Metastability and Clock Domain Crossing

After this lecture you should be able to compute the effect of input and clock frequency changes on the MTBF and implement clock domain crossing synchronizers to minimize the probability of a metastable event.

Input Synchronization

Input signals often control the values loaded into multiple flip-flops. For example, an input that controls state transitions in a state machine can affect the values loaded into the flip-flops that hold the encoded state. When an input changes, differences in propagation delays will cause different parts of the circuit to see different values at different times. This can result in behaviour that depends on when the input changes relative to the clock. A circuit that exhibits unpredictable behaviour as a result of the timing of its input(s) is said to have a *race condition.*

Such problems can be avoided by registering the inputs using a flip-flop and using the output of this flip-flop output to drive the rest of the logic. This results in a delay of up to 1 clock period before the circuit can respond to the changed input. Usually this is an acceptable trade-off for improved reliability.

Exercise 1: Draw the schematic of an input synchronizer.

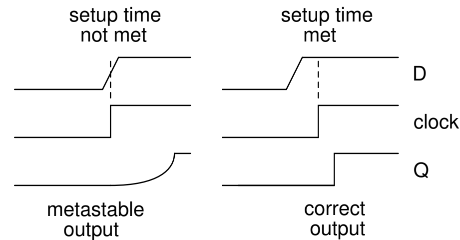
As a general rule, **always synchronize (register) asynchronous inputs.**

Metastability

Introduction

The proper operation of a clocked flip-flop depends on the input being stable for a certain period of time before the rising edge of the clock (the setup time) and after the clock edge (the hold time).

If the setup or hold time requirements are not met and an input transition happens during a short time window around the transition of the clock, then the output level could take a long time to settle to a valid logic level (longer than t_{CO}, the guaranteed clock-to-output delay). This is called *metastable* behaviour or *metastability*:



This longer-than-expected delay combined with different propagation delays through different parts of the circuit can result in some parts of the circuit treating the flip-flop output as low and some as high. This will cause unpredictable, possibly catastrophic, behaviour.

In the synchronous circuits we have studied thus far we have been able to prevent metastability by clocking all flip-flops from the same clock and doing timing analysis to ensure that the maximum propagation delay of any combinational logic path is less than the clock period minus the flip-flop setup time and clock-to-output delay.

However, some inputs will be asynchronous – have no time relationship – to a circuit's clock. Examples include switches or sensors that can change at any time, and an signal controlled by a different oscillator. It these cases it is *impossible* to ensure that the setup and hold times will be met and this will eventually lead to metastable behaviour and (likely) incorrect behaviour of the device.

It is important to realize that all practical logic circuits with asynchronous inputs will eventually fail due to metastability. However, the designer should try to ensure that these failures happen very infrequently (e.g. once per 10^3 or 10^6 years of operation) so that other causes of failure predominate.

Computing MTBF

The average time between metastable outputs (mean time between failures or 'MTBF') is given by the formula:

$$\text{MTBF} = \frac{e^{C_2 t_{MET}}}{C_1 f_{clk} f_{data}}$$

where C_1 and C_2 are constants that depend on the technology used to build the flip-flop, t_{met} is the duration of the metastable output, and f_{clk} and f_{data} are the frequencies of the synchronous clock and the asynchronous input respectively.

FPGA manufacturers no longer seem to publish values for C_1 and C_2 that would allow you to calculate the MTBF but instead incorporate MTBF calculation into their timing analyzers.

However, the equation shows that the MTBF is proportional to the clock and data input periods and increases exponentially with the duration of the metastable output.

As a general rule, metastability issues can be ignored for inputs with low average frequencies (e.g. keypads) but need to be considered when a signal crosses clock domains – when the launch and latch clocks are asynchronous – and the frequencies are 10's of MHz or higher (e.g. high-speed data interfaces).

Reducing Metastability Probability

The simplest approach is to slow down the clock since this provides a longer time for the output of the flip-flop to reach a stable output value (increases t_{MET}). Because the MTBF increases exponentially with t_{MET} a small reduction in clock frequency can be enough to increase the MTBF to an acceptable value. However, many cases reducing the clock rate is undesirable.

Another approach is to use flip-flops with shorter setup and hold times – and correspondingly smaller C_1 and larger C_2 values. The trade-off is that such flip-flops typically have higher current consumption. Whenever possible, these "metastable-hardened" flip-flops should be used on asynchronous inputs. The flip-flops on FPGA inputs are typically of this type.

If these approaches do not result in the desired degree of reliability, it is possible to use two (or more) flip-flops in series. This arrangement is called a synchronizer. Even if the output of the first flip-flop has not settled to a valid logic level before the setup time of the second flip-flop, it is unlikely to be at an invalid level at the rising edge of the second flip-flop. Thus all the logic driven by the second flip-flop will see the same logic level.

Exercise 2: Draw a synchronizer circuit. Does increasing the clock rate increase or decrease the MTBF?

A synchronizer can still fail if both the first and second flip-flops have metastable outputs after t_{CO}, but this is much less likely than for a single flip-flop. By adding additional flip-flops to the synchronizer chain the probability of a metastable output from the last flip-flop can be made arbitrarily low. The disadvantage of using a synchronizer is that the input will now be delayed by additional clock period(s).

Glitches

Glitches are short temporary changes in outputs that are caused by different propagation delays in a circuit. There are two reasons why glitches are undesirable.

The first set of problems is related to noise and power. Since glitches are short pulses much of their energy is at high frequencies and this power couples easily onto adjacent conductors. This induces noise into other circuits and reduces their noise immunity. Glitches also cause power supply current spikes which result in voltage transients on the power supply lines. Another problem with glitches is that in CMOS logic families current consumption is proportional to the number of transistor switchings and glitches lead to increased current consumption.

The second set of problems arises when the digital output of one circuit is used as a clock in another circuit (e.g. to drive a counter or register). In this case glitches cause undesired clock edges (similar to switch bounce). In synchronous (single-clock) circuits these glitches are not a problem.

Glitches can be reduced by modifying the design of the combinational logic. However, this usually introduces additional logic. Glitches on signals that are confined to short paths within a circuit or inside a chip are usually tolerated. However, when outputs are brought off a chip, board or system (e.g. onto a bus) it is good practice to eliminate glitches.

The simplest way to eliminate glitches is to use a registered output signal. The output of a flip-flop changes only once, on the clock edge, and thus eliminates any glitches on its input. There are two ways to register outputs. Often it is possible to use register outputs directly such as when an output is already in a data register or when the signals are state machine state registers. The second method is to pass the signal through an additional flip-flop before it is output. The disadvantage of this method is that the output will be delayed by up to one clock period.

As a general rule, **always register chip outputs.**