

Reconfigurable Embedded Systems: Übung #2

Abgabe am Montag, 27. Oktober 2014

Günther Schindler

Inhaltsverzeichnis

Seriell-Parallel-Wandler	3
4-Bit-Multiplexer	5

Seriell-Parallel-Wandler

Folgender VHDL-Code (SP_Decoder) beschreibt das Verhalten eines Seriell-Parallel-Wandlers. Ein Eingangssignal (din) wird taktweise eingelesen und auf vier Ausgangssignale (LED_REG) nach dem vierten Takt geschaltet. Realisiert wurde dieses Schieberegister mit einer Finite State Machine (FSM) mit vier Zuständen (S0-S4). Der asynchrone Reset setzt die Ausgangssignale sofort auf '0000' und die FSM in den Grundzustand S0.

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

entity SP_Decoder is
port( clk, reset, din: in std_logic; LED_REG: out std_logic_vector(3 downto 0));
end SP_Decoder;

architecture Behavioral of SP_Decoder is
signal INT_REG: std_logic_vector(3 downto 0) := (others => '0');
type states is (S0, S1, S2, S3);
signal state: states := S0;
begin
    process(clk, reset)
    begin
        if reset = '1' then
            INT_REG <= (others => '0');
            LED_REG <= (others => '0');
            state <= S0;
        elsif rising_edge(clk) then
            case state is
                when S0 =>
                    LED_REG <= INT_REG;
                    INT_REG <= INT_REG(2 downto 0) & din;
                    state <= S1;
                when S1 =>
                    INT_REG <= INT_REG(2 downto 0) & din;
                    state <= S2;
                when S2 =>
                    INT_REG <= INT_REG(2 downto 0) & din;
                    state <= S3;
                when S3 =>
                    INT_REG <= INT_REG(2 downto 0) & din;
                    state <= S0;
            end case;
        end if;
    end process;
end Behavioral;
```

Die zugehörige Testbench ("tb_SP_Decoder") simuliert hintereinander die drei Bitsequenzen '1101', '1001' und '0101'. Zur Synchronisation verwendet die Testbench den asynchronen Reset.

Die Prozedur "clk_procedure" wird eingesetzt um einen freilaufenden Takt zu vermeiden. Sie invertiert den Takt und wartet eine halbe Taktperiode (5 ns).

```

...
procedure clk_procedure(
    signal clk_IN: in std_logic;
    signal clk_OUT: out std_logic
)is
begin
    clk_OUT <= not clk_IN;
    wait for clk_period/2;
end clk_procedure;
...

```

Im Impulsprozess (stim_proc) wird durch zweimaligen Aufruf der clk_procedure ein Taktimpuls simuliert. Bitweise werden die Testsequenzen durch din eingelesen. Ist jeweils die 4-Bit-Sequenz eingelesen und der vierte Taktzyklus vollendet, wird mit der assert-Anweisung das Ausgangssignal LED_REG überprüft und eine Ausgabe an die Standardausgabe vorgenommen.

```

...
stim_proc: process
    begin
        reset <= '1';
        clk_procedure(clk, clk);
        clk_procedure(clk, clk);
        reset <= '0';
        -- first sequenz: 1101
        assert false report "INFO: Starting sequence '1101'" severity note;
        din <= '1';
        clk_procedure(clk, clk);
        clk_procedure(clk, clk);
        din <= '1';
        clk_procedure(clk, clk);
        clk_procedure(clk, clk);
        din <= '0';
        clk_procedure(clk, clk);
        clk_procedure(clk, clk);
        din <= '1';
        clk_procedure(clk, clk);
        clk_procedure(clk, clk);
        -- second sequenz: 1001
        din <= '1';
        clk_procedure(clk, clk);
        assert (LED_REG = "1101") report "ERROR: Output is not equal '1101', failure!" severity error;
        assert (LED_REG /= "1101") report "INFO: Output is equal '1101', ok!" severity note;
        assert false report "INFO: Starting sequence '1001'" severity note;

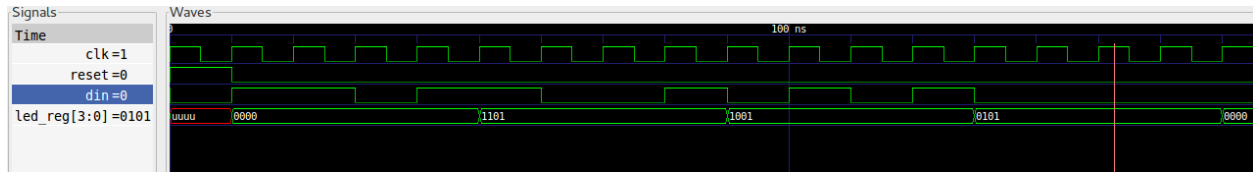
        ...

        wait;
    end process;

```

Die analyse der Waveform bestätigt das korrekte Verhalten des Seriell-Parallel-Wandlers. Nachdem der reset auf '0' gesetzt wird, werden bitweise die Signale auf die Eingangsleitung (din) übertragen. Nach dem vierten

Taktzyklus (angefangen ab der ersten steigenden Flanke nachdem reset = '0') wird das Signal '1101' auf die Ausgangssignalleitung (LED_REG) geschaltet. Nach jeweils weiteren vier Taktzyklen wird LED_REG '1001' und danach '0101'.



Auch die assertion-Ausgaben der Testbench bestätigen das korrekte Verhalten.

```
tb_SP_Decoder.vhd:59:16:@10ns:(assertion note): INFO: Starting sequence '1101'
tb_SP_Decoder.vhd:76:16:@55ns:(assertion note): INFO: Output is equal '1101', ok!
tb_SP_Decoder.vhd:77:16:@55ns:(assertion note): INFO: Starting sequence '1001'
tb_SP_Decoder.vhd:92:16:@95ns:(assertion note): INFO: Output is equal '1001', ok!
tb_SP_Decoder.vhd:93:16:@95ns:(assertion note): INFO: Starting sequence '0101'
tb_SP_Decoder.vhd:108:16:@135ns:(assertion note): INFO: Output is equal '0101', ok!
tb_SP_Decoder.vhd:109:16:@135ns:(assertion note): INFO: Stop testsequence
```

4-Bit-Multiplexer

Folgender VHDL-Quellcode (MUX) zeigt eine Implementierung eines 4-input Multiplexers, basierend auf einer Look-up-table (LUT).

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

```
entity MUX is
```

```
port (D3, D2, D1, D0: in std_logic; SEL: in std_logic_vector(1 downto 0); Y: out std_logic);
end MUX;
```

```
architecture Behavioral of MUX is
```

```
begin
```

```
    process (SEL, D3, D2, D1, D0)
```

```
    begin
```

```
        case SEL is
```

```
            when "00" => Y <= D0;
```

```
            when "01" => Y <= D1;
```

```
            when "10" => Y <= D2;
```

```
            when "11" => Y <= D3;
```

```
            when others => NULL;
```

```
        end case;
```

```
    end process;
```

```
end Behavioral;
```

Die Testbench (tb_MUX) des Multiplexers prüft alle möglichen Kombinationen von Datensignale (D3 - D0) und Steuersignal (SEL).

Um die Steuersignale zu variieren wird die Prozedur p_inc_slv verwendet, welche die übergeben Adresse inkrementiert.

```
procedure p_inc_slv (
    signal r_IN: in std_logic_vector(1 downto 0);
    signal r_OUT: out std_logic_vector(1 downto 0)
```

```

) is
begin
  r_OUT <= std_logic_vector(unsigned(r_IN) + 1);
end p_inc_slv;

```

Im Impulsprozess werden die Datensignale über eine for-Schleife variiert (0000, 0001, ..., 1000). Eine weitere for-Schleife variiert die Steuersignale (00,01,...,11). Die Testbench verwendet einen freilaufenden Takt.

```

...
signal ADDR: std_logic_vector(1 downto 0) := (others => '0');
signal CNT : std_logic_vector(3 downto 0) := (others => '0');
...
clk_process :process
begin
  clk <= '0';
  wait for clk_period/2;
  clk <= '1';
  wait for clk_period/2;
end process;
...
stim_proc: process
begin
  ...
  for J in 0 to 4 loop
    for I in 0 to 3 loop
      SEL <= ADDR;
      p_inc_slv(ADDR,ADDR);
      wait for clk_period;
    end loop;
    CNT <= (others => '0');
    CNT(J) <= '1';
  end loop;
  wait;
end process;

```

Wie folgende Waveform der Simulation zeigt, liefert der Multiplexer die gewünschten Ergebnisse. Ist das Select-Signal (SEL) beispielsweise auf Adresse '00', wird das Datensignale D0 an den Ausgang (Y) geschaltet. Bei Adresse '01' wird D1 geschaltet, etc.

