# Lab 7
# Verilog – Sequential Design(2)

**Johnson Liu**

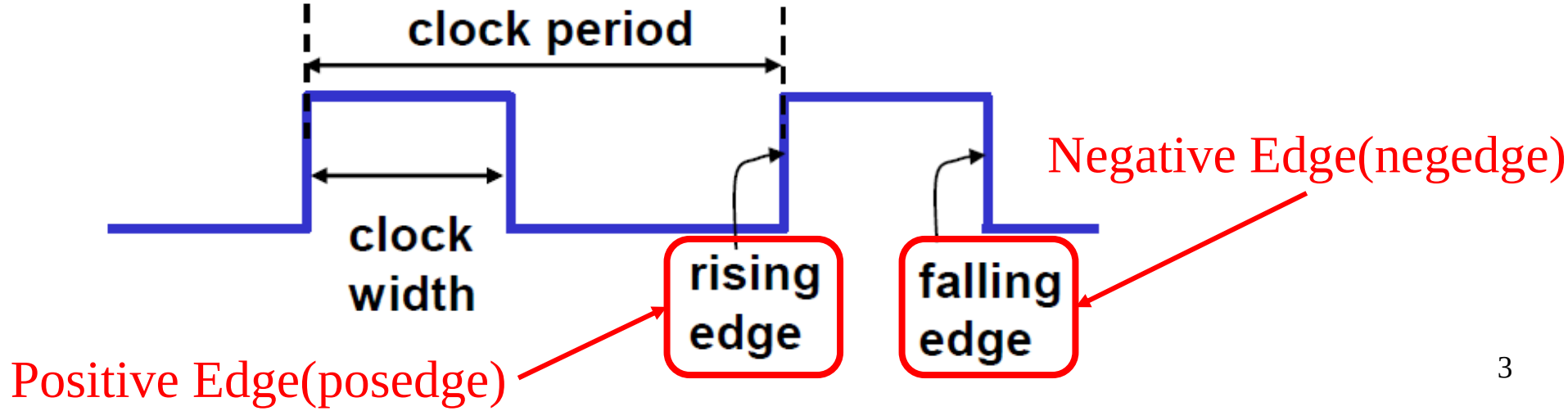
Department of Electrical Engineering
National Cheng Kung University

# Outline

1. Synchronous Reset vs. Asynchronous Reset
2. Register
3. Implementation(1) Johnson Counter
4. Blocking vs. Non-Blocking
5. Implementation(2) Stack
6. Implementation(3) Queue
7. TA Checking & Laboratory Report
8. Reference

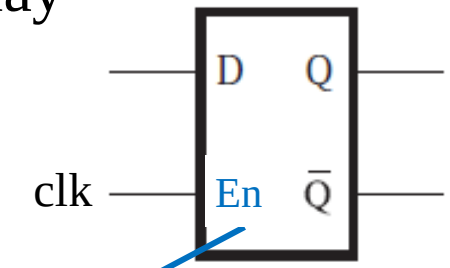
# Recall: Clock

- Clock period: the time between successive transitions in the same direction. (second/cycle)
- Clock frequency: the reciprocal of clock period. (cycle/second)
- Clock width: the time interval during which clock is equal to 1.
- Duty cycle: the ratio of the clock width and clock period
- Active high: the circuit changes occur at the rising edge or during the logic is 1.
- Active low: the circuit changes occur at the falling edge or during the logic is 0.

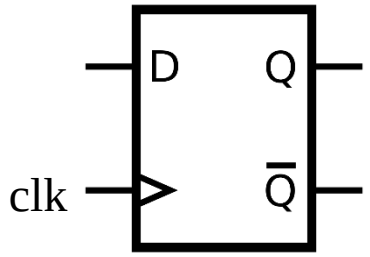


# Recall: Latch & Flip-Flop

- The state of a latch or flip-flop is switched by a change of the control input like clock.
- Flip-flops and latches are used as data storage elements.
- D: Data/Delay




D-Latch




D-FF

Level Trigger

| E/C | D | Q | $\bar{Q}$ | Comment |
|---|---|---|---|---|
| 0 | X | $Q_{prev}$ | $\bar{Q}_{prev}$ | No change |
| 1 | 0 | 0 | 1 | Reset |
| 1 | 1 | 1 | 0 | Set |

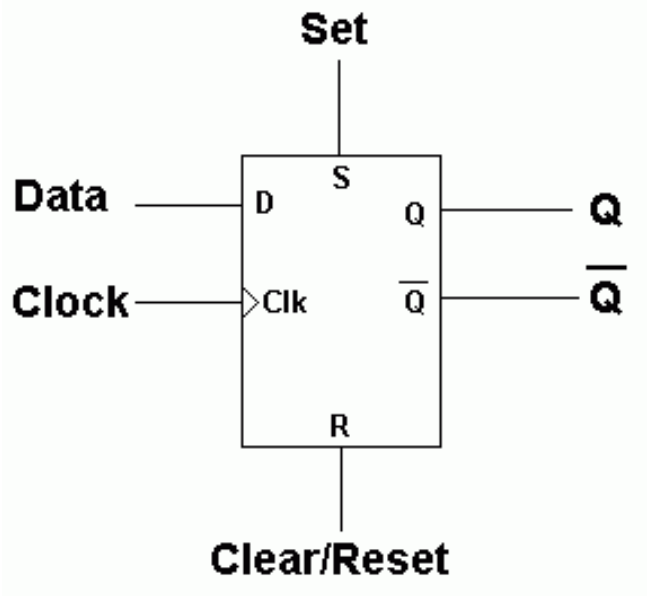
Edge Trigger

| Clock | D | $Q_{next}$ |
|---|---|---|
| Rising edge | 0 | 0 |
| Rising edge | 1 | 1 |
| Non-rising | X | Q |

# Synchronous Reset
# vs.
# Asynchronous Reset

# Reset

- A reset clears any pending errors or events and brings a system to normal condition or an initial state.
- Two types of Reset in Verilog:
  1. Synchronous Reset
  2. Asynchronous Reset

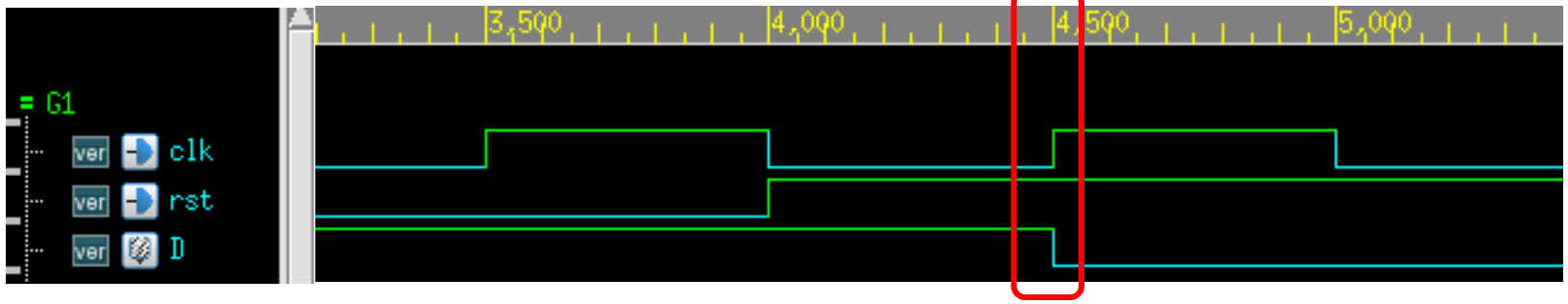


# Synchronous Reset

- Synchronous Reset will reset the Flip-Flop when clock is triggered and reset signal is active.

```
module ResetSynchronous(clk, rst);
    input clk, rst;
    reg D;

    always @(posedge clk)
    begin
        if(rst)
        begin
            D <= 1'd0;
        end
        else
        begin
            D <= 1'd1;
        end
    end
endmodule
```

Synchronous

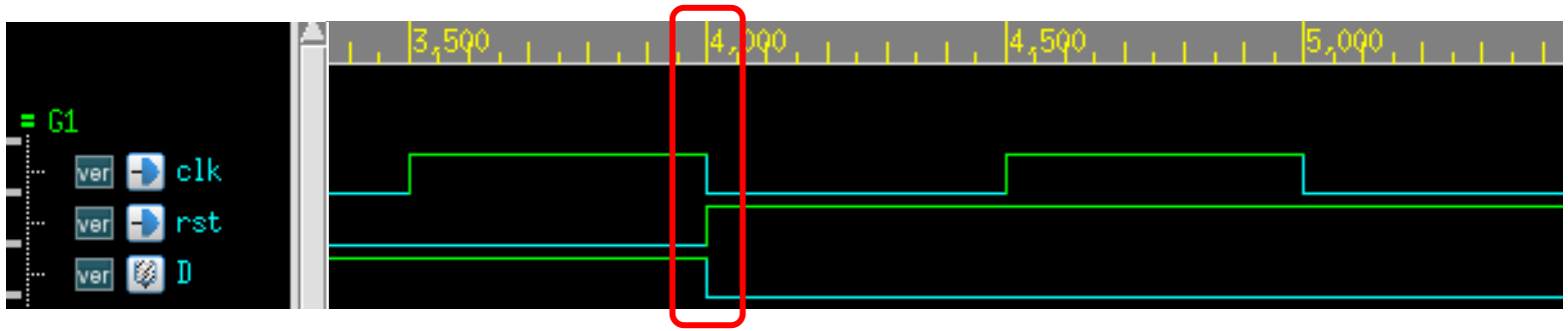


# Asynchronous Reset

- Asynchronous Reset will reset the Flip-Flop whenever reset signal is active.

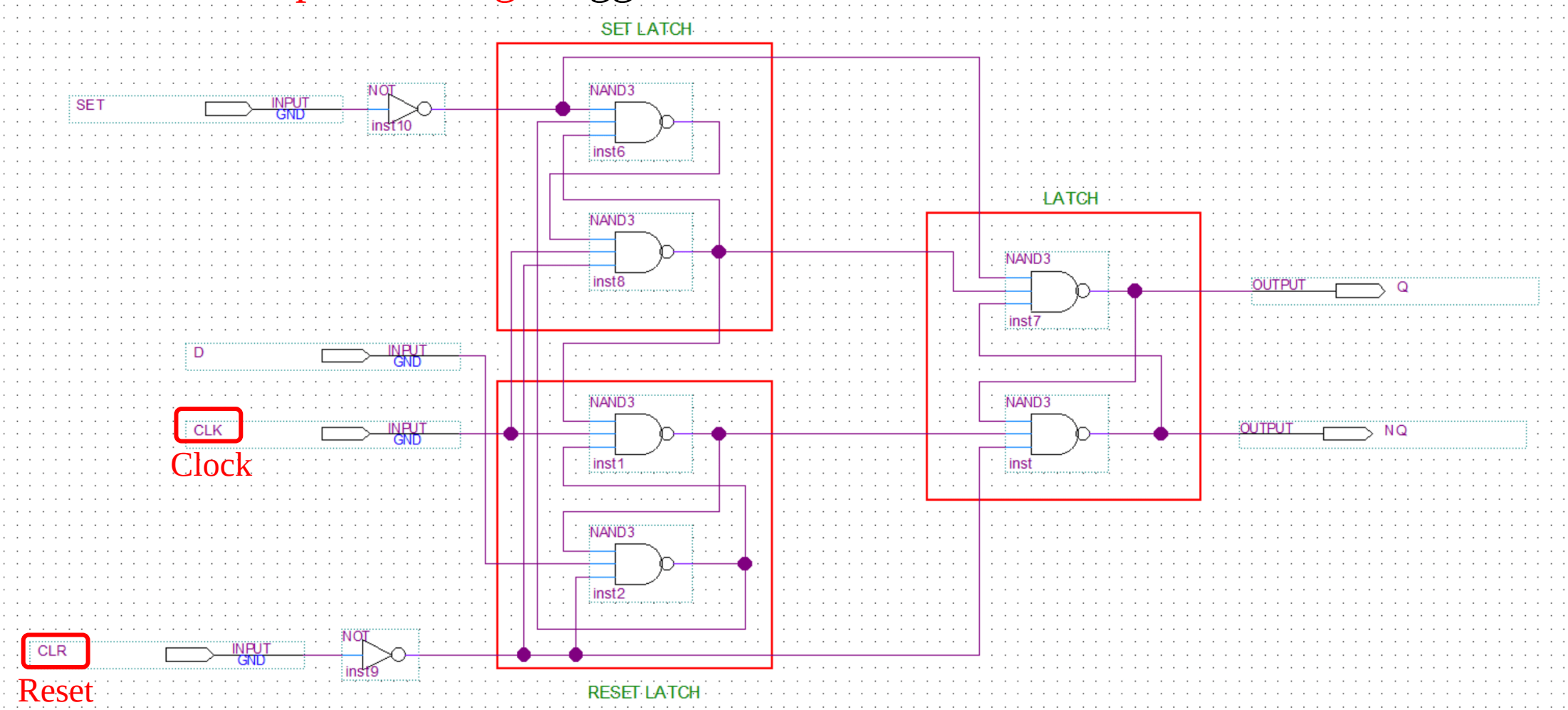
```
module ResetSynchronous(clk, rst);
    input clk, rst;
    reg D;

    always @(posedge clk or posedge rst)
    begin
        if(rst)
        begin
            D <= 1'd0;
        end
        else
        begin
            D <= 1'd1;
        end
    end
endmodule
```

Asynchronous



# Asynchronous Set/Reset D Flip-Flop

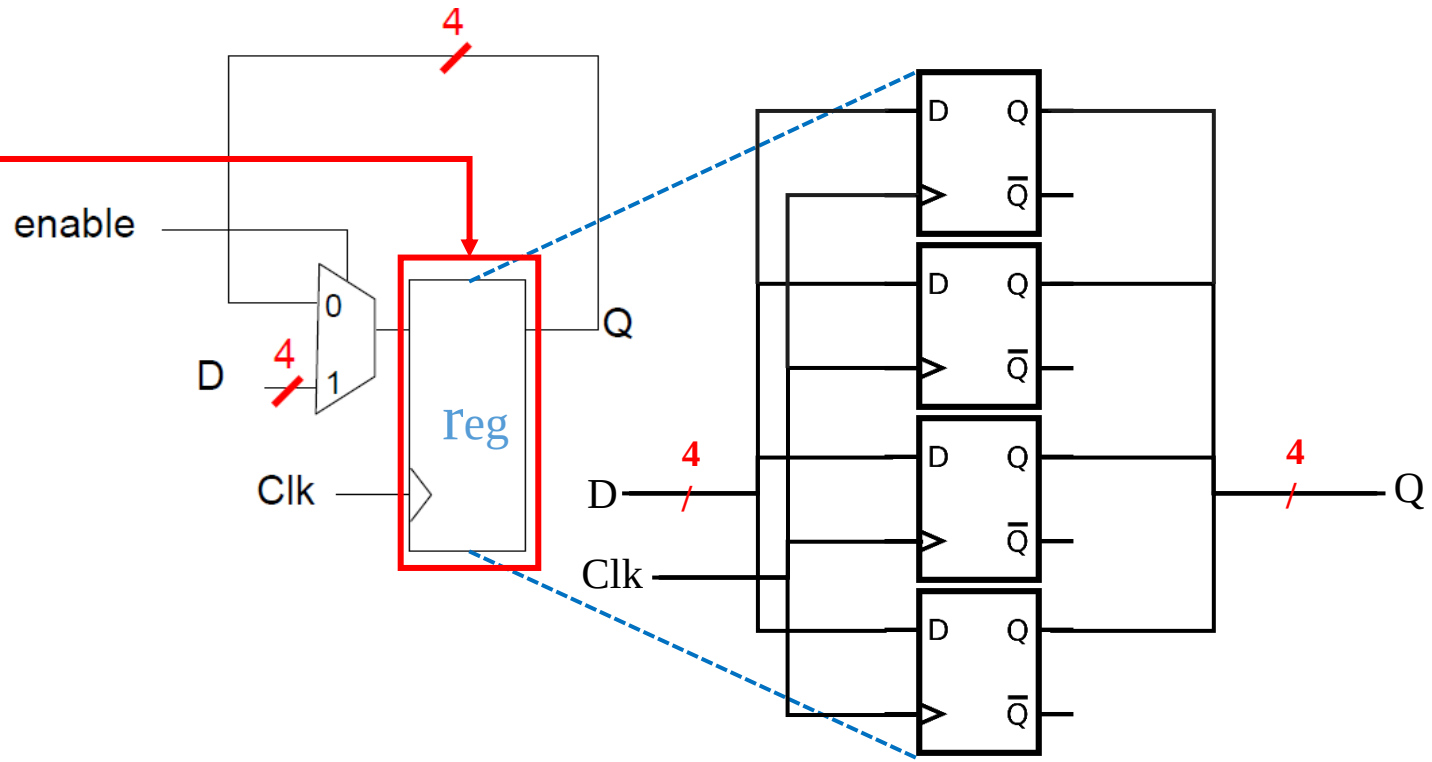
- Set & Reset are active-high.
- Clock is positive edge triggered.



# Register

# Register

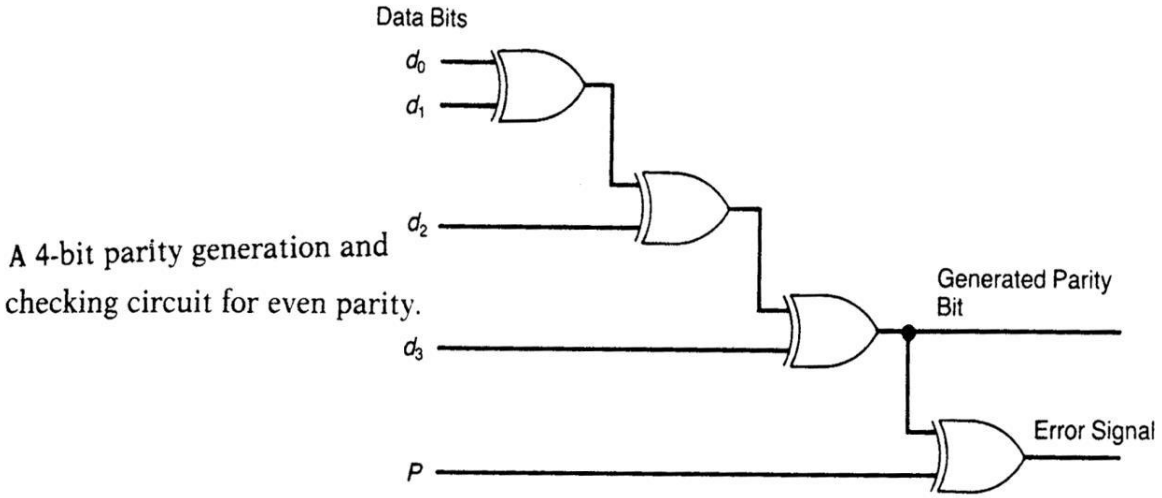
```
module DFF_REG (clk, enable, D, Q);
    input clk, enable;
    input [3:0] D;
    output [3:0] Q;
    reg [3:0] Q;
    always @(posedge clk)
    begin
        if (enable)
        begin
            Q <= D;
        end
        else
        begin
            Q <= Q;
        end
    end
endmodule
```



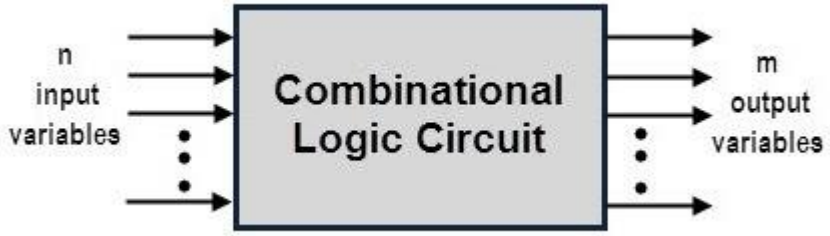
# Blocking
# vs.
# Non-Blocking

# Recall: Combinational Circuit

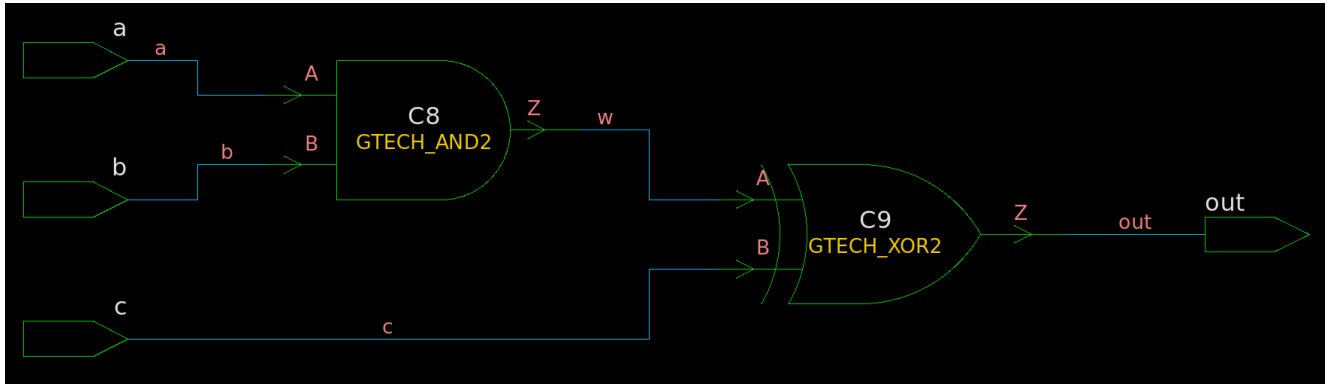
- A combinational circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs.




A 4-bit parity generation and checking circuit for even parity.



# Recall: Blocking Assignment




A

```
module Blocking(a, b, c, out);
    input a, b, c;
    output out;
    // Wire
    reg out;
    always @(*)
    begin
        out = (a & b) ^ c;
    end
endmodule
```

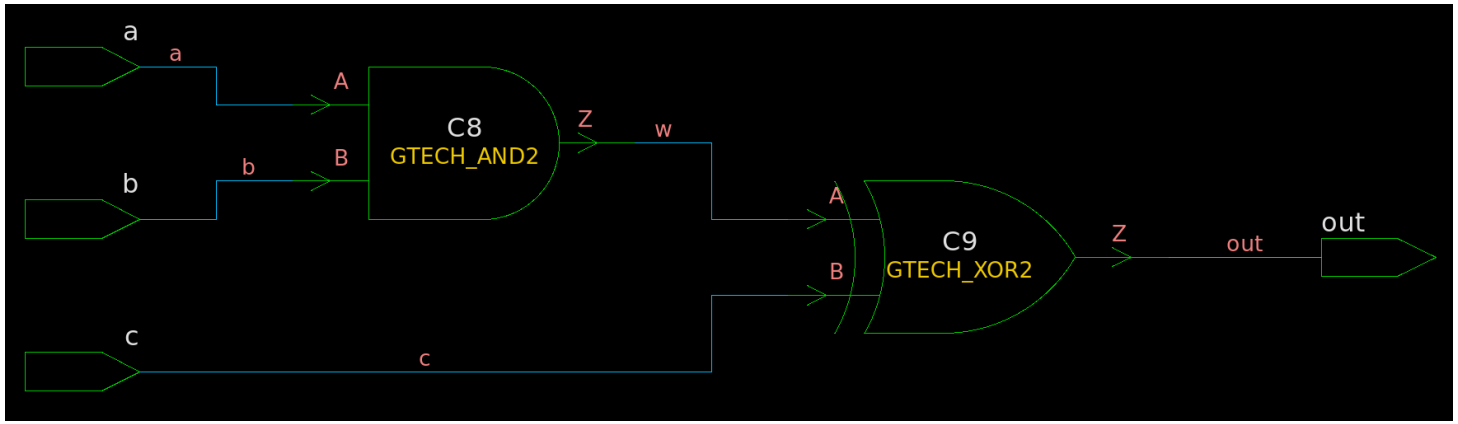
B

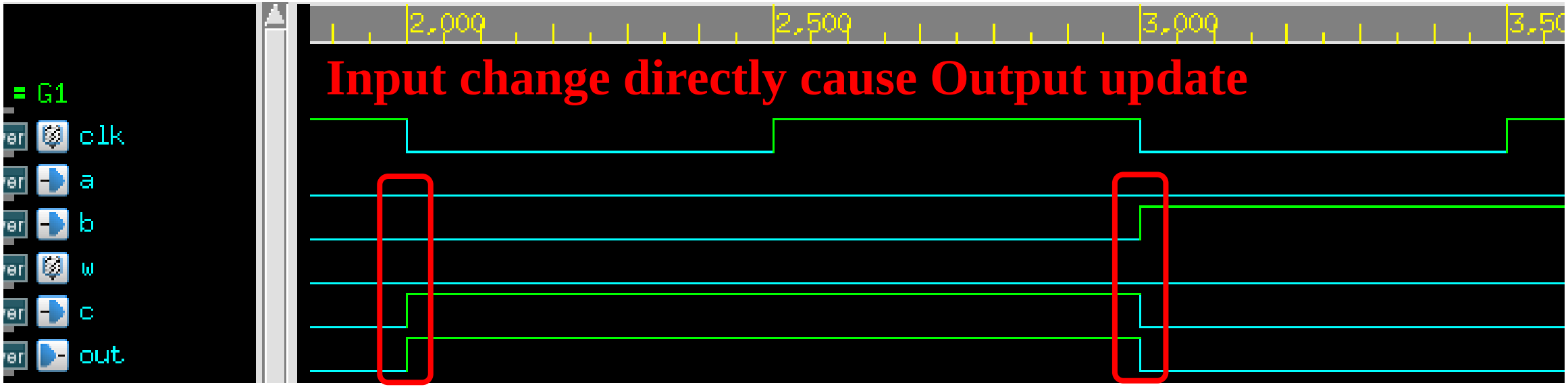
```
module Blocking(a, b, c, out);
    input a, b, c;
    output out;
    // Wire
    reg w;
    reg out;
    always @(*)
    begin
        w = a & b;
        out = w ^ c;
    end
endmodule
```

C

```
module Blocking(a, b, c, out);
    input a, b, c;
    output out;
    // Wire
    reg w;
    reg out;
    // Continuous Assignment
    assign w = a & b;
    assign out = w ^ c;
endmodule
```

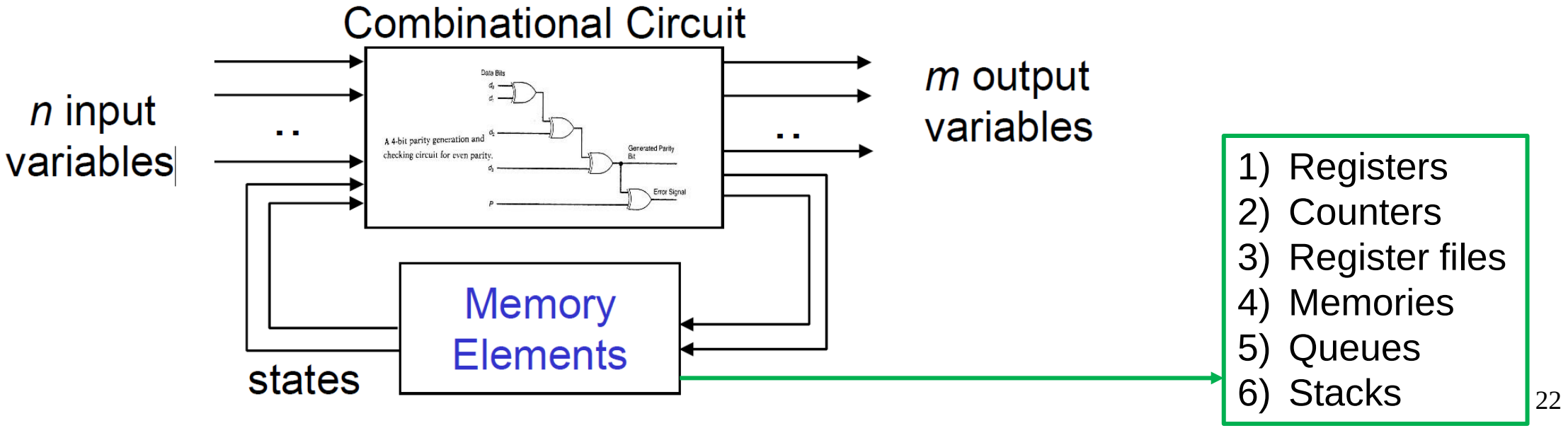
# Recall: Blocking Assignment Waveform



# Recall: Sequential Circuit

- A sequential circuit is a system whose outputs at any time are determined from the present combination of inputs and the previous inputs or outputs, so sequential components contain memory elements.
- Sequential circuit can be break into two phases:
  1. Evaluate (Before Clock Edge) => Combinational Circuit
  2. Update (On Clock Edge) => Memory Element

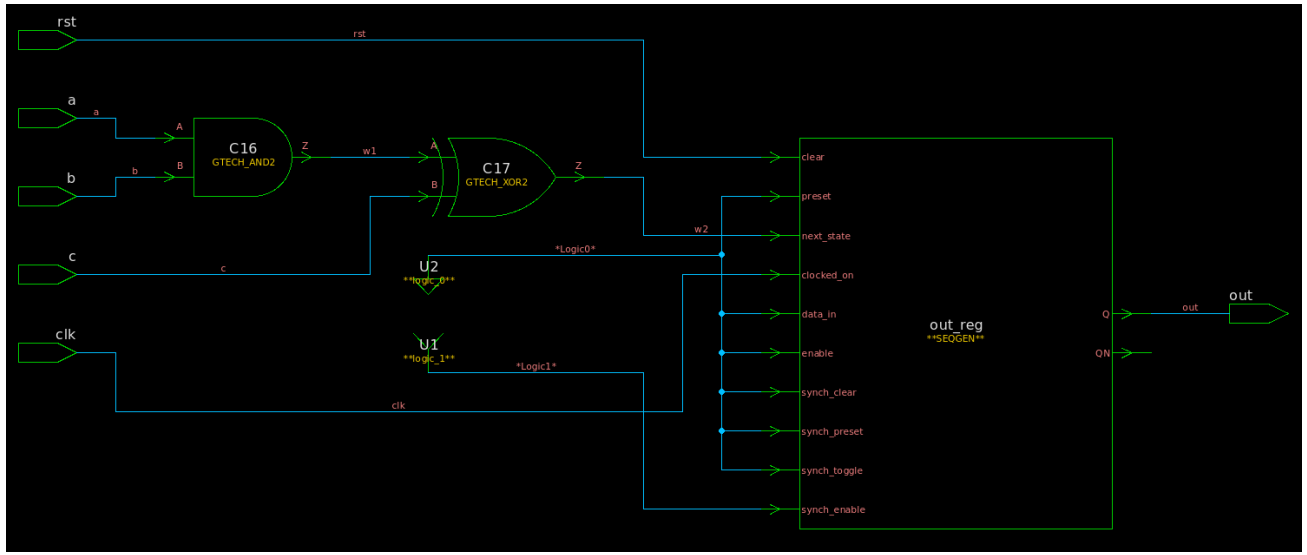


# Non-Blocking Assignment

```
module NonBlocking(clk, rst, a, b, c, out);
    input clk, rst, a, b, c;
    output out;
    // Register
    reg out;
    always @(posedge clk or posedge rst)
    begin
        if (rst)
        begin
            out <= 1'b0;
        end
        else
        begin
            out <= (a & b) ^ c;
        end
    end
endmodule
```

Sequential Block

- Non-Blocking Assignment: <=
- Use in Sequential Block (Always Block with Clock)



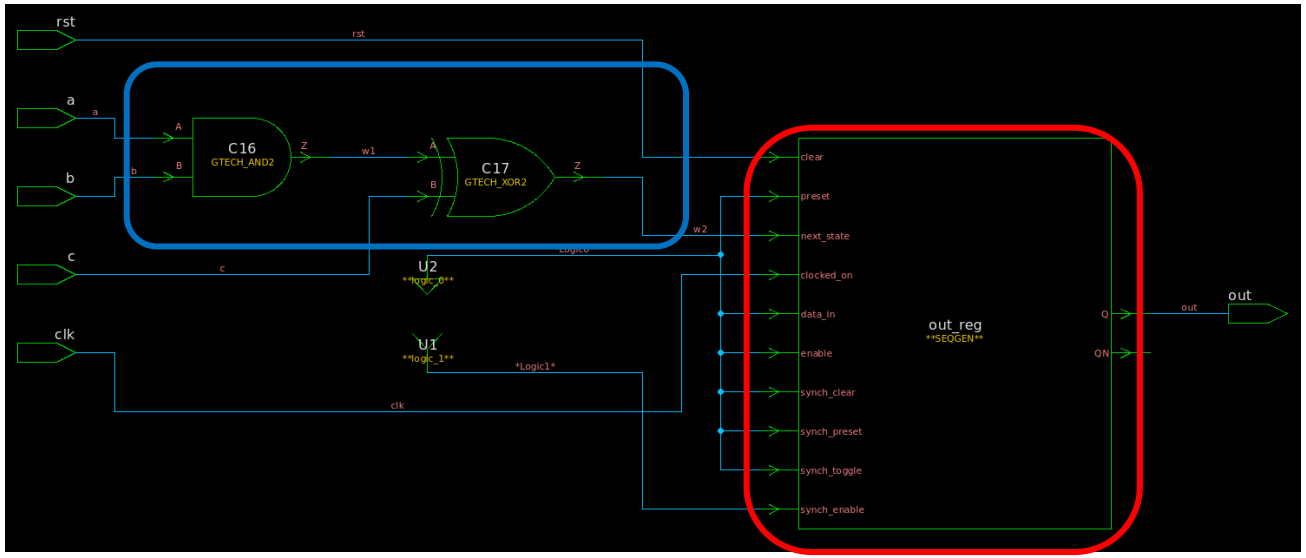
# Non-Blocking Assignment

```
module NonBlocking(clk, rst, a, b, c, out);
    input clk, rst, a, b, c;
    output out;
    // Wire
    reg w1;
    reg w2;
    // Register
    reg out;
    // NonBlocking Assignment
    always @(posedge clk or posedge rst)
    begin
        if (rst)
        begin
            out <= 1'd0;
        end
        else
        begin
            out <= w2;
        end
    end
```

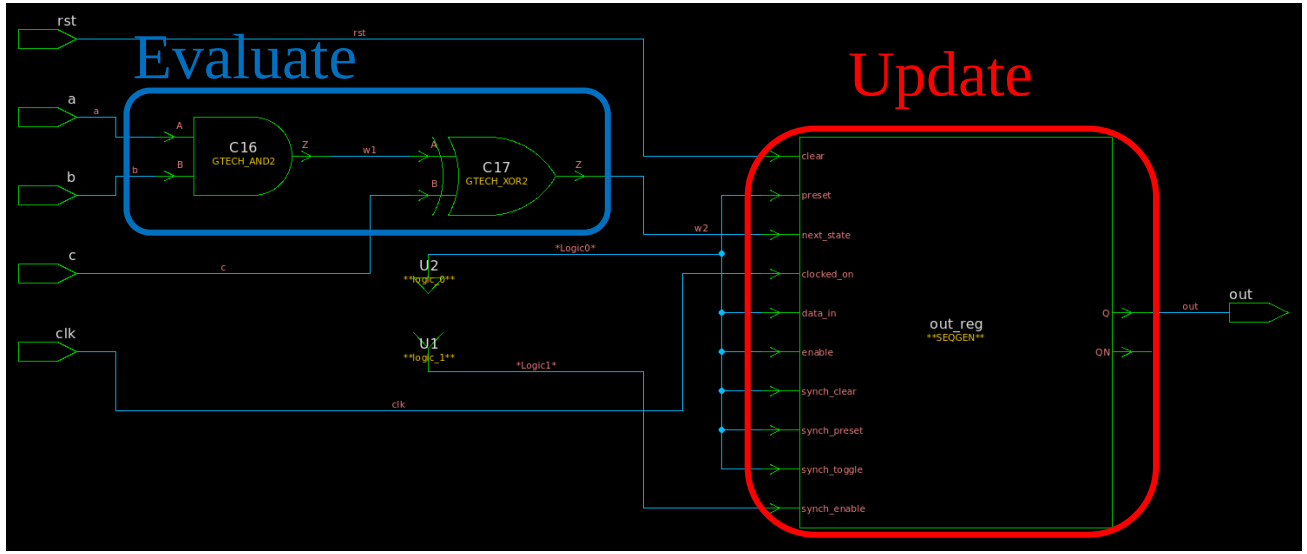
Sequential Block

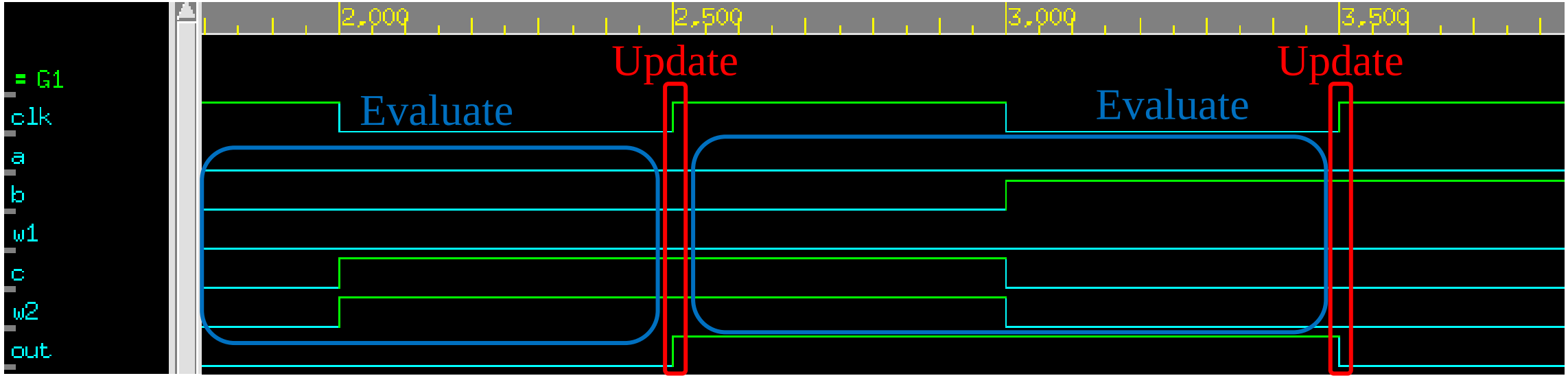
```
    // Blocking Assignment
    always @(*)
    begin
        w1 = a & b;
        w2 = w1 ^ c;
    end
endmodule
```

Combinational Block



# Non-Blocking Assignment Waveform



# Different in Assignment

- Blocking Assignment (=) are order sensitive
- Non-Blocking Assignment (<=) are order independent Blocking

1. Blocking

```
initial
begin
    a = #12 1;
    b = #3 0;
    c = #2 3;
end
```

2. Non-Blocking

```
initial
begin
    d <= #12 1;
    e <= #3 0;
    f <= #2 3;
end
```

- initial: Simulation start at 0.
- #n: Delay of n time units.

Time →

| Timestamp | 0 | 2 | 3 | 12 | 15 | 17 |
|---|---|---|---|---|---|---|
| a | x | x | x | 1 | 1 | 1 |
| b | x | x | x | x | 0 | 0 |
| c | x | x | x | x | x | 3 |
| d | x | x | x | 1 | 1 | 1 |
| e | x | x | 0 | 0 | 0 | 0 |
| f | x | 3 | 3 | 3 | 3 | 3 |

# Test Yourself

1. Blocking

```
initial
begin
   ...
   A = 1;
   B = 0;
   ...
   A = B;
   B = A;
```

B = ? is used
A = ? is used

```
initial
begin
   ...
   A = 1;
   B = 0;
   ...
   B = A;
   A = B;
```

A = ? is used
B = ? is used

---

2. Non-Blocking

```
initial
begin
   ...
   A <= 1;
   B <= 0;
   ...
   A <= B;
   B <= A;
```

B = ? is used
A = ? is used

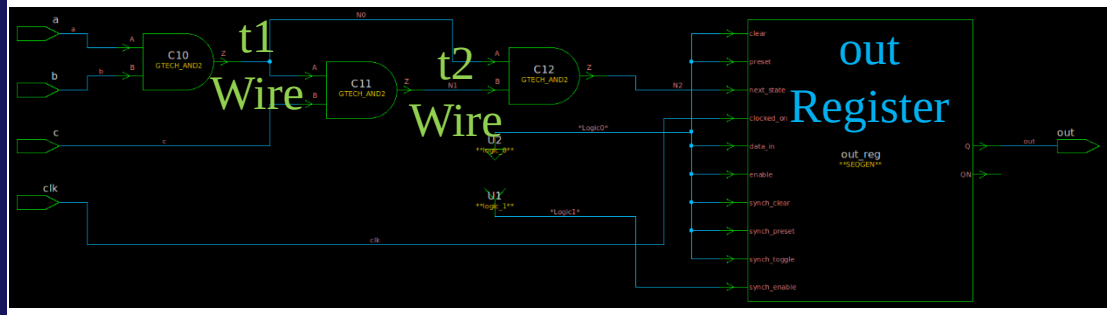
```
initial
begin
   ...
   A <= 1;
   B <= 0;
   ...
   B <= A;
   A <= B;
```

A = ? is used
B = ? is used

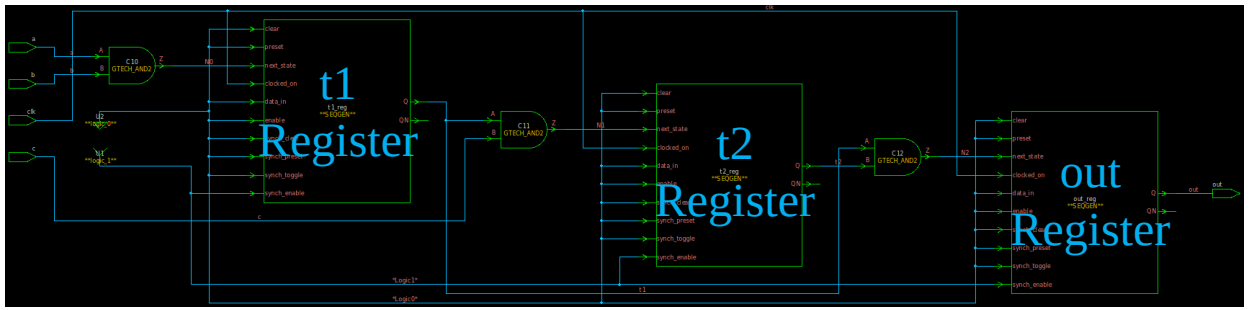
！ ！ ！LHS in initial or always statements should declare as “reg”. ！ ！ ！

# Different in Sequential Block

```
module test(clk, a, b, c, out);
    input clk, a, b, c;
    output out;
    reg t1, t2;
    reg out;
    always @(posedge clk)
    begin
        t1 = a & b;        ①
        t2 = t1 & c;       ②
        out = t1 & t2;     ③
    end
endmodule
```




```
module test(clk, a, b, c, out);
    input clk, a, b, c;
    output out;
    reg t1, t2;
    reg out;
    always @(posedge clk)
    begin
        t1 <= a & b;       ①
        t2 <= t1 & c;      ① old t1 is used
        out <= t1 & t2;    ① older t1 & old t2 is used
    end
endmodule
```



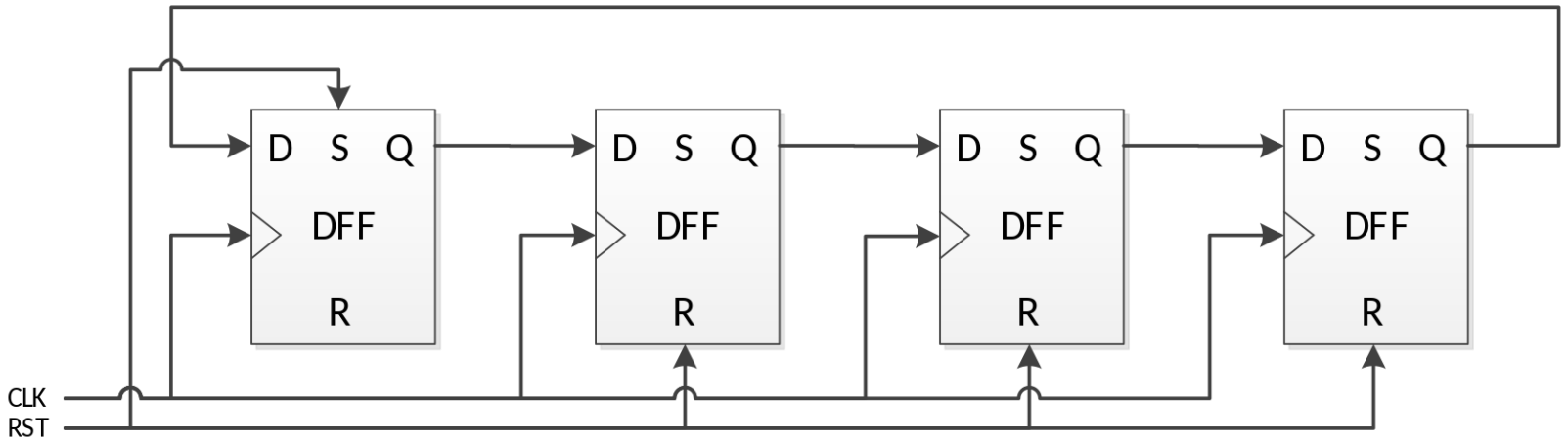
# Implementation(1)
# Johnson Counter

# Ring counter

- A ring counter is a type of counter composed of flip-flops connected into a shift register.
- The output of the last flip-flop fed to the input of the first, making a "circular" or "ring" structure.

| Straight ring counter | | | | |
|---|---|---|---|---|
| State | Q0 | Q1 | Q2 | Q3 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 |
| *0* | 1 | 0 | 0 | 0 |
| *1* | 0 | 1 | 0 | 0 |
| *2* | 0 | 0 | 1 | 0 |
| *3* | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 |

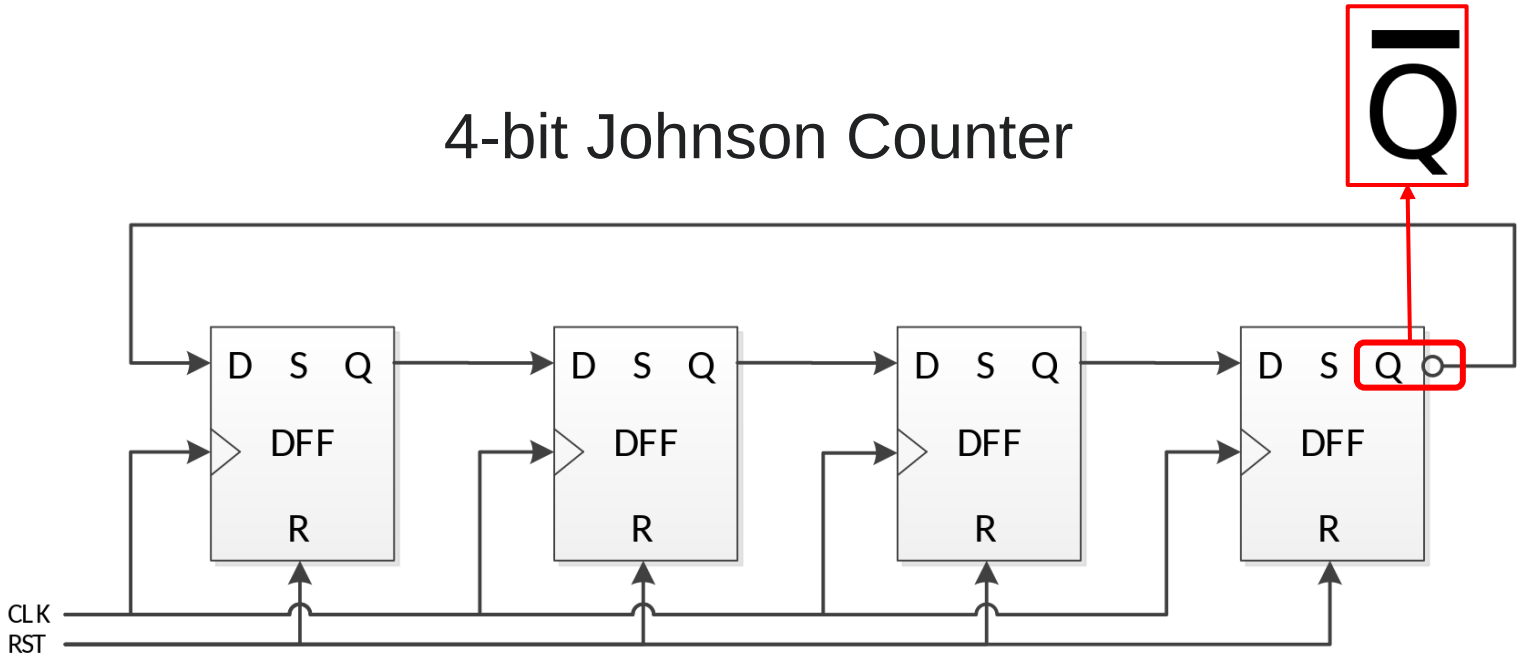
4-bit Straight Ring Counter



# Johnson Counter

- Johnson Counter can represent 2N states, while Straight Ring Counter can only represent N states. (N is the number of bits in the code)
- Inversion of the Q signal from the last shift register feeding back to the first shift register's D input.

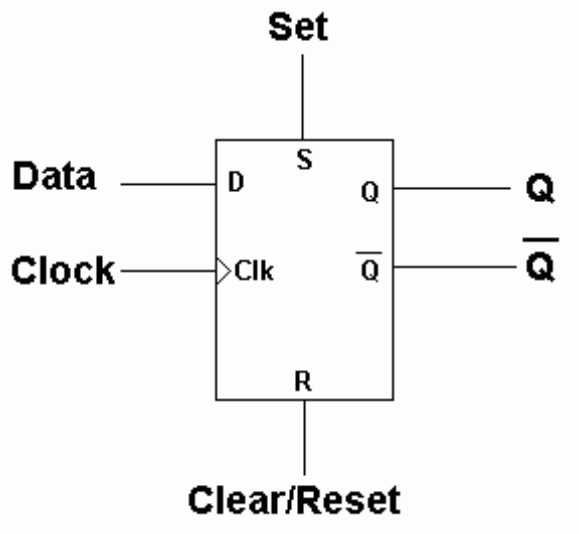
| Johnson counter | | | | |
|---|---|---|---|---|
| State | Q0 | Q1 | Q2 | Q3 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 |
| 4 | 1 | 1 | 1 | 1 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 0 | 1 |
| *0* | 0 | 0 | 0 | 0 |



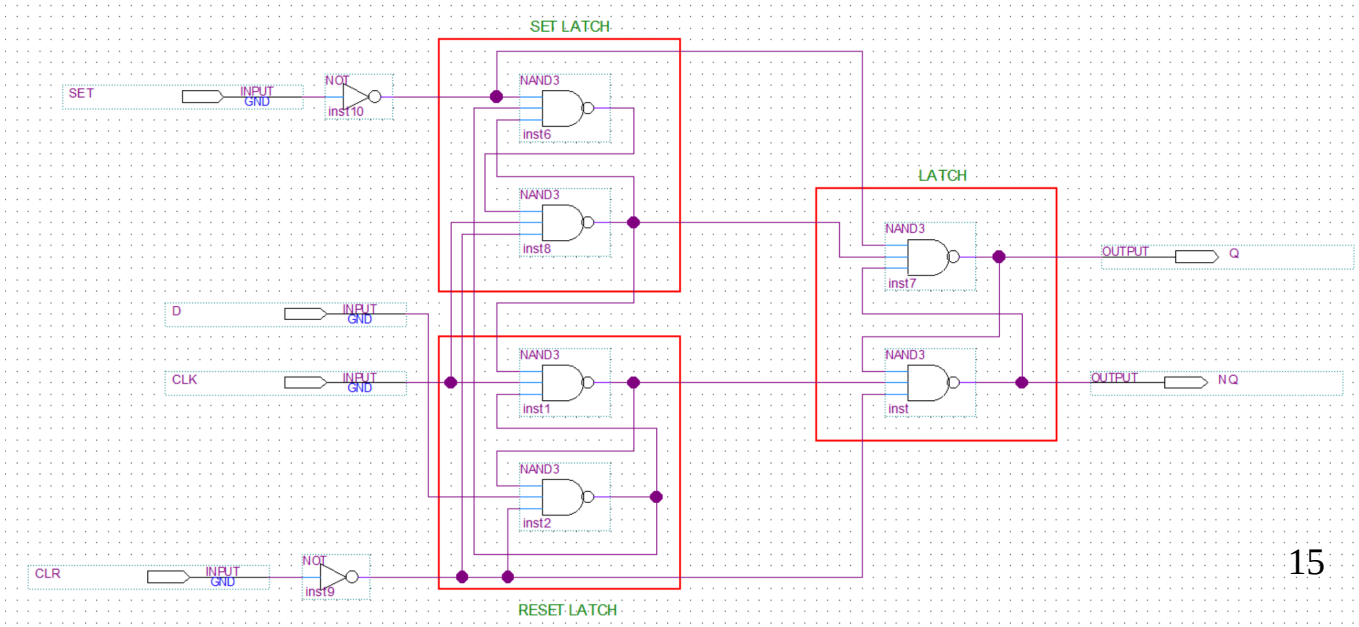
# Implement 6-bit Johnson Counter

- Draw 6-bit Johnson Counter circuit with following D Flip-Flop Template, and paste it into Report.
- Use 6 Asynchronous Set/Reset D Flip-Flop to create 6-bit Johnson Counter, and pass TA's testbench.

D Flip-Flop Template




Asynchronous Set/Reset D Flip-Flop



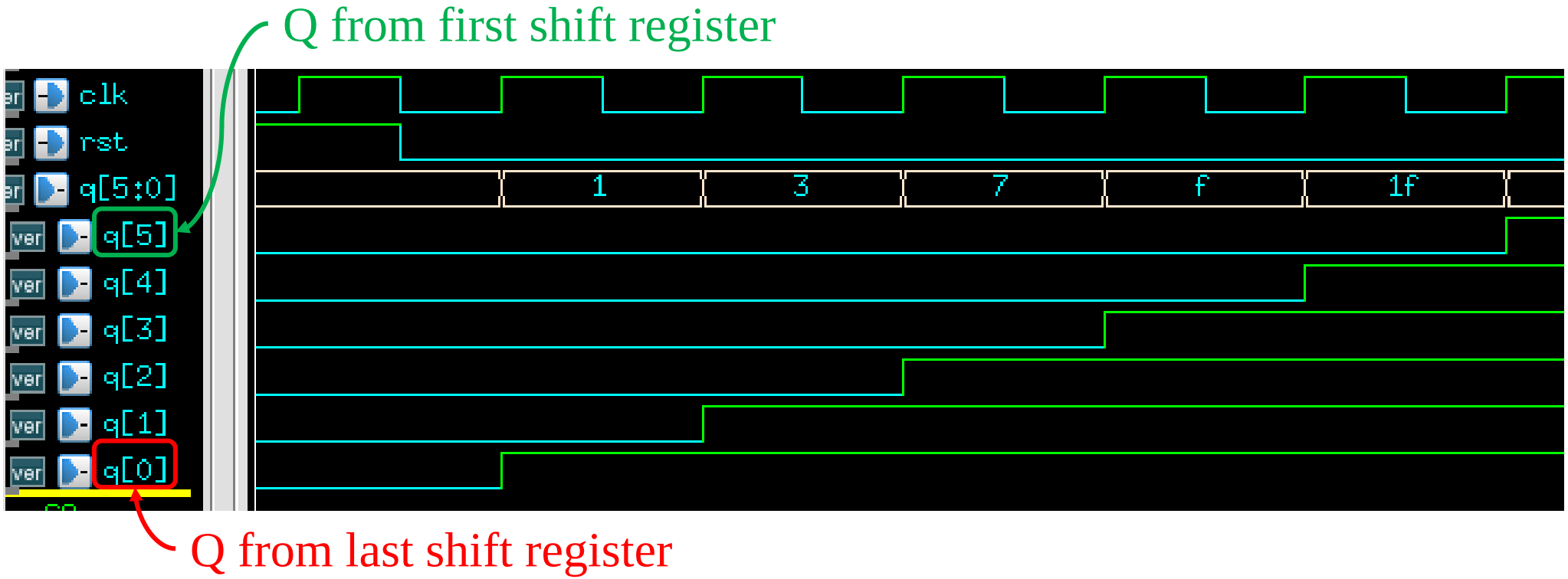
# Implement 6-bit Johnson Counter

- I/O Interface of Module JohnsonCounter in JohnsonCounter.v

| Name | I/O | Width | Description |
|---|---|---|---|
| clk | I | 1 | System clock signal. This system is synchronized with the positive edge of the clock. |
| rst | I | 1 | Active-high asynchronous reset signal. |
| q | O | 6 | A Vector with element of Q6～Q1 from D-FF, with MSB from Q6 and LSB from Q1.<br>D-FF 1 is first shift register.<br>D-FF 6 is last shift register. |

# Implement 6-bit Johnson Counter

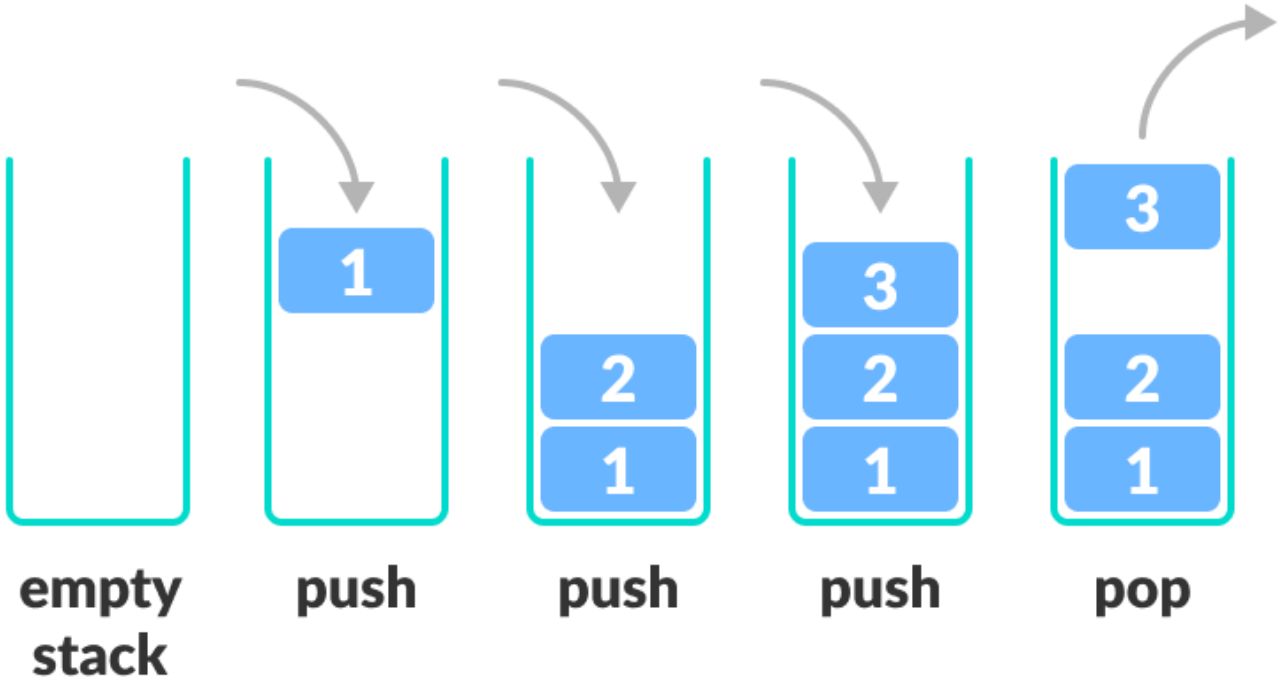
- Waveform of 6-bit Johnson Counter



# Implementation(2)
# Stack

# Stack

- A stack is an abstract data type that serves as a collection of elements.
- Two main principal operations:
  1. Push => Adds an element to the collection.
  2. Pop => Removes the most recently added element.
- The order in which elements come off a Stack gives rise to its alternative name, LIFO (Last In, First Out).



# Implement Stack with 8 Element

- Each element in Stack is 8-bit width.
- Three operation support:
  1. Push => Adds an 8-bit element to the stack.
  2. Pop => Removes an 8-bit element from the stack.
  3. Clear => Clear all element in the stack, and don't need to output.
- Recommendation:
  1. Use the combination of Vector and Array to create Stack's memory element, for example `reg [7:0] memory [7:0];`
  2. Use a variable to indicate which element is the newest one.
  3. Evaluate in Combinational Block.
  4. Update register & memory element in Sequential Block.

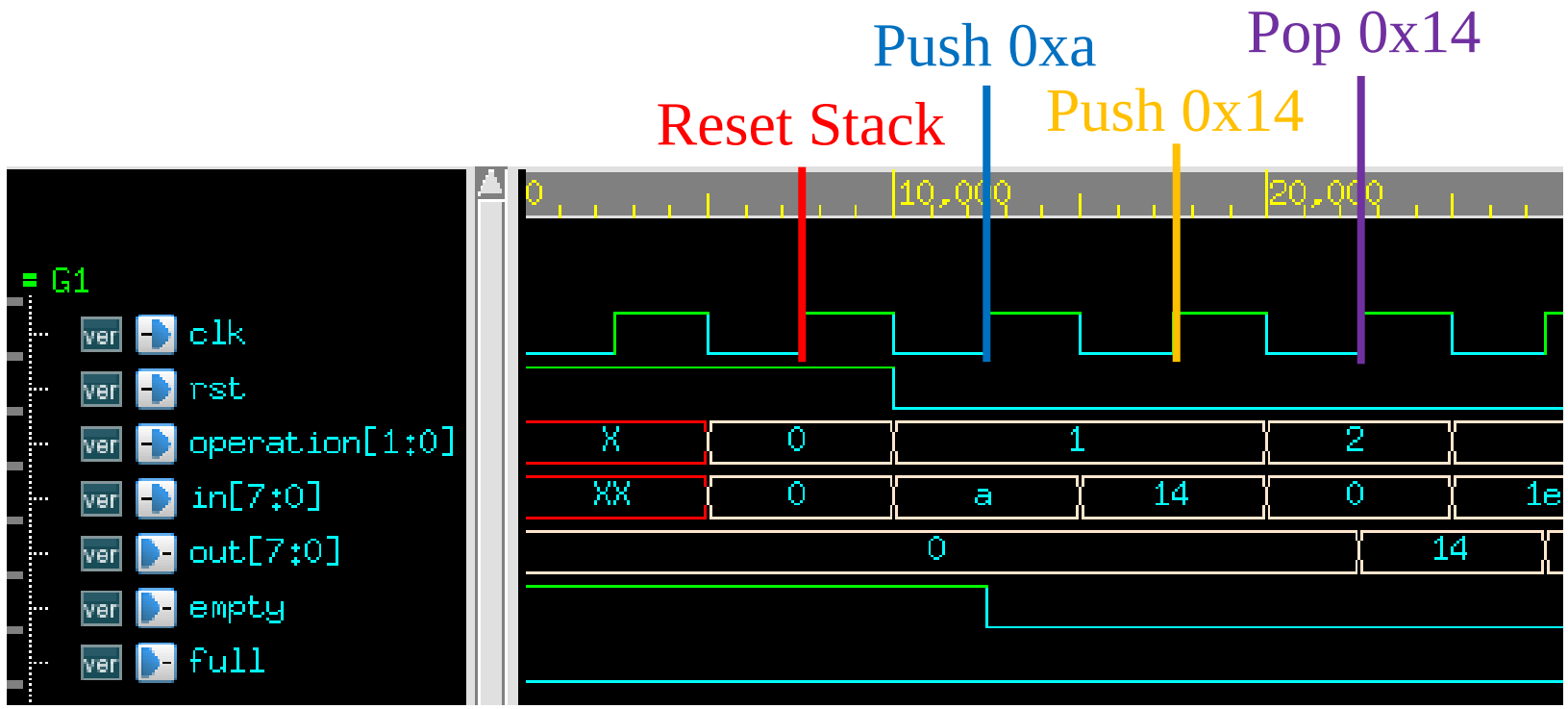
# Implement Stack with 8 Element

- I/O Interface of Module Stack in Stack.v

| Name | I/O | Width | Description |
|---|---|---|---|
| clk | I | 1 | System clock signal. This system is synchronized with the positive edge of the clock. |
| rst | I | 1 | Active-high asynchronous reset signal. |
| operation | I | 2 | Operations.<br>Idle = 2'b00, Push = 2'b01, Pop = 2'b10, Clear = 2'b11 |
| in | I | 8 | Push element. |
| out | O | 8 | Pop element. |
| empty | O | 1 | High when stack is empty. |
| full | O | 1 | High when stack is full. |

# Implement Stack with 8 Element

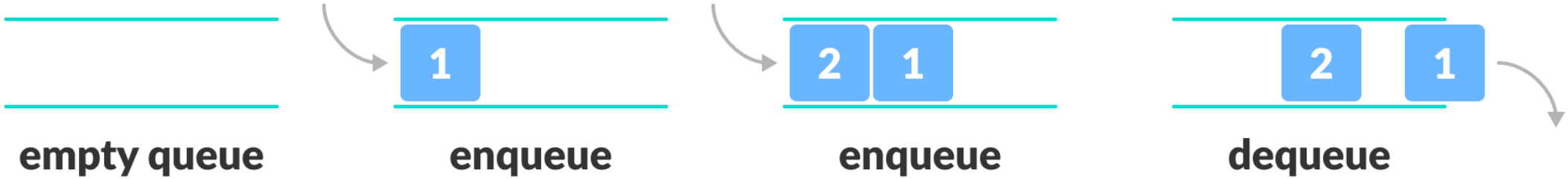
- Waveform of Stack with 8 Element



# Implementation(3)
# Queue

# Queue

- A queue is an abstract data type that serves as a collection of elements..
- Two main principal operations:
  1. Enqueue => Adds an element to the rear of the queue.
  2. Dequeue => Removes an element from the front of the queue.
- The order in which elements come off a Queue gives rise to its alternative name, FIFO (First In, First Out).



# Implement Queue with 8 Element

- Each element in Queue is 8-bit width.
- Three operation support:
  1. Enqueue => Adds an 8-bit element to the queue.
  2. Dequeue => Removes an 8-bit element from the queue.
  3. Clear => Clear all element in the queue, and don't need to output.
- Recommendation:
  1. Use the combination of Vector and Array to create Queue's memory element, for example `reg [7:0] memory [7:0];`
  2. Use a variable to indicate which element is the newest one.
  3. Always dequeue first (oldest) element in memory, and shift the memory by one element, and subtract index variable by 1.
  4. Evaluate in Combinational Block.
  5. Update register & memory element in Sequential Block.

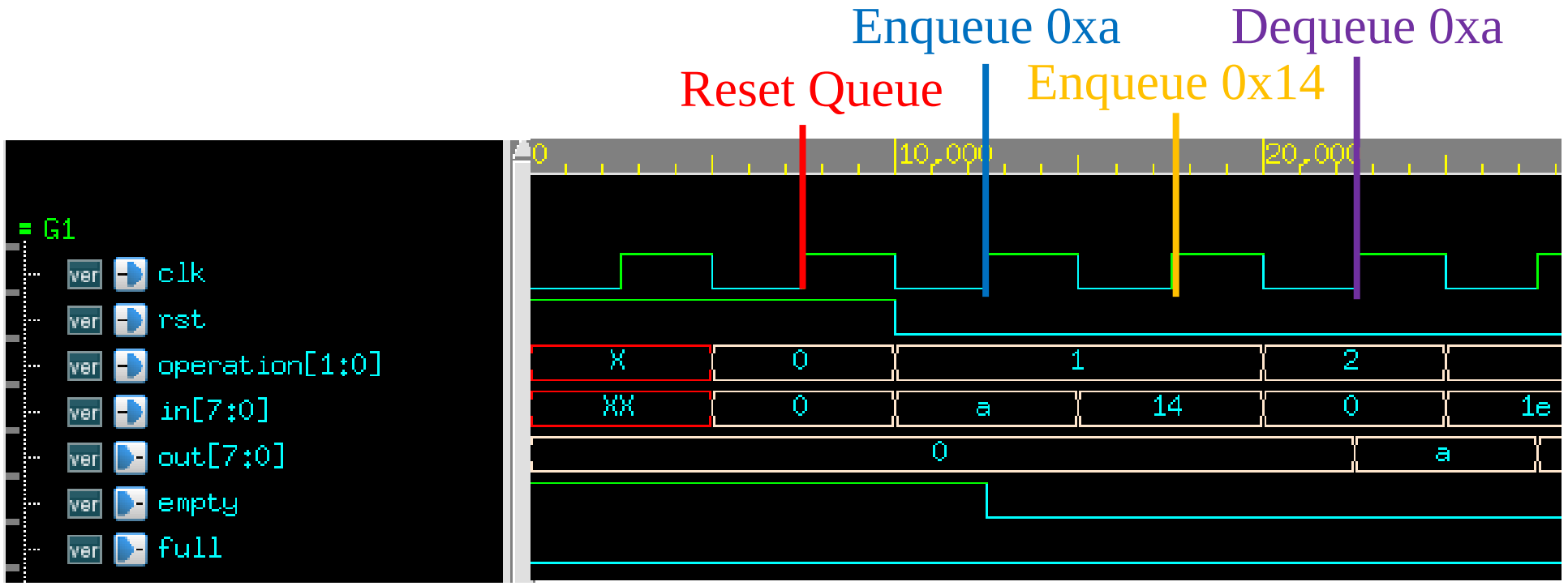
# Implement Queue with 8 Element

- I/O Interface of Module Queue in Queue.v

| Name | I/O | Width | Description |
|---|---|---|---|
| clk | I | 1 | System clock signal. This system is synchronized with the positive edge of the clock. |
| rst | I | 1 | Active-high asynchronous reset signal. |
| operation | I | 2 | Operations.<br>Idle = 2'b00, Enqueue = 2'b01, Dequeue = 2'b10,<br>Clear = 2'b11 |
| in | I | 8 | Enqueue element. |
| out | O | 8 | Dequeue element. |
| empty | O | 1 | High when queue is empty. |
| full | O | 1 | High when queue is full. |

# Implement Queue with 8 Element

- Waveform of Stack with 8 Element



# TA Checking
# &
# Laboratory Report

# TA Checking & Laboratory Report

- TA Checking
  1. Implementation(1)’s Code & Pass Result
  2. Implementation(2)’s Code & Pass Result
  3. Implementation(3)’s Code & Pass Result
- Laboratory Report
  1. Test Yourself’s Answer
  2. Implementation(1)’s 6-bit Johnson Counter Circuit
  3. Implementation(2)’s idea and waveform
  4. Implementation(3)’s idea and waveform
  5. How do you think about this time’s laboratory class?

# Reference

# Reference

- Latch & Flip-Flop & Register: https://reurl.cc/em7x2m
- Combinational vs Sequential: https://reurl.cc/OkL1l7
- Blocking vs Non-Blocking: https://reurl.cc/GbLVMG
- Verilog HDL Design: https://pse.is/3rypr5

# Thank for Listening