EE-334
Digital System Design

Custom Digital Circuits

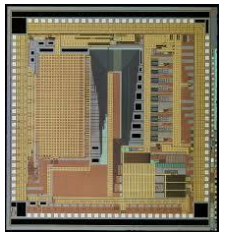
Principles of Synchronous

Register Transfer Level (RTL) Design

Andreas Burg

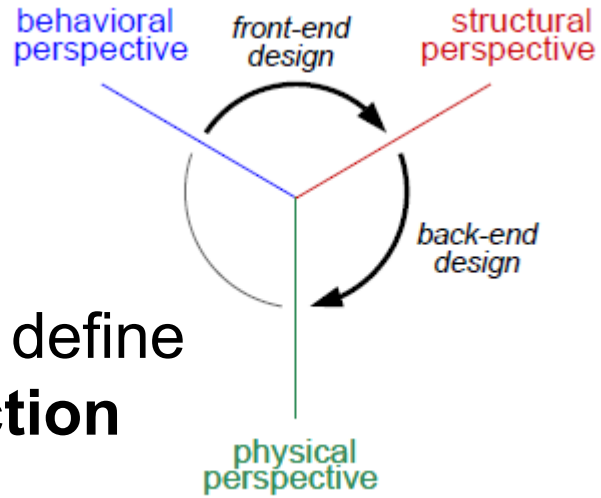
Focus on the Design of **Custom Digital Circuits**

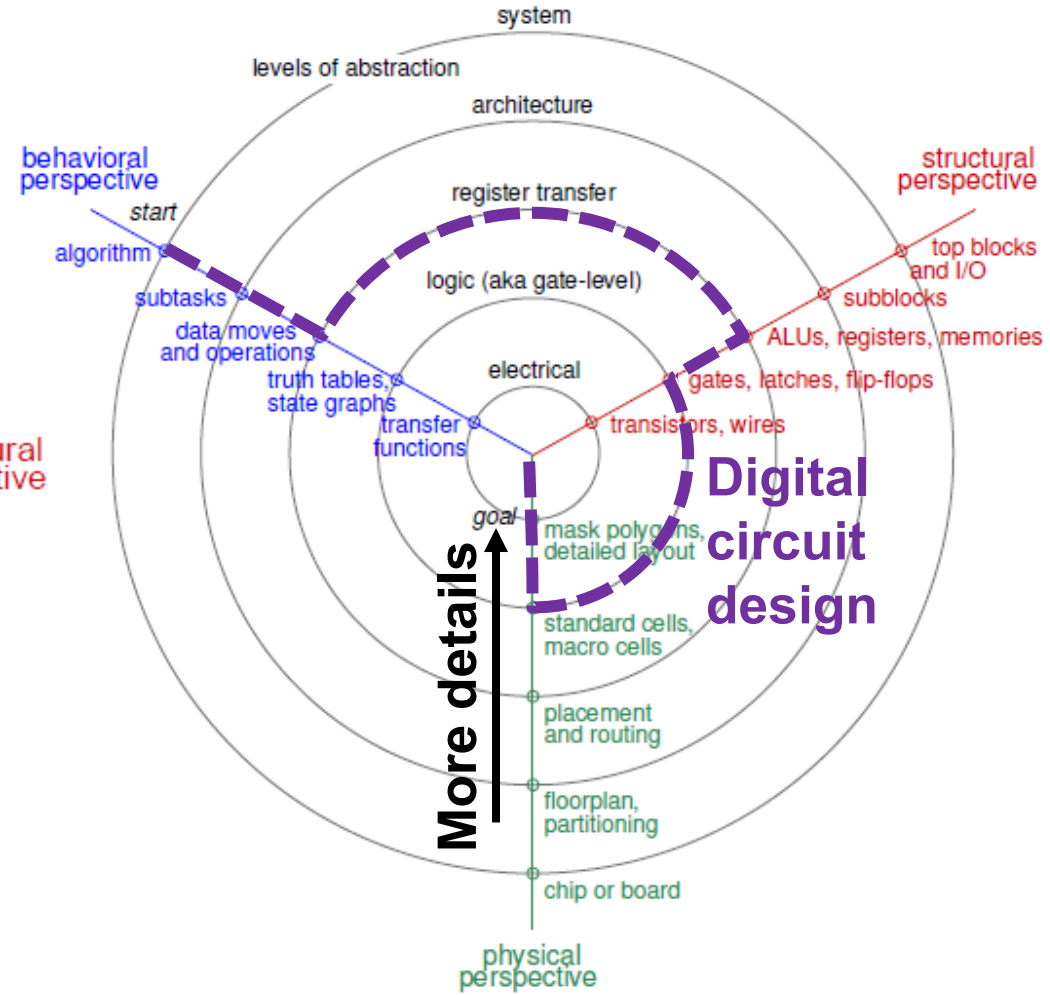
- **Where do we need custom digital circuits** in a digital system?
  - Interface logic between standard components that speak different protocols
  - Dedicated hardware for digital signal processing with high performance or low power
  - Microprocessors that execute software code
- **Implementation options for custom digital circuits**
  - Custom integrated circuits
  - Field Programmable Gate Arrays (FPGAs)



Implementing Algorithms with Registers and Logic

- Design process visualized by the **Y-Diagram** [Gajski and Kuhn,1983]
- Similar to the system design, we can **establish multiple views**, representing different **domains**
  - Behavioral
  - Structural
  - Physical
- **For each domain**, we also define **different levels of abstraction** (amount of visible details)





- **Design process brings us from the behavioral domain to the center of the chart, i.e., a physical circuit with all its details**

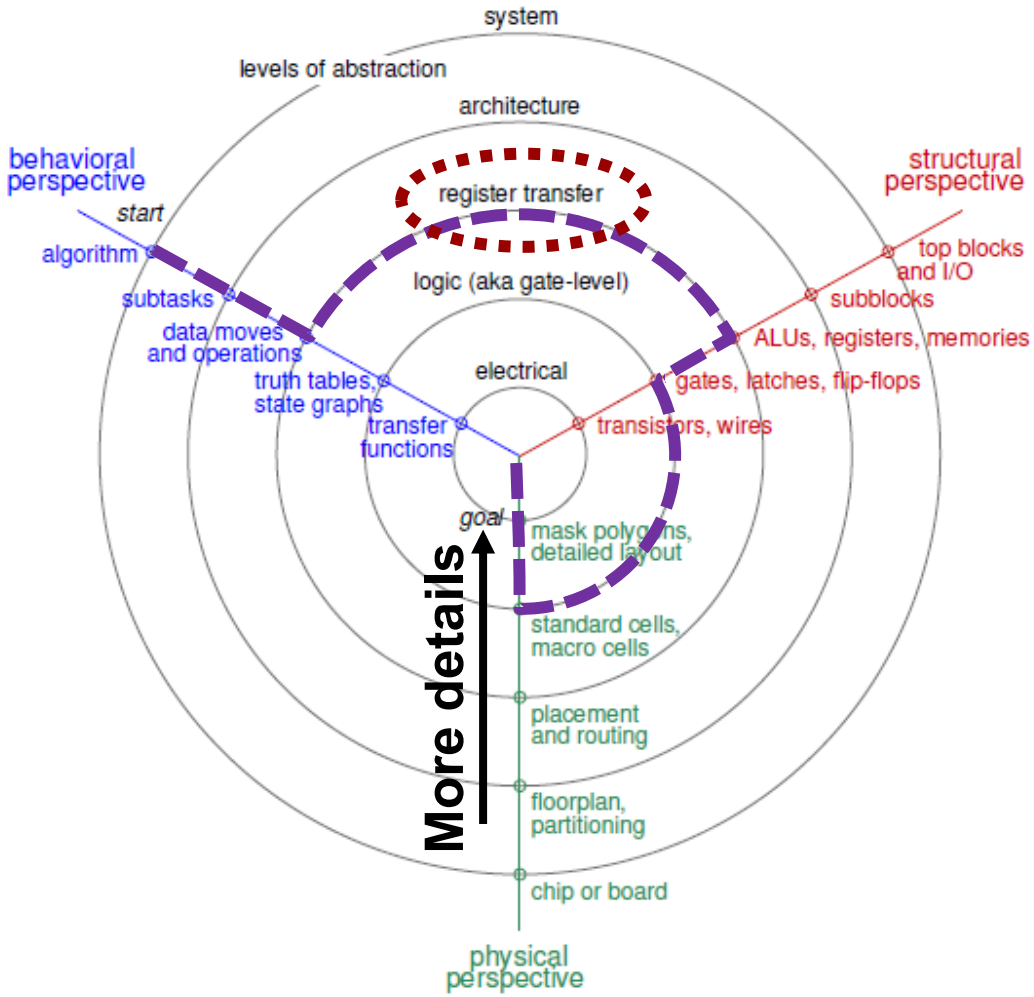
Lets Now Clarify the Title of this Lecture…

- **What is "Synchronous Register Transfer Level (RTL) Design"?**

  - An intermediate representation **between the behavioural and structural perspective**

  - Close enough to the algorithm view to
    - allow convenient mapping of algorithm operations into hardware resources

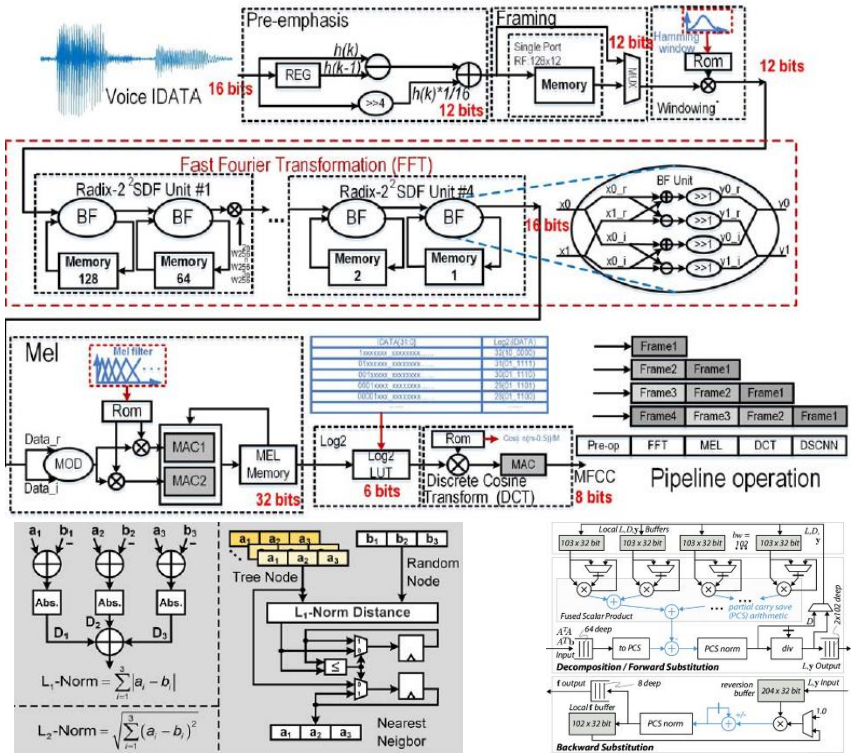
  - Close enough to the structural view to
    - represent complexity tradeoffs and
    - allows to automate the step to more details



Always **Start with** a **Block Diagram**

- **Circuits are** a **concatenation of components** that are **connected by wires**
  - Components receive input signals and generate output signals
  - Wires connect the components (can be hierarchical)
- **Block diagrams are the** natural **way to describe** and discuss **circuits**

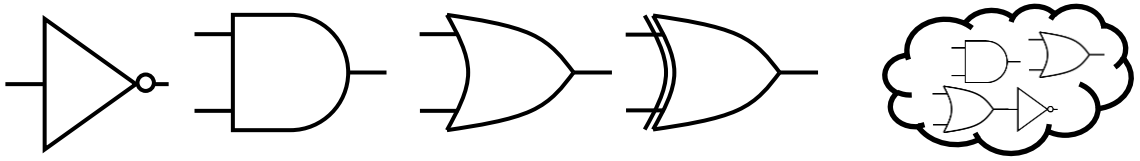
Block Diagram examples from ISSCC papers



- **Block diagrams have many advantages**
  - They show the components you use: **basis for area (complexity) estimate**
  - They are **essential to estimate** the “longest path” and **the maximum** clock **frequency**
  - They show the connections: **basis for estimating communication requirements**
  - The are the starting point for writing **VHDL**
  - They **help to avoid severe mistakes in your VHDL**
  - They are the **best way to discuss circuits**

Fundamental Components of Digital Circuits

- **Combinatorial logic**: memoryless
  - **Built from basic** Boolean **logic gates**
  - **Output is only a function of the** current **input**
  - Combinational logic has no state
- **Nets (wires):** memoryless
  - Connect components, carrying logic signals
  - **One input (driver)** and one or multiple outputs



A	Z
0	1
0	0

NOT

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

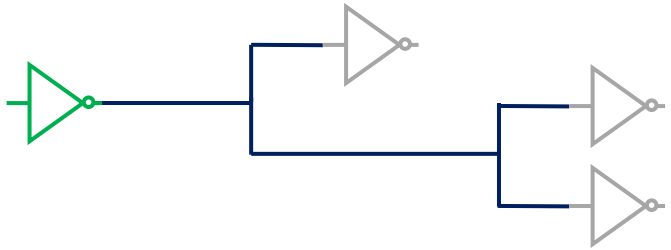
OR

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

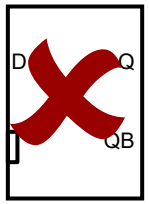
XOR

A	B	Z
.	.	.
.	.	.
.	.	.
.	.	.

Any arbitrary Boolean function

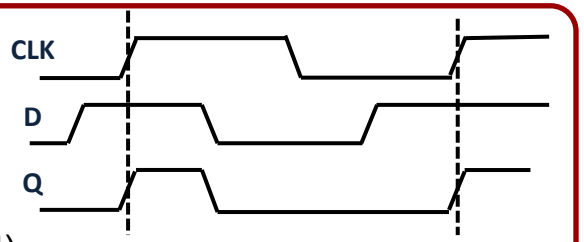


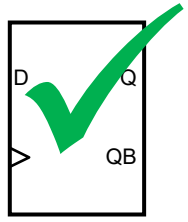
- **Storage** (sequential) **elements**: memory
  - Output depends on an internal state, defined by previous inputs



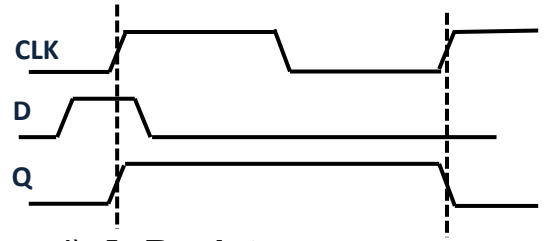
CLK	D	Q	QB
0	-	Q	QB
1	1	1	0
1	0	0	1

Latch
(high transparent)





CLK	D	Q	QB
0	-	Q	QB
1	-	Q	QB
↑	0	0	1
↑	1	1	0

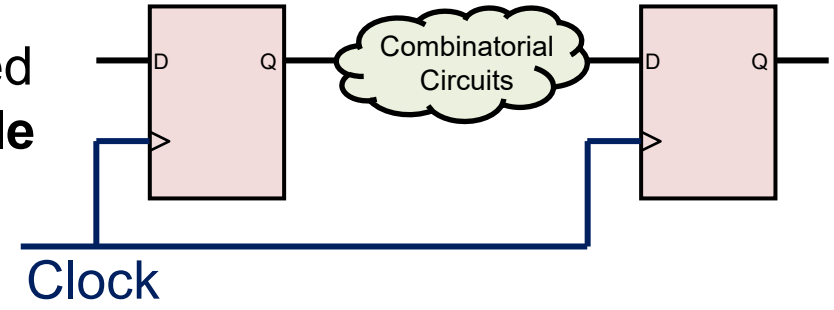


Flip-Flop (positive-edge triggered) ➔ **Register**

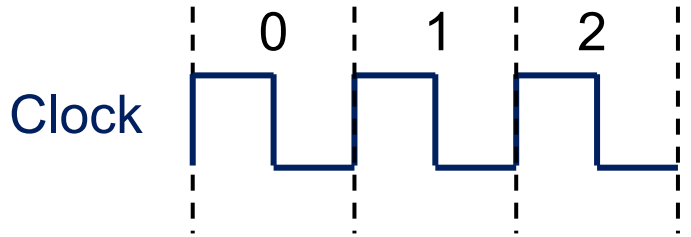
Lets Now Further Clarify the Title of this Lecture…

- **"Synchronous Register Transfer Level (RTL) Design" is both an architecture template and a design methodology**

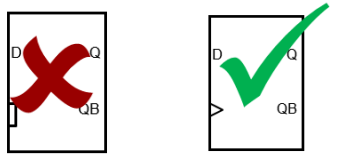
  - **Register Transfer Level (RTL):** refers to how data is processed **Data is modified with combinational logic** or processed **while it is moved between registers** (**for us only Flip-Flops**).



  - **Synchronous Digital Circuits:** refers to the timing
    In synchronous circuits, **all state changes happen instantly at the same time** under the **control of one common clock**.
    - **Common clock provides** the **time-reference** for RTL operations



  - We consider only **Positive Edge Triggered Synchronous Circuits**
    - **State changes** happen instantly **only on** the **positive edge of the clock**

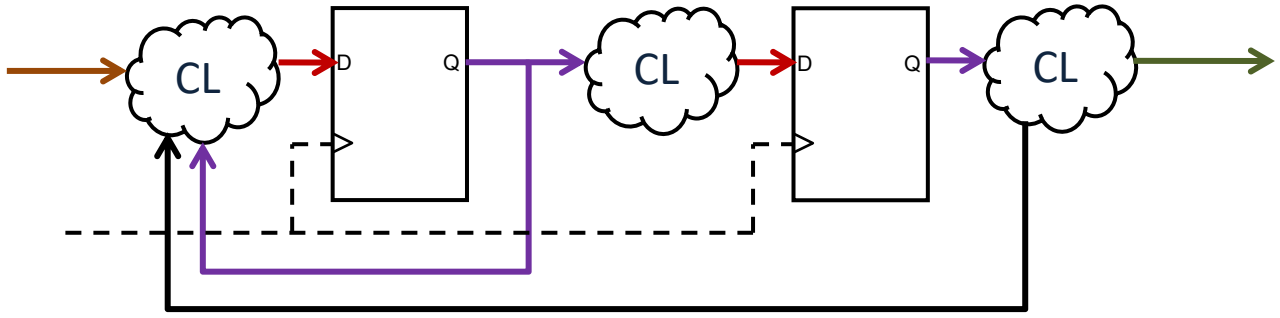


Synchronous RTL Design Follows a Template

- Including Primary inputs and outputs, we arrive at a **generic design template**
  - **Sequential elements** (memory) **keep the present state** (data)
  - **Combinational logic** (CL) defines
    - **Next state**, i.e., content of data registers in the next cycle
    - **Primary outputs**

  based on the inputs and the **present state** (e.g., as boolean or arithmetic expression)

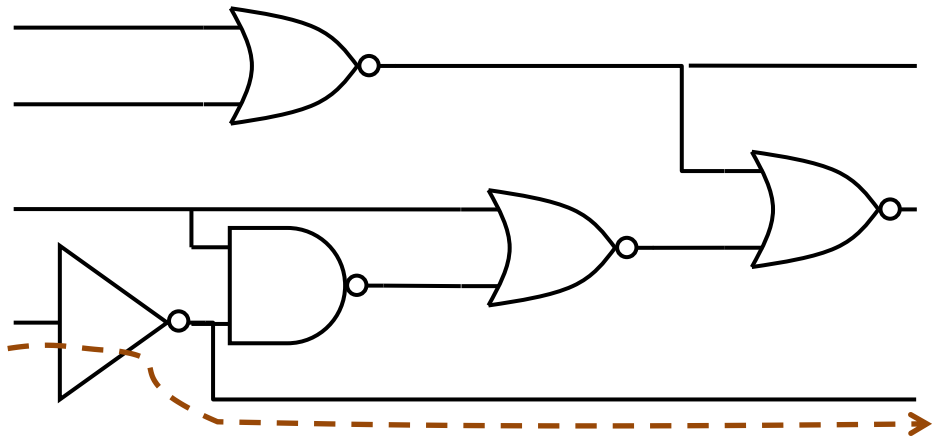
- The state is kept in registers:
  - **Present state**: output (Q) of registers
  - **Next state**: input (D) of registers (stored on next positive clock edge)

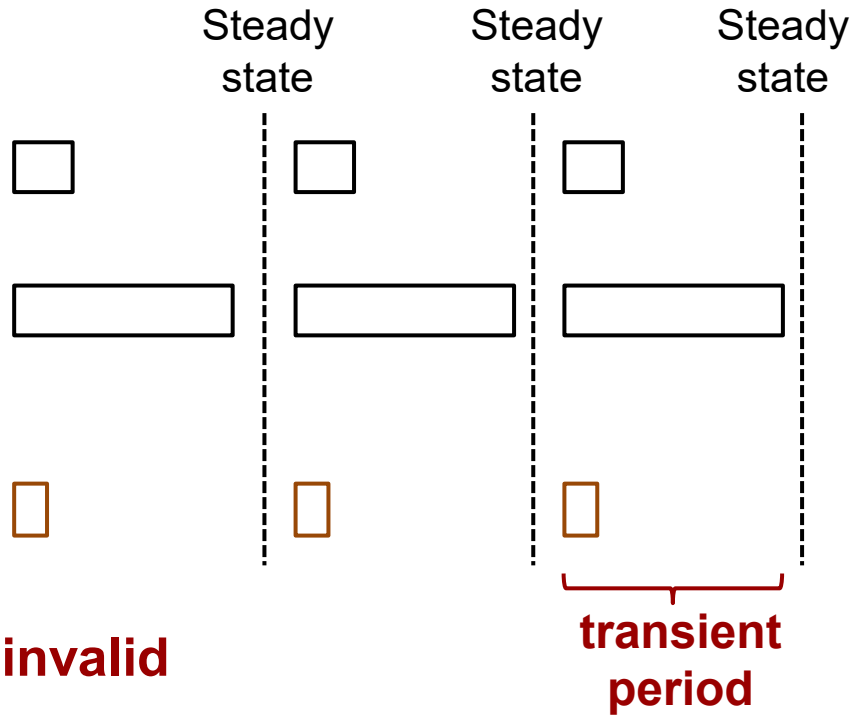


RTL design defines the storage elements and the logic that modifies the data as it is transferred between these storage elements

Sync. Design Abstracts from Component Delays

- **Problem**: real **circuits** (logic gates) **have** (different) **delays**
  - When an input changes, the circuit goes to a transient state
  - Time for a change to propagate from an input to is different for each input and output





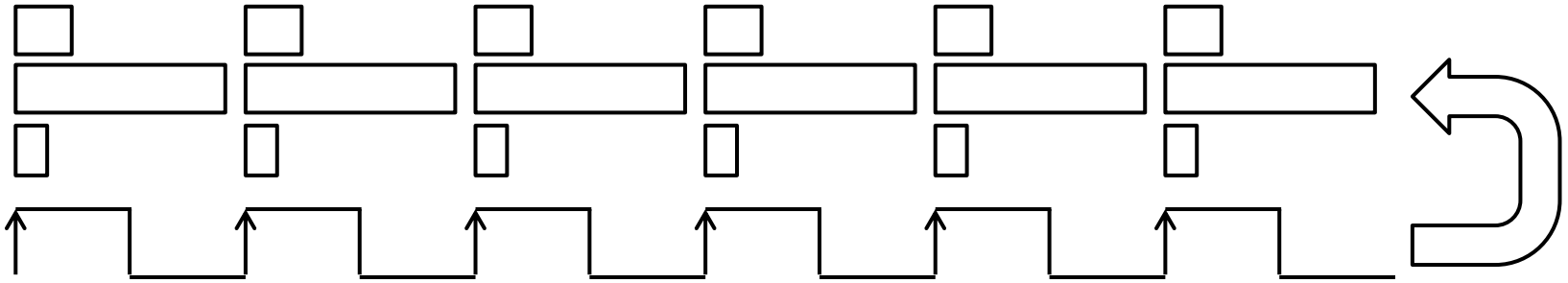
  - During the **transient period**, the **output** of a circuit **is invalid**

- **Solution**: **store new state only** when circuit is back **in steady state**

Synchronous Design: Abstraction to Discrete Time

- **Basic idea**: operation in discrete time steps
  - Start from a coherent (aligned) set of inputs and state variables
  - Wait until all outputs have stabilized (system in steady state)
  - Store the results and new state -> triggers a change in the inputs

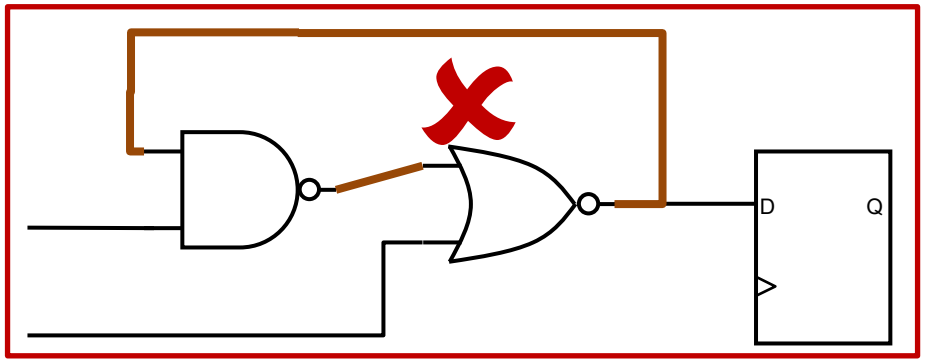
- A clock signal provides the timing reference (heartbeat)
  - **Sufficiently long clock period** ensures correct operation



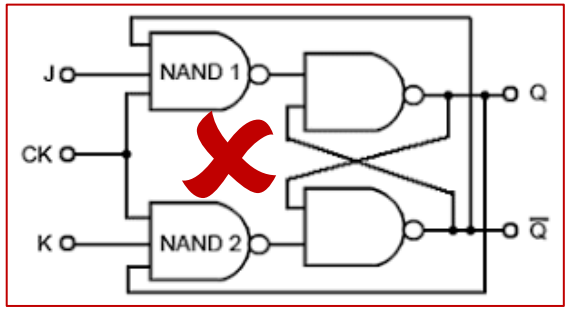
- **Changing the state any other time than the positive clock edge is dangerous**

“Combinational Loops” are Forbidden

- A **combinational loop** exists in your RTL circuit if
  - a **signal path** that goes **through a combinational circuit**
  - **arrives back at** an input of **the same gate**
  - **without passing** “through” **a register**



- Combinational loops do not fit the synchronous design paradigm
  - They have no start and no end
  - They are not synchronized with the clock

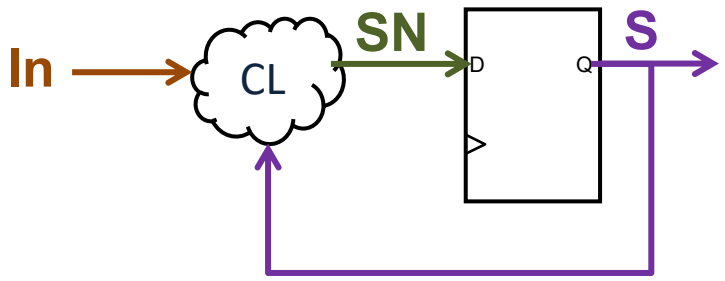


- Hence, it is **difficult (impossible) to guarantee** that they reach **a stable state**

- **Combinatorial loops are not allowed under no circumstances**, even when they are “deactivated”

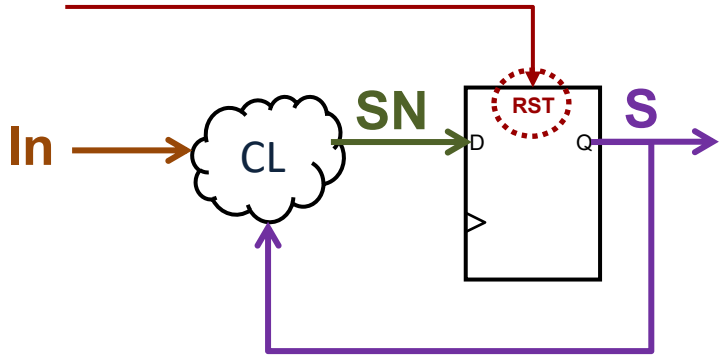
Guaranteeing an Known Initial State

- **At power-up**, the **content/state of storage elements is unknown**
  - Further **evolution of** the **stats** is also **unknown**



Cycle	0	0	0	1	2	3	4
S	'X'	'X'	'X'	'X'	'X'	'X'	'X'
SN	'X'	'X'	'X'	'X'	'X'	'X'	'X'
In	'0'	'0'	'0'	'1'	'0'	'1'	'0'

- **Initial asynchronous RESET** solves the problem



Cycle	0	0	0	1	2	3	4
RST	'0'	'1'	'1'	'0'	'0'	'0'	'0'
S	'X'	'0'	'0'	'1'	'0'	'1'	'0'
SN	'X'	'-'	'0'	'0'	'1'	'1'	'0'
In	'-'	'-'	'0'	'1'	'0'	'1'	'0'

Signal Classes in Synchronous Design

There are three types of signals in a synchronous circuit

- **Asynchronous reset**: brings all bistables (flip-flops), i.e., the entire system, into a known and stable state, independent of their inputs or the clock signal; **they are used only once at power up**

- **Clock**: **only signal that triggers state transitions** and storage of results in memory elements; determines the duration of the work phases and thereby also the speed of the operation (limited by the speed of the combinational elements)

- **Data/control**: represent data and control information in the circuit; they can be used in logic and can be stored in sequential elements (determine the value of state variables), but **they never trigger any state transition by themselves**

Each signal belongs to one and only one of the above categories.

Synchronous Design: The Golden Rule(s)

- **Separate the design of the logic from timing** (see static timing analysis)
  - Clear-cut **separation between** signals that decide
    - **When a state transitions should take place** (clocks)
    - **What data values should be stored**
  - Clock and asynchronous reset signals never participate in logic operations (exception: clock gating, but only if you know what you are doing)
  - **Data/control signals** never trigger a state transition
  - **Combinational Loops** are **not allowed**

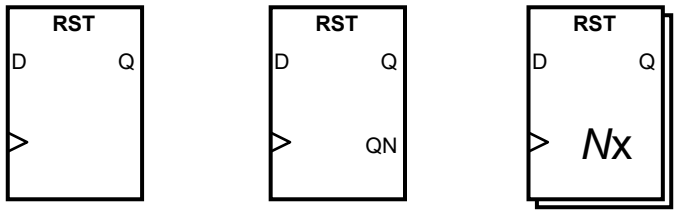
- **Conservative side:**
  - **All storage element**s do **have** an **asynchronous reset**
  - Do not use clock gating unless really needed and if so adhere to its special rules
  - Use **only a single clock** in the entire circuit; if you need multiple clocks, keep them synchronized and use frequencies that are integer multiples of each other

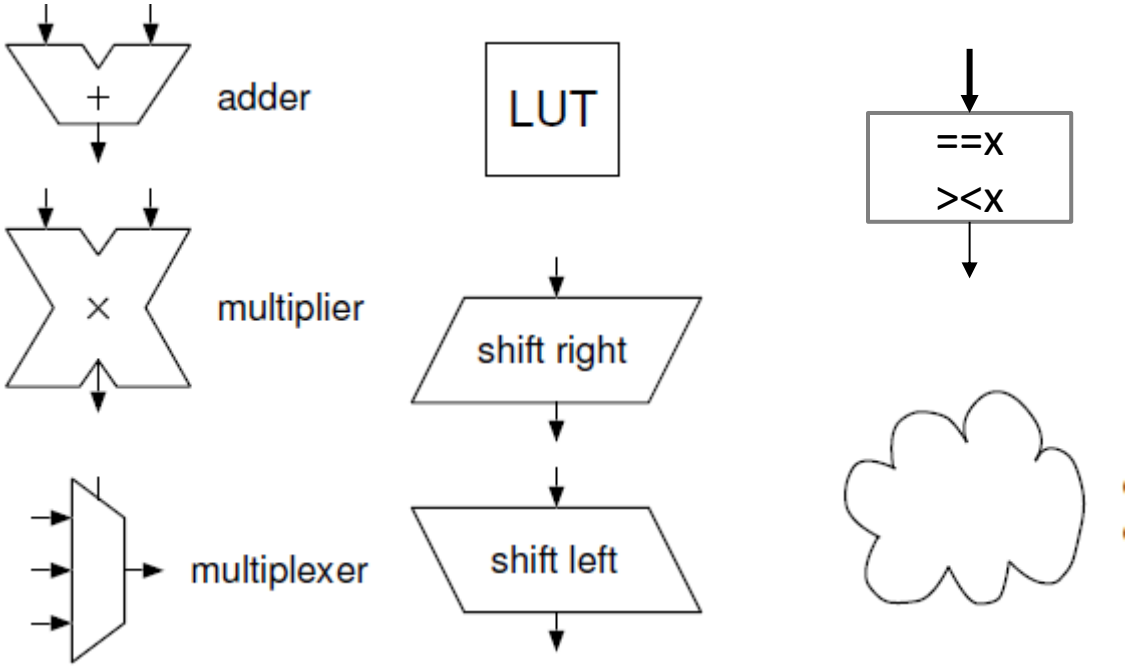
RTL Design Abstraction Summary

- RTL description of a synchronous digital system **fully describes**
  - directly
    - The hardware as **collection of storage elements and combinational logic**
    - completely and unambiguously **all hardware resources and their connections**
  - indirectly (not visible in the block diagram, but deterministic given by the RTL design)
    - the **discrete-time behaviour** of a synchronous circuit
- But is **does not contain any information on**
  - The delay of the combinational logic or the timing requirements of sequential elements
- **RTL design is done with block diagrams!**
  - **Golden rule**: if you can not draw it, something is wrong!

Common Complex Components

- Components are not limited to basic logic gates and single-bit Flip-Flops
  - RTL Synthesis allows to describe combinational logic in a compact way
  - Not all details (e.g., internals of an adder) need always to be worked out
  - Multi-bit signals (e.g., busses) can typically be collapsed into one signal

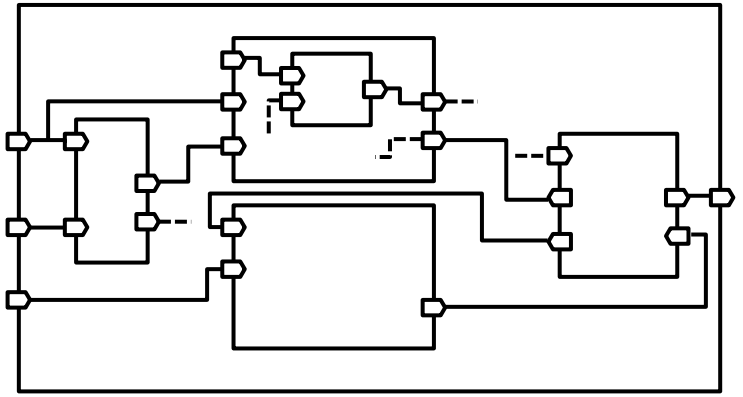




- Choose the **level of detail based on**
  - what you **want to show**
  - what is **necessary** to show to **understand** the circuit

Managing Structural Complexity with Hierarchy

- **Most systems** are too complex and too heterogeneous to be considered as one big piece and **must be broken up into sub-circuits** (divide-et-impera)
  - Partition design into functional blocks (*instances of design entities*)
  - Specify how these blocks communicate with each other (interfaces: signals and protocols)
  - Define clear and *informative* names for all blocks, signals, and ports and annotate them in the block diagram

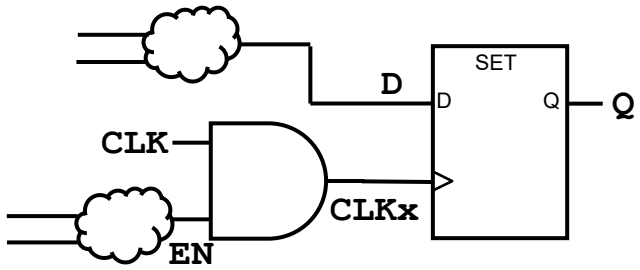


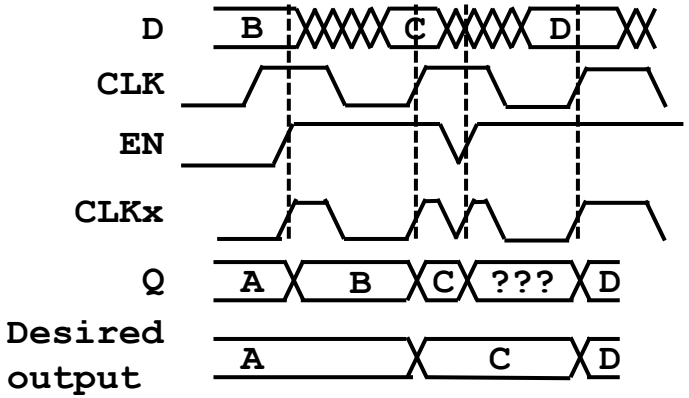
- Design Hierarchy: recursive partitioning of design entities into smaller units
  - Hierarchy is useful for:
    - functional partitioning
    - hiding details
    - reducing complexity
    - re-use of components

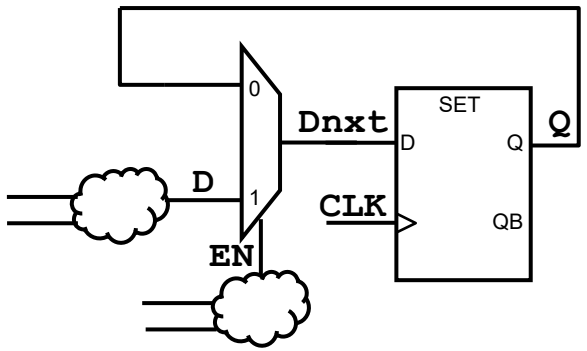
- **Top-down design: start from top**
  - **Partitioning of a (sub-)unit into further/smaller sub-units**
  - **Refinement: implementation of sub-units**
- **Bottom-up design: start from bottom**
  - **Assemble pieces to build up a larger design/unit**

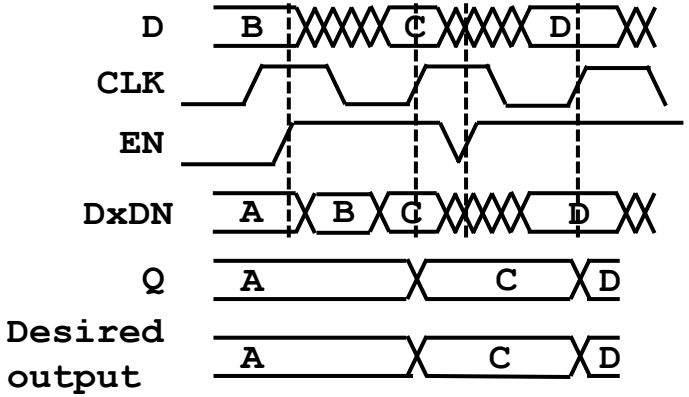
Synchronous Design: A Famous Example

- Flip-flop with enable



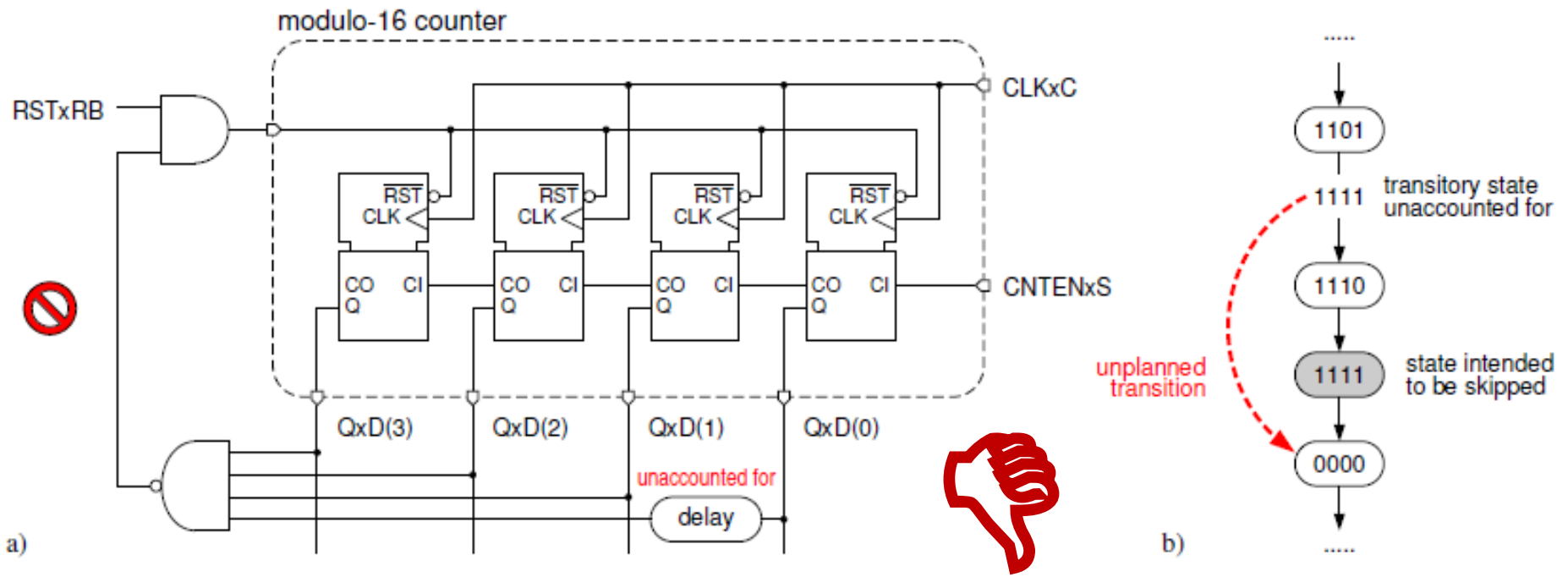






Synchronous Design: Another Famous Example

- **Counter with RESET**:
  - 4-bit counter that skips “1111” and return immediately to “0000”

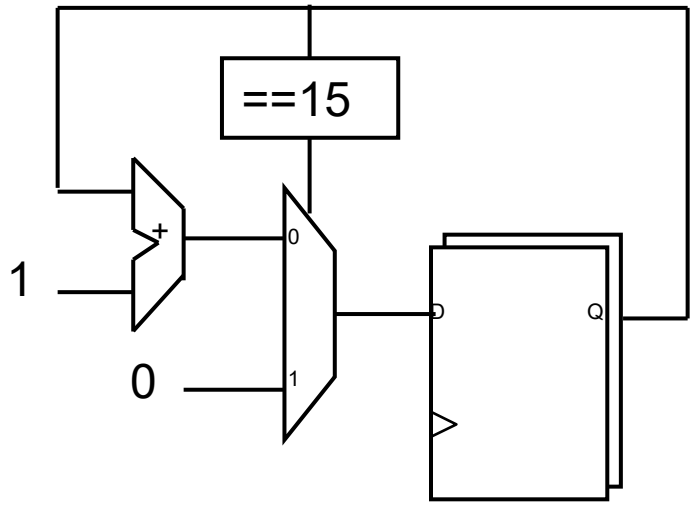


- **How would you realize this in a fully synchronous design style??**

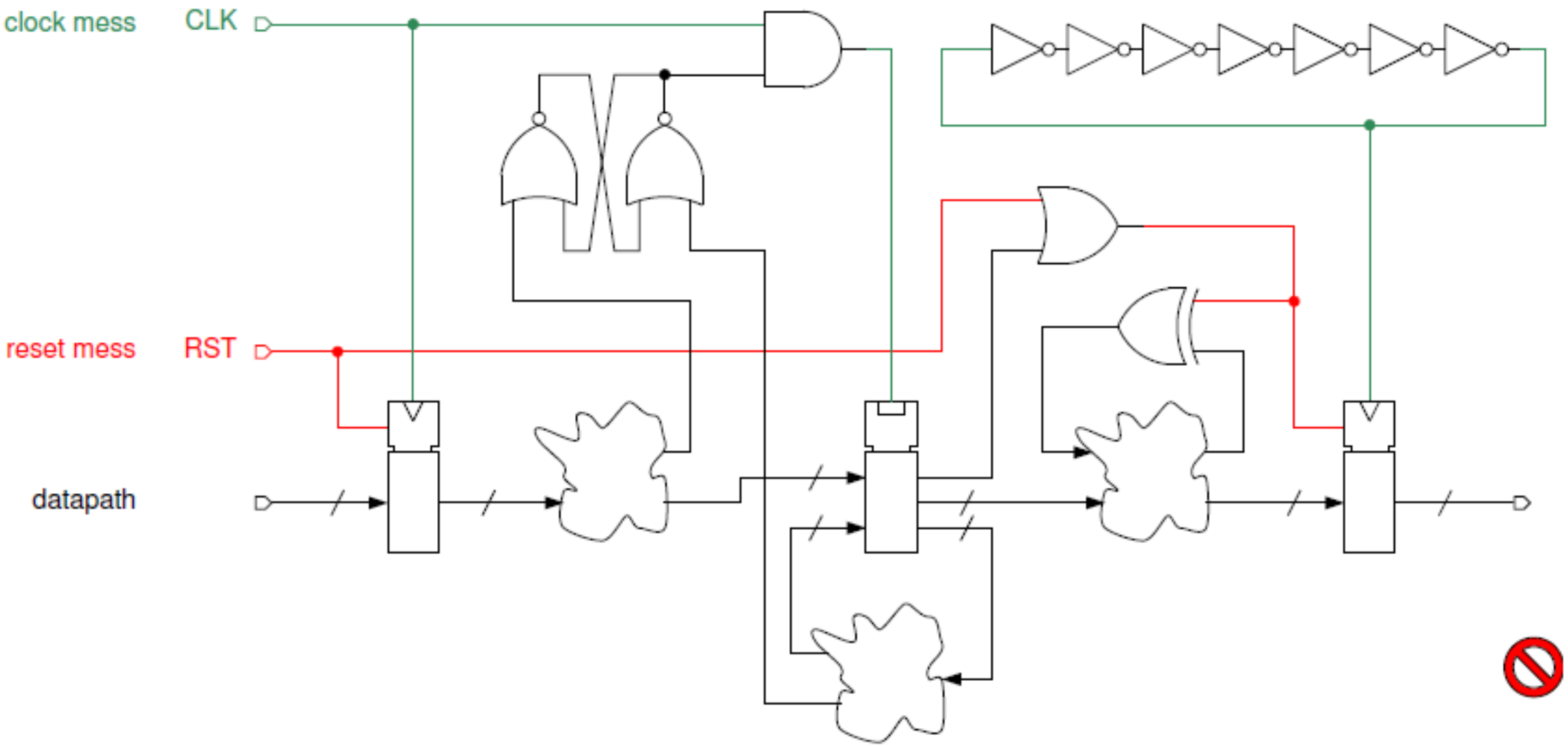
RTL Design Example

- Counter
  - Count from 0 to 15 and wrap around to 0

- What do you need:
  - 16 possible states (0-15) => 4 bits to store the state => 4 registers
  - Adder to increment the counter by 1 in each clock cycle
  - Some means to recognize when to add and when to reset
  - Some means to reset the state(counter)

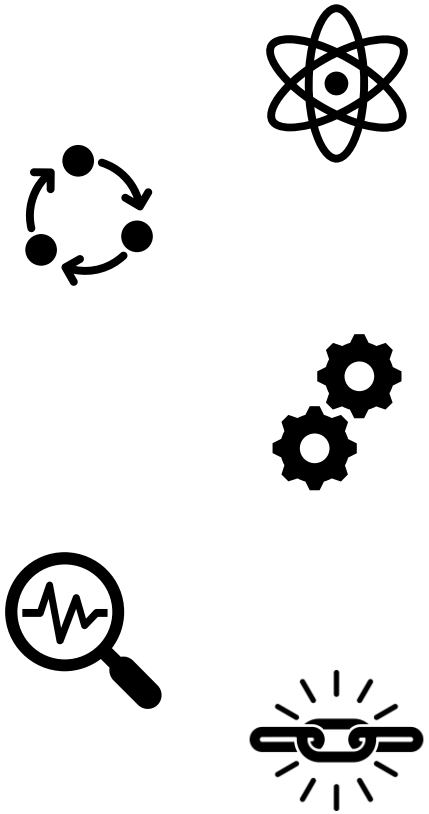


Find the Design Issues



Pros and Cons of Synchronous RTL Design

- **"Synchronous RTL Design" is the industry standard design style** for digital integrated circuits (and FPGAs)
- **Advantages**:
  - **Easy to abstract** complex circuits with **simple rules**
  - **Well defined design methodology** with very few ingredients
  - **Compatible with electronic design automation** (EDA) **tools** to translate an abstract description into a circuit with all the details
  - Circuits are relatively **easy to verify and debug**
  - Provides a very **high level of robustness and reliability**



Pros and Cons of Synchronous RTL Design

- **"Synchronous RTL Design" is the industry standard design style** for digital integrated circuits (and FPGAs)
- **Disadvantages and dangers**:
  - Simple, but (at first sight) very **restrictive rules**
  - **Essential to strictly adhere to the rules** at all costs,

    even though it is sometimes tempting to break them
  - **Verification fails to detect issues when rules are broken**
  - **Breaking the rules** even once **jeopardizes the** entire **design**
  - **Significant differences to programming** software

! DO NOT BREAK THE RULES