Holistic Approach to IO Timing Verification Using Portable Stimulus and Assertions

Amitesh Khandelwal and Praveen Kumar
Infineon Semiconductor Asia Pac Pte Ltd,
8 Kallang Sector, Singapore. 349282

***Abstract*-This paper gives an overview of how to verify IO timings of a complex SoC with auto generated Assertions using the concept of portable stimulus.**

I. INTRODUCTION

One of the major challenges for Static Timing Analysis (STA) is to guarantee the IO timing parameters specified in the datasheet for each and every supported variation (in terms of frequency and load). A farrago of different peripherals with multiple modes creates myriad scenarios that are too difficult and error prone to be done accurately for a static tool therefore augmentation is required by alternate methodologies.

We used SystemVerilog assertions that are run in the dynamic simulations to complement STA in verifying IO timings for IPs like Gigabit Ethermac, SPI, MSC and SDMMC etc. Please refer to [2] for details on these IPs. Since these IPs have several timing parameters for each pin-clock combination and each pin can appear on several top level ports depending upon muxing options used, the total number of assertions required to verify all the timing parameters can easily run into hundreds. This challenge is overcome by using Perspec[tm] [3] to generate these assertions automatically by modelling basic assertion in SLN.

'SLN' which stand for System-Level Notation is a proprietary language from Cadence[tm]. The Perspec[tm] tool documentation also covers the details of this language. It is similar to the 'e' language in terms of syntax so the adaptation is faster if you are already familiar with 'e'.

Basic SystemVerilog assertions are modelled in SLN using the Perspec[tm] tool to create a template. All the desired timing parameters are given to the tool in csv format. Perspec[tm] then applies the template to all the pin-clock combinations and generates assertions. These assertions are then added to the top level testbench and run along with dynamic simulation. Any deviation is reported in the logs.

II. IO timing parameters specified in the datasheet

Different peripherals have different IO timings. However they can be classified in the following three categories in general.

a) Clock Duty Cycle Deviation
b) Output Delay
c) Setup and Hold

Below figure 1 and 2 provide examples of such IO timings for MSC (Micro Second Channel) module. The MSC is a serial interface that is especially designed to connect external power devices. The serial data transmission capability minimizes the number of pins required to connect such external power devices.

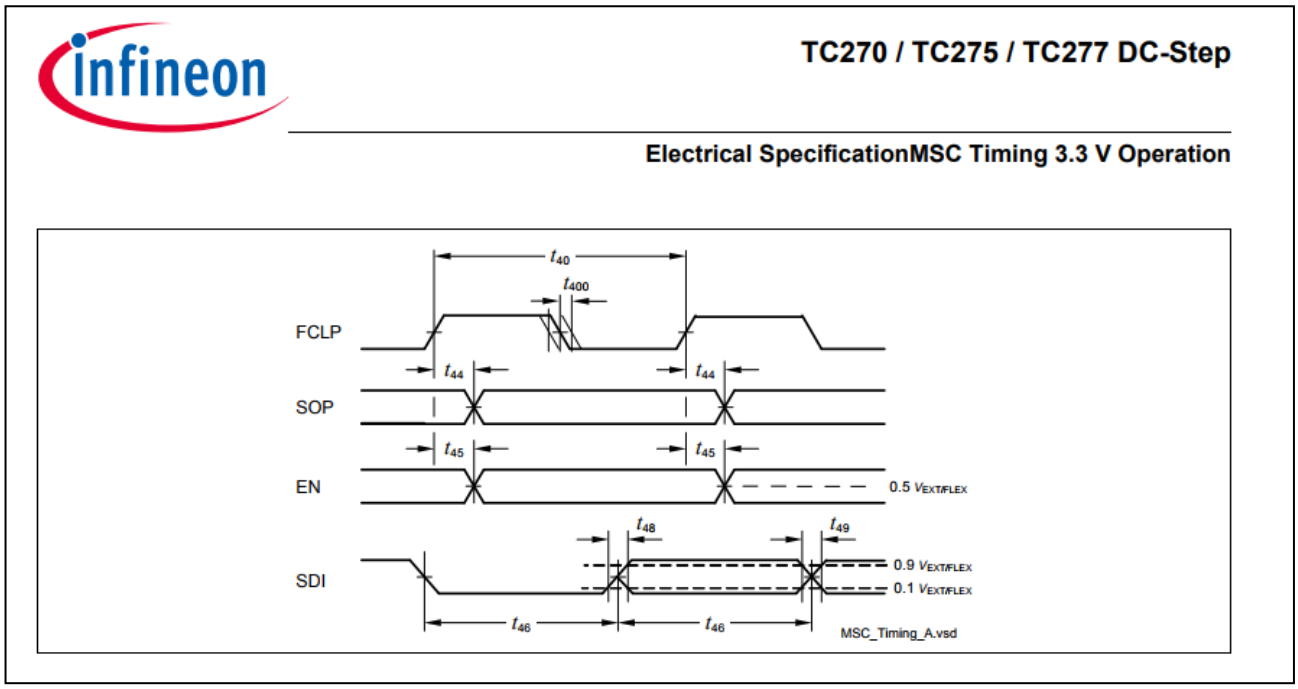


Figure 1 : IO Timing diagram of MSC module (Refer to 2 for module details).

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
FCLPx clock period [1]	t_{40} CC	2 * T_A	-	-	ns	MPm/MP+m/MPRm; C_L=50pF
Deviation from ideal duty cycle [2] [3]	t_{400} CC	-8	-	15+0.04 * C_L	ns	MPm/MP+m; 0 < C_L < 200pF
SOPx output delay [4]	t_{44} CC	-11	-	9	ns	MPm/MP+m; C_L=50pF
ENx output delay [4]	t_{45} CC	-15	-	11	ns	MPm/MP+m/MPRm; C_L=50pF
		-33	-	-4	ns	MPm/MP+m/MPRm; C_L=0pF

Figure 2 : IO Timing parameters of MSC module (Refer to 2 for module details).

Typically the parameters are specified in the datasheet. Datasheet also contains the expected values which we need to verify.
Sometimes values may not be present in the datasheet (for example for a new lead product) and in such cases STA report itself can be used as a reference.

III. Extraction of parameter values from datasheet

If parameters are not available in the datasheet, we can use the values dumped from the STA tool. Below figure 3 shows an example of the MSC timing report. Using a Perl script this report was converted into CSV format as shown in figure 4.

```
================================================================================
Parameter                                                Symbol      Min     Max
--------------------------------------------------------------------------------
Clock duty cycle deviation, CMOS mode                    t400      -3.24    2.79
SO   output delay, CMOS mode                             t44       -2.94    3.34
EN   output delay, CMOS mode                             t45       -2.98    3.10

================================================================================
Detailed Report:
================================================================================
--------------------------------------------------------------------------------
Interface 0:
--------------------------------------------------------------------------------

Clock duty cycle deviation at p11_6:PPFAST.msc0_abra_fclpb0, CMOS mode                t400      -2.64    2.25
Clock duty cycle deviation at p11_6:PPFAST.msc0_fclpb0, CMOS mode                     t400      -2.62    2.27
Clock duty cycle deviation at p13_0:LVDSTX.msc0_abra_fclna_cmos0, CMOS mode           t400      -2.56    2.31
Clock duty cycle deviation at p13_0:LVDSTX.msc0_fclna_cmos0, CMOS mode                t400      -2.62    2.32
Clock duty cycle deviation at p13_1:LVDSTX.msc0_abra_fclpa_cmos0, CMOS mode           t400      -2.44    2.28
Clock duty cycle deviation at p13_1:LVDSTX.msc0_fclpa_cmos0, CMOS mode                t400      -2.37    2.30
Clock duty cycle deviation at p13_2:LVDSTX.msc0_abra_fclpa_cmos1, CMOS mode           t400      -2.66    2.21
Clock duty cycle deviation at p13_2:LVDSTX.msc0_fclpa_cmos1, CMOS mode                t400      -2.59    2.23

SO  output delay to p11_6:PPFAST.msc0_abra_fclpb0, port p11_9:PPFAST, CMOS mode       t44       -2.01    1.66
SO  output delay to p11_6:PPFAST.msc0_abra_fclpb0, port p13_2:LVDSTX, CMOS mode       t44       -2.32    1.49
SO  output delay to p11_6:PPFAST.msc0_abra_fclpb0, port p13_3:LVDSTX, CMOS mode       t44       -2.23    1.03
SO  output delay to p11_6:PPFAST.msc0_fclpb0, port p11_9:PPFAST, CMOS mode            t44       -2.19    2.23
SO  output delay to p11_6:PPFAST.msc0_fclpb0, port p13_2:LVDSTX, CMOS mode            t44       -2.63    2.11
SO  output delay to p11_6:PPFAST.msc0_fclpb0, port p13_3:LVDSTX, CMOS mode            t44       -2.47    1.74
SO  output delay to p13_0:LVDSTX.msc0_abra_fclna_cmos0, port p11_9:PPFAST, CMOS mode  t44       -1.87    1.82
SO  output delay to p13_0:LVDSTX.msc0_abra_fclna_cmos0, port p13_2:LVDSTX, CMOS mode  t44       -2.17    1.64
SO  output delay to p13_0:LVDSTX.msc0_abra_fclna_cmos0, port p13_3:LVDSTX, CMOS mode  t44       -2.09    1.19
SO  output delay to p13_0:LVDSTX.msc0_fclna_cmos0, port p11_9:PPFAST, CMOS mode       t44       -2.10    2.37
SO  output delay to p13_0:LVDSTX.msc0_fclna_cmos0, port p13_2:LVDSTX, CMOS mode       t44       -2.54    2.24
SO  output delay to p13_0:LVDSTX.msc0_fclna_cmos0, port p13_3:LVDSTX, CMOS mode       t44       -2.38    1.87
SO  output delay to p13_1:LVDSTX.msc0_abra_fclpa_cmos0, port p11_9:PPFAST, CMOS mode  t44       -1.43    2.16
SO  output delay to p13_1:LVDSTX.msc0_abra_fclpa_cmos0, port p13_2:LVDSTX, CMOS mode  t44       -1.73    1.99
SO  output delay to p13_1:LVDSTX.msc0_abra_fclpa_cmos0, port p13_3:LVDSTX, CMOS mode  t44       -1.64    1.53
```

Figure 3: STA report

```
MSC
#Mode, #Sig_Port, #Clk_Port, #TParam, #Min, #Max, #Details
Clock duty cycle deviation at,-,p11_6,t400,-2.64,2.25, PPFAST.msc0_abra_fclpb0  CMOS mode
Clock duty cycle deviation at,-,p11_6,t400,-2.62,2.27, PPFAST.msc0_fclpb0  CMOS mode
Clock duty cycle deviation at,-,p13_0,t400,-2.56,2.31, LVDSTX.msc0_abra_fclna_cmos0  CMOS mode
Clock duty cycle deviation at,-,p13_0,t400,-2.62,2.32, LVDSTX.msc0_fclna_cmos0  CMOS mode
Clock duty cycle deviation at,-,p13_1,t400,-2.44,2.28, LVDSTX.msc0_abra_fclpa_cmos0  CMOS mode
Clock duty cycle deviation at,-,p13_1,t400,-2.37,2.30, LVDSTX.msc0_fclpa_cmos0  CMOS mode
Clock duty cycle deviation at,-,p13_2,t400,-2.66,2.21, LVDSTX.msc0_abra_fclpa_cmos1  CMOS mode
Clock duty cycle deviation at,-,p13_2,t400,-2.59,2.23, LVDSTX.msc0_fclpa_cmos1  CMOS mode
output delay to,p11_9,p11_6,t44,-2.01,1.66, PPFAST.msc0_abra_fclpb0  port p11_9:PPFAST  CMOS mode
output delay to,p13_2,p11_6,t44,-2.32,1.49, PPFAST.msc0_abra_fclpb0  port p13_2:LVDSTX  CMOS mode
output delay to,p13_3,p11_6,t44,-2.23,1.03, PPFAST.msc0_abra_fclpb0  port p13_3:LVDSTX  CMOS mode
output delay to,p11_9,p11_6,t44,-2.19,2.23, PPFAST.msc0_fclpb0  port p11_9:PPFAST  CMOS mode
output delay to,p13_2,p11_6,t44,-2.63,2.11, PPFAST.msc0_fclpb0  port p13_2:LVDSTX  CMOS mode
output delay to,p13_3,p11_6,t44,-2.47,1.74, PPFAST.msc0_fclpb0  port p13_3:LVDSTX  CMOS mode
output delay to,p11_9,p13_0,t44,-1.87,1.82, LVDSTX.msc0_abra_fclna_cmos0  port p11_9:PPFAST  CMOS mode
output delay to,p13_2,p13_0,t44,-2.17,1.64, LVDSTX.msc0_abra_fclna_cmos0  port p13_2:LVDSTX  CMOS mode
output delay to,p13_3,p13_0,t44,-2.09,1.19, LVDSTX.msc0_abra_fclna_cmos0  port p13_3:LVDSTX  CMOS mode
output delay to,p11_9,p13_0,t44,-2.10,2.37, LVDSTX.msc0_fclna_cmos0  port p11_9:PPFAST  CMOS mode
output delay to,p13_2,p13_0,t44,-2.54,2.24, LVDSTX.msc0_fclna_cmos0  port p13_2:LVDSTX  CMOS mode
output delay to,p13_3,p13_0,t44,-2.38,1.87, LVDSTX.msc0_fclna_cmos0  port p13_3:LVDSTX  CMOS mode
output delay to,p11_9,p13_1,t44,-1.43,2.16, LVDSTX.msc0_abra_fclpa_cmos0  port p11_9:PPFAST  CMOS mode
output delay to,p13_2,p13_1,t44,-1.73,1.99, LVDSTX.msc0_abra_fclpa_cmos0  port p13_2:LVDSTX  CMOS mode
output delay to,p13_3,p13_1,t44,-1.64,1.53, LVDSTX.msc0_abra_fclpa_cmos0  port p13_3:LVDSTX  CMOS mode
output delay to,p11_9,p13_1,t44,-1.59,2.74, LVDSTX.msc0_fclpa_cmos0  port p11_9:PPFAST  CMOS mode
output delay to,p13_2,p13_1,t44,-2.02,2.62, LVDSTX.msc0_fclpa_cmos0  port p13_2:LVDSTX  CMOS mode
output delay to,p13_3,p13_1,t44,-1.86,2.25, LVDSTX.msc0_fclpa_cmos0  port p13_3:LVDSTX  CMOS mode
output delay to,p11_9,p13_2,t44,-1.62,2.08, LVDSTX.msc0_abra_fclpa_cmos1  port p11_9:PPFAST  CMOS mode
output delay to,p13_2,p13_2,t44,-,-, LVDSTX.msc0_abra_fclpa_cmos1  port p13_2:LVDSTX  CMOS mode
output delay to,p13_3,p13_2,t44,-1.83,1.45, LVDSTX.msc0_abra_fclpa_cmos1  port p13_3:LVDSTX  CMOS mode
output delay to,p11_9,p13_2,t44,-1.76,2.67, LVDSTX.msc0_fclpa_cmos1  port p11_9:PPFAST  CMOS mode
```

Figure 4: CSV generated from the STA report

Alternatively if the datasheet contains timing values, it can be directly used. Below figure 5/6 shows examples of CSV which are using datasheet parameters for Ethermac and SDMMC

```
ETH,,,,,,,,
#Mode, #Check_Mode, #Sig_Port, #Clk_Port, #TParam, #Min, #Max, #Offset
rgmii,Clock duty cycle deviation at,8,p11_4,t19,-0.8,0.8,,
rgmii,Clock duty cycle deviation at,8,p11_12,t19,-0.8,0.8,,
rgmii,output delay to,p11_3,p11_4,t20,-0.5,0.5,2,
rgmii,output delay to,p11_2,p11_4,t20,-0.5,0.5,2,
rgmii,output delay to,p11_1,p11_4,t20,-0.5,0.5,2,
rgmii,output delay to,p11_0,p11_4,t20,-0.5,0.5,2,
rgmii,output delay to,p11_10,p11_12,t21,-2.6,-1,3,
rgmii,output delay to,p11_9,p11_12,t22,-2.6,-1,3,
rgmii,output delay to,p11_8,p11_12,t23,-2.6,-1,3,
rgmii,output delay to,p11_7,p11_12,t24,-2.6,-1,3,
```

Figure 5: Ethermac timing parameters extracted from the datasheet

```
SDMMC,,,,,,,,
#Mode, #Check_Mode, #Sig_Port, #Clk_Port, #TParam, #Min, #Max, #Offset
sdmmc,Clock duty cycle deviation at,20,p15_1,t1,-0.001,0.001,,
sdmmc_out,output delay to,p15_3,p15_1,t5,-3,3,4,
sdmmc_out,output delay to,p20_7,p15_1,t5,-3,3,4,
sdmmc_out,output delay to,p20_8,p15_1,t5,-3,3,4,
sdmmc_out,output delay to,p20_10,p15_1,t5,-3,3,4,
sdmmc_out,output delay to,p20_11,p15_1,t5,-3,3,4,
sdmmc_out,output delay to,p20_12,p15_1,t5,-3,3,4,
sdmmc_out,output delay to,p20_13,p15_1,t5,-3,3,4,
sdmmc_out,output delay to,p20_14,p15_1,t5,-3,3,4,
sdmmc_out,output delay to,p15_0,p15_1,t5,-3,3,4,
sdmmc_in,output delay to,p15_3,p15_1,t5,2.5,6.3,0,
```

Figure 6: SDMMC timing parameters extracted from the datasheet

IV: Fundamental Assertions

For the verification of the three different types of IO timings, we use two fundamental assertions. These are called check_delay and check_deviation.

```
module check_delay(input clk, input data, input enable_cond);

parameter string msg = "AssertFailed : Output delay violation";
parameter realtime pMaxDelay = 20ns;
parameter realtime pMinDelay = 10ns;
parameter realtime pDataOffset = 0ns; //This should be more than modulo(maxium of all the min delays)

wire data_del;
assign #pDataOffset data_del = data;

realtime tClkEdge =0;
realtime tDataEdge=0;
//store delta time
always begin
  @(clk);
  if(enable_cond) begin
     tClkEdge = $realtime;
  end
end

always begin
  @(data_del);
  if(enable_cond) begin
     tDataEdge = $realtime - tClkEdge - pDataOffset;
     a_delta_delay: assert ((tDataEdge < pMaxDelay) && (tDataEdge >= pMinDelay))
       else
         begin
           $error(msg);
           $display("\t(actual=%3.3f, min=%3.3f,max=%3.3f)\n",tDataEdge, pMinDelay,pMaxDelay);
         end
   end
 end

 endmodule
```

```
module check_deviation(input clk,input enable_cond);

parameter string msg = "AssertFailed : clock cyle deviation violation";
parameter realtime pMaxDelay = 20ns;
parameter realtime pMinDelay = 10ns;
parameter realtime pClkPeriod = 200ns;

realtime tClkPosEdge =0;
realtime tClkDeviation=0;
//store delta time
always begin
  @(posedge clk);
  if(enable_cond) begin
     tClkPosEdge = $realtime;
     @(negedge clk);
     tClkDeviation = $realtime - tClkPosEdge - pClkPeriod/2;
     a_check_deviation: assert ((tClkDeviation < pMaxDelay) && (tClkDeviation >= pMinDelay))
       else
         begin
           $error(msg);
           $display("\t(actual=%3.3f, min=%3.3f,max=%3.3f)\n",tClkDeviation, pMinDelay,pMaxDelay);
         end
  end
end
endmodule
```

Figure 7: Fundamental assertions used to verify the IO timings.

The key here is to use parameters for the values that are checked. For the Clock duty cycle deviation check we use three parameters, pClkPeriod, pMinDelay and pMaxDelay. For the output delay IO timing check we also use three parameters, pDataOffset, pMinDelay and pMaxDelay. The pDataOffset helps to overcome the scenarios where negative timings may be present.

V. Generating all the assertions required for complete IO timing verification

Once we have the fundamental assertions ready we can move on to the generation of the specific assertions for each of the specified IO timings which we described in Figure 4/5/6.

We used Cadence[tm] Perspec[tm] tool to achieve this but this can be also done using scripts or other tools. The basic idea is to describe the parameters in a tabular fashion as done in Figure 4/5/6 and then apply these values to instantiate the fundamental assertions for each of the parameter.

```
    <[var index : uint = 0;]>
    <[table from csv_to_table(LIST_OF_MODULES, "TC38x") with { ]>
      <[table from csv_to_table(DATASHEET_PARAMS_FILE,"<#Peripheral>", ((csv_column("Check_Mode") == "Clock duty c
ycle deviation at"))) with { ]>
       wire   <##Peripheral>_dcd<(index)>_enable;
       assign <##Peripheral>_dcd<(index)>_enable = 1;
       check_deviation #(.pClkPeriod(pClkPeriod), .pMaxDelay(<#Max>ns), .pMinDelay(<#Min>ns),.msg("AssertFailed DC
DMAX<(index)>: Duty Cycle Deviation in <#Mode> mode for clock on port <#Clk_Port> ")) <##Peripheral>_dcd<(index)>
(<#Clk_Port>, global_enable_iotiming_checks && <##Peripheral>_enable_iotiming_checks && <##Peripheral>_dcd<(index)
>_enable);
      <[index = index+1;]>
      <[};]>
      <[index = 0;]>
      <[table from csv_to_table(DATASHEET_PARAMS_FILE,"<#Peripheral>", ((csv_column("Check_Mode") == "output delay
 to"))) with { ]>
         wire   <##Peripheral>_od<(index)>_enable;
         assign <##Peripheral>_od<(index)>_enable = 1;
         check_delay #(.pDataOffset(<#Offset>ns), .pMaxDelay(<#Max>ns),.pMinDelay(<#Min>ns), .msg("AssertFailed O
DMIN<(index)>:Output Delay Deviation in  <#Mode> mode, clk=<#Clk_Port>, sig=<#Sig_Port>")) <##Peripheral>_od<(inde
x)> (<#Clk_Port>,<#Sig_Port>, global_enable_iotiming_checks && <##Peripheral>_enable_iotiming_checks && <##Periphe
ral>_od<(index)>_enable);
      <[index = index+1;]>
      <[};]>
    <[};]>
```

Figure 8: Perspec[tm] abstraction to instantiate fundamental assertions in SLN language.

```
      wire   ETH_dcd0_enable;
      assign ETH_dcd0_enable = 1;
      check_deviation #(.pClkPeriod(pClkPeriod), .pMaxDelay(0.8ns), .pMinDelay(-0.8ns),.msg("AssertFailed DCDMAX0
: Duty Cycle Deviation in rgmii mode for clock on port p11_4 ")) ETH_dcd0 (p11_4, global_enable_iotiming_checks &&
 ETH_enable_iotiming_checks && ETH_dcd0_enable);
      wire   ETH_dcd1_enable;
      assign ETH_dcd1_enable = 1;
      check_deviation #(.pClkPeriod(pClkPeriod), .pMaxDelay(0.8ns), .pMinDelay(-0.8ns),.msg("AssertFailed DCDMAX1
: Duty Cycle Deviation in rgmii mode for clock on port p11_12 ")) ETH_dcd1 (p11_12, global_enable_iotiming_checks
&& ETH_enable_iotiming_checks && ETH_dcd1_enable);
         wire   ETH_od0_enable;
         assign ETH_od0_enable = 1;
         check_delay #(.pDataOffset(2ns), .pMaxDelay(0.5ns),.pMinDelay(-0.5ns), .msg("AssertFailed ODMIN0:Output
Delay Deviation in  rgmii mode, clk=p11_4, sig=p11_3")) ETH_od0 (p11_4,p11_3, global_enable_iotiming_checks && ETH
_enable_iotiming_checks && ETH_od0_enable);
         wire   ETH_od1_enable;
         assign ETH_od1_enable = 1;
         check_delay #(.pDataOffset(2ns), .pMaxDelay(0.5ns),.pMinDelay(-0.5ns), .msg("AssertFailed ODMIN1:Output
Delay Deviation in  rgmii mode, clk=p11_4, sig=p11_2")) ETH_od1 (p11_4,p11_2, global_enable_iotiming_checks && ETH
_enable_iotiming_checks && ETH_od1_enable);
         wire   ETH_od2_enable;
         assign ETH_od2_enable = 1;
         check_delay #(.pDataOffset(2ns), .pMaxDelay(0.5ns),.pMinDelay(-0.5ns), .msg("AssertFailed ODMIN2:Output
Delay Deviation in  rgmii mode, clk=p11_4, sig=p11_1")) ETH_od2 (p11_4,p11_1, global_enable_iotiming_checks && ETH
_enable_iotiming_checks && ETH_od2_enable);
         wire   ETH_od3_enable;
         assign ETH_od3_enable = 1;
         check_delay #(.pDataOffset(2ns), .pMaxDelay(0.5ns),.pMinDelay(-0.5ns), .msg("AssertFailed ODMIN3:Output
Delay Deviation in  rgmii mode, clk=p11_4, sig=p11_0")) ETH_od3 (p11_4,p11_0, global_enable_iotiming_checks && ETH
_enable_iotiming_checks && ETH_od3_enable);
         wire   ETH_od4_enable;
         assign ETH_od4_enable = 1;
```

Figure 9: Perspec[tm] Generated instantiations of the fundamental check_delay and check_deviation assertions for Ethermac.

Perspec has built-in system function csv_to_table to read in tables but the tables must follow certain format. Each table must start with a table header followed by a row of column headers where each column header must start with a '#'. In figure 5 'ETH' is the table header and #Mode, #Check_Mode etc are the respective column headers. We use another table to list all the modules for which assertions are to be generated. This table (stored in list_of_modules.csv file) looks like the following (along with the datasheet_params.csv shown in Figure 5)

```
1 TC38x                                  1 ETH,,,,,,,,
2 #Peripheral                            2 #Mode, #Check_Mode, #Sig_Port, #Clk_Port, #TParam, #Min, #Max, #Offset
3 ETH                                    3 rgmii,Clock duty cycle deviation at,8,p11_4,t19,-0.8,0.8,,
4 SDMMC                                  4 rgmii,Clock duty cycle deviation at,8,p11_12,t19,-0.8,0.8,,
5 MSC                                    5 rgmii,output delay to,p11_3,p11_4,t20,-0.5,0.5,2,
6                                        6 rgmii,output delay to,p11_2,p11_4,t20,-0.5,0.5,2,
~                                        7 rgmii,output delay to,p11_1,p11_4,t20,-0.5,0.5,2,
~                                        8 rgmii,output delay to,p11_0,p11_4,t20,-0.5,0.5,2,
~                                        9 rgmii,output delay to,p11_10,p11_12,t21,-2.6,-1,3,
~                                        10 rgmii,output delay to,p11_9,p11_12,t22,-2.6,-1,3,
~                                        11 rgmii,output delay to,p11_8,p11_12,t23,-2.6,-1,3,
~                                        12 rgmii,output delay to,p11_7,p11_12,t24,-2.6,-1,3,
~                                        13
~                                        14 SDMMC,,,,,,,,
~                                        15 #Mode, #Check_Mode, #Sig_Port, #Clk_Port, #TParam, #Min, #Max, #Offset
list_of_modules.csv [RO]   6,0-1         All datasheet_params.csv [RO]                  4,43          Top
```

Figure 10: Example of the tables used in Perspec to generate assertions

Now let us examine the SLN code shown in Figure 8 to show how it reads the various tables to dump out assertions.

The first line defines a variable index and initializes it to 0.

```
<[var index : unit = 0;]>
```

This variable is used to generate instances of the fundamental assertions check_deviation and check_delay.

Line 2 and 3 in Figure 8 call the system function csv_to_table to read in the tables shown in Figure 10.
Line 4 then uses these tables to instantiate the fundamental assertion check_deviation.

```
<[table from csv_to_table(LIST_OF_MODULES, "TC38x") with { ]>
      <[table from csv_to_table(DATASHEET_PARAMS_FILE,"<#Peripheral>", ((csv_column("Check_Mode") == "Clock du-
ty cycle deviation at"))) with { ]>
      check_deviation  #(.pClkPeriod(pClkPeriod),  .pMaxDelay(<#Max>ns),  .pMinDelay(<#Min>ns),.msg("AssertFailed
DCDMAX<(index)>:  Duty  Cycle  Deviation  in  <#Mode>  mode  for  clock  on  port  <#Clk_Port>  "))  <##Peripher-
al>_dcd<(index)>  (<#Clk_Port>,  global_enable_iotiming_checks  &&  <##Peripheral>_enable_iotiming_checks  &&  <##Pe-
ripheral>_dcd<(index)>_enable);
```

The #Peripheral would be replaced with ETH, SDMMC and MSC in each iteration.

The function csv_to_table takes additional arguments so we can iterate over rows with specific values in a given column. Here although ETH has rows for both ‘clock duty cycle deviation’ and ‘output delay’, we only want to instantiate check_deviation assertion for ‘clock duty cycle deviation’ rows.

Once specific table and rows are identified, we can extract the values of the required parameters and add them to our code using #<column header>. For example <#Max> would be replaced by 0.8ns.

```
check_deviation #(.pClkPeriod(pClkPeriod), .pMaxDelay(0.8ns), .pMinDelay(-0.8ns),.msg("AssertFailed DCDMAX0: Duty Cycle Deviation in rgmii mode for clock on port p11_4 ")) ETH_dcd0 (p11_4, global_enable_iotiming_checks && ETH_enable_iotiming_checks && ETH_dcd0_enable);
check_deviation #(.pClkPeriod(pClkPeriod), .pMaxDelay(0.8ns), .pMinDelay(-0.8ns),.msg("AssertFailed DCDMAX1: Duty Cycle Deviation in rgmii mode for clock on port p11_12 ")) ETH_dcd1 (p11_12, global_enable_iotiming_checks && ETH_enable_iotiming_checks && ETH_dcd1_enable);
check_deviation #(.pClkPeriod(pClkPeriod), .pMaxDelay(0.001ns), .pMinDelay(-0.001ns),.msg("AssertFailed DCDMAX8: Duty Cycle Deviation in sdmmc mode for clock on port p15_1 ")) SDMMC_dcd8 (p15_1, global_enable_iotiming_checks && SDMMC_enable_iotiming_checks && SDMMC_dcd8_enable);
```

Notice how the ports and module names are replaced taking values from the parameters listed in Figure 5 and 6. By nesting the tables, we can generate assertions for all the ports of all the modules with a few lines of code.

The check_delay assertion is instantiated in a similar way. We then instantiate all the generated assertions in a System Verilog module and integrated it with the existing top level testbench.

The check_delay and check_deviation can be implemented independently in any language and the SLN code will still work as long as the CSV format is maintained.

VI. Running and debugging the assertions

The generated assertions are compiled as a parallel top and run with the existing test suite where the targeted modules are doing tx/rx so that the pins are exercised and assertions are triggered. Any deviation is reported in the logfiles and can be debugged in the waves as usual. Figure 10 shows one such typical failure when the assertions are run in dynamic simulation

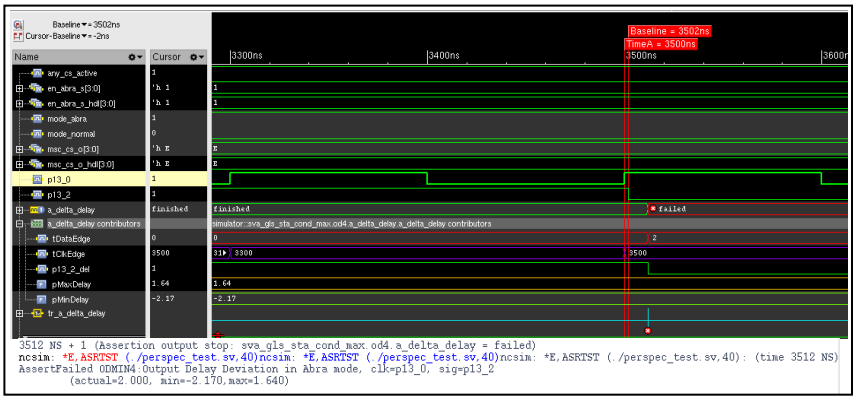


Figure 11: Assertion failure in dynamic simulation

Above example shows a failure for the MSC module where the output delay was observed as 2ns however the pMaxDelay is 1.64ns. This could be seen in the failure message printed in the logfile.

Fortunately this was not a real bug as the testcase did not set the output pads in the fastest mode.

VII. Usage and challenges in running multiple modules:

As mentioned in introduction we successfully used this methodology in some of our IPs like Gigabit Ethermac, Quad SPI and SDMMC etc. However there were some adaptions required when testing different modules.

Some modules work on both edges and here the fundamental assertions were adapted to work on both edges. However the SLN code was the same.

Some testcases applied multiple modes in a single testcase. So assertions were required to be enabled/disabled automatically instead of being enabled all the time.

Many a times a module pin would toggle as soon as the module is enabled. This would not follow the timing parameters since no actual tx/rx would have started. Additional enable/disable logic was added to overcome such issues.

Once these basic issues were sorted out, it was really easy to run the assertions for different modules. Below we provide two such examples for Ethermac and SDMMC.

Gigabit Ethermac: Gigabit Ethernet, a transmission technology based on the Ethernet frame format and protocol used in local area networks (LANs), provides a data rate of 1 billion bits per second (one gigabit). Gigabit Ethernet is defined in the IEEE 802.3 standard and is currently being used as the backbone in many enterprise networks.

The reduced gigabit media independent interface (RGMII) has become a widely used alternative to the gigabit media independent interface (GMII) by offering lower pin count which enables board space, and cost, savings. The RGMII standard achieves this by reducing parallel data bus width and through double data rate (DDR). RGMII specifies that the clock and data will be generated simultaneously by the transmitting source which requires a skew be introduced between clock and data. The skew can be achieved by PCB trace routing or by an internal delay in the transmitting or receiving node. The skew imposed on the clock and data shall be chosen carefully to ensure meeting the requirements of the interface.

There are tough timing budget for Gigabit Ethermac to meet the requirement. There are timing skew requirement as low as 500 ps. Here are the timing requirements at a glance:

Symbol	Parameter	Min	Typ	Max	Units
TskewT	Data to Clock output Skew (at Transmitter)	-500	0	500	ps
TskewR	Data to Clock input Skew (at Receiver)	1	1.8	2.6	ns
TsetupT	Data to Clock output Setup	1.2	2		ns
TholdT	Data to Clock output Hold	1.2	2		ns
TsetupR	Data to Clock input Setup	1	2		ns
TholdR	Data to Clock input Hold	1	2		ns
Tcyc	Clock Cycle Duration (1)	7.2	8	8.8	ns

Figure 12: Gigabit Ethermac IO timing requirements

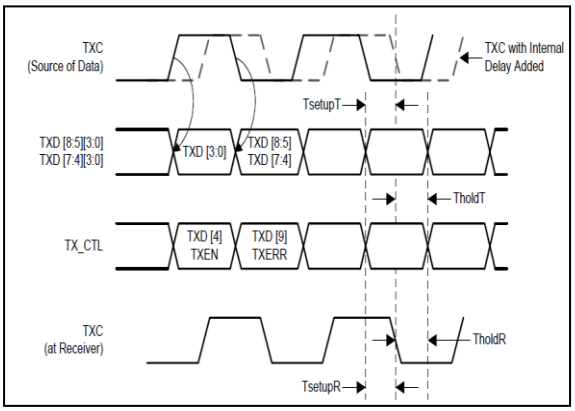


Figure 13: RGMII TX Timing Diagram

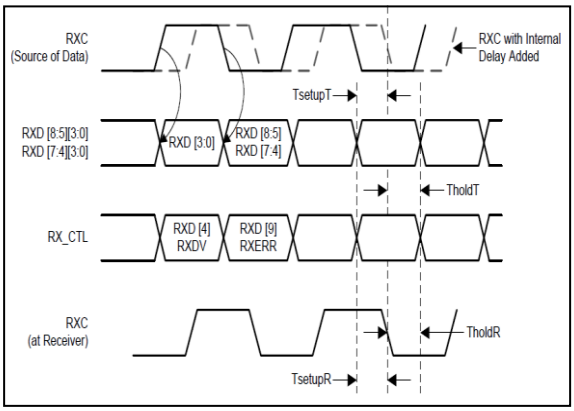


Figure 14: GMII RX Timing Diagram

With above so strict timing requirements GEthermac become really critical in terms of meeting the IO timing requirement and checking the interface for correct timing. To quickly verify and ensure that STA assumption are correct and meeting the requirement as per the standard specification we used the Portable Stimulus to generate System Verilog assertions using Perspec.

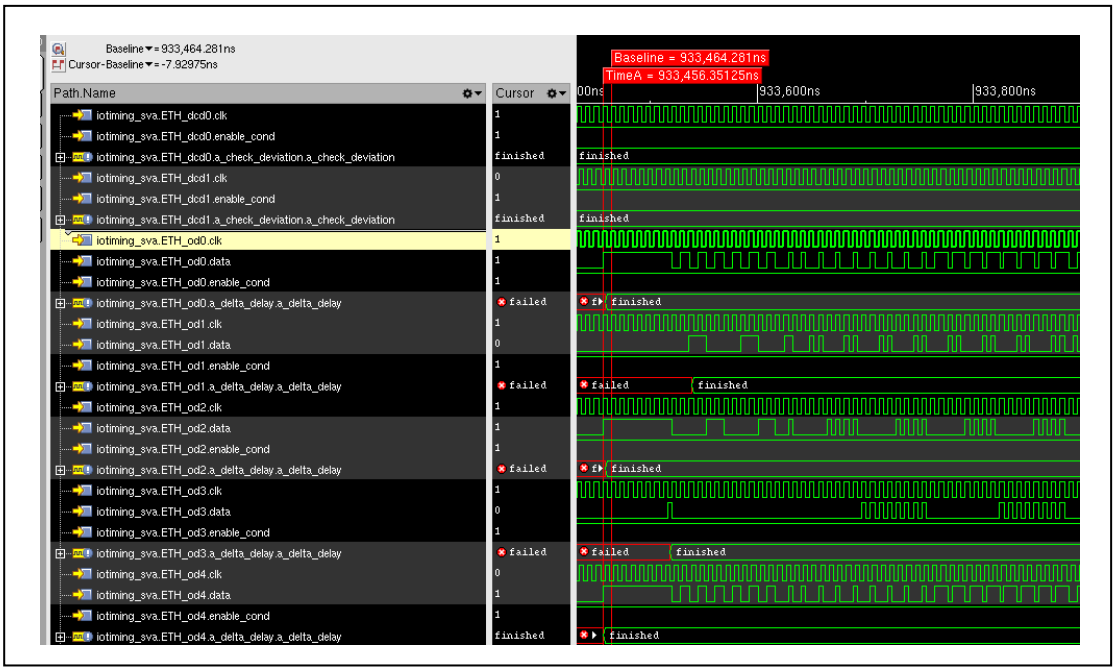


Figure 15: Gigabit Ethermac Results

From above snapshot it can be seen that full frame transmission is achieved in dynamic simulation and continuous checking of "clock Deviation" and "setup/Hold" are checked at I/O interface.

SDMMC (SD/MMC Card Controller): The purpose of the SDMMC module is to enable communication to external managed NAND Flashes using the SD or eMMC interface.

SDMMC supports following modes:

a. Communication to eMMC memories
    1. Communication using 1-, 4- or 8-data lines
    2. Legacy MMC card- and High-speed SDR mode supported
b. Communication to SD-cards
    1. Communication using 1- or 4-data lines.
    2. Default- and High Speed Mode supported.

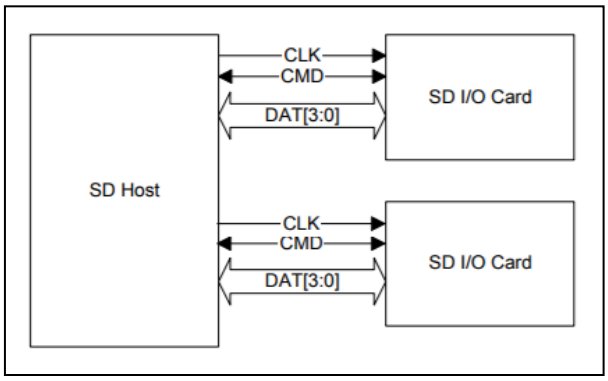


Figure 16: SDMMC controller Pin Connections.

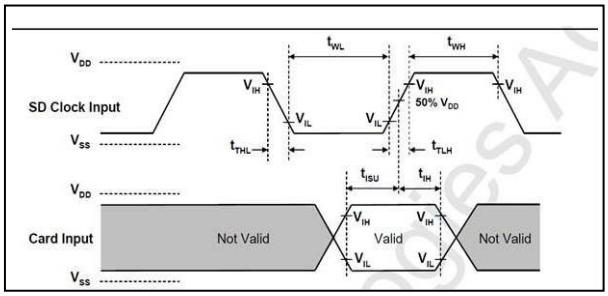
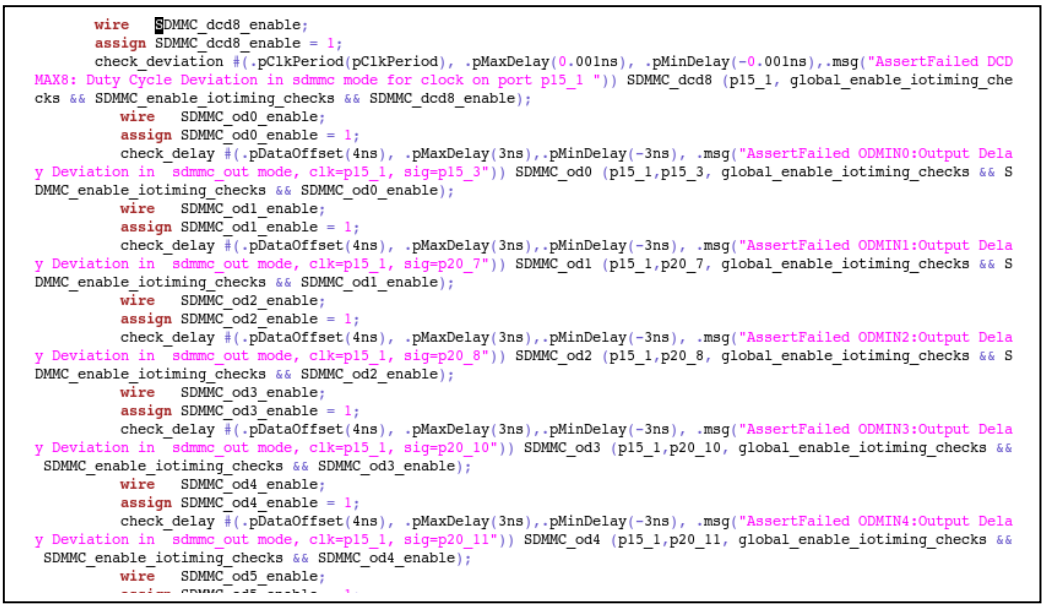


Figure 17: SDMMC IO timing diagram.

Parameter	Symbol	Min	Max	Unit	Notes
Clock frequency in data transfer mode	fPP	0	50	MHz	CL≤10pF
Clock low time	tWL	7.0		ns	
Clock high time	tWH	7.0		ns	
Clock rise time	tTLH		3	ns	
Clock fall time	tTHL		3	ns	
CMD, DAT input setup time	tISU	6		ns	
CMD, DAT input hold time	tIH	2		ns	
CMD, DAT output delay time during data transfer mode	tODLY		14	ns	
CMD, DAT output hold time	tOH	2.5		ns	

Figure 18: SDMMC IO timing parameters.



```
        wire   SDMMC_dcd8_enable;
        assign SDMMC_dcd8_enable = 1;
        check_deviation #(.pClkPeriod(pClkPeriod), .pMaxDelay(0.001ns), .pMinDelay(-0.001ns),.msg("AssertFailed DCD
MAX8: Duty Cycle Deviation in sdmmc mode for clock on port p15_1 ")) SDMMC_dcd8 (p15_1, global_enable_iotiming_che
cks && SDMMC_enable_iotiming_checks && SDMMC_dcd8_enable);
          wire   SDMMC_od0_enable;
          assign SDMMC_od0_enable = 1;
          check_delay #(.pDataOffset(4ns), .pMaxDelay(3ns),.pMinDelay(-3ns), .msg("AssertFailed ODMIN0:Output Dela
y Deviation in  sdmmc_out mode, clk=p15_1, sig=p15_3")) SDMMC_od0 (p15_1,p15_3, global_enable_iotiming_checks && S
DMMC_enable_iotiming_checks && SDMMC_od0_enable);
          wire   SDMMC_od1_enable;
          assign SDMMC_od1_enable = 1;
          check_delay #(.pDataOffset(4ns), .pMaxDelay(3ns),.pMinDelay(-3ns), .msg("AssertFailed ODMIN1:Output Dela
y Deviation in  sdmmc_out mode, clk=p15_1, sig=p20_7")) SDMMC_od1 (p15_1,p20_7, global_enable_iotiming_checks && S
DMMC_enable_iotiming_checks && SDMMC_od1_enable);
          wire   SDMMC_od2_enable;
          assign SDMMC_od2_enable = 1;
          check_delay #(.pDataOffset(4ns), .pMaxDelay(3ns),.pMinDelay(-3ns), .msg("AssertFailed ODMIN2:Output Dela
y Deviation in  sdmmc_out mode, clk=p15_1, sig=p20_8")) SDMMC_od2 (p15_1,p20_8, global_enable_iotiming_checks && S
DMMC_enable_iotiming_checks && SDMMC_od2_enable);
          wire   SDMMC_od3_enable;
          assign SDMMC_od3_enable = 1;
          check_delay #(.pDataOffset(4ns), .pMaxDelay(3ns),.pMinDelay(-3ns), .msg("AssertFailed ODMIN3:Output Dela
y Deviation in  sdmmc_out mode, clk=p15_1, sig=p20_10")) SDMMC_od3 (p15_1,p20_10, global_enable_iotiming_checks &&
 SDMMC_enable_iotiming_checks && SDMMC_od3_enable);
          wire   SDMMC_od4_enable;
          assign SDMMC_od4_enable = 1;
          check_delay #(.pDataOffset(4ns), .pMaxDelay(3ns),.pMinDelay(-3ns), .msg("AssertFailed ODMIN4:Output Dela
y Deviation in  sdmmc_out mode, clk=p15_1, sig=p20_11")) SDMMC_od4 (p15_1,p20_11, global_enable_iotiming_checks &&
 SDMMC_enable_iotiming_checks && SDMMC_od4_enable);
          wire   SDMMC_od5_enable;
```

Figure 19: Assertions Generated for SDMMC:

VIII. Summary:

As described in [1] the use of SVA to verify IO timings is very useful. However manually creating all the required assertions could be very challenging. Deploying the power of portable stimulus using Cadence's Perspec[tm] tool helps to overcome these challenges with minimal effort in modelling and scripting.

A successful regression with these assertions running in full timing gate level simulation gives a very high confidence for the IO timing sign off.

References:

[1] Using System Verilog Assertions in Gate-Level Verification Environments, Mark Litterick, Verilab, Munich, Germany. (mark.litterick@verilab.com) : http://www.verilab.com/files/sva_gate_paper_dvcon2006.pdf

[2] Datasheet for 32bit Infineon Microcontroller : https://www.infineon.com/dgdl/Infineon-TC27xDC_DS_v10-DS-v01_00-EN.pdf?fileId=5546d46259d9a4bf015a846b292f74ce

[3] Perspec System Verifier – Usecase driven SoCVerification: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/system-design-verification/perspec-system-verifier-ds.pdf