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.

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.

- Throughput
- Latency
- Flow Control

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.

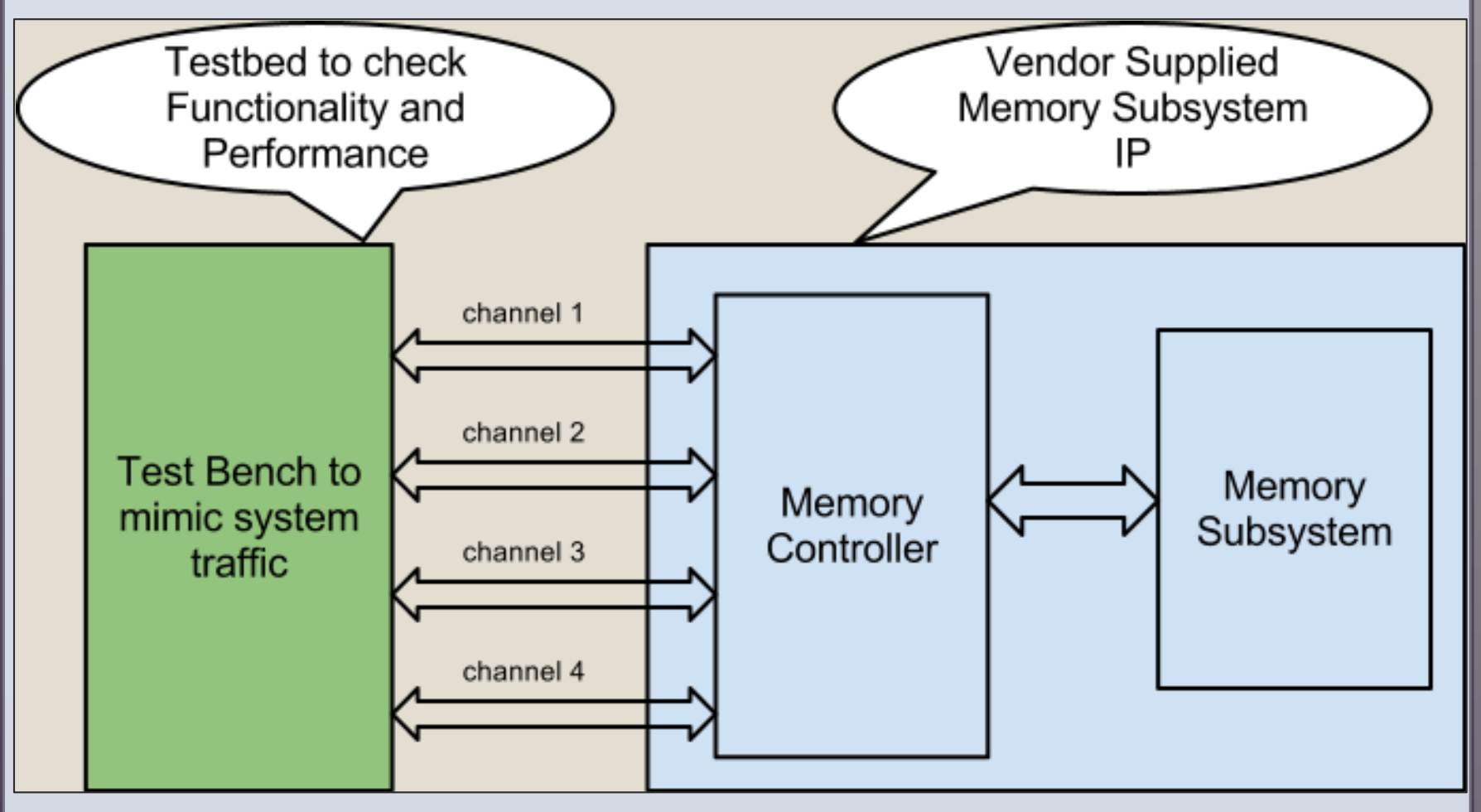
Testbed to check Functionality and Performance
Vendor Supplied Memory Subsystem IP
Test Bench to mimic system traffic
channel 1
channel 2
channel 3
channel 4
Memory Controller
Memory Subsystem

Figure 1: Test setup

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.

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.

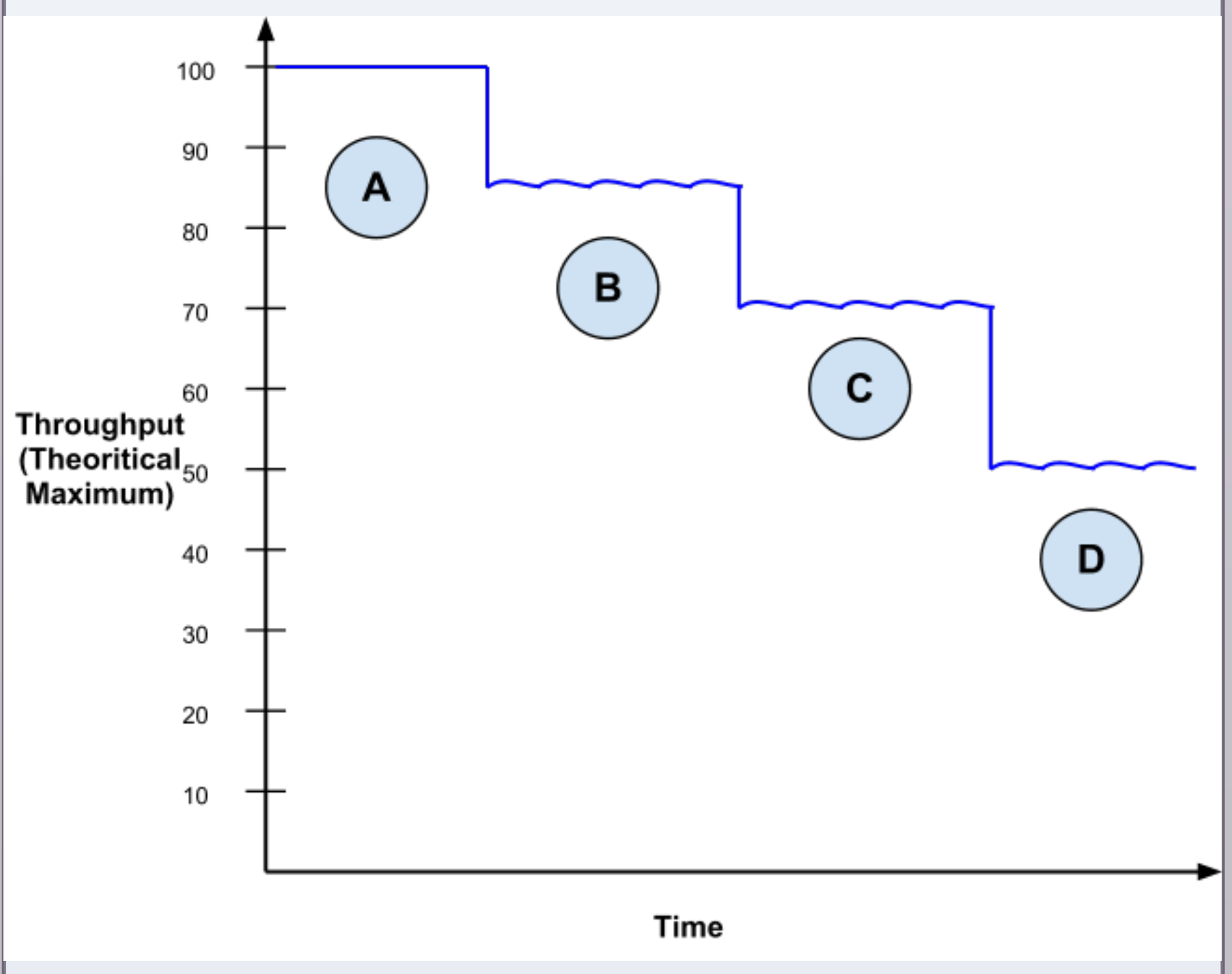

Figure 2: Throughput of memory subsystem under various usage patterns

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.

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.

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.

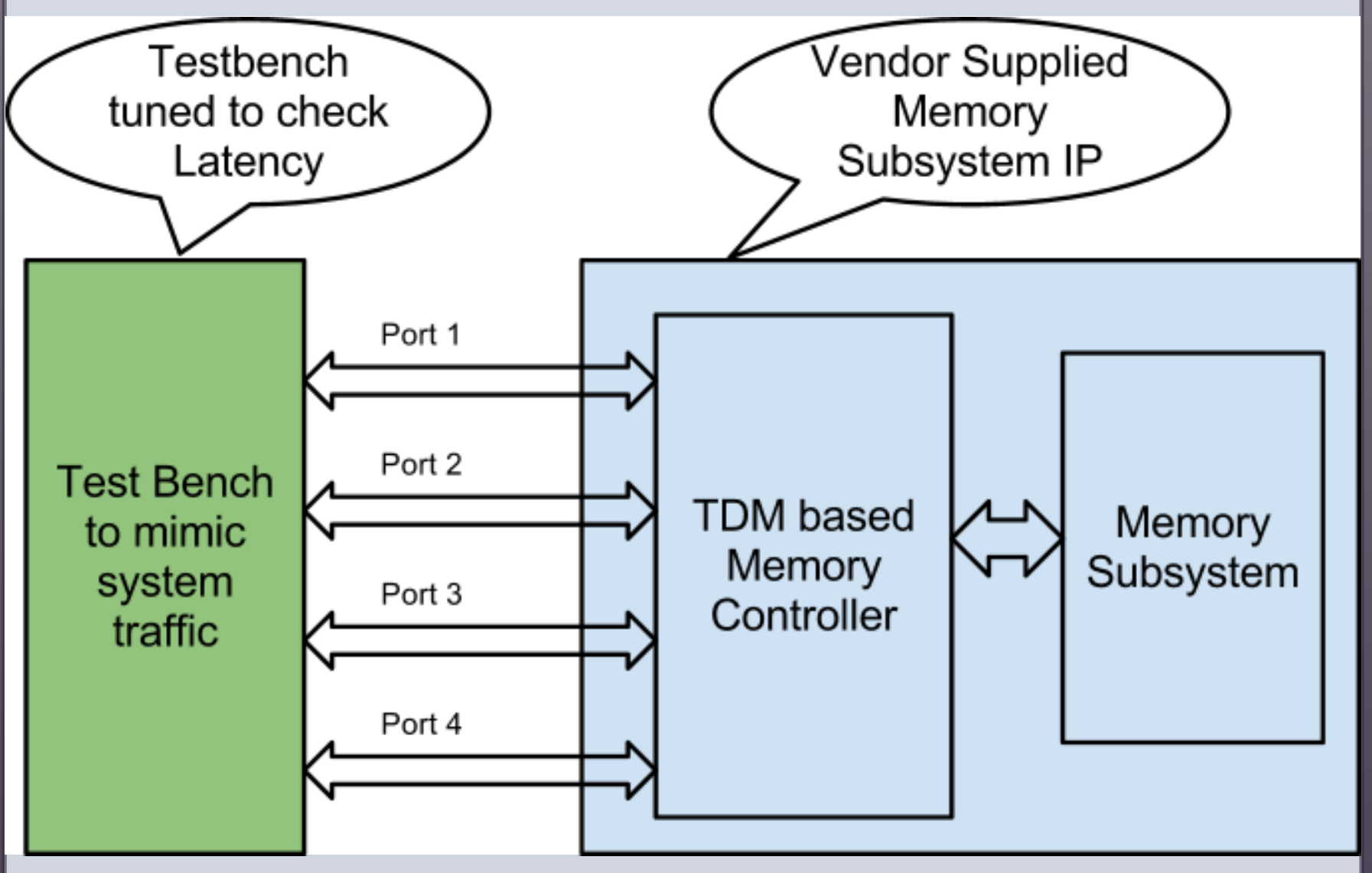
Testbench tuned to check Latency
Vendor Supplied Memory Subsystem IP
Test Bench to mimic system traffic
Port 1
Port 2
Port 3
Port 4
TDM based Memory Controller
Memory Subsystem

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.

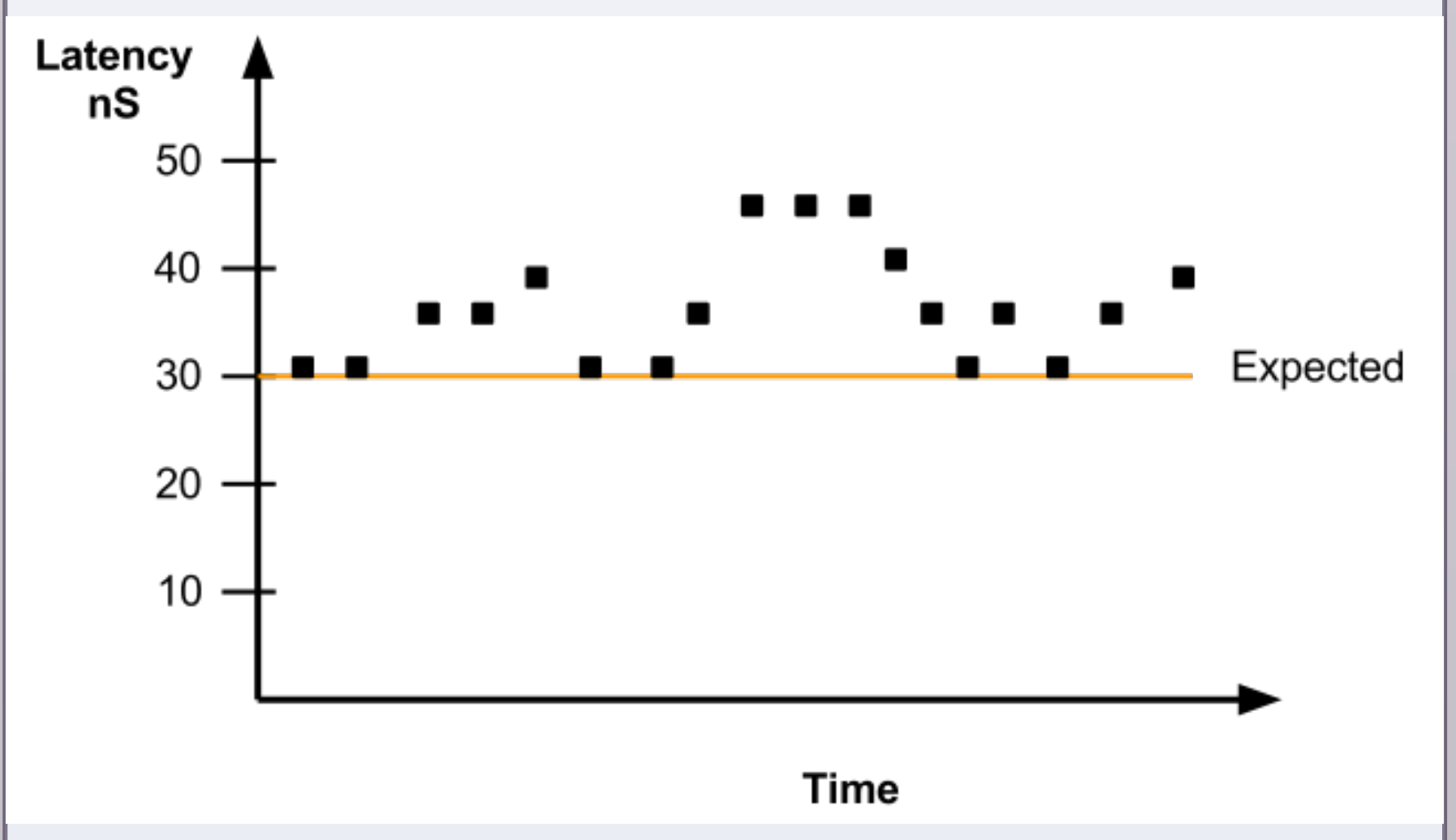

Figure 4: Graph showing observed latency issue

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.

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.

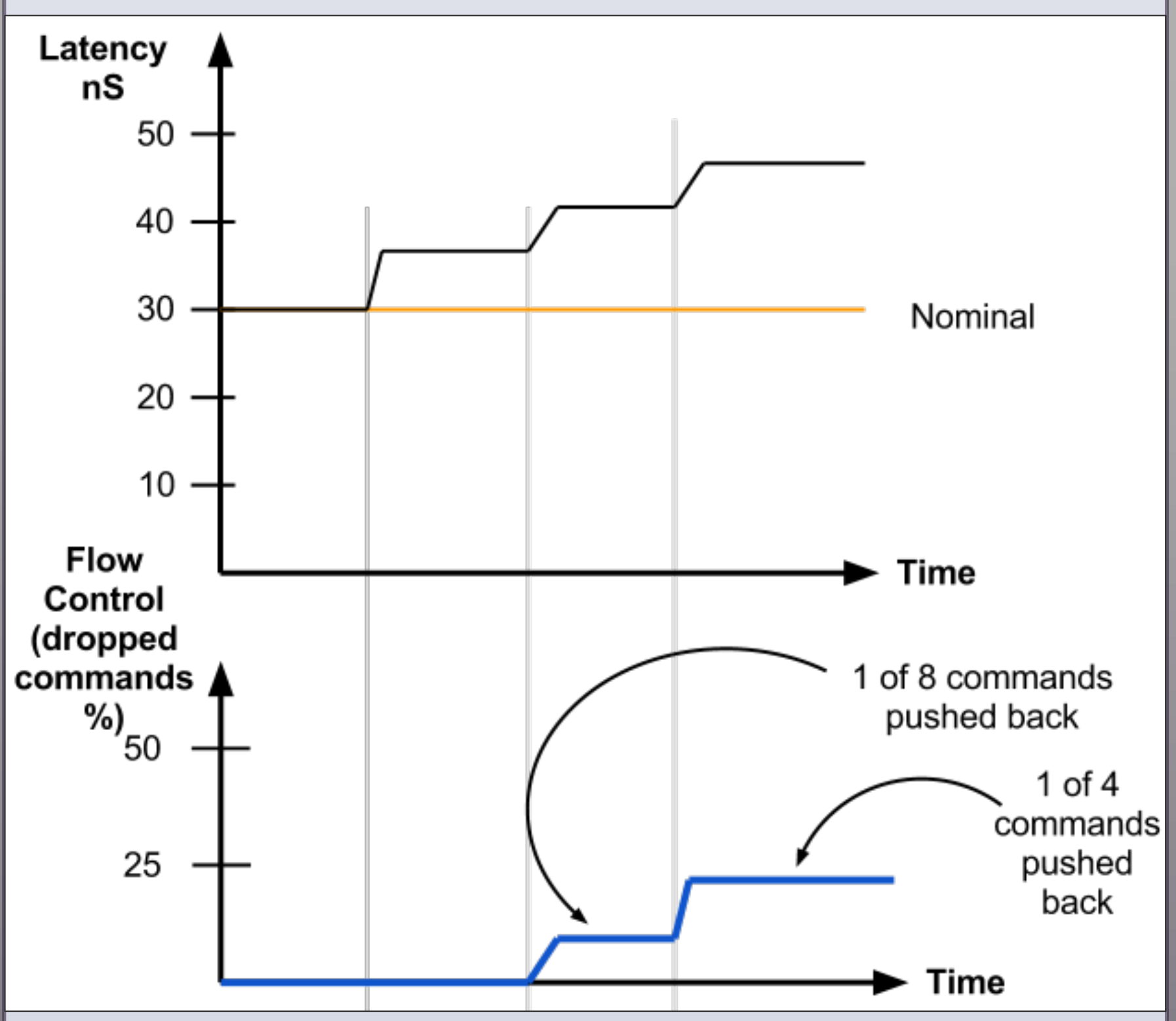

Figure 5 : Graph showing latency along with flow control

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.

Run the test bench which had a reporting and scaling mechanism. It created a smooth throughput number from instantaneous bandwidth measured every clock.

A gnuplot script is run to display data in .csv file graphically.

The details of each block are given below.

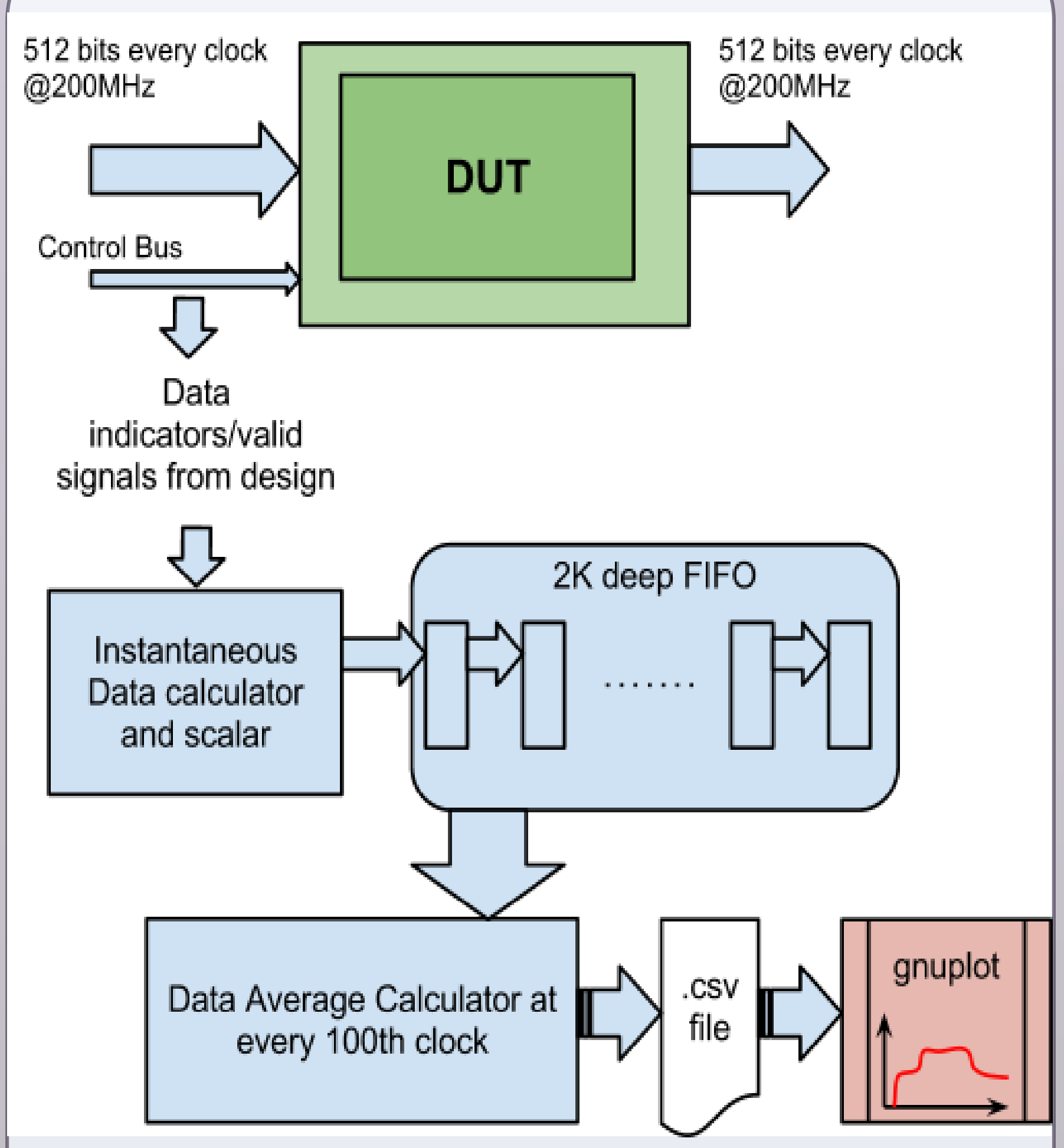
512 bits every clock @200MHz
512 bits every clock @200MHz
DUT
Control Bus
Data indicators/valid signals from design
Instantaneous Data calculator and scalar
2K deep FIFO
....
Data Average Calculator at every 100th clock
.csv file
gnuplot

Figure 6: Block diagram of reporting mechanism

A sample gnuplot

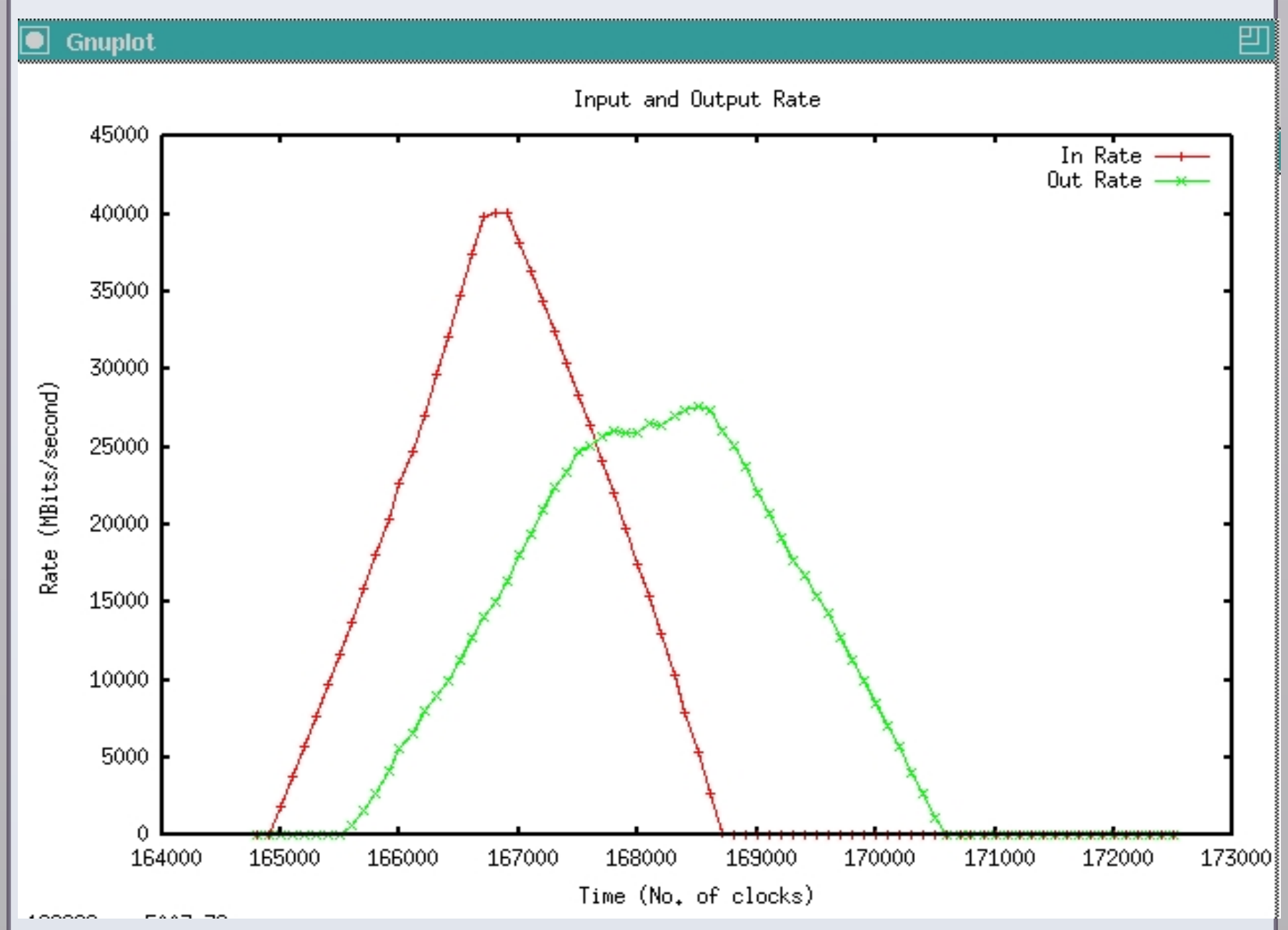

Figure 7: Graph showing Input and Output rates

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.