FSM 2 :
Finite State Machines

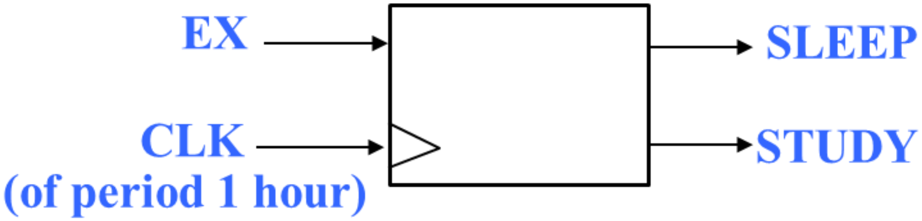
Let’s Try This Out…

Bob is stressed out as he has too many deadlines and isn’t sleeping enough! He has decided to develop a FSM to regulate his time between sleeping and studying.

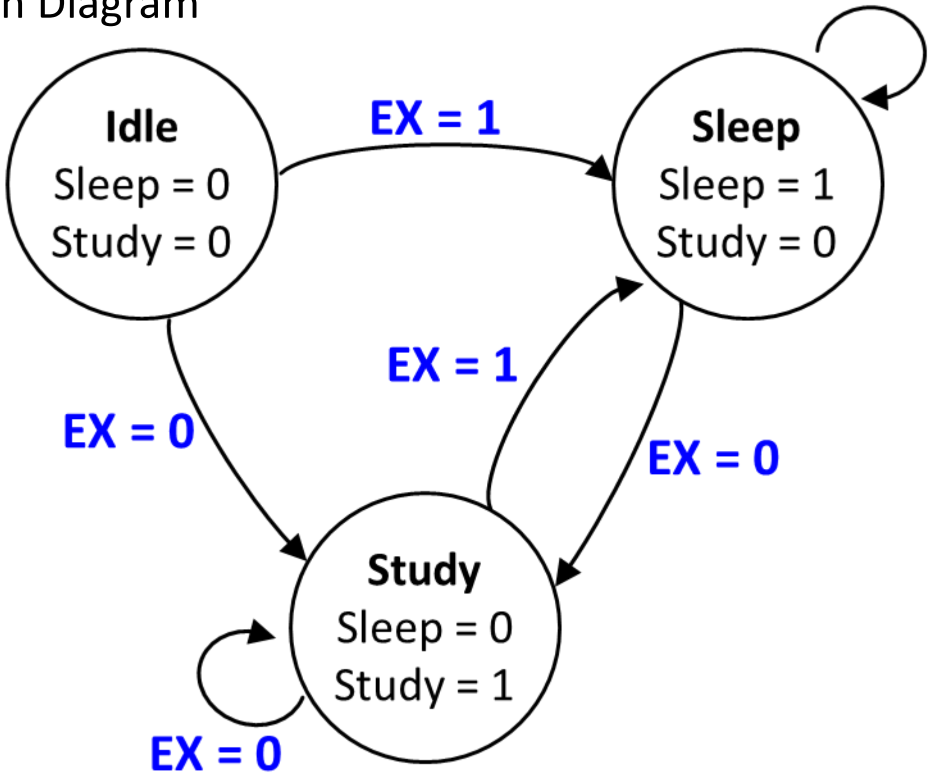
- When he is idle, there are two actions he can do next : SLEEP or STUDY.
- To prevent exhaustion, he buys a body exhaustion sensor and checks it every hour. When he is exhausted, the sensor output EX will be TRUE.
- When he is exhausted, he should sleep (SLEEP is TRUE). When he is not exhausted, he should study (STUDY is TRUE).
- Implement his FSM using D Flip Flops and gates.

Step 1 : Block Diagram, STD

1) Block Diagram

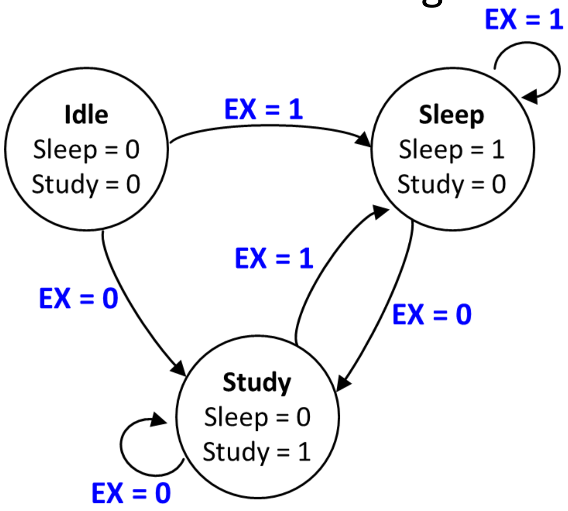


2) State Transition Diagram



Step 2 : Next State Table

State Transition Diagram



Current State Input Next State

S	EX	S+
IDLE	0	STUDY
IDLE	1	SLEEP
SLEEP	0	STUDY
SLEEP	1	SLEEP
STUDY	0	STUDY
STUDY	1	SLEEP

State	S_1S_0
IDLE	00
SLEEP	01
STUDY	10

S_1S_0	EX	S_1+S_0+
0 0	0	1 0
0 0	1	0 1
0 1	0	1 0
0 1	1	0 1
1 0	0	1 0
1 0	1	0 1

Step 2 : Next State Table

S_1S_0	EX	S_1+S_0+
0 0	0	1 0
0 0	1	0 1
0 1	0	1 0
0 1	1	0 1
1 0	0	1 0
1 0	1	0 1

D_1=
1
0
1
0
1
0

D_0=
0
1
0
1
0
1

D_1 :

EX \ S_1S_0	00	01	11	10
0	1	1	X	1
1	0	0	X	0

$S_1+ = \overline{EX}$

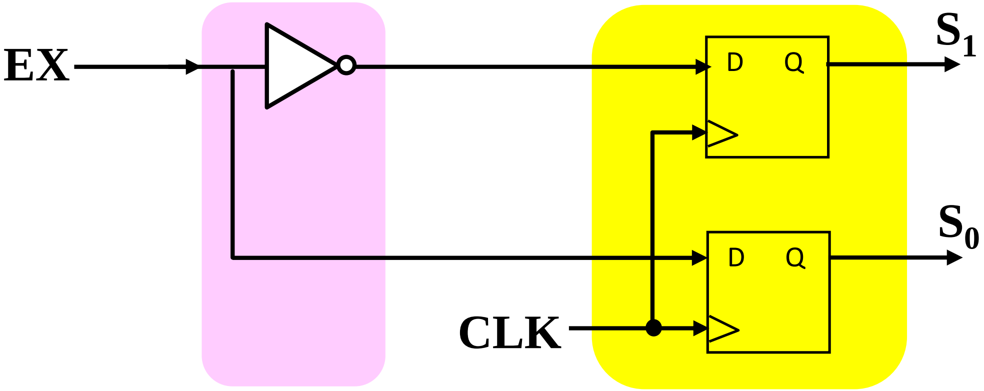
D_0 :

EX \ S_1S_0	00	01	11	10
0	0	0	X	0
1	1	1	X	1

$S_0+ = EX$

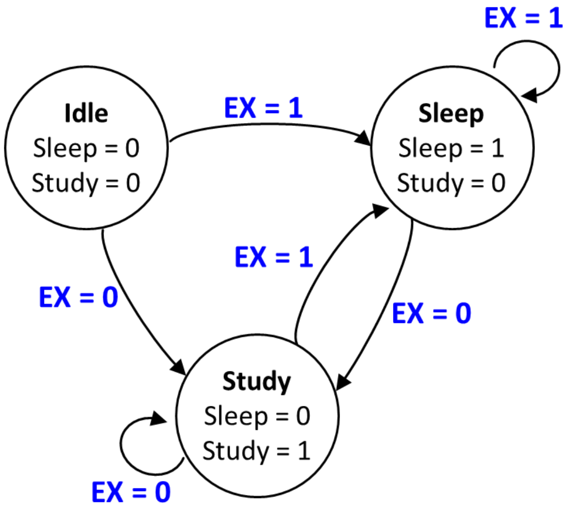
Step 2 : Next State Table

If the state assignments were to change, would the next state logic change?



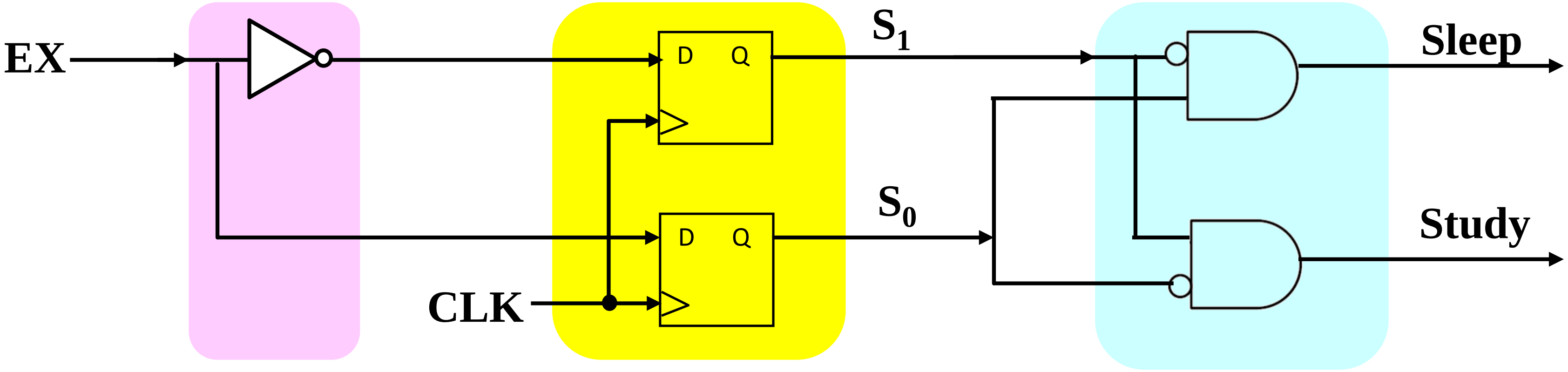
Step 3 : Output Logic

State	S_1S_0
IDLE	00
SLEEP	01
STUDY	10



S_1S_0	Sleep	Study
0 0	0	0
0 1	1	0
1 0	0	1

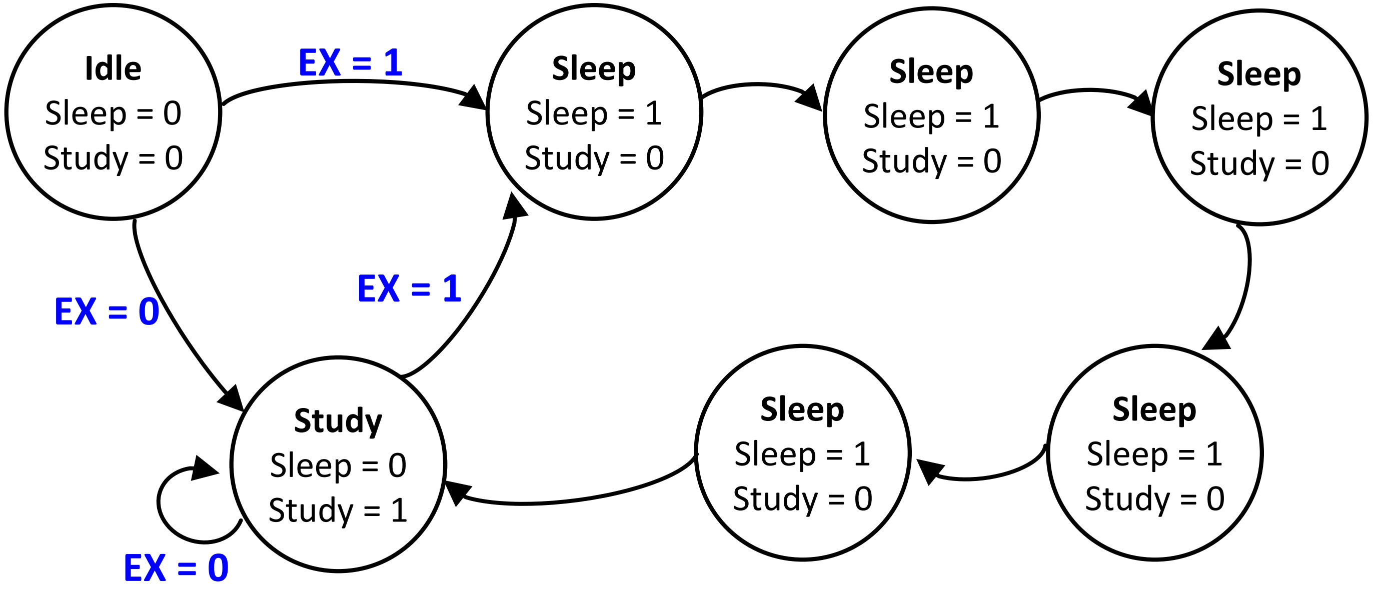
$$\text{Sleep} = \overline{S_1}S_0 \qquad \text{Study} = S_1\overline{S_0}$$



Changing Sleep Time…

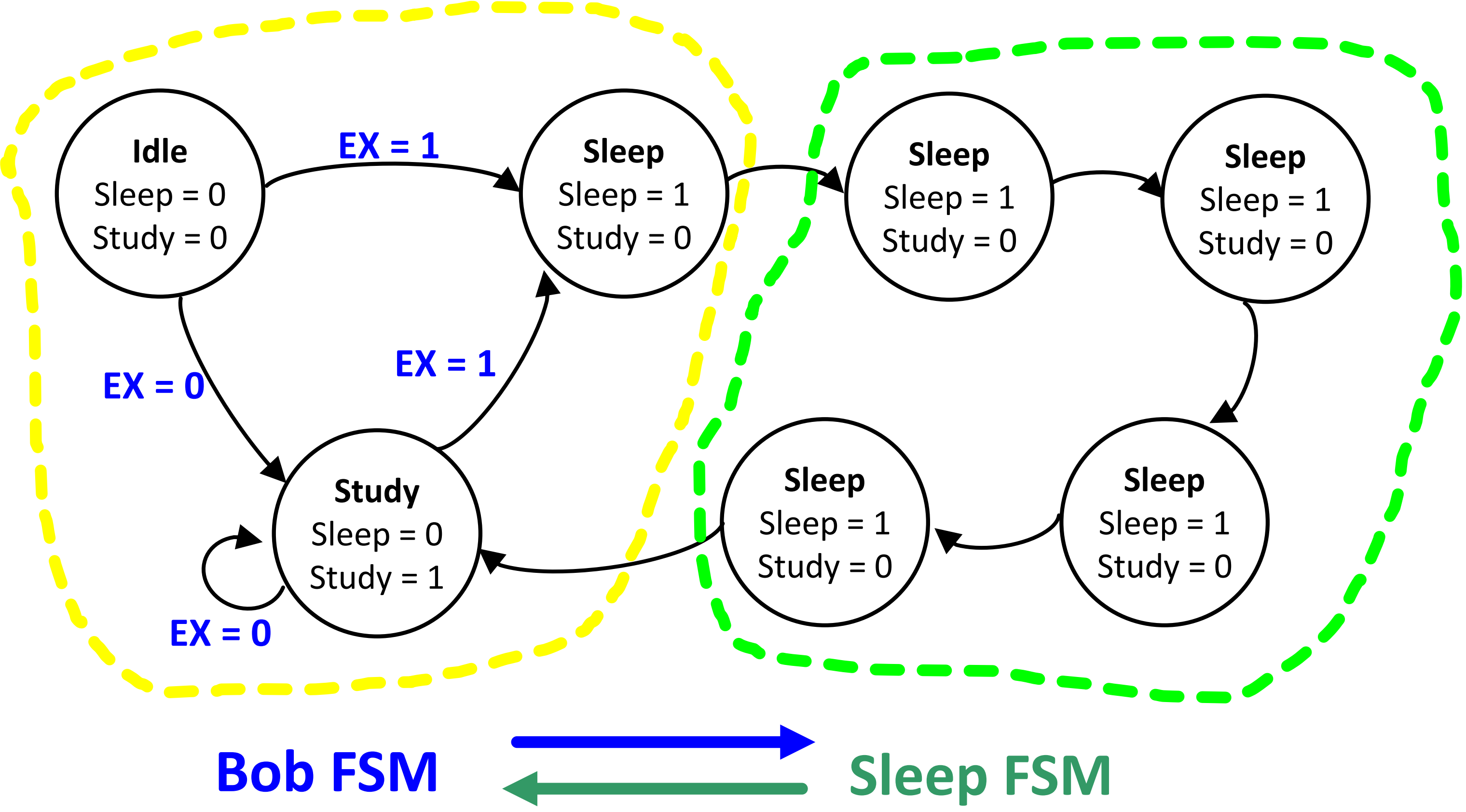
Bob has decided that he is sleep state for too long and has decided to fix his sleep time to 5 hours.

How can we modify his state transition diagram?

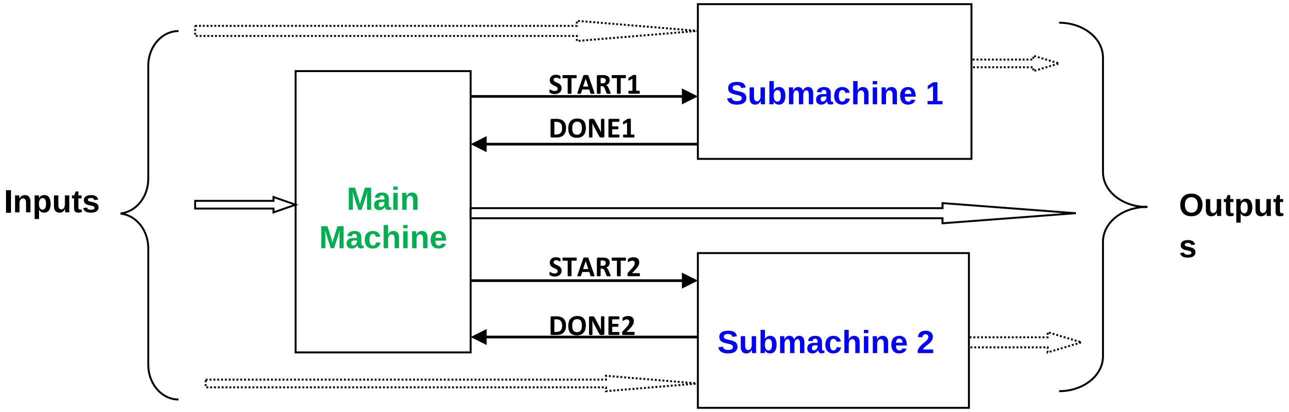


Modular Design of FSMs

Designing complex FSMs is often easier if they can be broken down into simpler FSMs that interact.



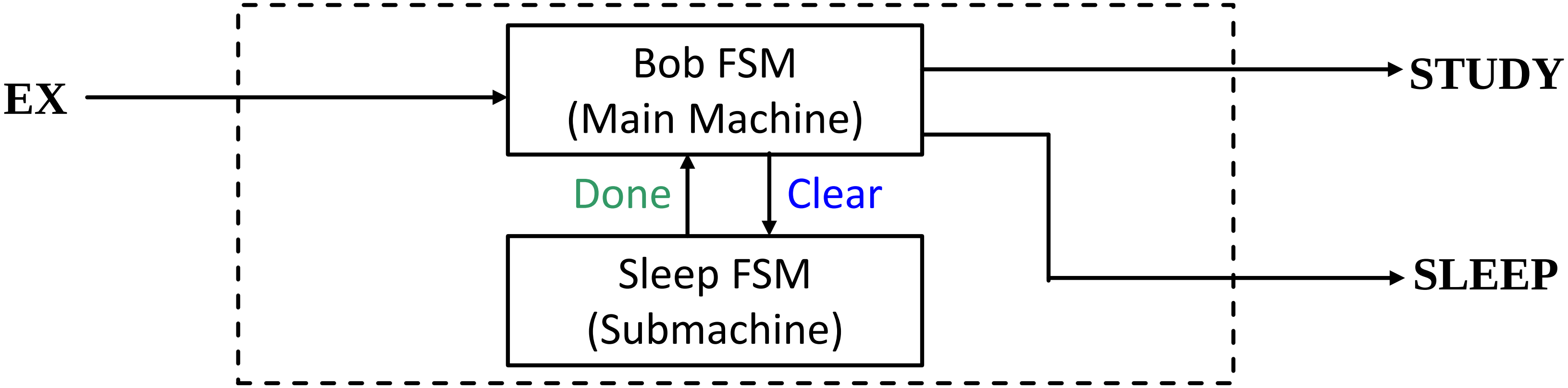
Modular Design of State Machines



- **Main machine :** executes main algorithm, controls the submachines & get the job done. Commands & gets feedback signals from submachines.
- **Submachines** respond to external inputs & commands from main machine. Can give outputs as well as feedback to the main machine.
- Common examples of submachines are ***counters***, ***shift registers***, etc.
- Sometimes the main machine is called the *controller* and the **submachines** are called *controlled circuit elements* or ***architectural elements***.
- Trick here is to **modularize** appropriately, and pick suitable components for the submachines that simplify the design problem.

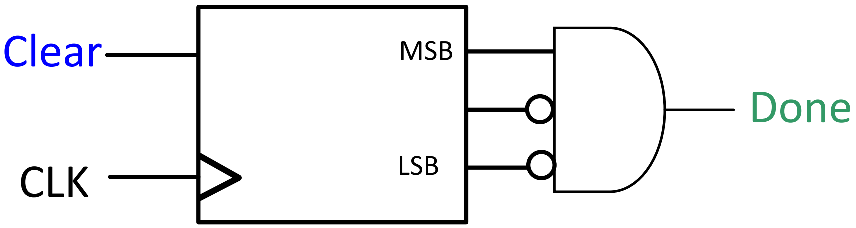
Modularizing…

What is a natural submachine that we can use?



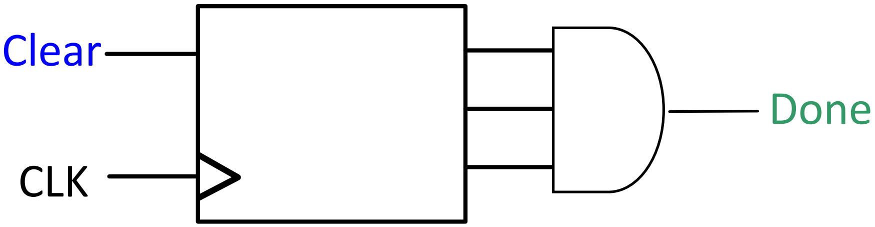
What inputs would you provide? What outputs would you want?

Mod-5 Counter



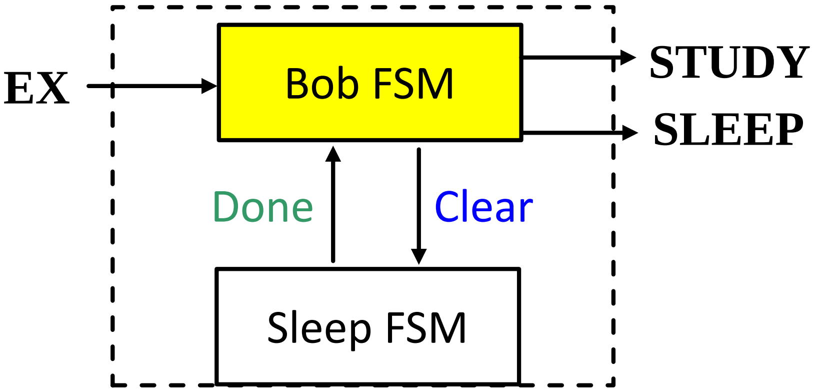
Done = 1 *when* count = 4

Mod-X Counter

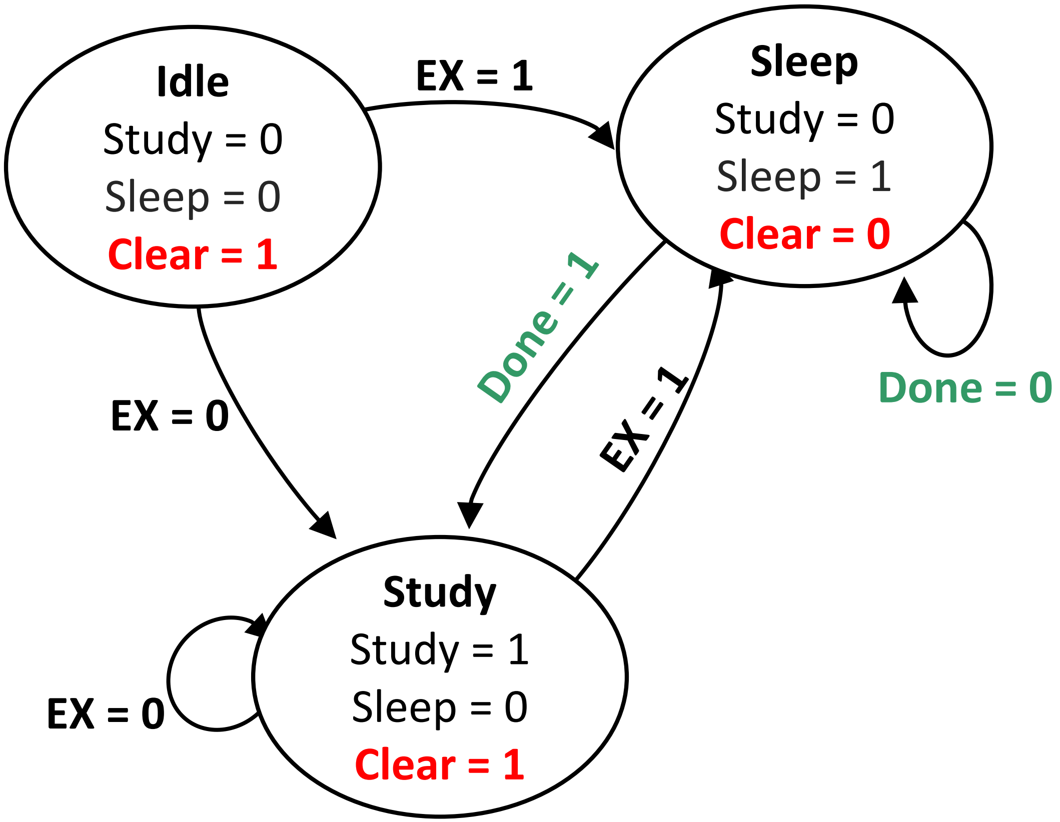


Done = 1 *when* count = (X-1)

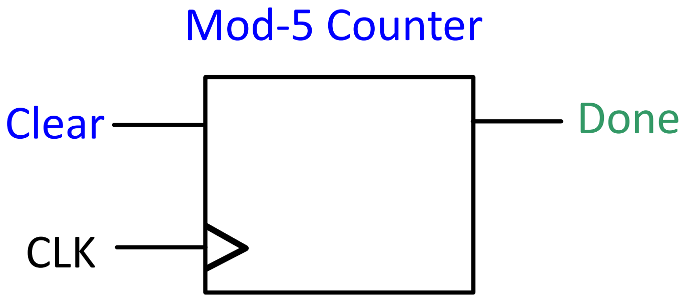
Modified FSM



Bob FSM

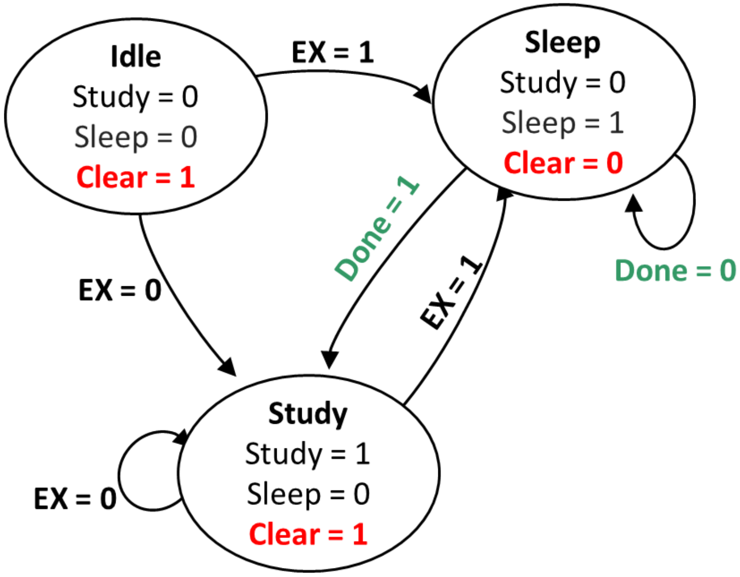


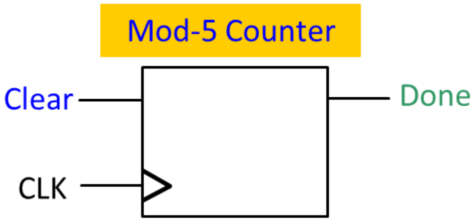
Sleep FSM



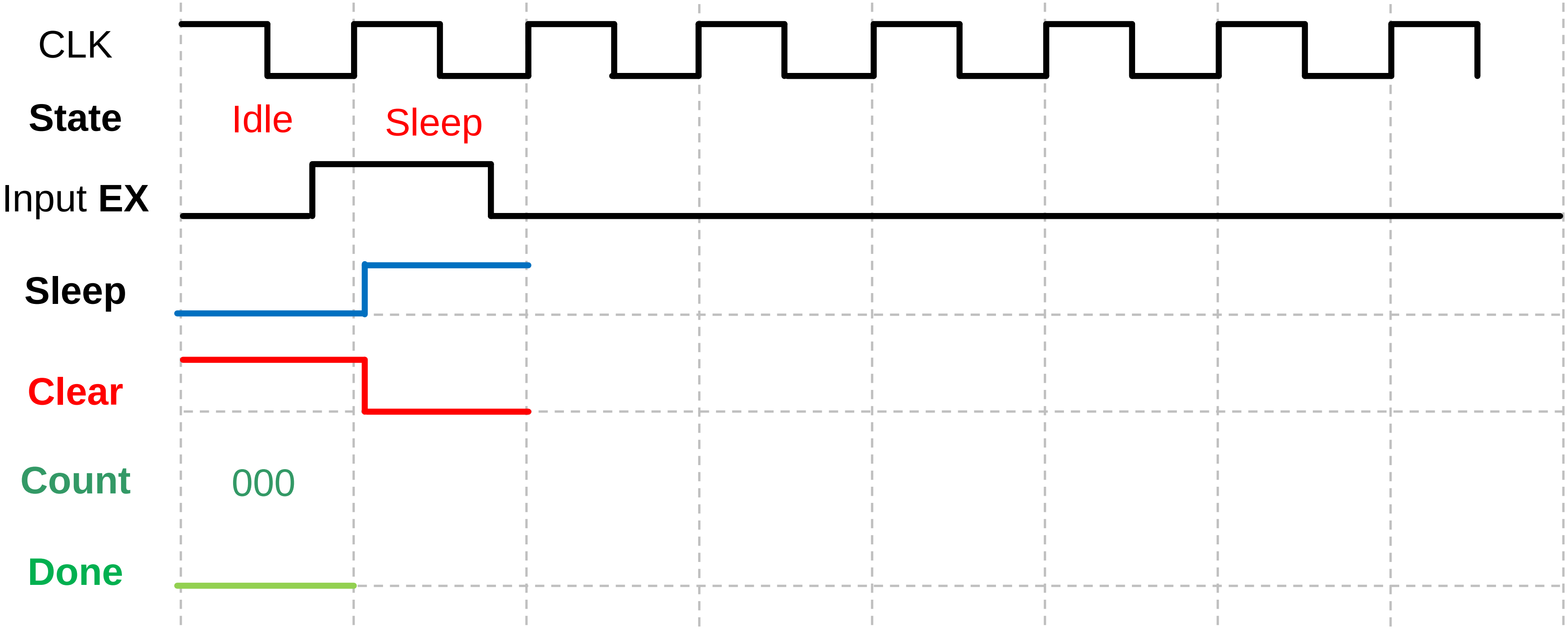
Done = 1 *when* count = 4 (100)

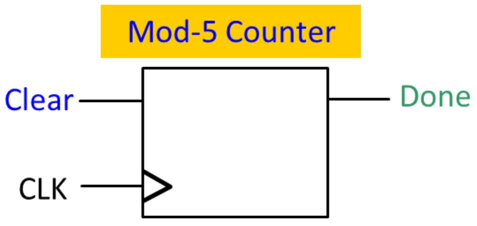
Fill in the timing diagram for Bob FSM and the counter.





Done = 1 *when* count = 4 (100)





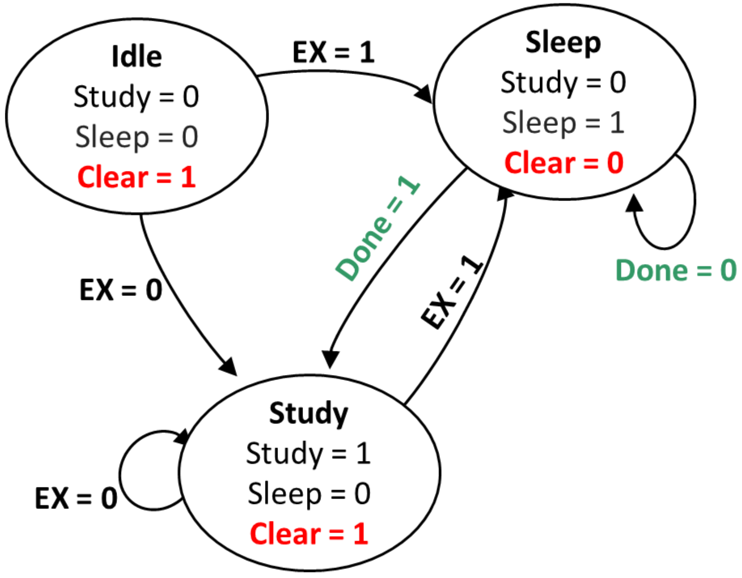
Mod-5 Counter

Clear

CLK

Done

Done = 1 *when* count = 4 (100)



Idle
Study = 0
Sleep = 0
Clear = 1

Sleep
Study = 0
Sleep = 1
Clear = 0

Study
Study = 1
Sleep = 0
Clear = 1

EX = 1

EX = 0

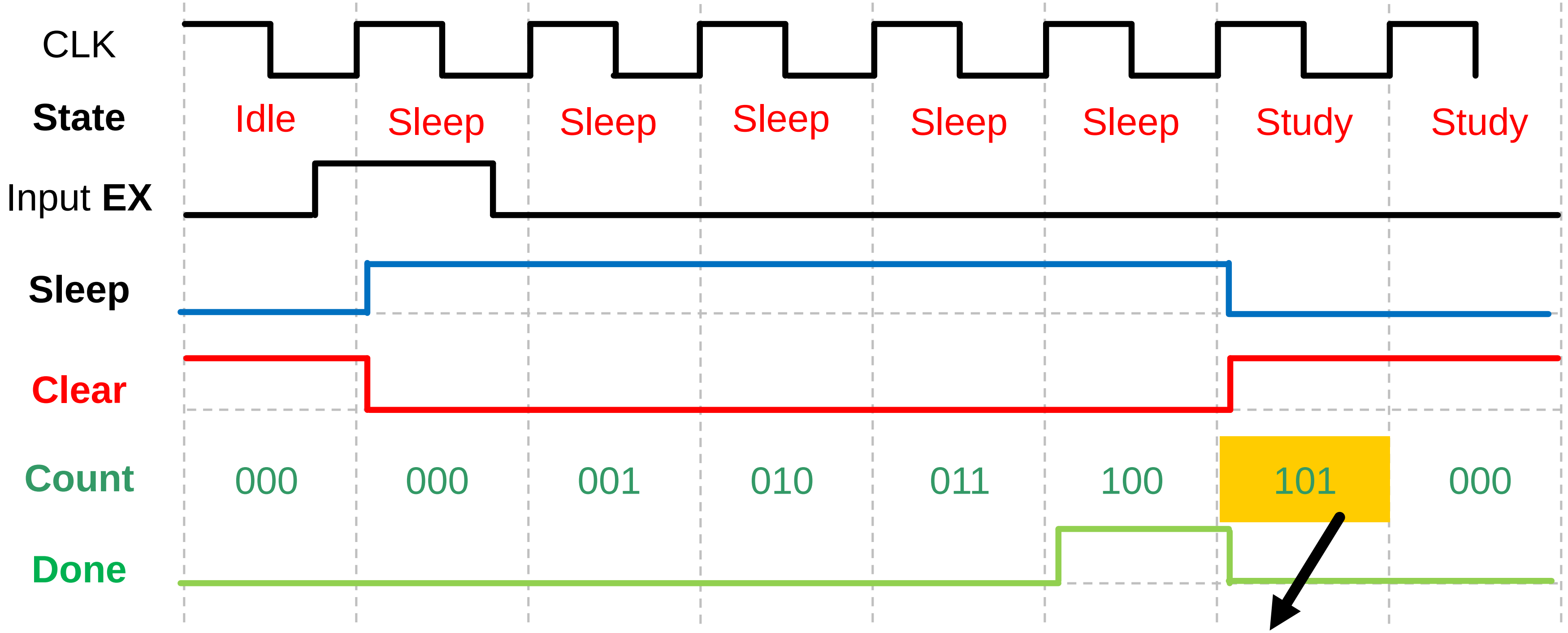
Done = 1

EX = 1

Done = 0

EX = 0

Timing Diagram:



For a mod-5 counter, this should be 000. However, notice that even if the count continues to increase, operation of Bob FSM is not affected!