

# Control Unit Synthesis Targeting Low-Power Processors\*

Chuan-Yu Wang and Kaushik Roy  
Electrical Engineering, Purdue University, West Lafayette, IN

## Abstract

With demands for reliability and further integration, reducing power consumption becomes a critical concern in today's processor design. Considering the different techniques to minimize power consumption and promote system's reliability, reducing switching activity of CMOS circuits is a promising area to be explored. Motivated by these, we propose two optimization schemes which can be incorporated into processor's control unit synthesis to lower power dissipation. The first one, a low-power decoding scheme, utilizes graph embedding and logic minimization techniques to refine the decoding structure in processor's control unit. To get further optimization for those control units in Nanoprogrammed or Microprogrammed architecture, the second scheme is proposed to optimally assign ZERO or ONE to the don't-care bits distributed in Nanocontrol memory or Control memory, to significantly reduce switching activity within the control unit and/or on the path from control unit to data processing unit. To achieve these two goals efficiently, we have used Pseudo-Boolean programming to optimize the synthesis parameters. Based on a subset of 8086 instruction set, experimental results show that 25.8 percent improvement is obtained by properly encoding instruction opcodes, and 4.9 to 16.6 percent improvement can be obtained from optimal don't-care bits assignment.

## 1 Introduction

Due to the rapid progress in VLSI technology, more and more circuits are being integrated into a single chip. With the effects of electromigration (EM) and overheating, power consumption has become a limiting factor for further integration. High power consumption may cause area overhead for cooling and reduce the reliability of the chips. It also limits the maximum clock rate of chips and shortens the operating life of batteries. In today's high-performance VLSI designs, it is therefore imperative to incorporate low-power techniques into automatic synthesis tools to achieve ultra low power.

Prior efforts to reduce power dissipation in CMOS circuits can be categorized into optimizations at the circuit level, at the logic gate level, and at higher levels. In [1], Chandrakasan et al. proposed parallelism

and pipelining to compensate the performance penalty for a low voltage power supply. In [2], Wilhelm et al. tried to reduce supply voltage and preserve high driving ability of circuits at the same time. Taking transistors' capacitances into account, Prasad et al. [4] proposed reordering the input signals to series-connected transistors in logic gates for lower power based on signal activity measure. Roy et al. [5] encoded states of FSM (Finite State Machine) to reduce switching activity at the input to the combinatorial module. Technology mapping targeting lower power dissipation have been proposed in [12]. In [6], the clock signal is gated to reduce circuit's switching activity when the next state in a FSM is the same as the current one. In [7], Gray code address encoding was proposed. Gray codes have the property that consecutive code numbers have a Hamming distance of one. Hence, for sequential addressing scheme, considerable power savings can be achieved for long address lines associated with large capacitance.

In this paper, we address two optimization procedures for processor's control unit synthesis (shown in Figure 1) to reduce the power consumption: *Optimization ONE* - to reduce power dissipation to decode the opcodes, a two-phase decoding scheme is proposed. It optimally assigns opcodes to minimize the switching activity at the input to the PLA decoders, and utilizes *logic minimization* to reduce the transistor count in PLA, and *Optimization TWO* - specifically for *Microprogrammed* or *Nanoprogrammed* architectures, we minimize switching activity at the output of control memory (*CM*) or nanocontrol memory (*nCM*). This optimization will reduce the switching activity within the control unit and on the path to data processing unit. We achieve this by optimally assigning *don't-care bits*.

The paper is organized as follows. Section 2 briefly introduces Pseudo-Boolean functions, which were used for optimizing synthesis parameters. Section 3 illustrates the low-power decoding scheme. Section 4 discusses optimal assignment of don't-care bits (distributed in *CM* or *nCM*) with polynomial time complexity. Section 5 presents the implementations and experimental results. Conclusions are given in Section 6.

## 2 Pseudo-Boolean Functions

*Definition* : A *Pseudo-Boolean* function is a function

\*This research was supported in part by NSF CAREER award and by IBM Corporation

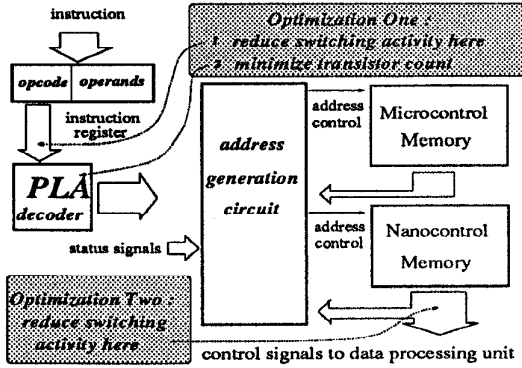


Figure 1: Two optimizations for control unit synthesis.

in which each variable's value could be either ZERO or ONE.

In our approach to achieve low-power design, we have used the the Pseudo-Boolean programming to optimize the design parameters. The transformation of functions from Boolean algebra to Pseudo-Boolean expressions can be performed using the following three equations:

1.  $a \text{ OR } b = a + b - ab$
2.  $a \text{ AND } b = ab$  ( $a$  multiplied by  $b$ )
3.  $\text{NOT } a = 1 - a$

For example, the Boolean function  $T_b = (a \text{ AND } b) \text{ OR } (c \text{ AND } d)$ , can be transformed into a corresponding Pseudo-Boolean expression  $T_p = ab + cd - abcd$ .

### 3 A Low-Power Decoder

To implement a low-power decoding system, we propose *encoding* opcodes to reduce the switching activity at the input to the decoders and *logic minimization* to reduce the literal count at each AND gate of PLA. This scheme reduces power consumed within decoders and the power to drive the decoders.

#### 3.1 Building the Transition Graph for Microinstructions : $TG_m$

To form the basis for our optimization procedures, we build a *transition graph* which represents the transition frequencies between all pairs of macroinstructions. In such a graph, each vertex denotes an opcode, and each edge is associated with a weight equal to the transition frequency between the two vertices (opcodes) connected by this edge. The process to build  $TG_m$  from a given set of benchmarks is straightforward. To obtain the weight of each edge in  $TG_m$ , we compute transition frequencies by inspecting the dynamic instruction stream while executing benchmarks. After calculating the transition frequencies between all pairs of instructions,  $TG_m$  can be built as follows : For each opcode of instruction  $I_x$ , a vertex marked  $x$  is created in  $TG_m$ . As long as the transition frequency between instruction  $I_x$  and  $I_y$  is not zero, we build an *edge* connecting vertex  $x$  and  $y$ , and associate a value  $E_{xy}$  which is equal to the frequency.

### 3.2 PHASE I - Encoding Opcodes

We define the *cost* of graph  $TG_m$  as :

$$T_m = \sum_{e_{ij} \in TG_m} E_{ij} * H(i, j), \quad (1)$$

where  $e_{ij}$  is the edge connecting vertex  $i$  and  $j$ ,  $E_{ij}$  is the value of the edge  $e_{ij}$ , and  $H(i, j)$  is the *Hamming Distance* between the codings of vertex (opcode)  $i$  and  $j$ . To minimize the average number of bit-switchings on the path to PLA, we minimize function  $T_m$  by optimally encoding the vertices (opcodes) in  $TG_m$ . Depending on the size of instruction set, we propose an exact and an approximate procedure as follows.

#### An Exact Procedure :

For a small instruction set, Pseudo-Boolean programming can be applied to obtain the optimum encodings of opcodes. The procedure can be described in the following two steps.

[STEP 1] Express  $T_m$  by Pseudo-Boolean expression - A binary variable  $b_{kx}$  can be associated with the  $x$ th bit of opcode (vertex)  $k$ . Now, Boolean-difference can be used to express (1) as:  $T_m = \sum_{\text{over all edges in } TG_m} E_{ij} * (\sum_{l=1}^L (b_{il} \oplus b_{jl}))$ , where  $L$  is the length of coding.  $T_m$  can be further transformed into a Pseudo-Boolean function :  $T_m = \sum_{\text{over all edges in } TG_m} E_{ij} * (\sum_{l=1}^L (b_{il} + b_{jl} - 2b_{il}b_{jl}))$ . To guarantee there are no two opcodes having the same code, the following *restriction set* can be attached to  $T_m$ :  $R = \{\forall i, j \prod_{l=1}^L (1 - (b_{il} \oplus b_{jl})) = 0\}$ .  $R$  can be transformed into a Pseudo-Boolean expression:  $R = \{\forall i, j \prod_{l=1}^L (1 - b_{il} - b_{jl} + 2b_{il}b_{jl}) = 0\}$ .

[STEP 2] Transforming this problem into an unconstrained programming model - We multiply a positive number  $M$ , which is greater than the maximum value of  $T_m$  (for this problem,  $M$  could be chosen as  $1 + L \sum_{\text{over all edges in } TG_m} E_{ij}$ ), to each term  $(1 - b_{il} - b_{jl} + 2b_{il}b_{jl})$  in  $R$ . ( $M * R$ ) is then added to  $T_m$  to form a new function  $T_u$  :

$$T_u = \sum_{e_{ij} \in TG_m} E_{ij} * \left( \sum_{l=1}^L (b_{il} + b_{jl} - 2b_{il}b_{jl}) \right) + M * \sum_{e_{ij} \in TG_m} \prod_{l=1}^L (1 - b_{il} - b_{jl} + 2b_{il}b_{jl})$$

where  $e_{ij}$  is an edge of  $TG_m$ . It is obvious that if the value of binary variables in  $T_u$  does not satisfy the restriction set  $R$ , it is impossible for the resulting value of  $T_u$  to reach its minimum. If the value of binary variables satisfies  $R$ ,  $T_u$  is equal to  $T_m$ . As a result, function  $T_m$  and  $T_u$  have the same minimum value and reach their minimums simultaneously. Typical branch-and-bound algorithms in Pseudo-Boolean programming (such as [8]) can be applied to minimize  $T_u$  to obtain optimal opcode encodings.

Table 1: Implicant table of a decoder

| Index of implicant | Input Part (opcodes) | Output Part (addresses) |
|--------------------|----------------------|-------------------------|
| P1                 | 001                  | 0000                    |
| P2                 | 100                  | 0100                    |
| P3                 | 010                  | 0110                    |
| P4                 | 101                  | 1001                    |
| P5                 | 000                  | 1100                    |

Table 2: Implicant table after logic minimization

| Index of implicant | Input Part (opcodes) | Output Part (addresses) |
|--------------------|----------------------|-------------------------|
| P1                 | 0*1                  | 0000                    |
| P2                 | 1*0                  | 0100                    |
| P3                 | *1*                  | 0110                    |
| P4                 | 1*1                  | 1001                    |
| P5                 | 000                  | 1100                    |

#### An Approximate Procedure :

For a large instruction set, we propose to apply *graph embedding* [9] techniques on  $TG_m$  to obtain acceptable encodings within reasonable CPU time. *Graph embedding* techniques are typically used to assign binary codes to the vertices of a graph based on given edge weights. For our analysis, we apply the *wedge clustering* algorithm in [3] to *embed* codings to the vertices of  $TG_m$  (Fig. 4) to minimize  $T_m$ . The application is straightforward, and hence, the algorithm descriptions have been omitted.

### 3.3 PHASE II - Logic Minimization

After the opcodes have been optimally encoded, the *implicant table* is defined for the PLA decoder. Because distinct opcodes of macroinstructions are decoded to access distinct addresses in  $CM$ , we cannot reduce the cardinality - which is always equal to the size of the instruction set. But there are don't-care terms which can be used to minimize the literal count of each implicant's input part. As a result, the AND gates which implement the input parts have the minimum transistor count. By above reasoning, logic minimization can be applied to reduce the power consumption to decode opcodes. After logic minimization, the PLA which implements the decoder has the minimum transistor count in its input plane. For example, for the decoder with the implicant table in Table 1, the resultant table after logic minimization is shown in Table 2.

## 4 Assigning Don't-Care Bits

We will explain how to optimally assign the don't-care bits in a nanocontrol memory with polynomial time complexity (i.e. polynomial in the number of don't-care bits in  $nCM$ ). This optimization minimizes the switching activity at the output of  $nCM$ , and thus reduces the switching activity in control and data processing unit.

### 4.1 Building the Transition Graph for Nanoinstructions : $TG_n$

The transition frequencies (between opcodes of instructions) and occurrence frequencies (of opcodes of instructions) can be easily obtained by executing a set of benchmark programs. Based on these measurements, the *nanoinstruction transition frequencies* can be computed from the given nanoinstruction sequences associated with instructions.

After calculating the transition frequencies for all pairs of nanoinstructions, the transition graph of nanoinstructions ( $TG_n$ ) can be built as follows. For each nanoinstruction ( $m$ ), a vertex (marked  $m$ ) is created in  $TG_n$ . An edge associated with the the transition frequency of nanoinstruction  $m$  and  $l$  is created to connect vertex  $l$  and  $m$ .

### 4.2 Formulating the Problem

The average switching activity at the output of the  $k$ th ( $k \leq$  (bit dimension of nanocontrol memory)) bit position of nanocontrol memory can be expressed as the cost of  $TG_n$  :  $T_k = \sum_{\text{over all edges in } TG_n} P_{ij} H_k(i, j)$ , where  $T_k$  is the objective function we want to minimize,  $P_{ij}$  is the transition frequency between nanoinstruction  $i$  and  $j$  (and  $P_{ij}$  is also the value associated with the edge connecting vertex  $i$  and  $j$  in  $TG_n$ ), and  $H_k(i, j)$  represents the Hamming distance between the  $k$ th bits of vertex (nanoinstruction)  $i$  and  $j$ .

The term  $H_k(i, j)$  can be alternatively expressed as  $bit_{ik} \oplus bit_{jk}$ , where  $bit_{ik}$  is the  $k$ th bit of the coding for nanoinstruction  $i$ , and  $bit_{jk}$  is the  $k$ th bit of the coding for nanoinstruction  $j$ . We can transform  $(bit_{ik} \oplus bit_{jk})$  into a Pseudo-Boolean expression  $bit_{ik} + bit_{jk} - 2bit_{ik}bit_{jk}$ . Then the average switching activity at the output of the  $k$ th column can be formulated as :

$$T_k = \sum_{e_{ij} \in TG_n} P_{ij} (bit_{ik} + bit_{jk} - 2bit_{ik}bit_{jk}) \quad (2)$$

where  $e_{ij}$  is an edge of graph  $TG_n$ , and  $bit_{ik}$  ( $bit_{jk}$ ) is either a constant (0 or 1) or a don't-care bit. After substituting the values of  $bit_{ik}$  ( $bit_{jk}$ ) into  $T_k$  and simplification, the resultant form of  $T_k$  is a quadratic Pseudo-Boolean function. Considering the *Burroughs D-machine* as an example, the word dimension of its  $nCM$  is 123. If the distribution of don't-care bits in its  $nCM$  is 35 %, each column of  $nCM$  has about 43 don't-care bits which have to be optimally assigned.

We will show in the following subsection that  $T_k$  can be minimized with polynomial complexity.

### 4.3 Assignment of Don't-Care Bits

For  $k$ th column in  $nCM$ , we obtain the corresponding Pseudo-Boolean function  $T_k$  ( $k \in \{1, 2, 3, \dots, \text{the bit dimension of } nCM\}$ ) which represents the average switching activity at the output of this column. Then the don't-care bits in this column can be optimally assigned by minimizing  $T_k$ . After eliminating the constant term,  $T_k$  is a quadratic function of the following form:

$$T_k = \sum_{j=1}^v (C_{jj}d_j) - \sum_{i=1}^v \sum_{j=1, j \neq i}^v (C_{ij}d_jd_i) \quad (3)$$

where  $v$  is the number of binary variables in  $T_k$ ,  $C_{jj}$  is the coefficient of the linear term, and  $C_{ij}$  is the coefficient of the quadratic term.  $C_{jj}$  and  $C_{ij}$  are always greater than or equal to 0.

For the Pseudo-Boolean function  $T_k$  of the form in (3), *Network Flow* algorithms have been proposed to minimize it in polynomial time (polynomial in the number of binary variables in the function) [10] [11]. Here we briefly explain the process proposed in [10]:

[STEP 1] Build the corresponding *network*  $\Lambda$  for function  $T_k$ : The *Source* vertex (denoted by  $S$ ) and *Sink* vertex (denoted by  $D$ ) are created in  $\Lambda$ . For each binary variable  $d_m$  in  $T_k$ , an *intermediate* vertex marked  $m$  is created in  $\Lambda$ . Therefore, there are  $(v + 2)$  vertices in network  $\Lambda$ , where  $v$  is the number of variables in  $T_k$ . For each coefficient  $C_{ij}$  of the term  $d_i d_j$  in  $T_k$  ( $i \neq j$ ), we build an edge in  $\Lambda$  which connects the corresponding vertices of variable  $d_i$  and  $d_j$ , and associate this edge with the value of  $C_{ij}$ . For each variable  $d_j$  in  $T_k$ , we check the value of  $\max(\sum_{k=1, k \neq j}^v C_{kj} - C_{jj}, 0)$ , where  $v$  is the number of variables in  $T_k$ ,  $C_{kj}$  is the coefficient of the term  $d_k d_j$  in  $T_k$ , and  $C_{jj}$  is the coefficient of the term  $d_j$  in  $T_k$ . If this value is greater than or equal to 0, an edge associated with this value is built in  $\Lambda$  to connect vertex  $j$  and Source vertex  $S$ . For each variable  $d_j$  in  $T_k$ , we also check the value of  $\max(C_{jj} - \sum_{k=1, k \neq j}^v C_{kj}, 0)$ , where  $v$  is the number of variables in  $T_k$ ,  $C_{kj}$  is the coefficient of the term  $d_k d_j$  in  $T_k$ , and  $C_{jj}$  is the coefficient of the term  $d_j$  in  $T_k$ . If this value is greater than or equal to 0, an edge associated with this value is built in  $\Lambda$  to connect vertex  $j$  and Sink vertex  $D$ .

[STEP 2] Finding the minimum cut (the cut with the minimum capacity) in  $\Lambda$ : Once  $\Lambda$  has been built, *maximum flow* algorithms such as *Ford-Fulkerson method*, or the *preflow-push algorithm* can be applied to  $\Lambda$  to find a *minimum cut* in  $\Lambda$ . The complexity of these maximum flow algorithms are polynomial in the number of vertices in  $\Lambda$ . By the minimum cut, the binary variables in  $T_k$  are assigned ONE if their corresponding intermediate vertices in  $\Lambda$  are in the same set with Source  $S$ , or assigned ZERO if their corresponding in-

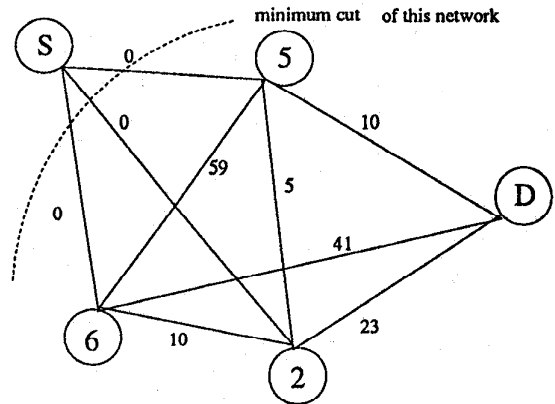


Figure 2: Corresponding network of  $T_x$

Table 3: Reduction (%) of the Switching Activity by Optimal Encoding (encoding length=5)

| Set of Inputs | Bubble (% Red) | Insert (% Red) | Select (% Red) |
|---------------|----------------|----------------|----------------|
| 1             | 5.49           | 30.23          | 37.08          |
| 2             | 12.65          | 31.61          | 35.49          |
| 3             | 10.93          | 28.16          | 35.05          |
| 4             | 11.98          | 34.23          | 36.32          |
| 5             | 8.49           | 27.11          | 36.68          |
| 6             | 14.59          | 30.58          | 35.98          |
| 7             | 14.48          | 32.12          | 35.68          |
| 8             | 10.03          | 29.62          | 36.60          |
| 9             | 11.20          | 31.89          | 36.48          |
| 10            | 9.26           | 30.09          | 36.77          |
| Ave.          | 10.91          | 30.56          | 36.21          |

intermediate vertices in  $\Lambda$  are in the same set with Sink  $D$ .

By the above procedure, the don't-care bits in  $nCM$  can be optimally assigned column by column with polynomial complexity. For example, for  $T_x = 38d_2 + 74d_5 + 110d_6 - 10d_2d_5 - 20d_2d_6 - 118d_5d_6$ , the resultant network  $\Lambda$  has 5 vertices and 9 edges (six edges have the capacity greater than 0) and is shown in Figure 2. By finding the minimum cut of  $\Lambda$ , the optimal assignment to minimize  $T_x$  is  $(d_2, d_5, d_6) = (0, 0, 0)$ .

## 5 Implementation and Results

The architecture used for our experimentation has an instruction set of size 20, which is a subset of 8086 instruction set. The size of nanocontrol memory is 40 bits by 40 bits. Besides the don't-care bits, the number of ZEROs and ONES in  $nCM$  are about even. The maximum length of the nanoinstruction sequences associated with the macroinstructions is 30 (i.e. a macroin-

Table 4: Reductions (%) of the Switching Activity by Optimal Encoding (encoding length=6)

| Set of Inputs | Bu. (% Red.) |       | In. (% Red.) |       | Se. (% Red.) |       |
|---------------|--------------|-------|--------------|-------|--------------|-------|
|               | 3-6          | Ran.  | 3-6          | Ran.  | 3-6          | Ran.  |
| 1             | 20.48        | 21.77 | 42.92        | 43.81 | 47.73        | 49.21 |
| 2             | 24.56        | 24.54 | 43.07        | 43.61 | 46.88        | 47.93 |
| 3             | 22.61        | 23.12 | 40.43        | 40.73 | 46.51        | 47.41 |
| 4             | 26.19        | 25.80 | 45.18        | 46.09 | 47.44        | 48.69 |
| 5             | 22.47        | 23.36 | 40.72        | 41.09 | 47.61        | 48.92 |
| 6             | 26.88        | 26.62 | 42.70        | 43.01 | 47.28        | 48.28 |
| 7             | 26.16        | 25.92 | 43.49        | 43.97 | 47.02        | 47.99 |
| 8             | 24.05        | 24.46 | 42.28        | 42.83 | 47.56        | 48.85 |
| 9             | 24.90        | 25.00 | 43.75        | 44.45 | 47.54        | 48.81 |
| 10            | 23.71        | 24.19 | 42.73        | 43.38 | 47.68        | 49.01 |
| Ave.          | 24.25        | 24.47 | 42.72        | 43.29 | 47.23        | 48.51 |

struction induces at most 30 nanoinstructions). We have three programs (*bubble sort*, *selection sort* and *insertion sort*) as the benchmark set of this architecture. We run these programs to obtain the dynamic measurements for building transition graphs ( $TG_m$  and  $TG_n$ ). We utilize the *wedge clustering* algorithm (mentioned in Section 3.2) to optimally encode opcodes. For each column in  $nCM$ , *Ford-Fulkerson* method is utilized to find the *minimum cut* of the corresponding network of its  $T_k$ . According to this minimum cut, we obtain the optimal assignments for the don't-care bits in this column.

Results of *Optimally Encoding Opcodes* are shown in Table 3 and 4. We encode the opcodes with different encoding length (L) in these two tables. In Table 3, the minimum encoding length L=5 is chosen. Resultant switching activities on the path to PLA are compared with 2 different encoding schemes - *Graph-Embedding*, and *Random Assignment*. We implement the *Graph-Embedding* scheme by *wedge clustering* algorithm mentioned in Section 3.2. The *Random Assignment* scheme randomly selects 20 distinct binary codes (with L=5) and randomly assigns them to the opcodes. In Table 3, the average reduction of bit switchings is 25.8 %.

With the encoding length equal to 6 (L=6), we summarize the reductions of switching activity (%) of *Graph-Embedding over 3 out of 6 encoding* and *Graph-Embedding over Random Assignment* respectively in Table 4. The *Random Assignment* scheme randomly selects 20 distinct binary codes (with L=6), and randomly assigns them to our instruction set. The *3 out of 6 encoding* scheme randomly selects 20 distinct binary codes with 3 ONEs and 3 ZEROs in the codings, then randomly assigns them to the opcodes. The *Graph-Embedding* scheme is also implemented by the *wedge clustering* algorithm. In table 4, the average reduction of switching activities is 38.4 % by graph-embedding.

Results of Optimal Don't-Care Bit Assignments are shown in Table 5 and 6. Each table has its own set of nanoinstruction sequences associated with the macroin-

Table 5: Reductions (%) of the Switching Activity - For the first set of nanoinstruction sequences

| Distr.         | In. Set | Bu. (% Red.) |       | In. (% Red.) |       | Se. (% Red.) |       |
|----------------|---------|--------------|-------|--------------|-------|--------------|-------|
|                |         | Uni.         | Ran.  | Uni.         | Ran.  | Uni.         | Ran.  |
| 35% don't care | 1       | 4.90         | 15.41 | 5.62         | 16.70 | 5.47         | 16.77 |
|                | 2       | 4.51         | 15.18 | 5.38         | 16.49 | 5.30         | 16.58 |
|                | 3       | 4.55         | 15.26 | 5.38         | 16.41 | 5.26         | 16.53 |
|                | 4       | 4.67         | 15.09 | 5.45         | 16.62 | 5.39         | 16.68 |
|                | 5       | 4.70         | 15.42 | 5.55         | 16.58 | 5.41         | 16.72 |
| 30% don't care | 1       | 3.22         | 10.84 | 5.73         | 15.71 | 5.89         | 16.31 |
|                | 2       | 3.28         | 11.17 | 5.53         | 15.50 | 5.62         | 15.87 |
|                | 3       | 3.26         | 11.18 | 5.37         | 15.10 | 5.55         | 15.77 |
|                | 4       | 3.31         | 11.01 | 5.73         | 15.91 | 5.76         | 16.09 |
|                | 5       | 3.21         | 10.94 | 5.49         | 15.19 | 5.81         | 16.18 |
| 40% don't care | 1       | 4.96         | 20.23 | 6.39         | 22.10 | 6.52         | 22.00 |
|                | 2       | 4.60         | 19.62 | 6.12         | 21.72 | 6.25         | 21.68 |
|                | 3       | 4.59         | 19.75 | 5.98         | 21.65 | 6.18         | 21.61 |
|                | 4       | 4.81         | 19.68 | 6.32         | 21.88 | 6.39         | 21.84 |
|                | 5       | 4.77         | 19.97 | 6.16         | 21.90 | 6.44         | 21.90 |
| Ave.           |         | 4.22         | 15.38 | 5.74         | 17.96 | 5.81         | 18.16 |

structions. The average length of these two sequences is 13.9 and 30 respectively. In each table, reductions of switching activities are compared with three different distributions of don't-care bits (35 %, 30 % and 40 %). For each combination of the set of nanoinstruction sequences,  $nCM$  and the set of the data to be sorted, we compare the resultant switching activities at the output of  $nCM$  with 3 different assignment schemes - MAXIMUM FLOW, RANDOM and UNIFORM. The MAXIMUM FLOW scheme assigns those don't-care bits by the procedure in Section 4. The RANDOM scheme randomly assigns ZERO or ONE to the don't-care bits in  $nCM$ . The UNIFORM scheme assigns ZERO (or ONE) to all don't-care bits uniformly. From Table 5 and 6, The average reduction (of switching activities) of MAXIMUM FLOW over UNIFORM is 4.91 %, and the reduction of MAXIMUM FLOW over RANDOM is 16.65 %.

## 6 Conclusions and future work

In the paper, we propose methodologies to reduce the processor's power consumption by refining its control unit. To achieve the low-power decoder, we apply branch-and-bound algorithms to encode macroinstructions optimally, and the PLA for decoding can be minimized by logic minimization. Furthermore, we apply Maximum Flow algorithms to assign microcode don't-care bits optimally in polynomial time. Experiments show 25.8% reduction of bit switching can be achieved by optimally encoding macroinstructions, and 4.9% to 16.6% reduction can be achieved by optimally assigning the don't-care bits.

Table 6: Reductions (%) of the Switching Activity - For the second set of nanoinstruction sequences

| Distr.         | In. Set | Bu. (% Red.) |       | In. (% Red.) |       | Se. (% Red.) |       |
|----------------|---------|--------------|-------|--------------|-------|--------------|-------|
|                |         | Uni.         | Ran.  | Uni.         | Ran.  | Uni.         | Ran.  |
| 35% don't care | 1       | 4.77         | 14.38 | 4.94         | 14.79 | 4.91         | 14.71 |
|                | 2       | 4.50         | 14.42 | 4.89         | 14.80 | 4.86         | 14.74 |
|                | 3       | 4.48         | 14.44 | 4.88         | 14.82 | 4.85         | 14.75 |
|                | 4       | 4.54         | 14.34 | 4.91         | 14.77 | 4.89         | 14.73 |
|                | 5       | 4.48         | 14.44 | 4.93         | 14.83 | 4.90         | 14.72 |
| 30% don't care | 1       | 4.33         | 13.14 | 4.75         | 13.44 | 4.86         | 13.58 |
|                | 2       | 4.48         | 13.32 | 4.77         | 13.56 | 4.83         | 13.61 |
|                | 3       | 4.43         | 13.25 | 4.72         | 13.49 | 4.82         | 13.61 |
|                | 4       | 4.50         | 13.34 | 4.81         | 13.58 | 4.85         | 13.60 |
|                | 5       | 4.39         | 13.20 | 4.71         | 13.40 | 4.85         | 13.59 |
| 40% don't care | 1       | 3.87         | 18.92 | 4.87         | 21.18 | 4.99         | 21.36 |
|                | 2       | 3.93         | 18.87 | 4.78         | 20.95 | 4.84         | 21.05 |
|                | 3       | 3.90         | 18.90 | 4.70         | 20.79 | 4.81         | 20.98 |
|                | 4       | 3.96         | 18.88 | 4.89         | 21.17 | 4.91         | 21.20 |
|                | 5       | 3.89         | 18.88 | 4.76         | 20.92 | 4.90         | 21.26 |
| Ave.           |         | 4.29         | 15.51 | 4.52         | 16.43 | 4.87         | 16.49 |

## References

- [1] A.P.Chandrakasan, S.Sheng, and R.W.Brodersen, "Low-Power CMOS Digital Design," *Journal of Solid-State Circuit*, Vol.27, No.4, pp.473-483, April,1992.
- [2] W. Wilhelm, P. Weger, "2V Bipolar Low-Power Logic", *IEEE Intl. Solid-State Circuits Conf.*, pp.94-95, 1994.
- [3] S. Devadas, H. Ma, A. R. Newton and A. S. Vincentelli, "MUSTANG : State Assignment of Finite State Machines Targeting Multilevel Logic Implementations", *EEE Trans. Computer-Aided Design*, Vol. 7, No. 12, pp.1290-1299, December, 1988.
- [4] S.Prasad and K.Roy, "Circuit Optimization for Minimization of Power Consumption under Delay Constraint," *IEEE VLSI Design Conf.*, New Delhi, 1995.
- [5] K.Roy and S.Prasad, "Circuit Activity Based Logic Synthesis for Low Power Reliable Operations", *IEEE Trans. VLSI Systems*, Vol.1, No. 4, pp.503-513, December, 1993.
- [6] L.Benini, P.Siegel and G.De Micheli, "Saving Power by Synthesizing Gated Clocks for Sequential Circuits", *IEEE Design and Test of Computers*, Vol. 11, No. 4, pp.32-41, winter,1994.
- [7] C.Su, C.Tsui, and Alvin M. Despain, " Saving Power in the Control Path of Embedded Processors", *IEEE Design and Test of Computers*, Vol. 11, No. 4, pp.32-41, winter,1994.
- [8] P.L.Hammer and U.N.Peled, "On the Maximization of a Pseudo-Boolean Function", *J.A.C.M.*, Vol. 19, No. 2, pp.265-282, April 1972.
- [9] M.R.Garey and D.S.Johnson, *Computer and Intractability, A Guide to the Theory of NP-Completeness*, W.H.Freeman, New York, second edition, 1979.
- [10] J.C.Picard and H.D.Ratliff, "Minimum Cuts and Related Problems" , *Networks* 5, pp.357-370, 1975.
- [11] J.Rhys, "A Selection Problem of Shared Fixed Costs and Network Flows" , *Management Sci.*, Vol. 17, pp.200-207, 1970.
- [12] C. Tsui, M. Pedram, and A. Despain, "Technology Decomposition and Mapping Targeting Low Power Dissipation," *ACM/IEEE Design Automation Conf.*, 1993, pp.68-73.