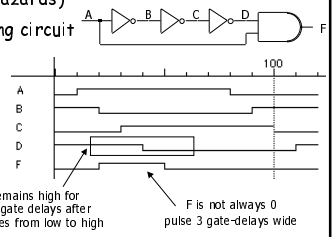
## State Machine Signaling

- Timing Behavior
  - Glitches/hazards and how to avoid them
- FSM Partitioning
  - What to do when the state machine doesn't fit!
- State Machine Signaling
  - State Machine Retiming
  - Introducing Idle States (synchronous model)
  - Four Cycle Signaling (asynchronous model)
- Dealing with Asynchronous Inputs
  - Metastability and synchronization



## Momentary Changes in Outputs

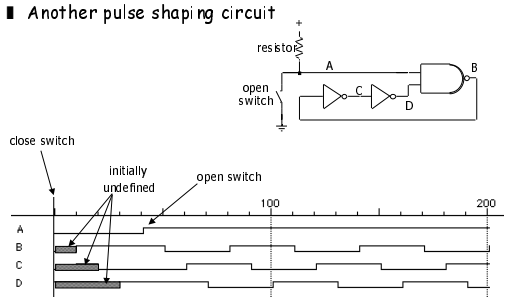
- Can be useful—pulse shaping circuits
- Can be a problem—incorrect circuit operation (glitches/hazards)
- Example: pulse shaping circuit
  - A' · A = 0
  - delays matter in function






## Oscillatory Behavior

- Another pulse shaping circuit






## Hazards/Glitches

- Hazards/glitches: unwanted switching at the outputs
  - Occur when different paths through circuit have different propagation delays
    - As in pulse shaping circuits we just analyzed
  - Dangerous if logic causes an action while output is unstable
    - May need to guarantee absence of glitches
- Usual solutions
  1) Wait until signals are stable (by using a clock): preferable (easiest to design when there is a clock - *synchronous* design)
  2) Design hazard-free circuits: sometimes necessary (clock not used - *asynchronous* design)

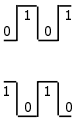


## Types of Hazards

- Static 1-hazard
  - Input change causes output to go from 1 to 0 to 1


- Static 0-hazard
  - INput change causes output to go from 0 to 1 to 0

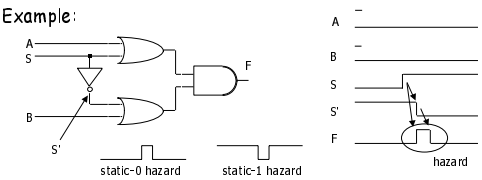

- Dynamic hazards
  - Input change causes a double change from 0 to 1 to 0 to 1 OR from 1 to 0 to 1 to 0





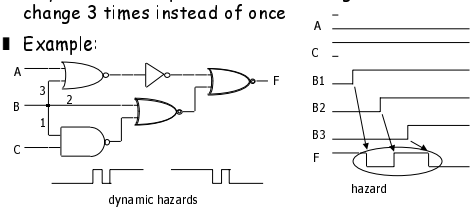

## Static Hazards

- Due to a literal and its complement momentarily taking on the same value
  - Thru different paths with different delays and reconverging
- May cause an output that should have stayed at the same value to momentarily take on the wrong value
- Example:



## Dynamic Hazards

- Due to the same versions of a literal taking on opposite values
  - Thru different paths with different delays and reconverging
- May cause an output that was to change value to change 3 times instead of once
- Example:



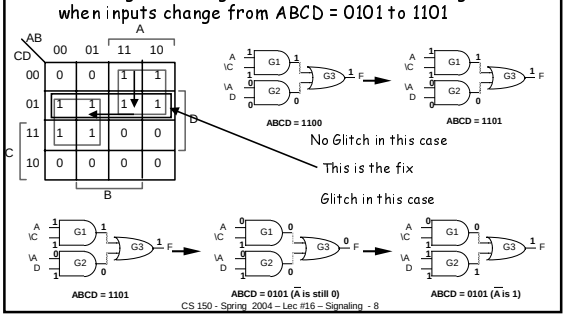






## Eliminating Static Hazards

- Following 2-level logic function has a hazard, e.g., when inputs change from ABCD = 0101 to 1101

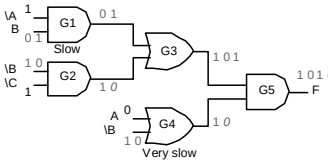








## Eliminating Dynamic Hazards

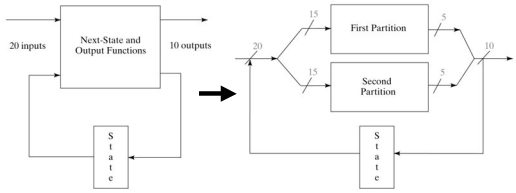



- Very difficult!
- A circuit that is static hazard free can still have dynamic hazards
- Best approach:
  - Design critical circuits to be two level and eliminate all static hazards
  - OR, use good clocked synchronous design style



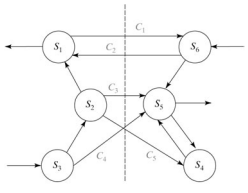
## FSM Partitioning

- Why Partition?
  - What if programmable logic is limited in number of inputs and outputs that can be used in a particular device?
  - For PALs/PLAs, the number of product terms are limited, thus limiting the complexity of the next state and output functions






## Partitioning the State Machine

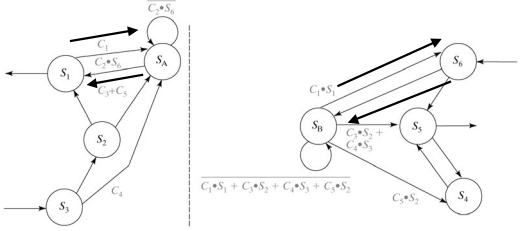



- Suppose that FSM is partitioned so that states at the right are in one partition and states at the left are in the other
- How do you support intersignaling between the state machine partitions?
- It is usually a good idea to partition the machine so there are as few cross links as possible (min cut set in graph theoretic terms)

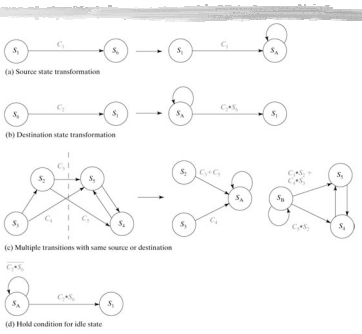


## Partitioning the State Machine

- Solution: introduce idle states $S_A$ and $S_B$
  - Machine at left enters $S_A$ allowing machine at right to exit $S_B$
  - When machine at right returns to $S_B$, machine at left exits $S_A$

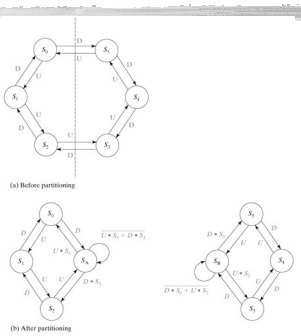


## Rules for Introducing Idle States





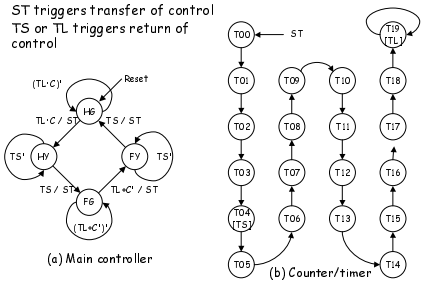

## Example: Partitioning the Up/Down Counter





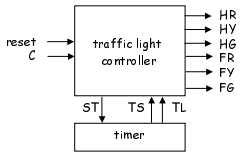

## Example Partitioning: Traffic Light Controller

- Main Controller vs. Counter/Timer
  - ST triggers transfer of control
  - TS or TL triggers return of control






## Partitioned FSM Block Diagram

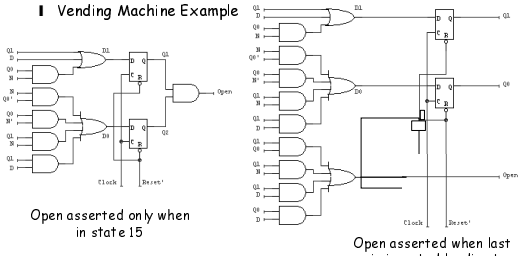



- Interface between the two partitions are the signals ST, TS, TL
- NOTE: Main Controller and Timer use the same clock and are operating in a synchronous mode



## State Machine Retiming

- Moore vs. (Async) Mealy Machine
  - Vending Machine Example



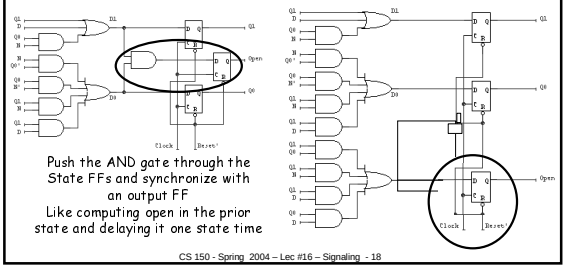

Open asserted only when in state 15

Open asserted when last coin inserted leading to state 15



## State Machine Retiming

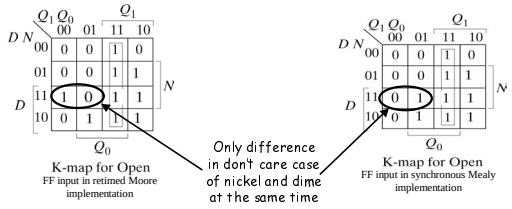
- Retiming the Moore Machine: Faster generation of outputs
- Synchronizing the Mealy Machine: Add a FF, delaying the output
- These two implementations have identical timing behavior




Push the AND gate through the State FFs and synchronize with an output FF
Like computing open in the prior state and delaying it one state time

## State Machine Retiming

- Timing behavior is the same, but are the implementations really identical?



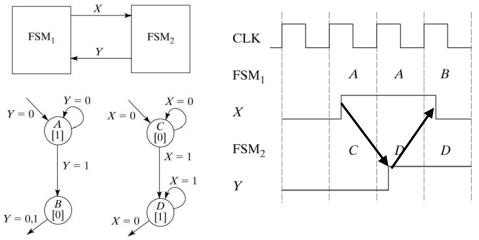

K-map for Open
FF input in retimed Moore implementation

K-map for Open
FF input in synchronous Mealy implementation



## Generalized Inter-FSM Signaling

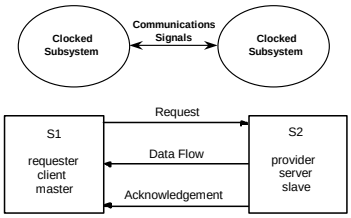
- Interlocked Synchronized Signaling






## Asynchronous Signaling

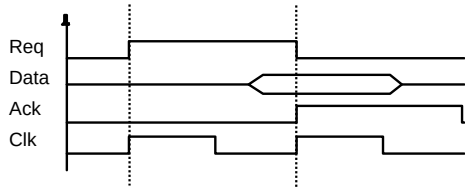
- Also known as "speed-independent" signaling
  - Requester/client/master vs. Provider/Server/Slave






## Asynchronous Signaling

- First consider the common clock case (synchronous)

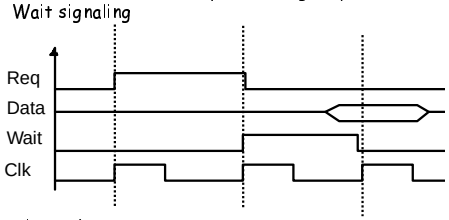



- Master asserts Request
- Slave recognizes request, processes request, indicates completion by asserting Acknowledgement
- Master accepts results, removes Request
- Slave see Request removed, removes Acknowledge



## Asynchronous Signaling

- What if Slave can't respond in single cycle? Solution: Wait signaling

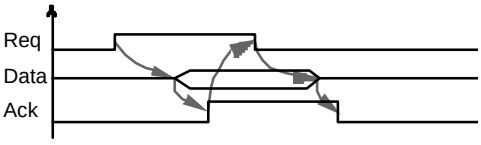



- Slave inhibits master by asserting wait
- When slave unasserts wait, master knows request has been processed, and can latch results



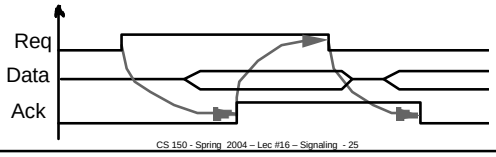
## True Asynchronous Signaling

- Now remove the assumption of a single common clock
- How do we make sure that receiver has seen the sender's signal? Solution: Interlocked signaling
- Four cycle signaling: assert Req, process request, assert ack, latch result, remove Req, remove Ack and start again
- Sometimes called "Return to Zero" signaling



## True Asynchronous Signaling

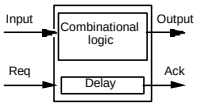
- Alternative scheme: Two-Cycle Signaling
  - Non-return-to-zero signaling
  - Transaction start by Req lo-to-hi, finishes Ack lo-to-hi
  - Next transaction starts by Req hi-to-lo, finishes Ack hi-to-lo
  - Requires EXTRA state to keep track of the current sense of the transitions—faster than 4 cycle case, but usually involves more hardware






## True Asynchronous Timing

- Self-Timed Circuits
  - Uses Req/Ack signaling as described
  - Components can be constructed with NO internal clocks
  - Determines on its own when the request has been processed
  - Concept of the delay line simply slows down the pass through of the Req to the Ack—usually matched to the worst case delay path
  - Becoming MORE important for large scale VLSI chips were global clock distribution is a challenge





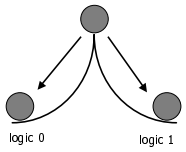

## Metastability and Asynchronous inputs

- Clocked synchronous circuits
  - Inputs, state, and outputs sampled or changed in relation to a common reference signal (called the clock)
  - E.g., master/slave, edge-triggered
- Asynchronous circuits
  - Inputs, state, and outputs sampled or changed independently of a common reference signal (glitches/hazards a major concern)
  - E.g., R-S latch
- Asynchronous inputs to synchronous circuits
  - Inputs can change at any time, will not meet setup/hold times
  - Dangerous, synchronous inputs are greatly preferred
  - Cannot be avoided (e.g., reset signal, memory wait, user input)

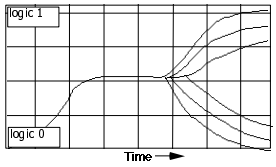


## Synchronization Failure

- Occurs when FF input changes close to clock edge
  - FF may enter a metastable state - neither a logic 0 nor 1 -
  - May stay in this state an indefinite amount of time
  - Is not likely in practice but has some probability




small, but non-zero probability that the FF output will get stuck in an in-between state

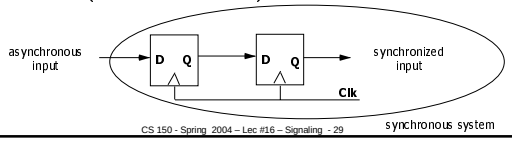



oscilloscope traces demonstrating synchronizer failure and eventual decay to steady state



## Dealing with Synchronization Failure

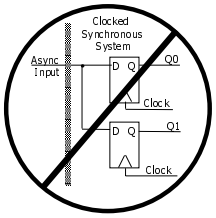
- Probability of failure can never be reduced to 0, but it can be reduced
  - (1) slow down the system clock: this gives the synchronizer more time to decay into a steady state; synchronizer failure becomes a big problem for very high speed systems
  - (2) use fastest possible logic technology in the synchronizer: this makes for a very sharp "peak" upon which to balance
  - (3) cascade two synchronizers: this effectively synchronizes twice (both would have to fail)

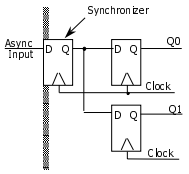





## Handling Asynchronous Inputs

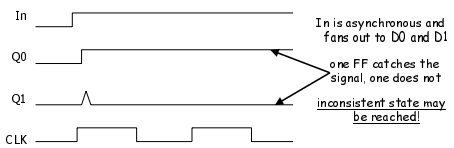
- Never allow asynchronous inputs to fan-out to more than one flip-flop
  - Synchronize as soon as possible and then treat as synchronous signal



## Handling Asynchronous Inputs (cont'd)

- What can go wrong?
  - Input changes too close to clock edge (violating setup time constraint)






## Signaling Summary

- Glitches/Hazards
  - Introduce redundant logic terms to avoid them OR use synchronous design!
- FSM Partitioning
  - Replacing monolithic State Machine with simpler communicating state machine
  - Technique of introducing idle states
- State Machine Retiming
  - Rearranging the logic/FFs in a state machine to speed up its output signaling behavior
- Machine-to-machine Signaling
  - Synchronous vs. asynchronous
  - Four vs. Two Cycle Signaling
- Asynchronous inputs and their dangers
  - Synchronizer failure: what it is and how to minimize its impact