

# Einführung in Computer Microsystems

## 8. Aufgabenblatt

### Sommersemester 2007

#### Aufgabe 1: Adressdekoder

Implementieren Sie als Verilog-Modul einen Adressdekoder für einen Bus mit einer CPU als Initiator/Master. Der Dekoder soll mit möglichst wenigen Gattern auskommen und für jeden der folgenden Slave-Teilnehmer ein separates **SELECT**-Signal erzeugen:

- RAM 64KB
- ROM 8KB
- ROM 8KB
- ROM 4KB
- 8 Register zu je 16 Bit
- 4 Register zu je 32 Bit

Jede Adresse adressiert ein Byte. Finden Sie eine geeignete Address-Map, Memory-Aliasing ist erlaubt. Es darf höchstens ein **SELECT**-Signal gleichzeitig aktiv sein. Prüfen Sie mit einer geeigneten Testbench, ob alle Adressen korrekt dekodiert werden.

#### Lösung:

Verilog-Modul:

```
`timescale 1ns / 1ps
module addr_decoder(ADDR, SELECT_RAM, SELECT_ROM8_1, SELECT_ROM8_2,
                   SELECT_ROM4, SELECT_REGISTER16, SELECT_REGISTER32);
    input          [16:0] ADDR;
    output         SELECT_RAM;
    output         SELECT_ROM8_1;
```

```

output          SELECT_ROM8_2;
output          SELECT_ROM4;
output reg [7:0] SELECT_REGISTER16;
output reg [3:0] SELECT_REGISTER32;

integer i;

// Binaere Zerlegung des Adressraumes
assign SELECT_RAM    = ~ADDR[16];
assign SELECT_ROM8_1 = ADDR[16] & ~ADDR[14] & ~ADDR[13];
assign SELECT_ROM8_2 = ADDR[16] & ~ADDR[14] & ADDR[13];
assign SELECT_ROM4   = ADDR[16] & ADDR[14] & ~ADDR[12];

// 4 KByte-Bereich bei 64K+16K+4K selektiert
wire ADDR64_16_4 = ADDR[16] & ADDR[14] & ADDR[12];

always @(*) begin
    // Einzelne Register dekodieren -- 16 Bit, jede 2. Adresse
    for (i = 0; i < 8; i = i + 1)
        SELECT_REGISTER16[i] = ADDR64_16_4 & ~ADDR[4] && (ADDR[3:1] == i);
    // Einzelne Register dekodieren -- 32 Bit, jede 4. Adresse
    for (i = 0; i < 4; i = i + 1)
        SELECT_REGISTER32[i] = ADDR64_16_4 & ADDR[4] && (ADDR[3:2] == i);
end

endmodule

```

Testbench:

```

`timescale 1ns / 1ps
module tb_addr_decoder_v;

    // Inputs
    reg [16:0] ADDR;

    // Outputs
    wire SELECT_RAM;
    wire SELECT_ROM8_1;
    wire SELECT_ROM8_2;
    wire SELECT_ROM4;
    wire [7:0] SELECT_REGISTER16;
    wire [3:0] SELECT_REGISTER32;

    integer i;

    // Instantiate the Unit Under Test (UUT)
    addr_decoder uut (
        .ADDR(ADDR),
        .SELECT_RAM(SELECT_RAM),
        .SELECT_ROM8_1(SELECT_ROM8_1),
        .SELECT_ROM8_2(SELECT_ROM8_2),
        .SELECT_ROM4(SELECT_ROM4),
        .SELECT_REGISTER16(SELECT_REGISTER16),
        .SELECT_REGISTER32(SELECT_REGISTER32)
    );

```

```

// Alle Signale ausgeben bei
// Aenderung aller Signale ausser ADDR
always @(SELECT_RAM, SELECT_ROM8_1, SELECT_ROM8_2,
        SELECT_ROM4, SELECT_REGISTER16, SELECT_REGISTER32)
    $display("ADDR=%h_SELECT_RAM=%b_SELECT_ROM8_1=%b_SELECT_ROM8_2=%b_SELECT_ROM4=%b_SELECT_REGISTER16=%b_SELECT_REGISTER32=%b",
            ADDR, SELECT_RAM, SELECT_ROM8_1, SELECT_ROM8_2,
            SELECT_ROM4, SELECT_REGISTER16, SELECT_REGISTER32);

initial begin
    // Initialize Inputs
    ADDR = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    // Ueberstreichungen ganzen Adressraum
    for (i = 0; i < 2**17; i = i + 1) begin
        ADDR = i; #1;
    end
    $stop;
end

endmodule

```

## Simulationsausgabe:

```

ADDR=00000 SELECT_RAM=1 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=0000000x SELECT_REGISTER32=0000
ADDR=00000 SELECT_RAM=1 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0000
ADDR=10000 SELECT_RAM=0 SELECT_ROM8_1=1 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0000
ADDR=12000 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=1 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0000
ADDR=14000 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=1 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0000
ADDR=15000 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0000
ADDR=15000 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000001 SELECT_REGISTER32=0000
ADDR=15002 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000010 SELECT_REGISTER32=0000
ADDR=15004 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000100 SELECT_REGISTER32=0000
ADDR=15006 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00001000 SELECT_REGISTER32=0000
ADDR=15008 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00010000 SELECT_REGISTER32=0000
ADDR=1500a SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00100000 SELECT_REGISTER32=0000
ADDR=1500c SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=01000000 SELECT_REGISTER32=0000
ADDR=1500e SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=10000000 SELECT_REGISTER32=0000
ADDR=15010 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0001
ADDR=15014 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0010
ADDR=15018 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0100
ADDR=1501c SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=1000
ADDR=15020 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000001 SELECT_REGISTER32=0000
ADDR=15022 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000010 SELECT_REGISTER32=0000
ADDR=15024 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000100 SELECT_REGISTER32=0000
ADDR=15026 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00001000 SELECT_REGISTER32=0000
ADDR=15028 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00010000 SELECT_REGISTER32=0000
ADDR=1502a SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00100000 SELECT_REGISTER32=0000
ADDR=1502c SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=01000000 SELECT_REGISTER32=0000
ADDR=1502e SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=10000000 SELECT_REGISTER32=0000
ADDR=15030 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0001
ADDR=15034 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0010
ADDR=15038 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0100
ADDR=1503c SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=1000
...

```

## Vielfache Wiederholung der Registerselektion durch Memory-Aliasing

```

...
ADDR=1ffe0 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000001 SELECT_REGISTER32=0000
ADDR=1ffe2 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000010 SELECT_REGISTER32=0000
ADDR=1ffe4 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000100 SELECT_REGISTER32=0000
ADDR=1ffe6 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00001000 SELECT_REGISTER32=0000
ADDR=1ffe8 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00010000 SELECT_REGISTER32=0000
ADDR=1ffea SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00100000 SELECT_REGISTER32=0000
ADDR=1ffec SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=01000000 SELECT_REGISTER32=0000
ADDR=1ffee SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=10000000 SELECT_REGISTER32=0000
ADDR=1fff0 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0001

```

```

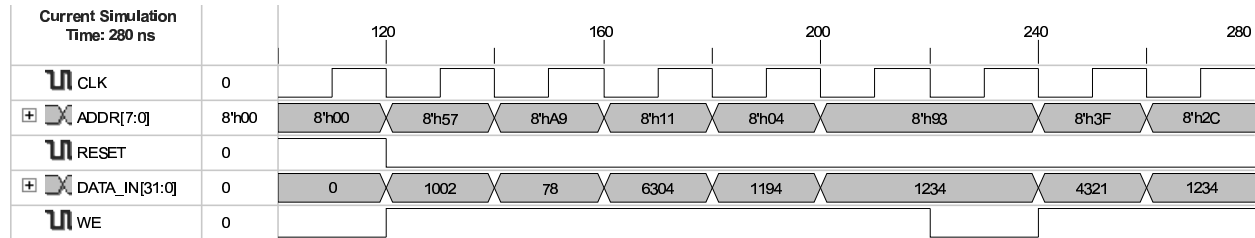
ADDR=1fff4 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0010
ADDR=1fff8 SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=0100
ADDR=1fffc SELECT_RAM=0 SELECT_ROM8_1=0 SELECT_ROM8_2=0 SELECT_ROM4=0 SELECT_REGISTER16=00000000 SELECT_REGISTER32=1000

```

## Aufgabe 2: Transaktionen

Gegeben sind eine Waveform und ein Verilog-Modul.

Waveform:



Verilog-Modul:

```

`timescale 1ns / 1ps
module memory_mapped_regs (CLK, RESET, ADDR, DATA_IN, WE, DATA_OUT);
    input CLK;
    input RESET;
    input [7:0] ADDR;
    input [31:0] DATA_IN;
    input WE;
    output [31:0] DATA_OUT;

    reg [31:0] A, B, C, D, E, F, G;

    assign DATA_OUT = ADDR[6] ? D : ADDR[5:4] == 2'b00 ? A :
                        ADDR[5:4] == 2'b01 ? B : ADDR == 42 ? E :
                        ADDR[7] ? C : ADDR[3:2] == 2 ? G : F;

    always @(posedge CLK or posedge RESET) begin
        if (RESET) begin
            A <= 1; B <= 2; C <= 3; D <= 4;
            E <= 5; F <= 6; G <= 7;
        end
        else
            case ({WE, ADDR[7], ADDR[4:2]})
                5'h17: A <= DATA_IN;
                5'h1F: B <= DATA_IN;
                5'h13: C <= DATA_IN;
                5'h14: D <= DATA_IN;
                5'h1A: E <= DATA_IN;
                5'h15: F <= DATA_IN;
                5'h1C: G <= DATA_IN;
                default: begin
                    G <= 42;
                    A <= DATA_IN;
                end
            endcase
    end
endmodule

```

```
end
endmodule
```

Die in der Waveform dargestellten Transaktionen werden als Stimulus in das Verilog-Modul eingegeben. Welche Werte enthalten danach die Register A bis G und auf welchen Adressen können sie jeweils ausgelesen werden? Hinweis: Das Modul enthält einige Designfehler beim Address-Decoding...

### Lösung:

Die Register haben nach den Transaktionen folgende Werte:

| Register | Wert (dezimal) |
|----------|----------------|
| A        | 4321           |
| B        | 2              |
| C        | 1234           |
| D        | 6304           |
| E        | 78             |
| F        | 1002           |
| G        | 42             |

Die nachstehende Testbench ermittelt die Registeradressen durch Auslesen der bekannten Reset-Werte, indem der gesamte Adressbereich überstrichen wird. Anschließend werden die Transaktionen der Waveform als Teststimuli formuliert und die sich daraus ergebenden Registerwerte ausgegeben:

```
`timescale 1ns / 1ps
module tb_memory_mapped_regs_v;

    // Inputs
    reg CLK;
    reg RESET;
    reg [7:0] ADDR;
    reg [31:0] DATA_IN;
    reg WE;

    // Outputs
    wire [31:0] DATA_OUT;

    integer i;

    // Instantiate the Unit Under Test (UUT)
    memory_mapped_regs uut (
        .CLK(CLK),
        .RESET(RESET),
        .ADDR(ADDR),
        .DATA_IN(DATA_IN),
        .WE(WE),
        .DATA_OUT(DATA_OUT)
    );

    // Erzeuge einen Taktzyklus
```

```

task DoClockCycle;
  begin
    #10;
    CLK = 1;
    #10;
    CLK = 0;
  end
endtask

// Alle Signale ausgeben bei
// Aenderung von DATA_OUT
always @(DATA_OUT)
  $display("ADDR=%h_DATA_OUT=%1d", ADDR, DATA_OUT);

initial begin
  // Initialize Inputs
  CLK = 0;
  RESET = 0;
  ADDR = 0;
  DATA_IN = 0;
  WE = 0;

  // Wait 100 ns for global reset to finish
  #100;

  // Add stimulus here
  // Reset
  RESET = 1;
  DoClockCycle;
  RESET = 0;

  // Ueberstreiche ganzen Adressraum
  // und finde Registerleseadressen
  $display("Finde_Registeradressen:");
  for (i = 0; i < 2**8; i = i + 1) begin
    ADDR = i; #1;
  end

  // Erzeuge Transaktionen der Waveform
  $display("Erzeuge_Transaktionen:");
  WE = 1;
  ADDR = 8'h57; DATA_IN = 1002;
  DoClockCycle;
  ADDR = 8'hA9; DATA_IN = 78;
  DoClockCycle;
  ADDR = 8'h11; DATA_IN = 6304;
  DoClockCycle;
  ADDR = 8'h04; DATA_IN = 1194;
  DoClockCycle;
  ADDR = 8'h93; DATA_IN = 1234;
  DoClockCycle;
  WE = 0;
  DoClockCycle;
  WE = 1;
  ADDR = 8'h3F; DATA_IN = 4321;

```

```

DoClockCycle;
ADDR = 8'h2C; DATA_IN = 1234;
DoClockCycle;
$display("A=%0d_B=%0d_C=%0d_D=%0d_E=%0d_F=%0d_G=%0d",
        uut.A, uut.B, uut.C, uut.D, uut.E, uut.F, uut.G);
$stop;
end

endmodule

```

Simulationsausgabe des Adresstests nach Reset, **DATA\_OUT=1**  $\mapsto$  Register A bis **DATA\_OUT=7**  $\mapsto$  Register G:

```

ADDR=00 DATA_OUT=1
ADDR=10 DATA_OUT=2
ADDR=20 DATA_OUT=6
ADDR=28 DATA_OUT=7
ADDR=2a DATA_OUT=5
ADDR=2b DATA_OUT=7
ADDR=2c DATA_OUT=6
ADDR=38 DATA_OUT=7
ADDR=3c DATA_OUT=6
ADDR=40 DATA_OUT=4
ADDR=80 DATA_OUT=1
ADDR=90 DATA_OUT=2
ADDR=a0 DATA_OUT=3
ADDR=c0 DATA_OUT=4

```

Daraus ergibt sich folgende Address-Map der Register A bis G:

| Register | Adresse (hexadezimal) |
|----------|-----------------------|
| A        | 00-0F, 80-8F          |
| B        | 10-1F, 90-9F          |
| C        | A0-BF                 |
| D        | 40-7F, C0-FF          |
| E        | 2A                    |
| F        | 20-27, 2C-37, 3C-3F   |
| G        | 28-29, 2B, 38-3B      |