# COL215 Digital Logic and System Design

Designing synchronous sequential circuits

25 August 2016

# Outline

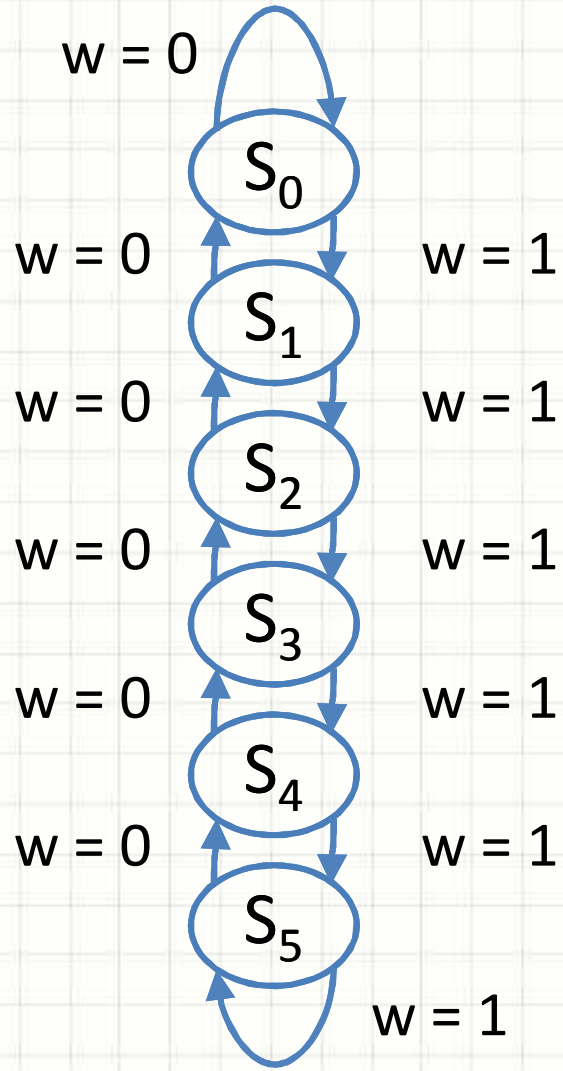
- Saturating up/down counter
- Arbiter example
- FSMs in VHDL
- Moore and Mealy models

# Saturating up-down counter

Design a synchronous up-down counter with the following specifications

- It has one input w and one 3-bit output Z
- Output Z is the count
- Input w decides whether count goes up (w = 1) or down (w = 0)
- The count saturates at 0 while counting down and at 5 while counting up.
- Initially, the count is 0

# State transition diagram



# State transition table

| Present state | Next state | | Output z |
|---|---|---|---|
| | w = 0 | w = 1 | |
| $S_0$ | $S_0$ | $S_1$ | 0 0 0 |
| $S_1$ | $S_0$ | $S_2$ | 0 0 1 |
| $S_2$ | $S_1$ | $S_3$ | 0 1 0 |
| $S_3$ | $S_2$ | $S_4$ | 0 1 1 |
| $S_4$ | $S_3$ | $S_5$ | 1 0 0 |
| $S_5$ | $S_4$ | $S_5$ | 1 0 1 |

# State transition table

state encoding is trivial here

| Present state | Next state | | Output z |
|---|---|---|---|
| | w = 0 | w = 1 | |
| 0 0 0 | 0 0 0 | 0 0 1 | 0 0 0 |
| 0 0 1 | 0 0 0 | 0 1 0 | 0 0 1 |
| 0 1 0 | 0 0 1 | 0 1 1 | 0 1 0 |
| 0 1 1 | 0 1 0 | 1 0 0 | 0 1 1 |
| 1 0 0 | 0 1 1 | 1 0 1 | 1 0 0 |
| 1 0 1 | 1 0 0 | 1 0 1 | 1 0 1 |

present state = $y_2\, y_1\, y_0$

next state = $Y_2\, Y_1\, Y_0$

output = $z_2\, z_1\, z_0$

= $y_2\, y_1\, y_0$

# Implementation using D FFs

| $y_2\ y_1\ y_0$ | $Y_2$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 0 |
| 0 1 0 | 0 | 0 |
| 0 1 1 | 0 | 1 |
| 1 0 0 | 0 | 1 |
| 1 0 1 | 1 | 1 |

| $y_2\ y_1\ y_0$ | $Y_1$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 1 |
| 0 1 0 | 0 | 1 |
| 0 1 1 | 1 | 0 |
| 1 0 0 | 1 | 0 |
| 1 0 1 | 0 | 0 |

| $y_2\ y_1\ y_0$ | $Y_0$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 1 |
| 0 0 1 | 0 | 0 |
| 0 1 0 | 1 | 1 |
| 0 1 1 | 0 | 0 |
| 1 0 0 | 1 | 1 |
| 1 0 1 | 0 | 1 |

$Y_2 = y_2\, y_1'y_0 + w\, y_2\, y_1' + w\, y_2'y_1\, y_0$

$Y_1 = w'y_2\, y_1'y_0' + w'y_2'y_1\, y_0 + w\, y_2'y_1\, y_0' + w\, y_2'y_1'y_0$

$Y_0 = y_2\, y_1'y_0' + y_2'y_1\, y_0' + w\, y_2\, y_1' + w\, y_2'y_0'$

# Implementation using T FFs

| $y_2\ y_1\ y_0$ | $T_2$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 0 |
| 0 1 0 | 0 | 0 |
| 0 1 1 | 0 | 1 |
| 1 0 0 | 1 | 0 |
| 1 0 1 | 0 | 0 |

| $y_2\ y_1\ y_0$ | $T_1$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 1 |
| 0 1 0 | 1 | 0 |
| 0 1 1 | 0 | 1 |
| 1 0 0 | 1 | 0 |
| 1 0 1 | 0 | 0 |

| $y_2\ y_1\ y_0$ | $T_0$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 1 |
| 0 0 1 | 1 | 1 |
| 0 1 0 | 1 | 1 |
| 0 1 1 | 1 | 1 |
| 1 0 0 | 1 | 1 |
| 1 0 1 | 1 | 0 |

$T_2 = w' y_2\ y_1' y_0' + w\ y_2' y_1\ y_0$

$T_1 = w' y_2\ y_1' y_0' + w' y_2' y_1\ y_0' + w\ y_2' y_0$

$T_0 = y_2' y_0 + y_2' y_1 + w' y_2\ y_1' + w\ y_1' y_0'$

# Implementation using JK FFs

| $y_2\ y_1\ y_0$ | $J_2K_2$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 - | 0 - |
| 0 0 1 | 0 - | 0 - |
| 0 1 0 | 0 - | 0 - |
| 0 1 1 | 0 - | 1 - |
| 1 0 0 | - 1 | - 0 |
| 1 0 1 | - 0 | - 0 |

| $y_2\ y_1\ y_0$ | $J_1K_1$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 - | 0 - |
| 0 0 1 | 0 - | 1 - |
| 0 1 0 | - 1 | - 0 |
| 0 1 1 | - 0 | - 1 |
| 1 0 0 | 1 - | 0 - |
| 1 0 1 | 0 - | 0 - |

| $y_2\ y_1\ y_0$ | $J_0K_0$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 - | 1 - |
| 0 0 1 | - 1 | - 1 |
| 0 1 0 | 1 - | 1 - |
| 0 1 1 | - 1 | - 1 |
| 1 0 0 | 1 - | 1 - |
| 1 0 1 | - 1 | - 0 |

$J_2 = w\, y_2' y_1 y_0$

$J_1 = w' y_2 y_1' y_0' + w\, y_2' y_0$

$J_0 = w\, y_2' + y_2 y_1' + y_2' y_1$

$K_2 = w' y_1' y_0'$

$K_1 = w' y_2' y_0' + w\, y_2' y_0$

$K_0 = y_2' + w' y_1'$

# D FF implementation with don't cares

| $y_2\ y_1\ y_0$ | $Y_2$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 0 |
| 0 1 0 | 0 | 0 |
| 0 1 1 | 0 | 1 |
| 1 0 0 | 0 | 1 |
| 1 0 1 | 1 | 1 |
| 1 1 0 | - | - |
| 1 1 1 | - | - |

| $y_2\ y_1\ y_0$ | $Y_1$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 1 |
| 0 1 0 | 0 | 1 |
| 0 1 1 | 1 | 0 |
| 1 0 0 | 1 | 0 |
| 1 0 1 | 0 | 0 |
| 1 1 0 | - | - |
| 1 1 1 | - | - |

| $y_2\ y_1\ y_0$ | $Y_0$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 1 |
| 0 0 1 | 0 | 0 |
| 0 1 0 | 1 | 1 |
| 0 1 1 | 0 | 0 |
| 1 0 0 | 1 | 1 |
| 1 0 1 | 0 | 1 |
| 1 1 0 | - | - |
| 1 1 1 | - | - |

$Y_2 = y_2\, y_0 + w\, y_2 + w\, y_1\, y_0$

$Y_1 = w' y_2\, y_0' + w' y_1\, y_0 + w\, y_1\, y_0' + w\, y_2' y_1' y_0$

$Y_0 = y_2\, y_0' + y_1\, y_0' + w\, y_2 + w\, y_0'$

# T FF implementation with don't cares

| $y_2\ y_1\ y_0$ | $T_2$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 0 |
| 0 1 0 | 0 | 0 |
| 0 1 1 | 0 | 1 |
| 1 0 0 | 1 | 0 |
| 1 0 1 | 0 | 0 |
| 1 1 0 | - | - |
| 1 1 1 | - | - |

| $y_2\ y_1\ y_0$ | $T_1$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 1 |
| 0 1 0 | 1 | 0 |
| 0 1 1 | 0 | 1 |
| 1 0 0 | 1 | 0 |
| 1 0 1 | 0 | 0 |
| 1 1 0 | - | - |
| 1 1 1 | - | - |

| $y_2\ y_1\ y_0$ | $T_0$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 | 1 |
| 0 0 1 | 1 | 1 |
| 0 1 0 | 1 | 1 |
| 0 1 1 | 1 | 1 |
| 1 0 0 | 1 | 1 |
| 1 0 1 | 1 | 0 |
| 1 1 0 | - | - |
| 1 1 1 | - | - |

$T_2 = w' y_2 y_0' + w\, y_1 y_0$

$T_1 = w' y_2 y_0' + w' y_1 y_0' + w\, y_2' y_0$

$T_0 = y_2' y_0 + y_2' y_1 + w' y_2 + w\, y_0'$

# JK FF implementation with don't cares

| $y_2\ y_1\ y_0$ | $J_2K_2$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 - | 0 - |
| 0 0 1 | 0 - | 0 - |
| 0 1 0 | 0 - | 0 - |
| 0 1 1 | 0 - | 1 - |
| 1 0 0 | - 1 | - 0 |
| 1 0 1 | - 0 | - 0 |
| 1 1 0 | - - | - - |
| 1 1 1 | - - | - - |

| $y_2\ y_1\ y_0$ | $J_1K_1$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 - | 0 - |
| 0 0 1 | 0 - | 1 - |
| 0 1 0 | - 1 | - 0 |
| 0 1 1 | - 0 | - 1 |
| 1 0 0 | 1 - | 0 - |
| 1 0 1 | 0 - | 0 - |
| 1 1 0 | - - | - - |
| 1 1 1 | - - | - - |

| $y_2\ y_1\ y_0$ | $J_0K_0$ | |
|---|---|---|
| | w' | w |
| 0 0 0 | 0 - | 1 - |
| 0 0 1 | - 1 | - 1 |
| 0 1 0 | 1 - | 1 - |
| 0 1 1 | - 1 | - 1 |
| 1 0 0 | 1 - | 1 - |
| 1 0 1 | - 1 | - 0 |
| 1 1 0 | - - | - - |
| 1 1 1 | - - | - - |

$J_2 = w\ y_1\ y_0$

$J_1 = w'y_2\ y_0' + w\ y_2'y_0$

$J_0 = w + y_2 + y_1$

$K_2 = w'y_0'$

$K_1 = w'y_0' + w\ y_0$

$K_0 = y_2' + w'$

# What about states “110” and “111”?

| $y_2\ y_1\ y_0$ | $Y_2$ | |
|---|---|---|
| | w’ | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 0 |
| 0 1 0 | 0 | 0 |
| 0 1 1 | 0 | 1 |
| 1 0 0 | 0 | 1 |
| 1 0 1 | 1 | 1 |
| 1 1 0 | 0 | 1 |
| 1 1 1 | 1 | 1 |

| $y_2\ y_1\ y_0$ | $Y_1$ | |
|---|---|---|
| | w’ | w |
| 0 0 0 | 0 | 0 |
| 0 0 1 | 0 | 1 |
| 0 1 0 | 0 | 1 |
| 0 1 1 | 1 | 0 |
| 1 0 0 | 1 | 0 |
| 1 0 1 | 0 | 0 |
| 1 1 0 | 1 | 1 |
| 1 1 1 | 1 | 0 |

| $y_2\ y_1\ y_0$ | $Y_0$ | |
|---|---|---|
| | w’ | w |
| 0 0 0 | 0 | 1 |
| 0 0 1 | 0 | 0 |
| 0 1 0 | 1 | 1 |
| 0 1 1 | 0 | 0 |
| 1 0 0 | 1 | 1 |
| 1 0 1 | 0 | 1 |
| 1 1 0 | 1 | 1 |
| 1 1 1 | 0 | 1 |

$Y_2 = y_2\, y_0 + w\, y_2 + w\, y_1\, y_0$

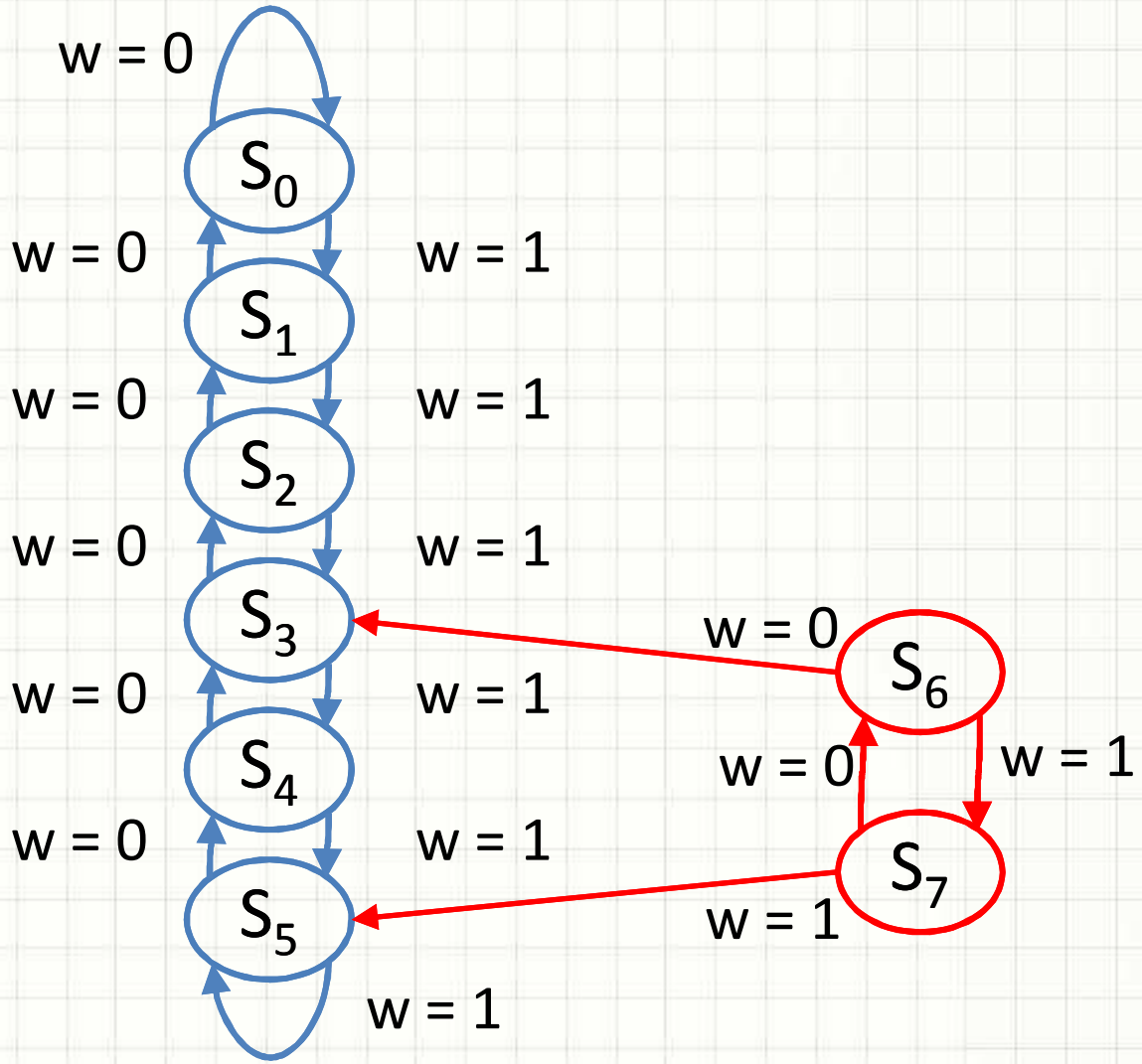
$Y_1 = w' y_2\, y_0' + w' y_1\, y_0 + w\, y_1\, y_0' + w\, y_2' y_1' y_0$

$Y_0 = y_2\, y_0' + y_1\, y_0' + w\, y_2 + w\, y_0'$

# State transition table

| Present state | Next state | | Output z |
|---|---|---|---|
| | w = 0 | w = 1 | |
| 0 0 0 | 0 0 0 | 0 0 1 | 0 0 0 |
| 0 0 1 | 0 0 0 | 0 1 0 | 0 0 1 |
| 0 1 0 | 0 0 1 | 0 1 1 | 0 1 0 |
| 0 1 1 | 0 1 0 | 1 0 0 | 0 1 1 |
| 1 0 0 | 0 1 1 | 1 0 1 | 1 0 0 |
| 1 0 1 | 1 0 0 | 1 0 1 | 1 0 1 |
| 1 1 0 | 0 1 1 | 1 1 1 | 1 1 0 |
| 1 1 1 | 1 1 0 | 1 0 1 | 1 1 1 |

# State transition diagram



# Revisiting arbiter of 3 port switch

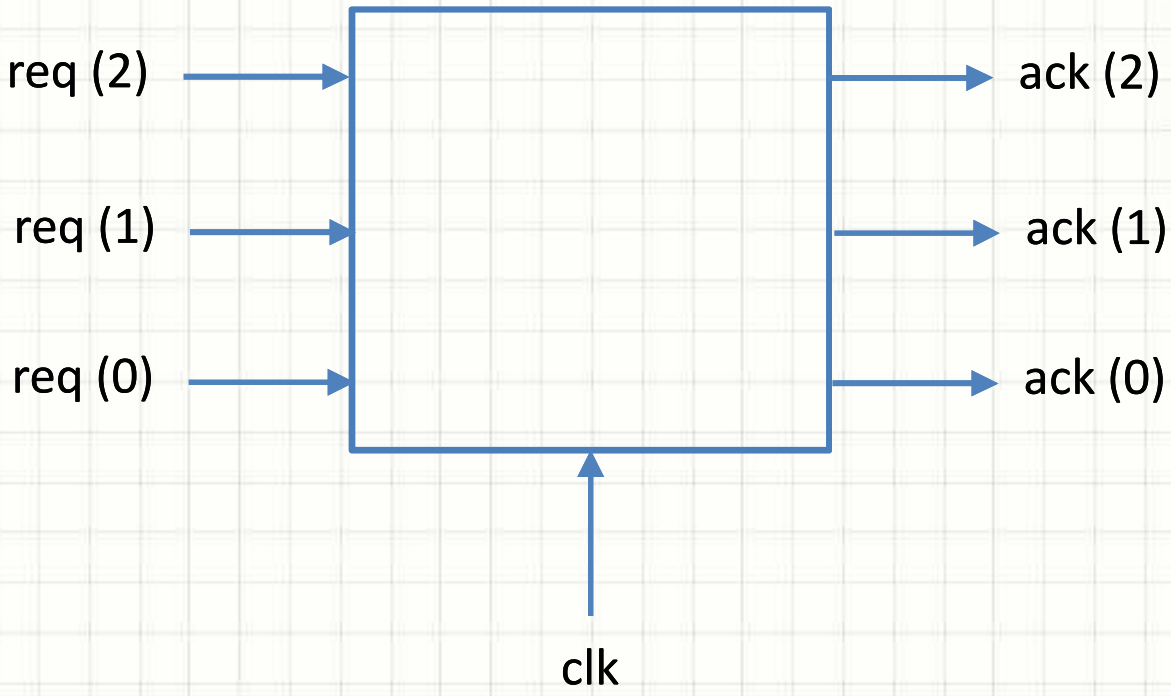
## Previous design :

- Arbitration done by priority encoder
- Fixed priorities
- Connections through low priority ports get interrupted

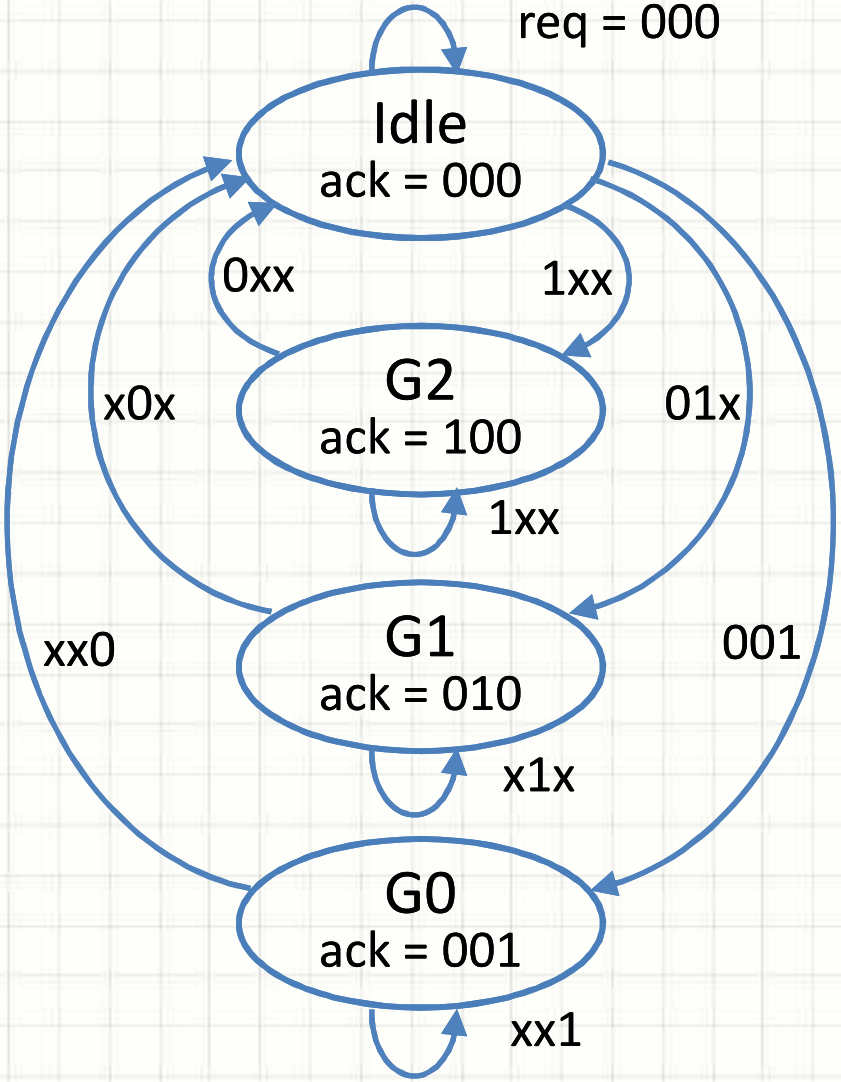
## Improved design :

- Prevents interruption
- It remembers which port is already communicating
- Sequential circuit rather than combinational

# Arbiter



# State transition diagram



# Design in VHDL - entity

```
ENTITY arbiter IS
   PORT (clk : IN bit;
         req : IN bit_vector (2 DOWNTO 0);
         ack : OUT bit_vector (2 DOWNTO 0)
        );
END arbiter;
```

# Design in VHDL - architecture

```
ARCHITECTURE FSM OF arbiter IS
   TYPE state_type IS (Idle, G0, G1, G2);
   SIGNAL state : state_type;
BEGIN
   PROCESS (clk)
      . . . define next state here
   END PROCESS;
   WITH state SELECT
      . . . define output here
END FSM;
```

# Next state process

```
PROCESS (clk)
BEGIN
   IF clk'EVENT AND clk = '1' THEN
      CASE state IS
         WHEN Idle =>  . . .
         WHEN G2 => . . .
         WHEN G1 => . . .
         WHEN Go => . . .
       END CASE;
   END IF;
END PROCESS;
```

# Case statement

```
CASE state IS
   WHEN Idle =>
      IF      req (2) = '1' THEN state <= G2;
      ELSIF req (1) = '1' THEN state <= G1;
      ELSIF req (0) = '1' THEN state <= G0;
      END IF;
   WHEN G2 =>
      IF req (2) = '0' THEN state <= Idle; END IF;
   WHEN G1 =>
      IF req (1) = '0' THEN state <= Idle; END IF;
   WHEN G0 =>
      IF req (0) = '0' THEN state <= Idle; END IF;
END CASE;
```

# Selected signal assignment

```
WITH state SELECT
ack <= “000”    WHEN Idle ,
       “100”    WHEN G2 ,
       “010”    WHEN G1 ,
       “001”   WHEN G0;
```

# Next state and output together

```
CASE state IS
   WHEN Idle =>
      IF      req (2) = '1' THEN ack <= "100"; state <= G2;
      ELSIF req (1) = '1' THEN ack <= "010"; state <= G1;
      ELSIF req (0) = '1' THEN ack <= "001"; state <= G0;
      ELSE                                   ack <= "000"; END IF;
   WHEN G2 =>
      IF req (2) = '0' THEN ack <= "000"; state <= Idle;
      ELSE                          ack <= "100"; END IF;
   WHEN G1 =>
      IF req (1) = '0' . . .
   WHEN G0 =>
      IF req (0) = '0' . . .
END CASE;
```

# Moore and Mealy models

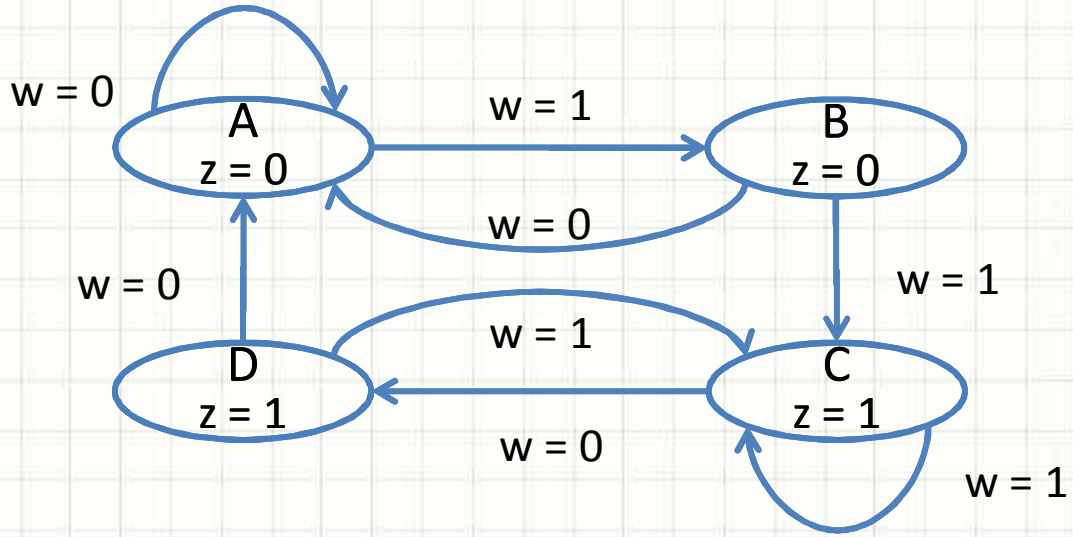
Moore

- Output is a function of present state only

Mealy

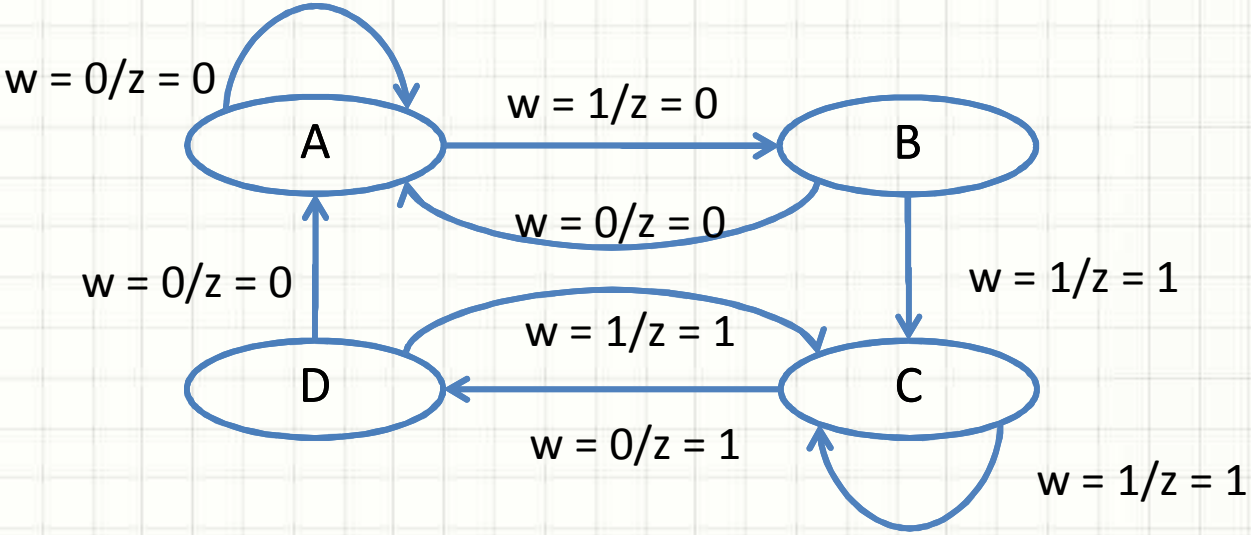
- Output is a function of present state as well as inputs

# Moore FSM example



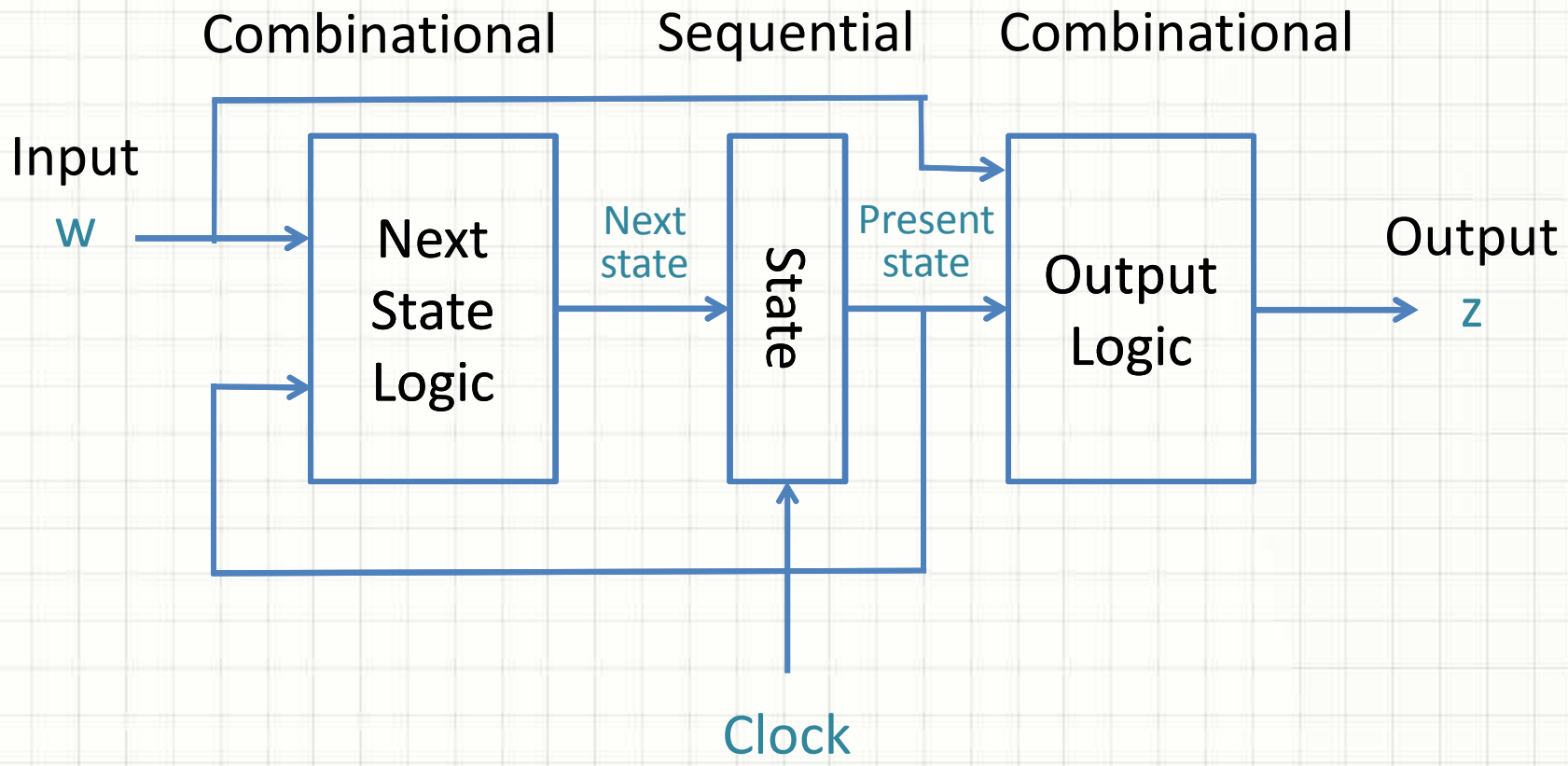

| Present state | Next state | | Output z |
|---|---|---|---|
| | w = 0 | w = 1 | |
| A | A | B | 0 |
| B | A | C | 0 |
| C | D | C | 1 |
| D | A | C | 1 |

# Mealy FSM example




| Present state | Next state/Output z | |
|---|---|---|
| | w = 0 | w = 1 |
| A | A/0 | B/0 |
| B | A/0 | C/1 |
| C | D/1 | C/1 |
| D | A/0 | C/1 |

# Circuit structure for Mealy FSM



# Next state for Moore/Mealy

```
CASE state IS
   WHEN A =>
      IF w = '1' THEN state <= B; END IF;
   WHEN B =>
      IF w = '1' THEN state <= C; ELSE state <= A; END IF;
   WHEN C =>
      IF w = '0' THEN state <= D; END IF;
   WHEN D =>
      IF w = '0' THEN state <= A; ELSE state <= C; END IF;
END CASE;
```

# Output for Moore FSM

```
WITH state SELECT
    z <= '0' WHEN A ,
         '0' WHEN B ,
         '1' WHEN C ,
         '1' WHEN D;
```

# Output for Mealy FSM

```
CASE state IS
   WHEN A =>
     z <= '0';
   WHEN B =>
      IF w = '1' THEN z <= '1'; ELSE z <= '0'; END IF;
   WHEN C =>
      z <= '1';
   WHEN D =>
      IF w = '0' THEN z <= '0'; ELSE z <= '1'; END IF;
END CASE;
```

# Mealy FSM output – another way

```
CASE state IS
   WHEN A =>
      z <= '0';
   WHEN B =>
      z <= w;
   WHEN C =>
      z <= '1';
   WHEN D =>
      z <= w;
END CASE;
```

# Book sections covered

Chapter 2: Logic/VHDL (2.10 – 2.12)

Chapter 4: Combinational circuits (4.4 – 4.5)

Chapter 7: Sequential components (7.1 – 7.16)

Chapter 8: Sequential circuit design (8.1 – 8.4, 8.7 – 8.9)

Appendix A: VHDL reference

# THANKS