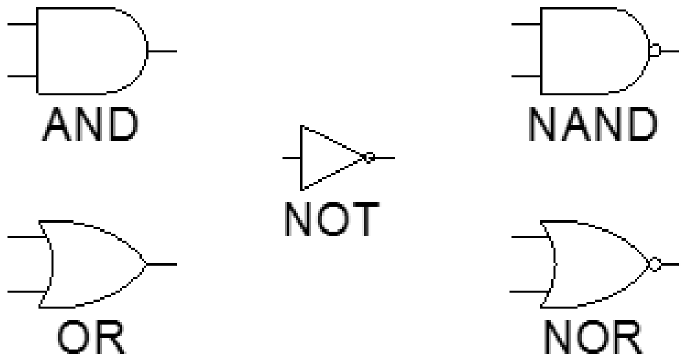
FROM LAST TIME…

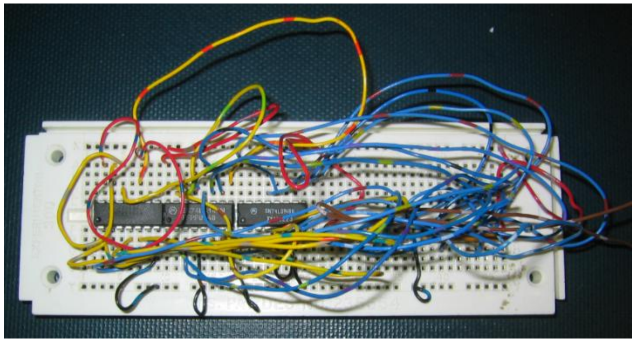
Computer Systems (Combinational Logic)

- Why do computer systems matter?
- Boolean algebra
- Boolean simplification
- Implementing Boolean logic



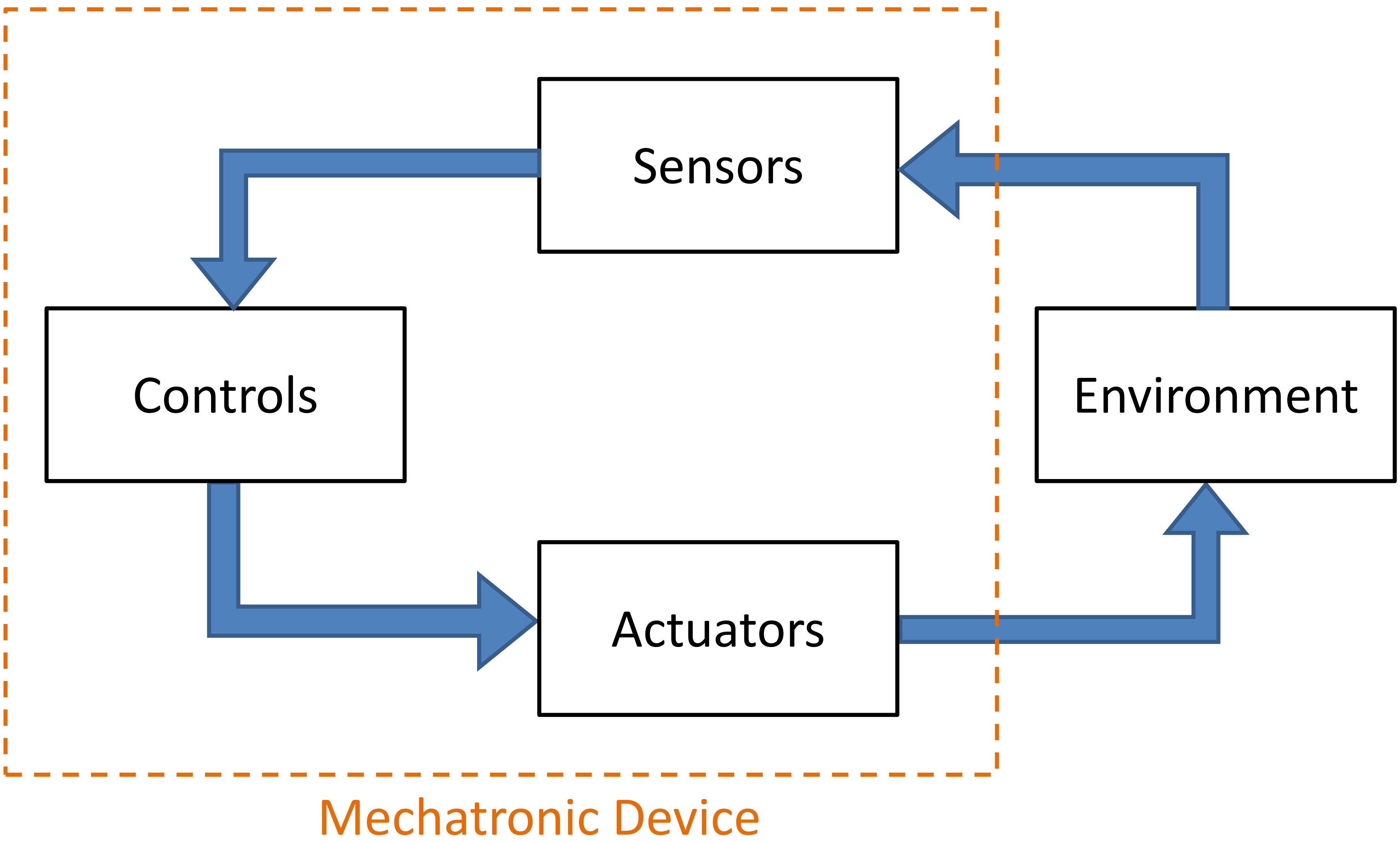
$$y = (B1 \cdot B2) + (B1 \cdot \overline{B2}) + (\overline{B1} \cdot B2) + (\overline{B1} \cdot \overline{B2})$$

$$y = (B1 + B2) \cdot (B1 + \overline{B2}) \cdot (\overline{B1} + B2) \cdot (\overline{B1} + \overline{B2})$$

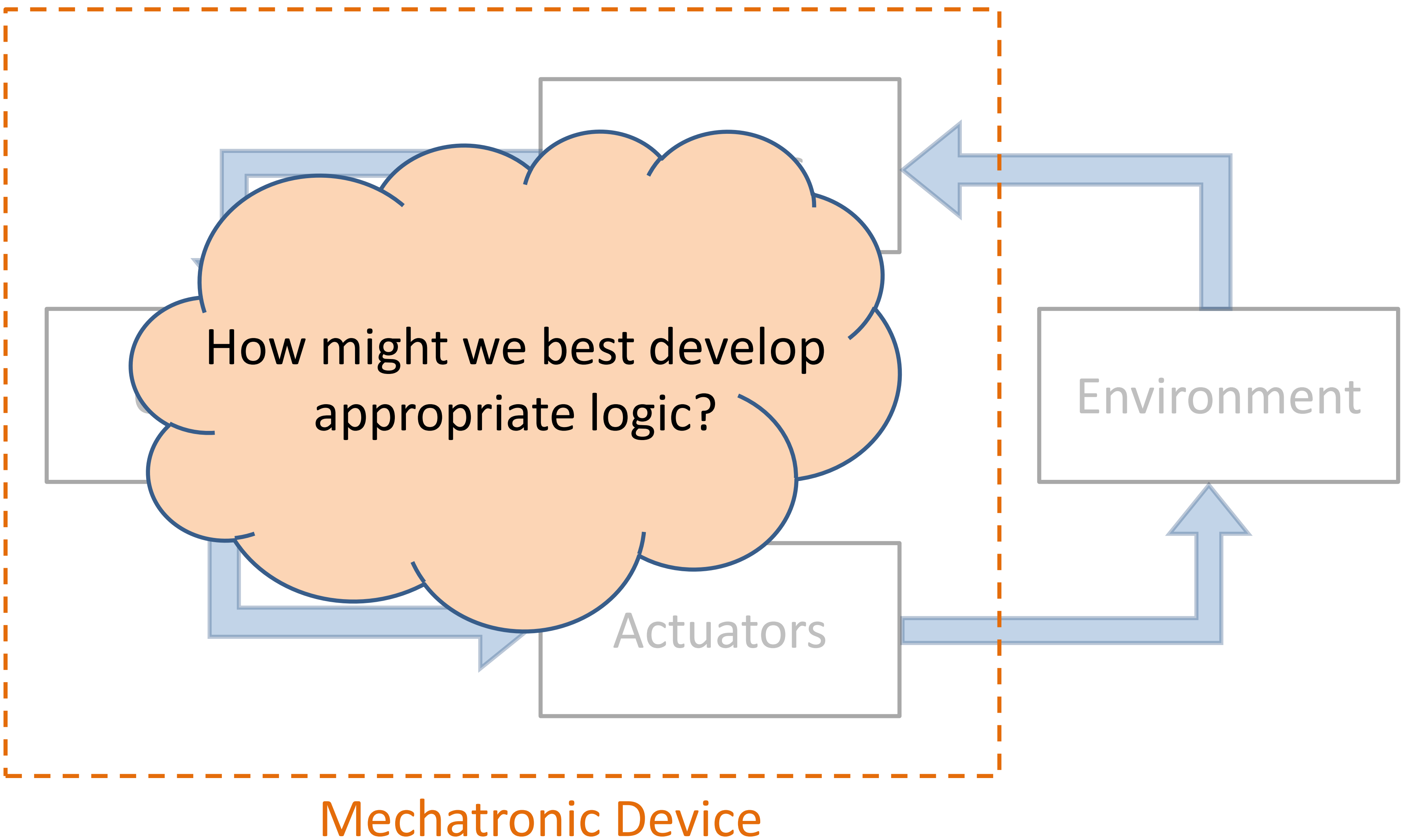


UNIT 4:
SEQUENTIAL LOGIC

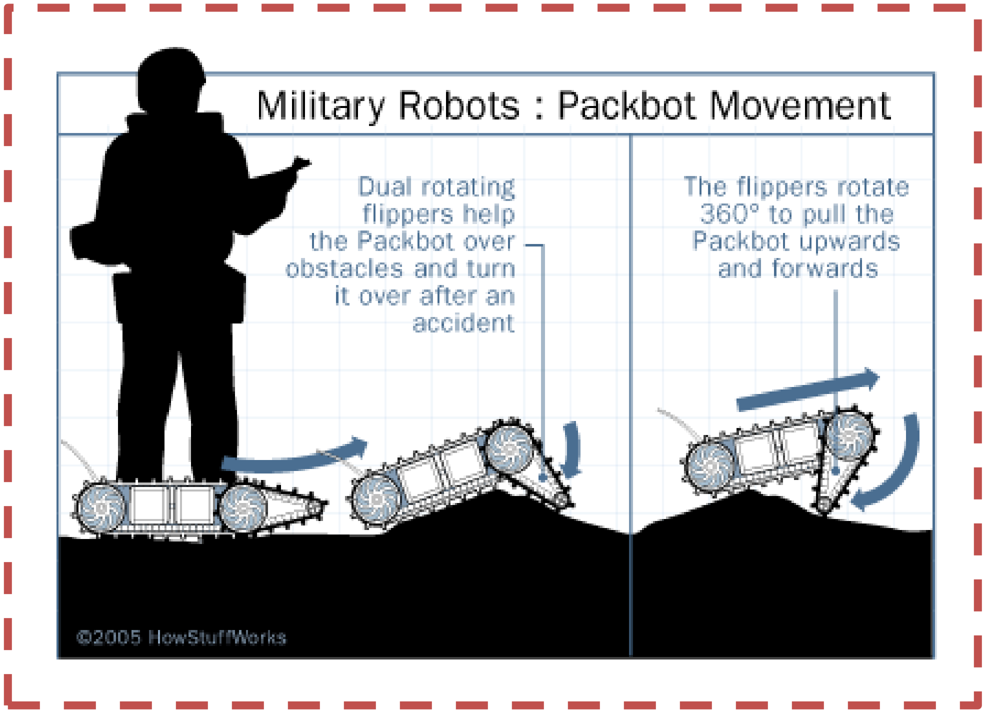
A MECHATRONIC DEVICE INTERACTS WITH ITS ENVIRONMENT



A MECHATRONIC DEVICE INTERACTS WITH ITS ENVIRONMENT



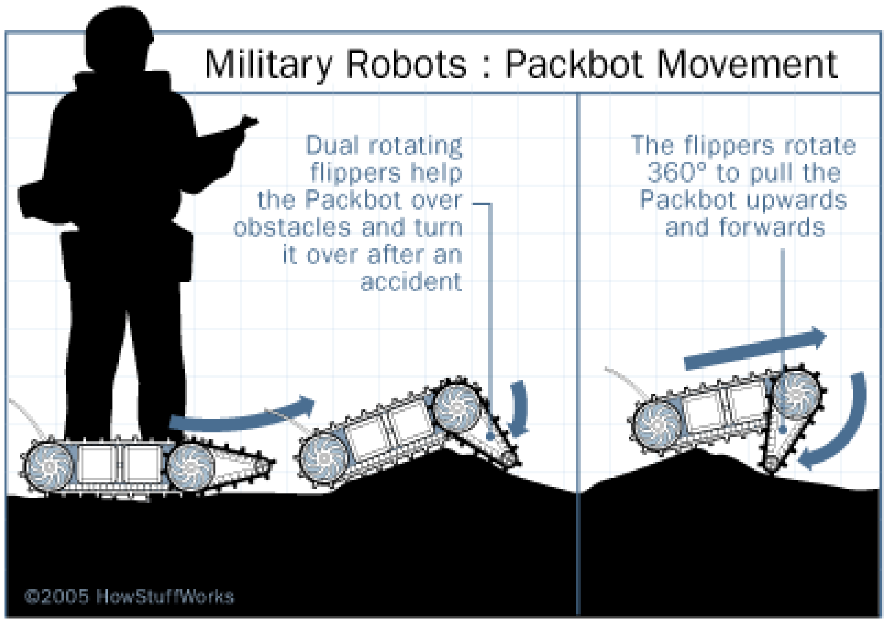
CONTROLLER DESIGN DEPENDS ON SYSTEM COMPLEXITY



Component-level control:
Classical or modern control theory

Supervisory control:
SCADA, state machine, etc.

A FINITE STATE MACHINE ALLOWS FOR SUPERVISORY CONTROL




```mermaid
flowchart LR
    A[Drive forward] --> B[Flippers rotate] --> C[Drive forward with flippers]
```

PROGRAMMING MAY BE IMPERATIVE OR DECLARATIVE

Imperative (Procedural):

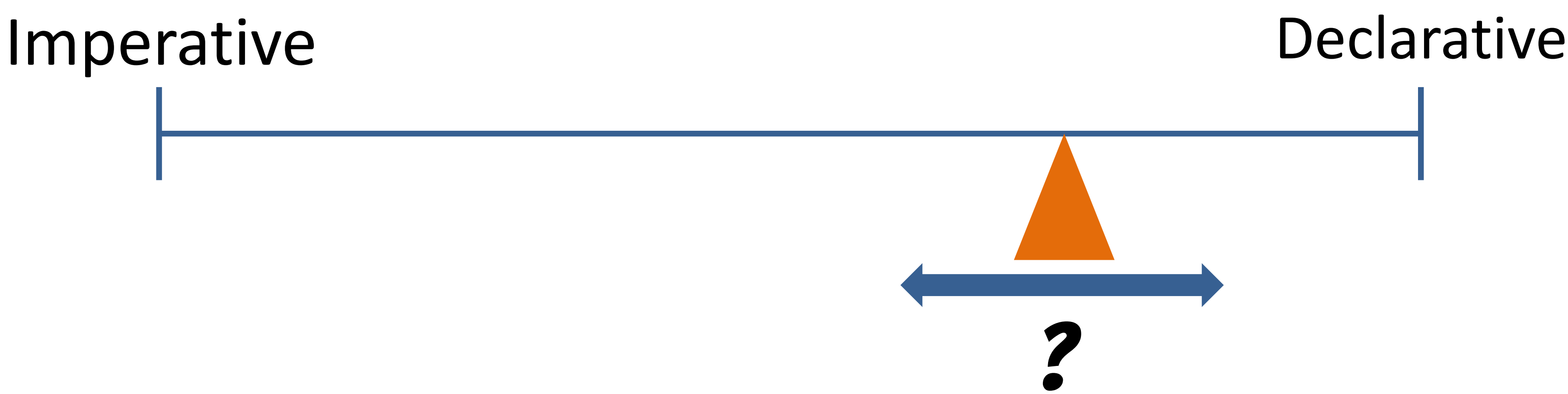
- Describe "how" something gets done
- Express control flow without describing logic
- Specify order in which statements are evaluated

Declarative:

- Describe "what" needs to be done
- Express logic without describing control flow
- Ignore order in which things happen

PROGRAMMING MAY BE IMPERATIVE OR DECLARATIVE

Real-world solutions require a mixture of imperative (procedural) and declarative (supervisory) logic. Our goal is finding an effective balance.



PROGRAMMING MAY BE IMPERATIVE OR DECLARATIVE

Microprocessors are procedural as they follow machine code algorithms that have been set forth in a particular order.

Most programming languages are imperative (procedural):

- C/C++
- Python
- Ruby
- Perl

However, some languages are declarative:

- SQL
- Markup languages (HTML, XSLT, etc.)

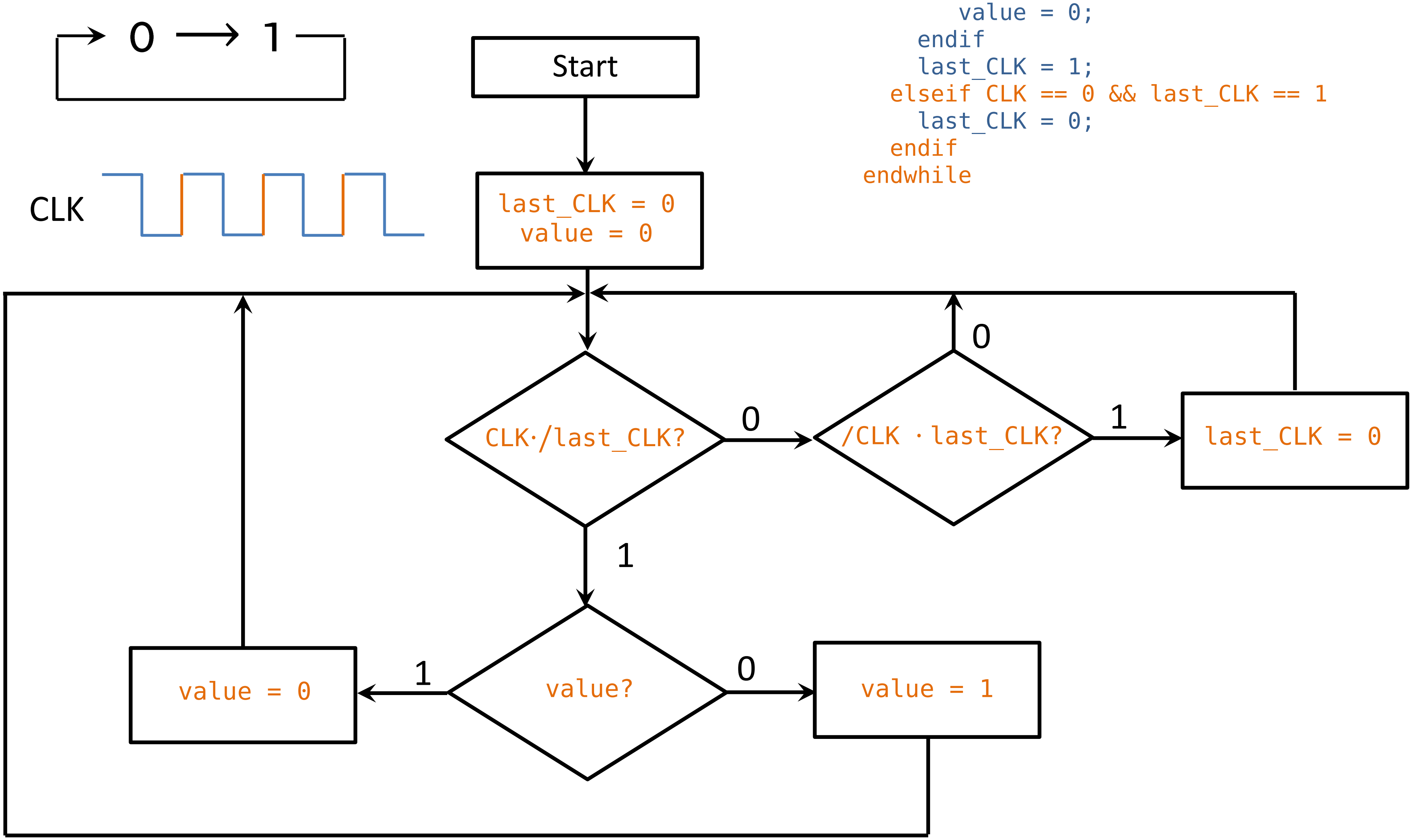
PROGRAMMING MAY BE IMPERATIVE OR DECLARATIVE

Imperative (procedural) ***models*** are closer to hardware implementation, but can be brittle, with small changes having unintended consequences.

Declarative ***models*** tend to be more robust, and easier for domain-experts to interpret, but can overlook important implementation details.

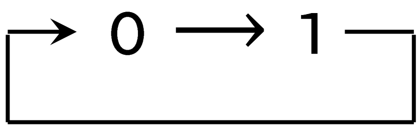
1-BIT UP COUNTER

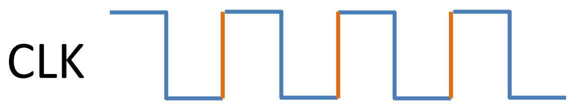
```
last_CLK = 0; value = 0;

while 1    % put into infinite loop
  input CLK;
  if CLK == 1 && last_CLK == 0
    if value = 0
           value = 1;
    else
       value = 0;
    endif
    last_CLK = 1;
  elseif CLK == 0 && last_CLK == 1
    last_CLK = 0;
  endif
endwhile
```

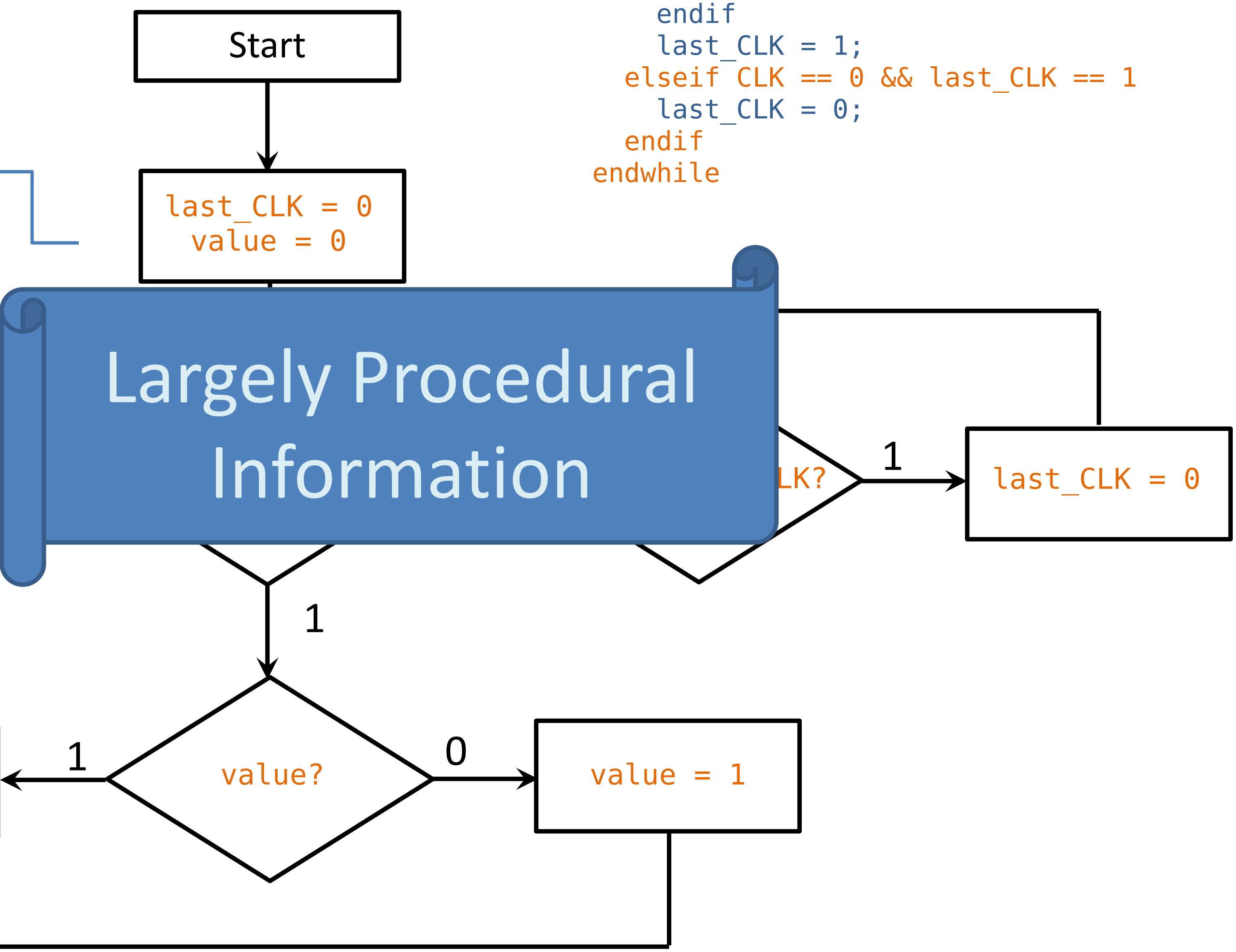


1-BIT UP COUNTER

```
last_CLK = 0; value = 0;

while 1    % put into infinite loop
  input CLK;
  if CLK == 1 && last_CLK == 0
    if value = 0
       value = 1;
    else
      value = 0;
    endif
    last_CLK = 1;
  elseif CLK == 0 && last_CLK == 1
    last_CLK = 0;
  endif
endwhile
```



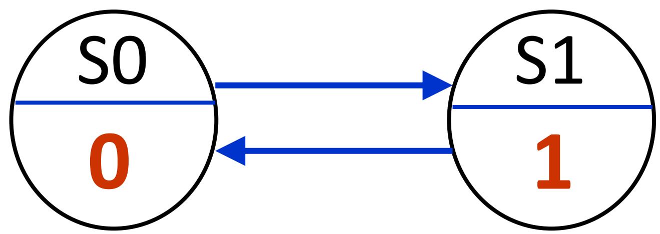




1-BIT UP COUNTER

State Transition Diagram:

This model tells us nothing about when transitions should occur, nor how the transition is to be accomplished.



Largely Declarative Information

DECLARATIVE PROGRAMMING IS OFTEN EFFECTIVE

It is frequently the case with Mechatronics that:

- desired outcomes are known in advance
- device behavior does not vary with time
- transitions are accomplished by activating actuators, rather than writing code

Thus, a declarative notation is *often* (but not always) more effective for designing and debugging a control system.

A FINITE STATE MACHINE (FSM) PROVIDES A DECLARATIVE MODEL

- Has a finite number of states
- Can only be in one state at a time
- Changes states via transitions

A FINITE STATE MACHINE (FSM) PROVIDES A DECLARATIVE MODEL

Before we can control the state, we must retain state knowledge. Therefore, we need to understand data storage elements…

UNIT 4:
SEQUENTIAL LOGIC

PART A:
DATA STORAGE ELEMENTS

SINGLE-BIT MEMORY DEVICES COME IN TWO FORMS

- Latches
- Flip-Flops

Four possible functions for acting on a single bit:

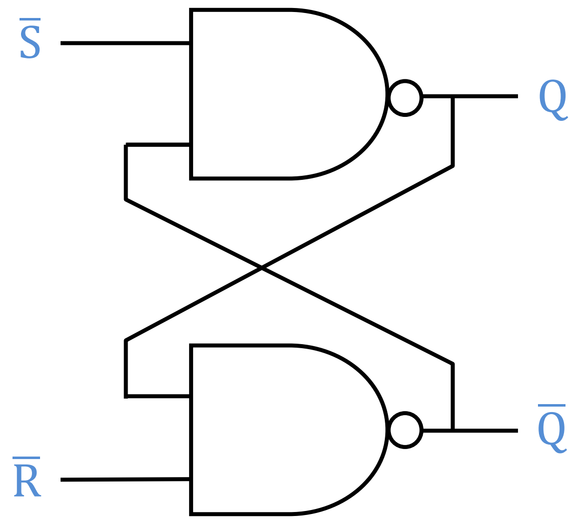
- Set
- Reset (clear)
- Toggle
- No operation

THERE ARE FOUR TYPES OF SINGLE-BIT MEMORY DEVICES

Four designations refer to manner in which inputs are varied to accomplish bit manipulation:

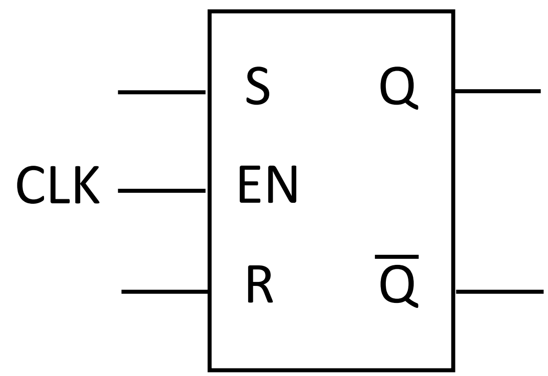
- **S-R** (Set/Reset)
- **D** (Data or Delay)
- **T** (Toggle)
- **J-K** (S-R + Toggle)

A LATCH CHANGES OUTPUT ON INPUT CHANGE



- Input-output change will include appropriate propagation delay
- Constructed from logic gates
- Always asynchronous

A FLIP-FLOP IS A LATCH MODIFIED TO REQUIRE A CLOCK SIGNAL



- Constructed from a latch
- Output changes with clock level (or clock transition)
- Naming convention (latch vs. flip-flop) sometimes loosely interpreted, look for functional description

FLIP-FLOP TRIGGERING DEPENDS ON A CLOCK SIGNAL

Level-triggered

- Specific clock level (voltage)
- Semi-asynchronous

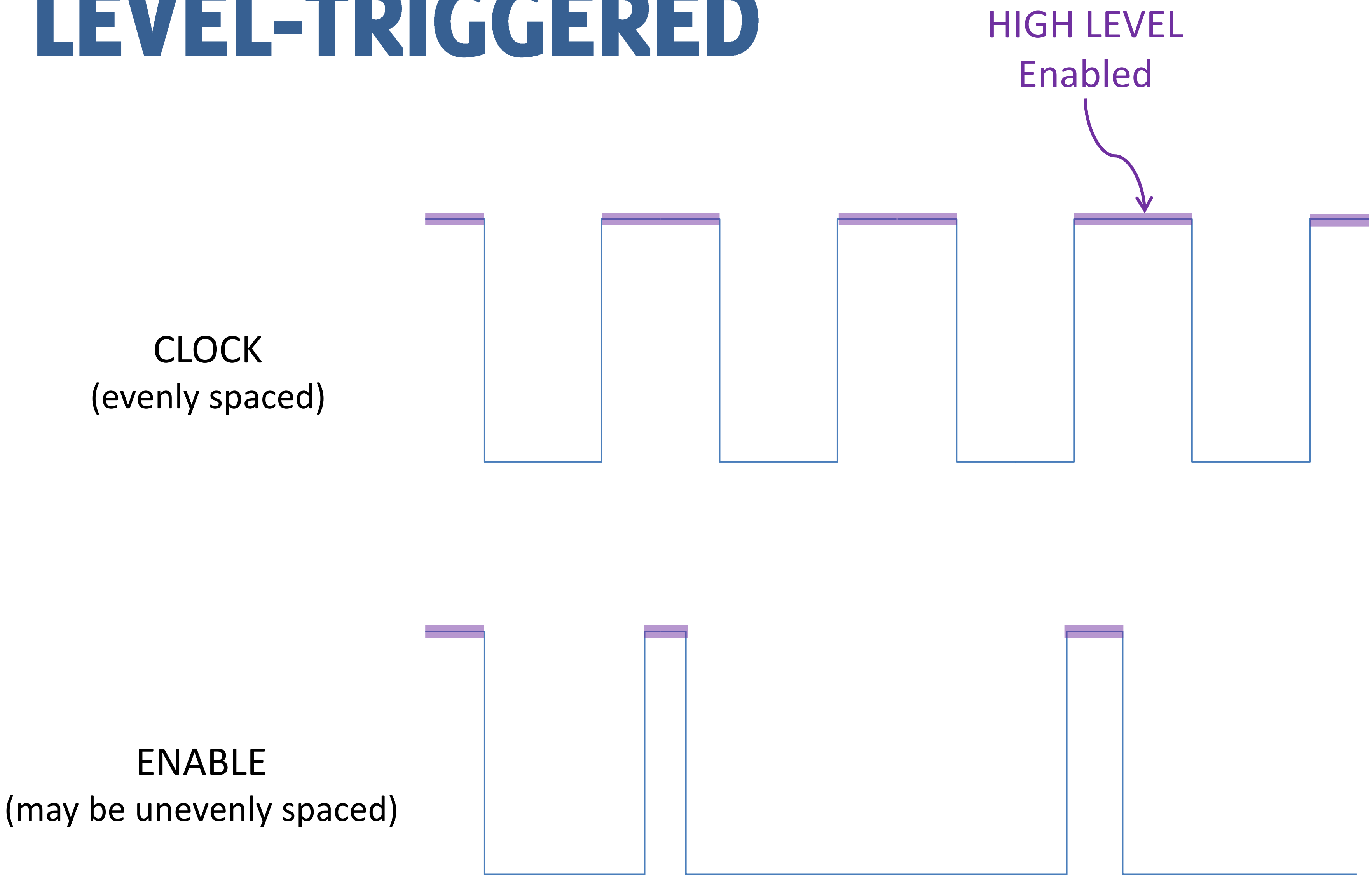
Pulse-triggered

- Two transitions, in opposite directions
- Synchronous

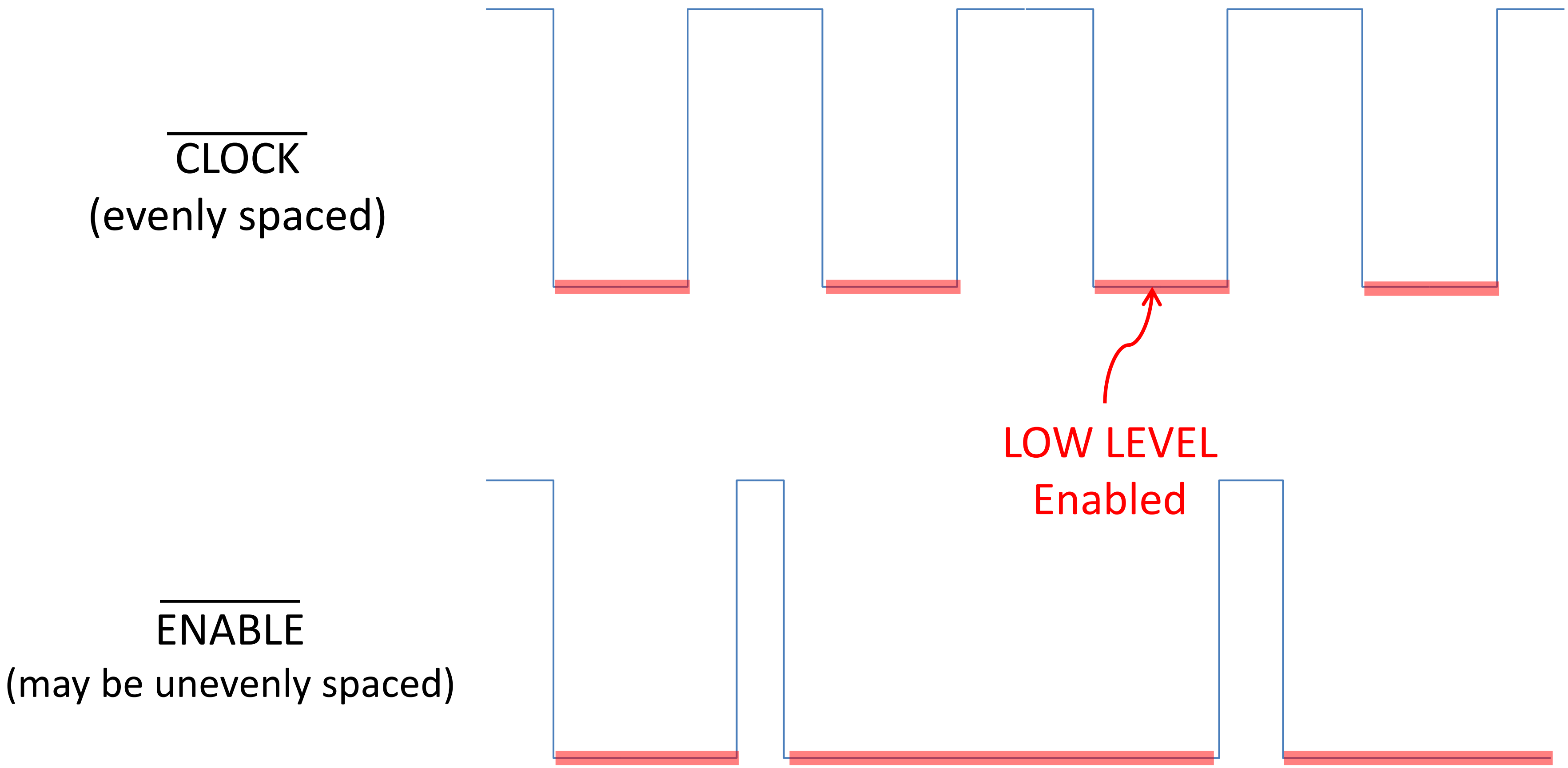
Edge-triggered

- positive or negative transition
- synchronous

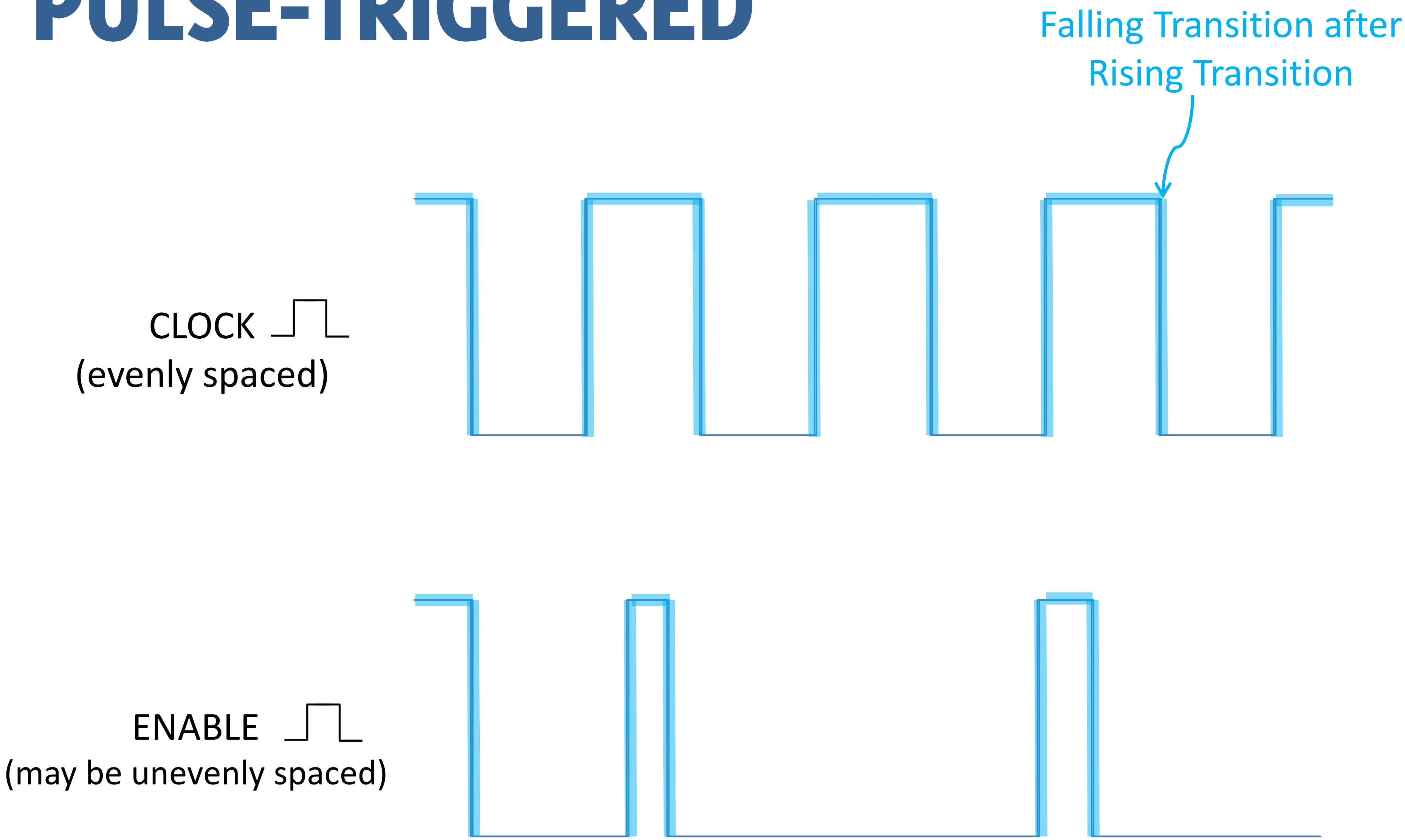
LEVEL-TRIGGERED



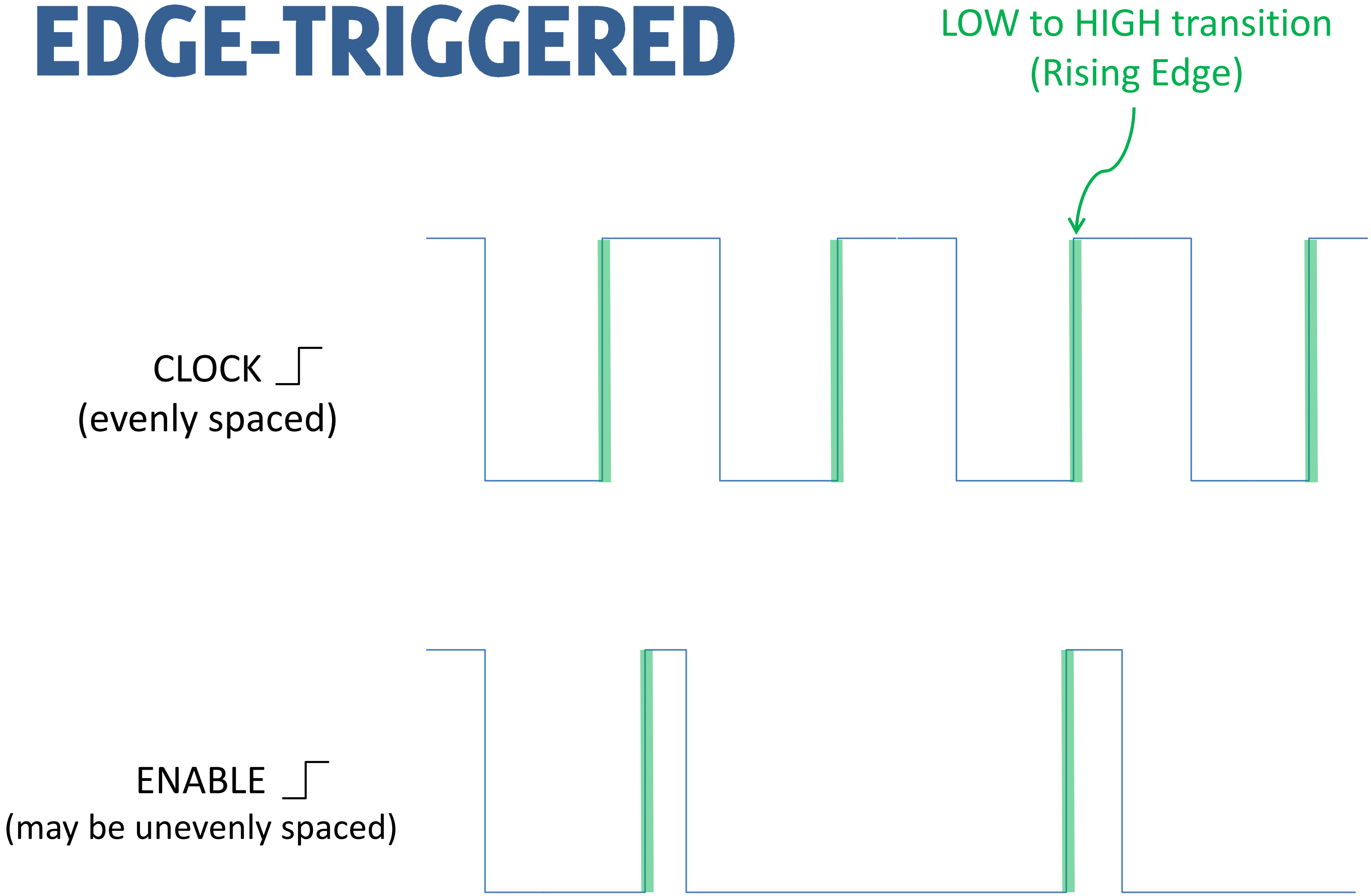
LEVEL-TRIGGERED



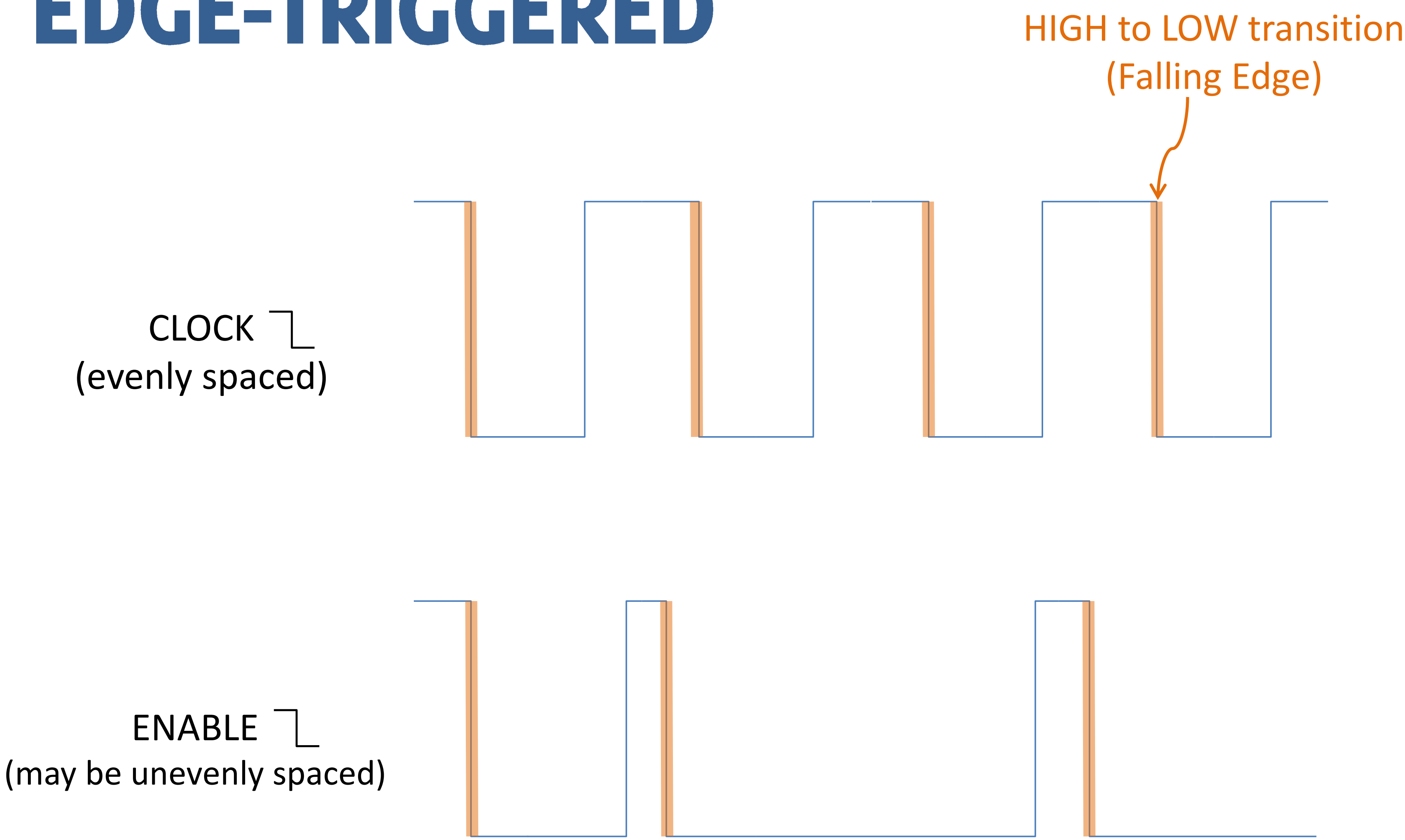
PULSE-TRIGGERED



EDGE-TRIGGERED



EDGE-TRIGGERED

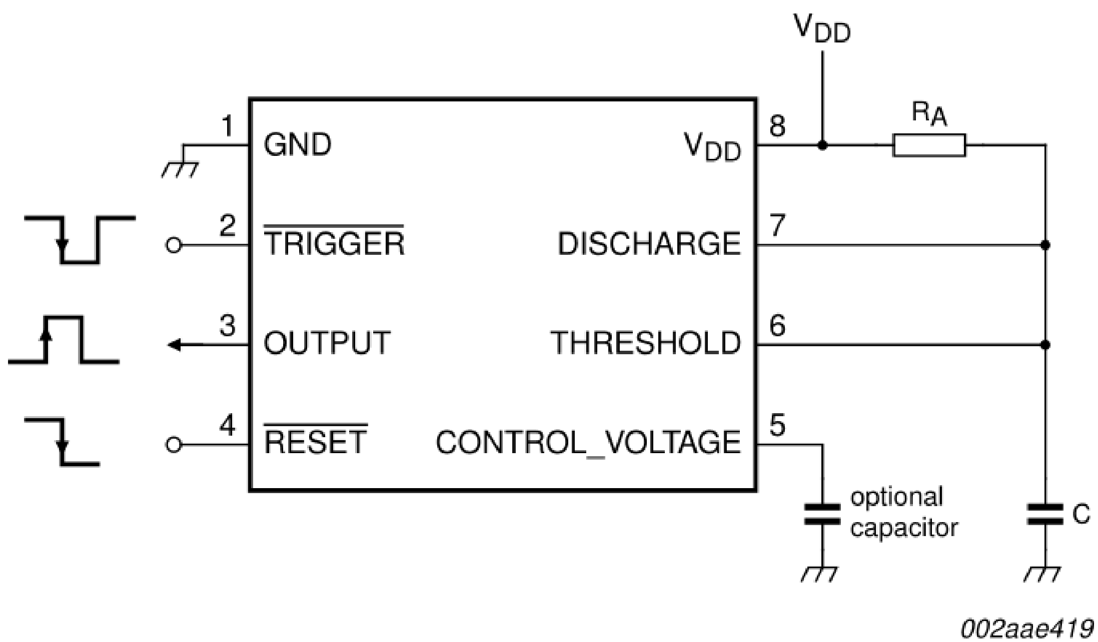


TRIGGERING NOTATION

NXP Semiconductors

ICM7555

General purpose CMOS timer



$V_{DD} \leq 18$ V; $t = 1.05\ R_AC$

Fig 16. Monostable operation

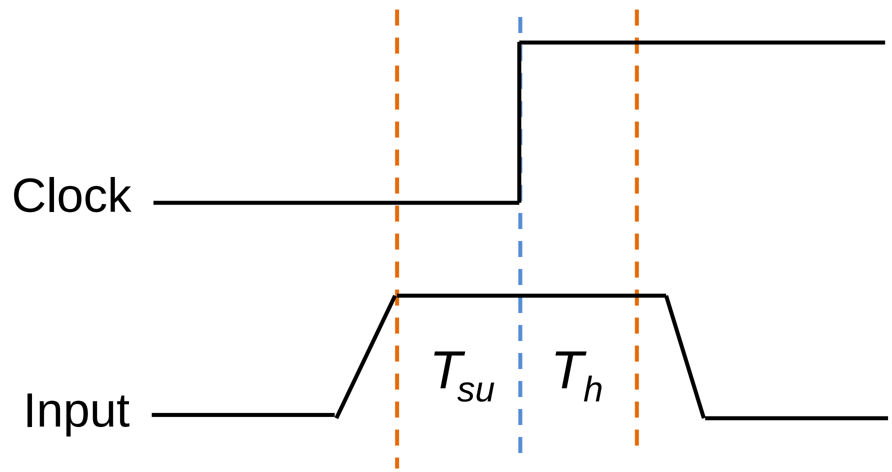
MEMORY DEVICE RESPONSE IS NOT INSTANTANEOUS

Setup Time (T_{su}):

Minimum time that input must remain stable ***before*** clock transition

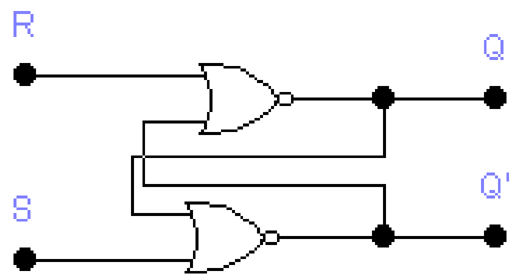
Hold Time (T_h)**:**

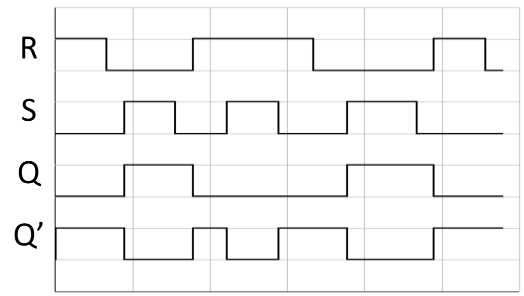
Minimum time that input must remain stable ***after*** clock transition



S-R (SET/RESET) LATCH

Cross-coupled NOR gates





R	S	Q	Q'
0	0	**Q**	**Q'**
0	1	**1**	**0**
1	0	**0**	**1**
1	1	**not allowed**	

- The S-R latch can hold its current state
- By asserting R (reset), the state can be set to 0 and by asserting S (set) the state is set to 1
- S=R=1 is forbidden because it leads to oscillation, and an uncertain output dependent on propagation delays

WRITE A NEW TRUTH TABLE TO INCLUDE NEXT STATE Q*

R	S	Q	Q'
0	0	**Q**	**Q'**
0	1	**1**	**0**
1	0	**0**	**1**
1	1	**not allowed**	

next state

R	S	Q	Q^*
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**0**
1	1	0	**X**
1	1	1	**X**

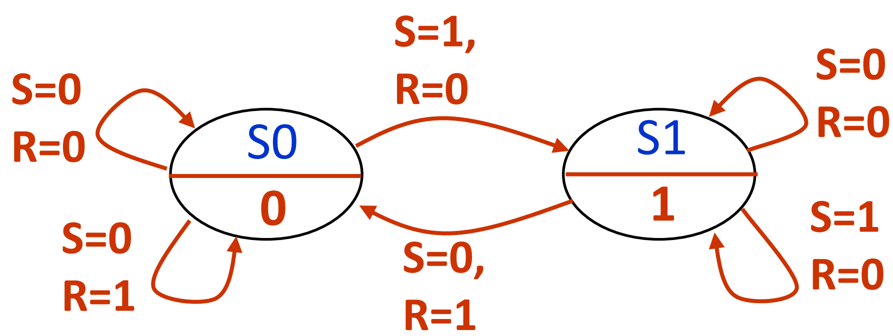
The new truth table represents a **sequential** logic device (circuit)

- Next state Q* is a function of inputs S and R ***and*** current state Q
- No additional output is defined for this device

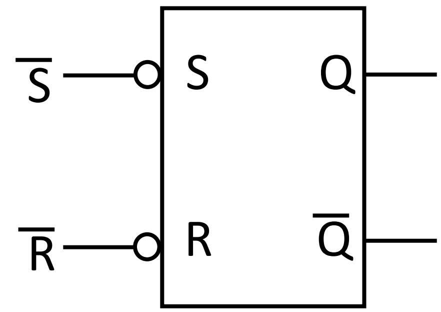
NEXT STATE EXPRESSED AS K-MAP OR STATE TRANSITION DIAGRAM

$$Q^* = S + Q \cdot \bar{R}$$

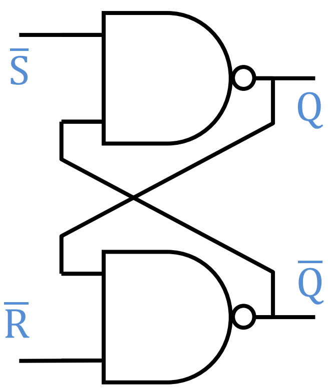
S-R Latch		*R S* 00	01	11	10
Q	0	0	1	X	0
	1	1	1	X	0

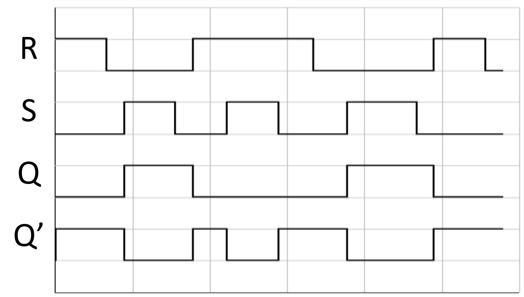


$\overline{S}$-$\overline{R}$ LATCH

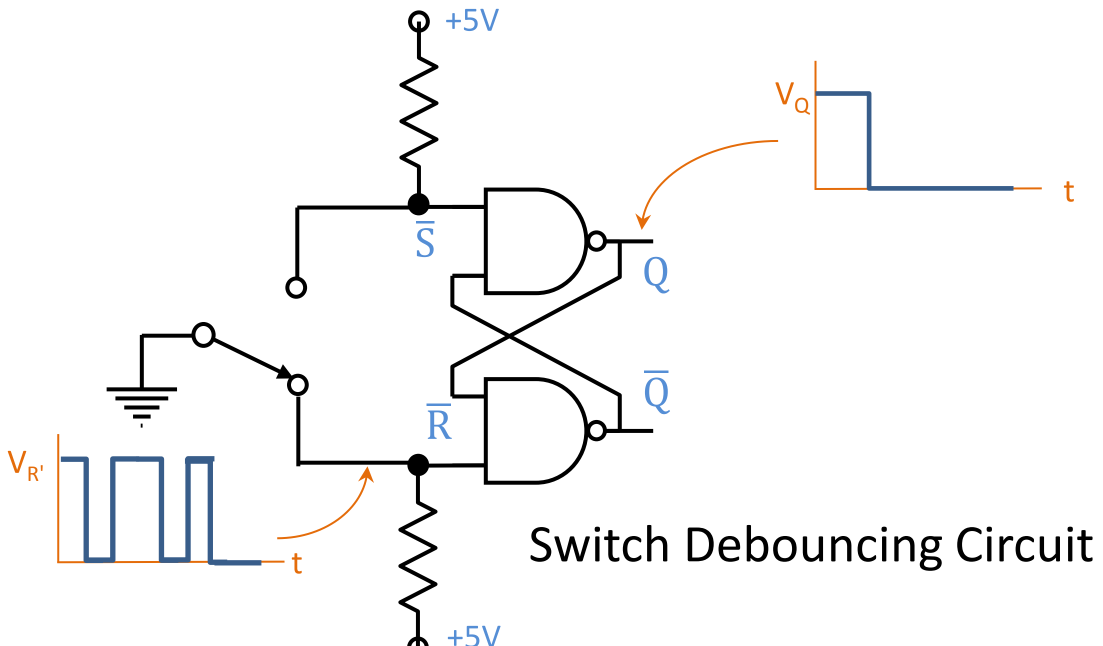


Cross-coupled NAND gates



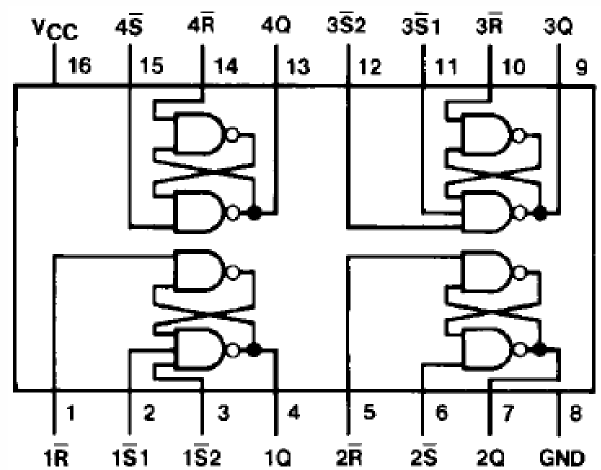


R	S	/R	/S	Q	Q’
0	0	1	1	**Q**	**Q’**
0	1	1	0	**1**	**0**
1	0	0	1	**0**	**1**
1	1	0	0	**not allowed**	



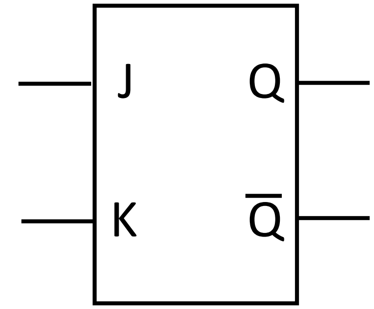
Switch Debouncing Circuit

Connection Diagram

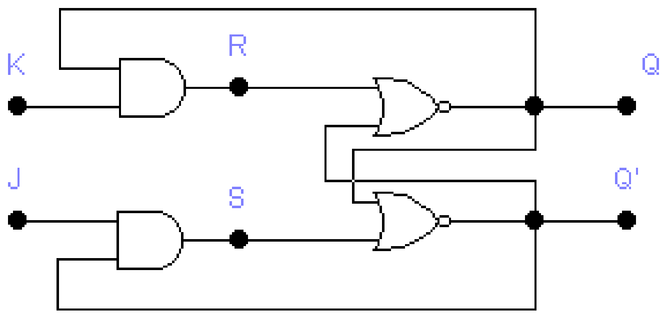


74279 Quad $\overline{S}$-$\overline{R}$ Latch

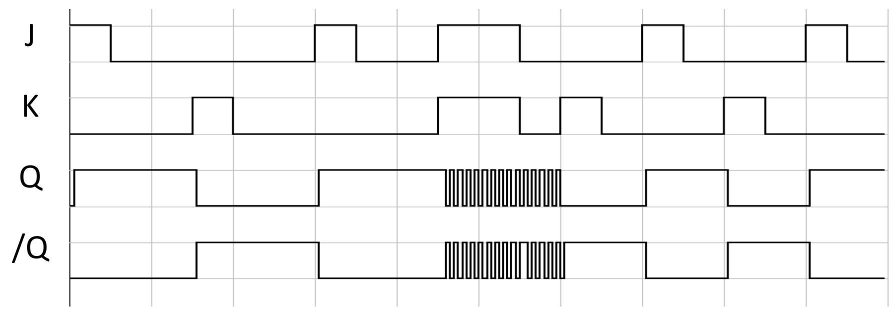
J-K LATCH



Output sets (Q=1) when J = 1 and resets (Q = 0) when K = 1. Output toggles when J = K = 1.



J	K	Q	Q^*	
0	0	0	0	**Q**
0	0	1	1	
0	1	0	0	**0**
0	1	1	0	
1	0	0	1	**1**
1	0	1	1	
1	1	0	1	**toggle**
1	1	1	0	

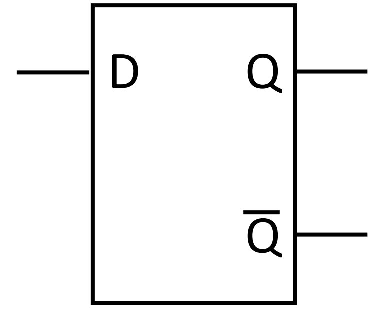


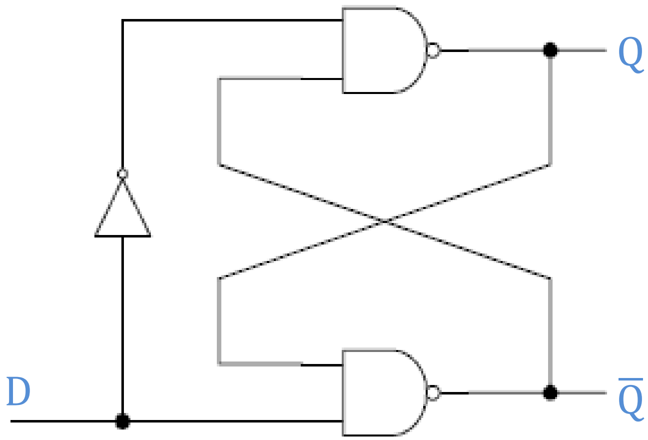
J-K Latch

Q \ JK	00	01	11	10
0	**0**	**0**	**1**	**1**
1	**1**	**0**	**0**	**1**

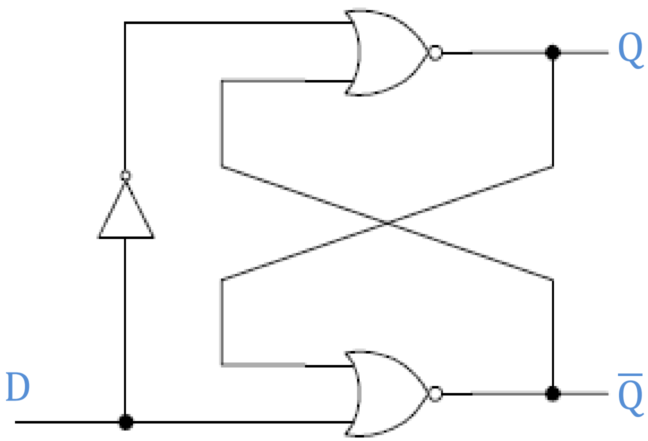
$$Q^* = \overline{Q} \cdot J + Q \cdot \overline{K}$$

D (DATA) LATCH





NAND Gate Implementation

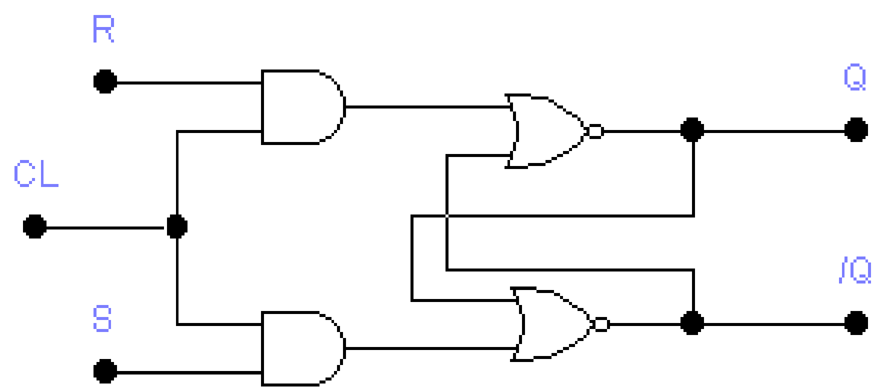


NOR Gate Implementation

D	*Q*	*Q**	*/Q**
0	X	**0**	**1**
1	X	**1**	**0**

Restricted input not possible

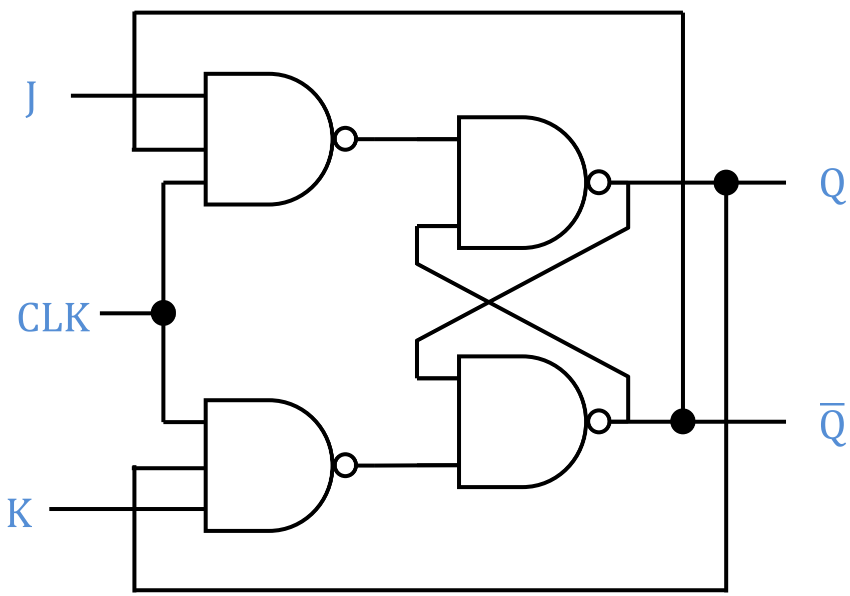
LEVEL-TRIGGERED S-R FLIP-FLOP IS ENABLED BY CLOCK VALUE



- Behaves like an S-R latch when CL is asserted, i.e. when CL = 1. Retains its previous state when CL = 0.
- Change in flip-flop state is triggered by CL signal level. This is sometime referred to as "level triggering."
- Circuit may also be called a "gated S-R latch."

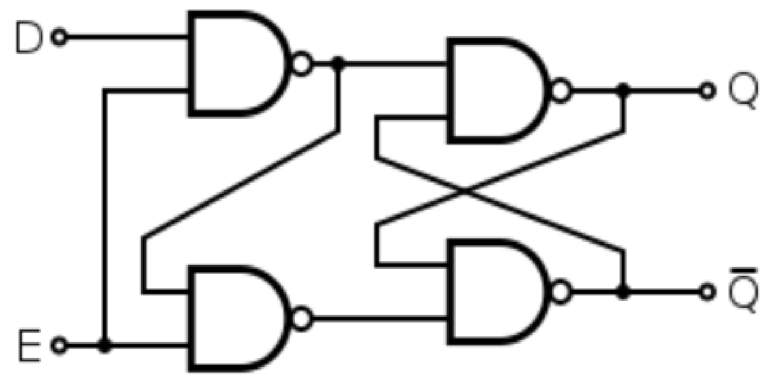
LEVEL-TRIGGERED J-K AND D FLIP-FLOPS ALSO POSSIBLE

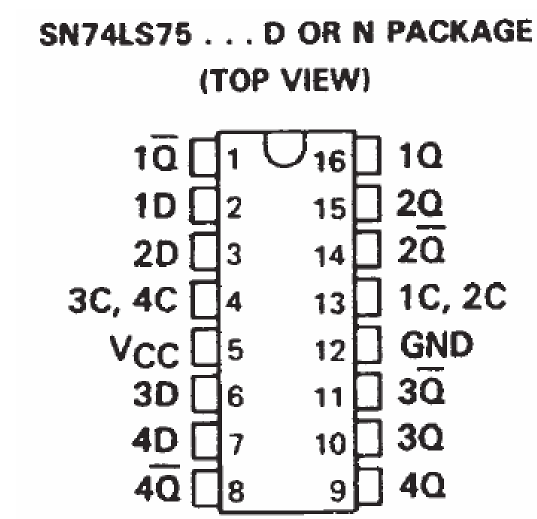
J-K Level-Triggered Flip-Flop



Restricted state permits toggling action

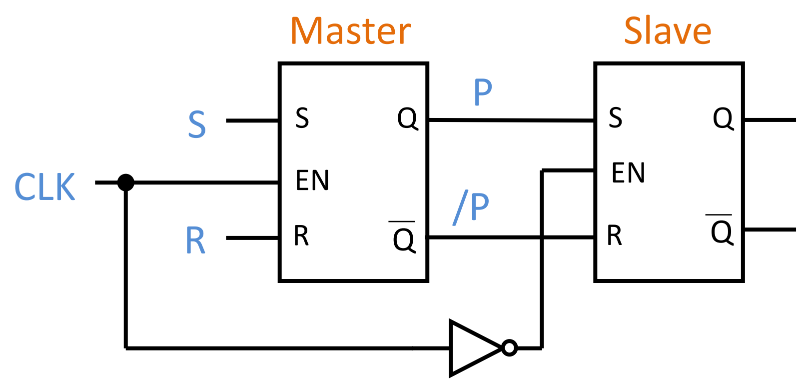
D Level-Triggered Flip-Flop





Texas Instruments SN7475
4-Bit Bi-Stable Latches

PULSE-TRIGGERED FLIP-FLOP

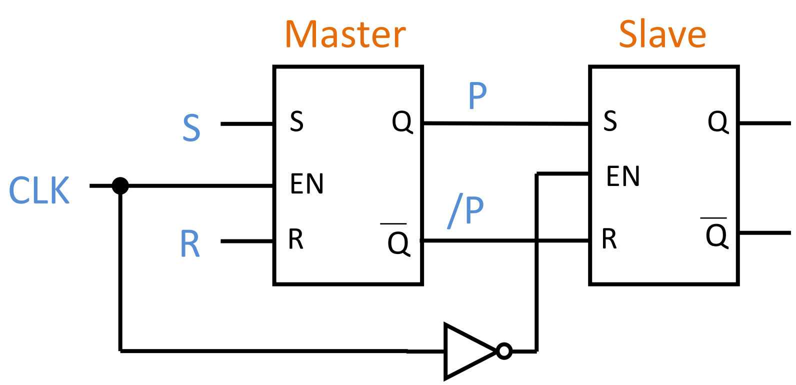


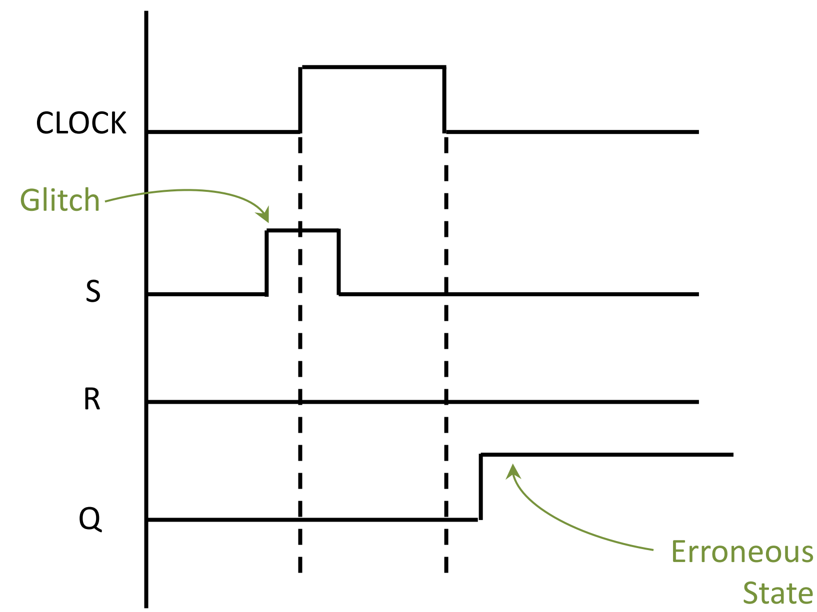
Allow no more than one state change during each clock period. Accomplished by using two stages:

- First stage accepts set and reset inputs at the ***rising*** edge of the clock signal, then generates an internal output state P
- Second stage accepts P and /P as inputs on the ***falling*** edge of the clock signal, and changes the output at the falling edge of the clock

PULSE-TRIGGERED FLIP-FLOP

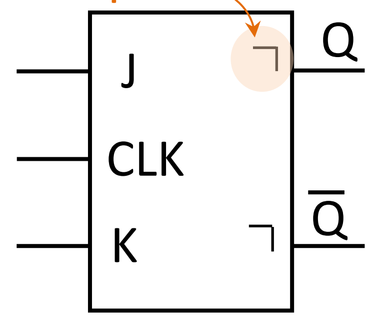
- Output transition is delayed until falling clock edge
- Downside: Catching Ones and Zeros
  - Any glitches at the inputs to a pulse-triggered flip-flop can cause unintended state changes
  - Must control "setup" time

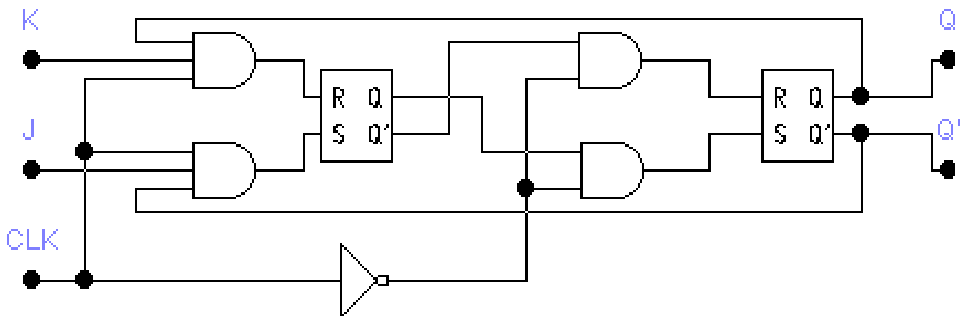


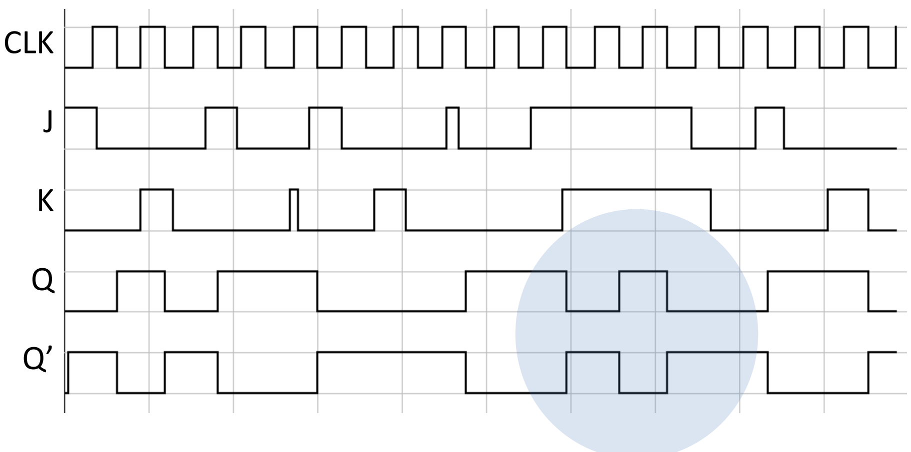


J-K PULSE-TRIGGERED FLIP-FLOP

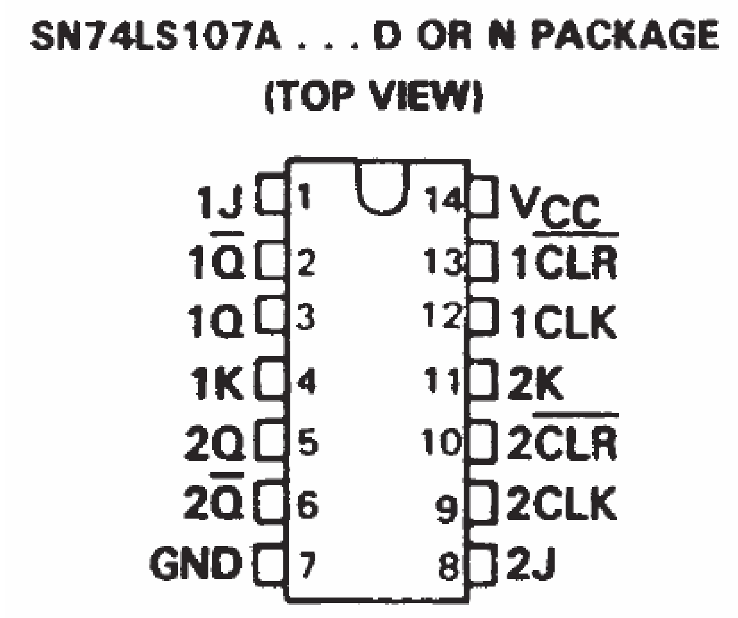
Postponed Output







Controlled Toggling!

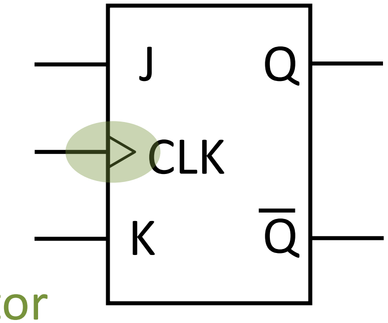


Texas Instruments SN74LS107
Dual J-K Flip-Flops with Clear

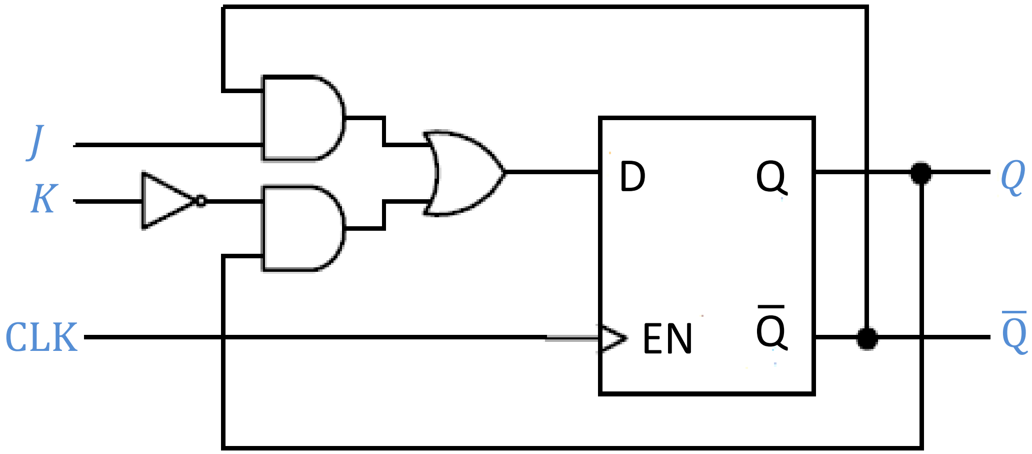
EDGE-TRIGGERED FLIP-FLOPS

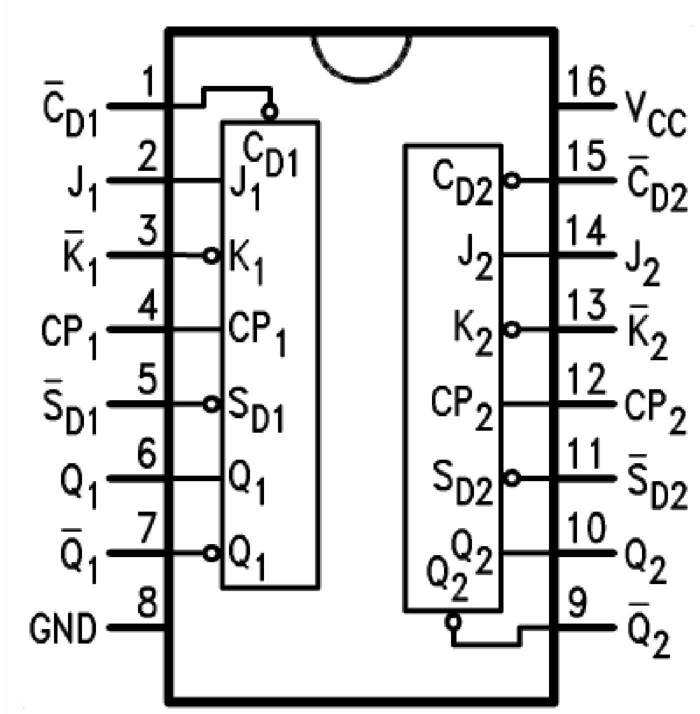
- Don't want to wait around for falling clock edge? Use an edge-triggered flip-flop.
- Sensitive to inputs for only a short time around the rising or falling clock pulse edge, allowing for faster operation than pulse-triggered flip-flops.
- Also known as "data lockout" flip-flops.

J-K EDGE-TRIGGERED FLIP-FLOP



Dynamic Input Indicator

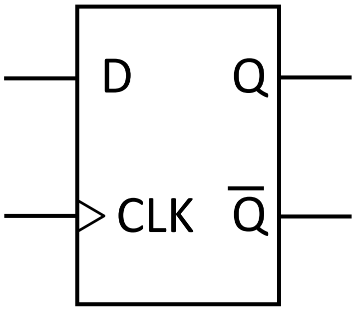


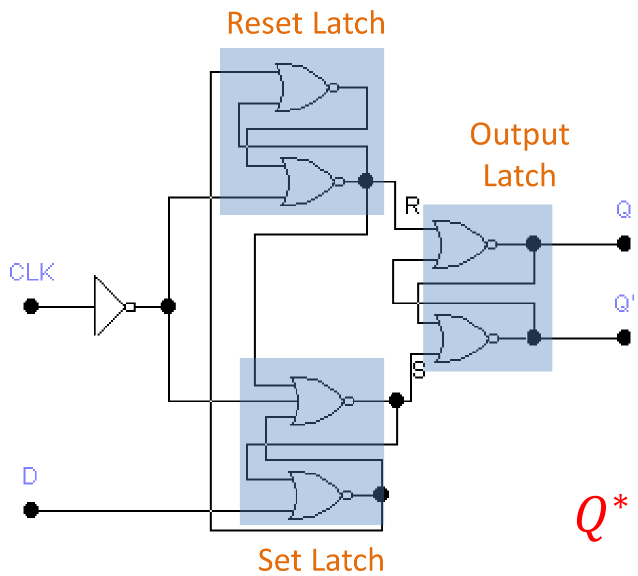


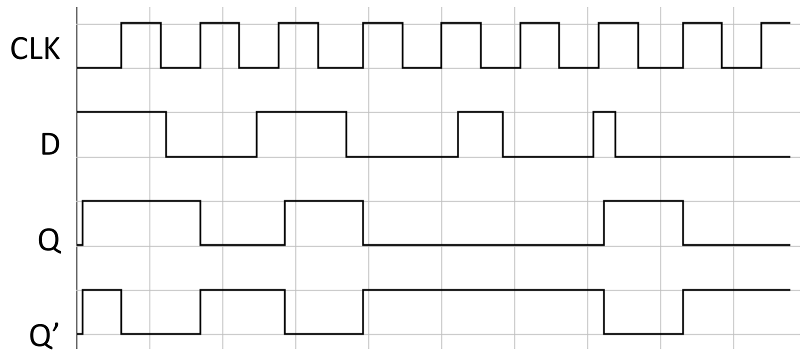
74109 Dual J-K' Positive Edge-Triggered Flip Flop

J	K	CL	Q^*
0	0	↑	Q
0	1	↑	0
1	0	↑	1
1	1	↑	$/Q$
X	X	0	Q
X	X	1	Q

D EDGE-TRIGGERED FLIP FLOP







$$Q^* = D$$

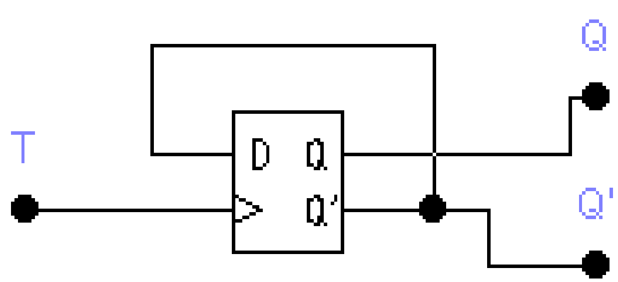
D	CLK	Q	Q*
0	↑	**X**	**0**
1	↑	**X**	**1**
×	0	**Q**	**Q**
×	1	**Q**	**Q**

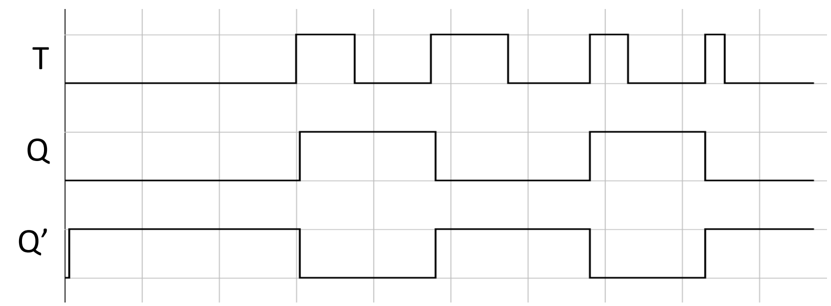
T (TOGGLE) FLIP-FLOP

- The output (stored state) is complemented when the input is asserted
- Not found in standard part list, but can be easily constructed using other types of flip-flops

Example:

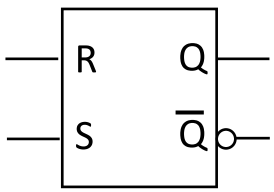
- Using an D flip-flop to construct a T flip-flop



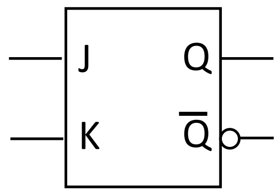


TTL LATCHES AND FLIP-FLOPS

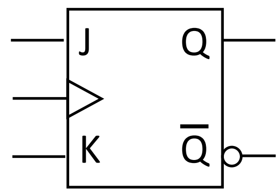
Schematic Symbols:



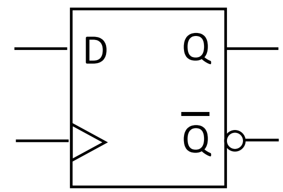
S-R Latch



J-K Latch

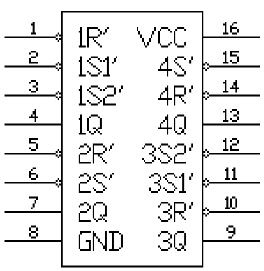


Edge-Triggered J-K Flip-Flop



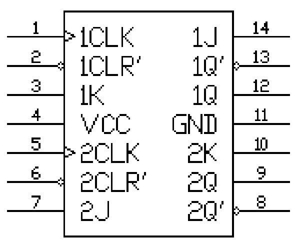
Edge-Triggered D Flip-Flop

TTL Components:



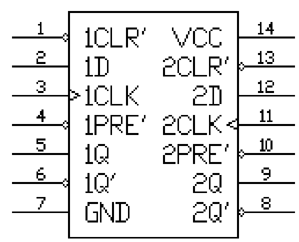
74279

Quad S-R Latch



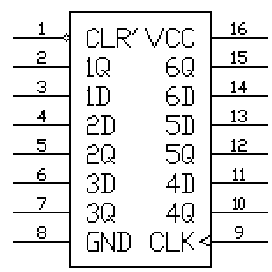
7473

Dual J-K Flip-Flop with CLR



7474

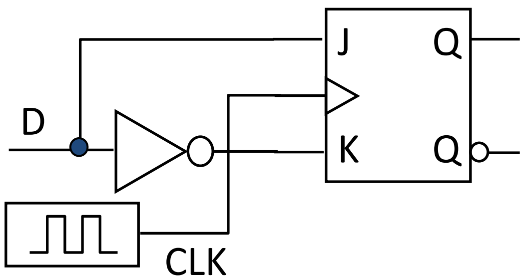
Dual D Flip-Flop with PRE CLR



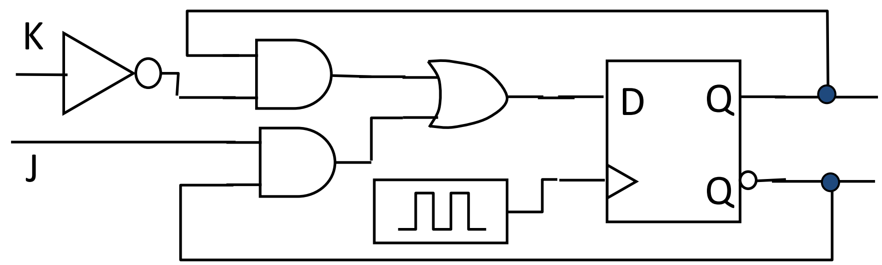
74174

Hex D Flip-Flop CLR

FLIP-FLOPS CAN BE CONSTRUCTED FROM OTHER FLIP-FLOPS



D flip-flop implemented by J-K flip-flop



J-K flip-flop implemented by D flip-flop

General Conversion – use excitation table as truth table to build interface combinational logic:

Q	Q^*	R	S	J	K	T	D
0	0	X	0	**0**	**X**	**0**	**0**
0	1	0	1	**1**	**0**	**1**	**1**
1	0	1	0	**0**	**1**	**1**	**0**
1	1	0	X	**X**	**0**	**0**	**1**
		$\underbrace{\quad\quad}_{\textit{S-R}}$		$\underbrace{\quad\quad}_{\textit{J-K}}$		$\underbrace{\quad}_{\textit{T}}$	$\underbrace{\quad}_{\textit{D}}$

SINGLE-BIT MEMORY DEVICE SUMMARY

S-R Latches

- Used as components in implementing master/slave or edge-triggered flip-flops
- Can be used for (and should only be used for) debouncing switches

J-K Flip-Flops

- Often results in the lowest gate count implementation of next-state combinational logic
- Requires two inputs per device – more complicated wiring

D Flip-Flops

- Requires only one input – attractive if wiring is an issue, as with most VLSI technologies

T Flip-Flops

- Good building blocks for *counters*
- Can be easily formed using J-K or D flip-flops

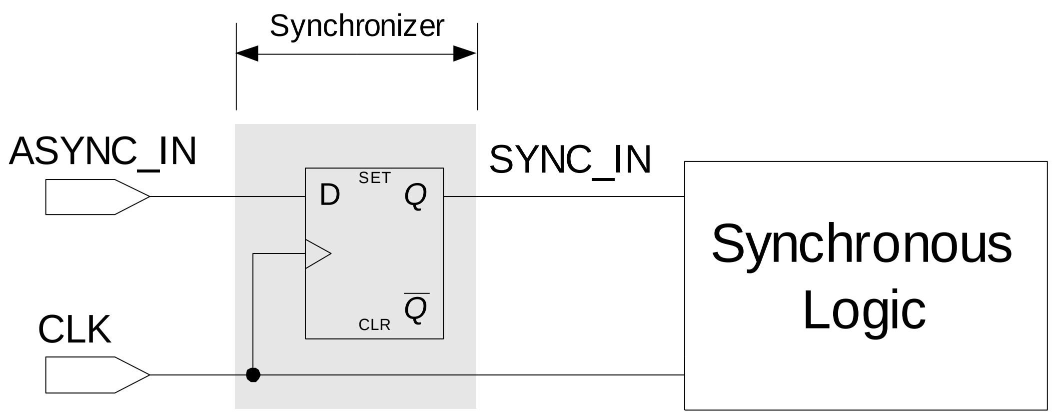
SINGLE-BIT MEMORY DEVICE SUMMARY

For the most part:

- Use SR-latches only for debouncing. Don't try to build your own flip-flops
- Use edge-triggered D flip-flops frequently
- Apply edge-triggered J-K flip-flops when needed, such as building T flip-flops

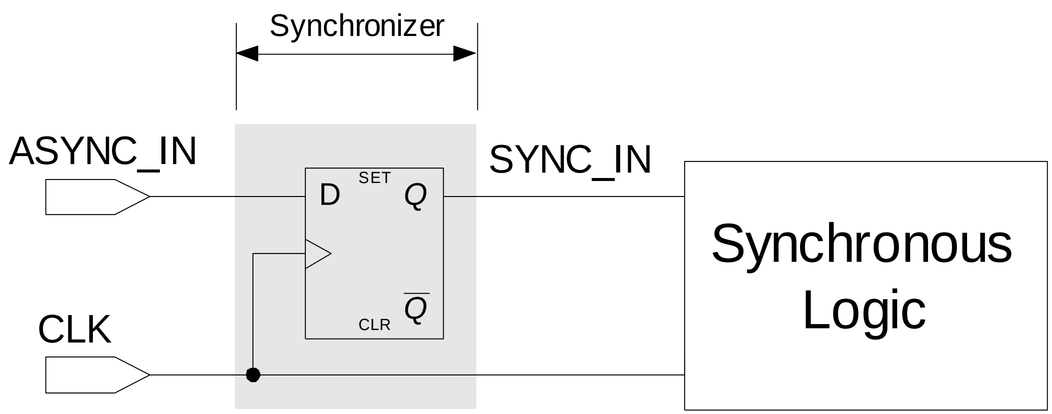
Everything else is an exception case

AN EDGE-TRIGGERED FLIP-FLOP CAN SYNCHRONIZE INPUTS



- All logic is synchronized with the same clock signal
- Provides similar noise margin in the temporal domain that binary quantization provides in the signal domain
- Allows a synchronous state machine

GOOD SYNCHRONOUS DESIGN PRACTICES



1. Use a single clock, and transition on a single edge as much as possible.
2. Avoid asynchronous presets and clears on flip-flops whenever possible.
3. Do not gate your clock signal!
4. Asynchronous inputs should be passed through synchronizers (usually composed of a D flip-flop with clock.)
5. Never fan-out asynchronous inputs: synchronize at a circuit boundary and fan-out a synchronized signal

COMING UP…

Computer Systems (Sequential Logic)

- Finite state machines
- FSM Examples
- Finite state reduction