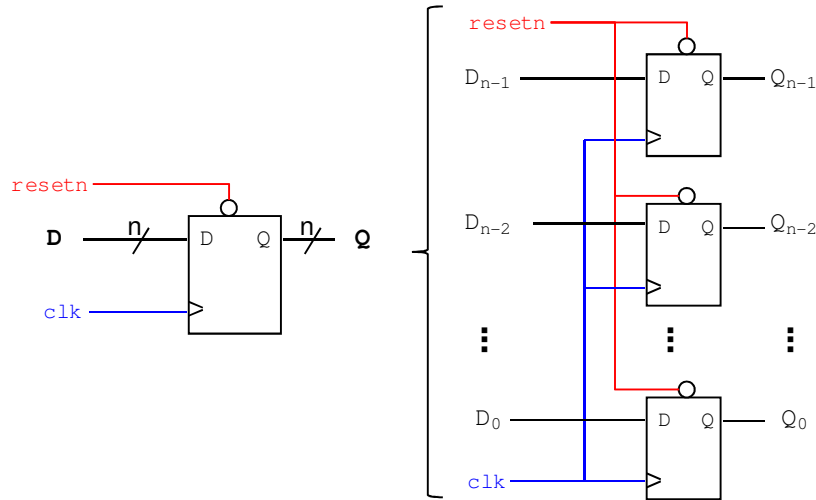


Notes - Chapter 6

REGISTERS:

n-bit Register: This is a collection of 'n' D-type flip flops, where each flip flop independently stores one bit. The flip flops are connected in parallel. They also share the same 'resetn' and 'clock' signals.

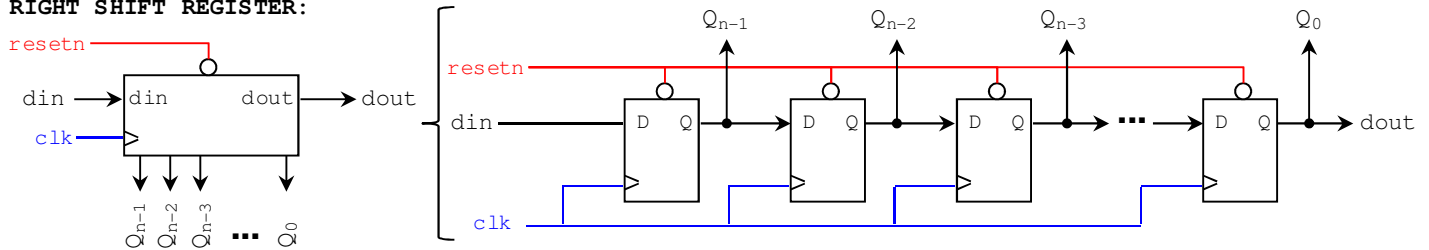


n-bit Shift register: This is a collection of 'n' D-type flip flops, connected serially. The flip flops share the same 'resetn' and 'clock' signals.

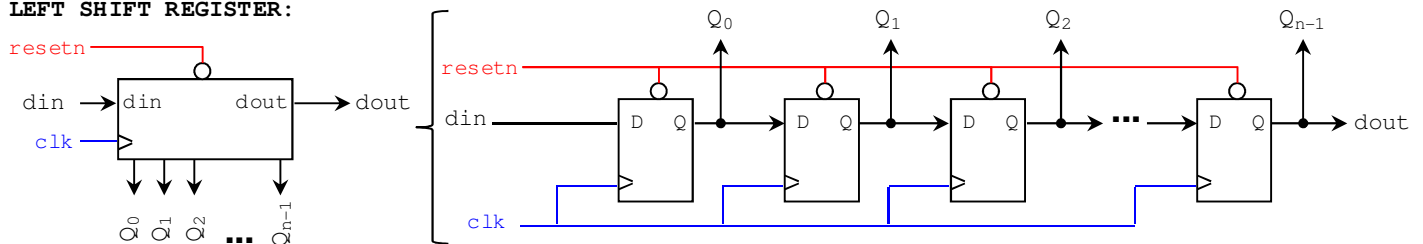
The serial input is called 'din', and the serial output is called 'dout'. The flip flop outputs (also called the parallel output) is called $Q = Q_{n-1}Q_{n-2} \dots Q_0$. Depending on how we label the bits, we can have:

- *right shift register:* The input bit moves from the MSB to the LSB, and
- *left shift register:* The input bit moves from the LSB to the MSB.

RIGHT SHIFT REGISTER:

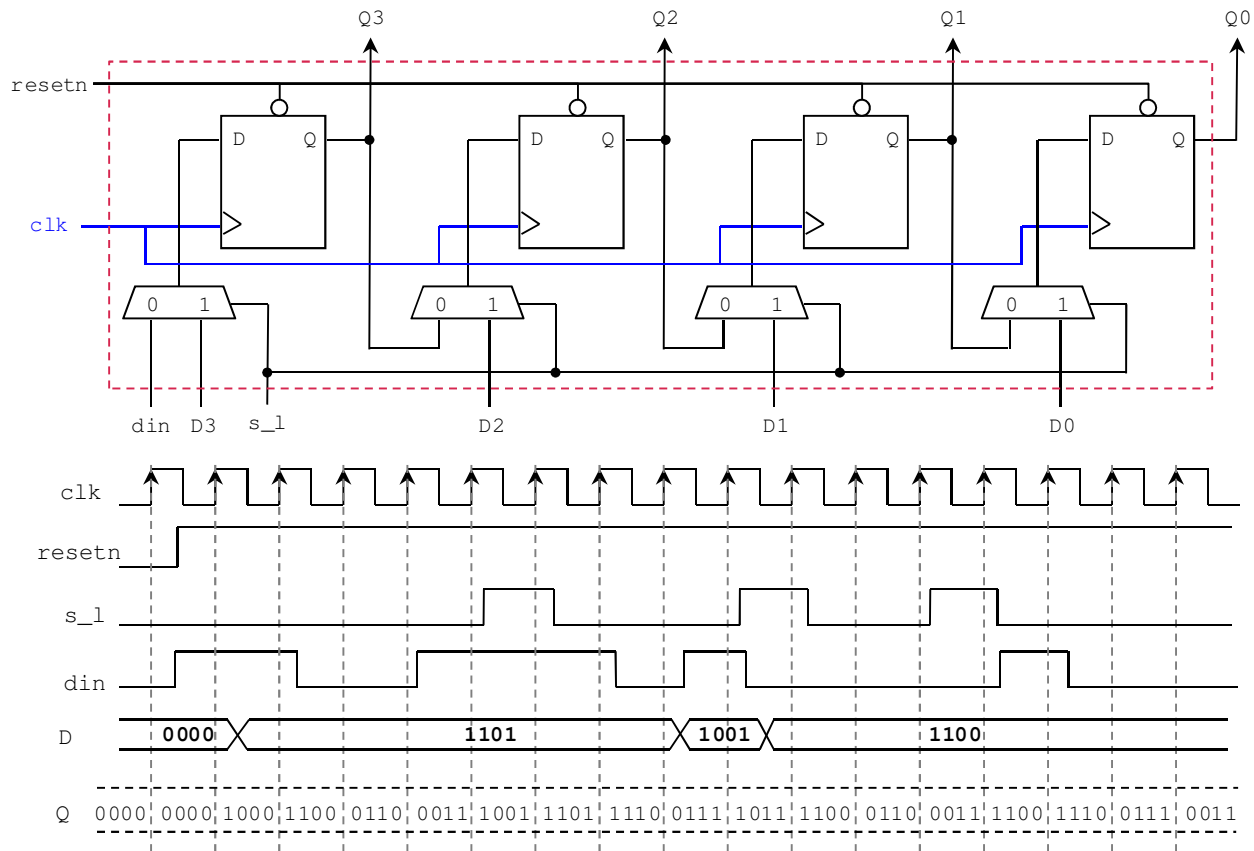


LEFT SHIFT REGISTER:



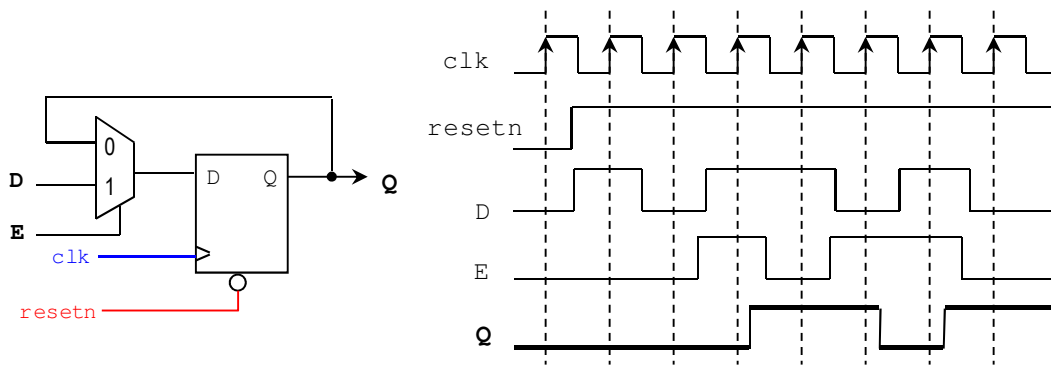
Parallel access shift register:

This is a shift register in which we can write data on the flip flops in parallel. The figure below shows a 4-bit parallel access shift register.

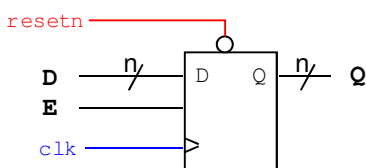


Adding enable to flip flops:

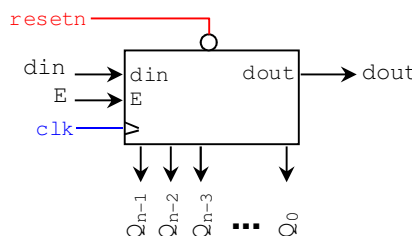
- In many instances, it is very useful to have a signal that controls whether the value of the flip flop is kept. The following circuit represent a flip flop with synchronous enable. When E = '0', the flip flop keeps its value. When E = '1', the flip flop grabs the value at the input D.
- We can thus create n-bit register and n-bit shift registers with enable. Here, all the flip flops share the same enable input.



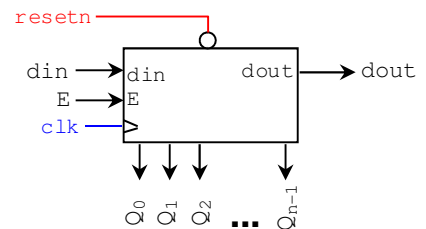
REGISTER:



RIGHT SHIFT REGISTER:



LEFT SHIFT REGISTER:

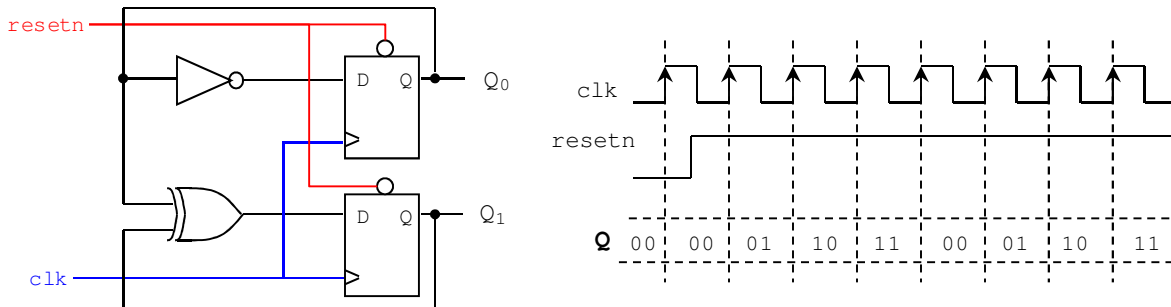


SYNCHRONOUS COUNTERS

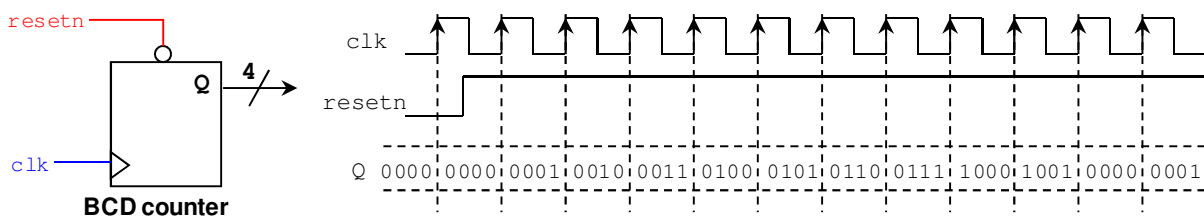
- Counters are useful for: counting the number of occurrences of a certain event, generate time intervals for task control, track elapsed time between two events, etc. Counters are made of flip flops and combinatorial logic. They are usually designed using Finite State Machines (FSM)s.
- Synchronous counters change their output on the clock edge (rising or falling). Each flip flop shares the same `clock` input signal. If the initial count is zero, each flip flop shares the `resetrn` input signal.

COUNTER CLASSIFICATION:

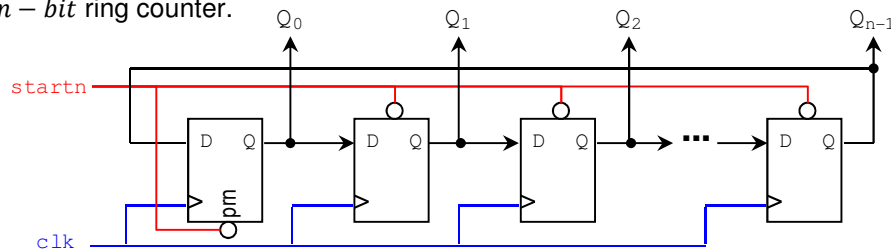
a) **Binary counter:** An $n - bit$ counter counts from 0 to 2^{n-1} . The figure depicts a 2-bit counter.



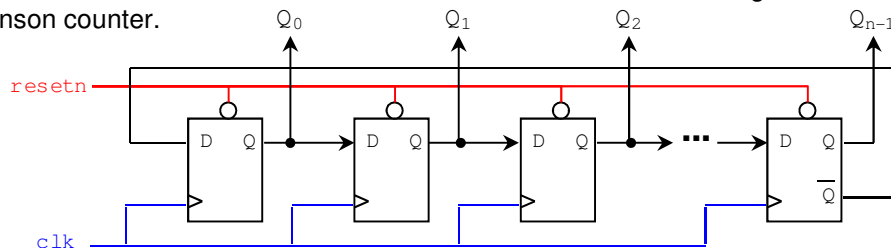
b) **Modulus counter:** A counter *modulo* N counts from 0 to $N-1$. Special case: BCD (or decade) counter: Counts from 0 to 9.



- c) **Up/down counter:** Counts both up and down, under command of a control input.
- d) **Parallel load counter:** The count can be given an arbitrary value.
- e) **Counter with enable:** If `enable` = 0, the count stops. If `enable` = 1, the counter counts. This is usually done by connecting the `enable` inputs of the flip flops to a single `enable`.
- f) **Ring counter:** Also called one-hot counter (only one bit is 1 at a time). It can be constructed using a shift register. The output of the last stage is fed back to the input to the first stage, which creates a ring-like structure. The asynchronous signal '`startn`' sets the initial count to 100...000 (first bit set to 1). Example (4-bits): 1000, 0100, 0010, 0001, 1000, ... The figure below depicts an $n - bit$ ring counter.

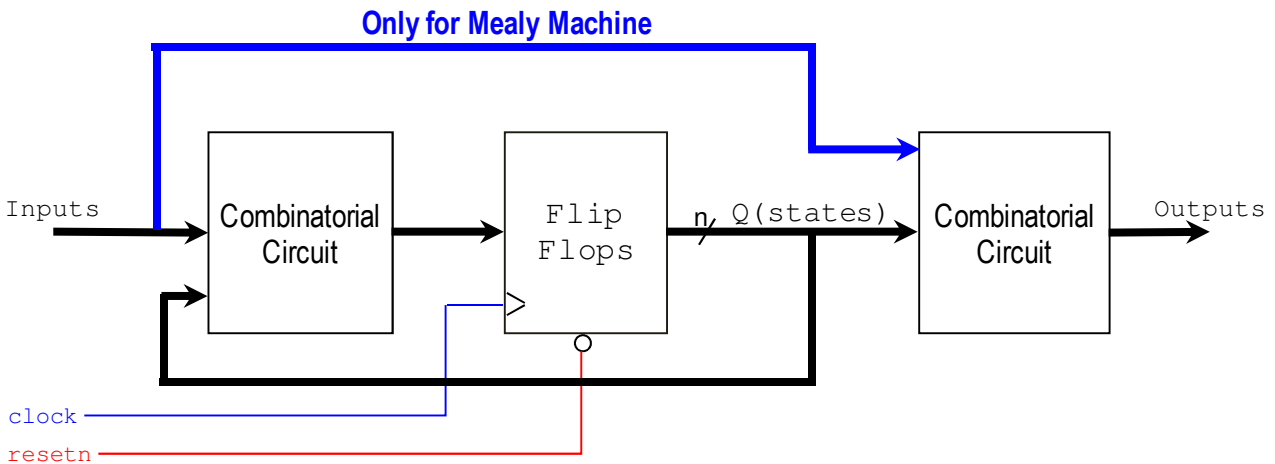


- g) **Johnson counter:** Also called twisted ring counter. It can be constructed using a shift register, where the \bar{Q} output of the last flip flop is fed back to the first stage. The result is a counter where only a single bit has a different value for two consecutive counts. All the flip flops share the asynchronous signal '`resetrn`', which sets the initial count to 000...000. Example (4 bits): 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000, ... The figure below depicts an $n - bit$ Johnson counter.



FINITE STATE MACHINES:

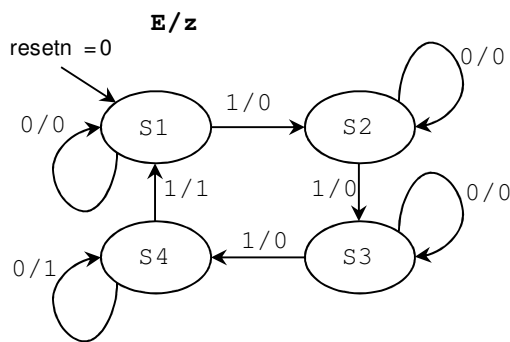
- Sequential circuits are also called Finite State Machines (FSMs), because the functional behavior of these circuits can be represented using a finite number of states (flip flop outputs).
- The signal 'resetn' sets the flip flops to a initial state.
- Classification:
 - Moore machine: Outputs depend solely on the current state of the flip flops.
 - Mealy machine: Outputs depend on the current state of the flip flops as well as on the input to the circuit.



- Any general sequential circuit can be represented by the figure above (Finite State Machine model).
- A sequential circuit with certain behavior and/or specification can be formally designed using the Finite State Machine method: drawing a State Diagram and coming up the Excitation Table.
- Designing sequential circuits using the Finite State Machine method is a powerful in Digital Logic Design.

Example: 2-bit gray-code counter with enable and 'z' output: 00, 01, 11, 10, 00, ... The output 'z' is 1 when the present count is '10'.

- First step:* Draw the State Diagram and State Table. If we were to implement the state machine in VHDL, this is the only step we need.



E	PRESENT STATE	NEXT STATE	NEXT COUNT	z
0	S1	S1	00	0
0	S2	S2	01	0
0	S3	S3	11	0
0	S4	S4	10	1
1	S1	S2	01	0
1	S2	S3	11	0
1	S3	S4	10	0
1	S4	S1	00	1

- Second step:* State Assignment. We assign unique flip flop states to the our state labels (S1, S2, S3, S4). Notice that this is arbitrary. However, we can save resources if we assign each state to the count that we desire. Then, the output 'count' is just the flip flops' outputs.

- ✓ S1: Q = 00
- ✓ S2: Q = 01
- ✓ S3: Q = 11
- ✓ S4: Q = 10

- Third step: Excitation table. Here, we replace the state labels by the flip flop states:

E	PRESENT STATE		NEXTSTATE		
	$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$	z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	1	1	1	0
0	1	0	1	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	1	1	0	0
1	1	0	0	0	1

- Fourth step: Excitation equations and minimization. $Q_1(t+1)$ and $Q_0(t+1)$ are the next state of the flip flops, i.e. these signals are to be connected to the inputs of the flip flops.

$Q_1(t+1)$

$E \backslash Q_1 \backslash Q_0$	00	01	11	10
0	0	1	0	0
1	0	1	1	1

$Q_0(t+1)$

$E \backslash Q_1 \backslash Q_0$	00	01	11	10
0	0	0	0	1
1	1	1	0	1

z

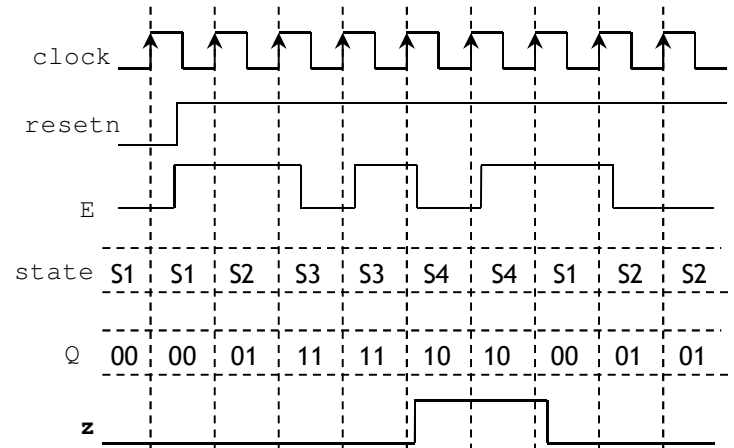
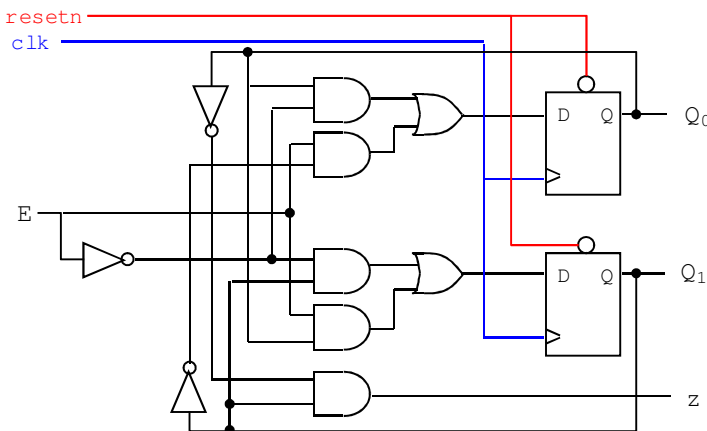
$E \backslash Q_1 \backslash Q_0$	00	01	11	10
0	0	1	1	0
1	0	0	0	0

$$Q_1(t+1) = \bar{E}Q_1 + EQ_0$$

$$Q_0(t+1) = EQ_1 + \bar{E}Q_0$$

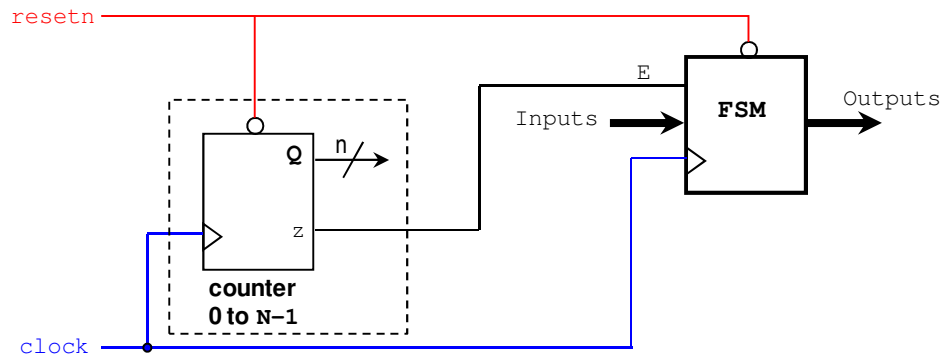
$$z = Q_1\bar{Q}_0$$

- Fifth step: Circuit implementation:



Modifying the rate of change of a Finite State Machine:

- We usually would like to reduce the rate at which FSM transitions occur. A straightforward option is to reduce the frequency of the clock input. But this is a very complicated problem when a high precision clock is required.
- Alternatively, we can reduce the rate at which FSM transitions occur by including an enable signal in our FSM. Then we assert the enable signal only when we need it. The effect is the same as reducing the frequency of the input clock.
- The figure below depicts a counter modulo-N (from 0 to N-1) that generates a pulse (output signal 'z') of one clock period every time we hit the count 'N-1'. The number of bits the counter is given by $n = \lceil \log_2 N \rceil$. The effect is the same as reducing the frequency of the FSM to f/N , where f is the frequency of the clock.
- A modulo-N counter is better designed using VHDL behavioral description, where the count is increased by 1 every clock cycle and 'z' is generated by comparing the count to 'N-1'. A modulo-N counter could be designed by the State Machine method, but this can be very cumbersome if N is a large number. For example, if $N = 1000$, we need 1000 states.



- ✓ As an example, we provide the timing diagram of the counter from 0 to N-1, when N=10. notice that 'z' is only activated when the count reaches "1001". This 'z' signal controls the enable of a state machine, so that the FSM transitions only occur every 10 clock cycles, thereby having the same effect as reducing the frequency by 10.

