Figure 4-1 Serial Adder with Accumulator

<table>
<thead>
<tr>
<th>t</th>
<th>X</th>
<th>Y</th>
<th>c_i</th>
<th>sum_i</th>
<th>c_{i+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0101</td>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0010</td>
<td>1011</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0001</td>
<td>1101</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>1110</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1100</td>
<td>0111</td>
<td>0</td>
<td>(1)</td>
<td>(0)</td>
</tr>
</tbody>
</table>
Figure 4-2  Control State Graph and Table for Serial Adder

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State N=0</th>
<th>Next State N=1</th>
<th>Present Output (Sh) N=0</th>
<th>Present Output (Sh) N=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>S1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>S2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td>S3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>S0</td>
<td>S0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Constraints on Input Labels for Every State $S_k$ *(From Page 123-124)*

1. If $I_i$ and $I_j$ are any pair of input labels on arcs exiting state $S_k$, then $I_iI_j = 0$ if $i \neq j$.

2. If $n$ arcs exit state $S_k$ and the $n$ arcs have input labels $I_1, I_2, ..., I_n$, respectively, then $I_1 + I_2 + ... + I_n = 1$.

![Diagram of state transitions](image)

$(X_1)(X_1'X_2') = 0$
$(X_1)(X_1'X_2) = 0$
$(X_1'X_2')(X_1'X_2) = 0$
$X_1 + X_1'X_2' + X_1'X_2 = 1$

Inputs are $X_1 X_2 X_3$

$(X_1 = X_2 = 1$ not allowed)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>000 001 010 011 100 101 110 111</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_k$</td>
<td>$S_k$ $S_k$ $S_q$ $S_q$ $S_p$ $S_p$</td>
</tr>
</tbody>
</table>
Multiplication of $13_{10}$ by $11_{10}$ in Binary – From Page 124

![Diagram showing binary multiplication process]

Initial contents of product register: $0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1$ (M (11))

After addition: $1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1$

After shift: $0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1$

(Add multiplicand since $M=1$)

After addition: $1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1$

After shift: $0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0$ (M)

(Skip addition since $M=0$)

After shift: $0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$

(Add multiplicand since $M=1$)

After addition: $1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$

After shift (final answer): $0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$ (143)

Dividing line between product and multiplier.
Figure 4-3 Block Diagram for Binary Multiplier
Figure 4-4 State graph for Binary Multiplier Control
Figure 4-5(a) Behavioral Model for 4 x 4 Binary Multiplier

-- This is a behavioral model of a multiplier for unsigned binary numbers. It multiplies a
-- 4-bit multiplicand by a 4-bit multiplier to give an 8-bit product.
-- The maximum number of clock cycles needed for a multiply is 10.

library BITLIB;
use BITLIB.bit_pack.all;
entity mult4X4 is
  port (Clk, St: in bit;
       Mplier,Mcand : in bit_vector(3 downto 0);
       Done: out bit);
end mult4X4;
architecture behave1 of mult4X4 is
  signal State: integer range 0 to 9;
  signal ACC: bit_vector(8 downto 0); -- accumulator
  alias M: bit is ACC(0); -- M is bit 0 of ACC
begin
  process
  begin
    wait until Clk = '1'; -- executes on rising edge of clock
    case State is
      when 0=>
        if St='1' then
          ACC(8 downto 4) <= "00000"; -- Begin cycle
          ACC(3 downto 0) <= Mplier; -- load the multiplier
          State <= 1;
        end if;
    end case;
  end process;
end behave1;
Figure 4-5(b) Behavioral Model for 4 x 4 Binary Multiplier

```vhdl
when 1 | 3 | 5 | 7 => -- "add/shift" State
    if M = '1' then
        ACC(8 downto 4) <= add4(ACC(7 downto 4), Mcand, '0');
        State <= State + 1;
    else
        ACC <= '0' & ACC(8 downto 1); -- Shift accumulator right
        State <= State + 2;
    end if;
when 2 | 4 | 6 | 8 => -- "shift" State
    ACC <= '0' & ACC(8 downto 1); -- Right shift
    State <= State + 1;
when 9 => -- End of cycle
    State <= 0;
end case;
end process;
Done <= '1' when State = 9 else '0';
end behave1;
```
Figure 4-6 Multiplier Control with Counter

(a) Multiplier control

(b) State graph for add-shift control

(c) Final state graph for add-shift control
<table>
<thead>
<tr>
<th>Time</th>
<th>State</th>
<th>Counter</th>
<th>Product Register</th>
<th>St</th>
<th>M</th>
<th>K</th>
<th>Load</th>
<th>Ad</th>
<th>Sh</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>S0</td>
<td>00</td>
<td>0000000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t1</td>
<td>S0</td>
<td>00</td>
<td>0000000000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>S1</td>
<td>00</td>
<td>0000001011</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t3</td>
<td>S2</td>
<td>00</td>
<td>0110110111</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t4</td>
<td>S1</td>
<td>01</td>
<td>0011011011</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t5</td>
<td>S2</td>
<td>01</td>
<td>1001111011</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t6</td>
<td>S1</td>
<td>10</td>
<td>0100111110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t7</td>
<td>S1</td>
<td>11</td>
<td>0010011111</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t8</td>
<td>S2</td>
<td>11</td>
<td>1000111111</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t9</td>
<td>S3</td>
<td>00</td>
<td>0100011111</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 4-3  4-bit Multiplier Partial Products

<table>
<thead>
<tr>
<th></th>
<th>$X_3$</th>
<th>$X_2$</th>
<th>$X_1$</th>
<th>$X_0$</th>
<th>$Y_3$</th>
<th>$Y_2$</th>
<th>$Y_1$</th>
<th>$Y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Multiplicand</strong></td>
<td>$X_3Y_0$</td>
<td>$X_2Y_0$</td>
<td>$X_1Y_0$</td>
<td>$X_0Y_0$</td>
<td><strong>Multiplier</strong></td>
<td>$X_3Y_1$</td>
<td>$X_2Y_1$</td>
<td>$X_1Y_1$</td>
</tr>
<tr>
<td></td>
<td>$C_{12}$</td>
<td>$C_{11}$</td>
<td>$C_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1st row carries</strong></td>
<td>$C_{13}$</td>
<td>$S_{13}$</td>
<td>$S_{12}$</td>
<td>$S_{11}$</td>
<td>$S_{10}$</td>
<td>$C_{22}$</td>
<td>$C_{21}$</td>
<td>$C_{20}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1st row sums</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Partial product 0</strong></td>
<td>$X_3Y_2$</td>
<td>$X_2Y_2$</td>
<td>$X_1Y_2$</td>
<td>$X_0Y_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$C_{22}$</td>
<td>$C_{21}$</td>
<td>$C_{20}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2nd row carries</strong></td>
<td>$C_{23}$</td>
<td>$S_{23}$</td>
<td>$S_{22}$</td>
<td>$S_{21}$</td>
<td>$S_{20}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2nd row sums</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Partial product 2</strong></td>
<td>$X_3Y_3$</td>
<td>$X_2Y_3$</td>
<td>$X_1Y_3$</td>
<td>$X_0Y_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$C_{32}$</td>
<td>$C_{31}$</td>
<td>$C_{30}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>3rd row carries</strong></td>
<td>$C_{33}$</td>
<td>$S_{33}$</td>
<td>$S_{32}$</td>
<td>$S_{31}$</td>
<td>$S_{30}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>3rd row sums</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Partial product 3</strong></td>
<td>$P_7$</td>
<td>$P_6$</td>
<td>$P_5$</td>
<td>$P_4$</td>
<td>$P_3$</td>
<td>$P_2$</td>
<td>$P_1$</td>
<td>$P_0$</td>
</tr>
<tr>
<td><strong>Final product</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Figure 4-7** Block Diagram of 4 x 4 Array Multiplier
From Page 133

\[
\begin{array}{ccc}
0.111 & \text{(+7/8)} & \leftarrow \text{Multiplicand} \\
\times 0.101 & \text{(+5/8)} & \leftarrow \text{Multiplier} \\
\hline
(0.00)0111 & \text{(+7/64)} & \leftarrow \text{Note: The proper representation of the fractional partial} \\
(0.)0111 & \text{(+7/16)} & \text{products requires extension of the sign bit past the binary} \\
0.100011 & \text{(+35/64)} & \text{point, as indicated in parentheses. (Such extension is not} \\
& & \text{necessary in the hardware.)}
\end{array}
\]

\[
\begin{array}{ccc}
1.101 & \text{(-3/8)} \\
\times 0.101 & \text{(+5/8)} \\
\hline
(1.111)1101 & \text{(-3/64)} & \leftarrow \text{Note: The extension of the sign bit provides} \\
(1.)1101 & \text{(-3/16)} & \text{proper representation of the negative products.} \\
1.110001 & \text{(-15/64)}
\end{array}
\]
From Pages 133 – 134

\[
\begin{array}{cccccc}
0.101 & (+5/8) \\
\times & 1.101 & (-3/8) \\
\hline
(0.00)0101 & (+5/64) \\
(0.)0101 & (+5/16) \\
(0.)011001 \\
1.011 & (-5/8) \quad \text{Note: The two's complement of the multiplicand is added at this point.} \\
\hline
1.110001 & (-15/64)
\end{array}
\]

\[
\begin{array}{cccccc}
1.101 & (-3/8) \\
\times & 1.101 & (-3/8) \\
\hline
(1.111)1101 & (-3/64) \\
(1.)1101 & (-3/16) \\
(1.)110001 \\
0.011 & (+3/8) \quad \text{Add the two's complement of the multiplicand} \\
\hline
0.001001 & (+9/64)
\end{array}
\]
Figure 4-8  Block Diagram for 2's Complement Multiplier

The diagram shows a block diagram for a 2's complement multiplier. The diagram includes a 5-bit full adder, a 1's complementer, and various control signals. The inputs and outputs are labeled as follows:

- **ACC**: Accumulator
- **Cin**: Carry input
- **Cm**: Complement signal
- **Clk**: Clock signal
- **Load**, **Sh**, **Ad**: Load, Shift, Add signals
- **Done**, **St**: Done, Start signals
- **M**: Multiplicand
- **product**: Product output
Figure 4-9  State Graph for 2's Complement Multiplier
Figure 4-10  Block Diagram for Faster Multiplier

LOAD
Sh
AdSh

CONTROL
M

Done
St
Cm

A (accumulator)

3 2 1 0

4-BIT FULL ADDER Cin

1's COMPLEMENTS

Cm

multiplicand

B

3 2 1 0

multiplier

product
Figure 4-11  State Graph for Faster Multiplier

- S0
  - St'/0
  - St/Load
  - –/Done
- S1
  - M/AdSh
  - M'/Sh
- S2
  - M/AdSh
  - M'/Sh
- S3
  - M/AdSh
  - M'/Sh
- S4
  - M/AdSh
  - M'/Sh
- S5
  - M/Cm AdSh
  - M'/Sh
library BITLIB;
use BITLIB.bit_pack.all;

entity mult2C is
  port (CLK, St: in bit;
       Mplier, Mcan: in bit_vector(3 downto 0);
       Product: out bit_vector(6 downto 0);
       Done: out bit);
end mult2C;

architecture behave1 of mult2C is
  signal State: integer range 0 to 5;
  signal A, B: bit_vector(3 downto 0);
  alias M: bit is B(0);
begin
  process
    variable addout: bit_vector(4 downto 0);
  begin
    wait until CLK = '1';
    case State is
      when 0 => -- initial State
        if St='1' then
          A <= "0000"; -- Begin cycle
          B <= Mplier;
          State <= 1;
        end if;
      end case;
    end process;
  end process;
end mult2C;
Figure 4-12(b) Behavioral Model for 2’s Complement Multiplier

```
when 1 | 2 | 3  => -- "add/shift" State
  if M = '1' then
    addout := add4(A,Mcand,'0'); -- Add multiplicand to A and shift
    A <= Mcand(3) & addout(3 downto 1);
    B <= addout(0) & B(3 downto 1);
  else
    A <= A(3) & A(3 downto 1); -- Arithmetic right shift
    B <= A(0) & B(3 downto 1);
  end if;
State <= State + 1;
when 4  => -- add complement if sign bit
  if M = '1' then -- of multiplier is 1
    addout := add4(A, not Mcand,'1');
    A <= not Mcand(3) & addout(3 downto 1);
    B <= addout(0) & B(3 downto 1);
  else
    A <= A(3) & A(3 downto 1); -- Arithmetic right shift
    B <= A(0) & B(3 downto 1);
  end if;
State <= 5;  wait for 0 ns;
  Done <= '1';  Product <= A(2 downto 0) & B;
when 5  => -- output product
  State <= 0;
  Done <= '0';
end case;
end process;
end behave1;
```
**Figure 4-13 Command File and Simulation Results for (+5/8 by -3/8)**

-- command file to test signed multiplier  
list CLK St State A B Done Product  
force st 1 2, 0 22  
force clk 1 0, 0 10 - repeat 20  
-- (5/8 * -3/8)  
force Mcand 0101  
force Mplier 1101  
run 120

<table>
<thead>
<tr>
<th>ns</th>
<th>delta</th>
<th>CLK</th>
<th>St</th>
<th>State</th>
<th>A</th>
<th>B</th>
<th>Done</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0000</td>
<td>0000</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>2</td>
<td>+0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0000</td>
<td>0000</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>10</td>
<td>+0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0000</td>
<td>0000</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>20</td>
<td>+1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0000</td>
<td>1101</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>22</td>
<td>+0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0000</td>
<td>1101</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>30</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0000</td>
<td>1101</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>40</td>
<td>+1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0010</td>
<td>1110</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>50</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0010</td>
<td>1110</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>60</td>
<td>+1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0001</td>
<td>0111</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>70</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0001</td>
<td>0111</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>80</td>
<td>+1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0011</td>
<td>0011</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>90</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0011</td>
<td>0011</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>100</td>
<td>+2</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1111</td>
<td>0001</td>
<td>1</td>
<td>11100001</td>
</tr>
<tr>
<td>110</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1111</td>
<td>0001</td>
<td>1</td>
<td>11100001</td>
</tr>
<tr>
<td>120</td>
<td>+1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1111</td>
<td>0001</td>
<td>0</td>
<td>11100001</td>
</tr>
</tbody>
</table>
library BITLIB;
use BITLIB.bit_pack.all;
entity testmult is  end testmult;
architecture test1 of testmult is
component mult2C
  port(CLK, St: in bit;
      Mplier,Mcand : in bit_vector(3 downto 0);
      Product: out bit_vector (6 downto 0);
      Done: out bit);
end component;
constant N: integer := 11;  type arr is array(1 to N) of bit_vector(3 downto 0);
                           "1000", "0000", "1111", "1011");
                           "1000", "1101", "1111", "0000");
signal CLK, St, Done: bit;  signal Mplier, Mcand: bit_vector(3 downto 0);
signal Product: bit_vector(6 downto 0);
begin
  CLK <= not CLK after 10 ns;
  process
  begin
    for i in 1 to N loop
      Mcand <= Mcandarr(i);  Mplier <= Mplierarr(i);  St <= '1';
      wait until rising_edge(CLK);  St <= '0';  wait until falling_edge(Done);
    end loop;
  end process;
mult1: mult2c port map(Clk, St, Mplier, Mcand, Product, Done);
end test1;
Figure 4-15 Command File and Simulation of Signed Multiplier

-- Command file to test results of signed multiplier
list -NOtrigger Mplier Mcand product -Trigger done
run 1320

<table>
<thead>
<tr>
<th>ns</th>
<th>delta</th>
<th>mplier</th>
<th>mcand</th>
<th>product</th>
<th>done</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+1</td>
<td>0101</td>
<td>0111</td>
<td>0000000</td>
<td>0</td>
</tr>
<tr>
<td>90</td>
<td>+2</td>
<td>0101</td>
<td>0111</td>
<td>0100011</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>+2</td>
<td>0101</td>
<td>1101</td>
<td>0100011</td>
<td>0</td>
</tr>
<tr>
<td>210</td>
<td>+2</td>
<td>0101</td>
<td>1101</td>
<td>1110001</td>
<td>1</td>
</tr>
<tr>
<td>230</td>
<td>+2</td>
<td>1101</td>
<td>0101</td>
<td>1110001</td>
<td>0</td>
</tr>
<tr>
<td>330</td>
<td>+2</td>
<td>1101</td>
<td>0101</td>
<td>1110001</td>
<td>1</td>
</tr>
<tr>
<td>350</td>
<td>+2</td>
<td>1101</td>
<td>1101</td>
<td>1110001</td>
<td>1</td>
</tr>
<tr>
<td>450</td>
<td>+2</td>
<td>1101</td>
<td>1101</td>
<td>0001001</td>
<td>0</td>
</tr>
<tr>
<td>470</td>
<td>+2</td>
<td>0111</td>
<td>0111</td>
<td>0110001</td>
<td>1</td>
</tr>
<tr>
<td>570</td>
<td>+2</td>
<td>0111</td>
<td>0111</td>
<td>0110001</td>
<td>1</td>
</tr>
<tr>
<td>590</td>
<td>+2</td>
<td>0111</td>
<td>1000</td>
<td>0110001</td>
<td>0</td>
</tr>
<tr>
<td>690</td>
<td>+2</td>
<td>0111</td>
<td>1000</td>
<td>1001000</td>
<td>1</td>
</tr>
<tr>
<td>710</td>
<td>+2</td>
<td>1000</td>
<td>0111</td>
<td>1001000</td>
<td>0</td>
</tr>
<tr>
<td>810</td>
<td>+2</td>
<td>1000</td>
<td>0111</td>
<td>1001000</td>
<td>1</td>
</tr>
<tr>
<td>830</td>
<td>+2</td>
<td>1000</td>
<td>1000</td>
<td>1001000</td>
<td>0</td>
</tr>
<tr>
<td>930</td>
<td>+2</td>
<td>1000</td>
<td>1000</td>
<td>1000000</td>
<td>1</td>
</tr>
<tr>
<td>950</td>
<td>+2</td>
<td>1101</td>
<td>0000</td>
<td>1000000</td>
<td>0</td>
</tr>
<tr>
<td>1050</td>
<td>+2</td>
<td>1101</td>
<td>0000</td>
<td>0000000</td>
<td>1</td>
</tr>
<tr>
<td>1070</td>
<td>+2</td>
<td>1111</td>
<td>1111</td>
<td>0000000</td>
<td>0</td>
</tr>
<tr>
<td>1170</td>
<td>+2</td>
<td>1111</td>
<td>1111</td>
<td>0000001</td>
<td>1</td>
</tr>
<tr>
<td>1190</td>
<td>+2</td>
<td>0000</td>
<td>1011</td>
<td>0000001</td>
<td>0</td>
</tr>
<tr>
<td>1290</td>
<td>+2</td>
<td>0000</td>
<td>1011</td>
<td>0000000</td>
<td>1</td>
</tr>
<tr>
<td>1310</td>
<td>+2</td>
<td>0101</td>
<td>0111</td>
<td>0000000</td>
<td>0</td>
</tr>
</tbody>
</table>
library BITLIB;
use BITLIB.bit_pack.all;

entity mult2Cs is
port (CLK, St: in bit;
      Mplier,Mcand : in bit_vector(3 downto 0);
      Product: out bit_vector (6 downto 0);  Done: out bit);
end mult2Cs;

architecture behave2 of mult2Cs is
  signal State, Nextstate: integer range 0 to 5; signal A, B: bit_vector(3 downto 0);
  signal AdSh, Sh, Load, Cm: bit; signal addout: bit_vector(4 downto 0);
  alias M: bit is B(0);
begin
  process (state, st, M)
  begin
    Load <= '0'; AdSh <= '0'; Sh <= '0'; Cm <= '0'; Done <= '0';
    case State is
      when 0=> -- initial State
        if St='1' then Load <= '1'; Nextstate <= 1; end if;
      when 1 | 2 | 3 => -- "add/shift" State
        if M = '1' then AdSh <= '1'; else Sh <= '1'; end if;
        Nextstate <= State + 1;
      when 4 => -- add complement if sign
        if M = '1' then Cm <= '1'; AdSh <= '1'; end if;
        else Sh <= '1'; end if;
      nextstate <= 5;
    end case;
  end process;
end behave2;
when 5 => 
    done <= '1';
    nextstate <= 0;
end case;
end process;

addout <= add4(A, Mcand, '0') when Cm = '0' else add4(A, not Mcand, '1');

process
begin
    wait until CLK = '1';
    if Load = '1' then
        A <= "0000";
        B <= Mplier;
    end if;
    if AdSh = '1' then
        A <= (Mcand(3) xor Cm) & addout(3 downto 1);
        B <= addout(0) & B(3 downto 1);
    end if;
    if Sh = '1' then
        A <= A(3) & A(3 downto 1); B <= A(0) & B(3 downto 1);
    end if;
    State <= Nextstate;
end process;

Product <= A(2 downto 0) & B;
end behave2;
Figure 4-17  Realization of Multiplier Control Network
Figure 4-18(a) Model for 2's Complement Multiplier Using Control Equations

-- This model of a 4-bit multiplier for 2's complement numbers
-- implements the controller using a counter and logic equations.

```
library BITLIB;
use BITLIB.bit_pack.all;

entity mult2CEQ is
    port(CLK, St: in bit;
        Mplier, Mcand: in bit_vector(3 downto 0);
        Product: out bit_vector(6 downto 0));
end mult2CEQ;

architecture m2ceq of mult2CEQ is
    signal A, B, Q, Comp: bit_vector(3 downto 0);
    signal addout: bit_vector(4 downto 0);
    signal AdSh, Sh, Load, Cm, Done, Ld1, CLR1, P1: bit;
    Signal One: bit:='1';
    Signal Din: bit_vector(3 downto 0) := "0100";
    alias M: bit is B(0);
begin
    Count1: C74163 port map (Ld1, CLR1, P1, One, CLK, Din, open, Q);
P1 <= Q(2); CLR1 <= not Q(3); Done <= Q(3); Sh <= not M and Q(2);
AdSh <= M and Q(2); Cm <= Q(1) and Q(0) and M;
Load <= not Q(3) and not Q(2) and St; Ld1 <= not Load;
Comp <= Mcand xor (Cm & Cm & Cm & Cm); -- complement Mcand if Cm='1'
addout <= add4(A,Comp,Cm); -- add complementer output to A
```

process
begin
    wait until CLK = '1'; -- executes on rising edge
    if Load = '1' then
        A <= "0000";
        B <= Mplier;
    end if;
    if AdSh = '1' then
        A <= (Mcand(3) xor Cm) & addout(3 downto 1);
        B <= addout(0) & B(3 downto 1);
    end if;
    if Sh = '1' then
        A <= A(3) & A(3 downto 1);
        B <= A(0) & B(3 downto 1);
    end if;
    if Done = '1' then
        Product <= A(2 downto 0) & B;
    end if;
end process;
end m2ceq;
Parallel Divider for Positive Binary Numbers – From Page 144

---

**divisor** \[\begin{array}{c} 1101 \\ \end{array} \] **quotient** \[\begin{array}{c} 1010 \\ \end{array} \] **dividend** \[\begin{array}{c} 1000111 \\ \end{array} \]

\[1101 \]
\[0111 \]
\[0000 \]
\[1111 \]
\[1101 \]
\[0101 \]
\[0000 \]
\[0101 \]

(135 \( ÷ \) 13 = 10 with a remainder of 5)

---

load: \[\begin{array}{c} 0 1 0 0 0 0 1 1 1 \\ \end{array} \] (can't subtract) \[\begin{array}{c} 1 1 0 1 \\ \end{array} \]

shift L: \[\begin{array}{c} 1 0 0 0 0 1 1 1 0 \quad 1 1 0 1 \quad 0 1 1 1 1 1 1 1 1 1 \\ \end{array} \]

subtract: \[\begin{array}{c} 0 0 0 1 1 1 1 1 1 1 \\ \end{array} \]

shift L: \[\begin{array}{c} 0 0 1 1 1 1 1 0 0 \quad 1 1 0 1 \quad 0 0 1 0 1 1 0 1 0 1 \quad 0 1 0 1 \quad 0 0 1 0 1 1 0 1 0 \\ \end{array} \]

Note that after the shift, the rightmost position in the dividend register is "empty."

---

Dividing line between dividend and quotient

First quotient digit

Third quotient digit

---

done remainder quotient
Figure 4-19 Block Diagram for Parallel Binary Divider

- **Dividend Register**: X8 to X0
- **Subtractor and comparator**: Y3 to Y0
- **Control**
  - Input: St (Start Signal), Sh, Su, C
  - Output: V (overflow indicator)
- **Clock**
Figure 4-20 State Diagram for Divider Control Circuit

<table>
<thead>
<tr>
<th>State</th>
<th>StC 00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>S0</td>
<td>S1</td>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>S0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td>S2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S3</td>
<td>S4</td>
<td>S3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S4</td>
<td>S5</td>
<td>S4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S5</td>
<td>S0</td>
<td>S0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>StC 00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>Load</td>
<td>Load</td>
</tr>
<tr>
<td></td>
<td>Sh</td>
<td>V</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Sh</td>
<td>Su</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Sh</td>
<td>Su</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Sh</td>
<td>Su</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Su</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Control Signals for Signed Divider

\[ LdU \] Load upper half of dividend from bus

\[ LdL \] Load lower half of dividend from bus

\[ Lds \] Load sign of dividend into sign flip-flop

\[ S \] Sign of dividend

\[ Cm1 \] Complement dividend register (2's complement)

\[ Ldd \] Load divisor from bus

\[ Su \] Enable adder output onto bus (Ena) and load upper half of dividend from bus

\[ Cm2 \] Enable complementer (\( Cm2 \) equals the complement of the sign bit of the divisor, so that a positive divisor is complemented and a negative divisor is not)

\[ Sh \] Shift the dividend register left one place and increment the counter

\[ C \] Carry output from adder (If C = 1, the divisor can be subtracted from the upper dividend.)

\[ St \] Start

\[ V \] Overflow

\[ Qneg \] Quotient will be negative (\( Qneg = 1 \) when sign of dividend and divisor are different)
Figure 4-21 Block Diagram for Signed Divider
Figure 4-22 State Graph for Signed Divider Control Network
library BITLIB;
use BITLIB.bit_pack.all;

entity sdiv is
  port(Clk,St: in bit;
       Dbus: in bit_vector(15 downto 0); Quotient: out bit_vector(15 downto 0);
       V, Rdy: out bit);
end sdiv;

architecture Signdiv of Sdiv is
  constant zero_vector: bit_vector(31 downto 0):=(others=>'0');
  signal Sign,C,NC: bit;
  signal Divisor,Sum,Compout: bit_vector(15 downto 0);
  signal Dividend: bit_vector(31 downto 0);
  alias Q: bit_vector(15 downto 0) is Dividend(15 downto 0);
  alias Acc: bit_vector(15 downto 0) is Dividend(31 downto 16);
begin
  -- concurrent statements
  compout <= divisor when divisor(15) = '1' -- 1's complemener
  else not divisor;
  Addvec(Acc,compout,not divisor(15),Sum,C,16); -- 16-bit adder
  Quotient <= Q; Rdy <= '1' when State=0 else '0';
Figure 4-23(b) VHDL Model of 32-bit Signed Divider

```vhdl
process
begin
    wait until Clk = '1'; -- wait for rising edge of clock
    case State is
        when 0 =>
            if St = '1' then
                Acc <= Dbus; -- load upper dividend
                Sign <= Dbus(15); State <= 1;
                V <= '0'; Count <= 0; -- initialize overflow// initialize counter
            end if;
        when 1 =>
            Q <= Dbus; State <= 2; -- load lower dividend
        when 2 =>
            Divisor <= Dbus;
            if Sign = '1' then -- two's complement Dividend if necessary
                addvec(not Dividend,zero_vector,'1',Dividend,NC,32);
            end if; State <= 3;
        when 3 =>
            Dividend <= Dividend(30 downto 0) & '0'; -- left shift
            Count <= Count+1; State <= 4;
        when 4 =>
            if C = '1' then -- C
                v <= '1'; State <= 0;
            else -- C'
                Dividend <= Dividend(30 downto 0) & '0'; -- left shift
                Count <= Count+1; State <= 5;
            end if;
        end case;
end process;
```
Figure 4-23(c) VHDL Model of 32-bit Signed Divider

```vhdl
when 5 =>
  if C = '1' then
    ACC <= Sum; -- subtract
    Q(0) <= '1';
  else
    Dividend <= Dividend(30 downto 0) & '0'; -- left shift
    if Count = 15 then
      count <= 0; State <= 6;
    else
      Count <= Count+1;
    end if;
  end if;
when 6 =>
  if C = '1' then
    ACC <= Sum; -- subtract
    Q(0) <= '1';
  else if (Sign xor Divisor(15))='1' then
    addvec(not Dividend,zero_vector,'1',Dividend,NC,32);
  end if;
  state <= 0;
end case;
end process;
end signdiv;
```
library BITLIB;
use BITLIB.bit_pack.all;
entity testsdiv is
end testsdiv;
architecture test1 of testsdiv is
component sdiv
  port(Clk,St: in bit;
       Dbus: in bit_vector(15 downto 0);  Quotient: out bit_vector(15 downto 0);
       V, Rdy: out bit);
end component;
constant N: integer := 12; -- test sdiv1 N times

type arr1 is array(1 to N) of bit_vector(31 downto 0);
type arr2 is array(1 to N) of bit_vector(15 downto 0);
constant dividendarr: arr1 := (X"0000006F",X"07FF00BB",X"FFFFFE08",
                             X"FF80030A",X"3FF80000",X"3FFF7FFF",X"C0008000",X"C0008000",
                             X"C0008001",X"00000000",X"FFFFFFFF",X"FFFFFFFF”);
constant divisorarr: arr2 := (X"0007", X"E005", X"001E", X"EFFA", X"7FFF", X"7FFF", X"7FFF", X"7FFF",
                             X"8000", X"7FFF", X"0001", X"7FFF", X"0000”);
signal CLK, St, V, Rdy: bit; signal Dbus, Quotient, divisor: bit_vector(15 downto 0);
signal Dividend: bit_vector(31 downto 0); signal count: integer range 0 to N;
begin
  CLK <= not CLK after 10 ns;
  process
  begin
    for i in 1 to N loop
      St <= '1';
      Dbus <= dividendarr(i) (31 downto 16);
      wait until rising_edge(CLK);
      Dbus <= dividendarr(i) (15 downto 0);
      wait until rising_edge(CLK);
      Dbus <= divisorarr(i);
      St <= '0';
      dividend <= dividendarr(i); -- save dividend for listing
      divisor <= divisorarr(i); -- save divisor for listing
      wait until (Rdy = '1');
      count <= i; -- save index for triggering
    end loop;
  end process;
  sdiv1: sdiv port map(Clk, St, Dbus, Quotient, V, Rdy);
end test1;
**Figure 4-25 Simulation Test Results for Signed Divider**

-- Command file to test results of signed divider
list -hex -Notrigger dividend divisor Quotient V -Trigger count
run 5300

<table>
<thead>
<tr>
<th>ns</th>
<th>delta</th>
<th>dividend</th>
<th>divisor</th>
<th>quotient</th>
<th>v</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+0</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>470</td>
<td>+3</td>
<td>0000006F</td>
<td>0007</td>
<td>000F</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>910</td>
<td>+3</td>
<td>07FF00BB</td>
<td>E005</td>
<td>BFFE</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1330</td>
<td>+3</td>
<td>FFFFFFFE08</td>
<td>001E</td>
<td>FFF0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1910</td>
<td>+3</td>
<td>FF80030A</td>
<td>EFFA</td>
<td>07FC</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2010</td>
<td>+3</td>
<td>3FFF8000</td>
<td>7FFF</td>
<td>0000</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2710</td>
<td>+3</td>
<td>3FFF7FFF</td>
<td>7FFF</td>
<td>7FFF</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2810</td>
<td>+3</td>
<td>C0008000</td>
<td>7FFF</td>
<td>0000</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>3510</td>
<td>+3</td>
<td>C0008000</td>
<td>8000</td>
<td>7FFF</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>4210</td>
<td>+3</td>
<td>C0008001</td>
<td>7FFF</td>
<td>8001</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>4610</td>
<td>+3</td>
<td>00000000</td>
<td>0001</td>
<td>0000</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>5010</td>
<td>+3</td>
<td>FFFFFFFF</td>
<td>7FFF</td>
<td>0000</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>5110</td>
<td>+3</td>
<td>FFFFFFFF</td>
<td>0000</td>
<td>0002</td>
<td>1</td>
<td>C</td>
</tr>
</tbody>
</table>