

Introduction of System Level Architecture Exploration Using the SpecC Methodology

L. Cai, D. Gajski

University of California, Irvine
Dept. of Information and Computer Science
Irvine, California 92697, USA

M. Olivarez

Motorola – Architecture and Systems Lab
7700 West Parmer Ln.
Austin, Texas 78729, USA

ABSTRACT

To implement chip design on satisfactory target architectures, architecture exploration should be done at higher levels of abstraction, in the earliest design stages. Using the SpecC language, an executable system level design language, system level architecture exploration can proceed easily and smoothly as the system specification is being created. A SpecC methodology of system level architecture exploration is introduced within this paper to illustrate this process. The design of a JPEG encoder is used as an example to illustrate the system level architecture exploration methodology.

1. INTRODUCTION

In the SoC design process, the work of mapping functionality into target architectures, called architecture exploration, is one of the main problems facing the SoC designer. At the RTL level, performing architectural exploration is too cumbersome to satisfy the time to market requirement. Therefore, a system level architecture model is required by the industry to implement fast architecture exploration.

Using the SpecC language, a system specification language developed at UC Irvine, one can implement architecture exploration easily and efficiently [1] and therefore explore more target architectures at the system level. Furthermore, the final result of an implementation on a target architecture model, at the system level, can be smoothly and consistently changed to an RTL level model by using the rest of the SpecC methodology.

In this paper, the system level architecture exploration methodology is introduced as part of the SpecC methodology. The existing SpecC methodology mainly focuses on refining specification level into architecture level [1] of abstraction. System level architecture exploration focuses on changing the implementation from one target architecture to another target architecture, at the specification level of abstraction. This methodology of system level architecture exploration allows the designer to compare implementations on different target architectures and make trade-off decisions at the highest level of design abstraction. It also defines a method to improve the current implementation by changing some part of the target architecture to meet the design needs. In this paper, a JPEG encoder example is used to illustrate this process.

The sections of this paper are organized as described here. Section 2 summarizes the SpecC language as well as the existing SpecC Methodology, which is described in [1]. In section 3, a description of the JPEG encoder algorithm, which is used for our

tests, is given. Section 4 introduces the system level architecture exploration methodology. In section 5, a JPEG encoder model has been architecturally explored. Finally, in section 6, conclusions are presented.

2. SPECC

2.1 SpecC Methodology

The SpecC methodology is a design methodology to implement design from pure specification into full implementation [1]. It defines four levels of modeling, from the most abstract level to the most detailed level. The first level of abstraction is the specification model, which represents pure specification. The second model, which is the architecture model, represents the implementation on target architectures with the abstraction of computation and communication specifications. The communication model, which is third in the hierarchy, represents the implementation on target architectures with the abstraction of computation but detailed communication model. The fourth or implementation model is the synthesizable RTL model.

Besides the four models, the SpecC methodology also defines the method of transitioning between these models. Among them, architecture exploration refines the specification model to architecture model. This paper is focused on this transition. System level architectural exploration is easily implemented because of the flexibility of the models defined by the SpecC methodology.

2.2 SpecC Language

The SpecC methodology describes abstraction at the system level using a specification language called SpecC [1,2]. SpecC is a super-set of C, which extends ANSI-C to allow easy reuse of the existing algorithmic and behavioral C descriptions that are common practice. SpecC contains all the features required to support system-level design and specification.

3. JPEG ENCODER

The JPEG image compression standard [3] was chosen for its simplicity as well as the existence of SpecC code. Figure 1 shows the block diagram of the DCT based encode for a gray scale image. It consists of four blocks: the image fragmentation block, the DCT block, the quantization block and the entropy coding block as shown in Figure 1. Although each block can be further broken down, the capabilities of the smaller components becomes less significant for reuse and model abstraction.

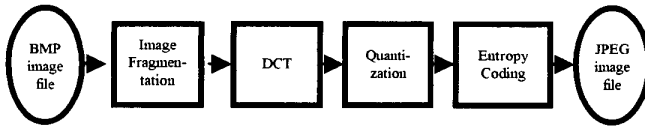


Figure 1. Block Diagram for the JPEG encoder.

4. SYSTEM LEVEL ARCHITECTURE EXPLORATION

4.1 Architecture Exploration Methodologies

In addition to SpecC, current tools available for architecture exploration are Cadence's VCC [10] and YXI's eXploration Compiler [7], as well as research efforts such as Ptolemy [9] and SystemC [8]. In the use of Ptolemy, UML is the resource for the creation of the system specification. This requires a change from the C reference specification, with explorations occurring as manipulation of the UML code. VCC can also perform architectural exploration similar to the SpecC Language, but it has added overhead in the capture of the C reference model into VCC as well as the cost of a proprietary solution. SystemC, as of version 1.0, lacks the ease of higher level abstraction needed to easily explore communications and concurrent architectures from the software model. SystemC along with new tools from Synopsys does provide a method of synthesizing cycle accurate C models that can be compiled with the GNU C++ compiler. YXI's eXploration Compiler currently uses behavioral VHDL as its specification for performing architectural explorations. Of the efforts mentioned, YXI's approach seems to automate the process the most, but requires the components necessary to be defined within the database it searches for solutions. When compared to the tools mentioned, SpecC makes the process of creating a system model from the C reference specification easiest. The language is limited so that introducing unnecessary lower levels of abstraction can be more easily avoided, and removes much of the overhead for modeling concurrency and communications. Using this approach, SpecC tools will be able to integrate with other leading system level design tools.

4.2 System Level Architecture Exploration with SpecC

Architecture exploration is a method to refine the specification model into the architecture model. However, in the design process, different implementation models based on different target architectures are always compared. Therefore, unbiased metrics need to be obtained. Furthermore, the most common way to improve the current design is to modify existing target architectures to achieve better performance. This need to explore very different architecture models can be implemented using the methodology, which we call System Level Architecture Exploration. This exploration methodology includes four types of basic operations as shown in Figure 2.

The first block includes three types of System Level Architecture Exploration sub-operations: Function Moving exploration, Component Separation exploration, and Component Merging

exploration. Function Moving exploration moves one function block from one component into another existing component within the target architecture. Component Separation exploration moves one function block from one component into a new component in the target architecture. Component Merging exploration merges two components in the target architecture into one component.

The second block includes two types of sub-operations to be done as a second pass of exploration: Parallel Execution and Sequential Execution exploration. Parallel Execution exploration schedules two components, which communicate with each other, to make them execute in parallel. Sequential execution exploration schedules two components, which communicate with each other to make them execute sequentially. The second sub-operation is useful to remove unnecessary additions in the target architecture, which will not effect the system.

The third block includes Architecture Pipeline and Component Sharing exploration. Architecture Pipeline exploration doubles a system limiting component and its functionality in the target architecture. Communications information is add to make the functions run in parallel to improve the overall system performance. The Component Sharing exploration is reverse of architecture pipeline exploration.

The fourth block contains our last two types of system level architecture exploration sub-operations: Shared Global Memory and Message Passing exploration. Shared global memory exploration changes the communication from a message passing method into a shared global memory method. Message passing exploration is the reverse process of shared global memory exploration.

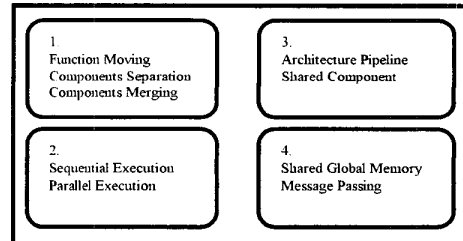


Figure 2. System Level Architecture Exploration operation types.

Based on the SpecC language, the system level architecture exploration is implemented manually, but without difficulty. With the guidelines mentioned in [6], automated tools can be implemented for the system level architecture exploration. Furthermore, system level architecture exploration can be merged into the SpecC methodology, which smoothly leads to the final stage of the design process, implementation.

5. JPEG EXAMPLE

5.1 System Level Execution Time Concept and Simulation Environment

The estimation time used in system level design is at a very abstract level. Estimation times of leaf nodes, such as the

Quantization block, are used as the smallest time units. The total execution time of the design should be the dynamic sum of the estimation time of leaf nodes, without going into any more granularity of timing. This is the idea of system level execution time versus the concept of cycle accurate modeling. The system level execution time is defined as the execution time, which is calculated based on each leaf function node's execution time. It is a crude method, but very useful in the early stages of designing.

Profiling tools of chosen processors can achieve the execution time of leaf nodes. Such examples are Motorola's development tools for various processors or estimation tools of ASIC architectures. If a component to run the specified function exists in a reusable library, the execution time can be obtained directly from the database. After using profiling tools and estimation tools, the estimation time should be written back to SpecC model using the *waitfor* keyword. This step will take the untimed system level specification model into timed system level specification model. After rewriting, the executable specification can be executed to achieve the total estimation time of the design. Estimations for this experiment were taken from profiles of the Motorola DSP56000 for the full software implementation. DCT estimations were also taken from processors and specifications of hardware to be built.

In the JPEG encoder example, a DSP56600 processor with maximum clock frequency of 60MHz (which is called SW in this paper) and an ASIC block to be integrated (which is called HW in this paper) are chosen as architecture components. From the view of easy implementation. The estimation times of JPEG's four leaf nodes on SW, which implement four blocks in section 3, are given in Table 1 [4].

Since the DCT block causes most of the execution time, it is a good candidate to be run in HW. In this paper, the two cases are tested. In the first case, a reused HW, which has the execution time of 650us per 8*8 pixel block for DCT is chosen, and the target architecture is searched. In the second case, the estimation time for HW is unknown. Therefore, the 7 HW solutions, which have different time constraints (as shown in Table 2), are tested to explore the trade-off between different HW solutions and different target architectures.

Estimates for communication, are also needed. Communication overhead between SW and HW is assumed as one pixel per SW cycle. For an 8*8 pixel block, the overhead is 1us.

In this paper, a bitmap (bmp) file, which includes 180 8*8 pixel blocks, is used as input of the testbench, and the expected timing constraint is assumed to be 90ms. To count the overall JPEG encoder execution time, a time simulator, which is also written in the SpecC language, is added to the simulation environment. The time simulator runs parallel with the specification of the JPEG encoder to tabulate the total execution time.

Handledata (HD)	DCT (DCT_8)	Quantization (QZ)	Huffman (HF)
142 us	745 us	92 us	162 us

Table 1. Estimated execution time for leaf nodes on SW for each 8*8 pixel block.

HW 1	HW 2	HW 3	HW 4	HW 5	HW 6	HW 7
650 us	600 us	500 us	400 us	300 us	200 us	100 us

Table 2. Timing constraint for DCT on 7 different hardware solutions for each 8*8 pixel block.

5.2 System Level Architecture Exploration Process

As shown in the outlined steps following, five target architecture models are explored [6]. In Pure_SW model, the four leaf nodes are implemented in the SW. In SW_HW sequential model, a HW is used to run DCT block while other leaf nodes remain in SW. In this model, since HW and SW work on the same 8*8 pixel block input and they have to wait each other until the other part has finished its execution, SW and HW are run sequentially. In SW_HW parallel model, SW and HW are scheduled to run different 8*8 pixel blocks input concurrently. In SW_2HW parallel model, two HW and one SW are used to run different 8*8 pixel blocks input concurrently. In SW_2HW_shared mem parallel model, a global memory is used to implement shared memory communication, instead of message passing communication, based on SW_2HW_parallel model.

To implement these target architectures, four main types of system level architecture exploration sub-operations: component separation, parallel execution, architecture pipeline, and shared global memory exploration are utilized, as shown in the following steps.

- 1). Pure SW on reference processor.
- 2). Component Separation exploration is performed on the specification model to find the bottlenecks in the software so hardware accelerators can be added.
- 3). SW_HW Sequential exploration testing is performed to see if the change of adding a hardware module to run in-line with the software, lowers the time required for the system to meet requirements.
- 4). Parallel Execution exploration is added to the specification model to test what happens when the DCT hardware runs as the HD, QZ, and HF modules are run on the reference processor.
- 5). SW_HW Parallel exploration testing allows the SW and HW components to be changed from sequential to parallel execution to see if the changes made allow the system to meet the requirements.
- 6). The specification model is changed using Architecture Pipeline exploration, to see what happens when the architecture is pipelined in respect to hardware and software, with like components being run in parallel.
- 7). SW_2HW Parallel exploration testing allows the testing of a pipelined architecture along with using multiple components. This begins creating the specification for RTOS requirements.
- 8). The specification model then adds Shared Global Memory exploration to model the specification more

as the real system may operate. Although one configuration was used, various memory configurations can be created at this time, in a short amount of time.

- 9). The SW_2HW Shared Memory Parallel model is tested to ensure the requirements are met by the specification. These steps can begin again if needed such that the JPEG model can utilize more hardware components for all for other sections.

5.3 Result

The time estimation results shown in Table 3 have been obtained by running the simulations along with the SpecC simulator. The underlined data satisfies the time constraint of 90ms. From Figure 3, which is derived from Table 3, the following solutions can be achieved. First, either the SW_2HW_parallel or SW_2HW_memory models can achieve the desired result, using the HW, which can execute the DCT in 650us. This result should incur the lowest cost in size and power. The second scenario would have the hardware run either sequentially, or in parallel. To use either of these solutions, the DCT hardware would have to execute in at least 400us (100us for sequential), thus incurring the largest cost in size, and power.

	Pure_SW (ms)	SW_HW _sequential (ms)	SW_HW _parallel (ms)	SW_2HW _parallel (ms)	SW_2HW _memory (ms)
HW1	205.56	188.82	117.94	<u>72.33</u>	<u>72.69</u>
HW2	205.56	179.82	108.94	<u>72.23</u>	<u>72.59</u>
HW3	205.56	161.82	90.94	<u>72.04</u>	<u>72.40</u>
HW4	205.56	143.82	<u>72.94</u>	<u>71.94</u>	<u>72.30</u>
HW5	205.56	125.82	<u>72.03</u>	<u>71.84</u>	<u>72.20</u>
HW6	205.56	107.82	<u>71.88</u>	<u>71.82</u>	<u>72.18</u>
HW7	205.56	<u>89.82</u>	<u>71.82</u>	<u>71.82</u>	<u>72.18</u>

Table 3. Execution time of JPEG encoder for different hardware solutions

Using the SpecC language and system level architecture exploration guidelines [1][6], the JPEG explorations mentioned in this paper can be achieved in few hours.

6. SUMMARY

This paper introduces a new methodology for system level architecture exploration. This is the methodology of modifying a design implementation from one target architecture model to another target architecture model, within the architecture level of the SpecC methodology [1]. Since this model is in an abstract level, the process of system level architecture exploration is fast.

System level architecture exploration refines the architecture level of SpecC methodology. Using this methodology within the SpecC design flow, the fast estimation of implementation, and finding the target architecture earlier in the design cycle, can be achieved, as shown in this JPEG encoder example. One of the strengths of the SpecC Language is the limitations placed on the

language to allow for the ease to create synthesized logic. This differentiating feature, contrasting SpecC from the tools mentioned in Section 4.1, removes the ambiguities that can be added into the model buy using a multipurpose tool. The SpecC Methodology and Language, along with the steps defined here, aids in moving architectural exploration into higher levels of abstraction, and will thus cut down on the design cycle time.

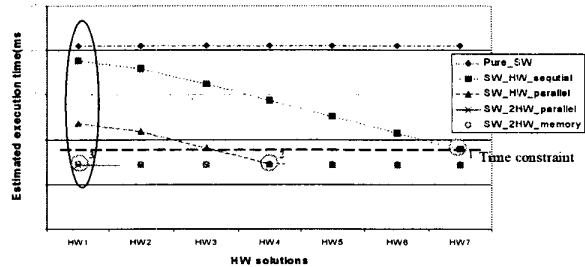


Figure 3. Execution times to perform JPEG encoding using various hardware solutions.

We would like to add thanks to Gordon McGregor of Motorola for his help with this work.

7. REFERENCES

- [1] D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, March, 2000
- [2] A. Gerstlauer, S. Zhao, D. Gajski, A. Horak. "Design of a GSM Vocoder using the SpecC Methodology". University of California, Irvine, Technical Report ICS-99-xx, February 1999.
- [3] V. Bhaskaran, K. Konstantinides. *Image and Video Compression Standards*, Second Edition, Kluwer Academic Publisher, 1997
- [4] L. Cai, J. Peng, C. Chang, A. Gerstlauer, H. Li, A. Selka, C. Siska, L. Sun, S. Zhao, D. Gajski. "Design of a JPEG Encoding System", University of California, Irvine, Technical Report ICS-99-xx, September 1999.
- [5] H. Yin, H. Du, T. Lee, D. Gajski. "Design of a JPEG Encoder using SpecC Methodology". University of California, Irvine, Technical Report ICS-00-xx, July 2000
- [6] L. Cai, M. Olivarez, D. Gajski. "System Level Architecture Exploration using the SpecC Methodology". Irvine Technical Report ICS-00-xx, Sep 2000.
- [7] Y Explorations Inc., *eXplorations Compiler (XC): Synthesis, Features, and Options, version 1.2*. August 1999.
- [8] Synopsys, Inc., CoWare, Inc., Frontier Design, Inc., "SystemC Version 1.0 Draft Specification". Open SystemC Initiative, 1999.
- [9] J. Davis, C. Hylands, B. Kienhuis, E. Lee. "Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java". UCB/ERL Memorandum M99/40, June 12, 2000.
- [10] G. Martin. "UML and VCC White Paper". Cadence Design Systems, Inc., Version 1.1, December 8, 1999.