

THIS CODE IS PART OF ESTEREL STUDIO BASIC BLOCS EXAMPLES. IN NO EVENT SHALL ESTEREL EDA TECHNOLOGIES SA BE LIABLE TO YOU FOR ANY LIABILITY, LOSS OR DAMAGE, INCLUDING, WITHOUT LIMITATION, DIRECT, INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR LOSS OF USE OR OTHER ECONOMIC LOSS, EVEN IF ESTEREL EDA TECHNOLOGIES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. YOU AGREE THAT YOU WILL BEAR THE ENTIRE RISK OF USING THE CODE AND THAT YOU WILL INDEMNIFY, DEFEND, AND HOLD ESTEREL EDA TECHNOLOGIES SA HARMLESS AGAINST ANY CLAIMS ARISING OUT OF THIS AGREEMENT, ANY OTHER AGREEMENT.

## Two Phase Multiclock Synchronizer Esterel v7 Design Version 2

M. Kishinevsky<sup>1</sup> (2005), G. Berry<sup>2</sup> (2007)  
www.esterel-technologies.com

(c) Esterel EDA Technologies 2006, 2007

This multi-clock design implements, simulates, and formally verifies the *two-phase handshake push synchronizer* used to safely transfer information between asynchronous clock zones by correctly taking care of metastability issues.

We made the two-phase synchronizer generic in the data it can handle and made some minor interface and name changes to make the synchronizer easier to use.

See also the four-phase synchronizer Synchronizer4 in the same paper and in the Esterel Studio design library. The four-phase synchronizer is simpler but has a bigger latency.

### 1. Synchronizer2 Interface

The Synchronizer2 multiclock design deals with two clocks: the Sclk clock is assumed to be the sender's clock, while the Rclk clock is assumed to be the receiver's clock. No relation is supposed between these clocks, which can be completely asynchronous and whose edges may coincide in time by chance. The design of course works with any kind of partially synchronized clocks, but it may be overkill in this case.

#### 1.1. Interface on the sender side

The Esterel v7 on the sender side is as follow, where Sclk clocks all the signals:

```
input DataIn : temp MyType; // Input data with a valid bit
output Ack; // Previous transfer cycle acknowledged
output Ready; // Ready to transfer new data
```

To send a value, the sender should sustain the full signal DataIn that carries the value to be sent and whose status high signals the intention of sending. The DataIn signal should

---

<sup>1</sup> Intel Strategic CAD LAB, Hillsobro, OR, Michael.Kishivevsky@intel.com

<sup>2</sup> Esterel Technologies, Villeneuve-Loubet, France, Gerard.Berry@esterel-technologies.com

be sustained until the Ready output falls, this cycle included. Its value is sampled at that time and transmitted.

The synchronizer sends back the Ack acknowledge to tell the sender that the value has been correctly transmitted and the Ready signal to tell the sender that the synchronizer is ready to transfer a new value.

After each transfer, the Ready signal is always asserted one Sclk cycle before the Ack signal. Therefore, one can also ignore the Ready signal and use only the Ack signal. We use Ready to make the interface fully compatible with that of the four-phase Synchronizer4, for which no simple relation exists between Ack and Ready.

## 1.2. Interface on the receiver side

The interface on the receiver side consists of a single full signal, clocked by Rclk:

```
output DataOut : temp MyType; // Output data with a valid bit
```

This output is emitted for a single cycle when a value is received. Beware, it must be caught and memorized by the receiver circuit.

## 2. Synchronizer2 structure

The model consists of two active modules, Sender2 and Receiver2, here described in mixed textual / graphical form. The generic GenericSynchronizer2 multiclock unit puts Sender2 and Receiver2 in parallel, respectively clocked by Sclk and Rclk. The Sender2 and Receiver2 modules involve #ifdef directives to switch between synthesis and simulation verification modes:

- In synthesis mode, the default mode, the code is the normal one, ready to be synthesized to VHDL or Verilog for hardware implementation.
- In simulation / verification mode, triggered by the esterelv7 command line option -define SIMULATION, new inputs are added to mimic metastability errors by negating the normal output of resynchronization registers on input rising or falling edges.

## 2. Esterel Studio Project

The Esterel Studio project is called Synchronizer2.etp. Load it in Esterel Studio before trying any of the actions below.

## 3. Simulation with Esterel Studio

- Set the Simulation configuration active. The main module is called Simulation. It instantiates the object type as unsigned<1000> and the initial value as 0.

- Play the Simulation.esi input file, which shows data transfer with asynchronous and irregular clocks (note that clock events can coincide).
- Look at waves. The signals are pictured in causal order, which we find easier to read. The wave definition file is Simulation.xml in the project directory.

## 4. Formal verification with Esterel Studio

Set the Verification configuration active. The objectives are to formally prove that the protocol is correct and study its latency. By default, we put no constraints on the clocks. To give an example of a clock constraint, out-commenting some self-explanatory code at the end of the Verification module states a constraint that clocks can never be synchronous.

The Verification main module is a single-clock module, whose own clock acts as a fictitious event clock faster than the sender and receiver clocks. The sender and receiver clocks are mimicked by pure signals relative to this fictitious superclock. The Verification module puts the following elements in parallel:

- The Synchronizer module, acting on unsigned<4> type. Here, it is run as a single-clock module using the mcrun Esterel v7 statement, with clock signals renaming input clocks.
- A feeder that create a stream of modularly increasing numbers. To make the input stream as general as possible, an additional free input is used to decide if a new number is actually sent when the synchronizer is ready to transmit one.
- A checker that verify that the received values are modularly increasing.
- Assertions to generate some traffic.

### 4.1. Correctness verification

- Verify the assertion ValuesOK with full model-checking using the Prove button. The fact that we use unsigned<4> means that the verification cycle over 0,1,2,3. This is of course not the general case, but enough to show any real bug. For types bigger that unsigned<4>, performing a full proof is too long but one can still perform appropriate verification using bounded model-checking (BMC button) since the control depth of the design is not very long.
- Try introducing an error in the protocol, try changing “if V” by “if A2” in Sender4 as indicated by a comment. The error is immediately found by bounded model checking. Play the counter-example.

## 4.2. Latency evaluation

- Call Bounded Model Checking (BMC) or full verification (Prove) on the assertions `FirstOutput`, `SecondOutput`, and `ThirdOutput`. The lengths of the counter-examples indicate the latencies in absence of metastability.
- For worst-case latencies, force metastability on all edges by assuming the `ReceiverMetaError` and `SenderMetaError` always present in the Esterel Studio verification panel and call BMC again

## 5. VHDL / Verilog generation

Use VHDL or Verilog configuration to generate the actual multiclock circuit. The main module is called `Synchronizer4`. In the distribution, it uses type `bool[32]` and initial value `{32{0}}`. Change this type to suit your needs. There are of course no more `MetaError` inputs to drive metastability, which now comes from sheer nature.