Instruction-Set Customization for Real-Time Embedded Systems

Huynh Phung Huynh and Tulika Mitra
School of Computing
National University of Singapore
{huynhph1,tulika}@comp.nus.edu.sg

Abstract

Application-specific customization of the instruction set helps embedded processors achieve significant performance and power efficiency. In this paper, we explore customization in the context of multi-tasking real-time embedded systems. We propose efficient algorithms to select the optimal set of custom instructions for a task set under two popular real-time scheduling policies. Our algorithms minimize the processor utilization through customization while satisfying the task deadlines and the constraint on silicon area. Experimental evaluation with various task sets shows that appropriate customization can achieve significant reduction in the processor utilization and the energy consumption.

1 Introduction

Current generation embedded system designs are characterized by the increasing demand on higher performance under stringent time-to-market constraints. In this context, application-specific customizable processor cores strike the right balance between performance and design efforts. A customizable processor is, in general, configurable w.r.t. the micro-architectural parameters. More importantly, a customizable processor may support application-specific extensions of the core instruction-set. Custom instructions encapsulate the frequently occurring computation patterns in an application. They are implemented as custom functional units (CFU) in the datapath of the existing processor core. CFUs improve performance through parallelization and chaining of operations. Some examples of commercial customizable processors include Lx, ARC^{TM} core, Xtensa and Stretch S5.

In this work, we explore customization in the context of multi-tasking real-time embedded systems. Most embedded system designs today support concurrency through multi-tasking. Moreover, they often employ real-time scheduling policies to meet strict timing constraints. Customizable processor cores appear to be quite promising in this scenario. First, custom instructions may reduce the processor utilization for a task set through performance speedup of the individual tasks. This improvement may enable a task set that was originally unschedulable to satisfy all the timing requirements. This is a far better option than choosing a high-frequency and energy-inefficient processor core to meet the deadlines. Second, a lower processor utilization opens up the possibility to execute non-real-time (best-effort) tasks alongside real-time tasks. Finally, a lower utilization can exploit voltage scaling opportunities to lower the operating frequency/voltage of the processor and still meet the computational requirements of the tasks. As energy consumption scales quadratically with the operating voltage ($E \propto V^2$), a small change in voltage can have a significant impact on the energy consumption.

One of the major challenges in the effective deployment of customizable processors is the development of the design-automation tool chain. In particular, a major research focus in the recent past has been automated identification of suitable instruction-set extensions for an application [7]. Given a single sequential application (a task), the goal here is to select a set of custom instructions that optimizes certain design criteria (such as power or performance) under pre-defined design constraints (such as silicon area).

Multi-tasking real-time embedded systems add substantial complexity to this design space exploration process. The optimization problem in this context is to minimize the processor utilization (through custom instructions) while satisfying the task deadlines under an area constraint. Clearly, a naïve approach of optimizing the execution time of each task in isolation will miss certain opportunities. We have to take into account the complex interplay among the tasks enabled by the real-time scheduling policy.

We propose efficient algorithms to select the *optimal* set of custom instructions for a multi-tasking real-time workload. We consider two popular real-time scheduling policies: a static priority based Rate-Monotonic Scheduler (RMS) and a dynamic priority based Earliest Deadline First (EDF) scheduler. For EDF scheduling policy, we employ a dynamic programming solution whereas for RMS schedul-

ing, we resort to an efficient branch-and-bound based search algorithm. Our experimental evaluation with a large number of workloads confirms the benefit of processor customization in real-time systems.

2 Related Work

The design space exploration problem to select suitable custom instructions for an application consists of two steps[7]. The first step identifies a large set of candidate patterns from the program's dataflow graph and their frequency via profiling [2, 4, 11]. Given this library of patterns, the second step selects a subset to maximize the performance under different design constraints. Various approaches proposed for this step include dynamic programming [1], 0-1 Knapsack [5], greedy heuristic [4], and ILP [8].

Customization for task graphs has been addressed in [10]. They propose an iterative algorithm to perform the assignment and scheduling of a task graph on to a heterogeneous multiprocessor platform together with processor customization in an integrated manner. However, they do not consider real-time constraints. A heuristic to select custom instructions such that the worst-case execution time (which is critical for real-time systems rather than the average-case) of a task is minimized has been proposed earlier by our group [12]. This intra-task customization approach is complimentary to inter-task customization for multi-tasking real-time systems.

3 Customization for Real-Time Systems

3.1 Problem Formulation

In the classic model of a real-time system, a set of tasks are executed periodically. Each task T_i is associated with a period P_i and a worst-case execution time C_i. An instance of the task T_i is released periodically once every P_i time units. The task instance should complete execution by its deadline, which is typically defined as the end of the period. The goal of real-time scheduling is to meet the deadline of every task. Schedulability analysis determines whether a specific set of tasks can be successfully scheduled using a specific scheduler. Given a set of N independent, preemptable, and periodic tasks on a uniprocessor, let U be the total utilization of this task set. U quantifies the fraction of processor cycles used by a task set. Therefore, a *necessary* condition for feasible scheduling of a task set is

$$U = \sum_{i=1}^{N} U_i = \sum_{i=1}^{N} \frac{C_i}{P_i} \leq 1 \qquad (1)$$

We would like to explore the opportunities opened up by instruction-set customization in this context. Each task

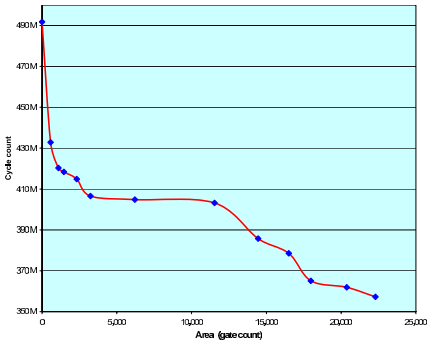


Figure 1. *Application performance versus additional gate count for different processor configurations corresponding to g721 encoding task.*

T_i has a set of custom instructions enhanced configurations with different performance/silicon area tradeoff. The higher is the area cost of a custom instruction configuration, the better is its performance. Let $config_{i,j}$ (for $j = 1 \ldots n_i$) be the j^{th} configuration corresponding to task T_i and n_i is the number of configurations for task T_i. In addition, let $cycle_{i,j}$ and $area_{i,j}$ denote the application performance in processor cycles and gate count of $config_{i,j}$. We assume that $config_{i,1}$ corresponds to the configuration without any custom instruction, i.e., $area_{i,1} = 0$ and $cycle_{i,1} = C_i$ (the task performance without any enhancement). For example, Figure 1 shows the set of configurations corresponding to g721 encoding task.

Given (1) a set of independent, preemptable, and periodic tasks, (2) a specific scheduling policy (RMS or EDF), and (3) a total area budget $AREA$ for the custom instructions, our goal is to select an appropriate configuration for each task such that the task set is schedulable and the total utilization U is minimized.

3.2 Customization for EDF Scheduling

Earliest Deadline First (EDF) is an optimal dynamic priority scheduling policy. It executes at any instant, the ready task with the closest deadline. If more than one ready tasks have the same deadline, EDF randomly selects one for execution. A task set is schedulable under EDF policy if the total utilization (U) is less than or equal to 1 (Equation 1).

We develop an algorithm to select the appropriate configuration for each task such that the total utilization of the task set is minimized. As the value of total utilization determines the feasibility of an EDF schedule, the algorithm, by definition, works towards meeting task deadlines. If the minimum utilization returned is greater than 1, then the task set cannot be scheduled even with custom instruction enhancements.

We propose a pseudo-polynomial time dynamic programming algorithm that returns the *optimal* solution. Let $U_i(A)$ be the *minimum* total utilization of tasks $T_1 \ldots T_i$ under an area budget A. Then $U_i(A)$ can be defined recursively.

$$U_i(A) = \min_{\substack{j=1\ldots n_i \\ area_{i,j} \leq A}} \left(\frac{cycle_{i,j}}{P_i} + U_{i-1}(A - area_{i,j}) \right) \quad (2)$$

That is, given an area A, we explore all possible configurations for T_i and choose the one that results in minimum utilization for tasks $T_1 \ldots T_i$. The base case for task T_1 is

$$U_1(A) = \min_{\substack{j=1\ldots n_1 \\ area_{1,j} \leq A}} \left(\frac{cycle_{1,j}}{P_1} \right) \quad (3)$$

The minimum utilization for tasks $T_1 \ldots T_N$ under area budget $AREA$ then corresponds to $U_N(AREA)$.

Algorithm 1: Custom Instructions selection under EDF

Input: Task Set $T_1, \ldots, T_N$ with configurations; Area constraint: AREA
Result: Minimum utilization
for A = 0 to AREA in steps of Δ **do**
  $U_1(A) \leftarrow \min_{\substack{j=1\ldots n_1 \\ area_{1,j} \leq A}} \left(\frac{cycle_{1,j}}{P_1} \right)$
for A = 0 to AREA in steps of Δ **do**
  for i=2 to N **do**
    $U_i(A) \leftarrow \min_{\substack{j=1\ldots n_i \\ area_{i,j} \leq A}} \left(\frac{cycle_{i,j}}{P_i} + U_{i-1}(\lfloor \frac{A - area_{i,j}}{\Delta} \rfloor \times \Delta) \right)$
return U_N(AREA);

Algorithm 1 encodes this recursion as a bottom-up dynamical programming algorithm. The step value Δ determines the granularity of area (typically area equivalent of 1K logic gates for most tasks). It is chosen based on the minimum area difference between two successive configurations for any task. The time complexity of this algorithm is $O(N \times \frac{Area}{\Delta} \times x)$ where $x = max_{i=1\ldots N}(n_i)$.

3.3 Customization for RMS

Rate Monotonic Scheduler (RMS) is an optimal static-(fixed-) priority scheduling policy using the task's period as the task's priority. RMS executes at any instant the ready task with the shortest period, i.e., the task with the shortest period has the highest priority. If more than one ready tasks have the same period, RMS randomly selects one for execution. Unlike EDF, however, there exist task sets with $U \leq 1$ that are not schedulable under RMS. There are no known polynomial time exact schedulability tests for RMS. In this work, we use a recently proposed exact schedulability test [3] that is more efficient than the previously proposed tests.

Theorem 1 *Given a periodic task set $T_1, \ldots, T_N$ in increasing order of periods*

1. *T_i can be scheduled using RMS if and only if:*

$$L_i = \min_{t \in S_{i-1}(P_i)} \frac{\sum_{j=1}^{i} \left\lceil \frac{t}{P_j} \right\rceil C_j}{t} \leq 1$$

where $S_i(t)$ is defined by the following recurrent expression:

$$\begin{cases} S_0(t) = \{t\} \\ S_i(t) = S_{i-1}\left(\left\lfloor \frac{t}{P_i} \right\rfloor P_i \right) \cup S_{i-1}(t) \end{cases}$$

2. *The entire task set is schedulable using RMS if and only if:*

$$\max_{i=1\ldots N} L_i \leq 1$$

Due to the double recurrent form of its definition, the worst-case cardinality of a generic $S_i(t)$ set is 2^i. In case two sets, $S_{i-1}\left(\left\lfloor \frac{t}{P_i} \right\rfloor P_i \right)$ and $S_{i-1}(t)$, overlap, the cardinality reduces.

The complexity of the schedulability test renders the design space exploration under the RMS policy more difficult compared to the EDF policy. Given a task set scheduled with RMS, it is possible to have two customized configurations p and p' such that $U(p) < U(p')$ but p' meets all the task deadlines whereas p does not. That is, we can no longer minimize only the total utilization without checking the feasibility of the schedule.

We propose a Branch and Bound search algorithm to select appropriate configuration for each task such that the entire task set is schedulable under Theorem 1 and the total utilization of the task set is minimized. Branch-and-bound deals with optimization problems over a search space that can be presented as the leaves of a search tree. The search is guaranteed to find the optimal solution, but its complexity in the worst case is as high as that of exhaustive search. The pseudo code is given as Algorithm 2.

Each level i in the branch-and-bound search tree corresponds to the choice of configuration for the task T_i. Thus, each node at level i corresponds to a partial solution with the configurations about the tasks T_1 up to T_i. Whenever we reach a leaf node of the search tree, we have a complete solution. The power of branch-and-bound algorithm comes from the effective pruning of the design space. We prune the design space under the following conditions.

First, during the traversal of the search tree, the minimum utilization achieved so far at any leaf node is kept as a bound $MinU$. At any non-leaf node m in the search tree, we compute a lower bound, $bound(m)$, on the minimum possible utilization at any leaf node in the subtree rooted at m. The lower bound is computed by summing up the utilization of the tasks that have been enhanced with custom instructions and the minimum utilization of the remaining tasks (which is the utilization when enhanced with the best possible custom instruction configuration). If $bound(m) \geq MinU$, then the search space corresponding to

the subtree rooted at m can be pruned. Moreover, at any level, the configuration with the minimum execution time is considered first. This ensures greater possibility of obtaining a low utilization value $MinU$ quickly and thereby achieve effective pruning during the subsequent traversal.

Second, we select the appropriate configuration for each task in the order of decreasing priority, i.e., the highest priority task is considered first. Recall that RMS is a static priority preemptive schedule. A higher priority task can preempt a lower priority task but not the other way round. Suppose we have a partial solution where the configurations corresponding to the first $i-1$ high priority tasks (T_1 to T_{i-1}) have been chosen. Suppose further that the tasks T_1 to T_{i-1} all meet their respective deadlines with the chosen configurations. Any lower priority task, such as T_i, cannot preempt the higher priority tasks and hence the higher priority tasks will not miss their deadlines due to the introduction of T_i. Thus, the task traversal order ensures that at level i of the search tree we only need to check the schedulability of task T_i (i.e., whether $L_i \leq 1$ in Theorem 1). Moreover, if T_i fails to meet its deadline, we can prune the subtree rooted at the corresponding node.

Finally, if the area constraint is violated at any node, then the subtree rooted at the corresponding node is pruned.

Algorithm 2: Custom Instructions selection under RMS

Input: Task Set T_1 **to** T_N with configurations; Area constraint: AREA
Output: Minimum utilization
begin
  U $\leftarrow$ 0; optimalSoln $\leftarrow \emptyset$; A $\leftarrow$ AREA; MinU $\leftarrow \sum_{i=1}^{N} \frac{C_i}{P_i}$;
  /* T_1 is highest priority task */;
  `search` (T_1, U, A, $\emptyset$);
  return *MinU*;
end
Function `search` (T_i, U, A, Soln)
for $config_{i,j}$ ($j \in 1$ **to** n_i) in increasing order of execution time **do**
  if ($area_{i,j} \leq A$) ***and*** T_i is schedulable with $cycle_{i,j}$ **then**
    partialSoln $\leftarrow$ Soln $\cup$ config$_{i,j}$; A $\leftarrow$ A $-$ area$_{i,j}$;
    U $\leftarrow$ U $+ \frac{\text{cycle}_{i,j}}{P_i}$;
    if $T_i = T_N$ **then**
      if $U < MinU$ **then**
        MinU $\leftarrow$ U; optimalSoln $\leftarrow$ partialSoln;
      return ;
    if `bound`($partialSoln$) $< MinU$ **then**
      `search` (T_{i+1}, U, A, partialSoln);

4 Experimental Evaluation

We use 8 benchmarks from MiBench and one benchmark from Mediabench (`g721_encoder`) for our experiments. We create six task sets using these benchmarks; each task set consists of four benchmarks as shown in Table 1.

We choose the Xtensa [6] processor platform from Tensilica for our experiments. Xtensa is a configurable processor core allowing application-specific instruction-set extensions. We use the XPRES compiler provided by Tensilica to generate the custom instructions from the C code corresponding to a task. Multiple custom instruction configurations are generated for each task based on the trade off between area and performance (see Figure 1). The maximum performance gain for the individual tasks vary from 3.5% to 27% with area budget ranging from 1K to 23K logic gates.

Task set	Benchmarks
1	crc32, sha, jpeg_decoder, blowfish
2	blowfish, adpcm_decoder, crc32, jpeg_encoder
3	adpcm_encoder, blowfish, jpeg_decoder, crc32
4	sha, susan, crc32, g721_encoder
5	adpcm_decoder, jpeg_decoder, crc32, blowfish
6	crc32, sha, blowfish, susan

Table 1. *Composition of Task sets*

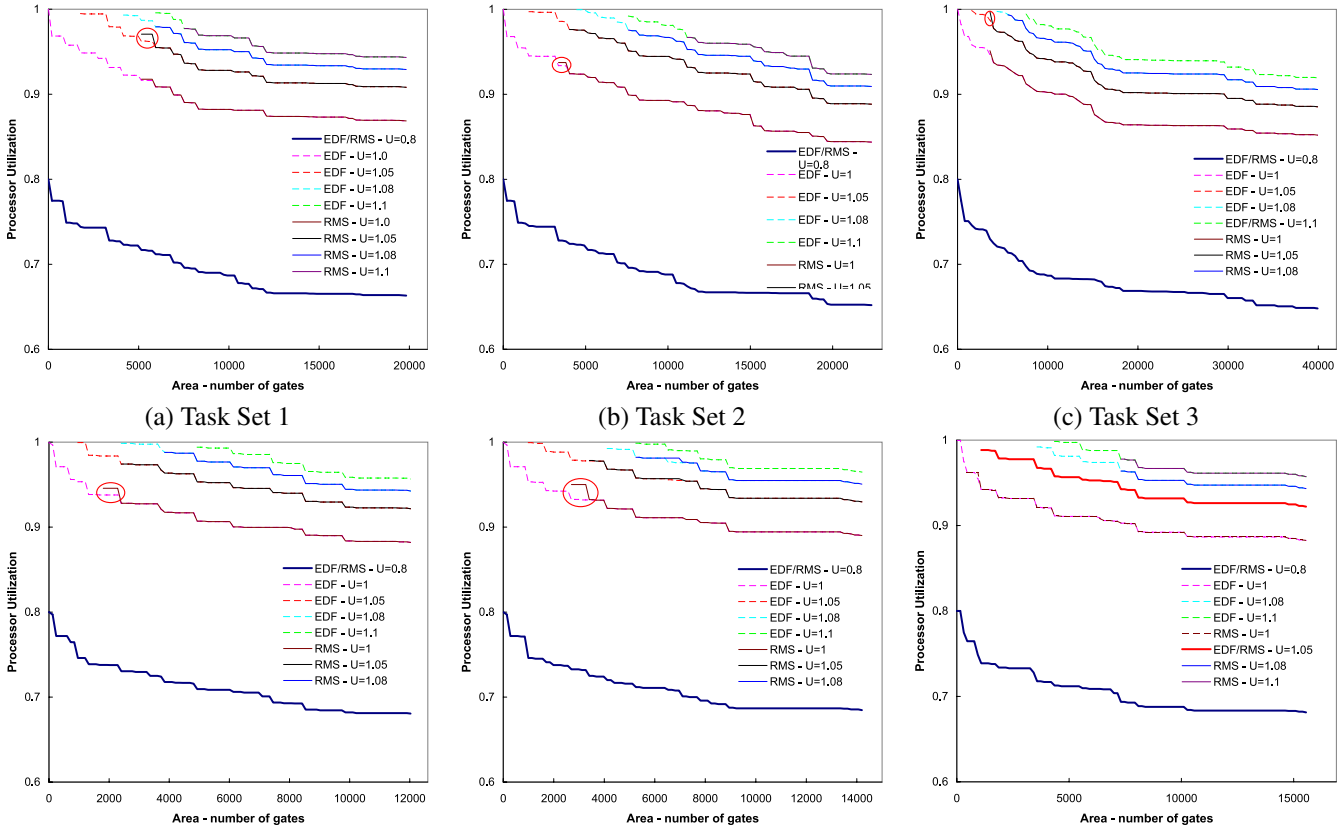
To set the periods for the tasks, we choose a total utilization for the task set (without any custom instructions) and then select the periods to achieve the corresponding utilization. Let C_i be the execution time of task T_i without using custom instructions. Then we set the period P_i for each task T_i as $P_i = \alpha_i \times C_i$ such that $\sum_{i=1}^{N} \frac{C_i}{P_i} = U$. We choose five different utilization factors U = 0.80, 1.00, 1.05, 1.08 and 1.10. A task set is EDF-schedulable if U = 0.8 or 1.0; but may or may not be RMS-schedulable. In this case, we are interested in finding out how much we can reduce the utilization by using custom instructions. For $U > 1.0$, a task set can only become schedulable by using custom instructions. The greater the original utilization factor, the more difficult it is to schedule the tasks using custom instructions.

For each task set, we vary the hardware area constraint from 0 to Max_Area at an interval of $0.01 \times Max\_Area$. The Max_Area for each task set is simply the summation of the maximum area requirements of the constituent tasks. A task set enhanced with custom instructions at Max_Area explores the limit of speedup achievable. The stricter the area constraint, the more difficult it is to schedule a task set and/or achieve lower utilization.

The average runtime of our configuration selection algorithm is 0.4 sec and 0.068 sec under EDF or RMS scheduling policy, respectively. The experiments are performed on a Pentium 4 3GHz CPU with 1GB of RAM.

4.1 Performance

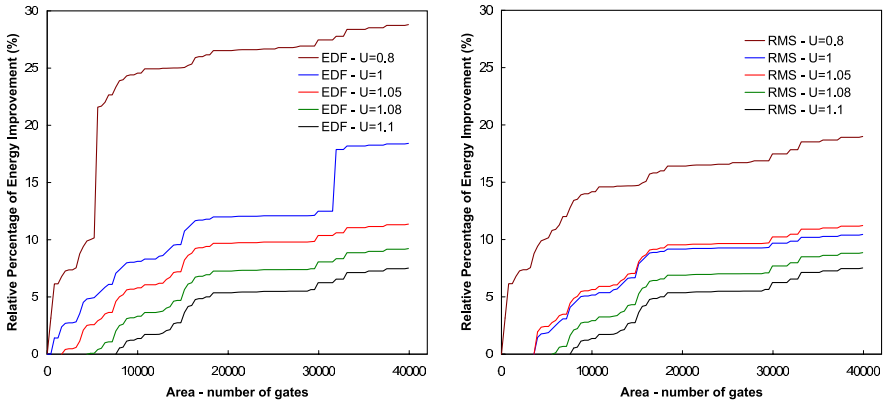
Figure 2 shows the utilization versus hardware area trade-off for the different task sets. For each task set and an original utilization factor, we apply both RMS and EDF scheduling policies. Our algorithms take less than 0.1 sec to return the solution for any task set and scheduling policy. The Y-axis shows the reduced utilizations. The utiliza-



(a) Task Set 1 (b) Task Set 2 (c) Task Set 3

(d) Task Set 4 (e) Task Set 5 (f) Task Set 6

Figure 2. Utilization versus Area for different task sets under EDF and RMS scheduling policies.



(a) EDF Policy (b) RMS Policy

Figure 3. Area versus Energy for Task Set 3 under EDF and RMS scheduling policies.

tion of a task set decreases with increasing hardware area because we can accommodate more custom instructions. Overall, we get up to 19% reduction in utilization. On an average, we get about 14% (13%) reduction in utilization at roughly 75% (50%) of *Max_Area* .

The reduced utilization values for a given area constraint are mostly identical for RMS and EDF. At original utilization $U = 0.8$, all our task sets are RMS-schedulable (they are, by definition, EDF schedulable). As adding custom instructions strictly improves the execution time of each task, the task sets remain scheduable for all possible configurations. We choose configurations for each task such that the total utilization is minimized and the task set is schedulable. Therefore, RMS and EDF scheduling policies select identical custom instruction configurations at a given area.

However, at original utilization $U = 1.0$, a task set without custom instructions enhancements may not be RMS-schedulable. Indeed, all our task sets are not schedulable under RMS policy at $U = 1.0$. Therefore, given a strict area budget, we fail to schedule the task sets under RMS policy even using custom instructions. As the area budget increases, a task set becomes RMS-schedulable and produces identical reduced utilization for both policies. In general, at any original utilization value greater than 1.0, a task set under the EDF policy becomes schedulable earlier compared to the RMS policy. The highlighted portions in the figure shows the design points where a task set is schedulable under both EDF and RMS policy; but produces different reduced utilization values.

4.2 Energy

A lower processor utilization opens up the opportunity to lower the operating frequency/voltage of the processor through voltage scaling technology. This may result in substantial energy savings. We employ the static voltage scaling algorithms for read-time systems proposed in [9]. Given a scheduling policy (RMS or EDF), the voltage scaling algorithm chooses the lowest operating voltage, frequency pair such that the task set still remains schedulable.

We first select the optimal customization for the task set under an area constraint. We apply static voltage scaling to obtain the lowest operating voltage/frequency corresponding to the original (no custom instructions) and the optimal configuration. We compare the energy consumptions corresponding to these two configurations over the hyper-period (the least common multiple of the task periods) of the task set. At some original utilizations, the task set is not schedulable without customization. In these cases, we perform the comparison w.r.t the first schedulable solution. We scale the frequency values from 300MHz (1.2 Volt) to 633MHz (1.6 Volt).

Figure 3 shows the relation between the hardware area and energy consumption under RMS and EDF scheduling policies. We can obtain up to 30% energy reduction. On an average, the energy reduction is 10% for RMS and 14% for EDF at 75% of *Max_Area*. Better energy savings for EDF is an artifact of the voltage scaling algorithm [9]. It can use aggressive voltage scaling for EDF policy due to its simpler schedulability test ($U \leq 1.0$). But for RMS, it uses a conservative schedulability condition that is sufficient but not necessary. In other words, it misses out certain opportunities and may select a higher operating frequency.

5 Conclusion

We explore instruction-set customization for multi-tasking real-time embedded systems. We propose algorithms to select inter-task optimal customizations under EDF and RMS scheduling that achieve significant reduction in processor utilization and overall energy consumption.

Acknowledgments

This work was supported by NUS project R252-000-171-112 and A*Star SERC project R-252-000-258-305.

References

[1] M. Arnold and H. Corporaal. Designing domain-specific processors. In *CODES*, 2001.

[2] K. Atasu, L. Pozzi, and P. Ienne. Automatic application-specific instruction-set extensions under microarchitectural constraints. In *DAC*, 2003.

[3] E. Bini and G. Buttazzo. The space of rate monotonic schedulability. In *RTSS*, 2002.

[4] N. Clark, H. Zhong, and S. Mahlke. Processor acceleration through automated instruction set customization. In *MICRO*, 2003.

[5] J. Cong et al. Application-specific instruction generation for configurable processor architectures. In *FPGA*, 2004.

[6] R. E. Gonzalez. Xtensa: A configurable and extensible processor. *IEEE Micro*, 20(2), 2000.

[7] P. Ienne and R. Leupers, editors. *Customizable Embedded Processors*. Morgan Kauffman, 2006.

[8] J. Lee, K. Choi, and N. Dutt. Efficient instruction encoding for automatic instruction set design of configurable asips. In *ICCAD*, 2002.

[9] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, 2001.

[10] F. Sun, S. Ravi, A. Raghunathan, and N.K.Jha. Application-specific heterogeneous multiprocessor synthesis using extensible processors. *IEEE TCAD*, 25(9), 2006.

[11] P. Yu and T. Mitra. Scalable custom instruction identification for instruction-set extensible processors. In *CASES*, 2004.

[12] P. Yu and T. Mitra. Satisfying real-time constraints with custom instructions. In *CODES+ISSS*, 2005.