**3 Types of synthesizable logic**

* Purely combinational
  + Every input must be in the sensitivity list of use always\_comb
  + Every output must be assigned a value for every possible combination of the inputs
    - Use defaults and no dangling ifs
* Sequential
  + Outputs only change at the clock edge (synchronously)
    - always\_ff @ (pos/negedge clk)
  + Allows for SYNC resets but not ASYNC
  + Rule 1: only the clk should be in the sensitivity list
  + Rule 2: only signals that change on the same edge of the clock can be included
* Sequential with async reset
  + Both reset and clk in the sensitivity list

**Types of assignments**

* Blocking (=)
  + Evaluation and update RHS immediately
  + Used for assignments to registers (FFs) in sequential blocks
* Non-blocking (<=)
  + Evaluation deferred until all RHS have be eval'd
  + Used assignments to registers in combinatorial blocks
* Do not mix assignment types for the same signal

**Datapaths**

* Explicit
  + Next state logic and output logic separated
* + Combined state transition and output logic

**Buffers and tri-state drivers**

* Tri-state: 1,0, z (high impedance)

**RTL to hardware**

* RTL → synthesis → placement → routing → timing analysis → bitstream (.sof)

**FPGA architecture**

* Array logic blocks: LUTs, ff, mux
* Outer perimeter I/O blocks

**Midterm questions:**

* Xor : 1 for odd numbers

**Timing**

Gate delay

* Parasitic capacitance of a transistors
* No current flows through the gate of a transistor
* Delay for a gate is the time of the logic gate to charge its associated parasitic capacitor on its output
* Charge time = R\*C Resistance across the source to drain of the gate\*parasitic capacitance
* Delay of n gates = n\*R\*C → we have control over n
* R dependent on the size of the logic gate (large = higher R) and the length of wire connecting the gates (think of resistivity)
* C dependent on the size of the logic gate, the number of gates driven, and the length of wire (wires have little capacitance/unit length but still)
  + Driving multiple gates:
    - The parasitic capacitance will be in parallel with the driven gate capacitances and therefore are summed together

Sources of delay

* Logic gate delay: on the order of 0.1ns
* Routing delay: around the order of 1 ns

Modelling delay

* Usually lump together routing and logic delays

Concepts

* Combinational timing constraints
  + Gate propagation: time for a comb gate output to change when the inputs change
    - Path delay: delay through a series of comb gates
    - Use ffs to synchronize the release of the input and then an ff to synchronize the outputs as well
  + Critical path: determines the min clock period → t\_clkmin >= t\_critical, fmax = 1/tclkmin
* FF timing contraints
  + Clock to Q: time for the output of a ff to settle after a posedge occurs
  + Setup time: input to a ff cannot change for a certain amount of time PRIOR to the edge
  + Hold time: input to a ff cannot change for a certain amount of time AFTER the edge

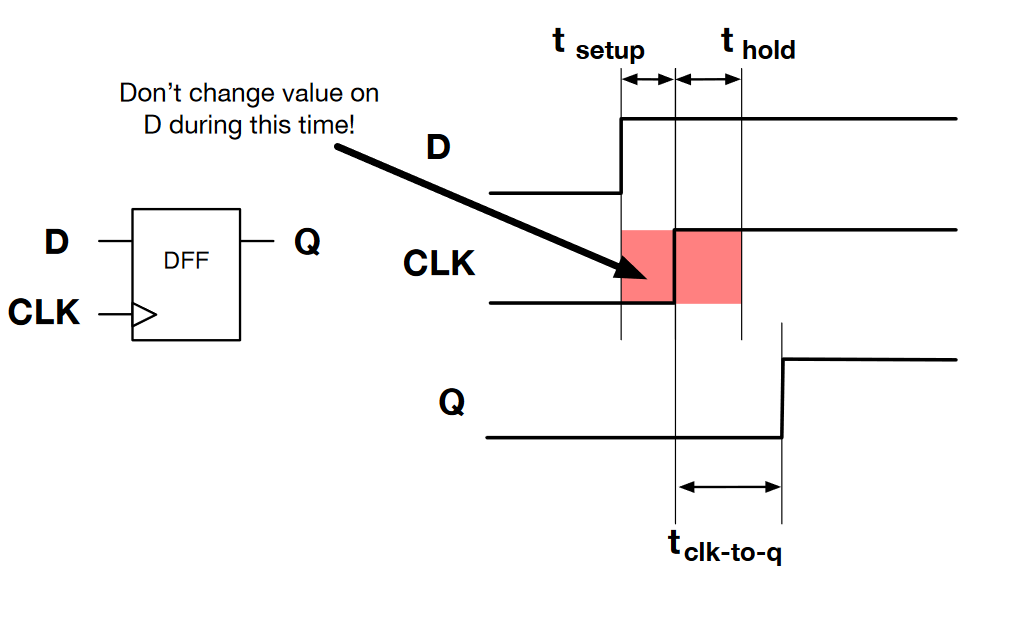
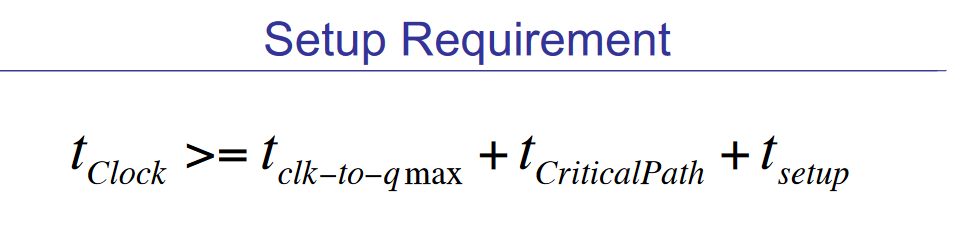
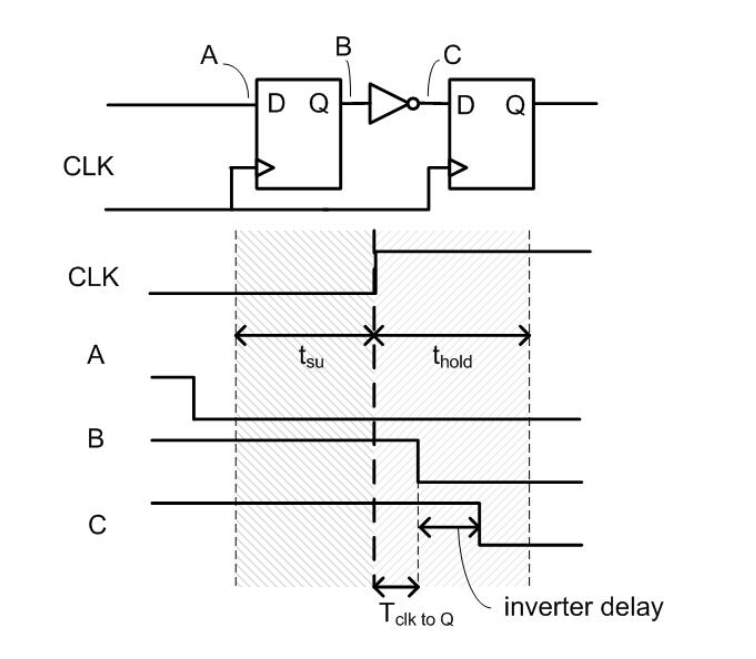
T\_clock >= t\_clktoq + t\_criticalpath + t\_setup

\*\*\*t\_clktoq usually includes hold time (they start at the same time), but if hold time > tclktoq then add to the t\_clock instead of tclktoq (one will always contain the other)

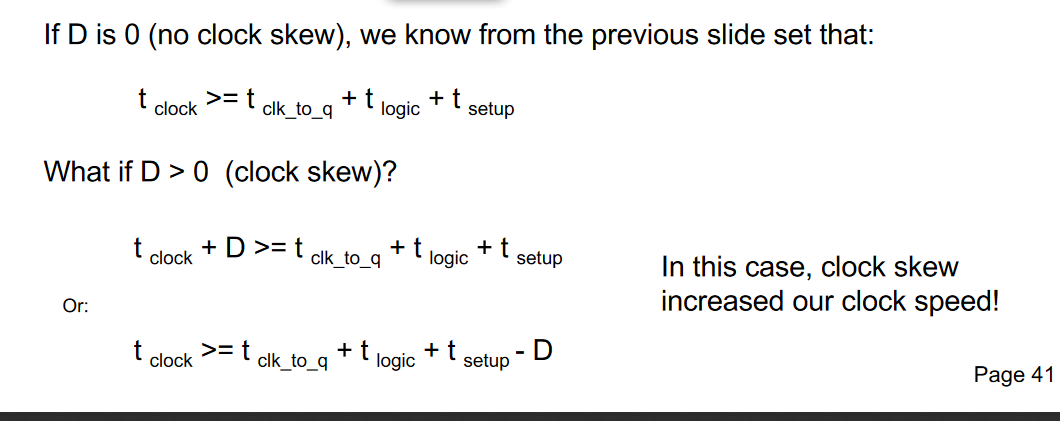
* + T\_clktoq\_min + t\_path >= t\_hold
* Circuit speed
  + Min clock period / max clock frequency

1GHz = 1ns

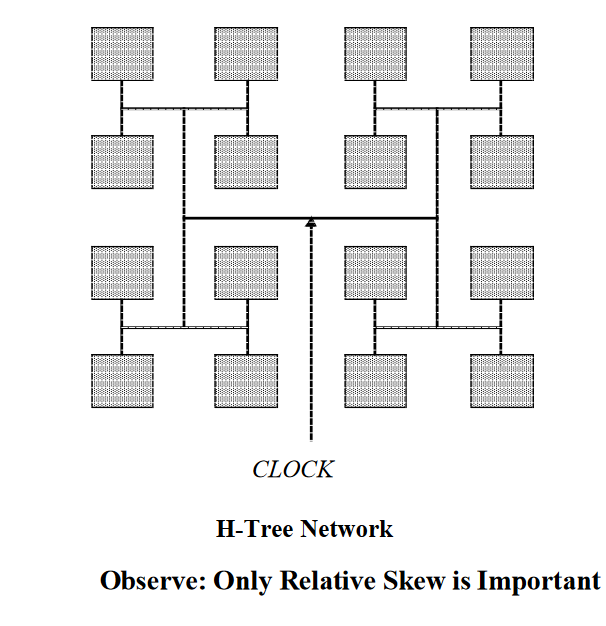
Draw clktoq diagram



* Timing closure: ensuring a design meets timing constraints



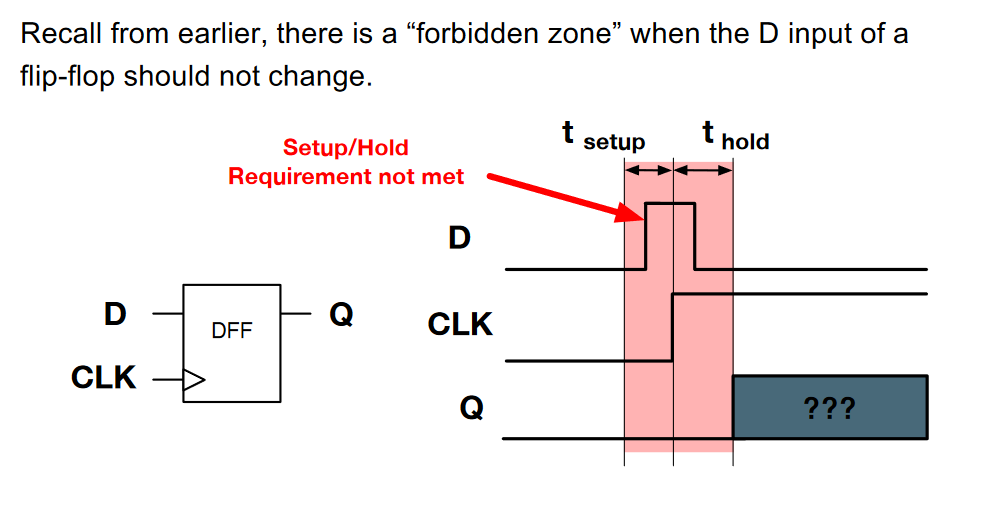
Clock skew

* No guarantee that the clock arrives at all ffs simultaneously
* If delay is in the same direction as the path, then the tclock will decrease and f will increase
* If delay is in the opposite direction of the path, then tclock will increase and f decrease
* Avoiding skew:
  + H-tree: grey are logic blocks
  + Global clock and regional clocks (GCLK, RCLK)
  + PLLs

PLLs

* Fractional or integer

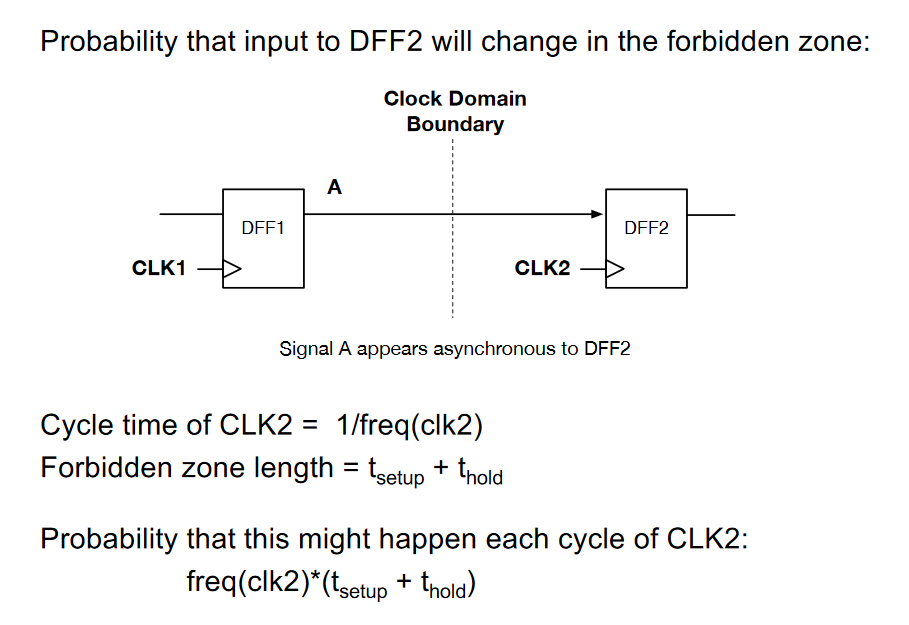
Glitch

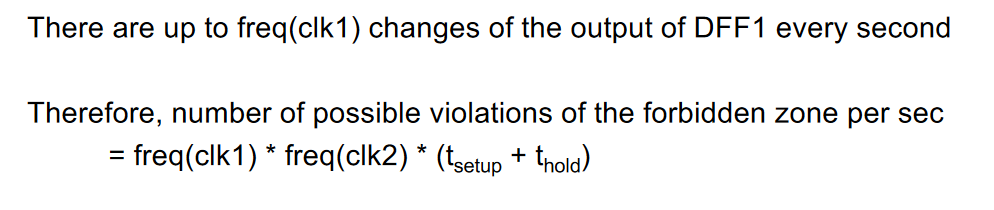
* Def: an unwanted short pulse that occurs before a signal settles to its intended value
* Causes
  + Unequal arrive times of inputs to a comb block
  + Other elec effects

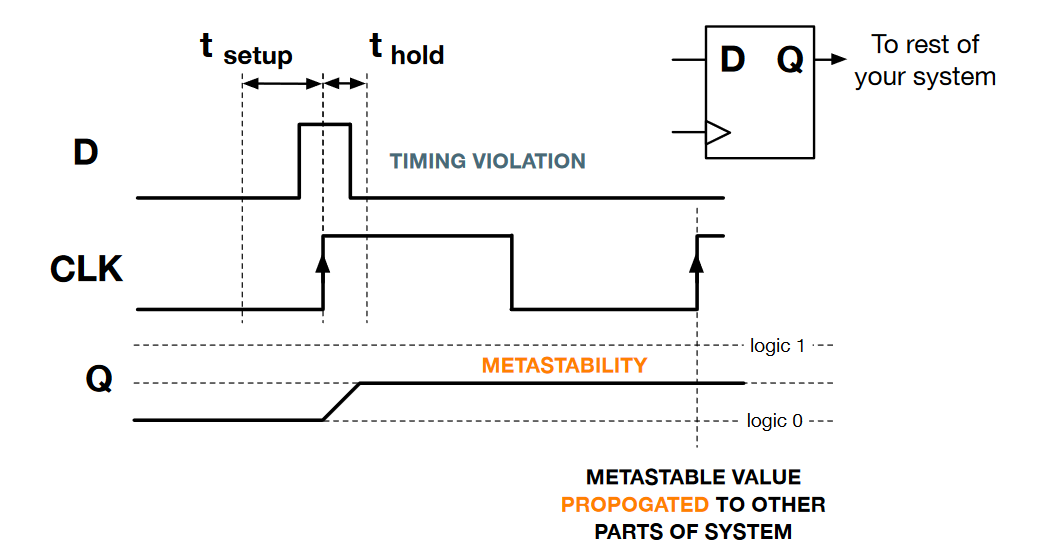
**Metastability**

* Concepts: clocks domains, and signals crossing domains

Problem with sigs crossing domains

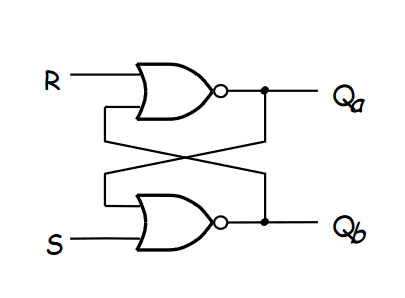
* “Forbidden zone” = tsetup + thold
* If clocks in the domains truly asynch, then no guarantee that a clock does not arrive at a ff during the forbidden zone
* Zone len = tsetup + thold
* P(forbidden) = f\_clk2 \* (tsetup + thold)

Asynch signals

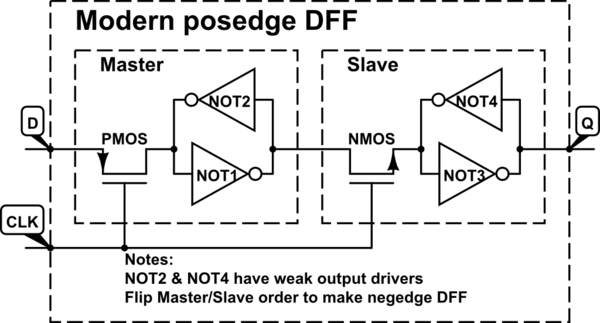
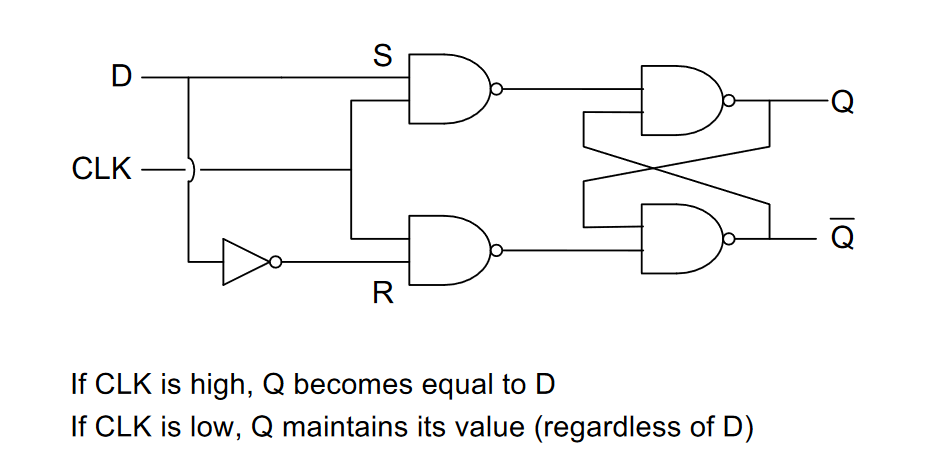
* Example: button or sensor (real-world)
* Non - zero chance of input changing too close to clock edge
* 

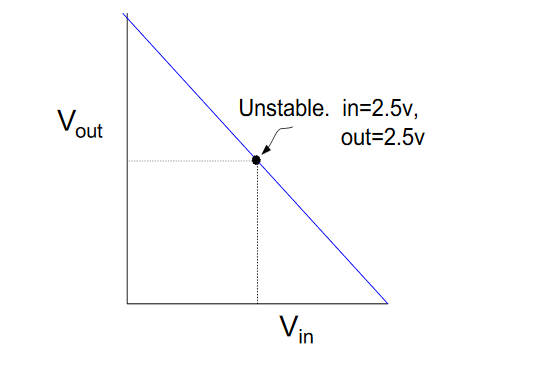
Violating ff setup or hold times

* Ex: D changes during forbidden zone: see the forbidden zone image
* Q could get correct or wrong value or become metastable
  + VERY BAD RIPPPP (system-wide failure)
  + Metastable value could propage to the whole system



Architecture of FFs

* S-R latch
  + S high = Qa high
  + R high = Qa low
  + S low, R low, Qa stays the same
* We use gated D latch
  + S = D, R = ~D
  + Level sensitive with CLK

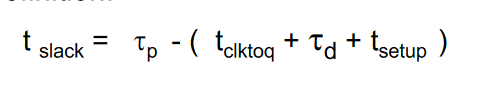
Inverters

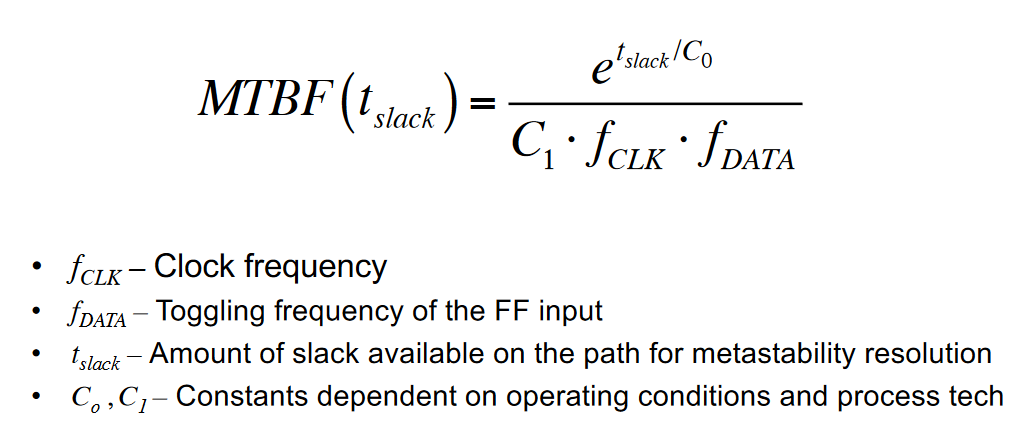
* Xfer function linear and decreasing

Metastability and ffs

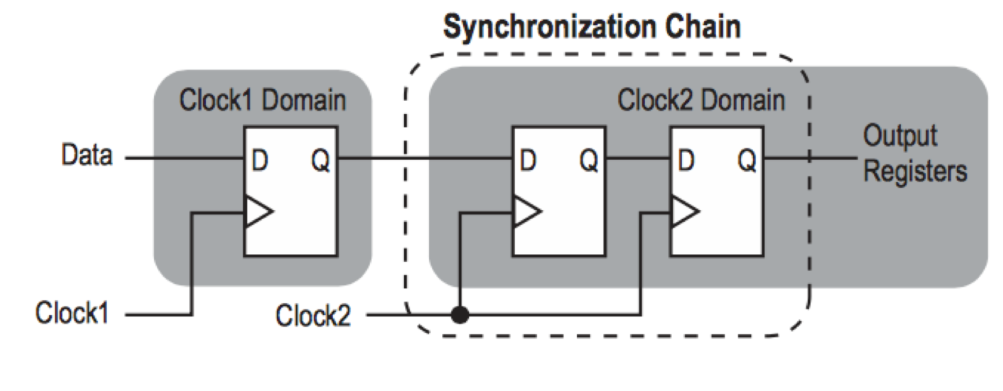
* If D changes in too close to clk (in setup time)
* If rise time of Q too slow

Avoiding metastability

* Ensuring that possible metastable component has enough time (slack) to settle before propagation continues

Mean time between failure (MTBF)

* Estimates avg time between 2 failure causing instances of metastability
* Implications
  + Higher MTBF = more robust design
  + Req. MTBF design dependent
    - Think application and its outcomes (medical vs. leisure)

Dealing with Metastability

* Soln 1: eliminate the possibility of metastability
  + Use only async circuits (not feasible)
* Soln 2: handle metastability when it occurs
  + Metastability detector that changes the settle time
* Soln 3: reduce the prob that metastability would cause system failure
  + Double-register each async input
* Soln 4: FIFO (Megafunction), very common
  + Write side of the FIFO is clocked by the sender
  + Read side is clocked but the receiver
  + Advantage: flow control
  + Already implemented in embedded RAM

**Power**

Importance of power:

* Difficult to feed and circulate large currents
* Cheaper device if more power efficient
* Improves portability

Power dissipation

* Dynamic power dissipation: whenever a node/wire switches states (pos/negedge)
  + Pdyn = a(activity)\*frequency\*Capacitance of a wire\*V(logic high voltage)^2
    - Estimate a through simulation, or statistical method(simpler, but inaccurate)
* Stator power dissipation: leakage, even when the chip is off

FPGAs

* Use a lot of power
* Lots of transistors = more leaking, = more capacitance = more Pdyb
* Use tools to estimate power usage
* Optimize design

Clock gating

* Clock dist. Networks consume a lot of power
* Clock gating allows for clks to be powered down for unused portions of the FPGA

Simplifying design

* Ex: use shifting and subtraction instead of multiplication

Bus invert coding: send majority low instead of high along with a polarity bit

Pipelining and glitches and power

* Adding ffs decreases glitches which decreases Pdyn
* However, too much pipelining could increase power consumption due to increase in used space
  + Also issues with clocking and more burden on H-trees
  + Always tradeoffs

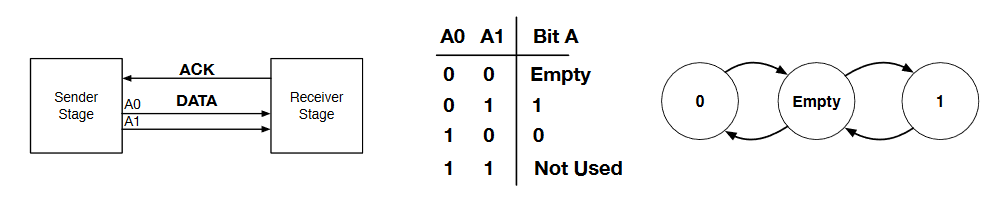
Tools

* Will reduce the length of connections in critical and high activity paths

**Async Datapaths**

Handshaking protocols

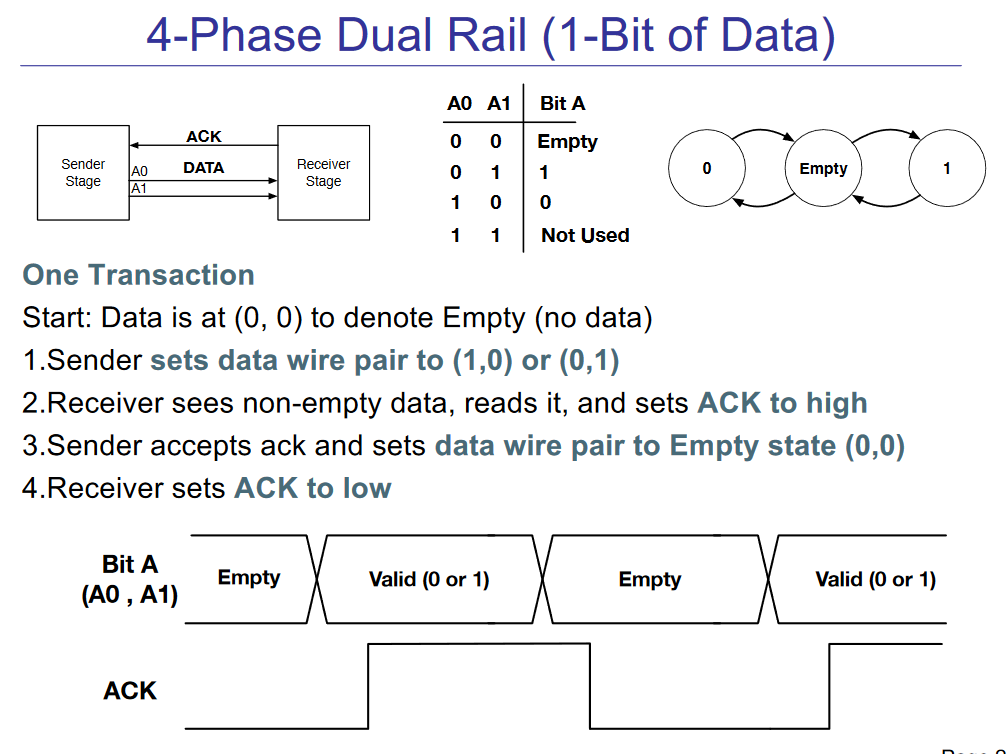
* 4 phase
  + 1: Sender issues data and sets ‘READY’ high
  + 2: Receiver sets ‘ACK’ high (acknowledge)
  + 3: Sender accepts the acknowledgement and disables ‘READY’
  + 4: Receiver acknowledges the disablement of ‘READY’ and disables ‘ACK’
* 2 phase (Non-return to zero or NRZ or transition signaling)
  + 1: Sender toggles ‘READY’
  + 2: Receiver acknowledges the setting high of ‘READY’ of the toggle and toggles ‘ACK’
* 4 has simpler hardware but takes more interaction between the S and R whereas 2 will have more complex hardware but minimizes that interaction

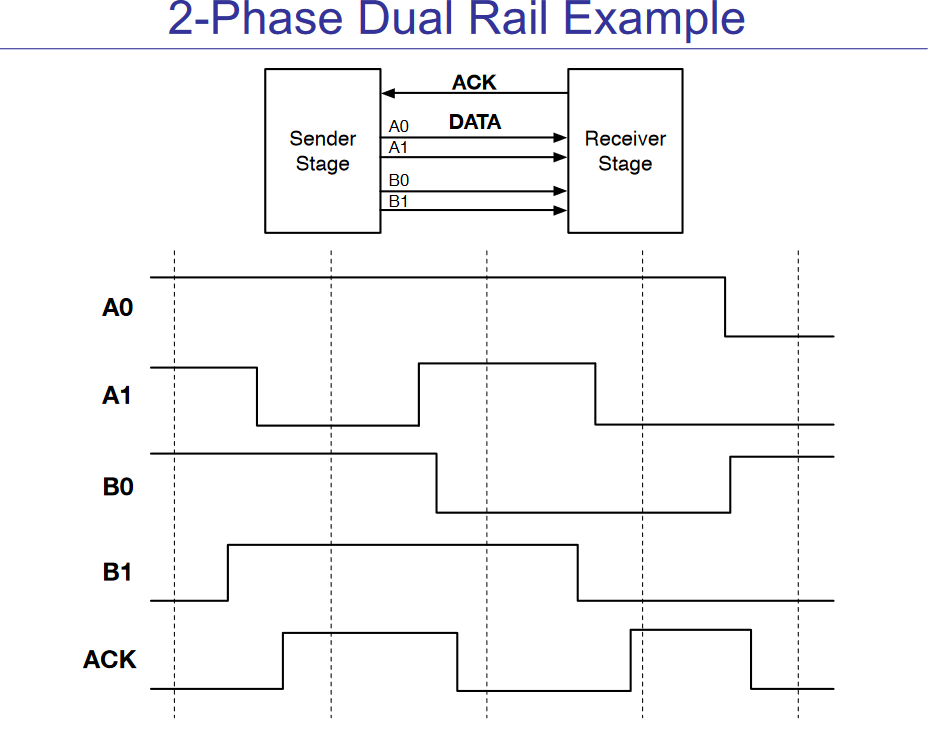
Data encoding

* Single-rail (Bundled)
  + No encoding, 1 bit = 1 wire
* Dual-rail
  + Each data bit encoded with 2 bits
  + Removes need for a ‘READY’ signal as it is encoded in the data

Combining handshakes and encoding

* Ex: 4 ph, dual rail, 1 bit

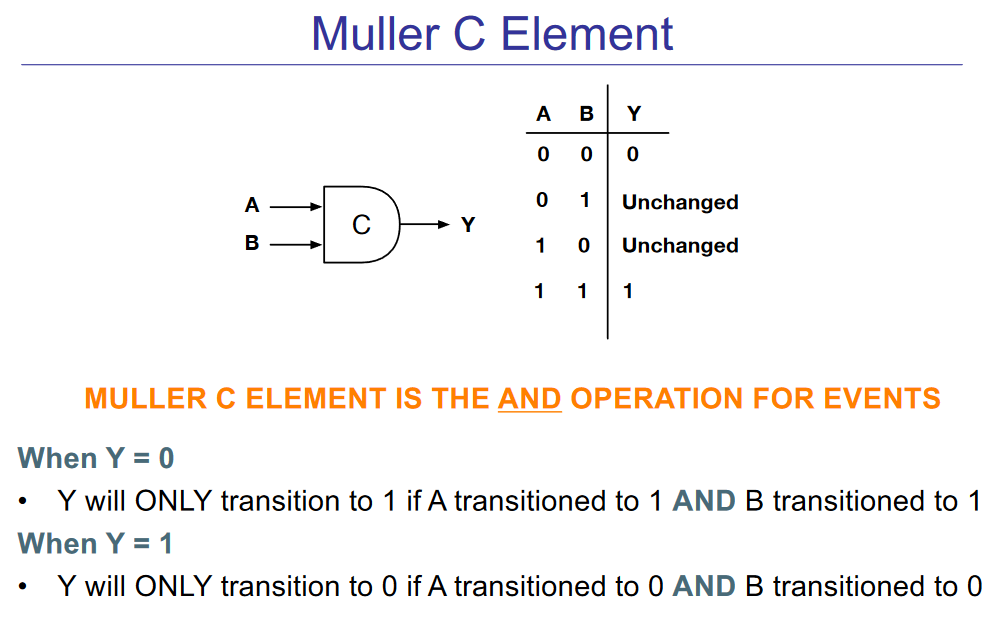
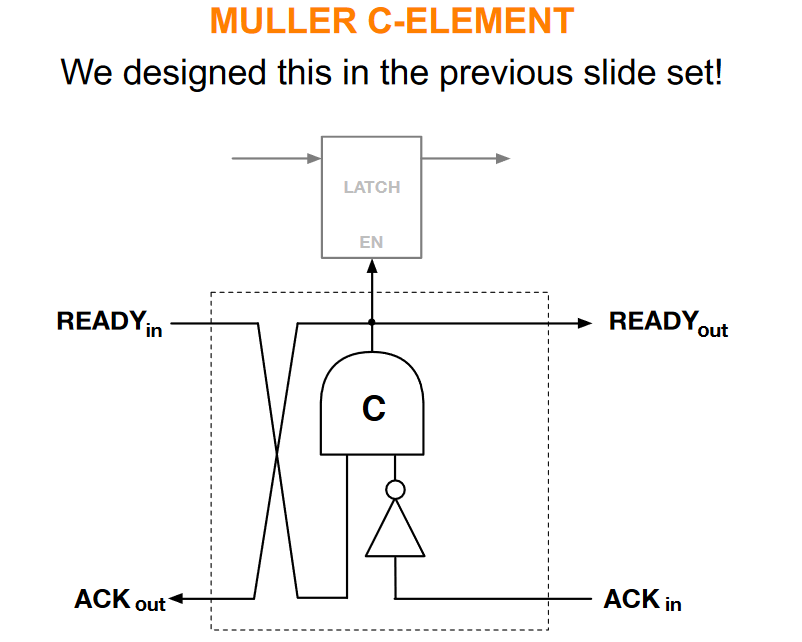




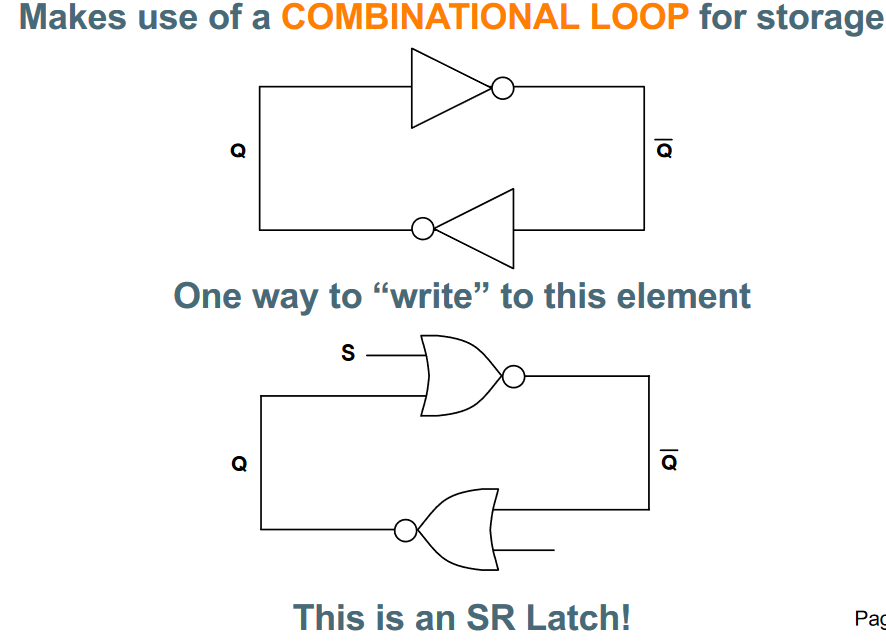
* Ex: 2-ph, dual rail, 2 bits

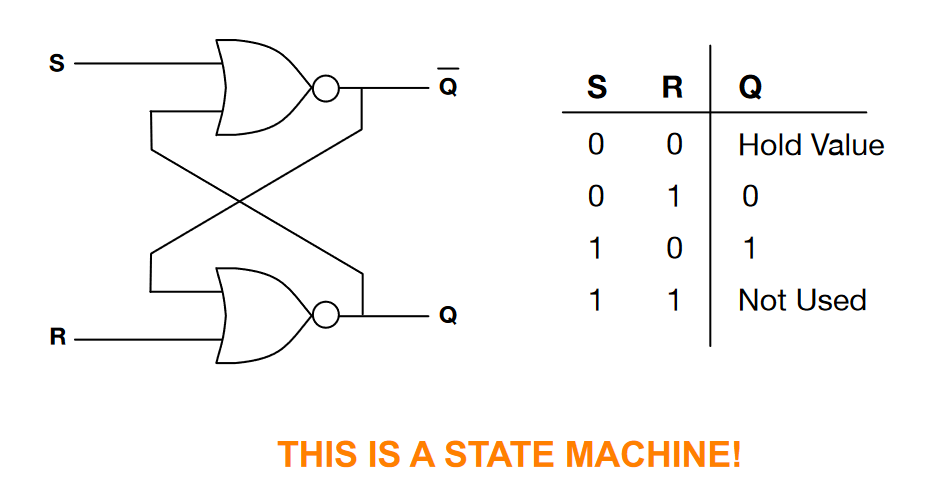
Control circuits

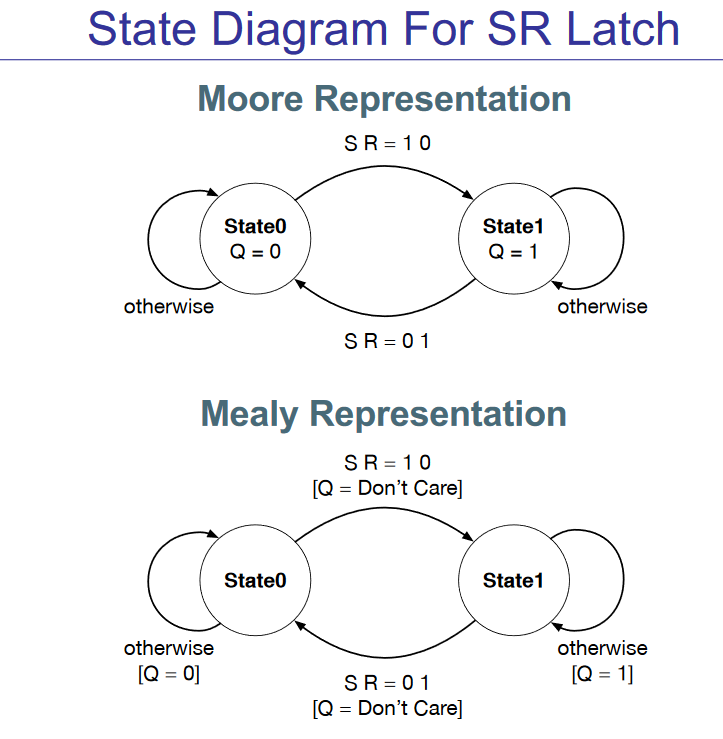
* Ex: muller c-element
  + Hazard free!



**Asynchronous FSMs**

**\*\*\* add both Muller C and pulse counter examples to note sheet**

Latches and flops are asynchronous



Async fsm process: same as with sync

* Start with a State Diagram
* Assign Encoding to States
* Create State Transition Table
* Create Karnaugh Map from State Transition Table
* Write boolean equations for next state

Stability

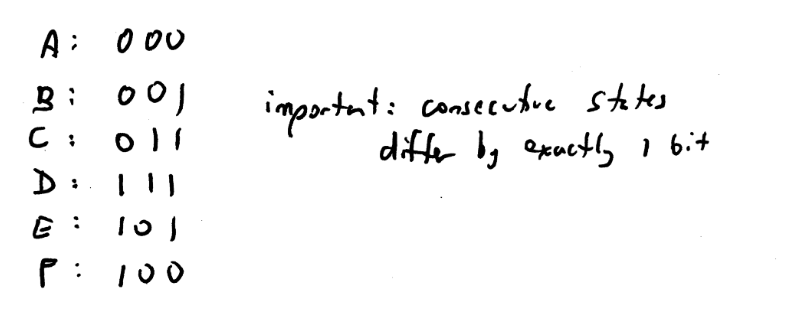
* Stable if for a given set of next-state logic input values, the circuit reaches a particular state
* Issues because async transitions to the next state “immediately” (some propagation delay)
* For an async fsm to be useful, it must eventually reach a stable state

Terminology

* Glitch: A temporary unwanted pulse/spike in a signal
* Hazard: A possibility of glitching due to circuit structure. In other words, a circuit with the potential to glitch is said to have a hazard.
* Static hazard: the potential for glitching to occur when the signal value should not change (remain static) transitions when there shouldn’t be any.
  + Occur if no overlapping in k-map
  + Static hazard elimination: redundancy to ensure overlapping
* Dynamic hazard:The potential for glitching to occur when the signal value should change (dynamic). Multiple transitions when there should be one.

Asynch FSM

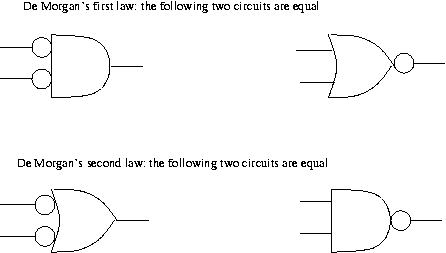
* Hamming state encoding: consec states must differ by exactly one bit



**Random**

* Logic instead of reg and wires
* Parameterized modules and module instantiations
* Typedef enum {state1, state2, …} state\_def;
* Generate blocks:

Genvar j;

Generate

for() begin : block\_name

End

Endgenerate

DeMorgan’s law

Kmap example setup

