Timing methodologies

- Rules for interconnecting components and clocks
  - guarantee proper operation of system when strictly followed
- Approach depends on building blocks used for memory elements
  - will focus on systems with edge-triggered flip-flops
  - found in programmable logic devices
- many custom integrated circuits focus on level-sensitive latches
- Basic rules for correct timing:
  - (1) correct inputs, with respect to time, are provided to the flip-flops
  - (2) no flip-flop changes state more than once per clocking event

Timing methodologies (cont'd)

- Definition of terms
  - clock: periodic event, causes state of memory element to change
    - can be rising edge or falling edge or high level or low level
  - setup time: minimum time before the clocking event by which the
    - input must be stable (Tsu)
  - hold time: minimum time after the clocking event until which the
    - input must remain stable (Th)

Comparison of latches and flip-flops

- Positive edge-triggered flip-flop
  - setup and hold times
  - minimum clock width
  - propagation delays (low to high, high to low, max and typical)

Comparison of latches and flip-flops (cont'd)

- Type
  - unlocked latch
  - level-sensitive latch
  - master-slave flip-flop
  - negative edge-triggered flip-flop

Typical timing specifications

- Positive edge-triggered D flip-flop
  - setup and hold times
  - minimum clock width
  - propagation delays (low to high, high to low, max and typical)

Cascading edge-triggered flip-flops

- Shift register
  - new value goes into first stage
  - while previous value of first stage goes into second stage
  - consider setup/hold/propagation delays (prop must be > hold)
Cascading edge-triggered flip-flops (cont'd)

- Why this works:
  - propagation delays exceed hold times
  - clock width constraint exceeds setup time
  - this guarantees following stage will latch current value before it changes to new value

- Assumes infinitely fast distribution of the clock

Clock skew

- The problem:
  - correct behavior assumes next state of all storage elements determined by all storage elements at the same time
  - this is difficult in high-performance systems because time for clock to arrive at flip-flop is comparable to delays through logic

- Effect of skew on cascaded flip-flops:

```
In  Q0  Q1
CLK0

Tsu 4ns
Tp 3ns
Th 2ns
```

```
In  Q0  Q1
CLK1

Tsu 4ns
Tp 3ns
Th 2ns
```

Summary of latches and flip-flops

- Development of D-FF:
  - level-sensitive used in custom integrated circuits
  - can be made with 4 switches
  - edge-triggered used in programmable logic devices
  - good choice for data storage register

- Historically J-K FF was popular but now never used
- similar to R-S but with 1-1 being used to toggle output (complement state)
- good in days of TTL/SSI (more complex input function: \( D = JQ' + K'Q \))
- not a good choice for PALs/PLAs as it requires 2 inputs
- can always be implemented using D-FF

- Preset and clear inputs are highly desirable on flip-flops
- used at start-up or to reset system to a known state

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 –
  - it may stay in this state an indefinite amount of time
  - this is not likely in practice but has some probability

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 stable 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).