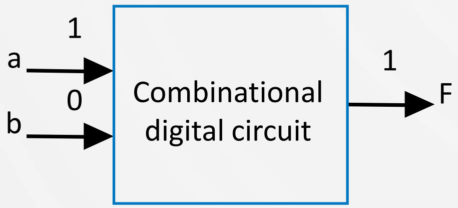
FSM PRIMER

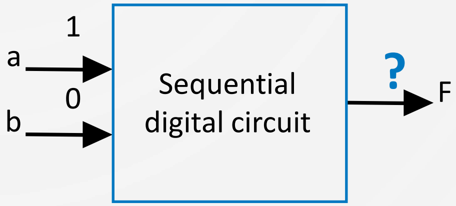
FPGA/VERILOG FROM SCRATCH

DAY 3

C3.1: FSM INTRODUCTION

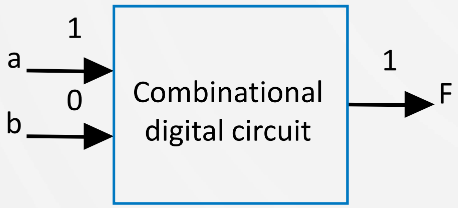
- Before diving into finite state machines, we need to know what the sequential circuit is?

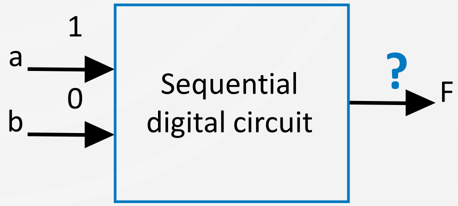




C3.1: FSM INTRODUCTION

- Before diving into finite state machines, we need to know what the sequential circuit is?





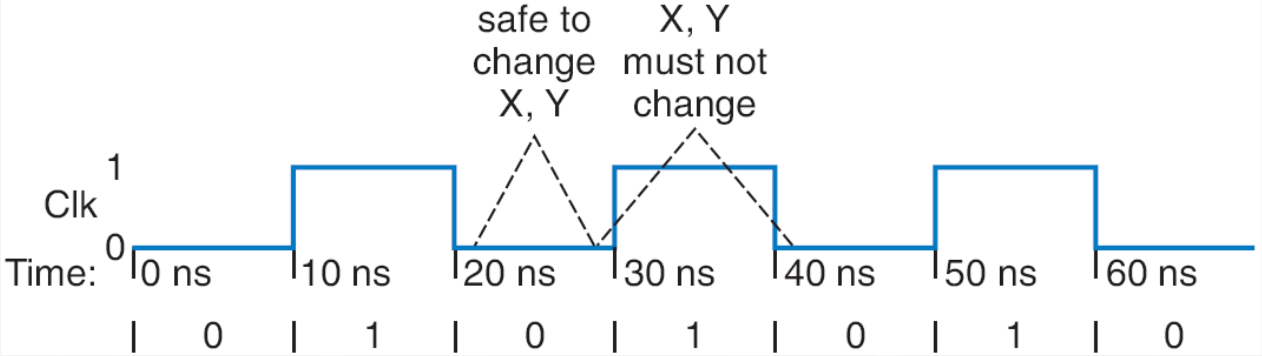
Must know sequence of past inputs to know output

- **Sequential circuit**
  - Output depends not just on present inputs (as in combinational circuit), but on past sequence of inputs
    - **Stores** bits, also known as having "state"
  - Simple example: a circuit that counts up in binary

CLOCK SIGNAL

- If the sequential circuits need not just current inputs but also the past state of the circuit, then we need to add **storage** and **time** features.

CLOCK SIGNAL



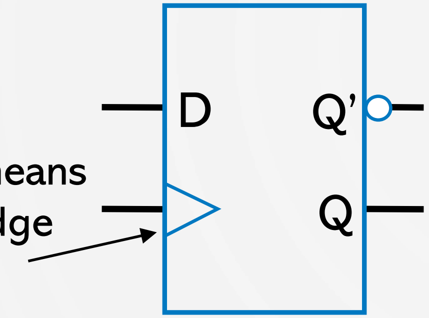
X, Y are irrelevant to this slide

- **Clock period**: time interval between pulses
  - Above signal: period = 20 ns
- **Clock cycle:** one such time interval
  - Above signal shows 3.5 clock cycles
- **Clock frequency**: 1/period
  - Above signal: frequency = 1 / 20 ns = 50 MHz
    - 1 Hz = 1/s

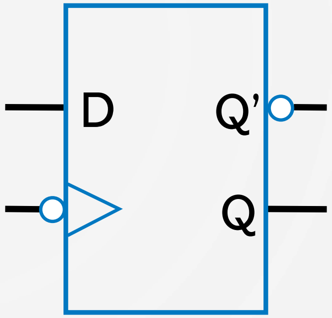
Freq	Period
100 GHz	0.01 ns
10 GHz	0.1 ns
1 GHz	1 ns
100 MHz	10 ns
10 MHz	100 ns

D FLIP-FLOP

The triangle means clock input, edge triggered



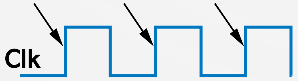
Symbol for rising-edge triggered D flip-flop



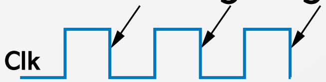
Internal design: Just invert servant clock rather than master

Symbol for falling-edge triggered D flip-flop

rising edges



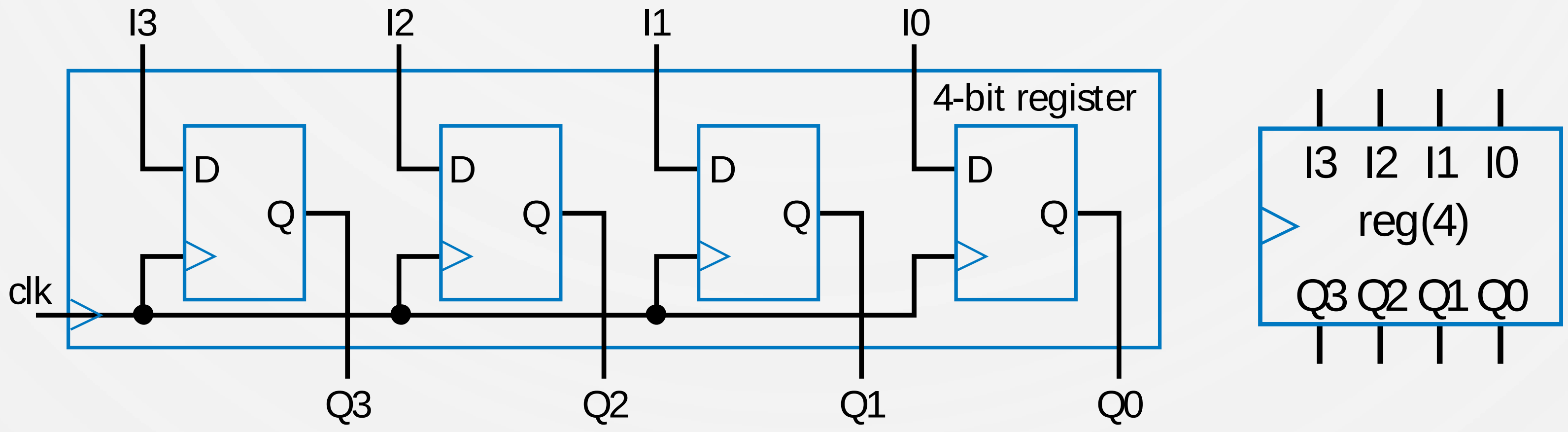
falling edges



- Is it enough to solve our storage problem?

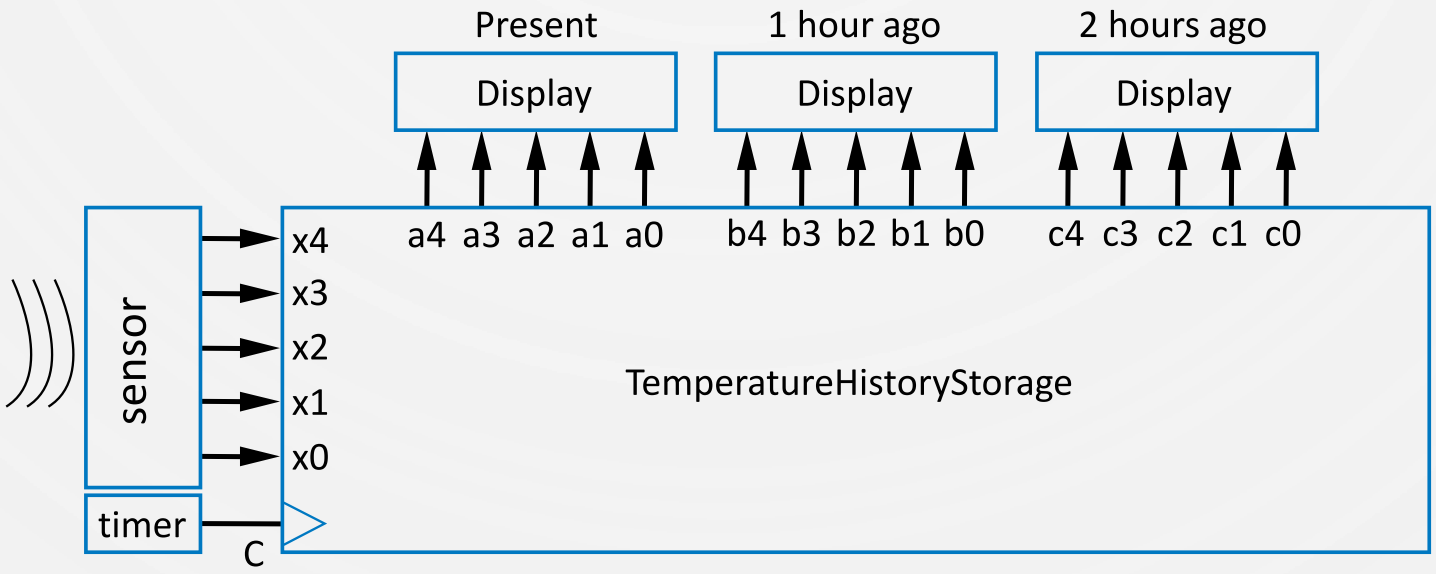
BASIC REGISTER

- Typically, we store multi-bit items
  - e.g., storing a 4-bit binary number
- ***Register***: multiple flip-flops sharing clock signal
  - From this point, we'll use registers for bit storage
    - No need to think of latches or flip-flops
    - But now you know what's inside a register

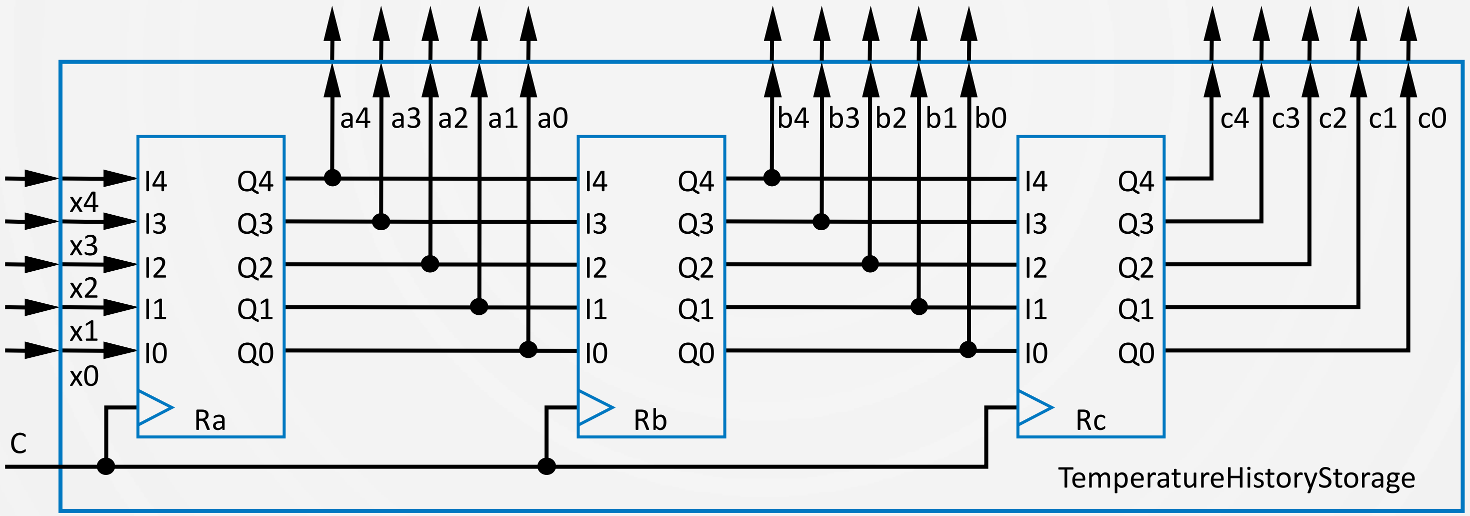


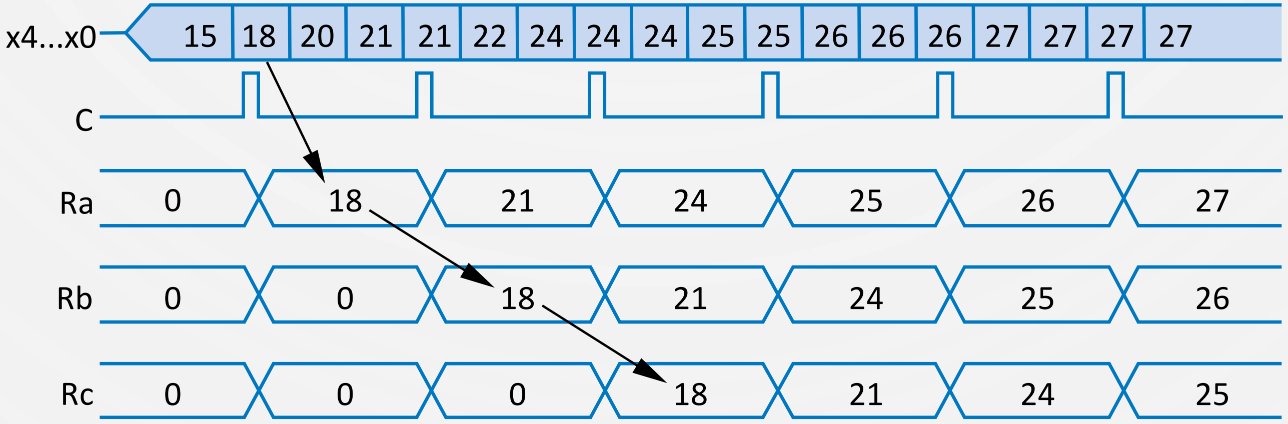
EXAMPLE USING REGISTERS: TEMPERATURE DISPLAY

- Temperature history display
  - Sensor outputs temperature as 5-bit binary number
  - Timer pulses C every hour
  - Record temperature on each pulse, display last three recorded values



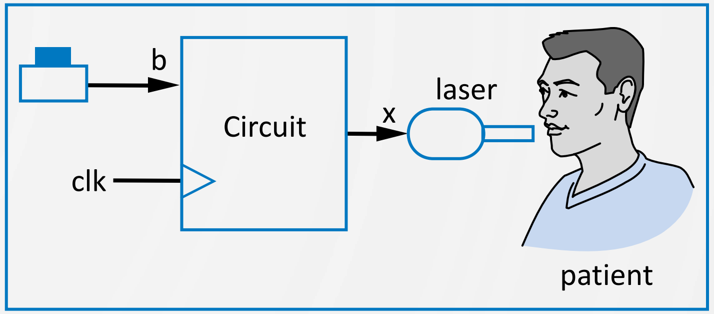
EXAMPLE USING REGISTERS: TEMPERATURE DISPLAY

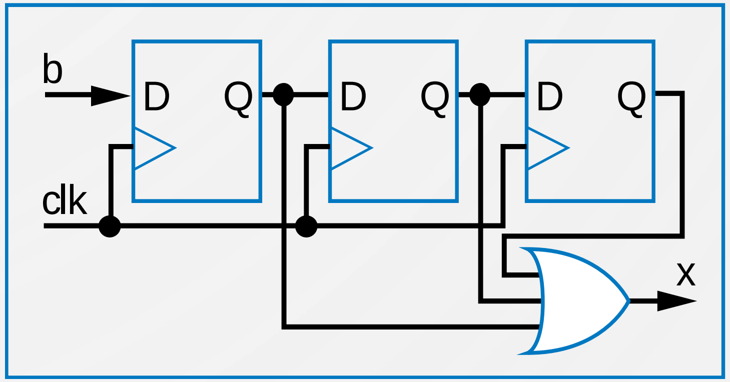




FINITE-STATE MACHINES (FSM)

- Want sequential circuit with particular behavior over time
- **Example: Laser timer**
  - Push button: x=1 for 3 clock cycles
  - How? Let's try three flip-flops
    - b=1 gets stored in first D flip-flop
    - Then 2nd flip-flop on next cycle, then 3rd flip-flop on next
    - OR the three flip-flop outputs, so x should be 1 for three cycles



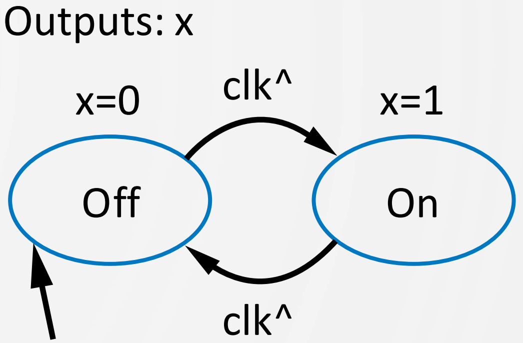


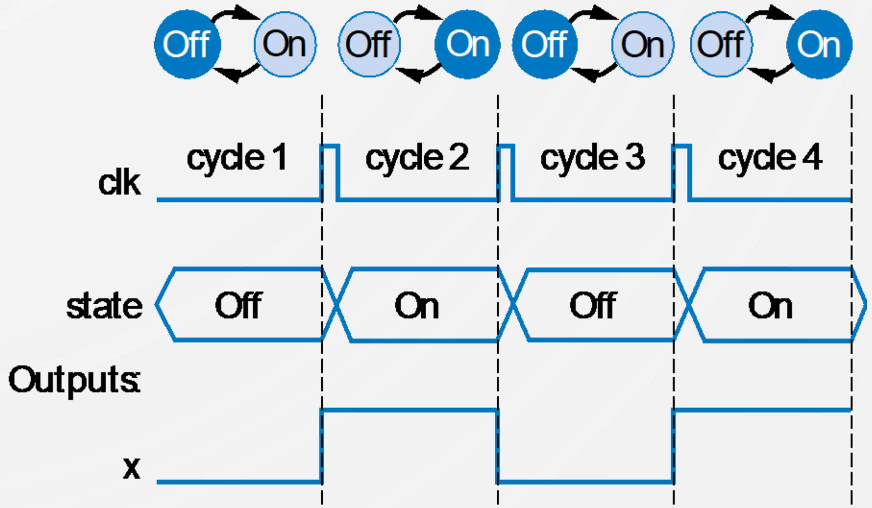
NEED A BETTER WAY TO DESIGN SEQUENTIAL CIRCUITS

- Trial and error is not a good design method
  - Will we be able to "guess" a circuit that works for other desired behavior?
    - How about counting up from 1 to 9? Pulsing an output for 1 cycle every 10 cycles? Detecting the sequence 1 3 5 in binary on a 3-bit input?
  - A circuit built by guessing may have undesired behavior
    - Laser timer: What if press button again while x=1? x then stays one another 3 cycles. Is that what we want?
- Combinational circuit design process had two important things
  1. A formal way to describe desired circuit behavior
     - Boolean equation, or truth table
  2. A well-defined process to convert that behavior to a circuit
- We need those things for sequence circuit design

DESCRIBING BEHAVIOR OF SEQUENTIAL CIRCUIT: FSM

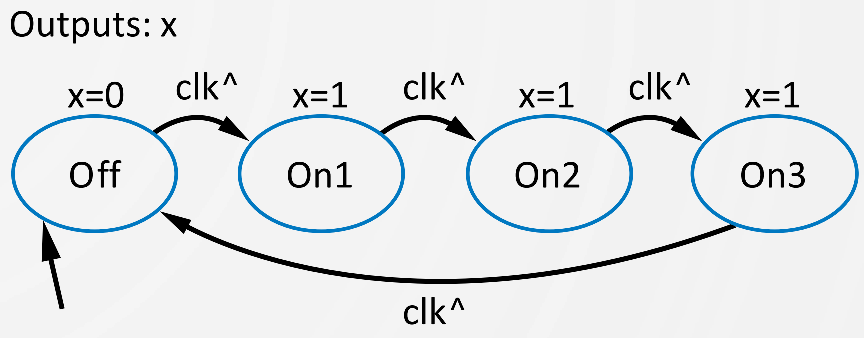
- **Finite-State Machine (FSM)**
  - A way to describe desired behavior of sequential circuit
    - Akin to Boolean equations for combinational behavior
  - List states, and transitions among states
    - Example: Make x change toggle (0 to 1, or 1 to 0) every clock cycle
    - Two states: "Off" (x=0), and "On" (x=1)
    - Transition from Off to On, or On to Off, on rising clock edge
    - Arrow with no starting state points to initial state (when circuit first starts)

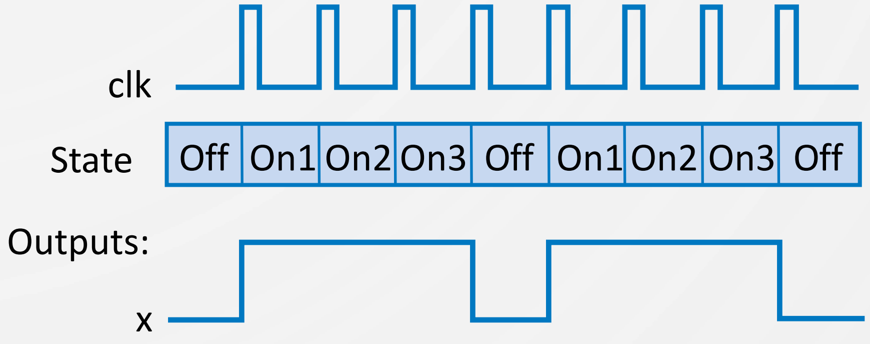




C3.2: FSM EXAMPLES: 0,1,1,1,REPEAT

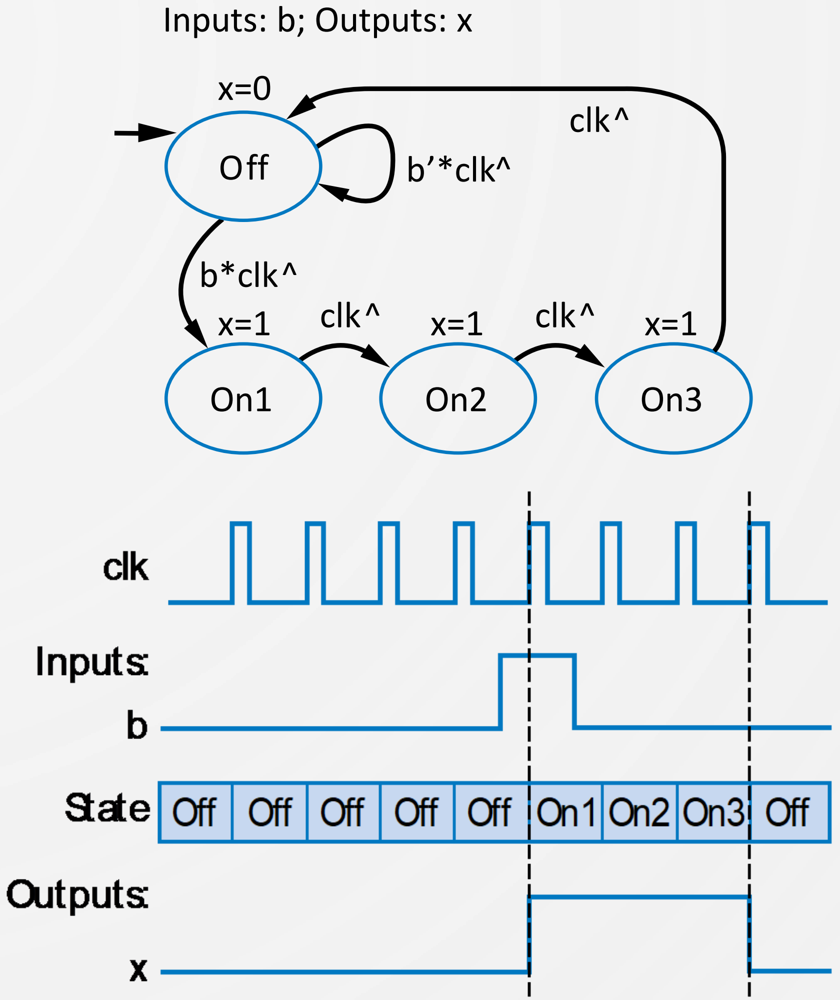
- Want 0, 1, 1, 1, 0, 1, 1, 1, ...
  - Each value for one clock cycle
- Can describe as FSM
  - Four states
  - Transition on rising clock edge to next state





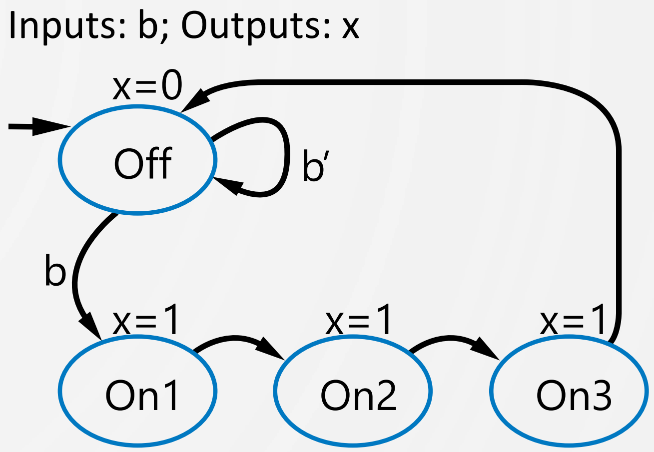
EXTEND FSM TO THREE-CYCLES HIGH LASER TIMER

- Four states
- Wait in “Off” state while b is 0 (b’)
- When b is 1 (and rising clock edge), transition to On1
  - Sets x=1
  - On next two clock edges, transition to On2, then On3, which also set x=1
- So x=1 for three cycles after button pressed



FSM DEFINITION

- FSM consists of
  - Set of states
    - Ex: {Off, On1, On2, On3}
  - Set of inputs, set of outputs
    - Ex: Inputs: {x}, Outputs: {b}
  - Initial state
    - Ex: "Off"
  - Set of transitions
    - Describes next states
    - Ex: Has 5 transitions
  - Set of actions
    - Sets outputs while in states
    - Ex: x=0, x=1, x=1, and x=1



We often draw FSM graphically, known as ***state diagram***

Can also use table (state table), or textual languages

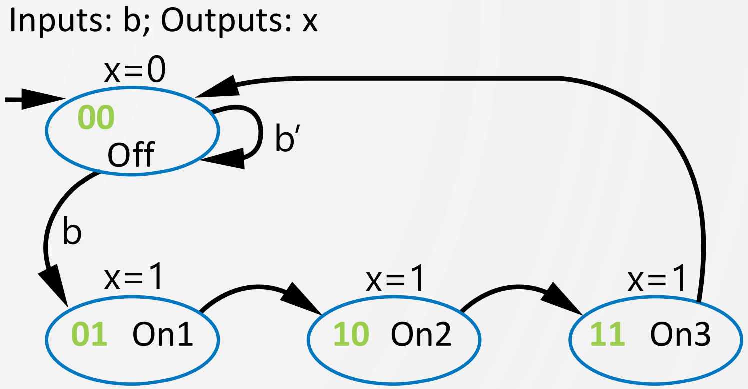
C3.3: FSM DESIGN

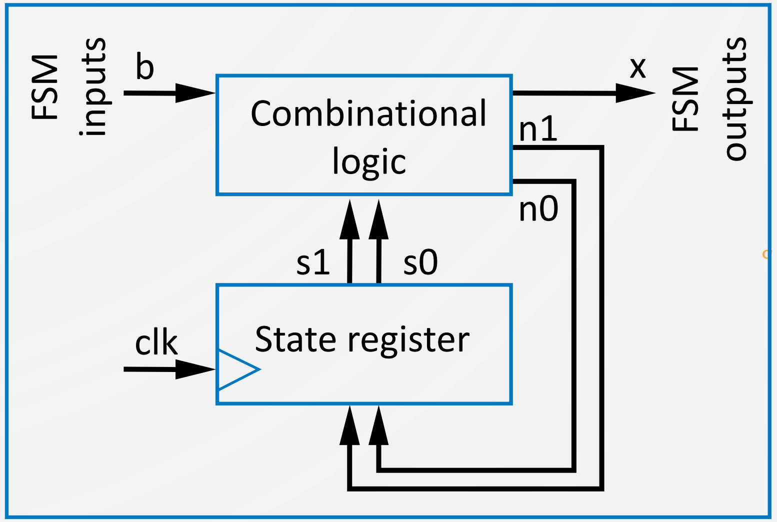
- Five step design process

	Step	Description
Step 1	*Capture the FSM*	Create an FSM that describes the desired behavior of the controller.
Step 2	*Create the architecture*	Create the standard architecture by using a state register of appropriate width, and combinational logic with inputs being the state register bits and the FSM inputs and outputs being the next state bits and the FSM outputs.
Step 3	*Encode the states*	Assign a unique binary number to each state. Each binary number representing a state is known as an ***encoding.*** Any encoding will do as long as each state has a unique encoding.
Step 4	*Create the state table*	Create a truth table for the combinational logic such that the logic will generate the correct FSM outputs and next state signals. Ordering the inputs with state bits first makes this truth table describe the state behavior, so the table is a state table.
Step 5	*Implement the combinational logic*	Implement the combinational logic using any method.

DESIGN EXAMPLE : LASER TIMER EXAMPLE

- Step 1: Capture the FSM
  - Already done
- Step 2: Create architecture
  - 2-bit state register (for 4 states)
  - Input b, output x
  - Next state signals n1, n0
- Step 3: Encode the states
  - Any encoding with each state unique will work

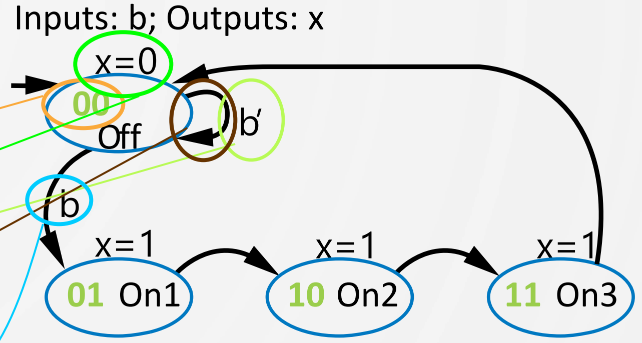


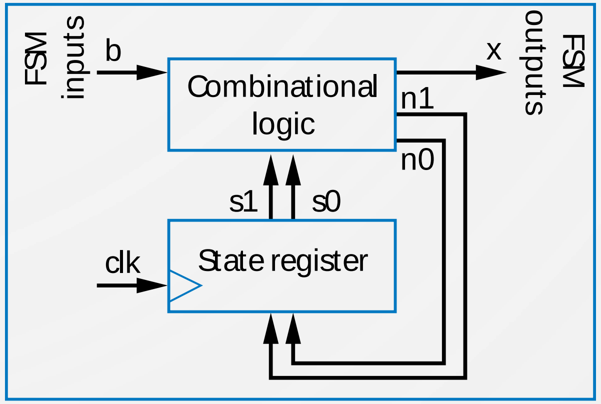


DESIGN EXAMPLE: LASER TIMER EXAMPLE (CONT)

- Step 4: Create state table

	Inputs			Outputs		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0

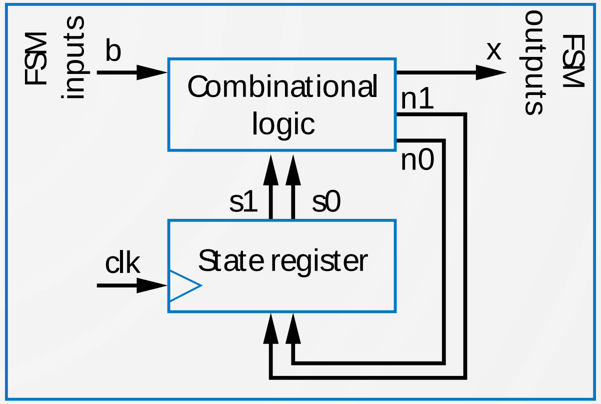




DESIGN EXAMPLE : LASER TIMER EXAMPLE (CONT)

- Step 5: Implement combinational logic

	Inputs			Outputs		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0



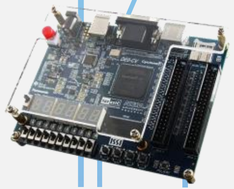
$x = s1 + s0$ (note from the table that x=1 if s1 = 1 or s0 = 1)

$n1 = s1's0b' + s1's0b + s1s0'b' + s1s0'b$

$n1 = s1's0 + s1s0'$

$n0 = s1's0'b + s1s0'b' + s1s0'b$

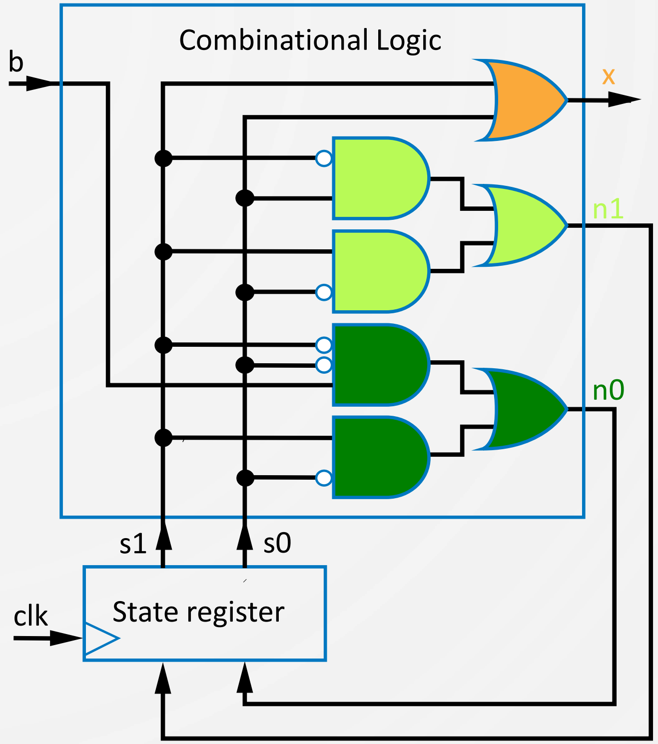
$n0 = s1's0'b + s1s0'$



DESIGN EXAMPLE : LASER TIMER EXAMPLE (CONT)

- Step 5: Implement combinational logic (cont)

	Inputs			Outputs		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0



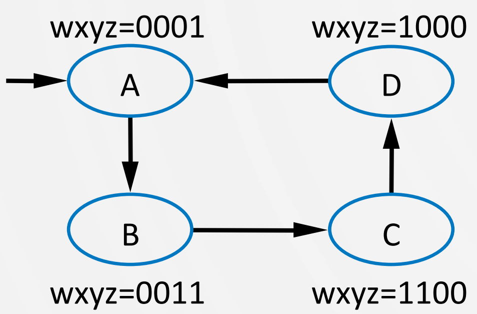
$x = s1 + s0$

$n1 = s1's0 + s1s0'$

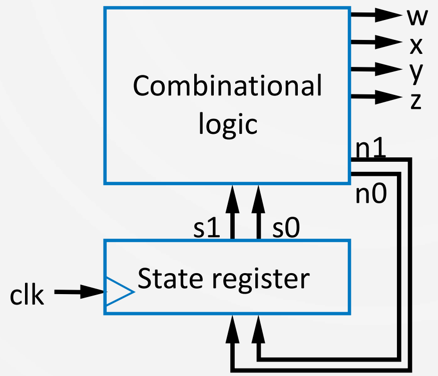
$n0 = s1's0'b + s1s0'$

DESIGN EXAMPLE: SEQUENCE GENERATOR

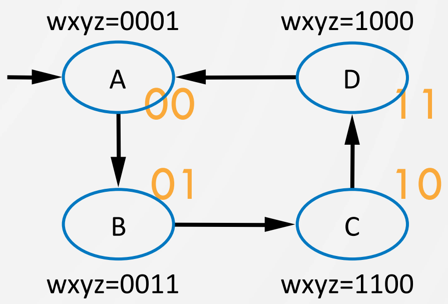
- Want generate sequence 0001, 0011, 1100, 1000, (repeat)
  - Each value for one clock cycle. e.g., to control magnets of a “stepper motor”



Step 1: Create FSM



Step 2: Create architecture

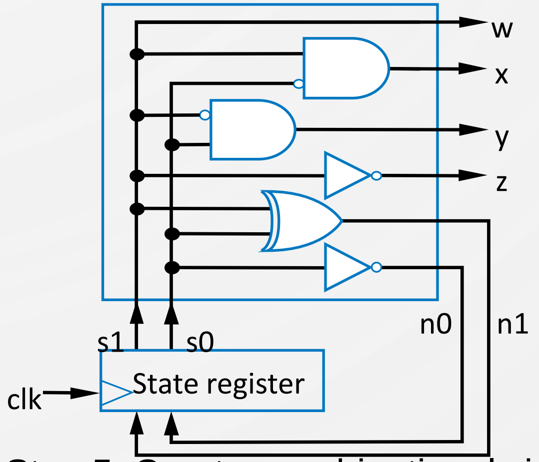


Step 3: Encode states

	Inputs		Outputs					
	s1	s0	w	x	y	z	n1	n0
A	0	0	0	0	0	1	0	1
B	0	1	0	0	1	1	1	0
C	1	0	1	1	0	0	1	1
D	1	1	1	0	0	0	0	0

Step 4: Create state table

w = s1
x = s1s0’
y = s1’s0
z = s1’
n1 = s1 xor s0
n0 = s0’



Step 5: Create combinational circuit