



18.05.2006

# Technische Grundlagen der Informatik II

## 3. Übung – Rechenwerke

### Sommersemester 2006

#### Aufgabe 1: Subtraktion von positiven Dualzahlen

Subtrahieren Sie die folgenden Dualzahlen unter Berücksichtigung der Borge-Bits (nach der ersten Methode in der Vorlesung, Vollsubtrahierer-Prinzip). Ergibt sich ein nicht mehr darstellbares negatives Ergebnis? Geben Sie dazu auch die entsprechenden dezimalen Werte an.

**a)**  $1100\ 0001 - 0011\ 0100$

**Lösung:**

$$\begin{array}{r} 1100\ 0001 \\ - 0011\ 0100 \\ \hline \underline{111\ 1} \quad \text{Borge-Bits} \\ = 1000\ 1101 \quad (\text{darstellbar}) \end{array}$$

$$193 - 52 = 141$$

**b)**  $1\ 0010 - 1\ 1000$

**Lösung:**

$$\begin{array}{r} 1\ 0010 \\ - 1\ 1000 \\ \hline \underline{(1)1} \quad \text{Borge-Bits} \\ = (1)1\ 1010 \quad (\text{nicht mehr darstellbar}) \end{array}$$

Unter Berücksichtigung des Borrow-Outs ergibt sich mit dem negativen Stellengewicht (hier -32) der richtige Wert. Das Ergebnis kann auch als 2K-Zahl interpretiert werden, wenn die Borrow-Out-Stelle zusätzlich hinzugenommen wird.

$$18 - 24 = 26 - (32) = -6$$

Das Ergebnis ist negativ geworden, es ist nicht mehr als positive Dualzahl darstellbar, also gab es einen Überlauf. Der Überlauf kann zum Vergleich benutzt werden: Die zweite Zahl war größer als die erste.

## Aufgabe 2: Addierer in Verilog

a) Beschreiben und simulieren Sie einen Halbaddierer in Verilog. Entwerfen Sie zusätzlich eine Testumgebung, so dass alle Wertkombinationen ausgegeben werden.

**Lösung:**

```
module HalfAdder(A, B, Sum, Carry);
  input A, B;
  output Sum, Carry;

  assign Sum = A ^ B;
  assign Carry = A & B;
endmodule
```

```
module TestBench;
  reg A, B;
  wire Sum, Carry;

  HalfAdder ha(A, B, Sum, Carry);

  initial begin
    $monitor("A: %d, B: %d, Sum: %d, Carry: %d\n", A, B, Sum, Carry);
    A <= 0; B <= 0;
    #1;
    A <= 1; B <= 0;
    #1;
    A <= 1; B <= 1;
    #1;
  end
endmodule
```

b) Beschreiben und simulieren Sie in Verilog einen Volladdierer unter Benutzung des Moduls aus Teilaufgabe a) als Untereinheit. Passen sie dazu die Testumgebung aus Teilaufgabe a) an.

**Lösung:**

```
module FullAdder(A, B, CarryIn, Sum, CarryOut);
  input A, B, CarryIn; output Sum, CarryOut;

  wire sum1, carry1, carry2;

  HalfAdder ha1(A, B, sum1, carry1);
  HalfAdder ha2(CarryIn, sum1, Sum, carry2);

  assign CarryOut = carry1 | carry2;
endmodule

module TestBench;
```

```

reg A, B, Cin;
wire Sum, Carry;

FullAdder fa(A, B, Cin, Sum, Carry);

initial begin
    $monitor("A: %d, B: %d, Cin: %d, Sum: %d, Carry: %d\n",
            A, B, Cin, Sum, Carry);
    A <= 0; B <= 0; Cin <= 0;
    #1;
    A <= 1; B <= 0; Cin <= 0;
    #1;
    A <= 0; B <= 0; Cin <= 1;
    #1;
    A <= 0; B <= 1; Cin <= 1;
    #1;
    A <= 1; B <= 1; Cin <= 0;
    #1;
    A <= 1; B <= 1; Cin <= 1;
    #1;
end
endmodule

```

c) Implementieren und simulieren Sie einen 1-Bit-Addierer mit Übertragserzeugung (carry generate, G) und Übertragsweiterleitung (carry propagate, P) aus Gatterprimitiven. Passen sie dazu die Ihre Testumgebung an.

### Lösung:

```

module adderGP(A, B, CarryIn, Sum, G, P);
    input A, B, CarryIn;
    output Sum, G, P;

    xor (Sum, A, B, CarryIn);
    or (P, A, B);
    and (G, A, B);
endmodule

module TestBench;
    reg A, B, Cin;
    wire Sum, G, P;

    adderGP agp(A, B, Cin, Sum, G, P);

    initial begin
        $monitor("A: %d, B: %d, Cin: %d, Sum: %d, G: %d, P: %d\n",
                A, B, Cin, Sum, G, P);
        A <= 0; B <= 0; Cin <= 0;
        #1;
        A <= 1; B <= 0; Cin <= 0;
        #1;
        A <= 0; B <= 0; Cin <= 1;
        #1;
        A <= 0; B <= 1; Cin <= 1;
        #1;
    end
endmodule

```

```

    A <= 1; B <= 1; Cin <= 0;
    #1;
    A <= 1; B <= 1; Cin <= 1;
    #1;
end
endmodule

```

**d)** Implementieren Sie den Carry-Generator für einen 4-Bit Addierer mit Carry Look Ahead (CLA). Benutzen Sie dazu Kapitel 3 (Mikroalgorithmen und Rechenwerke), Folie 14 der Vorlesung als Vorlage.

**Lösung:**

```

module cg(G, P, Cin, C, Gout, Pout);
    input  [4:1] G, P;
    input          Cin;
    output [4:2] C;
    output          Gout, Pout;

    assign Gout = G[4] | (P[4] & G[3]) | (&P[4:2] & G[1]);
    assign Pout = &P[4:1];
    assign C[2] = G[1] | (P[1] & Cin);
    assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & Cin);
    assign C[4] = G[3] | (P[3] & G[2]) | (&P[3:2] & G[1]) | (&P[3:1] & Cin);
endmodule

```

**e)** Implementieren und simulieren Sie einen 4-Bit-Addierer mit Übertragsvorausberechnung und Overflow. Verwenden Sie dabei die Module aus den Teilaufgaben c) und d). Ihre Testumgebung soll charakteristische Testfälle berücksichtigen.

**Lösung:**

```

module cla4(A, B, Cin, S, Cout, Gout, Pout, Ov);
    input  [4:1] A, B;
    input          Cin;
    output [4:1] S;
    output          Cout, Gout, Pout, Ov;

    wire [4:1] G, P;
    wire [4:2] C;

    adderGP agp1(A[1], B[1], Cin, S[1], G[1], P[1]),
            agp2(A[2], B[2], C[2], S[2], G[2], P[2]),
            agp3(A[3], B[3], C[3], S[3], G[3], P[3]),
            agp4(A[4], B[4], C[4], S[4], G[4], P[4]);

    cg cg1(G, P, Cin, C, Gout, Pout);

    assign Cout = G[4] | (P[4] & C[4]);
    assign Ov = Cout ^ C[4];
endmodule

module TestBench;
    reg  [4:1] A, B;

```

```

reg      Cin;
wire [4:1] Sum;
wire     Carry, G, P;

cla4 cla(A, B, Cin, Sum, Carry, G, P, Ov);

initial begin
    $monitor("A: %b, B: %b, Cin: %b, Sum: %b, Carry: %b, G: %b, P: %b, Ov: %b\n",
             A, B, Cin, Sum, Carry, G, P, Ov);
    A <= 0; B <= 0; Cin <= 0;
    #1;
    A <= 8; B <= 0; Cin <= 0;
    #1;
    A <= 0; B <= 0; Cin <= 1;
    #1;
    A <= 0; B <= 7; Cin <= 1;
    #1;
    A <= 5; B <= 10; Cin <= 0;
    #1;
    A <= 5; B <= 10; Cin <= 1;
    #1;
    A <= 8; B <= 8; Cin <= 0;
    #1;
    A <= 15; B <= 15; Cin <= 0;
    #1;
    A <= 15; B <= 15; Cin <= 1;
    #1;
end
endmodule

```

### Aufgabe 3: Multiplikation von 2K-Dualzahlen

Berechnen Sie gemäß folgendem Beispiel die Multiplikation wie in der Vorlesung (Kapitel 3, Mikroalgorithmen und Rechenwerke, Folie 44) behandelt.

$$\underbrace{1010}_{-6_{10}} * \underbrace{1101}_{-3_{10}} = \underbrace{00010010}_{18_{10}}$$

$$\begin{array}{r} \underline{1010 * 1101} \\ 1010 \\ 0000 \\ 1010 \\ \underline{1010} \\ 10000010 \\ 0110 \quad \text{Komplement 1. Faktor} \\ \underline{0011} \quad \text{Komplement 2. Faktor} \\ \underline{100010010} \end{array}$$

**a)**  $1101 * 1001$

**Lösung:**

$$\underbrace{1101}_{-3_{10}} * \underbrace{1001}_{-7_{10}} = \underbrace{00010101}_{21_{10}}$$

$$\begin{array}{r} \underline{1101 * 1001} \\ 1101 \\ 0000 \\ 0000 \\ \underline{1101} \\ 1110101 \\ 0011 \quad \text{Komplement 1. Faktor} \\ \underline{0111} \quad \text{Komplement 2. Faktor} \\ \underline{100010101} \end{array}$$

**b)**  $0101 * 1100$

**Lösung:**

$$\underbrace{0101}_{5_{10}} * \underbrace{1100}_{-4_{10}} = \underbrace{11101100}_{-20_{10}}$$

$$\begin{array}{r} \underline{0101 * 1100} \\ 0000 \\ 0000 \\ 0101 \\ \underline{0101} \\ 0111100 \\ \underline{1011} \quad \text{Komplement 1. Faktor} \\ \underline{011101100} \end{array}$$