A Comparative Study of Modeling at Different Levels of Abstraction in System on Chip Designs: A Case Study

Suryaprasad Jayadevappa, Ravi Shankar, Imad Mahgoub
Department of CS&E, Florida Atlantic University, FL, USA.
({sjayadev, ravi, imad} @ cse.fau.edu)

Abstract
Abstraction is a powerful technique for the design and implementation of complex systems. A model developed at a higher level of abstraction allows one to tackle complexity by initially hiding the details and elaborating them later. A higher level of abstraction typically has a positive effect on the simulation speed and ease of development of the model, but could affect the accuracy of the model developed. In this paper, we study the effect of model abstraction of a peripheral device developed at a higher level of abstraction using SystemC, and at the register transfer level using Verilog. The parameters compared are accuracy, simulation speed, flexibility, time to develop, code length and ease of verification. In our study we show that by raising the level of abstraction, one not only achieves better simulation speed, flexibility, ease of verification but also reduces time to develop and shorten code length. All this is achieved while being able to maintain almost the same accuracy.

1. Introduction

Present day technological advances provide us with the capability to integrate more and more functionality into a single chip. This has led to new design paradigms including, System-on-Chip (SoC), System Level Design, and Platform-based Design (PBD). In SoC designs all the functionality of a system is captured on a single chip, leading to increased performance, reduced power consumption, lower costs, and reduced size. SoC design brings with it new challenges and difficulties. The designs are large, complicated, and involve software and hardware components. These designs have to be modeled at a high level of abstraction before partitioning into hardware and software components for final implementation.

From the hardware (HW) design point of view, hardware description languages (HDLs) such as Verilog and VHDL, in conjunction with hardware simulation and synthesis tools, have proven highly valuable. Efforts have been made to extend these approaches to work for multiprocessor-based SoC design, but these tend to require a significant amount of information about the partitioning of the system into coarse-grained blocks at the start. With the increased complexity new approaches seem inevitable.

We are at a transition stage similar to the move from schematic-based design to hardware description language (HDL)-based design. A few years ago, just the notion of a new language (such as C) for hardware design, raised technical and suitability issues. Hardware engineers accustomed to using HDLs find it difficult to accept that a software development and modeling language such as C/C++ could be useful for hardware design.

To accelerate hardware design, designers use software models of the hardware that they build. These models developed at high level of abstraction are used to validate the functionality and evaluate performance. SystemC a relatively new language, is aimed at facilitating model development above the register transfer level (RTL) [1,2]. SystemC 2.0 provides the ability to capture designs at various levels of abstraction. The single language solution to express designs at different levels of abstraction makes SystemC language a strong contender for system level design language. Built around the C++ language, SystemC inherits the reputation of C++ as a multi-paradigm language, and offers additional capabilities for HW design. SystemC is close enough to C++ to develop SW intellectual property (IP) blocks, and is well suited for developing HW soft IP’s.

Earlier to SystemC, a typical design at system level was created using C/C++, Matlab, Spreadsheet or some variant. Next these designs had to be manually translated into RTL to capture the architecture. This conversion would typically lead to many design errors making previous work unfit for reuse. SystemC provides various features to perform system level modeling and simulation, which are missing in the generic HDL’s such as VHDL and Verilog. But SystemC is not intended to replace Verilog or VHDL. Supporting models at various levels of abstraction is a key feature in SystemC. Synthesizability of the SystemC models currently lacks commercial tool support.

In our current work, we discuss the effect of modeling at different levels of abstractions using SystemC and
Verilog. We start with an initial discussion of current related literature in Section 2. In Section 3 and 4 we discuss the effect of abstraction with respect to accuracy, speed of simulation, code length, ease of verification and flexibility. We have modeled a peripheral device, the watchdog timer both at the transaction level using SystemC and at register transfer level using Verilog. In Section 5, we discuss the results and in section 6 we provide conclusions.

2. Related Work

Currently there is very little published literature that compares modeling design abstraction in two different languages making our work unique. Though there is some published material on comparing Verilog and VHDL it typically involves designs modeled at a similar level of abstraction. Also, there has been very little effort to compare Verilog and SystemC. The comparisons have been restricted to the language features including syntax currently available on the web [3].

Gerstlauer et al. [4] discuss system level abstraction semantic. Their focus has been on using SpecC to show behavioral and architectural modeling of a JPEG encoder. Verkest et al. [5] discuss the usage of C++ for system level design. They have considered system level design to involve two activities: (a) IP creation, involving IP implementation and refinement from an abstract algorithm to RTL design, and (b) IP integration, which ensures correct IP interaction with its environment. To support HW design, Pasko et al. [6] consider an extension of C++ based design methodologies with meta-language concepts. Doucet et al. [7] use C++ for describing components, and a scripting language for performing the dynamic composition of components in a design. Villar et al. [8] discusses the design of embedded system using SystemC. Charest et al. [9] demonstrate the benefits of the multi-paradigm design using SystemC for HW design. Zhu [10] proposes an extension to RTL named Meta-RTL to raise the level of abstraction in current RTL languages.

As reported by Mokhoff [11], transaction-level modeling (TLM) being the topic of discussion at Design Automation and Test in Europe (DATE ‘03) conference as a possible answer to some of the design and verification challenges of complex system-on-chip design. TLM is a powerful technique that allows the designer to model the communication infrastructure of a design and to experiment with and evaluate different communication architectures. TLM is considered as a possible successor to traditional methodologies based on register-transfer-level (RTL) descriptions. They also discuss how modeling at transaction level could speed the design process and lower the risk of design failures, and the implications of this on the tools, languages and intellectual property (IP) used in the design/verification process.

3. Design Abstraction in Modeling

Dealing with complexity by various design representations is prevalent throughout engineering. These representations appear in different disciplines as: subsystem decomposition, objects, functional decomposition, hierarchy, and component subassemblies. In addition, most disciplines use simplifying assumptions in order to make analysis tractable. Such use of layering or simplification is referred to as "abstraction". Abstraction involves using simplified, high-level representations of designs as a way to master complexity. Figure 1 captures the different levels of abstraction as seen in hardware system design. As the level of abstraction increases the ability to support system designs of higher complexity gets better.

Designing the most appropriate system architecture is the key to an efficient implementation of complex algorithms such as those found in advanced applications. Over the years, it has always been recognized that raising the level of abstraction at which the creative work is done is the most effective method for improving design productivity. Today's design methodology for embedded systems is often platform based [12]. Finding the best algorithm/architecture trade-off is important for this design technique.

![Figure 1. Levels of abstraction in hardware modeling](image_url)

In our study, we compare a peripheral model developed at transaction level with that at the RTL. These levels are shown as the higher two layers in Figure 1. TLM can be defined as a high-level approach of modeling digital systems where details of communication among modules are separated from the details of the implementation of the functional units or of the communication architecture [1]. At transaction level one is more interested in the functionality of the data transfers and less in the actual implementation. In a TLM though the model developed would be clock accurate, they tend to simulate faster than the RTL model due to less implementation details. One can also perform architectural explorations when support for common interfaces is available and the modules are suitably parameterized. With the increase in level of abstraction, the accuracy is expected to deteriorate but simulation speed is expected to improve. Modeling of fewer details could be attributed as one of the reason.
Details as observed in our study are discussed in subsequent sections.

4. Modeling - Transaction level and RTL

Watchdog timer (WDT) is an important peripheral device typically used to detect a hung processor. It is used in various Embedded Control Systems; Critical Microprocessor (µP) and Microcontroller (µC) Monitoring; Industrial Controllers; Automotive; Telecommunications; and Networking systems. When enabled, the WDT will count down or “times out” after a predefined period. A time-out will force a device RESET, Interrupt or enable an external signal. To prevent a WDT time-out Reset, the user must periodically re-initialize the Watchdog Timer status. Programming the registers of the WDT suitably enables it to output the appropriate signal after timeout and also allows one to vary the duration of timeout.

The watchdog timer model was developed at two different levels of abstraction namely, transaction level and RTL. The TLM is developed using SystemC and the RTL model is developed using Verilog. The complete code of both SystemC and Verilog can be downloaded from our website [3]. The SystemC TLM developed is clock accurate and the Verilog model developed is clock and pin-accurate. The comparative study of the SystemC and Verilog models is based on the following parameters:

- Accuracy: Actual v/s the expected output.
- Simulation Speed: Time duration taken to simulate the model.
- Time to Model: Total duration involved in developing the peripheral model.
- Code Length: The actual number of lines of code in SystemC and Verilog.
- Ease of Verification: Effort necessary to test or verify the model developed.
- Flexibility: Ability to integrate the model developed into larger system.

4.1. Accuracy

In our context, accuracy is measured as the ability to model the required functionality correctly. When the timer counter value is loaded with 0x0F and divide-by-8 counter value used, the output signals will be set at 120 clock ticks. Clock accuracy is achieved by enabling the processes sensitive to either positive or negative edge of the clock. SystemC has constructs such as `sensitive_pos<<clock`, that can be associated with processes. This is very similar to the `@ (posedge clock)` construct available in Verilog. We tested the SystemC TLM model and the Verilog RTL models with the timer counter value set to 0x0F and programming the control register to divide by 8 value. Models developed in SystemC and Verilog set the appropriate output signals at 120 and 122 clock ticks, hence exhibiting almost similar functional accuracy as shown in Figure 2 and 3.

The difference in clock cycles seen in the Verilog model output is due the extra cycles taken for setting the necessary initial state. Though both the TLM and RTL models were clock accurate, only the RTL model is pin accurate. The pin accuracy is due to the implementation detailing available. The address, data and control signal sizes are explicitly laid out to ensure that the model is ready for synthesis.

![Figure 2. WDTimer Waveform trace output for SystemC TLM code](image-url)
4.2. Speed

The speed of simulation would typically be higher at a higher level of abstraction. The main reason for this would be due to fewer details of the model. Another reason for faster simulation speed could be attributed due to the reduced usage of concurrent threads. This was possible by using simple functional calls instead of signals for transferring address, data and control. In the SystemC TLM, only the counter and the modules controlling the output signal are modeled as concurrent threads and need to be sensitive to the clock. The remainder of the modules communicate using simple function calls. The Verilog RTL model consisted of 14 concurrent threads in three sub-modules (Counter, Prescaler and Controller). All the three sub-modules were sensitive to the clock. Hence the simulator had to keep track of changes in all these sub-modules for every clock-transition. But the advantage with the RTL model is that it is synthesizable. Currently there are tools in place which are automated to support synthesis for models described in Verilog, whereas such a tool is absent for SystemC. To simulate the watchdog timer with the simple testbench it took around 4.0 seconds for the Verilog register transfer model running LDV on Red Hat Linux 7.2. Whereas for a similar testbench for SystemC transaction level model running the OSCI simulator on VisualC++ 6.0 it took a little less than 3.4 seconds. A more detailed and exhaustive testbench would have brought out the differences in simulation timing to a larger extent.

4.3. Time to Model

Time spent in developing a model typically starts from the specification stage and includes time spent on understanding the textual specification, coding, debugging and verification. We kept track of the time we spent at various stages in developing these models at the two different levels of abstraction. There was a learning curve involved in learning SystemC and transaction level modeling, as we were new to it. This time was compensated by the initial time we spent in studying the watchdog timer model description before starting to develop the Verilog RTL model. Further our expertise in C++ and knowledge of Verilog helped us in ramping up our time in developing the SystemC model. The total time includes the time spent for studying the functional specification of the watchdog timer, developing the model in SystemC and Verilog, debugging and finally testing. The time spent on learning SystemC and TLM was included to the time involved in developing the SystemC model. For our study we have considered a day’s effort as eight hour. It took us seventy three hours to develop and test the SystemC TLM model when compared to ninety hours for RTL Verilog model.

4.4. Code length

Quite often code length is used as a factor while comparing two languages. Though we are using two different languages for modeling the same functionality, we have used them to model at different levels of abstraction. Our intention is to study the effect of code length based on the level of abstraction rather than the language used. The SystemC TLM for the watchdog timer contained 375 lines of source code. This included both the header and source file in SystemC. Verilog RTL model code was split among three files for the different modules (prescaler, counter, and controller) developed and contained 450 lines of code. All comments used to improve the readability of the code were not considered in the code length count. The driver and
monitor code that were developed to test the model were also not considered towards the code length.

4.5. Ease of Verification

Verification or testing is needed to prove the functionality and correct working of the model being developed. Also studies indicate that verification or testing typically takes almost 50% of the total time allotted for the model development. To verify the functionality of the SystemC TLM and Verilog RTL models the procedure was very similar. We had to first create the appropriate top-level netlist which had the model instantiation and port binding. Once this netlist is created, the appropriate driver and monitor functions were created. Developing the driver and monitor for SystemC transaction model is simplified due to the use of transactions. The verification test bench created contained reads and writes to the specific registers. This was provided in the `sc_main()` where the tracing was also included. A small snippet of the SystemC top level code used for testing is shown in Figure 4.

The Verilog testbench being a bit more detailed needed suitable initialization of the model. This involved ensuring suitable sensitivity of the various signals involved on the necessary clock transition. Figure 5 provides the Verilog code snapshot used for verification. It is used along with the top-level netlist which had to be created anyway.

Testing in Verilog was intended for functional purpose which was sufficient to test the module correctness. Additional test code was included for a more exhaustive verification.

```
#include "systemc.h"
#include "wdt.h"
sc_main(int argc, char *argv)
{   unsigned int address = 0x00000000;    unsigned int data = 0xABC00F ; int   rdwr = 0;
    sc_signal<bool>       RESET;   sc_signal<bool>      Interrupt;   sc_signal<bool>      External;   sc_signal<bool>    nRESET;
    sc_clock CLK("CLK",10, SC_NS);  // Clock Instantiation
    wdtimer wdtimer1("wdtimer::wdtimer1");  // WDTimer Module Instantiation
    wdtimer1.p_n_reset(nRESET);      wdtimer1.clock(CLK);    wdtimer1.ResetSignal(RESET);
    wdtimer1.InterruptSignal(Interrupt);     wdtimer1.ExternalSignal(External);
    wdtimer1.transfer_action(address,&data,rdwr);   ………  }
```

**Figure 4. Test Module of WDTimer Code Snippet in SystemC.**

```
`timescale 1ns/1ns
`define clkperiod 10
module top;
wire WDT_EXTRN;   wire WDT_RST;    wire WDT_IRQ;     reg clk; reg sel ; reg enable; …
    wdtimer wdtimer1      // Module Instantiation Top Level Netlist
(   .pclk  (clk),  .p_n_reset  (p_n_reset),  .psel   (sel), .penable  (enable),  .pwrite  (write),    .paddr   (addr),  .pwdata  (wdata),
    .cpu_debug   (cpu_debug),  .prdata  (rdata),     .ex_wdz   (WDT_EXTRN),  .wd_rst    (WDT_RST),  .wd_irq  (WDT_IRQ));
initial          begin
    p_n_reset = 1;        sel = 1;           write = 0;       enable = 0;          addr = 6'h0F;     clk = 0;      end
always     forever #hlfclkperiod clk = ~clk;  //clock creation
initial begin:stimulus //Setting Initial conditions
    #clkperiod p_n_reset = 0; #clkperiod p_n_reset = 1;
    write_register(zmr_addr, {12b0,12hABC,3'b000,2'b11,3'b100,4'b0010});
    write_register(cr_addr, {14b0,12h248,4'b0000,2'b00});
    #100000 $stop;     end  ….  endmodule
```

**Figure 5. Test Module of WDTimer Code Snippet in Verilog.**

In the respective top level modules, a general clock with similar frequency was created, then it was ensured that the correct signal or transaction values were set so that the functionality of the WDT could be verified. Though the testbenches created for testing both Verilog RTL and SystemC TLM were simple they were sufficient to prove the functional accuracy of the models developed.

4.6. Flexibility

Given an abstraction-based view of design, it is possible to consider different ways to address the problem of design for change. A simple approach is to execute the design process so as to isolate likely effects of changes into single subsystems or components. While manual redesign of single subsystems is a good start for dealing with change, it may be possible to accomplish even more.
In some disciplines, synthesis tools are becoming available that do direct synthesis from a higher level description. The design abstraction framework suggests that design for change can be accomplished by having designers work at a higher level of abstraction than the actual system components, then let synthesis tools create the lower layer of refined designs. Clearly this will result in a tradeoff of design cost and speed vs. efficiency of implementation. This must be evaluated on a case-by-case basis.

A complete SystemC framework developed at the transaction level consisted of a simple bus module which is freely available for download [2], a master for generating the read and write transactions, a simple memory, and the watchdog timer as shown in Figure 6. The SystemC top level netlist is created for the simple architecture. All the models were at transaction level facilitated by the simple bus model. Due to the use of interfaces and transactions it was very easy to integrate the watchdog timer model into this framework. It took us less than thirty hours to develop and test this simple framework. Majority of the time was spent on understanding the simple bus structure and its interfaces for master and slave. Though we did not develop a similar framework for the RTL model, our past experiences and various studies indicate that developing such a framework would need a group effort with sufficiently large amount of time being spent not only in developing the various models, but also in integrating and testing them.

The fact that all the sub-modules within the model need not be updated on every clock cycle unlike in Verilog, hence decreasing the simulation time due to lesser overhead on the simulator. Simple function calls replaced signals and SystemC data types replaced hardware specific data types. Figure 7 provides a plot of the simulation time taken for SystemC TLM and Verilog RTL models respectively. The time indicated in our study refers to the development time in hours and is presented in Figure 8. The duration indicated here includes the time taken in understanding the technical specification document of the peripheral model technical and leading all the way to modeling and testing. An advantage that could have been possible for the reduced amount of time for the SystemC TLM could be due to our earlier experience in programming with C++. However, the time spent in learning how to model at transaction level was substantial.

The code length indicates the actual lines of source code in SystemC TLM and Verilog RTL model. The reduction in code length is mainly attributed to the level of detailing modeled in the SystemC TLM as compared to the Verilog RTL model. A plot of the same is shown in Figure 9. The TLM developed in SystemC is flexible in the sense that we are able to connect the model to a simple bus by providing the necessary slave interface to connect to the bus as presented in Figure 6. From our previous experience and relevant study, similar integration is expected to take significant effort with the RTL model.

![Figure 6. Simple System Architecture](image)

![Figure 7. Simulation Speed](image)

![Figure 8. Total development time to model](image)

5. Results

According to our study, modeling at higher level of abstraction does not necessarily compromise on the functional accuracy. Similar accuracy was achieved in both models developed as the SystemC TLM developed was clock accurate. A timer value of 0x0F which counts down once every eight clock ticks, should set the output signal after 120 clock ticks. This is verified in Figure 3 and 4 for both the SystemC TLM and Verilog RTL models. The speed of simulation is better in case of SystemC TLM due to less implementation details and also due to the reduction in concurrent threads being modeled.
6. Conclusion

All successful modeling strategies have eliminated excessive design complexity while not over-limiting the design space. Cycle accurate RTL design, Instruction Set Architectures, and software objects all captured the essence of what designers had already been doing, or trying to do, in an elegant way. The fundamental question to be asked is what will the new design elements be? Designers need better tools, languages and methodologies to enable them to creatively and efficiently manipulate the diverse hardware and software elements together. Support for modeling the system at higher levels of design abstraction could be attributed to as an initial step. In our study, we modeled a simple peripheral at two different levels of abstractions, namely at RTL and TLM. To model at RTL we used Verilog and to model at TLM we used SystemC language. For comparing the models at different levels of abstraction the parameters we used include, Simulation Speed, Accuracy, Code length, Time to Develop, and Flexibility. Depending on the context the order of importance among the parameters could vary. However, we feel that the SystemC’s flexibility to easily integrate a new model into an already existing architecture should be a boon to IP developers and users. Also as shown in our study moving to a higher level of abstraction does not necessarily have to affect the accuracy of model, and it may have a positive effect in terms of reducing the simulation time and the code length.

Finally, in order to tackle design complexity a migration to higher level of abstraction would be a necessity.

References

[3] Link to our CVSI website [www.cvsi.fau.edu](http://www.cvsi.fau.edu)
URL: [http://www.eedesign.com/story/OEG20030317S0044](http://www.eedesign.com/story/OEG20030317S0044)