Asynchronous {Pipelines, dataflow}
Today

• HSRA commentary
• Clocking
• Synchronizers
• Dataflow in asynchronous systems
Big Picture

• Last week: **Synchronous** pipelines & data transactions

• This week: **Asynchronous** pipelines & data transactions

• Next week: {Synchronous, Asynchronous} FIFOs
Clocking Basics

• A clock signal
Clocking Basics

- A clock signal
- Setup time
Clocking Basics

- A clock signal
- Setup time
- Hold time
Clocking Basics

- A clock signal
- Setup time
- Hold time
- clk → Q
Clocking Basics

Quiz: Can $t_{setup}$ or $t_{hold}$ to be negative?

- $t_{setup} < 0$: D can change after the clock edge and the \textit{new} D will be recognized
- $t_{hold} < 0$: D can change before the clock edge and the \textit{old} D will be recognized

What about $\text{clk} \rightarrow Q$?
Metastability

• What happens when $t_{\text{setup}}$ or $t_{\text{hold}}$ are violated?

• Output unknown
  (somewhere between 0 and 1)

... until “resolution” occurs

at which point “out” could be either 0 or 1!
Metastability

• When can $t_{\text{setup}}$ or $t_{\text{hold}}$ be violated?

• One Clock
  Design doesn’t meet timing
  (You have bigger problems)

• Two Clocks: phase offset
  May or may not cause violations

• Two Clocks: different frequencies
  Will almost always cause violations
  Thought Q: Exceptions to this?
Metastability

• Resolution must occur within $t_r$

$$t_r = t_p - t_{clk\rightarrow Q} - t_{cl} - t_{su}$$

• Good news:
  chance to leave metastability increases exponentially with time

• Bad news:
  synchronization failure means… circuit failure
Synchronizers

• First flip-flop absorbs metastability
• Second flip-flop protects downstream logic

\[ t_r = t_p - t_{\text{clk}} \rightarrow Q - t_{\text{su}} \]
Synchronizers

• How can we do better? Increase $t_r$

$$t_r = N \times (t_p - t_{clk\rightarrow Q-} - t_{su})$$
Synchronizers

• Another “reliable synchronizer”

\[ t_r = N x t_p - t_{clk} \rightarrow Q - t_{su} \]
Synchronizers

• Synchronizer cost...
  – Area (but not much)
  – Cycle Delay

• Where does this matter?  Handshaking

• Case Study:
  Asynchronous FIFOs
Asynchronous Pipelines

• Recall... the FIFO interface that we call Ready/Valid

![Diagram of asynchronous pipelines]

• This worked in a single clock domain...

• Why?

  Transfers @ edge, both parties see change at the *same* time
Asynchronous Pipelines

• What happens in two clock domains?

• First: we must avoid metastability. Ideas?
Asynchronous Pipelines

- Step #1: Add synchronizers (prevents metastability)
Asynchronous Pipelines

- Step #1: Add synchronizers (prevents metastability)
- Step #2: Add a hold register (does this help here?)

Aside: Why not push data through parallel synchronizers?

This still doesn’t work!
Asynchronous Pipelines

• Problem:
  – It takes multiple cycles for a message from the receiver to reach the sender

• Why do we care?
  – What happens when the receiver says “stop?” (i.e. DataInReady = 0)

• Solution
  – Add buffering to the receiver
  – Add “almost full” like in lecture
Asynchronous Pipelines

• “Almost full” gives sender time to stop

- Same idea as what you saw in lecture
- What is the receiver starting to look a lot like?
Homework

• Thought problem
  – Based on what you have seen in lecture & today:
    Draw a block diagram for a synchronous FIFO
  – (More) reading will be posted
Acknowledgements & Contributors

Slides developed by Chris Fletcher (2/2010).

This work is based on ideas and discussions with:

Ilia Lebedev, Greg Gibeling, John Wawrzynek, Krste Asanovic,
and other fellow Spring 2009 CS294-48 students

This work has been used by the following courses:

– UC Berkeley CS150 (Spring 2010): Components and Design Techniques for Digital Systems