2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

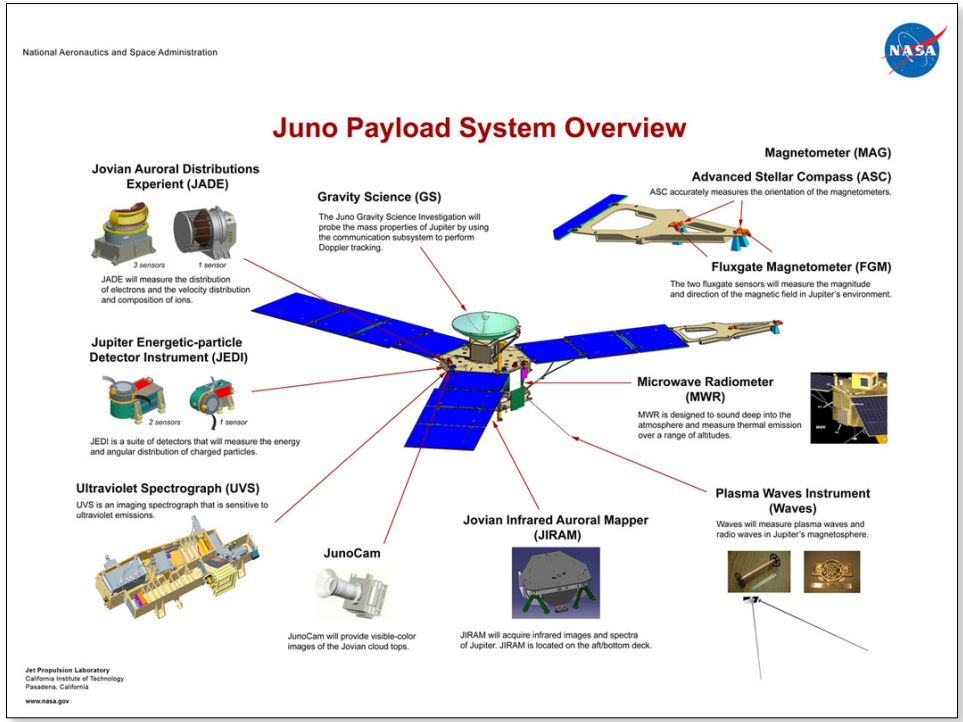
Using Formal to Exhaustively Determine Unsafe Clock Ratios Between Asynchronous Blocks

Eric Hendrickson, Verification Lead; Jet Propulsion Laboratory
Bill Au, Account Technology Manager; Mentor, A Siemens Business
Richard Llaca, Sr. Application Eng; Mentor, A Siemens Business
Joe Hupcey III, Verification Product Technologist; Mentor, A.S.B.

Jet Propulsion Laboratory
California Institute of Technology

Mentor®
A Siemens Business

A Typical DUT



- Controllers & instruments can operate at different clock rates
- Supported duty cycles & phases in each block can also differ
- Most blocks can’t “backpressure” their input data flow

The Verification Challenge

- How to verify the range of allowable clock ratios between any two blocks to determine if their correct operation is maintained?

- i.e. How to make sure a source block isn't over- or underclocking the receiving block?

- Double bonus:
  - Duty cycles and phases can be different
  - "Back pressure" signaling mechanisms are NOT supported (so we can't rely on "handshaking" to save us!)

The Simulation-Based Approach

- Parameterized, scripted battery of directed tests
- Starting with the “middle” clock frequency:
  - Set successively faster and slower clock signals
  - Move to larger deltas of clock duty cycles and clock phases
- The outputs of the tests are monitored for data coherency –
  i.e. keep checking the outputs until the data flow turns into garbage

Big Problem with the Sim-Based Flow: Checking All "Process Corners" is Impossible

- Legal, possible permutations of clock frequency, duty cycle, and phase relationships, is practically infinite

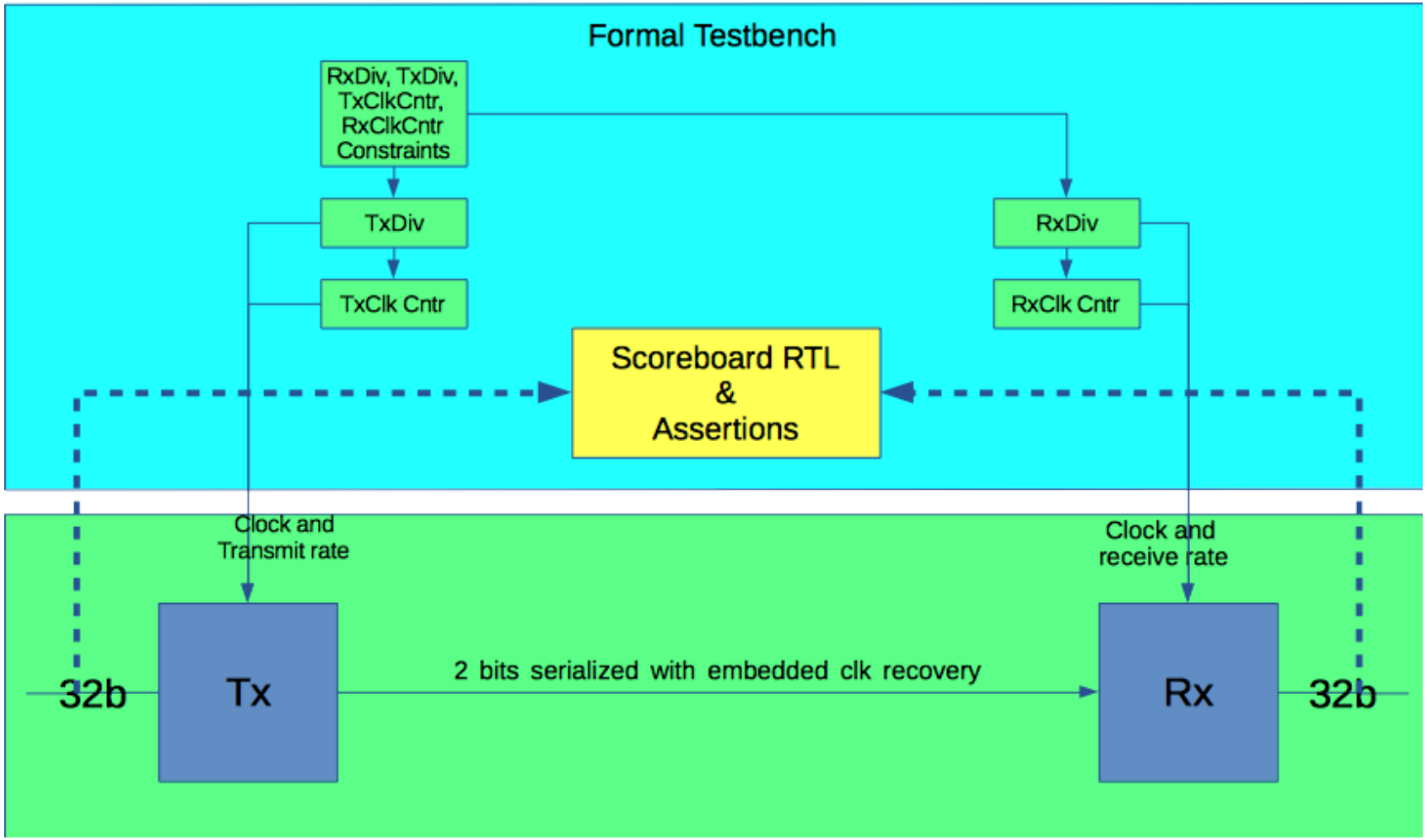
- What's the minimum number of tests we need to run?

Bottom Line

•The simulation flow would still let deadlocks "escape" even in designs that we were sure had been thoroughly analyzed and tested

•Only expensive, late-in-the-game system tests would save the day

Testbench Setup



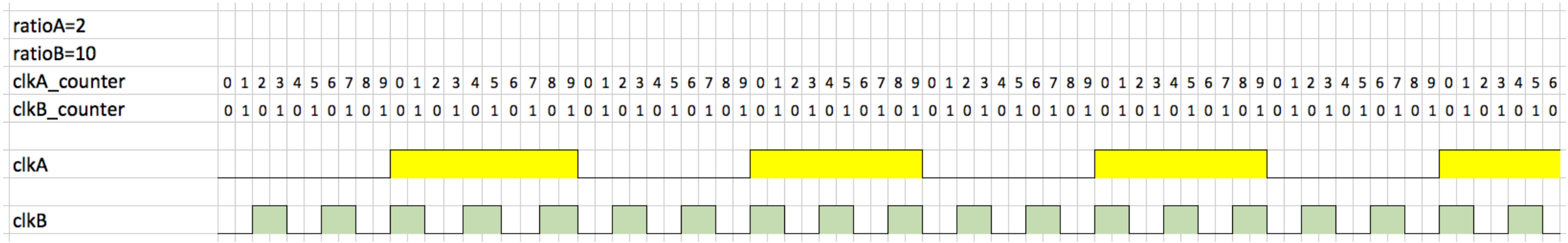
Theory of Operation

- Overall clocking relationship for permissible clock ratios need to be understood as well as permissible range of clock frequencies.
- DUT now includes generated clocks as part of COI of design.
- Use testbench constraints to generate clocks as part of the DUT.
- Constraints are a combination of SVA assumptions and testbench counters
- Counters will determine clock ratio frequency for source and destination clocks
- Counter thresholds that reset the counter are left as unconstrained inputs
  - When a threshold is selected, SVA constraint keeps it static through the entire analysis time
  - Overall threshold SVA constraint describes permissible range of ratios between source and destination clock
- Formal engine will select a static counter threshold for each clock ratio

Theory of Operation cont.

- Ex: Basic FIFO crossing that supports multiple clock ratios between domains A and B
  - Fundamental goal: Prove data going across a FIFO that supports multiple clock ratios is preserved
  - clockdomain A can be 50-200mhz, 50mhz increments, 4 ratios
  - clockdomain B can be 500-700mhz, 50mhz increments, 5 ratios
  - Simulation would be required to test 20 static clock configurations to be tested
  - Ratios simplify to 1-4 for A, 10-14 for B
  - We invert the ratios to determine the relationship of possible values for clkA and clkB relative to one another
    - If clkA ratio is 2, and clkB ratio is 10, then tb_clk should cause counterA to reset when it hits 10-1, and counterB to reset when it hits 2-1. Waveform explaining this relationship below.

Theory of Operation (cont)



Pseudocode

```
//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 [4:0] 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 - 1) ? clk_counter + 5'd1 : 5'd0;
                    clk_polarity <= (clk_counter == cntr_threshold - 1) ? !clk_polarity : clk_polarity
                    end
```

Results from jWire

- 3 bugs found
  - Reset of uninitialized flop was missing
  - Inactivity reset counter in the jWire Rx was causing data corruption right after poweron
  - Odd bit widths of parallel data into serdes interface would cause a polarity inversion on one of the reconstruction clocking bits on b2b transactions
- Run times expected
  - 6-700 properties, many parameterized cover properties for state space completeness.
  - 5-6 hours of run time on reduced design (parameterized bitwidth, fifo depth) to achieve full proof.
  - 10+ hours on bitwidths of 8 or higher for serdes implementation
  - For higher bitwidths, bug hunting flow may be more appropriate
- Conclusion
  - This type of proof is good for mission critical interfaces and blocks that can be parameterized to achieve simpler complexity.
  - Long run times can be expected, but when counter examples are found, they tend to be extreme corner cases that would’ve eventually failed in silicon that would’ve been extremely difficult to find in simulation.

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Summary

- With this formal-based approach, we are able to discover the limits of the clock ratios in days vs. weeks
- Leveraging the “non-determinism” property of formal analysis enables us to effectively sweep all possible inputs in parallel
- Exhaustive nature of the results significantly reduces risk!
- We expect to yield significant savings in whole system RTL simulations, through to final production tests on the flight hardware