Automatic Reconfigurable System-on-Chip Design with Run-Time Hardware/Software Partitioning

Yu Chen, Ren-Fa Li, and Qiang Wu
College of Computer and Communication, Hunan University, Changsha, China.
pnompenh@gmail.com

Abstract

Reconfigurable System-on-Chip (RSoC) is a promising alternative to deliver both flexibility and performance at the same time, and also a technical solution looking to the future needs of embedded applications. But the complex design process is impeding the development of extensive applications. This paper proposes an RSoC design methodology based on function-level programming model on account of the characteristics of the reconfigurable architecture. In the programming model, system designers use high-level language to complete functional specification by calling the co-function-library. Then the dynamic hardware/software partitioning algorithm will decide whether an invoked function should be running on hardware or software automatically. According to the partitioning result, the dynamic linker will switch functions’ execution mode in real time. And the above items can facilitate an automatic design flow through specification to the system implementation. Experiments and tests have verified the feasibility and efficiency of the automatic design flow.

1. Introduction

Reconfigurable System-on-Chip (RSoC) integrates programmable logic devices, microprocessors and other circuit modules into a single chip, and becomes an intermediate tradeoff between functional flexibility and high computing speed by taking advantage of the reconfiguration of programmable logic devices and programmability of microprocessors. Hence RSoC has been considered as a competitive technical solution which could meet the needs of complex requirements in embedded applications market.

RSoC typically contains one, or more, microprocessor and reconfigurable devices. The microprocessors execute software code while the reconfigurable devices implement the hardware logic circuit. For exploiting this kind of hybrid system, designers are always trapped in a fuzzy flow by using existing design methods and development tools: manually partition specification into hardware or software functions first, and then map different functions to microprocessors or reconfigurable devices. Obviously, hardware/software partitioning is an arduous challenge. Moreover, lots of involved issues have to be considered in order to obtain a high utilization ratio of system resources, including using the hardware description language and software description language separately, implementing dynamic reconfiguration, and designing communication methods between hardware and software. All these difficulties call for both rich development experience and high professional quality of a system designer, and ultimately impede further development and application of RSoC.

Dynamic hardware/software partitioning can help system designers saving large amount of time that spends on simulation and emulation and making run-time performance analysis directly, then distinguishing the bottleneck of the whole system and partitioning it to reconfigurable devices. This technology frees the designers from troublesome experiments and finally facilitates system development. So, to work on dynamic hardware/software partitioning is very important for the practicability of RSoC.

In this paper, looking to the application demands of RSoC, we provide a brand-new reconfigurable RSoC design methodology to realize an automatic design flow through specification to the system implementation based on function-level hardware/software programming model and run-time dynamic partitioning algorithm. Theoretically, this new methodology overcomes some flaws of traditional design methodology such as bad user transparency and automaticity. Practically, we also provide an RSoC development kit and application oriented co-function-library. Experiments show that our methodology and tool can evidently simplify the system development process and improve efficiency.
The structure of this paper is as follows: Section 2 gives an overview of related work. Section 3 describes the architecture of the RSoC development platform. Section 4 presents the design and implementation of hardware/software co-function-library. Section 5 then addresses the run-time dynamic hardware/software partitioning algorithm. Section 6 analyses the experiment results. Section 7 finally concludes this paper.

2. Related Work

Many researches focused on the design flow and methodology of reconfigurable System-on-Chip. [1] proposes a design flow instantiation for run-time RSoC. The design flow is roughly divided into two parts: system and implementation level. At system level, supports for hardware resource estimation and performance evaluation are applied. At implementation level, technology-dependent tools are used to realize the run-time reconfiguration. [2] presents an automatic design methodology for heterogeneous architectures based on reconfiguration pre-fetching technique and buffer-merging technique. [3] integrates some powerful partitioning algorithms into an unified system framework. [4] proposes a software control method used for increasing the utilization of hardware area in unit time by analyzing wrapper developing method in hardware applications.

In research area of building new system design tools, [5] provides a graphic tool to help system-on-chip design. The hardware/software partitioning is specified interactively by users. In order to enhance the portability of system, Emanuele Lattanzi and Aman Gayasen managed to achieve transparent load of hardware at system runtime by modifying the JAVA virtual machine [6]. [7] argues that it is necessary for operating system to make an abstraction of programmable devices. Furthermore, an operating system framework which supports reconfigurable devices has been defined.

Research on dynamic hardware/software partitioning mainly falls into two methods according to different partition levels and granularities. The first method is proposed by Greg Stitt on proceeding of DAC at 2003 [8] and continuously improved in subsequent work. They use specialized high-speed buffer to record the frequency of a loop in software program [10]. The machine code of the most frequent loop will be disassembled, and a data flow diagram will be generated and synthesized on-line to logic circuit by additional microprocessor [9]. The logic circuit will be configured to programmable devices. Designers need not to concern the partitioning of hardware/software by resigning it to dynamic partitioning algorithm and simply design the system in software style. This method starts from the bottom, so is called instruction-level hardware/software partitioning. Another method pays attention to system level, regards the hardware circuit that configured to programmable devices as hardware process (thread). Migration and switching of hardware/software process (thread) is implemented by modifying operating system to support hardware process (thread). So this method is called process-level (thread-level) hardware/software partitioning [10-13].

Instruction-level and process-level (thread-level) hardware/software partitioning have their own advantages and disadvantages. Instruction-level method has advantages of avoiding user intervention. It is unnecessary to modify complier and operating system. But it leads to some problems such as large overhead of hardware, limited performance enhancement and cannot reuse existing hardware design resources. Process-level (thread-level) hardware/software partitioning has advantages of treating hardware/software tasks as processes or threads managed by operating system together. Integrated development can be carried out due to the reuse of existing hardware resources [14]. But compared to instruction-level method, process (thread) scheduling, communication and synchronization are all controlled by software, certainly causes more time overhead.

3. Platform for Reconfigurable System-on-Chip Design

In this paper, we use high-level language (Java and C language) for system functional specification and Field Programmable Gate Array (FPGA) as the reconfigurable device. A unified and transparent programming model which is depicted in Figure 1 has been made up to abstract the difference of bottom hardware. The hardware/software co-function-library is designed for specification of systems. In the co-function-library, hardware functions are IP cores which are encapsulated by the same syntax as software functions, so that both hardware and software functions can be defined in a unified way. Designers program the functional specification directly by invoking different functions in the co-function-library and the whole design flow is compliant with software development process. When program is running, dynamic link control is in charge of linking function calls to real implementation. The dynamic link control
is implemented by a dynamic linker and a hardware/software partitioning algorithm. The dynamic linker provides the parameters needed by partitioning algorithm, gets the results generated by the partitioning algorithm, and finally maps the invocation of abstract functions to software instruction or hardware logic. In this case, a complete hardware/software partitioning flow is formed.

Figure 1. A unified and transparent programming model

3.1. Hardware/Software Co-Design Flow

The whole system development process which is depicted in Figure 2 includes three stages:

**System Specification**: designers finish the specification in C language by calling functions (interfaces of the functions are coded by C language) in co-function-library. Code for performance constraints can also be added, for example: specifying whether a function in library is executed by hardware or software; giving a time constraint to a function. All functions in co-function-library can either be executed by software running on microprocessors or hardware running on reconfigurable devices.

**Software Synthesis**: compiles and generates executable files of functional specification, and acquires information of invoked functions (hardware/software functions having the same interface), such as chip area needed for hardware accelerator, performance constraints, etc.

**Run-Time Dynamic Hardware/Software Partitioning**: executable files generated at software synthesis stage will be loaded and executed on RSoC. Hardware/Software partitioning algorithm decides whether invoked functions will be executed on microprocessors (software functions) or reconfigurable devices (hardware functions) according to the run-time information obtained by dynamic linker.

Figure 2. A function-level hardware/software co-design flow

3.2. An Integrated Development Platform for RSoC

The structure of a graphical integrated development platform supports the above co-design flow is depicted in Figure 3. And major modules of this platform are as follows:

**user interface**: provides graphical environment (as Figure 4) for designing functional specification and partitioning algorithm. In system specification mode (in real lines), designers input the code of functional specification and performance constraints. And in algorithm design mode (in dashed lines), designers input the code of partitioning algorithm and performance monitoring. A project is managed by a .project file, which provides the basic elements of software synthesis.

**compiler**: compiles the specification program into executable files and the partitioning algorithm into shared library. Converts the .project file into Makefile, and then invokes GCC (GNU C Compiler) to compile C code.

**CFF wrapper**: wraps the configuration bit files and physical address of hardware accelerator into the configuration files (.cif, Configure File Format).

**debugger**: use xmd, gdb tools to debug the performance constraints file on RSoC.

On this platform, functional specification and partitioning algorithm design can be carried out respectively in different flow.
3.3. RSoC Architecture

The architecture depicted in Figure 5 can commendably support the function-level programming model. A microprocessor and some other dedicated processing modules are integrated into reconfigurable logic module. Dedicated processing modules communicate with the microprocessor through shared bus or some particular internal networks. Relying on the ability of reconfiguration, dedicated modules will be configured dynamically. Hence a control and manage module is necessary to dynamically add or delete functional modules and realize the internal communication after reconfiguration. The dedicated hardware modules can be IP cores, or user defined function modules.

4. Hardware/software Co-Design Methodology

4.1. Hardware/Software Co-Function-Library Design

Our ultimate goal is not to develop a design platform for universal RSoC applications, because nowadays’ embedded applications are becoming so complex and diverse that it is not possible to quantitatively characterize in the early design phase. We propose a novel co-design strategy based on the above-mentioned programming model and choose image enhancement as our study case, which include a large number of repeated operations. According to the characteristics of the most frequently repeated computation, functions in the co-function-library are categorized to optimize the efficiency of applications. Functional specification and all software functions in the co-function-library are coded by high-level language, and hardware descriptions are encapsulated as software functions. The definition of hardware module includes two parts: a configuration file stored in a non-volatile memory which implements the basic function and software code which implements module invocation by microprocessor. To properly encapsulate a hardware module, a hardware interface function should be able to access these two parts after compiled. The workflow of this interface is depicted in Figure 6. First, configure hardware accelerators onto reconfigurable device by the configuration system call and then finish address mapping. After that, the interface function will get the ports number of hardware module from hardware resource manager, send the input and output parameters to the corresponding ports, and activate the hardware modules. When an accelerator finishes computing, an
anti-configuration system call will be invoked. It is important to note that address mapping is necessary because programs are using virtual addresses in GNU Linux operating system while hardware modules can only distinguish physical addresses.

Figure 6. Workflow of the hardware function interface

Hardware function can be either a slave device initialized by CPU or a master device which accesses the memory to automatically read and write parameters. For data synchronization, we have designed two state machines in the hardware interface. A user state machine interacts with user logic and a system state machine interacts with other modules. They can use on-chip memories to achieve data synchronization.

Hardware and software functions have different names so that applications could automatically select the operating mode of one function considering the current execution information. Meanwhile, using the same parameter list and return type for both hardware and software functions can provide the interface for dynamic link control.

4.2. Dynamic Link Control

Functional specification coded by C language can be compiled without modifying the current execution environment and compiler. However, a compiled executable file will only call software functions from the co-function-library because interfaces that the co-function-library provides only have the software function names. To solve this problem, hardware and software functions must be compiled into a shared library (dynamic link library). While functional specification is being compiled, the functions in the shared library will not be embedded in the executable file, but leaves a tag of function call. When the loaded executable file is executed, a jump will be made to the dynamic linker automatically for searching the implementation of a function in the shared library according to its name and other parameters. Then the dynamic linker gets the partitioning result from hardware/software partitioning algorithm to determine the linking object. Furthermore, by the help of dynamic linking mechanism, we can efficiently achieve different performance targets by using different hardware/software partitioning algorithms.

And for functional specification coded by Java language, we achieve a run-time reconfigurable Java virtual machine by modifying the Kaffe virtual machine as introduced in paper [15]. Hardware functions are packaged through the JNI interface to use dynamic link for hardware and software methods.

5. Run-Time Dynamic Hardware-Software Partitioning

5.1. Description of the Problem

In the above-mentioned hardware/software programming model, hardware/software partitioning algorithm will determine the implementation mode of the invoked abstract functions at run-time. First of all, definitions of related concepts are as follows: The input of the problem is expressed by a triad \( <F, \text{Area}_{\text{soft}}, T_{\text{all, software}} > \). \( \text{Area}_{\text{soft}} \) represents the largest area of available hardware; \( T_{\text{all, software}} \) represents the pure software execution time of one task; \( F \) corresponds to the paired hardware/software function, which comprises a quaternion group \( <C, \text{Time}_{\text{com}}, \text{HW}, X_{\text{sw}} > \). \( C \) corresponds to the times of a paired function has been called. \( \text{Time}_{\text{com}} \) represents the software execution time. \( X_{\text{sw}} \) represents the partitioning result of the current function. \( X_{\text{sw}} = 0 \) or 1, if \( X_{\text{sw}} = 1 \), the current function \( i \) will be executed on hardware, otherwise on software; \( \text{HW} \) represents implementation by hardware, which comprises \( <\text{Area}_{\text{hw}}, \text{Comm}_{\text{hw}}, \text{Time}_{\text{hw}}, T_{\text{hw}} > \). \( \text{Area}_{\text{hw}} \) represents the area occupied by a hardware function; \( \text{Comm}_{\text{hw}} \) represents the times of a function has been executed on hardware; \( \text{Comm}_{\text{hw}} \) represents the communication cost of a hardware function; \( \text{Time}_{\text{hw}} \) represents the hardware execution time; \( T_{\text{hw}} \) corresponds to the hardware configuration time. And the goal of the algorithm is to minimize the value of the following formula.

\[
\text{Minimize } T_{\text{all, software}} - \sum_{f} C_{\text{hw,sw}} \cdot (\text{Time}_{\text{sw}, f} - \text{Time}_{\text{com}, f} - \text{Comm}_{\text{hw}})
\]

\( C_{\text{hw,sw}} \) represents the times of functions have been executed on hardware;

\( \text{Time}_{\text{sw}, f} \) represents execution time of the software functions in set \( F \);
\( Time_{hw,now} \) represents execution time of the hardware functions in set \( F \);
\( Comm_{hw,now} \) represents communication time of the hardware functions in set \( F \);

Due to the limitation of hardware resources, resources occupied by a task must meet the certain constraints: at any time \( j \),
\[ \sum_{i \in F} X_{now, i} \cdot Area_{now} \leq Area_{total} . \]

**5.2. Partitioning Algorithm Design**

The above question can be regarded as a 0/1 knapsack problem, and be optimized by greedy rules. The partitioning result for each function will be affected by many factors. According to these factors, the tendency of a being partitioned function can be calculated as:
\[ F_i = f(X_{now, i} \cdot Time_{now, i} \cdot Time_{now, now} \cdot Comm_{now, i} \cdot Area_{now, i} \cdot C_{i, now} \cdot C) \quad i \in F \]

And the speed up ratio is the determinant of a function’s partitioning trend:
\[ f = \left\{ \frac{C_i \cdot Time_{now, i}}{(1 - X_{now, i}) \cdot T_{now, i} + C_i \cdot (Time_{now, now} + \text{Comm}_i)} \right\} \quad i \in F \]

\( C_i \) will be set 0 after a round of partitioning is finished, and the times of function \( i \) has been called will be recounted. The greater the value of \( f \), the higher possibility the function will be partitioned to hardware implementation. And the frequently called functions are more inclined to be partitioned to hardware implementation. Matrix \( C \) records the order of function call. \( C_{ij} \) represents the times of function \( j \) has been called immediately after function \( i \). The value of \( F_i \) will be calculated in each round of partitioning, and decide which function will be implemented by hardware.

Obviously, the above algorithm can be adjusted in terms of current system running state, because the value of \( x_i \) may be changed at run-time. It takes a certain period of time to attain the system run-time information, so the algorithm can run cyclically. In addition, the hardware configuration is time-consuming, the pre-configuration mechanism could be considered in the algorithm. If the current function \( i \) is partitioned to hardware implementation, function \( k \) which satisfied all the following conditions can be pre-configured on hardware.

\[
\begin{align*}
C_{ik} = \text{Max}(C_i) \\
X_{now, i} = 0 \\
Area_{now, i} \leq Area_{now, j} \\
i, j, k \in F
\end{align*}
\]

\( Area_{now} \) represents current usable hardware area. And the process of function-level hardware and software partitioning combined with pre-configuration mechanism is depicted in Figure 7.

Let \( p \) be the current partitioning result, \( p' \) be the new result, and \( T \) be the time function of \( p \). If \( T(p') \leq T(P) \), the new partitioning result is a better solution, otherwise discards the new partitioning result and maintain the original result. And through the evaluation process, the partitioning results will be continuously optimized.

**5.3. Algorithm Analysis**

Suppose that \( n \) functions in the co-function-library have been called in an application, and \( m \) functions \((m \leq n)\) in list are ready to be partitioned. Generally the time complexity of finding and deleting operations on function queue can be maintained at \( O(\log n) \), thus the complexity of the algorithm can be maintained at \( O(n \log n) \). As \( m \leq n \), the worst time complexity of the algorithm can be maintained at \( O(n \log n) \). Each paired function in the queue contains a number of fixed-size attribute values. Assuming that the space occupied by each paired function is \( k \), the space complexity of the algorithm is \( O(Km) \). As \( m \leq n \), the worst space complexity of the algorithm can be maintained at \( O(Kn) \).
5.4. Run-Time Partitioning Process

The entire run-time data flow is depicted in Figure 8. This data structure includes all parameters needed by the hardware/software partitioning algorithm, such as hardware and software function execution time, the times of functions have been called, the area of hardware functions, etc. Dynamic linker selects each function’s execution mode according to the partitioning result (on microprocessor or FPGA). And resource manager, a device driver, takes charge of the communications with the programmable hardware configuration controller, which inquires the needed hardware module state, configures the hardware, and updates records.

6. Experiments and Performance Analysis

6.1. Hardware/Software Functions Performance

Xilinx Virtex-II Pro XC2VP30 FPGA which supports partial dynamic reconfiguration is selected as our experimental platform. There are two tasks running at the same time: encryption and decryption functions of 3DES use one hardware accelerator, the configuration time is 3.0179526 seconds; encoding and decoding functions of Hamming use another accelerator, the configuration time is 2.3684901 seconds. Test 1K or 1M data process. The program time-consuming under different configuration is depicted in Figure 9. In the diagram, static represents accelerators has been configured before program is running; dynamic represents accelerators will be configured at run-time; all sw represents pure software implementation; all hw represents pure hardware implementation; 3des hw represents encryption/decryption function of 3DES using pure hardware implementation but hamming encoder/decoder using pure software implementation; hamming hw is just the opposite; random hw represents the implementation of the four functions is random.

The pure software implementation has the worst performance because of the very slow software execution speed. Dynamically allocated hardware implementation is almost the same high performance as static configuration when dealing with larger data, and one order of magnitude faster than the pure software implementation at least, because: 1. DES speedup and execution time is greater than the Hamming; 2. compared to the running time, 3DES configuration time can be ignored.

6.2. Run-Time Dynamic Hardware/Software Partitioning Algorithm Performance

The time consumption of three ways to achieve JPEG coding is depicted in Figure 10. The performance of hardware/software partitioning without dynamic reconfiguration (static) is the worst. The performance of partitioning with dynamic reconfiguration (dynamic reconfiguration) increases 9.93 percent than the former. With pre-configuration mechanism, the performance of function-level partitioning (per-configured) increases 18.44 percent than the partitioning without dynamic reconfiguration and increases 9.45 percent than the partitioning with the support of dynamic reconfiguration.

7. Conclusions

This paper proposes a new design methodology of reconfigurable system-on-chip based on function-level programming model. In this way, we built a graphical
development platform that provides an automatic design flow from the functional specification to system implementation for designers. On this platform, system designers can design run-time partitioning algorithms and easily specify the system by calling hardware/software functions in the co-function-library, which has been pre-customized and optimized for high-frequency computing tasks.

In view of the advantages of the function-level programming model and characteristics of reconfigurable device, we put forward a run-time hardware/software partitioning algorithm, which can adjust its partitioning scheme according to the run-time system information, make full use of pre-configuration mechanism and significantly improve the partitioning efficiency. Finally, experiments have proved the whole design method is feasible and effective.

Acknowledgments

This work was supported by the 863 National Hi-Tech Research and Development Program of China under Contract 2007AA01Z104.

References