Rechnerarchitektur
Sommersemester 25

ertis embedded real-time systems ulm university institute

Laborprojekt – Versuch 10

Synthese des RISC-V

Ziel des Versuchs

Dieser Versuch führt Sie in die Synthese und Ausführung eines einfachen RISC-V-Prozessors auf einem **DE10-Standard FPGA-Board** (Cyclone V) ein.

Versuchsziel

- Synthese des Designs mit Quartus Prime 23.1 Lite und Programmierung auf das **DE10-Standard Board**.

Praktische Umsetzung

- Übertragung des Projekts auf das FPGA mittels USB-Blaster.
- Beobachtung des Pipeline-Verhaltens anhand der LEDs und optional über die HEX-Ausgabe.

Voraussetzungen

- Ubuntu 24.04
- Intel Quartus Prime 23.1/24.1 Lite Edition (`https://fpgasoftware.intel.com/`)
- DE10-Standard FPGA-Board (Cyclone V)
- USB-Blaster-Treiber korrekt installiert

Aufgabe 1: Neues Projekt erstellen

1. Starten Sie Quartus und wählen Sie im Menü **File** → **New Project Wizard**.
2. Wählen Sie ein Verzeichnis, z. B. `~/projects/riscv_project`.
3. Verwenden Sie als Projektnamen `riscv`.
4. Überspringen Sie das Hinzufügen von Dateien (Sie verwenden später eigene Dateien).
5. Wählen Sie als Zielchip: **Cyclone V** → **5CSXFC6D6F31C6**.
6. Schließen Sie den Assistenten ab.

Aufgabe 2: HDL-Dateien hinzufügen, anpassen und kompilieren

1. In Quartus: **Project** → **Add/Remove Files**.
2. Fügen Sie Ihre `.vhd`, `.vhdl` oder `.v` Dateien hinzu.
3. Passen Sie Ihre RISC-V-Umsetzung wie folgt an:

   - Implementieren Sie z. B. `risc_v` mit folgenden Schnittstellen:

```
entity risc_v is
  port (
    KEY       : in  std_logic_vector(1 downto 0);
    SW        : in  std_logic_vector(9 downto 0);
    LEDR      : out std_logic_vector(7 downto 0);
    HEX0, HEX1 : out std_logic_vector(6 downto 0);
    HEX2, HEX3 : out std_logic_vector(6 downto 0);
    HEX4, HEX5 : out std_logic_vector(6 downto 0)
  );
end entity risc_v;
```

   - Die Schalter `SW(9 downto 0)` dienen der Eingabe von Konfigurations- oder Steuerparametern.
   - Die Taster `KEY(1 downto 0)` werden zum manuellen Takt- und Reset-Handling verwendet. Verbinden Sie diese wie folgt:

```
s_rst <= not KEY(0);
s_clk <= KEY(1);
```

   - Die `LEDR(7 downto 0)` zeigen den aktuellen Zustand eines gewählten Registers an:

```
LEDR(7 downto 0) <= s_registersOut(to_integer(unsigned(
    SW(4 downto 0))))(7 downto 0);
```

   - Die 7-Segmentanzeigen `HEX0` bis `HEX5` geben die letzten 24 Bit der aktiven Instruktion aus. Verwenden Sie dafür die Entity `hex_7seg_decoder.vhdl`:

```
-- Generate-Schleife für 6 Decoderinstanzen
gen_decoder : for i in 0 to 5 generate
  decoder_inst : entity work.hex_7seg_decoder
  port map (
    bin_in  => s_instrIF((i*4 + 3) downto i*4),
    seg_out => seg_patterns(i)
  );
end generate;

-- Zuweisung zu den HEX-Anzeigen
HEX0 <= seg_patterns(0);
HEX1 <= seg_patterns(1);
HEX2 <= seg_patterns(2);
HEX3 <= seg_patterns(3);
HEX4 <= seg_patterns(4);
HEX5 <= seg_patterns(5);
```

- Implementieren Sie den ersten Testfall aus dem letzten Versuch direkt in Ihrer Top-Level-Entity. Ergänzen Sie hierzu den Befehls-Cache:

```
s_instructions(1)  <= x"00900093"; -- ADDI x1, x0, 9
s_instructions(2)  <= x"00800113"; -- ADDI x2, x0, 8
s_instructions(3)  <= x"         "; -- OR x10, x1, x2
s_instructions(4)  <= x"         "; -- ADD x8, x1, x2
s_instructions(5)  <= x"         "; -- SUB x11, x1, x2
s_instructions(6)  <= x"         "; -- SUB x12, x2, x1
s_instructions(7)  <= x"         "; -- ADD x12, x2, x8
s_instructions(8)  <= x"         "; -- SUB x12, x2, x1
s_instructions(9)  <= x"         "; -- AND x1, x2, x1
s_instructions(10) <= x"         "; -- XOR x12, x1, x2
s_instructions(11) <= x"         "; -- LUI x13, 8
s_instructions(12) <= x"         "; -- LUI x13, 29
s_instructions(13) <= x"         "; -- AUIPC x14, 1
s_instructions(14) <= x"         "; -- AUIPC x14, 1
```

- Importieren Sie die zur Verfügung gestellte Datei `DE10.qsf`.
- Kompilieren Sie das Projekt über: **Compile** → **Start Compilation**.
- Prüfen Sie den Status im Fenster **Messages**.
- Kontrollieren Sie unter **Assignments** → **Pin Planner**, ob die Zuweisungen korrekt übernommen wurden.

Aufgabe 3: Programmierung des DE10-Boards

1. Schließen Sie das DE10 per USB-Blaster an.
2. Versorgen Sie das Board mit Strom und schalten Sie es über den roten Knopf ein.

3. Öffnen Sie den Programmer: **Tools** → **Programmer**.
4. Klicken Sie auf **Hardware Setup** und wählen Sie `USB-Blaster`.
5. Fügen Sie das `HPS+SoC` hinzu.
6. Klicken Sie auf **Auto Detect** – der Chip `5CSXFC6D6F31C6` sollte erscheinen.
7. Fügen Sie die generierte `.sof`-Datei hinzu.
8. Aktivieren Sie `Program/Configure` und starten Sie die Programmierung.
9. Testen Sie Ihre Umsetzung, indem Sie Instruktionen mit den HEX-Displays und Registerinhalte über die LEDs auswerten.

Aufgabe 4: Erweiterung: Eigener Algorithmus

Erweitern Sie Ihre Umsetzung, um einen selbstgewählten Algorithmus (z. B. aus Vorlesung oder Übung) auf dem System abzubilden. Erläutern Sie Ihrem Tutor die Ergebnisse. Sie können dabei über die bisherigen Ein-/Ausgabemöglichkeiten hinausgehen.

Aufgabe 5: Abgabe

Laden Sie die vollständige Ordnerstruktur (inkl. aller Dateien) als ZIP-Archiv auf Moodle hoch:

Name_Vorname_Versuch10.zip