Energy-Efficient Architecture for Embedded Software with Hard Real-Time Requirements in Partial Reconfigurable Systems

Zhigang Gao, Guojun Dai, Peng Liu
College of Computer Science and Technology,
Hangzhou Dianzi University,
Hangzhou 310018, P.R.China
{gaozhigang, daigj, perryliu}@hdu.edu.cn

Peifeng Zhang
College of Computer Science,
Zhejiang University,
Hangzhou 310027, P.R.China
bavon@zju.edu.cn

***Abstract*—Embedded systems, because of their application-specific characteristic, are one of the most widely used domains for reconfigurable systems. Aiming at the energy-saving problem, this paper presents a dynamic scheduling architecture. The contribution of this paper is twofold: First, this paper presents an Energy-efficient Architecture for Embedded Software (EAES), which uses a processor with dynamic voltage scaling (DVS) capability and FPGA modules as the hardware platform, and extends directed acyclic graph with AND and OR relationship as the task model. Second, this paper presents a dynamic energy-saving method, which combines the pre-assignment of tasks with dynamic runtime scheduling, minimizing the energy consumption of tasks while meets their deadline requirements. This is work in progress. So far, we have designed and implemented the required critical hardware and software functions. Currently, we are focusing on the rest functions and system integration.**

Keywords-Reconfigurable computing; embedded systems; DVS; DAG; energy savings.

I. INTRODUCTION

Reconfigurable computing, as a new computing paradigm, has the advantage of both the high performance of hardware implementation and the flexibility of software implementation, and increasingly becomes popular in some computation-intensive (such as image processing, encryption, and so on) and customized applications.

For embedded systems, the correctness of a system not only depends on logical rightness, but also depends on the non-functional properties, such as real time, energy consumption, safety, etc. The functions implemented by FPGAs (Field Programmable Gate Arrays) can be easily customerized according to application requirements. Moreover, FPGAs can speed up program execution, reduce time to market, improve fault tolerance, and reduce energy consumption. Therefore, FPGAs are wildly used in embedded systems.

In the research on reconfigurable computing, system architecture is an important aspect, which is used to improve performance, flexibility, and scalability. Much research has been carried on architectures of reconfigurable systems [2-7]. The Garp Architecture [3] combines reconfigurable hardware with a standard MIPS (Microprocessor without Interlocked Piped Stages) processor on the same die in order to obtain the advantages of both general and specific processing components. The SyCERS architecture [4] presented by Amicucci et al. is based on the SystemC and targets the design of reconfigurable embedded systems. The Pleiades project [5] at UC Berkeley seeks to achieve ultra-low power high-performance multimedia computing through the reconfiguration of heterogeneous system modules. The RaPiD project [7] focuses on defining coarse-grained adaptable architectures that solve the performance/power/price constraints posed by mobile/embedded systems platforms.

Most of the existing research aims at the development phase in order to improve performance, flexibility, and correctness, but not aims at the runtime phase. For embedded systems, the timing and energy are two most important constraints. Dynamic power consumption contributes a significant sector in low-power design. Timing and energy requirements are effected by scheduling mode, running time of tasks, and execution manner (software tasks or hardware tasks), which is relevant on the runtime context. In this paper, we focus on the energy-saving problem for hard real-time embedded systems, and present an energy-saving architecture EAES. EAES first assigns tasks to a processor or a FPGA module. When tasks are executed, they are dynamically scheduled into a processor or a FPGA module according to the evaluation for timing and energy constraints.

The rest of this paper is organized as follows. Section II presents the system architecture of EAES. Section III describes the task model and energy model. Energy-efficient task scheduling in EAES is presented in section IV. The ongoing work is given in section V. Finally, this paper concludes with section VI.

II. SYSTEM ARCHITECTURE

In this section, we present the system architecture of EAES. Before describing system architecture, we first introduce the hardware platform used in the system architecture.

The hardware platform consists of two boards: the master board and the slave board. The master board consists of three parts: master processor (MP), I/O devices, and memory. The MP is a processor which supports DVS in order to save energy at runtime of software.

978-0-7695-3825-9/09 $26.00 © 2009 IEEE
DOI 10.1109/EmbeddedCom-ScalCom.2009.76

The slave board consists of the slave processor (SP), the FPGA modules, the I/O devices, the CF card, and the memory.

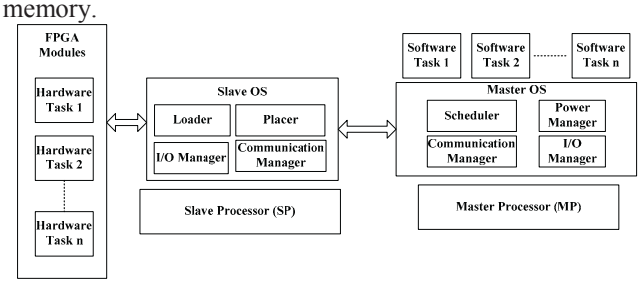

Figure 1. System platform.

The system architecture is shown in Figure 1. The right part of the system platform is the structure of the master processor. Its structure consists of three parts: Master Processor (MP), Master OS (operating system), and software tasks. The master OS is the manager of hardware resources and logical resources in the master board. It also provides the unified abstract for the tasks' running entity (software tasks or hardware tasks) and the energy-saving scheduling function. The scheduler is responsible for tasks' scheduling. The power manager sets the runtime voltage (i.e. the running frequency) of MP. The communication manager deals with the communication between MP and SP. The software task is the same as the task on normal embedded systems. The I/O manager manages the I/O devices on the master board. The left part of the system platform is the structure of the slave processor. Its structure consists of three parts: the Slave Processor (SP), Slave OS, and hardware tasks. The hardware tasks are those which are the equivalent of the software tasks, but they are implemented in hardware. Slave OS manages the hardware tasks and communicates with the master board. In Slave OS, the loader deals with the bitstream loaded to the FPGA module. The placer is responsible to find a suitable position to put a hardware task in a FPGA module. The communication manager deals with the communication between SP and MP. The I/O manager manages the I/O devices on the slave board.

III. TASK MODEL AND ENERGY MODEL

The directed acyclic graph (DAG) is wildly used to model tasks in reconfiguration research [17-18]. In a DAG $G(V, E)$, V is the set of nodes (i.e., the set of tasks), and E is the set of edges. The edges $E \subseteq V \times V$ represent the dependencies between tasks. An edge $e_{i,j}$ denotes the directed edge from the task τ_i to the task τ_j. Because a DAG does not describe tasks' characteristics and the communication cost between tasks in detail, in this paper, we present an extended DAG $EG(V^*, E^*)$, where V^* is the set of tasks, and the E^* is the set of edges. For $\tau_i, \tau_j \in V^*$, if $\tau_i \prec \tau_j$, τ_i is the *predecessor* of τ_j, and τ_j is the *successor* of τ_i; if there is no $\tau_k \in V^*$ which makes $\tau_i \prec \tau_k \prec \tau_j$, τ_i is the *immediate predecessor* of τ_j, and τ_j is the *immediate successor* of τ_i; if there is no $\tau_k \in V^*$ which makes $\tau_k \prec \tau_j$, τ_k is the *ancestor* of τ_j; if there is a $\tau_k \in V^*$ and there is no $\tau_j \in V^*$ making $\tau_k \prec \tau_j$, τ_k is a *leaf* of $EG(V^*, E^*)$. In an *EG*, a task may have more than one input edge and/or output edge, thus there are AND/OR relationship between tasks. Note that there may be more than one *leaf* in $EG(V^*, E^*)$. A task $\tau_i \in V^*$ is a eight-tuple { T_i, D_i, C_i^s , C_i^h , CF^i, P^{cf}, W, H }, where:

- T_i is the period of τ_i.
- D_i is the deadline of τ_i. If τ_i is a task with successor, its deadline is -1, representing without deadline; otherwise, its deadline (i.e. the end-to-end deadline in extended DAG) is a positive value.
- C_i^s is the worst-case execution time (WCET) of τ_i when τ_i is run on MP at the fully speed. When the working voltage (frequency) is reduced, the execution time of τ_i will be lengthened.
- C_i^h is the WCET of τ_i when τ_i is run on FPGA.
- CF^i is the configurable time of τ_i. In this paper, we assume it is a constant.
- P^{cf} is the energy consumption when configuring τ_i to FPGA. It is a constant.
- W is the width of the FPGA area occupied by τ_i when it is running as a hardware task.
- H is the height of the FPGA area occupied by τ_i when it is running as a hardware task.

We regard the software tasks and the hardware tasks as equivalent and transferable, so we use unique definition to denote a task wherever they run in MP or in a FPGA module.

In EG, an edge $e_i \in E^*$ is denoted as a three-tuple { CO_i^{in-s} , CO_i^{in-h} , CO_i^{ex} }, where CO_i^{in-s} is the communication overhead between software tasks; CO_i^{in-h} is the communication overhead between hardware tasks, CO_i^{ex} is the communication overhead between software tasks and hardware tasks. For simplicity, we assume both CO_i^{in-s} and CO_i^{in-h} are zero because they are in the same component, and CO_i^{ex} is a constant value.

Currently, most of processors are produced by CMOS technique. The power consumption for CMOS processors is dominated by dynamic power dissipation, and is defined as: $P = C_{eff} \cdot V_{dd}^2 \cdot f$ [8], where C_{eff} is the effective switching capacitance, V_{dd} is the supply voltage, and f is the operating frequency. We assume MP runs at a sequence of discrete operating frequencies with corresponding supply voltages. That is, MP has the parameters of $\{(f_1,V_1),\cdots, (f_{max},V_{max})\}$, where f is the operating frequency, and V is the supply voltage. The maximum power of MP, P^s_{max}, is the energy consumption of MP per cycle when MP is running at V_{max}. If MP is running on V_i, its power P_i is defined as $P_{max}V_i^2f_i/(V_{max}^2f_{max})$. Note that the WCET and the actual execution time (AET) of τ_i are all measured at the full speed of the processor where τ_i resides.

We assume the power of the FPGA in slave board when running a hardware task is P^h. The energy consumption of a

hardware task τ_i is $P^h \times e^h_i$, where e^h_i is the execution time of τ_i.

IV. ENERGY-EFFICIENT TASK SCHEDULING

In this section, we detail the scheduling process in EAES. When scheduling the tasks in externed DAG, it must meet the hard real-time requirements of tasks, and minimize the energy consumed by tasks.

The scheduling process in EAES consists of two phases: preplanning scheduling scheme and runtime scheduling. Both preplanning scheduling scheme and runtime scheduling have the goal of meeting timing requirements and minimal energy consumption. During preplanning scheduling scheme, each task is decided to be assigned to MP or FPGA, and this process is carried out offline. During runtime scheduling, tasks are decided whether to be executed in MP or in FPGA, and this process is carried out online by scheduler.

A. Preplanning Scheduling Scheme

Before present the scheduling scheme, we present three definitions which will be used.

Definition 1. If $\tau_i, \tau_j \in EG(V^*, E^*)$, and there is a path P1(τ_i, τ_{i+1},…, τ_m,…, τ_{j-1}, τ_j) where $\forall \tau_a, \tau_b \in$ P1, the following two conclusions hold: 1) $\tau_a \prec \tau_b$, or $\tau_b \prec \tau_a$, or τ_a and τ_b are in the AND path with the same predecessor and successor; 2) there is no another subpath P1`($\tau_i$, $\tau_{i+1}$, …,$\tau_n$,…, $\tau_{j-1}$, $\tau_j$) which makes $\sum_{\tau_q \in P1`} C^s_q < \sum_{\tau_k \in P1} C^s_k$. The P1 is called the *shortest path* between τ_i and τ_j.

Definition 2. If $\tau_i, \tau_j \in EG(V^*, E^*)$, and there is a path P1(τ_i, τ_{i+1}, …,τ_m,…, τ_{j-1}, τ_j) where $\forall \tau_a, \tau_b \in$ P1, The following two conclusions hold: 1) $\tau_a \prec \tau_b$, or $\tau_b \prec \tau_a$, or τ_a and τ_b are in the AND path with the same predecessor and successor; and 2) there has no another subpath P1`($\tau_i$, $\tau_{i+1}$, …,$\tau_n$,…, $\tau_{j-1}$, $\tau_j$) which makes $\sum_{\tau_k \in P1} C^s_k < \sum_{\tau_q \in P1`} C^s_q$. The P1 is called the *longest path* between τ_i and τ_j.

Definition 3. In $EG(V^*, E^*)$, assuming V_a is the set of ancestor tasks, and V_{lf} is the set of leaf tasks. For $\tau_{a1} \in V_a$ and $\tau_{b1} \in V_{lf}$, the set of all tasks in the possible path from τ_{a1} to τ_{b1} is called an *transaction* Tr(τ_{a1}, τ_{b1}). The longest path P1(τ_{a1}, τ_{a1+1}, …, τ_m,…, τ_{b1-1}, τ_{b1}) is called an t*ransaction's footprint*. All transactions' footprint in EG without repetition tasks is called the *transaction footprint set*.

In task scheduling with precedence constraints, list scheduling [13] is a wildly used scheduling method. In this paper, we use list scheduling to decide which tasks should be put into the ready queue. List scheduling can only keep the precedence constraints between tasks in EG, but it does not designate the execution order when multiple tasks are in the ready queue. In [12], Zhu et.al use longest-task-first (LTF) [14] to decide the execution order of task in the ready queue. Zhu et al. uses the same end-to-end deadline when an ancestor task has multiple corresponding leaf tasks. Moreover, they do not deal with the case when there are more than one ancestor task. In this paper, we consider the case when there are multiple ancestor tasks, and one ancestor task can have multiple leaf tasks with different deadlines. We assign the priority of tasks in the ready queue according to the following three rules:

1) For the tasks in the possible execution path of a transaction, smaller a transaction's deadlines is, higher the tasks' priority is. That is, assign tasks' priority by using the EDF (Earliest Deadline First) principle.
2) If a task is in the possible execution path of multiple transaction, its priority is the higher priority assigned by ruler 1.
3) For the tasks within the same transaction and be in the ready queue, longer a task's WCET is, higher the task's priority is. That is, assign tasks' priority by using the LTF (Longest Task First) principle.

During the preplanning scheduling scheme, it assigns tasks to MP or FPGA with the goal of meeting their deadlines requirements and minimizing the energy consumption of tasks. In an EG, because of complex relationship (such as precedence, AND, OR, etc), the execution of ancestor tasks may trigger multiple execution path, and may pass different execution path in different triggers. Because timing constraints is the foremost requirement for hard real-time systems, to guarantee tasks in EG could be scheduled under all execution path, we use the tasks contained in the transaction footprint set (denoted as TFS) of EG as the assigned tasks. As the tasks in different transaction are assigned priority according to EDF principle, if they are schedulable, other tasks in this transaction is schedulable. From the real-time scheduling theory, we know if it is schedulable when all transactions trigger at the same instant, the system is schedulable. In EAES, we regard the FPGA modules as an attaching component which could accelerate tasks' execution and reduce energy consumption in some case. Assigning tasks to MP or FPGA is a linear programming problem. We represent the constraints and the goal in equation (1)-(4).

$$A_{i,k} = 0,1, i = 1,2,\cdots,n; k = 1,2$$

$$A_{i,1} + A_{i,2} = 1, i = 1,2,\cdots,n \quad (1)$$

$$ET(Tr_i) \le D(Tr_i), i = 1,2,\cdots,m \quad (2)$$

$$\bigcup_{\tau_j \, in \, FPGA} A_j \subseteq A_{FPGA}, \forall t \quad (3)$$

$$E_{total} = \sum_{i=1}^{m} (P_{\max} \cdot C^s_i \cdot S_i) + \sum_{j=1}^{p} (P^h \cdot e^h_j + CF^j) \quad (4)$$

In equation (1), $A_{i,k}$ represents assigning the task τ_i to the execution component k. $A_{i,1}$=1 means τ_i is assigned to MP, and $A_{i,2}$=1 means τ_i is assigned to FPGA. $ET(Tr_i)$ is the completion time of Tr_i; D(Tr_i) is the deadline of Tr_i; A_j is the space occupied by the hardware task τ_j running at the instant t. A_{FPGA} is the schedulable area by placer in SP. Currently, there are many place algorithms have been presented. Because this paper does not focus on the space management

of FPGA, and the place algorithm also do not directly affect the result of this paper. The scheduler in SP can use a placement and space management algorithm presented in existing research papers, such as [19-20], etc.

The equation (1) means a task can either be assigned to MP or FPGA, but not both. The equation (2) means all transactions' completion time must be no more than their deadlines, i.e., the timing constraints of all transactions must be met. The equation (3) says the tasks assigned to FPGA at any time can be placed into FPGA by placer. The equation (4) is the energy function. In the right side of the equation (4), the first item is the energy consumption of software tasks (S_i is the energy scaling factor because of using DVS); the second item is the energy consumption of hardware tasks (it includes hardware tasks themselves' energy consumption, and the energy consumption when configuring them).

Equations (2)-(4) are dependent on the tasks' assignment and scheduling. The equation (2) and (3) can not calculate analytically because the tasks are scheduled by list scheduling and EDF heuristic. In this paper, we resolve the task assignment problem by three sequent steps: 1) generate a task assignment scheme satisfying the equation (1), use list scheduling and EDF heuristic to generate the scheduling of tasks in transaction footprint set; 2) verify whether equation (2) and equation (3) are met during scheduling generation. If either equation (2) or equation (3) could not be met, it shows this task assignment scheme is unsuitable, and we generate another task assignment scheme and verify them during generating scheduling; 3) if all tasks is schedulable, calculate the energy of tasks using equation (4).

Let's demonstrate the above steps by an example. Assuming there are two transactions Tr_1 and Tr_2, as shown in Figure 2. The Tr_1's footprint is τ_1, τ_2, τ_3, and τ_4, the Tr_2's footprint is τ_5, τ_6, τ_7, and τ_8.

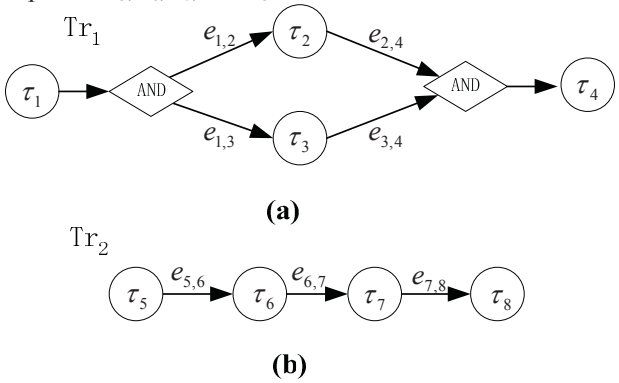

Figure 2. Two transaction Tr_1 and Tr_2.

If we generate a task assign scheme {$A_{1,1}$, $A_{2,2}$, $A_{3,2}$, $A_{4,1}$, $A_{5,1}$, $A_{6,1}$, $A_{7,2}$, $A_{8,1}$}, the task scheduling is shown in Figure 3. In Figure3, the horizon axis represents the time, and the vertical axis represents the execution speed (S) of tasks for software tasks or area for hardware tasks (i.e., the shadow area). From the Figure 3, we see the hardware tasks τ_2, τ_3, and τ_7, can be placed in FPGA, and the deadlines of the two transactions are meet.

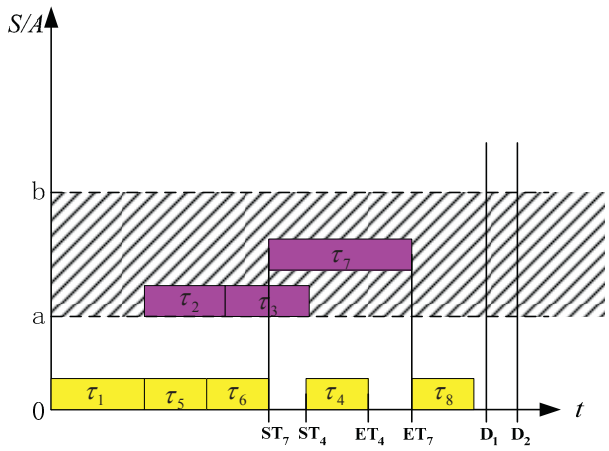

Figure 3. An scheduling scheme for Tr_1 and Tr_2.

After that, we calculate the energy savings of the scheduling scheme. We shift all software tasks forward as large as possible. If a task's immediate successor is a hardware task, its latest completion instant is the start instant of its successor in Figure 3; if a task has no immediate successor, its latest completion instant is the deadline of the transaction it belongs to. During tasks shift, the precedence relationship is kept. We don't shift hardware tasks in order to keep the scheduling result of placer. After task shift completion, we reduce software tasks' execution speed in order to save energy. The result of task shift and speed reduction for Figure 3 is shown in Figure 4. In Figure 4, τ_8 is shift forward, and there is some space for τ_4 to reduce its speed.

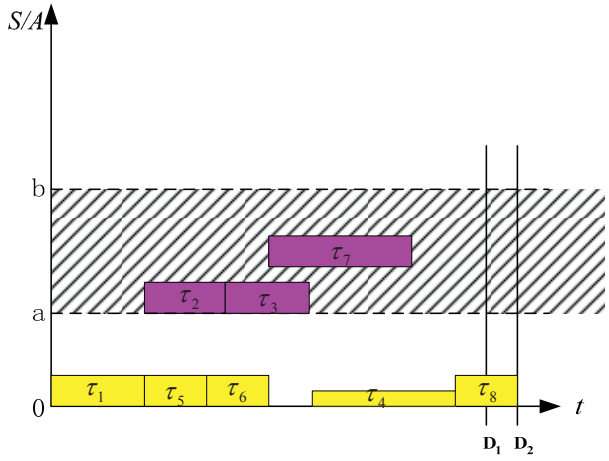

Figure 4. Energy savings for Tr_1 and Tr_2.

Note that there are some OR paths in EG, the tasks in OR paths beyond a transaction footprint must also be considered because they maybe pass at runtime. These tasks have shorter execution time, but their have different area requirements. For each OR path, we adjust the tasks' place position until they meet their transactions' deadline requirements.

B. Runtime Scheduling

In this phase, it reduces energy consumption by dynamic task assignment and dynamic slack reclaim and reuse. During the first phase, we assume tasks execute with their WCET, and all transactions arrive at the same time. However, in most case, tasks' actual execution time (AET) of tasks is less than their WCET [15, 16], which means there are dynamic slack can be reused to reduce the energy consumptions. Usually, transactions do not arrive at the same time, which means software tasks assigned to MP can be scheduled into FPGA in order to reduce energy consumption further. Before the scheduler in MP releases a software task τ_i, if its next task is a hardware task, it sends a preconfiguration command to SP with the parameter (i+1, C_i^s), where i+1 represents the task number, and C_i^s represents the WCET of τ_i. The placer in SP decides the placement position and configuration time. If continuous multiple tasks behind τ_i are hardware tasks, scheduler sends multiple commands to SP. If both τ_{i+1} and τ_{i+2} are software tasks, the scheduler sends the evaluation command with the parameter (i, 2) to SP for evaluating the energy savings effect when executing τ_{i+1} as a hardware task. If the inequation (5) holds.

$$P_{\max} \cdot (C_{i+1}^s + C_{i+2}^s) > P^h \cdot C_{i+1}^h + CF^{i+1} + P_{\max} \cdot C_{i+2}^s \cdot S \quad (5)$$

In inequation (8), the left side is the energy consumption when τ_{i+1} and τ_{i+2} are executed in MP, the right side is the energy consumption when τ_{i+1} is executed in FPGA and τ_{i+2} is executed in MP with reduced speed (where S is the energy scaling factor when τ_{i+2} reclaims and reuses the slack produced by τ_{i+1}). We do not consider the slack before τ_i because the scheduler uses greedy algorithm to reuse the slack time, and the unclaimed energy can also be claimed and reused by tasks behind τ_{i+2}. If SP returns a message that i+2 can be placed in FPGA, then MP believe it a pre-configurating hardware tasks.

In order to reduce energy savings of tasks, before τ_i is released, the scheduler uses the slack time reclaimed and the WCET of τ_i to calculate the lowest speed. After τ_i completes, the scheduler reclaims the slack according the execution time of τ_i in order to provide to the successor tasks.

V. ONGOING WORK

Currently, we are setting up the experimental hardware platform and implementing the operating system. We intend to use a Transmeta TM5800 processor as the MP to make up of the master board and a Virtex-4 FX ML410 as slave board. The TM5800 processor has finer voltage-scaling granularity, and provides a wider voltage adjusting scope. The Virtex-4 FX ML410 supports dynamic partial reconfigurable capability, and has abundant peripheral equipments, such as CF card, UARTs, USB, JTAG, etc. Currently, we have implemented the hardware task implement and provide the interface with OS. At the same time, we are extending the open source operating system μC/OS to support the power management and the hardware task management in order to use them into MP and SP. The scheduler's in master OS has completed, and the algorithm for energy reclamation and reuse has been implemented based on the extension for the algorithm in [21]. The loader in master OS has been completed.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presents an energy-efficient architecture for hard real-time embedded software EAES. EAES first assigns tasks to MP and FPGA with the goal of meeting hard real-time requirements of tasks and minimizing energy consumption. At runtime of tasks, EAES dynamically schedules tasks into FPGA, and reclaims and reuses slack time to reduce energy consumption further. Our future work is to complete the experimental platform and evaluate the performance of EAES.

ACKNOWLEDGMENT

This work has been supported by the National Natural Science Foundation of China (No. 60773042) and the Zhejiang Provincial Key Science and Technology Program (No. 2008C13076) and Zhejiang Provincial Major Science and Technology Projects (No. 2008C11108-1).

REFERENCES

[1] E.A. Lee, "Embedded Software," Advances in Computers. Vol. 56, London: Academic Press, 2002.

[2] I. Robertson and J. Irvine, "A design flow for partially reconfigurable hardware," ACM Transactions on Embedded Computing Systems, Vol. 3, No. 2, 2004.

[3] J. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," Proceedings of the 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines, 1997, pp. 12–21.

[4] C. Amicucci, F. Ferrandi, M. D. Santambrogio, and D. Sciuto, "SyCERS: a SystemC Design Exploration Framework for SoC Reconfigurable Architecture," Proc. 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms (ERSA 2006), 2006, pp. 63-69.

[5] Pleiades: Ultra-Low-Power Reconfigurable Computing. http://bwrc.eecs.berkeley.edu/Research/ Configurable_Architectures/.

[6] Modern Embedded Systems: Compilers, Architectures, and Languages. http://www.gigascale.org/mescal/.

[7] RaPiD: Research on Coarse-Grained Adaptable Architectures. http://www.cs.washington.edu/research/ projects/lis/www/rapid/.

[8] M. Wan, Y. Ichikawa, D. Lidsky, and L. Rabaey, "An Energy Conscious Methodology for Early Design Space Exploration of Heterogeneous DSPs," Proc. ISSS Custom Integrated Circuits Conference (CICC), 1998.

[9] S. Hauck, T. Fry, M. Hosler, and J. Kao, "The Chimaera Reconfigurable Functional Unit," FPGAs for Custom Computing Machines (FCCM), 1997, pp. 87-96.

[10] M. Lee, H. Singh, G. Lu, N. Bagherzadeh, F. Kurdahi, E. Filho, and V. Alves, "Design and Implementation of the MorphoSys Reconfigurable Computing Processor," Journal of VLSI Signal Processing-Systems for Signal, Image and Video Technology, 2000.

[11] G. Smit, P. Havinga, L. Smit, P. Heysters, and M. Rosien, "Dynamic Reconfiguration in Mobile Systems," Proc. International Conference on Field Programmable Logic and Applications (FPL 2002), 2002, pp. 171–181.

[12] D. Zhu, D. Mossé, and R. Melhem, "R.G.: Power-Aware Scheduling for AND/OR Graphs in Real-Time Systems," IEEE Trans. Parallel Distrib. Syst., Vol 15, no. 9, pp. 849-864, 2004.

[13] M.L. Dertouzos and A.K. Mok, "Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks," IEEE Trans. Software Eng., vol. 15, no. 12, pp. 1497-1505, 1989.

[14] D. Zhu, R. Melhem, and B.R. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems," IEEE Trans. Parallel and Distributed Systems, vol. 14, no. 7, pp. 686-700, 2003.

[15] R. Ernst and W. Ye, "Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification", Proc. 1997 Intl. Conf. Computer-Aided Design (ICCAD'97), 1997, pp. 598-604.

[16] C. Scordino and E. Bini, "Optimal Speed Assignment for Probabilistic Execution Times", Proc. 2nd Workshop Power-Aware Real-Time Computing (PARC'05), 2005.

[17] P. Saha and T. El-Ghazawi, "Extending Embedded Computing Scheduling Algorithms for Reconfigurable Computing System," Proc. 3rd Southern Programmable Logic (SPL '07), 2007, pp. 87-92.

[18] N. R. Satish, K. Ravindran, and K. Keutzer, "Scheduling Task Dependence Graphs with Variable Task Execution Times onto Heterogeneous Multiprocessors," Technical report, University of California, Berkeley, UCB/EECS-2008-42, April, 2008.

[19] X. Zhou, Y. Wang, X. Huang, and C. Peng, "Fast On-Line Task Placement and Scheduling on Reconfigurable Devices," International Conference on Field Programmable Logic and Applications (FPL 2007), 2007, pp. 132-138.

[20] M. Yuan, X. He, and Z. Gu, "Hardware/Software Partitioning and Static Task Scheduling on Runtime Reconfigurable FPGAs using a SMT Solver," Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008), 2008, pp. 295-304.

[21] Z. Gao, Z. Wu, and M. Lin, "Energy-Efficient Fixed-Priority Scheduling for Periodic Real-Time Tasks with Multi-priority Subtasks," The 2007 International Conference on Embedded Software and Systems, pp. 572-583, 2007.