SER 232

Computer Systems Fundamentals I

Dr. Heinrichs

Topics

- Logic Operators in Programming

Why Use Logical in Programs?

```
if(a == 5) {
  if(b != 6) {
    if(c > 25) {
      //Reached if 3 nested conditions met
    }
  }
}
```

Logical Operators

Operator	Description
&&	AND
||	OR
!	NOT

- Each value treated as a Boolean
  - False: zero
  - True: all non-zero values

Logical Operators

- AND and OR are used just like arithmetic operators
  - x = a && b
    - If *a* is 3 and *b* is 1 then *x* will be assigned true (or 1 in some languages)
    - If *a* is 0 and *b* is 2 then *x* will be assigned false (or 0)
  - y = c || d

- NOT can be placed to the left of an expression
  - y = !c
    - If *c* is 0 then *y* will be true (or 1)
    - If *c* is any other value then *y* will be false (or 0)

Combining Conditional Statements

```
if(a == 5) {
  if(b != 6) {
    if(c > 25) {
      //Reached if 3 nested conditions met
    }
  }
}
```

Combining Conditional Statements

```
if( (a == 5) && (b != 6) && (c > 25) ) {

  // Equivalent expression inside a single
  // conditional

}
```

Bitwise Operators

- Operation carried out separately for each bit in one value with the bit in the same position of another value
  - Often referred to as bit manipulation or bit twiddling

Operator	Description
&	AND
|	OR
^	XOR
~	NOT
<<	Shift left
>>	Shift right

Bitwise Operators

- Bitwise AND, OR, and XOR are also utilized like arithmetic operators
  - x = a & b
    Suppose *a* is 1101_2 and *b* is 1001_2

```
  1101
& 1001
------
  1001
```

Bitwise Operators

- Bitwise NOT is also placed to the left of an expression
  - y = ~c
    - If *c* is 10110_2 then *y* will be 01001_2

SER 232

Computer Systems Fundamentals I

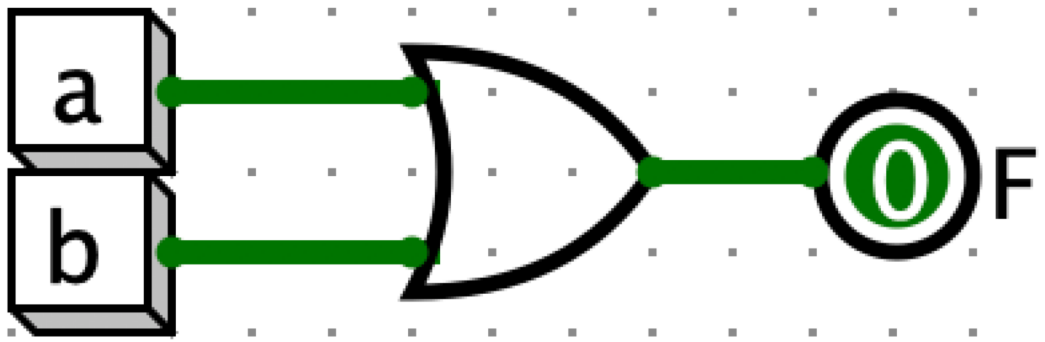
Dr. Heinrichs

Topics

- Combinational vs. Sequential Logic

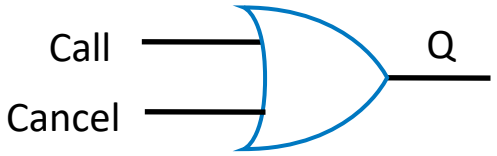
Combinational Circuit

- Every circuit we had so far was combinational
- Output solely based on input(s)
- No memory / capability to store information
- Simple circuit with buttons:
  - Press any button: Change output to F = 1
  - Release button: Revert back to initial state, F = 0



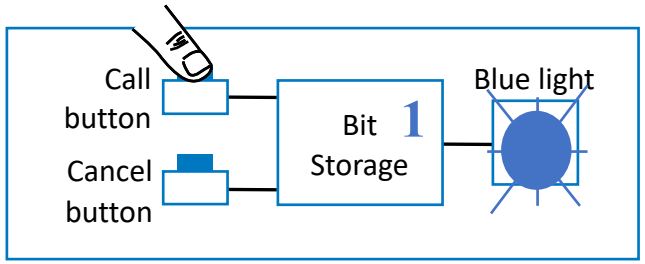
Sequential Circuit

- Output depends on current and *previous* input values
- Example: Flight attendant call button
  - Press call: light turns on
    - **Stays on** after button released
  - Press cancel: light turns off
    - Stays off after button released

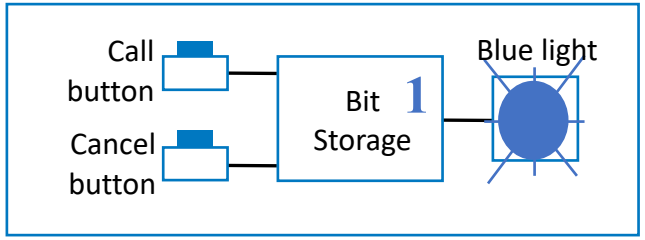


Doesn't work. Q=1 when Call=1, but doesn't stay 1 when Call returns to 0

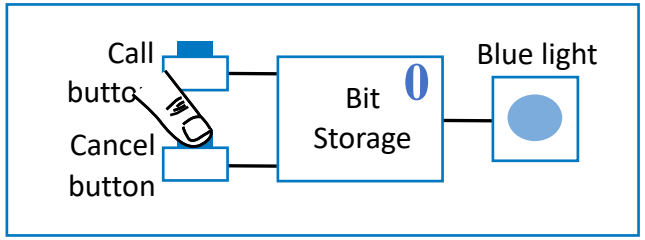
Need some form of "feedback" in the circuit



1. Call button pressed – light turns on



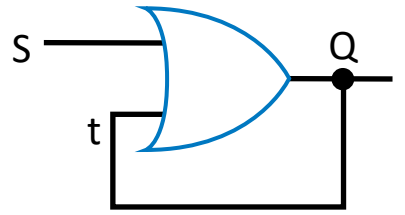
*2. Call button released – light **stays on***



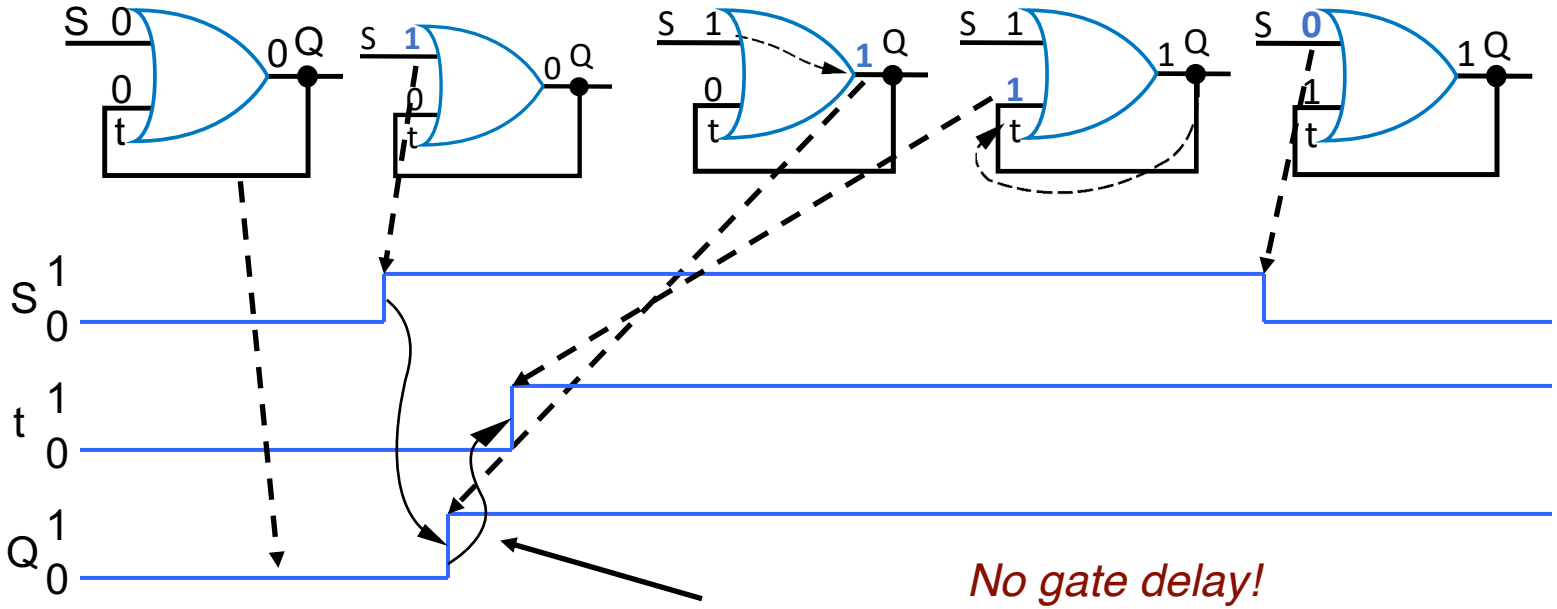
3. Cancel button pressed – light turns off

Source: Digital Design, Frank Vahid

Feedback



- Need some sort of feedback
  - Does circuit on the right do what we want?
  - No: Once Q becomes 1 (when S=1), Q stays 1 forever – no value of S can bring Q back to 0



No gate delay!
Only to visualize the order of changes

Source: Digital Design, Frank Vahid

Combinational vs. Sequential Logic

- Combinational circuits
  - Outputs only depend on current input values
  - Any previous inputs our output values are not considered
  - No state / memory
- Sequential circuits
  - Outputs depend on current input values and the sequence of previous input values
  - Sequential circuits have a state / memory

SER 232

Computer Systems Fundamentals I

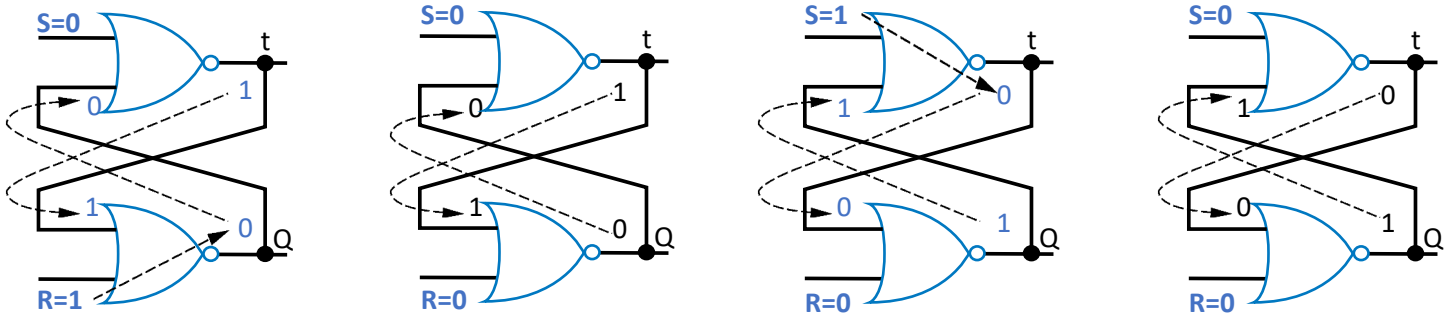
Dr. Heinrichs

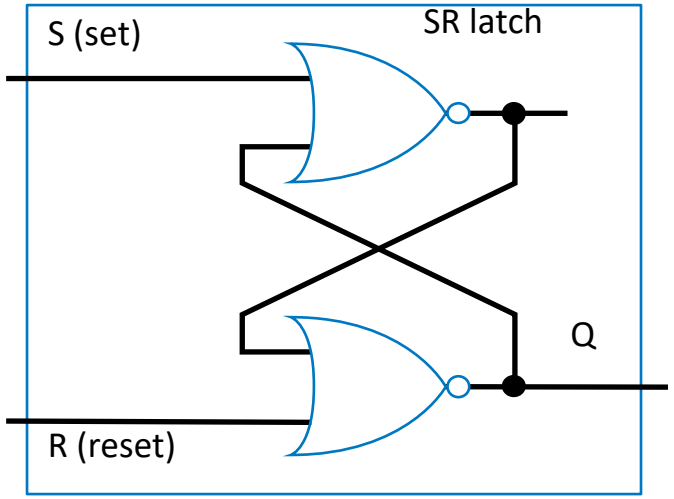
Topics

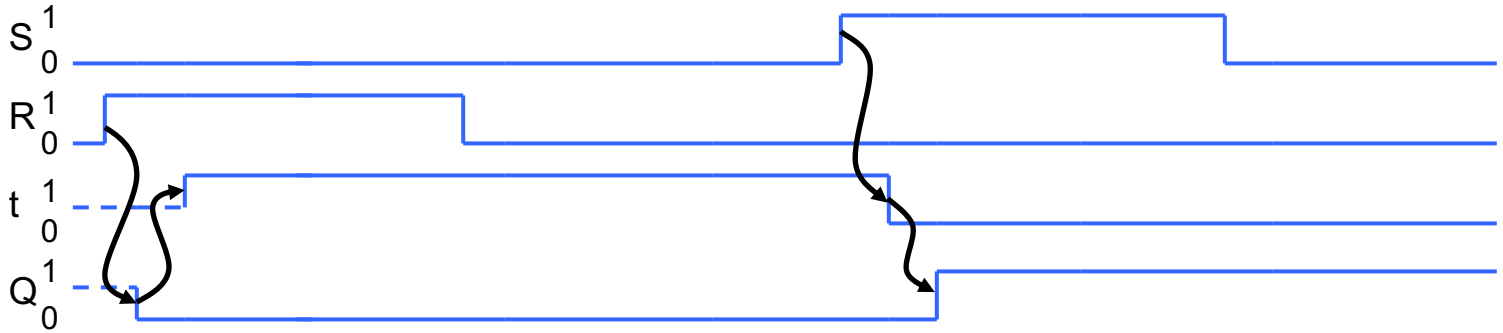
- SR Latch

SR Latch

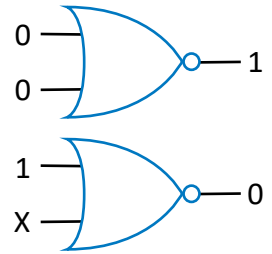
- Does the circuit to the right, with cross-coupled NOR gates, do what we want?
  - Yes!







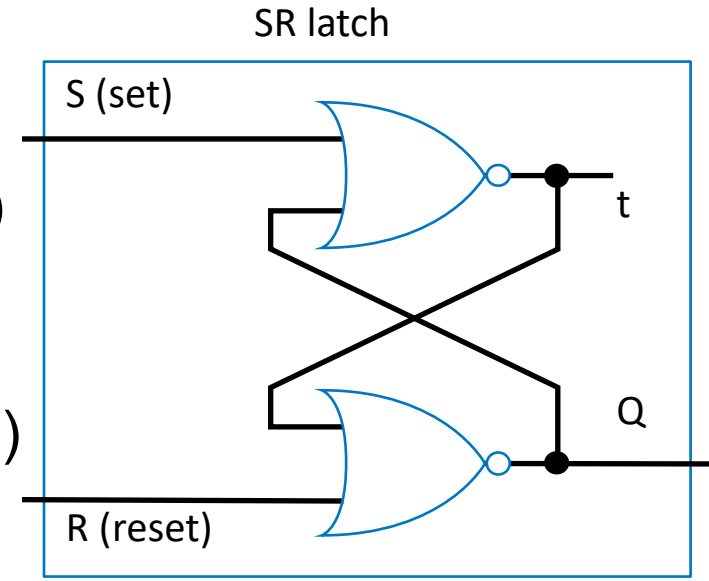
Recall NOR…



Source: Digital Design, Frank Vahid

SR Latch

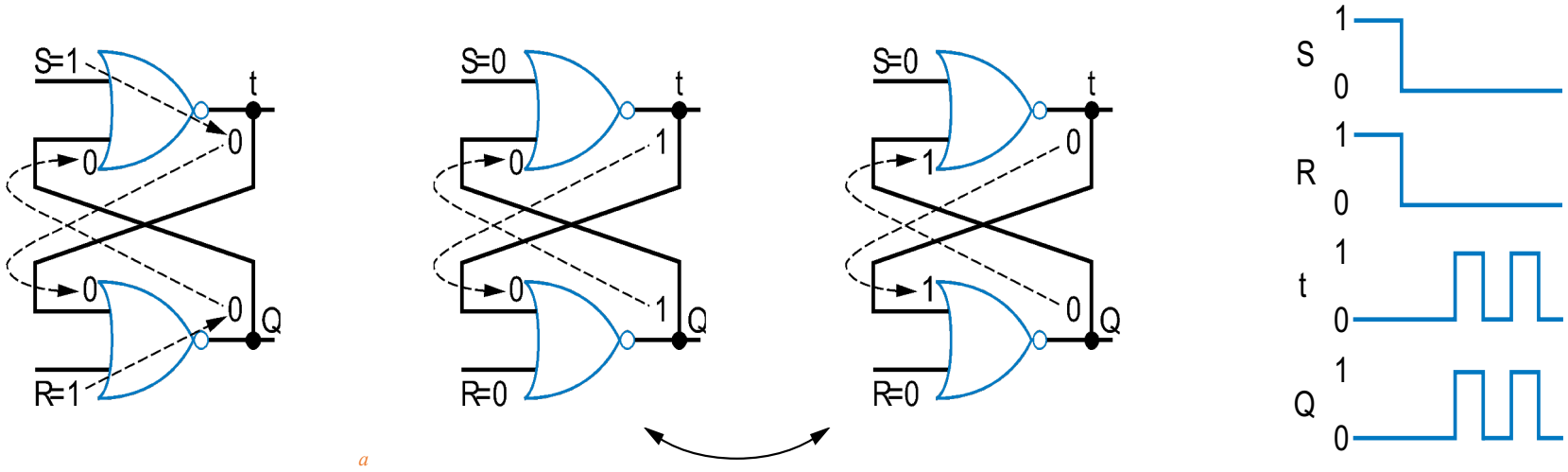
- How long must a button be pressed to have the output set to 1 or reset to 0?
  - Minimum = **delay of 2x NOR gates** (+ wires in between, which is close to 0)
- If `Q=1`; what value does t have?
  - **0**
- If `Q=0`; When latch is stable, what value(s) can `R` have? (`S=0`)
  - **0 or 1**
- If `Q=0`; When latch is stable, what value(s) can `S` have ? (`R=0`)
  - **0**



Source: Digital Design, Frank Vahid

Problem with SR Latch

- Problem:
  - If S=1 and R=1 and simultaneously go back to 0, we don't know what value Q will take

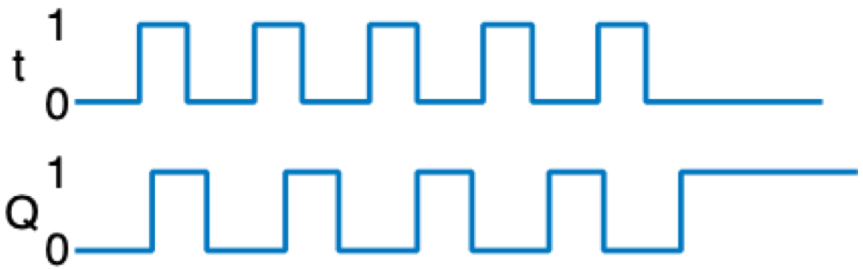


a

Source: Digital Design, Frank Vahid

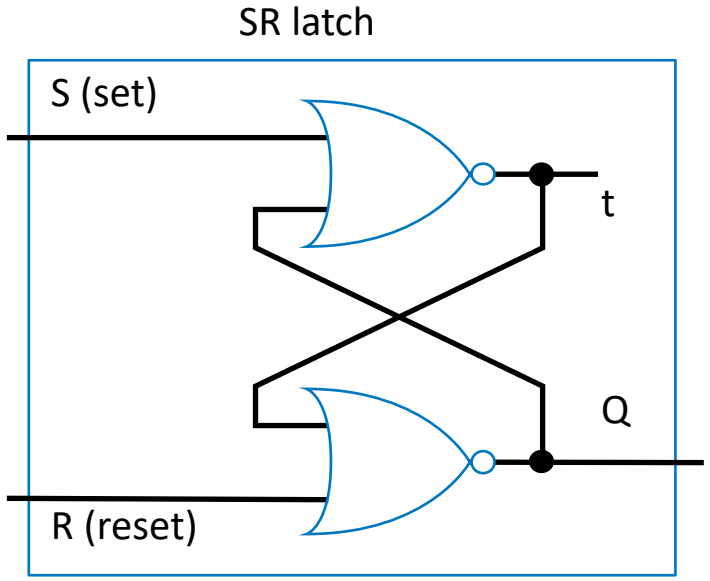
Problem with SR Latch

- Problem:
  - If S=1 and R=1 and simultaneously go back to 0, we don't know what value Q will take
  - Q may oscillate. Then, because one path will be slightly longer than the other, Q will eventually settle to 1 or 0 – but we don't know which. Known as a race condition.



Source: Digital Design, Frank Vahid

Problem with SR Latch

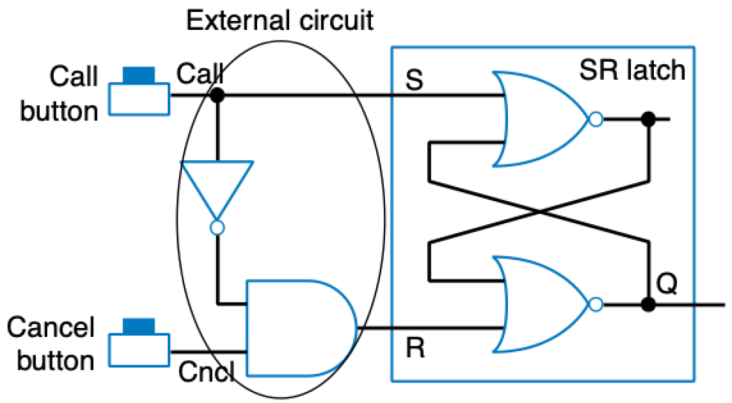


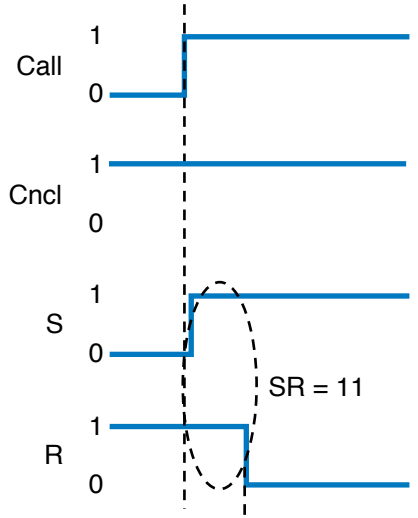
- If S=1 and R=1, what values do Q and t have?
  - **Q=0, t=0**
- When precisely does the oscillation problem occur?
  - **S and R change to 0 at the same time**
- Would the SR Latch oscillate if one NOR gate has twice the delay of the other NOR gate?
  - **No**. Q will always have the same value depending on which NOR gate is slower.

Source: Digital Design, Frank Vahid

SR Latch

- Designer might try to avoid problem using external circuit
  - Circuit should prevent SR from ever being 11
  - But 11 can occur due to different path delays

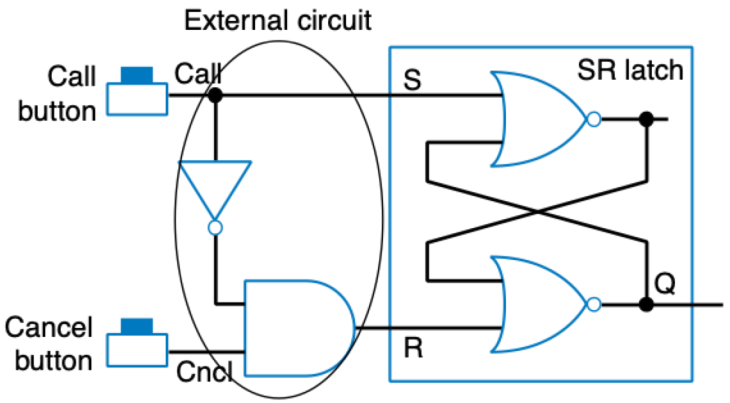


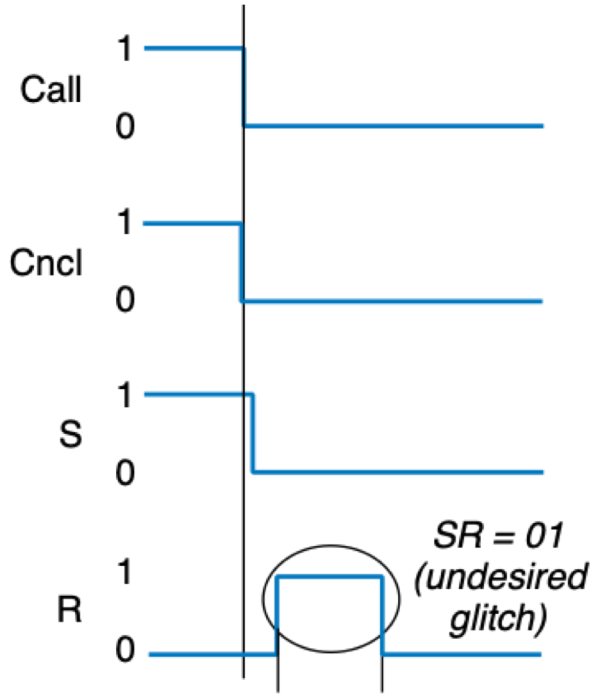


Source: Digital Design, Frank Vahid

SR Latch

- Glitch can also cause undesired set or reset

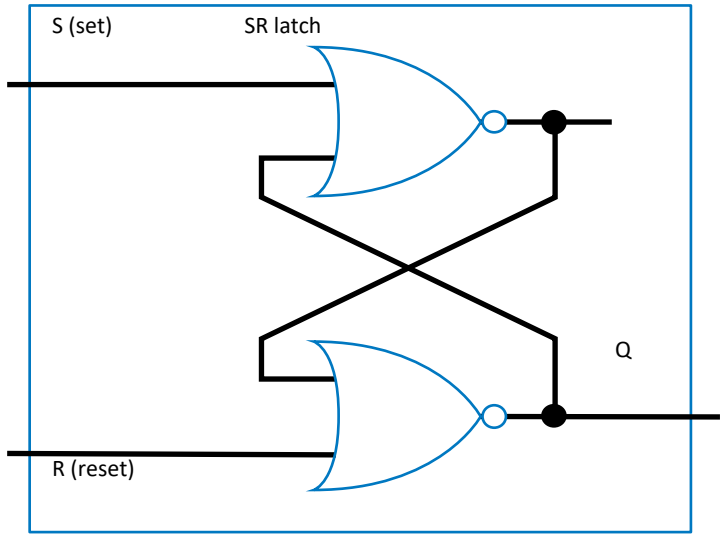


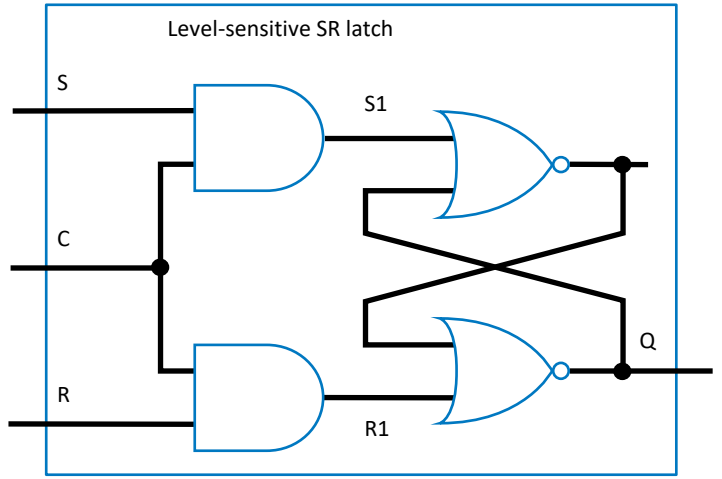


Source: Digital Design, Frank Vahid

Level-Sensitive SR Latch

- Add enable input “C”
- Only let S and R change when C=0
- Set C=1 after time for S and R to be stable
- When C becomes 1, the stable S and R value passes through the two AND gates to the SR latch’s S1 R1 inputs.

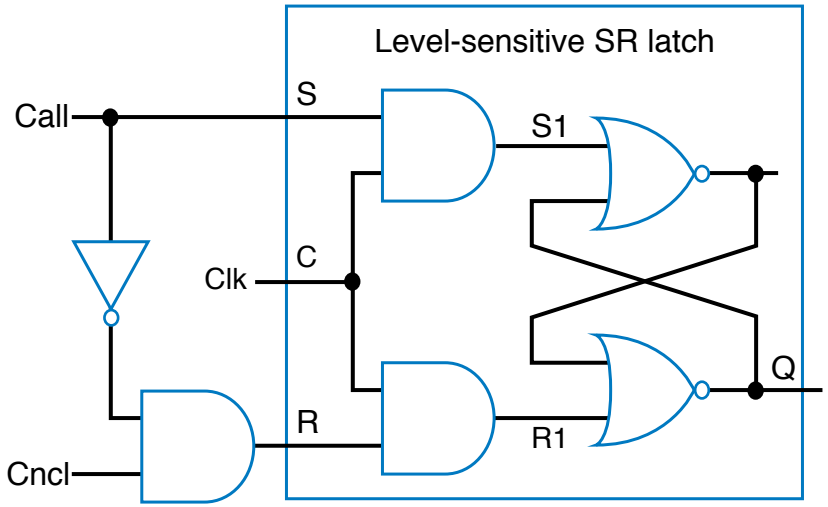


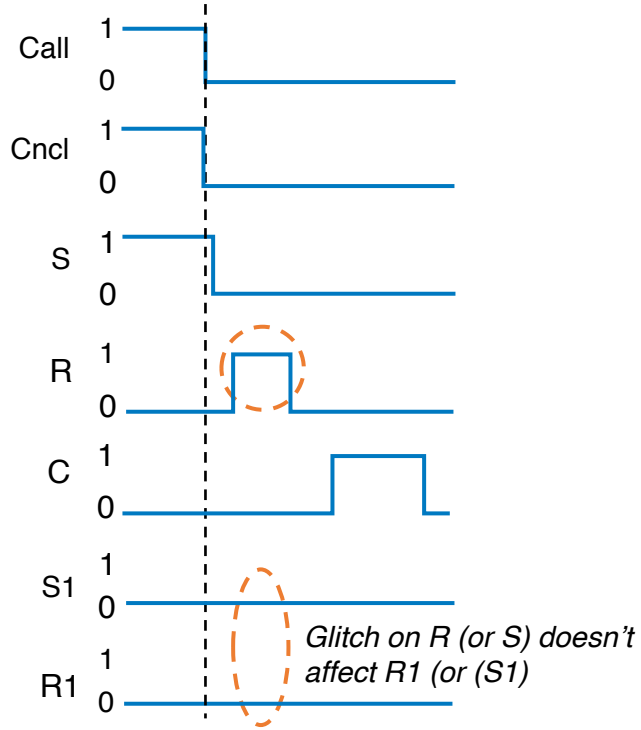


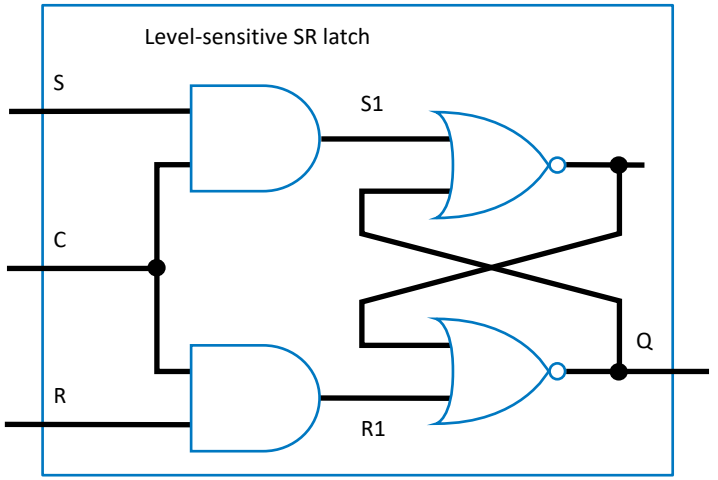
Source: Digital Design, Frank Vahid

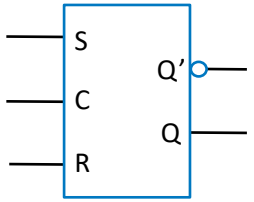
Level-Sensitive SR Latch

- Ensure circuit in front of SR never sets SR=11, except briefly due to path delays









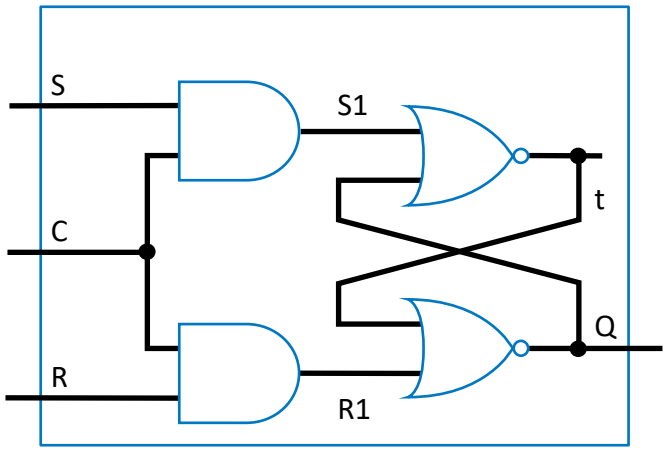
Level-sensitive SR latch symbol

Source: Digital Design, Frank Vahid

Level-Sensitive SR Latch

- In which situation can Q change?
  - **C=1**
- In which situation can t change?
  - **C=1**
- If C is always 1, is there a difference to the standard SR Latch?
  - **Yes**, we have an additional AND gate delay, but behavior is identical.

Level-sensitive SR latch



Source: Digital Design, Frank Vahid

SER 232

Computer Systems Fundamentals I

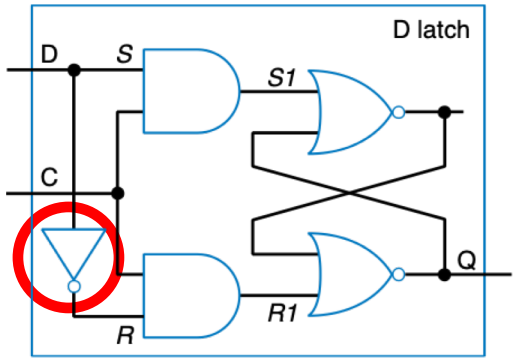
Dr. Heinrichs

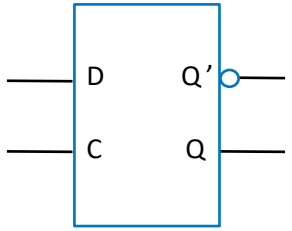
Topics

- D Latch

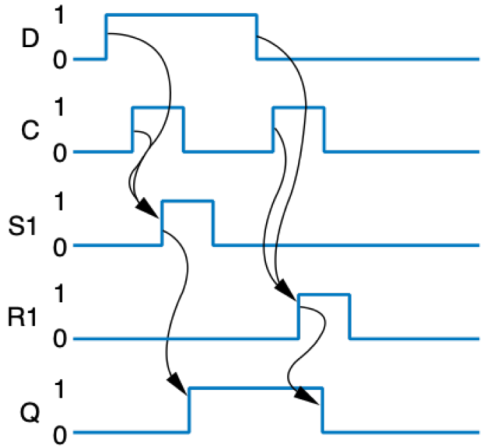
Level-Sensitive D Latch

- SR latch requires careful design to ensure SR=11 never occurs
  - D latch relieves designer of that burden
  - Inserted inverter ensures R always opposite of S





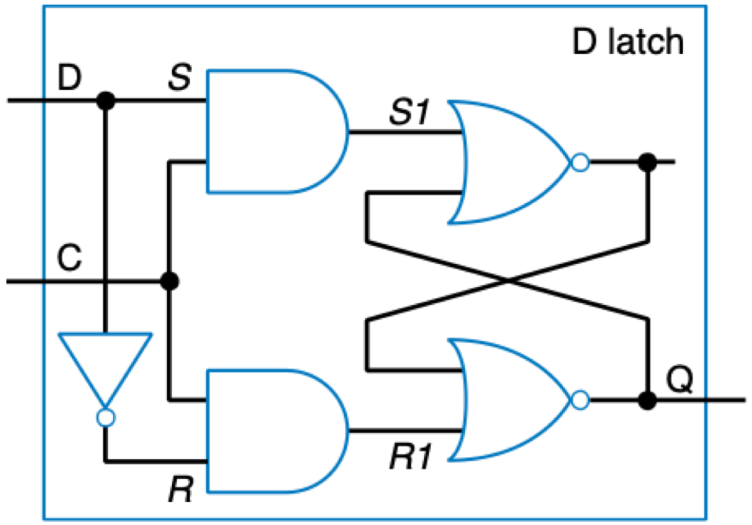
D latch symbol



Source: Digital Design, Frank Vahid

Level-Sensitive D Latch

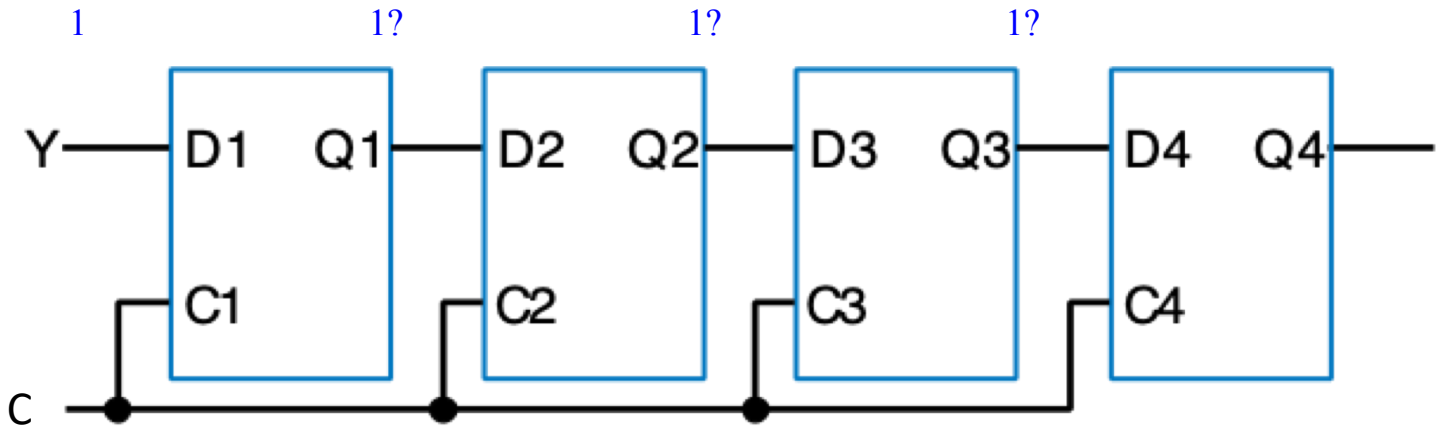
- How is a 1 stored?
  - **C=1 & D=1**
- How is a 0 stored?
  - **C=1 & D=0**
- What happens if D=1 and C=0?
  - **No change**, Q is whatever it was before, since C=0 disables the latch.
- If C is always 1, what will Q be?
  - **Whatever value D has**, after the delay of the whole circuit.



Source: Digital Design, Frank Vahid

Level-Sensitive D Latch

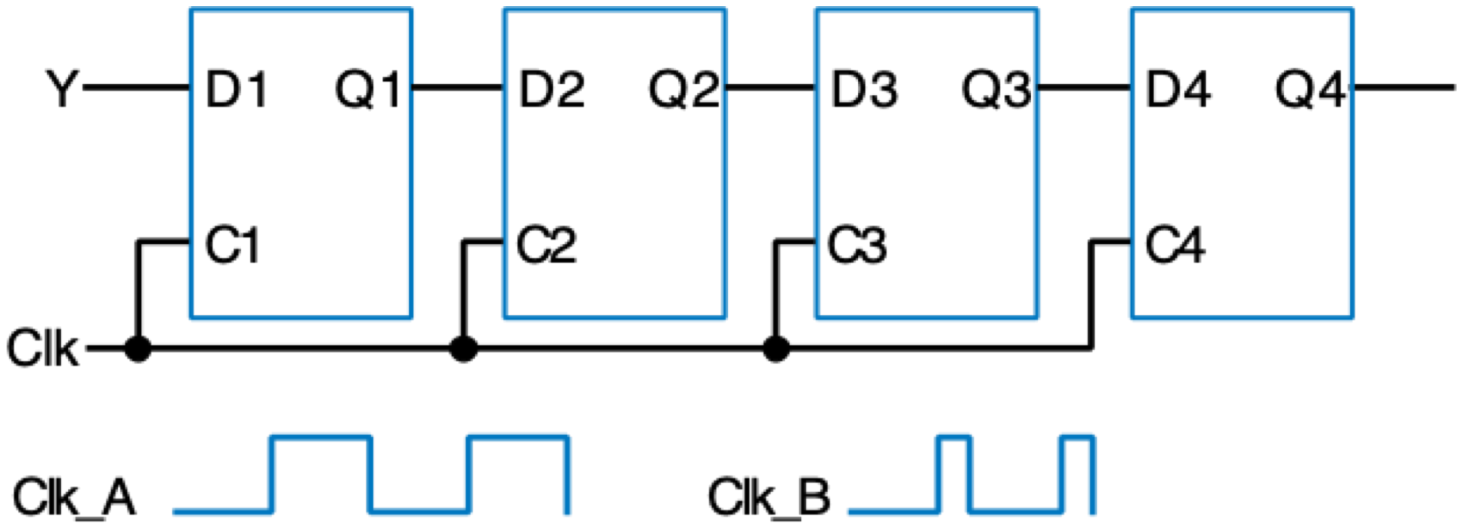
- D latch still has a problem (as does SR latch) when multiple latches are connected after another:
  - When C=1, through how many latches will a signal (e.g. Y = 1) travel?



Source: Digital Design, Frank Vahid

Level-Sensitive D Latch

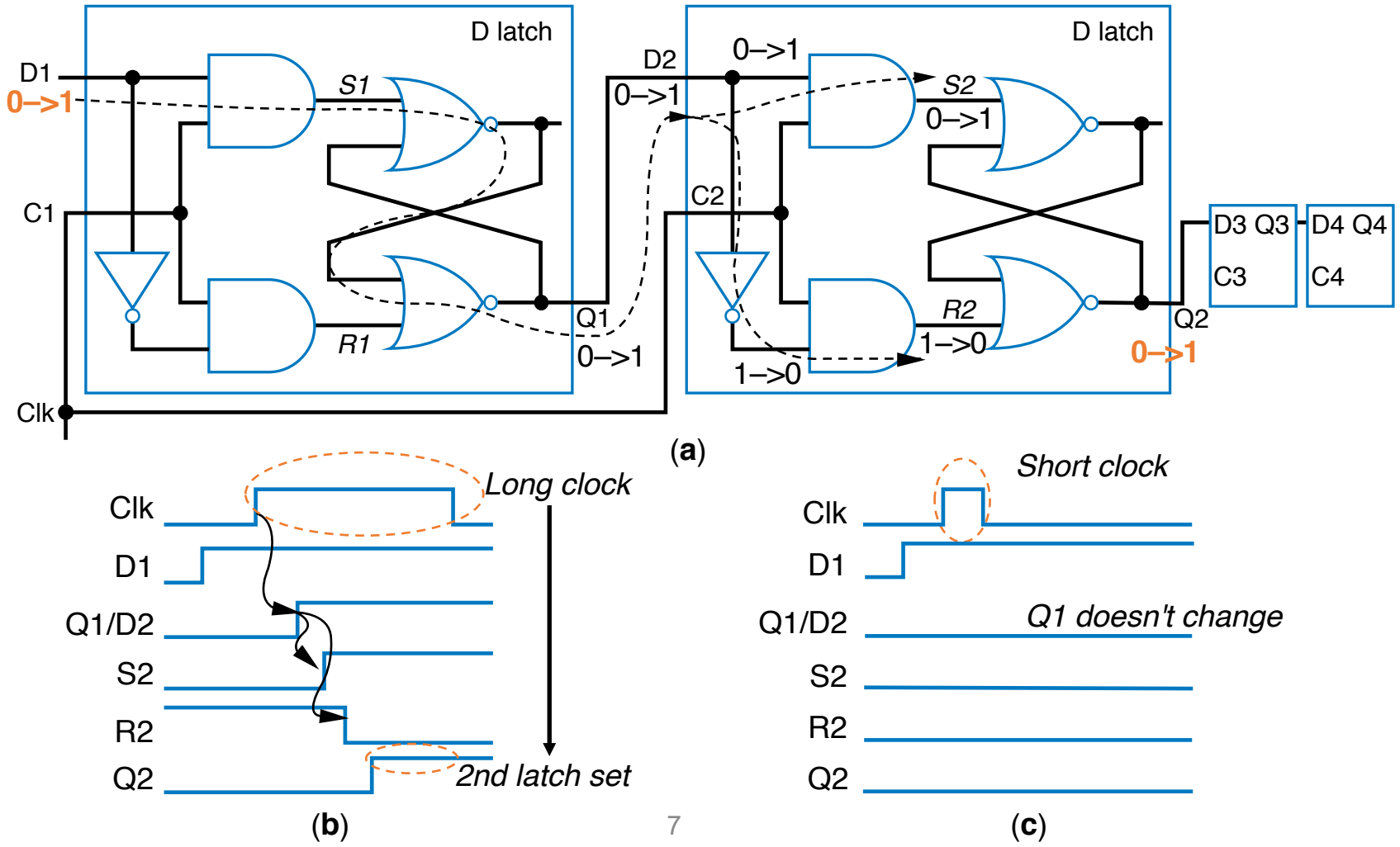
- When C=1, through how many latches will a signal travel?
  - Depends on how long C=1
    - Long clock period – signal may travel through multiple latches
    - Short clock period – signal may travel through fewer latches
- Clk_A: signal may travel through multiple latches
- Clk_B: signal may travel through fewer latches



Source: Digital Design, Frank Vahid

Level-Sensitive D Latch

- When C=1, through how many latches will a signal travel?



Source: Digital Design, Frank Vahid

Level-Sensitive D Latch

- Different clock speeds will change the behavior of Level-Sensitive D Latch
  - Should not be the case!
- Almost all modern CPUs will change their frequency depending on different aspects like load or core temperature

SER 232

Computer Systems Fundamentals I

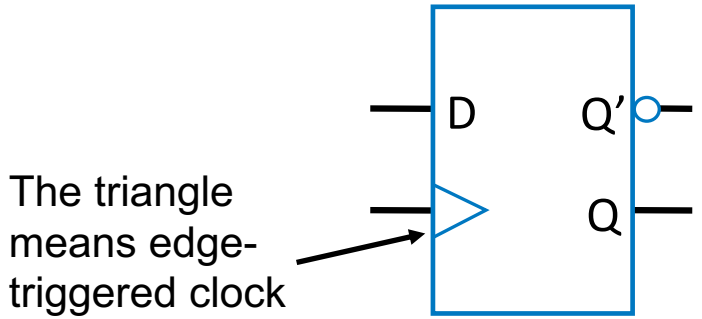
Dr. Heinrichs

Topics

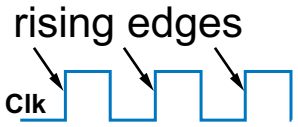
- D Flip Flop

D Flip Flop

- Solution: Store bit instantly and only on rising edge of clock
  - Edge-triggered D flip-flop
- A lot of different designs of D flip-flops possible
  - One design: *Master-Minion Design*
    - Utilizes two D latches



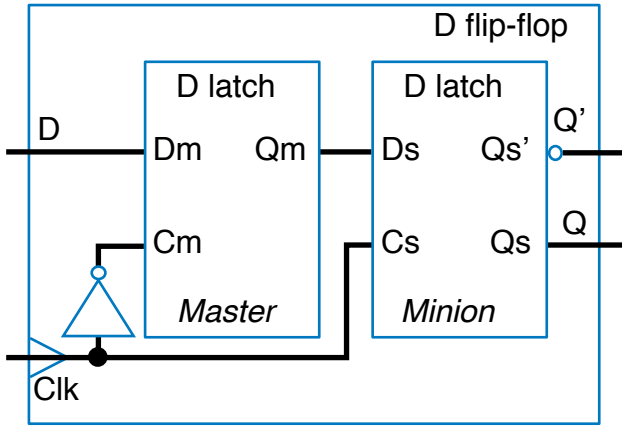
Symbol for rising-edge triggered D flip-flop

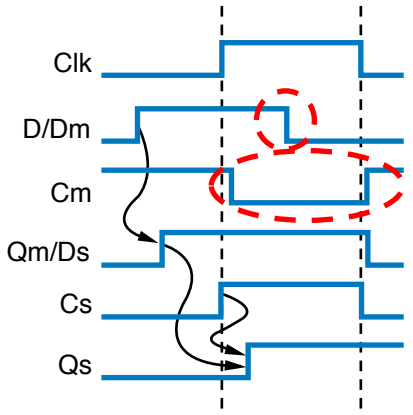


Source: Digital Design, Frank Vahid

D Flip Flop

- Master-Minion Design
  - Clk = 0 – master enabled, loads D, appears at Qm. Minion disabled.
  - Clk = 1 – Master disabled, Qm stays same. Minion latch enabled, loads Qm, appears at Qs.
  - Thus, value at D (and hence at Qm) when Clk changes from 0 to 1 gets stored into Minion



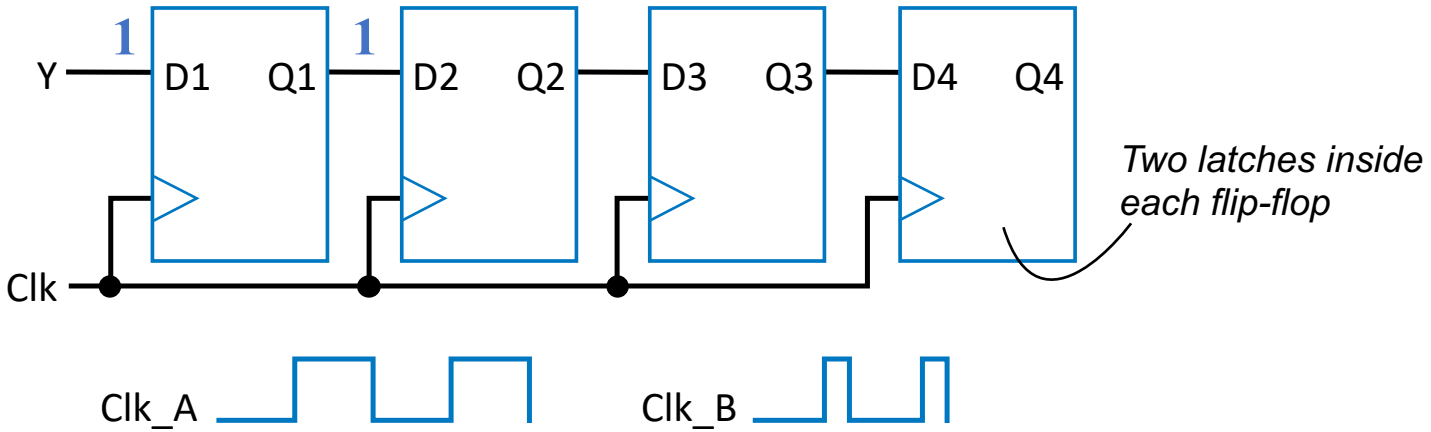


***Qm** does not change (since Cm = 0) until clock changes again*

Source: Digital Design, Frank Vahid

D Flip-Flop

- Solves problem of not knowing through how many latches a signal travels when C=1



Source: Digital Design, Frank Vahid

SER 232

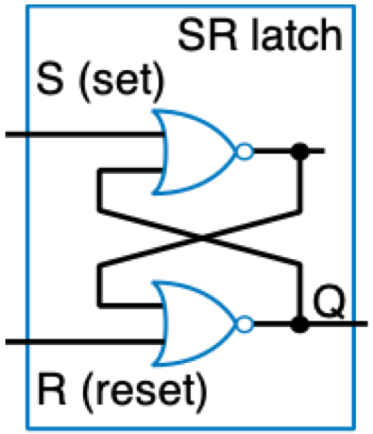
Computer Systems Fundamentals I

Dr. Heinrichs

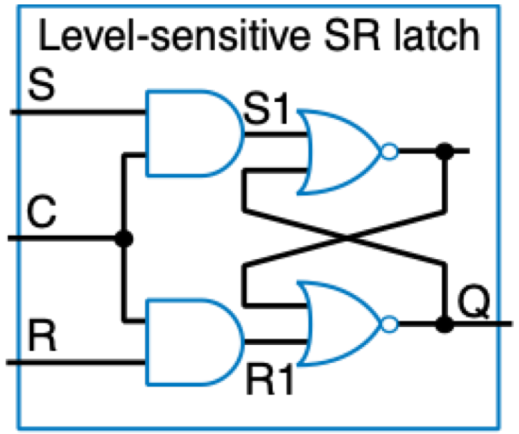
Topics

- Register

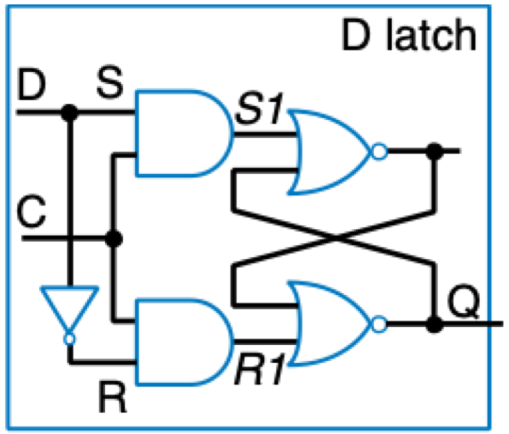
Summary



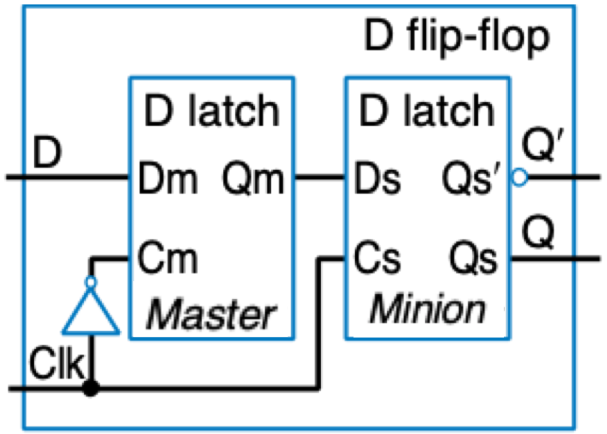
Feature: S=1 sets Q to 1, R=1 resets Q to 0.
Problem: SR=11 yields undefined Q, other glitches may set/reset inadvertently.



Feature: S and R only have effect when C=1. An external circuit can prevent SR=11 when C=1.
Problem: avoiding SR=11 can be a burden.



Feature: SR can't be 11.
Problem: C=1 for too long will propagate new values through too many latches; for too short may not result in the bit being stored.

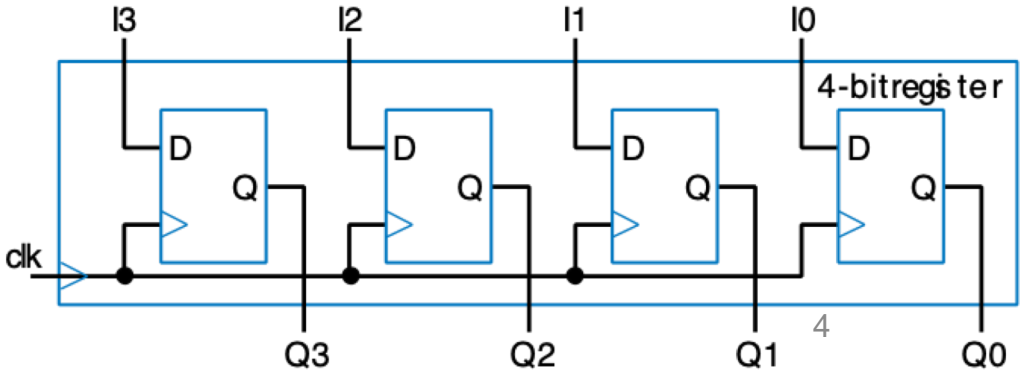


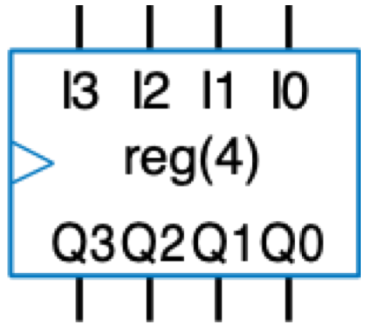
Feature: Only loads D value present at rising clock edge, so values can't propagate to other flip-flops during same clock cycle. *Tradeoff:* uses more gates internally, and requires more external gates than SR—but transistors today are more plentiful and cheaper.

Source: Digital Design, Frank Vahid

Register

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





Source: Digital Design, Frank Vahid