FPGA TechNote:
Asynchronous signals and Metastability

This *Doulos FPGA TechNote* gives a brief overview of metastability as it applies to the design of FPGAs.

The first section introduces metastability and gives links to resources.

The second part looks specifically at making calculations using typical metastability data, and also at the automation of metastability calculation that has recently been introduced in the Altera Quartus II software.

A PDF version of this document, together with the code examples, can be downloaded from our web site:

`www.doulos.com/knowhow/fpga`

Our KnowHow pages are often updated with new material – call back soon!

Contents

Asynchronous Signals and Metastability

This introductory part of the TechNote introduces the problems we aim to solve, and the available methods.

Introduction

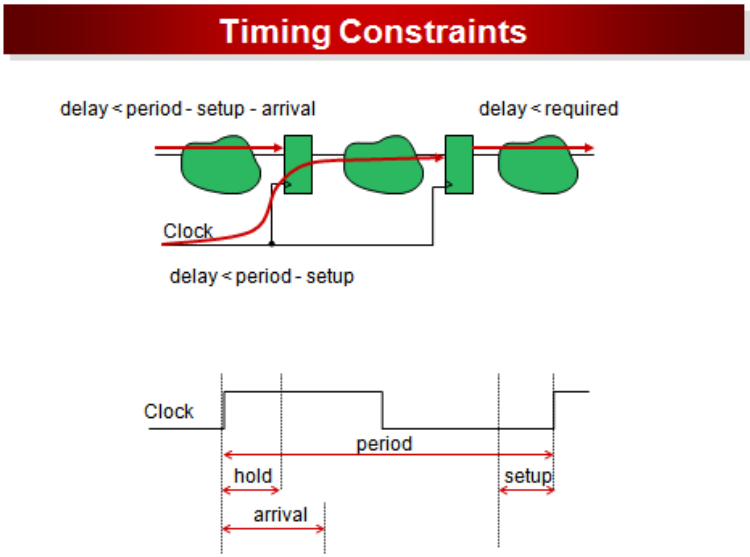
Synchronous Design and Synchronization

Modern digital design is normally carried out using a synchronous design methodology. A design consists of combinational logic and edge-triggered flip-flops.

Combinational logic is logic such that the output of the logic is purely a function of the steady-state values of the inputs – in other words it has no memory.

Edge-triggered flip-flops are used to remember the state of a design. They are triggered by one edge of a clock signal. As long as the input to an edge-triggered flip-flop is available some time before the clock edge (the setup time), the value on that input will be sampled correctly.

Here's a simplified diagram.



Designers tend to concentrate mainly on register-to-register delay, as this governs the maximum clock frequency at which a circuit can run.

Inputs that are triggered by the same clock may have an arrival time specified so that they are guaranteed to settle before the following clock edge.

However in the real world, some signals may not be related to the clock

- A signal may come from another chip running on a different clock
- A signal may be an external input, for instance a push-button
- A signal may be from another clock domain inside the same chip.

In all these cases the designer needs to take care of synchronization.

Metastability

When an external asynchronous signal is sampled by a flip-flop, it will be sampled correctly if it changes before the input setup time of the flip-flop, and after the input hold time.

If the signal changes too close to the clock edge (within the time interval from the input setup time to the input hold time) it is possible that the signal will be sampled with the wrong value. However on the following clock edge, the correct value should be seen.

However if the input signal changes extremely close to the clock edge, it is possible for that signal to enter a metastable state – an analogue value which is neither a zero nor a one. The signal will then eventually settle to either a zero or a one randomly.

The big problem with metastability is if the “analogue” value at the output of the flip-flop lasts such a long time that it affects other flip-flops.

In that case, it is possible for downstream circuitry to end up in an illegal state – that is two downstream flip-flops could sample the same sampled signal at the same time, yet end up with different output values; because that sampled signal is in an indeterminate (analogue) state.

That could mean, for instance, that a state machine reaches a supposedly “impossible” state.

The most important thing to realise about metastability is that there is no cure. Metastability can always occur if you sample an asynchronous signal – the question is only *how often?* Because of its probabilistic nature, metastability is characterized by the mean time between failures (MTBF).

Metastability documentation

There is a lot of documentation available for metastability.

General descriptions

- The FPGA FAQ http://www.fpga-faq.org/FAQ_Pages/0017_Tell_me_about_metastables.htm
- The Massachusetts Institute of Technology course Computation Structures contains a lecture about metastability http://6004.csail.mit.edu/currentsemester/

Vendor provided information

- Actel Application Note AC308 “Metastability Characterization Report for Actel Antifuse FPGAs”
- Actel Application Note “Metastability Characterization Report for Actel Flash FPGAs”
- Altera “Understanding Metastability in FPGAs” white paper wp-01082
- Altera AN473 “Using DCFIFO for data transfer between asynchronous clock domains”
- Altera Quartus II 9.0 Handbook Volume 1 section 7 “Managing Metastability with the Quartus II software”
- Lattice Semiconductor Corporation Technical Note 1055 “Metastability in Lattice Devices”
- Xilinx Application Note: Virtex II Pro Family XAPP094 “Metastable Recovery in Virtex II Pro FPGAs”

Calculation of Metastability

In the references above you will find a full discussion of metastability. The following equation can be derived for the MTBF due to metastability

$$MTBF = \frac{e^{K_2 * t_r}}{F_1 * F_2 * K_1}$$

Where

- **K_1** Metastability catching set-up time.
- **K_2** Re-convergence quality factor, which is proportional to the gain-bandwidth product in the internal feedback path of the first latch inside the flip-flop.
- **F_1** and **F_2** system (sampling) and external (asynchronous) event frequencies. We assume than there is no correlation between F1 and F2.
- **t_r** acceptable extra delay (**r** representing “recovery” or “resolution”).

The constants **K_1** and **K_2** are due to the detailed electronic design of the flip-flop.

The value of **t_r** is the amount of extra settling time that can be allowed before the (possibly metastable) signal is sampled by the next downstream flip-flop.

Regardless of the actual values involved, it can be seen that increasing **t_r** makes the MTBF exponentially better (longer).

Interpreting the Metastability Equation

Note that you have to be careful when encountering the equation above to look at the definition of **t_r**. When deriving the equation from fundamentals, the value of total time **t** to resolve a meta-stable state is the total settling time available, which comprises of Tco (the clock to

output delay of the flip-flop) + $\mathbf{t_r}$ the additional time available to resolve to a known value. In other words

$t = Tco + t_r$

Thus the equation could be written

$$MTBF = \frac{e^{K_2 * (Tco + t_r)}}{F_1 * F_2 * C_1}$$

which can be re-arranged as

$$MTBF = \frac{e^{(K_2 * Tco)} e^{(K_2 * Tr)}}{F_1 * F_2 * C_1}$$

Then by absorbing the constant term $e^{(K_2 * Tco)}$ into C_1 (to give our original K_1) the original equation can be obtained.

Some of the papers above also use the contstant $1/K_2$, normally writing this as the greek letter τ (tau).

Increasing the MTBF

To increase the MTBF we want to allow as much time as possible after the clock edge. If you imagine the case of 1 flip-flop feeding into the first flip-flop of the main circuit, then the settling time $\mathbf{t_r}$ = clock period – Tco – Tsu – Tpd where

Tco is the clock to output delay of the first flip-flop, Tsu is the input setup time of the following flip-flop, and Tpd is the propagation delay of any combinational logic (or wiring) between the two flip-flops.

To make Tpd as small as possible, and hence make $\mathbf{t_r}$ as large as possible, we can simply cascade two flip-flops. To a first

approximation, this multiplies the MTBF for one flip-flop by itself (i.e the result is the square of the MTBF).

This technique is known as a two stage synchroniser.

More than one asynchronous input

What happens if you have more than one independent asynchronous input? In that case each input will need separate synchronization. If you have *n* inputs then the combined MTBF is given by

$$\frac{1}{MTBF} = \frac{1}{MTBF_1} + \frac{1}{MTBF_2} + \cdots \frac{1}{MTBF_n}$$

This means that the worst MTBF tends to dominate. For instance if you have two MTBFs of 1000 years and one of 1 year, the combined MTBF will be very close to (just less than) 1 year.

Practical Example

Let's now create a simple example problem. Rather than try and obtain metastability constant values, we will use the new features of Altera® Quartus® II 9.0 to show the effect of synchronization on our circuit.

Circuit Description

We will use a simple counter. As modern devices and flip-flops are very fast, we will write the counter in a "bad" style, to try and show the effect of metastability.

The counter has a number of inputs including an UpDn signal which causes it to count up (when UpDn is true) or down (when UpDn is false); and a Load signal, which causes data to be loaded in from a parallel input port.

We will assume all signals are synchronized to the counter clock except for UpDn and Load – which are assumed to be asynchronous.

The counter has been written is such a way that there is a long path from UpDn through to the adder of the counter. This is so that when we synchronize this path, the metastability recovery time is reduced by extra combinational logic.

You can download the counter code (both VHDL and Verilog versions) from our website at http://www.doulos.com/knowhow/fpga

To synchronize the asynchronous inputs we create a synchronizer design entity which is parameterizable for the number of stages of synchronization. Here is the VHDL code for that synchronizer:

```
library IEEE;
use IEEE.Std_logic_1164.all;
entity sync is
  generic (nStages : positive);
  port (Clock : in Std_logic;
        D: in Std_logic;
        Q : out std_logic);
end;

architecture RTL of sync is
  signal delay  : std_logic_vector(nStages-1 downto 0);
begin

  g1: for I in delay'RANGE generate
    -- first stage of synchronizer
    -- Capture external input
    g2: if I = 0 generate
       process(clock)
       begin
         if rising_edge(Clock) then
           delay(0) <= D;
         end if;
       end process;
    end generate;

    -- subsequent stages - will not be used if nStages = 1
    g3: if I /= 0 generate
       process(clock)
       begin
         if rising_edge(Clock) then
           delay(I) <= delay(I-1);
         end if;
       end process;

    end generate;
  end generate;

  -- assign from final stage of synchronizer to external output
  Q <= delay(nStages-1);
end;
```

By setting nStages to 1 or 2 we can create a 1 or 2 stage synchronizer.

Setting up Altera Quartus II

To get started you'll need to create a project in Altera Quartus II. The easiest way to do this is to use the File > New Project Wizard... menu.

To see the same results described in this TechNote, select the device EP3C5E144C8 (a Cyclone® III device).

You will need to add the files sync.vhd and counter.vhd to the project.

Using Altera Quartus II for Metastability

First, note that at the time of writing only certain device families are supported for metastability calculation: Arria® II GX, Stratix® IV, Stratix III, and Cyclone III. That is why we specified a Cyclone III device above.

Setting up the Timing Analyzer

Set up the TimeQuest Timing Analyzer as follows:

- Select menu Assignments > Settings
- In the left pane, select Timing Analysis Settings
- To the right, select Use TimeQuest Timing Analyzer during Compilation
- In the left pane, select TimeQuest Timing Analyzer
- To the right, click on the ... next to the SDC FileName: box, and find the file counter.sdc (which is in the download from the website).

This file sets the clock frequency.

- Click Add to add the counter.sdc file to the project.
- Now at the bottom of the same form, set Synchronizer Identification: to Auto
- Click OK

The last step tells Quartus to search for chains of registers automatically – note however that Quartus will *not* carry out

metastability analysis until you have manually identified valid synchronization chains.

Identifying Synchronization Chains

With the default setting of 1 stage of synchronization on UpDn and Load, you should find that Quartus fails to identify any synchronization chains. This is because the default behaviour is to look for at least 2 flip-flops, and we have set our synchronization chain length to 1.

To verify this

- Compile the design
- Run TimeQuest (double-click the clock icon)
- In TimeQuest, double-click Update Timing Netlist
- Double-click Report Metastability

You should see the message

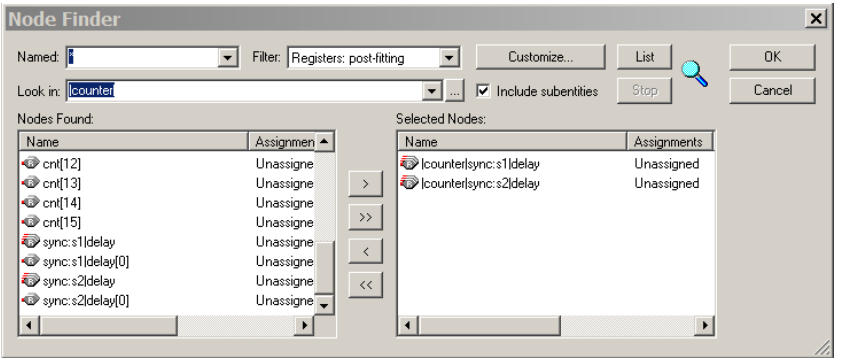
```
Info: No synchronizer chains to report.
```

Now you can close TimeQuest.

To identify synchronization chains

- Select menu Assignments > Assignment Editor
- In the assignment editor, use the drop-down list under Assignment Name, and select Synchronizer Identification
- Now double-click under To so a small right-pointing arrow appears
- From the arrow, select Node Finder...
- In the node finder, set Filter... to Registers: Post Fitting
- Click List
- Select sync:s1|delay and sync:s2|delay, and add them to the SelectedNodes

There's a screenshot at the top of the next page:

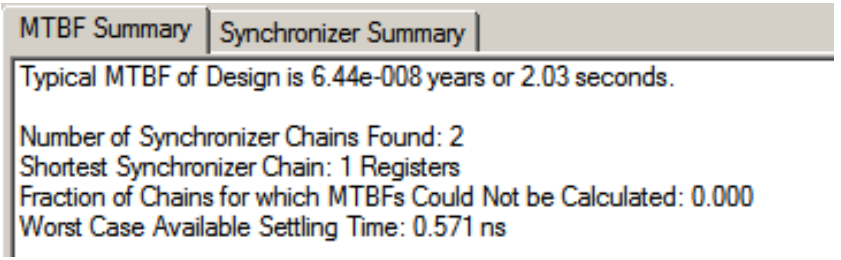


- Click OK to close the dialogue box.

The final step is to set the value for these assignments in the Assignment Editor (which should still be open).

- In the column labelled Value, select “Forced if asynchronous” for both assignments.
- Save the assignments – once saved, you can quite the assignments editor.

Now run compilation and TimeQuest again, and run the metastability report. This time you should see



By carefully writing a bad design (!) and only using one stage of synchronization we have achieved a very bad MTBF of 2 seconds!

However before changing our chains to length 2, it would be a good idea to understand what this figure is really means. Altera Quartus is automatically calculating a value based on the performance of the flip-

flips in a Cyclone III chip with speed grade 8 – but what about the frequencies?

The sampling frequency is set by the clock, which was set in the counter.sdc constraint file. If you look in there, you'll see we set a clock period of 4.3ns (frequency of 233MHz).

The Quartus software assumes that the inputs we are sampling are changing at 1/8 of the clock frequency (about 29 million transitions per second).

In other words, the calculation is assuming that UpDn and Load are both changing asynchronously 29 million times per second.

If this is not realistic, then you can adjust the assumptions by setting the assignment Synchronizer Toggle Rate in the Assignment Editor.

You'll also see from the Report Metastability output in TimeQuest the figure

```
Worst Case Available Settling Time 0.571 ns
```

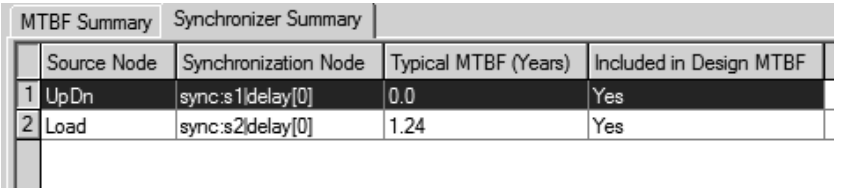
Improving Metastability

To improve the metastability we could

- reduce the clock rate
- reduce the input toggle rate (for UpDn and Load)
- choose a faster speed grade device
- re-design our code to remove the large amount of combinational logic between UpDn and the destination registers in the counter
- increase the length of the synchronization chain

Let's try the last one of these, increase the synchronization chain length.

First, let's find out what is the main contributor to our metastability. From the metastability report it is possible to get the individual contribution of each asynchronous input, which is shown below:



MTBF Summary | Synchronizer Summary

	Source Node	Synchronization Node	Typical MTBF (Years)	Included in Design MTBF
1	UpDn	sync:s1|delay[0]	0.0	Yes
2	Load	sync:s2|delay[0]	1.24	Yes

From this you can see that UpDn is the main culprit.

By increasing the synchronization chain to length 2 on the UpDn input, we obtain an MTBF of “greater than 1 billion years” for the UpDn input, and 1.24 years for the Load input – which gives a combined MTBF of 1.24 years.

By making the synchronization chain length 2 for the Load signal as well, the combined MTBF becomes over 1 billion years – which should be good enough!

Conclusion

We have summarised literature for metastability, and shown the governing equation – pointing out the care needed to correctly understand the various constants involved.

We have then worked through a practical example using the metastability calculation features in Altera Quartus II 9.0 to show the effect of adding stages to input synchronization of asynchronous signals.

For modern fast CMOS processes a 2 stage synchronizer will almost definitely be sufficient to increase MTBF to acceptable levels, but it is easily possible that a single stage of synchronization is not sufficient.

Finally, it is wise not to ignore metastability even though modern chips are very fast – it only takes one bad synchronizer (or even worse an unsynchronized asynchronous input) to drastically reduce MTBF.

And Finally...

We hope you find this Technote useful. Please visit www.doulos.com/knowhow for more valuable hints and tips, and to download example code to go with this Technote.