Multi-Parametric Improvements for Embedded Systems using Code-Placement and Address Bus Coding

Sri Parameswaran
The University of New South Wales
Kensington, NSW
sridevan@cse.unsw.edu.au

Jörg Henkel
NEC USA Inc.
Princeton, NJ
henkel@nec-lab.com

Haris Lekastas
NEC USA Inc.
Princeton, NJ
lekatsas@nec-lab.com

Abstract

Code placement techniques for instruction code have shown to increase an SOCs performance mostly due to the increased cache hit ratios and as such those techniques can be a major optimization strategy for embedded systems. Little has been investigated on the interdependencies between code placement techniques and interconnect traffic (e.g. bus traffic) and optimization techniques combining both. In this paper we show as the first approach of its kind that a carefully designed known code placement strategy combined and adapted to a known interconnect encoding scheme does not only lead to a performance increase but it does also lead to a significant reduction of interconnect-related energy consumption. This becomes especially interesting since future SOC bus systems (or more general: "networks on a chip") are predicted to be a dominant energy consumer of an SOC. We show that a high-level optimization strategy like code placement and a lower-level optimization strategy like interconnect encoding are NOT orthogonal. Specifically, we report cache miss reduction ratios of 32% in average combined with bus related energy savings of 50.4% in average (with a maximum of up to 95.7%) by means of our combined optimization strategy. The results have been verified by means of diverse real-world SOC applications.

1 Introduction

The advent of silicon technologies will lead to System on Chips (SOC) that will reach one billion transistor designs within the next few years. A major reason preventing the integration of several hundred million transistors on a single chip (indeed, this would be possible through today's mainstream 0.13μ silicon technologies and wafer technologies) is the energy dissipation problem. The per- square-mm and per-time generated heat energy can hardly be carried off-chip without substantial (i.e. costly) effort and thus prevent those designs finding their way into mainstream consumer products. An additional constraint is implied by mobile computing/communication/entertainment devices which draw their current from capacity-limited batteries. The problem has been addressed at various levels of abstraction starting from new silicon technologies, through gate-level, RT-level, architectural-level and eventually to system-level approaches. However, it can be observed that many proposed approaches are orthogonal (at least as far as many methods within a certain abstraction-level are concerned). In other words, power saving/reducing methods are often designed, without having complementary methods in mind and thus complicating or even preventing the effective implementation of one or more other power saving methods. There is currently promising evidence by new research at the system-level that the tuning of parameters of various system parts can lead to substantial power savings. Work in this direction is typically carried out at a high-level of abstraction and thus cannot capture subtle architectural characteristics.

Benchmark	Before	After
Compress	*527685559*	*535616238*
Mpeg	*420232997*	*401858247*
Rstest	*415164716*	*410877294*

Table 1: Average number of transitions before and after code placement for three benchmarks

In this paper we present the first approach that combines, adapts and optimizes two methods for significantly reducing the power consumption of communication dominated systems. We present a code placement strategy that, among others, reduces the communication overhead between CPU and the memory hierarchy. Code placement reduces the power by reducing memory accesses and by reducing the number of wait cycles in the processor. However, code placement does not reduce the average number of bus transitions (see Table 1). Thus a bus encoding mechanism is needed to amplify the power savings in the processor and memory.

This encoding leads to ultra-low bus power consumption combined with an effectively increased bus bandwidth that yields a higher system performance as well (we will later on explain why bus power and interconnect power in general will be a major contributor to the system power consumption in future silicon technologies). We achieve interconnect energy savings of 50.4% compared to the case where a single method is applied and thus report energy savings that top all other known stand-alone energy saving techniques addressing SOC interconnect.

The paper is structured as follows: the next Section 1.1 gives an introduction to the related work in both code placement and bus encoding. Our techniques for code placement and bus encoding are then introduced in detail in Sections 2 and 3, respectively. The validation environment is given in Section 4 and the experimental part follows in Section 5 and finally conclusions arrived at in Section 6.

1.1 Related work

In the following we report on the most relevant work on the two areas of code placement techniques and bus encoding techniques. Both areas are crucial to our approach which is the first one to combine, adapt and optimize two previously separately treated low energy methods to achieve an ultra-low energy dissipating interconnect for SOC designs.

Several articles have appeared in recent literature about reducing cache misses by reorganizing data or instructions in the cache. The work on cache misses has predominantly concentrated on data cache optimisations [4][13][12]. Hwu et al, in [1], McFarling in [2] and [3], Chow in [7], Tomiyama and Yasuura in [11], Kirovski et al in [10], Kirk in [8] and [9], Li and Wolf in [5], and Parameswaran in [15] have given various methods to reduce instruction cache misses in microprocessor based systems. All of these systems are at the function level.

In [27], algorithms were presented in order to reduce the total cache misses at the assembler block level.

In recent work it has been recognized that inter-wire capacitances increasingly contribute to the power consumed on a bus

0-7803-7659-5/03/$17.00 ©2003 IEEE.

system. Various approaches have tried to address this problem and bus-related power consumption in general.

Initial work on bus encoding has been conducted by Stan/Burleson [16]. The basic idea is to transfer an inverted word through the bus whenever it can reduce the Hamming distance between a word and its previous word. Later in [17], they introduced Limited-Weight Codes (LWC) for low power encoding and provided optimal statistic performance for random data.

The above schemes belong to the class of space-time redundant encoding, where bus sizes are augmented. While the above encodings were developed for random input, researchers started to address data source properties. Panda/Dutt [20] developed a scheme to map arrays in memory for reducing energy on address bus.

Exploiting the characteristic that consecutive memory accesses tend to have a consecutive addresses, Mehta et al. introduced Gray code for address bus [21]. To further reduce the energy on an address bus, Benini et al. proposed a prediction scheme taking high regularity of data on address buses into consideration [23]. E. Musoll et al. [18] proposed the WZE (Working-Zone Encoding) scheme to exploit locality of memory reference.

Meanwhile, theoretical approaches for bus encoding were developed. In [25], Ramprasad et al. started to use a general communication model to analyze bus-encoding schemes giving lower bounds on average signal transition activities. The work introduced so far is focussed on reducing transition activities on a bus based on the assumption that the inter-wire parasitic capacitances are negligible. However, with the advent of deep sub-micron technology, inter-wire parasitic capacitances become a major issue. Sotiriadis/Chandrakasan stated in [22] that simply minimizing transition might not lead to optimal power reduction. They developed a model to incorporate the inter-wire capacitances of a bus and search the code space to find the best codes for inputs based on their bus power model. In the work conducted by Kim et al. [24], two new schemes have been introduced for low power buses. Schemes addressing data properties for deep sub-micron technologies are proposed by Henkel/Lekatsas [19] through re-arranging bus lines and then applying local bus invert. Macchiarulo et al. [26] have shown that the layout of an address bus can be arranged for low power consumption.

1.2 Motivation and Focus

Deep sub-micron silicon designs of 0.10u and beyond lead to a shift in optimization strategies for SOCs for several reasons:

1. due to the small feature sizes, inter-wire capacitances of, for instance, bus lines become dominating (compared to intrinsic bus line capacitances). Hence, the relative share of energy consumption of the SOC buses compared to all other components will increase by up to around 30% in future designs.

2. architectural optimizations like improved code placement techniques do not only increase the performance (as shown in [27], they may also dramatically change the extension of the traffic on the buses involved. For example, an improved code placement technique might lead to a higher cache hit ratio and thus a) reduce the number of related bus transactions between the cache and the main memory b) shift the bus traffic to the processor-to-cache bus instead and c) decrease the amount of total related bus transactions.

This leads to interdependencies that were previously not considered to be orthogonal. Now, with deep sub-micron designs emerging, code placement (and other, higher level optimization techniques for SOCs) has a direct impact on the bus-related energy consumption and as such influences the energy consumption of the whole SOC. Previously considered relevant was the energy savings that come with the reduced execution time (improved performance) of an application with a more efficient code placement. Now, it does matter in which way a code placement technique implicitly shifts transactions from one bus to another and it does matter how efficiently these buses make use of bus encoding schemes to reduce the energy consumption. In this context it also does matter how long (physically) those buses are inducing a direct relationship between a code placement technique and physical parameters.

This work focuses on these relationships and presents, as the first approach, a quantification and optimization of interdependencies between:

a) code placement on the one side and processor-to-cache and cache-to-main_memory bus lengths on the other side

b) code placement and energy-saving bus encoding schemes

We will show that these interdependencies and their exploitation lead to a reduction of 50.4% in average (maximum of 95.7%) of the address bus energy consumption of a whole SOC.

1.3 Assumptions

The following assumptions hold for the approach introduced later in this section:

1) The systems considered are single microprocessor systems, with memory and instruction cache which is configured for a single application. This is quite common in embedded systems.

2) The size of code block placed is no bigger than the size of the cache. This assumption is quite valid in embedded systems where the basic blocks are usually small enough to fit into small cache sizes. If the task is too large for the cache it is possible to break up the task into smaller granules such that each granule will fit into the cache.

3) Only Level-1 caches are available for use. Once again in an embedded system, where frequently there is no cache at all, it is unlikely that more than a single level of cache is going to be available for use.

4) The caches are direct mapped. High-speed systems frequently use direct mapped systems in order to speed up the system as much as possible. This assumption makes it easier to analyse due to the deterministic mapping to cache from memory.

5) The problem is sufficiently large so that the total size of the instructions (in bytes) are several times larger the size of cache. This is a reasonable assumption in a realistic system.

2 Allocation of Assembly Level Basic Blocks in Cache and Memory

This section details the code placement methodology. Note here that the methodology looks at the code at the assembler level. The basic blocks here are blocks of assembler instructions which are executed together. This methodology contains an algorithm with two parts. The first part places basic blocks in the cache so that basic blocks with high frequency are swapped out as little as possible. The second part of the algorithm takes the placed basic blocks and maps them into main memory. This algorithm is performed as a preprocessing step, taking the application's original instructions in memory and re-mapping them to different locations.

In order to re-map instructions, it was necessary to identify basic blocks. After the identification of basic blocks, we had to identify which blocks executed consecutively. We identified them by running the application through an instruction set sim-

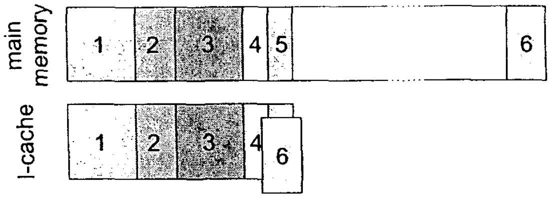

Figure 1: The memory allocation example

ulator and finding blocks of instructions which were always executed together. The number of basic blocks within applications under consideration varied from 100 - 900. A more comprehensive study of the approach is given in [27].

2.1 Part 1: Cache Allocation

The methodology used for ordering basic blocks in the cache is as follows. All the loops containing a particular basic block are grouped into a single super loop. Thus a loop will be only a member of one super loop. Each super loop's execution frequency (f_{sl}) is defined as the addition of all the execution frequencies of the component loops. The super loops are ordered in descending order of frequency. The ordered super loop list is given as $sl_1, sl_2, ...sl_p$.

For super loop sl_1, the basic blocks within it are taken in order (from highest to lowest frequency of execution of basic blocks - f_b) and these basic blocks (only whole basic blocks are allowed) are allocated to the cache from the lowest address to the highest until the cache is completely filled or there are no remaining basic blocks within that super loop. Once this step is finished, and if there are any remaining basic blocks, we find the largest basic block from the remaining basic blocks of sl_1. This large basic block is allocated to the bottom of the cache, say with starting address A_{ls}, and ending at the end of the cache. After this we take the next largest basic block and allocate its starting address in the cache to A_{ls}. The ending address will be less than the final address of the cache. Then if another unallocated basic block can be found which can go into the space (below the basic block we just allocated, and above the last cache address), we allocate that basic block into the available space. We keep doing this until we reach the end of the cache. We take the next largest unallocated basic block, and start it at address A_{ls} and we repeat the process until all basic blocks are allocated. This is then repeated for all the other super loops in the ordered list.

2.2 Part 2: Memory Allocation

The memory allocation part of the algorithm takes the already placed basic blocks in the cache and directly maps them to the memory. Figure 1 shows an example of how the basic blocks are mapped to memory from the cache. In this figure blocks 1 to 5 are mapped directly on to the memory, but the block 6 is mapped to some memory locations further away, such that the mapped block will end up in the desired position in the cache. Thus if a basic block is mapped to the location from t_x to e_x in cache of the processor, then the basic block can be placed in memory in any one of the address ranges from addresses $t_x + i * S$ to $e_x + i * S$, where i is a positive integer and S is the size of the cache. However, since basic blocks in the cache will wrap around the cache, an offset Z_r, can be added to each basic block allocated from super loop sl_r, and the basic block can be placed from memory location $t_x + Z_r + i * S$ to memory location $e_x + Z_r + i * S$. This introduction of the offset allows the reduction in size of the total memory needed for the system.

2.2.1 Algorithm

```
Part 1. ordering basic blocks in cache
    For Each super loop in ordered list {
        Until Cache is filled {
            Allocate basic blocks to the cache in descending order of
            frequency $f_b$ until no more blocks can be allocated
        }
        Reorder Unallocated basic blocks in order of size and place
        in list $BB_u$ ($BB_u = bb_{u1}, bb_{u2}, bb_{u3}...bb_{uy}$, where y is the
        number of unallocated basic blocks for that super loop)
        Allocate $bb_{u1}$ from address $A_{ls}$ to end of cache (where $A_{ls}$ = S
        - sizeof($bb_{u1}$))
        Remove $bb_{u1}$ from $BB_u$
        Repeat until all basic blocks are allocated {
            Find the next largest unallocated Task $bb_{up}$ from the list
            $BB_u$
            Allocate $bb_{up}$ from address $A_{ls}$ to $A_{le}$ where $A_{le} =
            A_{ls} + sizeof(bb_{up})$
            Mark $bb_{up}$ from list $BB_u$ as allocated
            Move along the list $BB_u$ and place as many basic blocks
            as possible between $A_{le}$ and $S$
            Mark placed basic blocks as allocated }}
```

Part 2. Memory Allocation Let us assume that the super loops used in the system are $sl_1, sl_2, sl_3...sl_k$. The associated cache is C. For each super loop sl_r, the basic blocks to be executed in that super loop are as follows: $bb_1, bb_2, bb_3......bb_m$ where m is the number of basic blocks.

```
    Reorder super loops from largest sum to smallest sum of total basic
    block size and name them $sl_a, sl_b, sl_c...$
    For all basic blocks ordered in the cache allocation order in $sl_a$ do {
        $i = 0$
        While basic block is not allocated do {
            If memory locations $t_x + i * S$ to $e_x + i * S$ is free then
                Map basic block to address $t_x + i * S$ to $e_x + i * S$
            Else
                $i + +$ } }
    For all basic blocks ordered in descending order of size in the next super
    loop until the end of the super loop list do {
        Allocate largest basic block in the first available contiguous
        memory block ($M_x$ to $M_y$), which will hold the basic block
        Calculate $Z_r = (M_x mod S) - t_x$, where $t_x$ is the address
        in which the basic block being allocated starts in the cache at
        address 0 and $sl_r$ is the present super loop under consideration
        While basic block not allocated do {
            If memory locations $t_x + Z_r + i * S$ to $e_x + Z_r + i * S$
            is free then
                Map basic block to address $t_x + Z_r + i * S$ to $e_x +
                Z_r + i * S$
            Else
                i++ }}
```

3 Enhancing Code Placement's Efficiency through Adaption of Bus Coding

The code placement algorithm introduced in Section 2 reduces the number of cache misses. Hence, the traffic on the CPU-to-cache bus and the cache-to-main-memory bus is significantly reduced, leading to a higher performance of the whole system and a decrease of the interconnect energy dissipation. The aim of this Section is to adapt a bus encoding scheme that amplifies these two effects even further and thus leads to an ultra-low bus power consumption combined with an effectively increased bus-bandwidth that yields a higher system performance as well. Since we target sub-0.10μ technologies it is necessary to also provide means for cross-talk reduction since signal integrity is another major concern. In the following we introduce the bus encoding scheme to address these problems.

Normalized coupling value		i,j^{t+1}			
		00	01	10	11
i,j^t	00	0	1	1	0
	01	0	0	2	0
	10	0	2	0	0
	11	0	1	1	0

Table 2: Normalized coupling capacitance for all transition combinations of two adjacent bus lines i and j

3.1 Buses in Deep-submicron Designs

The closer geometrical proximity of adjacent bus lines in sub-0.10μ technologies leads to effects that are almost negligible in technologies not as advanced as 0.10μ and below: two adjacent bus lines form a parasitic capacitance between them. This effect does not only lead to cross-talk and delay effects, it also leads to an increased power consumption since the parasitic capacitance is charged and discharged when there is a voltage swing between two or more bus lines. Thus, each bus line's capacitance can be represented as

$$C_i = C_B + C_{C,left} + C_{C,right} \tag{1}$$

where C_B is the base (or intrinsic) capacitance (capacitance between bus line and metal layers) and $C_{C,left}$, $C_{C,right}$ are the left and right coupling capacitance between bus line I and it's left and right neighbor (if any) respectively. Table 2 shows the normalized coupling capacitance C_C between a bus line i and one of its neighbors j according to the values the bus lines take at time $T_1 = t$ and $T_2 = t + 1$. Obviously, the coupling effect is highest when both lines are subject to a transition in the opposite direction. We have measured the absolute capacitances C_B and C_C for a 0.10μ technology:

$C_B = 42.22pF/m$
$C_C = 35.89pF/m$

According to Equation 1 and the table above, the maximum capacitance for a bus line I we can expect is:

$$C_{i,max} = (42.22{+}2{\times}35.89{+}2{\times}35.89)pF = 185pF/m \tag{2}$$

Compared to the case where the inter-wire capacitances are negligible (i.e. $C_{i,max} = C_C = 42.22pF/m$ this is 4.4 times higher. This is why inter-wire effects have to be taken into consideration.

There are several means to diminish or at least reduce the problem of inter-wire capacitances:

a) Widen the distance between bus lines: this is typically not preferred since the total area of the bus systems grows too large.

b) Use P&R *place & route* tools that avoid side-by-side routing of bus lines. This is what is actually done in the newest generation of P&R tools. However, the interconnect complexity of one billion transistor SOCs with multiple bus hierarchies and long buses with many cores connected to them will prevent a satisfying solution at a feasible routing time (complexity of the routing problem).

c) Change the geometrical shape of bus lines: the bus lines themselves can be re-shaped. For example, the cross-sectional shape can be made narrower such that the distance between two bus lines increases without sacrificing space for the whole bus. However, the main disadvantage of this approach is that the cross-sectional area of a bus line is fixed, since the current-per-area ratio is fixed for any certain technology. That typically leads to solutions where the bus line is buried deeper into the substrate with the height being larger than the width of a bus line. However, even though the inter-wire capacitance decreases due to a decreasing distance between bus lines, it does increase due to the increased flank area of two opposing bus lines. In conclusion: what is won through a wider distance has to be, at least partly, given up through the effect of larger flank areas.

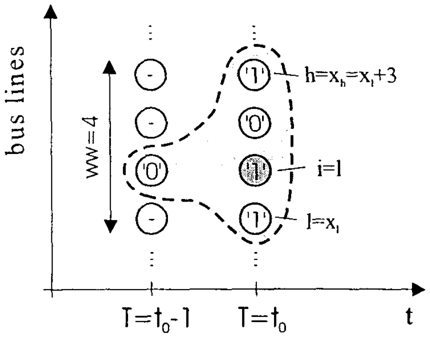

Figure 2: a) a32-bit bus partitioned in 8 windows of 4bit each b) and c) examples for calculating the TA measure for two cases within a window

d) Bus encoding techniques that take inter-wire capacitances into consideration when words are transmitted via a bus system.

Within this section, we focus on the latter technique, namely on finding a bus encoding technique that compliments the code placement technique introduced earlier and thus leads to an ultra-low bus power consumption combined with an effectively increased bus-bandwidth that yields a higher system performance

3.2 Reducing Power and Increasing Signal Integrity

In the following we introduce an encoding method that solves the problems discussed in sub-section 3.1 and that serves as an enhancement to further optimize the advantages achieved through the code placement techniques from Section2.

Let us first define what we call a *window*:

$$w_{l,h}(ww) = \{l, h|\ h - l = ww - 1, h > l, 0 \leq (h,l) \leq bw - 1\} \tag{3}$$

with l, h being the lower and upper border bit positions of the window, respectively, ww the window size in bits and bw the bus size in bits.

Now, let us first define what we call the Transition Activity TA for a window $w_{l,h}(ww)$. In order to make the formula easier readable we simply use w to denote the window. Furthermore, let us assume that b_x is the x'th bit within a window with B_x being the value of that bit (i.e. $B_x \in \{0,1\}$). Thus, we can define the TA measure as follows:

$$\text{TA}(w) = \sum_{\forall b_i \in w} \left(\left(B_i \oplus B_i^{-1}\right) + \left(B_i \oplus B_i^{-1}\right) \cdot \sum_{\forall b_j \in w, b_j \neq b_i} (B_i \oplus B_j)\right) \tag{4}$$

Thereby B_i^{-1} gives the value of bit b_i at time $t-1$ i.e. the temporal predecessing value. Thus, $B_i \oplus B_i^{-1}$ determines whether bit b_i has a high/low or low/high transition (=1) or not (=0). Accordingly this specific bit will contribute to the TA measure or not. Figure 2 gives an idea on how TA is measured using an example. There, the portion of the TA measure contributed by $i = a+1$ is demonstrated. The dotted line shows the scope that is important for the calculation of the respective TA portion. It equals to 2.

It is very important to note that TA as shown does NOT violate the causality principle as it might seem from the Figure 2. This is because the bus word referring to time $T = t - 1$ is stored in a register. But even the bus word for time t is stored in a register since the word is not yet put on the bus (it is just in the I/O register of a device) and thus TA does work as intended by Equation 4.

According to Equation 4 every value of a bit different to the bit under review is contributing 1 or 0 to the value of TA depending on whether it is different in value or not. That each contribution is equally sized (1 or 0 with no other values allowed) is justified by our capacitance measure that gives us values of base capacitance compared to coupling capacitances of the closest neighbors (a maximum of three left or right neighbors in a **4-bit window**) that are approximately the same and thus contribute the same to the power/energy consumption. Window sizes larger than 4 bit yielded lower energy savings since such a model would assume that inter-wire capacitances reach far beyond the closest neighbor (which is actually not the case). Window sizes less than 4 bits on the other side might be more beneficial in terms of power savings (3 would be ideal since it exactly reflects the physical relationship of *adjacent* bus lines) but the additional hardware effort cannot be justified.

In the next step we use the TA as a measure to determine whether we should invert the information in the window or not. Please note that the TA scheme is able to measure the impact of coupling capacitances. A Hamming Distance measure, as used for regular invert schemes would not lead to a reasonable improvement in power/energy consumption. It would only reduce the number of transitions. But the number of transitions does not necessarily reflect the amount of power/energy that is consumed. Our whole scheme works according to the following procedure: For all windows the TA measure is calculated (lines 1-2). If the measure exceeds half of the maximum value (dependent on the window size ww) then it is counted (lines 3-5). After all TA measures are calculated, it is determined whether more than half of the windows have a high TA value (Equation 4) If that is the case the information in the windows is transmitted inverted. Please note that decoding can be done inversely. Only 1 extra bit line is used for that since *all* windows will be inverted or not (majority vote).

Strategy of the Scheme

```
1)  For All   windows $w_i \in W$
2)    determine TA($w_i$)
3)    If   TA($w_i$) > $TA_{max}(ww)/2$
4)    Then
5)      $hi\_ta+ = 1$
6)
7)    If   $hi\_ta > (\#windows)/2$
8)    Then
9)      For All   windows $w_i \in W$
10)       $invert(w_i)$
11) done.
```

Figure 3: The strategy of the Scheme

Also, note that this code explains only the strategy. It does not in any way reflect the implementation that, of course, is in hardware.

Hardware related issues

The design bus encoding interface including an encode/decode pair uses approximately 400 gates; it does not incur an additional clock cycle; the critical path is between 2-3ns)).The whole encoding scheme has been designed with signal integrity in mind since this is another major issue in sub-0.10μ designs. As explained earlier, the scheme aims to minimize the switching activities within a certain window as the TA measure (Equation 4) shows. That means that the probability of switching within a window is being reduced and thus reduces the probability of violating the signal integrity through, for example, crosstalk between two adjacent bus lines. Bus lines located at the border of a window may still interfere with bus lines located at the border of adjacent windows. But note that due to the scheme, this effect is not any larger than in the non-encoded case.

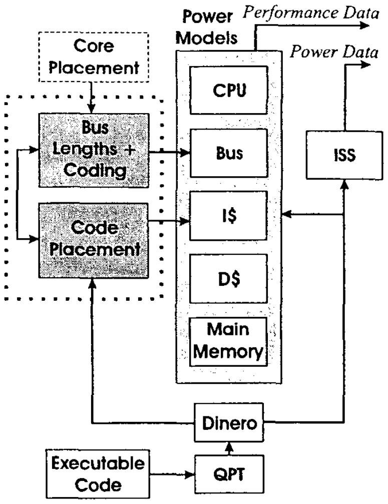

Figure 4: The whole set-up for power and performance estimation and including the focus of this work i.e. code placement strategy adapted to bus encoding and bus length determination.

Minimizing the remaining border-to-border effects could be achieved by increasing the window size. However, this would decrease the efficiency of the encoding scheme and thus it is contrary to the low power goal.

4 Validation Environment

We explain the experiments in Section 5, but in prelude to it we briefly introduce our validation environment.

It is the main goal of this work to show the efficiency of combining code placement and bus coding for an ultra-low power bus/interconnect for an SOC. These two methodologies are highlighted in a dashed box in Fig. 4. The bus lengths of the involved buses (i.e. buses between instruction cache and main memory and instruction cache and CPU) are crucial parameters for the power consumption. The lengths are determined by the results of the core placement (memory, cache and CPU). The results of the code placement and the bus coding scheme are fed into the power models of instruction cache and the bus system, respectively. Further power models in the environment are a CPU power model, a data cache power model and a main memory power model. All models plus the code placement and bus encoding mechanisms are fed by instruction traces through the *"QPT"/"Dinero"* tool set sequence [6]. The output of the environment is power and performance data. For more detailed information please refer to [14].

5 Experiments and Results

The target system the experiments were conducted on is shown in Fig. 5: it shows a chip layout with the interesting parts magnified: the CPU, the instruction cache ("*I$*"), and the main memory banks. The buses that are affected by our code placement and bus encoding methodologies are buses *"Bus1"* and *"Bus2"*. The length and/or the ratio of the lengths of these buses varies with the placement and relative size of all cores comprised within this SOC. Hence, the bus power consumption will not only depend on our methodologies (see Sections 2

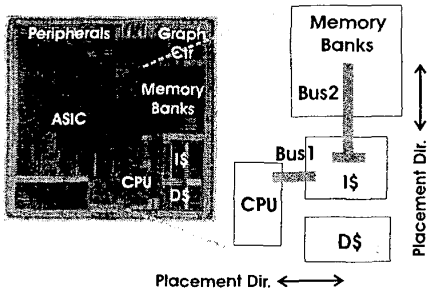

Figure 5: Chip-layout and buses "Bus1" and "Bus2" that are subject to extension/contraction according to the placement of the involved cores.

and 3) but also on the geometrical characteristics of *"Bus1"* and *"Bus2"*. This is one of the parameters that will be investigated in this section.

The experiments were conducted with the evaluation environment shown in Fig. 4. Here are the main steps:

1) Placing the instruction code according to Section 2.
2) Generating traces for the new code allocation.
3) Running the traces through the bus encoding scheme (Section 3).
4) Measuring power and performance with the evaluation environment.
5) Varying instruction cache sizes.
6) Varying the ratios of bus lengths *"Bus1"* to *"Bus2"* (see Fig. 5) according to different placement scenarios of the affected cores.
7) Repeating steps 1)-4) for all applied combinations of parameters.

We performed experiments on a set of five applications. The applications have been chosen with as much variety in characteristics as possible in order to show the wide application area of the methodology. Thus, the applications varied in size (8k to 200k), application area (video, animation, algorithmic etc.) and application domain (data dominated or control dominated). The applications used were: a complete MPEGII video encoder *mpeg*, a video trick animation algorithm *trick1*, the Whetston benchmark sequences *whetston*, the unix command compress *compress*, and a chromakey video mixer as part of a digital video studio equipment.

The results achieved are shown in Table 5. The first column gives the application name and the number of instructions executed for that application. The second column gives the cache sizes. The third column gives the cache miss rates before code placement and the fourth column gives the miss rates after code placement. The next five columns (columns 5-9) are results which have been obtained by simulating with bus lengths of 0.2mm for the cpu-cache bus and 3.8mm for the cache-memory bus. The fifth column gives the energy expended for a system without optimization. Column six shows the the energy expended with address coding only, and the seventh column shows energy consumption in the busses if only code placement was performed and finally in column eight we show the energy consumption when both address coding and code placement methodologies are applied. Column nine shows the percentage improvement between column eight and column five. Columns 10-14 are in a similar format to Columns 5-9 except that they are results of a simulation with 0.5mm and 3.5 mm for the respective bus lengths. Likewise, Columns 15-19 are results of a simulation with 0.8mm and 3.2 mm for the respective bus lengths.

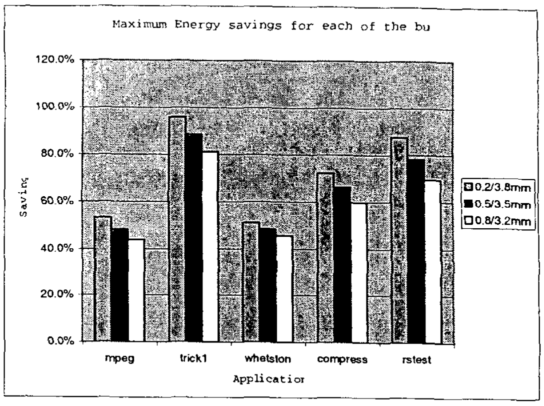

Figure 6: The graph of Max Savings

As can be seen from the figures, the energy savings of the bus system are quite substantial. The maximum savings achieved for each of the applications with differing bus lengths are given in Figure 6. Even though the energy consumption of the buses using current mainstream silicon technologies (i.e. 0.18μ, 0.13μ) is only around 5% to 20% of the whole SOC, it is anticipated that in sub-0.10μ technologies this portion raises to 20% to 30% [28].

On average, for the applications investigated in our experiments, the energy consumption of the bus system is reduced by 54.6% for the 0.2/3.8 mm (*"Bus1"* / *"Bus2"*) bus case, 50.4% for the 0.5/3.5 mm bus case and 46.6% for the 0.8/3.2 mm bus case.

Another factor is the relative lengths of the busses: at some point the relative length of the busses are going to be of importance. If the two busses were of equal length, then the energy consumption might have even increased. At that point the designer will have to decide upon the relative merits of energy saving, crosstalk reduction and performance. Typically, *"Bus1"* will be much shorter than *"Bus2"* since this is the preferable outcome of place&route according to the sizes of the involved cores (instruction cache, main memory banks, CPU). Our methodology favors this tendency as we can observe that results are best for a smaller *"Bus1"* to *"Bus2"* ratios for reasonable cache misses. As the cache misses increase beyond 50% the energy dissipation does not always favor smaller bus ratios. Note that the best results were achieved for "feasible" instruction cache sizes i.e. cache sizes that are neither too small nor too large as to fit the entire code in the instruction cache anyway. These are design points chosen by designers as the best compromise between effort (i.e. costs) and results (e.g. performance, power). In those cases (bold highlighted) we can always achieve a large drop in cache misses. The implications are a higher performance and a lower energy consumption. Also, the performance of the entire system increases substantially because of the reduction in the cache misses (due to the code placement strategy) as can be seen from columns 3 and 4.

Thus we can state that the two combined methods should be used in conjunction to produce ultra low power systems with increased performance and reduced crosstalk.

For certain applications it can be seen that address coding alone will produce superior results to that of the combined scheme (see Compress with a cache size of 64 in Table 2). This is due to the fact that the code placement algorithms has a number of jump instructions which are further away than in the non-code-placed algorithm and this is causing it to increase the bus activity. Note that for this application there was little reduction

Bus length ratio				0.2/3.8					0.5/3.5					0.8/3.2				
Appl. # Instr	**I-$ Size**	**Miss Ratio**	**Code-placed Miss Rat.**	**No-opt 105J**	**A Only 105J**	**C only 105J**	**A&C 105J**	**% Imp.**	**No-opt 105J**	**A Only 105J**	**C only 105J**	**A & C 105J**	**% Imp.**	**No-opt 105J**	**A Only 105J**	**C only 105J**	**A & C 105J**	**% Imp.**
mpeg 22406459	128	72.94%	71.34%	6.63	4.00	6.61	4.33	34.7%	6.31	3.81	6.30	4.13	34.6%	5.98	3.61	6.00	3.93	34.4%
	256	56.20%	51.26%	5.22	3.15	4.72	3.03	41.9%	5.13	3.10	4.71	3.02	41.2%	5.04	3.04	4.69	3.01	40.4%
	512	47.85%	36.35%	4.51	2.72	3.62	2.52	44.1%	4.54	2.74	3.80	2.65	41.7%	4.57	2.76	3.98	2.77	39.3%
	1024	**32.93%**	**20.77%**	**3.25**	**1.96**	**2.19**	**1.53**	**53.0%**	**3.49**	**2.11**	**2.60**	**1.81**	**48.1%**	**3.73**	**2.25**	**3.00**	**2.09**	**43.9%**
	2048	2.60%	1.39%	0.69	0.42	0.58	0.44	36.3%	1.36	0.82	1.27	0.96	29.6%	2.02	1.22	1.95	1.47	27.3%
trick1 1.03E+08	128	100.00%	100.00%	37.91	22.45	38.52	22.31	41.2%	34.92	20.67	35.48	20.55	41.2%	31.93	18.90	32.44	18.79	41.2%
	256	99.58%	97.16%	37.76	22.36	30.67	15.31	59.5%	34.79	20.60	28.32	14.14	59.4%	31.83	18.84	25.98	12.97	59.3%
	512	87.70%	62.15%	33.49	19.83	22.91	14.89	55.5%	31.24	18.49	22.23	14.45	53.8%	28.98	17.16	21.54	14.00	51.7%
	1024	**71.17%**	**0.04%**	**27.56**	**16.31**	**2.13**	**1.19**	**95.7%**	**26.29**	**15.56**	**5.30**	**2.96**	**88.7%**	**25.02**	**14.81**	**8.47**	**4.73**	**81.1%**
	2048	17.70%	0.00%	8.35	4.49	1.93	2.09	75.0%	10.29	6.09	4.82	3.64	64.6%	12.22	7.23	7.70	5.82	52.4%
whetston 1749402	128	97.25%	93.51%	0.48	0.33	0.47	0.32	33.4%	0.44	0.30	0.43	0.30	33.2%	0.41	0.28	0.40	0.27	32.9%
	256	**69.46%**	**47.81%**	**0.35**	**0.24**	**0.24**	**0.17**	**50.9%**	**0.34**	**0.23**	**0.24**	**0.17**	**48.3%**	**0.32**	**0.22**	**0.25**	**0.17**	**45.5%**
	512	6.18%	6.10%	0.05	0.04	0.05	0.04	31.1%	0.09	0.06	0.09	0.06	30.9%	0.12	0.08	0.12	0.09	30.8%
	1024	0.06%	0.06%	0.03	0.02	0.02	0.02	30.9%	0.07	0.04	0.06	0.04	30.9%	0.10	0.07	0.10	0.07	30.9%
compress 53280973	64	89.79%	89.79%	12.52	8.11	12.57	8.30	33.6%	11.65	7.55	11.69	7.73	33.6%	10.78	6.98	10.82	7.15	33.6%
	128	74.49%	53.55%	10.51	6.81	7.77	5.15	51.0%	9.97	6.46	7.69	5.10	48.8%	9.44	6.11	7.62	5.05	46.5%
	256	**54.73%**	**19.69%**	**7.91**	**5.13**	**3.10**	**2.18**	**72.5%**	**7.81**	**5.06**	**3.75**	**2.64**	**66.2%**	**7.71**	**4.99**	**4.40**	**3.10**	**59.8%**
	512	13.33%	2.92%	2.48	1.61	1.06	0.77	68.8%	3.28	2.13	2.05	1.49	54.6%	4.08	2.65	3.03	2.20	46.0%
	1024	2.39%	0.48%	1.04	0.68	0.77	0.55	47.4%	2.08	1.35	1.81	1.30	37.7%	3.13	2.03	2.86	2.05	34.4%
	2048	0.83%	0.10%	0.84	0.54	0.70	0.54	35.9%	1.91	1.24	1.73	1.33	30.6%	2.99	1.94	2.76	2.12	29.1%
rstest 28235416	128	86.86%	84.16%	7.79	4.44	7.72	4.26	45.4%	7.27	4.14	7.23	3.99	45.2%	6.75	3.85	6.74	3.72	45.0%
	256	49.23%	16.30%	4.62	2.63	1.88	1.13	75.6%	4.63	2.64	2.36	1.41	69.5%	4.64	2.64	2.84	1.70	63.3%
	512	**43.79%**	**4.35%**	**4.16**	**2.37**	**0.79**	**0.52**	**87.5%**	**4.25**	**2.42**	**1.39**	**0.92**	**78.3%**	**4.33**	**2.47**	**2.00**	**1.32**	**69.5%**
	1024	23.57%	1.30%	2.45	1.40	0.44	0.30	87.7%	2.83	1.61	1.09	0.75	73.6%	3.20	1.82	1.74	1.19	62.8%
	2048	23.52%	0.01%	2.45	1.40	0.47	0.28	88.4%	2.82	1.61	1.19	0.71	74.9%	3.19	1.82	1.90	1.14	64.5%

Table 3: Table of results

in cache misses.
We mentioned earlier that the signal integrity has been a concern in this work even though performance and power consumption were the main goals:

a) The code placement method leads to a decreased traffic on the CPU-to-cache bus and the cache-to-main-memory bus. This reduced traffic is *not* traded against an increased traffic elsewhere. Consequently, there is a lower vulnerability through crosstalk just through the minimized traffic on the buses.

b) In a second step, the data on these buses is encoded to increase signal integrity (see Section 3).

6 Conclusions

In this work we have expoited the interdependencies between a high-level optimization technique, namely code placement, and a lower-level optimization technique, namely bus encoding. It could be shown that these previously orthogonally handled techniques are in fact interdependent on each other due to the increasing influence of deep sub-micron effects. As a result we have achieved much higher interconnect energy savings than any of these methods can achieve when applied solely (according to related research): the average SOC interconnect energy savings are 50% with a maximum of 95.7% The performance improvements are shown by large reductions in cache misses.. We have validated the results by means of real-world SOC applications that range in size between 8k and 200k lines of code. As an added benefit, the probability of crosstalk effects is reduced by both code placement and bus encoding techniques.

References

[1] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu, *IMPACT: an architectural framework for multiple-instruction-issue processors* in Computer Architecture News. vol.19, no.3; May 1991;

[2] S. McFarling, *Program optimization for instruction caches,* in ASPLOS III Proceedings. Third International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, New York, NY, USA; 1989; x+303 pp. p.183 91, 1989.

[3] S. McFarling, *Procedure merging with instruction caches,* in SIGPLAN Notices. vol.26, no.6; June 1991; p.71

[4] P. Panda, N. Dutt, and A. Nicolau, *Memory organization for improved data cache performance in embedded processors,* , 1996.

[5] L. Yanbing and W. Wolf, *Hardware/software co-synthesis with memory hierarchies,* Design Automation Conference, 1998.

[6] M. D. Hill, J. R. Laurus, A. R. Lebeck et al., *WARTS: Wisconsin Architectural Research Tool Set,* Computer Science Department University of Wisconsin.

[7] F. Chow, *A portable machine-independent global optimizer—Design and measurements,* Tech. report 83-254, PhD thesis, Computer Systems Lab, Stanford Univ., 1983.

[8] D. B. Kirk, *SMART (strategic memory allocation for real-time) cache design,* in Proceedings. Real Time Systems Symposium (Cat. No.89CH2803 5). IEEE Comput. Soc. Press, Los Alamitos, CA, USA; 1989; pp. p.229 37, 1989.

[9] D. B. Kirk and J. K. Strosnider, *SMART (strategic memory allocation for real-time) cache design using the MIPS R3000,* in Proceedings. 11th Real Time Systems Symposium (Cat. No.90CH2933 0). IEEE Comput. Soc. Press, Los Alamitos, CA, USA; 1990; xi+341 pp. p.322 30, 1990.

[10] D. Kirovski, L. Chunho, M. Potkonjak, and S.-W. H. Mangione, *Application-driven synthesis of memory-intensive systems-on-chip,* IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 18, pp. 1316-1326, 1999.

[11] H. Tomiyama and H. Yasuura, *Code placement techniques for cache miss rate reduction,* ACM Transactions on Design Automation of Electronic Systems, vol. 2, 1997.

[12] C. Kulkarni, F. Catthoor, and H. De Man, *Code transformations for low power caching in embedded multimedia processors,* Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing IEEE Comput. Soc, Los Alamitos, CA, USA; p.292-7 , 1998.

[13] F. Catthoor, N. D. Dutt, and C. E. Kozyrakis, *How to solve the current memory access and data transfer bottlenecks: at the processor architecture or at the compiler level?,*Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 , 2000.

[14] Taken out to enable blind review.

[15] S. Parameswaran, *Code placement in Hardware Software Co-Synthesis to Improve Performance and Reduce Cost,* Design Automation and Test in Europe, pp. 627-633, 2001.

[16] M.R. Stan, W.P. Burleson, *Bus-Invert Coding for Low-Power I/O,* IEEE Tr. on VLSI Systems, Vol. 3, No. 1, pp.49-58, Mar. 1995.

[17] M. R. Stan, W. P. Burleson, *Low-Power Encodings for Global Communication in CMOS VLSI,* IEEE Tr. on VLSI Systems, Vol. 5, No. 4, pp.444-455, Dec 1997.

[18] E. Musoll, T. Lang, J. Cortadella, *Working-Zone Encoding for Reducing the Energy in Microprocessor Address Buses,* IEEE Tr. on VLSI Systems, Vol. 6, No. 4, pp. 568-572, Dec. 1998.

[19] J. Henkel, H. Lekatsas, *ABC: adaptive address bus coding for low power deep sub-micron designs,* Proc. of Design Automation Conference, pp 744-749, 2001.

[20] P.R. Panda, N.D. Dutt, *Low-Power Memory Mapping Through Reducing Address Bus Activity,* IEEE Tr. On VLSI Systems, Vol. 7, No. 3, pp. 309-320, Sept. 1999.

[21] H. Mehta, R.M. Owens, M.J. Irwin, *Some issues in gray code addressing,* Proc. Of IEEE Conf. On sixth. Great Lakes Symp. On VLSI, pp. 178-181, 1996.

[22] P.P. Sotiriadis, A. Chandrakasan, *Bus energy minimization by transition pattern coding (TPC) in deep sub-micron technologies,* IEEE/ACM International Conference on Computer Aided Design, pp. 322- 327, 2000.

[23] L. Benini, G. DeMicheli, E. Macii, D. Sciuto, C. Silvano, *Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems,* Proc. of The Seventh Great Lakes Symp. On VLSI, pp. 77-82, 1997.

[24] K. Kim, K. Baek, N. Shanbhag, C.L. Liu, S. Kang, *Coupling- driven signal encoding scheme for low-power interface design,* IEEE/ACM International Conference on Computer Aided Design, pp. 318-321, 2000.

[25] S. Ramprasad, N. R. Shanbhag, I. N. Hajj, *Information- Theoretic Bounds on Average Signal Transition Activity,* IEEE Transaction on VLSI vol. 7, no. 3, pp. 359-368, 1999.

[26] L. Macchiarulo, E. Macii, M. Poncino, *Low-Energy Encoding for Deep-Submicron Address Buses,* IEEE/ACM Proc. of International Symposium on Low Power Electronics and Design (ISLPED'01), pp.176-181, 2001.

[27] Taken out to enable blind review.

[28] National Technology Roadmap for Semiconductors, Semiconductor Industry Association (SIA), 1997.