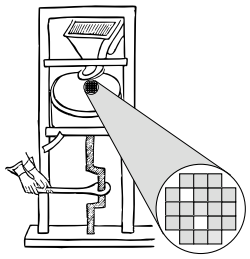
Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch
Holger Lange
Mathias Halbach (FG Rechnerarchitektur)



22.06.2006

Technische Grundlagen der Informatik II
7. Übung – MIPS ALU
Sommersemester 2006

Aufgabe 1: 1-Bit-ALU

a) Implementieren Sie die in Vorlesungskapitel 6, Folie 53 dargestellte 1-Bit-ALU in Verilog.

Lösung:

```
module alu(a, b, carryin, operation, carryout, result);
  input       a, b, carryin;
  input [1:0] operation;
  output      carryout, result;

  wire ab_and, ab_or, ab_sum, carryout, result;

  assign ab_and             = a && b;
  assign ab_or              = a || b;
  assign {carryout, ab_sum} = a + b + carryin;

  assign result = (operation == 0) ? ab_and
                                   : (operation == 1) ? ab_or
                                                      : ab_sum;

endmodule
```

b) Erweitern Sie die ALU aus a) um die **`Binvert`**-Funktion (Folie 54).

Lösung:

```
module alu(a, b, carryin, operation, binvert, carryout, result);
  input       a, b, carryin, binvert;
  input [1:0] operation;
  output      carryout, result;
```

```
  wire b_in, ab_and, ab_or, ab_sum, carryout, result;

  assign b_in               = binvert ? ~b : b;
  assign ab_and             = a && b_in;
  assign ab_or              = a || b_in;
  assign {carryout, ab_sum} = a + b_in + carryin;

  assign result = (operation == 0) ? ab_and
                                   : (operation == 1) ? ab_or
                                                      : ab_sum;
endmodule
```

Aufgabe 2: MIPS-ALU

Implementieren Sie die 32-Bit MIPS-ALU mit den Operationen **NOP**, **ADD**, **SUB**, **OR**, **AND** und **SLT** in Verilog. Die ALU soll nur einen Ausgang **alu_out** verwenden, über den die Ergebnisse aller Operationen ausgegeben werden. Die Eingänge der ALU sind **alu_a** (1. Operand), **alu_b** (2. Operand) und **alu_op** (Selektion der Operation). Verwenden Sie für die einzelnen ALU-Operationen die entsprechenden Verilog-Operatoren.

Lösung:

```
module alu(alu_out, alu_a, alu_b, alu_op);
   output [31:0] alu_out;
   input  [31:0] alu_a, alu_b;
   input  [2:0]  alu_op;

   reg    [31:0] alu_out;

   // Kontrollsignalkodierungen
   `define OP_NOP 0
   `define OP_ADD 1
   `define OP_SUB 2
   `define OP_OR  3
   `define OP_AND 4
   `define OP_SLT 5

   always @(alu_a or alu_b or alu_op)
     case(alu_op)
       `OP_ADD  : alu_out = alu_a + alu_b;
       `OP_AND  : alu_out = alu_a & alu_b;
       `OP_OR   : alu_out = alu_a | alu_b;
       `OP_SUB  : alu_out = alu_a - alu_b;
       `OP_SLT  : alu_out = {alu_a[31], alu_a} - {alu_b[31], alu_b} >> 32;
       `OP_NOP  : alu_out = 0;
       default :
         begin
           alu_out = 32'bx;
           $display("Unbekannte ALU Operation, %d", alu_op);
         end
     endcase
endmodule
```