An Embedded SOPC System Using Automation Design

Qingxu Deng, Hai Xu, Shuisheng Wei, Yu Han, Ge Yu
Department of Computer Science and Engineering
Northeastern University, China
dengqx@mail.neu.edu.cn, xhsoldier@163.com

Abstract

This paper presents an automation design-aided system for System-on-Programmable-Chip (SOPC). Platform-based design for SOPC issues become critical as implementation technology evolves towards complex integrated circuits and the time-to-market pressure continues relentlessly. Hence, new methodologies that emphasize Re-use and re-configurable are essential for embedded system design when using SOPC. We present a methodology that provides a platform automatically, which includes system level design specification, HW/SW partitioning, HW/SW optimization and system platform providing. The methodology can shorten the developing period and reduce power consumption. A case study on how to provide such a platform is also presented.

1. Introduction

With the development of embedded technology, Modern design requires a designer to have a unified view of software and hardware, seeing them not as completely different domains, but as two implementation options along a continuum of options varying in their design metrics, especially time-to-market, performance and power consumption. In recent years, the software and hardware co-design emerges with the constant development of design technology. First, an overall system analysis is needed; second, it is described in a specific computer language (generally C language) in spite of whether each part is implemented in software or hardware; last, simulation is required for an appropriate HW/SW partition. If the partition can’t satisfy the requirements, re-partitioning and simulation can be performed to yield a better partition. There are several mature co-design methodologies: POLIS, CORSAIR, COOL, etc. These methodologies are composed of systematic behavior description, cost estimation, partition of software and hardware, synthesize and simulate it to approach the rapid prototype of the board (except COOL). However, the methodologies mentioned above obviously have many shortcomings in common. During the stage of HW/SW partitioning, they only consider which tasks should be implemented in hardware or software, while task scheduling and hardware optimizing are ignored and the partitioning is manual. Although HW/SW co-design improves the development efficiency of embedded design to some extent, it is far from satisfying requirements. For example, the emerging of new products always has only several months in consumer electronics—cell phones, pagers, digital cameras, camcorders, videocassette recorders, portable video games, calculators, personal digital assistants, etc. These co-design methodologies cannot meet the increasing design demands. In order to speed up development processes, a new generation of design methodology called platform-based design emerged.

Platform based design is a new methodology in the field of embedded system co-design, which is based on co-design and aims at providing a system platform to shorten developing period for every kind of applications. Such platforms help a designer evaluate, simulate and program during the design stage, which will meet all the design metrics demands. We believe that the automation design of embedded system is the trend of embedded system design technology and the platform-based design is the foundation of it. Automation design of embedded system includes the following aspects: analyzing the system automatically (using FSM, UML etc.); building a mathematical model for the design metrics (Execution time, power etc.) and offering a platform scheme; simulating, synthesizing (the automation of synthesis and simulation) and optimizing the system platform to realize and compile the concrete codes (building general codes automatically for the system analysis). At present, these design aspects are immature. There is neither a unified methodology that can build such a system platform nor a modeling methods that can suit for the platform based design.
Especially, there is no special-purpose real-time operating system with HW/SW co-scheduling. Our goal is to build a system with HW/SW co-design and platform based design to achieve automatic design of embedded system.

The remainder of this paper is organized as follows. Section 2 presents a survey of related work in co-design and platform based design. Section 3 shows an implementation of the overall scheme. In section 4, modeling and formalization descriptions are presented. Section 5 describes how to build the Mathematical model of the hardware platform. The design and implementation of the real-time operating system with HW/SW co-scheduling are presented in Section 6. Section 7 gives a case study and concludes the paper.

2. Related Work

In this section, we briefly survey major related work in the area of the HW/SW co-design and platform based design. Frank Valid and Tony Givargis [1] proposed the HW/SW Co-design and provided the basic conceptions in this book. Balarin et al [2] put forward the POLIS which is a methodology of embedded system overall design based on HW/SW Co-design. K. Keutzer, et al [3] gave some conceptions of platform-based design and some questions that need special attention. Related aspects are described in [4], which is on the design of a wireless network platform with low power consumption.

Partitioning the HW/SW is most important in the design of platform-based design. HW-SW partitioning is an extensively studied “hard” problem with a plethora of approaches—dynamic programming [5], genetic algorithms [6], greedy heuristic algorithms [10], etc. Most of the initial work, [5], [7], focused on the problem of meeting timing constraints with a secondary goal of minimizing the amount of hardware. Subsequently there has been a significant amount of work on optimizing performance under area constraints, [8], [9], [10].

For the methods of partitioning HW/SW, some are partitioned at source code level since the tasks are generally pre-defined; some are partitioned at binary level, de-compile the binary codes to extract small parts that are frequently executed for implementation in hardware [11], [12].

We use a real-time operating system with HW/SW co-scheduling for the software platform. This is on how to co-schedule HW/SW in the OS and partition FPGA. The algorithms of task scheduling in the FPGA are as follows. Bazargan et al. [13] addressed the issue of placing application mappings onto a single device for hardware execution in a re-configurable computing system. They present a placement method that can be applied to dynamically adaptive hardware. The solution offered is a hybrid between the typical first-fit in a finite space and best-fit algorithms and trades quality of placement for speed. Walder et al. [14] combined an enhanced form of Bazargan’s partitioning algorithm and a placement-finding algorithm using 2D-hashing. The placement finding algorithm has linear time complexity. The update of the hash matrix is a quadratic time complexity in the worst case. The heuristic algorithm of partitioning FPGA [15] emphasized how to make different width of hardware tasks suit for reasonable partitions.

3. Main principle

The current embedded system design methodologies don’t provide the system platform, and only the designers with rich experiences can do HW/SW partitioning. Because of the shortcomings of the current embedded system design methodologies mentioned above, we present a methodology which is used for helping to design system platform. This methodology includes system modeling, formal description and system platform providing. System modeling and formal description means analyzing the system by FSM, EFSM and UML and decomposing it into many small tasks. Afterwards, we partition the tasks, decide the basic construction of hardware and provide an appropriate hardware platform automatically for the designers. As for software platform, we design and implement a real-time operating system with HW/SW co-scheduling. The main features of this operating system include supporting the POSIX1.1b and HW/SW co-scheduling. We will discuss this operating system in detail in section 6.

We believed that embedded system design will be automatic in future, and we present a methodology, which can provide an optimal system platform for designers. When the user submits the function and other constraints for the embedded system, a system platform which meet the user’s requirements can be given by using the tool-chain developed in light of our methodology mentioned above. Our aim is to design and implement embedded system platform automatically for SOPC, as Figure 1.

The basic hardware platform of the system
platform is composed of logic programmable chip, such as FPGA, CPLD etc and MCU as MIPS, ARM etc., or the more popular chips—SOPC as the Xilinx Virtex-4. User can download a general-purpose processor for CPU and other fields for FPGA. When designing an embedded system, basic hardware platform will not meet the design metrics for some occasions, so we will choose other single-purpose processors instead of FPGA. For examples, we will choose single-purpose processors such as DSP, CCD, etc. when performance or power consumption requirements are not satisfied. Therefore, this methodology needs a hardware library, which is used for storing chips and its abstract description, such as performance, power, cost, etc., and adding new emerging chips.

The software platform of the system platform is based on the real-time operating system with HW/SW co-scheduling that we have designed. Designers can add TCP/IP stack or device drivers to it for different requirements.

4. Modeling and Formalization Description

At the beginning of Co-design, the designer must describe the functions and the requirements of the system. This stage is called System Description. Two different methods are available. One is homogeneous modeling that describes the whole system with a uniform language and disregards concrete realization. Co-design with homogeneous modeling must analyze and decide which parts of the system should be implemented in hardware and which in software. The other is heterogeneous modeling that describes the hardware and software separately with different description languages. The analysis and verification of the systematic function tend to become difficult in the second method [16]. The three methodologies mentioned in section 3 adopt homogeneous modeling which is the trend in the future.

The system description of CORSAIR uses SDL (Specification and Description Language) and PMSC (Performance Message Sequence Chart) together to model the system. CORSAIR uses SDL, which is a systemic specification and description language based on object-oriented EFSM (Extended Finite State Machine) to describe behavioral function, while the systematic performance demands are described in PMSC. Combining SDL with PMSC is called SLD*[17]. COOL uses a subset of VHDL as its systemic description language. In Polis, designer uses a certain high-level language with EFSM (Esterel, VHDL, Verilog) to describe the system. The computation models of these methodologies mentioned above are all limited on EFSM basically, which are used in control or datapath applications. We present an improved method that offers a library of computation model. The designer can appoint computation models in a unified framework, hence to gain more freedom in the design process.

At present, UML (Unified Modeling Language) is a very promising and powerful tool since the systematic modeling language in designing embedded system. UML is object-oriented and the object is a cohesive entity, and it includes the attribute, behavior, state and relation among other objects and environment.

Chonlameth introduces UML in constraint-based co-design, which offers a good design method to it. But this method still has its weakness and needs to be improved.

Chonlanmeth adopts the use case in the initial analysis of embedded systematic design, then transforms them into activity diagrams and refines the important concepts and restrictions in every step of the activity diagram, which has reflected the advantages in combining UML with co-design. This idea is legible for simple embedded system, but it is not sufficient for complex embedded system. Because the active diagram is mainly applied in serial flow control, it is unsuitable to show the cooperation among the objects and the operation situation of the objects during their life cycles. Though active diagram is useful in describing the embedded system, the sequence diagram
and collaboration diagram are better in reflecting the characteristic of the embedded system. By adopting sequence diagrams, UML can express sequent time, relative time and absolute time, and show the time demand of the context. At the same time, it can support the control or datapath application, and it has overcome the limitation of other description languages.

In addition, to support the expression of general time demand (irrelevant environment, system whole life cycle) in system description stage [18], we expand UML to support it. So we can describe the interaction between context and system, and the mutual relationship among systemic objects without considering how to meet the requirements on execution time of state diagrams.

On the other side, there is a direct corresponding relationship between UML and SystemC which is a very promising design language currently. It is undoubted that it will improve the modeling method greatly and reduce the complexity of the system design.

5. Mathematical modeling of the hardware platform

The system can be separated to N tasks (a task may be composed of several functions or a function may be divided into several tasks, it is a slice of codes with relative independence). The hardware platform is divided into two parts: partitioning the hardware—its aim is to decide which tasks must run in hardware and which tasks may run in software or hardware; optimizing the hardware—its aim is to decide how to arrange the tasks that can run in hardware or software whether on earth they should run in hardware or software.

We present a primary Mathematical model for optimizing the hardware. In order to simplify the problem, we assume that there are N tasks \( T \{ T1, T2, \ldots Tn \} \) in the system, the tasks are serial, no commutation exists, and the design metrics that the user requires are that Energy should be less than \( Wr \) and the time for all tasks executing should be less than \( tr \). We assume that the tasks are listed in order—this can be done by software simulating. We assume that \( Ti > Tj \) if the running time of \( Ti \) is longer than \( Tj \). Then the problem is changed to how to decide a point \( n1 \) in \( T \). The tasks before \( n1 \) run in software and the tasks after \( n1 \) run in hardware.

\[
\begin{align*}
n1 &= \max\{n1 \mid W_{\text{total}} \leq Wr \text{ and } t_{\text{total}} \leq tr \}\quad W_{\text{total}} \text{ and } t_{\text{total}} \text{ are functions of } n1. 
\end{align*}
\]

This function can be decomposed to two sub-functions:

\[
\begin{align*}
\max\{n1 \mid W_{\text{total}} \leq W_r \} &= n1 = F(W_r, W_{\text{total}}) \\
\max\{n1 \mid t_{\text{total}} \leq t_r \} &= n1 = G(t_r, t_{\text{total}}) \\
n1 &= \min\{F(W_r, W_{\text{total}}), G(t_r, t_{\text{total}})\}
\end{align*}
\]

Procedure of resolving the functions:

\( W_{\text{sw}} \) is the energy consumed by CPU, \( W_{\text{hw}} \) is the energy consumed by FPGA, \( P_{\text{sw(idle)}} \) is the power of CPU when it is idle, \( P_{\text{sw(active)}} \) is the power of CPU when it is active, \( P_{\text{hw(static)}} \) is the power of FPGA when it is static, \( P_{\text{hw(active)}} \) is the power of FPGA when it is active, \( t_{\text{idle}} \) is the time of CPU when it is idle, \( t_{\text{active}} \) is the time of CPU when it is active, \( t_{\text{total}} \) is the time of FPGA when it is active, \( t_{\text{total}} \) is the time of FPGA when it is active and idle.

\[
\begin{align*}
W_{\text{total}} &= W_{\text{sw}} + W_{\text{hw}} \leq P_r \\
W_{\text{sw}} &= (P_{\text{sw(idle)}} * t_{\text{idle}} + P_{\text{sw(active)}} * t_{\text{active}}) / (t_{\text{idle}} + t_{\text{active}}) \\
W_{\text{hw}} &= (P_{\text{hw(active)}} * t_{\text{active}} + P_{\text{hw(static)}} * t_{\text{total}}) / t_{\text{total}} \\
\text{According to statistics:} \\
W_{\text{hw}} &= 1/2V^2f^\text{ciu}i\text{fi} = 5.9\mu W/\text{MHz} \\
\text{According to the assumptions above:} \\
t_{\text{total}} &= t_{\text{active}} + t_{\text{total}} + t_{\text{total}} \\
\text{Considering the cost metric:} \\
n1 &= \max\{n1 \mid W_{\text{total}} \leq W_r \text{ and } t_{\text{total}} \leq t_r \text{ and } S_{\text{total}} \leq S_r \}\quad S_{\text{total}} \text{ is the real cost and } S_r \text{ is the user’s requirement, } S_{\text{total}} \text{ is also the function of } n1.
\end{align*}
\]

The equations may yield no solutions in some situation. We will choose the single-purpose processor instead of FPGA when the equations have no solutions.

Designing such a system is complicated but effective. It will shorten the developing cycle and make the designing of embedded system no longer monopolized only by the experienced designers, and a common designer may develop a fairly good embedded system by adopting our methodology.

6. Designing and Realizing the Operation System with HW/SW Co-scheduling

Today, the increasing densities and reconfiguration modes of SRAM-based field-programmable gate arrays (FPGAs) and
configurable systems on a chip (CSoCs) advocate more dynamic uses of these components. Many promising application domains for re-configurable embedded systems, such as wearable computing, mobile systems, and network processors, combine high performance demands with frequent application changes. These impose strict demands on system implementation technology. High performance requires application-specific architectures, but flexibility and system agility require a programmable, adaptable approach.

Commercial RTOSs available for popular embedded processors enable the development of software-based real-time systems and significantly shorten the design period. But they typically take no advantage of hardware to implement any of their functions, probably because processors and custom hardware accelerators have historically resided on separate chips. Although sufficient for a limited number of RISC/DSP processors, these pure software kernels are of limited use for heterogeneous, dynamically re-configurable designs.

Because of the different advantages of hardware and software implementations, most systems consist of a combination of the two. Software, at the lowest level, consists of a sequence of instructions, and obviously requires a processor to interpret and execute them, and memory is used to store instructions and data. In addition, some parts of the function may be implemented directly on hardware, interacting with the software at appropriate points. The system design then concentrates on which parts are to be implemented in software and which in hardware.

The main problem in hardware/software co-design is how to design an embedded system that contains both hardware in the form of FPGA or ASICs and a microprocessor for which software must be written. FPGA technology has the potential to play a vital role in bridging the traditional divide between hardware and software. We construct a system platform which is composed of a set of these re-configurable hardware blocks and of instruction-set processors (ISP) can be used to combine two important assets: the flexibility of software and the performance of hardware, and to facilitate a natural style of hardware/software co-design embedded systems.

Gordon Brebner explores the issues involved in the resources management, and discusses the problems involved in presenting a software-oriented user with a larger virtual hardware resource that is implemented using smaller physical FPGA hardware, and puts forward a prototype operating system [23]. Herbert Walder and Marco Platzner further the study of re-configurable hardware operating systems in a top-down manner [24]. But they don’t provide feasible and unified hardware and software platform. To satisfy the hardware platform, we design and realize a real-time operation system with HW/SW co-scheduling in the platform-based design.

![Figure 2. Real-time operating system with HW/SW co-scheduling](image1)

**Figure 2. Real-time operating system with HW/SW co-scheduling**

A real-time operating system with HW/SW co-scheduling forms an abstraction that hides the details of the underlying technology from the developer. It can exert the flexibility of software, and facilitate a natural style of hardware/software co-design and design automation. The most important features are fast configuration and read-back capability. The execution semantics of HW/SW RTOS cross software and hardware and also provides the ability to dynamically reconfigure the processing structure implemented on both hardware and software. In our opinion, platform-based design must pay more attention to software in the platform definition process. The consequence for system platform shows that...
SOPC architectures can benefit greatly from early co-design with a HW/SW RTOS. The operating system objects and the task abstraction facilitate the re-use of tested and reliable code and circuitry. This can considerably speed up development cycles and shorten time-to-market. Porting applications is greatly simplified by operating systems that are available for different target platforms. The system can be repartitioned between different hardware components or between hardware and software components. A task that was previously implemented in software running on a CPU could be mapped to an FPGA to increase its performance. The partition and usage of FPGA area is shown in Figure 2.

In addition, we also consider to realize the enlighten algorithm in the OS between several current algorithms of partitioning the FPGA area. In the previous sections, we introduce how to partition HW/SW tasks and confirm which tasks should be implemented in hardware and which in software, but we adopt de-compiling technology to partition HW/SW dynamically when considering the restrictions of different kinds of commercial tools, as illustrated in Figure 3.

### 7. Experiments and conclusions

We use Memec Virtex-4 LC Development Kit in our experiment. The clock speed for the MIPS is 100 MHz at a supply voltage of 1.2V, and we used Xilinx’s Virtex Power Estimator [22] to estimate power for each example. Here, we use an operating system, and this cause the result different from those of no operating system adopting as for most of the test cases.

We examined several examples from Motorola’s Powerstone [21] benchmark suite: a voice encoder (adpcm), a cyclic redundancy check (crc), a data encryption standard (des), an engine controller (engine), a fax decoder (g3fax), a JPEG decoder (jpeg), a handwriting recognizer (summin), and a modem encoder/decoder (v42). We implement each example, using the input vectors in Powerstone, on an instruction set simulator for an MIPS microprocessor, augmented to output instruction traces.

Table 1 summarizes the relevant loop data for our benchmarks. Size indicates the total number of instructions in the program, and Loop Instr is the number of instructions in the region(s) moved to hardware. Loop Time is the percentage of total execution time taken by the region(s). CSL Size is the number of configurable logic blocks required by those

<table>
<thead>
<tr>
<th>Example</th>
<th>Performance (cycles)</th>
<th>Power (W)</th>
<th>% Energy Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>g3fax</td>
<td>7,800,000</td>
<td>3,560,000</td>
<td>599,500</td>
</tr>
<tr>
<td>adpcm</td>
<td>1,740,000</td>
<td>1,010,000</td>
<td>239,000</td>
</tr>
<tr>
<td>crc</td>
<td>71,000</td>
<td>35,310</td>
<td>23,200</td>
</tr>
<tr>
<td>des</td>
<td>143,000</td>
<td>37,520</td>
<td>18,200</td>
</tr>
<tr>
<td>engine</td>
<td>405,000</td>
<td>123,000</td>
<td>85,500</td>
</tr>
<tr>
<td>jpeg</td>
<td>3,500,000</td>
<td>85,500</td>
<td>37,520</td>
</tr>
<tr>
<td>summin</td>
<td>1,600,000</td>
<td>615,000</td>
<td>333,000</td>
</tr>
<tr>
<td>v42</td>
<td>1,925,000</td>
<td>423,000</td>
<td>108,000</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Benchmark loop information

<table>
<thead>
<tr>
<th>Example</th>
<th>Instruction Size</th>
<th>Loop Instr</th>
<th>Loop Time (%)</th>
<th>CSL Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>g3fax</td>
<td>450</td>
<td>540</td>
<td>1.34</td>
<td>450</td>
</tr>
<tr>
<td>adpcm</td>
<td>350</td>
<td>450</td>
<td>1.25</td>
<td>450</td>
</tr>
<tr>
<td>crc</td>
<td>350</td>
<td>450</td>
<td>1.25</td>
<td>450</td>
</tr>
<tr>
<td>des</td>
<td>350</td>
<td>450</td>
<td>1.25</td>
<td>450</td>
</tr>
<tr>
<td>engine</td>
<td>350</td>
<td>450</td>
<td>1.25</td>
<td>450</td>
</tr>
<tr>
<td>jpeg</td>
<td>350</td>
<td>450</td>
<td>1.25</td>
<td>450</td>
</tr>
<tr>
<td>summin</td>
<td>350</td>
<td>450</td>
<td>1.25</td>
<td>450</td>
</tr>
<tr>
<td>v42</td>
<td>350</td>
<td>450</td>
<td>1.25</td>
<td>450</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>1.25</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Benchmark test results

<table>
<thead>
<tr>
<th>Example</th>
<th>Performance (time)</th>
<th>Energy</th>
<th>Normal (time)</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>g3fax</td>
<td>1.44</td>
<td>0.0042</td>
<td>1.47</td>
<td>0.0042</td>
</tr>
</tbody>
</table>

Table 3. Test results with operating system supporting
regions.

The test results for our method are summarized in Table 2. The power column in the table presents power data for the microprocessor and for CSL when they are active. We estimated for the MIPS-based system that the interconnect power, namely the power consumed by the system buses and shared memory, would be about 0.1W. Furthermore, we are assuming a low-power state of 25% of the active state on the microprocessor, and the CSL’s low-power state consisted only of quiescent power, and thus the following equation is used to compute total power:

\[
\text{Total power} = \%\text{Sw} \times \text{Sw} + \%\text{CSL} \times (\text{CSL} + 0.25 \times \text{Sw}) + \text{Interconnect Power} + \text{Quiescent Power}
\]

In the equation, \%\text{Sw} is the percent of time spent in software, \%\text{CSL} is the percent time spent in the CSL, \text{Sw} is the power of the software when the microprocessor is active, and \text{CSL} is the power of the CSL when active. Note that the values shown for software power and CSL power in Table 2 do not include the interconnected and quiescent power. Interconnect and quiescent power are only included as part of the total power. \text{Sw} is the total number of cycles to execute the example completely in software. Loop in sw is the total cycles required by the loop when running in software. Loop in CSL is the number of cycles required by the loop when running in custom hardware. \text{Sw}/\text{CSL} is the number of cycles required to execute the entire program after partitioning. Speedup is the resulting speedup after partitioning.

Table 3 is the test results for all the examples executing at one time with the support of co-scheduling real-time operating system, compared with the normal real-time operating system. The Performance includes response time, configuration time and executing time.

The results of the experiments show that such a methodology is efficient. It produces an average energy savings of 26% for embedded-system benchmarks and accelerates about 1.44 times when there is no operating system supporting. When there is operating system supporting, our operating system is more efficient. We present a new methodology which is focused on embedded automatic design, and it combines almost all the aspects of embedded system design. The methodology can shorten the developing period, and is very efficient and convenient.

Acknowledgement

This work was partially supported by Natural Science Foundation of China (60473073), National 863 High-Tech Program (2004AA1112060).

Reference


