

ShanghaiTech University

EE 115B: Digital Circuits

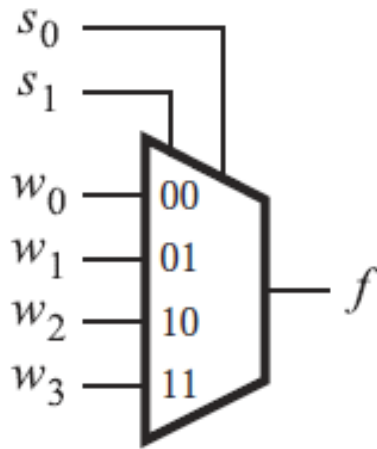
Fall 2022

Lecture 14

Hengzhao Yang
November 17, 2022

Example 2: 16-to-1 mux

- Truth table for 4-to-1 mux



(a) Graphical symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

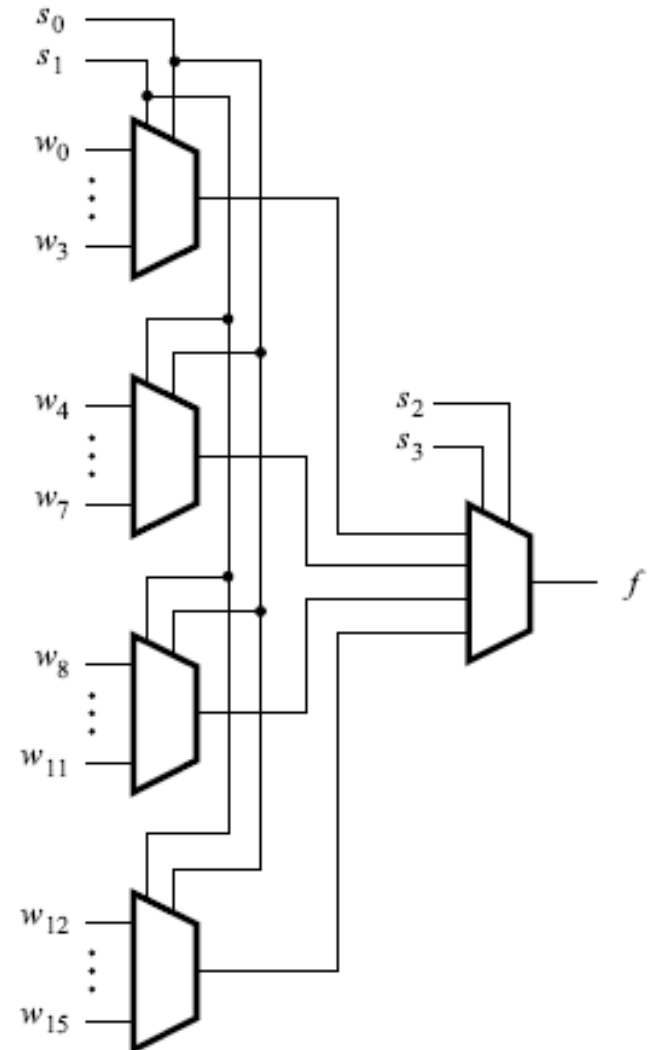


Figure 6.4 A 16-to-1 multiplexer.

Example 2: 16-to-1 mux

- VHDL code for 4-to-1 mux

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux4to1 IS
    PORT ( w0, w1, w2, w3 : IN    STD_LOGIC ;
          s                : IN    STD_LOGIC_VECTOR(1 DOWNTO 0) ;
          f                : OUT   STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE Behavior OF mux4to1 IS
BEGIN
    WITH s SELECT
        f <=  w0 WHEN "00",
              w1 WHEN "01",
              w2 WHEN "10",
              w3 WHEN OTHERS ;
END Behavior ;
```

Example 2: 16-to-1 mux

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux16to1 IS
    PORT ( w : IN    STD_LOGIC_VECTOR(0 TO 15) ;
          s : IN    STD_LOGIC_VECTOR(3 DOWNT0 0) ;
          f : OUT   STD_LOGIC ) ;
END mux16to1 ;

ARCHITECTURE Structure OF mux16to1 IS
    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;

    COMPONENT mux4to1
        PORT ( w0, w1, w2, w3 : IN    STD_LOGIC ;
              s                : IN    STD_LOGIC_VECTOR(1 DOWNT0 0) ;
              f                : OUT   STD_LOGIC ) ;
    END COMPONENT ;

    BEGIN
        Mux1: mux4to1 PORT MAP
            ( w(0), w(1), w(2), w(3), s(1 DOWNT0 0), m(0) ) ;
        Mux2: mux4to1 PORT MAP
            ( w(4), w(5), w(6), w(7), s(1 DOWNT0 0), m(1) ) ;
        Mux3: mux4to1 PORT MAP
            ( w(8), w(9), w(10), w(11), s(1 DOWNT0 0), m(2) ) ;
        Mux4: mux4to1 PORT MAP
            ( w(12), w(13), w(14), w(15), s(1 DOWNT0 0), m(3) ) ;
        Mux5: mux4to1 PORT MAP
            ( m(0), m(1), m(2), m(3), s(3 DOWNT0 2), f ) ;
    END Structure ;

```

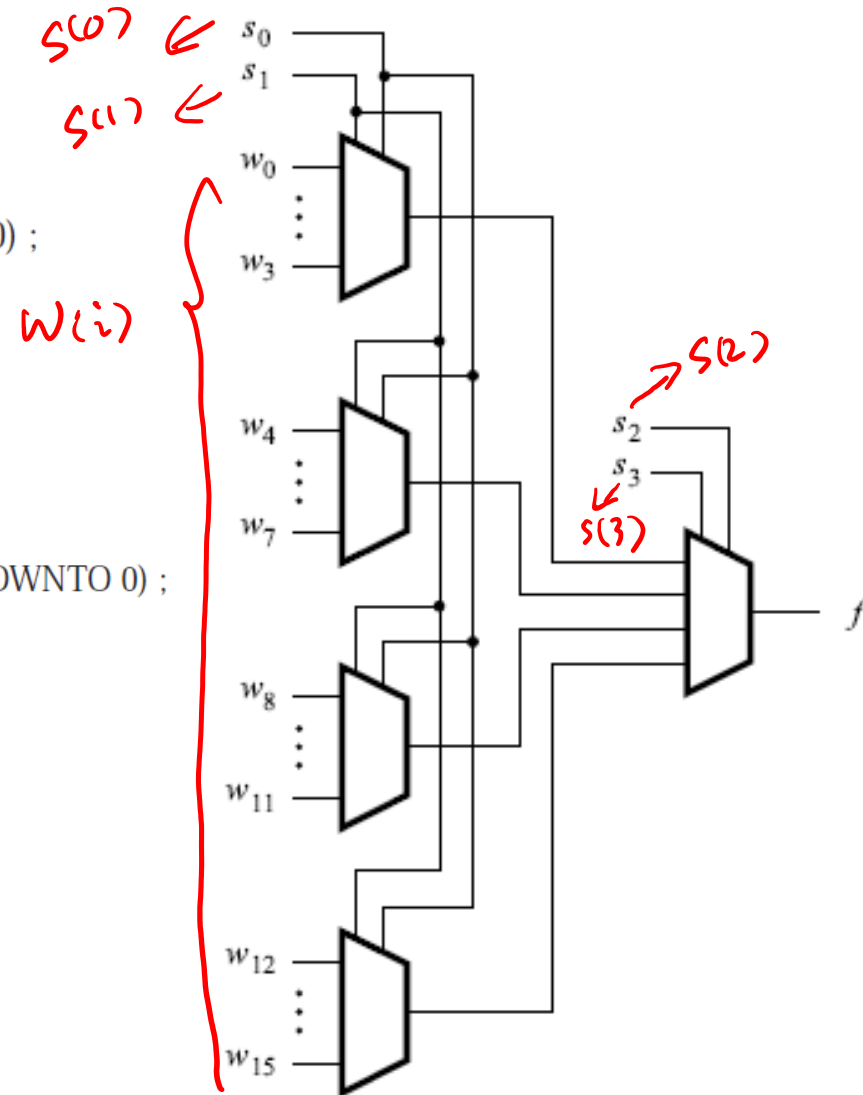
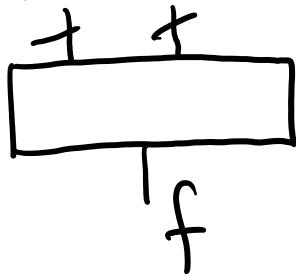


Figure 6.4 A 16-to-1 multiplexer.

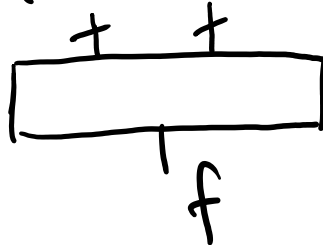
16-to-1 mux: operation

- Select signal
 - $s(3)s(2)$: select block
 - $s(1)s(0)$: select input

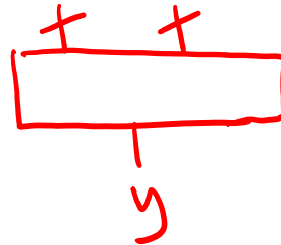
> 4-1 w_0, \dots, w_3 $s(1:0)$



> 16-1 $w(0:15)$ $s(3:0)$



$w(0:15)$ $s(3:0)$



$s(3)$	$s(2)$	$s(1)$	$s(0)$	f
0	0	0	0	$w(0)$
0	0	0	1	$w(1)$
0	0	1	0	$w(2)$
0	0	1	1	$w(3)$
0	1	0	0	$w(4)$
0	1	0	1	$w(5)$
0	1	1	0	$w(6)$
0	1	1	1	$w(7)$
1	0	0	0	$w(8)$
1	0	0	1	$w(9)$
1	0	1	0	$w(10)$
1	0	1	1	$w(11)$
1	1	0	0	$w(12)$
1	1	0	1	$w(13)$
1	1	1	0	$w(14)$
1	1	1	1	$w(15)$

FOR GENERATE statement

- A compact form to instantiate components
- Example: 16-to-1 mux
- Original code

```
Mux1: mux4to1 PORT MAP
    ( w(0), w(1), w(2), w(3), s(1 DOWNT0 0), m(0) ) ;
Mux2: mux4to1 PORT MAP
    ( w(4), w(5), w(6), w(7), s(1 DOWNT0 0), m(1) ) ;
Mux3: mux4to1 PORT MAP
    ( w(8), w(9), w(10), w(11), s(1 DOWNT0 0), m(2) ) ;
Mux4: mux4to1 PORT MAP
    ( w(12), w(13), w(14), w(15), s(1 DOWNT0 0), m(3) ) ;
Mux5: mux4to1 PORT MAP
    ( m(0), m(1), m(2), m(3), s(3 DOWNT0 2), f ) ;
```

FOR GENERATE statement

- Code using FOR GENERATE statement
 - “G1” is a label for this statement
 - “i” is a local variable whose scope is limited to this statement

```
G1: FOR i IN 0 TO 3 GENERATE
    Muxes: mux4to1 PORT MAP (
        w(4*i), w(4*i+1), w(4*i+2), w(4*i+3), s(1 DOWNT0 0), m(i) ) ;
END GENERATE ;
Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNT0 2), f ) ;
```

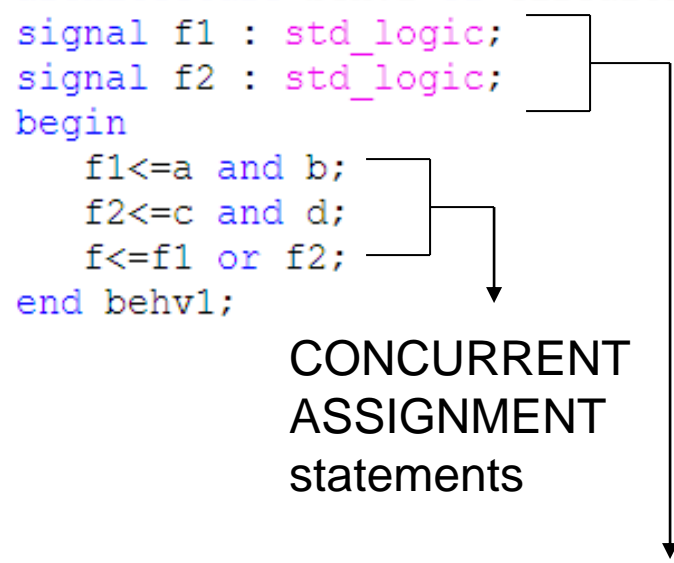
PROCESS

- Architecture
 - PROCESS statements
 - IF statements
 - CASE statements

Architecture: general form

```
architecture <arch_name> of <entity_name> is
  -- architecture declarations
  [SIGNAL declarations]
  [COMPONENT declarations]
  [CONSTANT declarations]
  [TYPE declarations]
  [ATTRIBUTE specifications]
begin
  -- architecture body
  [CONCURRENT ASSIGNMENT statements;]
  [COMPONENT instantiation statements;]
  [PROCESS statements;]
  [GENERATE statements;]
end <arch_name>;
```

```
architecture behv1 of circuit1 is
  signal f1 : std_logic;
  signal f2 : std_logic;
begin
  f1<=a and b;
  f2<=c and d;
  f<=f1 or f2;
end behv1;
```



CONCURRENT
ASSIGNMENT
statements

SIGNAL
declarations

PROCESS

- Multiple PROCESS statements are concurrent
- Statements inside a PROCESS are sequential
- Syntax

```
[process_label:]  
PROCESS (<signal1>, <signal2> ...)  
    [VARIABLE declarations]  
BEGIN  
    [WAIT statement]  
    [IF statements]  
    [CASE statements]  
    [FOR-LOOP statements]  
    [WHILE-LOOP statements]  
END PROCESS;
```

PROCESS

- Optional: process_label
- **Sensitivity list**: (<signal 1>, <signal 2> ...)
 - Specify signals that cause the PROCESS to be evaluated
 - When the value of any signal in the sensitivity list changes, the PROCESS is **active** and the statements inside the PROCESS are executed in sequential order
 - Statements inside PROCESS: IF, CASE, WAIT, LOOP

```
[process_label:]  
PROCESS (<signal1>, <signal2> ...)  
    [VARIABLE declarations]  
BEGIN  
    [WAIT statement]  
    [IF statements]  
    [CASE statements]  
    [FOR-LOOP statements]  
    [WHILE-LOOP statements]  
END PROCESS;
```

PROCESS

- Any assignments made to signals inside the PROCESS are not visible outside the PROCESS until all the statements in the PROCESS have been evaluated.
- Signal assignments only take effect at the end of the PROCESS.
- If multiple assignments are made to the same signal, only the last one has any visible effect.

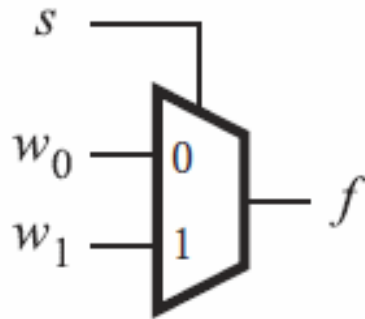
IF statement

- IF statement can only appear in a PROCESS
- Syntax

```
if <condition> then
    statements;
[elseif <condition> then
    statement; ]
[else
    statement; ]
end if;
```

Example: 2-to-1 mux

- Style 1



s	f
0	w_0
1	w_1

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s : IN  STD_LOGIC ;
          f          : OUT STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    PROCESS ( w0, w1, s )
    BEGIN
        IF s = '0' THEN
            f <= w0 ;
        ELSE
            f <= w1 ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Example: 2-to-1 mux

- Style 2
- Default assignment
 - $f \leq w0$;
 - This assignment is scheduled to occur after all statements in the PROCESS have been evaluated
 - Default value may be overridden

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s : IN  STD_LOGIC ;
          f          : OUT STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    PROCESS ( w0, w1, s )
    BEGIN
        f <= w0 ;
        IF s = '1' THEN
            f <= w1 ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Sequential statements

- Ordering of statements in a PROCESS can affect the meaning of the code

- Order 1

- Circuit: 2-to-1 mux

- Function: $f = \bar{s}w_0 + sw_1$

```
PROCESS ( w0, w1, s )  
BEGIN
```

```
    f <= w0 ;
```

```
    IF s = '1' THEN
```

```
        f <= w1 ;
```

```
    END IF ;
```

```
END PROCESS ;
```

- Order 2

- Function: $f = w_0$

```
PROCESS ( w0, w1, s )
```

```
BEGIN
```

```
IF s = '1' THEN
```

```
    f <= w1 ;
```

```
END IF ;
```

```
f <= w0 ;
```

```
END PROCESS ;
```


CASE statement

- CASE statement can only appear in a PROCESS
- Syntax

```
case <expression> is
    when <value> =>
        statements;
    when <value> =>
        statements;
    [ when others =>
        statements; ]
end case;
```

Example: 2-to-1 mux

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s : IN    STD_LOGIC ;
          f          : OUT STD_LOGIC ) ;
END mux2to1 ;

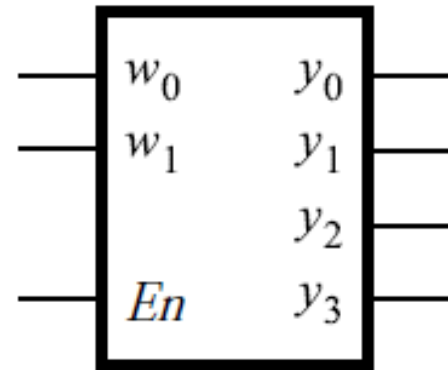
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    PROCESS ( w0, w1, s )
    BEGIN
        CASE s IS
            WHEN '0' =>
                f <= w0 ;
            WHEN OTHERS =>
                f <= w1 ;
        END CASE ;
    END PROCESS ;
END mux2to1 ;
```

Example: 2-to-4 binary decoder

- Truth table

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Truth table



(b) Graphical symbol

Example: 2-to-4 binary decoder

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY dec2to4 IS
    PORT ( w   : IN   STD_LOGIC_VECTOR(1 DOWNT0 0) ;
          En   : IN   STD_LOGIC ;
          y    : OUT  STD_LOGIC_VECTOR(0 TO 3) ) ;
END dec2to4 ;
    
```

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

```

ARCHITECTURE Behavior OF dec2to4 IS
BEGIN
    PROCESS ( w, En )
    BEGIN
        IF En = '1' THEN
            CASE w IS
                WHEN "00" =>
                    y <= "1000" ;
                WHEN "01" =>
                    y <= "0100" ;
                WHEN "10" =>
                    y <= "0010" ;
                WHEN OTHERS =>
                    y <= "0001" ;
            END CASE ;
        ELSE
            y <= "0000" ;
        END IF ;
    END PROCESS ;
END Behavior ;
    
```

Chapter 7 Basic Sequential Circuits

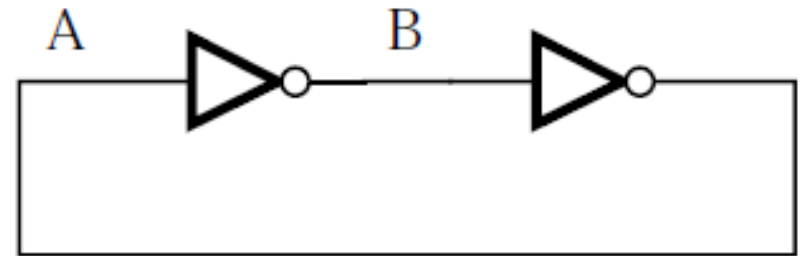
- Latches
- Flip-Flops
- Registers
- Counters

Logic circuits

- Combinational circuits
 - Outputs only depend on present inputs
 - Examples: muxes, demuxes, encoders, decoders, comparators, and adders
- Sequential circuits
 - Outputs depend on present inputs AND past behavior of the circuit
 - Examples: latches, flip-flops, registers, and counters

A memory element

- Storage/memory elements are circuits that store the values of logic signals.
- Two cascaded inverters
- Two possible states
 - $A=0, B=1$
 - $A=1, B=0$
- Not useful: states cannot be changed

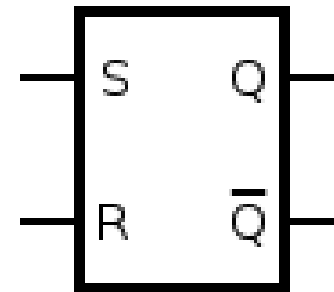
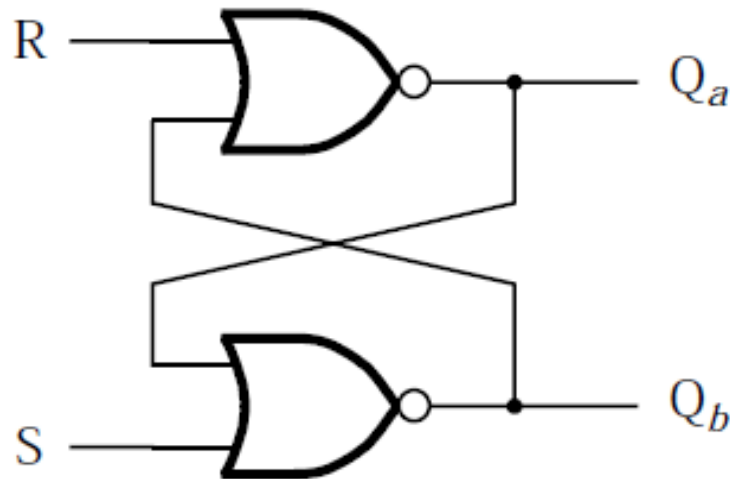


Latches

- A latch is a circuit that can store one bit of information.
- Latches
 - Basic SR latch
 - Gated SR latch
 - Gated D latch

Basic SR latch

- Basic SR latch
 - Circuit: two cross-coupled NOR gates
 - S: set signal
 - R: reset signal
 - Left to right: circuit and circuit symbol



Source: https://en.wikibooks.org/wiki/Digital_Circuits/Latches

Basic SR latch

- Characteristic table
 - $S=0, R=0$: latched/hold/no change
 - $S=0, R=1$: **reset** ($Q_a=0, Q_b=1$)
 - $S=1, R=0$: **set** ($Q_a=1, Q_b=0$)
 - $S=1, R=1$: illegal/not allowed/metastable

S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

