Synchronous Sequential Circuits - Part I

VOLKAN KURSUN

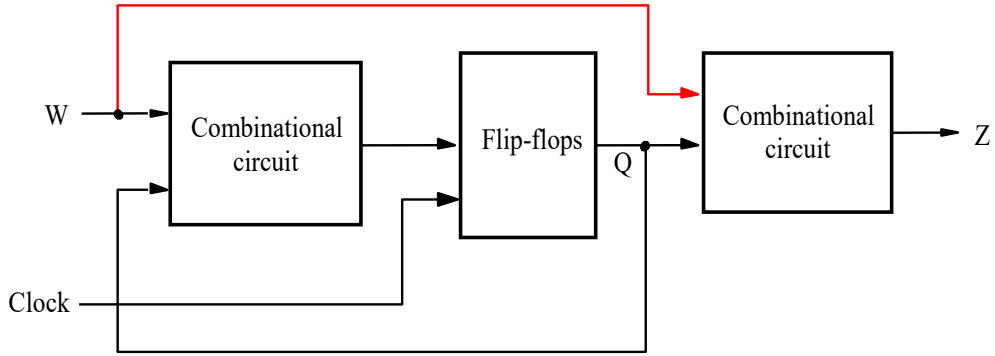
Some material from McGraw Hill

Outline

- Finite State Machines
- Moore State Model
- State Assignment
- Mealy State Model

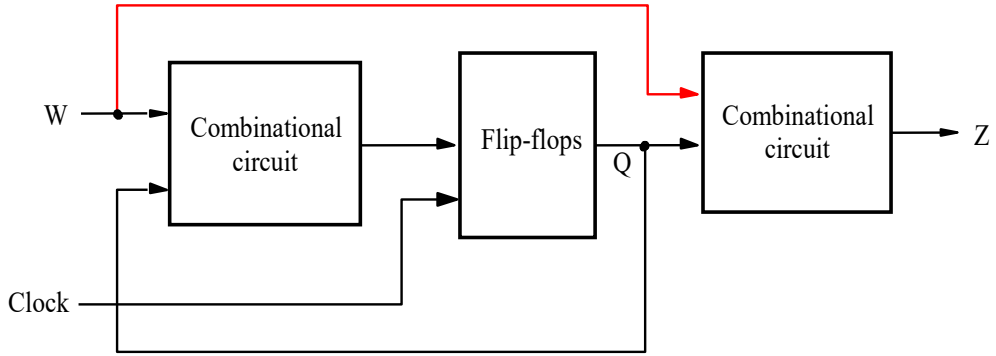
Sequential Circuits

- **Sequential circuit**: the outputs depend on the past behavior of the circuit and the present values of the inputs
- **Synchronous** sequential circuit: flip-flops are used to implement the state and **a clock signal** is used to control the operation of the sequential circuit



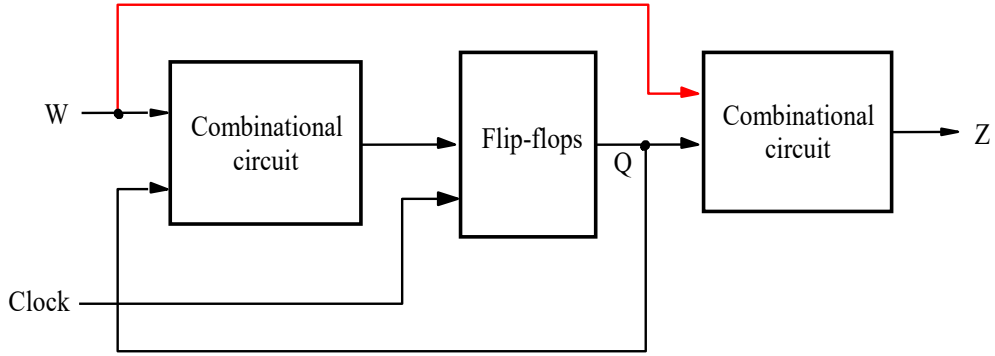
Synchronous Sequential Circuits

- **Sequential circuit state**: the **stored bits (Q) in the flip-flops** are referred to as the state
- **Synchronized with an edge of a clock signal**, the flip-flops change their state as determined by the combinational circuit that feeds the inputs to these flip-flops



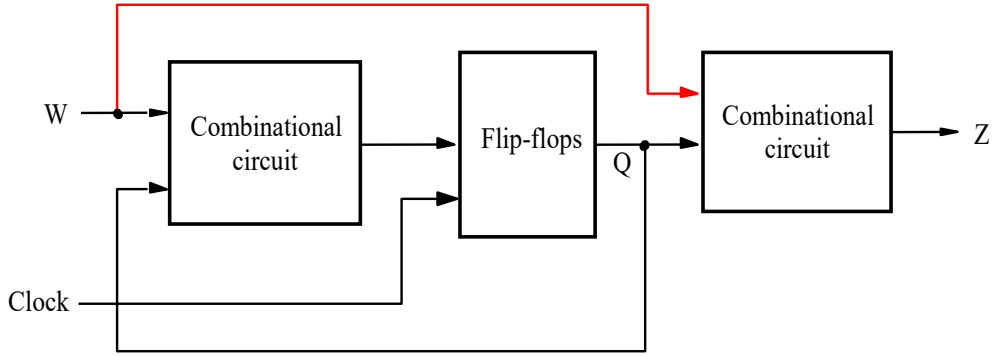
Synchronous Sequential Circuits

- **Only one transition from one state to another can occur during a clock cycle** since the flip-flops are edge triggered
- The combinational circuit that provides the inputs to the flip-flops has two sources: the primary inputs (W) and the present state of the flip-flops (Q)



Synchronous Sequential Circuits

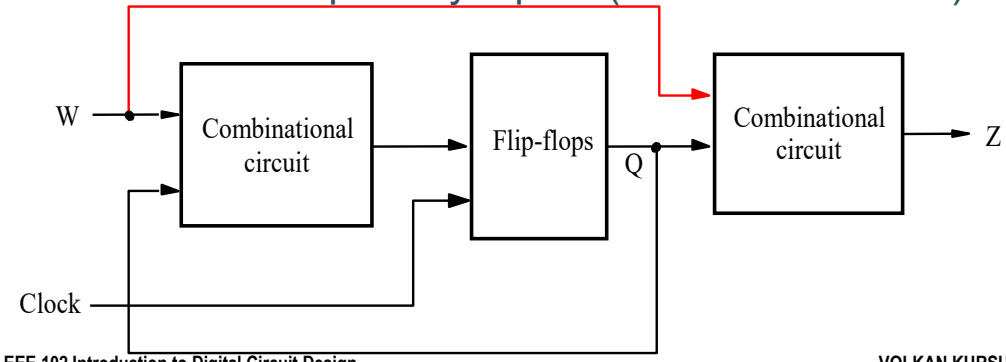
- The changes in state depend on both the present state (Q) and the primary inputs (W)
- Sequential circuits are also called **finite state machines**: behavior of a sequential circuit is represented with a finite number of states



Two Types of State Machines

- The outputs of the sequential circuit are produced by another combinational circuit

1) **Moore Machine**: the outputs depend only on the state of the circuit (the red line does NOT exist)
2) **Mealy Machine**: the outputs depend on both the state and the primary inputs (the red line exists)



Outline

- Finite State Machines
- Moore State Model
- State Assignment
- Mealy State Model

Moore Machine Example

- A **vehicle decelerator**: a binary signal w indicates if the speed is within limit. If w = 1 during two consecutive measurements, a control signal z is asserted to decelerate the car
- Assumptions:

1) The circuit has one input w and one output z
2) All changes occur with the positive edges of the clock signal
3) **If during two immediately preceding clock cycles the input w was 1**, the output z = 1. Otherwise, z = 0

Moore Machine Input-Output Sequences

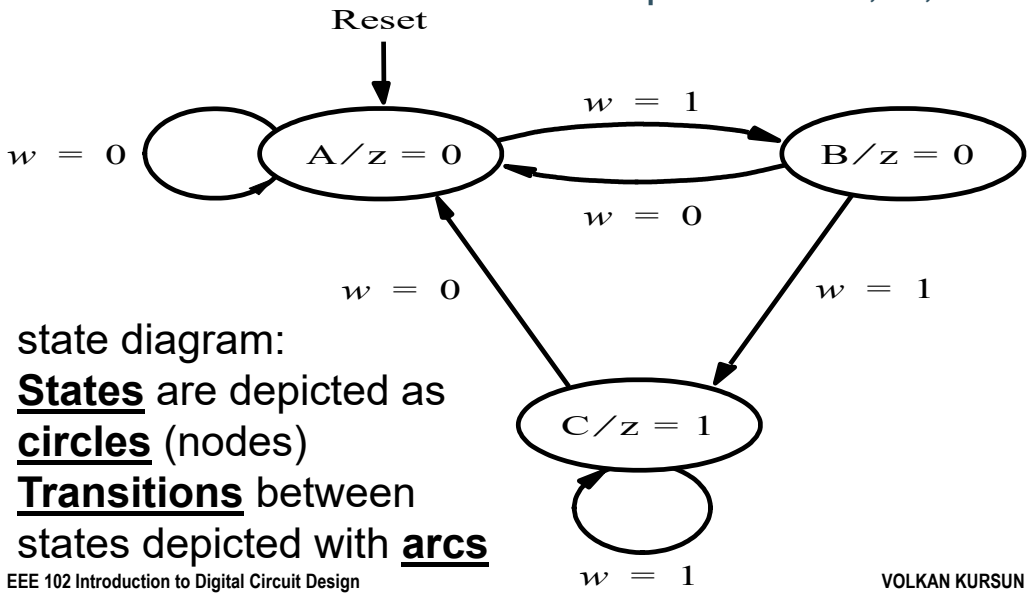
- Sequences of input and output signals:

Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

- **Step-1**: determine **the number of states** and the **transitions between states**
- A **state diagram** helps **to visualize the operation of the state machine**
- **Begin with an initial state**: this is the **state that the circuit should enter when power is first turned on or when reset**

Moore Machine State Diagram

- 3 states are sufficient to describe the behavior of this machine: states are depicted as A, B, C



state diagram:
States are depicted as **circles** (nodes)
Transitions between states depicted with **arcs**

Moore Machine State Table

- **State table**: all transitions from each present state to the next state for different values of the inputs are listed
- **Reset** is not listed in the table: the **first state in the table is implied to be the initial state**

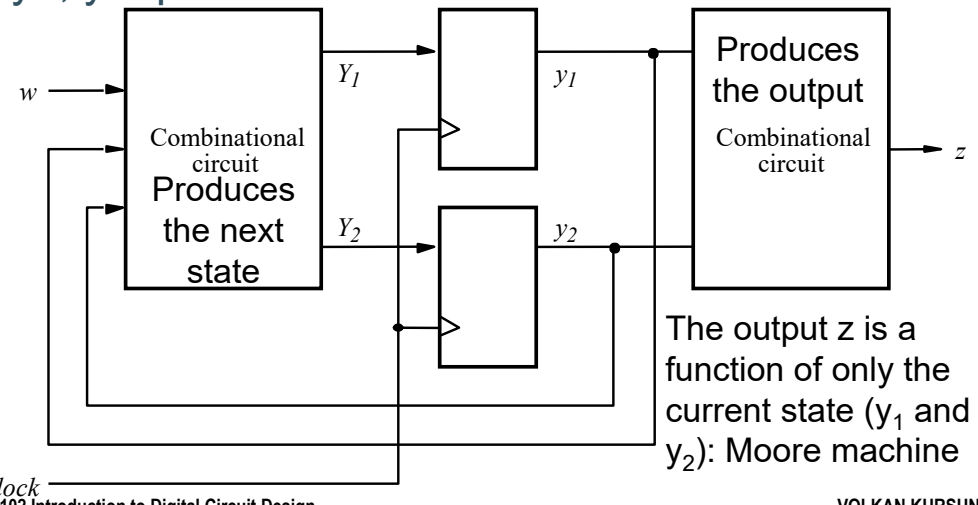
Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

Moore Machine State Assignment

- **State assignment**: the states are listed with letters A, B, and C in the state table
- For logic circuit implementation, **each state is represented with a binary number which is composed of state variables**
- **Each state variable is implemented with a flip-flop**
- To represent 3 states, 2 state variables are sufficient: y_1 and y_2
- The state variables are represented with flip-flops

Moore Machine Structure

- The circuit structure of the finite state machine:
- Y1, Y2: next state variables
- y1, y2: present state variables



The output z is a function of only the current state (y_1 and y_2): Moore machine

Moore Machine State Assigned Table

- **State-assigned table** (**truth table** for the state machine): assign specific binary numbers to each state
- **One** assignment (**there are other assignment possibilities too**):

State A is assigned 0b00, state B is assigned 0b01, and state C is assigned 0b10

	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

Moore Machine Combinational Circuits

- Use 2 D flip-flops to hold the present state variables y1 and y2
- **Derive the logic expressions for the next state and output functions** (**design** the combinational circuits that **produce the next state and outputs**)

	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

Moore Machine Next State Logic

Karnaugh map for Y1

w \ y_2y_1	00	01	11	10
0	0	0	d	0
1	1	0	d	0

Ignoring don't cares: $Y_1 = w\bar{y}_1\bar{y}_2$

Using don't cares: $Y_1 = w\bar{y}_1\bar{y}_2$

Karnaugh map for Y2

w \ y_2y_1	00	01	11	10
0	0	0	d	0
1	0	1	d	1

Ignoring don't cares: $Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$

Using don't cares: $Y_2 = wy_1 + wy_2 = w(y_1 + y_2)$

	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	dd	dd	d

Moore Machine Output Logic

- In a **Moore machine**, the **output** is a **function of only the present state variables y1 and y2**:

y_2 \ y_1	0	1
0	0	0
1	1	d

Ignoring don't cares: $z = \bar{y}_1y_2$

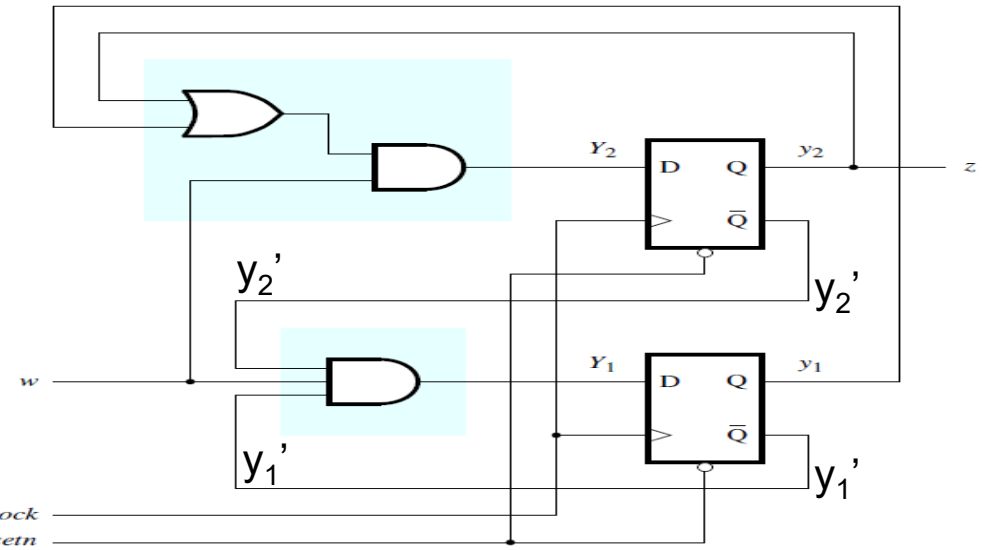
Using don't cares: $z = y_2$

	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	dd	dd	d

Moore Machine Circuit

- The circuit implementation is:

$Y_1 = wy_1'y_2'$, $Y_2 = w(y_1 + y_2)$, $z = y_2$



Moore Machine VHDL

```
1  LIBRARY ieee ;
2  USE ieee.std_logic_1164.all ;
3  ENTITY simple IS
4          PORT (   Clock, Resetn, w      : IN    STD_LOGIC ;
                    z                     : OUT  STD_LOGIC ) ;
5  END simple ;
7  ARCHITECTURE Behavior OF simple IS
8          TYPE State_type IS (A, B, C) ;
9          SIGNAL y : State_type ;
10 BEGIN
11         PROCESS ( Resetn, Clock )
12         BEGIN
13             IF Resetn = '0' THEN
14                 y <= A ;
15             ELSIF (Clock'EVENT AND Clock = '1') THEN
16                 CASE y IS
17                     WHEN A =>
18                         IF w = '0' THEN
19                             y <= A ;
20                         ELSE
21                             y <= B ;
22                         END IF ;
23                     WHEN B =>
24                         IF w = '0' THEN
25                             y <= A ;
26                         ELSE
27                             y <= C ;
28                         END IF ;
29                     WHEN C =>
30                         IF w = '0' THEN
31                             y <= A ;
32                         ELSE
33                             y <= C ;
34                         END IF ;
35                 END CASE ;
36             END IF ;
37         END PROCESS ;
38         z <= '1' WHEN y = C ELSE '0' ;
39 END Behavior ;
```

User defined signal TYPE with 3 possible values: A, B, or C

y signal represents the flip-flops that hold the state

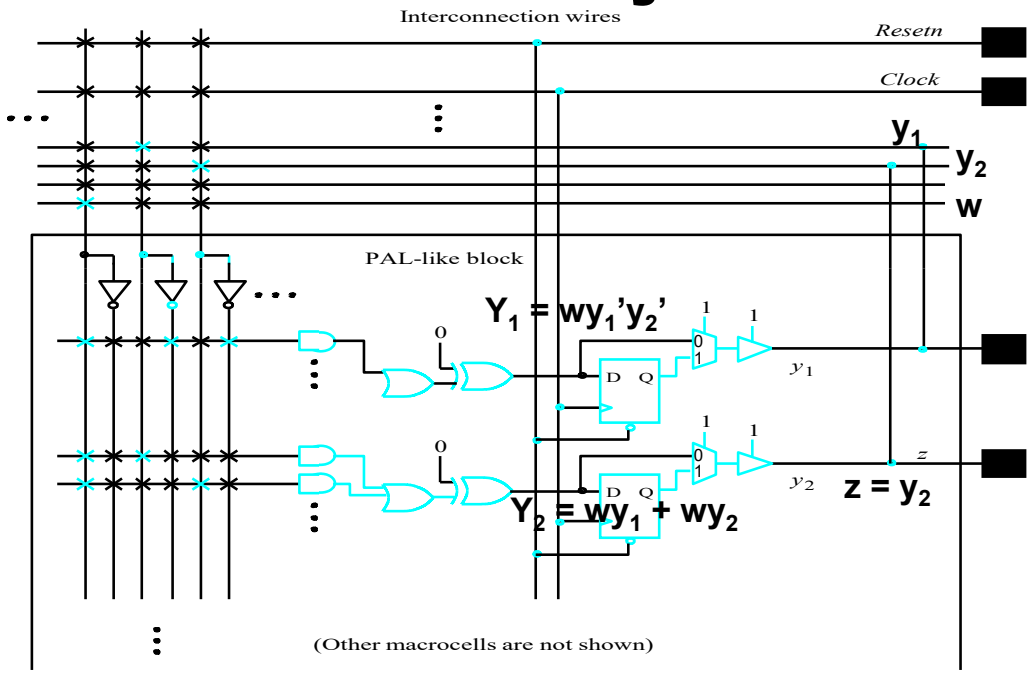
The number of flip-flops that hold the state is not specified in the code: **the synthesis tool assigns the states and chooses an appropriate number of flip-flops**

Sensitivity list: the signals that can cause the process to change y. **Resetn and Clock are the only two signals that can cause y to change**

The input w is NOT included in the sensitivity list since a change in w can NOT affect y until a change occurs in the Clock signal that may trigger a state change

If the machine is in state C, the output z must be asserted. Otherwise, the output z must be 0. **MOORE machine**

Moore Machine Synthesized



Moore Machine VHDL-2

```
ARCHITECTURE Behavior OF simple IS
    TYPE State_type IS (A, B, C) ;
    SIGNAL y_present, y_next : State_type ;
BEGIN
    PROCESS ( w, y_present )
    BEGIN
        CASE y_present IS
            WHEN A =>
                IF w = '0' THEN
                    y_next <= A ;
                ELSE
                    y_next <= B ;
                END IF ;
            WHEN B =>
                IF w = '0' THEN
                    y_next <= A ;
                ELSE
                    y_next <= C ;
                END IF ;
            WHEN C =>
                IF w = '0' THEN
                    y_next <= A ;
                ELSE
                    y_next <= C ;
                END IF ;
        END CASE ;
    END PROCESS ;
```

Alternative code for the state machine:

Signal y_present corresponds to the current state
Signal y_next corresponds to the next state
y_present and y_next are of the user defined TYPE State_type

The **first PROCESS describes the combinational circuit that produces the next state** y_next

The **second PROCESS describes the flip-flops that hold the present state** y_present

```
    PROCESS (Clock, Resetn)
    BEGIN
    IF Resetn = '0' THEN
            y_present <= A ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
            y_present <= y_next ;
    END IF ;
    END PROCESS ;

    z <= '1' WHEN y_present = C ELSE '0' ;
    END Behavior ;
```

Note: although statements within a process are evaluated and executed sequentially, all processes within the model are executed concurrently

Sequential Circuit Design Steps

1. Obtain the **specification** of the desired circuit
2. Derive a **state diagram**
3. Derive the corresponding **state table**
4. **Reduce the number of states** if possible
5. Decide on the **number of state variables**
6. Choose the **type of flip-flops** to be used
7. **Derive the logic expressions** needed to implement the circuit

asynchronous reset
asynchronous reset

FSM for Register Transfers

- Registers hold data for various operations in a computer system
- Sometimes it is necessary to swap the contents of two registers: a third register to store temporary data is used for the swap operation
- Example: swap the contents of registers R1 and R2

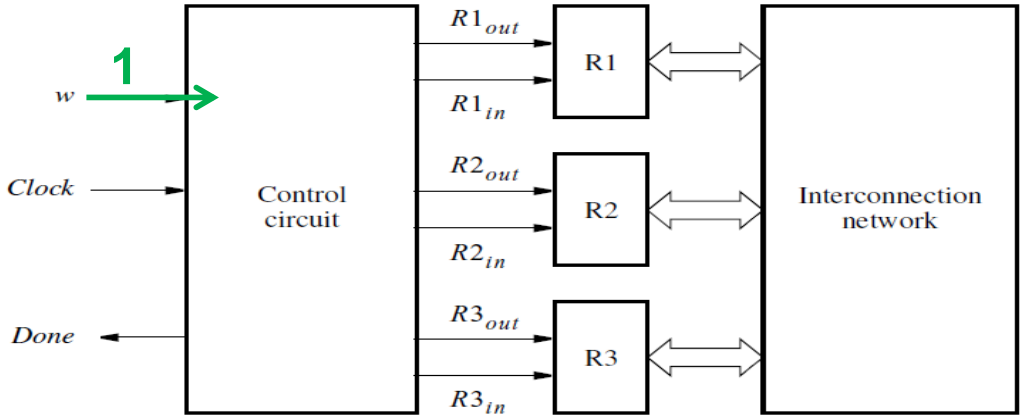
Step 1: transfer the contents of R2 to a temporary register R3, R3 ← R2

Step 2: transfer the contents of R1 to R2, R2 ← R1

Step 3: transfer the contents of the temporary register R3 to R1, R1 ← R3

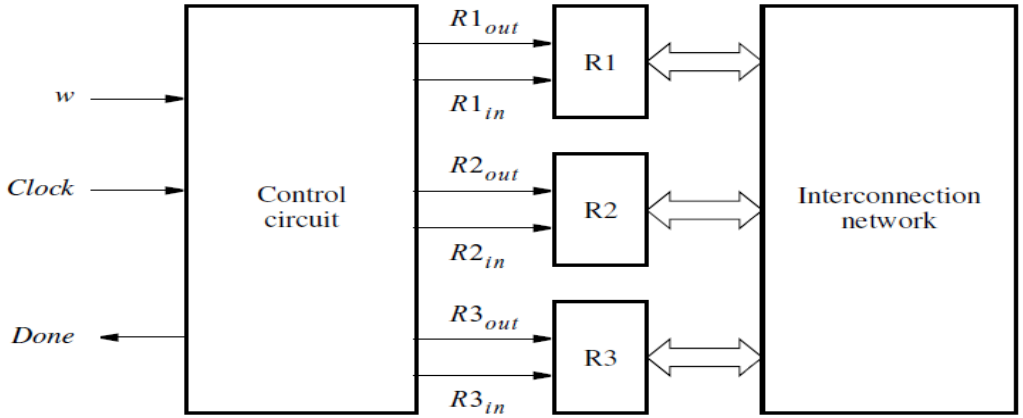
Register Interconnection

- Registers in a computer system are interconnected via an interconnection network
- Control circuit (FSM) issues the control signals to swap the contents of R1 and R2 in response to an external signal w: w = 1, swap R1 and R2



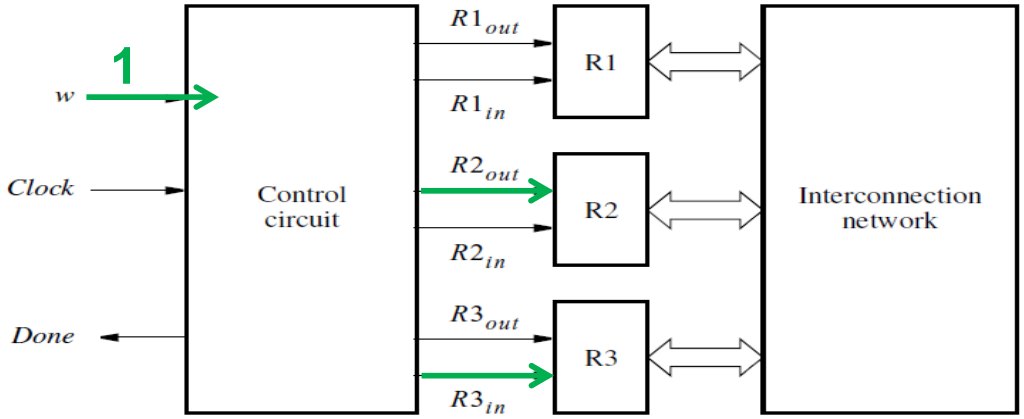
Register Transfer Control

- Rk_{out} and Rk_{in} signals are produced by the control circuit
- Rk_{out} signal: causes the contents of register Rk to be placed into the interconnection network
- Rk_{in} signal: causes the data from the network to be loaded into Rk



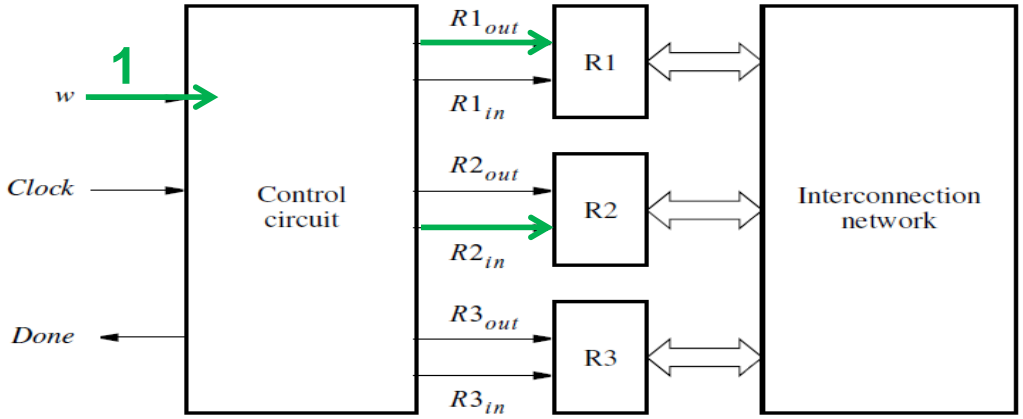
Register Swap Step 1

- Step 1: transfer the contents of R2 to a temporary register R3, R3 ← R2
- **$R2_{out}$ = 1**: place the contents of R2 into the interconnection network
- **$R3_{in}$ = 1**: data from the network loaded into R3



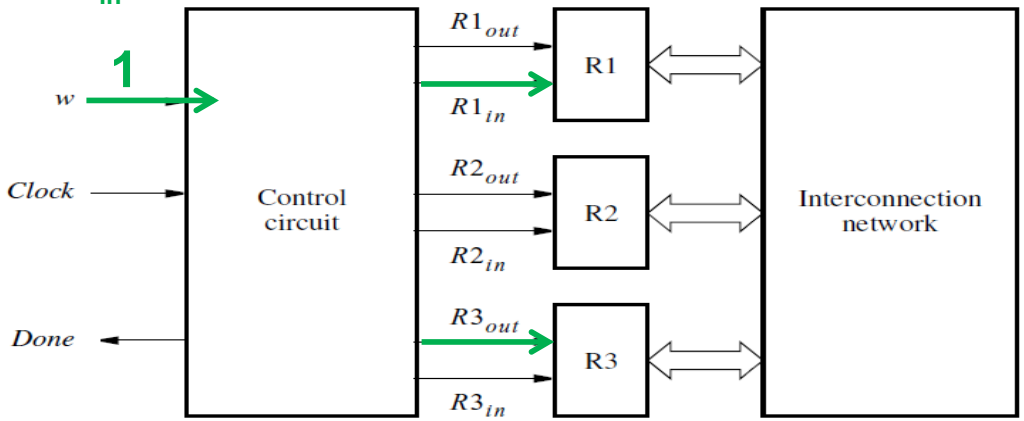
Register Swap Step 2

- Step 2: transfer the contents of R1 to R2, R2 ← R1
- **$R1_{out}$ = 1**: place the contents of R1 into the interconnection network
- **$R2_{in}$ = 1**: data from the network loaded into R2



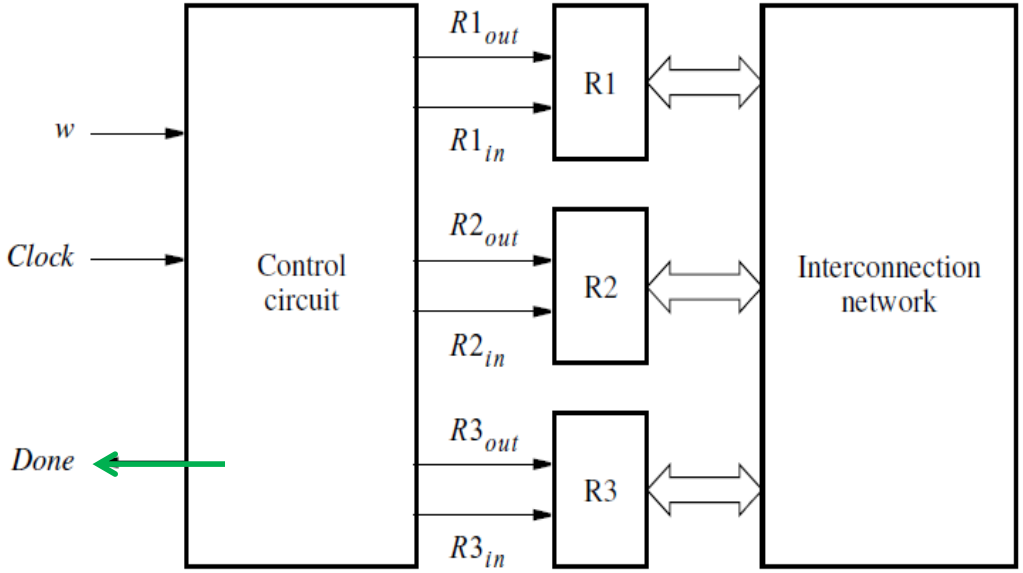
Register Swap Step 3

- Step 3: transfer the contents of the temporary register R3 to R1, R1 ← R3
- **$R3_{out}$ = 1**: place the contents of R3 into the interconnection network
- **$R1_{in}$ = 1**: data from the network loaded into R1



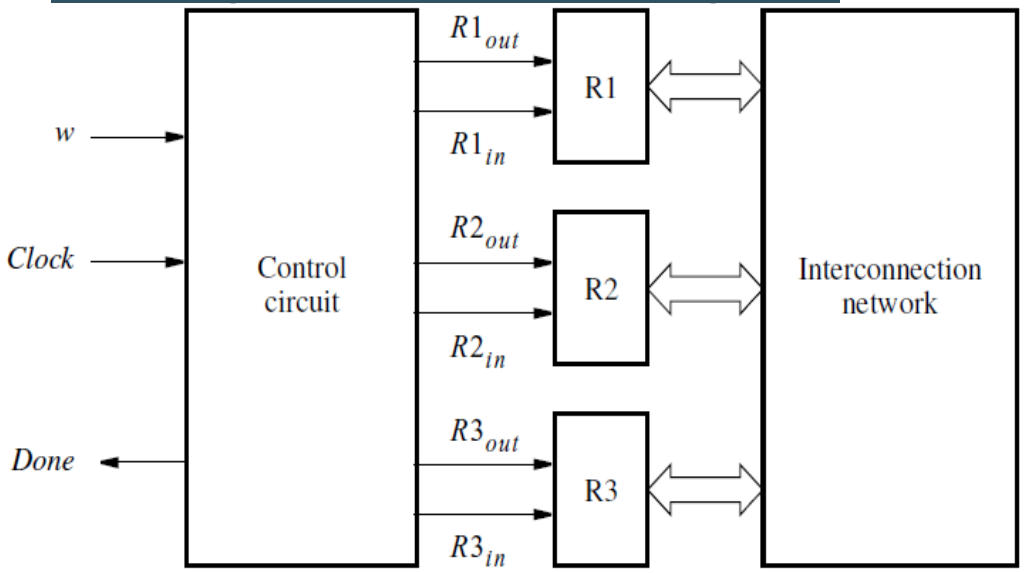
Register Swap Done

- **After the swap operation, control circuit asserts the Done signal**



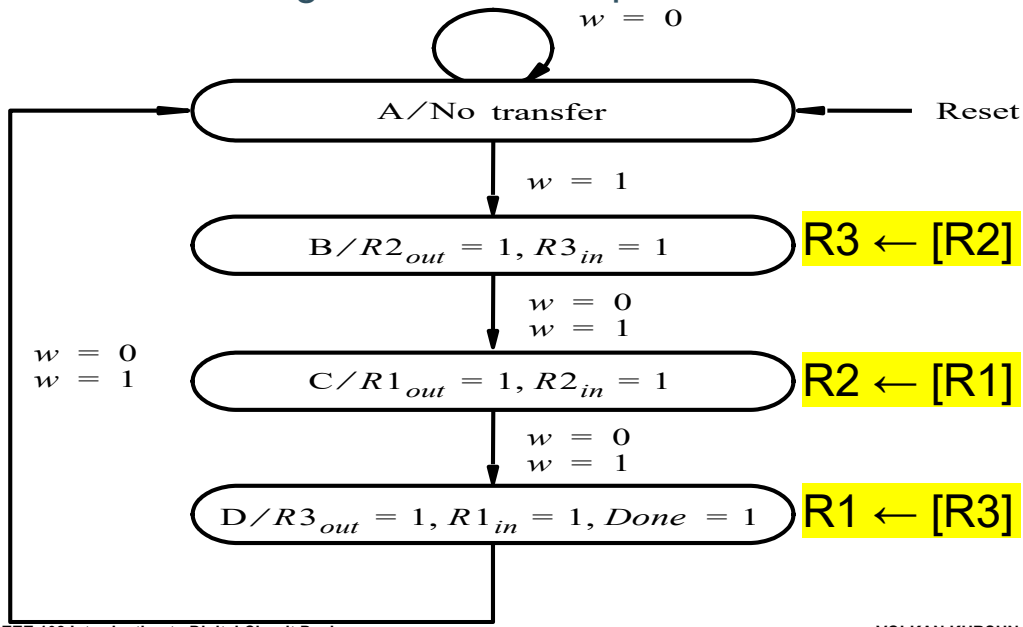
Moore Machine Example-2

- **Design the controller state machine for swapping the contents of two registers**:



Moore Machine State Diagram

- The state diagram for the swap machine:



Moore Machine State Table

Present state	Next state $w=0$	Next state $w=1$	Outputs $R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	A	B	0	0	0	0	0	0	0
B	C	C	0	0	1	0	0	1	0
C	D	D	1	0	0	1	0	0	0
D	A	A	0	1	0	0	1	0	1

- To distinguish these 4 states, 2 state variables are sufficient: y2 and y1
- One possible set of assignments for A, B, C, and D are: A = 0b00, B = 0b01, C = 0b10, D = 0b11

Moore Machine State Assignments

- The state-assigned table is:

	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Outputs $R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	00	01	0	0	0	0	0	0	0
B	01	10	10	0	0	1	0	0	1	0
C	10	11	11	1	0	0	1	0	0	0
D	11	00	00	0	1	0	0	1	0	1

- **Derive the next state expressions**:

Karnaugh map for Y2

y_2y_1 / w	00	01	11	10
0		1		1
1		1		1

$Y_2 = y_1\bar{y}_2 + \bar{y}_1 y_2$

Karnaugh map for Y1

y_2y_1 / w	00	01	11	10
0				1
1	1			1

$Y_1 = w\bar{y}_1 + \bar{y}_1 y_2$

Moore Machine Output Logic

	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Outputs $R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	00	01	0	0	0	0	0	0	0
B	01	10	10	0	0	1	0	0	1	0
C	10	11	11	1	0	0	1	0	0	0
D	11	00	00	0	1	0	0	1	0	1

- **Derive the output expressions**:

$R1_{out} = R2_{in} = y_2y_1'$

$R1_{in} = R3_{out} = Done = y_2y_1$

$R2_{out} = R3_{in} = y_2'y_1$

Moore machine outputs depend only on the present state variables

Moore Machine Circuit

Next state expressions:

$Y_1 = wy_1' + y_1'y_2$

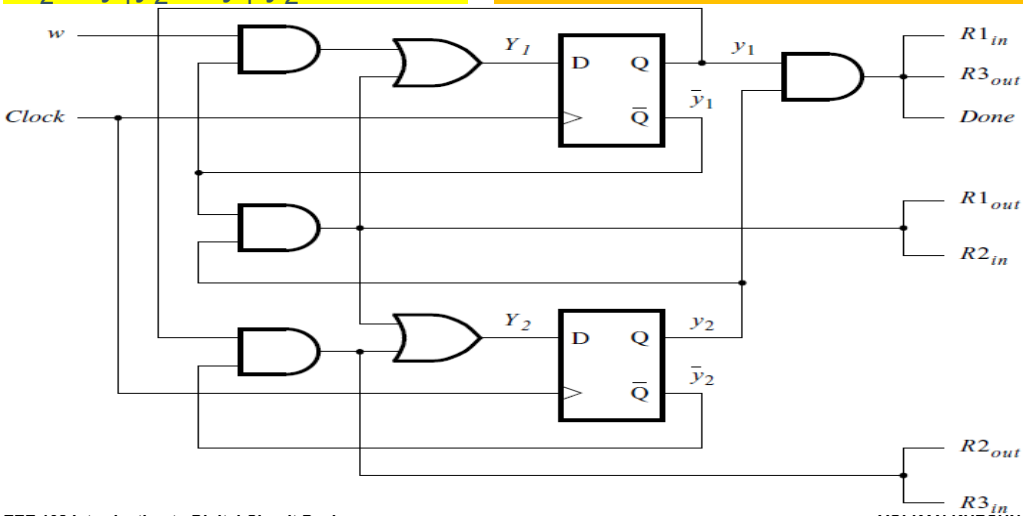
$Y_2 = y_1y_2' + y_1'y_2$

Output expressions:

$R1_{out} = R2_{in} = y_1'y_2$

$R1_{in} = R3_{out} = Done = y_1y_2$

$R2_{out} = R3_{in} = y_1y_2'$



Outline

- Finite State Machines
- Moore State Model
- State Assignment
- Mealy State Model

State Assignment

- Some state assignments may lead to simpler circuits than others
- Example: revisit the car decelerator design. Re-assign the states A, B, and C to 0b00, 0b01, and 0b11

	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	11	0
C	11	00	11	1
	10	dd	dd	d

Next State and Output Logic

$Y_2 = wy_1$

w \ y_2y_1	00	01	11	10
0	0	0	0	d
1	0	1	1	d

$Y_1 = w$

w \ y_2y_1	00	01	11	10
0	0	0	0	d
1	1	1	1	d

$z = y_2$

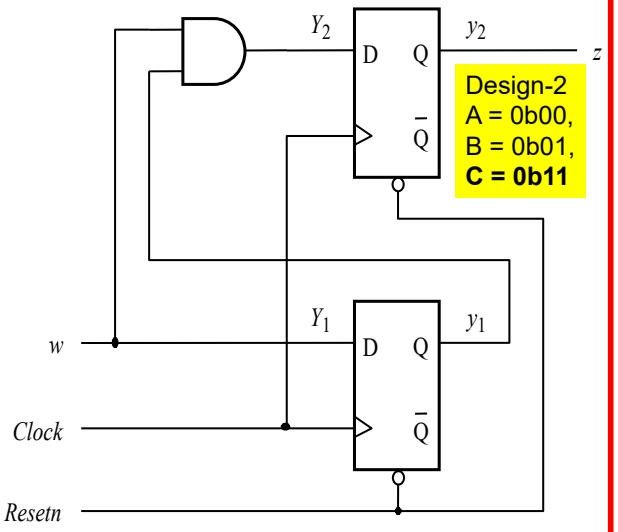
y_2 \ y_1	0	1
0	0	0
1	d	1

	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	11	0
C	11	00	11	1
	10	dd	dd	d

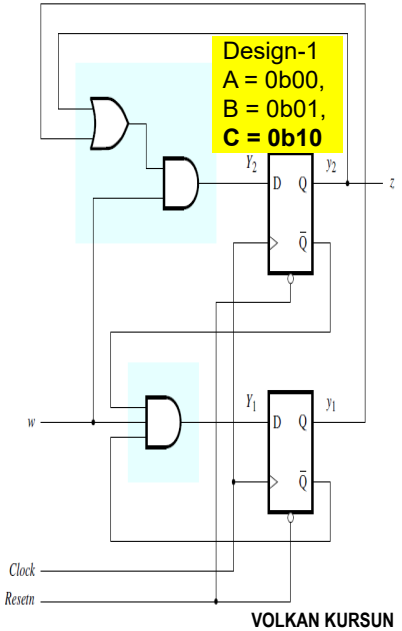
State Assignments Comparison

- Draw the circuit and compare to the first design:

$Y_1 = w$, $Y_2 = wy_1$, $z = y_2$



Design-2
A = 0b00,
B = 0b01,
C = 0b11



State Assignment VHDL

- It is possible to **specify the state assignments in VHDL code**

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

ENTITY simple IS
    PORT ( Clock, Resetn, w  : IN     STD_LOGIC ;
           z                 : OUT    STD_LOGIC ) ;
END simple ;

ARCHITECTURE Behavior OF simple IS
    SIGNAL y_present, y_next : STD_LOGIC_VECTOR(1 DOWNTO 0);
    CONSTANT A :  STD_LOGIC_VECTOR(1 DOWNTO 0) := "00" ;
    CONSTANT B :  STD_LOGIC_VECTOR(1 DOWNTO 0) := "01" ;
    CONSTANT C :  STD_LOGIC_VECTOR(1 DOWNTO 0) := "11" ;
BEGIN
    PROCESS ( w, y_present )
    BEGIN
        CASE y_present IS
            WHEN A =>
                IF w = '0' THEN y_next <= A ;
                ELSE y_next <= B ;
                END IF ;
            WHEN B =>
                IF w = '0' THEN y_next <= A ;
                ELSE y_next <= C ;
                END IF ;
            WHEN C =>
                IF w = '0' THEN y_next <= A ;
                ELSE y_next <= C ;
                END IF ;
            WHEN OTHERS =>
                y_next <= A ;
        END CASE ;
    END PROCESS ;
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            y_present <= A ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            y_present <= y_next ;
        END IF ;
    END PROCESS ;
    z <= '1' WHEN y_present = C ELSE '0' ;
END Behavior ;
```

y_present is a 2-bit STD_LOGIC_VECTOR
The CASE statement must cover all possible values of y_present (note that the first process is for the next state logic which is a combinational circuit). WHEN OTHERS clause is used to cover y_present = 0b10. The state 0b10 would normally never happen. However, if it occurs due to noise, the state machine transitions back to the initial reset state A in case of such an error.

In an alternative design, y_next could be assigned don't care WHEN y_present = 0b10: y_next <= "- -"

State Assignment Example-2

- Redesign the swap controller by re-assigning the states: interchange the binary numbers assigned to states C and D

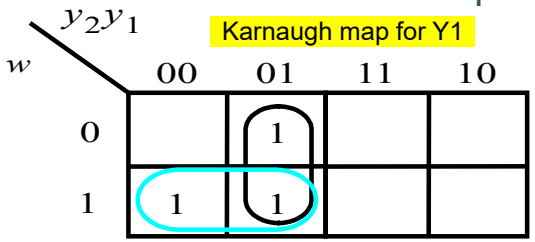
	Present state	Nextstate		Outputs						
		$w=0$	$w=1$							
	y_2y_1	Y_2Y_1	Y_2Y_1	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	00	01	0	0	0	0	0	0	0
B	01	11	11	0	0	1	0	0	1	0
C	11	10	10	1	0	0	1	0	0	0
D	10	00	00	0	1	0	0	1	0	1

State Assignment Example-2

	Present state	Nextstate		Outputs						
		$w=0$	$w=1$							
	y_2y_1	Y_2Y_1	Y_2Y_1	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	00	01	0	0	0	0	0	0	0
B	01	11	11	0	0	1	0	0	1	0
C	11	10	10	1	0	0	1	0	0	0
D	10	00	00	0	1	0	0	1	0	1

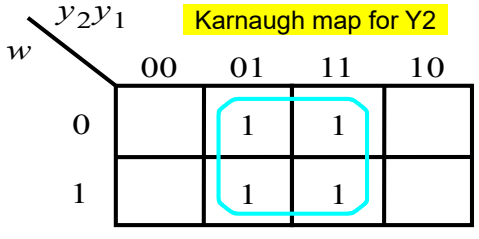
- Derive the next-state expressions:

Karnaugh map for Y1



$Y_1 = w\bar{y}_2 + y_1\bar{y}_2$

Karnaugh map for Y2



$Y_2 = y_1$

State Assignment Example-2

	Present state	Nextstate		Outputs						
		$w=0$	$w=1$							
	y_2y_1	Y_2Y_1	Y_2Y_1	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	00	01	0	0	0	0	0	0	0
B	01	11	11	0	0	1	0	0	1	0
C	11	10	10	1	0	0	1	0	0	0
D	10	00	00	0	1	0	0	1	0	1

- Derive the output expressions:

$R1_{out} = R2_{in} = y_2y_1$

$R1_{in} = R3_{out} = Done = y_2y_1'$

$R2_{out} = R3_{in} = y_2'y_1$

- The resulting circuit is simpler as compared to the first design based on a different state assignment

One-Hot Encoding

- In the previous examples, minimum number of flip-flops were used to represent the states
- An **alternative state assignment method** is to use as many state variables as there are states
- For each state, only one state variable is 1 (hot) while the other state variables are all 0s
- **Goal**: **simplify the combinational circuits** that produce the next state and outputs
- Simpler combinational circuits **may** lead to **higher clock frequency**
- **Tradeoff**: **increased number of flip-flops**

One-Hot Encoding Example

- Revisit the swap controller design: for the 4 states, use 4 state variables (treat the unused bit combinations as don't cares to simplify the logic circuits)

	Present state	Nextstate $w=0$	Nextstate $w=1$	Outputs						
	$y_4y_3y_2y_1$	$Y_4Y_3Y_2Y_1$	$Y_4Y_3Y_2Y_1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	0 001	0001	0010	0	0	0	0	0	0	0
B	0 010	0100	0100	0	0	1	0	0	1	0
C	0 100	1000	1000	1	0	0	1	0	0	0
D	1 000	0001	0001	0	1	0	0	1	0	1

One-Hot Encoding Example

- Derive the next state expressions:

$Y_1 = w'y_1 + y_4$, $Y_2 = wy_1$, $Y_3 = y_2$, $Y_4 = y_3$

	Present state	Nextstate $w=0$	Nextstate $w=1$	Outputs						
	$y_4y_3y_2y_1$	$Y_4Y_3Y_2Y_1$	$Y_4Y_3Y_2Y_1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	0 001	0001	0010	0	0	0	0	0	0	0
B	0 010	0100	0100	0	0	1	0	0	1	0
C	0 100	1000	1000	1	0	0	1	0	0	0
D	1 000	0001	0001	0	1	0	0	1	0	1

One-Hot Encoding Example

- Derive the output expressions:

$R1_{out} = R2_{in} = y_3$, $R1_{in} = R3_{out} = Done = y_4$

$R2_{out} = R3_{in} = y_2$

	Present state	Nextstate $w=0$	Nextstate $w=1$	Outputs						
	$y_4y_3y_2y_1$	$Y_4Y_3Y_2Y_1$	$Y_4Y_3Y_2Y_1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	0 001	0001	0010	0	0	0	0	0	0	0
B	0 010	0100	0100	0	0	1	0	0	1	0
C	0 100	1000	1000	1	0	0	1	0	0	0
D	1 000	0001	0001	0	1	0	0	1	0	1

3 Swap Controllers: Comparison

Design-1) Original design (**minimum number of flip-flops**):

Next state expressions:

$Y_1 = wy_1' + y_1'y_2$

$Y_2 = y_1y_2' + y_1'y_2$

Output expressions:

$R1_{out} = R2_{in} = y_1'y_2$

$R1_{in} = R3_{out} = Done = y_1y_2$

$R2_{out} = R3_{in} = y_1y_2'$

Design-2) Simplified design with **state reassignment** (**minimum number of flip-flops**):

$Y_1 = w\bar{y}_2 + y_1\bar{y}_2 \quad Y_2 = y_1$

$R1_{out} = R2_{in} = y_2y_1$, $R1_{in} = R3_{out} = Done = y_2y_1'$, $R2_{out} = R3_{in} = y_2'y_1$

Design-3): with **one-hot encoding**

The **next state expressions** :

$Y_1 = w'y_1 + y_4$, $Y_2 = wy_1$, $Y_3 = y_2$, $Y_4 = y_3$

The **output expressions** with **one-hot encoding**:

$R1_{out} = R2_{in} = y_3$, $R1_{in} = R3_{out} = Done = y_4$

$R2_{out} = R3_{in} = y_2$

One-Hot Encoding Observations

- The next state expressions with one-hot encoding:

$Y_1 = w'y_1 + y_4$, $Y_2 = wy_1$, $Y_3 = y_2$, $Y_4 = y_3$

- The output expressions with one-hot encoding:

$R1_{out} = R2_{in} = y_3$, $R1_{in} = R3_{out} = Done = y_4$

$R2_{out} = R3_{in} = y_2$

- These **output expressions are simpler** than the previous two designs that used minimum number of flip-flops for the state assignment
- However, the **number of flip-flops increased from 2 to 4 with the one-hot state assignment**

Outline

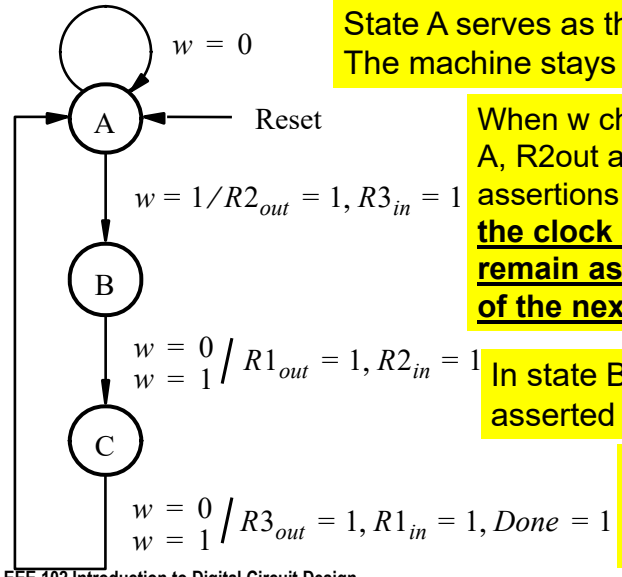
- Finite State Machines
- Moore State Model
- State Assignment
- Mealy State Model

Mealy State Model

- Mealy-type machines: the **outputs** are generated **based on both the present state of the circuit and the present values of the inputs**
- Provides **additional flexibility** in the design of sequential circuits (**number of states can be reduced** but the circuit behavior also changes)
- Example: re-design the register swap controller as a Mealy-machine
- The **Mealy implementation** requires 3 states: although 2 flip-flops are still needed, the **control signals are generated one cycle sooner** than the Moore machine which requires 4 states

Mealy State Model Example

- Example: re-design the register swap controller as a Mealy-machine



State A serves as the Reset state
The machine stays in state A as long as w = 0

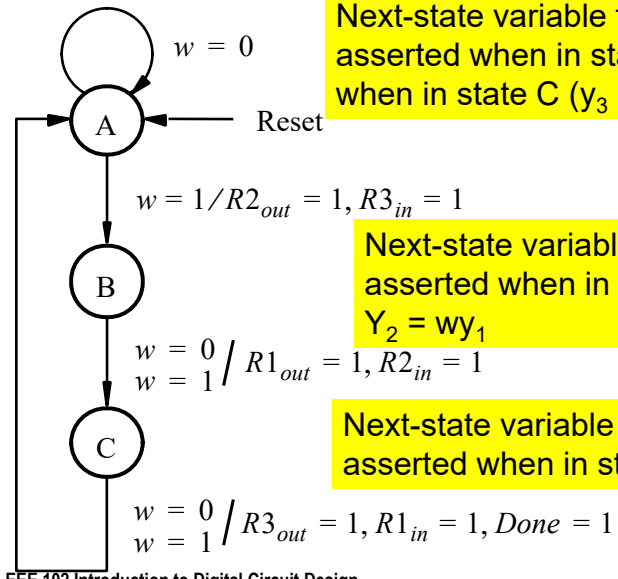
When w changes from 0 to 1 in state A, R2out and R3in are asserted (these assertions are **not synchronized with the clock edge and these signals remain asserted until the beginning of the next clock cycle**)

In state B, R1out and R2in are asserted regardless of the value of w

In state C, R3out, R1in, and Done are asserted regardless of the value of w

Mealy State Model Example

- Implement the Mealy machine with one-hot encoding: A = 0b001, B = 0b010, and C = 0b100



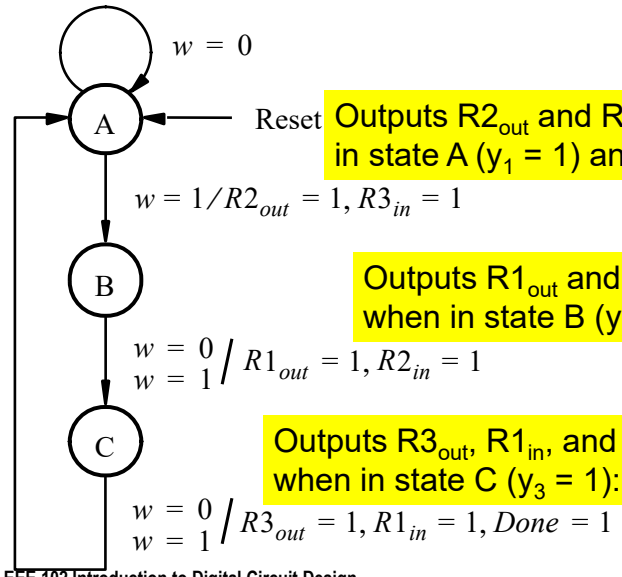
Next-state variable for flip-flop-1 (Y_1) must be asserted when in state A ($y_1 = 1$) and w = 0 or when in state C ($y_3 = 1$): $Y_1 = w'y_1 + y_3$

Next-state variable for flip-flop-2 (Y_2) must be asserted when in state A ($y_1 = 1$) and w = 1: $Y_2 = wy_1$

Next-state variable for flip-flop-3 (Y_3) must be asserted when in state B ($y_2 = 1$): $Y_3 = y_2$

Mealy State Model Example

- Implement the Mealy machine with one-hot encoding: A = 0b001, B = 0b010, and C = 0b100



Outputs $R2_{out}$ and $R3_{in}$ must be asserted when in state A ($y_1 = 1$) and w = 1: $R2_{out} = R3_{in} = wy_1$

Outputs $R1_{out}$ and $R2_{in}$ must be asserted when in state B ($y_2 = 1$): $R1_{out} = R2_{in} = y_2$

Outputs $R3_{out}$, $R1_{in}$, and Done must be asserted when in state C ($y_3 = 1$): $R3_{out} = R1_{in} = Done = y_3$