Hardware Design
Übungsblatt 1

18. November 2016

1. Starten Sie die Xilinx ISE und legen Sie ein Projekt mit dem Namen ihrer Wahl an. Folgen Sie dem Wizard und geben Sie folgende Daten ein:
   - 1 Tab: Projektname etc.
   - 2 Tab: Family: Virtex 5, Device: XC5VLX330, Package: FF1760, Speed: -1, Simulator: ISE Simulator
   - 3 Tab: New Source... → Wählen Sie „VHDL Module“ und geben Sie den Namen „ALU“ ein. Sie können optional auf den folgenden Seiten des „New Source“-Wizards bereits die Ports der generierten Entity definieren.
   - Schließen Sie die Erstellung des Projektes ab.
2. Implementieren Sie eine ALU über ein Verhaltensmodell wie folgt:

```
1  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.STD_LOGIC_ARITH.ALL;
   use IEEE.STD_LOGIC_UNSIGNED.ALL;

6  entity ALU is
       generic ( WD : integer := 16 );
       port ( a, b : in std_logic_vector(WD-1 downto 0);
              s : in std_logic_vector(1 downto 0);
              o : out std_logic_vector(WD-1 downto 0));
11 end ALU;
   architecture Behavioral of ALU is
   begin
       process (a,b,s)
       begin
16         if (s = "00") then
               o <= a+b;
           elsif (s = "10") then
               o <= a-b;
           elsif (s = "01") then
21             o <= not a;
           end if;
       end process;
```

```
    process (o)
26  begin
        if (clk'event and clk = '1') then
            s <= o;
        end if;
    end process;
31
end Behavioral;
```

- Was fehlt in der Beschreibung bzw. was ist eventuell inkorrekt? Korrigieren Sie die Entität und synthetisieren Sie diese? Wie sieht das Ergebnis der Synthese aus?
- Implementieren Sie den Addierer mit einem Überlaufbit.

3. Verifizieren Sie die Funktionsweise der ALU. Legen Sie dazu eine Testbench (TB) wie folgt an:
   - Rechtsklick im Source-Browser → New Source...
   - Wählen Sie VHDL Test Bench, geben Sie einen Namen ein, assoziieren Sie die TB mit der ALU-Entität und schließen Sie das Erstellen der TB ab.
   - Applizieren sie diverse Eingaben an den Eingangsports der ALU.
   - Simulieren Sie die Testbench:
     - Wählen Sie dazu oberhalb des Source-Browsers Behavioral Simulation
     - Selektieren Sie die TB im Source-Browser
     - Expandieren Sie Xilinx ISE Simulator im Process-Browser und doppelklicken Sie auf Simulate Behavioral Model

4. Implementieren Sie einen Carry-Ripple-Addierer über ein Strukturmodell.
   - Erstellen Sie hierzu Entitäten für einen Halb- und einen Volladdierer und nutzen Sie diese Module innerhalb der Addierer-Entität.

Halbaddierer	Volladdierer
$s = a \oplus b$	$s = a \oplus b \oplus c_{in}$
$c_{out} = ab$	$c_{out} = ab \vee ac_{in} \vee bc_{in}$

   - Implementieren Sie nun einen 4-Bit Carry-Ripple-Adder, der die erstellten Module miteinander verbindet. Hierzu müssen Sie Signale für die Summe und für die einzelnen Carry-Signale deklarieren und mit den Ports der Module verbinden.
   - Implementieren Sie den Addierer generisch. Deklarieren Sie analog zur ALU-Entität eine generische Variable und verwenden Sie folgendes Template zur Instanziierung der Komponenten.

```
<label> : for <laufvariable> in <start> to <ende> generate
    <instance_name> : <component_name>
3   port map (
        <port signal1> => <signal1>,
        ...
    );
end generate;
```

5. Erstellen Sie eine neue Entität für ein Schieberegister und programmieren Sie diese wie folgt:

```
entity shiftreg is
    port ( i, clk : in std_logic; o : out std_logic);
end shiftreg;
architecture Behavioral of shiftreg is
begin
    process (clk)
        variable a, b : std_logic;
    begin
        if (clk'event and clk = '1') then
            a := i; b := a; o <= b;
        end if;
    end process;
end Behavioral;
```

   Ändern Sie nun die Variablendeklaration und Zuweisung in eine Signaldeklaration bzw. Zuweisung. Synthetisieren Sie beide Versionen. Wo liegt der Unterschied?

6. Schreiben Sie ein Fibonacci Linear-Feedback-Shift-Register (LFSR)[1], wobei die Bitpositionen w_i (s.g. Taps), die in die Berechnung der Rückkopplung eingehen, konfigurierbar sein sollen. Dazu verfügt das Schieberegister über ein weiteres Register, welches das Konfigurationswort $P = p_0 p_1 \dots p_{n-1}$ enthält. Das Ausgabewort $W(t) = w_t w_{t-1} \dots w_{t-n-1}$ des LFSRs zum Zeitpunkt t ergibt sich nun aus dem vorherigen Ausgabewort $W(t-1) = w_{t-1} \dots w_{t-n-2}$ und der Rückkopplung

$$w_t = \bigoplus_{i=0}^{n-1} p_i w_{t-i-1}.$$

7. Generieren Sie mit dem Xilinx IP Generator einen RAM-Speicher (Block Memory) mit dem Sie anschließend einen FIFO realisieren. Der FIFO soll eine Breite und Tiefe von 16 Bit besitzen. Nutzen Sie einen Schreib- und einen Lesezähler zum adressieren der zu lesenden bzw. zu schreibenden Daten. Der FIFO soll über folgende Ein- und Ausgänge verfügen:
   - Reset: Solange dieser Eingang 0 ist, wird der Inhalt des FIFO's gelöscht.
   - Lesetakt: Eine positive Flanke auf diesem Eingang veranlasst das FIFO das nächste Wort aus dem internen Speicher am Ausgang anzulegen.
   - Schreibtakt: Eine positive Flanke auf diesem Eingang veranlasst das FIFO, das Wort, welches am Dateneingang ansteht, in den Speicher zu schreiben.
   - Dateneingang: zum Einlesen der Daten in das FIFO.
   - Dataausgang: Zur Ausgabe der Daten aus dem FIFO.
   - leer: Ist dieser Ausgang 1, so befinden sich keine Daten im FIFO.
   - voll: Ist dieser Ausgang 1, so können keine weiteren Daten ins FIFO geschrieben werden

Zur Bearbeitung der Aufgaben empfehlen wir unter anderem die nachfolgend aufgelistete Literatur.

[1] Siehe Wikipedia