Einführung in

Computer Microsystems

Sommersemester 2010

Wolfgang Heenes

TECHNISCHE UNIVERSITÄT DARMSTADT

9. Aufgabenblatt mit Lösungsvorschlag

16.06.2010

Aufgabe 1: Schnelles Potenzieren

Entwerfen Sie analog zur in der Vorlesung gezeigten Vorgehensweise zum systematischen Entwurf eine Schaltung für ein Verfahren zur schnellen Potenzierung ganzer Zahlen mit ganzzahligen Exponenten nach folgendem Pseudo-Code:

- *Eingaben Basis* `base`*, Exponent* `exp` *vorzeichenlos zu je 8 Bit*
- *Ausgabe Potenz* `pow` *ist 512 Bit breit*
- *Signal* `start=1` *startet Rechnung*
- *Signal* `done=1` *zeigt Abschluss der Rechnung an*

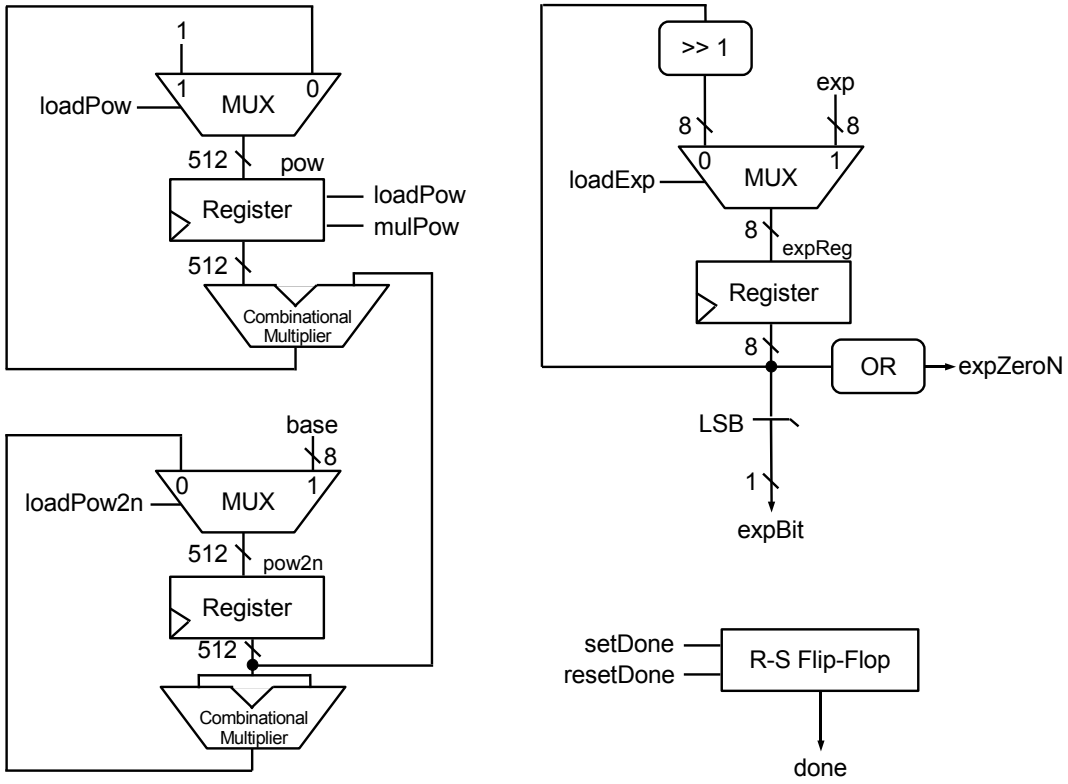
```
pow2n[511:0] := base[7:0];
pow[511:0] := 1;
done := 0;

WHILE (exp != 0) DO BEGIN
  if (exp[0] == 1)
    pow := pow * pow2n;
  pow2n := pow2n * pow2n;
  exp := exp >> 1;
END

done := 1;
```

Implementieren Sie Steuerwerk und Datenpfad als getrennte Verilog-Module. Finden Sie dazu geeignete Steuer- und Statussignale. Testen Sie die Funktion der Schaltung mit einer Testbench. Geben Sie den Zustandsgraphen des Steuerwerks und das Diagramm des Datenpfades an.

Lösungsvorschlag:

```
1: pow2n[511:0] := base[7:0];
   pow[511:0] := 1;
   done := 0;

2: IF (exp != 0) THEN BEGIN
   if (exp[0] == 1)
    pow := pow * pow2n;
    pow2n := pow2n * pow2n;
    exp := exp >> 1;
    GOTO 2;
   END

3: done := 1;
```

Ein möglicher Datenpfad:

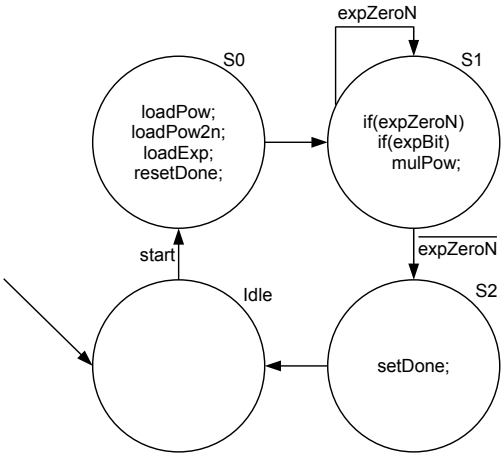


Verilog-Modul des Datenpfads:

```
‘timescale 1ns / 1ps
module fast_power_dp(
  input CLK, RESET, LOADPOW, MULPOW,
        LOADPOW2N, LOADEXP, RDONE, SDONE,
  input [7:0] EXP, BASE,
  output EXP_ZERO_N, EXP_BIT,
  output reg DONE,
  output reg [511:0] POW
);

  reg [7:0] EXP_REG; // Exponentenregister
  reg [511:0] POW2N; // (2**n)-te Potenzen von BASE

  // Statussignale fuer Steuerwerk
  // Exponentenregister ist nicht Null
  assign EXP_ZERO_N = |EXP_REG;
  // Aktuelles Exponentenbit
  assign EXP_BIT = EXP_REG[0];

  always @(posedge CLK or posedge RESET) begin
    if (RESET) begin
      DONE <= 0;
      POW <= 0;
      EXP_REG <= 0;
      POW2N <= 0;
    end
    else begin
      // DONE behandeln
      case ({SDONE, RDONE})
        2’b10: DONE <= 1;
        2’b01: DONE <= 0;
      endcase
      // EXP_REG behandeln
      if (LOADEXP)
```

```
        EXP_REG <= EXP;
      else // Exponent 1 Bit nach rechts schieben
        EXP_REG <= {1'b0, EXP_REG[7:1]};
      // (2**n)-te Potenzen der Basis berechnen
      if (LOADPOW2N)
        POW2N <= BASE;
      else
        POW2N <= POW2N * POW2N;
      // Ergebnisakkumulator POW behandeln
      if (LOADPOW)
        POW <= 1;
      else
        if (MULPOW)
          POW <= POW * POW2N;
    end
  end
endmodule
```

Zustandsgraph des Steuerwerks:




```
`timescale 1ns / 1ps
module fast_power_fsm(
  input CLK, RESET, START, EXP_ZERO_N, EXP_BIT,
  output reg LOADPOW, MULPOW, LOADPOW2N,
             LOADEXP, RDONE, SDONE
);

  parameter IDLE = 0,
            S0   = 1,
            S1   = 2,
            S2   = 3;

  reg [1:0] STATE, NEXTSTATE;

  // Kombinatorisch neuen Zustand und Ausgangsvektor berechnen
  // MULPOW haengt direkt von EXP_BIT ab => Mealy-Automat
  always @(STATE, START, EXP_ZERO_N, EXP_BIT) begin
    // Default Werte, Latches vermeiden
    // alles deaktivieren
    LOADPOW = 0; MULPOW = 0; LOADPOW2N = 0;
    LOADEXP = 0; RDONE = 0; SDONE = 0;
    NEXTSTATE = IDLE;
    case (STATE)
      IDLE: if (START)            // Auf Startsignal warten
              NEXTSTATE = S0;
      S0:   begin                 // Datenpfad initialisieren
              LOADPOW = 1; LOADPOW2N = 1;
              LOADEXP = 1; RDONE = 1;
              NEXTSTATE = S1;
            end
```

```
        S1:   if (EXP_ZERO_N) begin // Schleife EXP != 0
                if (EXP_BIT)
                  MULPOW = 1;
                NEXTSTATE = S1;
              end
              else
                NEXTSTATE = S2;
        S2:   begin                 // Ende der Berechnung anzeigen
                SDONE = 1;
                NEXTSTATE = IDLE;
              end
      endcase
    end

    // Neuen Zustand uebernehmen
    always @(posedge CLK, posedge RESET) begin
      if (RESET)  STATE <= IDLE;
      else        STATE <= NEXTSTATE;
    end

  endmodule
```

Haupt- oder Toplevel-Modul des Potenzrechners:

```
  `timescale 1ns / 1ps
  module fast_power(
    input CLK, RESET, START,
    input [7:0] BASE, EXP,
    output DONE,
    output [511:0] POW
  );

    wire EXP_ZERO_N, EXP_BIT, LOADPOW, MULPOW;
    wire LOADPOW2N, LOADEXP, RDONE, SDONE;

    // Instanziere Steuerwerk
    fast_power_fsm FAST_POWER_FSM (
      .CLK(CLK),
      .RESET(RESET),
      .START(START),
      .EXP_ZERO_N(EXP_ZERO_N),
      .EXP_BIT(EXP_BIT),
      .LOADPOW(LOADPOW),
      .MULPOW(MULPOW),
      .LOADPOW2N(LOADPOW2N),
      .LOADEXP(LOADEXP),
      .RDONE(RDONE),
      .SDONE(SDONE)
    );

    // Instanziere Datenpfad
    fast_power_dp FAST_POWER_DP (
      .CLK(CLK),
      .RESET(RESET),
      .LOADPOW(LOADPOW),
      .MULPOW(MULPOW),
      .LOADPOW2N(LOADPOW2N),
      .LOADEXP(LOADEXP),
      .RDONE(RDONE),
      .SDONE(SDONE),
      .EXP(EXP),
      .BASE(BASE),
      .EXP_ZERO_N(EXP_ZERO_N),
      .EXP_BIT(EXP_BIT),
      .DONE(DONE),
      .POW(POW)
    );
  endmodule
```

Testbench:

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

  // Inputs
  reg CLK;
  reg RESET;
  reg START;
  reg [7:0] BASE;
  reg [7:0] EXP;

  // Outputs
  wire DONE;
  wire [511:0] POW;

  // Instantiate the Unit Under Test (UUT)
  fast_power uut (
    .CLK(CLK),
    .RESET(RESET),
    .START(START),
    .BASE(BASE),
    .EXP(EXP),
    .DONE(DONE),
    .POW(POW)
  );

  // Erzeuge einen Taktzyklus
  task DoClockCycle;
    begin
      #5;
      CLK = 1;
      #5;
      CLK = 0;
    end
  endtask

  // Testdaten-Ausgabe
  initial
    $monitor("Zeit: %t   START=%b DONE=%b BASE=%d EXP=%d POW=%0d",
             $time, START, DONE, BASE, EXP, POW);

  initial begin
    // Reset der Schaltung
    RESET = 1;
    DoClockCycle;
    RESET = 0;
    DoClockCycle;
    // Berechne 2 hoch 0
    BASE = 2; EXP = 0;
    START = 1;
    DoClockCycle;
    START = 0;
    DoClockCycle;
    while (!DONE)
      DoClockCycle;
    // Berechne 2 hoch 31
    EXP = 31;
    START = 1;
    DoClockCycle;
    START = 0;
    DoClockCycle;
    while (!DONE)
      DoClockCycle;
    // Berechne 42 hoch 42
    BASE = 42; EXP = 42;
    START = 1;
    DoClockCycle;
    START = 0;
    DoClockCycle;
    while (!DONE)
      DoClockCycle;
```

```
    end

  endmodule
```

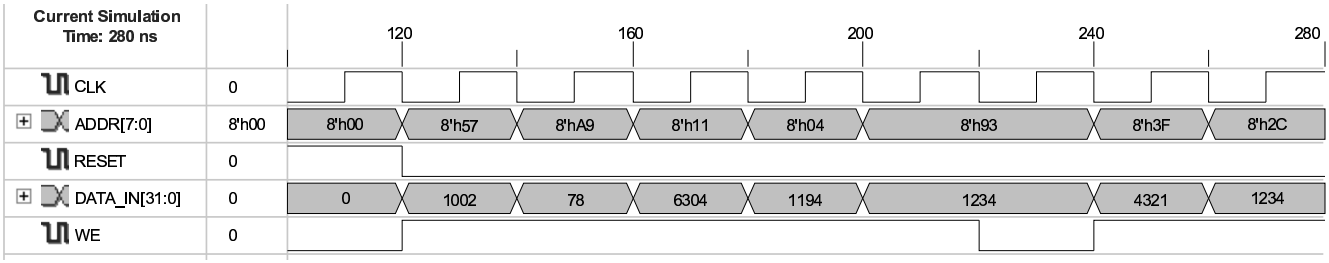
Simulationsausgabe:

```
Zeit:       0   START=x DONE=0 BASE=  x EXP=  x POW=0
Zeit:   20000   START=1 DONE=0 BASE=  2 EXP=  0 POW=0
Zeit:   30000   START=0 DONE=0 BASE=  2 EXP=  0 POW=0
Zeit:   35000   START=0 DONE=0 BASE=  2 EXP=  0 POW=1
Zeit:   55000   START=0 DONE=1 BASE=  2 EXP=  0 POW=1
Zeit:   60000   START=1 DONE=1 BASE=  2 EXP= 31 POW=1
Zeit:   70000   START=0 DONE=1 BASE=  2 EXP= 31 POW=1
Zeit:   75000   START=0 DONE=0 BASE=  2 EXP= 31 POW=1
Zeit:   85000   START=0 DONE=0 BASE=  2 EXP= 31 POW=2
Zeit:   95000   START=0 DONE=0 BASE=  2 EXP= 31 POW=8
Zeit:  105000   START=0 DONE=0 BASE=  2 EXP= 31 POW=128
Zeit:  115000   START=0 DONE=0 BASE=  2 EXP= 31 POW=32768
Zeit:  125000   START=0 DONE=0 BASE=  2 EXP= 31 POW=2147483648
Zeit:  145000   START=0 DONE=1 BASE=  2 EXP= 31 POW=2147483648
Zeit:  150000   START=1 DONE=1 BASE= 42 EXP= 42 POW=2147483648
Zeit:  160000   START=0 DONE=1 BASE= 42 EXP= 42 POW=2147483648
Zeit:  165000   START=0 DONE=0 BASE= 42 EXP= 42 POW=1
Zeit:  185000   START=0 DONE=0 BASE= 42 EXP= 42 POW=1764
Zeit:  205000   START=0 DONE=0 BASE= 42 EXP= 42 POW=17080198121677824
Zeit:  225000   START=0 DONE=0 BASE= 42 EXP= 42
 POW=150130937545296572356753417944249280335594212851135933799120557309952
Zeit:  245000   START=0 DONE=1 BASE= 42 EXP= 42
 POW=150130937545296572356753417944249280335594212851135933799120557309952
```

Aufgabe 2: Adressen und Inhalte

Gegeben sind eine Waveform und ein Verilog HDL Modul.

Waveform:



Verilog-Modul:

```
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
```

Die in der Waveform dargestellten Signale werden als Stimulus in das Verilog HDL Modul eingegeben. Welche Werte enthalten danach die Register A bis G und auf welchen Adressen können sie jeweils ausgelesen werden?

Lösungsvorschlag:

Die nachstehende Testbench ermittelt die Registeradressen durch Auslesen der Reset-Werte indem der gesamte Adressraum überstrichen wird.

```
module tb_memory_mapped_adr;
    // Inputs
    reg CLK;  reg RESET;  reg [7:0] ADDR;  reg [31:0] DATA_IN;  reg WE;
    // Outputs
    wire [31:0] DATA_OUT;

    // 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

    // Bei Änderung von DATA_OUT alle Signale ausgeben
    always @(DATA_OUT)
        $display("ADDR=%h␣DATA_OUT=%1d", ADDR, DATA_OUT);

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

        // Reset
        RESET = 1; DoClockCycle; RESET = 0;

        // Überstreiche ganzen Adressraum und finde Registerleseadressen
        for (i = 0; i < 2**8; i = i + 1) begin
            ADDR = i; #1;
        end
    end
endmodule
```

Simulationsausgabe des Adresstests (DATA_OUT=1 ↦ Register A bis DATA_OUT=7 ↦ 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

In folgender Testbench werden die Signaländerungen der Waveform als Teststimuli formuliert und die Registerwerte ausgegeben.

```
module tb_memory_mapped_wave;
    // Inputs
    reg CLK;  reg RESET;  reg [7:0] ADDR;  reg [31:0] DATA_IN;  reg WE;
    // Outputs
    wire [31:0] DATA_OUT;

    // 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;
        ...
    endtask

    initial begin
        // Initialize Inputs
        CLK = 0; RESET = 0; ADDR = 0; DATA_IN = 0; WE = 0;
        // Reset
        RESET = 1; DoClockCycle; RESET = 0;

        // Erzeuge Signale der Waveform
        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);
    end
endmodule
```

Die Register haben nach den Signaländerungen folgende Werte:

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