IEICE TRANS. INF. & SYST., VOL.E93–D, NO.8 AUGUST 2010

PAPER

A Concurrent Instruction Scheduling and Recoding Algorithm for Power Minimization in Embedded Systems

Sung-Rae LEE[†], **Ser-Hoon LEE**[††], *Nonmembers*, *and* **Sun-Young HWANG**[††a)], *Member*

SUMMARY This paper presents an efficient instruction scheduling algorithm which generates low-power codes for embedded system applications. Reordering and recoding are concurrently applied for low-power code generation in the proposed algorithm. By appropriate reordering of instruction sequences, the efficiency of instruction recoding is increased. The proposed algorithm constructs program codes on a basic-block basis by selecting a code sequence from among the schedules generated randomly and maintained by the system. By generating random schedules for each of the basic blocks constituting an application program, the proposed algorithm constructs a histogram graph for each of the instruction fields to estimate the figure-of-merits achievable by reordering instruction sequences. For further optimization, the system performs simulated annealing on the generated code. Experimental results for benchmark programs show that the codes generated by the proposed algorithm consume 37.2% less power on average when compared to the previous algorithm which performs list scheduling prior to instruction recoding.

key words: *embedded system, low-power, instruction scheduling, recoding*

1. Introduction

The demand for embedded application devices that operate on batteries, such as mobile phones, MP3 players and notebook computers, has been growing explosively. Chips are being designed in response to the growing requirements of embedded applications. Whereas battery capacity growth has been rather slow, battery power requirement has been increasing due to the clock frequency increase to achieve high performance levels to satisfy system specifications and integration requirements. Thus, the importance of low-power design is increasingly critical in embedded system designs [1]. Various low-power design methods have been proposed and adopted including gated clocking, instruction recoding, DVS (Dynamic Voltage Scaling) [2], and DPM (Dynamic Power Management) [3].

ASIPs (Application Specific Instruction-set Processors) have the flexibility of general purpose processors and the high performance of ASICs. ASIPs are suitable for embedded system designs, where chips of high performance need to be designed within a short period of time. A retargetable compiler is essential for application analysis and code generation in the design of ASIPs [4]–[6]. By equipping a retargetable compiler with an efficient scheduling algorithm, application code with low power consumption can be generated. In [7], the power consumption of ASIP instruction memory was found to account for 30 percent or more of the entire processor's power consumption. Minimizing power consumption at the instruction bus is critical in low-power ASIP design. Numerous studies have reported that a minimization of bus line bit-switching can reduce bus power consumption. Lately it has been found that the amount of crosstalk between bus lines is also a critical power consumption factor as the buses are conventionally routed close to each other due to advances in deep submicron processes.

Instruction recoding is a technique which was originated from the low-power state assignment algorithm in designing sequential circuits. In the low-power state assignment algorithm, the state stationary probability vector and the total transition probability are calculated, whereupon the state is encoded to reduce bit switching frequency [8]. Instruction recoding is performed through graphic modeling of fields which constitute an instruction, such as opcodes or registers. Histogram graphs are constructed by mapping opcodes in a given instruction sequence as vertices, and connecting a pair of vertices whose opcodes appear in sequence as an edge. Edge weight is assigned according to the frequency of occurrence of the adjacent vertices. Low-power codes are generated by assigning a new binary code to each vertex, such that the hamming distance between a pair of vertices with large edge weight is minimized. In [9], instruction recoding was performed by simply considering the memory bit-line switching activities. Crosstalk is also taken into consideration, and the result of applying this recoding scheme is presented in [7] and [10]. In [11], instruction recoding is performed after low-power instruction scheduling based on the assumption that instruction recoding and scheduling are orthogonal. However, the number of symbol transitions depends on the order of instruction sequence. Instruction scheduling, which determines the order of instruction sequence, cannot be orthogonal to instruction recoding. Instruction recoding and low-power scheduling are mutually dependent. Results of instruction recoding can be different depending on the order of instruction sequence, and the low-power scheduling which uses instruction binary cannot be performed without recoded binary codes. In [10], a register perturbation method is presented for effective recoding through appropriate register reassignment. It adjusts register working sets in the direction of increasing the variance of histogram edge weights and the self-loop transition count.

Manuscript received December 28, 2009.
Manuscript revised March 21, 2010.

[†]The author is with the Telechips Inc., KORAD Bldg., 1000–12 Daechi-dong, Gangnam-gu, Seoul, 135–280, Korea.
[††]The authors are with the Department of Electronic Engineering, Sogang University, C.P.O. Box 1142, Seoul, 100–661, Korea.
a) E-mail: hwang@sogang.ac.kr

DOI: 10.1587/transinf.E93.D.2162

This is based on the fact that power consumption can be reduced by reassigning binary codes to the symbols with least hamming distance that are connected by an edge with the largest weight in the histogram graph. However, optimal results cannot be guaranteed because the change in the order of instruction sequence through scheduling is not considered.

In this paper we present an efficient instruction scheduling algorithm which can increase the variance of edge weights and the self-loop transition count in the histogram graph for efficient recoding. This paper consists of five sections. Section 2 describes the background and previous research results. Section 3 presents the proposed low-power code generation algorithm. Experimental results are presented in Sect. 4, and conclusions and future research are presented in Sect. 5.

2. Background and Related Works

In this section we present the bus power model and related works.

2.1 Bus Power Model

In a deep sub-micron process, the bus power consumption arises largely by self capacitance between the bus line and the ground, and the coupling capacitance between bus lines. Changes in voltage levels at each bus line result in charge/discharge of the self capacitance and power is consumed. Power consumption P_{self} by self capacitance is expressed in Eq. (1).

$$P_{self} = \frac{1}{2}\alpha C_{self} V_{DD}^2 \tag{1}$$

Here, α is the total number of single-bus-line bit transitions and C_{self} is the self capacitance. Power consumption by coupling capacitance is a serious power consumption factor as the bus line spacing is narrowed with the progress in deep sub-micron technology. In the case of a voltage change in a bus line due to coupling capacitance, it affects the adjacent bus line and becomes an additional power consumption factor. The effective coupling capacitance based on signal changes with adjacent bus lines of b_i and b_{i+1} are as shown in Table 1. C_{couple} is the actual coupling capacitance between bit lines.

When the same transitions occur along two adjacent bit lines, the coupling effect does not occur. When a transition occurs only at a bit line, the effective capacitance becomes C_{couple}, whereas when transitions occur in opposite directions at two adjacent bit lines, the effective capacitance becomes $2C_{couple}$. Thus, the power consumption P_{couple} from coupling capacitance is as expressed in Eq. (2) [12].

$$P_{couple} = \frac{1}{2}(\beta C_{couple} + \gamma\, 2C_{couple}) V_{DD}^2 \tag{2}$$

Here, β is the count where the effective capacitance between bit lines is C_{couple}, and γ for $2C_{couple}$. Total bus power P_{tot}, when considering both the self capacitance and the coupling capacitance, is the sum of power consumptions by the two factors and is expressed as in Eq. (3) where we set C_{couple}/C_{self} as λ [7].

Table 1 Crosstalk model.

b_{i+1} \ b_i	$0 \to 0$	$0 \to 1$	$1 \to 0$	$1 \to 1$
$0 \to 0$	0	C_{couple}	C_{couple}	0
$0 \to 1$	C_{couple}	0	$2C_{couple}$	C_{couple}
$1 \to 0$	C_{couple}	$2C_{couple}$	0	C_{couple}
$1 \to 1$	0	C_{couple}	C_{couple}	0

$$\begin{aligned} P_{tot} &= P_{self} + P_{couple} \\ &= \frac{1}{2}(\alpha + \beta\lambda + 2\gamma\lambda) C_{self} V_{DD}^2 \end{aligned} \tag{3}$$

2.2 Related Works

2.2.1 Instruction Recoding

Instruction recoding was proposed to generate low-power code suitable for a specific application by analyzing the performance pattern of the application program and reassigning the binary code. Histogram graphs are used for the analysis of application performance patterns. Histogram graph $H(V, E)$ is a weighted undirected graph, where vertex $v \in V$ is the symbol for considered instruction field and undirected edge $e \in E$ has the transition frequency between a vertex pair as the weight. In our paper we categorize edge types into self-loop edges, which connect to the same vertex, and transition edges, which connect to different vertices in histogram graph. For a transition between the same symbol, no dynamic power consumption arises at bus lines. In the instruction recoding algorithm, self-loop edges are not considered, but only transition edges are considered for performing instruction recoding.

Instruction recoding is performed on each of the fields that constitute an instruction. For opcode reassignment, an opcode histogram graph is constructed by the execution sequence of an opcode field within instructions and recoding is performed in the direction of reducing power consumption through analysis of the constructed opcode histogram graph. For register name adjustment, a register histogram graph is constructed through analysis of a register's usage sequence and recoding is performed to minimize the power consumption.

Figure 1 shows code sequence of an example basic block and the histogram graph constructed from it. In this example, we assume that basic block frequency is one. When basic block frequency is n, each of the edge weights of the constructed histogram graph becomes n times the present value. To solve the instruction recoding problem, Chattopadhyay et al. [7] obtained initial solution using MWP (Maximum Weighted Path) algorithm and applied

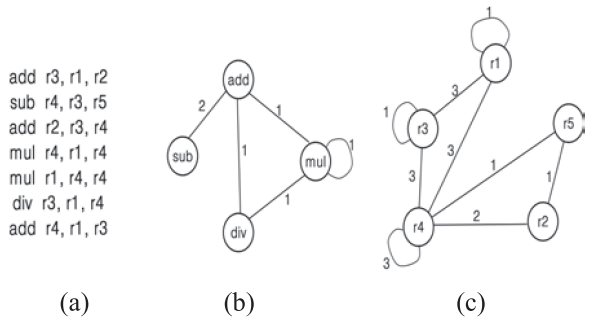

Fig. 1 Example of histogram graph. (a) Code sequence, (b) Opcode histogram graph, (c) Register histogram graph.

simulated annealing with the initial solution in implementing their recoding scheme.

2.2.2 Instruction Scheduling

Instruction scheduling can be used for diverse purposes such as improving the runtime performance of the target processor, avoiding pipeline hazards and reducing the power consumption. Instruction scheduling is performed by first decomposing a program code into basic blocks and constructing a data dependence graph for each basic block. A reservation table and an Instruction latency table are necessary for instruction scheduling. A reservation table includes resource information used in each instruction cycle, whereas an instruction latency table includes information on the latency to resolve RAW (Read After Write), WAW (Write After Write) and WAR (Write After Read) hazards. The instruction scheduler uses the reservation table for resource conflict prevention and the instruction latency information for data hazard prevention. Latency of a basic block can be measured through instruction scheduling of the basic block.

Cold scheduling for reducing power consumption through scheduling was first proposed in [13]. In their work, instruction scheduling was performed to minimize the hamming distance between instruction sequences under the assumption that the larger the hamming distance of instruction binary code, the greater the dynamic power consumption from switching activity. In [14], dynamic power consumed, when instruction pairs are performed in sequence, is obtained in a matrix format through simulation, and the TSP (Traveling Salesman Problem) algorithm was applied to perform cold scheduling. In [15], cold scheduling was expanded to VLIW and horizontal/vertical scheduling was proposed.

3. Proposed Instruction Scheduling Algorithm

In this section we present the motivation for this research, the figure-of-merit formulated and used in the algorithm, and the proposed scheduling algorithm for maximizing the figure-of-merit.

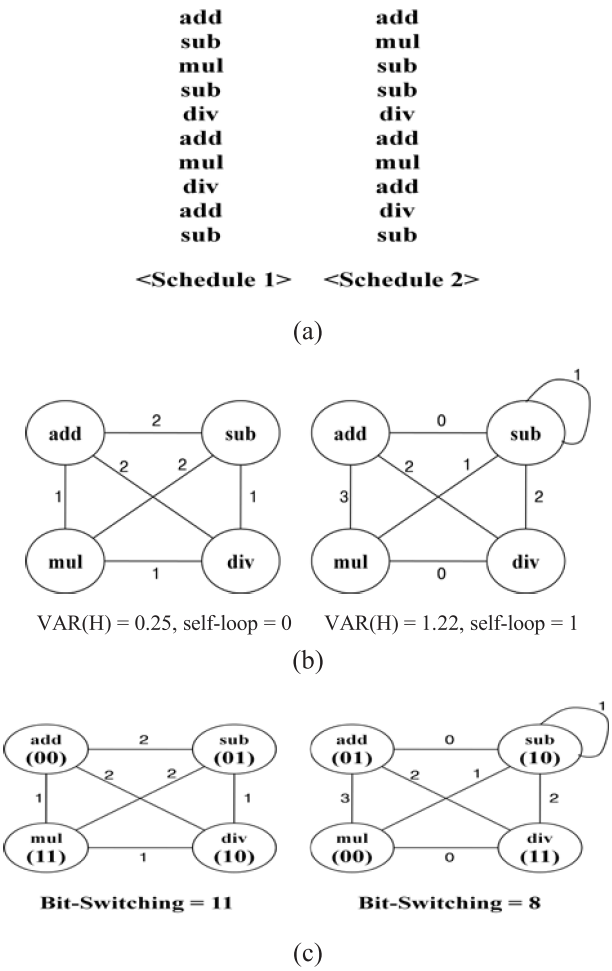

Fig. 2 Algorithm motivation example (a) Different scheduling results, (b) Constructed histogram graphs, (c) Optimal recoding results.

3.1 Motivation

Figure 2 (a) shows two different code sequences which are results of different instruction scheduling. The histogram graph for the two code sequences can be constructed as in Fig. 2 (b). Note that the edge weights of the histogram graph are different because the numbers of adjacent opcode pairs are not the same in these code sequences. And the second code sequence has larger values of self-loop edge weights and variance of transition edge weights than first one. Figure 2 (c) shows results of instruction recoding of each histogram graph. The instruction recoding results of the second histogram graph have less bit transitions than the first one. Power consumption depends on instruction order, and if the histogram graph has a larger sum of self-loop edge weights and a larger variance of transition edge weights, instruction recoding results are more optimal in terms of power consumption.

Instruction recoding adjusts the instruction binary code so that power consumption can be reduced in proportion to edge weights in a histogram graph. When the variance

of transition edge weights is increased, the inequality of edge weights is increased. An increased inequality of edge weights means that a small number of specific code sequences appear more frequently. The larger the sum of self-loop edge weights for transitions between the same symbols, which do not require dynamic power consumption, the greater the power saving effect of a code sequence will be. In summary, we can generate a code with less power consumption through recoding in instruction scheduling processes by maximizing the sum of self-loop edge weights and the variance of transition edge weights.

3.2 FM (Figure-of-Merit)

To generate a code sequence with minimal power consumption, instruction scheduling and recoding results must be considered together. The recoding effect can be increased by maximizing the sum of self-loop edge weights and the variance of transition edge weights through instruction reordering. The figure-of-merit $FM(H_f(S))$ on the histogram graph $H_f(S)$, calculated upon analysis of a specific instruction field f for a given schedule S in consideration of the above two factors, is derived as in Eq. (4).

$$VAR(H_f(S)) = \frac{1}{N}\sum_{i\in E-L}(w_i - w_{avg})^2$$
$$FM(H_f(S)) = \left(1 + \frac{(\sum_{i\in L} w_i)}{(\sum_{i\in E} w_i)}\right) * \left(1 + \frac{VAR(H_f(S))}{(\sum_{i\in E-L} w_i)^2}\right) \quad (4)$$

Here, N is the number of vertex pairs connected by transition edges, L is the set of self-loop transition edges and $E - L$ is the set of transition edges which connect to different vertices in histogram graph. The sum of self-loop edges are normalized by dividing by the sum of total edge weights, and $VAR(H_f(S))$ is normalized using the sum of transition edge weights. One is added to both two terms to make them greater than zero. The reason of normalizing $VAR(H_f(S))$ is that as the sum of self-loop edge weights is changed, so is the sum of transition edge weights, resulting in a change in the transition edge variance. Transition edge variance is a measure for inequality between transition edge weights and must be obtained independent of the sum of self-loop edge weights. Therefore, $VAR(H_f(S))$ is normalized to eliminate any dependency between the sum of self-loop edge weights and the variance of transition edges.

The figure-of-merit $FM(S)$ for a schedule S that considers all instruction fields f that constitute an instruction I is given in Eq. (5).

$$FM(S) = \sum_{f\in I} \frac{width(f)}{width(I)} * FM(H_f(S)) \quad (5)$$

The total figure-of-merit $FM(S)$ is the sum of the normalized values of figure-of-merit $FM(H_f(S))$ for individual field f with respect to the ratio of the field f width to the total instruction width.

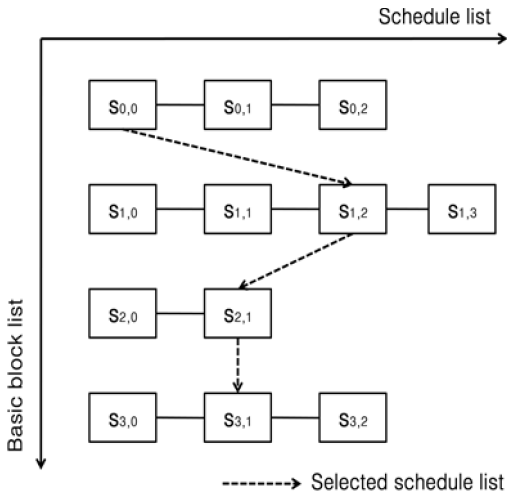

Fig. 3 Schedule selection.

3.3 Algorithm Overview

It is desirable that instruction scheduling be performed by constructed histogram graphs for the entire program in accordance with the figure-of-merit presented herein. When executing codes with binary code assigned through recoding, the figure-of-merit for the entire program needs to be maximized to minimize the power consumption. Since most instruction scheduling is performed in basic block level, global optimization for an entire program is difficult to implement. Here, we present the method where multiple schedule instances are generated for each basic block through random scheduling, whereupon an optimal solution is determined through proper selection of schedules for basic blocks. The proposed algorithm is performed within the limits of available computing power and memory capacity.

Figure 3 shows a summary of the proposed method. In the figure, a square represents the j-th random schedule $s_{i,j}$ for a basic block i. An optimal solution is determined from all schedules through a schedule selection process. Selection of an optimal solution requires global optimization considering all schedule consequences of basic blocks and is a difficult task. In this paper the proposed schedule selection scheme employs a 'greedy selection' process where a global schedule is constructed by processing a basic block sequentially from the first to the last basic block. An optimal solution is obtained through simulated annealing as a post optimization step.

3.4 Random Scheduling

The proposed algorithm generates random code sequences that satisfy semantics of each basic block as the first step. The random scheduling process considers the runtime performance as well as latency because power consumption tends to increase in proportion to latency. In the random scheduling, the latency obtained from the heuristic minimum-latency list scheduling is set as an upper bound,

```
Make_Schedules_for_BBs (BB_SET[ ])
begin
  for each BB in BB_SET[ ] do
  begin
    list_schedule_solution = LIST_SCHEDULE (BB);
    /* Upper bound in latency of considered BB */
    latency_UB = LATENCY (list_schedule_solution);
    Insert list_schedule_solution to Schedules_for_BBs[BB];
    /* try to find solutions by random scheduling */
    for i = 0 until ITERATION_COUNT (BB)   do
    begin
      new_schedule = RANDOM_SCHEDULE (BB);
      acceptable = False;
      if (LATENCY (new_schedule) <= latency_UB)
      begin
        acceptable = True;
        for each schedule solutions in Schedules_for_BBs[BB]do
        if (LATENCY (s) == LATENCY (new_schedule))
        begin
            /* measure similarity between two schedule solution */
            similarity_measure = COMPARE (s, new_schedule);
          if (similarity_measure>θth *LATENCY(new_schedule)) then
          begin
            accpetable = False;
            break;
          end
        end
      if (acceptable)
        Insert new_schedule to Schedules_for_BBs[BB];
      end
    end
  end
  return Schedules_for_BBs[ ];
end
```

Fig. 4 Proposed random scheduling algorithm.

whereupon the subsequent random schedules whose latency values are less than the upper bound are stored in the solution list. Here, the number of solutions in the solution list is limited so that solutions with fewer similarities to other solutions are kept to ensure a diverse schedule solution set. The similarity check is useful for obtaining an optimal solution in that it prevents solution duplicity and ensures mutually exclusive and diverse patterns. Multiple random schedules are attempted for each basic block in forming the solution list. The iteration count of random schedule generation for basic blocks is designed to be proportional to the product of the basic block size and the basic block frequency. This ensures that more diverse schedule instances are obtained for larger basic blocks and for basic blocks with higher execution frequency to facilitate determination of an optimal solution. Figure 4 shows a pseudo code for the procedure of generating a solution set through random scheduling.

For each BB in the basic block set BB_SET[],

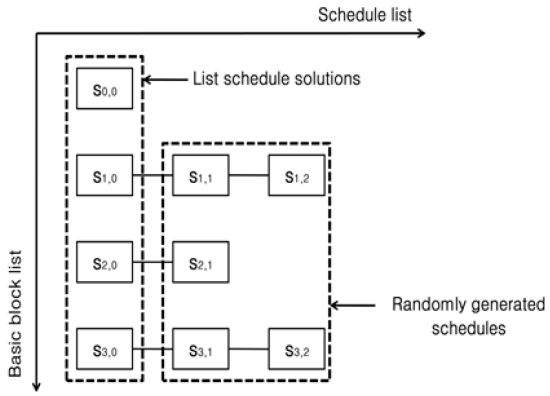

Fig. 5 Result of random scheduling.

performance oriented list scheduling is performed to generate solutions. It sets the latency of list_schedule_solution as the latency upper bound and includes it in Schedules_for_BBs[BB]. If any random schedule solution is not found in BB, list_schedule_solution becomes the only one solution. The algorithm generates random schedules for a suitable number of iterations for each BB. If a schedule's latency is less than latency_UB and its similarity measure with respect to other solutions with the same latency is small, it is inserted into Schedules_for_BBs[BB]. If a schedule's latency is less than latency_UB and no other solution with the same latency has been obtained, it is also inserted into Schedules_for_BBs[BB]. We generate a random schedule (new_schedule) and compare its latency with each of the schedules in Schedules_for_BBs[BB]. If two schedules have the same latency, the similarity of new_schedule is measured. Our instruction scheduler inserts a NOP instruction, if any instruction cannot be issued. Thus, two instruction sequences can be compared on a control step basis. At each control step, if the same instruction appears in two schedules, similarity_measure is increased by one. If similarity_measure is smaller than $\theta_{th} * \mathrm{LATENCY}(\mathrm{new\_schedule})$, the schedule is inserted into Schedules_for_BBs[BB]. Here, θ_{th} is the similarity threshold and has a value between 0 and 1. The larger the similarity threshold, the greater is the probability of discarding a new_schedule upon comparison. The above process is repeated for ITERATION_COUNT(BB).

Figure 5 shows the schedule set generated randomly. In our experiment, about 34% of randomly generated schedules are accepted when θ_{th} is set to be 0.7. Because all schedule solutions have less latency than performance-oriented list schedule solution in each basic block, execution performance can be preserved.

3.5 Schedule Selection

For each basic block, a schedule with an optimal result is selected from among multiple schedule solutions. We denote the global schedules of a specific application program as $S_{program}$ and the selected schedule of the ith basic block

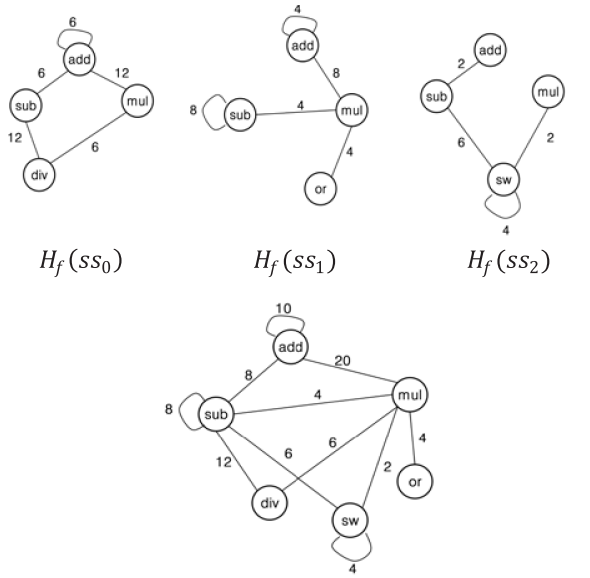

Fig. 6 Merge of histogram graphs.

as ss_i. $S_{program}$ consists of basic block schedules. Here, for convenience, we denote it as a set with basic blocks as its elements. FM of $S_{program}$ formed to obtain an optimal schedule of a specific application program must be the maximum as presented in Eq. (6). n is the number of basic blocks.

$$S_{program} = \{ss_0, ss_1, ss_2, \cdots ss_{n-1}\},\ maximize\ (FM(S_{program})) \quad (6)$$

$FM(S_{program})$ for $S_{program}$ can be expressed as Eq. (7) by applying Eqs. (4) and (5).

$$FM(S_{program}) = \sum_{f \in I} \left[\frac{width(f)}{width(I)} * \left(1 + \frac{(\sum_{i \in L} w_i)}{(\sum_{i \in E} w_i)}\right) * \left(1 + \frac{VAR(H_f(S_{program}))}{(\sum_{i \in E-L} w_i)^2}\right)\right] \quad (7)$$

$H_f(S_{program})$ can be attained by merging histogram graphs $H_f(ss_i)$ for all basic block schedules ss_i. Examination of Fig. 6 shows that, for opcode field f of an application program consisting of 3 basic blocks, $H_f(S_{program})$ can be obtained by merging $H_f(ss_0)$, $H_f(ss_1)$ and $H_f(ss_2)$.

Since $H_f(S_{program})$ is obtained by merging selected histogram graphs $H_f(ss_i)$ for each basic block schedule ss_i, $VAR(H_f(S_{program}))$ for a program consisting of n basic blocks can be expressed as Eq. (8) [16]. Here, $COV(H_f(ss_i), H_f(ss_j))$ is the covariance of two histogram graphs $H_f(ss_i)$ and $H_f(ss_j)$.

$$VAR(H_f(S_{program})) = \sum_{i=0}^{n-1} VAR(H_f(ss_i)) + 2 \sum_i \sum_{j<i} COV(H_f(ss_i), H_f(ss_j)) \quad (8)$$

```
GREEDY_SELECTION (Schedules_for_BBs[ ])
begin
  S_g = ø;
 for each BB_i in BB_SET[ ] do
 begin
  for each s_{i,j} in Schedules_for_BBs[BB_i] do
  begin
   for each k ∈ K do
                /* K is a set of schedules up to BB_{i-1} */
   begin
        S_t = S_g(i − 1, k) + s_{i,j};
    if (FM (S_t)) > FM (S_g(i, j)) then S_g(i, j) = S_t; end
   end
  end
  Find S_result[ ] whose FM value is maximum among
                schedules S_g(n − 1, k) for k ∈ K
  /* K is a set of schedules for last basic block BB_{n-1} */
  return S_result[ ];
end
```

Fig. 7 Greedy selection algorithm.

For $H_f(S_{program})$ maximization, ss_i must be selected so that self-loop transition of $H_f(S_{program})$ for each field is large, and the sum of transition variances of all ss_i and the covariance of transition edge weights of all ss_i are large. It is an NP-hard problem to maximize the covariance of transition edge weights of all ss_i while satisfying a number of other constraints. In this paper, we employ the method of a greedy selection followed by simulated annealing. For the greedy selection we use the dynamic programming method to achieve local cost maximization via a bottom-up approach while proceeding with the schedule selection to obtain the overall schedule. Let $s_{i,j}$ be the j-th random schedule of the i-th basic block and $S_g(i, j)$ be the set of selected schedules for up to $s_{i,j}$. $S_g(i, j)$ can be determined by selecting one with the largest value of figure-of-merit from among the schedules obtained by inserting $s_{i,j}$ into $S_g(i-1, k)$, for all k's, which are the schedules up to $(i-1)$-th basic block. Figure 7 shows a pseudo code of the greedy selection algorithm.

S_g is initialized to NULL at start. For all BB_i of BB_SET[ ], S_t becomes $S_g(i, j)$ where $FM(S_t)$ is larger than $FM(S_g(i, j))$ on the set S_t of random schedule solutions $s_{i,j}$ and $S_g(i-1, k)$, for all k's, which are the schedules up to $(i-1)$-th basic block BB_{i-1}. Global schedule $S_{result}[\ ]$, which has the largest figure-of-merit among selected schedule sets on the last $(n-1)$-th basic block, is returned.

3.6 Further Optimization

Our greedy selection process forms a schedule for the entire program by generating a schedule set that maximizes the figure-of-merit. Although this process is efficient, global

```
SIMULATED_ANNEALING (Schedules_for_BBs[ ], S_result[ ])
begin
  T = 100;
  FM_cur = FM (S_result[ ]);
  while (T > 1)
  begin
    BB_sel = SELECT_BB ( );
    ss_sel = SELECT_SCHEDULE (Schedules_for_BBs[ ]);
    ss_t = S_result[BB_sel];
    S_result [BB_sel] = ss_sel;
    FM_new = FM (S_result[ ]);
    difference = FM_new - FM_cur;
    if (difference > 0 || EXP (difference / T) > RANDOM ( )) then
      FM_cur = FM_new;
    else
      S_result[BB_sel] = ss_t;
    T = 0.9 * T;
  end
end
```

Fig. 8 Simulated annealing algorithm.

optimization cannot be guaranteed because schedules are generated by maximizing local figure-of-merit. Therefore, we apply a simulated annealing [17] scheme for further optimization. The simulated annealing process takes an initial solution and improves it iteratively. It obtains near-global optima beyond local optima.

Figure 8 shows the pseudo code of the applied simulated annealing algorithm. The simulated annealing procedure takes the global schedule $S_{result}[\ ]$ obtained by the greedy selection algorithm shown in Fig. 7 as initial solution. Selecting a schedule ss_{sel} not included in the current schedule set at each iteration, the procedure creates a new schedule set by replacing the schedule of the corresponding basic block with ss_{sel}. To ensure that optimization is focused on basic blocks of higher importance, basic blocks are randomly selected with a priority on the execution frequency of each basic block. A new schedule set is accepted based on its acceptance probability. It is naturally accepted if its figure-of-merit is larger than that of the current schedule. Otherwise, the algorithm returns to the previous condition prior to the schedule set swapping.

4. Experimental Results

To measure the performance of the proposed algorithm, we implemented the algorithm in the retargetable compiler with MIPS R3000 [18] as the target processor and performed an experiment. We used our retargetable simulator to measure execution performance and power consumption. For optimization purposes, we applied loop unrolling to make basic blocks larger. The larger basic block size has an advantage in that more random schedules are generated, which increases the chances of generating optimal result. We measured the reduction in power consumption between existing algorithms and the proposed algorithm using benchmark programs such as qsort (quick sort), fft (fast Fourier transform), ARIA (encryption algorithm) [19], JPEG (image compression algorithm), mat_inv (matrix inversion) and edge_detect (edge detection algorithm).

The proposed algorithm works greedily and has polynomial time complexity. Because of its exhaustive random scheduling and schedule selection, the proposed algorithm needs longer compile time. For example, JPEG encoder program consisting of 417 basic blocks, it takes about 2 hours to compile.

In the development environment for embedded systems, programs are developed, compiled, and downloaded to instruction ROM. They will be used without modification for their life time. Generated code quality is very important in embedded systems where resources are limited. High code quality can be achieved by compiler optimization effort, which requires longer compilation time. Because of the properties of one time compilation and high code quality requirement, longer compilation time is not critical in developing embedded systems. Our code generation algorithm is valuable in developing the programs running in embedded system environment.

To show the effectiveness of the proposed algorithm, the values of constituent factors in the FM of Eq. (4) were measured. For this, we measured the normalized sum $(\sum_{i\in L} w_i)/(\sum_{i\in E} w_i)$ of self-loop edges of the histogram graphs for the entire instruction field, and the normalized variance of transition edge $VAR(H_f(S))/(\sum_{i\in E-L} w_i)^2$. The results are presented in Table 2 and Table 3. As shown in the tables, our proposed algorithm generates instruction sequences which have larger sum of edge weights and variance of transition edges.

Table 4 compares the number of execution cycles of the codes generated by list schedule, min-latency schedule obtained from random schedules, and the schedule obtained by the proposed algorithm. For each of the benchmark programs, our proposed algorithm shows an increased performance compared with the list scheduling algorithm by about 2∼4%. Because we gathered random scheduling solutions in the list scheduling latency bound, the proposed algorithm shows better performance than the performance-oriented list scheduling algorithm. Thus, the proposed algorithm generates application codes consuming less power without performance degradation.

To measure power consumption cost, we defined the PCC (Power Consumption Cost) suggested in [7] as in Eq. (9) and used it as our measure of performance.

$$PCC = \alpha + \beta\lambda + 2\gamma\lambda \tag{9}$$

λ was set to be 3 in our experiment using a 0.18 μm process technology, as was used in Ref. [20]. The values of α, β, γ were obtained by simulation performed after assigning binaries to opcodes and register operands. Those values are

Table 2 Comparison of sum of self transition.

Benchmarks				No optimization	List scheduling	Proposed	
Name	# BBs	# lines in C	# lines in binary code			Random + Greedy selection	Simulated-annealing applied
qsort	17	52	142	0.173203	0.176303 (+1.8%)	0.200038 (+15.5%)	0.203569 (+17.5%)
fft	84	134	794	0.201902	0.202419 (+0.25%)	0.232538 (+15.2%)	0.246888 (+22.3%)
ARIA	270	502	3542	0.181827	0.182356 (+0.29%)	0.223538 (+22.9%)	0.240028 (+32.0%)
JPEG	417	1104	3377	0.188029	0.190881 (+1.5%)	0.241213 (+28.3%)	0.251181 (+33.6%)
mat_inv	92	212	957	0.217238	0.214559 (-1.2%)	0.246444 (+13.4%)	0.277413 (+27.7%)
edge_detect	117	427	1039	0.212912	0.212434 (-0.2%)	0.260119 (+22.2%)	0.279288 (+31.2%)
average	-	-	-		+0.4%	+19.6%	+27.4%

Table 3 Comparison of transition variance.

Benchmarks	No optimization	List scheduling	Proposed	
			Random + Greedy selection	Simulated-annealing applied
qsort	0.000275	0.000272 (-1.1%)	0.000388 (+41.1%)	0.000403 (+46.5%)
fft	0.000541	0.000543 (+0.4%)	0.000747 (+38.1%)	0.000800 (+47.9%)
ARIA	0.000269	0.000266 (-1.1%)	0.000380 (+41.3%)	0.000436 (+62.1%)
JPEG	0.000215	0.000218 (+1.4%)	0.000271 (+26.0%)	0.000278 (+29.3%)
mat_inv	0.000321	0.000324 (+0.9%)	0.000435 (+35.5%)	0.000467 (+45.5%)
edge_detect	0.000336	0.000339 (+0.9%)	0.000488 (+45.2%)	0.000540 (+60.7%)
average		+0.23	+37.9%	+48.7%

Table 4 Comparison of execution cycles.

Benchmarks	List scheduling	Min-latency schedule obtained from random schedules	Proposed	
			Random + Greedy selection + Recoding	Simulated-annealing applied
qsort	41,456	38,960 (-6.0%)	40,247 (-2.9%)	39,780 (-4.0%)
fft	149,998	141,295 (-5.8%)	144,346 (-3.7%)	143,399 (-4.3%)
ARIA	70,172	67,610 (-3.7%)	68,241 (-2.7%)	69,051 (-1.5%)
JPEG	10,122,706	9,763,486 (-3.5%)	9,942,327 (-1.7%)	9,842,331(-2.7%)
mat_inv	881,937	837,199 (-5.1%)	853,997 (-3.1%)	862,742 (-2.1%)
edge_detect	7,977,693	7,467,009 (-6.4%)	7,793,807 (-2.3%)	7,698,221(-3.5%)
average	-	-5.1%	-2.7%	-3.0%

※ The figures in the parenthesis are obtained when compared with the list scheduling results in the first column.

for the entire instruction fields. To demonstrate performance of our proposed algorithm against existing algorithms, we compared the results obtained by applying different methods. Table 5 lists PCC values for the benchmark application programs.

When comparing with the cases only recoding and only list scheduling are applied and the case list scheduling plus recoding is applied, we find that about 20~30% reduction of power consumption is obtained through recoding. Comparison of the proposed random scheduling followed by recoding with greedy selection and the case of list scheduling only yielded about a 35~55% reduction of power consumption. As Table 4 shows, numbers of execution cycles of the codes generated by the proposed algorithm are comparable to those of the min-latency schedule. In the proposed algorithm, recoding performance is greatly increased by maximizing self transitions and transition variance. However, the efficiency of recoding cannot be utilized in the min-latency schedule. The power consumption was achieved for all benchmark programs with respect to recoding only. Recoding with additional simulated annealing resulted in about a 45~65% reduction of power consumption, showing power

Table 5 Comparison of PCC values.

Benchmarks	No power optimization	List scheduling only	Min-latency schedule*	Recoding only	List scheduling + Recoding [7]	Min-latency schedule* + Recoding	Proposed	
							Random + Greedy selection + Recoding	Simulated-annealing applied
qsort	690,194	681,723 (-1.2%)	660,129 (-4.4%)	501,080 (-27.4%)	486,345 (-29.5%)	469,183 (-32.0%)	409,686 (-40.6%)	316,728 (-54.1%)
fft	3,397,982	3,277,839 (-3.5%)	3,189,123 (-6.1%)	2,495,771 (-26.6%)	2,533,770 (-25.4%)	2,519,238 (-25.9%)	2,041,986 (-39.9%)	1,618,803 (-52.3%)
ARIA	1,283,570	1,337,610 (4.2%)	1,219,273 (-5.0%)	940,438 (-26.7%)	920,277 (-28.3%)	931,992 (-27.4%)	604,731 (-52.9%)	441,675 (-65.6%)
JPEG	240,958,151	236,153,745 (-2.0%)	242,482,718 (0.6%)	187,321,245 (-22.3%)	178,768,386 (-25.8%)	172,621,714 (-28.4%)	155,396,607 (-35.5%)	130,314,207 (-45.9%)
mat_inv	22,184,565	20,698,419 (-6.7%)	21,130,273 (-4.8%)	16,771,531 (-24.4%)	16,694,322 (-24.7%)	16,172,819 (-27.1%)	12,617,034 (-43.1%)	10,618,542 (-52.1%)
edge_detect	144,681,027	146,174,640 (1.0%)	139,681,782 (-3.5%)	115,327,787 (-20.3%)	110,478,027 (-23.6%)	91,028,372 (-37.1%)	81,842,496 (-43.4%)	67,948,254 (-53.0%)
average	-	-1.36%	-3.85%	-24.6%	-26.2%	-29.6%	-42.6%	-53.9%

* Min-latency schedule is obtained from random schedules.

reduction in all benchmark programs over the case of applying greedy selection only. Thus, the proposed algorithm resulted in a significant performance improvement over existing recoding, which does not consider the order of instruction sequences.

5. Conclusion

In this paper, we proposed an instruction scheduling algorithm that generates a code for minimizing power consumption by considering both instruction scheduling and recoding simultaneously. The experimental results prove a significant performance improvement over the existing method. The algorithm generates schedules for each basic block through random scheduling and generates a code for the entire program by applying a schedule selection process on the generated schedules. The number of patterns without dynamic power consumption is increased by maximizing self-loop transitions, and the frequency of power saving patterns is increased through instruction recoding by maximizing the transition variance to ensure the generation of a low-power code.

The experiment results show that, whereas the recoding only method produced a power reduction of about only 20∼30%, application of the proposed method of recoding after scheduling yielded about a 45∼65% reduction of power consumption. The proposed algorithm is thus meaningful in that it can be a solution for generating low-power codes in the design of embedded systems, which are becoming increasingly high performance, highly integrated and require low-power designs.

Future research includes generation of low-power codes by maximizing the figure-of-merit presented in this paper by considering instruction scheduling and register allocation simultaneously.

Acknowledgement

This research work has been supported by the MEST (Ministry of Education, Science and Technology), through NRF (National Research Foundation) of Korea under Grant #2010-0008043.

References

[1] T. Makimoto and Y. Sakai, "Evolution of low power electronics and its future applications," Proc. ISLPED, pp.2–5, Aug. 2003.
[2] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," IEEE J. Solid-State Circuits, vol.35, no.11, pp.1571–1580, Nov. 2000.
[3] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.8, no.3, pp.299–316, June 2000.
[4] M. Jain, M. Balakrishnan, and A. Kumar, "ASIP design methodologies: Survey and issues," Proc. Int. Conf. VLSI Design, pp.76–81, Jan. 2001.
[5] R. Leupers, "Compiler design issues for embedded processors," IEEE Des. Test Comput., vol.19, no.4, pp.51–58, July-Aug. 2002.
[6] R. Leupers, M. Hohenauer, J. Ceng, H. Scharwaechter, H. Meyr, G. Ascheid, and G. Braun, "Retargetable compilers and architecture exploration for embedded processors," IEE Proc. Computers and Digital Techniques, vol.152, no.2, pp.209–223, March 2005.
[7] A. Chattopadhyay, D. Zhang, D. Kammler, E.M. Witte, R. Leupers, G. Ascheid, and H. Meyr, "Power-efficient instruction encoding optimization for embedded processors," Proc. Int. Conf. VLSI Design, pp.595–600, Jan. 2007.
[8] L. Benini and G. De Micheli, "State assignment for low power dissipation," IEEE J. Solid-State Circuits, vol.30, no.3, pp.258–268, March 1995.
[9] L. Benini, G. De Micheli, A. Macii, E. Macii, and M. Poncino, "Reducing power consumption of dedicated processors through instruction set encoding," Proc. GLSVLSI, pp.8–12, Feb. 1998.
[10] P. Petrov and A. Orailoglu, "Transforming binary code for low-power embedded processors," IEEE Micro, vol.23, no.3, pp.21–33, May-June 2004.
[11] R. Dimond, O. Mencer, and W. Luk, "Combining instruction coding and scheduling to optimize energy in system-on-FPGA," Proc. Symp. FCCM, pp.175–184, April 2006.
[12] E. Macii, M. Poncino, and S. Salerno, "Combining wire swapping and spacing for low-power deep-submicron buses," Proc. GLSVLSI, pp.198–202, April 2003.
[13] C. Su, C. Tsui, and A. Despain, "Saving power in the control path of embedded processors," IEEE Des. Test Comput., vol.11, no.4, pp.24–31, Winter 1994.
[14] K. Choi and A. Chatterjee, "Efficient instruction-level optimization

methodology for low-power embedded systems," Proc. Int. Symp. System Synthesis, pp.147–152, Sept. 2001.
[15] C. Lee, J. Lee, and T. Hwang, "Compiler optimization on instruction scheduling for low power," Proc. Int. Symp. System Synthesis, pp.55–60, Sept. 2000.
[16] S. Ross, Probability Models for Computer Science, Academic Press, 2002.
[17] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," Science Magazine, vol.220, no.220, pp.671–680, May 1983.
[18] G. Kane and J. Heinrich, MIPS RISC Architecture, Prentice Hall, 1992.
[19] D. Kwon, J. Kim, S. Park, S-H. Sung, Y. Schn, J-H. Song, Y. Yeom, E-J. Yoon, S. Lee, J. Lee, S. Chee, D. Han, and J. Hong, "New block cipher: ARIA," Proc. ICISC, pp.432–445, Nov. 2003.
[20] P. Sotiriadis and A. Chandrakasan, "Bus energy minimization by transition pattern coding (TPC) in deep sub-micron technologies," Proc. Int. Conf. ICCAD, pp.322–327, Nov. 2000.

Sung-Rae Lee received the B.S. degree in electronic engineering from Sogang University, Seoul, Korea, in 2007 and 2009. He is currently working at Telechips Inc. His current research interests include image processing, embedded system design and CAD systems.

Ser-Hoon Lee received the B.S. and M.S. degrees in electronic engineering from Sogang University, Seoul, Korea, in 2003 and 2005. He is currently working towards the Ph.D. degree in electronic engineering at Sogang University. His current research interests include embedded system design and CAD systems.

Sun-Young Hwang received the B.S. degree in electronic engineering from Seoul National University, Seoul, Korea, in 1976, the M.S. degree from Korea Advanced Institute of Science in 1978 and the Ph.D. degree in electrical engineering from Stanford University, California, U.S.A., in 1986. Since 1986, he has been with the Center for Integrated Systems at Stanford University, working on design of a high-level synthesis and simulation system. In 1986 and 1987, he held a consulting position at Palo Alto Research Center of Fairchild Semiconductor Corporation. In 1989, he joined the Department of Electronic Engineering at Sogang University, where he is now a professor. His current research interests include hardware/software co-design, DSP/VLSI systems design, and embedded system design.