Prof. Dr. Gundolf Kiefer *Hochschule Augsburg*
Michael Schäferling, M.Sc. *Fakultät für Informatik*

Hardware-Systeme – SS 2017

Praktikum: *VISCY-Prozessor* – Teil 2

In diesem Teil des Praktikums wirdeine Strukturbeschreibung der kompletten CPU erstellt werden. Dazu fehlt im wesentlichen noch das Steuerwerk. Weiterhin müssen noch einige kleinere Module (Multiplexer am Eingang des Registerfiles und am Adress-Ausgang sowie Logik zum Treiben des Tristate-Datenbusses) modelliert werden. Entwerfen Sie hier zunächst *ein vereinfachtes Steuerwerk*, das nur einen kleinen Teil der Befehle unterstützt.

Ziel ist es, die komplette CPU - mit reduziertem Befehlssatz - erfolgreich zu simulieren.

Vorbereitung (alle)

a) Welche Ein- und Ausgänge hat das Steuerwerk? Welche Bedeutung hat welches Signal? Orientieren Sie sich an der unten angegebenen Entity-Deklaration für das Steuerwerk.

b) Das Steuerwerk soll als Moore-Automat realisiert werden. Zeichnen Sie ein Zustandsübergangsdiagramm, in dem alle Übergangsbedingungen und für jeden Zustand die auszuführenden Register-Transfer-Operationen angegeben sind.

   Beachten Sie hier zunächst *nur* die ALU- Befehle sowie die Befehle LDIL und LDIH.

c) Schreiben Sie ein Assembler-Programm, mit dem sich möglichst vollständig überprüfen lässt, ob alle ALU- und LDI-Befehle funktionieren.

d) Diskutieren Sie das Zustandsübergangsdiagramm und das Assembler-Programm mit Ihrem Betreuer.

Aufgabe 1: Struktur der CPU

a) Erstellen Sie eine Struktur-Beschreibung der CPU. Wenn das Steuerwerk erst später bzw. von einem anderen Teammitglied entworfen wird, erstellen Sie hierzu eine passende Entity mit leerer Architektur.

   Die noch fehlende Multiplexer-Logik können Sie innerhalb der CPU-Beschreibung durch entsprechende Prozesse oder Datenflussanweisungen modellieren (es ist nicht notwendig, hierfür neue Entities zu erzeugen).

   Beachten Sie, dass der Tristate-Bus *data* sauber getrieben wird (vgl. Vorlesung). Am besten deklarieren Sie zwei interne Signale *data_in* und *data_out* und schreiben einen separaten Prozess, der die Signale *data* und *data_out* erzeugt.

b) Analysieren Sie die CPU mit GHDL, und korrigieren Sie eventuelle Syntax-Fehler.

c) Präsentieren Sie das Ergebnis Ihrer Arbeit dem übrigen Projekt-Team.

Schnittstelle der CPU:

```
entity CPU is
  port (
    clk, reset: in std_logic;
    adr: out std_logic_vector (15 downto 0);
    data: inout std_logic_vector (15 downto 0);
    rd, wr: out std_logic;
    ready: in std_logic
  );
end CPU;
```

Aufgabe 2: Vereinfachtes Steuerwerk

Das Steuerwerk soll als Moore-Automat realisiert werden und muss zunächst **nur die ALU- und LDI-Befehle** ausführen können.

a) Setzen Sie Ihr Zustandsübergangsdiagramm wie in der Vorlesung beschrieben in synthetisierbaren VHDL-Code um. Definieren Sie zwei Prozesse für den Zustandsübergang und die Ausgabe.

b) Synthetisieren Sie Ihr Steuerwerk. Überprüfen Sie die XST-Ausgabe, z.B. ob ungewollte Flipflops oder Latches erzeugt wurden. Welche Zustandscodierung hat XST gewählt?

c) Präsentieren Sie das Ergebnis Ihrer Arbeit dem übrigen Projekt-Team.

Schnittstelle des Steuerwerkes:

```
entity CONTROLLER is
  port (
    clk, reset: in std_logic;
    ir: in std_logic_vector(15 downto 0);  -- Befehlswort
    ready, zero: in std_logic;          -- weitere Statussignale
    c_reg_ldmem, c_reg_ldi,             -- Auswahl beim Register-Laden
    c_regfile_load_lo, c_regfile_load_hi,  -- Steuersignale Reg.-File
    c_pc_load, c_pc_inc,                -- Steuereingänge PC
    c_ir_load,                          -- Steuereingang IR
    c_mem_rd, c_mem_wr,                 -- Signale zum Speicher
    c_adr_pc_not_reg : out std_logic    -- Auswahl Adress-Quelle
  );
end CONTROLLER;
```

Aufgabe 3: Testbench

a) Erstellen Sie nach der unten angegebenen Vorlage eine Testbench für die gesamte CPU. Die Testbench soll simulieren, wie das in dem Array *mem_content* definierte Programm ausgeführt wird. Am Ende sollte automatisch geprüft werden, ob die CPU mit Ihrem Programm das richtige Ergebnis geliefert hat.

   Den Inhalt von *mem_content* können Sie mithilfe von *seppas* und *viscy2l* (Befehl "*dump*") automatisch erzeugen lassen.

b) Simulieren Sie den in Aufgaben 1 und 2 erstellten Prozessor mit der Testbench und beseitigen Sie alle Fehler. Besprechen Sie das Ergebnis mit Ihrem Betreuer!

c) Präsentieren Sie das Ergebnis Ihrer Arbeit dem übrigen Projekt-Team.

Aufgabe 4: Software

In der Praxis ist es üblich, mit der Entwicklung der Software schon zu beginnen, bevor funktionierende Hardware zur Verfügung steht.

a) Schreiben Sie ein Assembler-Programm, das zwei 15 Bit breite positive Ganzzahlen multipliziert. Die Faktoren sind in den Speicherstellen 0x100 und 0x101 gespeichert, das Ergebnis soll an die Adresse 0x102 geschrieben werden.

   Assemblieren Sie Ihr Programm mit *seppas* und testen Sie es mit dem Simulator.

b) Präsentieren Sie das Ergebnis Ihrer Arbeit dem übrigen Projekt-Team.

Anhang

Vorlage für die Testbench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;


ENTITY cpu_tb IS
END cpu_tb;


ARCHITECTURE behavior OF cpu_tb IS

  -- Component Declaration for the Unit Under Test (UUT)...
  COMPONENT cpu
    PORT (
      clk, reset, ready : IN std_logic;
      rd, wr : OUT std_logic;
      adr : OUT std_logic_vector(15 downto 0);
      data : INOUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;

  -- Signals...
  SIGNAL clk, reset, ready, rd, wr: std_logic;
  SIGNAL adr, data: std_logic_vector (15 downto 0);

  -- Parameters...
  constant clk_period: time := 10 ns;
  constant mem_delay: time := 25 ns;

  -- Memory content (created by viscy2l) ...
  -- hier Ausgabe von viscy2l einfügen
```

```
BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: cpu PORT MAP(
    clk => clk, reset => reset,
    rd => rd, wr => wr, ready => ready,
    adr => adr, data => data
  );

  -- Process to simulate the memory behavior...
  memory: process
  begin
    data <= "ZZZZZZZZZZZZZZZZ";
    ready <= '0';
    wait on rd, wr;
    if rd = '1' then
      wait for mem_delay;
      data <= mem_content (conv_integer ('0' & adr));
      ready <= '1';
      wait until rd = '0';
      data <= "ZZZZZZZZZZZZZZZZ";
      wait for mem_delay;
      ready <= '0';
    elsif wr = '1' then
      wait for mem_delay;
      mem_content (conv_integer ('0' & adr)) <= data;
      ready <= '1';
      wait until wr = '0';
      wait for mem_delay;
      ready <= '0';
    end if;
  end process;


  -- Main testbench process...
  tb : PROCESS

    procedure run_cycle is
    begin
      clk <= '0';
      wait for clk_period / 2;
      clk <= '1';
      wait for clk_period / 2;
    end procedure;

  BEGIN

    -- sinnvolles Hauptprogramm überlegen

    wait;     -- wait forever (stop simulation)
  END PROCESS;
END;
```