metody pro debouncing. Těchto metod existuje několik, bližší popis již překračuje rozsah této laboratorní úlohy. Pro snazší práci s tlačítky a odstranění tohoto problému je v přípravku DE10-Lite integrován debouncing tlačítek pomocí tzv. Schmittova klopného obvodu s hysterezí (Schmitt trigger). Obr. č. 4 ukazuje průběh bez a s použitím tohoto obvodu (převzato z manuálu přípravku).



Obr. č. 4: Využití Schmittova klopného obvodu pro odstranění zákmitů tlačítka.

Pro využití této možnosti odstranění zákmitů tlačítek na přípravku DE10-Lite stačí během přiřazení pinů v okně *Pin Planner* u daného tlačítka zvolit ve sloupečku *I/O Standard* požadovaný napěťový standard s aplikací Schmittova klopného obvodu (Schmitt trigger).

6 Základ realizace stopek s tlačítkovým vstupem a výstupem na displej

Na závěr nastíníme jednu z možností pro vytvoření stopek požadovaných v této laboratorní úloze.

Jako základ nejprve navrhneme a pomocí behaviorálního popisu v jazyce VHDL realizujeme synchronní čítač; jeho entitu nazvěme `citac`. Tento čítač bude mít hodinový vstup `clock`, nulovací vstup `reset` a také blokovací vstup `enable`. Účelem vstupu `enable` je dočasné pozastavení či spuštění čítače (blokování čítače), tedy v případě, kdy je na vstupu `enable` logická 1, `'1'`, čítač čítá, pokud je však na vstupu `enable` hodnota logická 0, `'0'`, čítač nečítá a je zastaven[5].

Výstupem čítače bude 4bitový logický vektor `stav`, abychom obsáhli všechny číslice 0-9 zobrazitelné na 7segmentovém displeji, a dále bude výstupem indikace přenosu do vyššího čítaného řádového místa pomocí `std_logic` výstupu, `prenos`. Ten využijeme pro indikaci přenosu do vyššího řádového místa stopek, tedy v okamžiku, kdy čítač jednotek sekund dočítá do hodnoty 9, při dalším stisknutí tlačítka (respektive příchodu hrany hodinového signálu) dojde k překlopení zpět na 0 a bude indikován přenos do vyššího řádového místa, tzn. výsledkem pak bude hodnota 10.

Port pro samotný výstup čítače, `stav`, deklarujeme jako typ `buffer`, abychom s ním mohli pracovat jako s čítačem, jak jsme si vysvětlili v kapitole 3. Kromě těchto portů však při deklaraci entity deklarujeme i `generic` sekci, která bude obsahovat parametr `modulo` pro parametrizaci maximálního rozsahu čítače; datový typ parametru `modulo` bude `integer` s výchozí hodnotou 9.

Jak jsme si vysvětlili v kapitole 4, pomocí parametrů můžeme měnit a upravovat vlastnosti a hodnoty navržené entity v případě jejího využití jako komponenty v rámci jiné top-level entity. V závěru kapitoly 4 jsme ukázali, že uvedený základní čítač pro realizaci stopek můžeme pomocí parametrizace vhodně upravit tak, aby při jeho použití jako komponenty na pozici jednotek sekund čítal v rozsahu 0–9, zatímco jako komponenta na pozici desítek sekund čítal pouze v rozsahu 0–5. Tento rozsah pak můžeme jednoduše nastavit právě pomocí parametru `modulo` v rámci parametrizace `generic`.

Vlastní architektura čítače `citac` se pak bude skládat v podstatě pouze z jednoho procesu, v citlivostním seznamu budou uvedeny vstupy `clock` a `reset`. V této laboratorní úloze budeme pro zjednodušení budit hodinový vstup `clock` stiskem tlačítka na přípravku (např. KEY 1), hodinový signál z oscilátoru budeme využívat až v následující laboratorní úloze. Druhé tlačítko přípravku (např. KEY 0) tedy využijeme pro vstup `reset`.

[5] Tento blokovací vstup `enable` využijeme až v navazující laboratorní úloze č. 6, kdy pomocí něho budeme stopky zastavovat a spouštět, ale v rámci komponenty základního čítače si jej vytvoříme již nyní.

Jak jsme si vysvětlili v předchozí kapitole, tlačítka jsou na přípravku DE10-Lite realizována v invertovaném režimu, budeme tedy pro detekci jejich stisku využívat sestupnou hranu výstupního signálu a jejich stisknutý stav pomocí hodnoty logická 0.

Čítač máme dle zadání navrhnout s asynchronním resetem (nulováním), využijeme tedy jeho VHDL kód z kapitoly 2.

Dále bude proces pokračovat `if` podmínkou detekce sestupné hrany hodinového vstupu `clock`. Do této podmínky však ještě vnoříme podmínku typu `if` pro blokovací vstup `enable` – pokud bude hodnota vstupu `enable` rovna logické 1, bude čítač čítat (tedy přičítat +1, případně dojde k jeho vynulování při dosažení maximální hodnoty čítače), při hodnotě vstupu `enable` logická 0 čítač nebude čítat (nebude vykonáno nic).

Uvnitř této blokovací podmínky bude již jádro vlastního čítače – poslední vnořená podmínka `if`, která bude kontrolovat, zda nedošlo k překročení nastavené maximální hodnoty čítače (`modulo` a `9`), pokud nikoliv, bude do čítače přičtena +1.

Maximální hodnotu čítače jsme nastavili pomocí parametru `modulo`, který je typu `integer`, výstup čítače, `stav`, je pro změnu typ `std_logic_vector` a pro jeho porovnání s maximem daným číslem `9` (také `integer`) a parametrem `modulo` (`integer`) je potřeba provést příslušné konverze.

Správně bychom měli převést typ `std_logic_vector` nejprve na typ `unsigned` a následně na `integer` (nebo naopak převést čísla `9` a `modulo` na `unsigned`). Avšak jednou z mála výjimek v jazyce VHDL při použití operací a datových typů je operace porovnání (=), kdy lze navzájem porovnat i bez konverze hodnotu datového typu `unsigned` a datového typu `integer`. Pokud dosáhla hodnota výstupu čítače, `stav`, maximální hodnoty, vynulujeme jej a nastavíme výstup pro indikaci přenosu, `prenos`, do hodnoty logická 0.

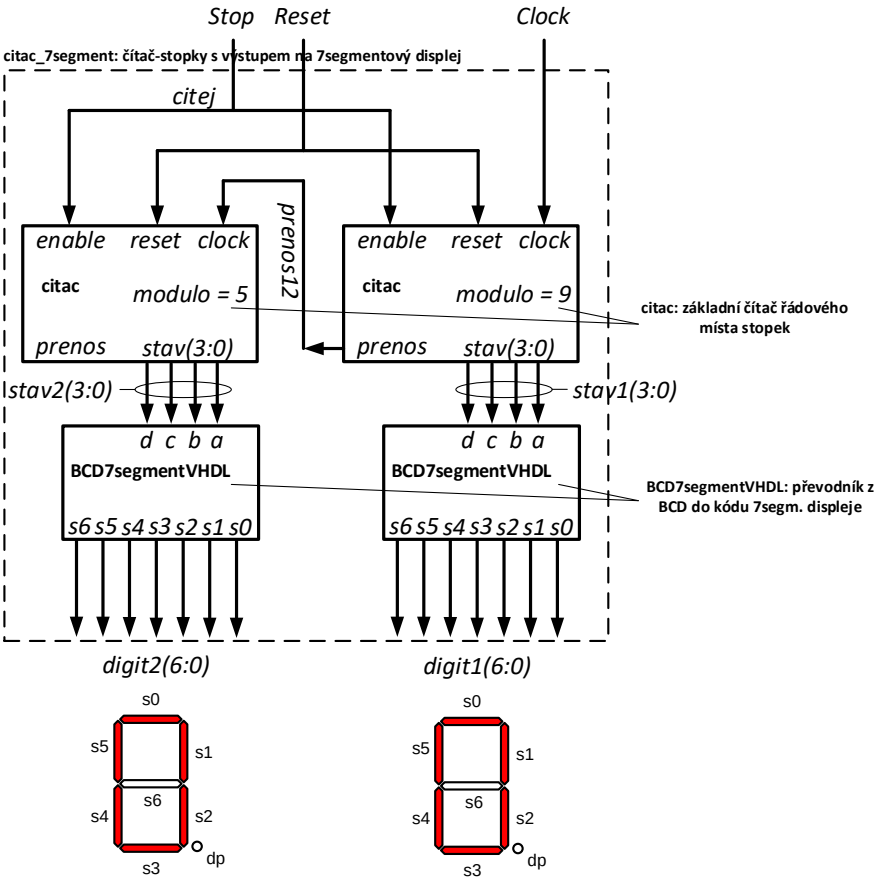
Nakonec ještě pro dokončení podmínky zapíšeme příkaz, kdy je hodnota čítače inkrementována o +1, je tedy nutné převést výstup čítače `stav` z typu `std_logic_vector` na `unsigned`, provést operaci `+1` a výsledek opět převést na `std_logic_vector` a uložit do výstupu čítače `stav`. Současně také v tomto případě nastavíme indikaci přenosu, `prenos`, zpět do hodnoty logické 1[6].

[6] Stejně jako pro tlačítka a vstupy `clock` a `reset`, i pro indikaci přenosu pomocí výstupu `prenos` využijeme invertovanou logiku, tzn. v případě přenosu do vyššího řádu čítání stopek nastavíme logickou 0. Je to z toho důvodu, že v další komponentě čítače na pozici desítek sekund použijeme tento `prenos` jako jeho hodinový vstup, který je invertovaný.

Kód č. 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity citac is
     generic (modulo : integer :=9);
     port (clock,reset,enable : in std_logic;
           prenos : out std_logic;
           stav : buffer std_logic_vector(3 downto 0));
end citac;

architecture Behavioral of citac is
begin
process (clock,reset)
begin
if reset = '0' then
     ... -- zde doplnte nulovani citace
elsif clock='0' and clock'event then
     if enable='1' then
          if unsigned(stav) = modulo or unsigned(stav) = 9 then
               ... -- zde doplnte nulovani citace
               ... -- zde doplnte nastaveni prenosu na '0'
          else
               ... -- zde doplnte stav+1
               ... -- zde doplnte nastaveni prenosu na '1'
          end if;
     end if;
end if;
end process;
end Behavioral;
```

Nyní již můžeme vytvořit hlavní top-level entitu čítače-stopek s výstupem na 7segmentový displej. Tato top-level entita (nazvěme ji např. `citac_7segment`) bude obsahovat dvojici předchozích čítačů `citac` jako komponent, každý pro jedno čítané řádové místo stopek, a také dvojici komponent převodníků z kódu BCD do kódu 7segmentového displeje pro zobrazení každého řádového místa stopek na jednom 7segmentovém displeji přípravku, `BCD7segmentVHDL`. Obr. č. 5 ukazuje výsledné schéma zapojení a propojení komponent.



Obr. č. 5: Schéma realizace čítače-stopek pomocí jednotlivých komponent.

Výsledná top-level entita `citac_7segment` má vstupní porty typu `std_logic` `Stop`, `Reset` a `Clock`, výstupem jsou 2 vektory `digit1`, `digit2` typu `std_logic_vector` o velikosti 7 bitů pro realizaci výstupu na dvojici 7segmentových displejů.

Dále v architektuře deklarujeme použití komponent `citac` a `BCD7segmentVHDL` a také čtveřice signálů, kterými propojíme jednotlivé komponenty a porty entity.

Signály `stav1`, `stav2` jsou 4bitové logické vektory, každý propojuje výstup čítače na daném řádovém místě a převodník z BCD kódu do kódu 7segmentového displeje pro zobrazení výstupu daného čítače na příslušný displej.

Signál `prenos12` slouží k propojení výstupu `prenos` z čítače čítajícího jednotky sekund do hodinového vstupu `clock` čítače čítajícího desítky sekund.

A konečně signál `citej` je určen pro realizaci spuštění a zastavení čítání čítačů prostřednictvím jejich vstupů `enable` a je ovládán skrze vstup (tlačítko) `Stop`. Základ výsledného VHDL kódu takto realizované top-level entity `citac_7segment` můžeme zapsat takto:

Kód č. 2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity citac_7segment is
     ...-- zde doplnte deklaraci portu
end citac_7segment;

architecture Structural of citac_7segment is
...-- zde doplnte deklaraci signalu stav1, stav2, prenos12
signal citej : std_logic :='1';

component citac is
     generic ...; -- zde doplnte deklaraci komponenty citac
     port ...;
end component;

component BCD7segmentVHDL is
     port ...; -- zde doplnte deklaraci komponenty BCD7segment
end component;

begin
citac_1: citac -- pouziti komponenty citac pro jednotky sekund
generic map(9) -- mapovani generic parametru modulo = 9
port map (...); -- mapovani portu

citac_2: citac -- pouziti komponenty citac pro desitky sekund
...; -- obdobne vytvorte mapovani citace desitek sekund

displej_1: BCD7segmentVHDL
port map (stav1(0),stav1(1),...);

displej_2: BCD7segmentVHDL
port map (...);

process(stop) -- proces detekce stisknuti tlacitka Stop
begin
if stop='0' and stop'event then
     citej<=not  citej;  --  po  stisku  tlacitka  Stop  dojde
k invertovani hodnoty signalu citej, ktery ovlada enable vstupy
end if;
end process;

end Structural;
```

Obr. č. 5 ukazuje, že výstup `prenos` z čítače čítajícího desítky sekund není připojen. Při mapování portů v jazyce VHDL lze tuto situaci řešit v případě použití jmenného mapování portů jednoduše tak, že daný port (v tomto případě `prenos`) vůbec v seznamu mapovaných portů neuvedeme[7], nebo korektnější způsob je použití klíčového výrazu `open` ať již při použití jmenného (`prenos => open`) nebo pozičního mapování (`...,open,...`), který v jazyce VHDL slouží právě k ošetření nezapojeného portu komponenty.

[7] Syntezátor nás však v tomto případě upozorní vygenerováním varování (warrning), že port komponenty není připojen, syntéza však proběhne a syntezátor automaticky nezapojený port odstřihne.