High-Level System Modeling and Architecture Exploration with SystemC on a Network SoC: S3C2510 Case Study

Hye-On Jang[1], Minsoo Kang[2], Myeong-jin Lee[2], Kwanyeob Chae[2], Kookpyo Lee[2], Kyuhyun Shim[2]

[1]Samsung Advanced Institute of Technology
Gyeonggi-Do, Korea
hyeon.jang@samsung.com

[2]Samsung Electronics Co., Ltd.
Gyeonggi-Do, Korea
{minsoo.kang , artistic.lee, corean, kookpyo.lee, kh.shim}@samsung.com

Abstract

This paper presents a high-level design methodology applied on a Network SoC using SystemC. The topic will emphasize on high-level design approach for intensive architecture exploration and verifying cycle accurate SystemC models comparative to real Verilog RTL models.

Unlike many high-level designs, we started the project with working Verilog RTL models in hands, which we later compared our SystemC models to. Moreover, we were able to use the on-chip test board performance simulation data to verify our SystemC-based platform.

This paper illustrates that in high-level design, we could have the same accuracy as RTL models but achieve over one hundred times faster simulation speed than that of RTL's. The main topic of the paper will be on architecture exploration in search of performance degradation in source.

1. Introduction

As SoC has started to dominate the ASIC world, the competitiveness of performance and price have increased and triggered the market to move accordingly. Customers request higher performance chip with lower price.

In order to meet such demands, many companies have tried to adopt high-level design methodology to measure and analyze the chip before fabrication or even before RTL development. Not only to adopt or create a new high-level design methodology, many companies also want to have an assurance on the performance after the chip is manufactured. To satisfy all the above and shorten the Turn-Around-Time (TAT), we integrated SystemC[2] into our design flow and applied Transaction Level Modeling (TLM) method.

One of our Network SoC's, S3C2510[1], was already in production. However, the performance of the network SoC did not meet the initial expectation. When the frame was transmitting at 98 Mbps, it was receiving it only at 52 Mbps. It was roughly losing half of its packets. Even though, the current performance was acceptable in the market for the time being, we had to diagnose the performance bottleneck and provide a solution for the improvement.

As many hardware engineers have agonized for many years, it was impossible to explore the architecture once the chip was fabricated. Even in Register Transfer Level (RTL), it was almost impossible to port Real Time Operating System (RTOS) and do many different cases of architecture exploration in limited design time.

Darringer et al.[7] at IBM explained the method for architecture exploration and model validation. Even though they showed architecture exploration in many different areas, they were not able to achieve the accuracy or the simulation speed as we have.

In this paper, we will present a method of architecture exploration in transaction level while keeping the RTL-like accuracy and achieve over one hundred times faster simulation speed than that of RTL's. Moreover, it will show method of RTOS porting and some unexpected problems on a virtual system. Finally, it will explore CPU profiling and bus utilization to identify the performance bottleneck.

2. Preliminary

High-level design methodology requires many stages of preparation and consensus between different groups in each level of abstraction all the way down to Verilog RTL. However, due to lack of known information and TLM techniques, we decided to create a system, which was already verified in RTL. S3C2510 system was in the market and would be the base platform for the later

models. We decided to create the same system with SystemC. Moreover, we gave roughly ±10% of error window to validate the SystemC-based platform.
Once the architecture was chosen, we needed to find the system bottleneck, which was causing the performance degradation. If the problem was found, then we were to suggest optimized architecture or a solution to achieve the maximum performance of 98 Mbps.

In order to solve the problem, we divided the tasks into modeling, system integration, RTOS porting, system simulation, and architecture exploration.

The requirements and specification document was prepared to create SystemC models and modify our RTOS, in this case uClinux[4]. The highest priority of modeling was having the accuracy as close to RTL's. Unlike other high-level design, we needed to make the models as accurate as RTL, because this system would become the base platform for the future developments. Since the SystemC models would be developed in cycle accuracy, we allowed the each model's error window to ±1% of RTL's.

3. Modeling

Our network SoC was mainly used in communication processors. It contained many modules, but we eliminated modules that were not directly related to Network Address Translation (NAT) performance. Figure1 shows the blocks that are only affecting the performance of NAT directly.

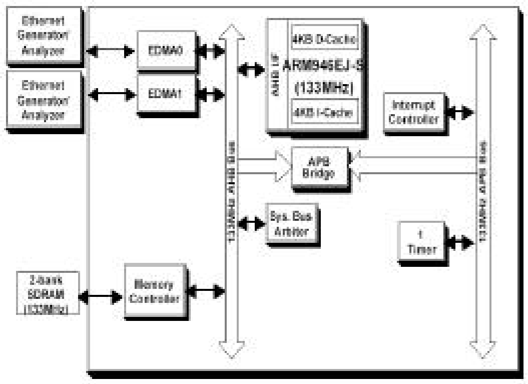

Figure1. NAT performance related blocks of S3C2510

We modeled two-channel Ethernet DMA, SDRAM controller, Arbiter, Interrupt controller, GDMA and timer in SystemC. The ARM processor and the AMBA bus were provided by Synopsys' DesignWare SystemC library[3]. CoCentric System Studio[3] from Synopsys was used to develop the models and the system as well. Later on, we also used CoCentric System Studio for simulation and architecture exploration.

Each model was developed and verified by engineers who also developed RTL model. Since we had RTL models as a reference, we were able to keep the accuracy of the SystemC models within 1% of the RTL's. To make seven models in total, EDMA, SDRAM controller, Arbiter, Interrupt Controller, Timer, Vector Generator, and GDMA, it took us three weeks to develop and verify. Once the models were completed, we had simple binary codes to test the functionalities of each block. Later, when the system was fully integrated, we compared the each model's signal waveform to RTL's waveform result. Since the models were all cycle accurate, we expected to have the same waveform as RTL's. Once the models were verified, we ran the bridge test. It was a hardware test to see if the system was routed properly. In other words, it was a test to see if the system was operating the right functions. When it was routed, we could measure the routing speed of the system. Figure2 shows the actual test board (SMDK2510) with S3C2510 chip on it.

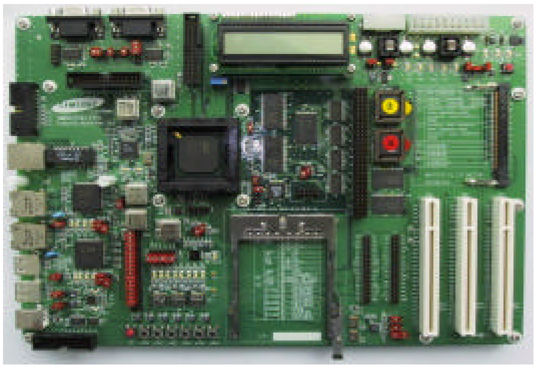

Figure2. SMDK2510 board with S3C2510 chip

Figure3-1 shows the SMDK2510's bridge test result and Figure3-2 shows the SystemC-based platform's bridge test result.

Packet Size (bytes)	Bridge test (Mbps)
1514	93.82
256	88.28
64	N/A

Figure3-1. SMDK2510's Bridge test result

Packet Size (bytes)	Bridge test (Mbps)	Error Rate (%)
1514	94.06	+0.25
256	88.57	+0.29
64	74.37	-

Figure3-2. SystemC-based system's Bridge test result

The bridge test did not use RTOS, but loaded a binary code to test functionality of Ethernet-DMA. In other words, if the whole system was working properly with Ethernet-DMA, we would see whether the system got routed or not, transmitting the frames and receiving them at the same rate.
According to Figure3-2, we could rely on the SystemC models and the SystemC-based platform's accuracy. There were three frame sizes for testing, but 1514 byte frame would be used mainly from now on. Since the error rate of the bridge test was within 1% of RTL's, we were able to trust the hardware models and started to port the RTOS.

4. RTOS porting

The difference between our SystemC-based platform and the RTL platform was the ARM model. RTL platform used ARM940T whereas SystemC-based platform used ARM946E-S. ARM946E-S was able to change the I/D cache size up to 1M bytes, could use Tightly Coupled Memory (TCM) and operated in incremental eight (INCR8) burst. To make ARM946E-S model as close to ARM940T, the I/D cache size of ARM946E-S was set to 4K/4K and the TCM was turned off.
Our system was based on 8-word operation. In order for incremental four (INCR4) to operate in 8-word, it had to run twice. That was because INCR4 only operated in 4-word. The read cycle was set to 5 cycles in SDRAM Controller. Thus, when INCR4 operated once, Read cycle + INCR4 = 5 + 4 = 9 cycles, it would run only 9 cycles. To make the 8-word operation, it had to run twice. Thus, it would be 9 cycles * 2 = 18 cycles. However, INCR8 operated in 8-word. When we applied the same equation, read cycle + INCR8 = 5 + 8 = 13 cycles, it only took 13 cycles to operate the 8-word operation. Thus INCR8 had about 27.8% of advantages than INCR4 in 8-word operation. This was a very rough calculation. If we had considered the write cycles and other operations it would show different figures. Figure4 shows the calculation of the incremental operation.

	INCR4	INCR8
2X	9cycles	
	18 cycles	**13 cycles**

Figure4. Calculation of INCR4 & INCR8

Even though there was 27.8% cycle difference between INCR4 and INCR8, the target performance deviation of the SystemC-based platform was set to ±10% from that of the network SOC. It was set based on the following facts. First, the system arbiter was designed to give the maximum 50% chance to the CPU to access the bus. Second, INCR8 was used only between the CPU and the memory. Finally, more than 20% of the bus resources were idle even for the worst case of incoming traffic.
Once we set the system's performance goal and verified the processor model, we started to make the modifications to the uClinux. The modifications were to comment out unused blocks and to change the processor type to ARM946E-S from ARM940T. Functionalities were not changed.
Once we started porting the RTOS, we did not know what to expect. We thought SystemC-based platform would not be too different than the test board for booting. The problem we faced was lack of communication and experience. The gap between hardware and software engineers was bigger than any of us expected. For example, software engineer needed to know the differences between RTL and SystemC models to make the modifications to the uClinux. However, to hardware engineers, SystemC models were exactly the same as RTL's in functionality. It was minor but turned out there were some differences between SystemC and RTL platform in system configuration, which hardware engineers would never suspect to look into. The gap was the way of seeing the system between hardware and software engineers. They speak different languages and care for different parameters. Moreover, software engineers had never worked on an environment such as this, software simulation for booting. They were used to working on a test board and spend one second to boot the OS to the system. However, in high-level platform, it took more than one second to boot the OS. It actually took us about fifty minutes to boot the RTOS.

5. Simulation Speed

The biggest benefit of using SystemC was in simulation speed. We took an advantage of that benefit.

Simulation Level	Sim. Speed (Cycles/Sec.)
Silicon	133M
RTL	112
SystemC w/ RTOS	18K

Figure5. Simulation Speed

We were able to achieve over hundred times faster simulation speed than Verilog RTL. Figure5 shows the simulation speed in different levels. We believed we could achieve even faster simulation speed depending on the modeling technique.
The hardware system we used was Blade2000 from Sun[6]. It provided 900Mhz CPU with 3Gbytes of memory.

6. Architecture Exploration

Once the RTOS was booted, we measured NAT performance of the SystemC-based platform.

Figure6 shows the NAT performance of the evaluation board of S3C2510 (SMDK2510) and SystemC-based platform. To have the exact measurement comparison at a common point of two systems, the measurement occurred at the point where there was no loss of packets. The evaluation board had no packet loss when Tx. transmitted the frame at 58.6 Mbps. The same method was applied to SystemC platform. When it transmitted the frame at 62.20 Mbps, there was no loss of packets.

With that in mind, the difference of NAT performance between two systems was about 6.14%. Considering the difference in processor model, we thought NAT performance was reasonable and the system was reliable enough to use it further. Figure6 shows the NAT performance of two systems, SMDK2510 and SystemC based platform.

Frame (byte)	**Eval. Board (Mbps)**	**SystemC Pf. (Mbps)**	**Error Rate**
1514	58.60	62.20	6.14%
256	10.50	11.08	5.52%
64	2.68	2.95	10.07%

Figure6. NAT throughput comparison

To do the architecture exploration to find the cause of performance degradation in given environment, we would explore the following areas. Firstly, we would change the I/D cache size, and see how it would affect the NAT performance. Secondly, we would explore the bus to see the utilization and the contention. Lastly, we would run synchronous mode of bus and CPU. Since the bus clock speed was pretty much fixed, we would just increase the CPU clock speed by doubling it. Then we would explore how it would affect the whole system's performance throughput.

6.1. I/D cache size variation

Since ARM946E-S supported I/D cache size from 4K bytes to 1M bytes, we decided to change the I/D cache size and see how it would affect the NAT performance.

ARM946E-S BUS/CPU 133/133			
I/D Cache size	4K/4K	8K/8K	16K/16K
Frame Size(Bytes)	1514	1514	1514
NAT (Mbps)	**58**	**98.6**	**98.6**
Tx (Mbps)	98.7	98.7	98.7
Packet loss rate	**41.2%**	**0.1%**	**0.1%**

Figure7. I/D cache size vs. NAT performance

Shown in Figure7, when the I/D cache size was increased to 8K/8K and above, we got the maximum NAT performance. If it was indeed the I/D cache size, which caused the performance degradation, we wanted to see what was going on inside the cache.

According to the cache profiling, shown in Figure8, we found some changes when the I/D cache size was increased to 8K/8K. Figure8 was the measurement of the I/D cache miss ratio at the same number of instruction was executed at 4K/4K and 8K/8K. At 4K/4K, the data cache read misses ratio was about 9.88% when the data cache write misses ratio was about 14.6%. When I/D cache was set to 8K/8K, the data cache read misses ratio was reduced to 3.4% and the write misses ratio to 3.93%.

I/D Cache size	4K/4K	8K/8K
Data cache read hits	353118	458581
Data cache read misses	**34903**	**15582**
Data cache write hits	212319	330843
Data cache write misses	**31080**	**12995**

Figure8. Cache miss ratio profiling

To see the effect of data cache read misses ratio, we made a simple ratio calculation in Figure9.

Ideal Case	100% hit	
	1	**1. cycles**
4K/4K	90%hit, 10%miss	
(0.9*1)+(0.1*13)=		**2.2 cycles**
8K/8K	97%hit, 3%miss	
(0.97*1)+(0.03*13)=		**1.36 cycles**

Hit: 1 cycle
Miss: 13 cycles

Figure9. Cache miss ratio calculation

In ideal case, we could assume that there would be 100% hit, no cache misses. We set 100% hit as one cycle. In 4K/4K run, since we had about 10% miss, we could assume 90% cache hits and 10% cache misses. As shown in Figure4, it took 13 cycles to operate 8-word. Thus, the equation came out to be (0.9*1)+(0.1*13) = 2.2 cycles for 4K/4K. The same rule applied to 8K/8K. Then the result came out to be 1.36 cycles. As shown in Figure9, 4K/4K took twice as much cycles to operate than that of 8K/8K.

It turned out by increasing the I/D cache size we were able to achieve the maximum NAT performance. However, we were not certain if this was the only cause of the performance degradation. Thus, we started to look into the bus to make sure.

6.2. Bus Utilization

To see the bus contention, we simply looked into how much CPU and other masters were taking the bus operation.

When the I/D cache size of 4K/4K and 8K/8K were compared, the biggest differences were with CPU (write) and E-DMA1. As shown in Figure1, E-DMA0 transmitted data to memory and E-DMA1 received them from the memory. By looking at Figure10-2, we could see that the CPU (write) operation had increased. Doing so caused the E-DMA1 to receive more data than when it was at 4K/4K. When CPU (write) operation increased, it could write more data to the memory. In other words, it executed more data and was able to write back to the memory. Naturally, more data to the memory meant E-DMA1 could receive more as well.

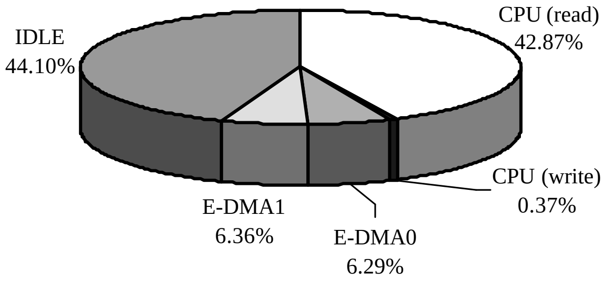

Figure10-1. Bus utilization at 4K/4K

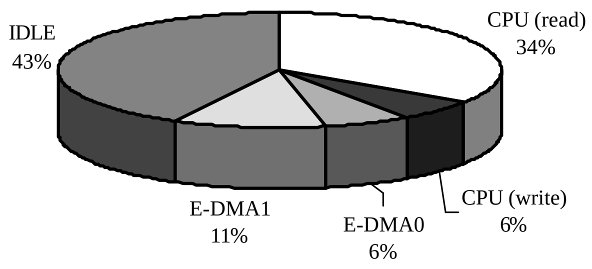

Figure10-2. Bus utilization at 8K/8K

6.3. Bus Contention

We wanted to see whether the bus was affecting the NAT performance regardless of I/D cache size. Thus, we set the I/D cache size at 8K/8K and transmitted frames at the rate of 85 Mbps where there was no packet loss while transmitting and receiving the data. In Figure11, 'Normal' represents S3C2510 as is, 'plus 1 TL' represents S3C2510 with one more master, thus total of four masters. Lastly, 'plus 2 TL' represents total of five masters on the S3C2510 bus. NAT perform for the three cases were all the same, 85 Mbps. Since NAT throughput was the same for all cases, the usage of ETH0 and ETH1 were pretty much the same. Even though by adding two traffic loaders to overload the bus operation, it did not affect the NAT throughput. Thus, we could safely assume that the bus operation did not have much impact on the NAT performance after all.

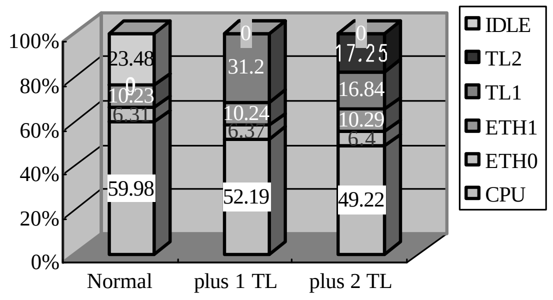

Figure11. Bus Contention by traffic loaders

6.4. Synchronous mode test

Since Bus was not the source of the performance degradation, we wanted to see whether CPU clock speed could affect the performance. We kept the Bus clock speed at 133Mhz and raised the CPU clock speed up to four times of its initial value.

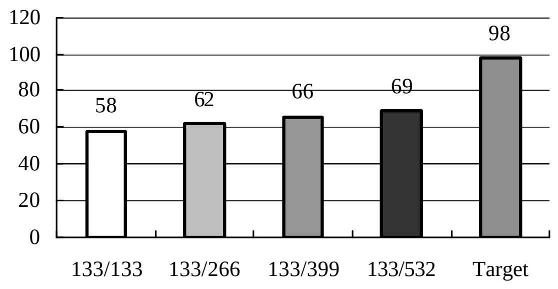

Figure12-1. Synch. Mode at 4K/4K

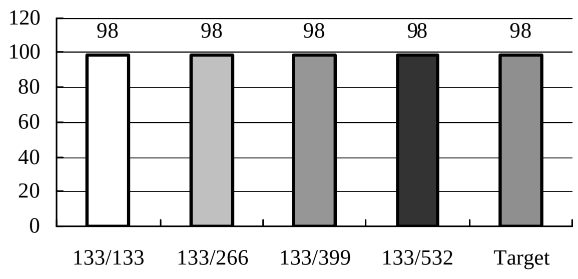

Figure12-2. Synch. Mode at 8K/8K

As shown in Figure12-1, if the cache was set to 4K/4K, CPU clock speed did not affect the NAT throughput much. However, when the cache was set to 8K/8K in Figure12-2, NAT throughput reached the maximum performance of 98 Mbps no matter how fast the CPU clock speed was. Thus, we concluded that it was neither the bus nor the CPU clock speed, which caused the NAT performance degradation but I/D cache size only.

7. Conclusions

In this paper, we presented how we approached high-level design with SystemC, targeting network SoC. Before we applied high-level design methodology, chip designers did not have a way to explore their blocks, not to mention the entire system. However, by applying system-level design methodology with SystemC, we were able to show it could be as accurate as RTL models depending on the modeling technique and could achieve more than hundred times faster simulation speed than that of RTL's.

Most importantly, engineers are now able to explore the systems to find problems and analyze them down to the source.

8. Future Works

Up until now, we only have focused on hardware oriented architecture exploration. However, intensive software, RTOS, exploration is required as well.

In this paper, we only showed the trend of problems and solutions, but we are planning on analyzing why such phenomenon was happening with more precise measurements and explanations.

9. References

[1] http://www.samsung.com/Products/Semiconductor/SystemLSI/Networks/PersonalNTASSP/CommunicationProcessor/S3C2510/S3C2510.htm, 2003.

[2] http://www.systemc.org, 2003.

[3] http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html, 2003.

[4] http://www.uclinux.org, 2003.

[5] http://www.arm.com

[6] http://www.sun.com/desktop/FinalSB2000ds.pdf

[7] J.A Darringer at el., "Early analysis tools for system-on-a-chip design", IBM J. RES. & DEV., VOL.46 NO.6, pp691-707, NOVEMBER 2002