A Smart Synchronizer - Pragmatic way to cross asynchronous clock domains.

Priyank Parakh
Advanced Micro Devices, Inc.
90 Central Street,
Boxborough, MA, USA.
(1)-978-795-2500

Steven J Kommrusch
Advanced Micro Devices, Inc.
2950 E. Harmony Rd,
Fort Collins, Co 80582
(1)-970-226-9500

ABSTRACT

In order to achieve satisfactory verification coverage in an asynchronous design, it is highly desirable to model a synchronizer with all the checks that can help catch the problem. Modeling uncertainty caused by metastable values at the output of the synchronizer is one of them. Due to the inherent determinism in RTL-level simulations, it is necessary to take explicit steps to model the inbuilt uncertainty associated with the act of synchronization. However, the task of effectively modeling this uncertainty in a manner that draws out design flaws can be a challenge.

Many papers published so far have emphasized the usage of very aggressive ways to model various checks inside the synchronizer but what they failed to do is to consider the overhead, like performance issues, false negatives etc, caused by these aggressive techniques.

This paper explains a more balanced approach which is smart enough to regulate the aggression of the synchronizer based on different synchronization activities.

Keywords:

Metastability, clock domain crossing, random delay, aggressive techniques, gray-encoded pointers, FIFO.

1. INTRODUCTION

Clock domain crossing (CDC) verification of the current generation of complex, deep sub-micron (DSM) designs has become a critical step in the design process. Today's large silicon devices are comprised of a growing number of IPs which in turn contain an ever increasing number of asynchronous clock domains from which stem a large number of cross over signals and a variety of design styles for synchronization of CDC signals. In order to achieve adequate verification coverage in a design like this, it is highly desirable to model a synchronizer with all the checks that can help catch problems. In hardware, the effects of metastabilty [1-6] are unpredictable, but normal simulation provides predictable results. Hence, due to the inherent determinism in RTL-level simulations, it is necessary to take explicit steps to model the inbuilt uncertainty associated with the act of synchronization.

2. PROBLEM STATEMENT

Modeling a clock-domain-crossing (CDC) synchronizer is a more involved process than initially meets the eye. Generally speaking, modeling techniques must strive to meet multiple goals, some of which can be at odds with each other. In addition, since simulation is a crucial medium to validate and debug design flaws, it is essential that reproducibility be an innate characteristic of the solution chosen. Various design teams across different companies have handled this task in a variety of ways which has caused misunderstandings and bugs found late in the design process

Many papers published so far have emphasized the usage of very aggressive ways to model various checks inside the synchronizer but what they failed to do is to consider the overhead caused by these aggressive techniques. For example, the most aggressive way to model a synchronization event is to randomly introduce an extra latency of one destination clock-cycle when propagating any transition. Unlike the real-life behavior of flip-flops, using this behavior will introduce synchronization uncertainty in cases well outside the "setup/hold" window that characterizes the physical design. This technique can be very effective for simple synchronization activities, but for more sophisticated ones like a "bussed" event with gray encoding this technique is at odds with the property that makes gray-coded counters so effective – only one bit changes per update.

2.1 Aggressive technique

Ultimately, the goal of a synchronizer model is to expose all clock domain crossing design flaws as quickly and efficiently as possible. In order to accomplish this goal, it is necessary to strive for an aggressive behavior in the model in case the design is left with undiscovered flaws. Out of all the modeling techniques discussed so far in the design world, one of the most commonly used techniques is to randomize the data with respect to the destination clock.

```
always @( posedge CK )

        prevD   <= currD;
        currD   <= D;

  // The net effect of this is to randomly decide whether to
  // take the current value of the signal to be synchronized,
  // or the value that was presented before the last transition.
  //
        Q <= ( random_evaluation ) ? prevD : currD;
```

Using this simplistic technique can be very effective for simple synchronization activities, but simplistic synchronization techniques tend to lead to design inefficiencies that can result in performance limitations or bloat to the design in an effort to recoup performance. To combat this, designers will often leverage properties of “bussed” events to achieve their CDC synchronization goals. A common example of a “bussed” event that is leveraged is that of gray-coded counters. Unfortunately, the simple technique described above is at odds with exactly the property that makes gray-coded counters so effective – only one bit changes per update.

3. SOLUTION

When considering an approach to modelling a synchronizer, two important criteria must be taken into account. To begin with we must know when to model the uncertainty and then how to model the uncertainty.

3.1 Modelling Uncertainty

It is the belief of the authors that the proposal outlined below achieves the goals in a manner that is as aggressive as possible while still yielding to the properties that are leveraged for “bussed” events. The proposal is divided into two layers. Basic layer which essentially is a single-bit synchronizer that classifies candidate events for randomized uncertainty modelling, and a filter layer which is a multi-bit level that provides a filter to qualify the candidates elected by the basic level. Each of these levels will be described using pseudo code, but the intent of the code snippet is to impart an understanding of the technique and not to present final code.

Basic layer

```
reg   currD;
reg   prevD;
reg   prevVld;

initial begin
  currD   = D;
  prevD   = D;
  prevVld = 1'b0;
end

always @( D or posedge CK ) begin

if ( posedge CK )

  // We are taking a sample, so past transitions are moot
  // for future samples as they would be stable by the time
  // that the next sample edge arrives.

  prevVld <= 1'b0;  // clear our candidacy for uncertainty
else begin

  // When we see a transition in the data, we mark our
  // candidacy for uncertainty modeling. This is our aggressive
  // mode of modeling.
  //
  // Note, the use of non-blocking assignments. Since the
  // clock edge takes precedence, we are not executing this
  // code concurrently with a clock edge. Consequently, the
  // choice of blocking vs. non-blocking should be a matter
  // of which simulates faster.

  prevVld <= 1'b1;
  prevD   <= currD;
  currD   <= D;
 end
end

always @( posedge CK )

 // Note, the signal 'transitionVld' is an input to this
 // level of the model and is the hook for the higher-level
 // filter to "dial-back"; the aggressiveness of the modeling
 // of "when"; to apply randomized uncertainty in the delay.

 if ( prevVld && transitionVld ) begin

  // Here, the technique for the random evaluation is not
  // detailed. The technique for this is discussed later.
  //
  // Nevertheless, it is important, for performance reasons,
  // to understand that the random stream should only be
  // pulled when an actual randomization decision is required.
  //
  // The net effect of this is to randomly decide whether to
  // take the current value of the signal to be synchronized,
  // or the value that was presented before the last transition.
  //
  // Note, it would have been simpler to use ~CurrD without
  // storing prevD, but that technique limits
  // us to { 0, 1 } as signal values, whereas the technique
  // outlined here is specifically capable of dealing with
  // { 0, 1, X } as signal values without polluting the results.

  Q <= ( random_evaluation ) ? prevD : CurrD;

 end else
  // Data not flagged for randomization was stable at ‘currD’
  Q <= CurrD;
```

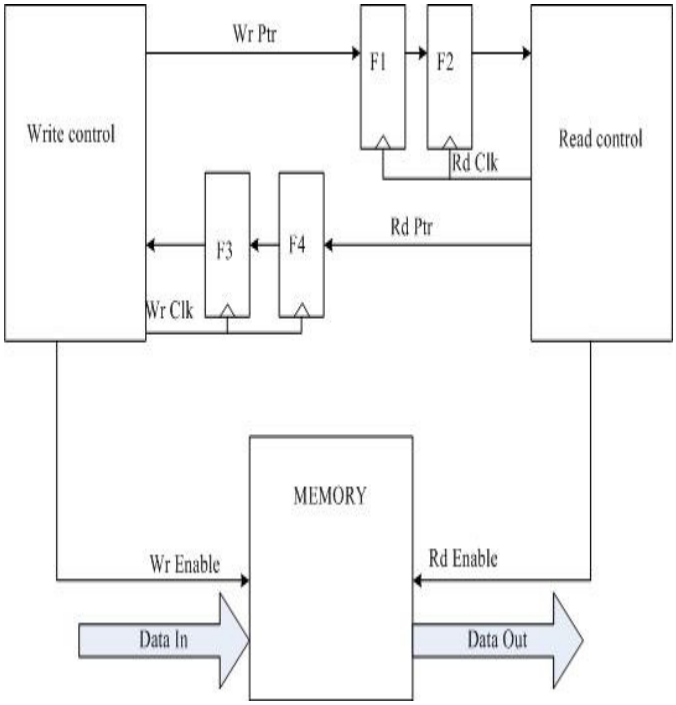
Multi-bit filter

```
input  [(VEC_SIZE-1):0]  D;
...

reg    [(VEC_SIZE-1):0]  currD;
reg    [(VEC_SIZE-1):0]  prevD;
reg    [(VEC_SIZE-1):0]  transitionVld;
integer                  idx;

initial begin
  currD = D;
  prevD = D;
end

always @( D ) begin
  prevD  = currD;
  currD <= D;

  for ( idx = 0; idx < VEC_SIZE; idx = idx+1 )

    // any bit that has changed since the last time the
    // data vector changed will be marked as a valid bit
    // for synchronization candidacy. Any bit that remains
    // unchanged will have its "candidacy" overridden since
    // it is known to be stable, thus preserving the
    // leveraged property of the "bussed" event.
    //
    // this signal is then passed down to the individual
    // lower-level synchronization models.

    transitionVld[idx] = ( prevD[idx] !== currD[idx] );
end
```

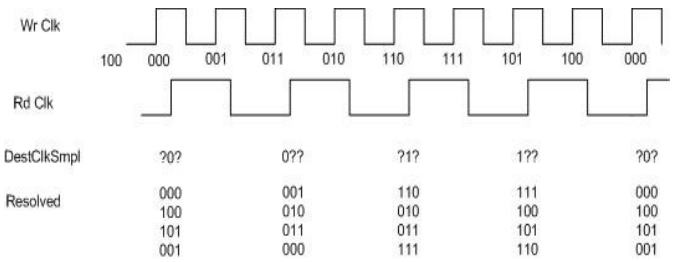
As can be seen from the code samples above, transitions on the input data are the primary means for determining when the capture should or should not be randomized. By sensitizing the code to transitions of the input data, we are inferring the source clock domain since each data transition should follow a clock edge in the source clock domain. Using this property, the aggression is “dialed-back” for bussed synchronization events by permitting randomization for only those bits that have changed in the most recent update. Any bits that have remained stable during the most recent update are blocked from having their capture randomized, even if the lower-level synchronizer has detected a transition since the last capture edge. This “filter” leverages the fact the property that a signal that has been stable for at least one source clock cycle will have been stable at the sample flop for a sufficiently long time that its capture will be certain.

4. CASE STUDY

Consider a FIFO, as shown above, used to transfer multi-bit data across a clock domain. Now, imagine a scenario where the destination clock is known to always be at least half as fast as the source clock. The designer might assume that the gray code counter could never change by more than two counts. But if the source clock is writing every cycle and destination clock is exactly half the source clock rate, then in real silicon, we might see a case where the counter jumps three counts in one cycle.



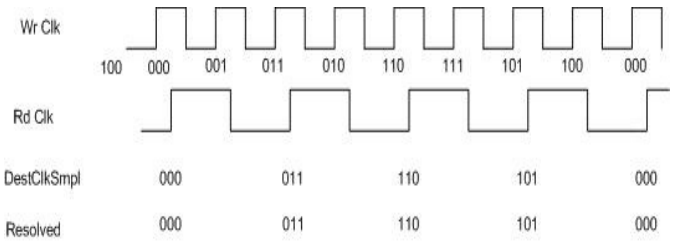
If the designer is using the aggressive technique described in this paper, his randomizer will try to process the data at every edge of the destination clock and will randomize any changed bit at the capture time.



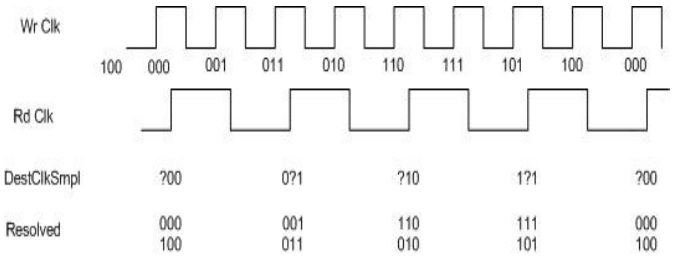
Now as can be seen above, the simulation data at the first destination clock is 000 and in the consecutive edge, due to the frequency relationship between the clock and the fact that we are gray encoded the pointers, the simulation data will be 011. This implies that in between two destination clock transition two bits have changed which means after standard aggressive randomization, simulation can see any of the four shown choices 000, 001, 010 and 011. If the simulation were to process 010 then it might eventually lead to false

functional failures as that entry has not even been written yet. This false negative might cause simulation errors and might consume both time and resource to fix.

To avoid this overhead, a designer sometimes takes a "drastic step" including shutting down the randomization model completely, hence using just back to back flops and gray encoded pointers. Following is the waveform of the "Resolved" value with randomization disabled.



Shown below is the case which might slip through the cracks if we were to follow the "drastic step" and how our synchronizer model will expose that.



In zero time delay simulation, the first Rd Clk rising edge sees 000 from the source domain, but the last bit to change was 1 (from 100), so in silicon, this bit might go metastable and hence our smart synchronizer will capture the transition and will introduce a random delay and will produce either 000 or 100. Similarly when the Rd clk captures 011, the actual value could be 001 or 011 and then the next destination clock cycle might produce 001 or 011, and the next might produce 010 or 110. So there is a random way to produce the sequence 000, 001, 110. This could actually happen in real silicon based on metastability resolution and so our smart synchronizer will correctly expose a design problem if the design could not handle a 3-step increment in the counter

5. SUMMARY

With a growing number of clocks in today's SOC designs, increased design complexity, and pressure for first silicon success, all clock and timing issues have become a verification challenge.

Existing techniques used to model uncertainty in synchronizers are too aggressive to be used effectively with all different synchronization schemes forcing the user to make a choice when it is safe to turn off and on this feature. The technique presented in this paper eliminates that decision making process and makes clock domain verification more stable and sustainable.

6. ACKNOWLEDGEMENT

The authors are grateful to the many imaginative designers whose innovations ended up in this paper. Their names are kept in confidence. The anonymous referees have provided interesting viewpoints which together have improved the quality of the paper; the authors alone should be blamed for any remaining mistake.

7. REFERENCE

[1] Chris Kwok, et al, "Using Assertion based Verification to verify Clock domain crossing", DVCON 2003

[2] Clifford E. Cummings, "Clock domain crossing design and Verification techniques using System Verilog", SNUG 2008

[3] Clifford E. Cummings, "Synthesis and Scripting techniques for Designing Multi- Asynchronous clock designs", SNUG 2001.

[4] Ginsor Ran, "Fourteen ways to fool your synchronizer" Asynchronous Circuits and Systems, 2003.

[5] Parakh Priyank, Sabbagh Roger, "Clock domain crossing verification of a CPU/GPU design", DVCON 2010.

[6] Patil, Girish, IFV Division, Cadence Design Systems. *Clock Synchronization Issues and Static Verification Techniques.* Cadence Technical Conference 2004