Pragmatic Simulation-Based Verification of Clock Domain Crossing Signals and Jitter using SystemVerilog Assertions

Mark Litterick (Verification Consultant)

mark.litterick@verilab.com

Copyright © 2006 Verilab

Introduction

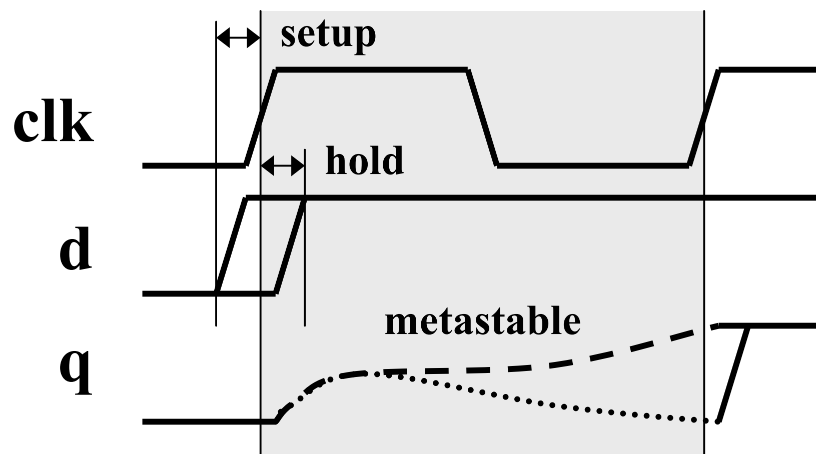
Clock Domain Crossing signals require special verification techniques

- Overview of metastability : what is the problem?
- Overview of ABV : why is it appropriate?
- Practical methodology using ABV for CDC verification
  - basic steps for CDC verification; independent of tools
- Analysis of SVA for synchronizer circuits
  - targeted at simulation-based validation
- CDC jitter emulation
  - using ABV and leveraging coverage to validate convergence
- Example project results

Copyright © 2006 Verilab

Metastability

- Violation of setup and hold times causes metastability
- Analogue effect in real-world transistor circuits
  - metastable value decays to 0 or 1; but can't predict which!
  - Mean Time Between Failures statistically predicts if metastable value will decay in time for next clock event

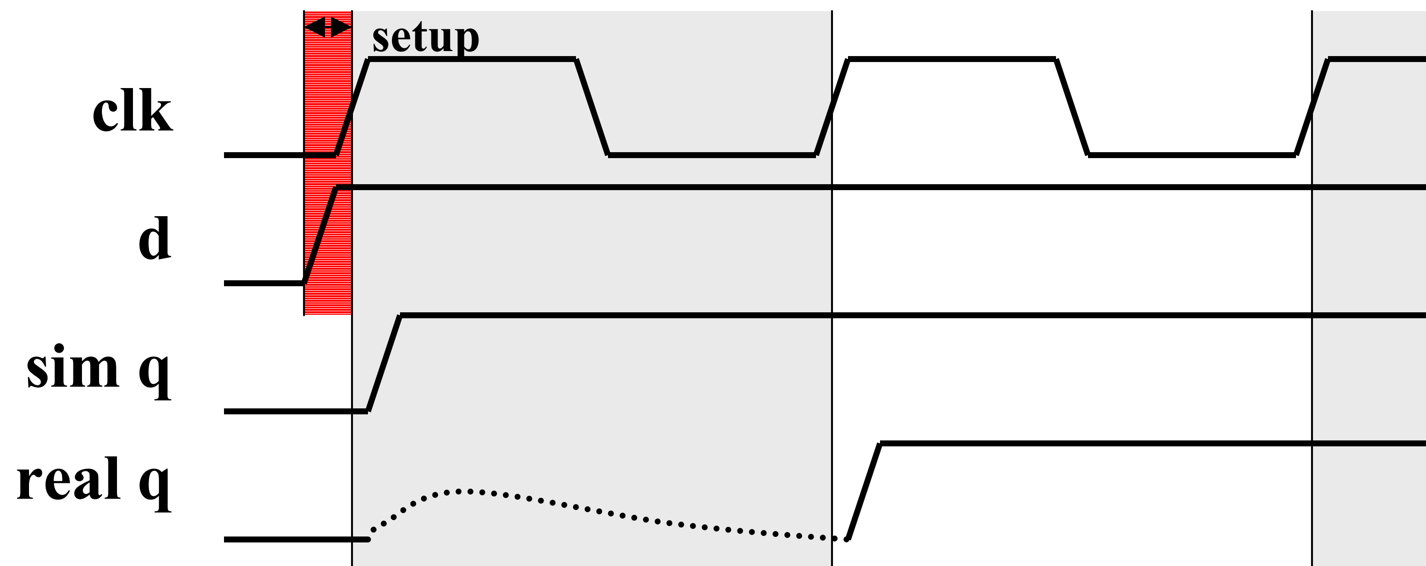


$$\text{MTBF} = \frac{e^{T_s/T_c}}{T_w F_c F_d}$$

T_s = settling window
T_c = settling time constant*
T_w = window of susceptibility*
F_c = destination clock frequency
F_d = data rate
*** technology dependant**

Metastability : Uncertainty

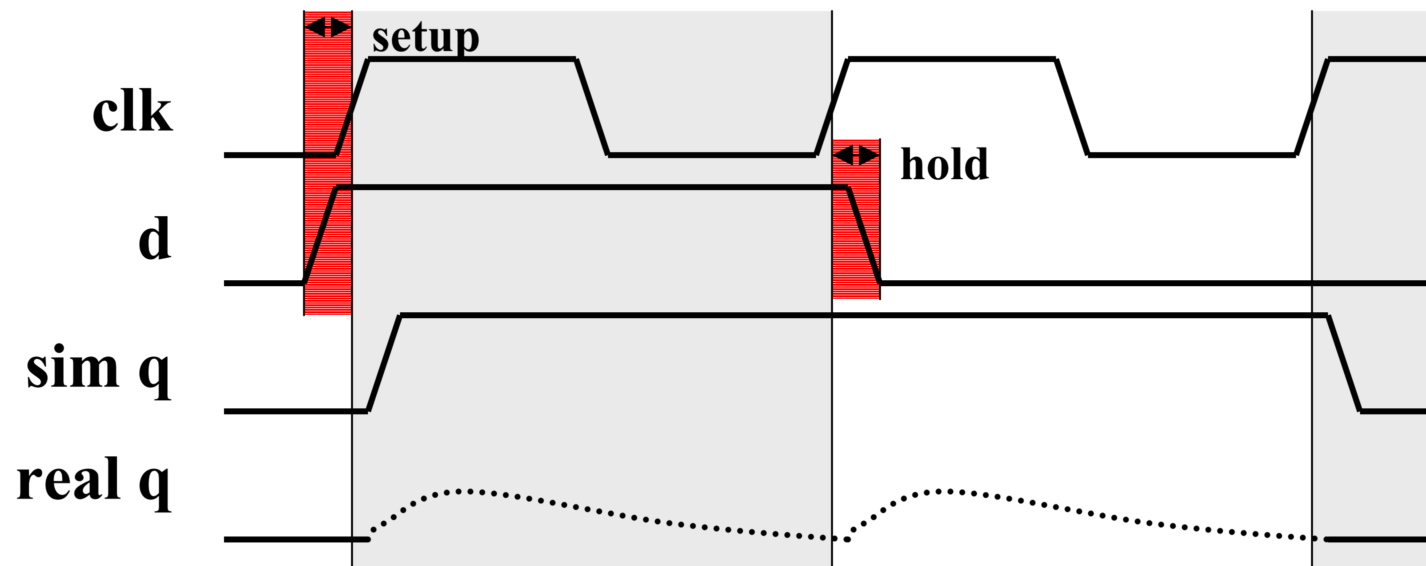
- Sampled value subject to uncertainty
  - sampled values that violate hold
    => next clock in simulation; 0 or 1 clock earlier in real-life
  - sampled values that violate setup (shown)
    => current clock in simulation; 0 or 1 clock later in real-life



Copyright © 2006 Verilab

Metastability : Filtering

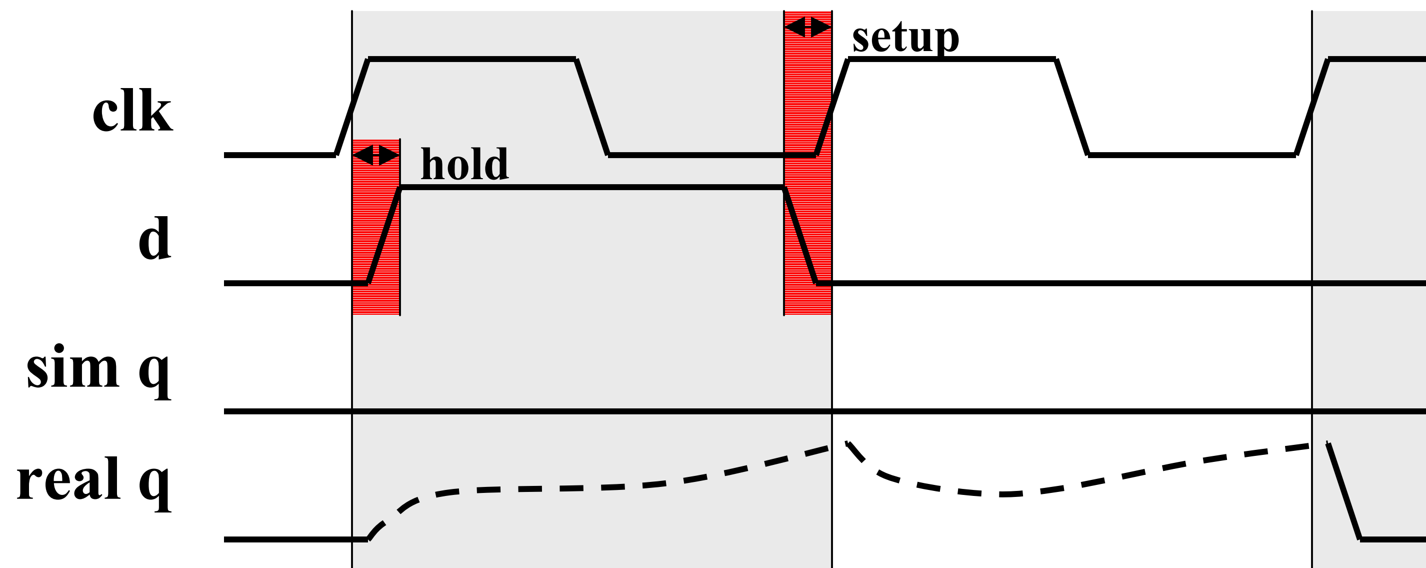
- Values propagated in simulation may get filtered in real-life
  - single clock samples that violate setup *or* hold
    => 1-clock pulse in simulation; 1 or 0 in real-life
  - two clock samples that violate setup *and* next hold (shown)
    => 2-clock pulse in simulation; 2, 1 or 0 in real-life



Copyright © 2006 Verilab

Metastability : Generation

- Values missed in simulation may get generated in real-life
  - unsampled values that violate hold *or* next setup
    => not seen in simulation; 0 or 1-clock pulse in real-life
  - unsampled values that violate hold *and* next setup (shown)
    => not seen in simulation; 0, 1 or 2-clock pulse in real-life

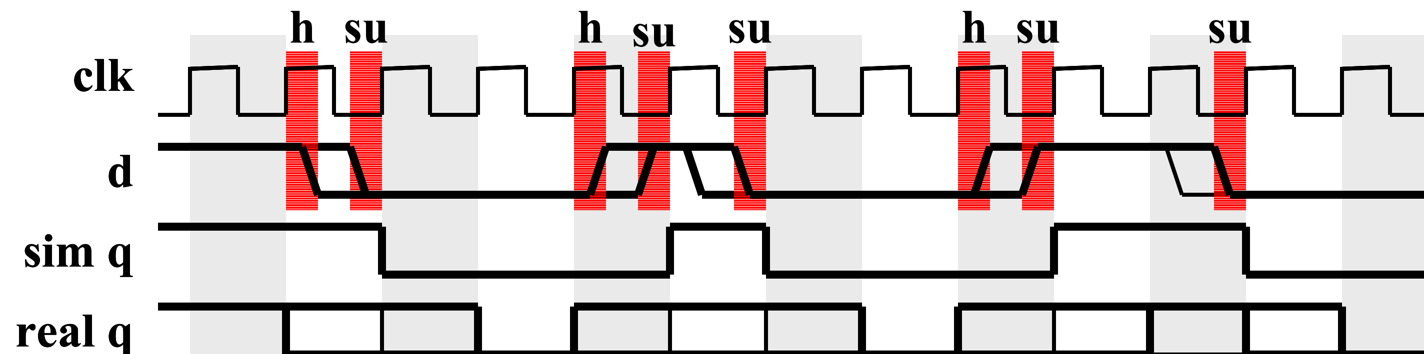


Metastability : CDC Jitter

- Combination of uncertainty, filtering and generation is called

  Clock Domain Crossing Jitter

- Affects all CDC signals with independent probabilities
  - including properly synchronized signals
- Real-world has unpredictable deviation from simulation
  - not visible in normal RTL or Gate-Level simulations
  - effect can be modeled for simulation or formal analysis



Copyright © 2006 Verilab

Assertion-Based Verification

- **Assertion-Based Verification** is a *methodology* for improving the *effectiveness* of a *verification* environment
  - define properties that specify expected behavior of design
  - check property assertions by simulation or formal analysis
- Benefits of ABV include:
  - improved error detection and reduced debug time due to observability
  - improved integration due to built-in self-checking
- ABV is a good fit for CDC verification:
  - low-level structural problem well encapsulated by ABV checks
  - SVA temporal expressions good for describing synchronizer protocols
  - CDC defects need to be identified at source to enable efficient debug
- ABV is only part of solution: require CDC methodology

Copyright © 2006 Verilab

CDC Verification Steps

1 Structural analysis to identify all CDC signals

2 Classification of CDC signal types

3 CDC signal verification
- operation of synchronizers
- usage of synchronizers

4 CDC jitter verification
- operation of complete device in presence of synchronizers

Copyright © 2006 Verilab

1. Structural Analysis

- Essential to identify all CDC signals
  - commercial automated CDC tools
  - script based in-house tools
  - manual inspection and code reviews
- Minimize risk by using structural CDC implementation
  - instantiate CDC building blocks; don't infer from distributed code
  - ensures metastable nets are not abused
  - encapsulate assertions as part of building blocks, not separate checkers
  - allows easier emulation of CDC jitter
- Repeat analysis for gate-level netlist and compare results
  - later in design cycle

Copyright © 2006 Verilab

2. Classification of Signals

- Classify CDC signals according to intent, for example:
  - regular signal (needing synchronized, e.g. interrupt)
  - level or pulse-based handshake protocol (groups of signals; some need synchronized, others need protocol checks only)
  - asynchronous FIFO (used when handshake latency is unacceptable)
  - static signal (e.g. only changed during configuration)
- All CDC signals should fall into an identified class
- All signal types can be checked using assertions
  - synchronizer operation and usage
  - associated handshake controller protocol
  - static checks during operation

Copyright © 2006 Verilab

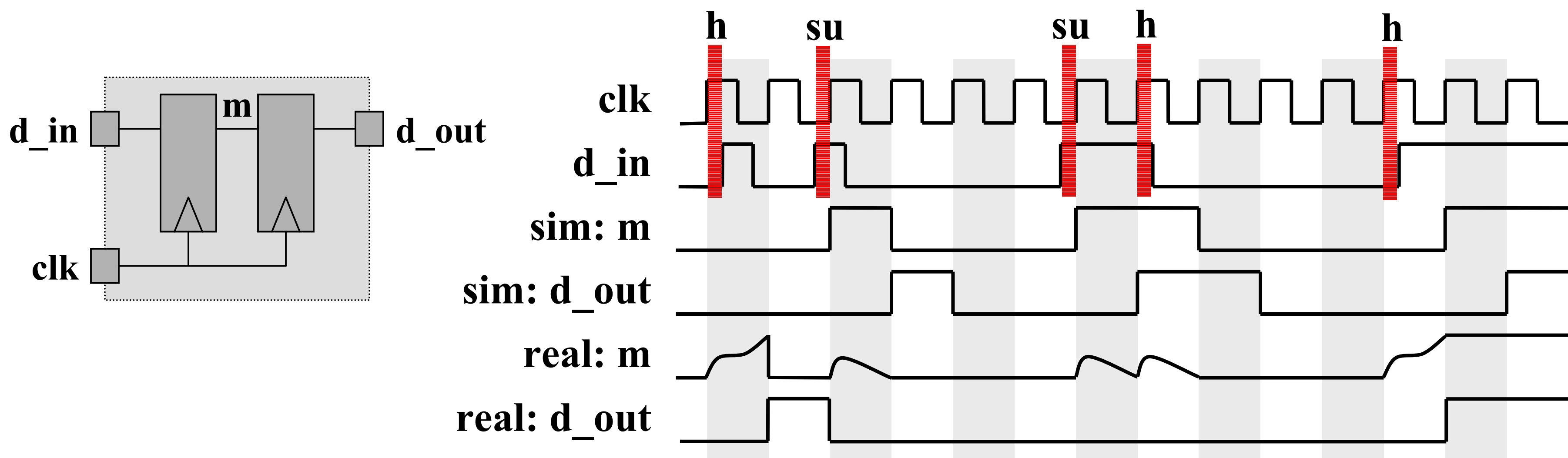
3. CDC Signal Verification

- Assert properties to check:
  - correct *operation* of synchronizer circuits
  - correct *use* of synchronizer circuits
- Encapsulate assertions in CDC building blocks
  - no assumed relationships between clocks
  - no manual specification parameters to specify relationships
  - clock relationships change dynamically and between derivatives
  - avoid manual instantiation of checkers and distributed code
- Validate assertions by simulation or formal analysis
  - assertions presented here are aimed at dynamic validation

Copyright © 2006 Verilab

2-Flip-Flop Synchronizer

- Sampled values must be stable for 3 destination clock edges
  - otherwise value may not be transported to destination clock domain
- Input changes must be sampled by destination clock
  - narrow values may get transported in real-life, but not in simulation
  - note: combinational logic not allowed on CDC paths



Copyright © 2006 Verilab

SVA for 2-FF Synchronizer

- *p_stability* check covers values sampled by clock
- *p_no_glitch* check validates transitions between clock edges
  - in this context *glitch* means any transition that is missed by destination clock domain

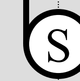
```
property p_stability;
  @(posedge clk)
    !$stable(d_in) |=> $stable(d_in) [*2];
endproperty

property p_no_glitch;
  logic data;
  @(d_in)
    (1, data = !d_in) |=>
  @(posedge clk)
    (d_in == data);
endproperty

assert property(p_stability);
assert property(p_no_glitch);
```

Copyright © 2006 Verilab

Timing Diagram for *p_no_glitch*

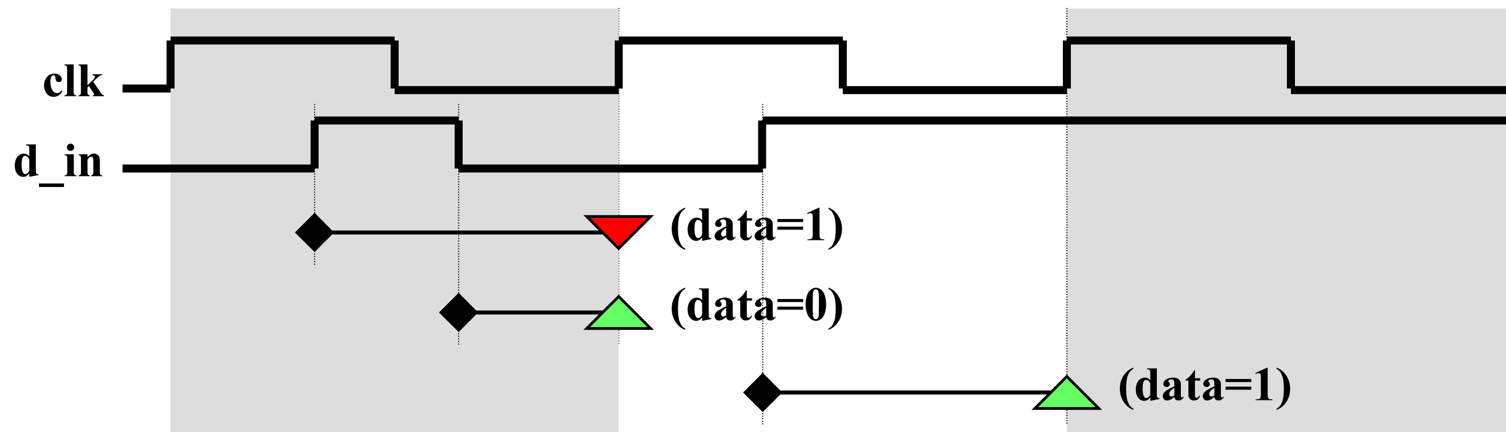
```
property p_no_glitch;
  logic data;
  @(d_in)
    (1, data = !d_in) |=>
  @(posedge clk)
    (d_in == data);
endproperty
```

d_in



@(posedge d_in) $sample(d_in) == 0

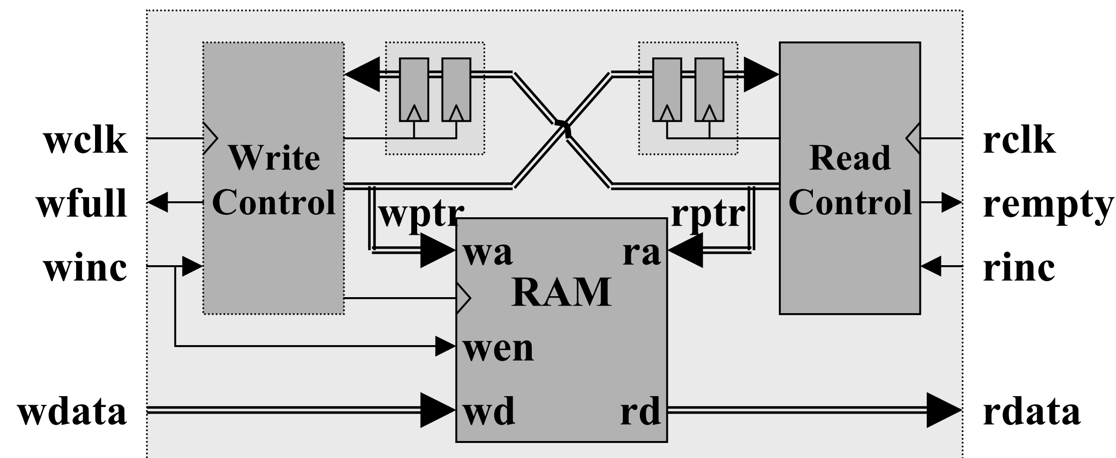
@(negedge d_in) $sample(d_in) == 1



Copyright © 2006 Verilab

Dual-Clock Asynchronous FIFO

- never write to a full FIFO
- never read from an empty FIFO
- no unknown control and status (anytime)
- no unknown data (during transfers)
- pointers gray-coded in source clock domain
- data integrity (correct values and order)



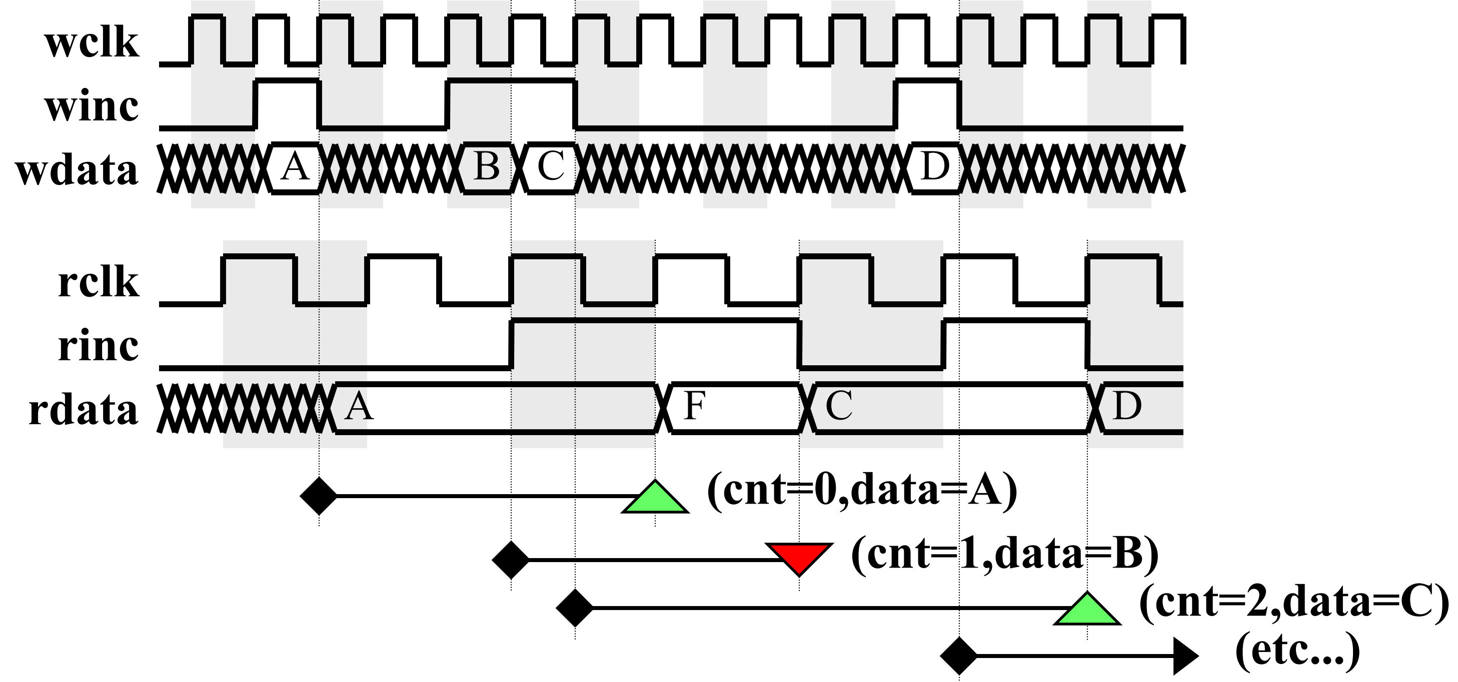
Copyright © 2006 Verilab

SVA for Data Integrity in Asynchronous FIFO

```
int wcnt, rcnt;
always @(posedge wclk) if (winc) wcnt = wcnt + 1;
always @(posedge rclk) if (rinc) rcnt = rcnt + 1;

property p_data_integrity;
  int cnt;
  logic [15:0] data;
  @(posedge wclk)
     (winc, cnt=wcnt, data=wdata) |=>
  @(posedge rclk)
     first_match(##[0:$] (rinc & (rcnt==cnt)))
     ##0 (rdata==data);
endproperty

assert property (p_data_integrity);
```

Copyright © 2006 Verilab

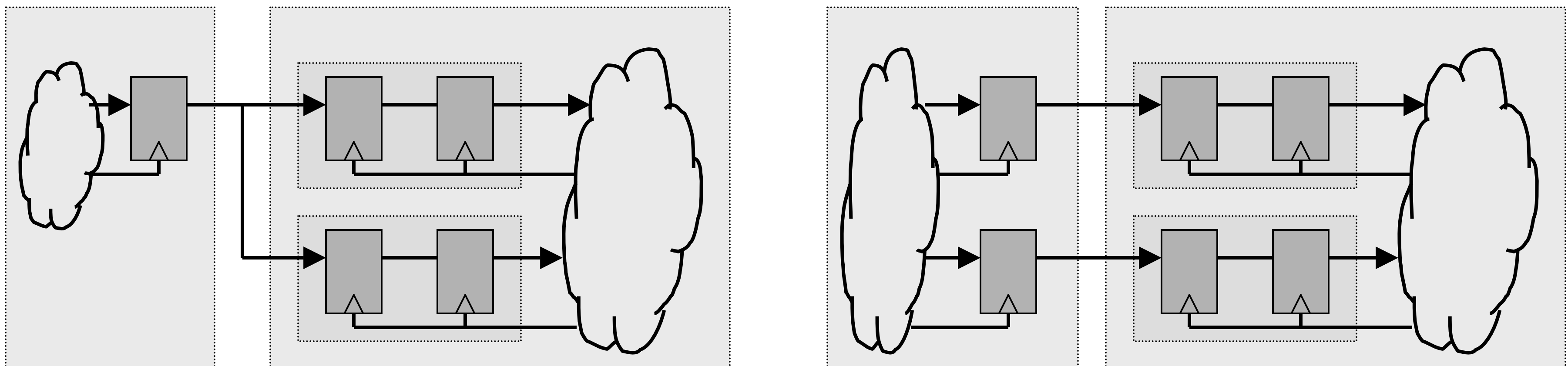
Timing Diagram for *p_data_integrity*



Copyright © 2006 Verilab

CDC Convergence

- Convergence of CDC signals can cause unreliable operation of synchronizer or downstream circuits
  - even with correctly synchronized CDC signals
  - independent CDC jitter causes unpredictable temporal relationships
  - logic must be coded to allow for all possible jitter timing relationships
  - convergence can be local (e.g. inputs to FSM) or remote (e.g. many sequential elements in destination domain before convergence)



Copyright © 2006 Verilab

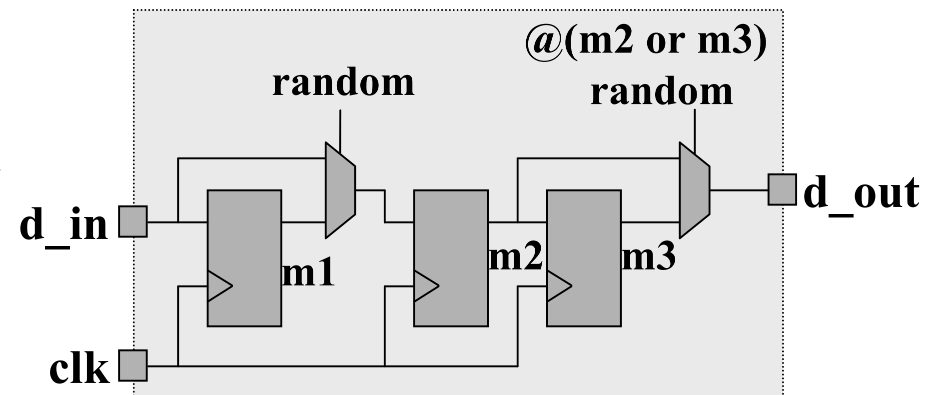
4. CDC Jitter Verification

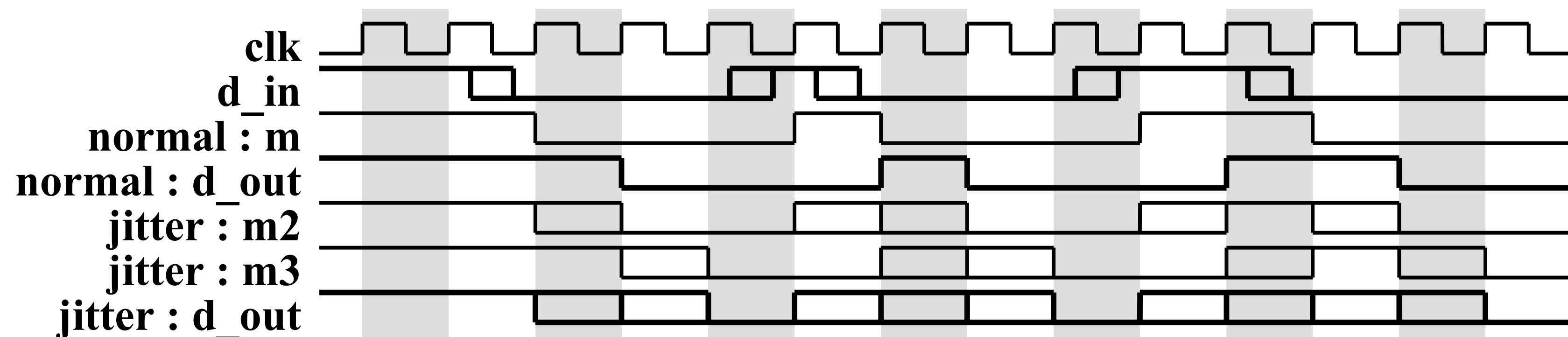
- Identify convergent CDC signals by **structural analysis**
- Assert *additional properties* in areas of convergence
- Emulate CDC jitter using model or formal tool
- Validate assertions by simulation or formal analysis
  - checks properties in *rest of design*, not just synchronizers
  - need reasonable assertion density in areas of convergence
  - formal is particularly effective for jitter convergence
    => since it can prove all possible jitter combinations
  - simulation only observes actual generated scenarios
    => add functional coverage points to improve effectiveness

Copyright © 2006 Verilab

CDC Jitter Model

```
always @(posedge clk) begin
  randcase
    1: m2 <= $past(d_in);
    1: m2 <= d_in;
  endcase
  m3 <= m2;
end
always @(m2 or m3)
  randcase
    1: d_out = m2;
    1: d_out = m3;
  endcase
```

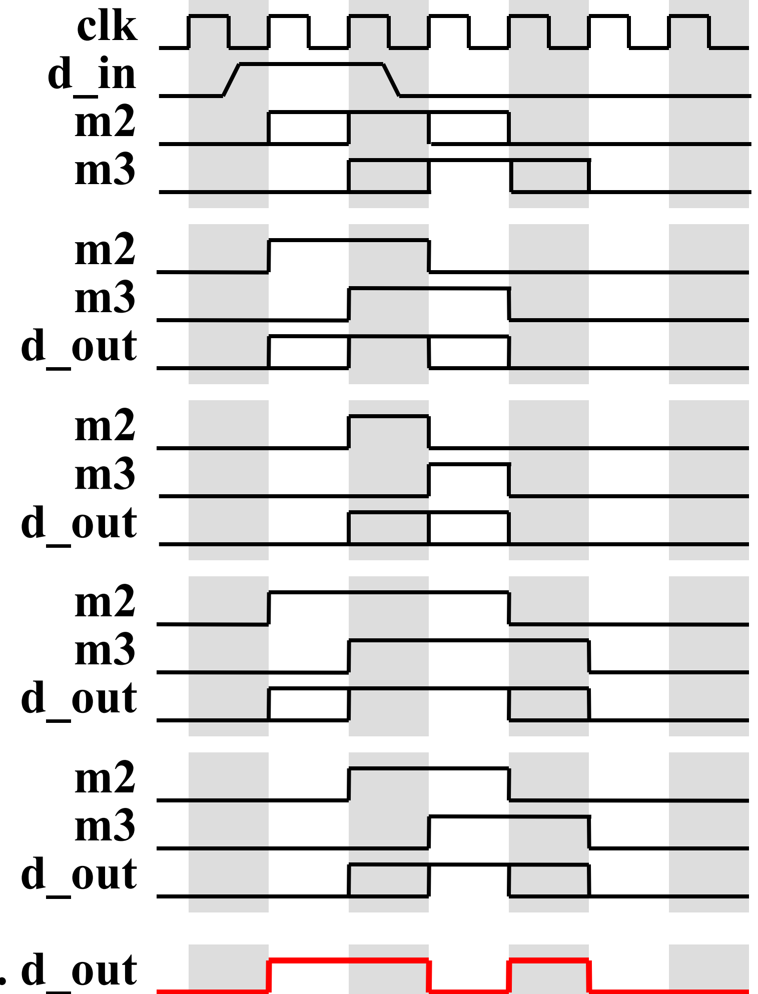




Timing Diagram for CDC Jitter Model

```
always @(posedge clk) begin
  randcase
    1: m2 <= $past(d_in);
    1: m2 <= d_in;
  endcase
  m3 <= m2;
end
```

```
always @(m2 or m3)
  randcase
    1: d_out = m2;
    1: d_out = m3;
  endcase
```



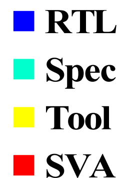
illegal conditions are never generated; e.g. d_out

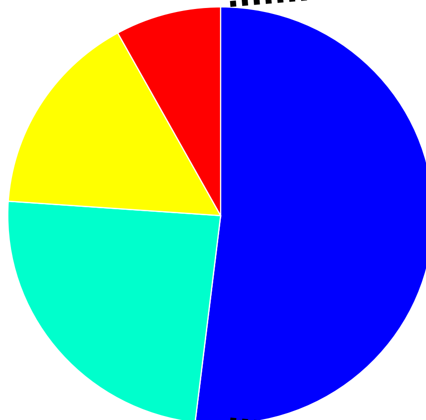
Copyright © 2006 Verilab

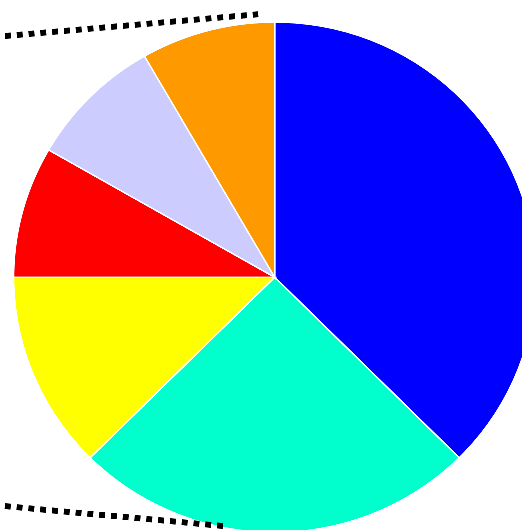
Example Project Results

- Hardware graphics accelerator for mobile applications
  - mature VHDL design with modifications and derivatives planned
  - established VHDL testbench environment including FPGA prototype
  - ABV introduced late in design cycle to subsystem level
  - many CDC defects were detected by ABV using simulation

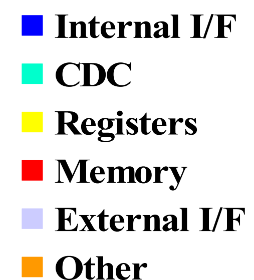
Issue Type







RTL Defect Type



Copyright © 2006 Verilab

Conclusion

- Practical assertion-based methodology for simulation flows
  - accessible to most FPGA and ASIC teams: no special tools
  - successfully applied to a number of different projects
- Structural approach to CDC building block implementation
  - minimizes risk of incorrect or abused synchronizers
  - few important properties on key CDC blocks reused as required
  - no assumptions about clock relationships - tougher design rules
- Special assertions improve quality of dynamic validation
  - static signal checks, glitch detection, multi-clock assertions
- Validate device operation in presence of CDC jitter
  - identify convergence, increase assertion density, emulate jitter effects, add functional coverage points

Copyright © 2006 Verilab mark.litterick@verilab.com