Test of Decompression module

Meysam Taassori

***Abstract* – This assay is trying to explain the process of test of a decompression module. This module is implemented using Verilog and simulated with Modelsim; all other steps from a Verilog code describing the design to having a chip have been done by Cadence tools. This chip is fabricated in MOSIS 600nm and tested by Tektronics family called LV500. This decompression module is designed to use in memory systems. Since there are a lot of similarities in data is going to be saved in different hierarchy of memory systems, compression and decompression can be useful for these system to tackle one of the most important problems of memory systems, latency of memories.**

I. Introduction

As technology advances, the designers have more transistors on a chip and in turn there are more faster cores on one die; therefore, the number of request to memory is increasing dramatically. On the other hand, memories are implemented and optimized just for capacity and in this situation they cannot cope with this number of fast cores. That is, this introduces performance bottleneck in future multicore systems. In new technologies, the contribution of power consumption in memory systems also is increasing; this side effect is so important to servers where we need so many memory capacities. These huge numbers of memory is so power hungry and this amount of power consumed in memory systems cause some other problems including cost, high error rate and so on. Briefly, the gap between CPUs and memory systems is going to be exacerbated.

One approach to tackle these problems is to use data compression in storage of data. According to some observation, there are a lot of similarities in data pattern resident in memory hierarchy, which make it possible for compression to help improve performance.

We implemented these compression schemes using Verilog to description it in structural level. In next step, this Verilog code is synthesized to provide us with the gate level of this system. This step is followed by placement and routing in 0.6 micrometer technology using Cadence tools.

The rest of this report is organized as follows: First the particular compression and decompression algorithms used in this chip are explained and the output of simulation of this module is depicted in section 3. Then the method of test of this module is discussed and MSA file used to test this module is introduced. In section 5, we are going to explain some difficulties we encountered in the test process.

II. Compression Algorithm

Frequent Pattern Compression (FPC) is one important way of compression to stop storing and sending frequent data. Although the frequent data differs from program to program, it is observed that all-zero bytes are more frequent than other type of bytes in most of programs. As a result, FPC is realized by detecting zero-based patterns and removing redundant parts. More precisely, a 64B cache line is decomposed to sixteen 4B words. Then for each word if it follows the pattern 0000, 000X, 00XY, or XYZT FPC merely keeps <u>00</u>,<u>01</u>X, <u>10</u>XY, or 11XYZT, respectively. Note that the underlined numbers represent bits, whereas the other parts showed by alphabets represent bytes, so FPC might lead to significant savings with a small overhead. In addition to simplifying the decompression process the underlined bits are kept in the first word as metadata. Figure 1 depicts the bitmap in an FPC-based compressed line.

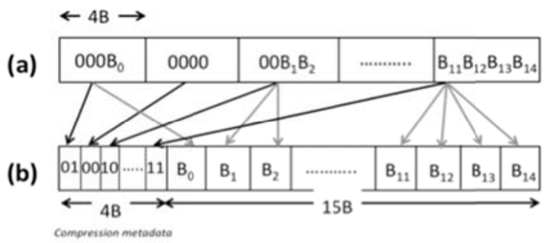

Figure 1: Bitmap of FPC Compression

Base Delta Intermediate (BDI) is a recent proposal that proved to be effective for desktop applications. BDI relies on the fact that the difference between words in one cache is often limited. As such, BDI keeps the difference (delta) of words with one base value instead of keeping whole words. The base value is typically the first word and the delta is typically valued in the range of [-128,127] to fit in a byte. Figure 2 illustrates bitmap for BDI.

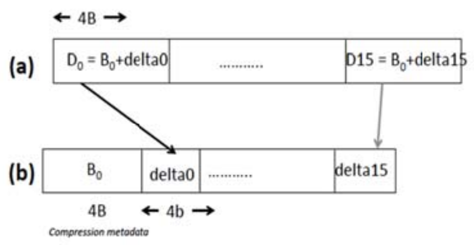

Figure 2: Bitmap of BDI

We use the implementation of FPC and BDI based on proposals in [1] and [2]. Figure 3 shows the block diagram for FPC compression unit.

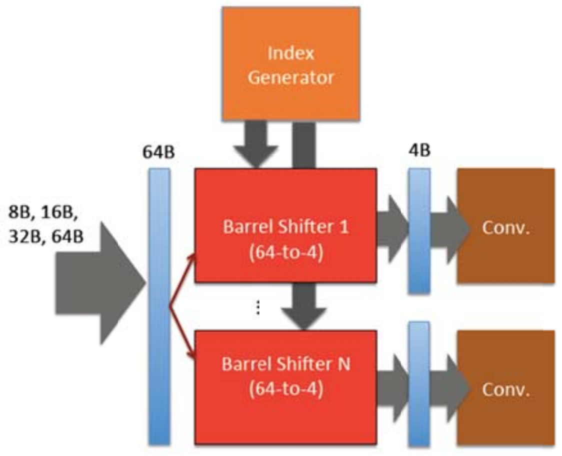

Figure 3: FPC Compression Unit

It mainly consists of three important parts: index generator, barrel shifter, and pattern convertor. The index generator simply generates an index. The index is the position of the first non-zero byte of a word in the compressed cache line, recalling that each word is compressed to 2-bit metadata kept in the first word and zero or more non-zero bytes. As we have 16 words per line, we need to generate 16 indices as well. Since the rate of data arrival is 4B per cycle, these indices are generated by one in each cycle.

The next important part is the barrel shifter. In Cycle *i* the barrel shifter shifts the cache line, *index(i)* position, so that the beginning of the non-compressed part of the word matches the LSB position. So zero, one, two, or four least significant bytes are the non-compressed part of Word *i*. The next component, the pattern convertor, decides how many of the first four bytes after shifting belong to Word *i* by checking the metadata associated with this word. It also fills zero bytes when necessary, again based on metadata.

In [2] the authors considered complete a barrel shifter in their design. We conclude that this is not necessary. In fact, just the first four bytes are needed. Therefore, we modified to Barrel shifter to shift the line and *just keep the first LSB after shift.* This significantly reduced the number of MUXes needed in the barrel shifter. As DBI kept the difference (delta) between each word and the first word, its decompression is simply a bunch of adders to add up the delta and base to find the original word.

III. Result of Simulation

To simulate the structural level model of the design, we use Modelsim. In this stage, we prepare a testbench for this design; our testbench is in charge of preparing different input signals such as clock, reset, and data_in. Then, the output signals will be get by this testbench and we can check it if they are exactly the same as what we expect. Figure 3 depicts the output and input waves of decompression. As shown in figure 3, inputs including “reset”, “clk”, “ready_in”, “number”, and “data_in” are set in Modelsim and the outputs including “data_out” is get from this simulator. “Reset” in this module would be set for several clock period to make sure that circuit is getting started from reset state. After reset is reset, the valid input would be set to module in every period of clock. Since, in this step, we are going to test the behavior of our circuit, timing has no meaning. It is worth mentioning that in this case, no cell has any delay and because of lack of definition of delay, timing is not defined in this step.

IV. Test Methodology

A. *Tester*

The tester we use in this test project is from family called LV500 made by Tektronix in 1989-1991. This tester has been designed in order to test of ASIC design. There are two kinds of tester named LV512, and LV514. Both these testers have many things in common but the most important difference is about test head. Test head means the number of input and output pins which are programmable separately. This tester can test design with a clock whose frequency is up to 50 MHz. this teset can handle 64000 test vectors. Another important specification of this system is ability to connect to network to upload and download the files. LV512, LV54 have 192 and 128 test channels and 12 and 8 sectors respectively.

As shown in figure 4, we design some input vector tests; design of this vectors is so important and we need to do it in such a way that this set of vectors make us sure that this test is as comprehensive as we could and need. The more

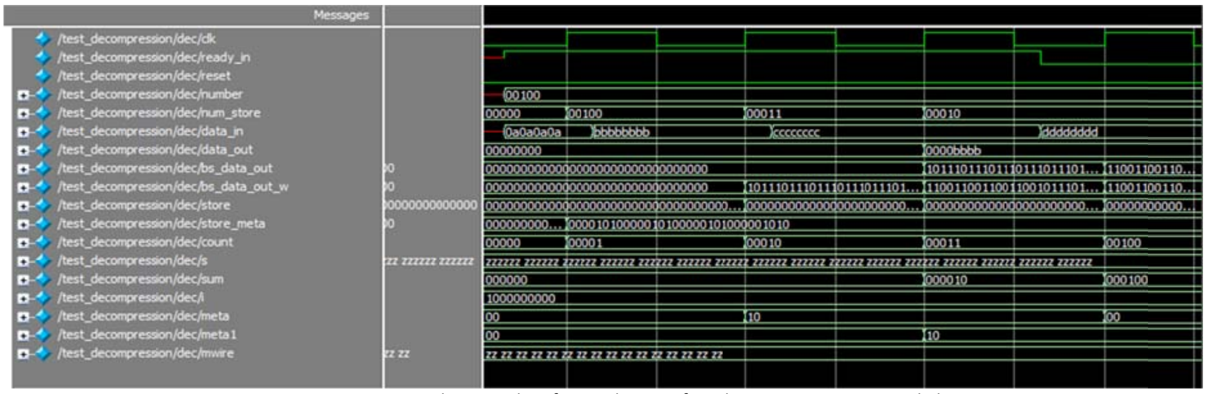
Messages
/test_decompression/dec/clk
/test_decompression/dec/ready_in
/test_decompression/dec/reset
/test_decompression/dec/number
/test_decompression/dec/num_store
/test_decompression/dec/data_in
/test_decompression/dec/data_out
/test_decompression/dec/bs_data_out
/test_decompression/dec/bs_data_out_w
/test_decompression/dec/store
/test_decompression/dec/store_meta
/test_decompression/dec/count
/test_decompression/dec/s
/test_decompression/dec/sum
/test_decompression/dec/i
/test_decompression/dec/meta
/test_decompression/dec/meta1
/test_decompression/dec/mwire

Figure 3 the result of simulation for decompression module

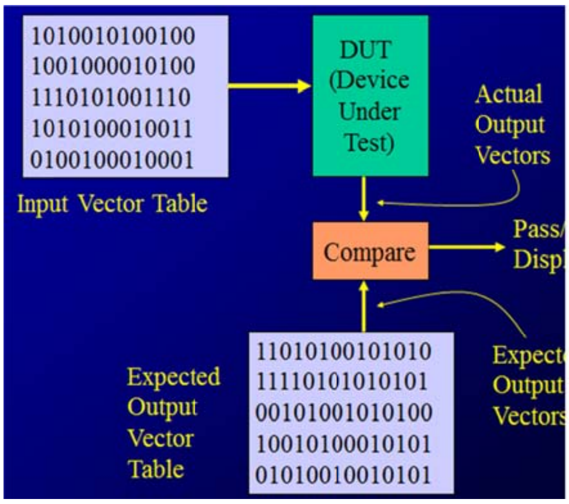

Figure 4 the methodology of test

powerful this set of tests is, the more reliable our design would be. This vectors are given to our design as inputs and the real output would be received by tester. On the other hand, we can define the outputs we are expecting from design. The comparison between real outputs and expected output is another duty of tester. Therefore, if the outputs are the same the design is working correctly; otherwise, there is an error or fault in our circuit. It is evident that to do this test, we need both input test vectors and expected outputs. Both of these prerequisites can be ready just by means of Modelsim. We can prepare a testbench for our design and then define some inputs as the test vector; after testing, Modelsim illustrates the expected outputs.

B. *Essential parts of a test*

Each test we are going to conduct needs some prerequisites; lack of one of these essentials can lead some unexpected problem and consequently no result we are expected would be achieved. These parts are mentioned as follows.

1. Wired DUT (Design Under Test) card: this card is in charge of connecting the pins of tester to input and output of our design called DUT. The mapping between input and output of our design and tester is needed.
2. Tester configuration: we need to configure voltage, current of tester properly. Moreover, we need to set timing of design and tester; it means that we need to clarify when inputs should be applied and when the outputs need to be checked. We need define for every channel of tester either it is "force" or compare"; the former is for input channel and the latter is dedicated to output channels. We should notice to the point that every channel of a tester might not be active and we are allowed to use just active ones. The structure of tester is composed of sectors and each sector contains some channels; each channel is dedicated to one input or output of our chip. For example, LV512 has 16 sectors named 0 to F and each sector has 16 channel; note that only sectors called 0, 3-B are useable. Figure 5 is depicting the card of LV512 and the active sectors are highlighted.

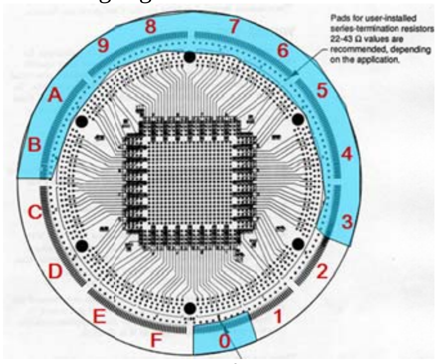
Pads for user-installed series-termination resistors 22-43 Ω values are recommended, depending on the application.
8
7
6
5
4
3
2
1
0
F
E
D
C
B
A
9

Figure 5 DUT Card of Tester LV512

3. A complete set of test vectors: we need to apply these vectors and check if the outputs are equal to what we are expected.

C. *Configuration of test*

In this sub chapter, we are going to explain entire configuration we need to consider before starting a test.

1. Mapping the output and input of our circuit to the sector and channel of tester. In this step, we need to define some groups including all pins of our design. In fact, all pins which needs a same timing can be considered as a group; for instance, we can consider a group named "data_in" for all pins of data and another group called data_out to consider all output pins of data. In each group we need to map the pins of that group to channels of tester. For example, data_out0 is mapped to 9th channel of sector A as follows

```
signal "data_out0" {
                    dut = "j10";
                    sector = 0hA;
                    channel = 0h9;
        }
```

   Each group defined in first step has some specifications including phase, when data should be compared, when data should be forced or even ignored.
2. Defining phase for every group. We can define one phase for every group of pins. Each phase has a delay showing when signal is getting up, another parameter is width depicting how long it would be up, and cycle length showing how long each period takes.
3. Defining the force format: if signal is input, we need to define force format for its group. This format can be chosen from the set of "R1", "R0", "DNRZL", and "DNRZT". This format along with the phase we defined in step 2 can define signals precisely.
4. Defining the compare format: if signal is output we need to compare it with what we expect. Compare format we can choose is compare edge T, edge L, and window;
5. In this step we need to define a template where we set phases and function of each group.
6. Finally, we need to define pattern; the patterns are just different inputs and different outputs based on behavior of our chip, meaning that we need to show what we expect from this system under test.

All these parts of configuration can be mentioned in a file with extension of ".msa". When we run this file in tester all these configuration parts would be done automatically.

V. Result of Test

After preparing msa file, we can start running the appropriate test. The result of tester is mentioned in this chapter. We got the result in two different cases, using logic analyzer, and using software of tester.

A. *Using logic Analyzer*

In this case, we connect the digital analyzer to the input and output ports of our chip which are connected to different channels and sectors of tester. Therefore, we have this opportunity to monitor all input ports and all output ports to make sure that what we have defined in msa file is working well. We assume different force formats for signal of clock and in turn we got some different output signals. In the following figures, channel 0, 1, 2-9, 10-15 are dedicated to "clock", "reset", "output signals" and "input signals" respectively.

1. Clock format is "NRZL"

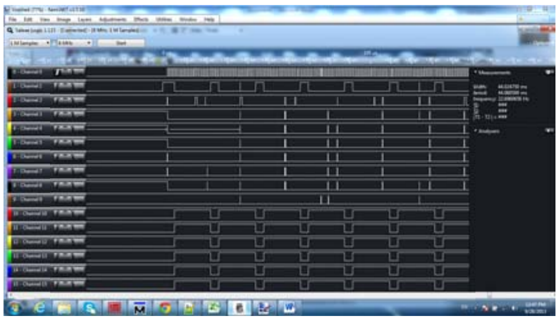

Figure 6 output of logic analyzer in case 1

2. Clock R0 and in pattern it is "0"

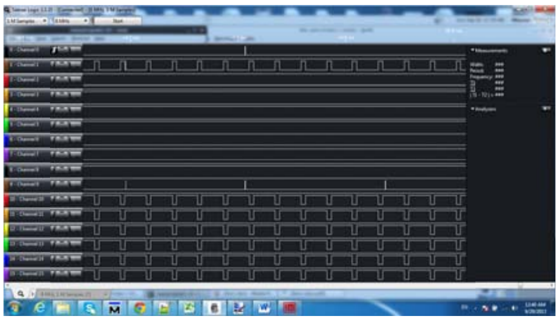

Figure 7 output of logic analyzer in case 1

Figure 6 and 7 shows that output of this chip is strange and it is not what we expected. Although the inputs of this circuit seem to be correct, the outputs are getting stuck at zero or we can see some sparks on that output.

B. *Using software of tester*

After entering the msa file, we configure the tester in such a way that all our patterns have been entered and we can start testing. When we changed the format of clock as mentioned in previous section, there were some differences in the output of system.

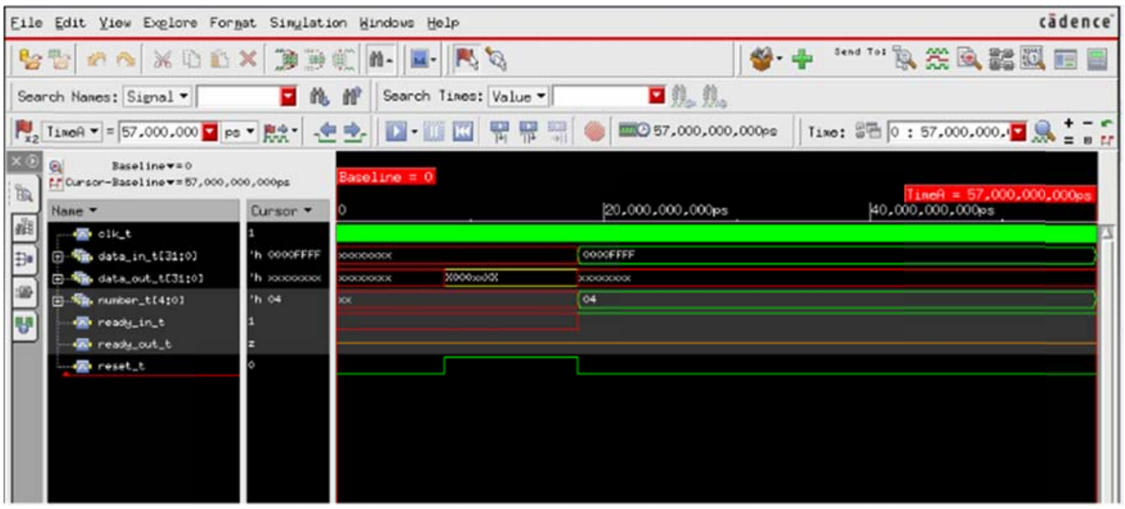

Figure 8 output of file sent fabrication

When we assume that the format of clock is NRZL, all output pins get stuck at 0 and there is no change when the input pins change in our pattern.
These two method of test, using software and logic analyzer, made us sure that the problem might be searched in our design not in configuration of tester. However, all other cases also were tested to make quite sure that this is not because of some wrong in configuration.

Finally, after investigating the files that we have sent to fabrication to have our chip fabricated, I found that mistakenly in those files there are many flip flops that are not connected to either “reset” or any function of reset. Therefore, we have many flip flops whose reset state is not known and they are starting from unknown state.

To make sure that our guess is right, we have conducted a simulation using NC-Verilog with the file we have sent fabrication. The result of this simulation indicates that our guess unfortunately was right and there are some serious problems in our original file. Figure 8 shows the result of this last experiment using. It indicates that even after reset is high for enough period of time, some outputs of our design are still unknown shown with red wave in figure 8.

VI. Conclusion

In this essay, we were explaining how to test an VLSI chip. We described our methodology for a complete test. Furthermore, we introduced the tester from family LV500 in this paper and how to configure this tester. Finally, we showed the results we earned with this tester in two methods, by means of logic analyzer, and software of tester as well. The results were upsetting and showed that our chip does not work properly. To make sure that this problem is not about our tester or any configuration we have set, we conducted another test using NC-Verilog; the result of this test also confirmed that our chip has some serious problems and does not work correctly. Our guess is that this problem is about the file we have sent for fabricating to fabrication.

VII. References

[1] B. Abali, H. Franke, D. E. Poff, R. A. Saccone, C. O. Schulz, L. M. Herger, and T. B. Smith. Memory expansion technology (MXT): software support and performance. IBM JRD, 2001.
[2] A. R. Alameldeen and D. A. Wood. Adaptive cache compression for high performance processors. In ISCA-31, 2004.