Hardware-Software Cosynthesis of Multi-Mode Multi-Task Embedded Systems with Real-Time Constraints

Hyunok Oh Soonhoi Ha
The School of Electrical Engineering and Computer Science
Seoul National University
Seoul 151-742, KOREA
TEL : +82-28807292
{oho,sha}@comp.snu.ac.kr

ABSTRACT
An embedded system is called multi-mode when it supports multiple applications by dynamically reconfiguring the system functionality. This paper proposes a hardware-software cosynthesis technique for multi-mode multi-task embedded systems with real-time constraints. The cosynthesis problem involves three subproblems: selection of appropriate processing elements, mapping and scheduling of function modules to the selected processing elements, and schedule analysis. The proposed cosynthesis framework defines an iteration loop of three steps that solve the subproblems separately. One of the key benefits of such a modular approach is extensibility and adaptability. Moreover, unlike the previous approaches, the proposed technique considers task sharing between modes and hardware sharing between tasks at the same time. We demonstrate the usefulness of the proposed technique with a realistic multi-mode embedded system that supports three modes of operation with 5 different tasks.

Keywords
Hardware-software cosynthesis, multi-mode, multi-task

1. Introduction
An embedded system is called multi-mode when it supports multiple applications by dynamically reconfiguring the system functionality. Such reconfigurability is desirable to cope with rapidly evolving standards and signal processing algorithms as well as to enhance the hardware utilization significantly. A multi-mode mobile terminal, for example, can be used for a PCS phone, MP3 player, and VOD terminal, by manually selecting the mode. We assume that an application defines a mode of the system and the system runs a single application at a time.

A single mode, in general, is a real-time multi-task system meaning that each application consists of a set of real-time tasks that should be scheduled within time constraints. Therefore, a critical design constraint of a multi-mode multi-task embedded system is to make all applications schedulable. Violation of this schedulability constraint can be detected using the schedulability tests applicable for real-time scheduling techniques[1][2].

The main focus of this paper is to find a minimum-cost system architecture that satisfies the schedulability constraints, given a real-time scheduling technique. We assume that a task is specified as an acyclic graph of which a node represents a function module such as DCT(Discrete Cosine Transform), MC (Motion Compensation), and so on. And there is a library of candidate processing elements, processors and IP blocks, with given timing information for each function module: how long it takes for each processing element(PE) to execute the function. A hardware implementation of a module may also be regarded as a processing element which takes infinite amount of time for other function modules. Then, the problem becomes selecting the appropriate processing elements and mapping function modules to the selected processing elements. We define this problem as the HW/SW cosynthesis problem.

While there have been some research efforts for cosynthesis of multi-task systems[3][4], only a few research results exist for multi-mode multi-task systems[5]. A naive approach of applying the cosynthesis techniques of multi-task systems directly to each mode separately is not optimal if a task is common in multiple modes. Therefore, the approach proposed in [5] considers the task sharing effect. Given task sets and processing elements, they examine the schedulability of each mode assuming that all tasks are run in a processor. If the schedulability constraint is violated, they single out the best task and the amount of execution time to be reduced to make all modes schedulable. They reduce the task execution time by implementing some code fragments to hardware component. However, they do not consider the resource sharing possibility between tasks so that they determine the best function module for hardware implementation separately for each selected task.

Compared with this previous approach, the proposed technique differs in two aspects. We address two issues at the same time: which processing elements to choose and which functional module to implement in hardware. Second, we consider the possibility of hardware resource sharing between tasks.

This research is supported by National Research Laboratory Program (number M1-0104-00-0015).
This work is sponsored by Brain Korea 21 project.
The RIACT at Seoul National University provides research facilities for this study.

The rest of the paper is organized as follows. In the next section, we state the problem and assumptions more clearly and present an example multi-mode multi-task system, experimented in this paper. Section 3 presents the structure of the proposed cosynthesis framework and section 4 explains the proposed algorithm in detail. And some experimental results are shown in section 5. We draw conclusion in section 6.

2. Preliminaries

In this section, we define some notations and terminologies and state the cosynthesis problem clearly with an example multi-mode system

A multi-mode system Π consists of a fixed number of modes $\{\Pi_i\}$ or $\Pi=\{\Pi_1, \Pi_2, \Pi_3,\ldots\}$. Each mode Π_i includes a number of tasks $\{\tau_j\}$ and each task τ_j is composed of modules $\{m_k\}$ that are functional blocks. The period T_{ij} and deadline D_{ij} of each task τ_j are defined separately for each mode Π_i. In case of a sporadic task, T_{ij} may be set as the minimum inter-arrival time between successive requests.

We are also given a library of candidate processing elements (PE's) $\{p_m\}$ that includes microprocessors, IPs, and ASIC core implementations of function modules. For each processing element p_m, its cost c_m and the worst case execution time $t_{m,k}$ of module m_k on p_m are assumed to be given for each module.

Then, the cosynthesis problem is to select a set of processing elements, $PE=\{p_n\}$, and to find a mapping $\phi:\{m_k\}\rightarrow\{p_n\}$ and scheduling of tasks $\{\tau_j\}$ to minimize the total cost of processing elements while satisfying the schedulability condition. Depending on which real-time scheduling technique is used, we use the appropriate schedulability test. We let $sl(\tau_j,PE)$ be the schedule length of task τ_j on the selected processing elements. Then, the utilization of mode Π_i, $U_{\Pi_i}(PE)$, becomes

$$U_{\Pi_i}(P)=\sum_{\tau_j\in\Pi_i}\frac{sl(\tau_j,P)}{T_{ij}} \qquad (1)$$

For instance, if a rate-monotonic scheduling technique is adopted, the schedulability test compares this utilization value with $n(2^{\frac{1}{n}}-1)$ where n is the number of tasks[1].

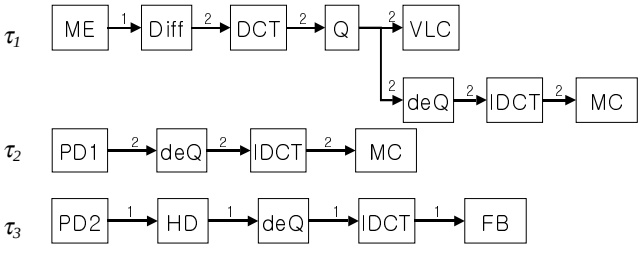
Figure 1 shows a real-life example of multi-mode embedded system experimented in this paper. The system supports 3 different modes of operation: video phone (Π_1), video player (Π_2), and MP3 player (Π_3). On the other hand, there are 5 different tasks: H.263 encoder (τ_1), H.263 decoder (τ_2), MP3 decoder (τ_3), G.723 encoder (τ_4), and G.723 decoder (τ_5). Figure 1(a) shows which tasks compose which mode of operation. For instance, the video phone mode runs 4 tasks $\{\tau_1,\tau_2,\tau_4,\tau_5\}$ concurrently.

The task period T_{ij} is dependent on the mode. Task τ_2 in Π_2 mode is scheduled twice as frequent as in Π_1 since the task decodes 20 frames per second in the video player mode Π_2 while it decodes 10 frames in Π_1. In this example, the task deadlines are set equal to the task periods. In case of audio encoder/decoder tasks, we assume that each invocation processes a buffered packet of 25ms voice samples to reduce the context switch overhead.

Each task is specified by an acyclic graph as shown in Figure 1(b). Note that three function modules are shared between tasks τ_1 and τ_2. In the graph, the annotated number on each arc indicates communication overhead to be counted if the source module and the sink module of the arc are mapped onto different processing elements. We do not show the graphs for tasks τ_4 and τ_5 assuming that they will not be broken down into multiple processing elements.

Mode	Π_1				Π_2		Π_3
Task	τ_1	τ_2	τ_4	τ_5	τ_2	τ_3	τ_3
Period	100	100	25	25	50	40	40
Deadline	100	100	25	25	50	40	40

(a)



(b)

	P_0(HW) : time(cost)	P_1(100)	P_2(900)
ME	ME_{hw}:17(100)	518	259
Diff	-	5.2	2.6
DCT	DCT_{hw}:5.6(20)	17	8.5
Q	Q_{hw}:4.6(24)	11.4	5.7
VLC	VLC_{hw}:6(20)	16	8
deQ	deQ_{hw}:2(10)	4	12
IDCT	$IDCT_{hw}$:5.8(20)	18	9
MC	MC_{hw1}:2.2(10), MC_{hw2}:1(30)	7.2	3.6
PD1	-	4.4	2.2
PD2	-	0.7	0.4
HD	-	3	1.5
deMQ	$deMQ_{hw}$:0.4(10)	1.2	0.6
IMDCT	$IMDCT_{hw}$:2(30)	5.7	2.9
FB	FB_{hw}:5(5)	10.2	5.1
τ_4	-	1.6	0.8
τ_5	-	1.8	0.9

(c)

Figure 1. An example multi-mode embedded system: (a) Modes of the system, and task periods and deadlines that depend on modes (b) Tasks specified by acyclic graphs (c) Module-PE profile table

Figure 1(c) shows the candidate processing elements and their cost and timing information. This table is called a *module-PE profile table*. The third and the fourth column indicate that there are two candidate microprocessors. We obtain the timing information for processing element P_1 from running the real code with the Armulator[6] assuming 500MHz ARM processor.

Processing element P_2 is about twice faster, but nine times more expensive. The first column lists the hardware implementations that will be regarded as separate processing elements. For each hardware implementation, the worst-case execution time and the hardware cost are given. For instance, MC_{hw1} has the value of 2.2 (msec) for the worst-case execution time and 10 for the cost. We admit that the numbers are not from the exact measurements. Packet decoding blocks PD1 and PD2, and Huffman decoding block HD have no hardware implementation.

3. Proposed Cosynthesis Framework

The cosynthesis problem involves three subproblems: selection of appropriate processing elements, mapping and scheduling of function modules to the selected processing elements, and schedule analysis. The proposed cosynthesis framework defines an iteration loop of three steps that attack the subproblems separately as depicted in Figure 2. The inputs to the cosynthesis framework are a library of candidate processing elements and a module-PE profile table as well as input task graphs.

The iteration starts with the module-PE allocation controller. The module-PE allocation controller selects a set of processing elements $\{p_n\}$ from the input candidate processing elements $\{p_m\}$ and constructs a reduced module-PE profile table that includes the selected processing elements only. This step is most critical since design objectives are considered when selecting the appropriate processing elements. If the design objective is to minimize the cost, we first select the cheapest processor first. The detailed mechanism will be explained in the next section.

The role of the next step is to schedule the acyclic graph of each task to the selected processing elements in order to minimize the schedule length. While the task graphs are given as inputs, the reduced module-PE profile table is obtained from the PE Allocation Controller step. Since this is a typical problem of heterogeneous multiprocessor (HMP) scheduling, we use any heterogeneous scheduler in this step. We obtain the schedule result and the schedule length $sl(\tau_j,PE)$ for task τ_j. We apply this step for each task graph separately. An interesting observation in this step is that the scheduler may not consume all selected processing elements to further reduce the system cost if possible.

The next step is the performance evaluation step. It first checks whether the design constraints are satisfied. Based on the schedule lengths of all tasks obtained from the previous step, we compute the utilization factors for schedulability analysis. If the schedulability constraint is satisfied, it may end the iteration and record the scheduling results. In case tradeoffs between multiple objectives are searched, it records the schedule results and restarts the iteration until all desired number of optimal points are collected. If any design constraint is violated, it passes the scheduling results and violation information to the PE Allocation Controller to select other processing elements. More detailed discussion can be found elsewhere [7].

One of the key benefits of such modular approach is extensibility. Without modification of the core mapping and scheduling step, we can add more design constraints to the performance evaluation step. More processing elements can be added to the PE Allocation Controller seamlessly. Even multiple design objectives can be considered without modifying the core mapping and scheduling step.

Second benefit is adaptability. We can easily change the mapping and scheduling algorithm even though our implementation uses a specific HMP scheduler, called the BIL scheduler[8], based on a list scheduling heuristic. It is reported that this specific scheduler performs reasonably good while time complexity is order of magnitude faster than other well-formulated algorithms. Another adaptation can be found in choosing the right schedulability test for a given real-time scheduler. Only performance evaluation step is modified to use the modified schedulability test.

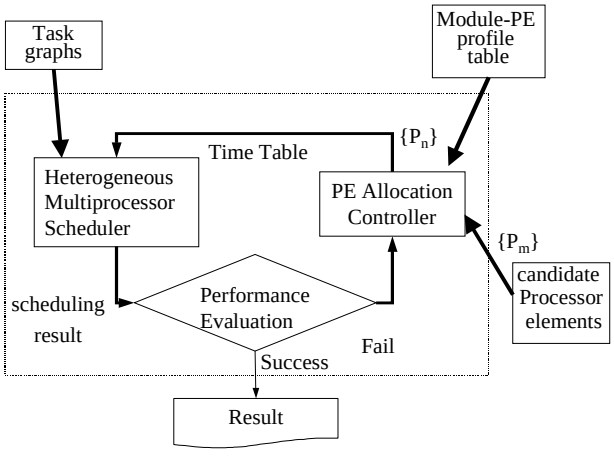


Figure 2. The proposed cosynthesis framework

3.1 Time Complexity

The worst case iteration counts of the proposed algorithm is p where p is the total number of the candidate processing elements because it adds one processing element at a time. For each iteration, we call the HMP scheduler $p_r \times N_t$ times where p_r is the number of remaining candidate processing elements and N_t is the number of tasks. To select the best processing element, we call the HMP scheduler once for each candidate processing element even though we can prune the search tree drastically in real implementation. If we let the time complexity of the HMP scheduler as S, the total time complexity becomes $O(Sp^2N_t)$. The time complexity of the HMP scheduler depends on the size of the task graphs and the number of selected processing elements.

Even for a single mode system, the proposed technique has more advantageous in terms of time complexity than other previous approaches such as genetic algorithms and integer linear programming. It is well-known that the ILP approach is prohibitively complex to solve even reasonable size problems. MOGAC[4] uses a genetic algorithm to solve cosynthesis problem for multi-task systems. It has much larger time complexity than ours due to two main reasons. First, the problem size is proportional to the number of tasks while the time complexity is proportional to the number of tasks in our proposed approach. If the problem size grows, the time complexity of a genetic algorithm grows much faster in general. Second, to satisfy the schedulability constraint, they consider a hyper-period that is the least common multiple of the tasks periods. If the task periods are different each other, the hyper-period can be huge and the problem size can be huge proportionally. On the other hand, the proposed algorithm uses the schedulability test without problem size increment, assuming that a real-time operating system is used.

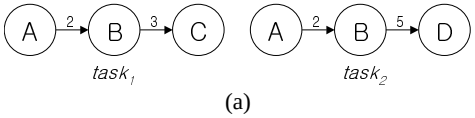
4. Processing Element Selection

In this section, we explain how the PE allocation controller selects the processing elements to achieve the design objectives. For simplicity, we assume that the design objective is to minimize the system cost. And we use a simple example to show how the algorithm proceeds.

Consider an example of Figure 3 that has two modes of operation and two different tasks. Mode Π_1 needs two tasks while mode Π_2 needs the second task only. Two tasks consist of three function modules respectively, while two function modules are shared between two tasks. We assume that the period of $task_1$ and $task_2$ in mode Π_1 is 40 and 60 respectively, and $task_2$ in mode Π_2 30. There are two candidate processors and 6 different hardware implementations for the constituent function modules as shown in the module-PE profile table of Figure 3(b).

Since the design objective is to minimize the system cost, initially the PE Allocation Controller allocates the cheapest processing element: P_1 in this example. The reduced module-PE profile table can be depicted as Figure 3(c) where infinite time indicates the corresponding processing element is not selected. Consequently the HMP scheduler maps all modules onto the PE's of minimum cost P_1 as displayed in Figure 3(d). We obtain two separate scheduling results for two task graphs.

	Π_1		Π_2
	$task_1$	$task_2$	$task_2$
Period	40	60	30
Deadline	40	60	30



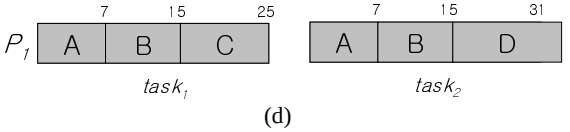
(a)

	P_0(HW):time(cost)	P_1(10)	P_2(60)
A	A_{hw}:1(12)	7	2
B	B_{hw1}:2(5),B_{hw2}:1(15)	8	3
C	C_{hw1}:2(10),C_{hw2}:1(20)	10	5
D	D_{hw}:4(10)	16	5

(b)

	P_0	P_1	P_2
A	∞	7	∞
B	∞	8	∞
C	∞	10	∞
D	∞	16	∞

(c)



(d)

Figure 3. (a) Modes and task graphs (b) Module-PE profile table (c) Initially reduced module-PE profile table (d) Scheduling results

The next step is the performance evaluation step that tests if tasks are schedulable. As discussed earlier, it is *utilization* that is a measure to determine the schedulability for a given real-time scheduler such as rate-monotonic, earliest deadline first and so on. If the utilization becomes larger than the given utilization constraint then the evaluation fails and more PE's need to be allocated. From the scheduling result in Figure 3(d), utilization U_{Π_1} of mode Π_1 becomes 1.14 ($=\frac{25}{40}+\frac{31}{60}$) and U_{Π_2} becomes 1.03 ($=\frac{31}{30}$). If we assume that the utilization constraint is 1.0, we should allocate more PE's in order to reduce the scheduling length of all tasks until utilization constraint is satisfied for all tasks.

Now, we arrive at the core of the selection technique. Among many candidate processing elements, we want to select another PE, which reduces the task execution times as much as possible, but minimizes the cost increment. We define the expected utilization decrement(*EUD*) and the expected cost increment(*ECI*) for each candidate processing element. Furthermore, we define the *slack* as the difference between the utilization constraint U^* and the current utilization in order to avoid reducing the utilization factor too much with more expensive PE.

$$Slack_{\Pi_i} = U_{\Pi_i}(PE) - U^* \quad (2).$$

While $ECI(p_n)$ is simply the cost of processing element p_n, $EUD(p_n)$ is defined as the difference between the utilization before allocating p_n and the utilization after allocating p_n.

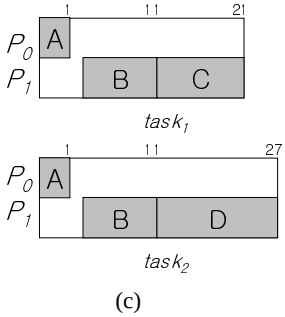
$$EUD(p_n) = \sum_{\Pi_i \in \Pi} \min(U_{\Pi_i}(PE) - U_{\Pi_i}(PE \cup \{p_n\}), Slack_{\Pi_i}) \quad (3).$$

	schedule length		EUD	ECI	$\frac{EUD}{ECI}$
	$task_1$	$task_2$			
A_{hw}	21	27	0.17	12	**0.014**
B_{hw}	24	31	0.03	5	0.005
B_{hw}	23	31	0.05	15	0.003
C_{hw}	20	31	0.125	10	0.013
C_{hw}	19	31	0.15	20	0.008
D_{hw}	25	24	0.15	15	0.010
P_2	10	10	0.17	60	0.003

(a)

	P_0	P_1	P_2
A	1	7	∞
B	∞	8	∞
C	∞	10	∞
D	∞	16	∞

(b)



(c)

Figure 4. (a) $\frac{EUD}{ECI}$ value for all candidate processing elements (b) Modified module-PE profile table after A_{hw} is selected (c) Scheduling results

After computing *EUDs* and *ECIs* of all PE's, we choose an entry that has the largest $\frac{EUD}{ECI}$ value among unselected PE's since the utilization is expected to decrease significantly with minimal cost increase. And we modify the reduced module-PE profile table and pass it to the HMP scheduler. It is not guaranteed however that

the modules are mapped to the newly selected PE. Modules will be scheduled onto the PE only when the total schedule length is actually reduced considering communication overheads.

Figure 4(a) represents *EUD* and *ECI* values of candidate processing elements at the onset of the second iteration. For example, $EUD(A_{hw})$ is the sum of $\min(1.14-(\frac{21}{40}+\frac{27}{60}),0.14)$ of mode Π_1 and $\min(1.03-\frac{27}{30},0.03)$ of mode Π_2. In this example, A_{hw} is chosen since its ratio is the largest. The HMP scheduling result is shown in Figure 4(c). Since we can schedule all tasks within the utilization constraint, we exit from the iteration loop.

The multi-function problem[9] is a cosynthesis problem to support multiple functions or applications of which only one is executed at any instant. Since the problem allows each mode or application to have one task, it is a sub-problem of the cosynthesis problem discussed in this paper. The fact that each mode has one task enables us not to compute utilization. Instead of utilization, the schedule length of each task can be used to compute expected utilization increment. The other procedures remains as described in the previous section.

5. Experimental Results

We apply the proposed technique to the multi-mode embedded system described in section 2. The HW speed and HW cost information is reasonably estimated while not obtained from real implementation.

For comparison, we first apply the proposed cosynthesis algorithm for each mode of operation separately and add up the estimated system cost at the end. Table 1 shows which processing elements are selected and what is the resultant system cost. While processor P_1 is commonly selected, different hardware implementations are selected for video phone and video player applications. As a result, 5 processing elements are selected and the system cost becomes 235.

Table 1. Results without considering multi-mode multi-task

Mode	Used PE's	Cost
video phone	ME_{hw}, $IDCT_{hw}$, P_1	220
video player	MC_{hw1}, FB_{hw}, P_1	115
MP3 player	P_1	100
Total	ME_{hw}, $IDCT_{hw}$, P_1, MC_{hw1}, FB_{hw}	235

Now, we apply the proposed algorithm to all modes together considering the resource sharing possibility. As shown in Table 2, the video player mode selects a different set of hardware implementations. Instead of selecting Motion Compensation and Filter Bank blocks, it selects IDCT block for HW implementation since the IDCT HW is already selected in the video-phone mode. Since resource sharing is successfully exploited in the proposed technique, the total system cost is reduced to 220.

The proposed algorithm has been implemented in C++ on a codesign framework[10]. It takes 0.1 seconds with Pentium 667 MHz processors. Considering the problem size of 3 modes, 5 tasks, 16 function modules, and 13 processing elements, the time complexity is reasonable.

Table 2. Results with considering multi-mode multi-task

Mode	Used PE's	Cost
video phone	ME_{hw}, $IDCT_{hw}$, P_1	220
video player	$IDCT_{hw}$, P_1	120
MP3 player	P_1	100
Total	ME_{hw}, $IDCT_{hw}$, P_1	220

We apply the proposed technique to the examples used in Hou's research[3]. They have three processing elements and four tasks which includes 10 modules. They tested three examples of task combination, which we interpret them as three different modes of operation: Π_1, Π_2 and Π_3 as shown Table 3. If we use the same task periods as [3], we cannot reduce system cost further since independent application of the cosynthesis algorithm to each mode also selects two processing elements. However if we prolong the period of τ_1 and τ_3 in Π_2 to 2000, then the system cost increases since the algorithm allocates lower cost PE for tasks in Π_2 instead of reusing PE's allocated for tasks in the other modes.

Table 3. Hou's task graphs : period and system cost

	Π_1		Π_2		Π_3	
	$task_1$	$task_2$	$task_1$	$task_3$	$task_3$	$task_4$
original period	240	240	240	200	200	380
relaxed period	240	240	2000	2000	200	380

	cost without considering multi-mode	cost with considering multi-mode
original period	170	170
relaxed period	190	170

6. Conclusions

In this paper, a HW/SW cosynthesis framework is proposed for multi-mode multi-task embedded systems with real-time constraints. The proposed iterative consynthesis procedure consists of three steps: selection of processing elements including ASIC core implementations, mapping and scheduling of task graphs onto the selected processing elements, and schedulability test.

Unlike the previous approaches, we take into account of task sharing between operation modes as well as HW resource sharing between tasks. As a result, the proposed algorithm achieves about 10% reduction of system cost with an example multi-mode embedded system, compared with an approach without considering the resource sharing opportunities. Since the time complexity of the proposed algorithm is only linear to the number of tasks, it is applicable for large size applications.

The key benefits of the proposed framework are extensibility and adaptability. Even though we concern about the schedulability and the system cost only in this paper, more design constraints and design objectives can be easily augmented. The main difficulty of using this approach to practical system design is constructing the module-PE profile table, which is assumed to be given in this paper.

7. REFERENCES

[1] C. L. Liu and J. W. Layland, "Scheduling algorithm for multiprogramming in a hard real time environment," Journal of ACM, vol. 20, pp. 46-61, Jan. 1973.

[2] N.Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," In Proc. of IEEE Workshop on Real-Time Operating Systems and Software, pp. 133-137, May 1991.

[3] J. Hou and W. Wolf, "Process partitioning for distributed embedded systems," in Proc. Int. Workshop Hardware-Software Codesign, pp. 70-76, March 1996.

[4] R. P. Dick and N. K. Jha, "MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Cosynthesis of Distributed Embedded Systems," IEEE Trans. on Computer-Aided Design of integrated circuits and systems, vol. 17, no. 10, pp. 920-935, Oct. 1998.

[5] Y. Shin, D. Kim, and K. Choi , "Schedulability-driven performance analysis of multiple mode embedded real-time systems," Proc. Design Automation Conf., pp. 495-500, June 2000.

[6] ARM Ltd., "Software Development Toolkit", available at http://www.arm.com/product/SDT/.

[7] Hyunok Oh and Soonhoi Ha, "A Hardware-Software Cosynthesis Technique Based on Heterogeneous Multiprocessor Scheduling", 7th International Workshop on Hardware/Software Codesign, pp. 183-187, May 1999.

[8] Hyunok Oh and Soonhoi Ha, "A Static Scheduling Heuristic for Heterogeneous Processors", Second International EuroPar Conference Proceedings, vol. II, August 1996.

[9] A. Kalavade and P. A. Subrahmanyam, "Hardware / Software Partitioning for Multi-function Systems", Proc. International Conference on Computer Aided Design, pp. 516-521, Nov. 1997.

[10] http://peace.snu.ac.kr/research/peace : PeaCE codesign Environment