SAME 2004

SESSION 3B: HIGH-LEVEL SYSTEM DESIGN

Multi-clock Design and Synthesis with Esterel

S. Bernardi, S. Lebailly, Texas Instruments,
B. Blanc, G. Berry, J. Dormoy Esterel Technologies
Villeneuve-Loubet, France

Multi-clock Design and Synthesis with Esterel

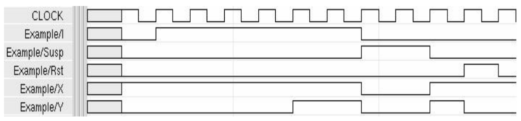
Abstract:

We describe a practical method for multi-clock design using the formal Esterel language and its programming environment, which is normally limited to single-clock design. The methodology has been applied to two designs at TI. The designs have been fully synthesized and verified.

Introduction

Esterel v7 [11] is a high-level language dedicated to synchronous hardware and embedded software programming. It is based on formal semantics presented in [8,9], which makes it unambiguous and synthesizable. The Esterel Studio tool provides an integrated environment for program capture, simulation, synthesis, and formal verification. Esterel programs are built from combinationally broadcast signals using concurrency, sequencing, pausing, preemption, and communication statements. All statements run on a unique base clock, which is mapped to the hardware clock for synchronous circuit designs. Here is a simple Esterel program example:

```
module Example:
input I, Rst, Susp;
output X, Y;
loop
  suspend
    sustain X
  ||
    await 5 I;
    sustain Y
  when Susp
each Rst
end module
```



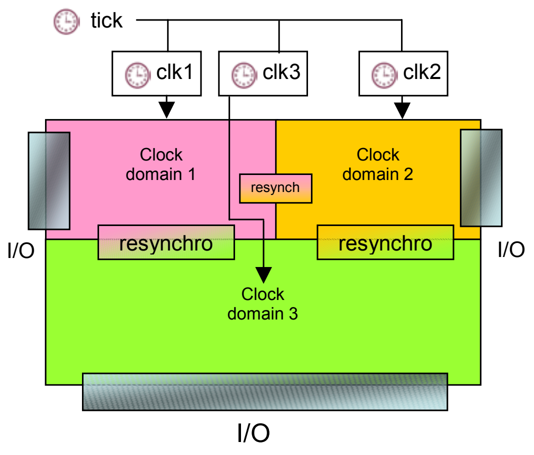
The internal concurrent statements behave as follows: the `sustain` X statement keeps X high; the concurrent `sustain Y` statement is only activated in the cycle where the 5th occurrence of `I` is received and it keeps `Y` high from then on. The enclosing `suspend` statement freezes the internal behavior and drives all outputs to inactive at each cycle where `Susp` is set. Finally, the outermost `loop-each` statement kills the behavior and restarts it afresh at any cycle where `Rst` is set.

Esterel is being used for control-dominated hardware designs by major electronics companies. It is used by TI for applications such as memory control, attachment interfacing, and power control. In some applications such as video attachment interfacing, the single-clock constraints appeared to be too strong: designs involving a variety of sensors and linked to software drivers by interfaces such as OCP usually involve several asynchronous clock zones linked by synchronizers designed to avoid or control metastability issues. The goal of this paper is to present a practical methodology to extend Esterel to multiple clocks. Asynchronous clocks are modeled as standard signals. Base clock (tick) is made fast enough to independently capture events from asynchronous input clock domains. Esterel Studio is used to design each clock domain as a classical single-clock block, and synchronizer models are coded in Esterel based on the conceptual base clock. For synthesis, the clock domains are treated as independent single-clock blocks and they are manually connected.

While similar to the multi-clock approaches presented in [7,10], Esterel methodology is more practical since it takes care of metastability issues for inter-zone communications. It has been successfully applied to two designs at TI.

How to build a multi-clock simulation model with Esterel

To build an Esterel Studio simulation environment for three-clock domains, we introduce three Esterel signals clk1, clk2, and clk3 that act as domain clocks. These signals are generated from the global Esterel base clock by a clock generator module. Each domain is represented by an Esterel module constrained to work only when its clock signal is active. Domains are directly connected to their external IOs. They are connected with each other using synchronizers described below. The global architecture is shown in the picture below.

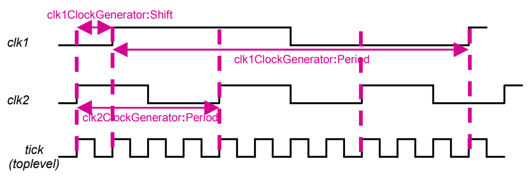


Clock generation

There are basically two ways for generating the domain clock signals: putting them as global inputs of the Esterel design, in which case they are fully driven by the simulation environment; or building them internally from the Esterel global tick, using for example, `Phase` and `Period` parameters as in the generator below:

```
module ClockGeneratorWithPhase:
constant Phase: integer;
constant Period: integer;
output Clock;
await Phase times tick;
loop
  emit Clock
each (Period/2) tick
end module
```

Here are the timing diagrams of two clocks with different phases.



Two clocks that are free inputs or are generated by two different instances of this module can occur simultaneously, which is useful to model random clock coincidence in the asynchronous world. Using Esterel incompatibility relations such as `clk1 # clk2`, one can forbid coincidence if desired. Phase and Period generation does not model dynamic skew, but is enough for most practical purposes.

Driving clock zones

We must constrain each domain to work only when its clock is active. This is exactly what the Esterel `suspend` construct achieves. In a sense, the `suspend` statement steals the tick to its body. The following code does the job:

```
   run ClockGeneratorCLK1
||
   run ClockGeneratorCLK2
||
   suspend
      run ClockDomainCLK1[…]
   when immediate not clk1
||
   suspend
      run ClockDomainCLK2[…]
   when immediate not clk2
```

Resynchronization for inter-zone communication

Esterel has pure and valued signals; for simplicity, here we consider only the former. Pure signals in Esterel are basically transient. They last one global clock tick and are broadcast to all receivers in this single clock tick. In our case, the global clock tick is the simulation tick, while the receivers are activated only by their domain clocks. Therefore, a signal could be lost if its receiver clock is not active at the global tick where it is emitted. We must make it persistent until the receiver clock is active. Here is how to transmit a signal from clock zone 1 to clock zone 2:

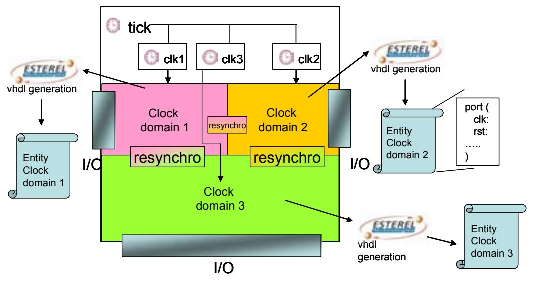
```
signal
   o1, i2
in
  suspend
    run p1 [o1 / output1]
  when immediate not clk1
||
  suspend
    run p2 [i2 / input2]
  when immediate not clk2
||
  sustain {
    i2 <= o1 and clk1,
    i2 <= pre(o1) and not clk1
  }
end signal
```

Notice the use of the sequential `pre` operator to keep the emitted value of `output1` in `o1`.

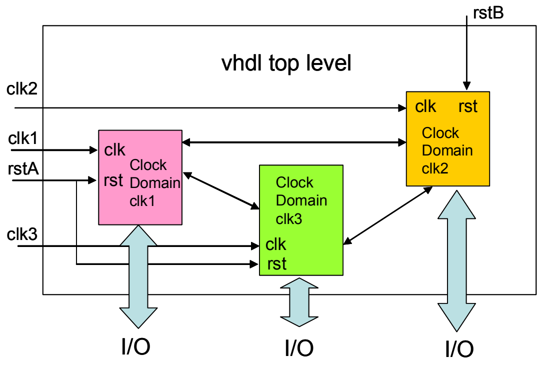
How to build the corresponding top-level VHDL

The above described Esterel top-level is valid for simulation and verification purposes only. It cannot be dumped into RTL as is. We use the following steps:

1) The modules of each clock domain are dumped into separate HDL files. Each file contains an entity with its own clock and reset signal.



2) We write a top-level VHDL entity that instantiates the clock domain components. We connect all ports, clocks, and resets. This is currently done manually.



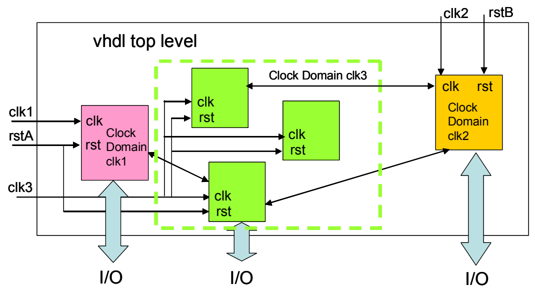
rstB
vhdl top level
clk2
clk1
rstA
clk3
clk rst Clock Domain clk1
Clock Domain clk3 clk rst
clk rst Clock Domain clk2
I/O
I/O
I/O

3) Within each clock domain, we perform modular compiling from Esterel to VHDL. This provides the following benefits:
   - Better sequential optimization for each module
   - Considerable reduction of compilation and synthesis time
   - Better-synthesized logic

   The drawbacks are:
   - The signals used to communicate between the VHDL components have to be connected manually
   - Esterel and VHDL model hierarchies do not exactly match

The next generation of Esterel compilers developed based on this experience will perform modular compiling wiring automatically.

The following picture presents a possible decomposition of Clock Domain 3:

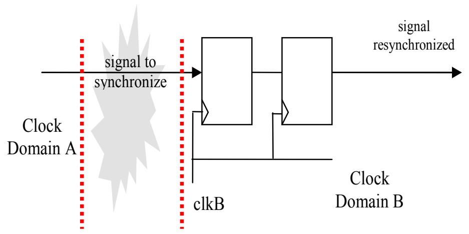


Re-synchronizing clock domains in Esterel

A signal sent from one clock domain to another one arrives asynchronously w.r.t. the receiver's clock, possibly violating the destination flip-flop setup or hold time; thus, causing a metastable condition and propagation of non-binary signals to other parts of the system [1-6]. The devices described below avoid metastability.

The double-stage synchronizer

A double-stage synchronizer is the most widely used method of stabilizing a signal in the destination-clock domain. If the first flip-flop enters the metastable condition, it has a full clock period to stabilize before the second flip-flop samples it. Only the second-stage value is propagated to the other parts of the system.



In practice, it is indispensable to minimize the wire propagation delay between the flip-flops and to avoid inserting any logic in between.

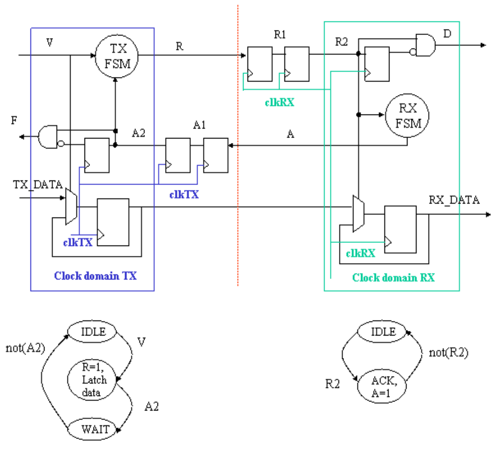
In the Esterel design, the synchronization structures must be implemented inside the control part. The code we use for them is the following:

```
module Resynchro:
input R;
output D : reg;
signal R1 : reg in
  sustain next {R1 <= R,
                    D <= R1}
end signal
end module
```

A more complex synchronization protocol

The double-stage synchronizer does not ensure that the signal remains stable long enough for the destination circuit to sample it once and only once. This can be a problem if a system asserts a signal in a fast source-clock domain and sends it to a very slow destination clock domain. The source must hold the signal for multiple clocks, for the logic of the slower destination-clock domain to detect it. On the other hand, if the system asserts a signal in a slow clock domain and sends it to a very fast clock domain, the receiver logic may detect the signal multiple times, mistaking it for multiple events. A handshake protocol such as the *push synchronizer* [6] solves the problem. We have used an Esterel implementation of it for another design.

The push synchronizer described in [6] comprises two synchronization circuits that envelop the data lines, implementing a complete handshake protocol. The Request (R) and Acknowledge (A) lines are synchronized by the receiver and sender respectively. The two synchronizers connect two simple finite state machines that implement the required protocol. A send request (V, true for a single cycle) latches data into REGs in the TX clock domain and starts the transmitter FSM. The synchronized request (R2) latches the data into REGs in the RX clock domain and triggers the receiver's FSM. The receiver is given a single-cycle "data-received" signal. The protocol is sometimes modified so that A is set as soon as the received data are latched, but removed only after the receiver had had an opportunity to use the data.



Experiments up to synthesis

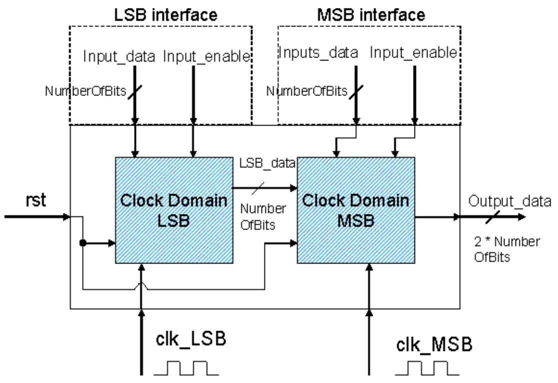
The multi-clock methodology and the resynchronization methods described above have been applied to two tests:

- The implementation of the push synchronizer for data-exchange devices
- The design of a complex video IP

The handshake protocol

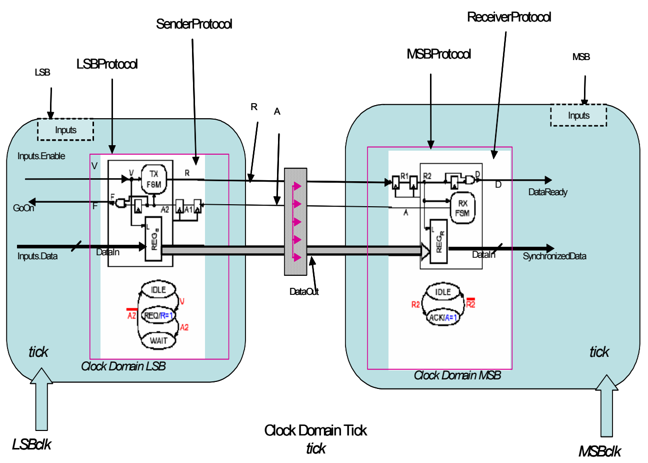
Description

The design is made of clock domains LSB and MSB transferring 4-bits data (*NumberOfBits* = 4). The LSB module receives an input bus *Inputs_data_LSB* and an enable signal *Inputs_enable_LSB*. When enable is set, LSB latches the bus with its own clock, and sends it out to MSB on *LSB_data* bus. The MSB module has an input bus *Inputs_data_MSB* with an enable *Inputs_enable_MSB*. When its enable is set, it latches its input in a register with its *clk_MS*B clock. It also resynchronizes the *LSB_data* bus from the LSB module. MSB concatenates the received LSB data with its own input data from *Inputs_data_MSB*, and sends it out to *Output_data* after a final re-sampling.



Esterel design

The Esterel simulation design uses three modules: the clock generator, to create clk_LSB and clk_MSB; the "clock domain LSB" module (TX); and the "clock domain MSB" module (RX). Synchronization is done using a push synchronizer. The TX module contains a sender protocol FSM, the synchronization registers producing the acknowledge signal A and the register latching the transmission data; the RX module contains the receiver protocol FSM, the resynchronization registers for the request signal R, and the output register to latch the data coming from TX side.



Simulations were run applying different clock frequency relationships through the variation of the clock generator parameters, showing correct behavior.

Synthesis results

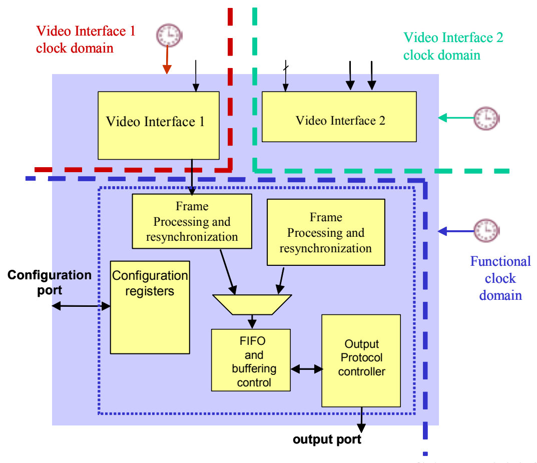
Two VHDL entities were generated for each clock domain and they were connected in a top-level VHDL. Synthesis was run with Synopsys Design Compiler, 2003.03, and the comparison with the hand-made VHDL showed no area overhead. Simulation was run through a VHDL test-bench, showing correct behaviors with all the frequency relationships foreseen.

Video IP design

Description and Esterel design

A more complex design was chosen to further test the methodology. The Video IP receives different frame formats from two video interfaces, working with different bus protocol and formats. It decodes them and packs the pixels into different data formats which are buffered into a FIFO and then transmitted outside using an OCP interface.

Three clock domains are involved: two for the video interfaces and one for the internal buffering & data output. The design structure is shown


Video Interface 1 clock domain
Video Interface 2 clock domain
Video Interface 1
Video Interface 2
Frame Processing and resynchronization
Frame Processing and resynchronization
Configuration port
Configuration registers
FIFO and buffering control
Output Protocol controller
Functional clock domain
output port

below:
A simulation framework has been implemented with three clock generators and the three main modules. The double-stage synchronizer has been used to secure data writing from the video interfaces to the buffering systems.

VHDL generation and synthesis results

A top-level VHDL has been created connecting the different clock domain components and the clock and reset pins have been connected to the core clock/reset input pins. Synthesis has been run with Synopsys Design Compiler, version 2003.03, and the results have been compared to the one obtained by the hand-written Verilog design, under the same synthesis constraints. No DFT or timing issues have been found, and a total 5% gain in area occupation has been obtained. The table below reports the percentage of gain in area occupation obtained for the three clock domains.

Functional Clock	**-1.6%**
Video Interface 1	**26%**
Video Interface 2	**2.11%**

The Esterel VHDL has been tested under a Reference Test Bench developed for the Verilog design, showing a correct behavior of the resynchronization structures.

Conclusion

We have presented a practical methodology for multi-clock design using Esterel. Individual clock domains are modeled and synthesized using the current Esterel Studio tool.

For system modeling, clocks are viewed as standard signals in a global Esterel simulation design, and communication synchronizers are created from Esterel modules.

For synthesis, individual clock domains are synthesized using the standard Esterel flow and linked by manually written HDL code. The overall methodology could be automated further in upcoming versions of Esterel Studio.

Bibliography

[1] – J.Jex and C.Dike, "A fast resolving BiNMOS synchronizer for parallel processor interconnect", *IEEE Journal of Solid-State Circuits*, vol. 30, pp.133-139, 1995.
[2] – D.J.Kinniment, A. Bystrov and A. Yakovlev, "Synchronization Circuit Performance", *IEEE Journal of Solid-State Circuits*, vol. 37, pp.202-209, 2002.

[3] – W.J.Dally and J.W.Poulton, *Digital System Engineering* (Eds.): Cambridge University Press, 1998.

[4] – T.H.-Y.Meng, *Synchrnonization Design for Digital Systems* (Eds.): Kluwer Academic Publishers, 1991.

[5] – D.J.Kinniment and J.V.Woods, “Synchronization and Arbitration Circuits in Digital Systems”, *Proceedings of IEE*, vol. 123, pp 961-966, 1976.

[6] – R. Ginosar, “Fourteen ways to Fool your Synchronizer”, *VLSI Systems Research Center, Technion – Israel Institute of Technology*, 2003.

[7] Basant Rajan, R. K. Shyamasundar: Multiclock Esterel: A Reactive Framework for Asynchronous Design. IPDPS 2000: 201-210

[8] G. Berry*, The Foundations of Esterel*, in Proof, Language, and Interaction, Essays in Honour of Robin Milner, MIT Press, YEAR="2000"

[9] G. Berry, *The Constructive Semantics of Pure Esterel,* draft book available at www.esterel-technologies.com, 2000

[10] G. Berry, E. Sentovich, *Multiclock Esterel*, Proc. CHARME'2001, Edinburgh, Springer-Verlag LNCS 2144.

[11] Esterel Technologies, The Esterel v7 Reference Manual, available at www.esterel-technologies.com, 2002