EE273 Lecture 13
Synchronizer Design

March 2, 2001

William J. Dally
Computer Systems Laboratory
Stanford University
billd@csl.stanford.edu

Logistics

- Reading
  - Sections 5.1, 5.2, 5.5, and 5.6
- Project
  - Due on 3/7
  - Automatic extension until 3/12
- Final Exam
  - Friday 3/23, 8:30AM to 10:30AM
  - Location TBD
- Upcoming Lecture Schedule
  - 3/5 – Power distribution – off-chip
  - 3/7 – Power distribution – on-chip and summary
  - 3/12 – Wrapup
  - 3/14 – Guest lecture

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

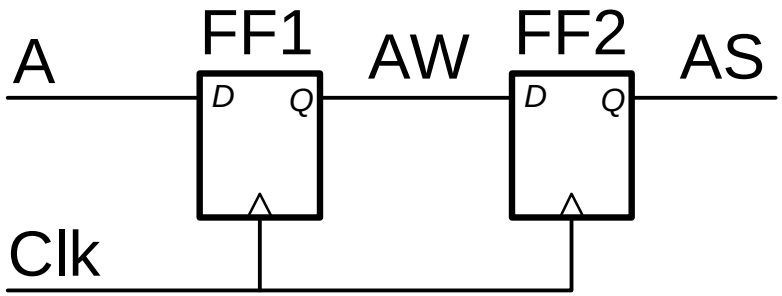
- The difficulty of synchronization depends on the relationship between events on the signal and events on the clock
- Synchronous
  - signal events always happen outside of the clock's keep-out region
    - same clock
- Mesochronous
  - signal events happen with a fixed but unknown phase relative to 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 pleisochronous
    - signal is synchronized to some periodic clock
- Asynchronous
  - signal events may occur at any time

Synchronization Hierarchy Summary

Type	Frequency	Phase
Synchronous	Same	Same
Mesochronous	Same	Constant
Plesiochronous	Small Difference	Slowly Varying
Periodic	Different	Periodic Variation
Asynchronous	N/A	Arbitrary

The Brute-Force Synchronizer

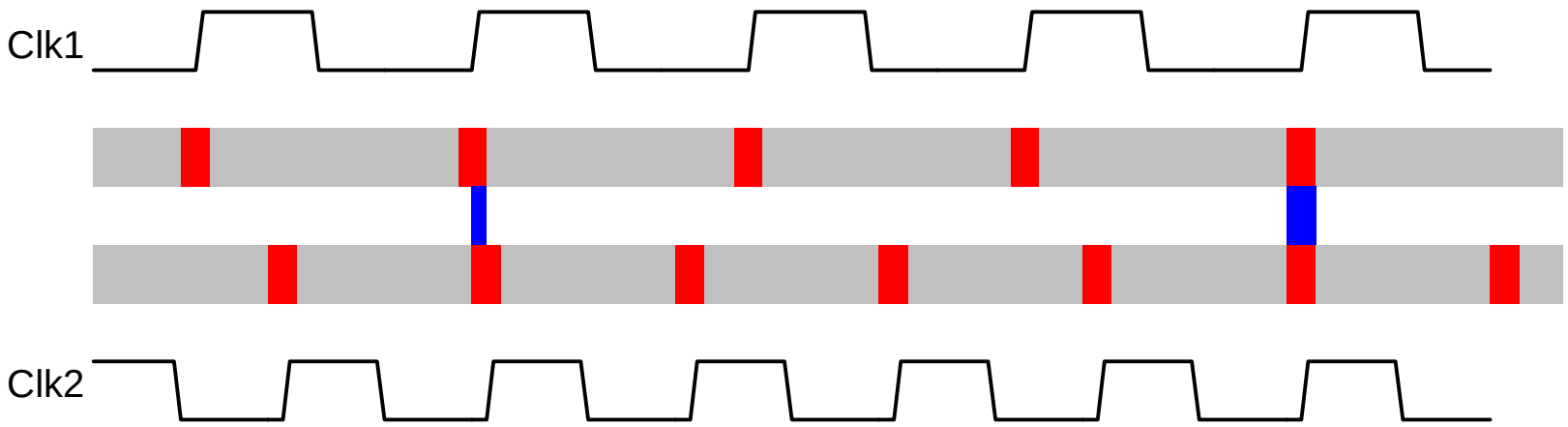
- How do we compare synchronizers?
  - synchronizer delay (in addition to the required $t_{cy}/2$)
  - failure rate
- For the brute-force synchronizer
  - $t_d=t_w+2(t_s+t_{dCQ})$
  - $f_f=t_a f_e f_{cy}\exp(-t_w/?_s)$
- Can we do better?



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

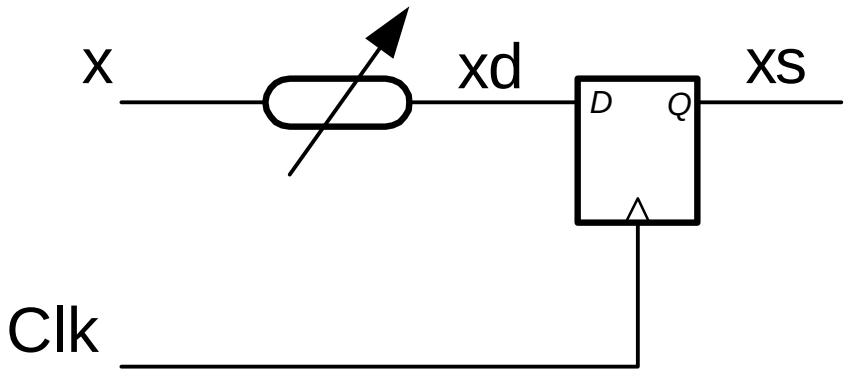


Mesochronous Synchronization

- 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

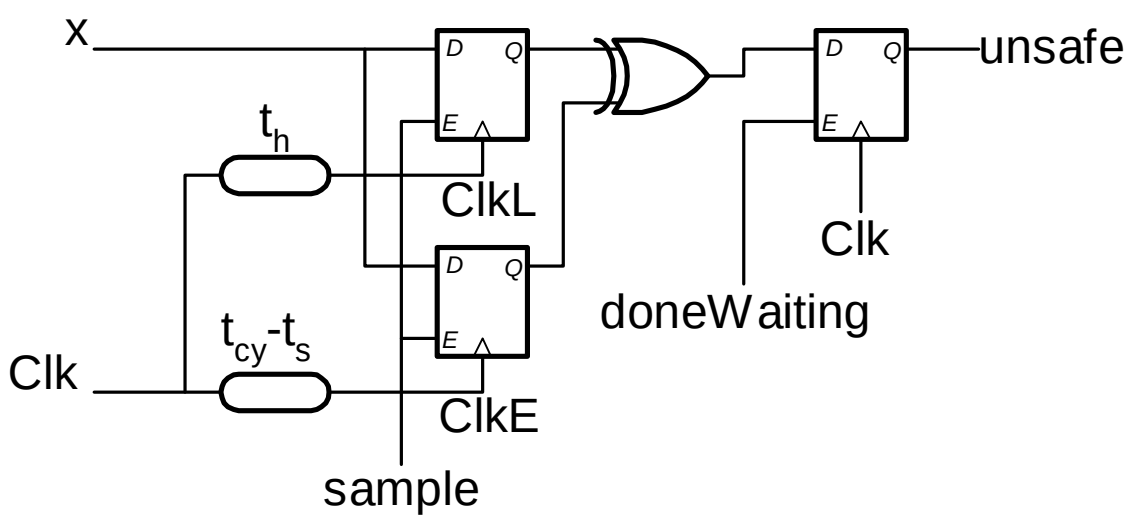
Delay-Line Synchronizer

- 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?
- What is the delay of this synchronizer?
- Do we need the flip-flop?



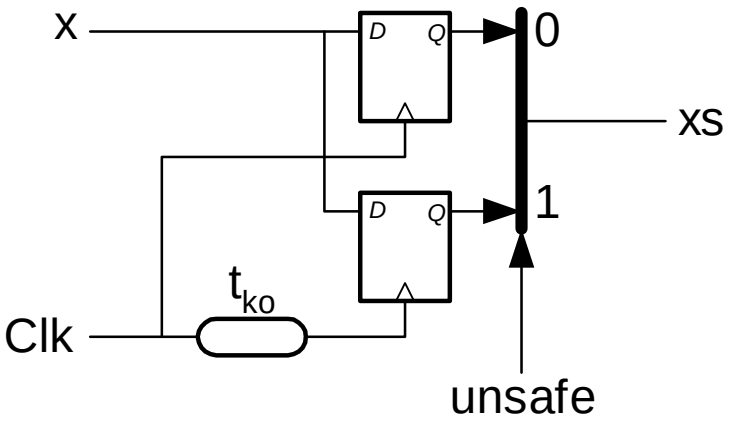
Detecting an Unsafe Signal

- 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



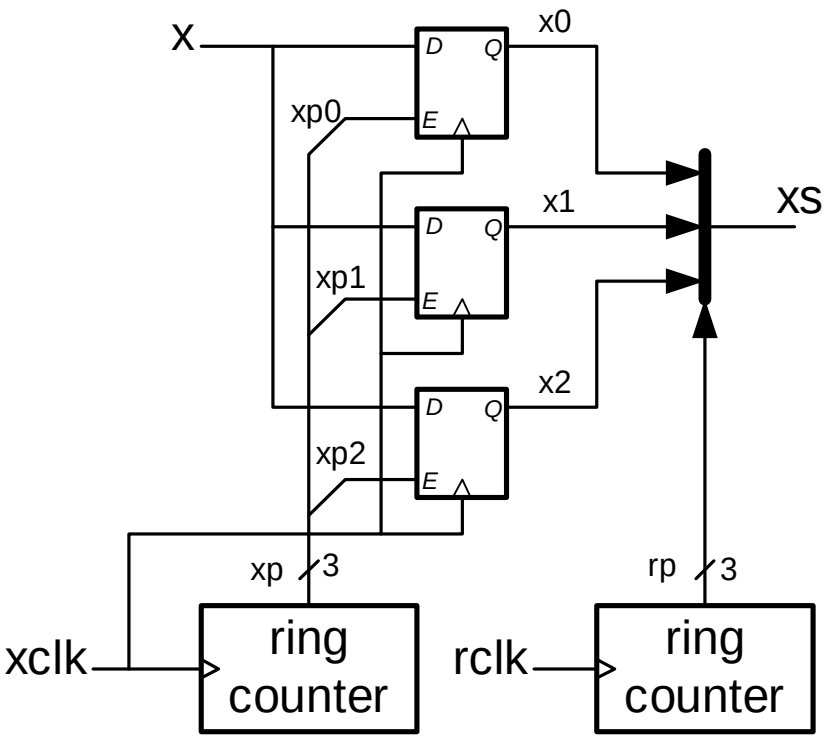
Two-Register Synchronizer

- The delay-line synchronizer has two problems
  1. Its 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?

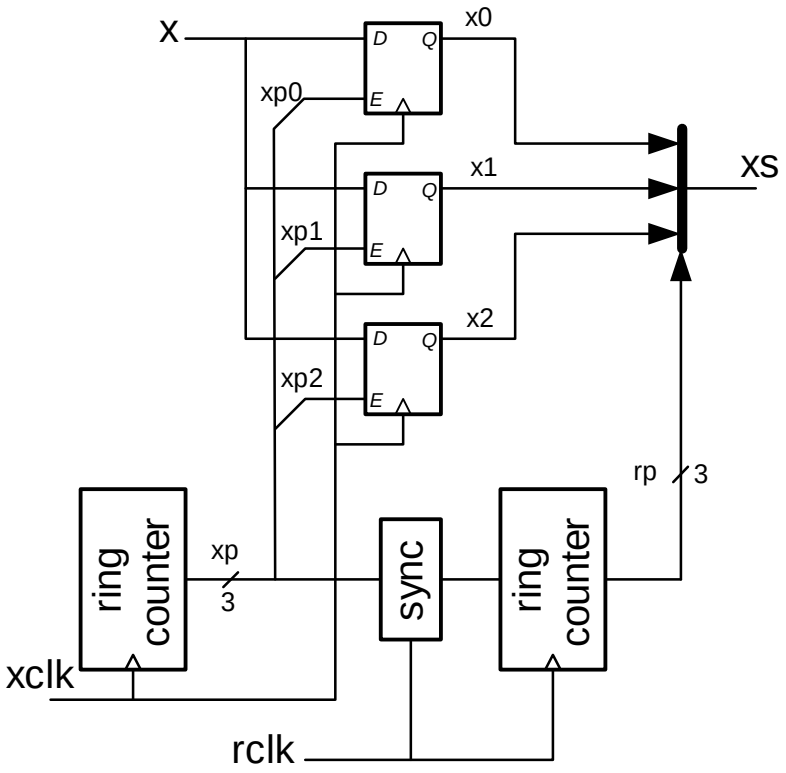


Plesiochronous Timing

- With plesiochronous timing, one clock is running slightly faster than the other
  - e.g., system with independent crystal oscillators with same nominal frequency (?200ppm)
- The same basic synchronizer types apply
  - delay line
  - two-register
  - FIFO
- But...
- we need to resynchronize periodically
  - e.g., once every 1,000 clocks
- we need flow control
  - have to match data rate of tx and rx even if clock rate is different
  - eventually the phase *wraps* and we either get 2 or 0 data elements during a particular clock
    - unless we make sure we are not sending data when the phase wraps

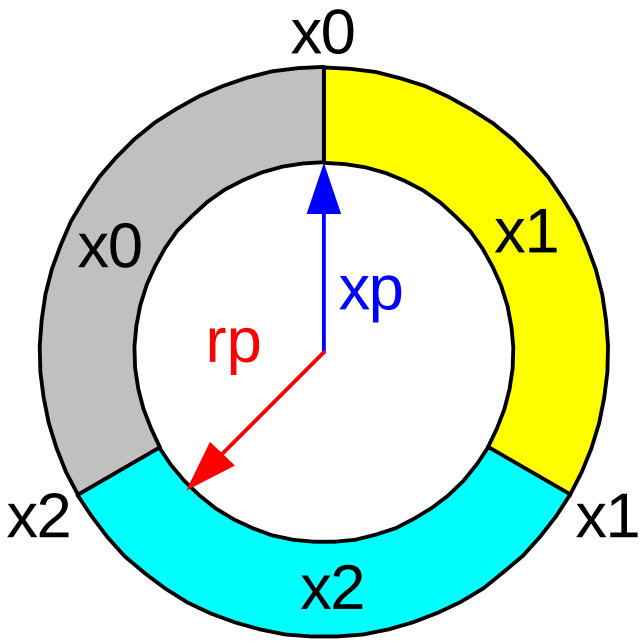
A Plesiochronous FIFO Synchronizer

- Insert data with transmit clock (xclk)
- Remove data with receive clock (rclk)
- Periodically update the receive pointer (rp) by synchronizing the transmit pointer (xp) to the receive clock
  - how do we know when to do this?
  - what do we do if rp increments by 2 or 0 when we update it?



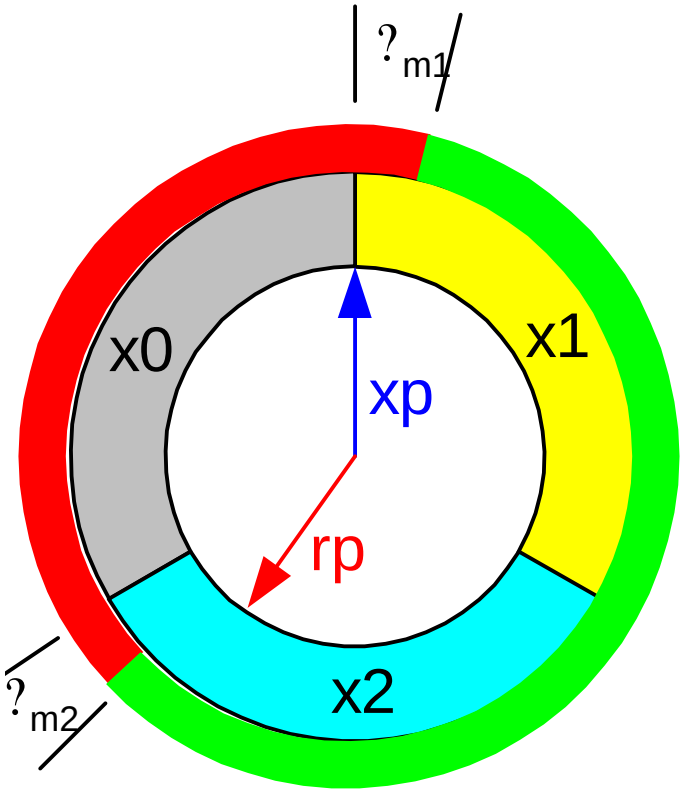
Phase-Slip Detection
Pointers on a Clock

- Think of xp and rp as hands on a clock
- When xp reaches 12:00, 4:00, and 8:00 data is clocked into x0, x1, and x2 respectively
- When rp is between 8:00 and 12:00 x0 is selected, from 12:00 to 4:00 x1 is selected, and from 4:00 to 8:00 x2 is selected. These values are sampled at the ends of these intervals
- What relative angles between rp and xp are *legal*?



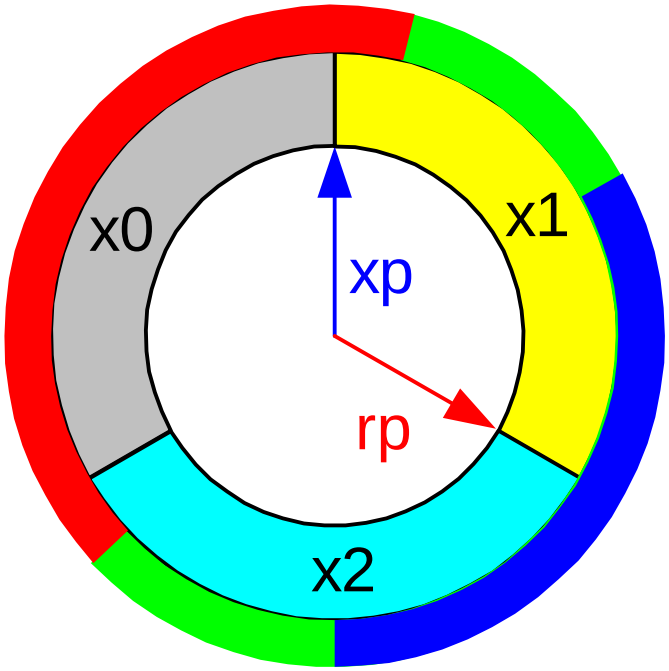
Phase Slip Detection
The Keepout Region

- When xp is straight up (sampling into x0), rp must be in the green region
  - otherwise data will change while its selected
  - ? $?_{m1}$ includes phase drift between updates and t_{dCQ} less t_{cmux}
  - ? $?_{m2}$ includes phase drift less t_{cCQ}
- Actually OK for data to change while selected as long as stable at end of selection
- To detect phase, sample xp with appropriate phase of rclk
  - advanced by $?_{m1}$ to detect fast receiver
  - retarded by $?_{m2}$ to detect fast transmitter
  - or just update to fastest legal time



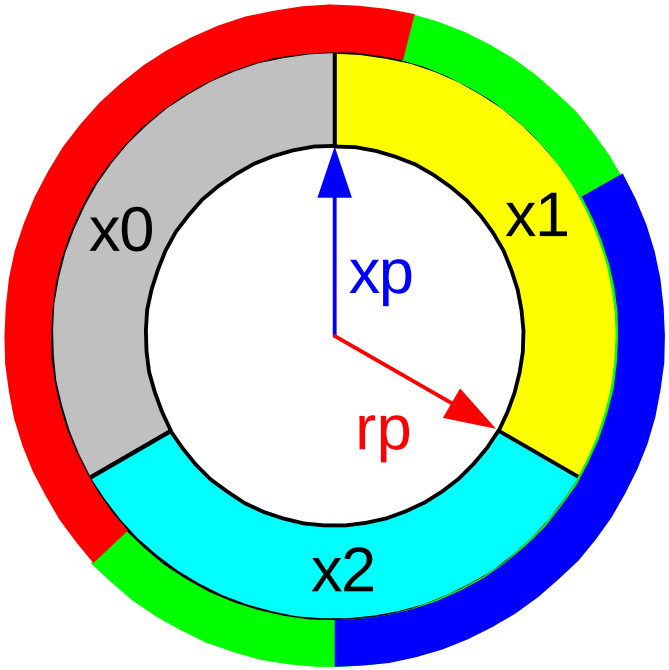
Phase Slip Detection
A Simple Approach

- If we make both margins 60°, we can just sample xp with rclk' and delay 1.5 cycles.
- At any point we want sampled xp to be two steps ahead of rp
  - so we can update rp to be sampled xp on positive edge of rclk
    - after waiting 1.5 + 3n rclks for synchronization
  - this approach may *dither* when right on the boundary



Data-Rate Mismatch

- Whichever approach we use, an update of RP may
  - skip a count 0,1,2,1,2,0,1,2,0,1,2
  - double a count 0,1,2,0,0,1,2,0,1,2,0
- If we aren't careful we can drop or duplicate a symbol
- One approach is to only update rp when the data contains a *null* symbol
  - OK to drop or duplicate
- Alternatively we can design the receiver to accept 0,1, or 2 symbols per clock



Open-Loop and Closed-Loop Flow Control

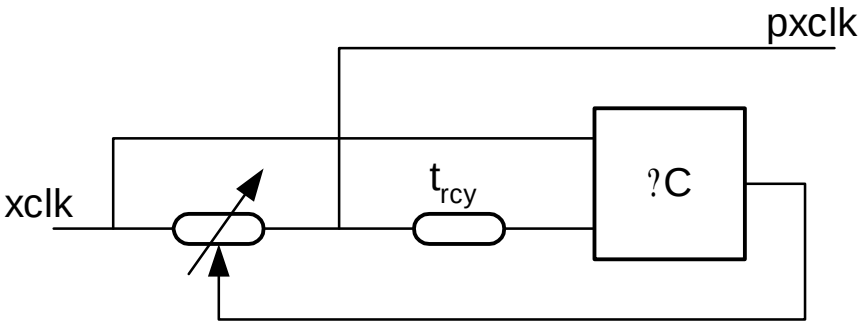
- We need to keep a fast transmitter from overrunning a receiver
- (or a slow transmitter from underrunning the receiver)
- Open-loop approach
  - insert *lots* of *nulls* into the data stream at the transmitter
  - enough so that rate of non-nulls is less than the rate of the slowest possible receiver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

Next Time

- Power Distribution