Laboratorní cvičení z předmětu CITE

Čítače, generické parametry, děličky hodin, detekce tlačítka

Martin Rozkovec, martin.rozkovec@tul.cz
Technická univerzita v Liberci | Fakulta mechatroniky, informatiky a mezioborových studií | ITE
Studentská 1402/2, 461 17 Liberec 1 | www.fm.tul.cz

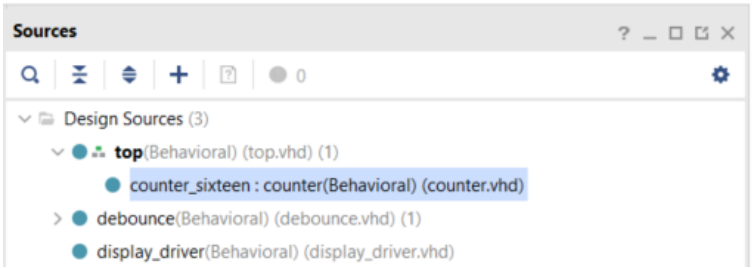
1. Z kurzu předmětu na elearning.tul.cz stáhněte a rozbalte projekt LAB08.zip.

   Cesta k projektu nesmí obsahovat diakritiku, mezery nebo speciální znaky. Vhodné umístění je v laboratoři A107 disk D:, ve kterém si můžete vytvořit podadresář.

2. Dvakrát klikněte na soubor LAB08.xpr uvnitř dekomprimované složky. Otevře se prostředí Vivado.

Čítač a generické parametry

1. Otevřete soubor **counter.vhd**.



2. Povšimněte si:
   a. Bloku **generic** a generického parametru **COUNTER_WIDTH**. Parametry z bloku můžeme dále v entitě i architektuře používat jako konstanty. Záznamy v bloku **generic** se zapisují stejně jako záznamy v bloku **port**, ale uvádíme u nich výchozí hodnotu pomocí operátoru okamžitého přiřazení.

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

entity counter is
    generic(
        COUNTER_WIDTH : integer := 4
    );
    port(
        cnt           : out std_logic_vector(COUNTER_WIDTH - 1 downto 0);
```

   b. Generickou konstantu jsme použili i pro deklaraci registru čítače. Ten je realizován signálem datového typu **unsigned** z knihovny **IEEE.NUMERIC_STD.ALL**. Datový typ

unsigned umožňuje používání aritmetických operací. V dnešním cvičení si vystačíme se sčítáním.

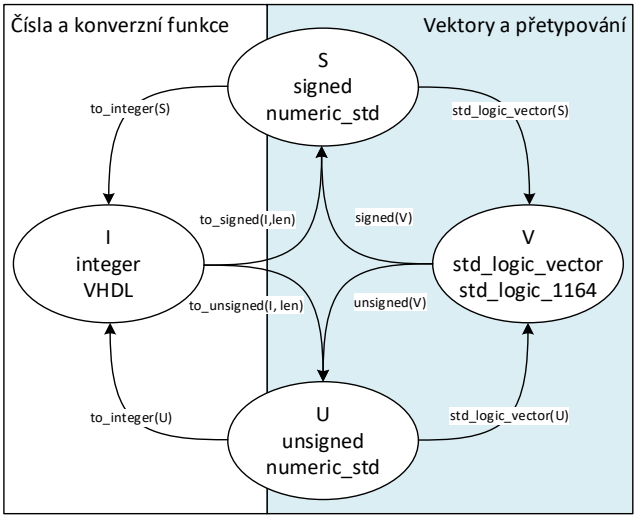
```
    signal counter_reg : unsigned(COUNTER_WIDTH - 1 downto 0);
begin
    process(clock)
    begin
        if rising_edge(clock) then
            counter_reg <= counter_reg + 1;
        end if;
    end process;
    cnt <= std_logic_vector(counter_reg);
```

3. Otevřete soubor **top.vhd** a povšimněte si:
    a. Instance čítače a mapování generického parametru na konstantní hodnotu 16. Instance čítače bude 16bitová namísto původní 4bitové.
    b. Způsobu, kterým jsme přiřadili na zatím neaktivní porty konstantní hodnoty.
4. Nahrajte do vývojové desky předpřipravený bitsream **top.bit**. Čítání je na lidské poměry poměrně rychlé, že? Zkusíme s tím něco udělat, ale nejprve čítač trochu vylepšíme.

5. Upravte postupně kód čítače v souboru **counter.vhd**. Chování čítače můžete ověřovat v připravené simulaci (**counter_tb.vhd**). Pro simulaci si vystačíme s malým, čtyřbitovým čítačem**.**

- Všechny podmínky, vkládejte hierarchicky dovnitř (pod) podmínku s rising_edge(clk).
- Datové typy std_logic_vector a unsigned nelze libovolně přiřazovat. Podrobně se tomu věnují skripta, pro naše účely vystačíme s přiložený „tahákem“. Nechte si ho.

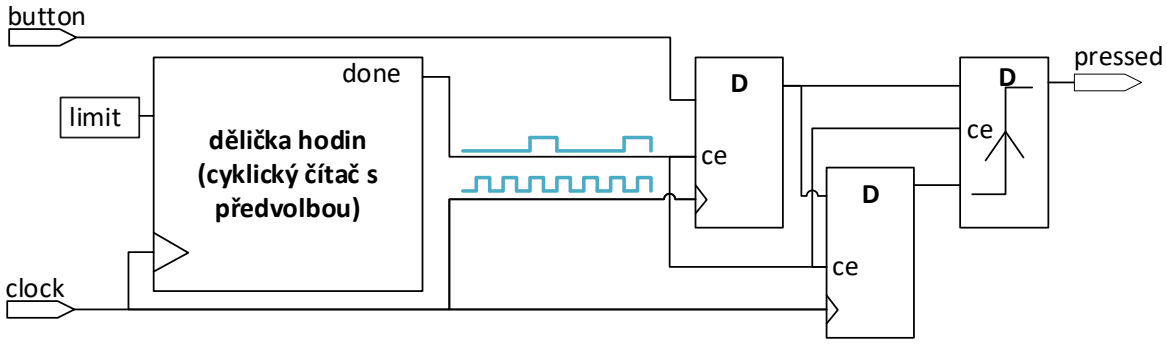
a. Obvod opatřete synchronním resetem řízeným vstupem **reset**. Je-li **reset** aktivní, nastaví čítač do 0.
b. Obvod opatřete povolením čítání s použitím **clock_enable**. Je-li **clock_enable** v log. 1, pak čítač čítá. Reset má větší prioritu než **clock_enable**.
c. Doplňte možnost zkráceného cyklu:
    i. Dosáhne-li čítač hodnoty **limit** (counter_reg = unsigned(limit), rel. operátory jsou =, /=, <=, >=)
        1. Je-li **repeat** v log. 0, pak přestane čítat
        2. Je-li **repeat** v log. 1, pokračuje hodnotou 0
d. Výstup **done** se nastaví na log.1, dosáhl-li čítač hodnoty limitu, jinak zůstává v log.0. Řešte jej jako kombinační logiku, tj. mimo proces čítače.



6. Až si budete jistí, že v simulaci vše funguje, budeme pokračovat.

Ošetření tlačítka a dělička hodin

1. Cílem bude ovládat vstup **clock_enable** čítače tlačítkem. Ošetření tlačítka provedeme tak, že budeme porovnávat současnou a předchozí hodnotu tlačítka a hledat změnu, která odpovídá náběžné hraně. Abychom vyloučili krátké zákmity budeme toto provádět na menší frekvenci, než je vstupní hodinová frekvence. Při návrhu budeme postupovat tak, že nebudeme vytvářet nové synchronizační signály – vystačíme si s původními hodinami.
2. Obvod je částečně připraven v kódu **debounce.vhd**. Do obvodu vstupuje signál od tlačítka **button** a hodinový signál **clock**. Výstupem obvodu je signál **pressed**. Blokové schéma obvodu je následující:



3. Soubor musíte upravit:
   a. Vypočítejte šířku čítače (konstanta **DIV100HZ_WIDTH**) a limit (konstanta **DIV100HZ_LIMIT**) tak, aby váš čítač generoval signál done každých 10ms (100 Hz). Frekvence hodinového signálu **clock** je 100 MHz. Získáme obvod, který periodicky generuje signál povolení hodin – děličku.
   b. Upravte podmínku detekce náběžné hrany na konci architektury. Nahraďte symboly U správnými logickými hodnotami:

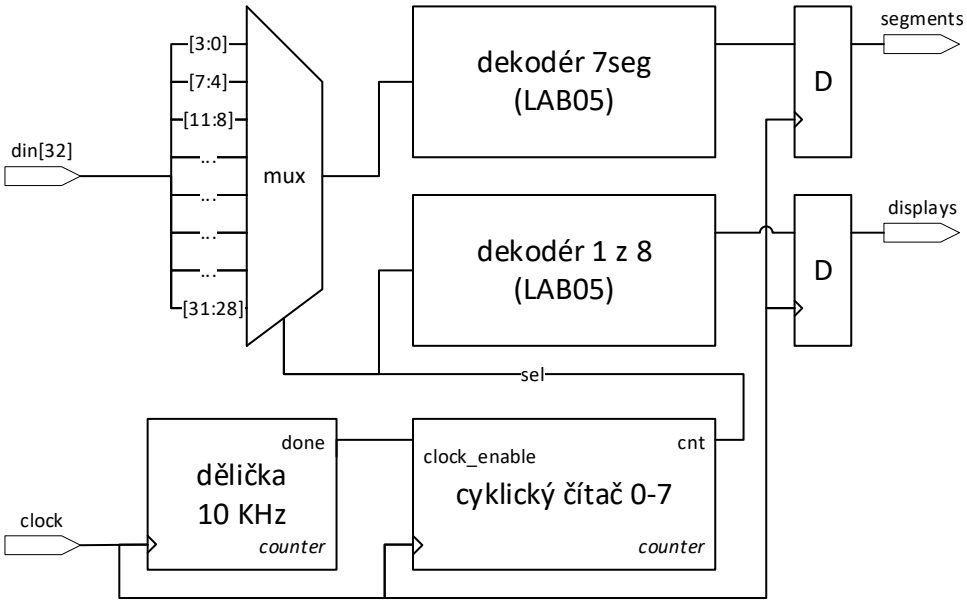
```
pressed <= '1' when actual_button_value = 'U' and last_button_value = 'U' else '0';
```

4. V **top.vhd** odkomentujte instanci **debounce** tak, abyste pomocí tlačítek řídili čítání čítače.
   a. Výstup **pressed** z **debounce** zapojte na port **clock_enable** vašeho čítače.
   b. Vstup **button** připojte na některé z tlačítek **btn_***

5. Do portu limit čítače namapujte **SW**.
6. Zbylá tlačítka namapujte na porty **reset** a **repeat**. Výstup **done** namapujte na **LED_BLUE**.
7. Vygenerujte bitstream a nahrajte jej do vývojové desky. Tlačítka jsou na pravé straně desky uprostřed.

Domácí úloha

8. Za domácí úkol sestavte obvod, který bude zobrazovat 32bitové číslo v hexadecimální soustavě na displejích vývojové desky. Při konstrukci využijte blokového schéma, vašich čítačů a dekodérů z této a předešlých hodin.



a. Připomeňme – na vývojové desce je 8 displejů, každý má 8 segmentů (7 pro číslice). V jednom okamžiku umíme zobrazit čtyřbitové číslo pouze na jednom z displejů.
b. Abychom zobrazili více čísel, musíme displeje a čísla rychle přepínat. Rozsvítíme první 4 bity na prvním displeji, chvíli počkáme, přepneme displej, rozsvítíme další 4 bity, atd. až do posledního displeje a posledních 4 bitů.
c. Frekvenci přepínání zvolíme 10 kHz – je dostatečně krátká na to, aby nám nepřišlo, že displeje blikají a dostatečně dlouhá na to, aby se LED pořádně rozsvítily. Děličku frekvence vytvoříme pomocí cyklického čítače – obdobným způsobem jako v dnešním cvičení u děličky hodin obvodu debounce. Vypočítáme limit a potřebnou šířku čítače.

d. Dále vytvoříme instanci cyklického čítače čítajícího v intervalu 0-7. Čítač bude inkrementovat každých 100 μs v rytmu děličky hodin. Výstup z čítače bude sloužit:
    i. pro výběr displeje pomocí dekodéru 1 z 8 zapojeného na **displays**. Ten jsme vyrobili v LAB05.
    ii. pro výběr 4 bitů z 32 bitů vstupního slova **din**. Hodnota nula volí **din** (3 downto 0), hodnota 1 **din**(7 downto 4), atd. až po 7, který vybírá **din**(31 downto 28). Výběr bitů realizujeme pomocí multiplexoru. Kód multiplexoru si můžete zjednodušit – využijte výstup z čítače konverzi std_logic_vector → unsigned → integer pro indexaci rozsahu.
e. Výstup multiplexoru zapojíme do dekodéru 7seg displeje z LAB 05, který bude budit **segments**.
f. Aby nám čísla při přepínání neblikala můžeme výstupy **segments** a **displays** opatřit registry.
g. Vstup reset na obrázku značen není, ale my jej realizujeme jako synchronní.
h. Obvod vytvořte v souboru LAB_08\LAB_08.srcs\sources_1\new\display_driver.vhd. Pro simulaci můžete použít nástroj Aldec HDL