ABSTRACT

A UML-based design flow for dynamically reconfigurable computing systems (DRCS) is proposed in this work. The proposed design flow is targeted at the execution speedup of functional algorithms in DRCS and at the reduction of the complexity and time-consuming efforts in designing DRCS. The most notable feature of the proposed design flow is a partitioning methodology based on the UML 2.0 sequence diagram, called Dynamic Bitstream Partitioning on Sequence Diagram (DBPSD). Partitioning guidelines are included in DBPSD to help designers make partitioning decisions at the class method granularity. To prove the feasibility of the proposed design flow and DBPSD partitioning methodology, an implementation example of DES (Data Encryption Standard) encryption system is presented in this article.

1. INTRODUCTION

Processors provide high flexibility in executing a wide range of applications, but they suffer from limited performance. Application specific integrated circuits (ASICs) provide superior performance, but are restricted by limited set of applications. Thus, a new computing paradigm called dynamically reconfigurable computing systems (DRCS) [1] is a promising solution, which provides an intermediate tradeoff between flexibility and performance.

The architecture of reconfigurable computing systems typically combine a microprocessor with reconfigurable logic resources, such as FPGA. An abstract model of such an architecture appears in Figure 1. To reduce overall communication, the workload is distributed as follows. The microprocessor executes the main control flow of an application and the operations that cannot be done efficiently in the reconfigurable logic. The reconfigurable logic performs the computing-intensive parts of the application. Shared memory is used for communication between the microprocessor and the reconfigurable logic.

The work in this article is concerned with the development of a design flow and of related supporting tools for dynamically reconfigurable computing systems. The proposed design flow takes a UML-based application model and facilitates the co-synthesis and rapid prototyping of dynamically reconfigurable computing systems. The outputs of our design flow are the ready-to-run software application and hardware bitstreams for a target platform. Furthermore, the primary focus of this article is on the hardware-software partitioning of UML models, which makes it different from previous researches.

Reconfigurable computing technologies not only increase the execution speed of application, but also reduce the power consumption and cost. For example, rather than requiring one chip to do image compression and another one to perform image decompression, the reconfigurable chip dynamically reconfigures itself for each algorithm as required.

Unfortunately, designing these kinds of systems have a high complexity and is a formidable task. Although many researches [2, 3] are ongoing in the academia and industry, but the lack of tools and design flows discourage designers from adopting the reconfigurable computing technology. Application development for such a software-hardware mixed architecture requires engineers with both software and hardware backgrounds. However, the typical application designer of reconfigurable systems is no expert in digital hardware knowledge. This leads to the need of a design flow especially easy for use by software engineers.

The article is organized as follows. In Section 2, we give a detailed discussion of each phase of our design flow. Section 3 is the originality of this article, where we present our partitioning methodology for UML models. In Section 4, an example is presented to show the feasibility of our design flow and partitioning methodology. In the last section, we conclude this article.
2. THE DESIGN FLOW

Figure 2 shows the proposed design flow, which is divided into three phases: design and implementation of the system software model, hardware synthesis, and software synthesis. In Figure 2, the ellipse boxes such as Class Diagram represent the workproduct in each phase. The rectangle boxes such as Rhapsody 5.0 represent the commercial-off-the-shelf tools used. The three-dimensional rectangle boxes such as SW/HW Extractor represent tools that are developed by ourself. Each phase of the proposed design flow will be examined further in the following subsections. Moreover, the target platform for verifying the proposed design flow is also constructed. In the final subsection, related issues of the proposed design flow are discussed.

2.1. Design and Implementation of System Software Model

Starting with the UML modeling tool, Rhapsody 5.0, UML 2.0 [4] diagrams such as the class diagram, sequence diagram, and state machine diagram are used to build the system model. The detailed behavior of the system model is implemented in the Java programming language. After the generation of the executable system model and the behavior verified, manual partitioning is done on the sequence diagrams by the user. XMI (XML Metadata Interchange) documents as an intermediate representation of the system software model are generated by Rhapsody 5.0. We then use our SW/HW Extractor to parse the partitioning information from the XMI documents to extract SW JAVA code and HW methods for the software synthesis phase and the hardware synthesis phase, respectively.

2.2. Hardware Synthesis

The purpose of this phase is to synthesize the hardware bitstreams which will be used by software applications, to perform some required computing-intensive operation. The HW Methods derived from SW/HW Extractor are the inputs of this phase. First, the Forge tool is used to transform the HW Methods into synthesizable Verilog HDL (Hardware Description Language) code. Secondly, the HW Interface Synthesizer adds PCI wrapper for the Verilog HDL code, then produces the HW Verilog HDL code with interface which enables communication with software. Finally, the XFlow tool will synthesize the HW Verilog HDL code with interface into the hardware bitstreams for execution in FPGA.

2.3. Software Synthesis

The purpose of this phase is to build executable Java Application, which is capable of invoking hardware method on demand. The SW JAVA Code which was derived from SW/HW Extractor are the inputs of this phase. Starting from SW JAVA Code, SW Interface Synthesizer is used to synthesize code for accessing hardware methods. After that, the produced SW JAVA Code with Interfaces is the final source code. Finally, the SW JAVA Code with Interfaces is compiled by the JAVA Compiler to generate ready-to-run Java program, called Reconfigurable JAVA Application. During the execution on the host processor, this Reconfigurable JAVA Application can reconfigure the required hardware methods into the FPGA for the acceleration of software execution.

2.4. Target Platform

Target platform allows system designers to verify the overall system behavior and to evaluate the overall system performance. Our target platform is a dedicated Xilinx Virtex-II FPGA (XC2V3000, 28,672 LUTs, at 40MHz) board connected to the host computer (Pentium 4 2.8GHz, 1GB RAM, Windows XP ) over the PCI (Peripheral Component Interconnect) interface.

2.5. Discussion

The advantage of the UML-based design entry is the comprehensible specifications when designing complex applications. The disadvantage is the informal semantics of the UML, thus some specifications cannot be specified precisely. When compared to the manual design of DRCS, the proposed design flow allows significant reduction of error-prone, tedious, and time-consuming tasks, such as hardware design and HW-SW interface synthesis.
3. THE HW/SW PARTITIONING ON SEQUENCE DIAGRAMS

Dynamically reconfigurable computing systems contain both a microprocessor (CPU) and the reconfigurable logic (FPGA), and thus require an application to be partitioned into software sections to be executed on the CPU and hardware sections to be executed on the reconfigurable logic.

3.1. The DBPSD Partitioning Methodology

We propose a *Dynamic Bitstream Partitioning on Sequence Diagrams (DBPSD)*, which is a partitioning methodology based on the UML 2.0 sequence diagrams and includes *partitioning guidelines* to help designers make prudent partitioning decisions at the granularity of class methods.

Sequence diagrams in UML 2.0 have been significantly enhanced for specifying complex control flows in one sequence diagram. As shown in the middle of Figure 3, the most obvious changes are the three rectangle boxes, called *interaction fragment*. The five-sided box with labels such as alt, opt, or loop at the upper left-hand corner is the *interaction operator* of the interaction fragment.

Figure 3. The Example of the Partitioning on UML 2.0 Sequence Diagram

The interaction operator gives the interaction fragment specific meaning. The alt operator provides the condition choice according to the evaluation result of the guards. For example, if the guard \( x > 1 \) is evaluated to true the M4() method will be called, else the [else] guard will be evaluated to true then the M5() method will be called. The opt operator is the if portion of the alt, that is the same as the if construct in C language. The loop operator defines that an interaction fragment shall be repeated several times.

When doing partitioning on the sequence diagrams, a designer may add a UML stereotype \(<\text{HWx}>>\) for a method to be implemented in hardware. For example, in Figure 3 the methods M2() and M3() are marked by the same stereotype \(<\text{HW1}>>\), but the method M4() is marked by another stereotype \(<\text{HW2}>>\). As a consequence, the methods M2() and M3() will be synthesized in the same bitstream, but the method M4() will be synthesized in another bitstream. Calling a hardware method that is synthesized in a different bitstream will require the FPGA to be reconfigured, therefore additional time overhead will be incurred.

The key performance penalties in dynamically reconfigurable computing systems are the FPGA reconfiguration time and the communication time between the CPU and the FPGA. These overheads are mainly dependent on the hardware restrictions. However, we can reduce the numbers of FPGA reconfigurations, so that reconfiguration overhead is decreased.

To reduce the number of FPGA reconfigurations, we need to take the control flow and execution order into consideration when doing partitioning on sequence diagrams. Hence, in DBPSD the following partitioning guidelines are provided:

**Guideline 1:** *Add the same stereotype \(<\text{HWx}>> for all computing-intensive methods.* For example, M1() \(<\text{HW1}>>\), M2() \(<\text{HW1}>>\), ... M12() \(<\text{HW1}>>\). If synthesis is feasible, only one bitstream is produced, thus only one reconfiguration action is needed.

**Guideline 2:** *Add the same stereotype \(<\text{HWx}>> to all dependent methods.* For example, because M3() is invoked by M2(), even if M3() is not computing-intensive, we should add the same stereotype \(<\text{HW1}>>\) to both M2() and M3().

**Guideline 3:** *For the alt operator, add the same stereotype \(<\text{HWx}>> to all the computing-intensive methods in all condition branches.* If the synthesis of the stereotyped methods is not possible, then start to move the last condition branch to another stereotype \(<\text{HWy}>>\) first, until synthesis is successful. For example, first add the same stereotype \(<\text{HW2}>>\) to M4() and M5(). If it is not synthesizable, change the stereotype of method M5() in last condition branch to \(<\text{HW3}>>\).

**Guideline 4:** *For the opt operator, add all of the methods inside this interaction fragment by a stereotype different from the methods outside this interaction fragment.* For example, if M7(), M8(), and M9() are not synthesizable in the same bitstream, then change the stereotype of M8() to \(<\text{HW4}>>\), because M8() may not always be invoked.

**Guideline 5:** *For the loop operator, add the same stereotype \(<\text{HWx}>> to all the computing-intensive methods inside this interaction fragment.* If the synthesis of the stereotyped methods is not possible, then start to move the less computing-intensive method to another stereotype \(<\text{HWy}>>\) first, until synthesis is successful. For example, first add the same stereotype \(<\text{HW5}>>\) to both M10() and M11(). If it is not synthesizable,
change the stereotype of method M11() in the last condition branch to <<HW6>>.

4. A DES ENCRYPTION SYSTEM EXAMPLE

To prove the feasibility of the proposed design flow and DBPSD partitioning methodology, an implementation example of DES encryption system is presented in this section.

Figure 4 is the class diagram for this example, which contains four classes: Controller (for the controlling of overall system), NIC (for the simulating of network interface), CRC (for calculating the CRC value), and DES (for encryption and decryption of message). Due to the limited space, the state machine diagram of this example is not shown.

Figure 4. The Class Diagram of the DES Encryption System Example

The sequence diagram which depicts the overall system control flow of this example is shown in Figure 5. DES is the computing-intensive part of this system, thus the partitioning focuses on its two methods: encrypt() and decrypt(). Figure 5 shows the optimal partition for this example. The other partitions and the implementation results are shown in Table 1.

Figure 5. The Sequence Diagram of the DES Encryption System Example

In Table 1, the most notable partitions are Partition 2 and Partition 3. The difference in the total execution time of Partition 2 and Partition 3 was not expected. The reason for this difference can be observed from the sequence diagram for this example. Different control flows affect the number of times each method is invoked. Thus, the worth of doing partitioning on sequence diagram is proved by this example.

Table 1. The Implementation Results

<table>
<thead>
<tr>
<th>Partition</th>
<th>encrypt()</th>
<th>decrypt()</th>
<th>Total Execution Time</th>
<th>FPGA Utilization (%)</th>
<th>Speedup Compared to Partition 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SW</td>
<td>SW</td>
<td>7200us</td>
<td>0%</td>
<td>+2.81x</td>
</tr>
<tr>
<td>2</td>
<td>SW</td>
<td>HW</td>
<td>2560us</td>
<td>18%</td>
<td>+1.48x</td>
</tr>
<tr>
<td>3</td>
<td>SW</td>
<td>SW</td>
<td>4880us</td>
<td>18%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>HW</td>
<td>HW</td>
<td>240us</td>
<td>36%</td>
<td>+30x</td>
</tr>
</tbody>
</table>

5. CONCLUSION

A UML-based design flow and its HW-SW partitioning methodology are presented in this article. The enhanced sequence diagram in UML 2.0 is capable of modelling complex control flows, thus the partitioning can be done efficiently on the sequence diagrams. Additionally, the real implementation results and information produced by the proposed flow such as application performance datum, hardware method execution time, FPGA reconfiguration time and communication overheads can used for simulation or evaluation. We see this design flow as a sound basis for carrying out further research on reconfigurable computing.

6. REFERENCES