Simulace hardware

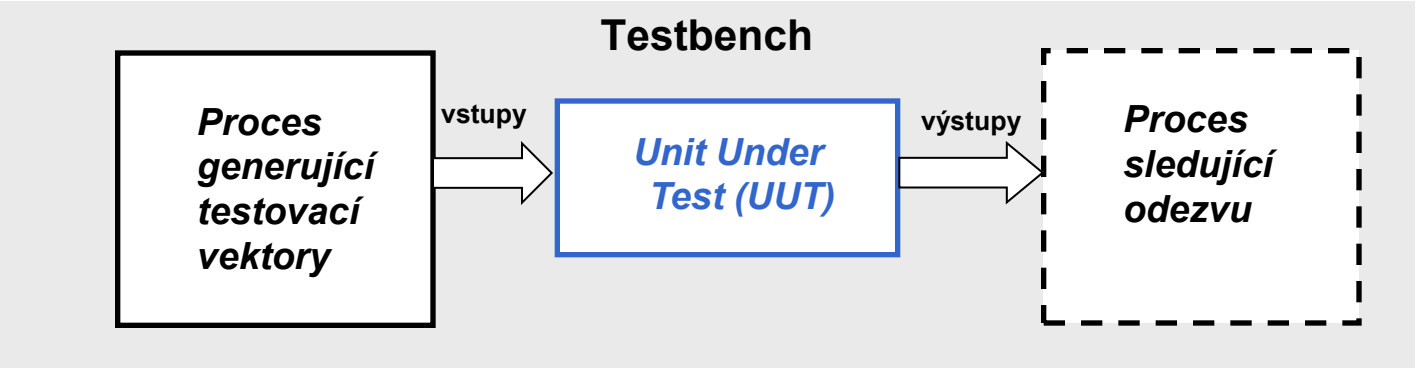
2025

Vojtěch Mrázek
mrazek@fit.vutbr.cz,

Verifikace a simulace pomocí nástroje ModelSIM (QuestaSIM)

Ověření funkčnosti - testbench

- Testbench je obvod, který aplikuje testovací vektory na vstupy testovaného obvodu a verifikuje získanou odezvu (není nezbytně nutné).
- Výsledkem může být časový diagram (waveform) nebo textový/binární soubor.
- Ve VHDL lze napsat jednoduchý i komplexní testbech (snadná přenositelnost).

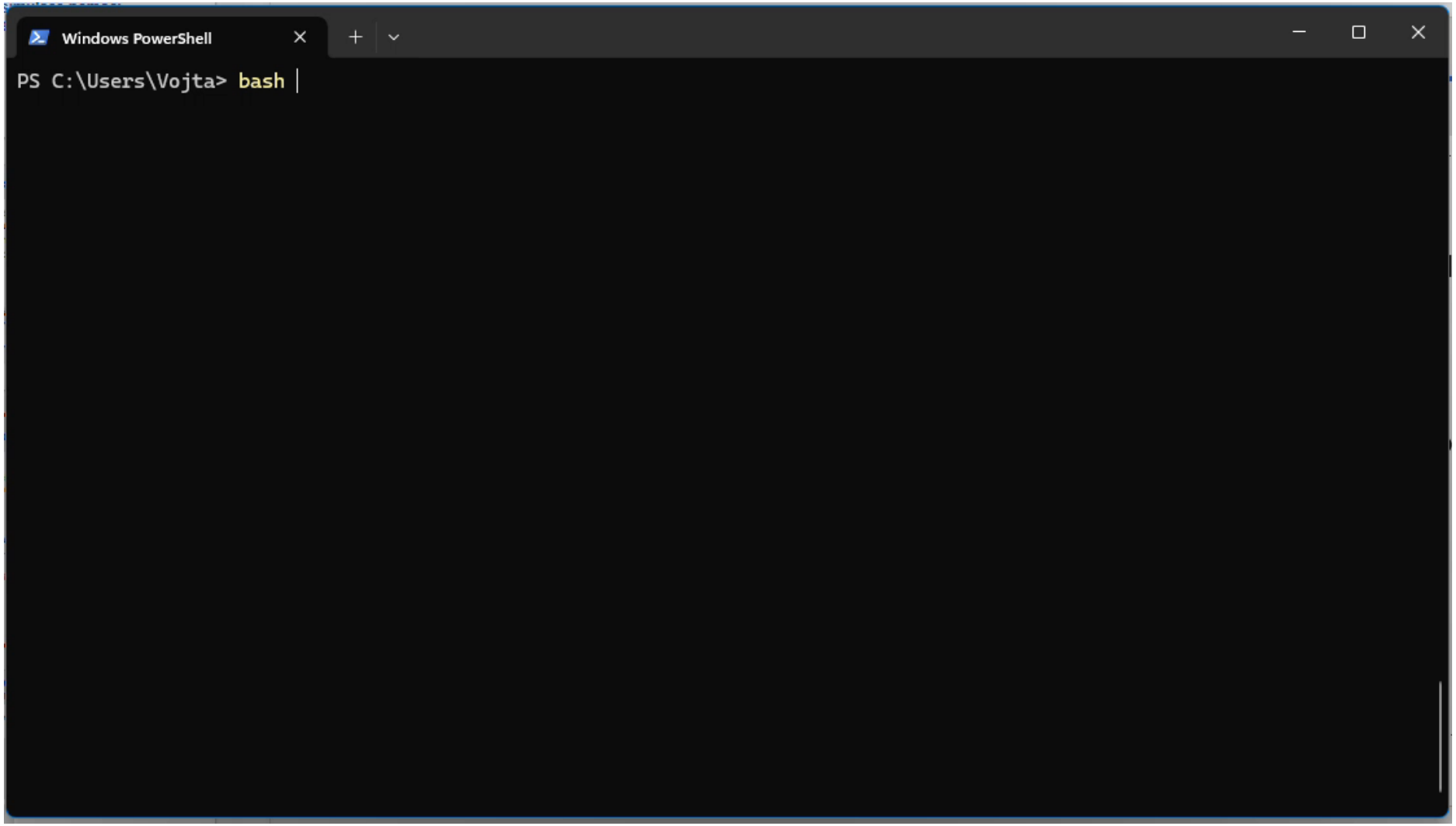


Nástroje pro simulaci HDL

- Pro simulaci lze použít
  - ModelSIM, QuestaSIM (vyžaduje licenci)
  - ~~Xilinx ISE Simulator (ISIM) (součást volně dostupného balíku Xilinx ISE Webpack)~~
  - Xilinx Vivado (objemný SW, umožňuje i post-synthesis simulaci)
  - ~~Synopsys VCS (Verilog)~~
  - GHDL

- Alternativně lze pracovat přímo na stroji fitkit-build
  (z Windows pomocí putty+Xming, WSL2 apod.)

Využití WSL2 (Windows)



Návrh hardware v praxi

Využití VHDL

Zdrojový kód (VHDL, Verilog, …)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity blk4 is
    port (
        i1,i2,i3,i4: in STD_LOGIC;
        o1: out STD_LOGIC
    );
end blk4;

architecture struc of blk4 is
begin
  o1 <= (i1 or i2) and (i3 and i4);
end struc;
```

RTL-level

simulace

Behaviorální simulace

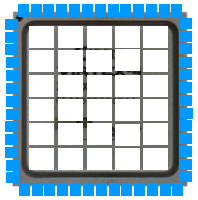
syntéza

simulační model

netlist

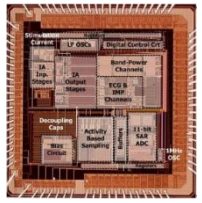
simulace

Funkční (časová) simulace



FPGA

ASIC



place&route, map

simulační model

netlist

simulace

Funkční (časová) simulace

Simulace pomocí nástroje GHDL (open-source)

Nástroje

- GHDL
  - Kompilátor VHDL do spustitelného souboru
  - Výstupem simulace je test podmínek (assertů) a waveform
- Zobrazení waveformu
  - GTKWave (zastaralé GUI)
  - Surfer
    - Instalace pro Linux / Windows (nejrychlejší) – 👍
    - Webová aplikace (https://app.surfer-project.org/) – soubory 😔
    - Doplněk do VSCode (surfer-project.surfer) – dostačuje 🤔
- Nevýhody
  - neumožňuje breakpointy pro ladění 😮
  - v základu pouze behavioární simulace (viz dále) ☠️

Postup simulace pro GHDL

- Skládá se ze analýzy (-a), elaborace (-e) a spuštění
- Nutné –fsynopsys pro podporu standardních knihoven
- Vytvoří se GHW (nebo VCD) soubor pro signály definované v top.wave

```
all: top.wave build

build: ram.vhd top.vhd sim.vhd
    ghdl -a -fsynopsys $^
    ghdl -e -fsynopsys top_sim
    ghdl -r -fsynopsys top_sim --wave=top.ghw --stop-time=10us --read-wave-opt=top.wave


top.wave:
    ghdl -r -fsynopsys top_sim  --stop-time=1us --wave=top.ghw --write-wave-opt=top.wave
```

Řešený příklad

Příklad – jednoduchý akcelerátor

- Navrhněte datovou cestu pro jednoduchý akcelerátor kódu pro sečtení všech čísel z paměti až po data s hodnotou 0.
- Implementujte stavový automat pro řízení.
- Data čtěte z dodané RAM.

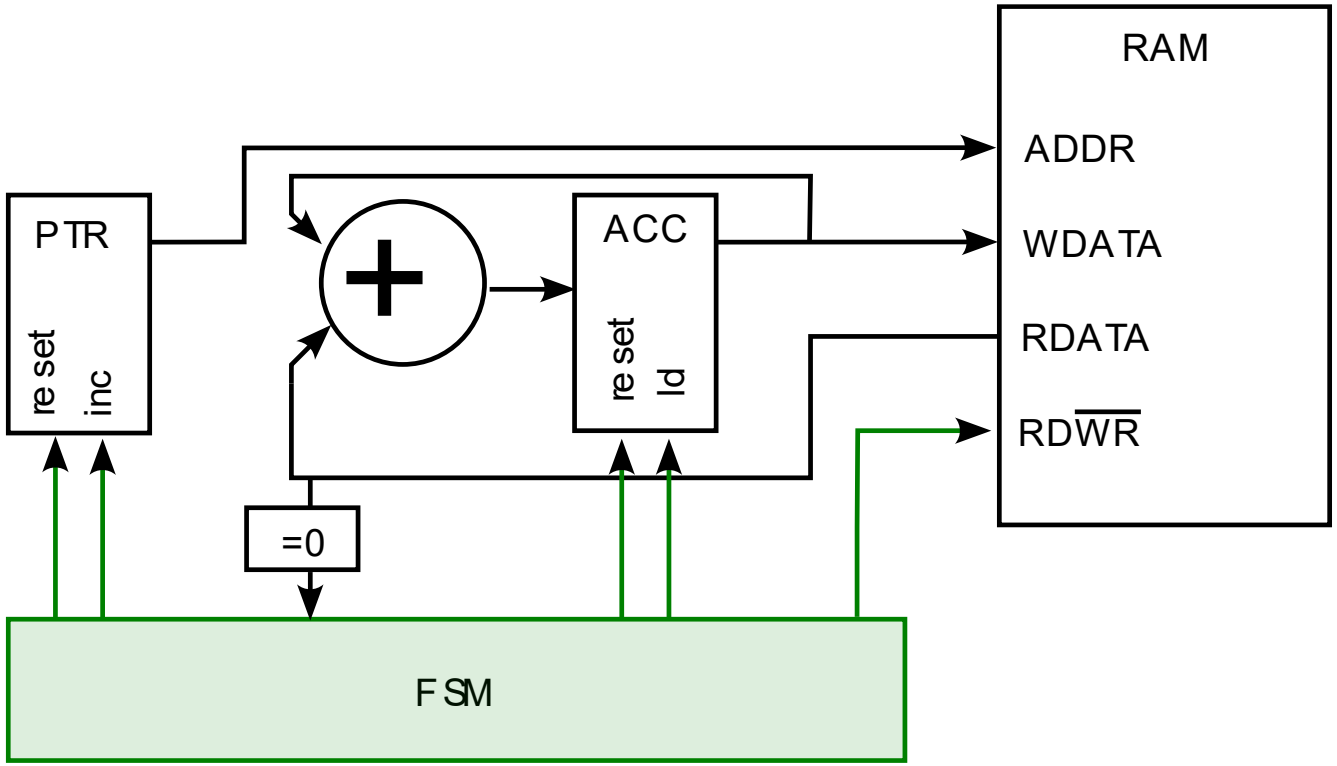
```
int acc = 0;
int * ptr = 0;

// Načítání
do {
    acc += *ptr;

// Ukončení a inkrementace
} while(*(ptr++) != 0 );

// Uložení
*ptr = acc;
```

Řešení - architektura




```
int acc = 0;
int * ptr = 0;

// Načítání
do {
    acc += *ptr;

// Ukončení a inkrementace
} while(*(ptr++) != 0 );

// Uložení
*ptr = acc;
```

Simulace pomocí komerčních nástrojů

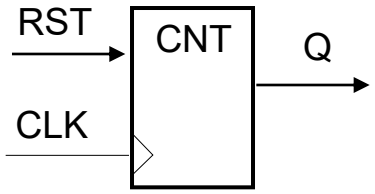
Questa, Vivado (post-synthesis)

Příklad: 8-bitový binární čítač
(cntr.vhd)

Synchronní čítač - obvod, který při každé vzestupné (nebo sestupné) hodinové hraně inkrementuje hodnotu na výstupu Q

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
ENTITY cntr IS
    PORT (
        CLK : IN STD_LOGIC;
        RST : IN STD_LOGIC;
        Q : BUFFER STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END cntr;
```

```
ARCHITECTURE behavioral OF cntr IS
BEGIN
    citac : PROCESS (RST, CLK)
    BEGIN
        IF RST = '1' THEN
            Q <= (OTHERS => '0');
        ELSIF CLK'event AND (CLK = '1') THEN
            -- elsif (CLK = '1') then -- chybny zapis
            Q <= Q + 1;
        END IF;
    END PROCESS;
END ARCHITECTURE;
```



Testbench pro čítač
(tb.vhd)

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY citac_tb IS
END citac_tb;
ARCHITECTURE behavior OF citac_tb IS
    --vstupy
    SIGNAL clk : STD_LOGIC := '0';
    SIGNAL rst : STD_LOGIC := '1';
    --vystupy
    SIGNAL q : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    -- instance uut (unit under test)
    uut : ENTITY work.cntr PORT MAP (
        CLK => clk, RST => rst, Q => q
        );
    clk <= NOT clk AFTER 5 ns;
    PROCESS
    BEGIN
        WAIT FOR 100 ns;
        WAIT UNTIL clk = '0';
        rst <= '0';
        WAIT UNTIL clk = '1';
        WAIT FOR 1 ns;
        ASSERT q = "00000001" REPORT "Chyba 1";
        WAIT UNTIL clk = '1';
        WAIT FOR 1 ns;
        ASSERT q = "00000010" REPORT "Chyba 2";
        WAIT;
    END PROCESS;
END;
```

Typické části

- deklarace signálů potřebných pro simulaci
- instance simulované komponenty (pomocí entity nebo komponenty)
- generátor hodinového signálu
- proces řídicí signál rst a sledující odezvu

Skript pro automatizaci simulace – pro ModelSim (QuestaSim)

(sim.fdo)

```
vlib work

vcom -explicit -93 src/cntr.vhd
vcom -explicit -93 +acc=rn src/tb.vhd

vsim -t 1ns -lib work citac_tb

view wave

add wave -divider "Vstupy"

add wave -label "reset" -color yellow /citac_tb/rst
add wave -label "hodinovy signal" -color magenta /citac_tb/clk

add wave -divider "Vystup"
add wave -radix unsigned -label "citac" /citac_tb/q

#vsechny signaly z UUT
add wave -divider "UUT"
add wave /citac_tb/uut/*

restart -f

force -freeze citac_tb/rst 1 0 600ns -cancel 1000ns


run 2 us
```

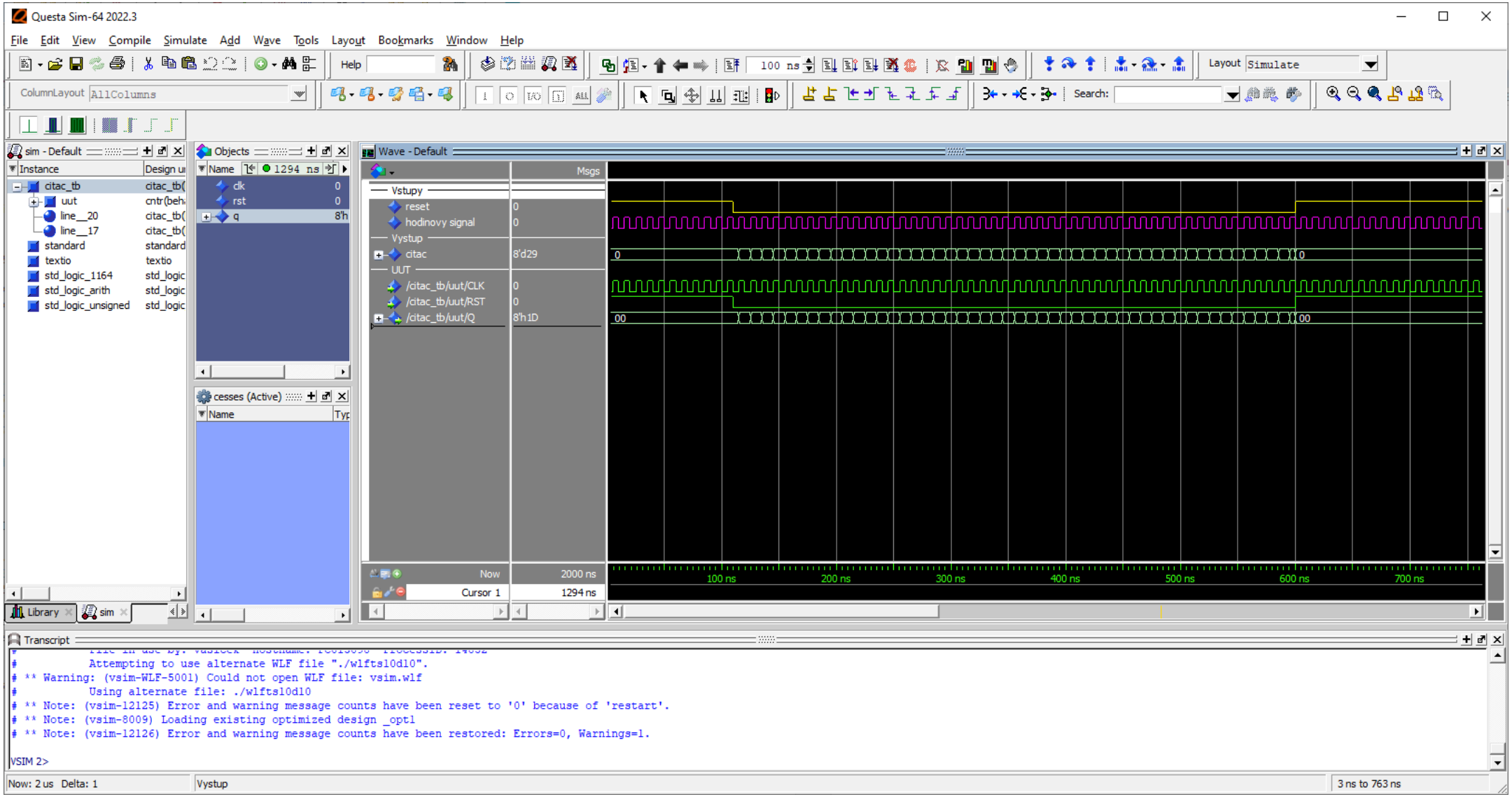
Typické části

- Kompilace zdrojových kódů
- Inicializace simulace
- Přidání signálů do waveform okna (volitelné)
- Nastavení hodnot signálů (volitelné)
- Spuštění simulace

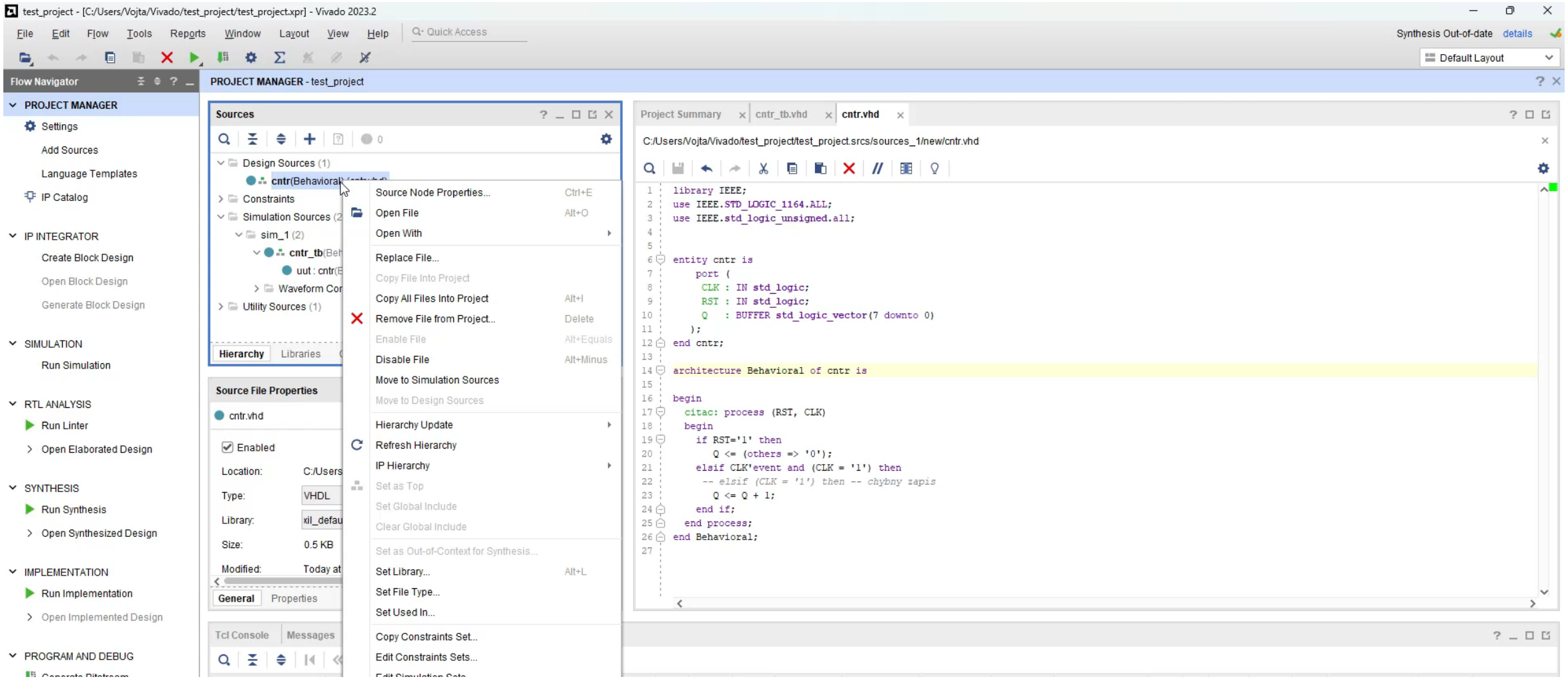
Spuštění simulace

vsim -do sim.fdo

Simulace

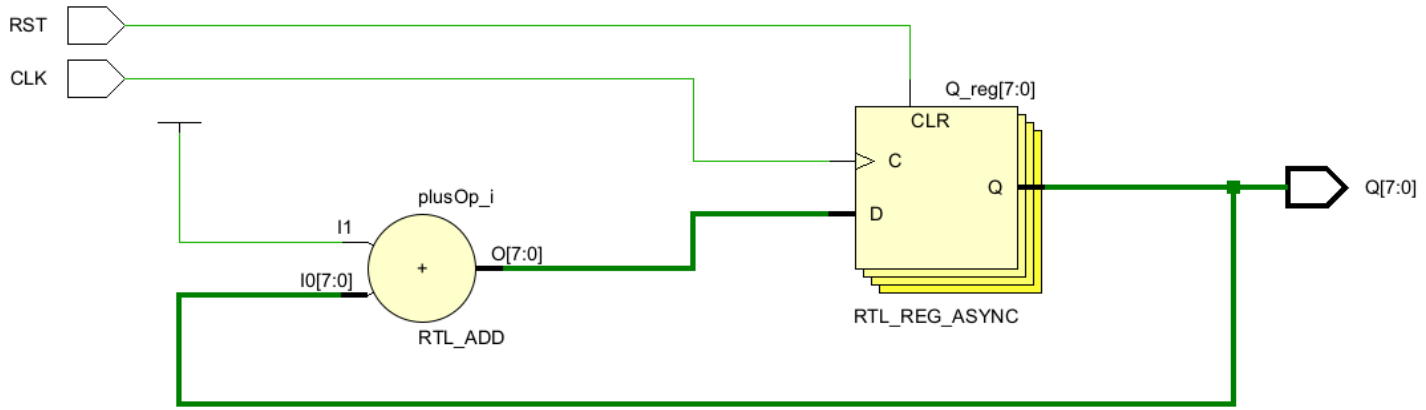

Questa Sim-64 2022.3
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Instance
citac_tb
uut
line__20
line__17
standard
textio
std_logic_1164
std_logic_arith
std_logic_unsigned
Objects
Name
clk
rst
q
Wave - Default
Msgs
Vstupy
reset
hodinovy signal
Vystup
citac
UUT
/citac_tb/uut/CLK
/citac_tb/uut/RST
/citac_tb/uut/Q
8'd29
8'h1D
Now
2000 ns
Cursor 1
1294 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
Transcript
** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
Using alternate file: ./wlftsl0dl0
** Note: (vsim-12125) Error and warning message counts have been reset to '0' because of 'restart'.
** Note: (vsim-8009) Loading existing optimized design _opt1
** Note: (vsim-12126) Error and warning message counts have been restored: Errors=0, Warnings=1.
VSIM 2>
Now: 2 us Delta: 1
Vystup
3 ns to 763 ns

Simulace: AMD Vivado

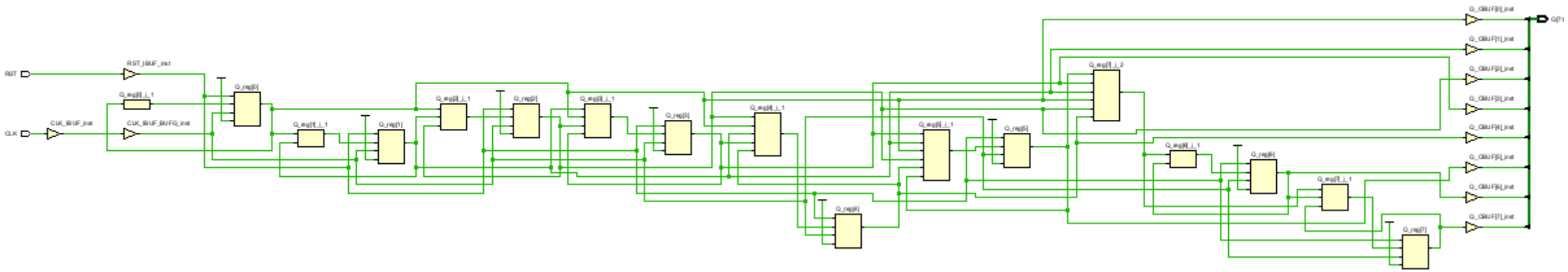


Syntéza

Proces převodu ze základních entit

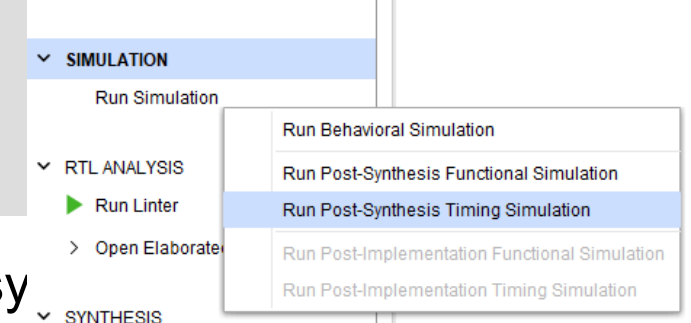


na reprezentaci pomocí LUT, Registrů a dalších elementárních jednotek.

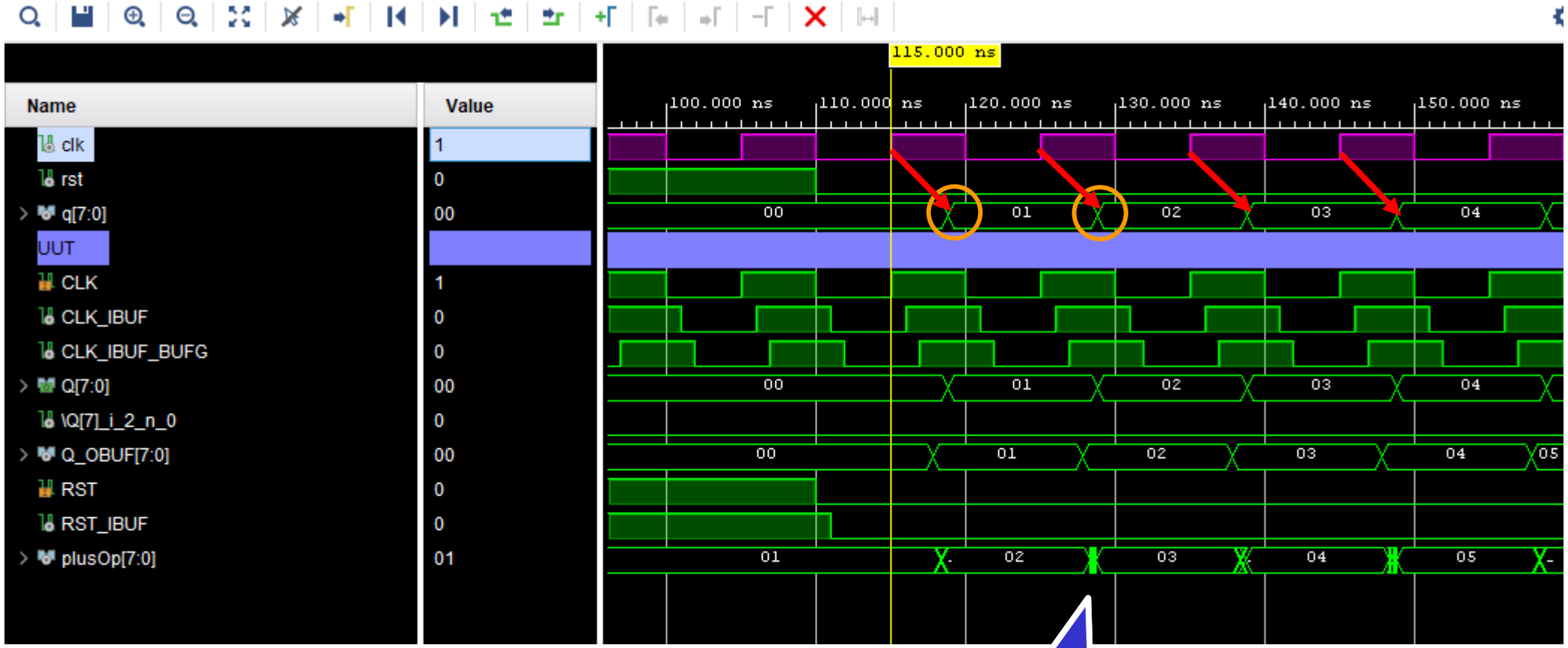


Simulace po syntéze

výsledek časové simulace (detail)



- Ke změně výstupu v praxi nikdy nedochází ihned po náběžné hraně, sy
  garantovat, že ke změně dojde vždy před novou náběžnou hranou (viz maximální pracovní
  frekvence)



Post-route simulation

Simulace po syntéze

chybná konstrukce registru ve funkční a časové simulaci

správný zápis (registr)

```
elsif CLK'event and (CLK = '1') then
```

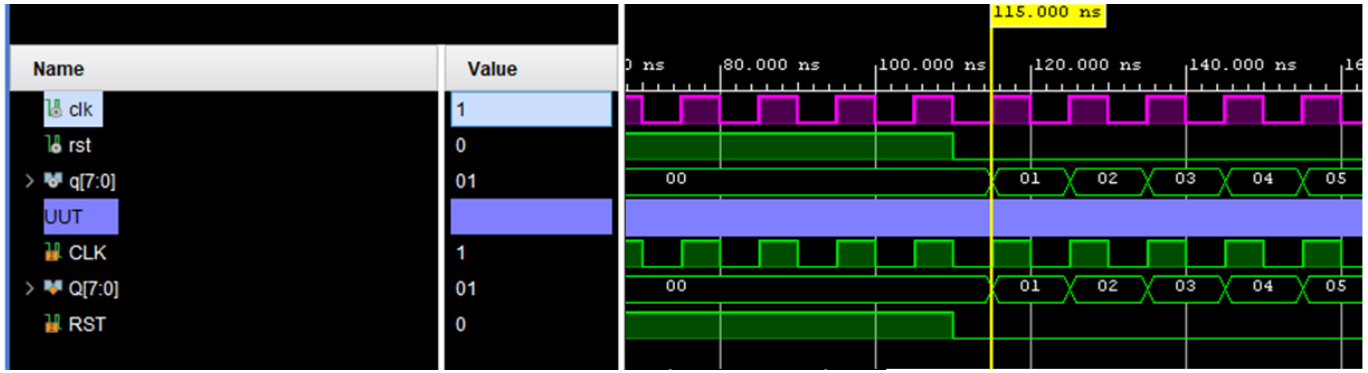
Rozdíl funkční a časové simulace
(chybně popsaný čítač zneužívající chování sensitivity listu)

chybně (latch)

```
citac: process (RST, CLK)
  begin
    if RST='1' then
       Q <= (others => '0');
    elsif (CLK = '1') then
       Q <= Q + 1;
    end if;
  end process;
```

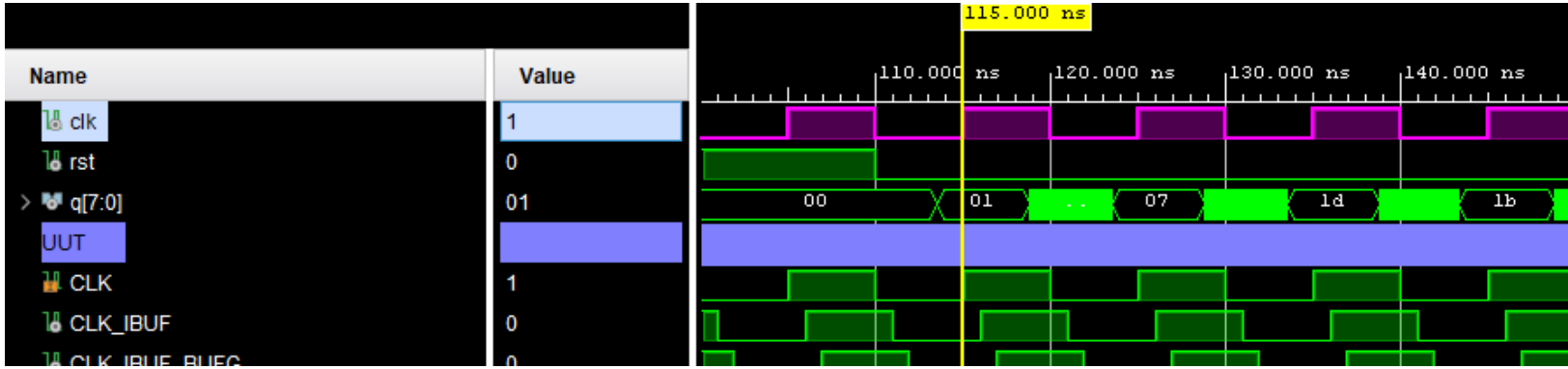
Behavioral Simulation

```
citac: process (RST, CLK)
```



Post-Route Simulation

```
citac: process (RST, CLK, Q)
```



Chybné konstrukce

většinou se projeví po syntéze

Pravidla, která je vhodné dodržovat

- Popis kombinačních obvodů
  - dataflow nebo proces s plným sensitivity listem (tzn. signály na pravé straně výrazů)

```
a <= b AND c;
```

```
PROCESS (b, c) IS
BEGIN
    a <= b AND c;
END PROCESS;
```

- Popis sekvenčních obvodů
  - proces s podmínkou na hodinovou hranu, sensitivity list obsahuje CLK případně RESET

```
PROCESS (clk) IS
BEGIN
    if clk = '1' and clk'event then
        -- ...
    end if;
END PROCESS;
```

- Proměnné a signály uvnitř procesu vždy inicializovat (nepoužívat k uchování stavu) – jinak vzniká nepříjemný LATCH

```
PROCESS (pstate) IS
BEGIN
    a <= '0';
    b <= '0';
    case pstate IS
        when s0 =>
            a <= '1';
        when s1 =>
            b <= '1';
        when others => null
    end;
END;
```

Chybné konstrukce

- Jeden konkrétní signál nelze ovládat z více procesů

```
architecture BAD of EXAMPLE is
  signal led : std_logic;
begin

process(CLK)
begin
  if (CLK'event) and (CLK='1') then
     if (...) then
       led <= '1'
     end if;
 end if;
end process;

process(CLK)
begin
  if (CLK'event) and (CLK='1') then
     if (...) then
       led <= '1';
     end if;
  end if;
end process;
```

```
architecture VALID of EXAMPLE is
  signal led : std_logic;
begin

process(CLK)
begin
  if (CLK'event) and (CLK='1') then
     if (...) or (...) then
        led <= '1'
     end if;
 end if;
end process;
```

Chybné konstrukce

- Špatný popis sekvenčního prvku s asynchronním nulováním. Asynchronní nulování musí mít vždy nejvyšší prioritu.

```
architecture BAD of EXAMPLE is
  signal led : std_logic;
begin

process(RESET, CLK)
begin
  if (RESET='1') then
     led <= '0';
  end if;
  if (CLK'event) and (CLK='1') then
     if (...) then
        led <= '1'
     end if;
 end if;
end process;
```

```
architecture VALID of EXAMPLE is
  signal led : std_logic;
begin

process(RESET, CLK)
begin
  if (RESET='1') then
     led <= '0';
  elsif (CLK'event) and (CLK='1') then
     if (...) then
        led <= '1'
     end if;
 end if;
end process;
```

Chybné konstrukce

- Špatný popis sekvenčního prvku s asynchronním nulováním. Asynchronní nulování musí mít vždy nejvyšší prioritu.

```
architecture BAD of EXAMPLE is
  signal led : std_logic;
begin

process(RESET, CLK)
begin
  if (CLK'event) and (CLK='1') then
     if (...) then
        led <= '1'
     end if;
  end if;
  if (RESET='1') then
     led <= '0';
  end if;
end process;
```

```
architecture VALID of EXAMPLE is
  signal led : std_logic;
begin

process(RESET, CLK)
begin
  if (RESET='1') then
     led <= '0';
  elsif (CLK'event) and (CLK='1') then
     if (...) then
        led <= '1'
     end if;
 end if;
end process;
```

Chybné konstrukce

- Špatný popis sekvenčního prvku s asynchronním nulováním
  Asynchronní nulování nesmí nikdy reagovat na synchronní událost

```
architecture BAD of EXAMPLE is
  signal led : std_logic;
begin

process(RESET, CLK)
begin
  if rising_edge(RESET) then
     led <= '0';
  end if;
  if falling_edge(RESET) and/or/if
    (CLK'event) and (CLK='1') then
     if (...) then
        led <= '1'
     end if;
    end if;
  end if;

end process;
```

```
architecture VALID of EXAMPLE is
  signal led : std_logic;
begin

process(RESET, CLK)
begin
  if (RESET='1') then
     led <= '0';
  elsif (CLK'event) and (CLK='1') then
     if (...) then
        led <= '1'
     end if;
 end if;
end process;
```

Nevhodné konstrukce

- Nevhodný popis sekvenčního chování, který vede na zhoršené parametry výsledného hardware (hodinový signál je veden datovou cestou)

```
architecture BAD of EXAMPLE is
    signal clk_slow : std_logic := '0';
begin

process(CLK)
begin
  if (CLK'event) and (CLK='1') then
     if (...) then
        clk_slow <= not clk_slow;
     end if;
  end if;
end process;

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

```
architecture VALID of EXAMPLE is
    signal clken : std_logic := '0';
begin

process(CLK)
begin
  if (CLK'event) and (CLK='1') then
   clken <= '0';
   if (...) then clken <= '1'; end if;
  end if;
end process;

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