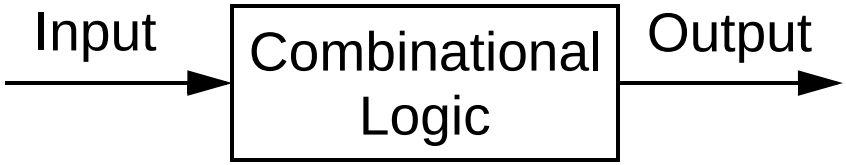
CHAPTER VII

SEQUENTIAL SYSTEMS - LATCHES & REGISTERS

SEQUENTIAL SYST.

INTRODUCTION

- So far...
  - So far we have dealt only with combinational logic where the output is formed from the current input.

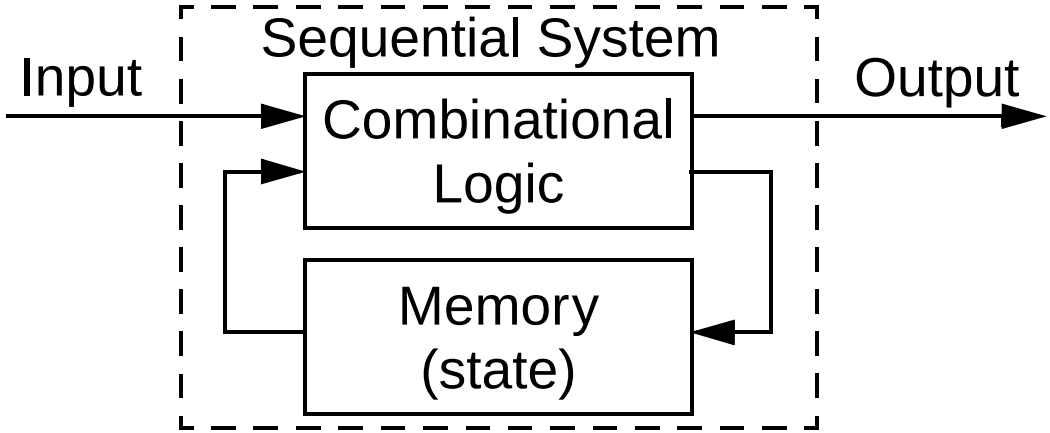


- Sequential systems
  - Sequential systems extend the idea of combinational logic by including a **system state**, or in other words **memory**, to our system.
  - This allows our system to perform operations that build on past operations in a *sequential* manner (*i.e.* one after another).
  - Timing diagrams will be needed to analyze the operation of many sequential systems.

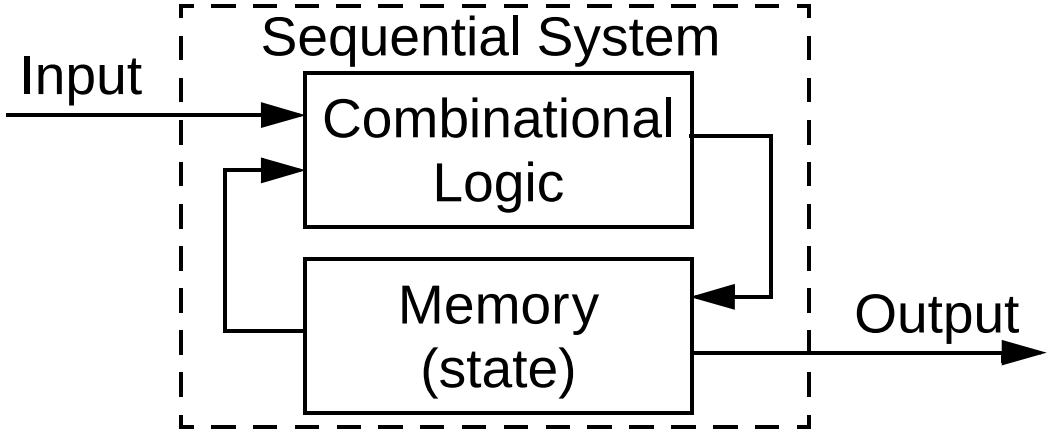
SEQUENTIAL SYST.

MEALY & MOORE MACHINES

- Mealy machine
  - Sequential system where output depends on current input and state.



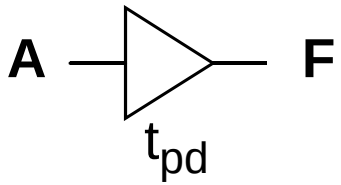
- Moore machine
  - Sequential system where output depends only on current state.



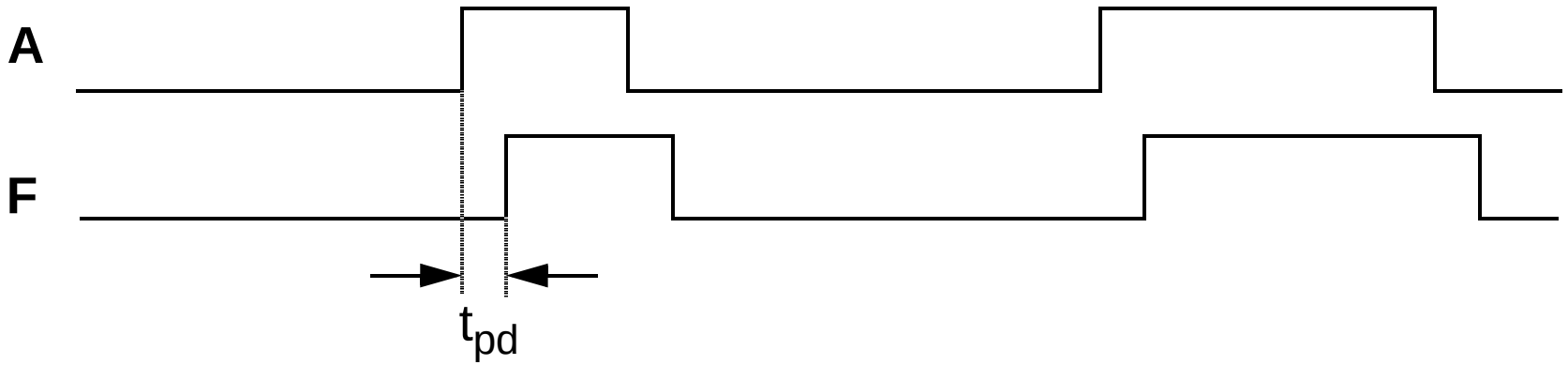
STORING BITS

STORING A BIT

- Since there are propagation delays in real components, this time delay can be used to store information.
  - For instance, the following buffer has a propagation delay of t_{pd}.



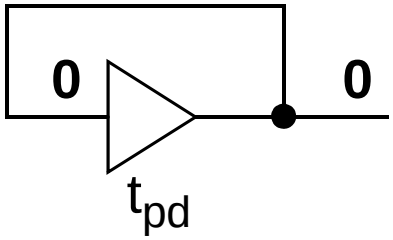
Timing Diagram

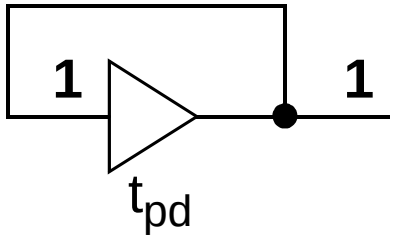


STORING BITS

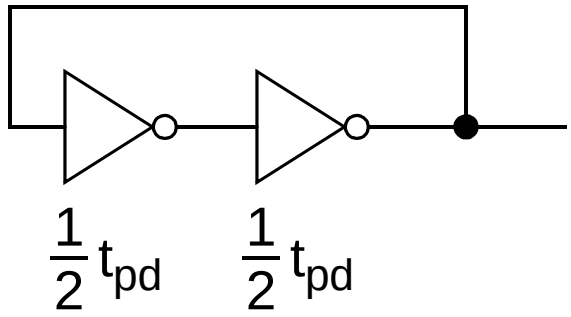
FEEDBACK LOOPS

- If we wish to store data for an indefinite period of time, then a feedback loop can be used to maintain the bit.





Can also use two inverters!

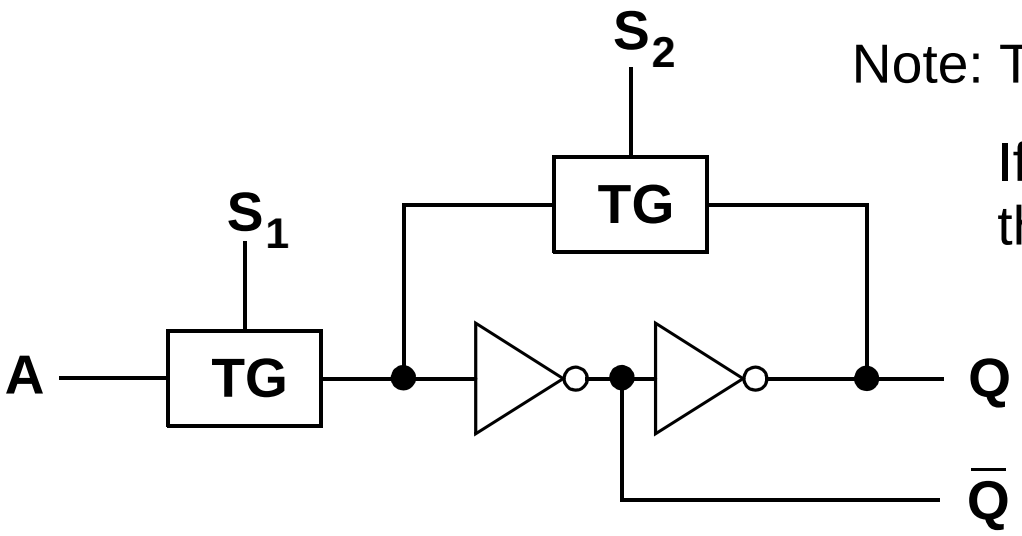


- How do we get the bit in there?

STORING BITS

LOADING A BIT

- To store a bit, we need a way of loading an input bit into the structure and making/breaking the connection in the feedback look.
  - One way of breaking connections is to use transmission gates.



Note: The latch is level-sensitive.

If **A** changes while $S_1 = 1$, then **Q** will change as well.

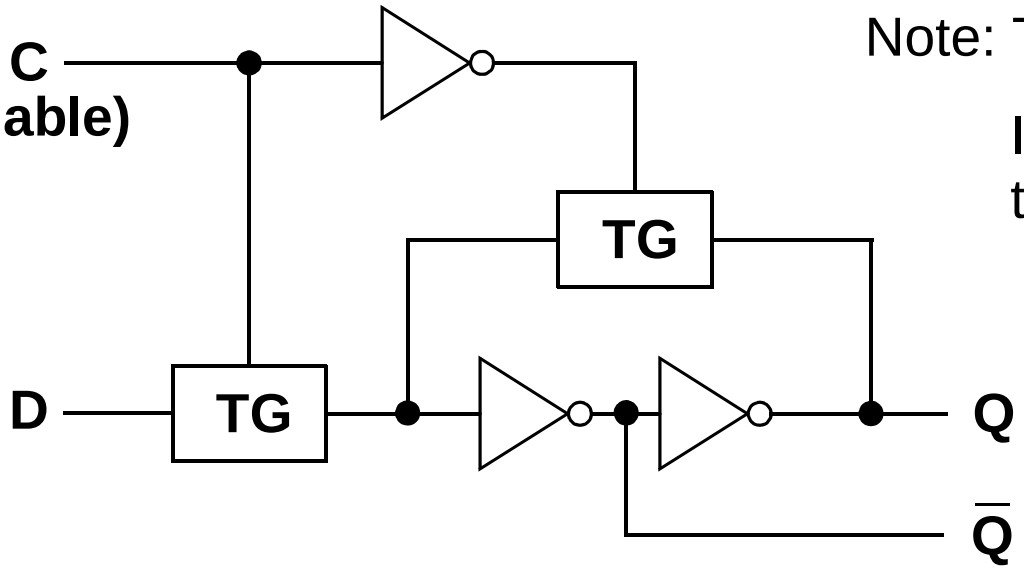
- **A** gets temporarily stored in the inverters when $S_1 = 1$ and $S_2 = 0$.

  Then setting $S_1 = 0$ and $S_2 = 1$, **A** gets held in the feedback loop.

LATCHES

D LATCH (WITH TG)

- The previous example is a data latch (**D latch**) if both S_1 and S_2 are controlled by a single line **C** as follows.



Note: The latch is level-sensitive.

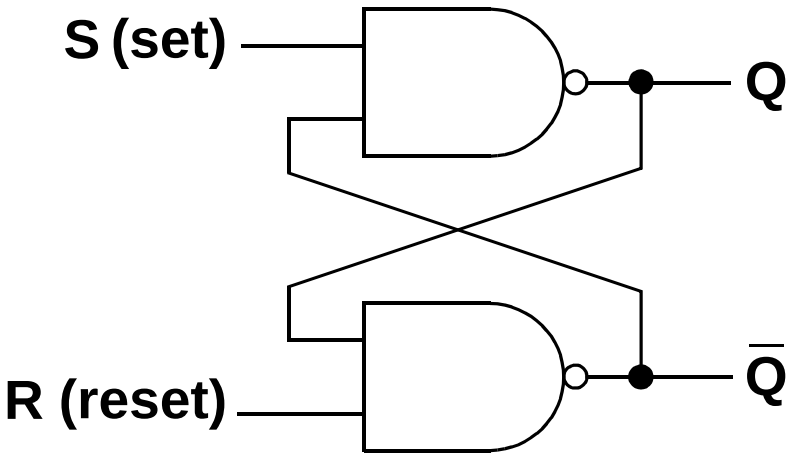
If **D** changes while **C = 1**, then **Q** will change as well.

- The control line **C** might be derived from the **clock signal**, or a signal from the **controller/sequencer** in the microprocessor.

LATCHES

$\bar{S}\bar{R}$ LATCH (NAND GATES)

- NAND gates can also be used to create a latch, this time an $\bar{S}\bar{R}$ latch.



S	R	Q	$\bar{Q}$	
1	0	0	1	
1	1	0	1	(after S = 1, R = 0)
0	1	1	0	
1	1	1	0	(after S = 0, R = 1)
0	0	1	1	

Recall:

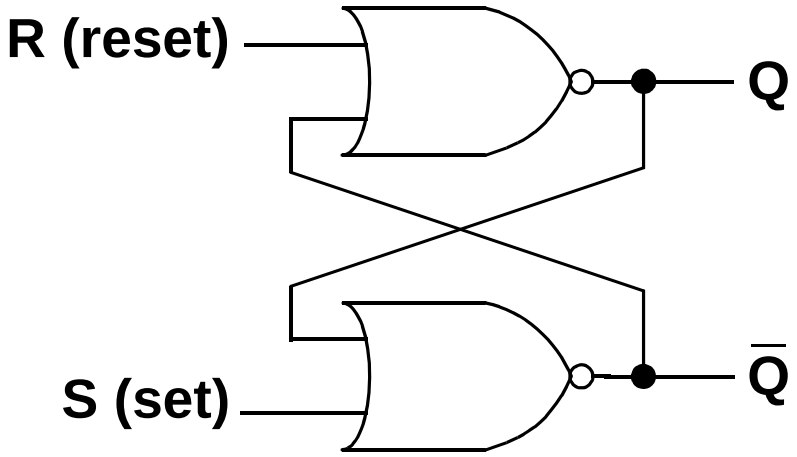
A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

- Notice that this latch is level-sensitive.

LATCHES

SR LATCH (NOR GATES)

- The SR latch also uses feedback to "store" a bit.



S	R	Q	$\overline{Q}$	
1	0	1	0	
0	0	1	0	(after S = 1, R = 0)
0	1	0	1	
0	0	0	1	(after S = 0, R = 1)
1	1	0	0	

Recall:

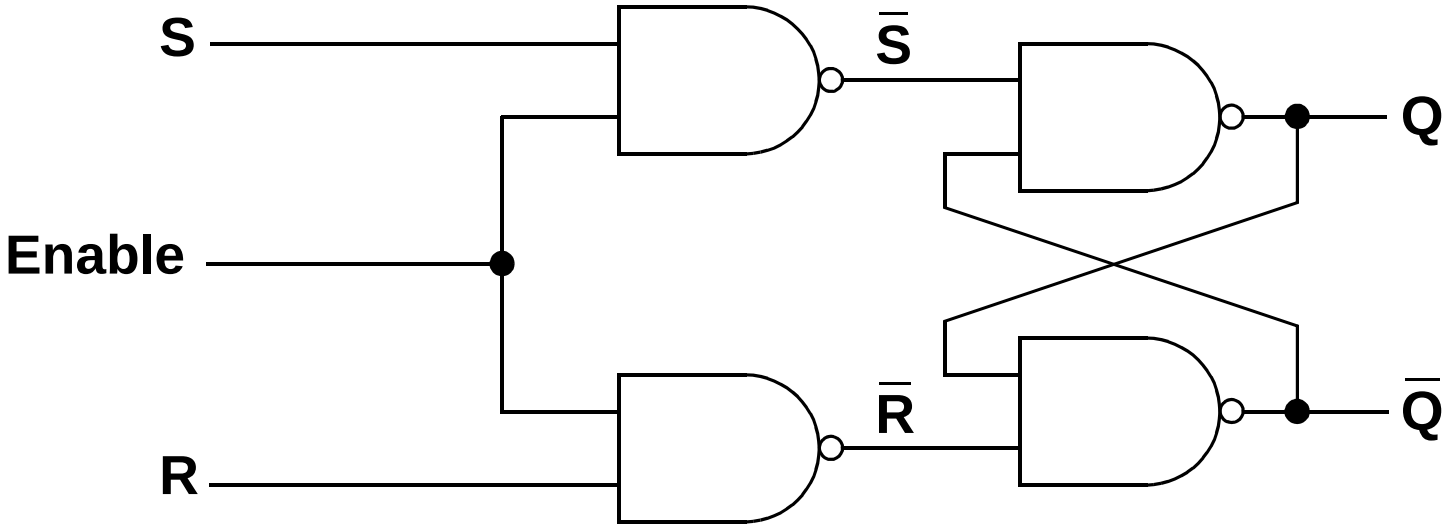
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

- Notice that this latch is level-sensitive.

LATCHES

SR LATCH WITH CONTROL

- A control line can be added to the $\overline{S}\overline{R}$ latch as follows forming an SR latch

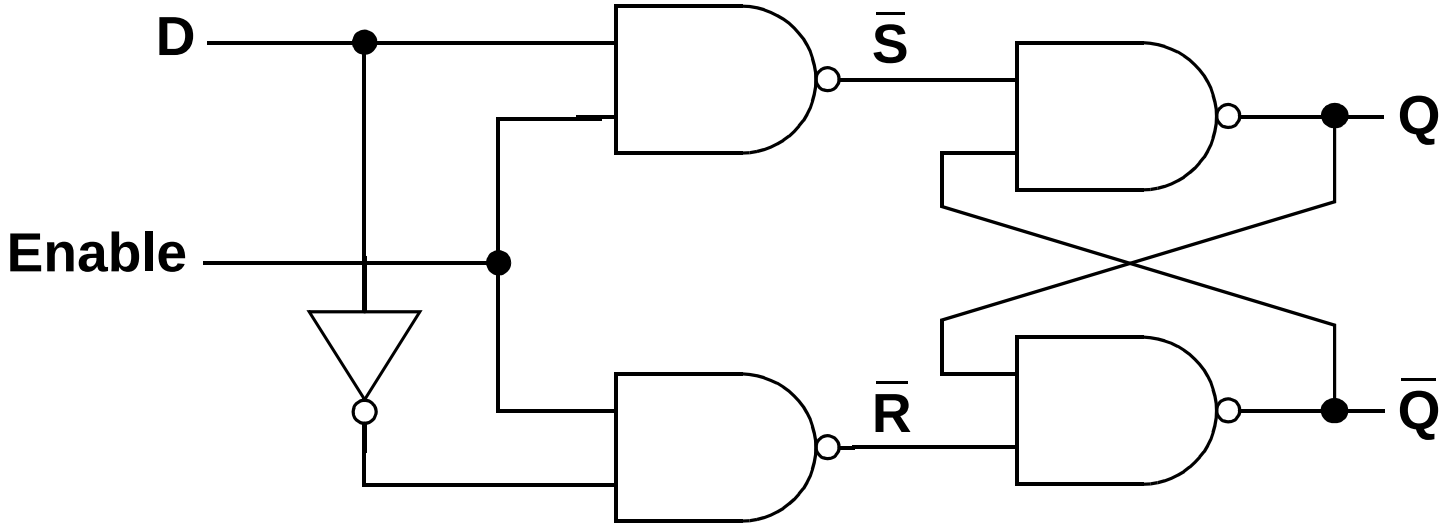


- This control line makes it possible to decide when the inputs **S** and **R** are allowed to change the state of the latch.

LATCHES

D LATCH (WITH $\bar{S}\bar{R}$ LATCH)

- A D latch can be implemented using what is effectively the SR latch with a control line as follows.

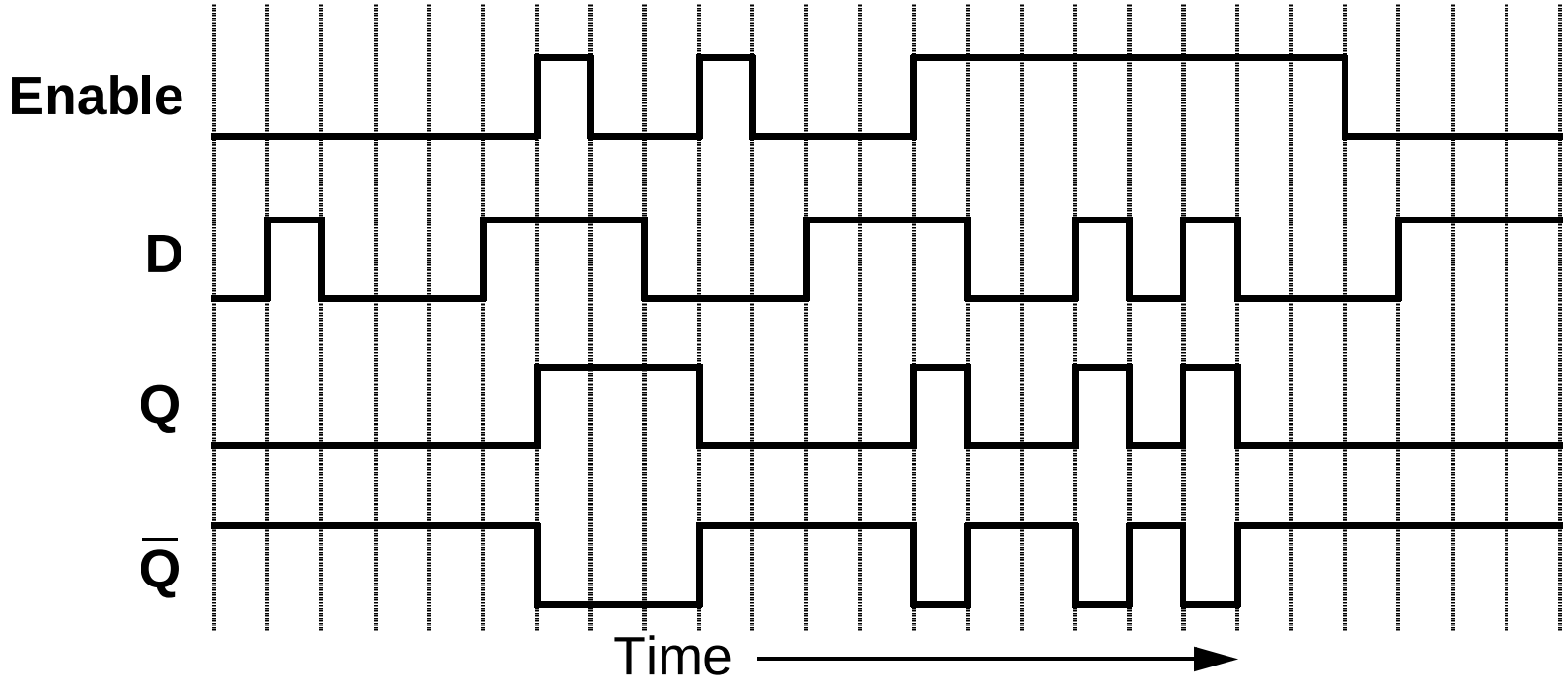


- Note that as long as **C** = **1**, that the latch will change according to the value of **D**.

LATCHES

TIMING DIAGRAMS

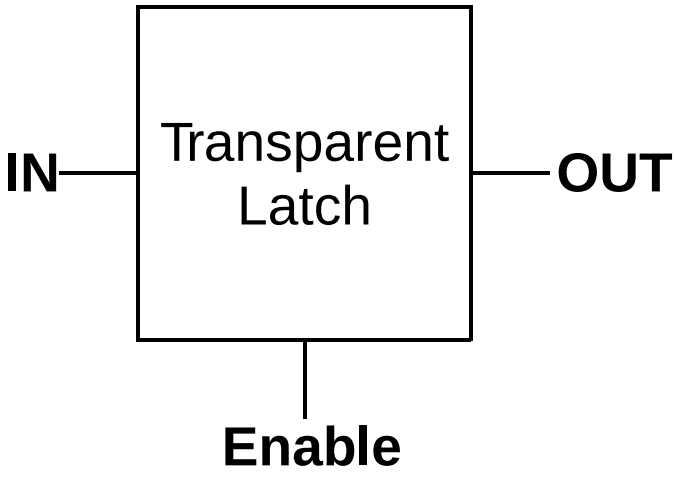
- Timing diagrams allow you to see how a sequential system changes with time using different inputs.
  - For instance, a timing diagram for a **D latch** might look like the following.

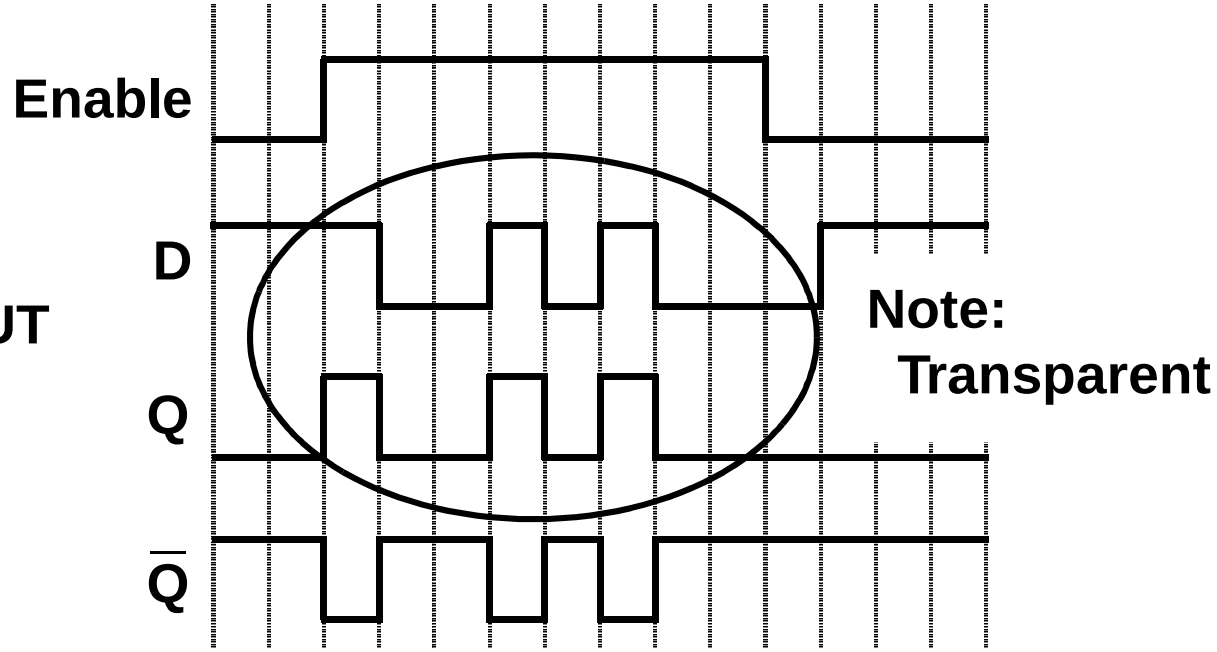


LATCHES

TRANSPARENCY (1)

- Latches like the D latch are termed **“transparent”** or **level-sensitive**.
  - This is because, when enabled, the **output follows the input**.

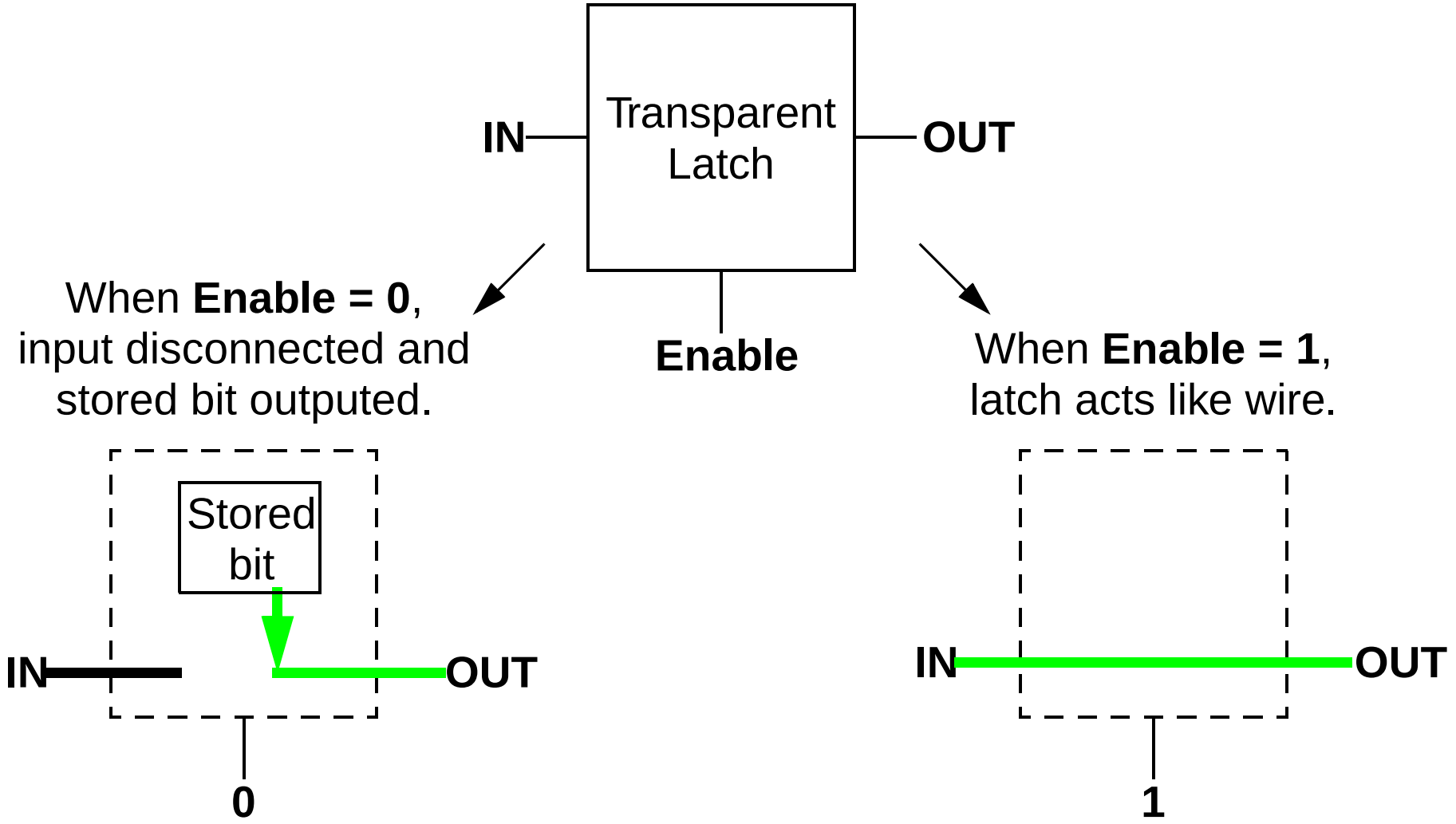




LATCHES

TRANSPARENCY (2)

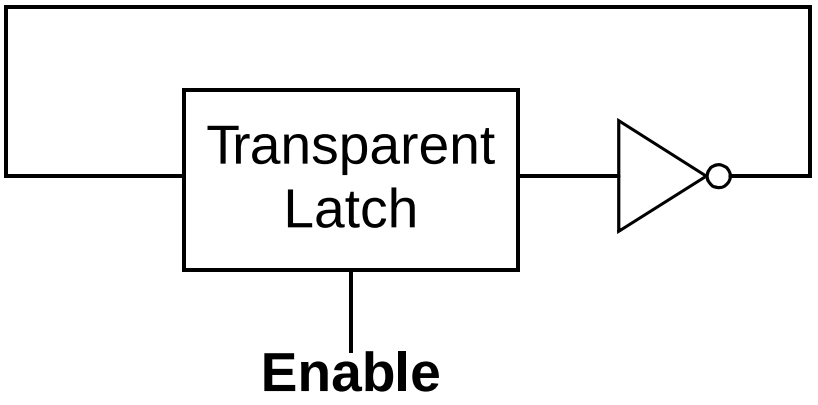
- The following behaviour is observed for Enable = 0 and Enable = 1.



LATCH EXAMPLE

PROBLEMS W/ TRANSPARENCY

- A problem with latches is that they are level-sensitive.
  - A momentary change of input changes the value passed out of the latch.
- This is a problem if the input of a latch depends on the output of the same latch.
  - Example: Design a system that flips a stored bit whenever **Enable** goes high. An inexperienced engineer might design the following.



How will this design behave?

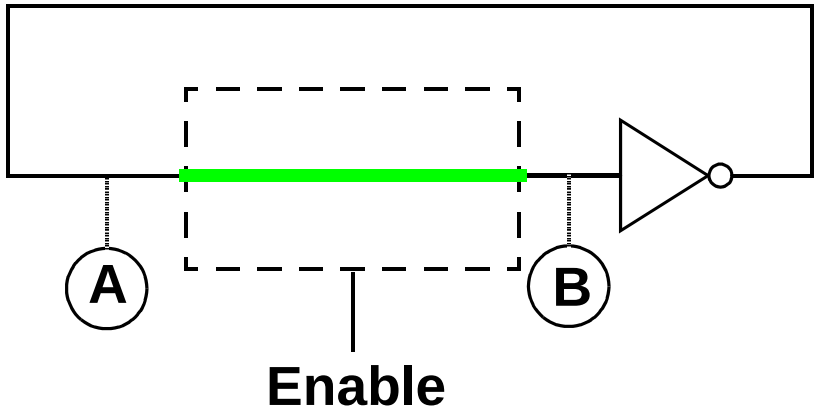
Will the bit flip once when the **Enable** signal goes high?

Answer: The output will follow the input, which happens to keep changing.

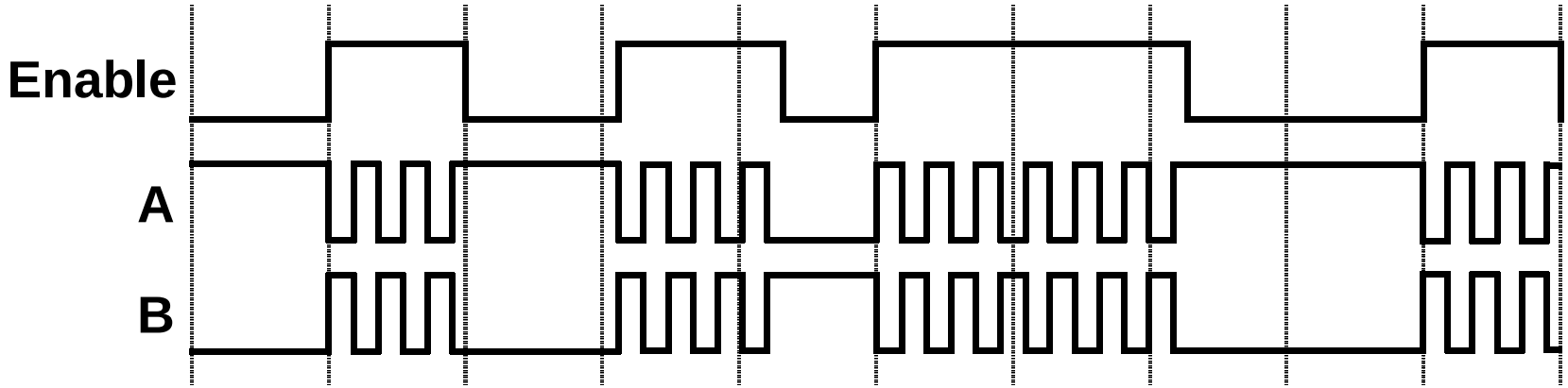
LATCH EXAMPLE

PROBLEMS W/ TRANSPARENCY

- Let's analyze the timing behaviour of this "poor" design.



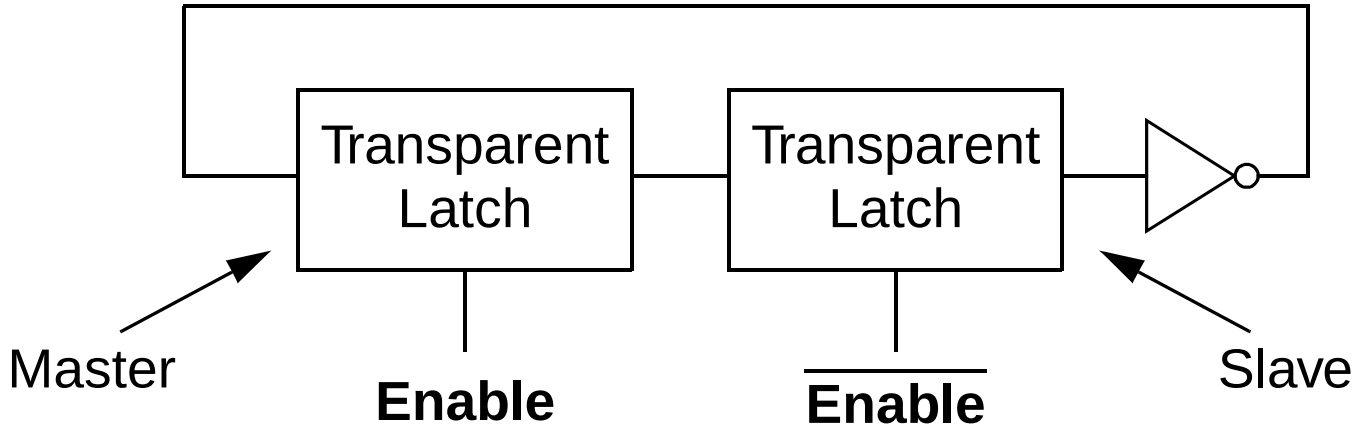
- Notice that instead of the desired bit flip when **Enable=1**, that the input oscillates. This is because the output depends directly on the input since **A** and **B** appear to be connected by a wire.



LATCH EXAMPLE

ELIMINATING TRANSPARENCY

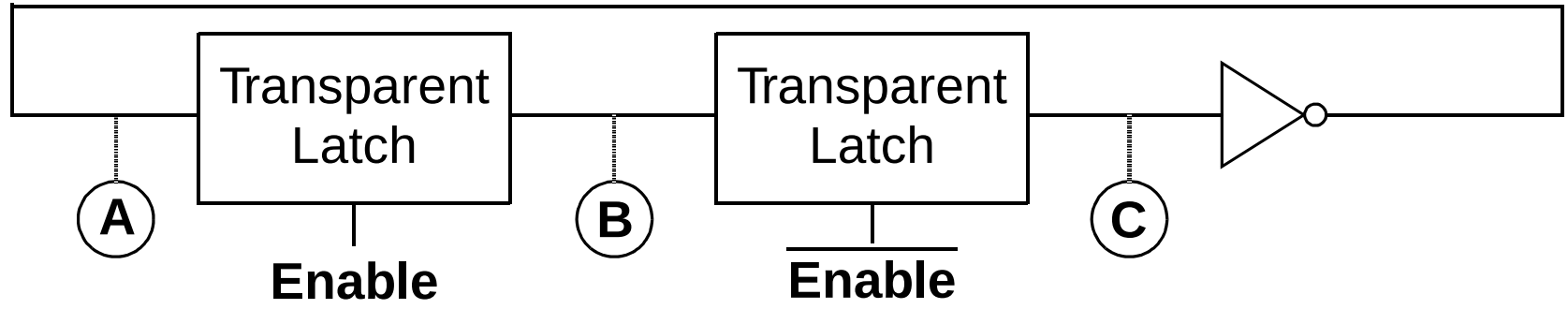
- The problem with transparent, level-sensitive latches can be fixed by splitting the input and output so that they are independent.
  - New solution: Consider the following improved design that flips a stored bit whenever **Enable** goes high. This design now uses a master and a slave transparent latches to separate the input from the output.

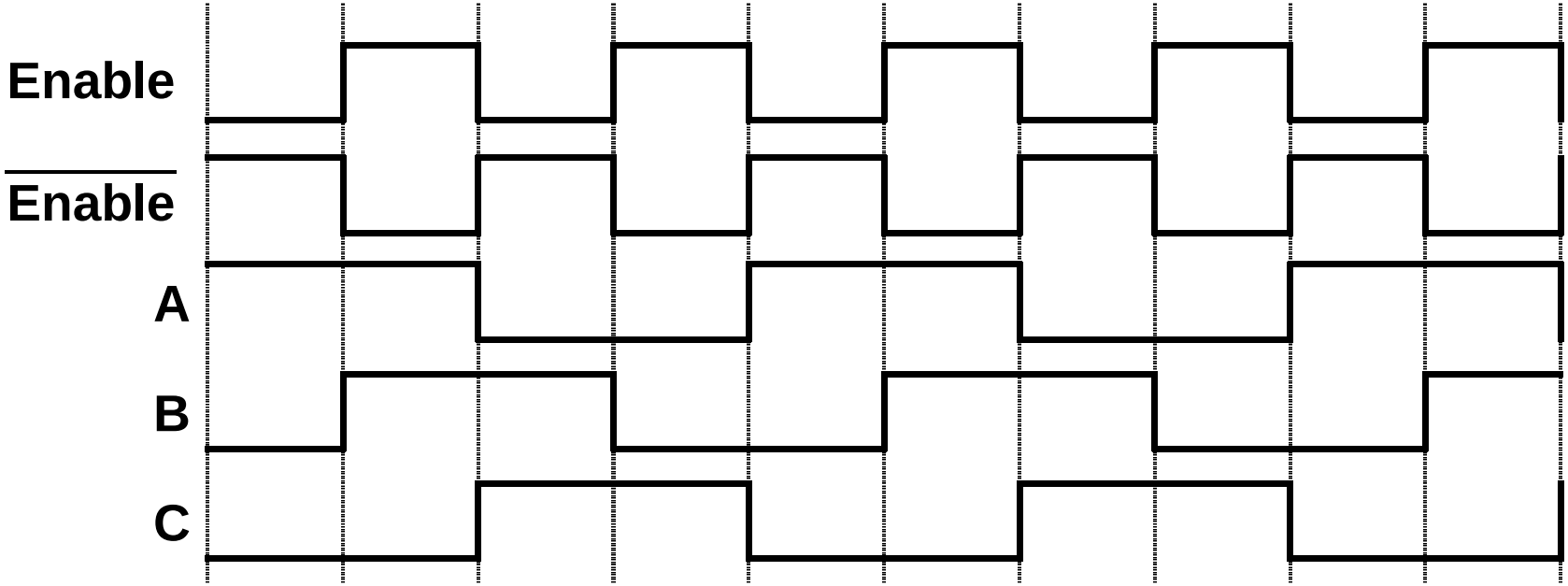


LATCH EXAMPLE

TIMING DIAGRAM

- Let's analyze the timing behaviour of this improved design.



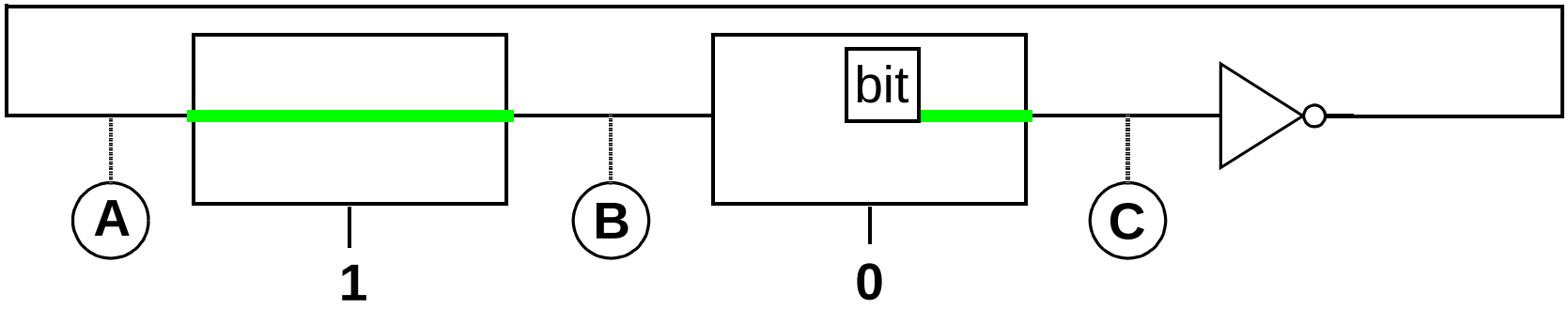


LATCH EXAMPLE

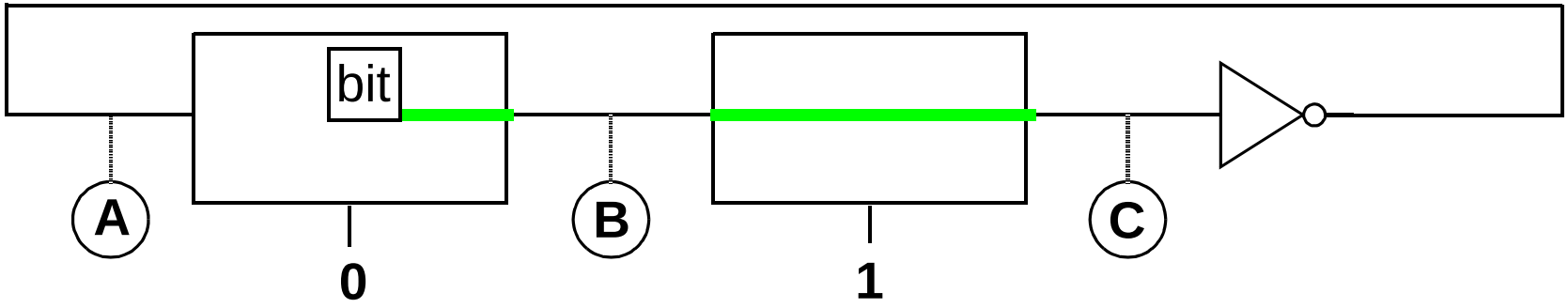
LATCH BEHAVIOUR

- The behaviour of the master and the slave transparent latches can be thought of as follows.

Enable = 1



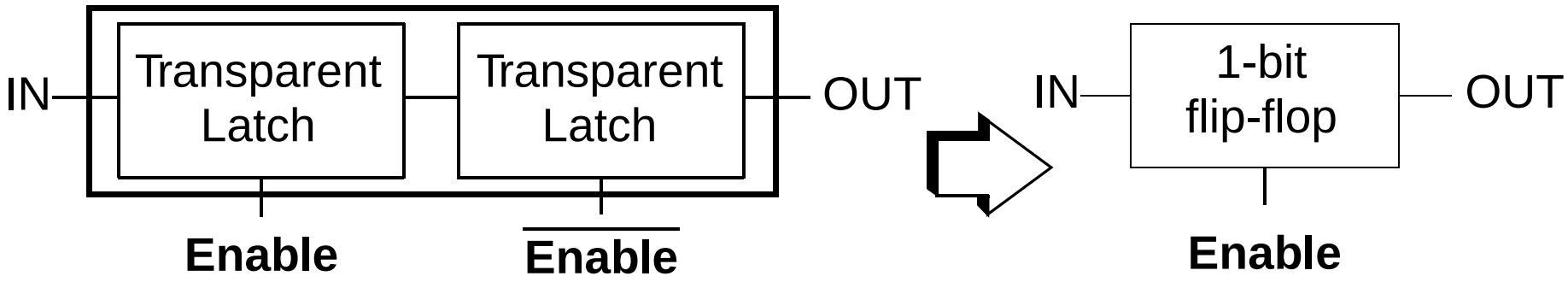
Enable = 0



FLIP-FLOPS

SINGLE BIT STORAGE

- A flip-flop is a single bit storage unit with two stages (master/slave):
  - First stage, or master, to accept input (flip)
  - Second stage, or slave, to give output as received by the first stage (flop)

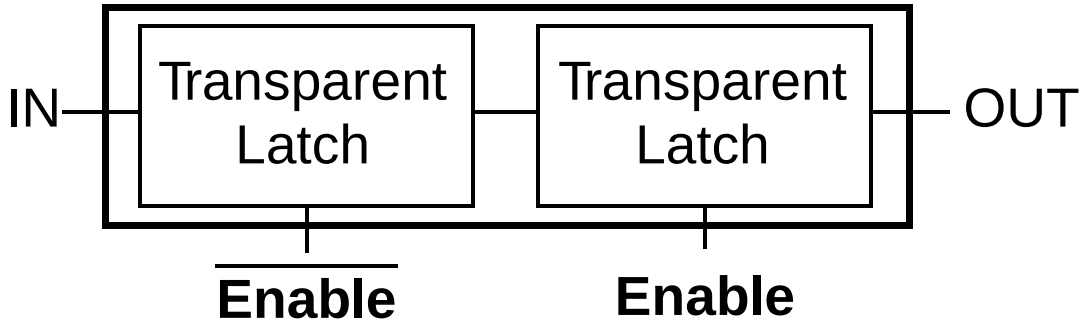


- A number of different types of flip-flops exist such as the SR, $\overline{S}\overline{R}$, D, and JK flip-flops. You may wish to review Chapter 4 regarding these types.

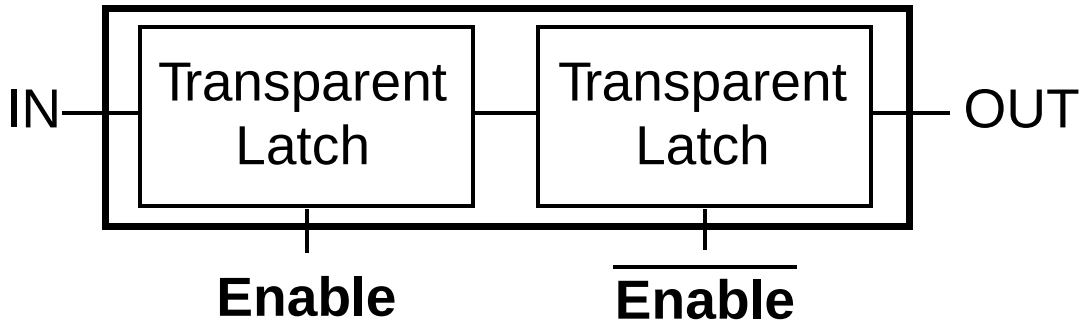
FLIP-FLOPS

EDGE TRIGGERED

- A common and useful type of flip-flop are edge triggered flip-flops.
  - Positive edge triggered flip-flops



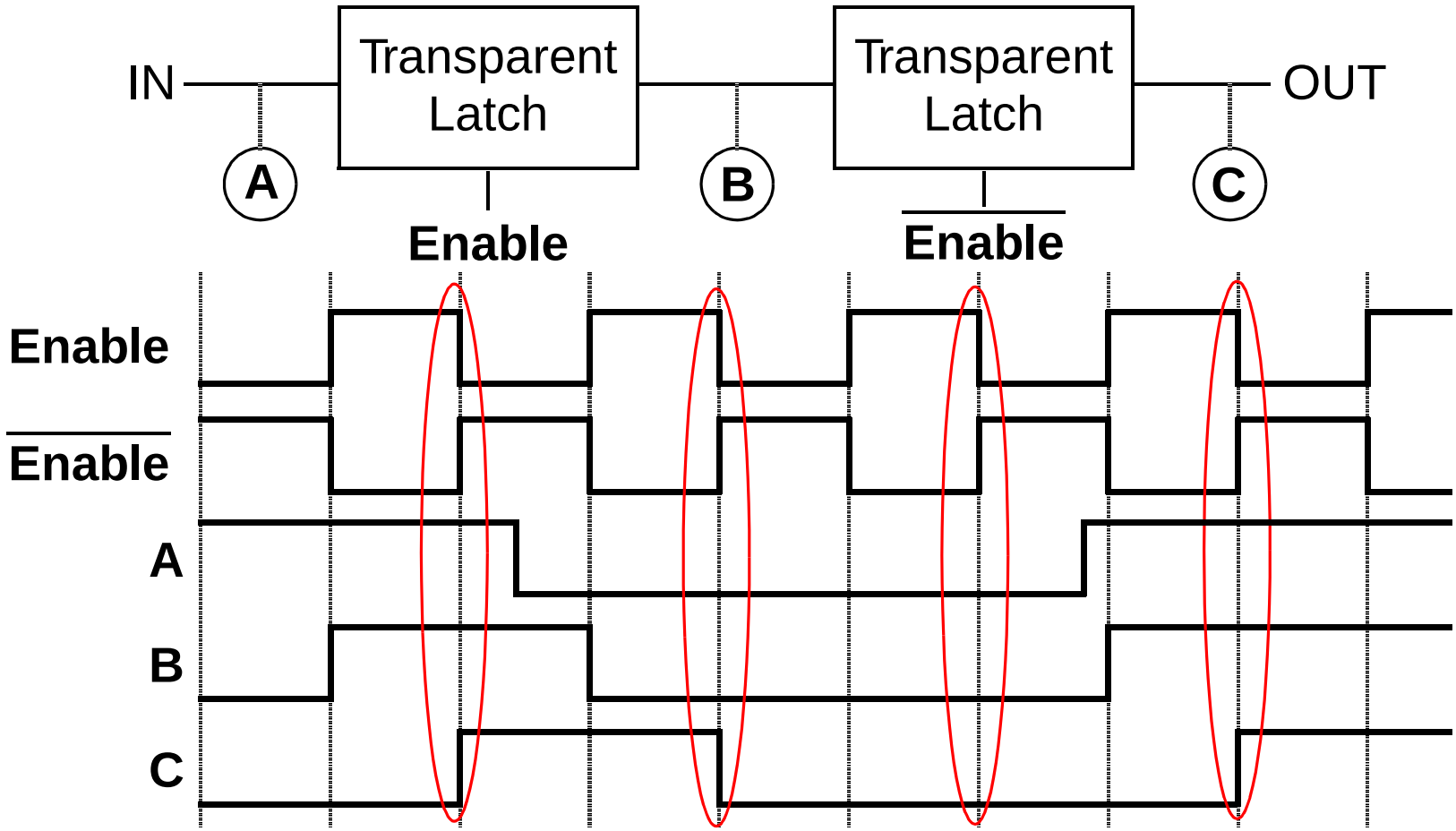
  - Negative edge triggered flip-flops



FLIP-FLOPS

NEGATIVE EDGE TRIGGERED

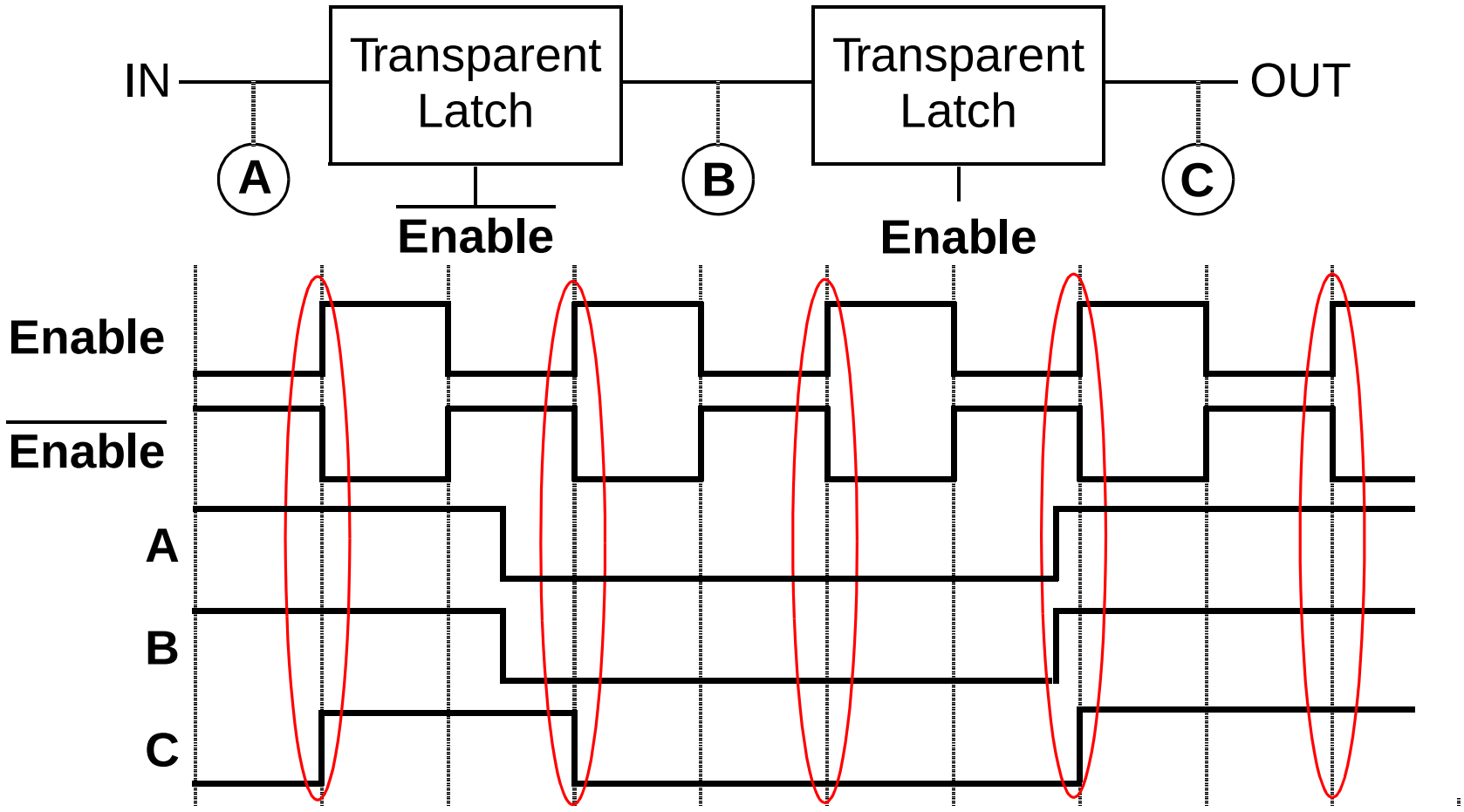
- The output C, which is also the bit stored, appears to change on the negative edge of the Enable transitions.



FLIP-FLOPS

POSITIVE EDGE TRIGGERED

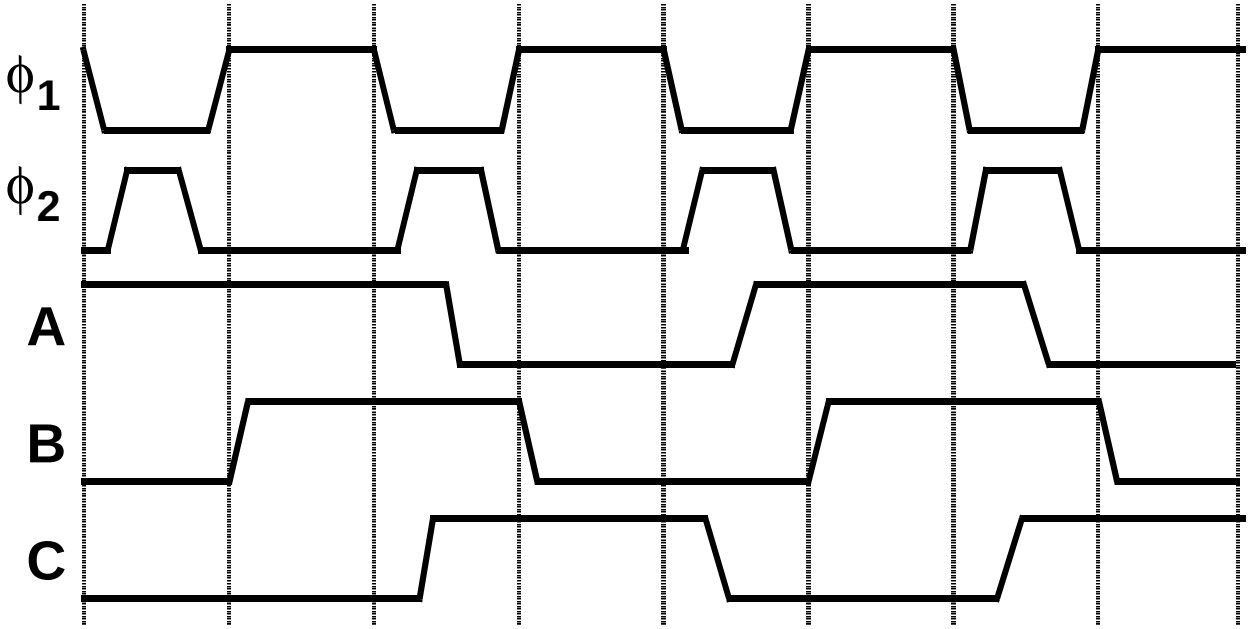
- The output C, which is also the bit stored, appears to change on the positive edge of the Enable transitions.



FLIP-FLOPS

NON-IDEAL W/ DUAL-PHASE

- The previous timing diagrams are in an ideal case. In reality, an implementation with delays might have the following timing diagram.



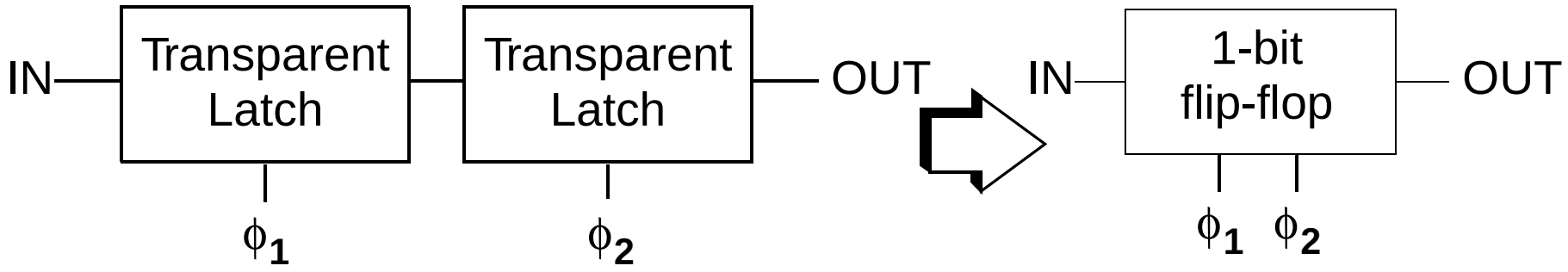
Propagation delays shift the outputs and slew transitions

- Notice that **Enable**/$\overline{\textbf{Enable}}$ are replaced with ϕ_1/ϕ_2, which are **non-overlapping** phases (normally generated from a dual-phase clock).

FLIP-FLOPS

DUAL-PHASE ENABLE

- Why use non-overlapping, dual-phase signals for the latch enable?
  - What happens if the latch enable input flip simultaneously?
  - How about if propagation delays cause one latche to change enable state slightly before the other?
  - The goal is to ensure that the **master latch has latched** the input **before the slave** latch tries takes this bit from the master.

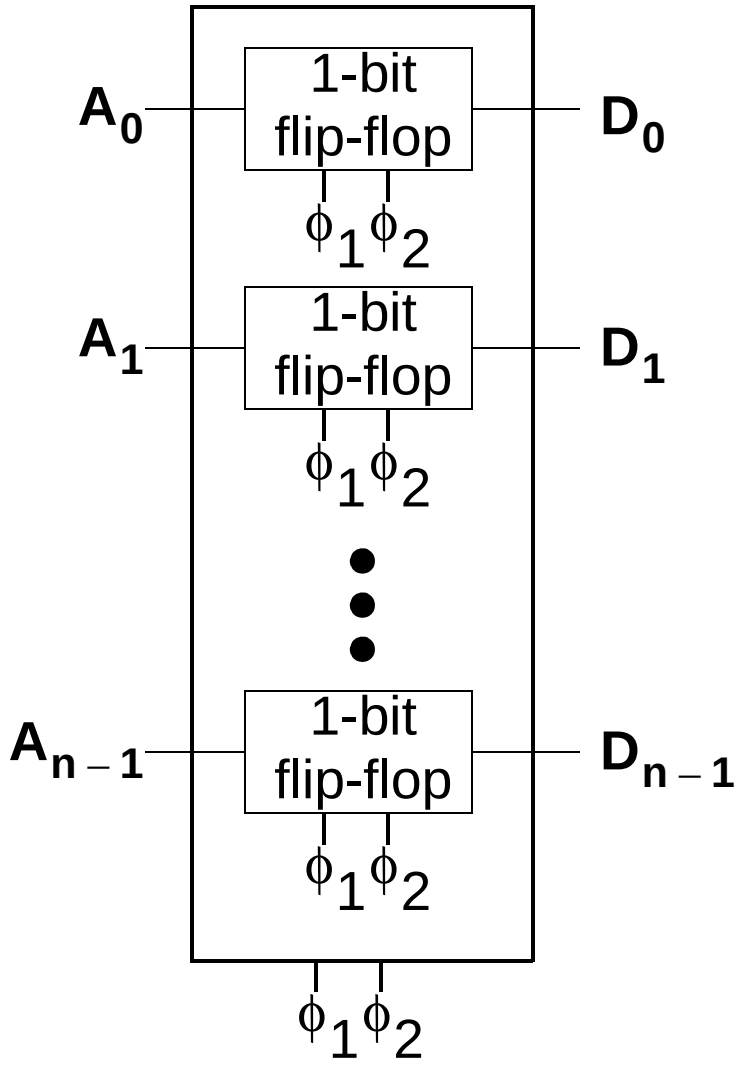


  - If the master has not latched, the slave sees the input transparently!!!
- A non-overlapping, dual-phase enable solves this problem.

REGISTERS

REGISTERS FROM FLIP-FLOPS

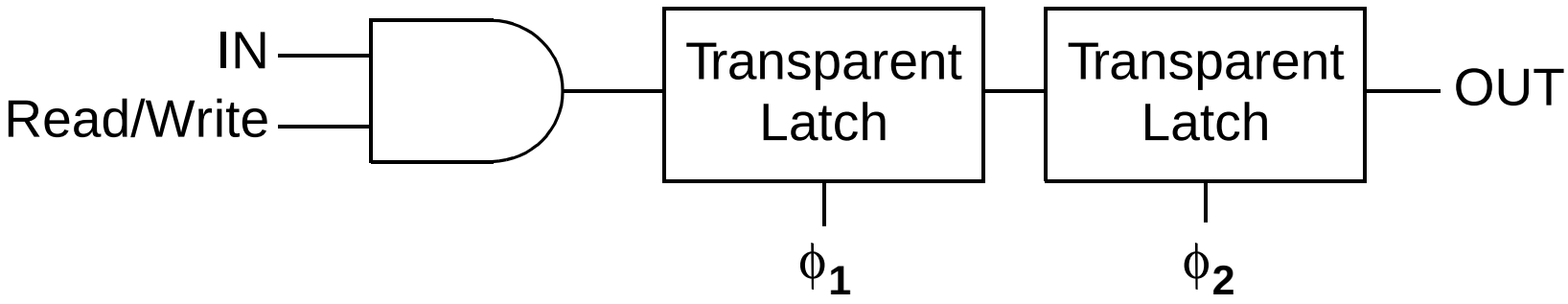
- In essence, a flip-flop is a 1-bit register.
- An n-bit register can be formed by groupign n flip-flops together.



REGISTERS

READ/WRITE CONTROL (1)

- When a clock is used, such as the non-overlapping, dual-phase clock ϕ_1 and ϕ_2, we want control over when a new value is written into a register (instead of writing a new value every clock cycle).
  - A read/write control is therefore required.
  - One “poor” design might be as follows for a 1-bit register.

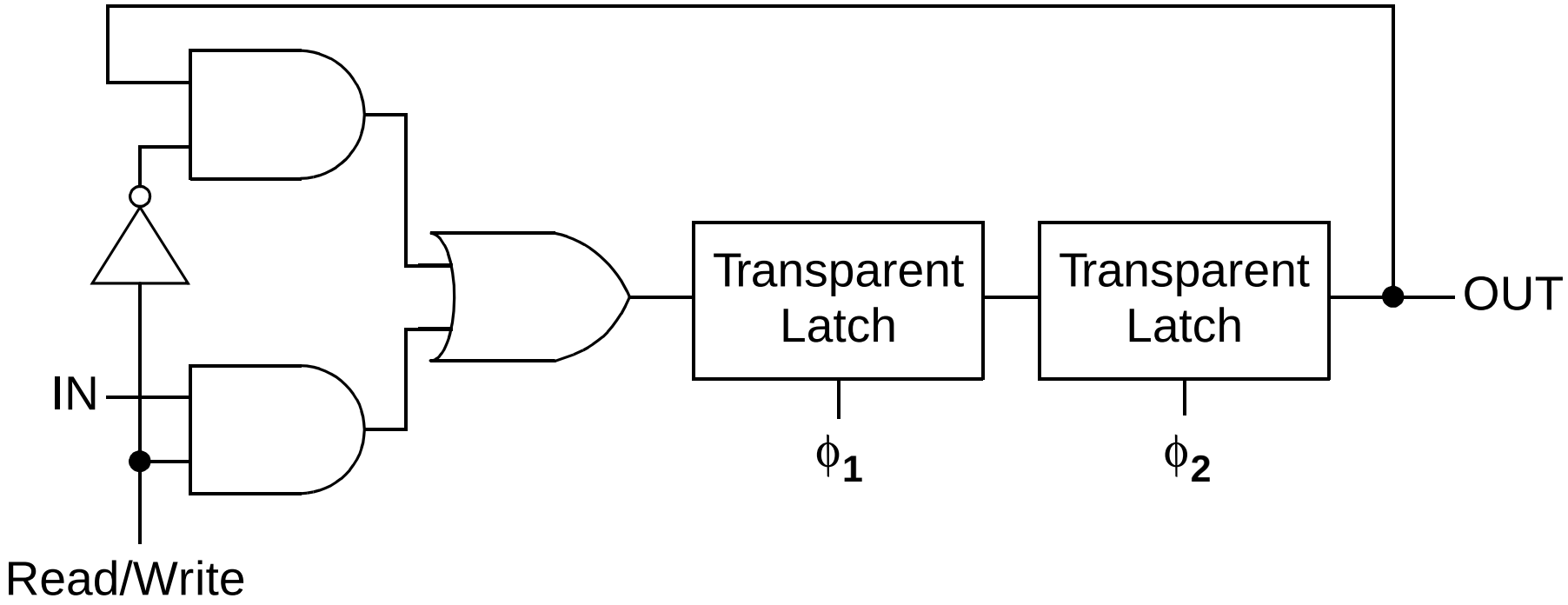


  - What is the problem with this design?

REGISTERS

READ/WRITE CONTROL (2)

- A better design might be as follows

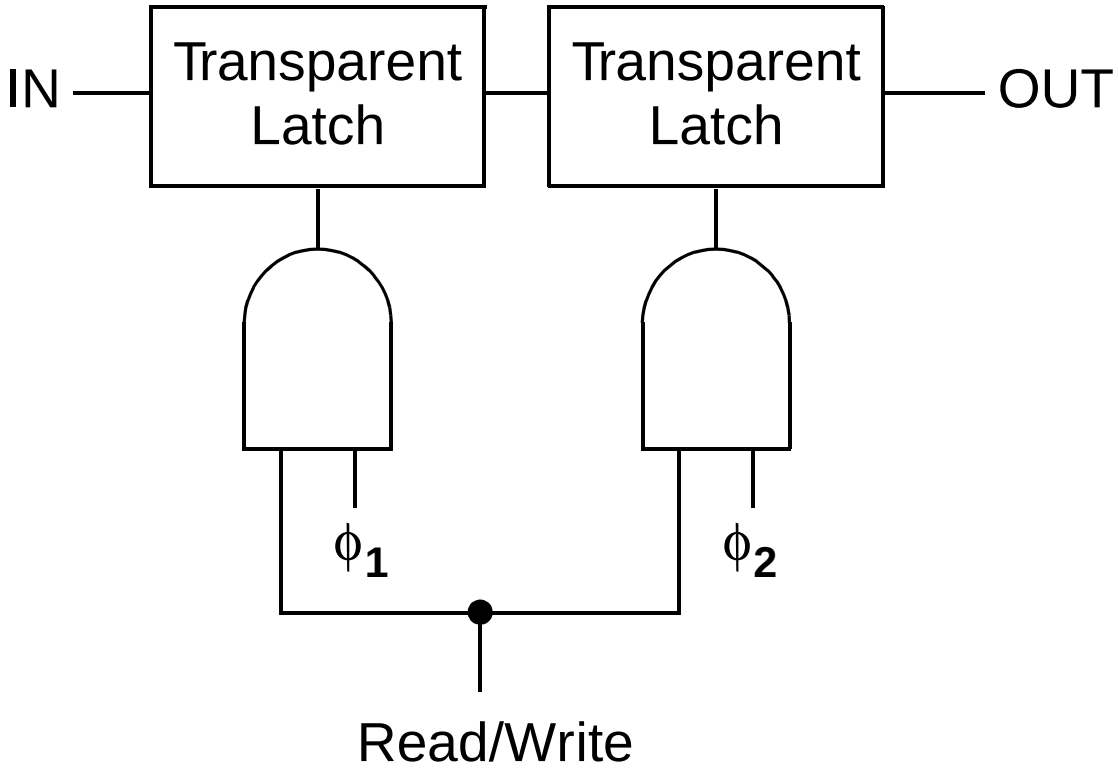


- When Read/Write = **0**, the output is feed back into the master latch.
- When Read/Write = **1**, the input is feed into the master latch.

REGISTERS

READ/WRITE CONTROL (3)

- A different design approach might be as follows



- What problems might exist with this design?
  - One issue might be that both latch enables are **0** when R/W = **0**.