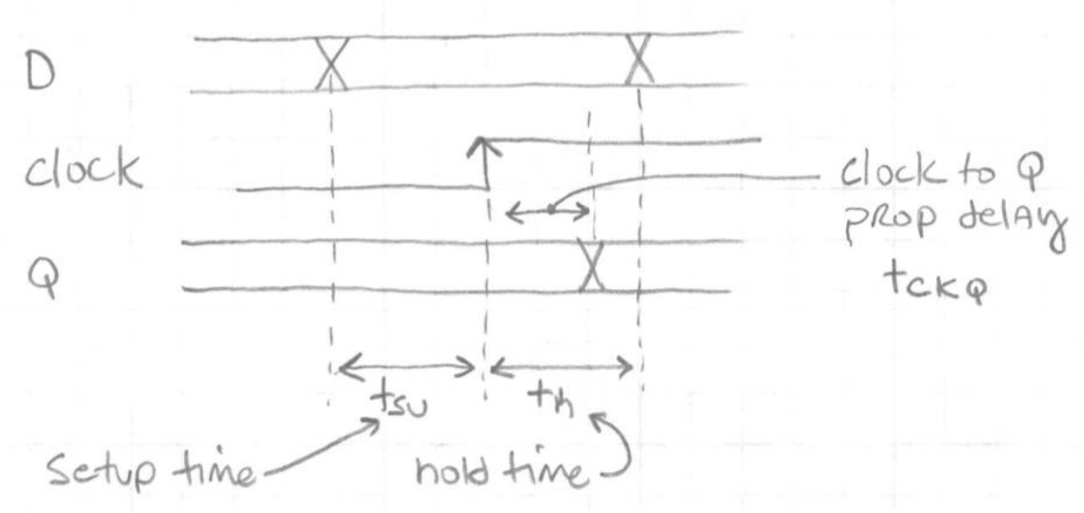
Metastability & Clock Domain Crossing Techniques

Gabriel Cojocaru

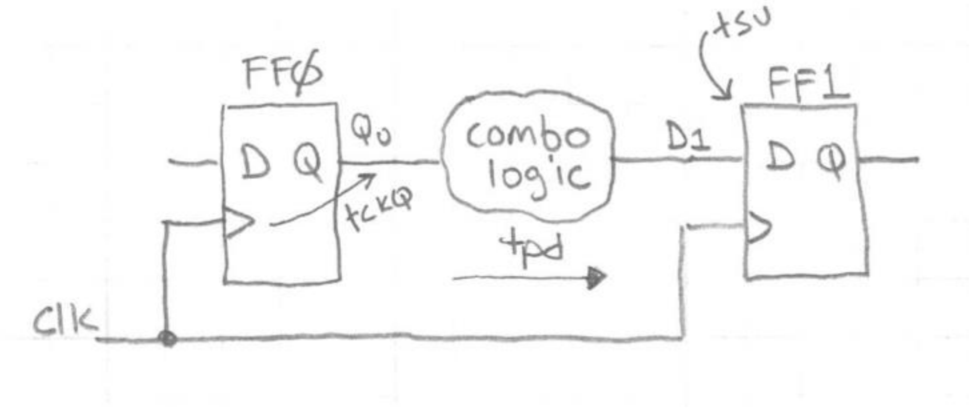
Review of Flip Flop Setup and Hold Time

- Considering D-type edge-triggered flip flops
- Just before and just after the clock edge, there is a critical time region where the D input must not change
- The region just before the clock edge is called **setup time** (tsu)
- The region just after the clock edge is called **hold time** (th)
- Every FF has minimum required values for tsu and th



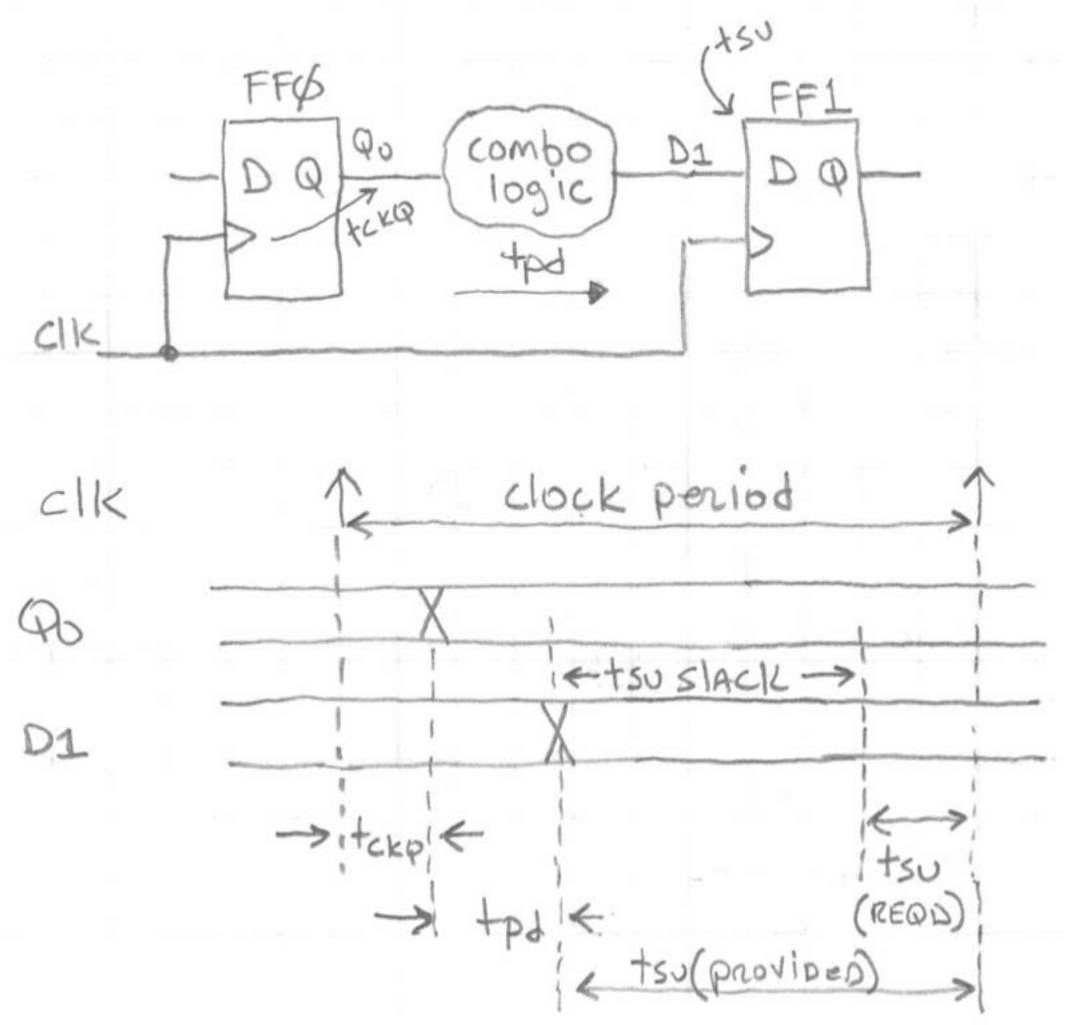
Review of Flip Flop Setup and Hold Time

- Synchronous circuit minimum cycle time is effected by setup time
- Between clock edges, the path between two FFs is composed of
  - clock to Q delay of FF0 (tckq)
  - propagation delay through combo logic (tpd)
  - the required setup time (tsu) of FF1



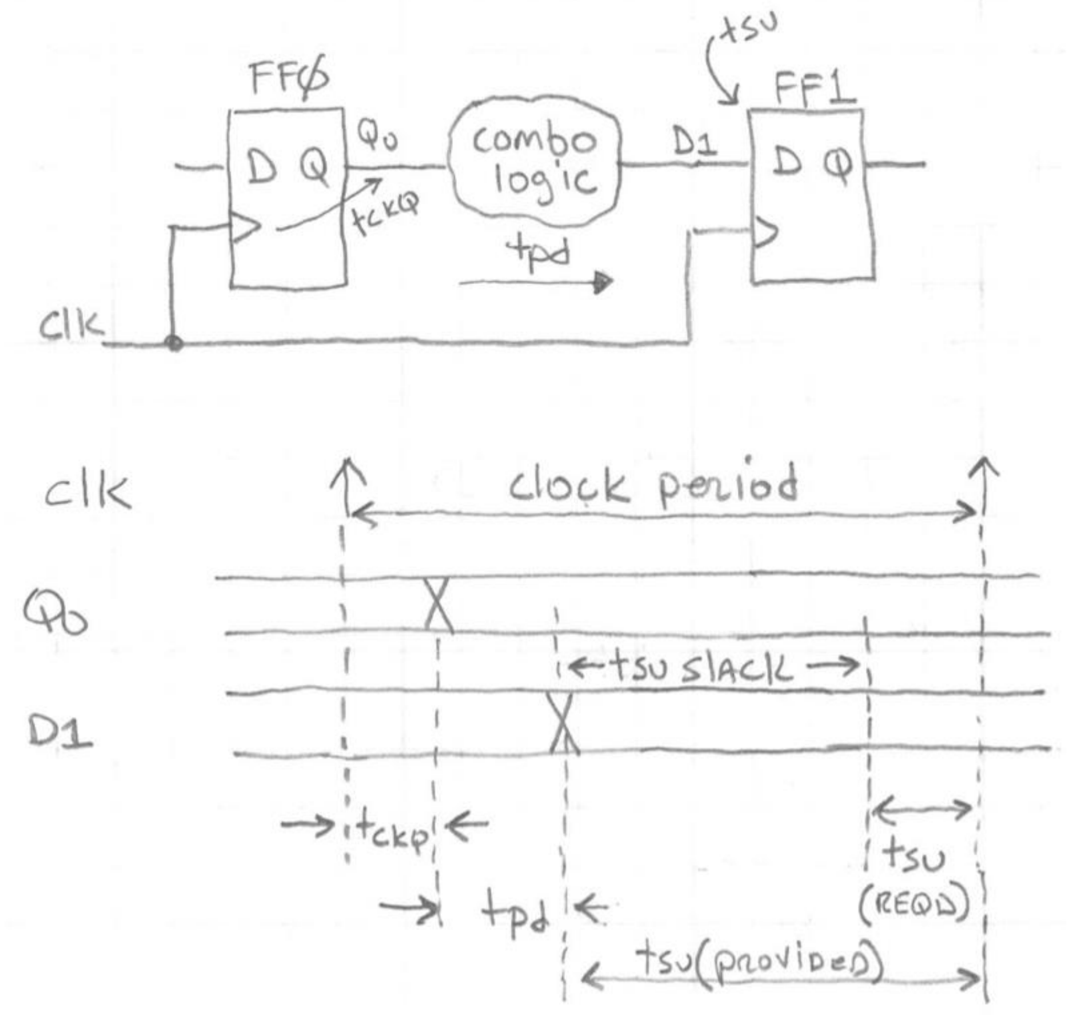
Review of Flip Flop Setup and Hold Time

- The circuit provides setup time as a function of the clock period Tckq, and the combinatorial logic delay Tpd
- The flip flop requires a certain Tsu to operate correctly
- The difference between the provided setup time and the required setup time is called **setup time slack**
    - Positive slack is good



Review of Flip Flop Setup and Hold Time

- In the same fashion, the flip flop requires the data at the D input be stable for a certain Th period after the clock edge to operate correctly
- The hold time is a function of the Tckq + Tpd (clock-to-Q and propagation delay)
- Hold time slack is the difference between the provided hold time and the required hold time

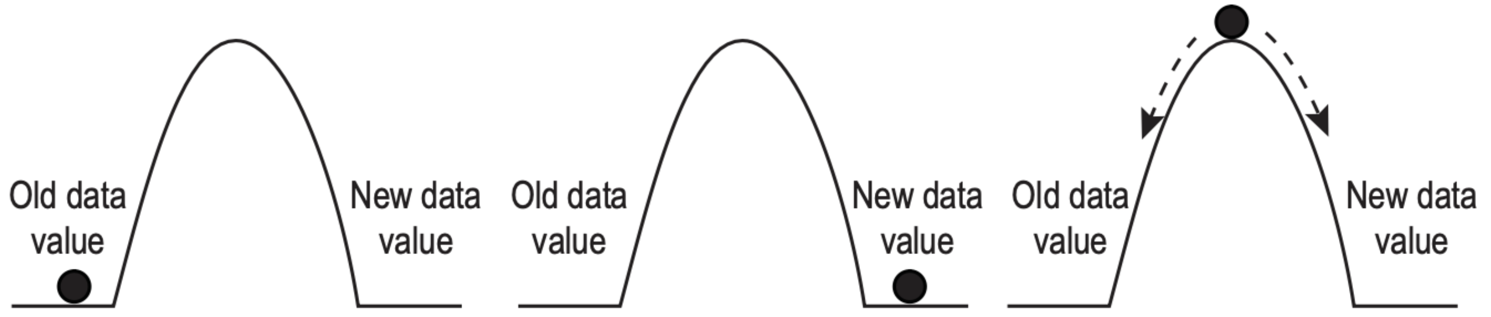


Metastability

- In modern ICs, data is often shared between IPs and functional blocks operating at different clock frequencies
- Many synchronous systems need to interface with asynchronous data sources
  - Interrupts from peripherals (I/O devices, key strokes, etc.)
  - Data transfers with different clock rates (PCIe, Ethernet, etc.)
  - Often these data sources have no known timing relationship to the cpu/system clock
- If not properly addressed, can cost the company a lot of money and someone their job!
- It is something that is often brought up in interviews

Metastability

- Refers to signals that aren't a stable 0 or 1 for a some period of time during normal operation
- Cannot be avoided, but the detrimental effects can be neutralized



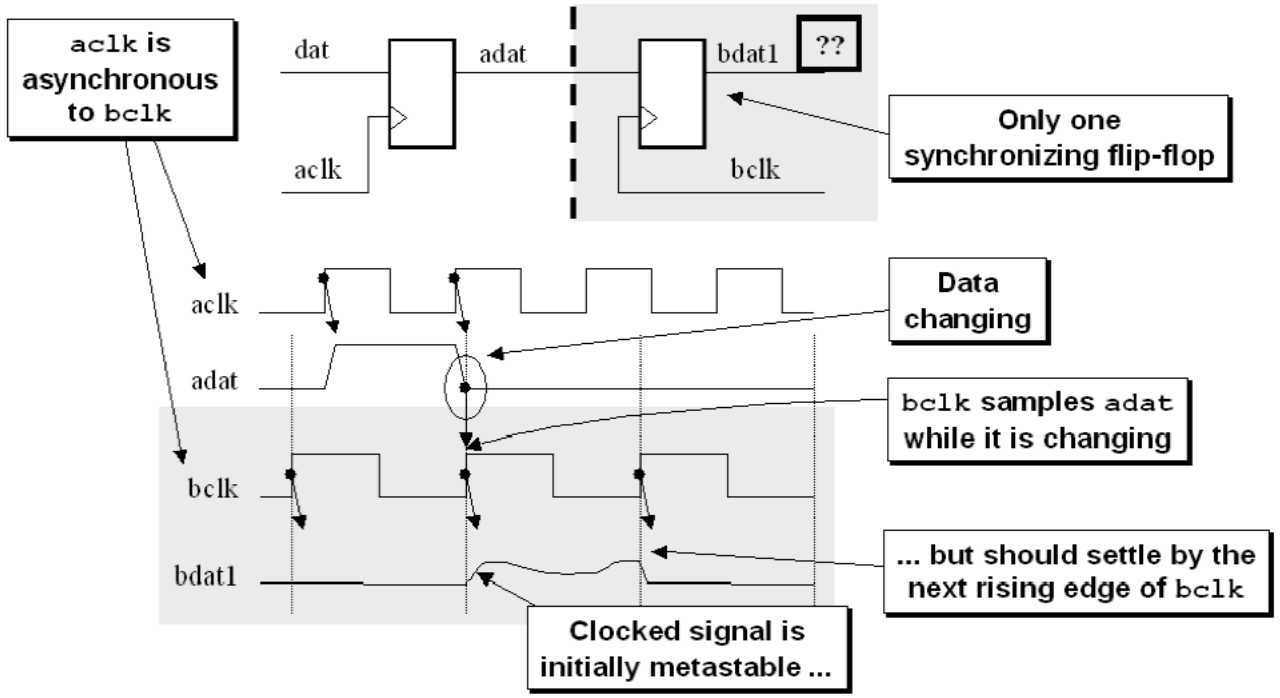
Signal transition occurs after clock edge and minimum t_H: Ball lands on the old data side.

Signal transition meets register t_{SU} and t_H: Ball lands on the new data side.

Signal violates register t_{SU} or t_H: Ball balances at top of hill or takes too long to reach the bottom. Output is metastable and violates t_{CO}.

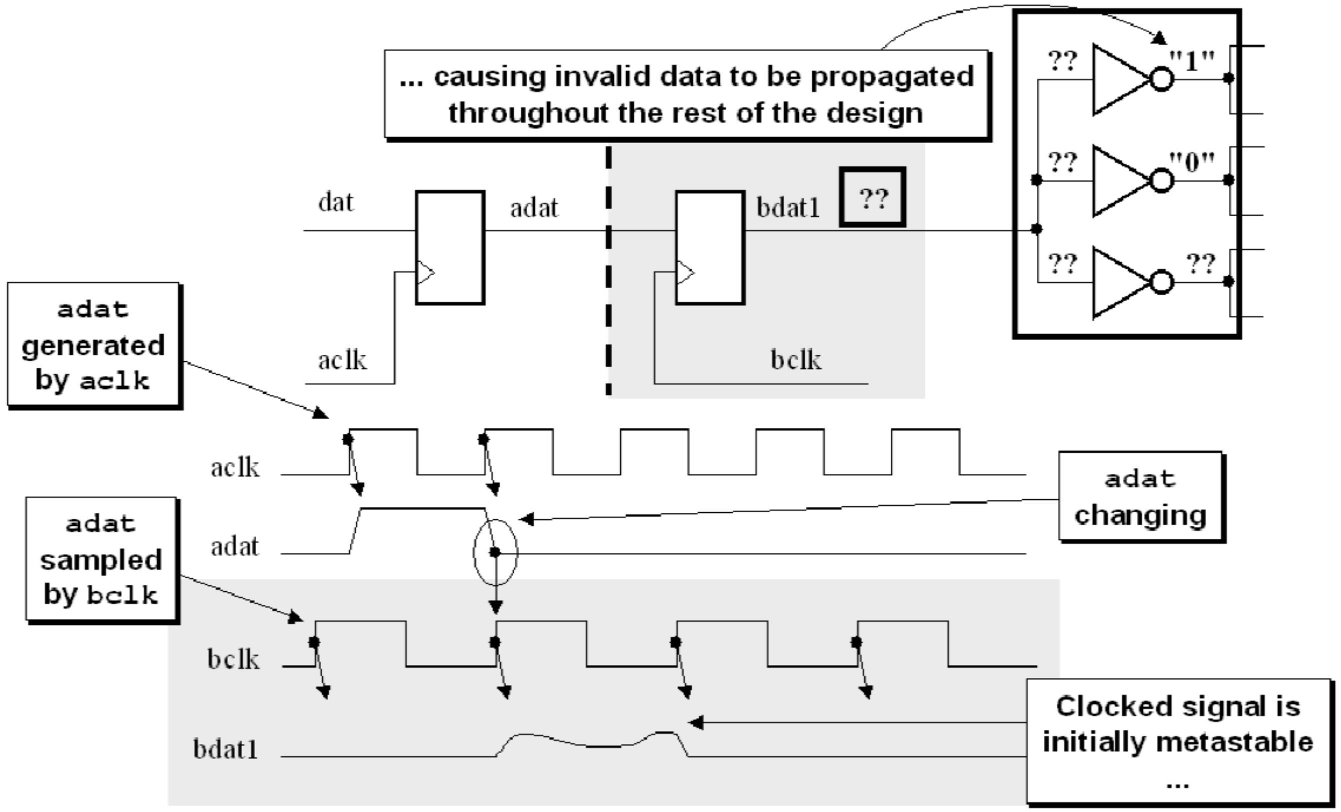
Metastability

- Signal generated in one clock domain that is changing too close to the rising edge of a clock in another clock domain causes the output to go metastable


aclk is asynchronous to bclk
dat
adat
aclk
bdat1
??
bclk
Only one synchronizing flip-flop
aclk
adat
bclk
bdat1
Data changing
bclk samples adat while it is changing
... but should settle by the next rising edge of bclk
Clocked signal is initially metastable ...

Metastability

- Why is it a problem?
  - Metastable outputs in the receiving clock domain can propagate to the rest of the design affecting state machines, control logic, etc.


... causing invalid data to be propagated throughout the rest of the design
??
"1"
??
"0"
??
??
dat
adat
bdat1
??
aclk
bclk
adat generated by aclk
aclk
adat changing
adat
adat sampled by bclk
bclk
Clocked signal is initially metastable ...
bdat1

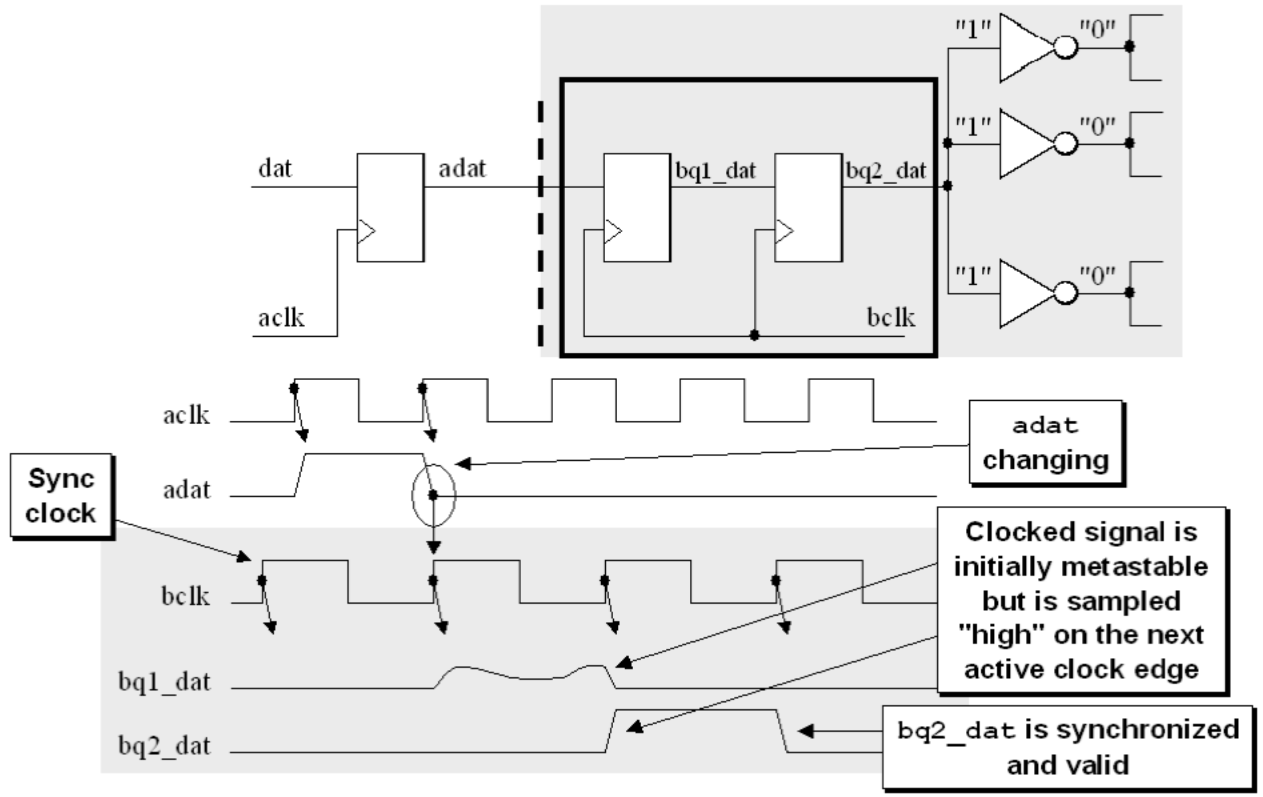
Metastability

- How do we solve this?
  - Synchronizers!

- Two scenarios
  - It is not necessary to sample every value in the receiving clock domain, but it is important that the sampled values are accurate. Ex: read and write gray code counters used in async FIFOs (which we will discuss later)
  - The signal needs to be properly acknowledged before it is allowed to change again

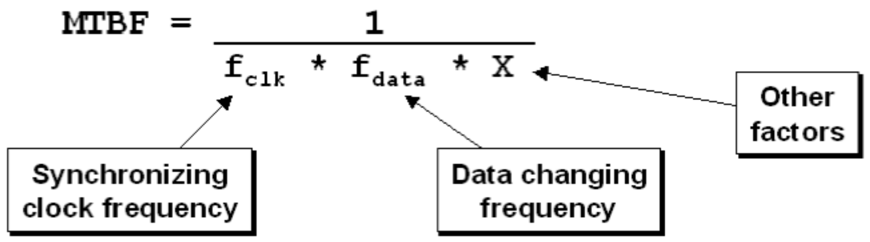
Two flip-flop synchronizer

- Most common synchronizer used by digital designers is a 2ff synchronizer


dat
adat
aclk
bq1_dat
bq2_dat
bclk
"1"
"0"
aclk
adat
bclk
bq1_dat
bq2_dat
Sync clock
adat changing
Clocked signal is initially metastable but is sampled "high" on the next active clock edge
bq2_dat is synchronized and valid

Two flip-flop synchronizer

- It is possible that the metastability will propagate through the second ff
- The probability of the time between synchronization failures (MTBF) is a function of multiple variables

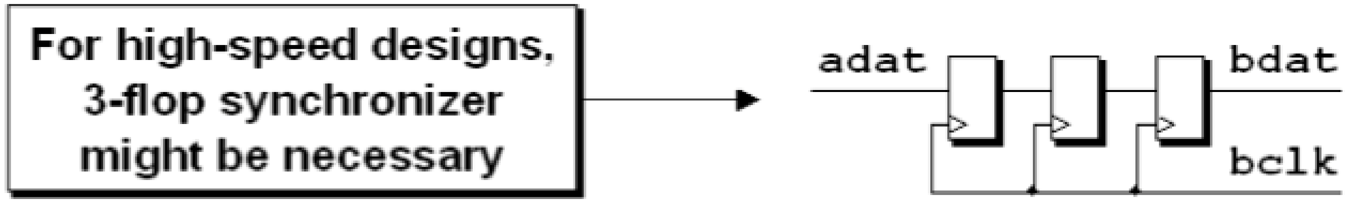


$$\text{MTBF} = \frac{1}{f_{clk} * f_{data} * X}$$

- We can see that high speed designs are more prone to synchronization issues

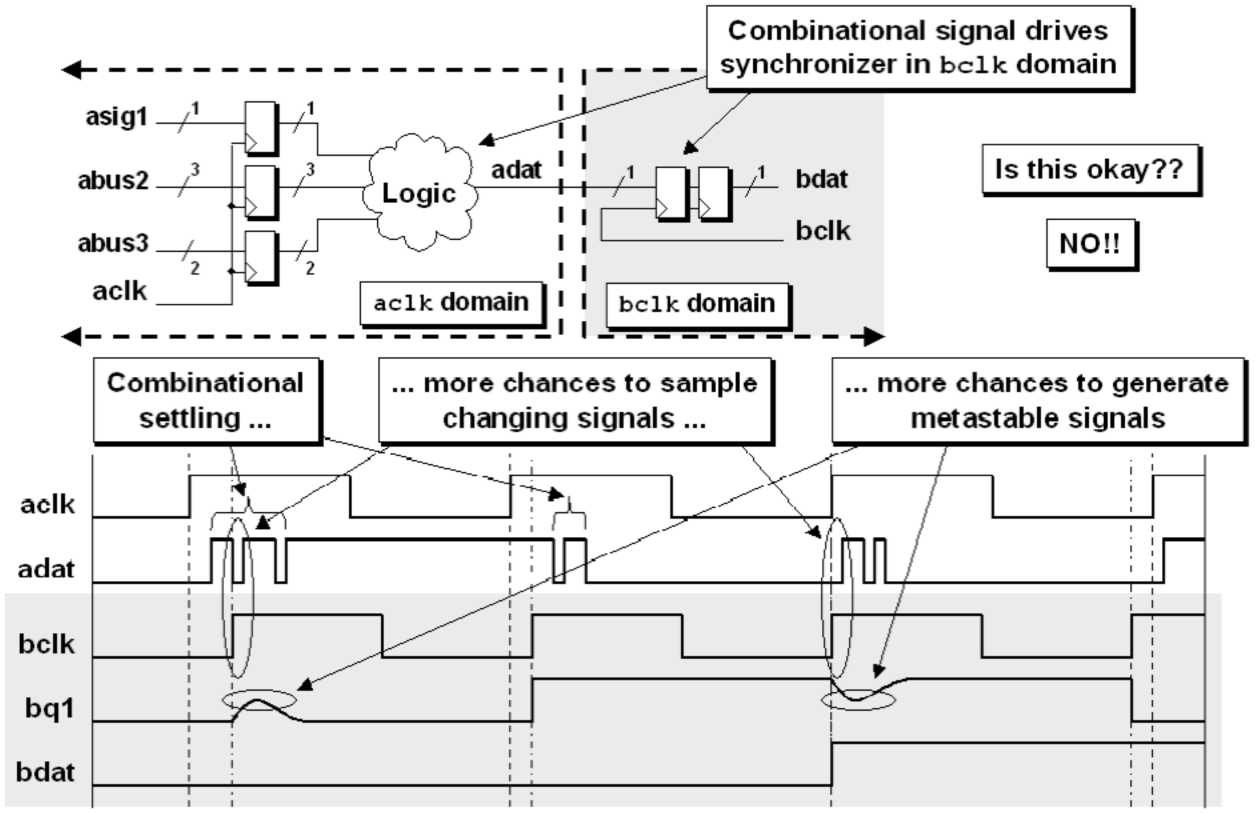
Three flip-flop synchronizer

- In very high-speed designs, MTBF of a two-flop synchronizer is too short, so a 3rd flop is added


For high-speed designs, 3-flop synchronizer might be necessary
adat
bdat
bclk

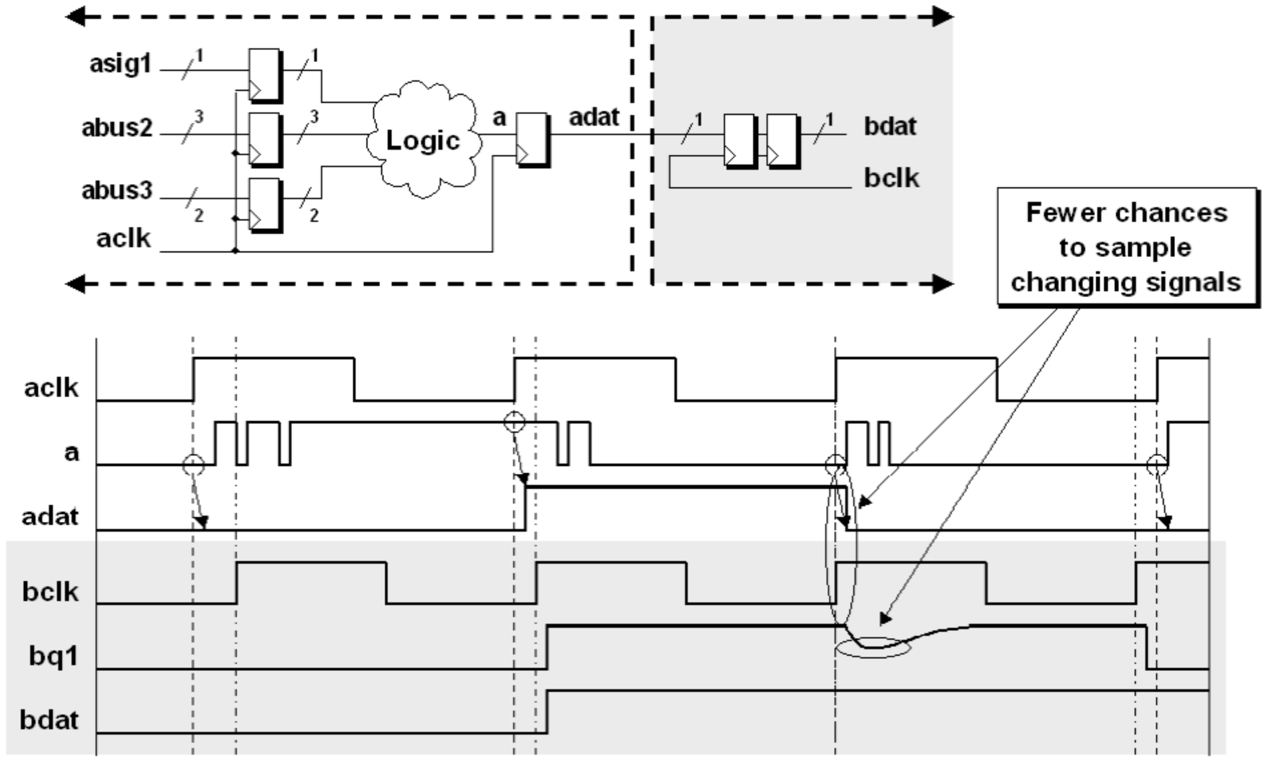
Synchronizing Signals From Sending Clock Domain

- Consider a case where signals in the sending clock domain aren't registered after passing through combinatorial logic as seen below
- Combinatorial settling increases the chance of data transitions or glitches on *adat*, reducing MTBF


Combinational signal drives synchronizer in bclk domain
asig1
abus2
abus3
aclk
Logic
adat
bdat
bclk
aclk domain
bclk domain
Is this okay??
NO!!
Combinational settling ...
... more chances to sample changing signals ...
... more chances to generate metastable signals
aclk
adat
bclk
bq1
bdat

Synchronizing Signals From Sending Clock Domain

- Therefore, signals that cross clock domains should first be sampled in the sending clock domain as shown here



Requirements for proper CDC

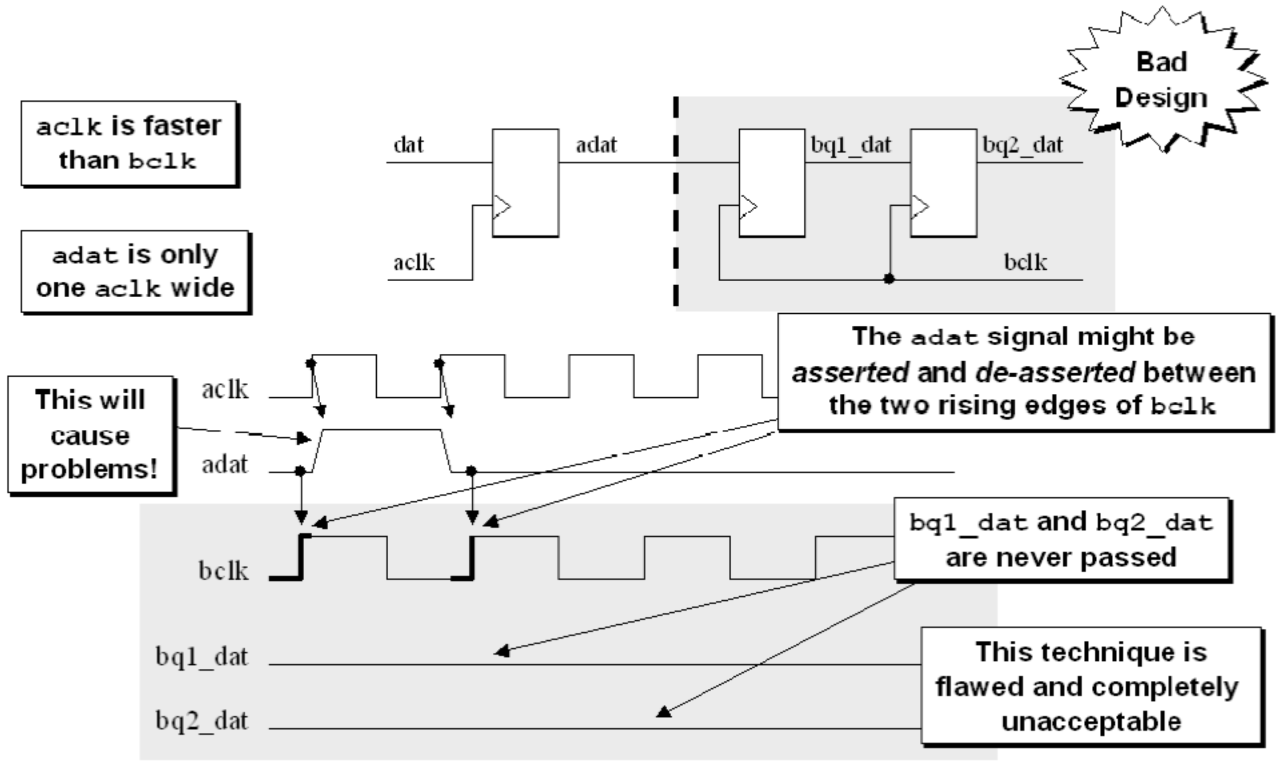
- Synchronizing slower signals into a faster clock domain is generally not an issue (at least 1.5X)
  - The signal will be sampled one or more times in the receiving clock domain
  - Two flip-flop synchronizers can be used for single bit signals
- Sampling faster signals in a slower clock domain introduces some problems

Synchronizing Fast Signals Into Slow Clock Domain

- One issue with synchronizers is the possibility that a signal in the sending clock domain changes multiple times before it can be sampled in the receiving clock domain
- When missing samples is not an option, two approaches are possible
  - Open-loop solution – capturing signals without acknowledgement
  - Closed-loop solution – passing back an ack signal into the sending clock domain

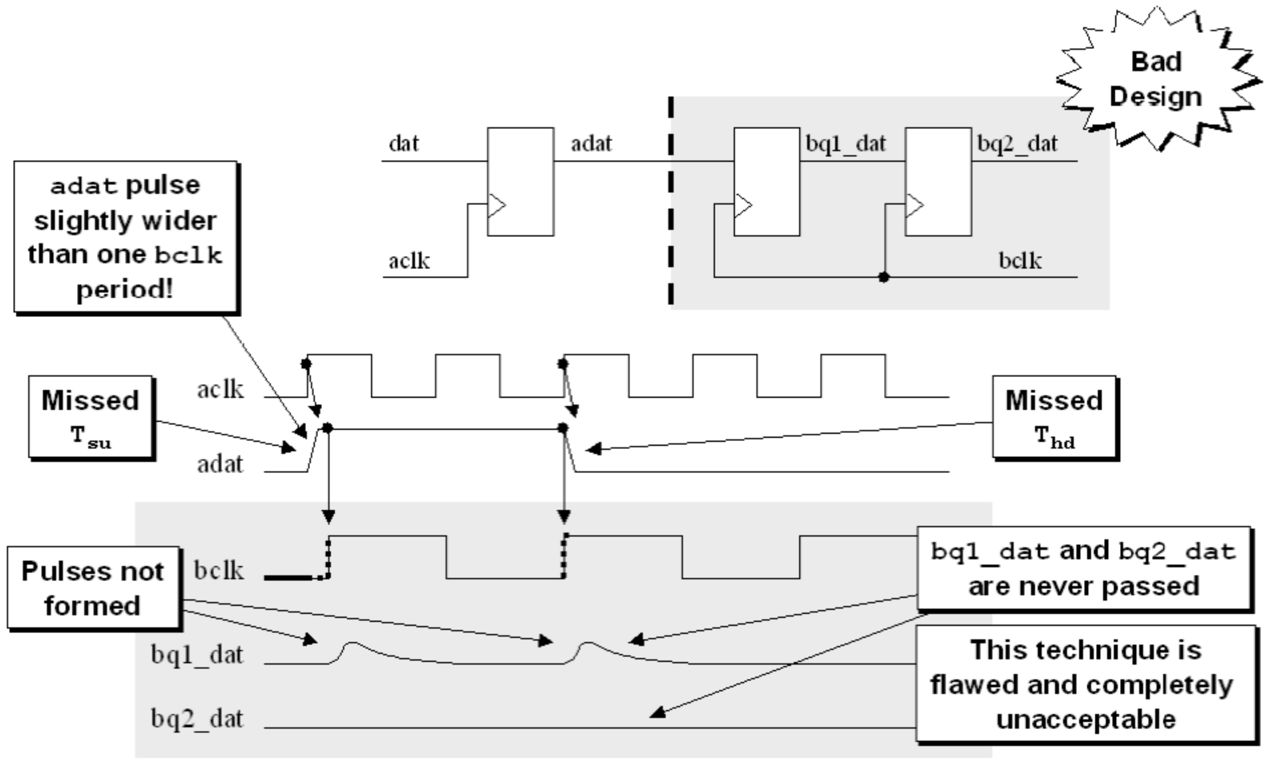
Problem 1: Fast Pulse

- The pulse can go low -> high -> low in the sending clock domain and be completely missed in the receiving clock domain


Bad Design
aclk is faster than bclk
adat is only one aclk wide
dat
adat
bq1_dat
bq2_dat
aclk
bclk
This will cause problems!
The adat signal might be *asserted* and *de-asserted* between the two rising edges of bclk
aclk
adat
bclk
bq1_dat
bq2_dat
bq1_dat and bq2_dat are never passed
This technique is flawed and completely unacceptable

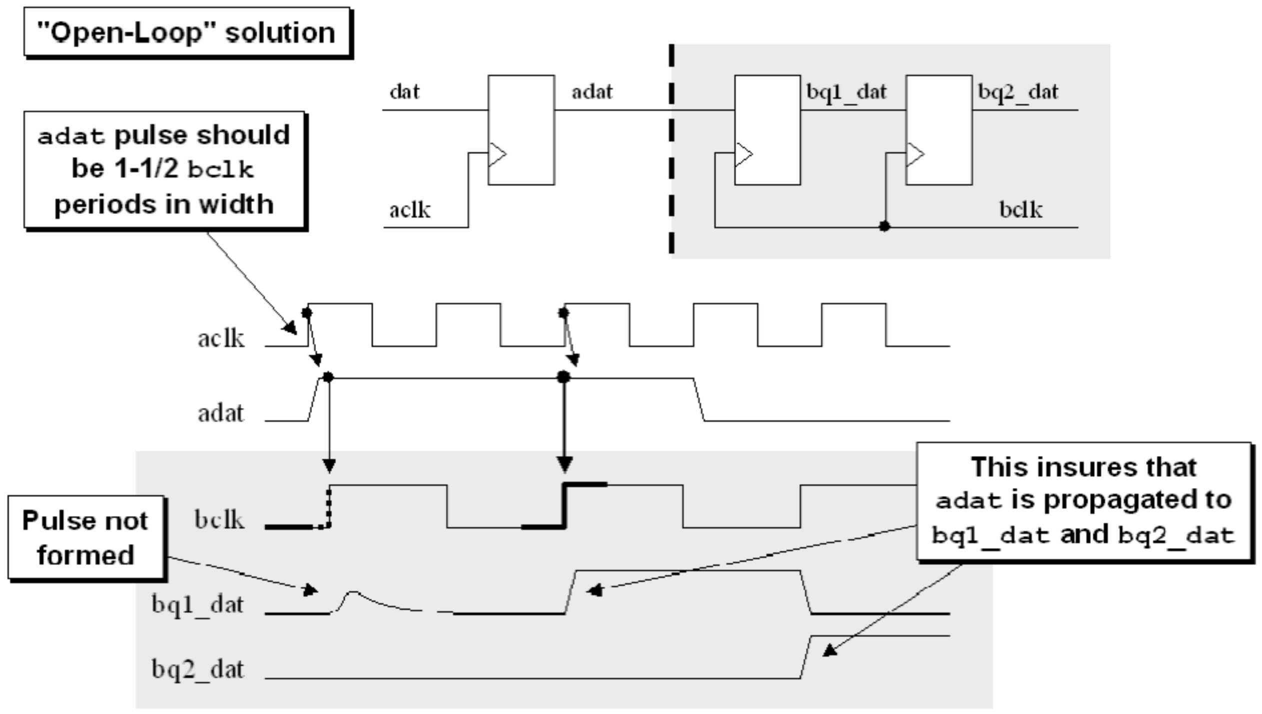
Problem 1: Long Pulse (but not long enough)

- If the pulse is slightly wider than the period of the receiving clock, although rarely, it can miss setup/hold times as shown


Bad Design
dat
adat
aclk
bq1_dat
bq2_dat
bclk
adat pulse slightly wider than one bclk period!
Missed T_{su}
aclk
adat
Missed T_{hd}
Pulses not formed
bclk
bq1_dat
bq2_dat
bq1_dat and bq2_dat are never passed
This technique is flawed and completely unacceptable

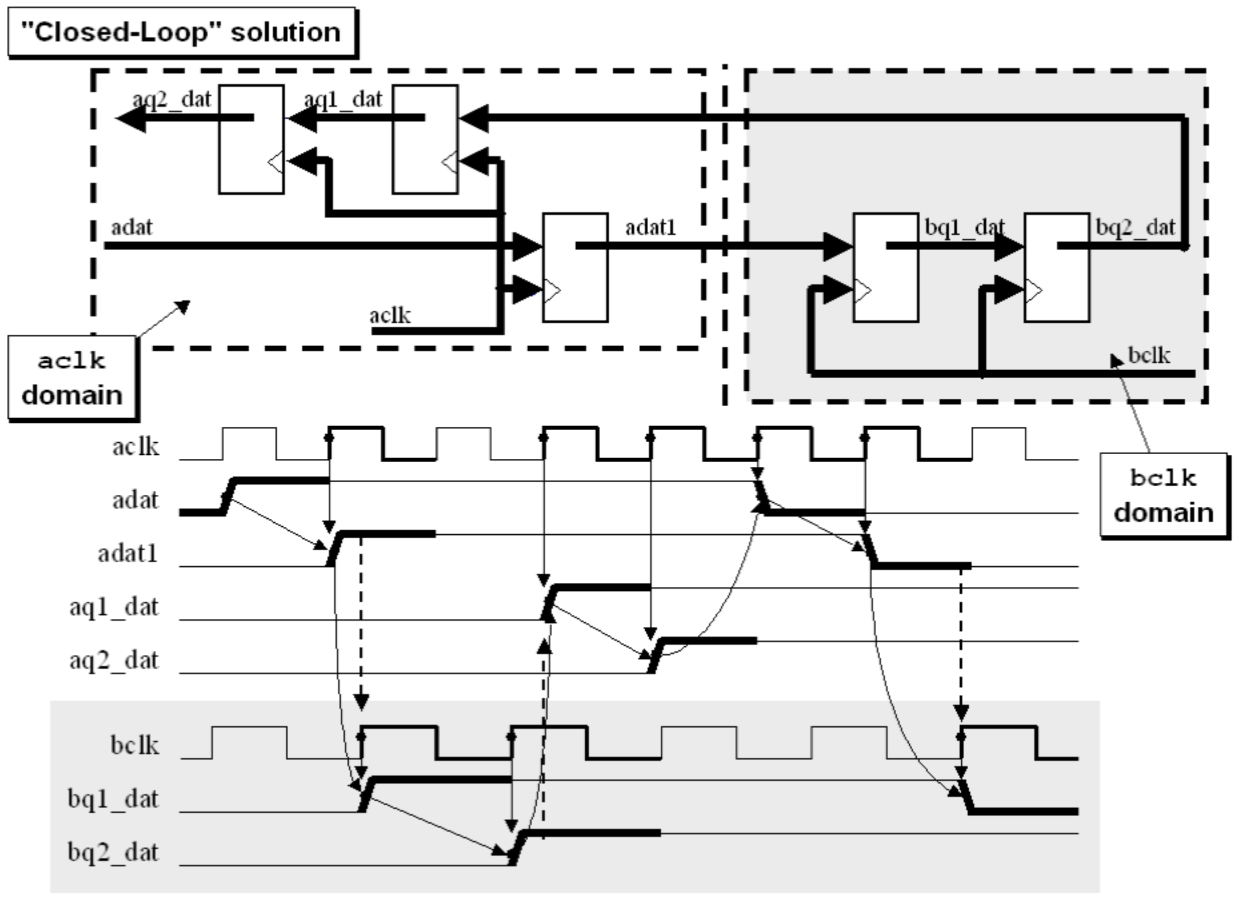
Open-loop solution – 2ff synchronizer

- Assert the CDC signal for a period of time longer than the period of the receiving clock


"Open-Loop" solution
dat
adat
aclk
bq1_dat
bq2_dat
bclk
adat pulse should be 1-1/2 bclk periods in width
aclk
adat
Pulse not formed
bclk
bq1_dat
bq2_dat
This insures that adat is propagated to bq1_dat and bq2_dat

Closed-loop solution

- Send a synchronized acknowledgment signal
- Safe technique to ensure the control signal was sampled correctly
- Introduces delay due to the extra synchronization logic



Multi-bit Signals CDC

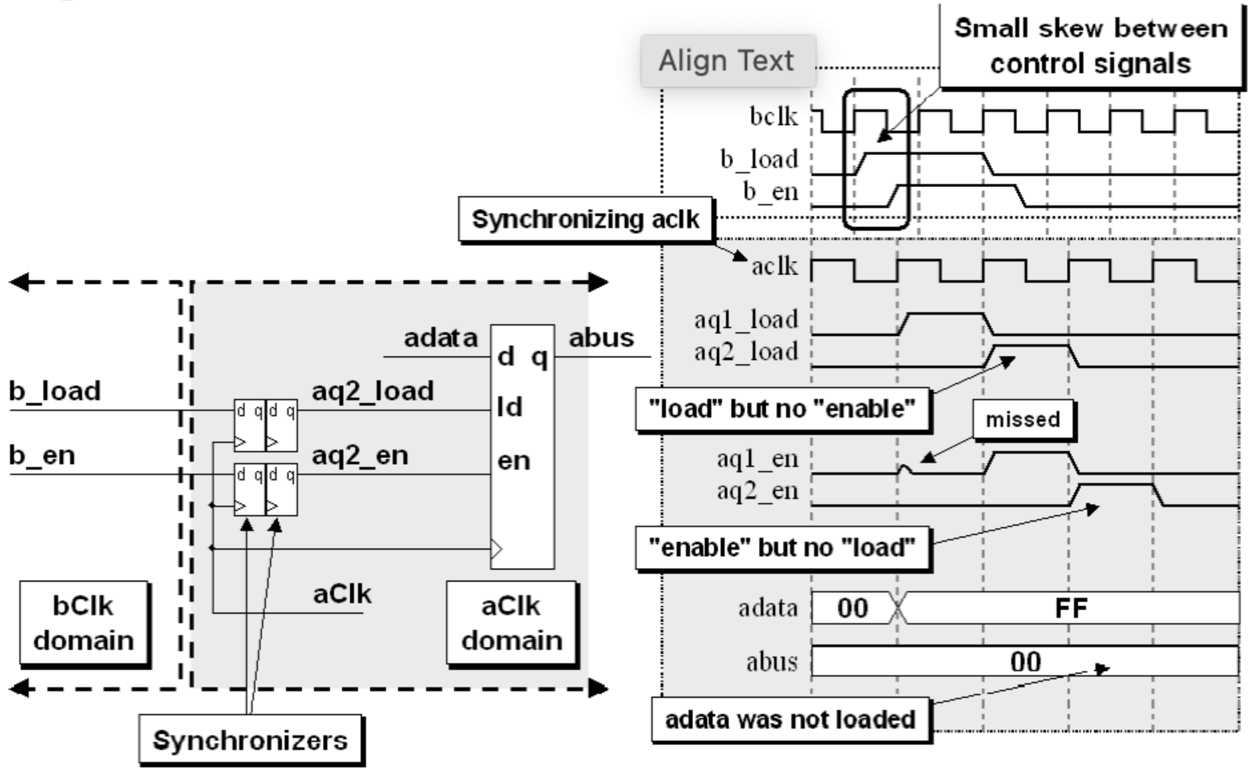
- Simple synchronizers do not guarantee safe data delivery
- In addition to signal synchronization, it is also important to ensure synchronized sampling of multi-bit data
- Multi-bit signals synchronized to one clock will experience data changing skews that might cause bits to be sampled on different rising edges in the new clock domain
    - Even with perfect trace length matching, difference in rise/fall times and process variations across a die introduce enough skew

Multi-bit CDC Strategies

- Multi-bit signal consolidation
  - When possible, consolidate multiple bits into one before crossing clock domains
- Multi-Cycle path formulation
  - Synchronized load signal to pass multiple unsynchronized CDC bits
- Pass multiple CDC bits using gray codes

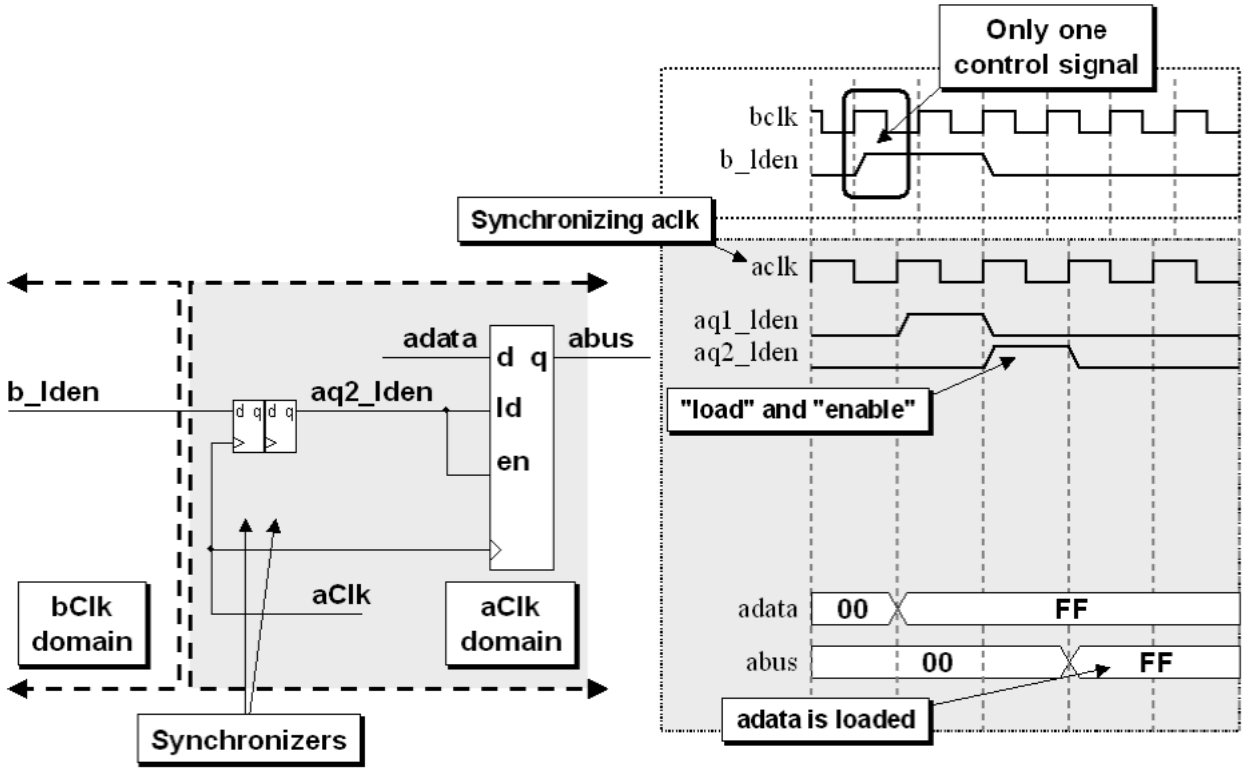
Bit Consolidation – Potential Issue

- If both control signals are needed to load data value into a register, but one of them synchronized on a different rising edge (due to skew), data would not be loaded into the register



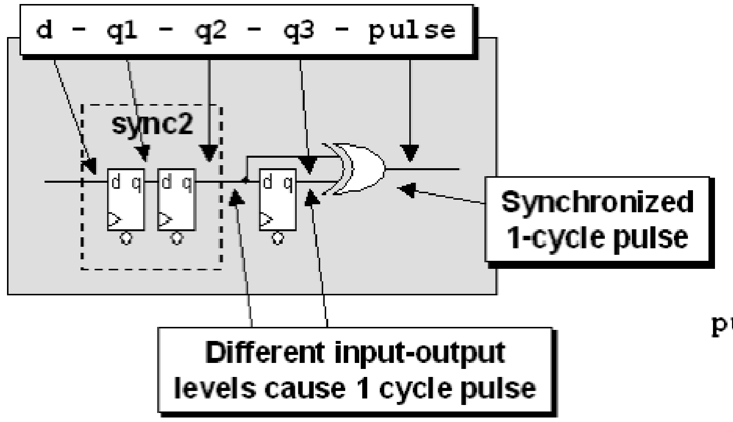
Bit Consolidation – Solution

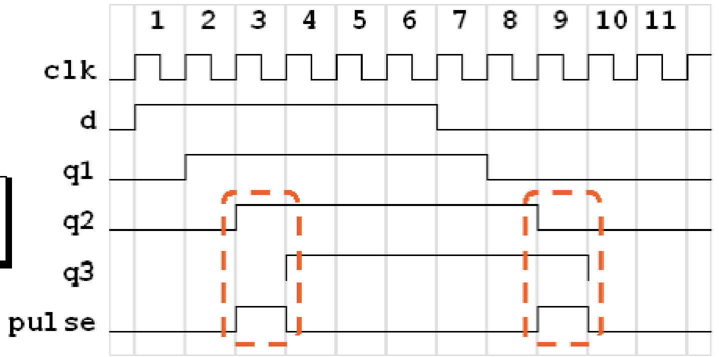
- Consolidate both signals into one b_lden and synchronize it



Multi-Cycle Path Formulation

- Simultaneously send unsynchronized data to a receiving clock domain, paired with a synchronized control signal
- Advantages:
  - The sending clock domain does not need to calculate the appropriate pulse width between clock domains
  - Only one control signal needs to be synchronized
- The control signal is passed through a synchronized pulse generator


d - q1 - q2 - q3 - pulse
sync2
d q
d q
d q
Synchronized 1-cycle pulse
Different input-output levels cause 1 cycle pulse



Multi-bit CDC using FIFOs

- Passing multiple bits, either data or control bits, can be done using asynchronous FIFOs
- An Async FIFO is a shared memory or register buffer where data is inserted in the write clock domain and read in the receiving clock domain
- Most of the hard work in designing an Async FIFO is done through the synchronization of the read and write pointers

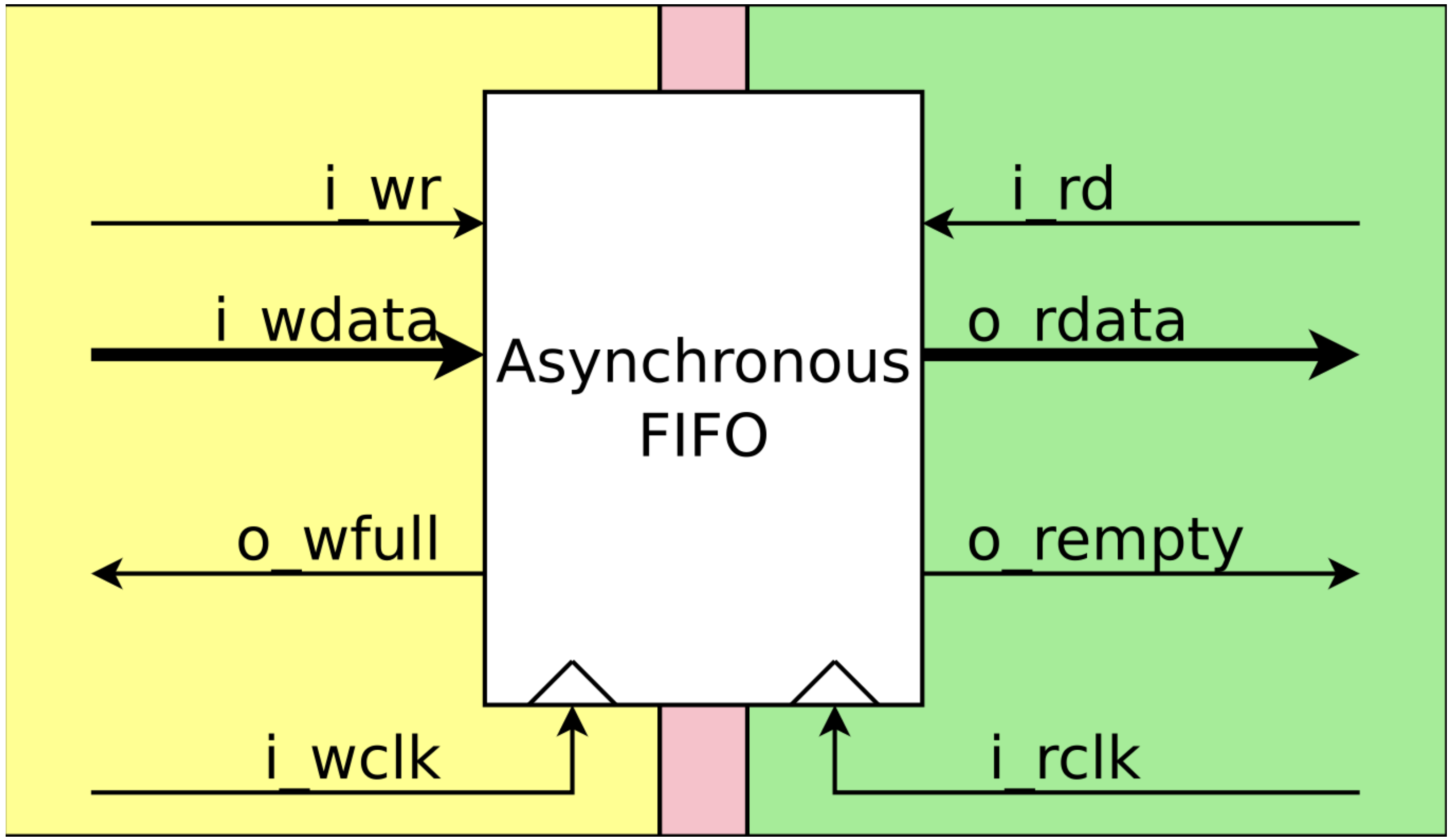
A bit on Gray Codes

- Gray code is a binary code where each successive value differs from the previous one by only one bit: 000,001,011,010…
- Eliminates the need with trying to synchronize multiple changing CDC bits
- Binary to Gray conversion can be simply be done like this:

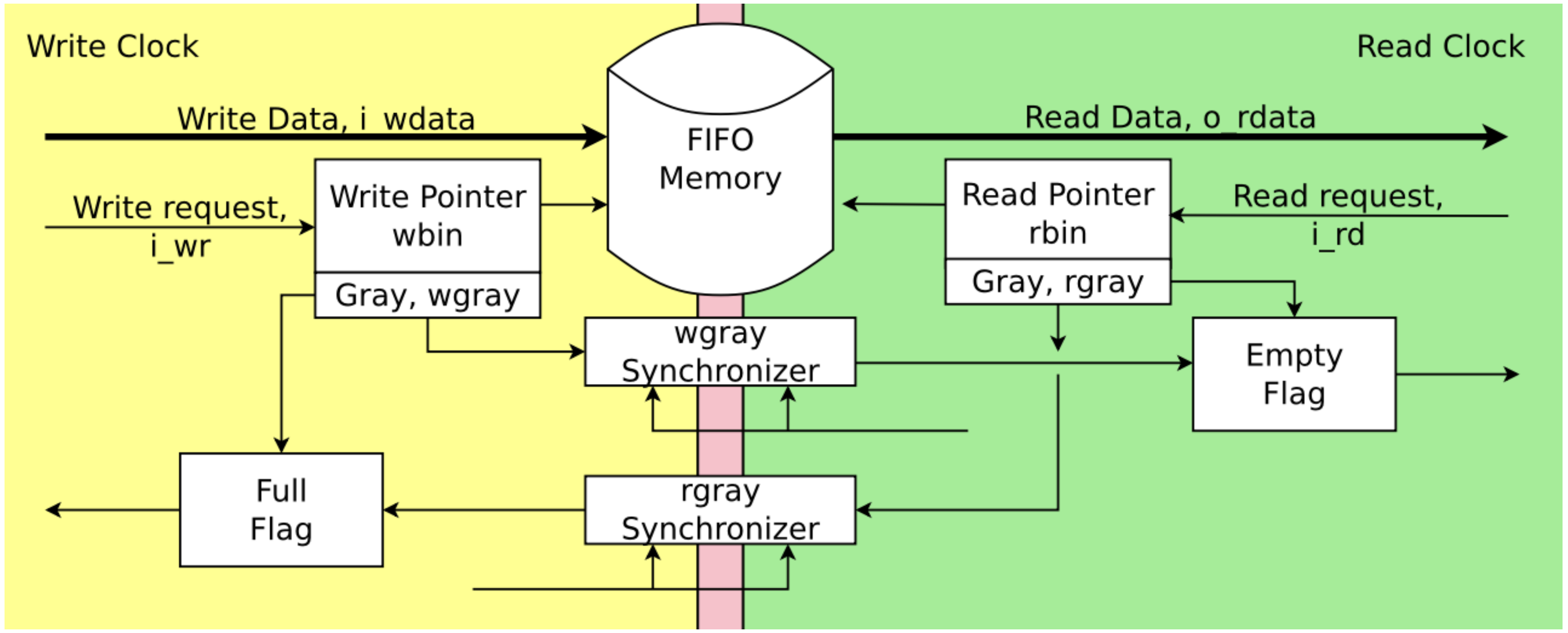
```
assign gray = bin ^ (bin >> 1);
```

Async FIFOs

- A basic async FIFO block diagram might look like this



Async FIFO – cont'd



Async FIFO – cont’d

- Most work is done around generating the empty and full flags
- FIFO Full flag is set when the write pointer catches up to the synchronized read pointer (in the write clock domain)
  - Extra attention needs to be paid to the case when the write pointer wraps around one more time than the read pointer
- Empty flag is set when the read pointer is equal to the write pointer synchronized in the read clock domain
- Assignment 6 will be based on this, and we will cover Async FIFOs in more detail on Friday

Questions?

Thank you!