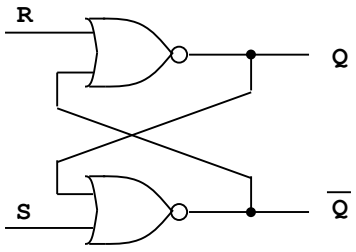
Unit 6 – Synchronous Sequential Circuits

Fundamental Circuits

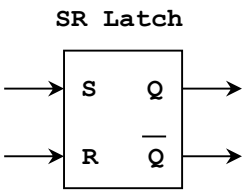
Asynchronous Circuits: Latches

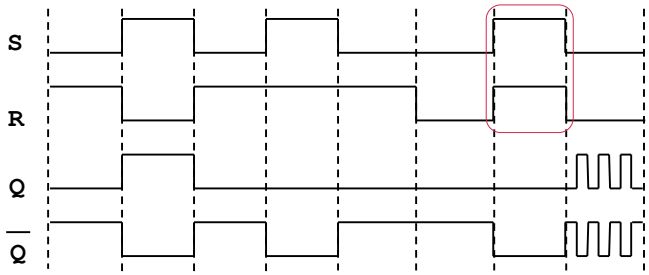
SR LATCH:



S	R	Q_{t+1}	$\overline{Q_{t+1}}$
0	0	Q_t	$\overline{Q_t}$
0	1	0	1
1	0	1	0
1	1	0	0

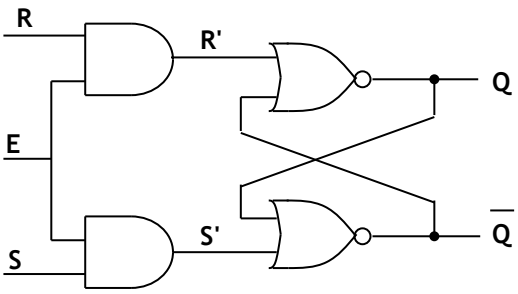
restricted





$Q_{t+1} \leftarrow \bar{R}S + \bar{R}Q_t$

SR LATCH WITH ENABLE:

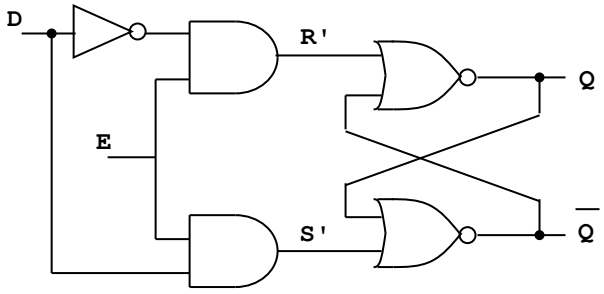


E	S	R	Q_{t+1}	$\overline{Q_{t+1}}$
0	x	x	Q_t	$\overline{Q_t}$
1	0	0	Q_t	$\overline{Q_t}$
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

$Q_{t+1} \leftarrow E\bar{R}S + \bar{E}Q_t + \bar{R}Q_t$

D LATCH WITH ENABLE:

- This is essentially an SR Latch with enable, where $R = not(D),\ S = D$. The D Latch always has an enable input.



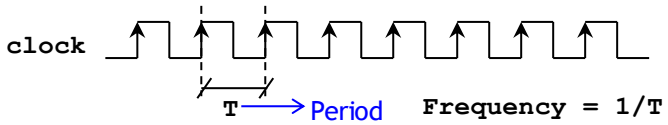
E	D	Q_{t+1}
0	x	Q_t
1	0	0
1	1	1

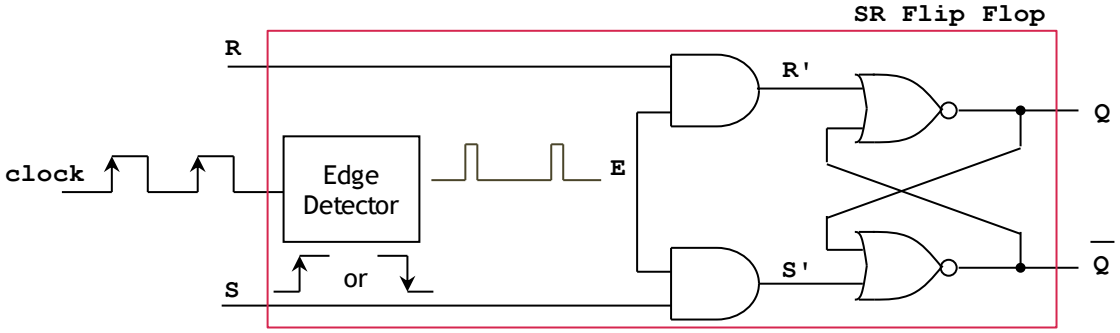
$Q_{t+1} \leftarrow \bar{E}Q_t + ED$

SYNCHRONOUS CIRCUITS: FLIP FLOPS

- Flip flops are made out of:
  - ✓ A Latch with an enable input.
  - ✓ An Edge detector circuit.

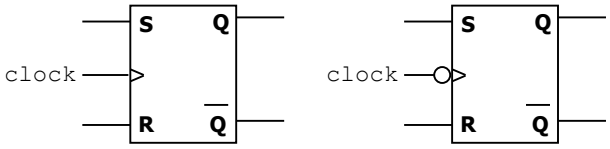
- The figure depicts an SR Latch, where the enable is connected to the output of an *Edge Detector* Circuit. The input to the Edge Detector is a signal called '***clock***'. A clock signal is a square wave with a fixed frequency.





- The edge detector circuit generates short-duration pulses during rising (or falling) edges. These pulses act as short-time enables of the Latch.
- The behavior of the flip flops can be described as that of a Latch that is only enabled during rising (or falling edges).

- Depending on what type of edge we are detecting, flip flops can be classified as:
  - ✓ Positive-edge triggered flip flop: The edge detector circuit generates pulses during rising edges.
  - ✓ Negative-edge triggered flip flop: The edge detector circuit generates pulses during falling edges.
  - ✓ Dual-edge triggered flip flop: The edge detector circuit generates pulses during both rising and falling edges. Current FPGA technology does not support dual-edge triggered flip flops.

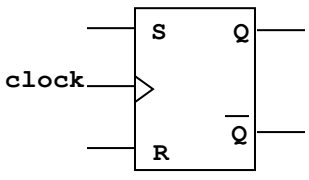


FLIP FLOP TYPES

SR Flip Flop

Excitation Table:

clock	S	R	Q_{t+1}	$\overline{Q_{t+1}}$
↑	0	0	Q_t	$\overline{Q_t}$
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	0	0



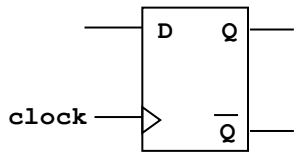
Excitation Equation:

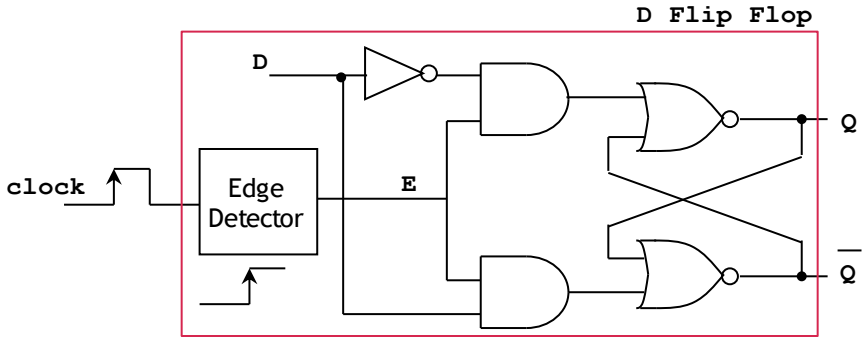
$Q_{t+1} \leftarrow S\bar{R} + Q_t\bar{S}\bar{R} = \bar{R}(S + Q_t\bar{S}) = \bar{R}(S + \bar{S})(S + Q_t) = \bar{R}S + \bar{R}Q_t$ (on the edge)

Note that when there are no rising edges, $Q_{t+1} = Q_t$

D Flip Flop

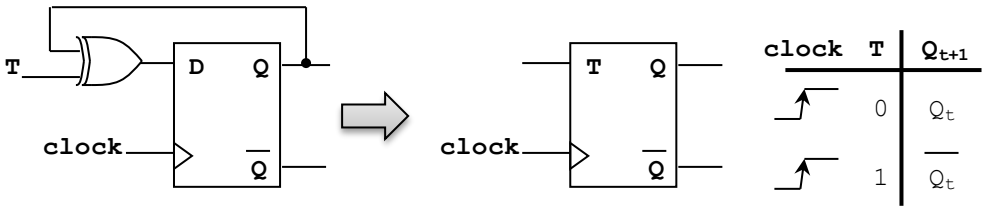
clock	D	Q_{t+1}
↑	0	0
↑	1	1





Excitation Equation: $Q_{t+1} \leftarrow D$

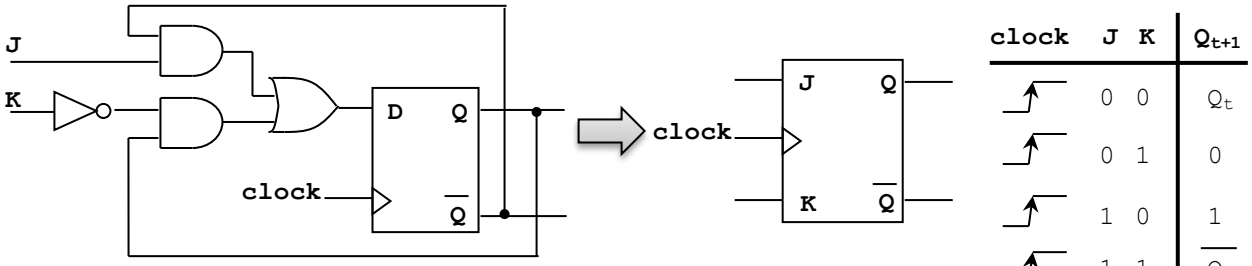
T Flip Flop



clock	T	Q_{t+1}
↑	0	Q_t
↑	1	$\overline{Q_t}$

Excitation Equation: $Q_{t+1} \leftarrow D = T \oplus Q_t$

JK Flip Flop



clock	J	K	Q_{t+1}
↑	0	0	Q_t
↑	0	1	0
↑	1	0	1
↑	1	1	$\overline{Q_t}$

Excitation Equation: $Q_{t+1} \leftarrow J\overline{Q_t} + \overline{K}Q_t$

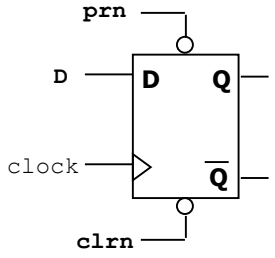
SYNCHRONOUS AND ASYNCHRONOUS INPUTS

Synchronous Inputs

- Typically, flip flops only change their outputs on the rising (or falling edge). Usually, a change on the inputs forces a change on the outputs. These inputs are known as *synchronous inputs*, as the inputs' state is only checked on the rising (or falling) edges. Example: Input D of a D flip flop, Inputs J, K of a JK flip flop.

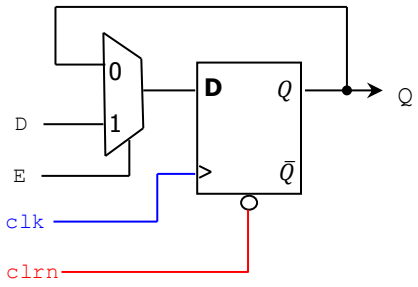
Asynchronous Inputs

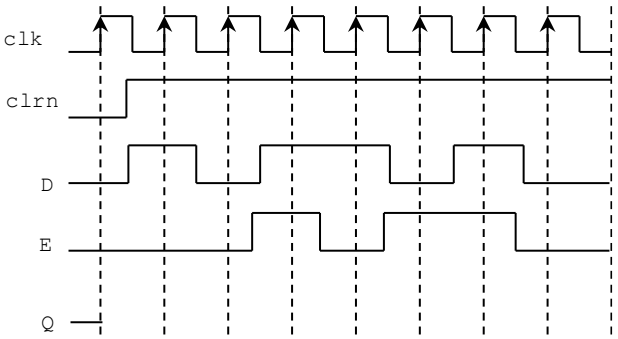
- However, in many instances, it is useful to have inputs that force the outputs to a value immediately, disregarding the rising (or falling edges). These inputs are known as *asynchronous inputs*. Common asynchronous inputs are prn and $clrn$ (they can be active-low or active high)
- The figure depicts a D Flip Flop with two asynchronous inputs:
  - ✓ prn: Preset (active low). When $prn = 0 \rightarrow Q = 1$.
  - ✓ $clrn$ (sometimes called $resetn$): Clear (active low). When $clrn = 0 \rightarrow Q = 0$.
  - ✓ If prn and $clrn$ are both 0, usually $clrn$ is given priority.
- A Flip flop can have more than one asynchronous inputs, or none.



PRACTICE EXERCISES I

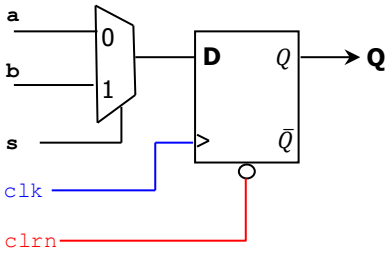
1. Complete the timing diagram of the circuit shown below:





2. Complete the VHDL description of the circuit shown below. Also, get the excitation equation for Q.

```
library ieee;
use ieee.std_logic_1164.all;

entity circ is
  port ( a, b, s, clk, clrn: in std_logic;
         q: out std_logic);
end circ;

architecture a of circ is

begin
  -- ???

end a;
```



$Q_{t+1} \leftarrow$

3. Complete the VHDL description of the circuit whose excitation table is shown below. What is the excitation equation for Q?

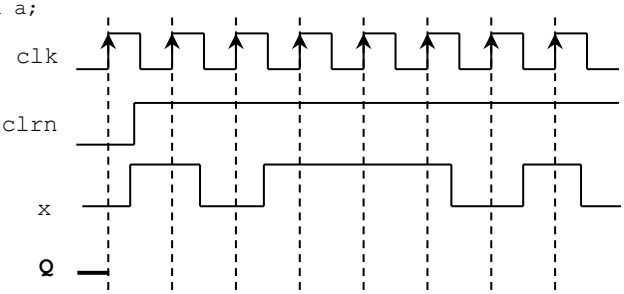
```
library ieee;
use ieee.std_logic_1164.all;

entity circ is
  port ( A, B, C, clrn, clk: in std_logic;
         q: out std_logic);
end circ;

architecture a of circ is
begin
  -- ???

end a;
```

clrn	clk	A	B	Q_{t+1}
1	↑	0	0	1
1	↑	0	1	C
1	↑	1	0	$\overline{Q_t}$
1	↑	1	1	Q_t
0	X	X	X	0

$Q_{t+1} \leftarrow$

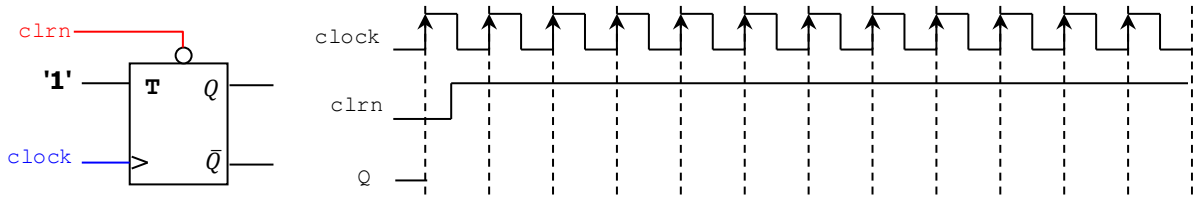
4. Complete the timing diagram of the circuit whose VHDL description is shown below. What is the excitation equation for Q?

```
library ieee;
use ieee.std_logic_1164.all;

entity circ is
  port ( clrn, x, clk: in std_logic;
         q: out std_logic);
end circ;

architecture a of circ is
   signal qt: std_logic;
begin
   process (clrn, clk, x)
   begin
     if clrn = '0' then
         qt <= '0';
      elsif (clk'event and clk = '1') then
         if x = '1' then qt <= not (qt); end if;
      end if;
   end process;
   q <= qt
end a;
```

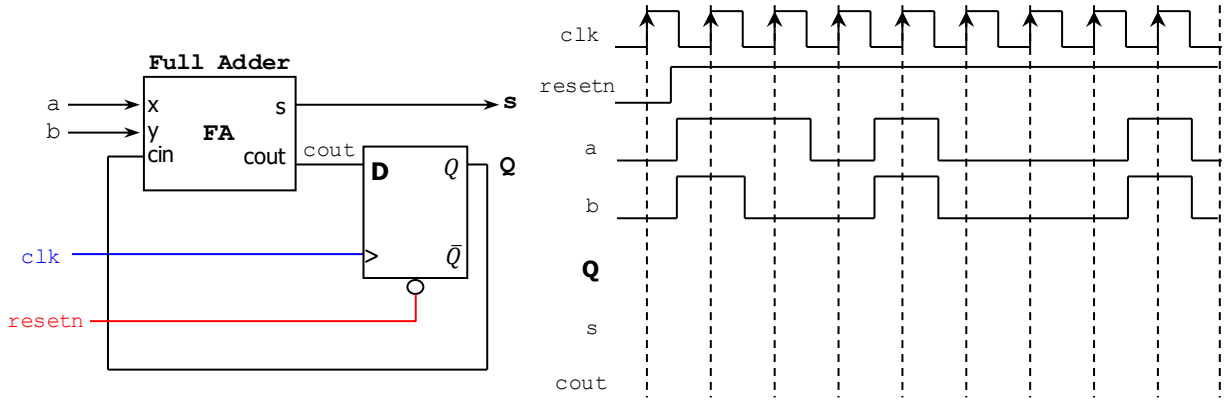


$Q_{t+1} \leftarrow$

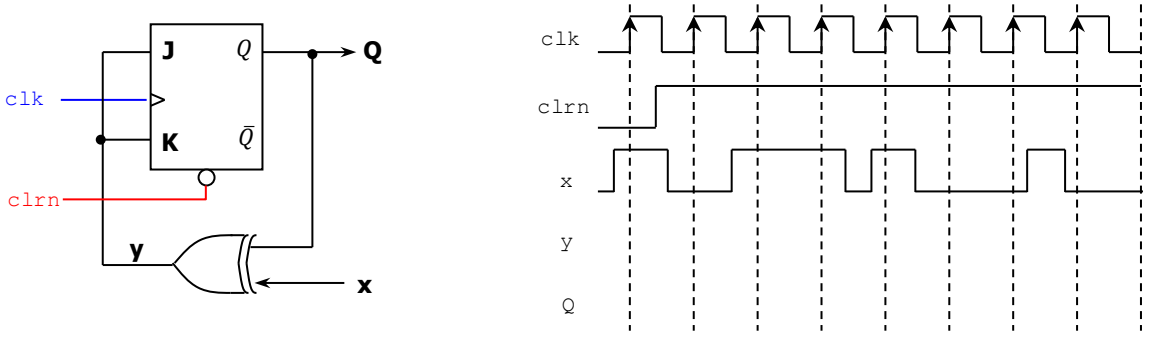
5. Complete the timing diagram of the circuit shown below. If the frequency of the signal clock is 25 MHz, what is the frequency (in MHz) of the signal Q?



6. Complete the timing diagram of the circuit shown below:



7. Complete the timing diagram of the circuit shown below. What is the excitation equation for Q?

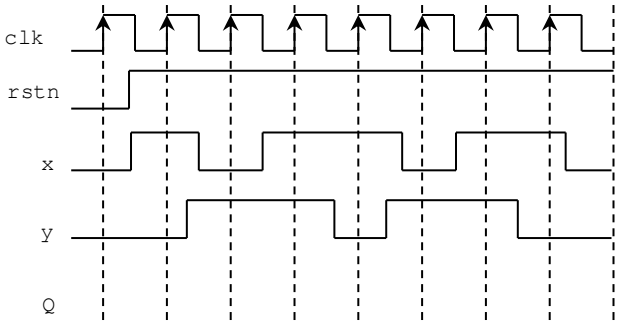


$Q_{t+1} \leftarrow$

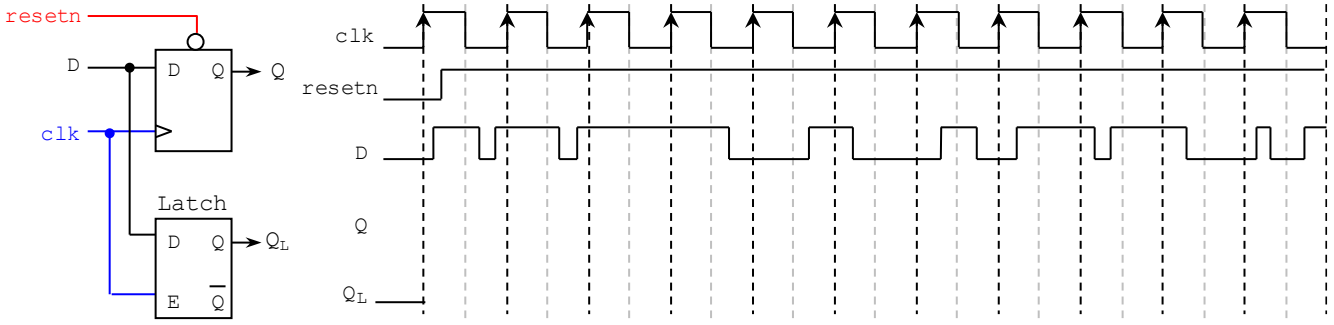
8. With a D flip flop and gates, sketch a circuit whose excitation equation is given by: $Q_{t+1} \leftarrow ab + (a \oplus b)Q_t$

9. Given the following excitation equation of a synchronous circuit with `rstn` and `clock`, complete the timing diagram.

$Q_{t+1} \leftarrow x \oplus y \oplus Q_t + \overline{xy}\,\overline{Q_t}$

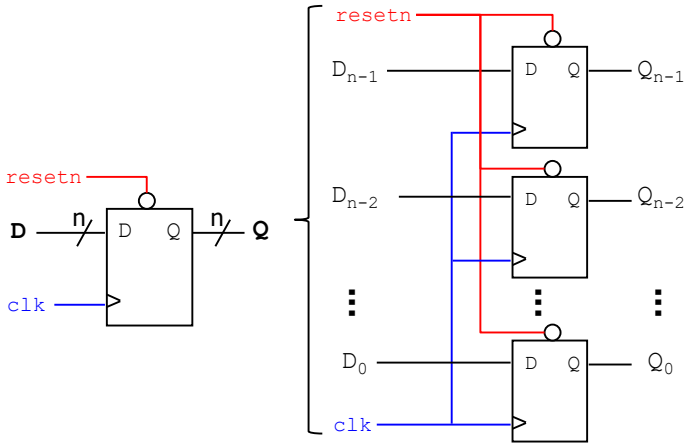


10. Complete the timing diagram of the circuit shown below:



SYNCHRONOUS CIRCUITS: REGISTERS

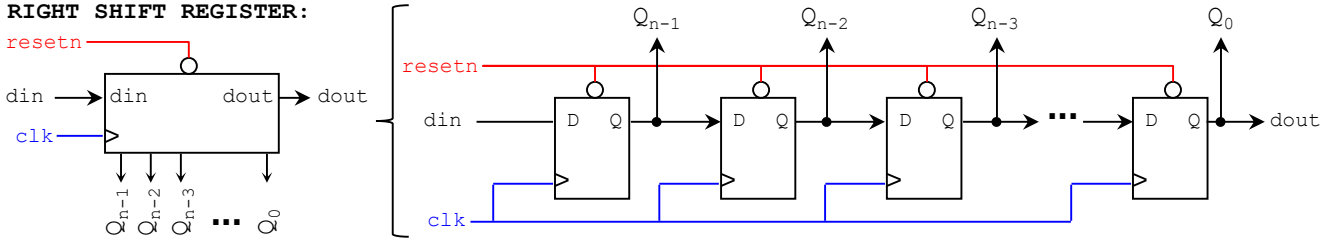
n-BIT REGISTER: This is a collection of 'n' D-type flip flops, where each flip flop independently stores one bit. The flip flops are connected in parallel. They also share the same `resetn` and `clock` signals.



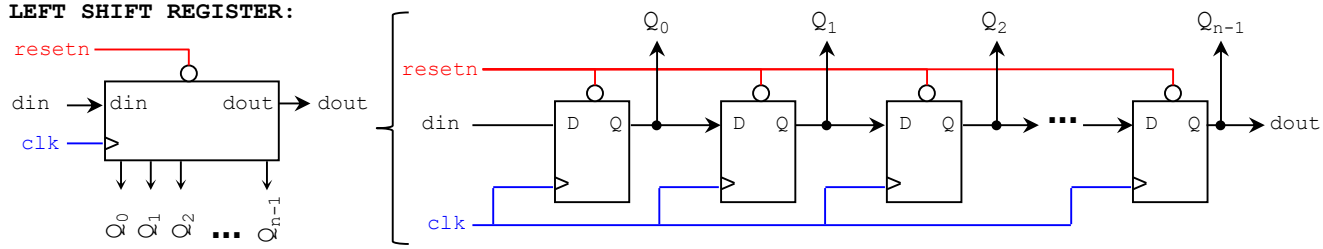
n-bit SHIFT REGISTER: This is a collection of 'n' D-type flip flops, connected serially. The flip flops share the same `resetn` and `clock` signals. The serial input is called 'din', and the serial output is called 'dout'. The flip flop outputs (also called the parallel output) are called $Q = Q_{n-1}Q_{n-2}\cdots Q_0$. 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.

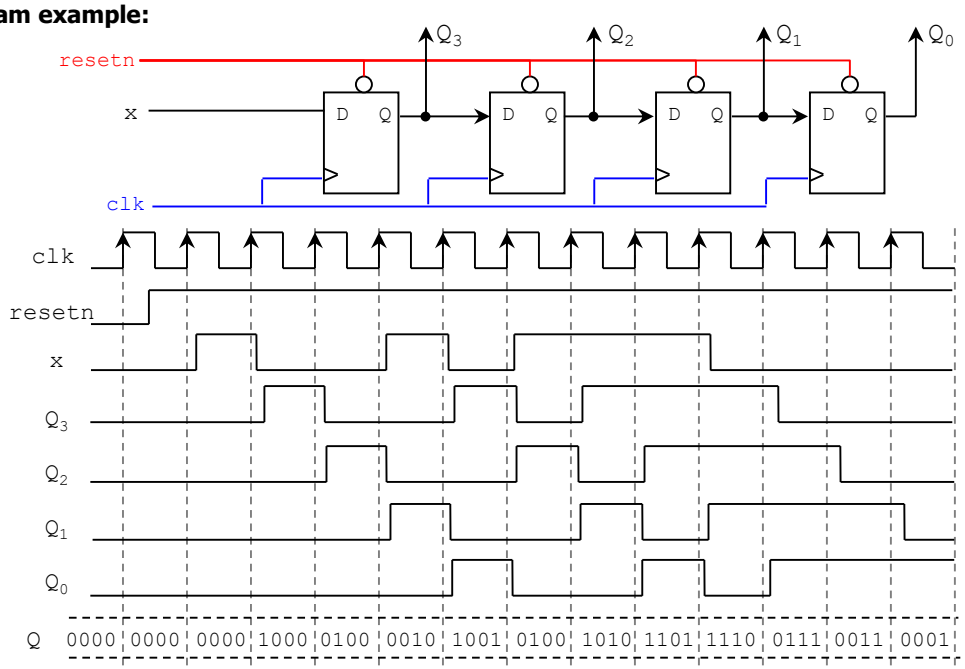
RIGHT SHIFT REGISTER:



LEFT SHIFT REGISTER:

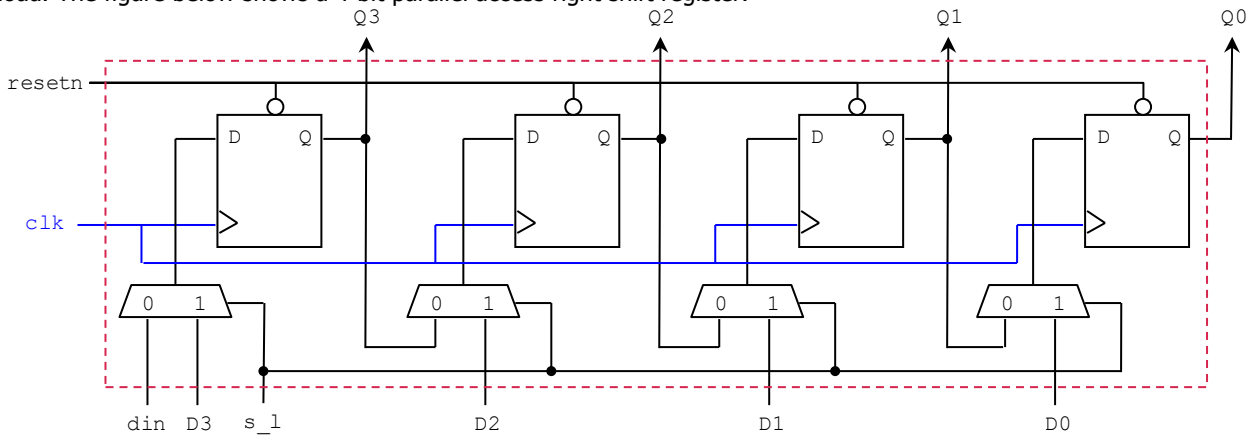


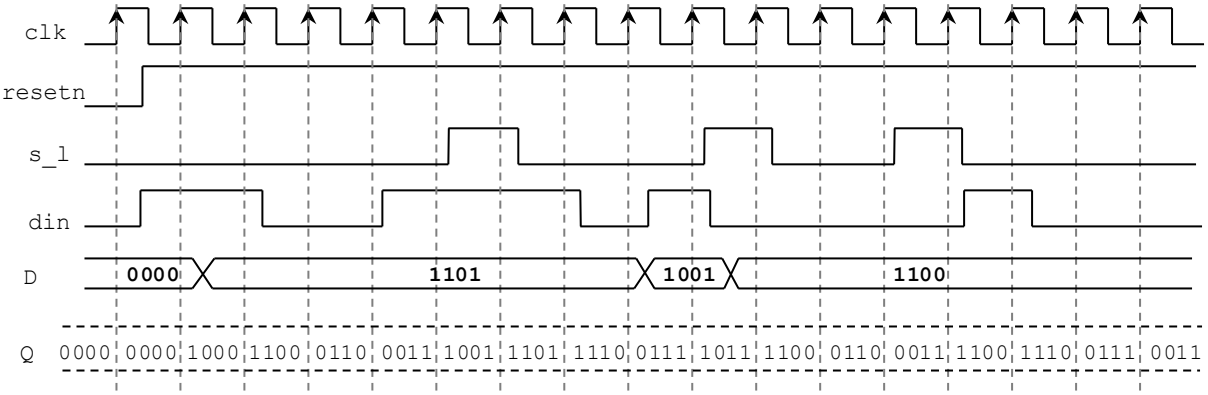
Timing Diagram example:



PARALLEL ACCESS SHIFT REGISTER:

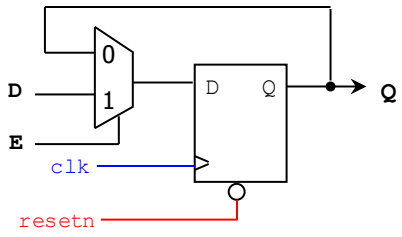
- This is a shift register in which we can write data on the flip flops in parallel. $s\_l = 0 \rightarrow$ shifting operation, $s\_l = 1 \rightarrow$ parallel load. The figure below shows a 4-bit parallel access right shift register.

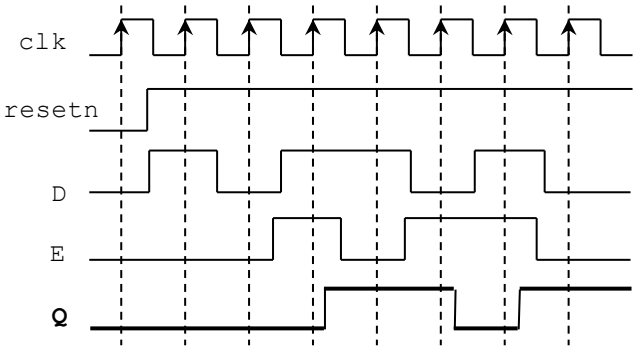




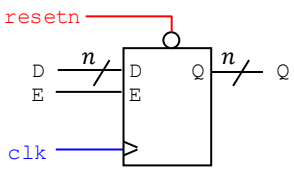
ADDING AN ENABLE INPUT TO FLIP FLOPS:

- 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.
  - ✓ E = '0': the flip flop keeps its value.
  - ✓ E = '1': the flip flop captures the value of the input D.
- We can thus create n-bit registers and n-bit shift registers with enable. Here, all the flip flops share the same enable input.
- Excitation equation: $Q_{t+1} \leftarrow \bar{E}Q_t + ED$

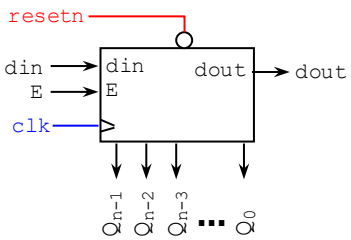




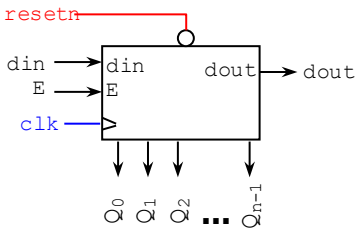
REGISTER:



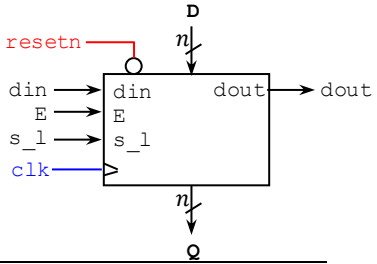
RIGHT SHIFT REGISTER:



LEFT SHIFT REGISTER:

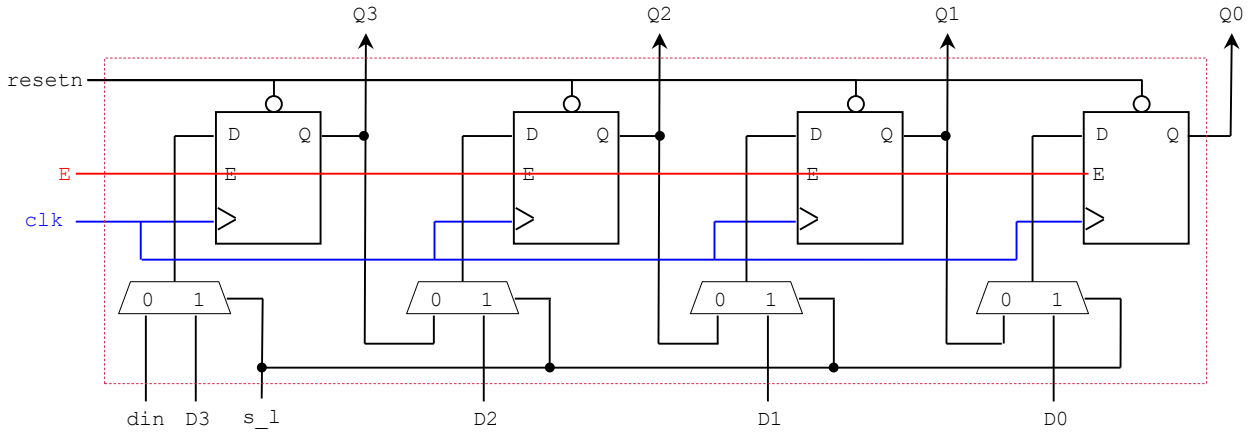


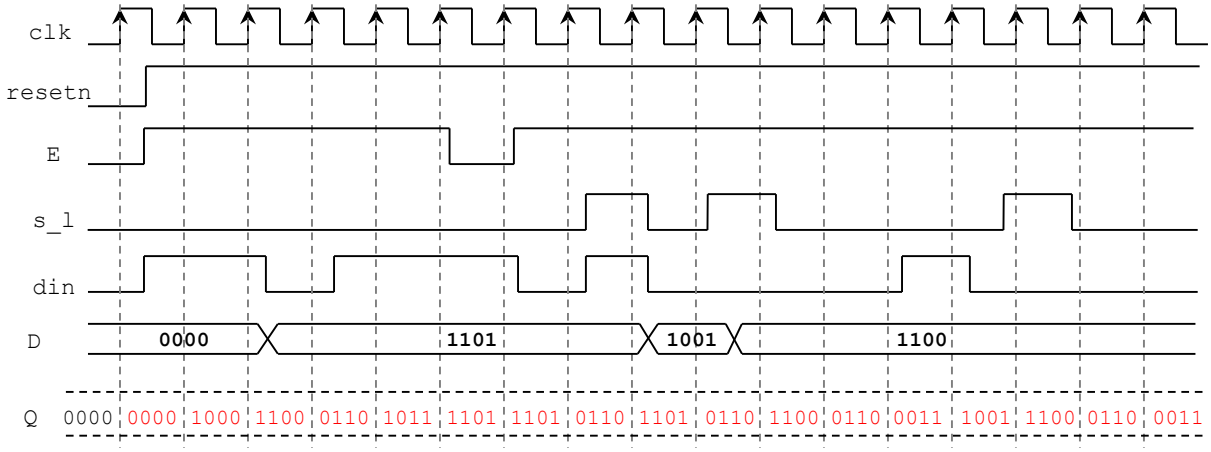
PARALLEL ACCESS SHIFT REGISTER:



Parallel access shift register with enable

- All the flip flops share the same enable input.

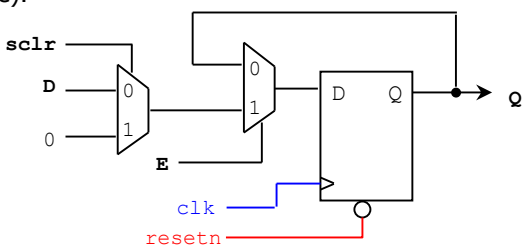




ADDING A SYNCHRONOUS CLEAR INPUT TO FLIP FLOPS

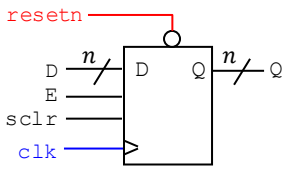
- In many instances, it is very useful to have a signal that can clear the value of the flip flop but only on the rising (or falling edges): synchronous clear ($sclr$). Typically, all synchronous signals are validated by enable. For example, for a D flip flop, the table show how the output state Q changes (on the rising edge):

E	$sclr$	Q (output state)
0	X	$Q \leftarrow Q$
1	0	$Q \leftarrow D$
1	1	$Q \leftarrow 0$

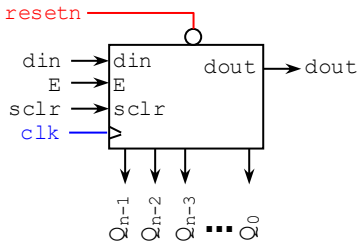


- We can thus create n-bit registers and n-bit shift registers with enable and $sclr$. Here, all the flip flops share the same enable and $sclr$ inputs.

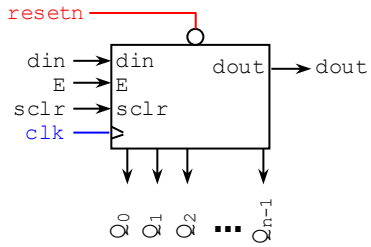
REGISTER:



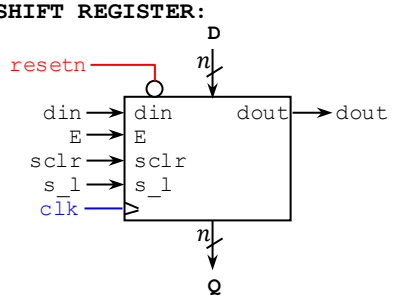
RIGHT SHIFT REGISTER:



LEFT SHIFT REGISTER:



PARALLEL ACCESS SHIFT REGISTER:

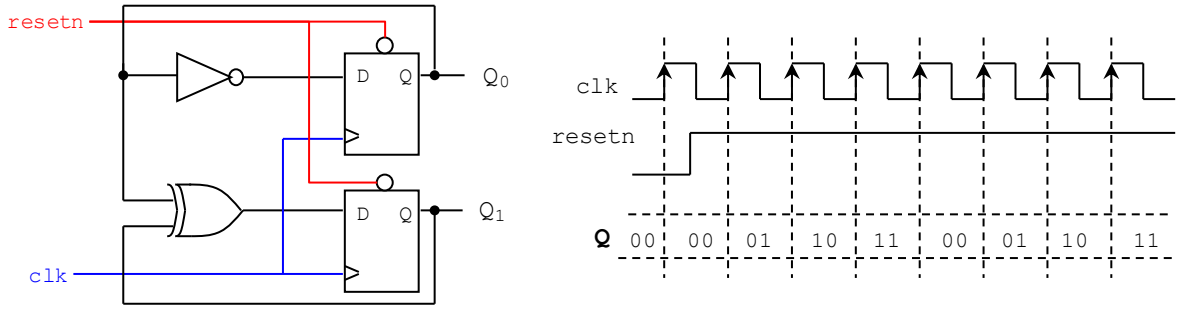


SYNCHRONOUS COUNTERS

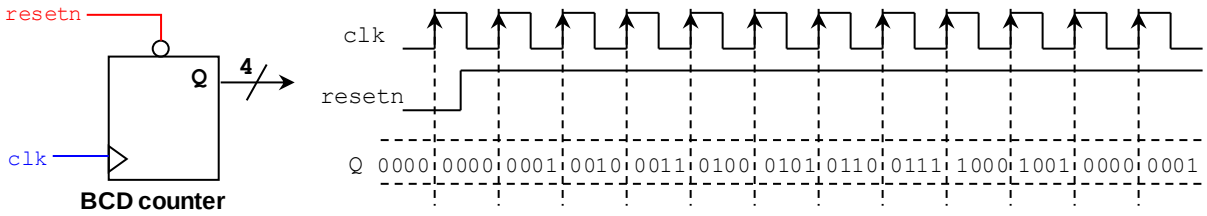
- Counters are useful for: counting the number of occurrences of a certain event, generate time intervals for task control, track elapsed time between two events, etc. Counters are made of flip flops and combinatorial logic. Common design techniques include using the Finite State Machine (FSM) method of a synchronous accumulator.
- 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 `resetn` 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.

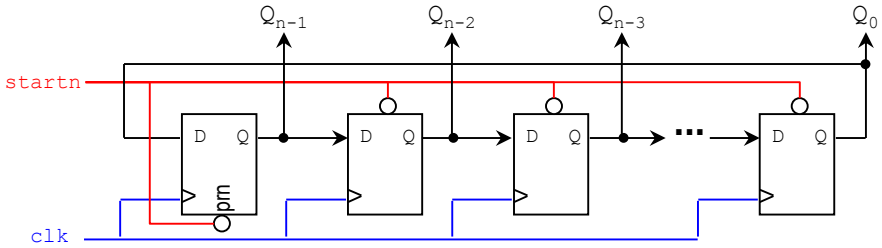


c) **Up/down counter**: Counts both up and down, under command of a control input.

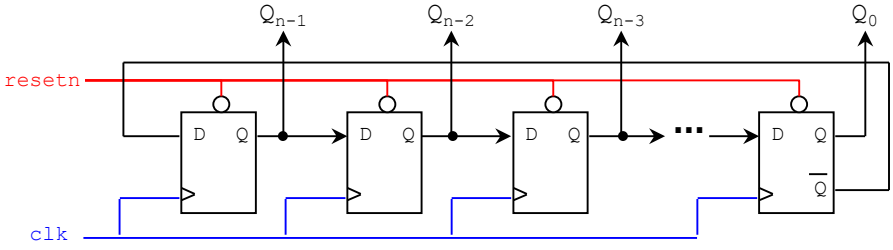
d) **Parallel load counter**: The count can be given an arbitrary value.

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.

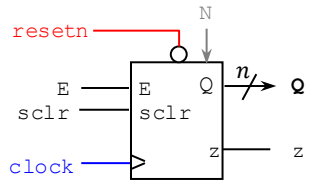


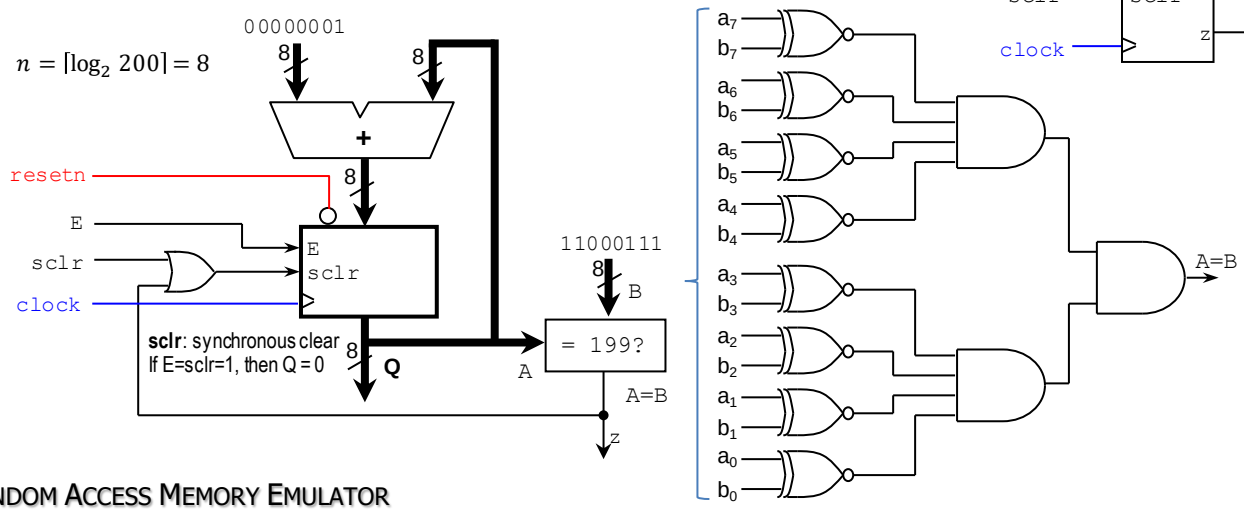
g) **Johnson counter**: Also called twisted ring counter. It can be constructed using a shift register, where the $\bar{Q}$ output of the last flip flop is fed back to the first stage. The result is a counter where only a single bit has a different value for two consecutive counts. All the flip flops share the asynchronous signal 'resetn', 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.



COUNTER DESIGN USING AN ACCUMULATOR:

- A generic modulo-N counter is shown on the right ($n = \lceil \log_2 N \rceil$). An example is shown below of a counter modulo-200 (it uses a register, an adder, a comparator, and an OR gate). Q: count. z: output signal asserted only when the maximum count (199) is reached.



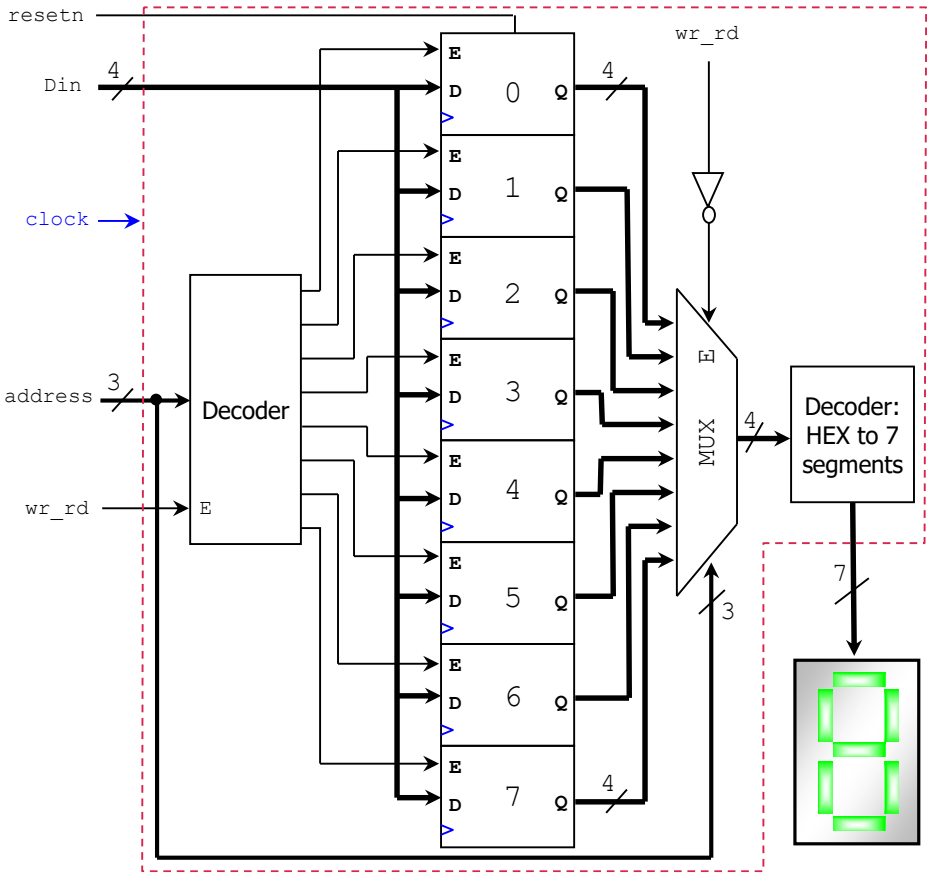


RANDOM ACCESS MEMORY EMULATOR

- The following sequential circuit represents a memory with 8 addresses, where each address holds a 4-bit data. The memory positions are implemented by 4-bit registers. The reset and clock signals are shared by all the registers. Data is written or read onto/from one of the registers (selected by the signal $address$).

- **Writing onto memory** ($wr\_rd = 1$): The 4-bit input data (D_in) is written into one of the 8 registers. The address signal selects which register is to be written. Here, the 7-segment display must show 0. For example: if address = "101", then D_in is written into register 5.

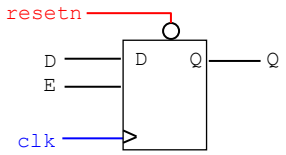
- **Reading from memory** ($wr\_rd = 0$): The MUX output appears on the 7-segment display (hexadecimal value). The address signal selects the register from which data is read.
  For example: If address = "010", then data in register 2 must appear on the 7-segment display. If data in register 2 is '1010', then the symbol 'A' appears on the 7-segment display.



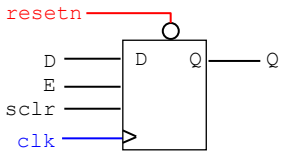
SUMMARY OF COMMON COMPONENTS AVAILABLE IN PARAMETERIZED VHDL CODE:

- The following components are popular in the design of digital logic circuits and their generic VHDL code is available:

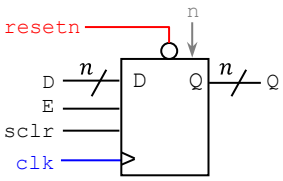
✓ D-type flip flop with enable: dffe



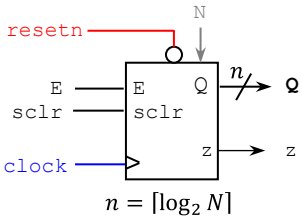
✓ D-type flip flop with enable and synchronous clear: dffes



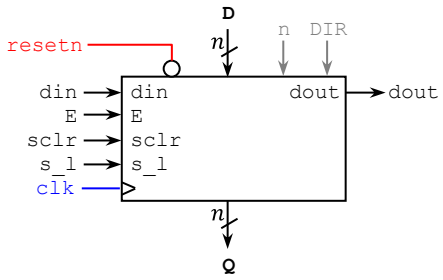
✓ n-bit register with enable and synchronous clear: my_rege
- Parameter (VHDL code): n (number of bits)



✓ Counter modulo-N with enable and synchronous clear: my_genpulse_sclr
- Parameter (VHDL code): N (sometimes COUNT is used)
- This circuit counts from 0 to N-1 (this is the maximum count).
- The number of bits of the output Q is given by $n = \lceil \log_2 N \rceil$
- z: output of a comparator between Q and N-1. z = '1' when Q = N-1, else z = '0'.
- Depending on the application, some inputs/outputs might not be used:
  - The output Q might not be used. Only z suffices.
  - The output z might not be used. The count Q suffices.
  - The input sclr might not be used. In this case, you need to tie it to '0' so it doesn't affect the counter behavior.
  - The enable input (E) might not be used. In this case, you need to tie it to '1', so the counter works.



✓ n-bit parallel access (right/left) register with enable and synchronous clear: my_pashiftreg_sclr
- Parameters (VHDL code):
  - n (number of bits)
  - DIR (direction): "LEFT", "RIGHT"
- din: serial input (sometimes also called 'w')
- dout: serial output (also called shiftout):
  - dout = Q_0 if DIR = "RIGHT"
  - dout = Q_{N-1} if DIR= "LEFT"
- Depending on the application, some inputs/outputs might not be used:
  - The output Q might not used. Only dout suffices.
  - The output dout might not be used. The output Q suffices.
  - The input sclr might not be used. In this case, you need to tie it to '0' so it doesn't affect the counter behavior.
  - The enable input (E) might not be used. In this case, you need to tie it to '1', so the counter works.
- s_l: It controls whether we shift in din (s_l= 0) or load (s_l=1) n-bit D. If you tie this input to a fixed value, you get:
  - If you tie s_l to '1', then the circuit becomes just an n-bit register.
  - If you tie s_l to '0', then the circuit is just a left/right shift register.



- Generic components: Summary of the behavior on the clock tick:

n-bit register: If E=0, the output is kept	Counter modulo-N: If E=0, the count stays.	n-bit Parallel access left/right shift register: If E=0, the output is kept

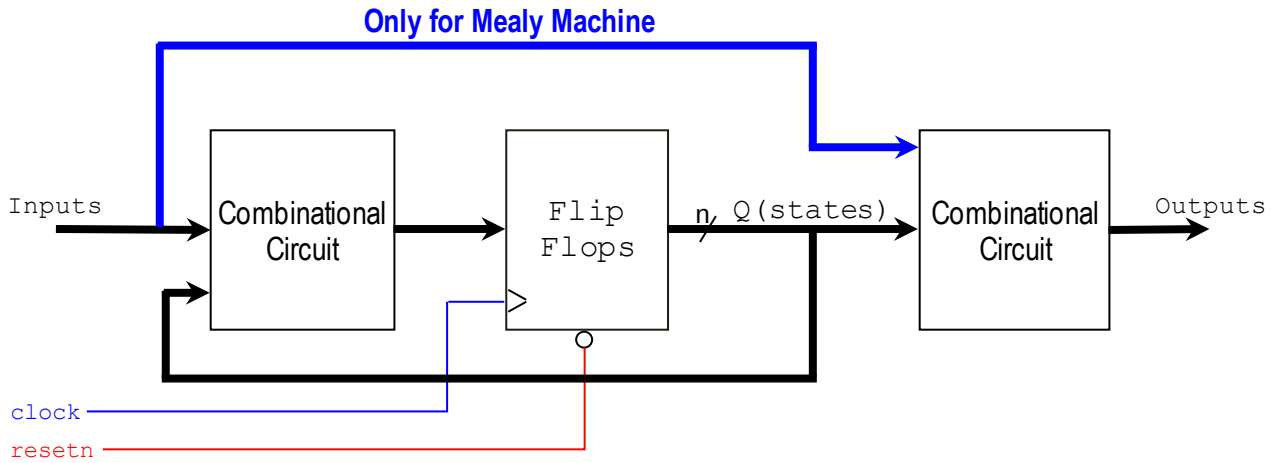
`if E = 1 then`
`  if sclr = 1 then`
`  Q ← 0`
`  else`
`  Q ← D`
`  end if;`
`end if;`
	`if E = 1 then`
	`  if sclr = 1 then`
	`  Q ← 0`
	`  elseif Q = N-1 then`
	`  Q ← 0`
	`  else`
	`  Q ← Q+1`
	`  end if;`
	`end if;`
	
	`* z = 1 if Q = N-1`
		`if E = 1 then`
		`  if sclr = 1 then`
		`  Q ← 0`
		`  elseif s_l = 1 then`
		`  Q ← D`
		`  else`
		`  Q ← shift in 'din'`
		`  (to the left or right)`
		`  end if;`
		`end if;`

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).
- Suppose you want to design a circuit that counts from 0 to 3 and issues a one-cycle pulse when the maximum count is reached. We can implement this circuit with a counter with some gates. What if we want a two-cycle pulse? We can, with painstaking effort, implement this circuit with a counter, flip flops, and logic gates. The Finite State Machine method offers a systematic way of implementing this circuit.

FSM MODEL

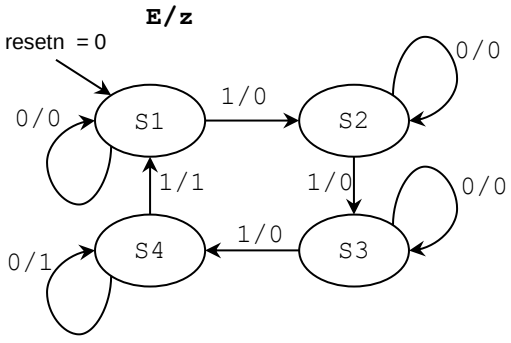
- The figure below represents the Finite State Machine model. Any digital circuit that performs some sequential control can be represented by this model.
- Classification:
  - **Moore machine**: Outputs depend solely on the current state of the flip flops.
  - **Mealy machine**: Outputs depend on the current state of the flip flops as well as on the input to the circuit.
    Important: Since the inputs can change at any time, the outputs are not restricted to only change on the clock edges.
- The signal $resetn$ sets the flip flops to an initial state.



- 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 Finite State Machines is a powerful method in Digital Logic Design.

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

- *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.



Input E	Present State	Next State	Outputs C_1	C_0	z
0	S1	S1	0	0	0
0	S2	S2	0	1	0
0	S3	S3	1	0	0
0	S4	S4	1	1	1
1	S1	S2	0	0	0
1	S2	S3	0	1	0
1	S3	S4	1	0	0
1	S4	S1	1	1	1

- *Second step*: State Assignment. We assign unique flip flop states to our state labels (S1, S2, S3, S4). This assignment is arbitrary (e.g.: S1: Q=10, S2: Q=11, S3: Q=00, S4: Q=01). However, we can simplify our procedure if we assign each state so that they follow the desired count:
  - ✓ S1: Q = 00
  - ✓ S2: Q = 01
  - ✓ S3: Q = 10
  - ✓ S4: Q = 11

  This way, the output C_1C_0 (the count) matches the states encoded in binary (i.e., FSM flip flops outputs): $C_1 = Q_1$, $C_0 = Q_0$.

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

Input	Present State		Next State		Outputs		
E	$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$	C_1	C_0	z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	0	0	0
1	0	1	1	0	0	1	0
1	1	0	1	1	1	0	0
1	1	1	0	0	1	1	1

- Note that $C_1C_0 = Q_1Q_0$. This is very common when designing counters using this method.
- As a result, it is customary to omit the output signals C_1 and C_0 in the State Diagram and Table, since Q_1Q_0 (the binary-encoded states) already represent the counter output.

- *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.

$Q_1(t+1)$

Q_0 \ EQ_1	00	01	11	10
0	0	1	1	0
1	0	1	0	1

$Q_0(t+1)$

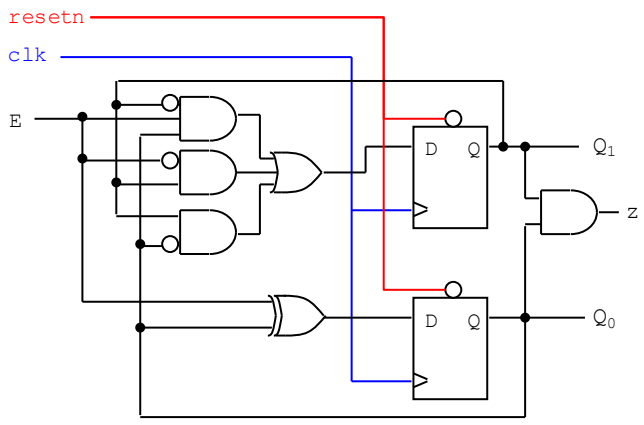
Q_0 \ EQ_1	00	01	11	10
0	0	0	1	1
1	1	1	0	0

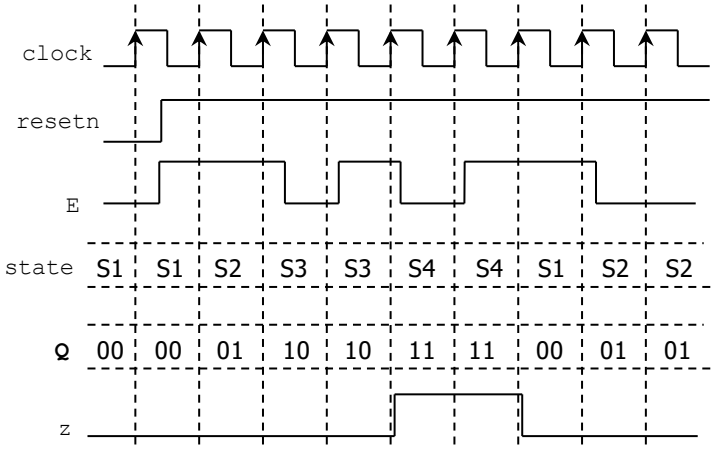
z

Q_0 \ EQ_1	00	01	11	10
0	0	0	0	0
1	0	1	1	0

$Q_1(t+1) \leftarrow Q_1\overline{Q_0} + \bar{E}Q_1 + E\overline{Q_1}Q_0$
$Q_0(t+1) \leftarrow E\overline{Q_0} + \bar{E}Q_0$
$z = Q_1Q_0$ → This is called Boolean output equation; it is always valid (not just on the clock edge)
Output z only depends on the present state. Outputs Q_1, Q_0 are the states and they only depend (in terms of the combinational output circuit) on the present state. Thus, this is a Moore FSM.

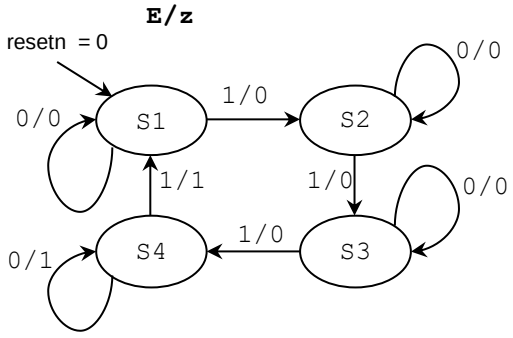
- *Fifth step:* Circuit implementation.





Example: 2-bit gray-code counter with enable and z output: 00, 01, 11, 10, 00, … The output z is 1 if 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.



Input	Present State	Next State	Outputs		
E			C_1	C_0	z
0	S1	S1	0	0	0
0	S2	S2	0	1	0
0	S3	S3	1	1	0
0	S4	S4	1	0	1
1	S1	S2	0	0	0
1	S2	S3	0	1	0
1	S3	S4	1	1	0
1	S4	S1	1	0	1

- *Second step*: State Assignment. We assign unique flip flop states to our state labels (S1, S2, S3, S4). This assignment is arbitrary. However, we can simplify our procedure if we assign each state so that they follow the desired count:
  - ✓ S1: Q = 00
  - ✓ S2: Q = 01
  - ✓ S3: Q = 11
  - ✓ S4: Q = 10

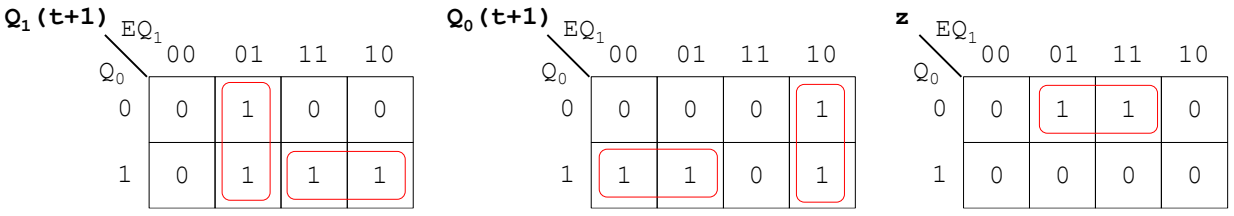
  This way, the output C_1C_0 (the count) matches the states encoded in binary (i.e., FSM flip flops outputs): $C_1 = Q_1$, $C_0 = Q_0$.

  - Alternatively, we could make S1: Q=00, S2: Q=01, S3: Q=10, S4: Q=11. Here the output C_1C_0 will not match Q_1Q_0 (the binary-encoded states), though it makes the encoding of the states more consistent.

- *Third step*: Excitation table. Here, we replace the state labels by the flip flop states (we use the simpler state assignment):

Input	Present State		Next State		Outputs		
E	$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$	C_1	C_0	z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	1	1	1	1	1	0
0	1	0	1	0	1	0	1
1	0	0	0	1	0	0	0
1	0	1	1	1	0	1	0
1	1	1	1	0	1	1	0
1	1	0	0	0	1	0	1

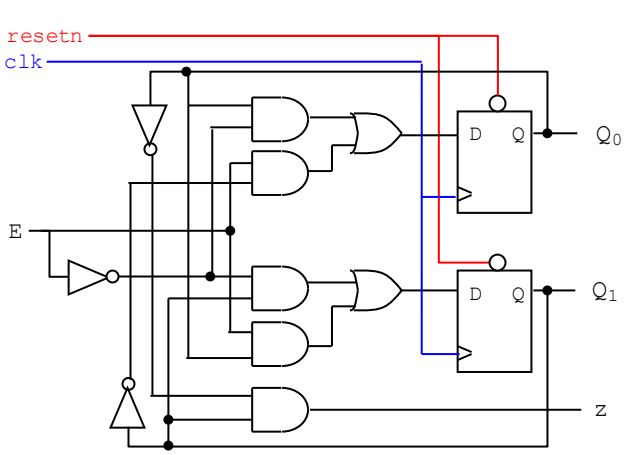
- *Fourth step*: Excitation equations and minimization. $Q_1(t+1)$ and $Q_0(t+1)$ are the next state of the flip flops, i.e. these signals are to be connected to the inputs of the flip flops.

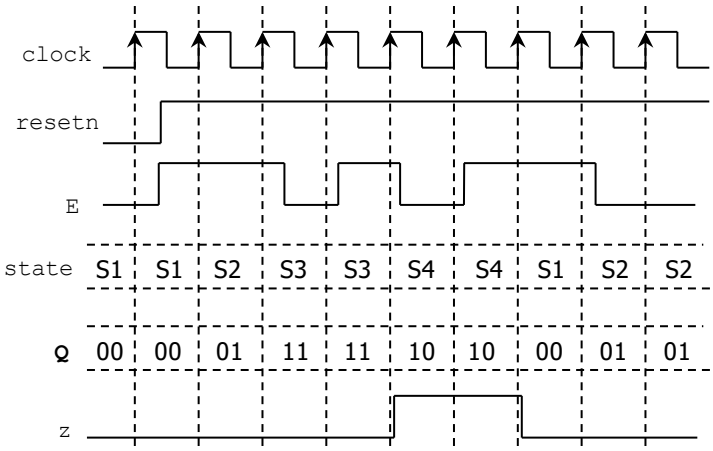


$$Q_1(t+1) \leftarrow \bar{E}Q_1 + EQ_0$$
$$Q_0(t+1) \leftarrow E\overline{Q_1} + \bar{E}Q_0$$
$$z = Q_1\overline{Q_0}$$

Output z only depends on the present state. Outputs Q_1, Q_0 are the states and they only depend (in terms of the combinational output circuit) on the present state. Thus, this is a Moore FSM.

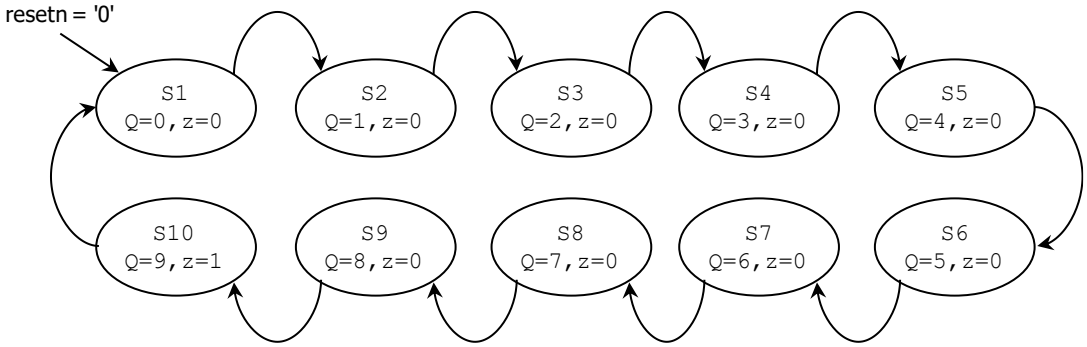
- *Fifth step:* Circuit implementation.



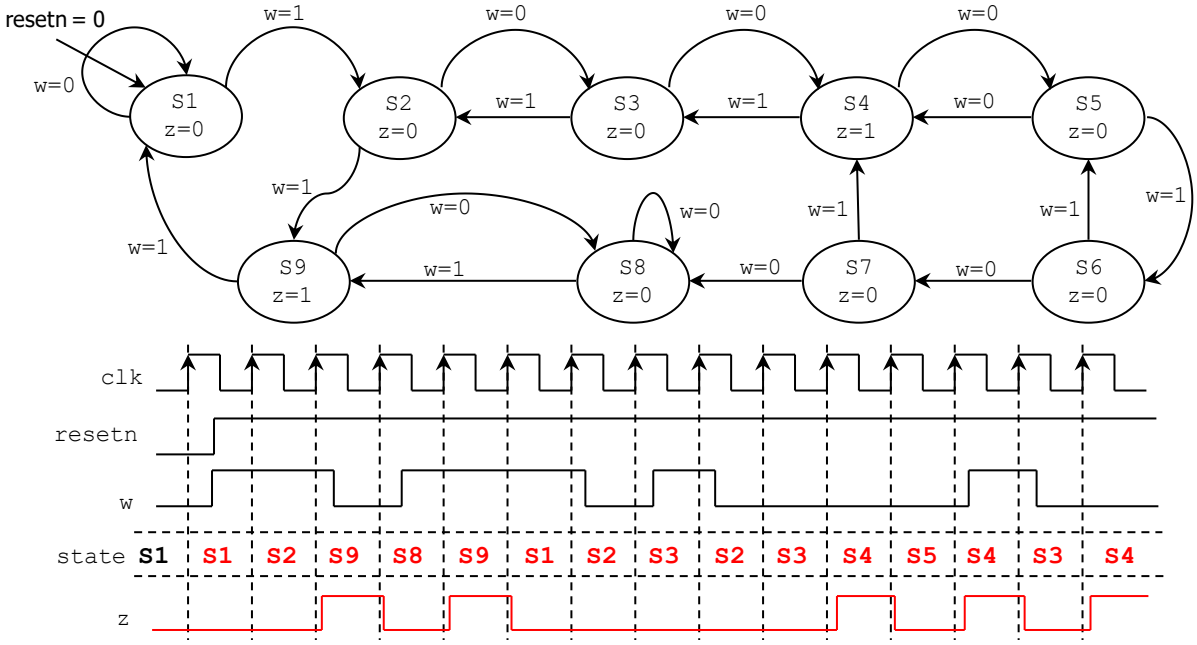


Note: In these 2-bit counters, the states are represented by the outputs of the flip flops: Q_1, Q_0. They also happen to be the outputs of the FSM. This is common in counters, as the count is usually the same as the flip flop outputs.

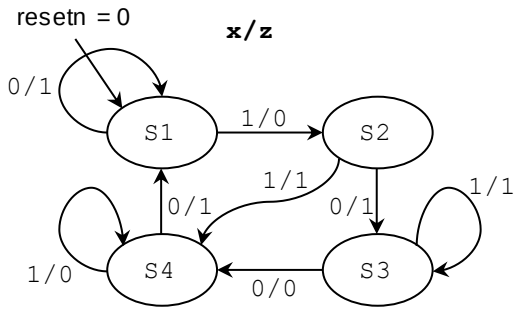
Example: BCD counter implemented as FSM. Outputs: $Q(3..0)$, z. When the count reaches 1001, z becomes 1. Moore FSM

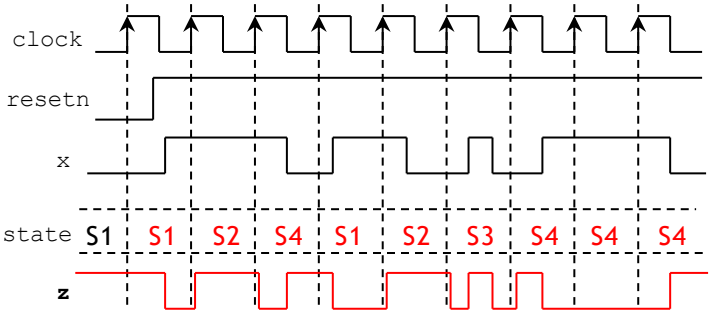


Example: FSM. Input: w. Output: z. This is a Moore FSM as z only depends on the present state.



Example: FSM. Input: x. Output: z. This is a Mealy FSM as z depends on the present state and the input. This implies that z can change during a clock cycle, as in the timing diagram:





We can get the excitation table and excitation equations from the FSM diagram:

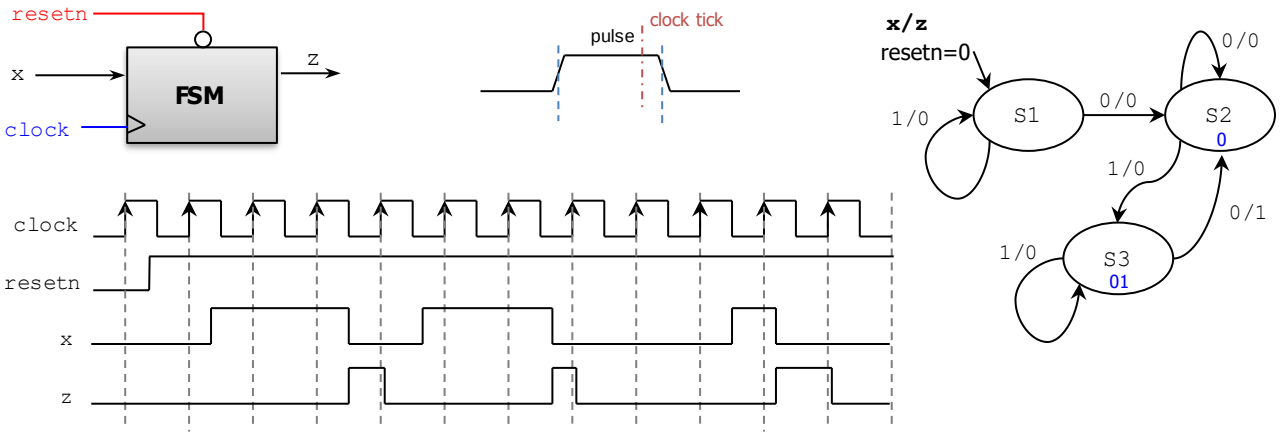
S1: Q = 00
S2: Q = 01
S3: Q = 10
S4: Q = 11

$Q_1(t+1) \leftarrow xQ_1 + (Q_1 \oplus Q_0)$
$Q_0(t+1) \leftarrow x\overline{Q_1} + xQ_0 + \bar{x}Q_1\overline{Q_0}$
$z = \overline{Q_1}Q_0 + \overline{x \oplus (Q_1\overline{Q_0})}$

Input	Present State		Next State		Outputs
x	Q_1 (t)	Q_0 (t)	Q_1 (t+1)	Q_0 (t+1)	z
0	0	0	0	0	1
0	0	1	1	0	1
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	1	0

EXAMPLE: Pulse Detector

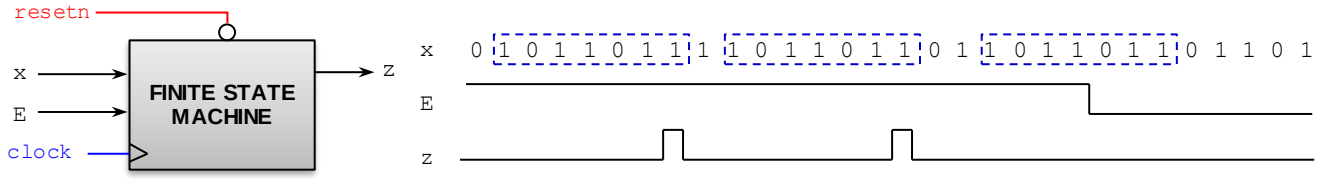
- Circuit with an input x and output z. The machine generates $z = 1$ when it detects a pulse. Note how in this design, the output z is 1 as soon as the $1 \rightarrow 0$ transition is detected.
  Assumption: For the circuit to detect a '1' or a '0' on x, the value needs to happen when a rising edge occurs.
- In order to have a clean output pulse, a common technique is to include a flip flop whose input is z
- The first state (S1) is to make sure that the pulse started at '0'. This is a Mealy FSM.

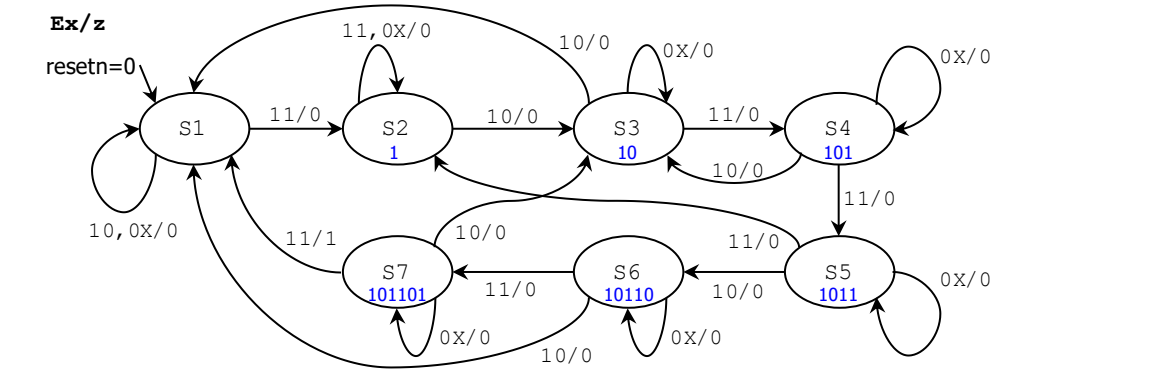


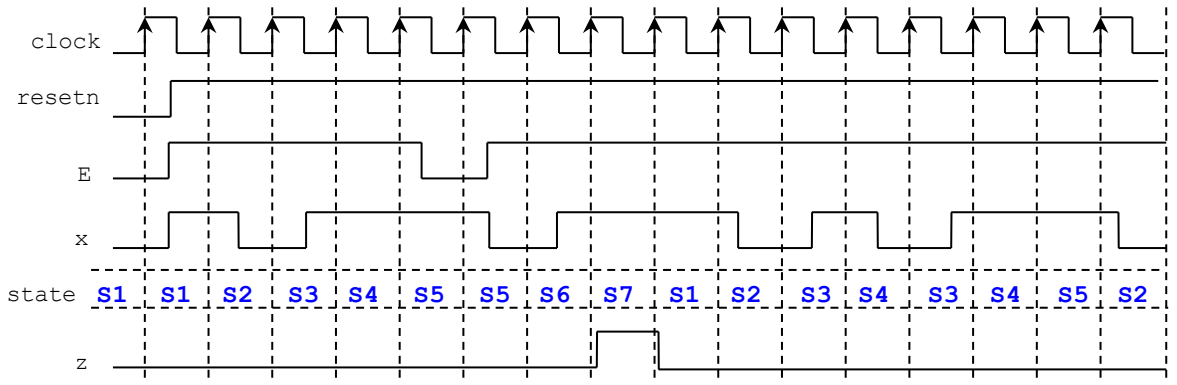
EXAMPLE: Sequence Detector

- Circuit with an input x and output z. The machine generates $z = 1$ when it detects the sequence 1011011. The value of a bit is detected on the clock edge. Right after the sequence is detected, the circuit looks for a new sequence.
  Assumption: For the circuit to detect a '1' or a '0' on x, the value needs to happen when a rising edge occurs.

  Signal E is an input enable: It validates the input x, i.e., if $E = 1$, x is valid, otherwise x is not valid. The figure below illustrates the behavior for a certain input stream. This is a Mealy FSM.



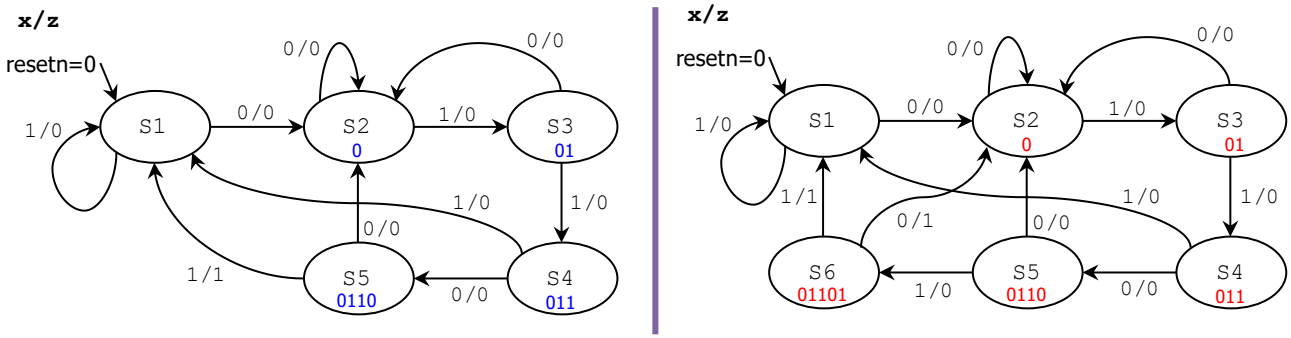


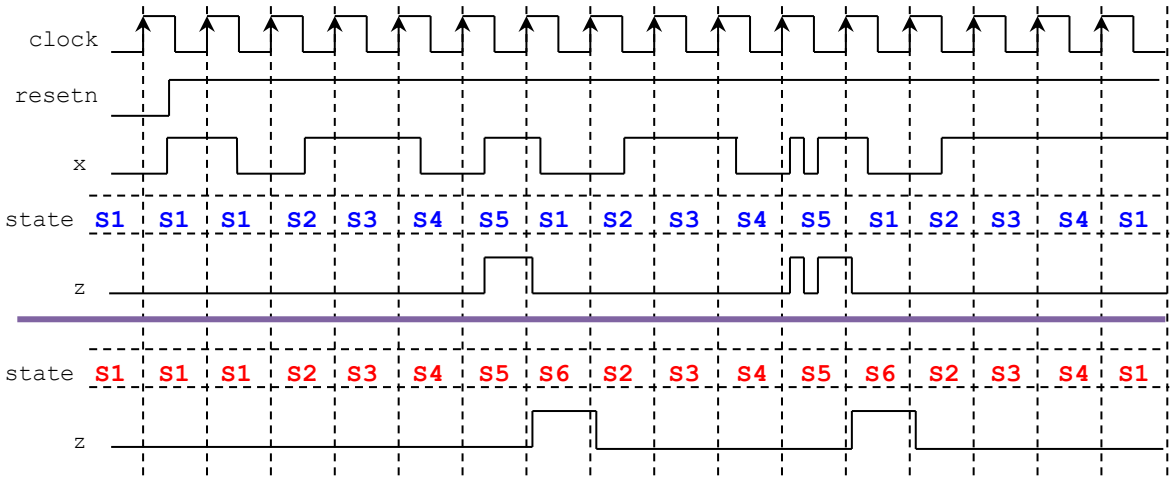


EXAMPLE: Sequence Detector (2 versions)

- Circuit with an input x and output z. The machine generates $z = 1$ when it detects the sequence 01101. The value of a bit is detected on the clock edge. Right after the sequence is detected, the circuit looks for a new sequence.

- Here, two FSM versions are presented:
  - ✓ (left): When the last bit ('1') is detected (i.e., $x = 1$ on the clock edge), the output z will be 0 right after the clock edge. Note that the output z will become 1 as soon as x is 1 in S5. This is a Mealy FSM.
    * In order to have a clean output pulse, a common technique is to include a flip flop whose input is z.
  - ✓ (right): The output z will be 1 for one clock cycle right after the last bit ('1') is detected on the clock edge. This requires 6 states. This is a Moore FSM.

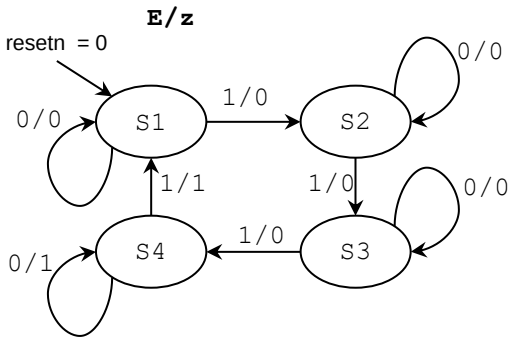


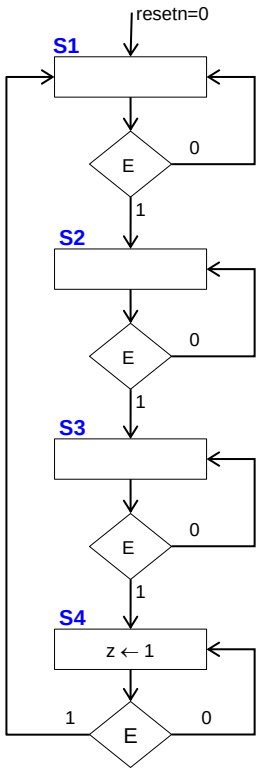


ALGORITHMIC STATE MACHINE (ASM) CHARTS:

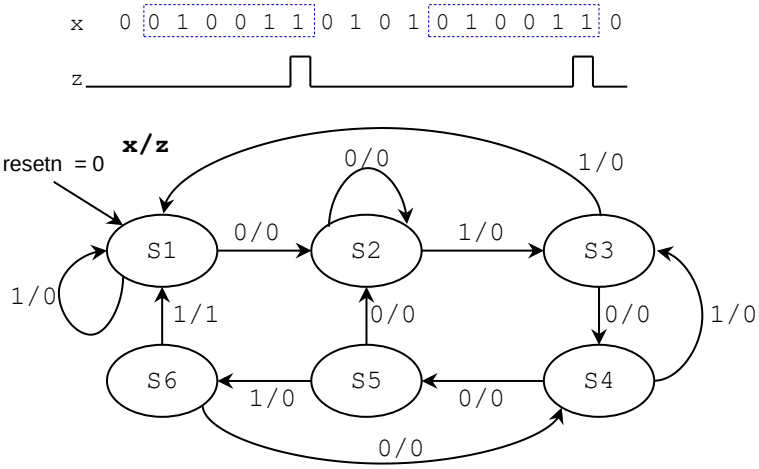
- In ASMs, if an output value is not specified for a certain circumstance (state, input values), it is assumed to be zero. We are only required to indicate the output values when they are assigned '1'.

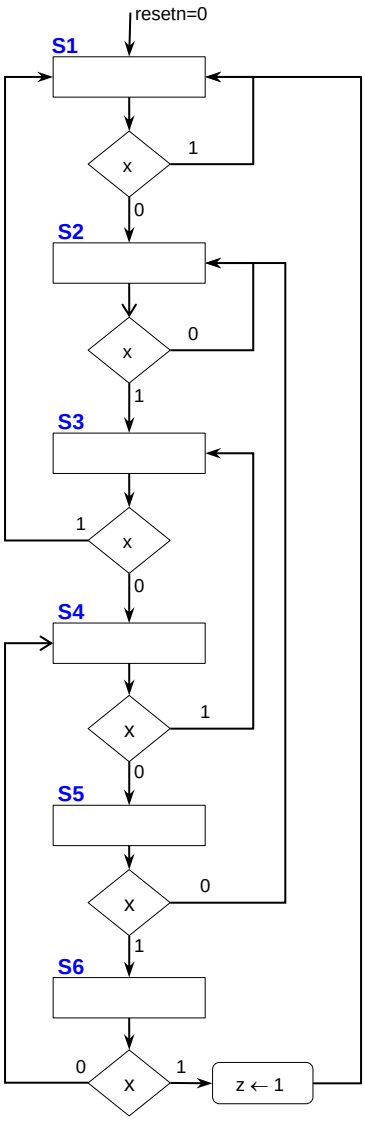
Gray counter, z=1 when Q=10





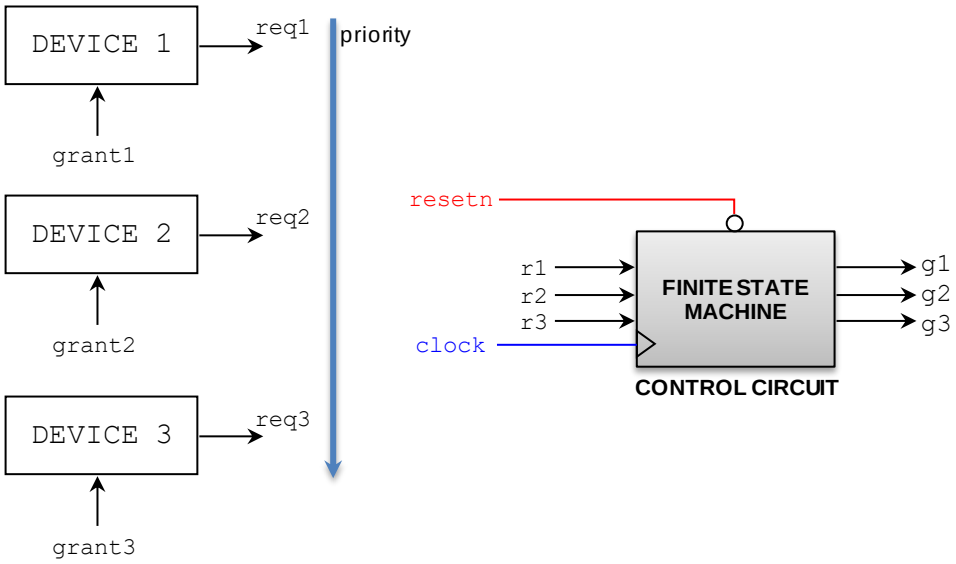
Sequence Detector: 010011



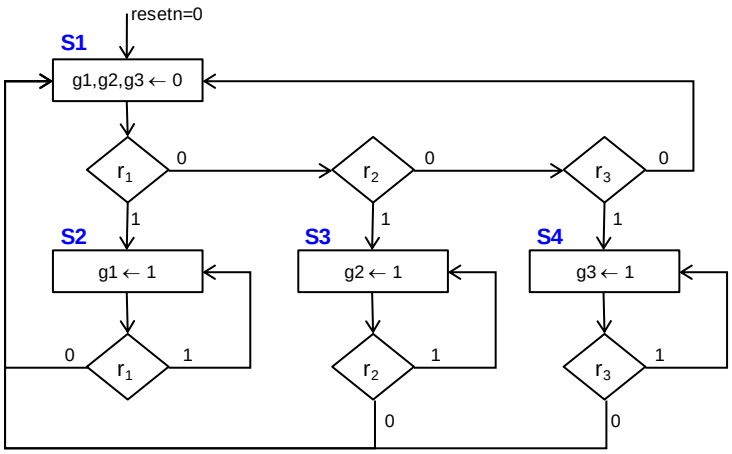


EXAMPLE: ARBITER CIRCUIT

- Three devices can request access to a certain resource at any time (example: access to a bus made of tri-state buffers, only one tri-state buffer can be enabled at a time). The FSM can only grant access to one device at a time.There should be a priority level among devices.
- If the FSM grants access to one device, one must wait until the request signal to that device is deasserted (i.e. set to zero) before granting access to a different device.



- **Algorithmic State Machine (ASM) chart:**



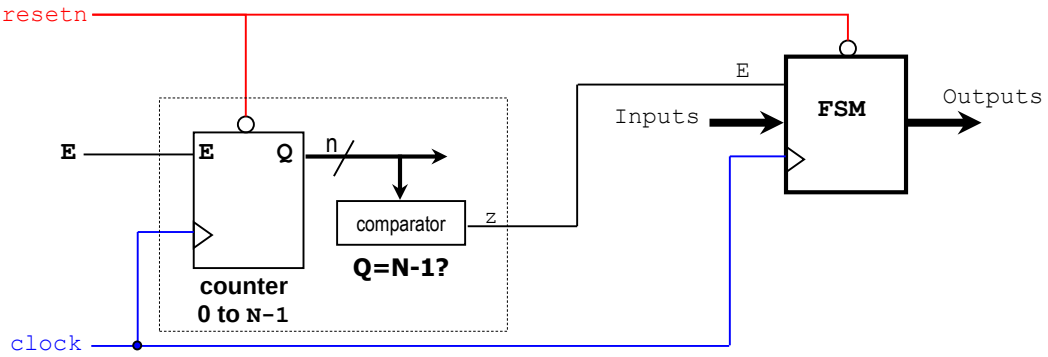
Modifying the rate of change of a Finite State Machine:

- We usually would like to reduce the rate at which FSM transitions occur. A straightforward option is to reduce the frequency of the input clock. But this is a very complicated problem when a high precision clock is required.

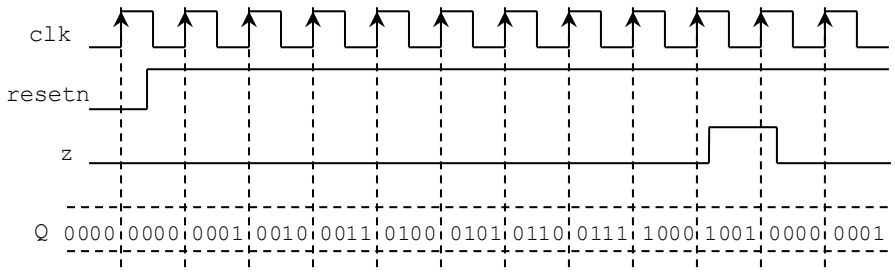
- Alternatively, we can reduce the rate at which FSM transitions occur by including an enable signal in our FSM: this means including an enable to every flip flop in the FSM. For any FSM transition to occur, the enable signal has to be '1'. Then we assert the enable signal only when we need it. The effect is the same as reducing the frequency of the input clock.

  ✓ The figure below depicts a counter modulo-N (from 0 to N-1) connected to a comparator 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.

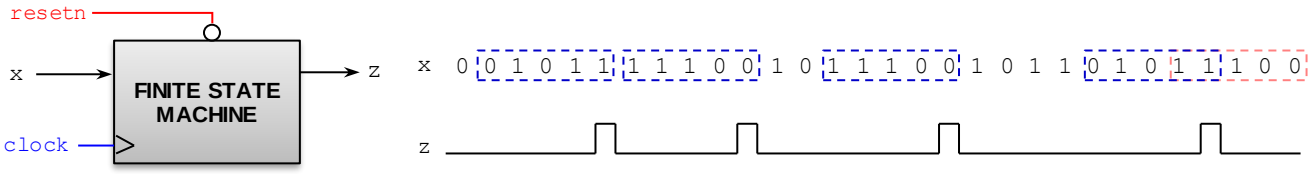


- We can apply the same technique not only to FSMs, but also to any sequential circuit. This way, we can reduce the rate of any sequential circuit (e.g.: another counter) by including an enable signal of every flip flop in the circuit.

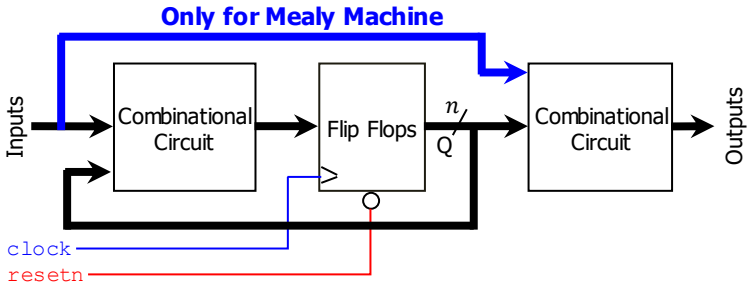
PRACTICE EXERCISES II

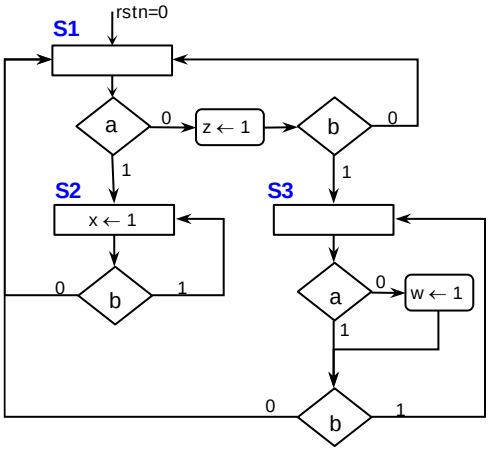
1. Two-pulse Detector: Draw the State Diagram (in ASM form) for a circuit that asserts a signal z when two pulses are detected on its input port x. Once the two pulses are detected, the FSM looks for a new pair of pulses.
   Assumption: For the circuit to detect a '1' or a '0' on x, the value needs to occur when a rising edge happens.

2. Sequence Detector: The circuit has to generate $z = 1$ when it detects the sequence 01011 or 11100. Once a sequence is detected, the circuit looks for a new sequence. Note that once the sequence starts being detected, we prioritize that sequence over the other (example: last sequence inside a dotted red rectangle is not considered).
   ✓ Draw the State Diagram. Also, provide the State Table and Excitation Table of this circuit.



3. Given the following state diagram (in ASM form):
   ✓ Is this a Mealy or a Moore Machine? Why?
   ✓ Get the excitation equations and the Boolean equations for x, z, and w.
   ✓ Sketch the circuit for this Finite State Machine. Identify the parts of your circuit that correspond to the FSM model components.





4. The following VHDL code describes an FSM. Get the excitation equations and Boolean equations for the output signals.

```
library ieee;
use ieee.std_logic_1164.all;

entity circ is
   port ( clk, rstn: in std_logic;
          a, b: in std_logic;
          x,w,z: out std_logic);
end circ;
```

```
architecture behavioral of circ is
   type state is (S1, S2, S3);
   signal y: state;
begin
  Transitions: process (rstn, clk, a, b)
  begin
     if rstn = '0' then y <= S1;
     elsif (clk'event and clk = '1') then
        case y is
           when S1 =>
               if a = '1' then y <= S2; else y <= S3; end if;
            when S2 =>
               if b = '1' then y <= S3; else y <= S1; end if;
            when S3 =>
               if a = '1' then y <= S3; else y <= S1; end if;
          end case;
       end if;
  end process;

  Outputs: process (y,a,b)
  begin
      x <= '0'; w <= '0'; z <= '0';
      case y is
         when S1 => if a = '1' then z <= '1'; end if;
         when S2 => x <= '1';
         when S3 => w <= '1';
      end case;
  end process;
end behavioral;
```