USING FORMAL TO EXAUSTIVELY DETERMINE UNSAFE CLOCK RATIOS BETWEEN ASYNCHRONOUS BLOCKS

Eric Hendrickson1, Bill Au2, Richard Llaca3, Joe Hupcey III4
1Verification Lead (Jet Propulsion Laboratory, Pasadena, CA, USA.
Tel: +1-818-354-4321 Eric.L.Hendrickson@jpl.nasa.gov),
2Account Technology Manager (Mentor Graphics, A Siemens Business, Fremont, CA, USA
Tel: +1-510-354-7400 Bill_AU@mentor.com),
3Senior Applications Engineer (Mentor Graphics, A Siemens Business, Austin, TX, USA.
Tel: +1-512-425-3000 Richard_Llaca@mentor.com),
4Verification Product Technologist (Mentor Graphics, A Siemens Business, Fremont, CA, USA
Tel: +1-510-354-7400 Joe_Hupcey@mentor.com).

Introduction:

If there is an overarching principal in spacecraft design, it can be summarized in a word: “reuse”.
The reuse of flight proven hardware is a time-honored practice that has been shown to dramatically reduce the chance of hardware failure in the demanding environments the spacecraft must endure. The systems design challenge this creates is that there are many different, IPs, processors, ASICs & FPGAs, and whole instruments (henceforth referred to as simply “blocks”) from different suppliers, and of different vintages, must be successfully integrated with any new hardware.

Drilling down a level into this integration challenge, these disparate hardware blocks rarely share the exact same operational clock frequency. As such, a critical design challenge is to make sure a source block isn’t over- or underclocking the receiving block. The degree of difficulty here is regularly increased by the fact that it is common for receiving blocks to NOT support any “back pressure” signaling mechanisms to tell the sending block to slow down or halt data transmission until it can catch-up. Furthermore, the duty cycles and phases of the default clocking in each block can also differ.

Obviously system designers readily avoid pairing blocks that are known to operate at drastically different clock frequencies. But in the majority of cases the operational clock frequencies of prospective source and destination block are “in the ball park” of each other, so the key challenge is somehow determine if there are any unsafe ratios within a given range of supported frequencies between source and destination clocks that will cause the system to malfunction.

Abstract: In this paper we will show how formal property checking can be used to exhaustively sweep the range of allowable clock ratios between any two blocks to determine if their correct operation is maintained. First, we obtain, or create, models of the sending and receiving blocks (in either VHDL or Verilog). After basic functional validation of these models, we then write the property set to reflect the functional requirements of each block. As the clocks between them allow for a range of frequency ratios, we employ the formal analysis principal of nondeterminism to lock-on to a particular supported frequency ratio and prove that the functional requirements are still met. Consider the following pseudo code example:

```
logic fast_clk;
logic ]N:0] some_ratio; //constrained to be stable and within the valid range
reg [M:0] barrel_counter;
logic ratio_hit_n;
reg clk1;
reg clk 2;
…
assign ratio_hit = barrel_counter % some_ratio;
…
…
clk1 <= !clk1;
```

clk2 <= !ratio_hit_n? !clk2: clk2;
…
…

We apply formal non-determinism technique to constraining a constant ratio within the supported range. That ratio will then control when the user created clocks are passed to the clock ports of the blocks, hence driving when the functional properties within the blocks to be verified.
Prior solutions: Before formal solutions were employed, verification engineers used a simulation-based, directed test approach. Starting with the "middle" clock frequency, a series of parameterized tests were run with successivelyfaster and slower clock signals, as well as larger deltas of clock duty cycles and clock phases. The outputs of the tests for monitored for data coherency – i.e. keep checking the outputs until the data flow turns to garbage.

Obviously a key question with this methodology is how many intermediate variations in the clocking should be tested with in-order to determine the all the "process corners" where the communications channel is still effective. In short, given all the possible permutations of clock frequency, duty cycle, and phase, you could honestly conclude that you would need to run millions of test cases – check 1Mhz, 1.1Mhz, 1.2Mhz, and so on – and even then, after running these simulations you still couldn't be 100% sure if the design would work when the clock was driven at 1.21ms, 1.23ms or 1.125ms with different duty cycles and phases.

Indeed, employing a directed test and constrained-random verification flow we would occasionally see deadlocks in designs that we were sure had been thoroughly tested and configured to never deadlock. Clearly, we needed to employ an exhaustive approach. As such, formal property checking was the obviously candidate solution given the exhaustive nature of its analysis.

Results: With this formal-based approach, we are able to exhaustively discover the limits of the clock ratios in a relatively short amount of time – days vs. weeks with prior approaches; and with much higher confidence in the results. This success carries over to the high-level system integration – both in the whole system RTL simulation as well as the ultimate, real hardware integration and final test in the production facilities.

The modules under analysis were a tx and rx block where clock recovery work is done on the rx side internally. Both tx and rx run at asynchronous clock frequencies, so there are internal parameters that are used in both the tx and rx side to limit the effective transmit and receive rate between the two blocks. They support a simple parallel interface and valid signal, and serialize and deserialize the data between the two links. This flexibility allows for simple point-to-point communications over dedicated interfaces across multiple boards where layout flexibility is important.

The transmit and receive rates that are determined by the clock frequencies of the tx and rx domains. Internal divide downs set by parameters create a relationship such that the tx max bandwidth will never exceed the rx max receive bandwidth.

Initially, formal verification was used to verify that data integrity was preserved across the tx serdes to the rx deserdes. Fixed parameters and clock ratios with static period delays were used across all the different configurations, and a basic scoreboard was written that captured the data going into the parallel input of the tx. The interface is meant for simple command and response for things like reconfiguration, register writes, DMAs, and telemetry response back to the spacecraft flight software. So there is only one outstanding transaction on the link at a time, no pipelining. This simplicity allowed for a simple scoreboarding mechanism to be used such that when the rx data valid went high, it was compared to the data captured in the scoreboard. This allowed a simple assertion to be written similar to the pseudocode below:

rx_data_valid |-> rx_data == captured_tx_data;

The module interface allows for both tx and rx sides to be set to an arbitrary bit width, as long as they match. Typical configurations are 32 bit for transmit and receive sides. Because this can cause state space explosion due to the number of counters on both the tx and rx sides, the verification configuration was parameterized down to 2, 4, and 8 bits. Full convergence using the above assertion across multiple bandwidth and clock configurations was achieved at 2 and 4 bits. 8 bits and above was run to ensure that no counterexamples were found, but full convergence was impossible after multiple 10-12 hour runtime sessions. Full convergence on 2 and 4 bits took about 3 and 6 hours respectively. Another set of assertions were used to check error conditions and link recovery mechanisms. This set of assertions disabled the data integrity checks and enabled unconstrained nets to introduce error conditions into the serial interface to test the timeout recovery mechanisms on the rx side and link reset capabilities based off of timeout. The purpose of this mechanism is to allow for link recovery due to potential SEU in a radiation environment such as low Earth orbit.

Some of the link recovery mechanisms are based off of an inactivity counter such that when the first serial received bit is received by rx, the rest should be received within a deterministic amount of time before the countdown expires. Inactivity following received bits will cause the counter to increment to a threshold, changes on the link will reset the counter. If the counter expires, indicating a locked up tx side, the rx will self-reset and reinitialize its internal shift counter. A few bug scenarios were found during power on reset, that within a certain time bound after reset, it was possible for the rx link inactivity timer to expire. A condition was found where immediately coming out of reset, the inactivity timer would begin to count. If the tx started transmitting data within this time window, then it was possible to create a scenario where the first bits of transmitted data would arrive right when the rx's inactivity timer expired. So any new transmitted data within the inactivity timer threshold window coming into the rx directly out of reset would've resulted in data corruption. This resulted in a minor fix to the inactivity counter to not count coming out of reset, a condition that was not caught in the lab.

The next steps after testing static clock ratios and transmit and receive rates was to create a generalized testbench configuration that would treat the parameters and clock ratios as part of the functional design. Testbench code has been created to use counters that can be constrained such that once they hit a specific threshold, they reset to zero. The threshold can be constrained as a free floating signal into the design such that no matter what value it picks, it stays static once it has been selected. This can be done easily using SVA system functions such as $stable(). A clocking bit is then used to track the polarity state of the clock that is generated based off of this counter and dynamic threshold. Pseudocode for this appears below:

```
//constraints
cntr_threshold > min_ratio && cntr_threshold < max_ratio;
$stable(cntr_threshold);

logic testbench_clk;        //fastest running testbench clock, set by the formal tool
logic testbench_rst;        //testbench reset
logic [3:0] cntr_threshold;
logic clk_polarity;
logic [4:0] clk_counter;

always_ff(@posedge testbench_clk)
        if(testbench_rst)
        …
        else begin
                clk_counter <= (clk_counter < cntr_threshold) ? clk_counter + 5'd1 : clk_counter;
                clk_polarity <= (clk_counter && clk_counter == cntr_threshold) ? !clk_polarity : clk_polarity;
                end
```

Using the above as a building block for creating a clock that can be dynamically selected allows the formal tool to explore which potential clock ratios in tx and rx will cause the data integrity assertion to break. When it does provide a counter example, it will report what the static clock ratios selected.

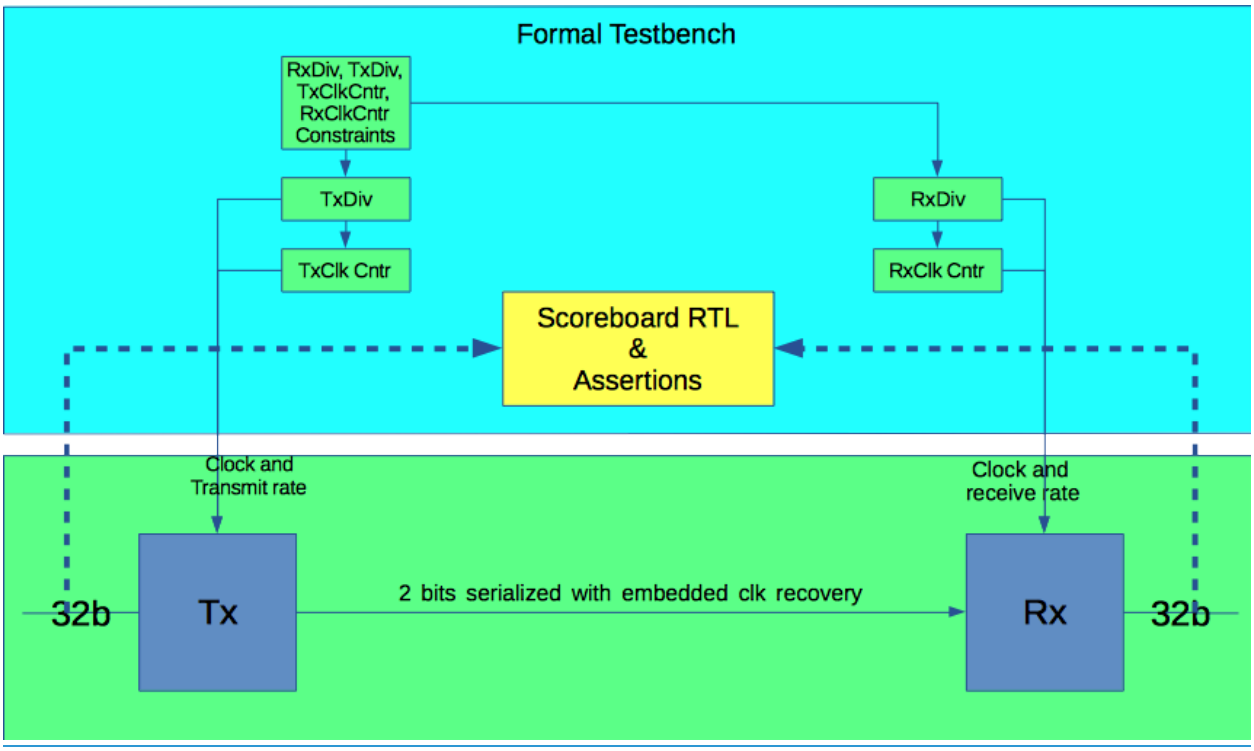
A couple of things must be considered when taking into account this approach. This code above assumes a 50% active/inactive duty cycle for the clocks into the DUT. If this assumption doesn't hold true, then the verification engineer must account for this in their formal testbench model. The other thing that isn't published here for proprietary reasons is the ultimate constraint in determining what the correct clk ratios are for tx and rx given a specific divide down parameter that limits the transmit and receive rates on both sides. Specifically referred to as RX_DIV and TX_DIV, these act as internal rate limiters to ensure that the transmitter never overwhelms the receiver. Since these were previously static parameters into the design, they must now be converted to dynamic inputs and constrained in a similar matter as the above cntr_threshold.

The downside to all of this extra logic is that it can cause state space explosion. Even at static RX_DIV, TX_DIV, and clock ratios, anything above 8 bits would need a couple of days to achieve full convergence on properties. Given that the design is parameterized around data width, reducing the design complexity via datawidth reduction is an acceptable approach to reducing the overall state space, especially if it is the result of counter width with no interesting counter threshold values except for the reset threshold. Another potential hang-up is that because the counter threshold is effectively half the clock period of the counter, there has to be some conversion done to convert clock speed and ratio to determine the value of the counter threshold. This is actually not that difficult.

Now that the TX_DIV, RX_DIV, and clock ratio threshold static behavior has been set, a final key constraint has to be written that determines what their expected values should be relative to one another. The specific equation isn't given in this presentation due to proprietary concerns, but at a high level, the constraint ensures that at a given TX_DIV+tx_clk_ratio, the maximum transmit rate, will not overwhelm the receiver. This is given as a fractional ratio of the permissible values and their relationship. Once this is defined by the designer, the relationship specified between these 4 values will be tested mainly by allowing the formal analysis engine to change these variables while searching for a counterexample in the data integrity assertion via the scoreboard. Because of state space explosion problem, full proof may or may not be achievable. But if full convergence is achieved on lower parameter bit widths, then it is usually acceptable given some assurances in design and parameterization approaches to assume that these will extended to larger bit width configurations.

When this is a concern, running a bug hunting flow at higher bit widths can add some confidence to this approach. But the designer should be aware if this will really be an issue. Odd bit widths were also tested, and a few data integrity issues were found in static configurations because the designer assumed that even bit widths, namely multiples of 2, would always be used. It is important, when making these assumptions, that fringe configurations are verified to ensure any design parameter simplification assumptions that might carry over to the actual configuration used in the real design configuration are valid.

At the time of writing, the dynamic variation of the tx and rx module is still undergoing development. Results will be reported when they are available. A similar approach has also been used on a previous design for a clock switch that was supposed to be able to switch between fast and slow clocks without creating glitches/runts, and supported a maximum and minimum set of input clock ratios. Limits on the assumption of what the max and min clock ratios were specified, and found bug scenarios where glitches and runts could be created, which resulted in changes to the design itself.


Formal Testbench
RxDiv, TxDiv, TxClkCntr, RxClkCntr Constraints
TxDiv
TxClk Cntr
RxDiv
RxClk Cntr
Scoreboard RTL & Assertions
Clock and Transmit rate
Clock and receive rate
32b
Tx
2 bits serialized with embedded clk recovery
Rx
32b