Parameterized and Re-usable Jitter Model for Serial and Parallel Interfaces

Malathi Chikkanna

Amlan Chakrabarti

AMD

Agenda

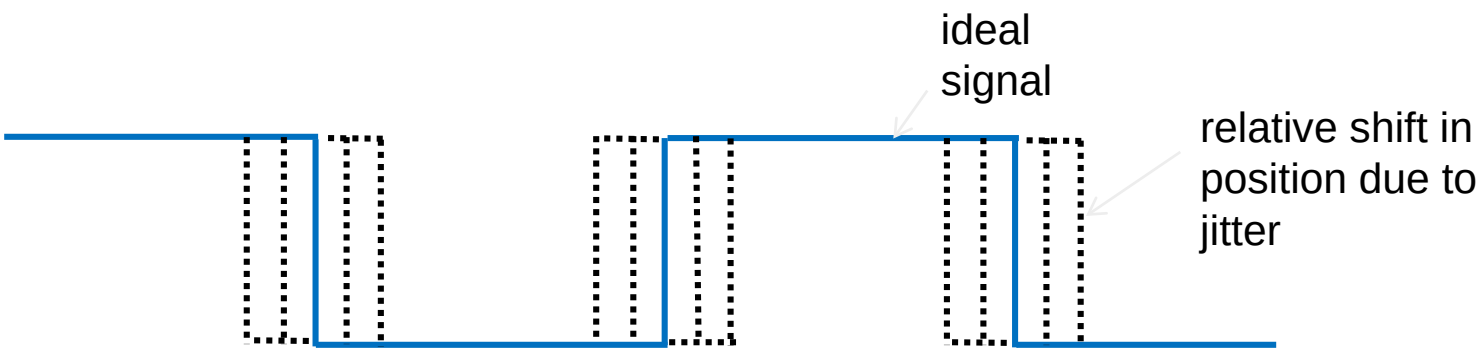
- Introduction
- Past approach to model jitter
- Proposed approach for modeling jitter
- Results
- Conclusion
- Acknowledgements
- Q & A

JITTER AND MODELING JITTER

INTRODUCTION

Jitter

- Jitter
  - Short term variation of a signal with respect to its ideal position in time
  - Sources of jitter – PLL, random thermal noise from a crystal, coaxial cables, traces, cross-talk, power supply noise etc.



Modeling Jitter on Serial/Parallel Interfaces

- Jitter model modifies the width of a data bit/data strobe.
- Serial interface
  - Clock may not be sent along with the data.
  - Clock recovered from the data using clock-data recovery (CDR).
  - Variation in the width of the data bit can expose inadequate depth of an elastic buffer.
- Parallel interface
  - Different bits of a data bus can experience varying amounts of jitter.
  - The distributions in the jitter model mimic this.

Types of Jitter

- Deterministic (sinusoidal, triangular, etc.)
  - Modeling of sinusoidal jitter

```
sj_offset + (sj_ampl*$sin(2*3.1416*curr_sj_freq*$realtime));
```

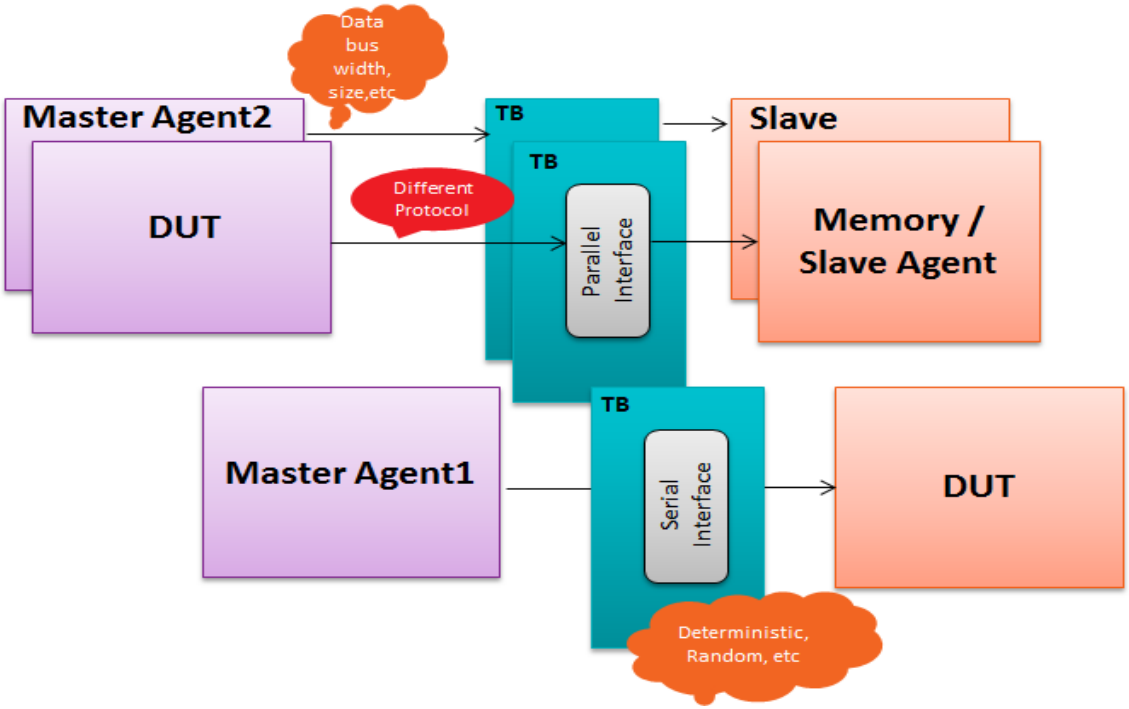
- Random (Gaussian, etc.)
  - rj_offset + $dist_normal(rj_seed, 0, rj_stdev)*10/1000;
- Combination of deterministic and random

NON-REUSABLE

PAST APPROACH TO MODEL JITTER

Past Approach

- Separate jitter model for every new interface

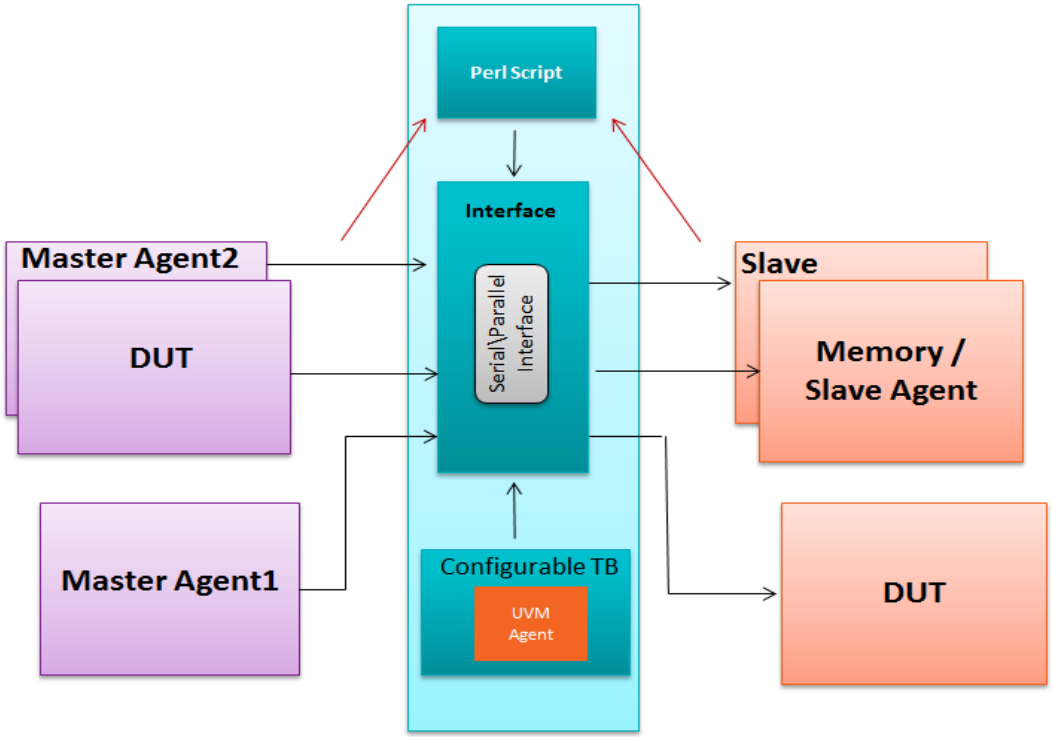


REUSABLE, CONFIGURABLE AND SCALABLE

PROPOSED SOLUTION

Proposed Solution

- Parameterized and reusable jitter model using UVM
- Can be used for any serial/parallel interface



Proposed Solution (Contd.)

- Jitter model parameters
  - Interface parameters
    - data bus port names, bus width
  - Testbench parameters
    - Type of jitter – sinusoidal, triangular, gaussian
- Components of the solution
  - Script
    - Input (interface parameters), Output (interface file)
  - Jitter Agent (in UVM)
    - Configurable and reusable

Script for Generating Interface File

- Script

```
my %CMDVars= ();
$CMDVars{interface_name} = "";
$CMDVars{data_sig_name}  = "";
$CMDVars{data_width}     = "";
$CMDVars{data_filename}      = "";
…

GetOptions( “interface_name=s"  =>\$CMDVars{interface_name},
        “data_sig_name=s “       =>\$CMDVars{data_sig_name},
        “data_width=s"           =>\$CMDVars{data_width},
        “data_filename=s"        =>\$CMDVars{data_filename},
         …
        "help"                   =>\$CMDVars{help} );
```

data.txt

```
Input = pad_dq_in, output = pad_dq_out, sig_width = 32
Input = pad_addr_in, output = pad_addr_out, sig_width = 10
….
```

script

```
open (DATA_FILE, “$data_filename.txt") or die ("file $ data_filename.txt file does not exist;");
open (INTF_FILE, ">$interface_name.sv") or die ("file $nterface_name.sv file does not exist;");
my @array = <DATA_FILE>;
print INTF_FILE "interface $interface_name\(input bit clk\);\n";
foreach (@array)
{
   my $data_out = $_ -> {“output"};
   my $data_out = $_ -> {“input"};
   my $size = $_ -> {“sig_width"};
   print INTF_FILE "  logic $data_out[$size:0]\;\n";
    …}
print INTF_FILE "endinterface \n“;
```

Interface File

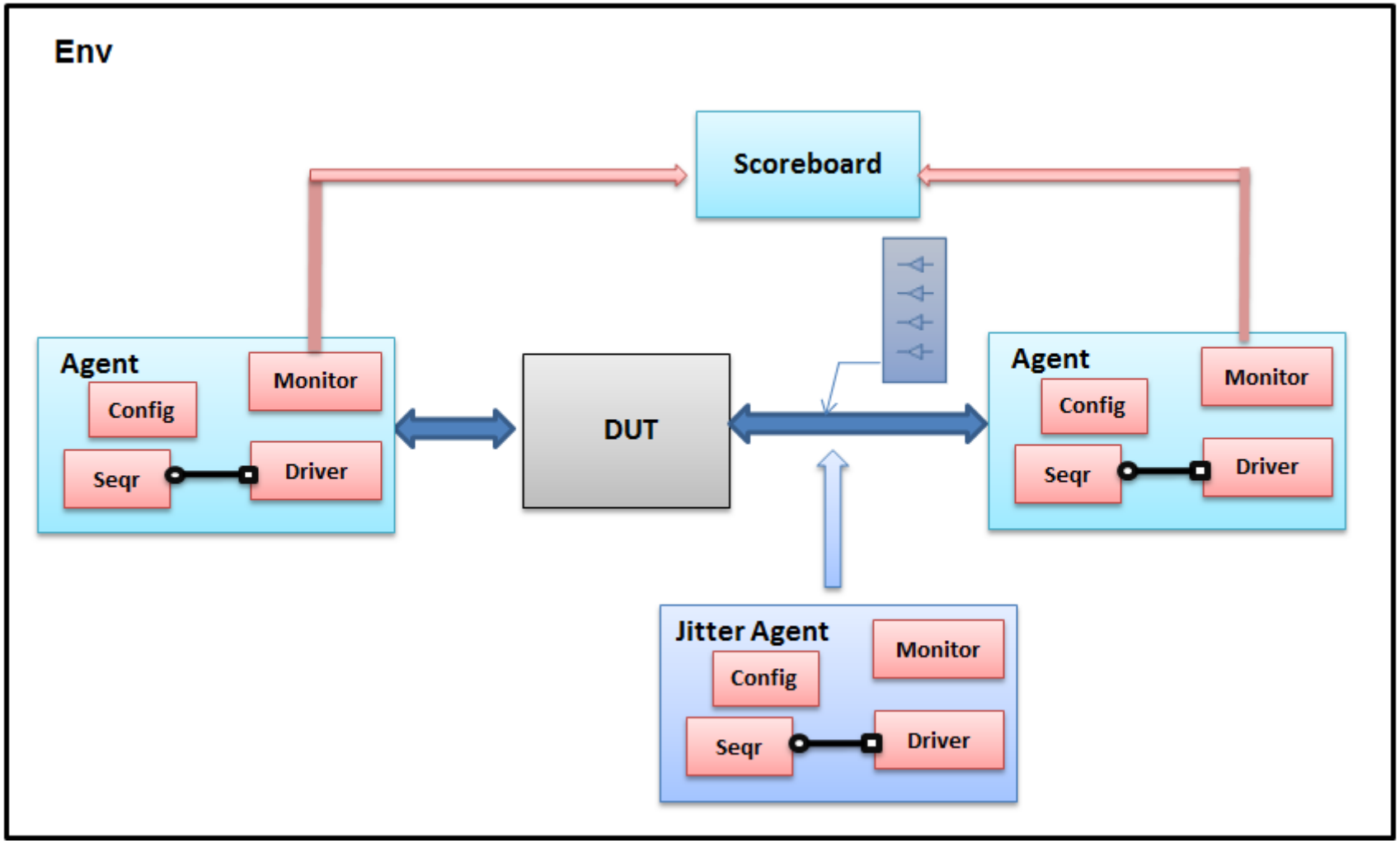
- Output interface file

```
Interface  pad_intf (input bit clk);
  logic pad_dq_out[31:0];
  wire pad_dq_in[31:0];

`ifdef POSEDGE_ONLY
always @(posedge pad_dq_in) begin
  pad_dq_out[0] = #pad_delay[0] pad_dq_in[0];
  pad_dq_out[1] = #pad_delay[1] pad_dq_in[1];
  …
  pad_dq_out[31] = #pad_delay[31] pad_dq_in[31];
end
`else
always @(pad_dq_in) begin
  pad_dq_out[0] = #pad_delay[0] pad_dq_in[0];
  pad_dq_out[1] = #pad_delay[1] pad_dq_in[1];
  …
  pad_dq_out[31] = #pad_delay[31] pad_dq_in[31];
end
`endif

endinterface
```

UVM TB Environment



Jitter Agent - Configuration Object

- Testbench parameters

```
class jitter_cfg extends uvm_object;

 //Parameters
   rand int rj_offset;
   rand int rj_enable;
   rand int rj_seed[32];
   rand int rj_rms;
    rand int no_rj;
    rand int sin_enable , sin_offset;
    real ampl;
    real curr_sj_freq;
   // 0 - 10 Khz, 1 - 100 Khz, 2 - 1 Mhz
    rand bit[1:0] sj_freq = 2;
   rand int max_delay, min_delay;
   rand delay_incr;
   rand int trij_offset;
    ....

function void post_randomize();
     if(sj_freq == 0)
       curr_sj_freq = 1e4/1e12; // 10 khz
     else if(sj_freq == 1)
       curr_sj_freq = 1e5/1e12; // 100 khz
     else if(sj_freq == 2)
       curr_sj_freq = 1e6/1e12; // 1 Mhz
  endfunction
```

Gaussian jitter parameters

Sinusoidal jitter parameters

Triangular jitter parameters

Config object

```
class jitter_trn extends uvm_sequence_item;

 //Parameters
rand int rj_enable;
rand int sj_enable;
rand int_rj_sj_enable;
....
endclass
```

Sequence item

Jitter Agent- Driver

Driver

```
class jitter_driver extends uvm_driver #(jitter_trn);
virtual pad_intf pad_if;
jitter_cfg cfgh;
real rj_array[];
real sin_jitter;
virtual task random_jitter();
foreach(rj_array[i]) begin // Random Jitter( Gaussian)
      rj_array[i] = cfgh.rj_offset + $dist_normal(cfgh.rj_seed[i], 0, cfgh.rj_rms)*10/1000;
  end
endtask
virtual task sin_jitter();
   sin_jitter = cfgh.sj_offset + (cfgh.ampl*$sin(m_2PI*cfgh.curr_sj_freq*$realtime));
endtask
```

Gaussian jitter

Sinusoidal jitter

```
class derived_jitter_driver extends jitter_driver;
…
task run_phase(uvm_phase phase);
 seq_item_port.get_next_item(tr);
 If(tr.rj_enable) begin
   rj_array = new[jitter_cfg.no_rj];
    random_jitter ();
    foreach(rj_array[i]) begin
        pad_if.pad_delay[i] = rj_array[j];
   end
```

1 or 2

Serial interface

```
class derived_jitter_driver extends jitter_driver;
…
task run_phase(uvm_phase phase);
 seq_item_port.get_next_item(tr);
@(pad_if.cb);
 If(tr.rj_enable) begin
  rj_array = new[jitter_cfg.no_rj];
   random_jitter ();
   foreach(rj_array[i]) begin
        pad_if.pad_delay[i] = rj_array[j];
   end
….
```

> 2

Parallel interface

Disabling the Jitter Agent

- Jitter model can be disabled through configuration object
  - Dynamic

tbcfg

```
class tb_cfg extends uvm_object;
 jitter_cfg jit_cfgh;
 rand int  jitter_model_enable;
// serial interface parameters
 rand int serial_if_enable;
  …
// parallel  interface parameters
 rand int parallel_if_enable;
 …
endfunction
```

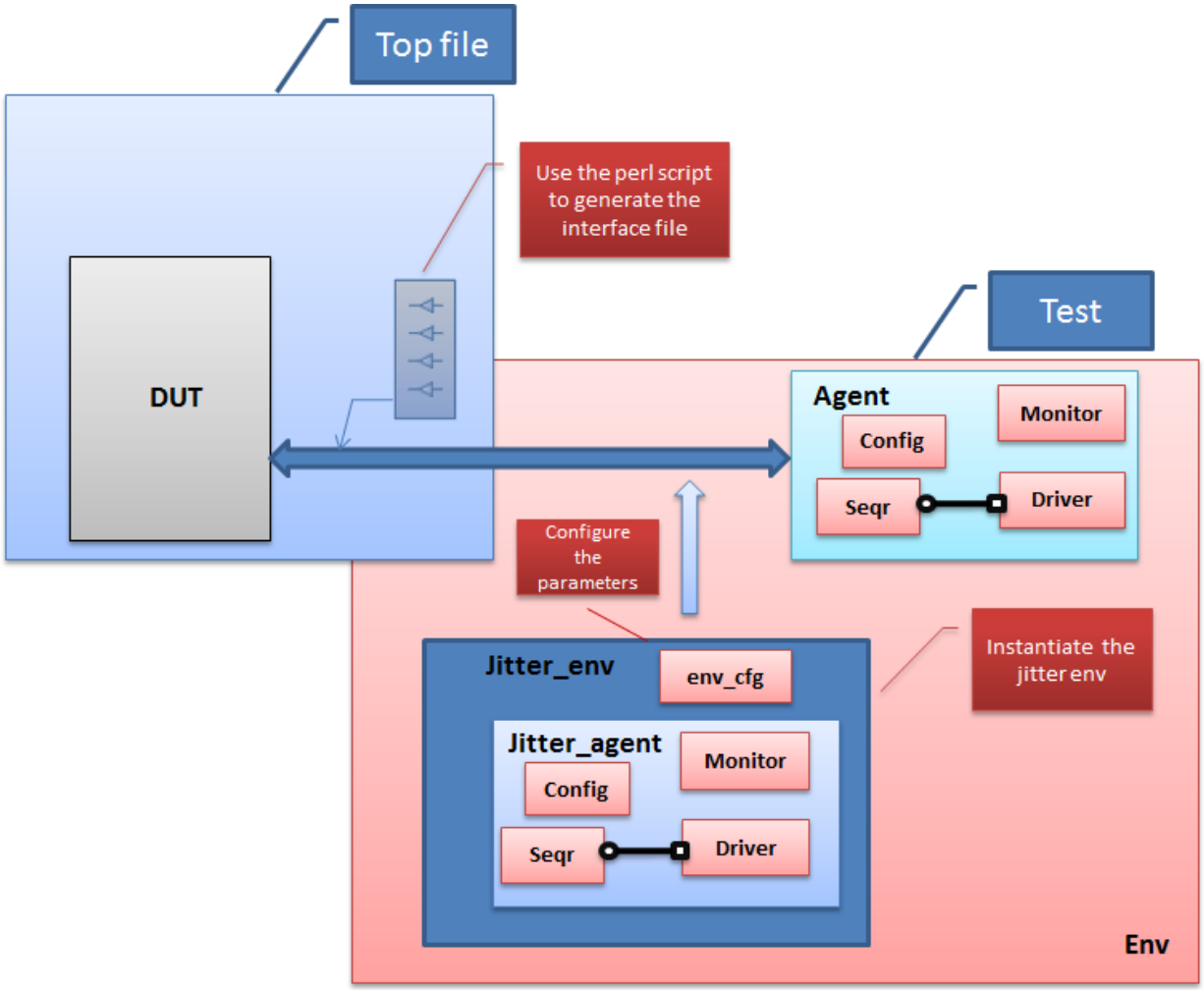
env

```
class jitter_env extends uvm_env;
jitter_agent       jitter_agent_i[*];
tb_cfg tb_cfgh;

function build_phase(uvm_phase phase);
If(tb_cfgh.jitter_model_enable) begin
      set_type_override_by_type(jitter_driver::get_type(), derived_jitter_driver::get_type());
     for (int chn=0; chn<tb_cfgh.num_of_chn; chn++)
      jitter_agent_i[chn] = jitter_agent::type_id::create($sformatf("jitter_agent_i[%0d]",chn),this);
end
endfunction
```

Jitter model enable

Reusing the Jitter Model for a New Interface

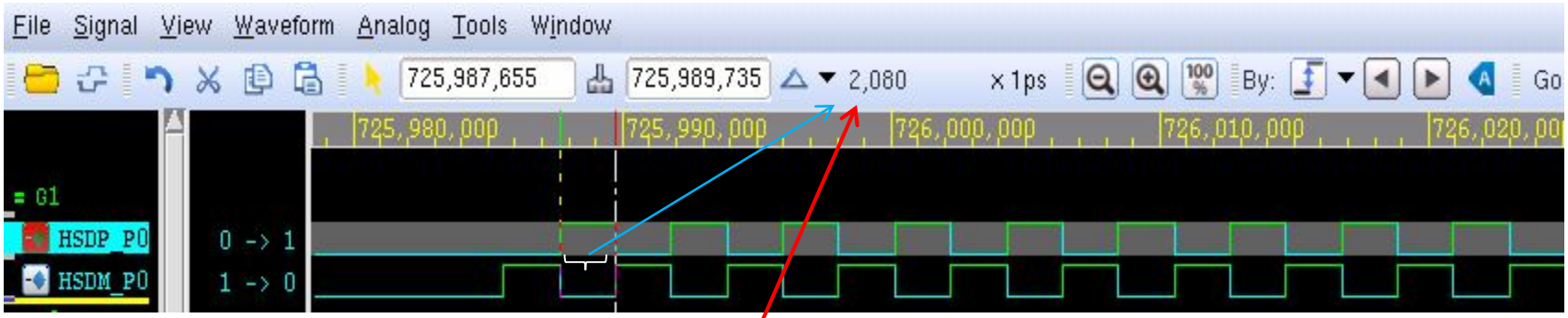


SERIAL AND PARALLEL INTERFACES

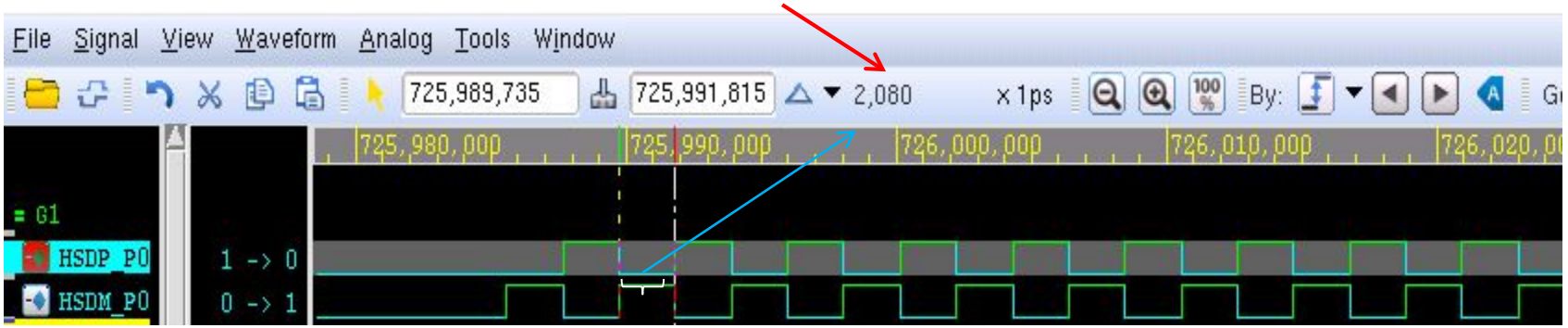
RESULTS

Waveforms for a Serial Interface

- Without jitter introduced

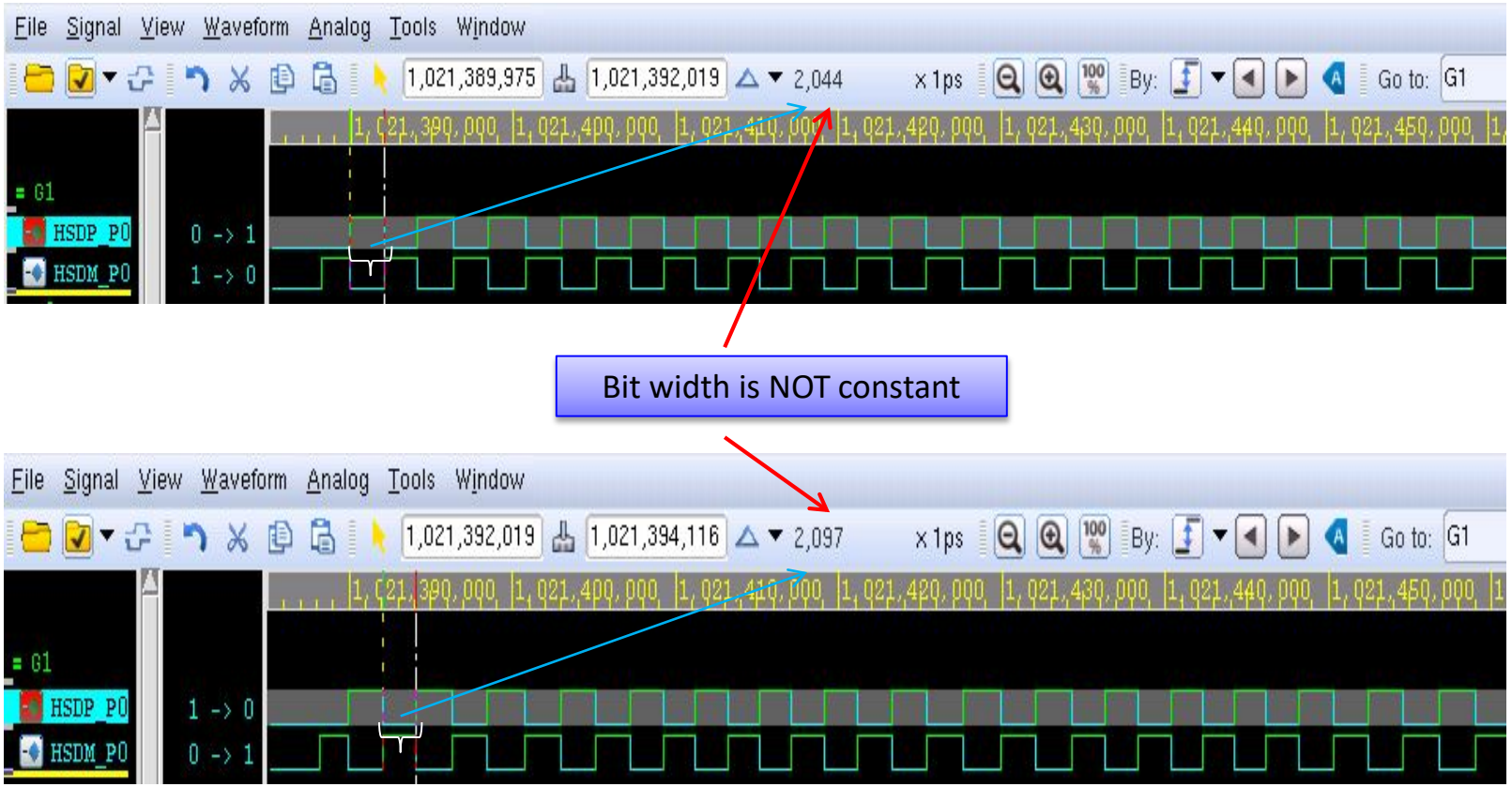


Bit width is constant

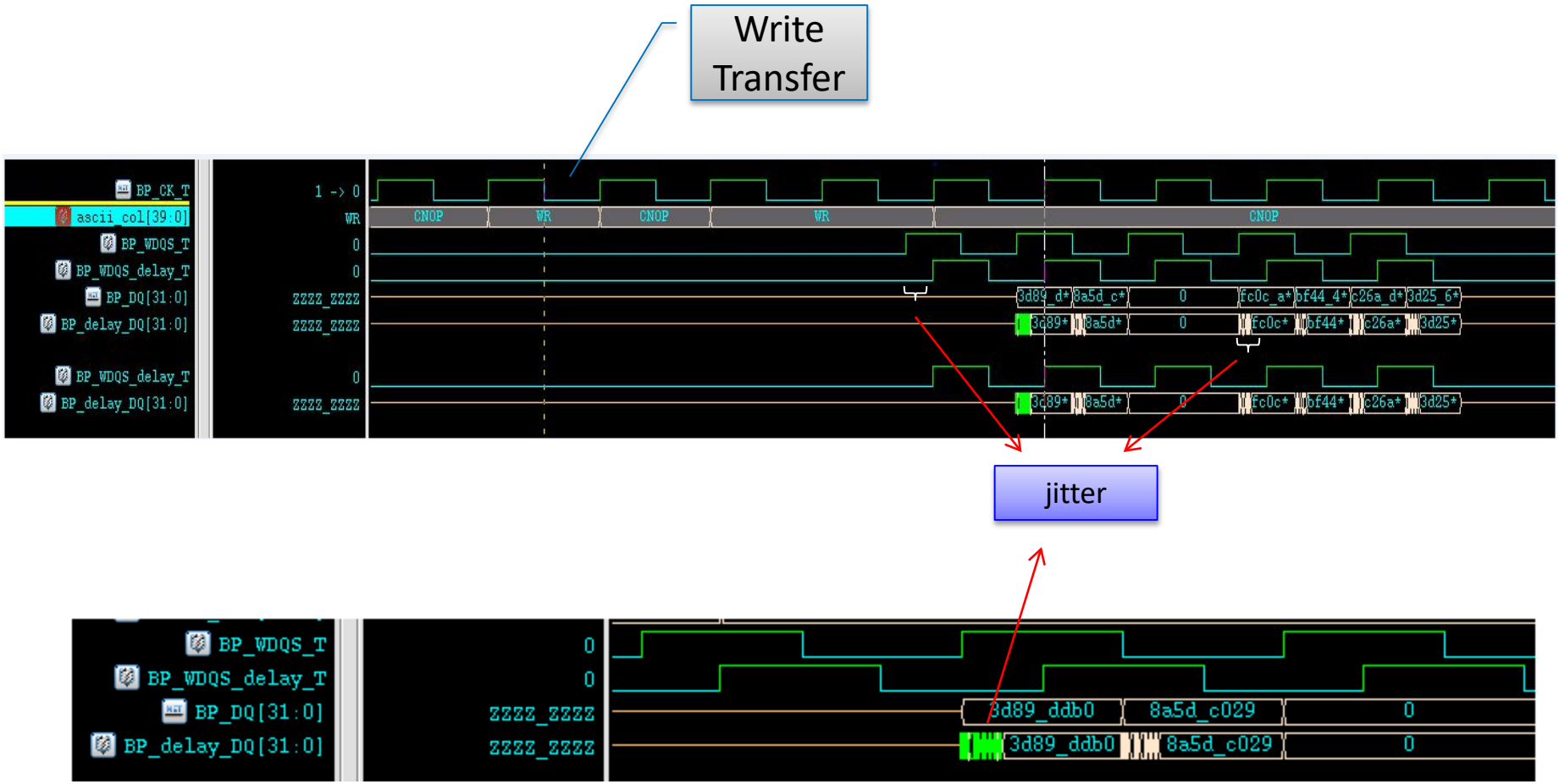


Waveforms for a Serial Interface (contd.)

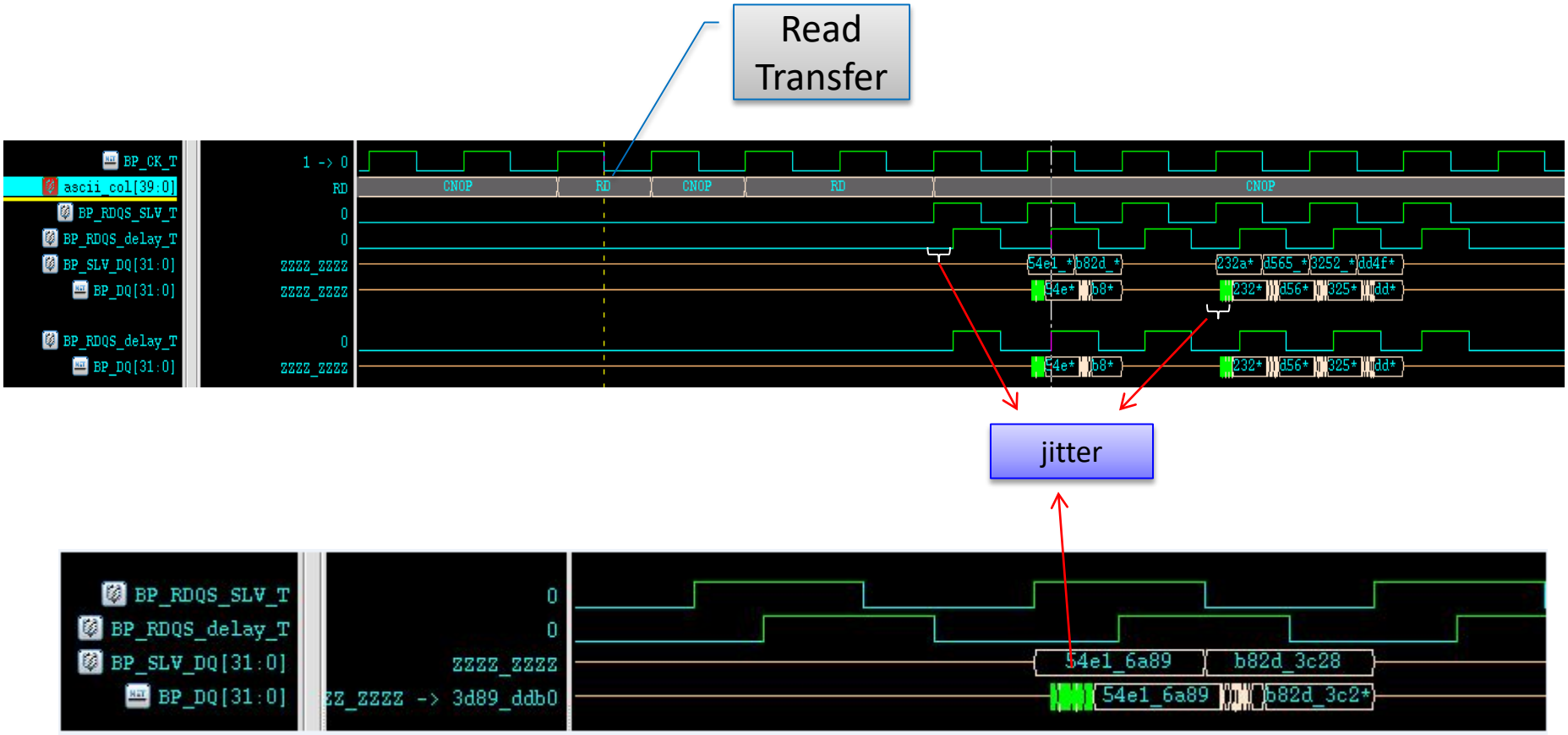
- With jitter introduced



Parallel Interface Write Transfer



Parallel Interface Read Transfer



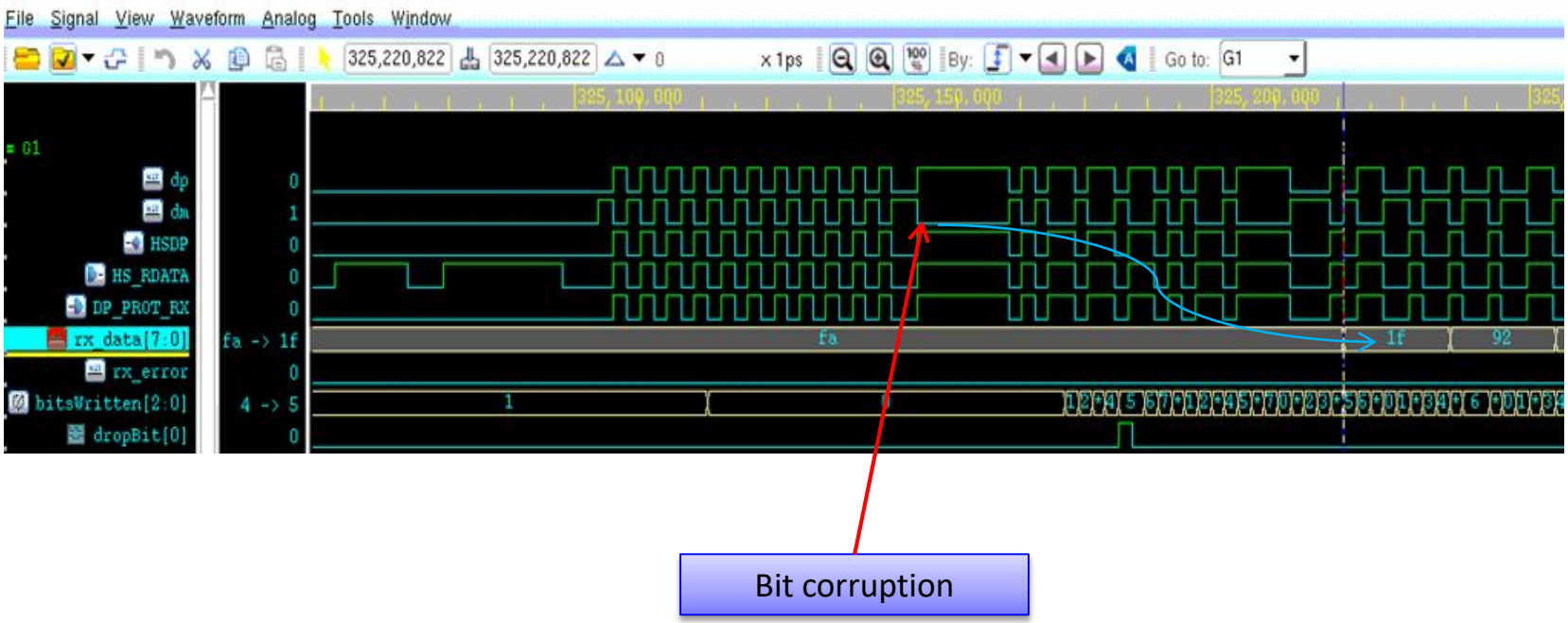
Bugs

- Bugs found on a serial interface

Category	Bug description
Received data corrupted	Final bit of a HS sync pattern is received in the first bit of a UI where 2 bits are processed. If this happens, the 2nd bit doesn't get written into the elastic buffer, but is discarded.
False assertion of receive error	Noise at the end of a HS chirp response is interpreted as a sync pattern. This falsely asserts receive error to the controller.

Waveform - Bug

- Waveform for the received data corruption



CONCLUSIONS

Conclusions

- A separate jitter model for each new interface lacks reusability
- With proposed approach, any new interface can be generated
- Jitter model can be quickly integrated in the environment
- Dynamic in nature
- **Saved 2 weeks** worth of effort when developing a jitter model from scratch for a new interface.

Acknowledgments

- We would like to acknowledge our colleagues for their valuable suggestions during this effort
  - Toni Simov
  - Atanaska Yanachkova
  - Govinda Raju

Thank You!

Questions