ECE 720-001
Project #1 Report
Ganqin Huang
March 19, 2014

1. INTRODUCTION

Driven by the strong demand for high-performance (but always power-limited) CPUs and SoCs for portable devices powered by batteries, the first decade of this century witnessed extensive development of electronics. With increasing pressure on today's SoC and ASIC design teams to deliver more aggressive designs in less time, a faster design and verification method is required.

While today's Register-Transfer-Level (RTL) design and verification flows are a step up from the gate-level flows of two decades ago, RTL flows are straining to meet the demands of most product teams. Many companies are looking to move to the next level of abstraction beyond RTL form to get a much-needed boost in design productivity. That next level of abstraction is based on transaction level models (TLMs).

Transaction-level modeling uses function calls, rather than signals or wires, for inter-module communications. It lets users analyze transactions such as reads or writes rather than worrying about the underlying logic implementation and timing. As a result, design iterations are reduced because TLM verification is much faster than RTL verification.

Therefore, it is worthwhile to investigate the advantages of this higher-level TLM in comparison with RTL model. In this report, a SoC system based on ARM Cortex is demonstrated both in TLM and RTL models, and the performances of running a Fibonacci program under both models are shown and compared.

The rest of this report is organized as follows. Section 2 discusses the design objectives of the project. Section 3 provides the main goals of this report, and Section 4 discusses the implementation of the Cortex SoC system, detailing how TLM surpasses RTL in simulation. Section 5 demonstrates and discusses the simulation results acquired, and section 6 provides the conclusions.

2. DESIGN OBJECTIVE

The primary objective of this report is writing TLMs to match the delays of the processor, bus, and memory-controller provided in RTL to be functionally correct. In this report, the TLM is implemented in language SystemC, which is based on C++, allowing full reuse of arithmetic C language functions. Based on the same amount of simulation time, efforts are dedicated to accurately model the behavior of the DRAM controller to match the delays of either the CORTEX-driven simulation or the stub-driven simulation.

The secondary objective is optimizing TLMs to get better cycles-per-second simulation performance than an RTL description. TLM uses function calls, rather than signals or wires, for inter-module communications. By making efficient use of time and memory and minimizing the numbers of function-calls and wait-statements, the speedup of TLM over RTL is remarkably raised.

To achieve the objectives, a System-on-Chip platform that includes the Synopsys DesignWare SRAM and static memory controller with the fibonacci.c program running is modeled with growing iterations. The simulation time and cycles per second performance of each simulation are listed in Section5 followed by detailed discussions.

3. MODEL AND DETAILS

To model the RTL and mimic its behaviors, firstly there are several problems to deal with:

1. How each module is connected and how data is transferred from processor to memory
2. What are the number of bits in row, column and bank address for SRAM controller
3. How to model bounty- burst operation, reset and refresh
4. How does data FIFO affects behavior of memory controller

The cortex_soc system in this report of transaction level model is consisted from a Cortexm0, which is provided by ARM System Generator Canvas, an AMBA AHB_Lite bus, and a SRAM module. To mimic the behaviors of DesignWare memory controller of RTL, between SRAM and bus, a memory controller is inserted. More specifically, the CortexM0 processor runs Fibonacci program and send the data through a 32-bit wide AHB bus to the memory controller, then the controller decodes data and further transfer them to the SRAM with respect to various operations and techniques that improves the overall performance. Meanwhile, thanks to the lack of burst operation in CortexM0, an additional stub module is added and switched to provide burst transactions when needed.

TLM uses function calls to model systems. Therefore, all of the contents needed to be transferred on bus, like command type, data, address, length, response type, etc. can be passed within a single member called generic payload provided by TLM2.0. By simply making function calls from initiator sockets to target sockets, the generic payload, as a parameter of block transport function, is passed and the transaction is performed.

As to the address, based on the waveforms of RTL model and the address format provided by Synopsys DW_memctl Databook[1] shown in Figure.1, addresses in TLM are extracted from AHB bus, and then manipulates memory of corresponding location. Considering parameters in RTL model, the 12-bit column address should be [1:12] of AHB bus signal, followed by the 2-bit bank address [13:14] and 13-bit row address [15:27]. The rest of simulations are based on these addresses extracted in each transaction.

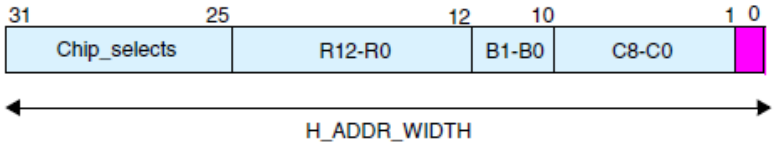

Figure 1. Address Formant

At the beginning of simulation, the reset operation is modeled based on the power-on command sequence in DW_memctl databook, namely t_reset=t_init+trp+t_mrd+t_xsr, where t_xsr is for leaving self-refresh process. For every 1040 cycles, SRAM refresh is executed and all banks are shutdown to refresh each rows with them at the end of this transaction.

Speaking of the behavior of burst reaches bounty, the memory controller firstly gets INCR signal, say INCR4, from the processor, and then transfer this signal to a WRAP4[2] burst in memory side, and each WRAP4 collects two words in two cycles respectively, if bounty is reached, this row will be closed firstly with a RAS delay followed by a RP delay of opening next row, and then command is issued again.

The advantages of FIFO are taken to increase the system performance. When a WRITE command comes, the memory controller puts it into the fifo and returns Ready immediately until the depth-8 fifo is full, in this way the process keeps issuing commands while controller takes care of all following

[1] Synopsys DesignWare SDRAM and Static Memory Controller Databook

[2] AMBA-PV Extensions to TLM 2.0 Developer Guide

commands, and take action in advance by analyzing following address, i.e. to set precharge-bit A10 high to precharge rows in advance.

4. RESULTS AND DISCUSSIONS

Results of simulations for each model are listed below in Table1. Iteration numbers 9, 12, 15, 18, 21 of Fibonacci program are simulated for the CORTEX-driven simulations, and 1000, 3000, 6000, 9000 iteration numbers are taken for stub-driven simulations. The transaction source files for stub in TLM comes from python-script-processed dump files of RTL which are optimized for address and size.

		CORTEX					STUB			
Iter		9	12	15	18	21	1000	3000	6000	10000
Cycle	TLM	70580	141127	391382	1373812	5456407	19474	56799	112562	186865
	RTL	72504	143602	389751	1372565	5475157	19563	57097	113292	188592
Diff		-2.65%	-1.72%	0.42%	0.09%	-0.34%	-0.45%	-0.52%	-0.64%	-0.92%
Instr num	TLM	11952	23499	63210	221416	881444	1000	3000	6000	10000
	RTL	10331	20828	57577	204913	820599	1000	3000	6000	10000
CPI	TLM	5.91	6.01	6.19	6.20	6.19	19.47	18.93	18.76	18.69
	RTL	7.02	6.89	6.77	6.70	6.67	19.56	19.03	18.88	18.86
Proc_ time	TLM	1.05	1.18	1.58	3.17	9.78	0.75	0.77	0.79	0.82
	RTL	8.84	16.93	44.7	155.6	618.21	3.641	7.273	12.691	19.894
Cycles /sec	TLM	67219.05	119599.15	247710.13	433379.18	557914.83	25965.33	73764.94	142483.54	227884.15
	RTL	8201.81	8482.10	8719.26	8821.11	8856.47	5372.97	7850.54	8926.96	9479.84
Ratio		8.20	14.10	28.41	49.13	63.00	4.83	9.40	15.96	24.04

Table 1 Results of CPI and Cycles per second of TLM and RTL

From table above, we could see a big breakthrough of TLM over RTL. Overall, the TLM is dozens of time faster than RTL, especially when iteration increases. The TLM not only accurately modeled all the delays in RTL which achieved remarkable results, but also modeled sophisticated logics like refresh, reset and other components which are less dominant in accuracy but impedes growing performance.

5. IMPROVEMENTS

To improve the speed of TLM, a bundle of modifications are made like optimizing algorithms, reducing arithmetic operations, rearrange data and wires, minimizing function calls and wait statements. For example, various delays are integrated into one data, like same_row_delay, in the constructor of memctl module to minimize the arithmetic operations to only once and reduce the number of calling wait statement. Likewise, different scenarios where delays match are combined together and simple tricky logics with almost same functions are used to model complicated parts like fifo. The drawbacks of this project are lack of detailed modeling like early-burst-termination operation. Though these details make TLM more accurate, they decrease the performance on the other hand.

6. CONCLUSION

This document has presented the format for the ECE 720-001 project1 report. Details on the design-objectives were given. The detailed model of the report was presented, stating that the problems and solutions in depth. The address-chosen method is stated and the CPI and cycles/sec performance of simulation are demonstrated and discussed. Also, techniques adopted to maximize performance are mentioned. Overall, thought sacrifices of performance are made to implement details functionalities like refresh and fifo, the TLM accurately modeled the components in RTL model