```
% Addierer 1 Bit mit Logikgleichungen %

SUBDESIGN adder1b
(
        cin,opa,opb     : INPUT;
        sum,cout        : OUTPUT;
)

BEGIN
        sum=cin$opa$opb;
        Cout=opa &opb # opa & cin # opb & cin; -- Kommentar
END;
```

```
TITLE "16 Bit Addierer";

SUBDESIGN adder16bit
(
        carryin                 : INPUT;
        opa[15..0],opb[15..0]   : INPUT;
        ergeb[15..0]            : OUTPUT;
        carryout                : OUTPUT;
)
BEGIN
                (carryout,ergeb[])=(0,opa[])+(0,opb[])+(0,carryin);
END;
```

```
TITLE " Addierer mit einer Wortbreite von n Bit";

PARAMETERS
(
        dim = 8
);

SUBDESIGN adder_n_bit
(
        carryin                         : INPUT;
        opa[dim-1..0],opb[dim-1..0]     : INPUT;
        ergeb[dim-1..0]                 : OUTPUT;
        carryout                        : OUTPUT;
)
BEGIN
                (carryout,ergeb[])=(0,opa[])+(0,opb[])+(0,carryin);
END;
```

```
TITLE " Arithmetisch- Logische Einheit";

SUBDESIGN alu_funk
(
        opcod[3..0]             : INPUT;
        carryin                 : INPUT;
        opa[7..0],opb[7..0]     : INPUT;
        ergeb[7..0]             : OUTPUT;
        carryout                : OUTPUT;
)
BEGIN
CASE opcod[] IS
```

```
        WHEN 0 =>
                (carryout,ergeb[])=(0,opa[])+(0,opb[]); % ADD %
        WHEN 1 =>
                (carryout,ergeb[])=(0,opa[])+(0,opb[])+(0,carryin); % ADC %

        WHEN 2 =>
                (carryout,ergeb[])=(0,opa[])-(0,opb[])-(0,carryin); % SUBB %
        WHEN 3 =>
                (carryout,ergeb[])=(0,opa[])+1; % INC %
        WHEN 4 =>
                (carryout,ergeb[])=(0,opa[])-1; % DEC %
        WHEN 5 =>
                ergeb[]=opa[]&opb[]; % UND %
        WHEN 6 =>
                ergeb[]=opa[]#opb[]; % ODER %
        WHEN 7 =>
                ergeb[]=opa[]$opb[]; % XOR %
        WHEN 8 =>
                ergeb[]=0; % CLEAR %
        WHEN 9 =>
                ergeb[]=!opa[]; % COMPLEMENT %
        WHEN 10 =>
                ergeb[]=opb[]; % Laden %

        WHEN OTHERS =>
                (carryout,ergeb[])=0;
END CASE;
END;
```

```
TITLE "Carry look ahead adder";

SUBDESIGN adderLookAh
(
        opa[3..0],opb[3..0],cin : INPUT;
        summe[4..0]             : OUTPUT;
)
VARIABLE
        ci[4..0]                : NODE;
        ai[3..0]                : NODE;
        gi[3..0]                : NODE;
        pi[3..0]                : NODE;
BEGIN
FOR i IN 0 to 3 GENERATE
        ai[i]=opa[i] # opb[i];
        gi[i]=opa[i] & opb[i];
        pi[i]=opa[i] $ opb[i];
        summe[i]=pi[i]$ci[i];
END GENERATE;
ci[0]=cin;
ci[1]=gi[0]#ai[0]&ci[0];
ci[2]=gi[1]#ai[1]&gi[0]#ai[1]&ai[0]&ci[0];
ci[3]=gi[2]#ai[2]&gi[1]#ai[2]&ai[1]&gi[0]#ai[2]&ai[1]&ai[0]&ci[0];
summe[4]=gi[3]#ai[3]&gi[2]#ai[3]&ai[2]&ai[1]&gi[0]#ai[3]&ai[3]&ai[1]&ai[0]&ci[0]
;
END;
```

```
%************************
Paralelmultiplizierer 4x4
mit if generate
************************%
SUBDESIGN multi4_mit_if
```

```
                                Beispiele_Arithmetik
(
        faka[3..0],fakb[3..0]           : INPUT ;
        prod[7..0]                      : OUTPUT;
)
VARIABLE
        zws[2..0][4..0]                 : NODE;
BEGIN
        prod[0]=faka[0]&fakb[0];
        zws[0][4..0]=(0,(faka[3..1]&fakb[0]))+(0,(faka[3..0]&fakb[1]));
        prod[1]=zws[0][0];
        FOR i  IN 1 TO 2 GENERATE
                zws[i][4..0]=(0,zws[i-1][4..1])+(0,(faka[3..0]&fakb[i+1]));
        prod[i+1]=zws[i][0];
        END GENERATE;
        prod[7..4]=zws[2][4..1];

END;
```

```
TITLE "Multiplizierer nxn Bit 2*n Bit Ergebnis";

PARAMETERS
(
        wortl = 8
);


SUBDESIGN multi_nxn_bit
(
        opa[wortl-1..0],opb[wortl-1..0]         : INPUT;
        prod[(2*wortl)-1..0]                    : OUTPUT;
)
VARIABLE
        zws[wortl-2..0][wortl..0]               : NODE;

BEGIN
        FOR i IN 0 to (wortl-2) GENERATE
                IF i==0 GENERATE

zws[i][]=(gnd,gnd,(opa[7..1]&opb[i]))+(GND,(opa[]&opb[i+1]));
                        prod[i]=opa[i]&opb[i];
                        prod[i+1]=zws[i][i];
                END GENERATE;
                IF (i>0) & (i<wortl-2) GENERATE
                        zws[i][]=(gnd,zws[i-1][8..1])+(gnd,(opa[]&opb[i+1]));
                        prod[i+1]=zws[i][0];
                END GENERATE;
                IF i==wortl-2 GENERATE
                        zws[i][]=(gnd,zws[i-1][8..1])+(gnd,(opa[]&opb[i+1]));
                        prod[(2*wortl)-1..wortl-1]=zws[i][];
                END GENERATE;
        END GENERATE;
END;
```

```
TITLE " Division 4Bit : 4Bit durch Vergleich und Subtraktion"
SUBDESIGN divis4
(
        zae[3..0],nen[3..0]     : INPUT;
        quo[3..0],rest[3..0]    : OUTPUT;
        zwz0[6..0]              : output;
        zwz1[5..0]              : output; -- zusätzliche Signale für die
Simulation
        zwz2[4..0]              : output;
```

```
        zwz3[3..0]              : output;

)
VARIABLE
        zwz[3..0][6..0]         : NODE;

BEGIN
        zwz0[6..0]=zwz[0][6..0];
        zwz1[5..0]=zwz[1][5..0];
        zwz2[4..0]=zwz[2][4..0];
        zwz3[3..0]=zwz[3][3..0];

        IF (nen[],B"000")> (0,zae[])THEN
                quo[3]=GND;
                (zwz[0][6..0])=(0,zae[3..0]);
        ELSE
                quo[3]=VCC;
                (zwz[0][6..0])=(0,zae[3..0])-(nen[],B"000");
        END IF;
        IF (nen[],B"00")> (zwz[0][5..0])THEN
                quo[2]=GND;
                zwz[1][5..0]=(zwz[0][5..0]);
        ELSE
                quo[2]=VCC;
                (zwz[1][5..0])=(zwz[0][5..0])-(nen[],B"00");
        END IF;
        IF (nen[],B"0")> (zwz[1][4..0])THEN
                quo[1]=GND;
                (zwz[2][5..0])=(zwz[1][5..0]);
        ELSE
                quo[1]=VCC;
                (zwz[2][4..0])=(zwz[1][4..0])-(nen[],B"0");
        END IF;
        IF nen[]> (zwz[2][3..0])THEN
                quo[0]=GND;
                zwz[3][3..0]=zwz[2][3..0];
        ELSE
                quo[0]=VCC;
                zwz[3][3..0]=zwz[2][3..0]-(nen[]);
        END IF;
        rest[3..0]=zwz[3][3..0];
END;
```

```
TITLE "Division nBit : nBit durch Vergleich und Subtraktion"
PARAMETERS
(
        zlen = 8
);

SUBDESIGN divis_n_par
(
        zae[zlen-1..0],nen[zlen-1..0]           : INPUT;
        quo[zlen-1..0],rest[zlen-1..0]          : OUTPUT;
)
VARIABLE
        zwz[zlen-1..0][2*zlen-2..0]             : NODE;
        zwvn[zlen-2..0]                         : NODE;

BEGIN
        zwvn[]=0;
        IF (nen[],zwvn[])> (0,zae[])THEN
                quo[zlen-1]=GND;
                (zwz[0][2*zlen-2..0])=(0,zae[zlen-1..0]);
        ELSE
```

```
                quo[zlen-1]=VCC;
                (zwz[0][2*zlen-2..0])=(0,zae[zlen-1..0])-(nen[],zwvn[]);
        END IF;
FOR i IN 1 TO (zlen-2) GENERATE
        IF (nen[],zwvn[zlen-2-i..0])> (zwz[i-1][2*zlen-2-i..0])THEN
                quo[zlen-1-i]=GND;
                zwz[i][2*zlen-2-i..0]=(zwz[i-1][2*zlen-2-i..0]);
        ELSE
                quo[zlen-1-i]=VCC;

(zwz[i][2*zlen-2-i..0])=(zwz[i-1][2*zlen-2-i..0])-(nen[],zwvn[zlen-2-i..0]);
        END IF;
END GENERATE;
        IF nen[zlen-1..0]> (zwz[zlen-2][zlen-1..0])THEN
                quo[0]=GND;
                zwz[zlen-1][zlen-1..0]=zwz[zlen-2][zlen-1..0];
        ELSE
                quo[0]=VCC;
                zwz[zlen-1][zlen-1..0]=zwz[zlen-2][zlen-1..0]-(nen[]);
        END IF;
        rest[zlen-1..0]=zwz[zlen-1][zlen-1..0];
END;
```
