ENGN4213/6213
Digital Systems and Microprocessors

Semester 1, 2023

Week 3, Lecture B:

Finite State Machines

Australian
National
University

What's in this lecture

- Finite State Machines (FSM)
  - Mealy machine
  - Moore machine
- Designing:
  - State diagrams and tables
  - Coding in Verilog
- FSM design examples

Resources

- Pre-recorded videos lectures
  - VL6: Finite state machines
- Wakerly (5th edition)
  - Chapter 9: State Machines
  - Chapter 12: Verilog State-Machine Coding Styles
- ENGN3213 old Reading Brick from 2008 - Chapter 7 (Finite State Machines)

Finite State Machine (FSM)

- Sequential circuit: the output (y) is a function (f_s) of current and past inputs (u)

$$y_s(t) = f_s(u(t), u(t-1), u(t-2), \dots)$$

- Time steps of input history required for a complete description depends on the system
  - Some systems may require a full input history which can be intractable

The state $\big(s(t)\big)$ of a sequential circuit is a collection of state variables whose values at any one time contain all the information about the past necessary to account for the circuit's future behavior.

- Output can be unambiguously described as a function of the current state and input

$$y_s(t) = f_s\big(s(t), u(t)\big)$$

- In digital design, state variables are binary values representing logic signals in the circuit
  - A circuit with n binary state variables has 2^n possible states
- In sequential circuits, the number (2^n) of state variables can be large but is finite. Hence the name **Finite-state Machine (FSM)**.
  - All sequential logic circuits can be interpreted as FSM

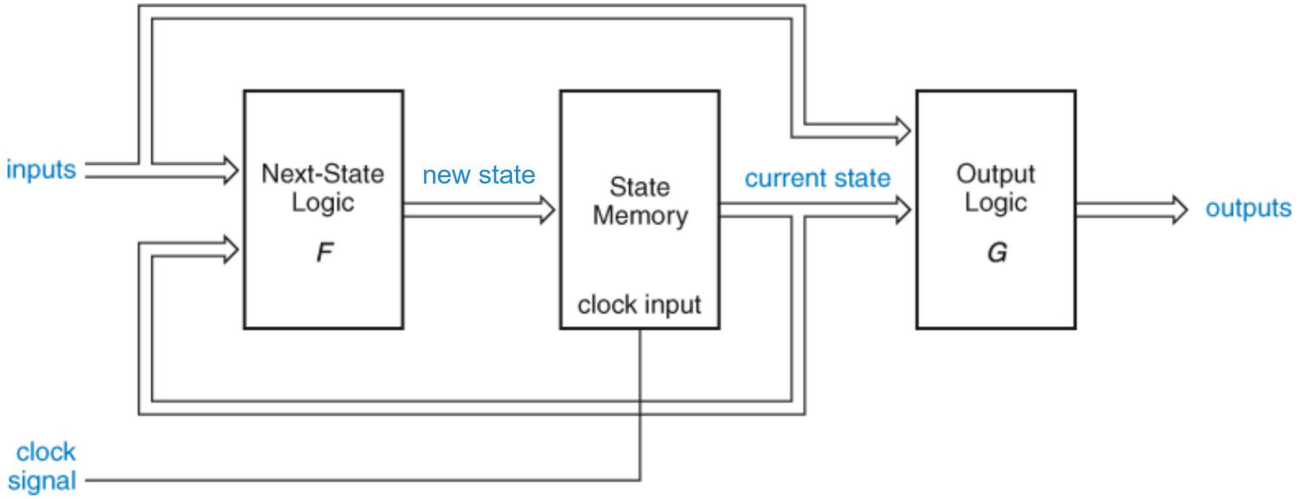
Finite State Machine (FSM)

Mealy machines

Mealy machines: A sequential circuit whose output depends on both the state and input

$$\text{Next State} = F(\text{Current State}, \text{Input})$$ **Next-State equation**

$$\text{Output} = G(\text{Current State}, \text{Input})$$ **Output equation**



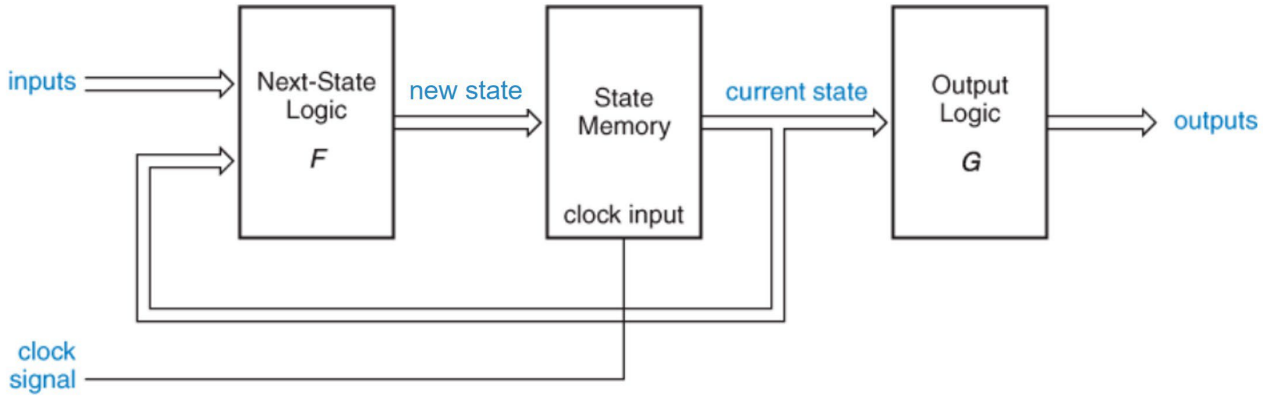
Finite State Machine (FSM)

Moore machines

Moore machines: A sequential circuit whose output depends only on the state alone

$$\text{Next State} = F(\text{Current State}, \text{Input})$$ **Next-State equation**

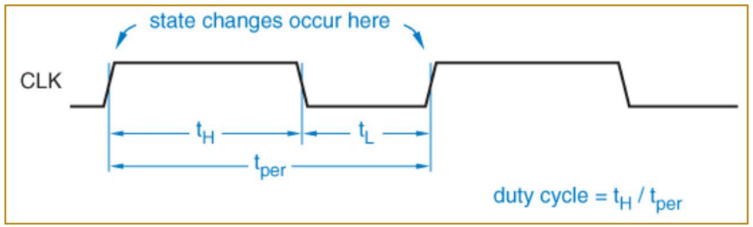
$$\text{Output} = G(\text{Current State})$$ **Output equation**



Finite State Machine (FSM)

Structure

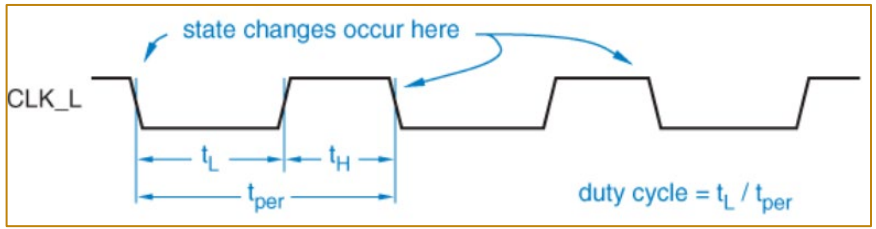
- Both Next-state logic F and Output logic G are strictly combinational logic circuits
- State memory is the true sequential part of the FSM design
  - Stores the current state of the machine
  - Implemented as a set of n flip-flops allowing 2^n distinct states.
  - Synchronous FSM: All flip-flops are connected to a common clock (edge-triggered D flip-flops are commonly used)
    - State transitions occur only at the active clock edge.
    - Between active edges, the state remains stable.



active high

$$\text{period} = t_{per}$$

$$\text{frequency} = \frac{1}{t_{per}}$$



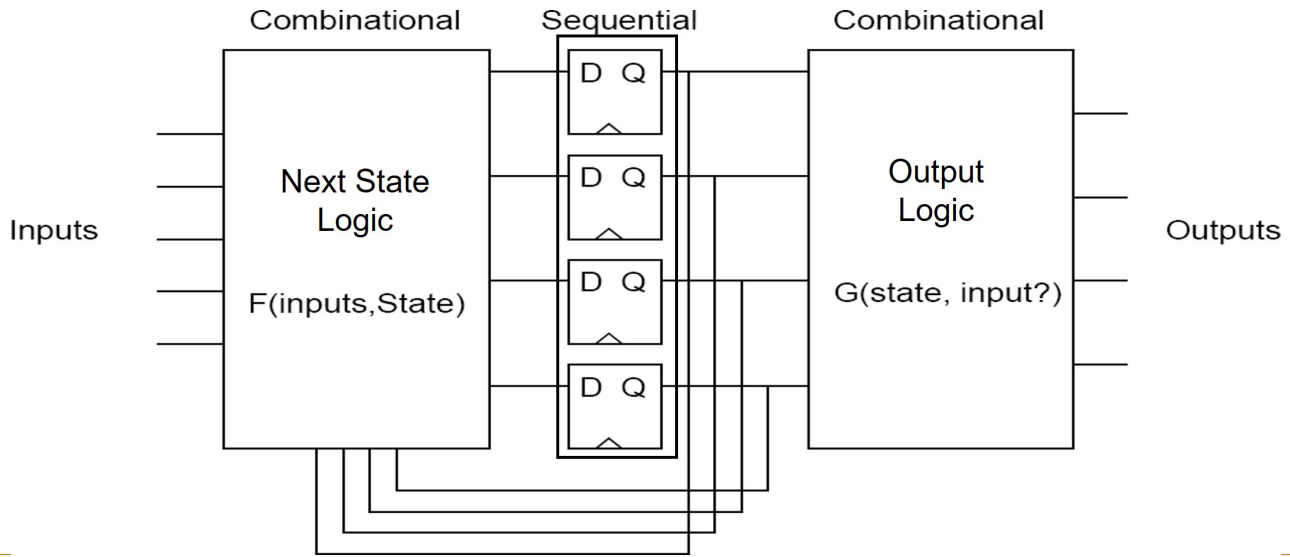
active low

Finite State Machine (FSM)

Structure

A synchronous FSM will consist of

- **Two combinational** blocks: Next-state logic F and Output logic G
- **One Sequential** block: State Memory



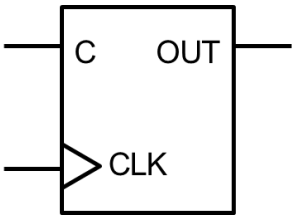
Finite State Machine (FSM)

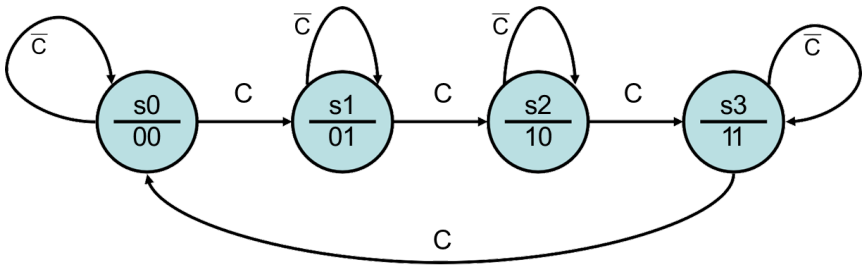
Design: State Diagrams

State Diagrams are a way to represent the FSM in one diagram

- States are symbolized by a bubble in the form of a circle or a box.
- Loops or branches between the state bubbles represent state transitions with the input condition for the transition written on them
- For a *Mealy machine,* the outputs depend on both the input and the state. Thus, outputs are written underneath the inputs on the loops/branches.
- For a *Moore machine,* outputs are written inside the state bubbles

Example: 2-bit counter (state=current count)





Finite State Machine (FSM)

Design: Tables

FSMs can also be described using tables:

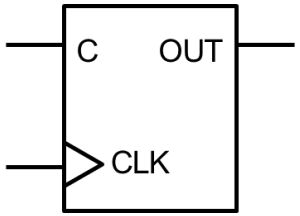
- Transition tables represent the relationship between the current state value and next state value as a function of the input
- State tables are very similar but more intuitive for complex states as the states are assigned a mnemonic description
- State/output tables also include the relationship between the state and the output. These can become complicated for Mealy machines as the output is also a function of the input.

You can draw one or more of these tables or condense them together as one single input | next state | output table.

Finite State Machine (FSM)

Design: Tables

Example: 2-bit counter (state=current count)



State Table

State name S	Next State S*	
	Input C=0	Input C=1
S0	**S0**	**S1**
S1	**S1**	**S2**
S2	**S2**	**S3**
S3	**S3**	**S0**

Transition Table

State value S	Next State S*	
	Input C=0	Input C=1
00	**00**	**01**
01	**01**	**10**
10	**10**	**11**
11	**11**	**00**

State/Output Table

State name S	Output OUT
S0	**00**
S1	**01**
S2	**10**
S3	**11**

Note: *the notation S* indicates the value of S "at the next time step" (next clock cycle).*

Finite State Machine (FSM)

Design: Key steps

1. Determine the inputs / outputs. Determine the states and give them mnemonic names
2. Draw up a state diagram.
3. Draw a state table.
4. Render the inputs and outputs in binary format, adopt an encoding for the states (you can use Verilog `parameter` to your advantage).
5. Draw a transition (next-state value) table.
6. Draw an output table
7. Use Boolean algebra (Karnaugh maps where practical) to obtain minimal combinational logic for next state and output.
8. Use the standard Verilog style to code and simulate your design and check for correct operation. Revise as appropriate.
9. Check for potential practical problems (e.g. non-ideal effects).

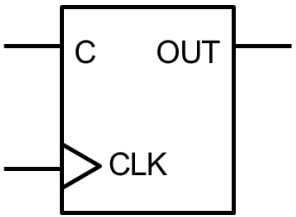
Finite State Machine (FSM)

Design: Key steps

Determining the states:

- Come up with as few states as possible:
  - Saving resources / easier to manage designs
  - Equivalent states: states associated with the same output and the same next-state transitions can be replaced by a single state.
- Use meaningful names!
- Once you have picked the states the number of required flip-flops is automatically derived

Example: 2-bit counter (state=current count)



Input: `C`, `CLK`
Output: `OUT`
States: `s0`, `s1`, `s2`, `s3`

Finite State Machine (FSM)

Design: Key steps

General rules to assign binary values to the states:

- Choose an initial state which is easy to initialize (00..0)
- Minimize the number of bit changes between adjacent states (Grey code can help)
- Exploit symmetries: if a state or group of states are very similarly related, they can be assigned the same code with the exception of 1 bit.
- If you have unused states, make use of the full range of binary numbers in order to achieve the above as much as possible
  - e.g., if you have 5 states (3 bits), don't stop at 101.
- If each bit (or subgroup of bits) has a meaning, exploiting these meanings can lead to simpler next-state logic.

Example: 2-bit counter (state=current count)

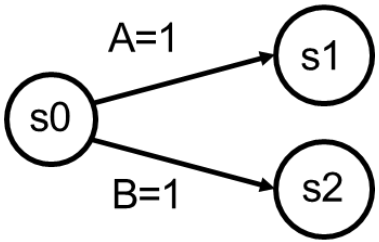
```
parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
```

Finite State Machine (FSM)

Design: Key steps

Designing good transition and output logic:

- Karnaugh maps are a good tool when working with simple designs
- When transition/output equations are complicated you can simplify the expressions using Boolean algebra to some extent.
  - But it might be hard to come up with a minimal description
  - CAD tools (synthesizers in particular) do take some of the trouble out of this by carrying out their own complexity reduction algorithms.
- If you are working with state diagrams, ensure that your diagram has no ambiguity before working on the transition logic!
  - Ambiguity can arise when different inputs cause the system to evolve to different states.



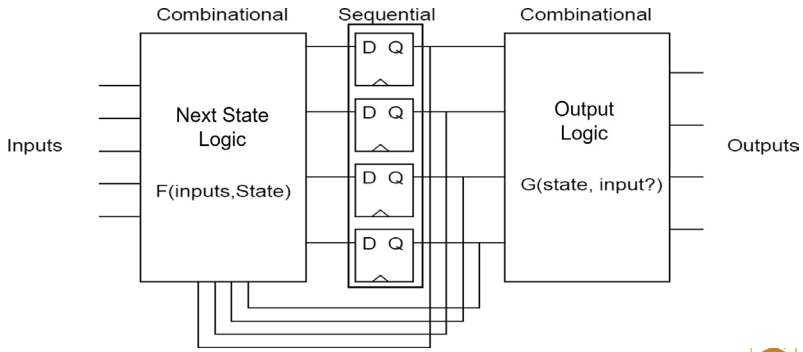
What if both A and B are 1?

Disambiguation required!

Finite State Machine (FSM)

Coding in Verilog

- State Memory (transitions): use a sequential `always` block sensitive to a clock-signal edge
- Next-State logic: use a combinational `always` block.
  - This block usually contains a case statement that enumerates all possible values of the current state.
- Output logic: use a combinational `always` block or `assign` statements.
  - It may or may not include a case statement, depending on the complexity of the output function.



Finite State Machine (FSM)

Coding in Verilog: Two-state Moore machine

```
module FSM
  (input wire clk, rst, in,
   output reg out);

parameter S0 = 1'b0, S1 = 1'b1;
reg state, Snext;

always @(posedge clk) begin
  //STATE MEMORY
  //sequential
  if(rst) state <= S0;
  else state <= Snext;
end

…

…

always @(*) begin
  //NEXT-STATE LOGIC
  //combinational
  case(state)
    S0: if(in) Snext = S1;
        else Snext = S0;
    S1: if(in) Snext = S0;
        else Snext = S1;
    default: Snext = S0;
  endcase
end

…

…

always @(*) begin
  //OUTPUT LOGIC
  //combinational
  case(state)
    S0: out = f(state);
    S1: out = g(state);
    default: out = h(state);
    //note: f,g,h arbitrary
  endcase
end

endmodule
```


Finite State Machine (FSM)

Coding in Verilog: Two-state Mealy machine

```
module FSM
  (input wire clk, rst, in,
   output reg out);

parameter S0 = 1'b0, S1 = 1'b1;
reg state, Snext;

always @(posedge clk) begin
  //STATE MEMORY
  //sequential
  if(rst) state <= S0;
  else state <= Snext;
end

…

…

always @(*) begin
  //NEXT-STATE LOGIC
  //combinational
  case(state)
    S0: if(in) Snext = S1;
        else Snext = S0;
    S1: if(in) Snext = S0;
        else Snext = S1;
    default: Snext = S0;
  endcase
end

…

…

always @(*) begin
  //OUTPUT LOGIC
  //combinational
  case(state)
    S0: out = f(in, state);
    S1: out = g(in, state);
    default: out = h(in,state);
    //note: f,g,h arbitrary
  endcase
end

endmodule
```


Finite State Machine (FSM)

Example: Simple Alarm System

Problem Statement:

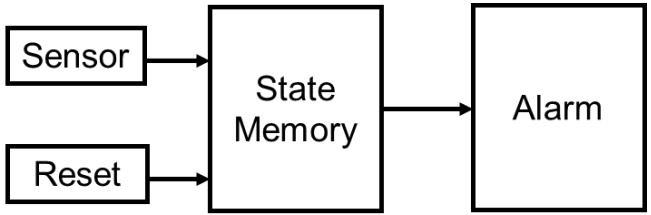
- The alarm should sound when the sensor is triggered by the burglar walking through
- The sound stops upon pressure of a reset button, but only if the sensor at that time is not picking up any intruders.

Finite State Machine (FSM)

Example: Simple Alarm System

Step 1: Identify the inputs, outputs, states

- Inputs:
  - Trigger (binary signal) – Sensor triggered/non triggered
  - Reset button (binary) - pressed/released
- Outputs:
  - Alarm bell (binary signal) - ON/OFF
- States:
  - Armed - Alarm is quiet
  - Sound - Alarm starts sound



Output is a function of the state alone

- When the state is Sound the output is 1 (Alarm ON)
- When the state is Armed the output is 0 (Alarm OFF)

This FSM is a *Moore machine*

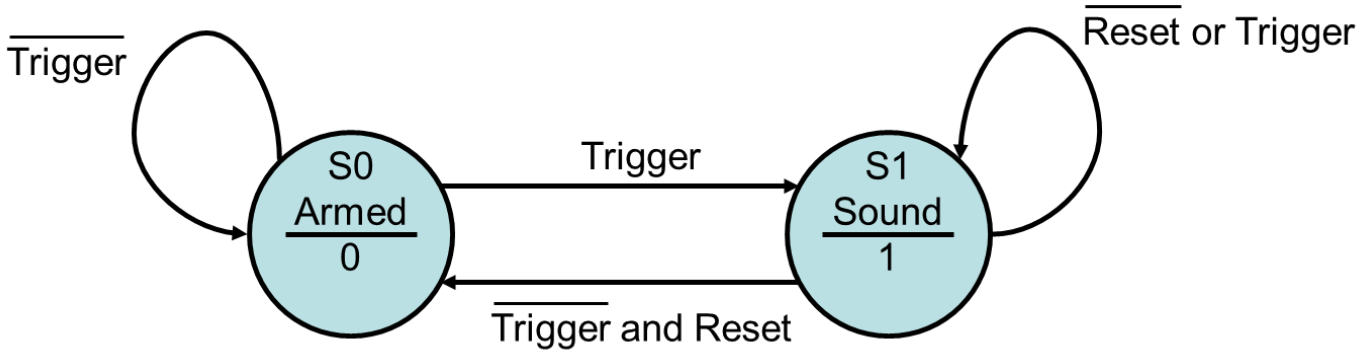
Finite State Machine (FSM)

Example: Simple Alarm System

Step 2: State diagram

Refer to the problem statement:

- The alarm should sound when the sensor is triggered by the burglar walking through
- The sound stops upon pressure of a reset button, but only if the sensor at that time is not picking up any intruders.



Finite State Machine (FSM)

Example: Simple Alarm System

Step 3: State table

- `A:` Armed
- `S:` Sound
- `T:` Trigger signal
- `R:` Reset signal

Step 4: Render the inputs, outputs, states in binary format

```
parameter A = 1'b0, S = 1'b1;
```

Step 5: Transition (next-state value) table

State Table

State	Input	Next State
A	$\bar{T}\bar{R}$	**A**
A	$\bar{T}R$	**A**
A	$T\bar{R}$	**S**
A	TR	**S**
S	$\bar{T}\bar{R}$	**S**
S	$\bar{T}R$	**A**
S	$T\bar{R}$	**S**
S	TR	**S**

Transition Table

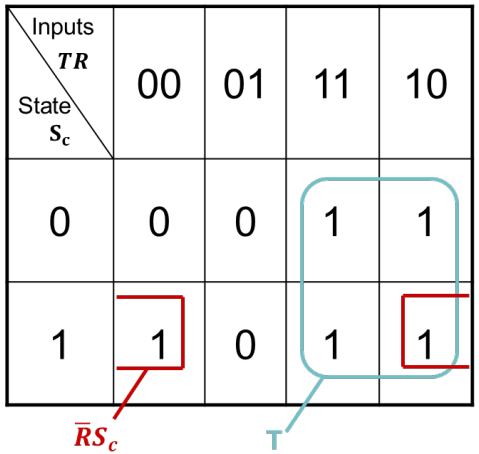
State	Input	Next State
0	$\bar{T}\bar{R}$	**0**
0	$\bar{T}R$	**0**
0	$T\bar{R}$	**1**
0	TR	**1**
1	$\bar{T}\bar{R}$	**1**
1	$\bar{T}R$	**0**
1	$T\bar{R}$	**1**
1	TR	**1**

Finite State Machine (FSM)

Example: Simple Alarm System

Step 6: Output table

Step 7: Use Boolean algebra (Karnaugh maps where practical) to obtain minimal next state and output combinational logic



Next state equation: $S_n = T + \bar{R}S_c$

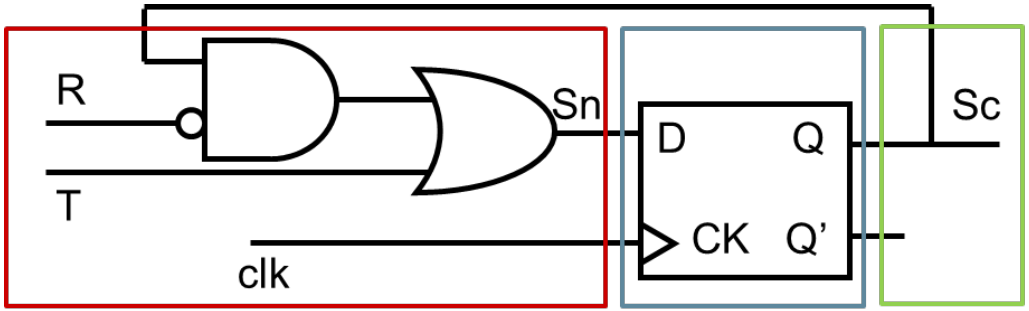
Next-state & Output Table

Current State S_c	Input	Next State S_n	Output
A (0)	$\bar{T}\bar{R}$	**A (0)**	**OFF**
A (0)	$\bar{T}R$	**A (0)**	**OFF**
A (0)	$T\bar{R}$	**S (1)**	**ON**
A (0)	TR	**S (1)**	**ON**
S (1)	$\bar{T}\bar{R}$	**S (1)**	**ON**
S (1)	$\bar{T}R$	**A (0)**	**OFF**
S (1)	$T\bar{R}$	**S (1)**	**ON**
S (1)	TR	**S (1)**	**ON**

Finite State Machine (FSM)

Example: Simple Alarm System

- **Next-state logic**: $S_n = T + \bar{R}S_c$
- **State Memory**: 1-bit state variable = 1 flip flop
- **Output Logic**: Output = state (the output logic is trivial)



Finite State Machine (FSM)

Example: Simple Alarm System

Step 8: Code and simulate your design in Verilog

```
module MyAlarm(
 input wire T, R, clk,
 output wire alarm_ON);

 reg Scurrent, Snext;

 always @(*)
     //Next-state logic
     Snext=T|(~R & Scurrent);

 always @(posedge clk)
     //State memory (flip flop)
     Scurrent <=Snext;

 //Output logic
 assign alarm_ON=Scurrent;

endmodule
```

```
parameter A = 1'b0, S = 1'b1;

always @(*) begin
//Next-state logic
 case(Scurrent)
   A: if(T) Snext = S;
      else Snext = A;
   S: if(T==0 & R==1) Snext = A;
       else Snext = S;
  default: Snext = A;
 endcase
end
```

Note: The next-state logic can be implemented in two ways as shown. The expression `Snext=T|(~R & Scurrent);` will be synthesized using gates, whereas the synthesis of case statements requires multiplexers.

Announcements

To do in Week - 4:

- Feedback Survey 1: opens on Monday, 13 March 2023, 9:00 AM in Wattle (Communication and Feedback)
- Attend in-class lectures
  - Tuesday: Finite state machines design examples
  - Wednesday: Wrap up of FPGA concepts
- Lab-3: Read the lab manual and try to attempt the tasks before attending your session.
  - Seven-segment Display (SSD)
  - Multiple SSDs using a display driver
  - Switch debouncer
- Tutorial-3: Download Tutorial-3 problem document and try to attempt the problem questions.
  - Verilog
- Watch pre-recorded videos before next Tuesday lecture
  - VL7: Practical aspects of design
  - VL8: RAM and ROM memories

THANK YOU

Acknowledgement

These presentation slides are the modified version of slides prepared by Dr. Jihui (Aimee) Zhang based on the original version by Dr. Nicolo Malagutti

Contact:

Amy Bastine
School of Engineering
ANU College of Engineering, Computing and Cybernetics
The Australian National University

Office: B147, Brian Anderson Building (Bldg. 115)
Email: amy.bastine@anu.edu.au

Australian National University