Energy Efficient Array Initialization Using Loop Unrolling with Partial Gray Code Sequence

Sumanta Pyne and Ajit Pal

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, West Bengal, 721 302, India
{spyne,apal}@cse.iitkgp.ernet.in

Abstract. The present work introduces a software technique to reduce energy consumed by the address bus of the on-chip data memory. This is done by reducing switching activity on the address bus of the on-chip data memory, with the help of loop unrolling with partial Gray code sequence. The present work introduces the translation of a loop with array initialization to its loop unrolled version with partial Gray code sequence. The expressions for switching activity consumed on the address bus of data memory are derived for both unrolled loop with and without partial Gray code sequence. The proposed translation method finds a relocatable base address of the array so that the partial Gray code sequence is maintained, without any energy-performance overhead and achieves a considerable amount of energy reduction without any performance loss. The proposed method achieves 25-50% reduction in switching activity on the address bus of on-chip data memory. The present work is evaluated on five benchmark programs and is suitable for programs where array initialization time is more than computation time.

Keywords: Energy reduction, array initialization, address bus of on-chip L1-data cache, switching activity, loop unrolling, unrolling factor, loop unrolling with partial Gray code sequence, translation.

1 Introduction

Energy/power consumed by VLSI circuits is directly proportional to the switching activity. The address and data bus connecting memory and processor are highly capacitive which leads to the switching power dissipation when $0-to-1$ and $1-to-0$ bit transitions occur on the buses at high frequency. As the technology scales down to the deep-submicron region, the inter-wire capacitance (C_I) becomes significant compared to the wire-to-substrate capacitance (C_L). As C_I is the dominant capacitance in deep sub-micron era, it has two significant effects, large propagation delay due to opposite transitions on adjacent wires [1,2,3] and power dissipation associated with driving the on-chip buses [2]. The expression for average bus wire power consumption can be written as $P_{avg} = \frac{1}{2} \times C_{bus} \times V_{dd}^2 \times n_{trans} \times f$ where, C_{bus} is the bus capacitance, V_{dd} is the supply voltage, f is the frequency of operation. $n_{trans} = \frac{\sum_{t=0}^{N-1} HD(d^t, d^{t+1})}{N}$, which

M.S. Gaur et al. (Eds.): VDAT 2013, CCIS 382, pp. 83–93, 2013.

is the average number of bit transitions (switching activity) on the bus caused by transfer of N bit patterns. $HD(d^t, d^{t+1})$ is the Hamming Distance between two consecutive bit patterns d^t and d^{t+1}. The present work reduces n_{trans} on the address bus of data memory. This work focuses on systems that use Harvard architecture employing independent data and instruction address buses. This work considers the address bus between L1-data cache and the processor, where the L1-data cache is on-chip. The present work exploits the sequential access of adjacent memory locations, when a set of sequential locations (like an array) is initialized within a small loop, and introduces loop unrolling with partial Gray code sequence. This technique reduces on-chip bus switching activity on the address bus of data memory, thereby saving energy. The proposed work does not require any extra hardware for encoding and decoding address on the address bus. Programmers and/or compilers can exploit this idea to reduce switching activity on address bus of data memory, when they encounter loops which initialize array of considerable size. Loop unrolling reduces the number of loop manipulation instructions by loop unrolling factor (uf) saving both time and energy. Array initialization using loop unrolling with partial Gray code sequence can save more energy by reducing switching activity on the address bus of on-chip L1-data cache.

1.1 Related Work

Several hardware based approaches to reduce bus switching activity has been proposed earlier which require extra hardware in the form of encoders and decoders which consume more silicon space (increasing the design cost), power and degrades performance. Since, the present work is a software based technique, the hardware approaches are not discussed. In [4] the authors proposed the idea for instruction scheduling to reduce switching activity. The authors of [5] studied Gray code addressing to reduce switching activity on the instruction address bits and introduced an instruction scheduling technique called cold scheduling to reorder instruction sequence to reduce the switching activities. In [6,7] Lee et al proposed a greedy bipartite-matching instruction scheduling scheme to reduce switching activity in the instruction bus. In [8] Parikh et al proposed instruction scheduling algorithms considering the activity of switching from one instruction to another instruction as circuit-state effect (circuit-state cost or inter-instruction cost). In [9] the authors proposed an algorithm to reduce both schedule length by 11.5% and bus-switching activities by an average of 19.4% for applications with loops. In [11] the authors proposed an algorithm to reduce bus-switching activities by 52.2% and schedule length by an average of 20.1% while performing scheduling and allocation simultaneously. The method proposed in [10] modifies operation placement orders within VLIW instructions to reduce the switching activity between successive instruction fetches by 34% on an average.

2 Present Work

2.1 Basic Approach

```
#define n 1000000
int main()
{
   register int i;
   int a[n];
   for(i = 0; i < n; i++)
      {
         a[i] = 0;
      }
   return 0;
}
```

(a) Original Program

```
#define n 1000000
int main()
{
   register int i;
   int a[n];
   for(i = 0; i < n; i = i + 16)
      {
         a[i] = 0;
         a[i + 1] = 0;
         a[i + 2] = 0;
         a[i + 3] = 0;
         a[i + 4] = 0;
         a[i + 5] = 0;
         a[i + 6] = 0;
         a[i + 7] = 0;
         a[i + 8] = 0;
         a[i + 9] = 0;
         a[i + 10] = 0;
         a[i + 11] = 0;
         a[i + 12] = 0;
         a[i + 13] = 0;
         a[i + 14] = 0;
         a[i + 15] = 0;
      }
   return 0;
}
```

(b) Original Program with loop unrolling

```
#define n 1000000
int main()
{
   register int i;
   int a[n];
   for(i = 0; i < n; i = i + 16)
      {
         a[i] = 0;
         a[i + 1] = 0;
         a[i + 3] = 0;
         a[i + 2] = 0;
         a[i + 6] = 0;
         a[i + 7] = 0;
         a[i + 5] = 0;
         a[i + 4] = 0;
         a[i + 12] = 0;
         a[i + 13] = 0;
         a[i + 15] = 0;
         a[i + 14] = 0;
         a[i + 10] = 0;
         a[i + 11] = 0;
         a[i + 9] = 0;
         a[i + 8] = 0;
      }
   return 0;
}
```

(c) Original Program with unrolled loop having partial Gray code sequence

Fig. 1. Programs for array initialization

Figure 1(a) shows a program where all the $'n'$ elements of an array $'a'$ are initialized to an integer value $'0'$. This array initialization involves sequential access of $'n'$ memory locations, where the index variable $'i'$ is stored in a CPU register. The program in Fig. 1(b) is the loop unrolled version of the program in Fig. 1(a), where the loop unrolling factor (uf) is $'16'$. Loop unrolling reduces the number of loop manipulation instructions by a factor $'uf'$, saving time and energy. The program in Fig. 1(c) is a loop unrolled version of the program in Fig. 1(a), where the array a's memory address references within the body of the unrolled loop follows a Gray code sequence. So, the number of $'0 - to - 1'$ and $'1 - to - 0'$ bit transitions for array a's memory address references within the body of the unrolled loop, referred as intra-iteration switching, is restricted to $'uf - 1'$. The present work refers a simple unrolled loop as LU, and an unrolled loop with partial Gray code sequence as LUG. The number of $'0 - to - 1'$ and $'1 - to - 0'$ bit transitions (switchings) on the address bus of the data memory for LU and LUG are referred as S_{LU} and S_{LUG}, respectively. Figure 2 shows the generalized form of the original loop, LU and LUG, where an array $'a'$ is initialized with $'value'$. Where, *value* is either a constant or a variable stored in CPU register. Let, *data_type* be the type of data stored in the array $'a'$, *data_type* may be *char,*

```
register int i;
data_type a[n];
for(i = 0; i < n; i++)
  {
    a[i] = value;
  }
```
(a) Original Loop

```
register int i;
data_type a[n];
for(i = 0; i < n; i = i + uf)
  {
    a[i] = value;
    a[i + 1] = value;
    a[i + 2] = value;
    a[i + 3] = value;
    ...
    a[i + uf − 2] = value;
    a[i + uf − 1] = value;
  }
```
(b) *LU*

```
register int i;
data_type a[n];
for(i = 0; i < n; i = i + uf)
  {
    a[i] = value;
    a[i + 1] = value;
    a[i + 3] = value;
    a[i + 2] = value;
    ...
    a[i + uf/2 + 1] = value;
    a[i + uf/2] = value;
  }
```
(c) *LUG*

Fig. 2. Generalized form of the original loop, *LU* and *LUG*

int, long int, float, double, etc. Each element belonging to a *data_type* consumes $sizeof(data\_type)$ bytes of memory space, where $sizeof(data\_type)$ is a power of 2. Let, $base\_address(a)$ be the base address or the starting address of the array $'a'$. Let, b be obtained by shifting $base\_address(a)$ $log_2 sizeof(data\_type)$ bits right. The value of b signifies the portion (bits) of the address of array elements involved in switching activity. The $log_2 sizeof(data\_type)$ least significant bits (lsbs) of the address of the array elements are not involved in switching activity, and remain same for all elements of the array. Let, $'a'$ be an array of $n = 2^{\alpha}$ elements and $uf = 2^{\beta}$ be the loop unrolling factor, where $\alpha \geq \beta$ and α, β are natural numbers. The unrolled loop iterates, $\frac{n}{uf} = 2^{\alpha-\beta} = 2^{\gamma}$ times. The expressions for S_{LU} and S_{LUG} are derived in sections 2.2 and 2.3, respectively, considering $base\_address(a) = 0$ and n is divisible by uf.

2.2 Derivation of S_{LU}

S_{LU} is dependent on intra-iteration switching (S_{LU_intra}) and inter-iteration switching (S_{LU_inter}). So, S_{LU} can be written as shown in equation (1)

$$S_{LU} = S_{LU\_intra} + S_{LU\_inter} \tag{1}$$

S_{LU_intra} is the total number of $'0-to-1'$ and $'1-to-0'$ bit transitions on the address bus of the data memory, which takes place due to the memory address references of the elements of array $'a'$, i.e. $a[i], a[i+1], ..., a[i+uf-2], a[i+uf-1]$ (in $(\frac{i}{uf}+1)^{th}$ iteration, where, $0 \leq i < n$ and i is a multiple of uf), within the body of the unrolled loop. For each iteration β lsbs of b follows the sequence $0, 1, 2, 3, \cdots, uf-2, uf-1$. So, S_{LU_intra} can be written as shown in equation (2)

$$S_{LU\_intra} = \frac{n}{uf} \times t_{\beta} \tag{2}$$

where t_{β} is the number of intra-iteration switchings per iteration. t_{β} can be expressed by the recurrence relation as shown in equation (3)

$$t_{\beta} = 2 \times t_{\beta-1} + \beta, \ for\ \beta > 1\ , and\ t_1 = 1 \tag{3}$$

The solution of the recurrence relation in equation (3) is shown in equation (4)

$$t_\beta = 2^{\beta+1} - \beta - 2 \tag{4}$$

Substituting equation (4) in equation (2) S_{LU_intra} is obtained in equation (5)

$$S_{LU\_intra} = \frac{n}{uf} \times (2^{\beta+1} - \beta - 2) = \frac{n}{uf} \times (2 \times uf - log_2 uf - 2) \tag{5}$$

Let, i_{curr} and i_{next} be the values of i in the current and next iterations,

			Inter-iteration bit transition for LU			Inter-iteration bit transition for LUG		
$Iteration$	i_{curr}	i_{next}	$b_{a[i_{curr}+uf-1]}$	$b_{a[i_{next}]}$	$\sharp Switching$	$b_{a[i_{curr}+\frac{uf}{2}]}$	$b_{a[i_{next}]}$	$\sharp Switching$
1	0	16	0000 1111	0001 0000	5	0000 1000	0001 0000	2
2	16	32	0001 1111	0010 0000	6	0001 1000	0010 0000	3
3	32	48	0010 1111	0011 0000	5	0010 1000	0011 0000	2
4	48	64	0011 1111	0100 0000	7	0011 1000	0100 0000	4
5	64	80	0100 1111	0101 0000	5	0100 1000	0101 0000	2
6	80	96	0101 1111	0110 0000	6	0101 1000	0110 0000	3
7	96	112	0110 1111	0111 0000	5	0110 1000	0111 0000	2
8	112	128	0111 1111	1000 0000	8	0111 1000	1000 0000	5

Fig. 3. Inter-iteration switching on the address bus of data memory for first eight iterations of LU and LUG in Fig. 2 (b) and (c), respectively, where, uf=16 and $base\_address(a) = 0$

$(Iteration(\eta), \sharp Switching)$
$(1, \beta+1)$
$(2, \beta+2), (3, \beta+1)$
$(4, \beta+3), (5, \beta+1), (6, \beta+2), (7, \beta+1)$
$(8, \beta+4), (9, \beta+1), (10, \beta+2), (11, \beta+1), (12, \beta+3), (13, \beta+1), (14, \beta+2), (15, \beta+1)$
$\cdots$
$(2^{\gamma-2}, \beta+\gamma-2+1), (2^{\gamma-2}+1, \beta+1), \cdots, (2^{\gamma-1}-2, \beta+2), (2^{\gamma-1}-1, \beta+1)$
$(2^{\gamma-1}, \beta+\gamma-1+1), (2^{\gamma-1}+1, \beta+1), \cdots, (2^{\gamma}-2, \beta+2), (2^{\gamma}-1, \beta+1)$

(a) Inter-iteration switching after each iteration from iteration 1 to iteration $2^\gamma - 1$

Iteration Range	*Total* $\sharp Switching$	
1	$\sigma_1 = \beta + 1$	$= 2^0 \times \beta + 2^1 - 1$
2 *to* 3	$\sigma_2 = \beta + 2 + \beta + 1 = 2 \times \sigma_1 + 1$	$= 2^1 \times \beta + 2^2 - 1$
4 *to* 7	$\sigma_3 = \beta + 3 + \beta + 1 + \beta + 2 + \beta + 1 = 2 \times \sigma_2 + 1$	$= 2^2 \times \beta + 2^3 - 1$
8 *to* 15	$\sigma_4 = 2 \times \sigma_3 + 1$	$= 2^3 \times \beta + 2^4 - 1$
$\cdots$	$\cdots$	$\cdots$
$2^{\gamma-2}$ *to* $2^{\gamma-1}-1$	$\sigma_{\gamma-1} = 2 \times \sigma_{\gamma-2} + 1$	$= 2^{\gamma-2} \times \beta + 2^{\gamma-1} - 1$
$2^{\gamma-1}$ *to* $2^\gamma - 1$	$\sigma_\gamma = 2 \times \sigma_{\gamma-1} + 1$	$= 2^{\gamma-1} \times \beta + 2^\gamma - 1$

(b) Total inter-iteration switching in mentioned iteration ranges

Fig. 4. Inter-iteration switching on the address bus of data memory for LU in Fig. 2(b), where, $base\_address(a) = 0$

respectively. Where, $i_{next} = i_{curr}+uf$. S_{LU_inter} is the total number of $'0-to-1'$ and $'1-to-0'$ bit transitions on the address bus of the data memory, which takes place due to the last memory address reference (of $a[i_{curr}+uf-1]$) in the $(\frac{i}{uf}+1)^{th}$ iteration (current iteration), and the first memory address reference (of $a[i_{next}]$) in the $(\frac{i}{uf}+2)^{th}$ iteration (next iteration). Let, $b_{a[i_{curr}+uf-1]}$, $b_{a[i_{next}]}$, $b_{a[i_{curr}+\frac{uf}{2}]}$ be the portion (bits) of the address of array elements $a[i_{curr}+uf-1]$, $a[i_{next}]$, $a[i_{curr}+\frac{uf}{2}]$, rescpectively, which are involved in inter-iteration switching activity. Figure 3 shows the inter-iteration switching on the address bus of data memory for first eight iterations of LU and LUG in Fig. 2 (b) and (c), respectively, where, uf=16 and $base\_address(a) = 0$. S_{LU_inter} can be obtained from Fig. 4. Figure 4(a) shows the inter-iteration switchings for each iteration (from iteration 1 to iteration $2^{\gamma}-1$). In Fig. 4(b) the total inter-iteration switching in mentioned iteration ranges forms a series whose summation forms the S_{LU_inter}. This can be written as, $S_{LU\_inter} = \beta \times (2^0 + 2^1 + 2^2 + \cdots + 2^{\gamma-2} + 2^{\gamma-1}) + (2^1 + 2^2 + 2^3 + \cdots + 2^{\gamma-1} + 2^{\gamma}) - \gamma = \beta \times (2^{\gamma} - 1) + (2^{\gamma+1} - 2) - \gamma$. The final expression for S_{LU_inter} can be written as shown in equation (6).

$$S_{LU\_inter} = (log_2 uf + 2) \times \left(\frac{n}{uf} - 1\right) - log_2 \frac{n}{uf} \tag{6}$$

Substituting equations (2) and (6) in equation (1), the expression of S_{LU} is obtained as $S_{LU} = \frac{n}{uf} \times (2 \times uf - log_2 uf - 2) + (log_2 uf + 2) \times \left(\frac{n}{uf} - 1\right) - log_2 \frac{n}{uf}$.

2.3 Derivation of S_{LUG}

S_{LUG} is dependent on intra-iteration switching (S_{LUG_intra}) and inter-iteration switching (S_{LUG_inter}). So, S_{LUG} can be written as shown in equation (7).

$$S_{LUG} = S_{LUG\_intra} + S_{LUG\_inter} \tag{7}$$

S_{LUG_intra} is the total number of $'0-to-1'$ and $'1-to-0'$ bit transitions on the address bus of the data memory, which takes place due to the memory address references of the elements of array $'a'$, i.e. $a[i], a[i+1], a[i+3], a[i+2], ..., a[i+\frac{uf}{2}+1], a[i+\frac{uf}{2}]$ (in $(\frac{i}{uf}+1)^{th}$ iteration, where, $0 \leq i < n$ and i is a multiple of uf), within the body of the unrolled loop. For each iteration β lsbs of b follows the Gray code sequence $0, 1, 3, 2, \cdots, \frac{uf}{2}+1, \frac{uf}{2}$. So, S_{LUG_intra} can be written as shown in equation (8)

$$S_{LUG\_intra} = \frac{n}{uf} \times (uf - 1) \tag{8}$$

where, $(uf-1)$ is the intra-iteration switching per iteration due to a Gray code sequence of address references within the body of the unrolled loop. S_{LUG_inter} is the total number of $'0-to-1'$ and $'1-to-0'$ bit transitions on the address bus of the data memory, which takes place due to the last memory address reference (of $a[i_{curr}+\frac{uf}{2}]$) in the $(\frac{i}{uf}+1)^{th}$ iteration (current iteration), and

$(Iteration(\eta), \sharp Switching)$
$(1, 2)$
$(2, 3), (3, 2)$
$(4, 4), (5, 2), (6, 3), (7, 2)$
$(8, 5), (9, 2), (10, 3), (11, 2), (12, 4), (13, 2), (14, 3), (15, 2)$
$\cdots$
$(2^{\gamma-2}, \gamma - 2 + 2), (2^{\gamma-2} + 1, 2), \cdots, (2^{\gamma-1} - 2, 3), (2^{\gamma-1} - 1, 2)$
$(2^{\gamma-1}, \gamma - 1 + 2), (2^{\gamma-1} + 1, 2), \cdots, (2^{\gamma} - 2, 3), (2^{\gamma} - 1, 2)$

(a) Inter-iteration switching after each iteration from iteration 1 to iteration $2^{\gamma} - 1$

Iteration Range	*Total* $\sharp Switching$	
1	$s_1 = 2$	$= 2$
2 *to* 3	$s_2 = s_1 + 3 \times 2^0 = 2 + 3$	$= 5$
4 *to* 7	$s_3 = s_2 + 3 \times 2^1 = 5 + 6$	$= 11$
8 *to* 15	$s_4 = s_3 + 3 \times 2^2 = 11 + 12$	$= 23$
$\cdots$	$\cdots$ $\cdots$	$\cdots$
$2^{\gamma-2}$ *to* $2^{\gamma-1} - 1$	$s_{\gamma-1} = s_{\gamma-2} + 3 \times 2^{\gamma-3}$	$= 3 \times 2^{\gamma-2} - 1$
$2^{\gamma-1}$ *to* $2^{\gamma} - 1$	$s_{\gamma} = s_{\gamma-1} + 3 \times 2^{\gamma-2}$	$= 3 \times 2^{\gamma-1} - 1$

(b) Total inter-iteration switching in mentioned iteration ranges

Fig. 5. Inter-iteration switching on the address bus of data memory for *LUG* in Fig. 2(c), where, $base\_address(a) = 0$

the first memory address reference (of $a[i_{next}]$) in the $(\frac{i}{uf} + 2)^{th}$ iteration (next iteration). S_{LUG_inter} can be obtained from Fig. 5. Figure 5(a) shows the inter-iteration switchings for each iteration (from iteration 1 to iteration $2^{\gamma} - 1$). In Fig. 5(b) the total inter-iteration switching in the mentioned iteration ranges forms a series whose γ^{th} term is obtained from the recurrence relation as shown in equation (9)

$$s_{\gamma} = s_{\gamma-1} + 3 \times 2^{\gamma-2},\ for\ \gamma > 1\ , and\ s_1 = 2 \tag{9}$$

The solution of the recurrence relation in equation (9) is shown in equation (10)

$$s_{\gamma} = 3 \times 2^{\gamma-1} - 1. \tag{10}$$

The S_{LUG_inter} can be obtained from the summation of the series obtained in Fig. 5(b). This can be written as $S_{LUG\_inter} = 3 \times (2^0 + 2^1 + 2^2 + \cdots + 2^{\gamma-1}) - \gamma = 3 \times (2^{\gamma} - 1) - \gamma$. The final expression for S_{LUG_inter} can be written as shown in equation (11).

$$S_{LUG\_inter} = 3 \times \left(\frac{n}{uf} - 1\right) - log_2 \frac{n}{uf} \tag{11}$$

Substituting equations (8) and (11) in equation (7), the expression of S_{LUG} is obtained as $S_{LUG} = \frac{n}{uf} \times (uf - 1) + 3 \times \left(\frac{n}{uf} - 1\right) - log_2 \frac{n}{uf}$. Table 1 compares S_{LU} and S_{LUG} considering $n = 2^{10}$. For any n, $n \geq uf$ reduction in S_{LUG} is minimum (25%) when $uf = 2^2$ and maximum (50%) when a loop is totally unrolled ($n = uf$). But, total loop unrolling is impractical due to hardware limitations of the system.

Table 1. Comparision between S_{LU} and S_{LUG}

uf	$\frac{n}{uf}$	S_{LU}	S_{LUG}	Gain(%)
2^2	2^8	2036	1525	25.09
2^3	2^7	2036	1270	37.62
2^4	2^6	2036	1143	43.86
2^5	2^5	2036	1080	46.95
2^6	2^4	2036	1049	48.47
2^{10}	1	2036	1023	49.75

2.4 Translation to *LUG*

In section 2.2 and 2.3 the expressions for S_{LU} and S_{LUG} have been derived, respectively, assuming $'0'$ as the $base\_address(a)$. But, in reality when the program in Fig. 2 will execute, the $base\_address(a)$ may not be $'0'$. The $base\_address(a)$ may vary for different executions because it depends on system's memory manager that allocates space for array a at runtime. So, it is not possible for a compiler to predict the actual base address $base\_address(a)$. The present work considers both b and n are divisible by uf. When the array a is allocated at compile time the compiler does not know the actual base address $base\_address(a)$, but knows the relocatable base address of the array, which is an offset address. The compiler finds a relocatable base address such that the logic values corresponding to the intra-iteration switching bits are 0, which implies that b is divisible by uf. If the array a is allocated in runtime then the dynamic memory allocation subroutine can be directed to find a base address such that b divisible by uf.

3 Experimental Results

The present work is evaluated on five benchmark programs on XEEMU simulator [12]. XEEMU is a power-performance simulator which simulates Intel's XScale processor. Each benchmark program (as described in Table 3) have array initialization loops (as in Fig. 2(a)) which are translated to LUG (as in Fig. 2(c)). Table 2 shows the reduction in switching activity, execution time, energy consumption by the translated loop (E_{TL}) and energy drawn by the address bus of dl1-cache ($E_{dl1-addr\_bus}$) for the programs in Fig. 1. Since $E_{dl1-addr\_bus}$ is directly propotional to S_{LUG} they experience equal amount of reduction. Table 4 shows the time taken and energy consumed by the benchmark programs having the original loop (Org), LU, and LUG. $SCount$ and $CSort$ with LUG achieves more gain in total energy (E_{Tot}) because their array initialization time (T_{init}) is much longer than computation time (T_{comp}). KS, TI and DFS with LUG have less gain in E_{Tot} because their T_{init} is much lesser than T_{comp}. Thus, LUG is more applicable for the programs having $T_{init} \geq T_{comp}$.

Table 2. Comparision of Switching activity, Time and Energy consumption of the programs in Fig. 1

Program	Metric	Value	Metric	LU's Gain wrt Org (%)	LUG's Gain wrt Org (%)	LUG's Gain wrt LU (%)
Original	♯Switching	1999986	♯Switching	-	-	-
(Org)	Time(ms)	36.90	Time	-	-	-
	E_{TL}(mJ)	28.10	E_{TL}	-	-	-
	$E_{dl1-addr\_bus}$(mJ)	5.69	$E_{dl1-addr\_bus}$	-	-	-
Loop Unrolling	♯Switching	1999986	♯Switching	0.0	-	-
(LU)	Time(ms)	27.60	Time	25.20	-	-
	E_{TL}(mJ)	19.90	E_{TL}	29.18	-	-
	$E_{dl1-addr\_bus}$(mJ)	5.69	$E_{dl1-addr\_bus}$	0.0	-	-
Loop Unrolling	♯Switching	1124989	♯Switching	-	43.75	43.75
with partial	Time(ms)	27.60	Time	-	25.20	0.0
Gray code	E_{TL}(mJ)	16.70	E_{TL}	-	40.56	16.08
sequence (LUG)	$E_{dl1-addr\_bus}$(mJ)	3.2	$E_{dl1-addr\_bus}$	-	43.75	43.75

Table 3. Benchmark Programs

Benchmark	Description	T_{init} T_{comp}
Symbol Count ($SCount$)	Finds frequency of symbols in a string of size $\omega = 10^3$, each symbol belongs to a set of $n = 2^{20}$ symbols. $uf = 2^4$	$O(n)$ $O(\omega)$
Counting Sort ($CSort$)	A linear time sort on an array of $m = 10^3$ integers, each integer lies between 0 and $n = 2^{20}$. $uf = 2^4$	$O(n)$ $O(m)$
0-1 Knapsack (KS)	Given costs and weights of $r = 3$ types of items, fill a knapsack of capacity $n = 10^6$ such that the sum of cost of the elements to fill it is maximum. $uf = 2^4$	$O(n)$ $O(r \times n)$
Treasure Island (TI)	Given an $n \times n$ grid, each coordinate of the grid has a cost, staring from the lower-left corner $(1, 1)$ one has to reach upper-right corner (n, n), either by moving upward or rightward in each step, such that the cost of the path traversed is maximum. $n = 2^9, uf = 2^4$	$O(n)$ $O(n^2)$
Depth First Search (DFS)	Depth First Traversal of a randomly generated graph with $n = 2^{10}$ vertices and $e = O(n^2)$ edges. $uf = 2^4$	$O(n)$ $O(max\{n, e\})$

4 Conclusion

The present work introduces a software based approach to reduce energy consumed on the address bus of the data memory. This is done by reducing switching activity on the address bus of the data memory, with the help of LUG. Translation of a loop with array initialization to LUG is introduced. The expressions for switching activity on the bus for LU and LUG are derived. The proposed translation technique finds a relocatable base address of the array so that the partial Gray code sequence is maintained, without any energy-performance overhead and achieves a considerable amount of energy reduction without any performance loss. The proposed method achieves 25-50% reduction in switching activity on the address bus of on-chip data memory. The proposed work is evaluated on five benchmark programs. LUG is more applicable for the programs having

Table 4. Comparision of Time and Energy consumption of the benchmark programs

Benchmark	Metric	*Org*	*LU*	*LUG*	Metric	*LU*'s Gain wrt *Org* (%)	*LUG*'s Gain wrt *Org* (%)	*LUG*'s Gain wrt *LU* (%)
SCount	*Time* (ms)	40.6	29.3	29.3	*Time*	27.83	27.83	0.0
	E_{Tot} (mJ)	30.7	21.1	17.7	E_{Tot}	31.27	42.34	16.11
	E_{TL} (mJ)	30.5	20.9	17.5	E_{TL}	31.47	42.62	16.26
CSort	*Time* (ms)	160.7	142.2	142.2	*Time*	11.51	11.51	0.0
	E_{Tot} (mJ)	116.7	102.2	99.7	E_{Tot}	12.42	14.46	2.44
	E_{TL} (mJ)	36.6	22.1	19.5	E_{TL}	39.61	46.72	11.76
KS	*Time* (ms)	766.1	751.1	751.1	*Time*	1.95	1.95	0.0
	E_{Tot} (mJ)	576.9	564.1	559.7	E_{Tot}	2.21	2.98	0.78
	E_{TL} (mJ)	52.0	39.3	35.0	E_{TL}	24.42	32.69	10.94
TI	*Time* (ms)	124.5	120.1	120.1	*Time*	3.53	3.53	0.0
	E_{Tot} (mJ)	91.2	88.0	87.4	E_{Tot}	3.5	4.16	0.68
	E_{TL} (mJ)	15.4	12.3	12.1	E_{TL}	20.12	21.42	1.62
DFS	*Time* (ms)	246.5	246.4	246.4	*Time*	0.04	0.04	0.0
	E_{Tot} (mJ)	180.23	180.212	180.211	E_{Tot}	0.01	0.01	0.0
	E_{TL} (mJ)	0.0583	0.0499	0.0466	E_{TL}	14.40	20.06	6.61

$T_{init} \geq T_{comp}$. The future work will investigate on other software techniques to reduce switching activity on address, data and control bus of instruction and data memory.

References

1. Caignet, F., Delmas-Bendhia, S., Sicard, E.: The Challenge of Signal Integrity in Deep-submicrometer CMOS Technology. Proceedings of the IEEE 89(4), 556–573
2. Sylvester, D., Hu, C.: Analytical Modeling and Characterization of Deepsubmicrometer Interconnect. Proceedings of the IEEE 89(5), 634–664
3. Victor, B., Keutzer, K.: Bus Encoding to Prevent Crosstalk Delay. In: Proceedings of ICCAD, pp. 57–63 (2001)
4. Tiwari, V., Malik, S., Wolfe, A.: Compilation Techniques for Low Energy: An Overview. In: Proceedings of Symposium on Low-Power Electronics, San Diego, CA (October 1994)
5. Su, C.-L., Tsui, C.-Y., Despain, A.M.: Reducing Power Consumption at Control Path of High Performance Microprocessors. IEEE Design and Test of Computers (December 1994)
6. Lee, C., Lee, J.K., Hwang, T.T.: Compiler Optimization on Instruction Scheduling for Low Power. In: Proceedings of 13th International Symposium on System Synthesis, pp. 55–60 (2000)
7. Lee, C., Lee, J.K., Hwang, T.T., Tsai, S.: Compiler Optimization on VLIW Instruction Scheduling for Low Power. ACM Transactions on Design Automation of Electronic Systems (TODAES) 8(2), 252–268
8. Parikh, A., Kim, S., Kandemir, M., Vijaykrishnan, N., Irwin, M.J.: Instruction Scheduling for Low Power. Journal of VLSI Signal Processing 37(1), 129–149

9. Shao, Z., Xiao, B., Xue, C., Zhuge, Q., Sha, E.H.M.: Loop scheduling with timing and switching-activity minimization for VLIW DSP. ACM Transactions on Design Automation of Electronic Systems (TODAES) 11(1), 165–185
10. Shin, D., Kim, J., Chang, N.: An Operation Rearrangement Technique for Low-Power VLIW Instruction Fetch. In: Proceedings of DATE, p. 809 (2001)
11. Shao, Z., Xiao, B., Xue, C., Sha, E.H.M.: Algorithms and analysis of scheduling for loops with minimum switching. Int. J. Computational Science and Engineering 2(1/2)
12. Herczeg, Z., Kiss, Á., Schmidt, D., Wehn, N., Gyimóthy, T.: XEEMU: An Improved XScale Power Simulator. In: Azémard, N., Svensson, L. (eds.) PATMOS 2007. LNCS, vol. 4644, pp. 300–309. Springer, Heidelberg (2007)