Laboratorní cvičení z předmětu CITE

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

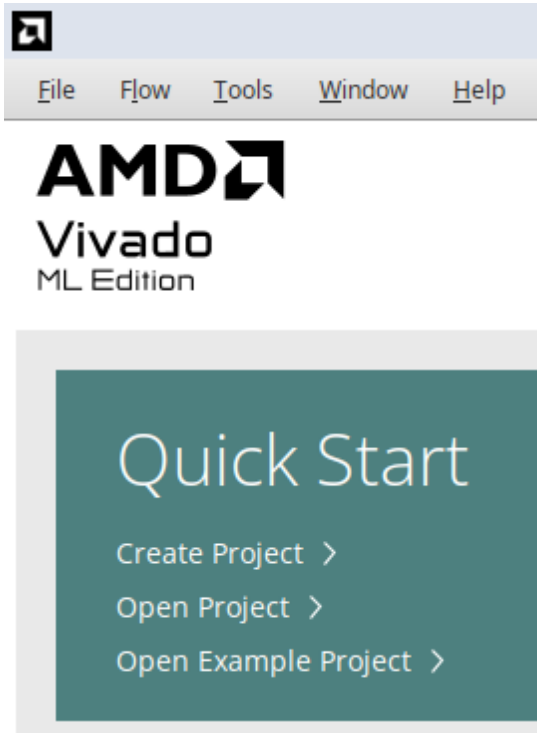
Proces, simulace, instance entity

1. Z kurzu předmětu na elearning.tul.cz stáhněte a rozbalte projekt LAB06.zip.

   Cesta k projektu nesmí obsahovat diakritiku, mezery nebo speciální znaky. Vhodné umístění je v laboratoři A107 vaše domovská složka, ve kterém si můžete vytvořit podadresář s vaším jménem bez diakritiky.

1. Otevřete Vivado – dvakrát klikněte na spouštěč na ploše:

2. V úvodním okně aplikace klikněte na Quick Start – Open Project:



3. Otevřete soubor LAB06.xpr uvnitř dekomprimované složky.

Process (pro kombinační logiku)

1. Seznamte se s příkazem **process**:

```
process[(citlivostní seznam)]
   [deklarační část]
begin
   [sekvenční část]
end process [návěští];
```

Paralelní příkaz **process** představuje způsob, jak spouštět jednotlivé příkazy sekvenčně (za sebou):

- Proces se (v simulaci) spustí při změně **signálu** uvedeného v **citlivostním seznamu**. Za signál můžeme v tomto kontextu považovat i **vstup**. Je-li seznam prázdný, pak se proces spouští neustále.
- VHDL 2008 umožňuje při popisu **kombinační logiky** nahradit seznam klíčovým slovem **all**.
- V **deklarační části** procesu můžeme vytvářet proměnné viditelné pouze v procesu.
- Za klíčové slovo begin vkládáme sekvenční příkazy.
- Mezi sekvenční příkazy patří
    - Příkaz **nepodmíněného přiřazení** (**<=**) do signálu.
    - Příkaz **nepodmíněného přiřazení do proměnné** (**:=**).
    - Podmínka **if** (obdoba podmíněného přiřazení) a **case** (obdoba výběrového přiřazení).
    - Cykly **for** a **while**.
    - Příkaz **wait**. Obsahuje-li proces příkaz **wait**, **nesmí** mít **citlivostní seznam**.
- Vlastnosti procesu:
    - Doba vykonání všech příkazů procesu je nekonečně krátká.
    - Všechna přiřazení do proměnných (:=) se provedou okamžitě.
    - Všechna ostatní přiřazení (<=) se provedou až po ukončení procesu nebo při zavolání příkazu **wait**.
    - Proces je paralelní z pohledu ostatních příkazů v těle architektury
    - Obsahuje-li proces **wait**, není syntetizovatelný do HW.

Syntax příkazu nepodmíněného přiřazení znáte z předchozích cvičení. Podmínka **if** a příkaz **case** jsou uvnitř procesu bližší tomu, co znáte z programovacích jazyků:

```
if podmínka1 then
        [sekvenční_příkazy1]
[elsif podmínka2 then
        [sekvenční_příkazy2]]
…
[elsif podmínkaN then
        [sekvenční_příkazyN]]
[else
        [sekvenční_příkazyE]]
end if;
```

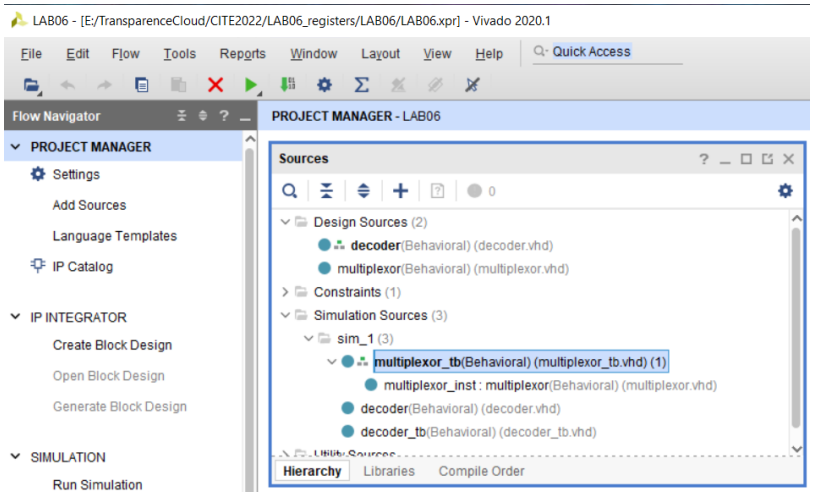
```
case výraz_výběru is
[when volba1 =>                 -- jednoduchá volba
        sekvenční_příkazy1;]
[when volba21 | volba22 =>      -- vícenásobná volba
        sekvenční_příkazy2;]
[when volbaN1 | volbaNN =>
        sekvenční_příkazyN;]
when others =>
        sekvenční_příkazyOthers;
end case;
```

2. Otevřete soubor **multiplexor.vhd**.
3. Do předem připraveného procesu doplňte popis **multiplexoru 1:4** s datovým vstupem **d**[4], výběrem signálu **s**[2] a výstupem **q**.
    a. Pro popis využijte příkaz **process** a podmínku **if**
    b. Doplňte správně citlivostní **seznam**
        i. Seznam musí obsahovat všechny signály, jejichž změna vede ke změně výstupů (přiřazení) v procesu
4. Zobrazte si schéma vytvořeného obvodu (RTL Analysis \ Schematic)

5. Otevřete soubor **decoder.vhd**
6. Doplňte popis dekodéru z 2bit binárního váhového kódu do kódu 1 ze 4. Hodnoty výstupu **q** ovládá vstup **d**.
    a. Využijte příkaz **process** a příkaz **case**
    b. Nezapomeňte na **citlivostní seznam**
7. Přepněte Vivado na zpracování souboru **decoder.vhd** (pravé tlačítko myši na název souboru a set as top) a zobrazte si schéma dekodéru.

Simulace, instance entity

1. Otevřete soubor **multiplexor_tb.vhd** z **Simulation Sources** \ **sim_1**. Tento soubor bude sloužit pro simulaci námi navrženého obvodu.



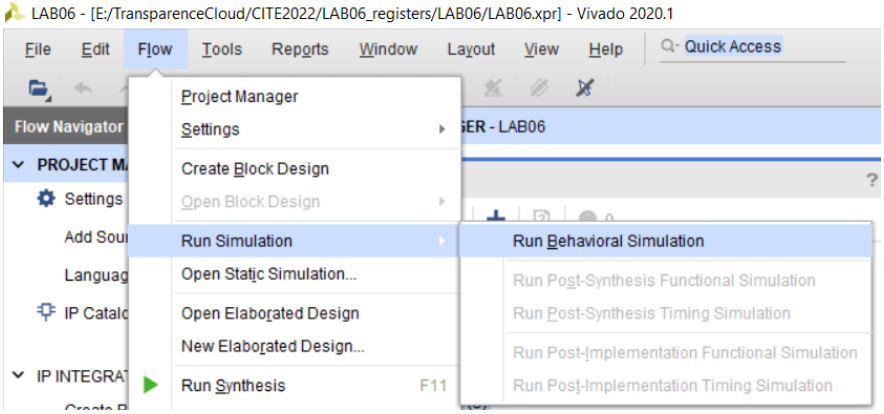
2. Povšimněte si:
   a. Obvod **multiplexor_tb** neobsahuje žádné vstupy, ani žádné výstupy.
   b. V deklarační části architektury se nachází deklarace signálů **mux_d**, **mux_s** a **mux_q (mux_q_cond)**.
   c. V těle architektury jsou dva příkazy. První je **příkaz instance entity** a můžete si jej představit jako vložení hotového obvodu multiplexoru do našeho nového obvodu:
      i. Řádek „*mux1 : entity work.multiplexor*“ vytvoří v simulačním obvodu zástupce (instanci) předem navrženého obvodu multiplexor, který se nachází v knihovně **work** (= výchozí knihovna VHDL projektu)
      ii. Při instanci **map**ujeme (připojujeme) **porty instance entity** vlevo (=>) na **signály** uvnitř architektury vpravo.
   d. Porovnejte **deklaraci** entity s **instancí** entity a všimněte si podobností:

```
entity multiplexor is
    port (
        d : in  std_logic_vector(3 downto 0);
        s : in  std_logic_vector(1 downto 0);
        q : out std_logic
    );
end multiplexor;
```

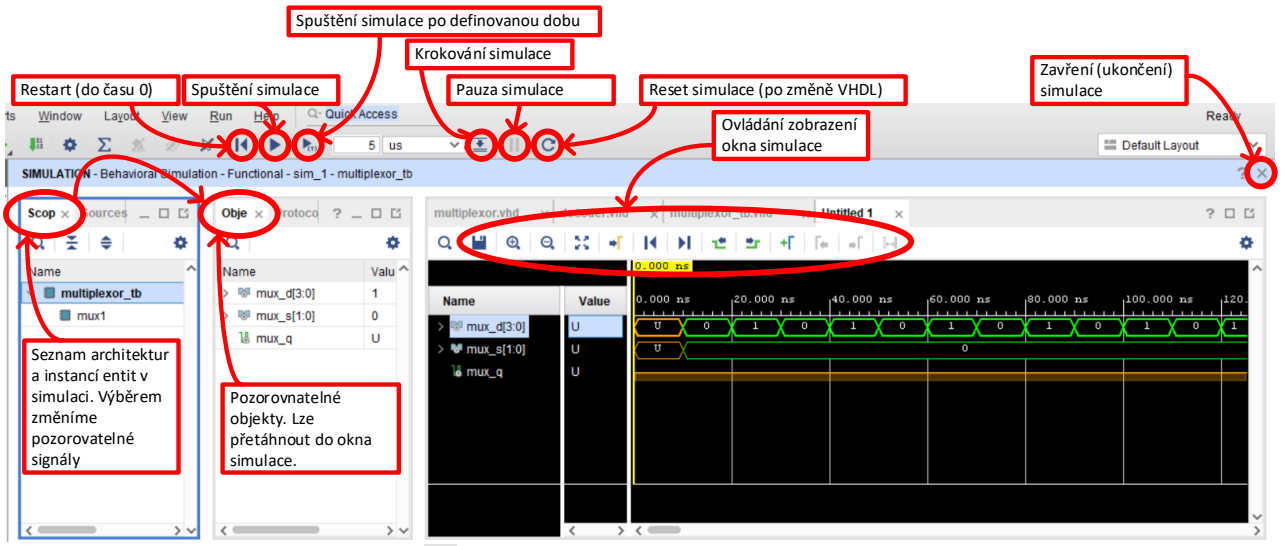
```
mux1 : entity work.multiplexor
    port map(
        d => mux_d,
        s => mux_s,
        q => mux_q
    );
```

e. Druhý příkaz je příkaz **process**, který realizuje posloupnost kroků, které povedou k „otestování“ – simulaci – našeho obvodu:

3. Spusťte simulaci:
   a. Klikněte pravým tlačítkem na položku Flow → Run Simulation → Run Behavioral Simulation:



   b. Prozkoumejte okno simulace:



   c. Restartuje simulaci ⏮ a krokujte ji ⏭:

d. Najedete-li myší nad signál v okně editoru, můžete vidět jeho aktuální hodnotu. Všimněte si, že na počátku simulace jsou všechny signály **U**ndefined. Ke změně nedojde okamžitě po jejich přiřazení, ale po zavolání druhého příkazu **wait**.

e. Pokračujte v krokování (F8). Všimněte si, že po dokončení procesu se opět spustí a čas mezi ukončením a opětovným startem je nulový.

4. V kódu **multiplexor_tb** doplňte:

a. příkazy **přiřazení** a **wait** tak, abyste vyzkoušeli funkci multiplexoru: Pomocí signálu **mux_s** ovládejte vstup **s** a postupně vybírejte vstupy $\mathbf{d_0}$, $\mathbf{d_1}$, $\mathbf{d_2}$, $\mathbf{d_3}$. Na každém vstupu vyzkoušejte přechod 0-1. Nejméně složitý postup ověření funkce lze shrnout následující tabulkou:

čas	0	1	2	3	4	5	6	7	8
d(0)	0	1	1	0	0	0	0	0	
d(1)	0	0	0	1	1	0	0	0	
d(2)	0	0	0	0	0	1	1	0	
d(3)	0	0	0	0	0	0	0	1	
s(0)	0	0	1	1	0	0	1	1	
s(1)	0	0	0	0	1	1	1	1	...
q	**d(0)**	**d(0)**	**d(1)**	**d(1)**	**d(2)**	**d(2)**	**d(3)**	**d(3)**	
q()	0	1	0	1	0	1	0	1	

b. Vytváření testu můžete algoritmizovat pomocí smyčky **loop** s iteračním schématem **for** a funkce **uint2slv**(val, size)

i. Smyčka for **i** in **rozah** loop <příkazy> end loop; iteruje integer proměnou **i** v uvedeném rozsahu (**to**, **downto**). Iterační proměnnou **i** lze využít pouze v příkazech uvnitř smyčky a nelze do ní přiřazovat.

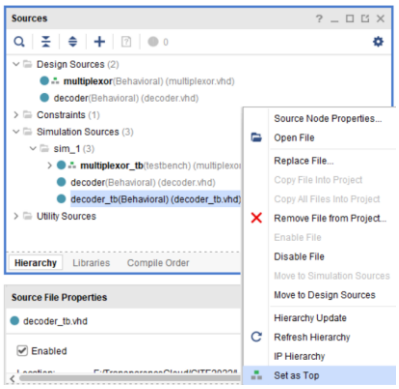
ii. uint2slv(val,size) vrací nezáporné číslo **val** ve formátu std_logic_vector o velikosti **size**.

c. Na konec procesu doplňte příkaz **wait**; Tím zabráníme opětovnému spuštění procesu.

d. Po uložení souboru je nutné soubor znovu zkompilovat a spustit simulaci:

e. Spusťte simulaci znovu a předveďte ji cvičícímu.

5. Zavřete simulaci. Přepněte **simulaci** na **decoder_tb.vhd**:



6. Vytvořte simulaci obvodu dekodéru.
    a. Do kódu vložte instanci obvodu dekodéru popsaného pomocí příkazu case v procesu.
    b. Vytvořte potřebné signály.
        i. Deklarace signálu může obsahovat inicializační hodnotu. Zkuste to:

```
signal dec_d : std_logic_vector(1 downto 0) := (others => '0');
```

    c. Přidejte proces, kterým budete budit vstupy.
    d. Prozkoumejte všechny kombinace vstupů a předveďte cvičícímu.

Instance entity pro syntézu

1. Zavřete simulaci. Přepněte Vivado na **syntézu** souboru **top.vhd**.
2. Do kódu pomocí příkazu instance entity vložte **dvojici** instancí multiplexoru a dva dekodéry.
3. Pro ovládání využijte vstupy **SW**[16], výstupy instancí namapujte na výstup **LEDs**[16].
    a. Využijte stejné přepínače pro datové vstupy multiplexorů; lišit se budou pouze ve vstupech výběru.
4. Vytvořte **bitstream**, naprogramujte FPGA a předveďte cvičícímu.