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)

![D Flip-Flop Diagram]

All measurements are made from the clocking event that is, the rising edge of the clock.

Synchronous System Model

- Register-to-register operation
- Perform operations during transfer
- Many transfers/operations occur simultaneously

![Synchronous System Model Diagram]

System Clock Frequency

- Register transfer must fit into one clock cycle
  - \( t_{su} + C.L. t_{su} + t_{su} < T_{ck} \)
  - Use maximum delays
  - Find the “critical path”
    - Longest register-register delay

![System Clock Frequency Diagram]

Short Paths

- Can a path have too little delay?
  - Yes: Hold time can be violated
  - \( t_{su} > t_{h} \)
  - Use min delay (contamination delay)
  - Fortunately, most registers have hold time = 0
  - But there can still be a problem! Clock skew...

![Short Paths Diagram]
Clock Skew

- Cannot make clock arrive at registers at the same time
- If skew > 0:
  - $t_{pd} > t_{su} + t_{skew}$
  - Clock skew can cause system failure
    - Can you fix this after you've fabbed the chip?

- If skew < 0:
  - $t_{pd} > t_{su} + CL + t_{su} + t_{skew}$
  - Can you fix this after fab?

Correct behavior assumes that all storage elements sample at exactly the same time
- Not possible in real systems:
  - clock driven from some central location
  - different wire delay to different points in the circuit
- Problems arise if skew is of the same order as FF contamination delay
- Gets worse as systems get faster (wires don't improve as fast)
  - 1) distribute clock signals against the data flow
  - 2) try to make the clock between two communicating components be as short as possible
  - 3) try to make all wires from the clock generator be the same length => clock tree

Nasty Example

- What can go wrong?
- How can you fix it?
Other Types of Latches and Flip-Flops

- D-FF is ubiquitous
  - simplest design technique, minimizes number of wires preferred in PLDs and FPGAs
  - good choice for data storage register
  - edge-triggered has most straightforward timing constraints
- Historically JK FF was popular
  - versatile building block, often requires less total logic
  - two inputs require more wiring and logic
  - can always be implemented using D-FF
- Level-sensitive latches in special circumstances
  - popular in VLSI because they can be made very small (4 T)
  - fundamental building block of all other flip-flop types
- Two latches make a D-FF
- Preset and clear inputs are highly desirable
  - System reset

Calculating probability of failure

This means register may sample a signal that is changing
- Preset and clear inputs are highly desirable
  - System reset

What About External Inputs?

- Internal signals are OK
  - Can only change when clock changes
- External signals can change at any time
  - Asynchronous inputs
  - Truly asynchronous
  - Produced by a different clock
- This means register may sample a signal that is changing
  - Violates setup/hold time
  - What happens?

Synchronization failure

- Occurs when FF input changes close to clock edge
  - the 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

Sampling external inputs

Calculating probability of failure

- For a single synchronizer
  \[ \text{Mean-Time Between Failure (MTBF)} = \exp \left( \frac{t}{\tau} \right) \left( \frac{T0}{f} \times a \right) \]
  
  where a failure occurs if metastability persists beyond time \( t \)
  - \( t \) is the resolution time - extra time in clock period for setting
  - \( T0 \) - \( a + T0 + \text{setup} \)
  - \( f \) is the frequency of the FF clock
  - \( a \) is the number of asynchronous input changes per second applied to the FF
  - \( T0 \) and \( \tau \) are constants that depend on the FF’s electrical characteristics
    (e.g., gain or steepness of curve)
  - example values are \( T0 = 4s \) and \( \tau = 1.5ns \)
  - sensitive to temperature, voltage, cosmic rays, etc.
  - Must add probabilities from all synchronizers in system
  \[ \text{1/MTBF}_{\text{system}} = \Sigma \text{1/MTBF}_{\text{synch}} \]
Metastability

- Example
  - Input changes at 1 MHz
  - System clock of 10 MHz, flipflop (t\text{setup} + t\text{hold}) = 5 ns
  - MTBF = \exp(5 ns / 1.5 ns) \approx 4.6 \times 10^{10} = 25 million years
  - If we go to 20 MHz then:
    MTBF = \exp(45 ns / 1.5 ns) \approx 4.6 \times 10^{13} = 1.33 seconds!
  - And we’re not even doing any logic!
- Must do the calculations and allow enough time for synchronization

What does this circuit do?

- What’s wrong with this?

Guarding against synchronization failure

- Give the register time to decide
  - Probability of failure cannot be reduced to 0, but it can be reduced
  - Slow down the system clock?
- Use very fast technology for synchronizer -> quicker decision?
- Cascade two synchronizers?

Stretching the Resolution Time

- Also slows the sample rate and transfer rate

Sampling Rate

- How fast does your sample clock need to be?
**Sampling Rate**

- How fast does your sample clock need to be? $f_{(clkB)} > 2f_{(clkA)}$

![Sequential Logic](image)

**Important Rule!**

- What if sample clock can’t go faster?
- If input clock is not available, no solution(?)
- If input clock is available (e.g. video codec)

![Sequential Logic](image)

**Increasing sample rate**

- The problem is the relative sample rate
- Slow down the input clock!

![Sequential Logic](image)

**Another Problem with Asynchronous inputs**

- What goes wrong here? (Hint: it’s not a metastability thing)
- What is the fix?

![Sequential Logic](image)

**More Asynchronous inputs**

- What is the problem?
- What is the fix?

![Sequential Logic](image)

**Important Rule!**

- Exactly one register makes the synchronizing decision

![Sequential Logic](image)
More Asynchronous inputs

- Can we input asynchronous data values with several bits?

![Sequential Logic Diagram]

More Asynchronous inputs

- How can we input asynchronous data values with several bits?

![Sequential Logic Diagram]

What Went Wrong?

- Each bit has a different delay
  - Wire lengths differ
  - Gate thresholds differ
  - Driver speeds are different
  - Register delays are different
  - Rise vs. Fall times
  - Clock skews to register bits
- Bottom line - “data skew” is inevitable
  - aka Bus Skew
  - Longer wires => More skew
- What is the solution??

![Sequential Logic Diagram]

Sending Multiple Data Bits

- Must send a “clock” with the data
  - Waits until data is stable
  - De-skewing delay
  - $f(clkB) > 2 f(clkA)$

![Sequential Logic Diagram]

Sending Multiple Data Bits

- Balancing path delays . . .
- What’s wrong with this solution?
- What’s the right way to do it?

![Sequential Logic Diagram]

Sending Multiple Data Bits

- The right way to do it . . .

![Sequential Logic Diagram]
Sending Multiple Data Bits

- Slightly different alternative...