EE273 Lecture 15
Synchronizer Design

March 5, 2003

Sarah L. Harris
Computer Systems Laboratory
Stanford University
slharris@cva.stanford.edu
Logistics

• Final Exam
  – Wednesday 3/19, 9:30AM to 11:30AM

• Upcoming Lecture Schedule
  – 3/10 – Guest Lecture – Ed Lee - Velio
  – 3/12 – Wrapup
A Quick Overview

• Synchronization Hierarchy
  • Mesochronous Synchronizers
    – delay-line synchronizer
    – two-register synchronizer
    – FIFO synchronizer
  • Plesiochronous Synchronizers
    – phase slip and flow control

• Periodic Synchronizers
  – clock prediction - looking into the future
Synchronization Hierarchy

- The difficulty of synchronization depends on the relationship between events on the signal and events on the clock
- **Synchronous**
  - signal events always happen outside of the clock’s keep-out region
    - same clock
- **Mesochronous**
  - signal events happen with a fixed but unknown phase relative to the clock
    - same frequency clock
- **Plesiochronous**
  - phase of signal events changes slowly with time
    - slightly different frequency clock
- **Periodic**
  - signal events are periodic
    - includes meso- and plesiochronous
    - signal is synchronized to some periodic clock
- **Asynchronous**
  - signal events may occur at any time
## Synchronization Hierarchy Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Frequency</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>Same</td>
<td>Same</td>
</tr>
<tr>
<td>Mesochronous</td>
<td>Same</td>
<td>Constant</td>
</tr>
<tr>
<td>Plesiochronous</td>
<td>Small Difference</td>
<td>Slowly Varying</td>
</tr>
<tr>
<td>Periodic</td>
<td>Different</td>
<td>Periodic Variation</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>N/A</td>
<td>Arbitrary</td>
</tr>
</tbody>
</table>
The Brute-Force Synchronizer

• How do we compare synchronizers?
  – synchronizer delay (in addition to the required $t_{cy}/2$)
  – failure rate

• For the brute-force synchronizer
  – $t_d = t_w + 2(t_s + t_{dCQ})$
  – $f_f = t_a e^{f_c y} e^{(-t_w/\tau_s)}$

• Can we do better?
Periodic Synchronizers
The Big Picture

• If an input signal is synchronized to some periodic clock, we can predict when its events are allowed to happen arbitrarily far into the future

• Thus, we can determine well in advance if the signal is safe to sample on a given clock cycle
  – if it is, we just sample it
  – if it isn’t, we delay the signal (or the clock) to make it safe

• This allows us to move the waiting time, $t_W$, out of the critical path.
  – we can make it very long without adding latency
Periodic Synchronizers
The Illustration

Clk1

Clk2
Mesochronous Synchronization

• The phase difference between the signal and the clock is **constant**
  - typical of systems where we distribute a master clock with no deskew
• Thus, we only need to synchronize once for all time!
• During reset check the phase
  - if its OK, sample the signal directly for ever
  - if its not, sample the signal after delay for ever
  - this phase check is the only asynchronous event we ever sample - and we can afford to wait a long time
Delay-Line Synchronizer

- For mesochronous and plesiochronous signals
- Delay signal as needed to keep transitions out of the *keep-out* region of the synchronizer clock
- How do we set the delay line?
- Do we need the flip-flop?
- What is the delay of this synchronizer?

\[ D \quad Q \]
\[ x \quad xd \quad xs \]
\[ Clk \]
Detecting an Unsafe Signal

- To see if a signal is unsafe, see if it changes in the forbidden region
  - sample just before and after the forbidden region and see if result is different

- These samples may hang the flip-flop in a metastable state
  - need to wait for this state to decay
  - if mesochronous we can wait a very long time since we only have to do this once

![Diagram of detecting an unsafe signal](attachment:image.png)
Two-Register Synchronizer

• The delay-line synchronizer has two problems
  1. It’s expensive, we need a delay line for each input
  2. We can’t use it with clocked receivers
• Both problems are solved by the two-register synchronizer
• We delay the clock rather than the data
  – sample the data with normal and delayed clock
  – pick the ‘safe’ output
• Can we just mux the clock?
FIFO Synchronizer

- A first-in-first-out (FIFO) buffer can be used to move the synchronization out of the data path.
- Clock the data into the FIFO in one clock domain (xclk).
- Mux the data out of the FIFO in a second clock domain (clk).
- Where did the synchronization move to?
- How do we initialize the pointers?
Open-Loop and Closed-Loop Flow Control

• We need to keep a fast transmitter from overrunning a receiver
• (or a slow transmitter from underrunning the receiver)
• Open-loop approach
  – insert lots of nulls into the data stream at the transmitter
  – enough so that rate of non-nulls is less than the rate of the slowest possible receiver
  – when the receiver underruns it inserts another null

• Closed-loop approach
  – receiver applies back pressure when it is about to be overrun
  – still has to insert nulls when it is underrun
Periodic Timing

- Transmit and receive clocks are periodic but at unrelated frequencies
  - e.g., modules in a system operate off of separate oscillators with independent frequencies
  - case where one is rationally derived from the other is an interesting special case

- In this situation, a single synchronization won’t last forever (like mesochronous) or even for a long time (like plesiochronous)

- However, we can still look into the future and predict clock conflicts far enough ahead to reduce synchronizer delay
Clock-Predictor Circuit

- Suppose we want to know the value of xclk, one rclk cycle ($t_{rcy}$) in the future.
- This is just a phase shift of $t_{xcy} - t_{rcy}$.
- It is easy to generate this phase shift using a simple timing loop.
- Note that we could just as easily predict xclk several rclk cycles in the future.
- So how do we build a synchronizer using this?
Asynchronous Timing

- Sometimes we need to sample a signal that is truly asynchronous
- We can still move the synchronization out of the datapath by using an asynchronous FIFO synchronizer
- However, this still incurs a high latency on the full and empty signals as we have to wait for a brute force synchronizer to make its decision
- We can still avoid delay in this case if we don’t really need to synchronize
  - often synchronization is just an expensive convenience