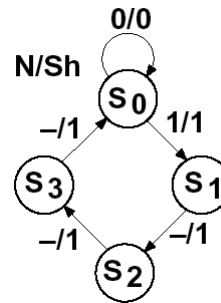




## 4.1 Design of a Serial Adder with Accumulator - Operation

	X	Y	$c_i$	$sum_j$	$c_{i+1}$
$t_0$	0101	0111	0	0	1
$t_1$	0010	1011	1	0	1
$t_2$	0001	1101	1	1	1
$t_3$	1000	1110	1	1	0
$t_4$	1100	0111	0	(1)	(0)

Present State	Next State		Present Output (Sh)	
	N=0	N=1	N=0	N=1
S <sub>0</sub>	S <sub>0</sub>	S <sub>1</sub>	0	1
S <sub>1</sub>	S <sub>2</sub>	S <sub>2</sub>	1	1
S <sub>2</sub>	S <sub>3</sub>	S <sub>3</sub>	1	1
S <sub>3</sub>	S <sub>0</sub>	S <sub>0</sub>	1	1



## 4.2 State Graphs for Control Networks

¥ Use variable names instead of 0s and 1s

—E.g.,  $X_i X_j / Z_p Z_q$

¥ if  $X_i$  and  $X_j$  inputs are 1, the outputs  $Z_p$  and  $Z_q$  are 1

(all other outputs are 0s)

—E.g.,  $X = X_1 X_2 X_3 X_4, Z = Z_1 Z_2 Z_3 Z_4$

¥  $X_1 X_4 / Z_2 Z_3 == 1 - - 0 / 0 1 1 0$

## 4.2 State Graphs for Control Networks - Constraints on Input Labels

¥ Assume:  $I$  — input expression =>  
we traverse the arc when  $I=1$

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

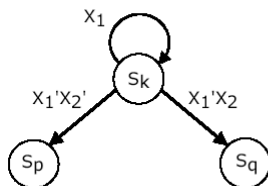
Assures that at most one input label can be 1 at any given time

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

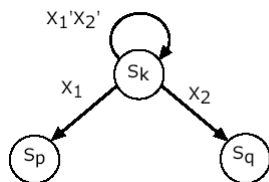
Assures that at least one input label will be 1 at any given time

1 + 2: Exactly one label will be 1 =>  
the next state will be uniquely defined for every input combination

## 4.2 State Graphs for Control Networks - Constraints on Input Labels (cont'd)



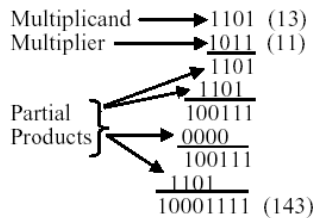
$$\begin{aligned} (X_1)(X_1'X_2') &= 0 \\ (X_1)(X_1'X_2) &= 0 \\ (X_1'X_2')(X_1'X_2) &= 0 \\ X_1 + X_1'X_2' + X_1'X_2 &= 1 \end{aligned}$$



Inputs are  $X_1 X_2 X_3$   
( $X_1 = X_2 = 1$  not allowed)

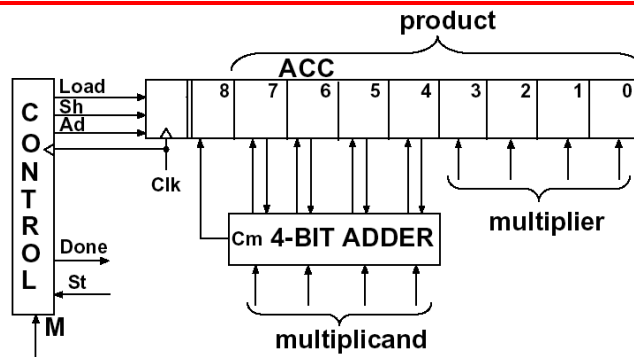
	000	001	010	011	100	101	110	111
$S_k$	$S_k$	$S_k$	$S_q$	$S_q$	$S_p$	$S_p$	-	-

### 4.3 Design of a Binary Multiplier - Terms



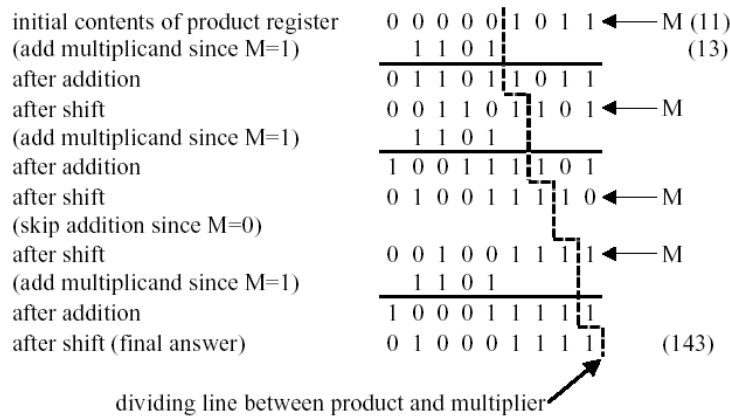
Note: we use unsigned binary numbers

### 4.3 Design of a Binary Multiplier - Block Diagram

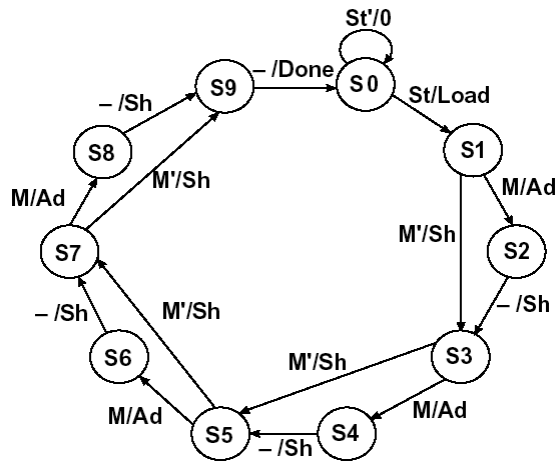


Ad —add signal // adder outputs are stored into the ACC  
 Sh —shift signal // shift all 9 bits to right  
 Ld —load signal // load multiplier into the 4 lower bits of the ACC and clear the upper 5 bits

### 4.3 Design of a Binary Multiplier - Multiplication Example



### 4.3 Design of a Binary Multiplier - State Graph



### 4.3 Design of a Binary Multiplier - Behavioral VHDL Model

```

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 =>                            -- initial State
        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;
  Done <= '1' when State = 9 else '0';
end behave1;

```

Spring 2004 Slide #11

Electrical and Computer Engineering

### 4.3 Design of a Binary Multiplier - Behavioral VHDL Model (cont'd)

```

      when 1 | 3 | 5 | 7 =>                 -- "add/shift" State
        if M = '1' then                   -- Add multiplicand
          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;

```

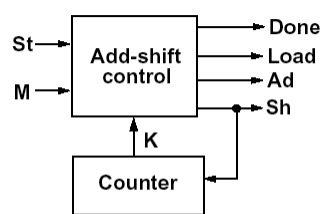
Spring 2004 Slide #12

Electrical and Computer Engineering

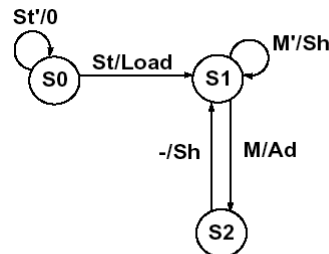
### 4.3 Design of a Binary Multiplier - Multiplier Control with Counter

- ¥ Current design: control part generates the control signals (shift/add) and counts the number of steps
- ¥ If the number of bits is large (e.g., 64), the control network can be divided into a counter and a shift/add control

### 4.3 Design of a Binary Multiplier - Multiplier Control with Counter



(a) Multiplier control

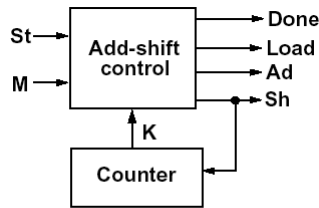


(b) State graph for add-shift control

Add-shifts control: tests  $St$  and  $M$  and generates the proper sequence of add and shift signals

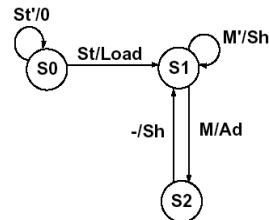
Counter control: counter generates a completion signal  $K$  that stops the multiplier after the proper number of shifts have been completed

### 4.3 Design of a Binary Multiplier - Multiplier Control with Counter

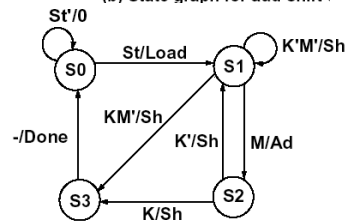


(a) Multiplier control

- ¥ Increment counter each time a shift signal is generated
- ¥ Generate K after n-1 shifts occurred



(b) State graph for add-shift



### 4.3 Design of a Binary Multiplier - Operation Using a Counter

Time	State	Counter	Product Register	St	M	K	Load	Ad	Sh	Done
t0	S0	00	00000000	0	0	0	0	0	0	0
t1	S0	00	00000000	1	0	0	1	0	0	0
t2	S1	00	000001011	0	1	0	0	1	0	0
t3	S2	00	011011011	0	1	0	0	0	1	0
t4	S1	01	001101101	0	1	0	0	1	0	0
t5	S2	01	100111101	0	1	0	0	0	1	0
t6	S1	10	010011110	0	0	0	0	0	1	0
t7	S1	11	001001111	0	1	1	0	1	0	0
t8	S2	11	100011111	0	1	1	0	0	1	0
t9	S3	00	010001111	0	1	0	0	0	0	1

### 4.3 Design of a Binary Multiplier - Array Multiplier

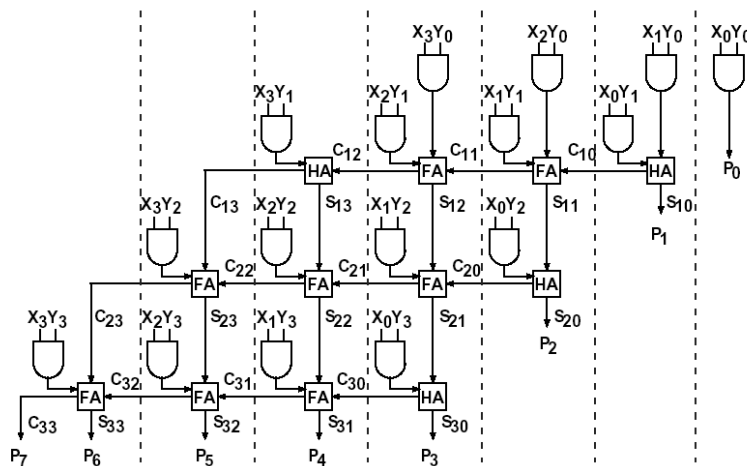
	X <sub>3</sub>	X <sub>2</sub>	X <sub>1</sub>	X <sub>0</sub>	Multiplicand				
	Y <sub>3</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>	Multiplier				
	X <sub>3</sub> Y <sub>0</sub>	X <sub>2</sub> Y <sub>0</sub>	X <sub>1</sub> Y <sub>0</sub>	X <sub>0</sub> Y <sub>0</sub>	partial product 0				
	X <sub>3</sub> Y <sub>1</sub>	X <sub>2</sub> Y <sub>1</sub>	X <sub>1</sub> Y <sub>1</sub>	X <sub>0</sub> Y <sub>1</sub>	partial product 1				
	C <sub>12</sub>	C <sub>11</sub>	C <sub>10</sub>		1st row carries				
	C <sub>13</sub>	S <sub>13</sub>	S <sub>12</sub>	S <sub>11</sub>	S <sub>10</sub>	1st row sums			
	X <sub>3</sub> Y <sub>2</sub>	X <sub>2</sub> Y <sub>2</sub>	X <sub>1</sub> Y <sub>2</sub>	X <sub>0</sub> Y <sub>2</sub>	partial product 2				
	C <sub>22</sub>	C <sub>21</sub>	C <sub>20</sub>		2nd row carries				
	C <sub>23</sub>	S <sub>23</sub>	S <sub>22</sub>	S <sub>21</sub>	S <sub>20</sub>	2nd row sums			
	X <sub>3</sub> Y <sub>3</sub>	X <sub>2</sub> Y <sub>3</sub>	X <sub>1</sub> Y <sub>3</sub>	X <sub>0</sub> Y <sub>3</sub>	partial product 3				
	C <sub>32</sub>	C <sub>31</sub>	C <sub>30</sub>		3rd row carries				
	C <sub>33</sub>	S <sub>33</sub>	S <sub>32</sub>	S <sub>31</sub>	S <sub>30</sub>	3rd row sums			
	P <sub>7</sub>	P <sub>6</sub>	P <sub>5</sub>	P <sub>4</sub>	P <sub>3</sub>	P <sub>2</sub>	P <sub>1</sub>	P <sub>0</sub>	final product

What do we need to realize  
Array Multiplier?

AND gates = ?  
FA = ?  
HA = ?

Spring 2004 Slide #17

### 4.3 Design of a Binary Multiplier - Array Multiplier (cont'd)



Spring 2004 Slide #18

### 4.3 Design of a Binary Multiplier - Array Multiplier (cont'd)

---

¥ Complexity of the N-bit array multiplier

—number of AND gates = ?

—number of HA = ?

—number of FA = ?

¥ Delay

— $t_g$  — longest AND gate delay

— $t_{ad}$  —longest possible delay through an adder

### 4.4 Multiplication of Signed Binary Numbers

---

¥ How to multiply signed binary numbers?

¥ Procedure

—Complement the multiplier if negative

—Complement the multiplicand if negative

—Multiply two positive binary numbers

—Complement the product if it should be negative

¥ Simple but requires more hardware and time than other available methods

## 4.4 Multiplication of Signed Binary Numbers

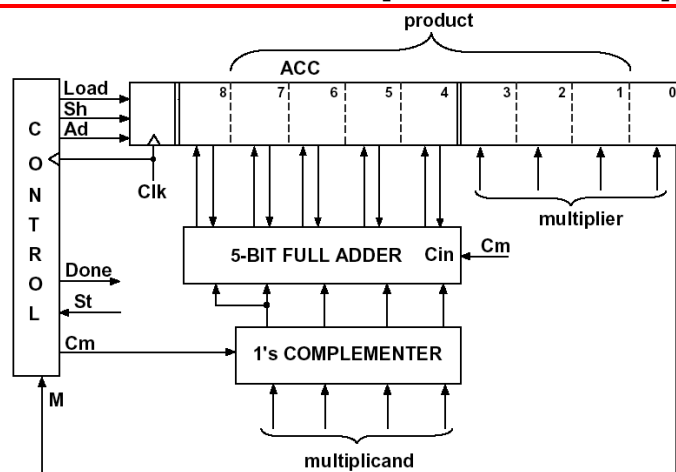
### ¥ Four cases

- Multiplicand is positive, multiplier is positive
- Multiplicand is negative, multiplier is positive
- Multiplicand is positive, multiplier is negative
- Multiplier is negative, multiplicand is negative

### ¥ Examples

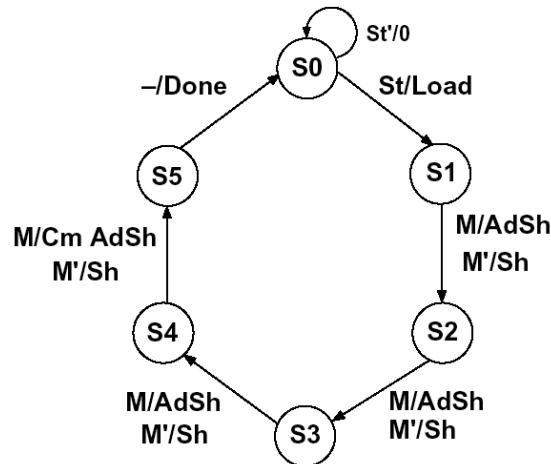
- 0111 x 0101 = ?    ¥ Preserve the sign of the partial product at each step
- 1101 x 0101 = ?    ¥ If multiplier is negative, complement the multiplicand before adding it in at the last step
- 0101 x 1101 = ?
- 1011 x 1101 = ?

## 4.4 Multiplication of Signed Binary Numbers - 2's Complement Multiplier





## 4.4 Multiplication of Signed Binary Numbers - New State Graph



## 4.4 Multiplication of Signed Binary Numbers - New Behavioral Model

```

library BITLIB;
use BITLIB.bit_pack.all;

entity mult2C 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 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; -- load the multiplier
          State <= 1;
        end if;
    end case;
  end process;
end behave1;
  
```

## 4.4 Multiplication of Signed Binary Numbers - New Behavioral Model

```

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;

```

Spring 2004 Slide #27

Electrical and Computer Engineering

## 4.4 Multiplication of Signed Binary Numbers - Simulation

```

-- 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

```

ns	delta	CLK	St	State	A	B	Done	Product
0	+1	1	0	0	0000	0000	0	0000000
2	+0	1	1	0	0000	0000	0	0000000
10	+0	0	1	0	0000	0000	0	0000000
20	+1	1	1	1	0000	1101	0	0000000
22	+0	1	0	1	0000	1101	0	0000000
30	+0	0	0	1	0000	1101	0	0000000
40	+1	1	0	2	0010	1110	0	0000000
50	+0	0	0	2	0010	1110	0	0000000
60	+1	1	0	3	0001	0111	0	0000000
70	+0	0	0	3	0001	0111	0	0000000
80	+1	1	0	4	0011	0011	0	0000000
90	+0	0	0	4	0011	0011	0	0000000
100	+2	1	0	5	1111	0001	1	1110001
110	+0	0	0	5	1111	0001	1	1110001
120	+1	1	0	0	1111	0001	0	1110001

Spring 2004 Slide #28

Electrical and Computer Engineering

## 4.4 Multiplication of Signed Binary Numbers - Test Bench

---

```
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);
  constant Mcandarr: arr := ("0111", "1101", "0101", "1101", "0111", "1000", "0111",
    "1000", "0000", "1111", "1011");
  constant Mplierarr: arr := ("0101", "0101", "1101", "1101", "0111", "0111", "1000",
    "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;
```

Spring 2004 Slide #29