Verifying functionality is simply not enough

Rajesh Bawankule
Sr. Research Engineer
Nokia-Siemens Networks
380 N Bernardo Av.
Mountain View, CA 94043
Rajesh.Bawankule@nsn.com

ABSTRACT

SoC designers increasingly incorporate a significant amount of IPs from third-party IP vendors. IP providers verify their IPs thoroughly from a functionality point of view but they often lack the understanding (and rightfully so) of a bigger picture in which their IPs may be used. Many IP providers also provide Verification IP (VIP) along with their design or implementation IPs to assist and speed up the verification process. Generally this process minimizes functional bugs as the vendor has spent considerable time and energy on finding and fixing functional issues. However, issues related to target throughput and how an IP will behave in a system context are more difficult to find.

In this paper, we describe our experience in creating test benches which quantify IP throughput to find out functional issues which cause throughput drops.

The key takeaway

The observation of throughput, latency, and flow control of an IP subsystem can reveal issues with the IP as well as the surrounding design which uses that IP.

Keywords

I.P., throughput, verification, functionality, specifications, latency, flow control.

1. Introduction

As a part of the Nokia-Siemens Networks's research group, we evaluate various IPs routinely. Evaluating them quickly and efficiently is the key. In one such project, we created a test infrastructure to validate the functionality and throughput of a new memory subsystem for networking gear 100G and beyond. This paper shows various lessons learned during that process. We also include code snippets and scripts that can be used by the readers for their projects.

The discussion is divided into 3 parts.

a. Throughput

b. Latency

c. Flow Control

2. Throughput

Throughput is how fast the IPs can execute certain functions in the real world. For example, a memory vendor may specify throughput in terms of bandwidth of 10GBytes of write data and 10GBytes of read data with 90% of utilization.

It was assumed that this memory will provide a simple SRAM-like interface to offer drop-in replacements for an existing design's memory interfaces. Like SRAM, it will provide simultaneously read and write into a flat address space.

The picture below shows the high level view of a test mechanism. The testing was done on RTL and netlist as well as on silicon at a later stage.

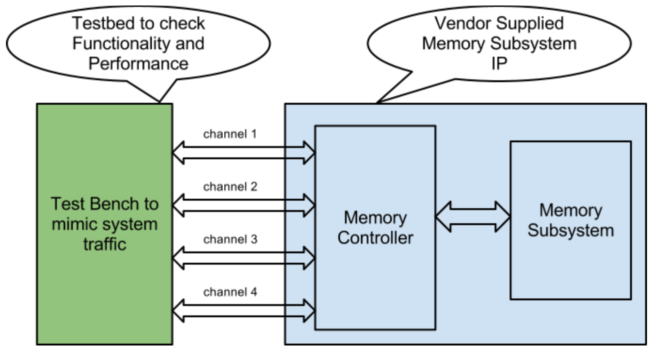
Testbed to check Functionality and Performance
Vendor Supplied Memory Subsystem IP
Test Bench to mimic system traffic
channel 1
channel 2
channel 3
channel 4
Memory Controller
Memory Subsystem

Figure 1: Test setup

2.1 Verifying functionality

In the first phase of verification, a set of assertions and data checkers were created to verify the basic functionality of memory subsystem IP. This was expected to pass easily, and no issues were discovered as the vendor had also completed similar testing in great detail.

2.2 Verifying Throughput

From past experience we have learned to take vendor claims of throughput with a grain of salt. We decided to create an exhaustive testing mechanism to verify throughput as it is difficult to design an IP subsystem to meet every customer's requirements.

In the second phase, a mechanism was created to measure

- instantaneous throughput

- latency for each transaction

- flow control / push back

In addition, these metrics are displayed graphically as the simulation was progressing.

2.3 Observations

The following graph shows the throughput of the memory subsystem under various usage patterns. The graph shows the normalized throughput when the usage patterns are changed over a period of time.

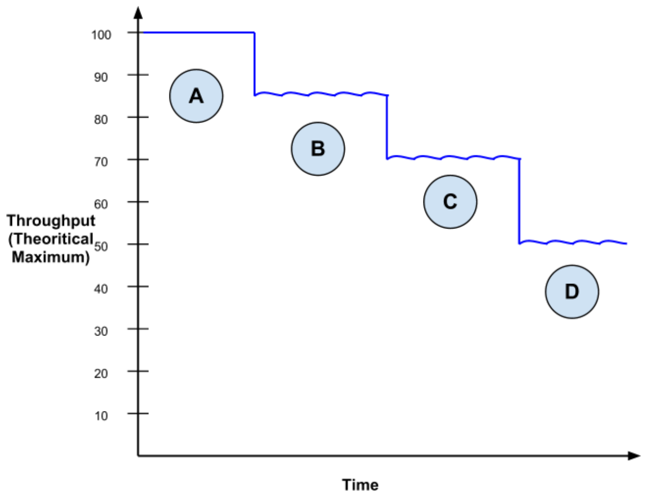

Figure 2: Throughput of memory under various usage patterns

A pattern or combination of patterns was discovered after long simulations which brought throughput down to 50%. The first usage pattern "A" corresponds to the ideal conditions or parameters. The second pattern "B" corresponds to what we believe to be a commonly used pattern. This pattern resulted in 85% of the published bandwidth. The last pattern "D" drives down memory bandwidth to 50% of published numbers. This was a cause for concern.

In our experiment the usage patterns are created by merely changing the 32 address bits, and thus the addressing pattern.

According to the specification, the IP vendor claimed that address bits can be used in flat fashion and there is no need to know their internal banking architecture.

In reality, the memory IP used a group of address bits for addressing various banks. Changing these bits rapidly or using these bits as lower address bits created internal bank conflicts and a bottleneck in the memory controller design.

2.4 Problems in the future avoided

Imagine if this drop in memory throughput went unnoticed. If the system around memory subsystem was unaware of this limitation and created transactions which fall under this pattern then it could have created an overall low performing system that would have been very difficult and time consuming to debug. This issue/bug within memory IP was not obvious by looking at the specifications.

3. Latency

The second important objective is to verify latency. Low latency is a key factor in most next generation networking gears. To achieve low latency it is imperative to use all IPs and especially interface IPs with the lowest possible latency.

We used the test setup and probing mechanism similar to what we used earlier to test throughput. The test mechanism is shown again.

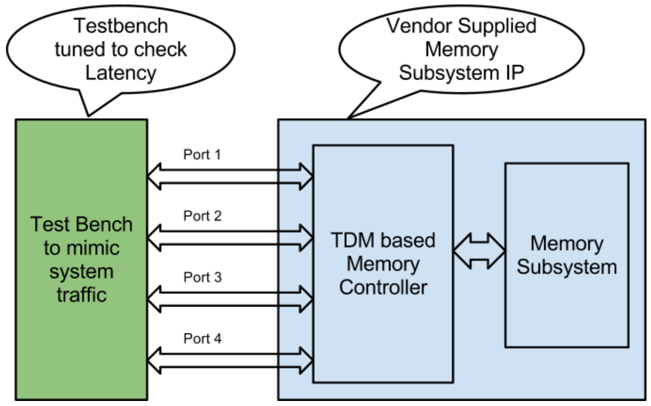
Testbench tuned to check Latency
Vendor Supplied Memory Subsystem IP
Test Bench to mimic system traffic
Port 1
Port 2
Port 3
Port 4
TDM based Memory Controller
Memory Subsystem

Figure 3: Test setup for Latency measurement

The read latency is the time difference between the time when a read operation is submitted to Memory Controller and the time when data is received. The latency numbers obtained for multiple read operations on one port at random times are shown below. The expected latency was 30nS.

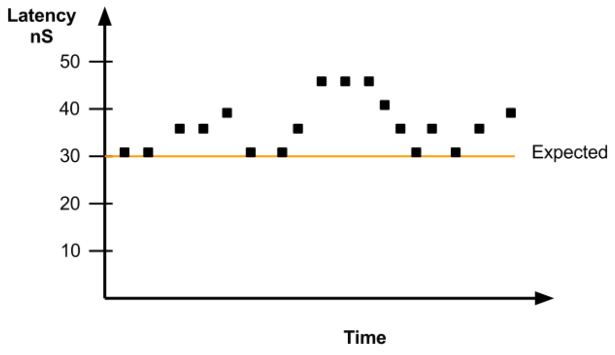

Figure 4: Graph showing observed latency issue

Further investigation revealed that the TDM architecture of memory controller was the cause. The latency increased based on the clock edge on which read request was launched.

4. Flow control

In almost all hardware designs, flow control / back pressure is one of the least tested functions. If it remains untested, it can lower the desired performance and even cause system lock ups. We created a mechanism to observe flow control signals visually to provide feedback of when and where things are going wrong.

The graph for flow control is obtained using the techniques similar to throughput calculations. The duty cycle of a signal used for back pressure can be used for displaying flow control.

Many times flow control / back pressure are tied to other issues like latency and throughput. The following diagram shows flow control observed along with latency numbers. This test uses the extended version of the test case used for latency observation. The test case was modified to hit the corner case repeatedly.

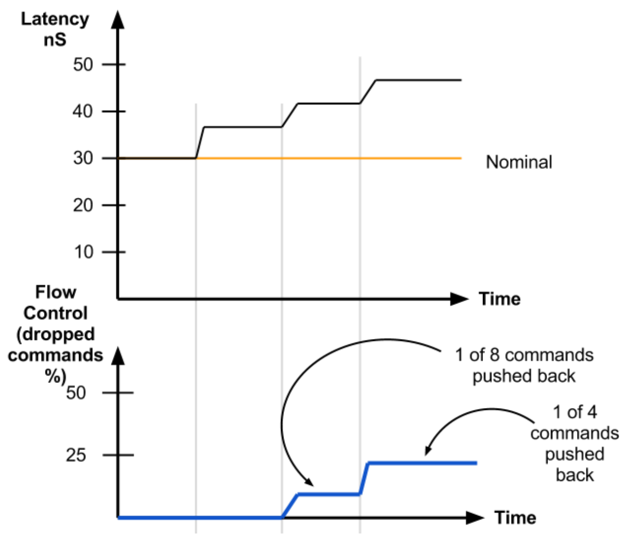

Figure 5 : Graph showing latency along with flow control

5. How it was done

A block diagram of overall reporting mechanism is shown in Figure 6. The graphical reporting mechanism was created using a two step approach.

1. Run the test bench which had a reporting and scaling mechanism. It created a smooth throughput number from instantaneous bandwidth measured every clock.
2. A gnuplot script is run to display data in .csv file graphically.

The details of each block are given below.

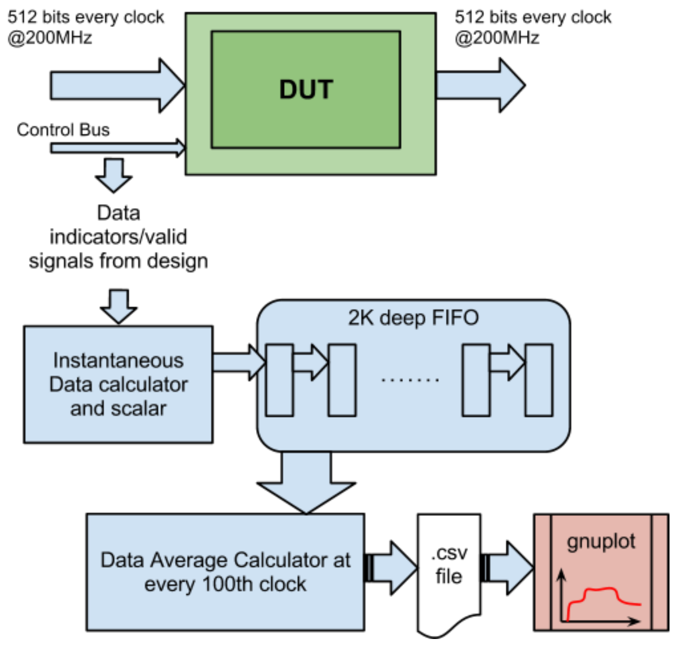
512 bits every clock @200MHz
DUT
512 bits every clock @200MHz
Control Bus
Data indicators/valid signals from design
2K deep FIFO
Instantaneous Data calculator and scalar
Data Average Calculator at every 100th clock
.csv file
gnuplot

Figure 6: Block diagram of reporting mechanism

5.1 Instantaneous Data Calculator

In the example above the DUT is operating at 200MHz and input data width is 512 bits. DUT can receive and send a theoretical maximum of 102.4 GBits. The input data bus is observed on every clock and an instantaneous number of bits is saved into a FIFO. A simple example is shown below. The code can be modified to take care of multiple queues as well as control signals like byte_enable which may indicate that only a part of 512 bits as valid.

```
// In FIFO
always @(posedge clk or posedge sys_rst) begin
   if (sys_rst) begin
     for(k0=0;k0<2000;k0=k0+1)
       in_fifo_0[k0]  <= 'h0;
   else begin
     if(config_done) begin
     // Shift the data to next higher location
       for(k0=1999;k0>0;k0=k0-1)
         in_fifo_0[k0] <= in_fifo_0[k0 - 1];
     // Write data amount on 0th location
       if(Valid_In) begin
        if(InQNum == 0) in_fifo_0[0] <= 'd512;
       end
      end
      else
        in_fifo_0[0] <= 'd0;
   end // else: !if(sys_rst)
end // always @ (posedge clk or posedge sys_rst)
```

The config_done or other signals can be used to selectively write to the csv file to make graph compact and easy to read.

5.2 Data Average Calculator

This block adds all the instantaneous data values stored in FIFO every 100 clocks and prints it out to a csv file. This file will be loaded in gnuplot to plot the chart.

It is essential to understand the averaging process as it affects the resolution as well as smoothness of the graph. In the following example the FIFO collects a maximum of 512 bits every clock for 2000 clocks.

```
integer handle_q0;
initial handle_q0 = $fopen("In_q0.csv");

reg [34:0] in_temp_0;
reg [35:0] in_rate_0,
always @(posedge clk or posedge sys_rst) begin
  if (sys_rst)
    in_rate_0  <= 'h0;
  else begin
    if(config_done) begin
      // for every 100th clock
      if(cc_100) begin
```

```
       in_temp_0 = 'h0;
         for(i=0;i<2000;i=i+1)
          in_temp_0 = in_temp_0 + in_fifo_0[i];
            // Data Averager
           in_rate_0 <= in_temp_0/'d10000;// GBits
           $fdisplay(handle_q0, "%0d,  %0d ",
              clock_counter,in_rate_0);
      end // if (cc_100)
    end // if (config_done)
 end // else: !if(sys_rst)
end // always @ (posedge clk or posedge sys_rst)
```

5.3 Using gnuplot

A sample csv file generated from incoming and outgoing traffic from a DUT is shown below.

```
164800,  0,  0
164900,  0,  0
165000,  1916,  0
165100,  3797,  0
165200,  5679,  0
165300,  7683,  0
165400,  9688,  0
165500,  11692,  0
165600,  13697,  668
165700,  15947,  1670
165800,  18040,  2672
165900,  20378,  4131
166000,  22629,  5591
166100,  24721,  6593
166200,  27060,  8017
166300,  29706,  9020
166400,  32133,  10022
166500,  34744,  11358
166600,  37416,  12695
166700,  39878,  14031
```

Notice the following key points

- The file does not start from time zero but from a point where actual transactions are enabled by “config_done” signal. It indicates that initial configuration setup is completed. This reduces lines with zero values and compacts the graph.
- Delay in output due to latency of DUT.

A sample gnuplot script is given below.

```
# Gnuplot script file for plotting data in file "test.csv"
# This file is called test.p
set   autoscale       # scale axes automatically
unset log             # remove any log-scaling
unset label           # remove any previous labels
set xtic auto         # set xtics automatically
set ytic auto         # set ytics automatically
set title "Input and Output Rate"
set xlabel "Time (No. of clocks)"
set ylabel "Rate (MBits/second)"
#set key 0.01,100
#set label "Yield Point" at 0.003,260
#set arrow from 0.0028,250 to 0.003,280
#set xr [100:300]
#set yr [0:2000000]
plot "test.csv" using 1:2 title 'In Rate' with linespoints, \
   "test.csv" using 1:3 title 'Out Rate' with linespoints
```

This script can be loaded in gnuplot from command line as shown below.

```
$ gnuplot
 G N U P L O T
 Version 4.0 patchlevel 0
 last modified Thu Apr 15 14:44:22 CEST 2004
 System: Linux 2.6.18-194.32.1.el5
 Copyright (C) 1986 - 1993, 1998, 2004
 Thomas Williams, Colin Kelley and many others
 This is gnuplot version 4.0.  Please refer to the documentation  for command syntax changes.  The old syntax will be accepted  throughout the 4.0 series, but all save files use the new syntax.
 Type `help` to access the on-line reference manual.
 The gnuplot FAQ is available from
 http://www.gnuplot.info/faq/
 Send comments and requests for help to
 <gnuplot-info@lists.sourceforge.net>
gnuplot> load 'test.p'
```

The log also shows where to obtain gnuplot and its documentation.

A graph obtained for a test csv is shown below.

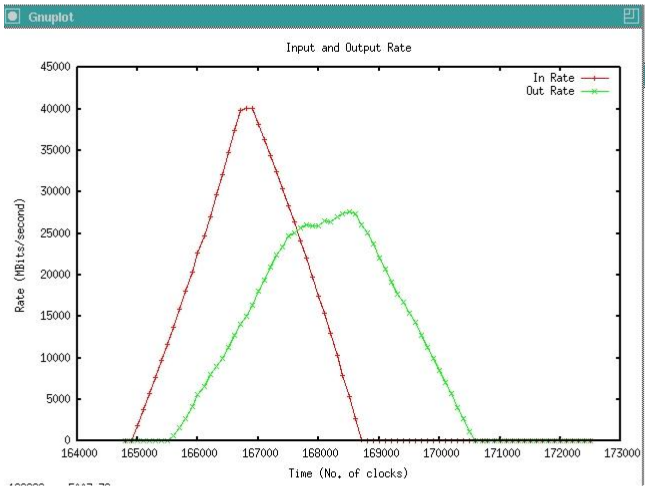

Figure 7: Graph showing Input and Output rates

5.4 Effects of depth of FIFO on smoothness

In our test case FIFO depth of 2000 was chosen based on traffic pattern and data rates. You should choose the depth of FIFO based on your application and simulation environment. The size of FIFO will affect the smoothness of graph. A larger size of FIFO acts as a low pass filter and smoothens out short bursts. A comparison of depth of FIFO versus smoothening is shown below.

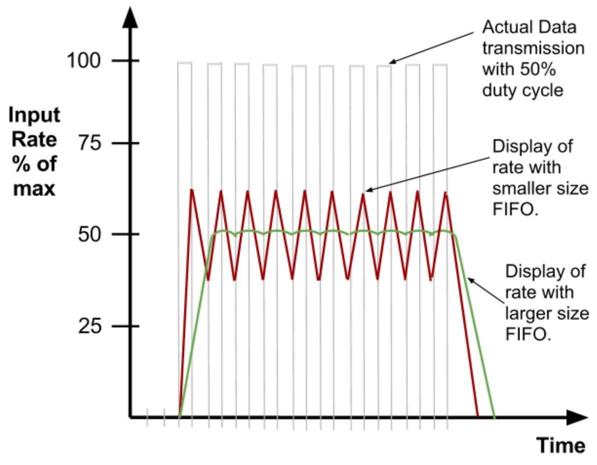

Figure 8: Graph showing effects of FIFO sizing

5.5 Future Enhancements

We outlined the quickly designed mechanism we used in our experiments. In the future we will enhance it with the following features.

Optimization of FIFO bits

The paper shows a simple mechanism of adding number of bits. The width of FIFO can be reduced just by saving number of bytes or even just 1 bit if byte enables are not used.

Hardware implementation

The current implementation is heavy in simulation. It is also difficult to synthesize due to large memory requirements. The implementation can be simplified to an accumulator style design. A suitable mechanism as well as algorithm can be chosen based on the application.

Various algorithms

This paper presents a simple average of moving window. It has the flaw of slow start and decay. The scheme worked for us as we were looking at number of clocks which was much larger than the FIFO.

Sophisticated algorithms like rolling average, weighted moving average, or exponential moving average can be used instead to show better results. These can remove the need for FIFOs and enable us to use faster and smaller designs suitable for hardware implementation.

Auto update mechanisms and usage other tools

The current implementation uses readily available gnuplot package. We need to load the csv file from command line to update the chart. An auto updating chart can be created by using better tools or using Tcl-Tk to show graph real time.