EECE-4740/5740 Advanced VHDL and FPGA Design

Lecture 3
Combinational and Sequential Circuits I

Cristinel Ababei
Marquette University
Department of Electrical and Computer Engineering

1

Overview

- Combinational circuits
  - Multiplexer, decoders, encoders, adders, comparators
- Sequential circuits
  - Regular sequential circuits
  - Finite State Machines

2

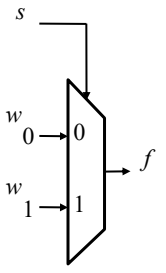
A VHDL Template for Combinational Logic

```
entity model_name is
    port( list of inputs and outputs );
end model_name;

architecture arch_name of model_name is
begin
    concurrent statement 1
    concurrent statement 2
    ...
    concurrent statement N;
end arch_name;
```

3

2-to-1 Multiplexer

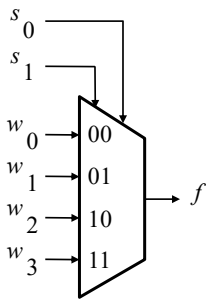
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
    f <= w0 WHEN s = '0' ELSE w1;
END dataflow ;
```



4

4-to-1 Multiplexer

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

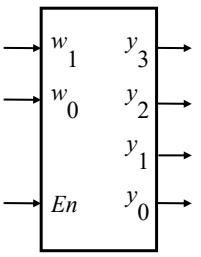


5

2-to-4 Decoder

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

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

ARCHITECTURE dataflow OF dec2to4 IS
      SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
      Enw <= En & w ;
      WITH Enw SELECT
            y <= “0001" WHEN "100",
                 "0010" WHEN "101",
                 "0100" WHEN "110",
                 “1000" WHEN "111",
                 "0000" WHEN OTHERS;
END dataflow;
```

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



6

4-bit Number Comparator: Unsigned

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

ENTITY compare IS
      PORT (A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                  AeqB, AgtB, AltB: OUT STD_LOGIC );
END compare;

ARCHITECTURE dataflow OF compare IS
BEGIN
      AeqB <= '1' WHEN A = B ELSE '0';
      AgtB <= '1' WHEN A > B ELSE '0';
      AltB <= '1' WHEN A < B ELSE '0';
END dataflow;
```

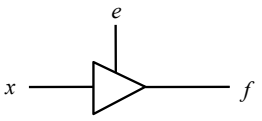
7

4-bit Number Comparator: Signed

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

ENTITY compare IS
      PORT (A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                  AeqB, AgtB, AltB: OUT STD_LOGIC );
END compare;

ARCHITECTURE dataflow OF compare IS
BEGIN
      AeqB <= '1' WHEN  A = B ELSE '0';
      AgtB  <= '1' WHEN  A > B ELSE '0';
      AltB  <=  '1'  WHEN  A < B ELSE '0';
END dataflow;
```

8

Tri-state Buffer

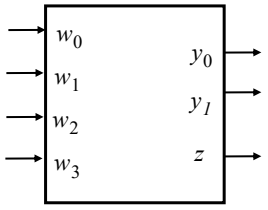


e	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1

```
ENTITY tri_state IS
  PORT ( e: IN STD_LOGIC;
         x: IN STD_LOGIC;
         f: OUT STD_LOGIC);
END tri_state;

ARCHITECTURE dataflow OF tri_state IS
BEGIN
        f <= x WHEN (e = '1') ELSE 'Z';
END dataflow;
```

9

Priority Encoder



w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

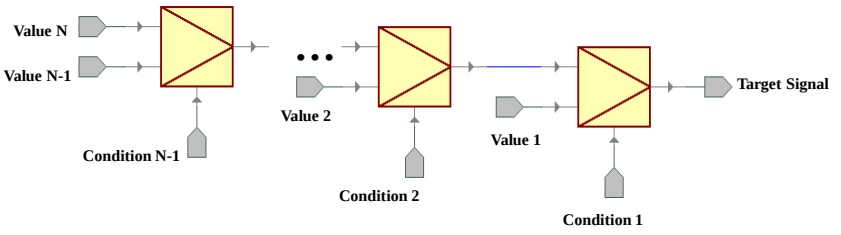
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY priority IS
      PORT ( w: IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
             y: OUT STD_LOGIC_VECTOR(1 DOWNTO 0) ;
             z: OUT STD_LOGIC ) ;
END priority ;

ARCHITECTURE dataflow OF priority IS
BEGIN
      y <=    "11" WHEN w(3) = '1' ELSE
              "10" WHEN w(2) = '1' ELSE
              "01" WHEN w(1) = '1' ELSE
              "00" ;
      z <= '0' WHEN w = "0000" ELSE '1' ;
END dataflow ;
```

10

Most often implied structure

When - Else

```
target_signal <= value1 when condition1 else
                 value2 when condition2 else
                    . . .
                 valueN-1 when conditionN-1 else
                 valueN;
```

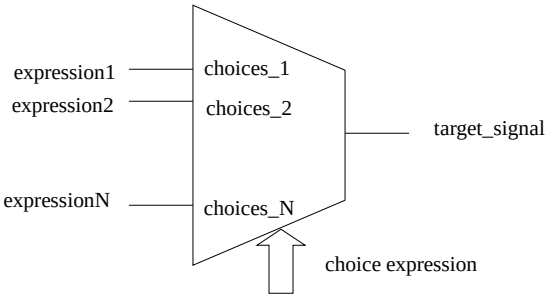


11

Most often implied structure

With - Select - When

```
with choice_expression select
   target_signal <= expression1 when choices_1,
                    expression2 when choices_2,
                       . . .
                    expressionN when choices_N;
```



12

Overview

- Combinational circuits
  - Multiplexer, decoders, encoders, adders, comparators
- Sequential circuits
  - Regular sequential circuits
  - Finite State Machines

13

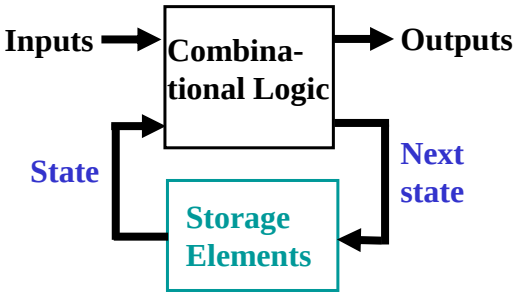
Sequential Circuits

- Regular sequential circuits
  - Sequential circuits
  - Storage elements: Latches & Flip-flops
  - Registers and counters
- Circuit and System Timing
- Finite State Machines (FSMs)
  - State tables & state diagrams

14

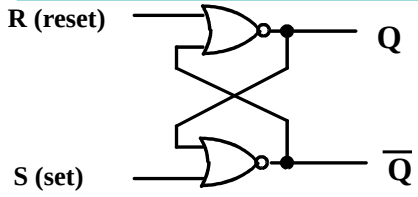
Sequential circuits – general description

- A Sequential circuit contains:
  - Storage elements: Latches or Flip-Flops
  - Combinational Logic: implements a multiple-output switching function
    - *Next state function:* Next State = *f*(Inputs, State)
    - *Output function: two types*
      Mealy: Outputs = *g*(Inputs, State)
      Moore: Outputs = *h*(State)

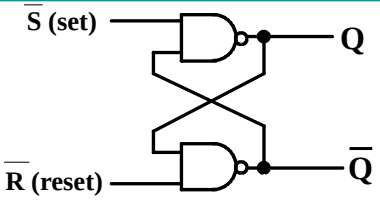


15

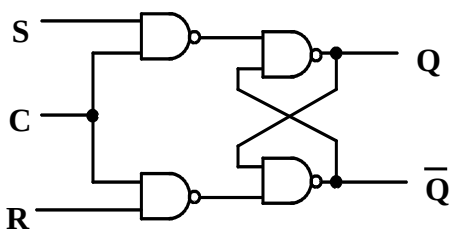
Latches



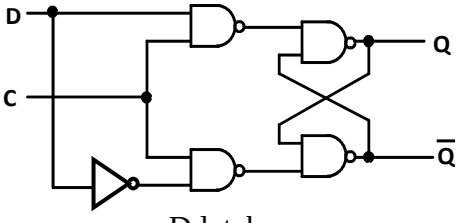
Basic S-R latch



Basic $\overline{S}$-$\overline{R}$ latch



Clocked S-R latch

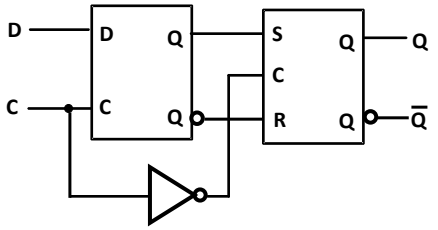


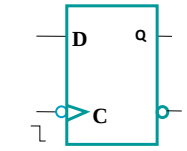
D latch

16

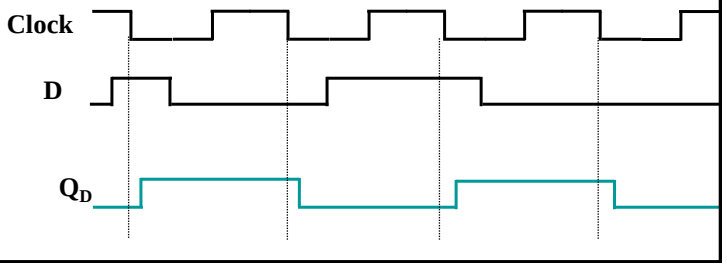
Edge-Triggered D Flip-Flop

- The change of Q is associated with the negative edge at the end of the pulse - *negative-edge triggered* flip-flop.





Triggered D



17

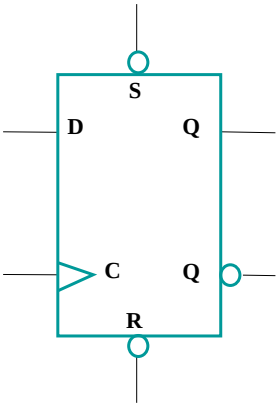
Modelling of Flip-Flops

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity FLOP is
   port (D, CLK : in std_logic;
         Q : out std_logic);
end FLOP;

architecture A of FLOP is
begin
  process
  begin
      wait until CLK'event and CLK='0';
          Q <= D;
   end process;
end A;
```

18

Direct Inputs

- At power-up or at reset, sequential circuit usually is initialized to a known state before it begins operation
  - Done outside of the clocked behavior of the circuit (i.e., asynchronously).
  - Direct R/S inputs
- For the example flip-flop shown
  - 0 applied to R resets the flip-flop to the 0 state
  - 0 applied to S sets the flip-flop to the 1 state



19

Positive Edge-triggered D Flip-flop with Asynchronous Set/Reset

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ASYNC_FF is
   port (D, CLK, SETN, RSTN : in std_logic;
         Q : out std_logic);
end ASYNC_FF;

architecture RTL of ASYNC_FF is
begin
    process (CLK, RSTN, SETN)
    begin
        if (RSTN = `1`) then
              Q <= `0`;
        elsif SETN ='1' then
              Q <= '1';
        elsif (CLK'event and CLK = '1') then
              Q <= D;
        end if;
   end process;
end RTL;
```

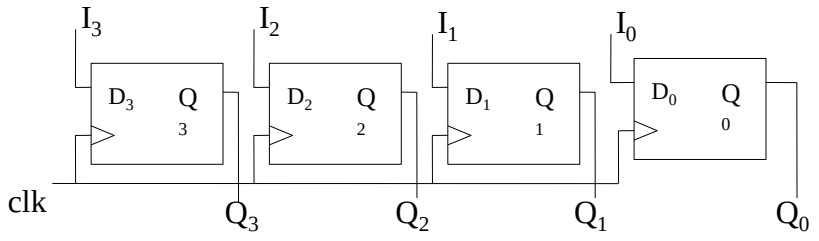
20

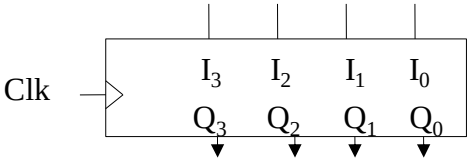
Registers

- Register: the simplest storage component in a computer, a bit-wise extension of a flip-flop.
- Registers can be classified into
  - Simple Registers
  - Parallel-Load Registers
  - Shift Registers

21

Simple Registers

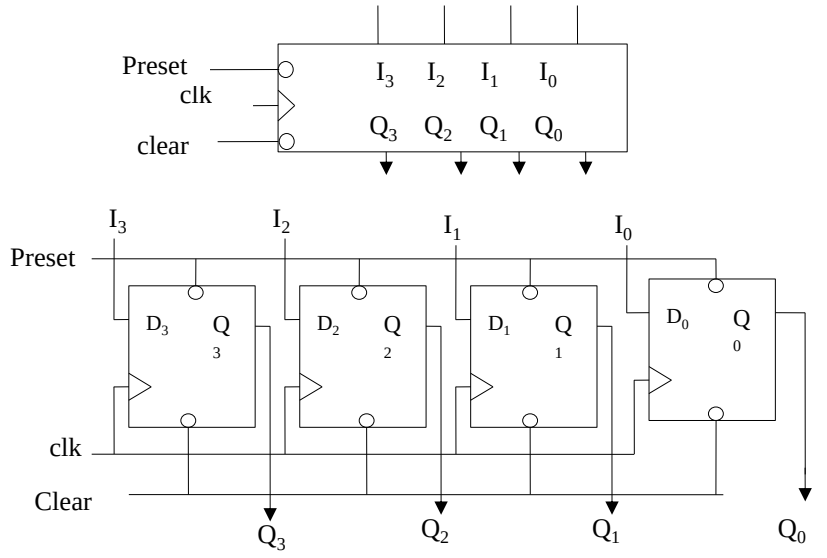




- A simple register consists of *N* flip-flops driven by a common clock signal.
- Has *N* inputs and *N* outputs in addition to the clock signal.

22

Register with asynchronous preset and clear



23

```
Library ieee;
USE ieee.std_logic_1164.all;
ENTITY simple_register IS
   GENERIC ( N : INTEGER := 4);
   PORT ( I : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
          Clock, Clear, Preset : IN  STD_LOGIC;
          Q : OUT STD_LOGIC_VECTOR (N-1 DOWNTO 0));
END simple_register;

ARCHITECTURE simple_memory OF simple_register IS
BEGIN
PROCESS (Preset, Clear, Clock)
BEGIN
       IF Preset = '0' THEN
              Q <= (OTHERS => '1');
       ELSIF Clear = '0' THEN
             Q <= (OTHERS => '0');
       ELSIF (Clock'EVENT AND Clock = '1') THEN
             Q <= I;
       END IF;
END PROCESS;
END simple_memory;
```

24

Parallel Load Registers

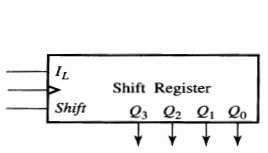
- In the previous registers, new data is stored automatically on every rising edge of the clock.
- In most digital systems, the data is stored for several clock cycles before it is rewritten. For this reason it is useful to be able to control WHEN the data will be entered into a register.
  - Use a control signal called Load or Enable. This allows loading into a register known as a parallel-load register.

25

```
Library ieee;
USE ieee.std_logic_1164.all;
ENTITY load_enable IS
  GENERIC ( N : INTEGER := 4);
  PORT ( D : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
         Clock, Resetn, load : IN STD_LOGIC;
         Q : BUFFER STD_LOGIC_VECTOR (N-1 DOWNTO 0));
END load_enable;

ARCHITECTURE rtl OF load_enable IS
   SIGNAL state : std_logic_vector(N-1 DOWNTO 0);
BEGIN
   PROCESS (Resetn, Clock) IS
   BEGIN
         IF Resetn = '0' THEN
            state <= (OTHERS => '0');
         ELSIF (Clock'EVENT AND Clock = '1') THEN
            IF load = '1' THEN
                  state <= D;
            ELSE
                  state <= state;
            END IF;
         END IF;
   END PROCESS;

   Q <= state;
END rtl;
```

26

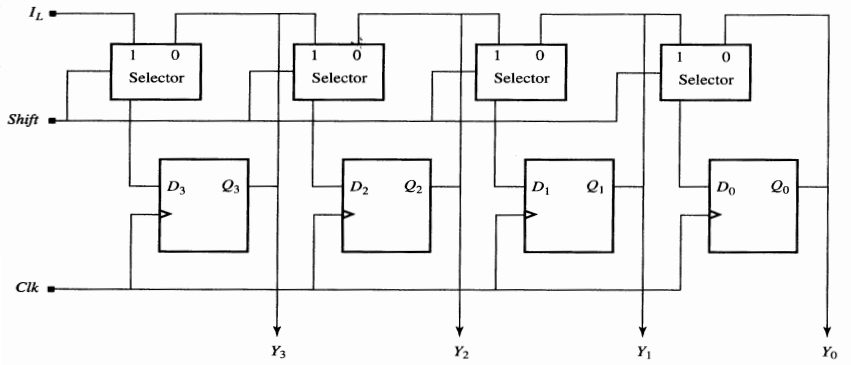
Serial-in/Parallel-out Shift Register



(a) Graphic symbol

PRESENT STATE	NEXT STATE
Shift	Q_3 Q_2 Q_1 Q_0
0	No change
1	I_L Q_3 Q_2 Q_1

(b) Operation table



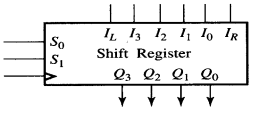
(c) Register schematic

27

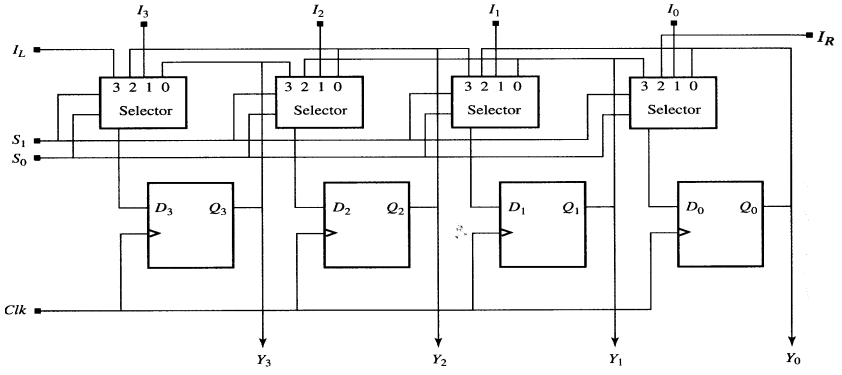
Parallel Load Shift Register

PRESENT STATE			NEXT STATE			
S_1	S_0	OPERATION	Q_3	Q_2	Q_1	Q_0
0	0	No change	Q_3	Q_2	Q_1	Q_0
0	1	Load input	I_3	I_2	I_1	I_0
1	0	Shift left	Q_2	Q_1	Q_0	I_R
1	1	Shift right	I_L	Q_3	Q_2	Q_1

(b) Operation table



(a) Graphic symbol



(c) Register schematic

28

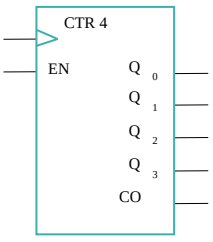
VHDL for Up-Counter

```
Library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY upcount IS
   GENERIC ( N : INTEGER := 4 );
   PORT ( Clock, Resetn, Enable : IN STD_LOGIC;
          Q: BUFFER STD_LOGIC_VECTOR (N-1 DOWNTO 0));
END upcount;

ARCHITECTURE cnt OF upcount IS
SIGNAL count : STD_LOGIC_VECTOR (N-1 DOWNTO 0);

BEGIN
PROCESS (Resetn, Clock)
BEGIN
        IF Resetn = '0' THEN
                count <= ( OTHERS => '0' );
                -- Use of others in aggregate makes your code generic.
                -- Thus you won't have to replace all "00000000" with
                -- "00" when you change your mind and vectors should
                -- be only 2 bits wide.
        ELSIF (Clock'EVENT AND Clock = '1') THEN
                IF Enable = '1' THEN
                        count <= count +1;
                ELSE
                        count <= count;
                END IF;
        END IF;
END PROCESS;

Q <= count;
END cnt;
```

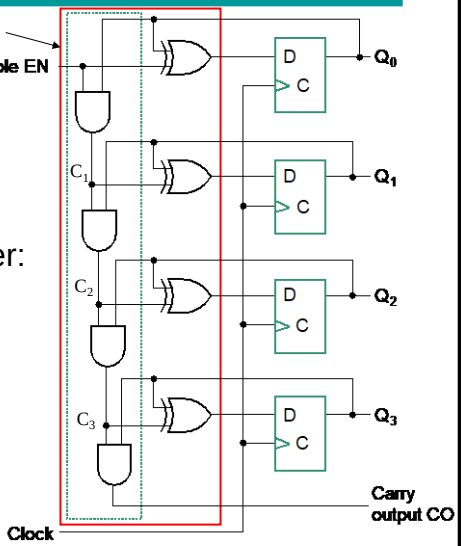
29

Synchronous Counters

- Internal logic
  - Incrementer: Q+0 or Q+1
- Contraction of a ripple carry adder with one operant fixed at 000X
- Symbol for synchronous counter:



Symbol

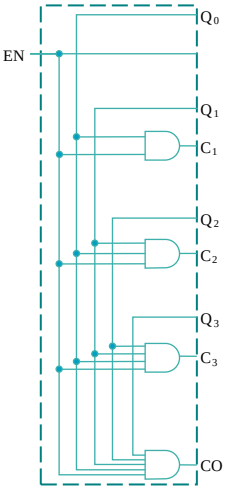


(a) Logic Diagram-Serial Gating

30

Synchronous Counters (Contd.)

- Contraction of carry-lookahead adder
  - Reduce path delays
  - Called parallel gating
  - Lookahead can be used on COs and ENs to prevent long paths in large counters



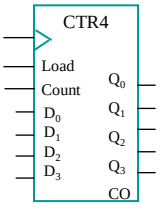
Logic Diagram-Parallel Gating

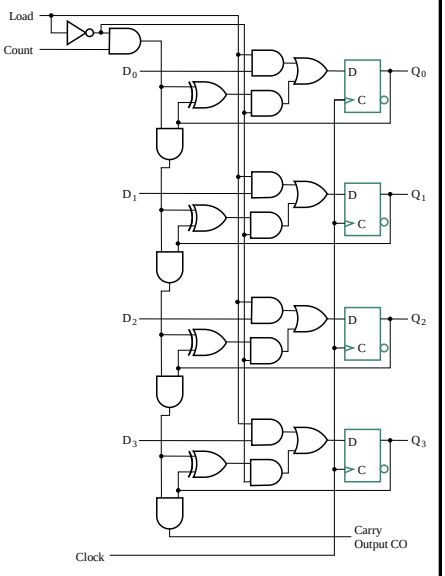
31

Counter with Parallel Load

Load	Count	Action
0	0	Hold Stored Value
0	1	Count Up Stored Value
1	X	Load D

- Add path for input data D
  - enabled for Load = 1
- Add logic to:
  - When Load = 1 disable count logic (feedback from outputs)
  - When Load = 0 and Count = 1 enable count logic





32

BCD Counter

```
architecture Behavioral of bcd_counter is
   signal regcnt : std_logic_vector(3 downto 0);
begin
   count: process (reset, clk) is
   begin
      if ( reset='1' ) then
          regcnt <= "0000";
       elsif ( clk'event and clk='1') then
          regcnt <= regcnt+1;
          if (regcnt = "1001") then
              regcnt <= "0000";
          end if;
      end if;
    end process;
end Behavioral;
```

33

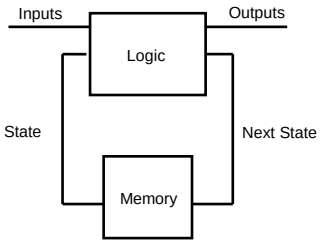
Overview

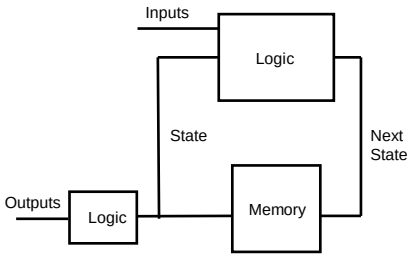
- Combinational circuits
  - Multiplexer, decoders, encoders, adders, comparators
- Sequential circuits
  - Regular sequential circuits
  - Finite State Machines (FSMs)

34

FSM types

- MEALY machine:
  - Outputs are dependent on current state and inputs
- MOORE machine:
  - Outputs are dependent on current state only



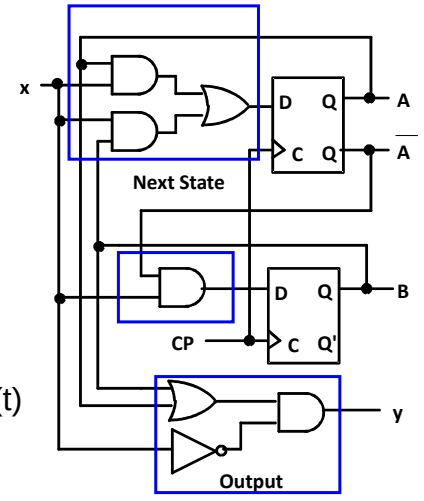


35

Example 1

- Input: x(t)
- Output: y(t)
- State: (A(t), B(t))

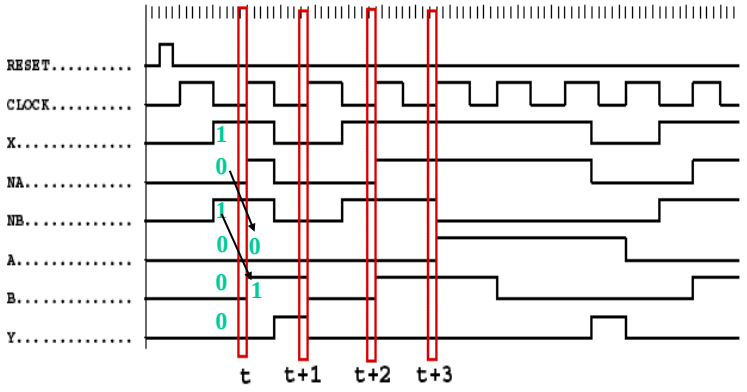
- Output Function?
  - $y(t) = \overline{x}(t)(B(t) + A(t))$
- Next State Function?
  - $A(t+1) = A(t)x(t) + B(t)x(t)$
  - $B(t+1) = \overline{A}(t)x(t)$



36

Example 1 (Contd.)

- Where in time are inputs, outputs and states defined?



$y(t) = \overline{x(t)}(B(t) + A(t))$

$A(t+1) = A(t)x(t) + B(t)x(t)$
$B(t+1) = \overline{A(t)}x(t)$

37

Typical Design Procedure for Sequential Circuits

- Formulation: Construct a state table or state diagram
- State Assignment: Assign binary codes to the states
- Flip-Flop Input Equation Determination: Select flip-flop types, derive flip-flop input equations from next state entries in the table
- Output Equation Determination: Derive output equations from output entries in the table
- Optimization - Optimize the equations
- Technology Mapping - Find circuit from equations and map to flip-flops and gate technology
- Verification - Verify correctness of final design

38

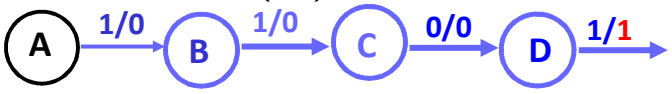
Example 2: Sequence Recognizer

- A sequence recognizer: produces an output '1' whenever a prescribed pattern of inputs occur in sequence
- Steps:
  - Begin in an initial state (typically "reset" state), when NONE of the initial portion of the sequence has occurred
  - Add states
    - That recognize each successive symbol occurring
    - The final state represents the input sequence occurrence
  - Add state transition arcs which specify what happens when a symbol **not** in the proper sequence has occurred
  - Add other arcs on non-sequence inputs which transition to states
    - The last step is required because the circuit must recognize the input sequence regardless of where it occurs within the overall sequence applied since "reset"

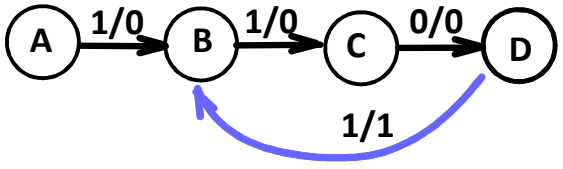
39

Example 2: Recognize 1101 as Mealy machine

- Define states for the sequence to be recognized
- Starting in the initial state ("A"):



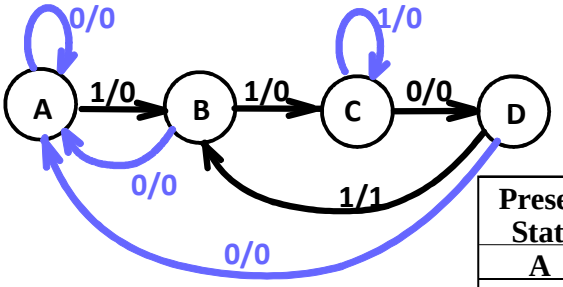
- Finally, output 1 on the arc from D means the sequence has been recognized,
  - To what state should the arc from state D go? Remember: 1101<u>1101</u> ?
  - The final 1 in the recognized sequence 1101 is a sub-sequence of 1101. It follows a 0 which is not a sub-sequence of 1101. Thus it should represent the same state reached from the initial state after a first 1 is observed.



40

Example 2: Recognize 1101 (Contd.)

- The other arcs are added to each state for inputs not yet listed. Which arcs are missing?
  - State transition arcs must represent the fact that an input subsequence has occurred.
  - Note that the 1 arc from state C back to C implies that State C means *two or more 1's have occurred*.

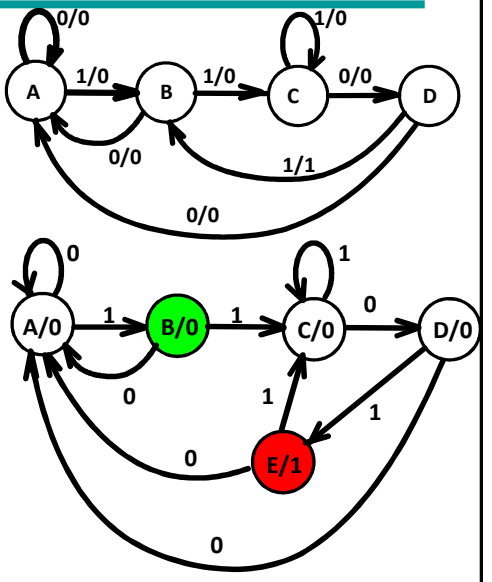


Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

41

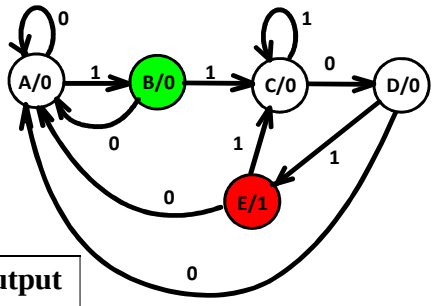
Example 2: Recognize 1101 as Moore machine

- **For Moore Model, outputs are associated with states.** Arcs now show only state transitions
- Add a new state E to produce the output 1
  - State E produces the same behavior in the future as state B, but it gives a different output at the present time. Thus these states do represent a *different abstraction* of the input history.
- The Moore model for a sequence recognizer usually has *more states* than the Mealy model.



42

Example 2: Moore Model (Contd.)

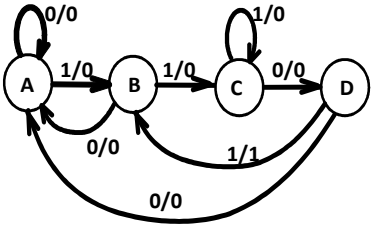


Present State	Next State x=0	Next State x=1	Output y
A	A	B	0
B	A	C	0
C	D	C	0
D	A	E	0
E	A	C	1

43

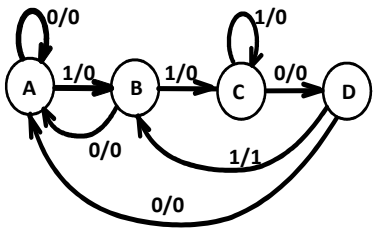
VHDL code using 3 processes: sequential recognizer

```
library ieee;
use ieee.std_logic_1164.all;

entity seq_rec_MEALY is
 port (CLK, RESET, X: in std_logic;
       Z: out std_logic);
end seq_rec;

architecture process_3 of seq_rec_MEALY is
  type state_type is (A, B, C, D);
  signal state, next_state: state_type;

begin
```



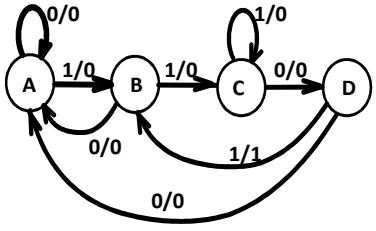
44

```
-- process 1: implements positive edge-triggered
-- flipflop with asynchronous reset
state_register: process (CLK, RESET)
begin
 if (RESET = '1') then
  state <= A;
 elsif (CLK'event and CLK = '1') then
  state <= next_state;
 end if;
end process;

-- process 2: implement output as function
-- of input X and state
output_function: process (X, state)
begin
 case state is
  when A => Z <= '0';
  when B => Z <= '0';
  when C => Z <= '0';
  when D => if X = '1' then Z <= '1';
            else Z <= '0';
            end if;
  end case;
end process;
```



45

```
-- process 3: next state-function implemented
-- as a function of input X and state
next_state_function: process (X, state)
begin
 case state is
   when A =>
     if X = '1' then next_state <= B;
     else next_state <= A;
     end if;
   when B =>
     if X = '1' then next_state <= C;
     else next_state <= A;
     end if;
   when C =>
     if X = '1' then next_state <= C;
     else next_state <= D;
     end if;
   when D =>
     if X = '1' then next_state <= B;
     else next_state <= A;
     end if;
 end case;
end process;

end architecture;
```



46

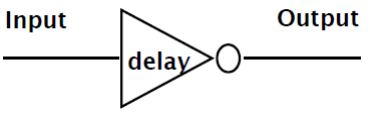
Summary

- Combinational circuits and regular sequential circuits are somewhat easier to describe in VHDL.
- Manual design of Finite State Machines becomes difficult for complex circuits. Automated tools come in handy in this case.

47

Appendix A: VHDL Delay Models

- Delay is created by scheduling a signal assignment for a future time
- Delay in a VHDL cycle can be of several types:
  - Inertial
  - Transport
  - Delta

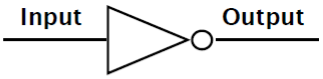


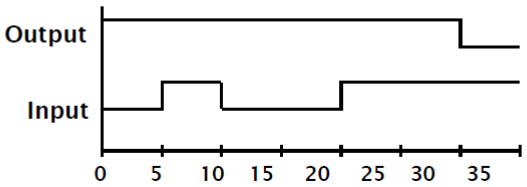
48

Inertial Delay

- Default delay type
- Allows for user specified delay
- Absorbs pulses of shorter duration than the specified delay

```
-- Inertial is the default
Output <= NOT Input AFTER 10 ns;
```



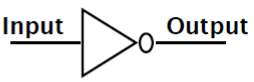


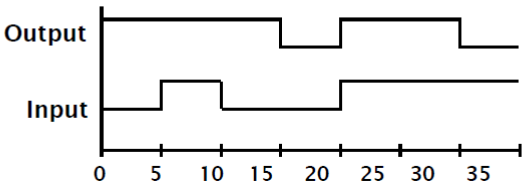
49

Transport Delay

- Must be explicitly specified by user
- Allows for user specified delay
- Passes all input transitions with delay

```
-- TRANSPORT must be specified
Output <= TRANSPORT NOT Input AFTER 10 ns;
```





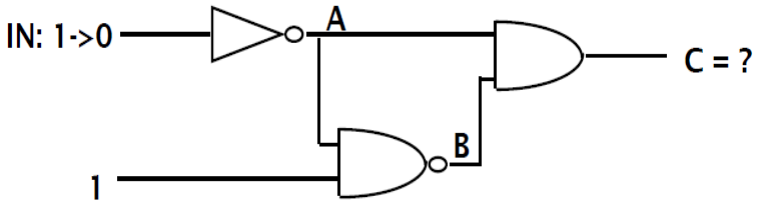
50

Delta Delay

- Delta delay needed to provide support for concurrent operations with zero delay
  - The order of execution for components with zero delay is not clear
- Scheduling of zero delay devices requires the delta delay
  - A delta delay is necessary if no other delay is specified
  - A delta delay does not advance simulator time or real/wall clock time
  - One delta delay is an infinitesimal amount of time
  - The delta is a scheduling device to ensure repeatability (or determinism)

51

Example – Delta Delay



Using delta delay scheduling

Time	Delta	Event
0 ns	1	IN: 1->0
		eval inverter
	2	A: 0->1
		eval NAND, AND
	3	B: 1->0
		C: 0->1
		eval AND
	4	C: 1->0
1 ns		

52