Labo Complex Digitaal Ontwerp: Register file and data path

Xander Rasschaert & Cas Truyers
Klas: S2Elecfs
Datum: 21/10/2024

Inleiding

In dit labo hebben we een CPU-datapad beschreven in VHDL door eerder ontworpen componenten, zoals de registers en de ALU, samen te voegen. Het doel was een CPU bus te maken om deze componenten te verbinden, zodat de ALU aangestuurd kon worden en de resultaten, samen met de flags, in registers opgeslagen konden worden. We hebben hiervoor gewerkt met de basic register component en ook een RegisterFile component (ook gebaseerd op de basic register component). Een belangrijk aandachtspunt was het encoden van inputs voor de RegisterFile en ervoor zorgen dat de registers de juiste databreedte hebben.

1 Analyse

In de eerste stap van de opdracht hebben we een Register_file.vhd component beschreven met behulp van een basic_register component. Hiervoor hebben we ons gebaseerd op het schema in de opgave. In de tweede stap hebben we de data_path.vhd file gemaakt. Hierin hebben we de Register_file, de basic_registers en de ALU met elkaar verbonden om een volledig CPU-datapad te creëren.

1.1 Code Fragmenten

1.1.1 register_file.vhd

```
architecture Behavioral of register_file is
    -- TODO: declare what will be used

    -- Making a type for all the register outputs - each register has a vector (array) of bits
    type data_reg_out is array (0 to C_NR_REGS-1) of std_logic_vector (C_DATA_WIDTH-1 downto 0);

    -- signals
    signal data_out_s : data_reg_out;

    -- components
    component basic_register is
    generic(
        C_DATA_WIDTH : natural := 8
    );
    port(
                clk : in  std_logic;
              reset : in  std_logic;
                 le : in  std_logic;
            data_in : in  std_logic_vector(C_DATA_WIDTH-1 downto 0);
           data_out : out std_logic_vector(C_DATA_WIDTH-1 downto 0)
    );
    end component;
begin

    -- TODO: describe how it's all connected and how it basic_register

    -- Setting up all the registers
    REG_FILE: for i in C_NR_REGS-1 downto 0 generate
        basic_reg: basic_register
        generic map(C_DATA_WIDTH => C_DATA_WIDTH)
        port map(
```

```
            clk => clk,
            reset => reset,
            le => (le and in_sel(i)), -- Enable only if 'le' is active and register is selected
            data_in => data_in,
            data_out => data_out_s(i) -- Store each register's output in data_out_s(index)
        );
    end generate REG_FILE;

    -- Process that handles selecting which register output goes to data_out
    process (out_sel, data_out_s)
    begin
        data_out <= (others => '0');
        for i in 0 to C_NR_REGS-1 loop
            if out_sel(i) = '1' then
                data_out <= data_out_s(i);
            end if;
        end loop;
    end process;

end Behavioral;
```

1.1.2 data_path.vhd

```
architecture Behavioral of data_path is
--The given signals and component declerations etc from the assignment are not
    pasted in this section but are necessary to work.
begin
    -- TODO: complete description
    -- Outputs (map to internal signals)
    cpu_bus <= cpu_bus_i;
    flags <= flags_i;
    -- Inputs (map to internal signals <optional>)
    reg_file_le_i <= reg_file_le;
    Rd_i <= Rdestination;
    Rs_i <= Rsource;
    alu_op_i <= alu_op;
    y_le_i <= y_le;
    z_le_i <= z_le;
    flags_le_i <= flags_le;

    -- CPU bus control: select what gets output on the CPU bus using with select
        concurrent statement
    with cpu_bus_sel select
        cpu_bus_i <= dibr when SFR_DBR,
                     z_i when SFR_Z,
                     pc when SFR_PC,
                     IV when SFR_IV,
                     reg_file_out_i when GP_REG ,
                     sp when SFR_SP,
                     ir_l when SFR_IR_L,
                     (others => '0') when others;

    The following 3 code blocks initializes the registers used in the CPU, we link
         these inputs and outputs based on the provided schematic:

    -- ALU secondary input (Y register)
    ALU_Y_REG : basic_register
        generic map(
            C_DATA_WIDTH => C_DATA_WIDTH
        )
        port map(
            data_in => cpu_bus_i,
            le => y_le_i,
            clk => clk,
            reset => reset,
            data_out => y_i
        );

    -- ALU output (Z register)
    ALU_Z_REG : basic_register
        generic map(
```

```
        C_DATA_WIDTH => C_DATA_WIDTH
    )
        port map(
            data_in=>alu_out_i,
            le=>z_le_i,
            clk=>clk,
            reset=>reset,
            data_out =>z_i
        );

-- ALU flags register, import here is that the standard C_DATA_WIDTH is not
    used but we set it on C_NR_FLAGS.
ALU_Flag_REG : basic_register
    generic map(
        C_DATA_WIDTH => C_NR_FLAGS
    )
    port map(
        data_in=> alu_flags_i,
        le=>flags_le_i,
        clk=>clk,
        reset=>reset,
        data_out=> flags_i
    );

-- We initialize the ALU and connect the inputs and outputs based on the given
     schematic:
ALU_INST : ALU8bit
    generic map(
        C_DATA_WIDTH => C_DATA_WIDTH
    )
    port map(
        X=>cpu_bus_i,
        Y=> y_i,
        Z=> alu_out_i,
        op=>alu_op_i,
        zf=> alu_flags_i(C_ZF),
        cf=> alu_flags_i(C_CF),
        ef=> alu_flags_i(C_EF),
        gf=> alu_flags_i(C_GF),
        sf=> alu_flags_i(C_SF)
    );

 -- The following code blocks describes a mux with decoding functionality.
     This was necessary since the input to the reg select is a 3-bit, but the
     register expects a 8 bit input.

with Rs_i select
reg_file_in_sel_i<= "00000001" when "000",
                    "00000010" when "001",
                    "00000100" when "010",
                    "00001000" when "011",
                    "00010000" when "100",
                    "00100000" when "101",
                    "01000000" when "110",
                    "10000000" when "111",
                    (others => '0') when others;

with Rs_i select
reg_file_out_sel_i<= "00000001" when "000",
                    "00000010" when "001",
                    "00000100" when "010",
                    "00001000" when "011",
                    "00010000" when "100",
                    "00100000" when "101",
                    "01000000" when "110",
                    "10000000" when "111",
                    (others => '0') when others;

-- Instantiating the register file and again linking as seen in the schematic.
Reg_File_Inst : register_file
    generic map(
        C_DATA_WIDTH => C_DATA_WIDTH,
        C_NR_REGS => C_NR_REGS
```

```
        )
        port map(
            data_in => cpu_bus_i,
            le => reg_file_le_i,
            in_sel => reg_file_in_sel_i,
            out_sel => reg_file_out_sel_i,
            clk=>clk,
            reset=>reset,
            data_out => reg_file_out_i
        );

end Behavioral;
```

2 Simulaties

2.1 register_file.vhd

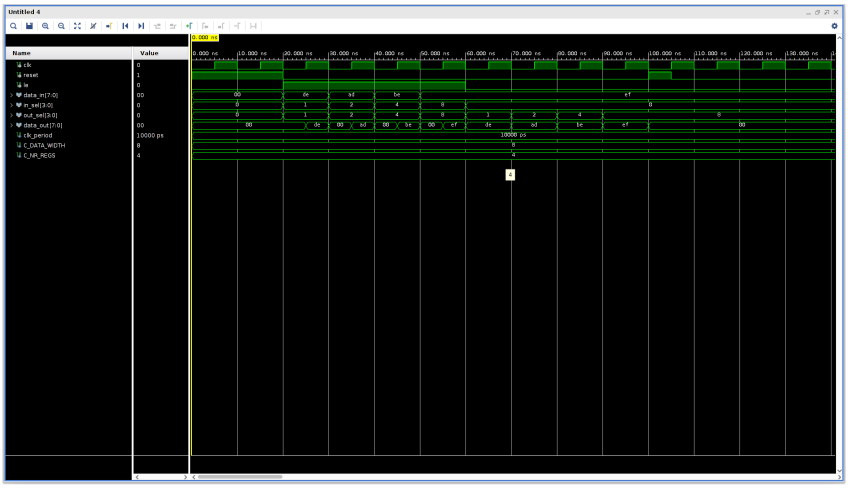

Figure 1: Golfvorm van de simulatie van register_file.vhd.

Simulatie Console Output:

SUCCESS: write register 0
SUCCESS: write register 1.
SUCCESS: write register 2.
SUCCESS: write register 3.
SUCCESS: read register 0.
SUCCESS: read register 1.
SUCCESS: read register 2.
SUCCESS: read register 3.
SUCCESS: reset.
Note: Simulation ended.
Time: 110 ns Iteration: 1

Figure 2: Console output van reg_file.vhd.

2.2 data_path.vhd

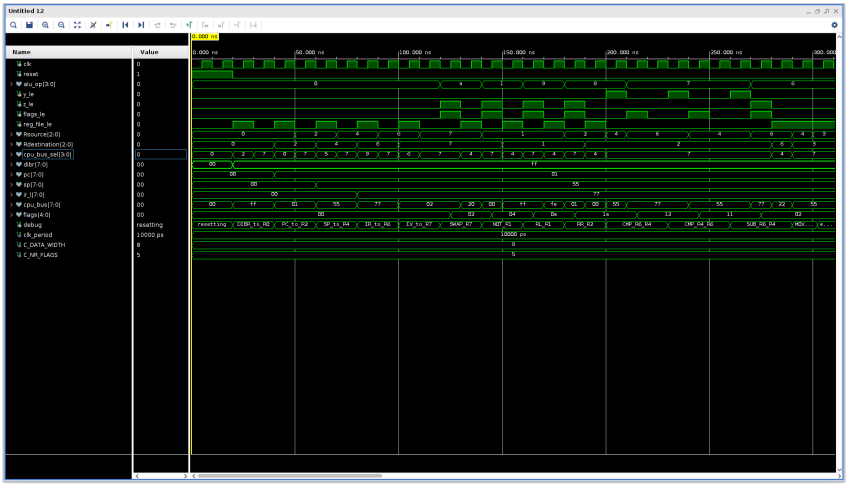

Figure 3: Golfvorm van de simulatie van data_path.vhd.

Simulatie Console Output:

```
SUCCESS: cpu bus selection (dibr)
SUCCESS: cpu bus selection (register 0)
SUCCESS: cpu bus selection (pc)
SUCCESS: cpu bus selection (register 2)
SUCCESS: cpu bus selection (sp)
SUCCESS: cpu bus selection (register 4)
SUCCESS: cpu bus selection (ir_l)
SUCCESS: cpu bus selection (register 6)
SUCCESS: cpu bus selection (iv)
SUCCESS: cpu bus selection (register 7)
SUCCESS: alu SWAP operation and cpu bus selection (alu output)
SUCCESS: alu NOT operation and cpu bus selection (alu output)
SUCCESS: alu RL operation and cpu bus selection (alu output)
SUCCESS: alu RL operation and cpu bus selection (alu output)
SUCCESS: alu CMP operation
SUCCESS: alu CMP operation
SUCCESS: alu SUB operation
SUCCESS: move R4 -> R3
Note:  Simulation ended.
Time:  302 ns Iteration:  0
```

Figure 4: Console output van data_path.vhd.

2.3 RTL-Schema's

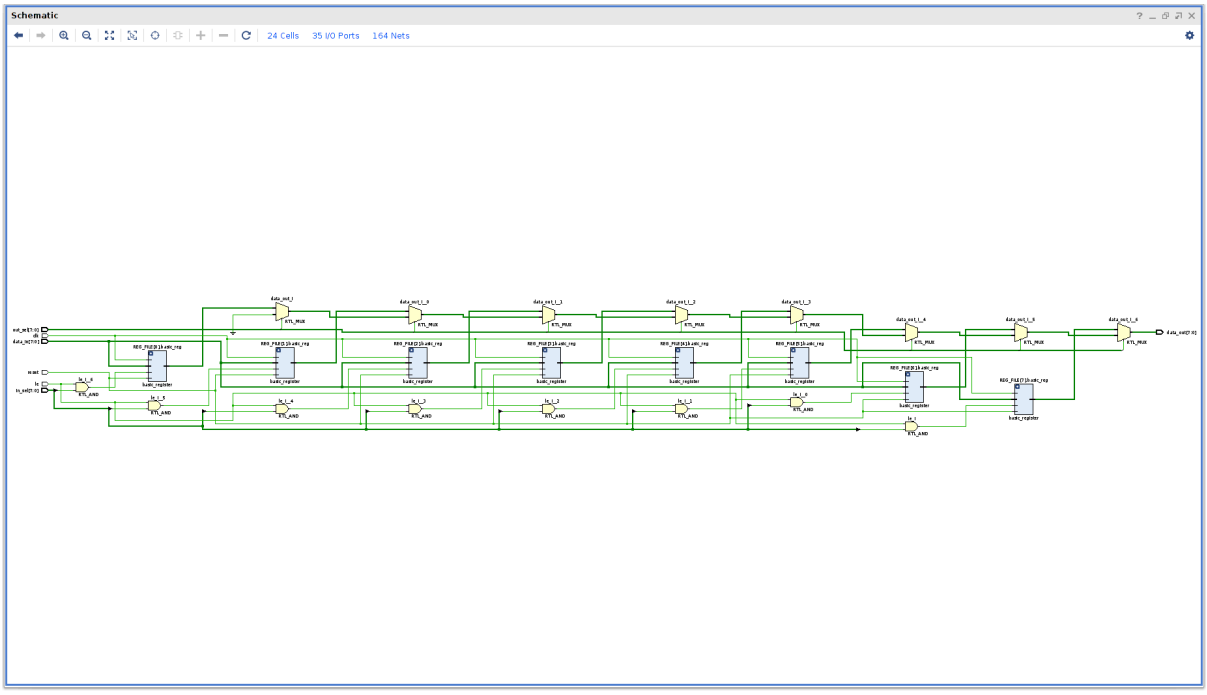

Figure 5: RTL schema van de register_file.vhd.

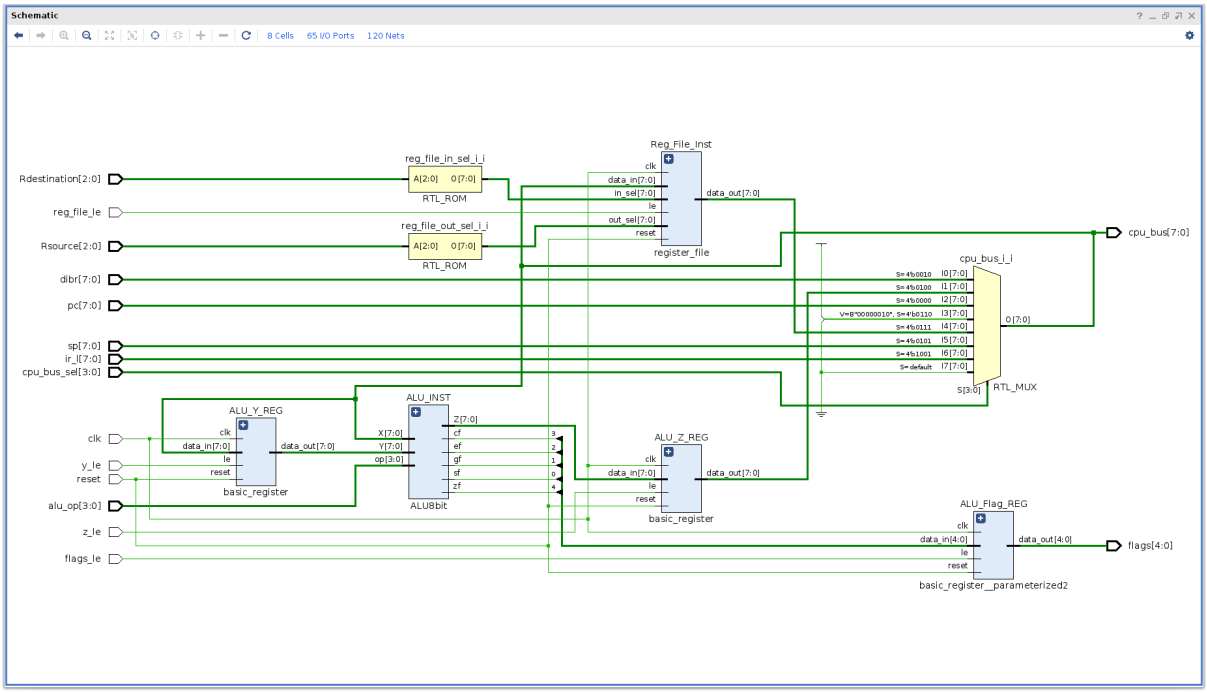

Figure 6: RTL schema van de data_path.vhd.

3 Besluit

3.1 Wat is gerealiseerd?

In dit labo hebben we een volledig werkend CPU-datapad gerealiseerd waarin verschillende componenten zoals de ALU, basic registers en een RegisterFile geïntegreerd zijn. Deze componenten zijn verbonden via een CPU-bus, wat het mogelijk maakt om data tussen de registers en de ALU te verplaatsen en te bewerken. Dit is succesvol getest geweest met een test bench in een behavioral simulation. Ook hebben we een encoding toegevoegd om de 3-bits selectiesignalen om te zetten naar een single encoded 8-bits formaat, wat nodig was voor de communicatie met de register_file register. Verder hebben we de vlaggen (flags) succesvol geïmplementeerd met een aangepast bitbreedte op basis van de hoeveelheid gebruikte vlaggen.

3.2 Wat kan er beter? Wat zijn de beperkingen?

Net zoals bij de vorige labo's is er geen grote focus gelegd op bepaald tijdsdoelen. De implementatie zou dus waarschijnlijk efficienter/sneller kunnen. Het zou eventueel ook verbeterd kunnen worden door het design zo te maken dat de encoding niet nodig is. Dit zal de efficientie van de implementatie bevorderen. Momenteel zien we geen verdere beperkingen, deze komen bij de volgende labo's eventueel wel nog naar boven wanneer we verder werken aan de CPU.

3.3 Hoe kunnen de problemen opgelost en uitbreidingen uitgevoerd worden?

Omdat we geen problemen zien met de huidige implementatie, kunnen we geen uitgebreid antwoord geven op welke problemen opgelost moeten worden.