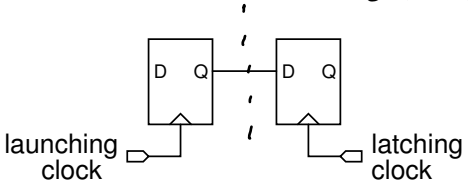
Clock Domain Crossing

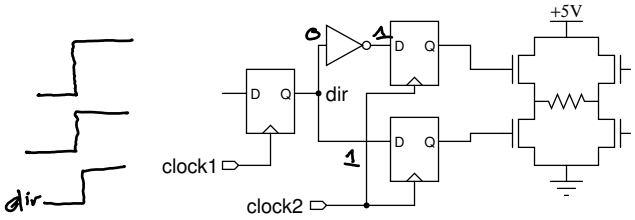
This lecture describes some potential pitfalls when signals cross asynchronous clock domains and how to avoid them. After this lecture you should be able to: estimate the effect of various design parameters on metastability MTBF and design synchronization circuits for single- and multi-bit signals.

Clock Domain Crossing

Care must be taken when designing interfaces between digital circuits that use independent clocks. This is called a "clock domain crossing" (CDC).



As an example of the consequences of incorrectly designed CDC, consider the H-bridge driver below:

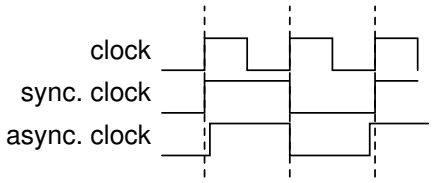


If the `direction` signal changes well before the rising edge of `clock2` then the two flip-flops are loaded with complementary values and the H-bridge functions as designed. But if `direction` changes just before `clock2`, the delay through the inverter can result in both transistors being turned on. This would result in a short-circuit and possibly permanent damage.

In addition to asynchronous inputs, digital circuits often use different clocks. For example, a cell phone may have one oscillator generating the processor's clock and another oscillator for the radio circuits.

When two clocks are synchronous – derived from the same source – the phase relationship between clock edges remains constant and static timing analysis can guarantee we will meet all timing requirements and avoid metastable behaviour.

On the other hand, when two clocks are asynchronous – independently generated – their clock edges will drift relative to each other. This means that it is *guaranteed* that at some point in time the setup and hold requirements of the latching clock will be violated. This will result in metastable behaviour.

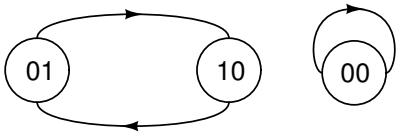


Consequences of Metastability

Metastable behaviour means that a flip-flop's output will not have settled to a valid logic value within the specified t_{CO}. This can result in different flip-flops latching different values at the next clock edge:

As shown in the example above, this violates fundamental assumptions made by the designer and the design software. The result is unpredictable behaviour with potentially disastrous consequences.

As another example, consider a state machine with one-hot state encodings of 01 and 10. Metastable behaviour could result in a transfer to an invalid state such as 00. The device would likely "lock up" in that state.



Although it's impossible to avoid metastable behaviour, we would like to be able to predict its likelihoood and find ways to reduce this probability to an acceptable level.

MTBF Calculations

Since we cannot predict when a metastable event will happen, we must treat failures caused by metastability as a random process. Analysis and experimental results show that the mean time between failures

(MTBF) due to metastable behaviour of a flip-flop can be approximated by:

$$\text{MTBF} = \frac{e^{T_s/C_2}}{C_1 f_d f_c}$$

where:

- T_s is the time available for the flip-flop output to reach a valid logic level. This will be equal to the slack calculated from timing analysis.
- C_2 is a time constant that depends on how fast the logic circuit drives the output towards a valid logic level. This is a function of RC time constants and circuit gains.
- C_1 is a time constant proportional to the time interval ("aperture") during which an input transition could cause a metastable output.
- f_d is the average rate at which the flip-flop input (D input) changes (sometimes called the "toggle rate")
- f_c is the latching flip-flop clock frequency

C_1 and C_2 also depend on the rise times of the clock and data signals, the manufacturing process and the operating conditions (PVT). In addition, ASIC libraries and FPGAs can have "metastable hardened" flip-flops designed for small values of C_1 and C_2. Since these have higher power consumption they are typically only used for CDC circuits.

Since accurate estimation of MTBF requires knowing the propagation delay between flip-flops and the constants for the specific flip-flops used, MTBF calculations are typically done by timing analysis software. For example, Intel's TimeQuest STA can recognize CDC synchronizers (described below) and generate MTBF estimates:

Synchronizer Chain #1: Typical MTBF is Greater than 1 Billion Years

Chain Summary | Statistics

	Property	Value
1	Source Node	a_in
2	Synchronization Node	a_sync
3	Typical MTBF (years)	Greater than 1 Billion
4	Included in Design MTBF	Yes

Exercise 1: A TimeQuest MTBF report says *Under typical conditions, an increase of 100ps in available settling time will increase MTBF values by a factor of 10.8.* What is C_2?

Increasing MTBF

Since MTBF increases exponentially with slack time (T_s), the most effective way to increase the MTBF is to increase the slack time.

Exercise 2: Using the values above, by how much would the MTBF increase if the slack time was increased by adding one cycle of a 50 MHz clock?

The MTBF is also inversely proportional to product of the clock and data input toggle frequencies. Thus another way to increase the MTBF is to minimize the clock frequency and data toggle rates.

Selecting Required MTBF

The selection of target MTBF depends on the consequences of a failure. The MTBF due to synchronization failure should be much longer than the product's overall MTBF due to other causes so that its doesn't impact the overall MTBF. The techniques described below can increase the MTBF due to synchronization failure to any required value at relatively low cost.

Exercise 3: What would be an acceptable MTBF due to CDC synchronization failure for a video game console? For an implantable cardiac defibrillator? Why?

Instead of the MTBF for one device operated continuously, a manufacturer may want to know the aggregate MTBF for all devices in the field in normal operation. The MTBF is inversely proportional to the number of clock cycles in which a metastable event could happen. This in turn is proportional to the the number of devices and their operating time.

Exercise 4: The MTBF estimate for a design is 1000 years. How often would you expect product failures in the field if there were 1 million devices being used one hour per day?

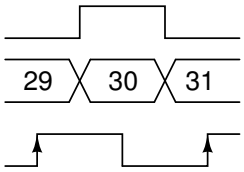
Missed Events

In addition to problems caused by metastability it's also possible that changes to a signal will be missed when a signal crosses from one clock domain to a slower one.

In some cases this will not matter. For example, skipping an intermediate temperature reading many not affect the operation of a furnace controller. But in other cases missing an event such as a short pulse could result in incorrect operation.

Thus it's important to take into account the launch and latch clock rates. However, since assumptions about clock rates can change as a design changes it's

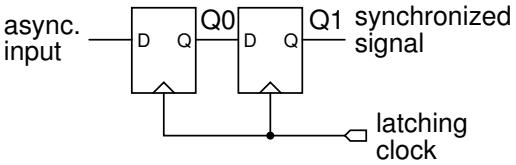
best to use synchronization techniques that are independent of the clock rates. These typically involve an exchange of handshaking signal as in the methods described below.

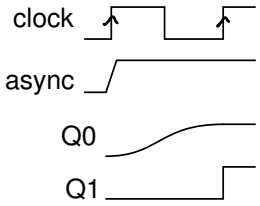


CDC Methods

Multi-flop Synchronizer

The most common way to increase the MTBF is to use two flip-flops in series to build a "synchronizer":



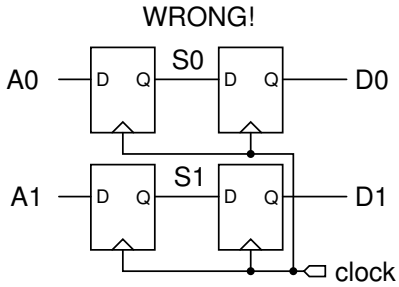


The use of two flip-flops with no intermediate delays allows for a settling time, T_s, approximately equal to one clock cycle. This is usually sufficient to ensure a sufficiently high MTBF at the output of the second flip-flop. However, if necessary, additional flip-flops can be placed in series.

An unavoidable consequence of using a synchronizer is that one (latch) clock cycle of delay is added to the signal.

Data Coherency

To synchronize the transfer of multiple bits you might be tempted to use multiple single-bit synchronizers in parallel.

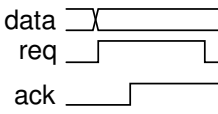


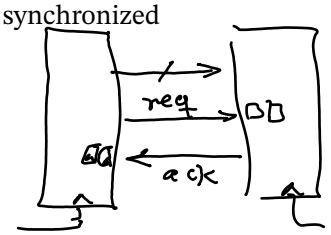
However, this approach can fail when the input changes close to the latching clock edge. Even if the output of each synchronizer output settles to a valid logic level, each synchronizer could settle to either the value before or after the input change. The binary value latched by this (bad) multi-bit synchronizer could be completely different than the value launched.

For example, a two-bit value changing from 01 to 10 could be captured as 00, 01, 10 or 11 depending on whether each synchronizer settles to the value before or after the change.

Four-Phase Handshake

Reliable transfer of data can be done using a "four-phase handshake." This requires generating a `request` status signal along with the data and using an `acknowledge` signal to indicate that the data has been captured. Both `req` and `ack` need to be synchronized because they cross clock domains.





The sequence of events is:

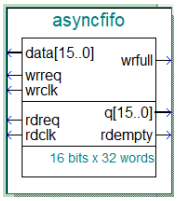
- `req` is asserted at the same time that the data is output
- `ack` is asserted after the data has been captured; this requires two (latching) clock cycles to ensure data coherency
- `req` is de-asserted to prepare for the next transfer
- `ack` is de-asserted to indicate that the receiver is ready for the next transfer

This method works regardless of the frequencies of the two clocks. However, the one- to two-cycle delay per transfer that is required to synchronize changes on the handshaking signals limits the transfer rate.

Asynchronous FIFOs

When faster CDC transfers are required, an asynchronous FIFO – a FIFO with independent read and write clocks – can be used. This requires a dual-ported memory with independent read and write ports. The design of an asynchronous FIFO is relatively complex because it's necessary to transfer read and write pointers between clock domains. However a FIFO allows for the fastest possible transfer rate – one word per read or write clock.

Since asynchronous FIFOs are commonly available as tested IP blocks there's no need to design your own. For example, the Intel FPGA IP library contains a FIFO that can be configured for asynchronous operation with independent read and write clocks:



Conclusion

Synchronization problems due to clock domain crossings can have serious consequences. Since they are random and potentially infrequent events they are unlikely to be found during testing and may only be discovered after a design goes into production.

To avoid CDC problems you should minimize clock domain crossings, use only commonly-accepted synchronization methods and ensure the MTBF calculated by CAD tools are appropriate for the application.