Behavioral Array Mapping into Multiport Memories Targeting Low Power *

Preeti Ranjan Panda and Nikil D. Dutt
Department of Information and Computer Science
University of California, Irvine, CA 92697-3425, USA

Abstract

Off-chip memories are typically used during behavioral synthesis to store large arrays that do not fit into on-chip registers. An important power-optimization problem that arises in this context is the minimization of signal transitions on the off-chip buses connecting the ASIC and the memory. We address the problem of system power reduction through transition count minimization on the multiported memory's address buses when these arrays are accessed from memory at execution time. We exploit regularity and spatial locality in the memory accesses and determine a power-efficient mapping of behavioral array references to physical locations as well as ports of a multiport memory. Our experiments on several image processing benchmarks show significant power savings through reduced transition activity on the memory address buses, compared to a straightforward mapping scheme.

1. Introduction

Most applications in the Image Processing and Digital Signal Processing domain that are synthesized into ASICs involve large arrays stored in off-chip memories. Often, high memory access bandwidth requirements necessitate the use of multiport memories to increase the effective speed of memory data access. The high memory access freqency makes the ASIC-memory interface an attractive region for power minimization. An important power-optimization problem that arises in this context is the minimization of signal transitions on the off-chip data and address buses. Most power-optimization efforts aim at reduction of signal transition count, due to dynamic power forming a significant fraction of the total power consumed in CMOS designs. Given that off-chip capacitances are three orders of magnitude larger than typical on-chip capacitances [1], we can effect significant power savings by reducing the switched capacitance of the off-chip address bus drivers through reduction of transition activity on the memory address bus. Furthermore, reduced activity in the address bus also leads to reduced activity in the memory address buffers and decoding circuitry. Studies have shown that power dissipation in the address decoder and address buffers of typical memory chips constitute a significant portion of the power consumed (upto 50%) in the memory chip [8]. Hence, design techniques leading to decrease in power dissipation in this part of the memory will significantly reduce the overall power dissipation of the application.

Power-related cost functions have been incorporated into phases of High-level Synthesis in [2, 3, 4]. However, they have not addressed the impact of memory accesses in their techniques. [8] presents transformations on the initial specification into a power-optimized form for reducing the number of memory accesses. [7] presents a technique for exploiting data encoding to reduce transition activity on I/O pins at the expense of a moderate increase in on-chip transitions, by analyzing data streams on I/O pins.

In [6], a technique for minimizing address bus transitions was presented, which exploited regularity and spatial locality in the memory accesses in a behavioral description and determined the mapping of behavioral array references to physical locations in a single memory. In this paper, we present a generalization of the array mapping techniques to an architecture employing multiport memories. Analysis of the behavioral memory access patterns at compile-time ensures a guaranteed reduction in address bus transition activity; our mapping strategy can thus result in additional power minimization after other behavioral power minimization techniques have been applied.

2. Problem Definition

The goal of this work is to assign arrays in a behavioral specification to memory locations, and ports of multiport memories, so as to minimize the transition count on the memory address buses when these arrays are accessed. Figure 1 shows the model we assume for a typical memory-intensive system, synthesized into an ASIC (consisting of datapath and control blocks) and a multiport memory module. We have a pair of data and address buses connecting

*This work was partially supported by NSF Grant CDA-9422095

0-8186-7755-4/96 $05.00 © 1996 IEEE

the ASIC to each memory port. The datapath is connected to an m-port memory, with ports $P_0..P_{m-1}$, consisting of m data buses $D_0..D_{m-1}$ and m address buses $A_0..A_{m-1}$. The address generator generates the m addresses for the address buses $A_0..A_{m-1}$, so that upto m data words can be accessed in parallel, in the same cycle. The address value is assumed to be latched before being fed to the off-chip driver.

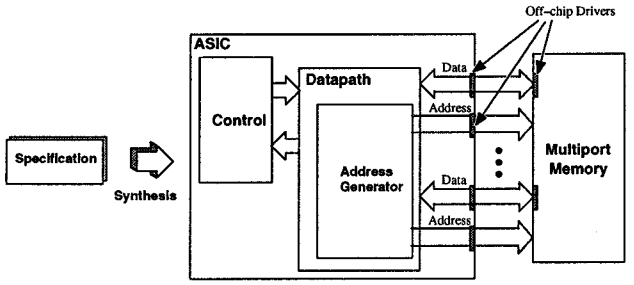

Figure 1. Synthesis model of a memory-intensive system

For a given number of memory accesses, it is difficult to design a hardware implementation for minimizing data bus transitions because the data is not known a priori. However, the sequences of memory access patterns in the behavior are known at the specification stage, giving us an opportunity to minimize power dissipation arising out of transitions on the memory address bus.

3. Mapping into a Single-Ported Memory

[6] presents a power-efficient technique for mapping of behavioral arrays into a single-ported memory. Essentially, the technique minimizes the number of bits transitioning on the address bus when successive data words are accessed from memory. Locality of reference considerations are taken in to account when choosing one of the following storage schemes for multi-dimensional arrays: (1) *row-major* (row by row) (2) *column-major* (column by column) and (3) *tile-based* (array is divided into rectangular tiles, and the tiles are stored in sequence). The dimensions of the tile follow from the smallest rectangular region enclosing the pattern formed by array accesses in the inner loop. Figure 2(c) shows an example of a tile derived from the access pattern in Figure 2(b).

We define a *maximal transition* as occurring when two logical addresses with a large difference are accessed in succession. A *minimal transition* occurs when this difference is small. In this case, *large* means comparable to the dimension of the array - we treat all consecutive accesses to elements in the same tile as minimal transitions. For example, if row-major storage is used for u in Figure 2(b), successive accesses to the pair of elements $(u[i][j], u[i][j+1])$ result in a minimal transition, whereas those to $(u[i][j], u[i+1][j])$ lead to a maximal transition. For each array, we choose a mapping scheme that minimizes the number of maximal transitions. In essence, we observe that if column-major mapping is used when the tile has n columns, there will be n maximal transitions in every inner loop iteration (assuming at least one element from every column is accessed). Similarly, row-major mapping would entail n maximal transitions in an n-row tile. In comparison, tile-based mapping leads to a maximum of 2 maximal transitions, independent of the tile size. We thus have a heuristic for a 2-level nested loop, involving a two-dimensional array (**Heuristic 1** in [6]) that chooses tile-based mapping when the tile has both dimensions greater than 2. Otherwise, it selects row- or column-major as appropriate.

4. Mapping into a Multiport Memory

The problem of mapping into a multiport memory requires the solution of two sub-problems: (i) determine the mapping style to use for each array, and (ii) assign specific array elements accessed within behavioral loops to memory ports, such that the total transition count on all the address buses is minimized.

An example port assignment for the code fragment in Figure 2(a) is shown in Figure 2(b), for a dual-port memory with ports P_0 and P_1. In every inner loop iteration, elements $u[i-1][j]$ and $u[i][j]$ are accessed from port P_0 and elements $u[i][j+1]$ and $u[i+1][j]$ are accessed from port P_1. Note that we wish to satisfy the maximal access throughput possible in order to retain the advantage of the multiport memory in the architecture. This means, for instance, that the assignment $-(P_0 : \{u[i-1][j], u[i][j], u[i+1][j]\}; P_1 : \{u[i][j+1]\})$ is not possible, as this assignment would require three memory read cycles (on port P_0) in the inner loop, as opposed to only two in the previous case.

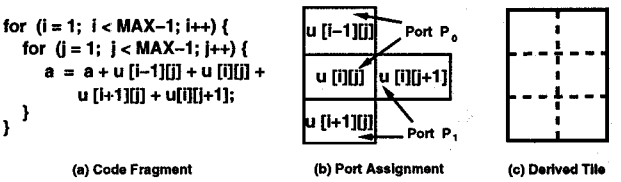

Figure 2. Port Assignment for Example Code Fragment

First, we describe a port assignment pattern for which *tile-based* mapping incurs no more than 2 *maximal transitions* per iteration. We then describe the special cases for which *row-* and *column major* styles perform equally well, or better.

Let S be the number of new array elements accessed in each iteration of the inner loop and the memory ports be $P_0..P_{m-1}$. For example, in Figure 2(b), $S = 4$ and $m = 2$.

The minimum number of cycles required to access all the elements, $T = \lceil S/m \rceil$. We assume that we are constrained to operate at the highest possible performance level, i.e., the S array elements should be accessed in no more than T cycles. This requires the assignment of upto T elements to each port.

Suppose the access pattern formed from the S new elements accessed in the inner loop moves horizontally for different iterations of the inner loop and the enclosing tile has dimensions $M \times N$. We call such a motion *row-first*. If the pattern moves vertically in successive inner loop iterations, we call the motion *column-first*. We use a procedure *PortAssignTILE* to determine the port assignment of tile elements. This procedure assigns the i-th group of T elements to port P_i.

Figure 3(a) shows a tile of size 5×4, with the elements $\{e_0 \ldots e_{16}\}$ marked, and the port assignments computed by procedure *PortAssignTILE*, with $m = 3$. Elements $\{e_0 \ldots e_5\}$ are assigned to port P_0; $\{e_6 \ldots e_{11}\}$ are assigned to port P_1; and $\{e_{12} \ldots e_{16}\}$ are assigned to port P_2.

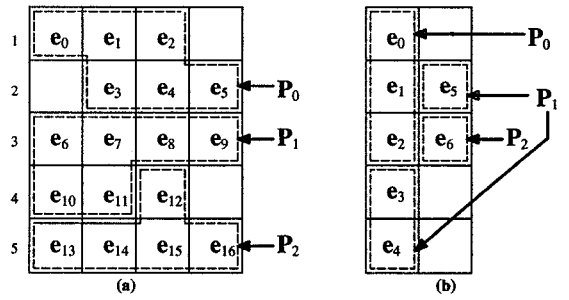

Figure 3. (a)Port Assignment for Tile-based Mapping (b) Port Assignment for 2-Column Tile

We observe, using the same reasoning as in Section 3, that there can be no more than 2 *maximal* transitions in every loop iteration. Suppose the *tile boundary* ([6]) is horizontal and separates rows 3 and 4 in Figure 3(a) (i.e., for a particular iteration, rows 1,2, and 3 are in the same vicinity in memory, and so are rows 4 and 5; but rows 3 and 4 are located far apart) . Since port P_1 has to access elements from two vertically adjacent (i.e., distant) tiles, it incurs one maximal transition. This occurs when element e_{10} is accessed after e_9. The rest of the accesses (including *all* accesses from ports P_0 and P_2) cause *minimal* transitions. It is easy to see that the port assignment strategy described above leads to 2 maximal transitions, no matter where the horizontal boundary lies. [1]

We use the same reasoning as in Section 3 to conclude that Row- and Column-major mapping can perform almost equally well when the tile has one of its dimensions ≤ 2.

[1]The other maximal transition in the current example occurs when P_1 accesses the corresponding e_6 in the next iteration.

Figure 3(b) shows an example array access pattern with a 2-column tile, with the port assignments as indicated, for $m = 3, S = 7$, which gives $T = \lceil 7/3 \rceil = 3$. This assignment produces one maximal transition for port P_1 in a column-major mapping. However, the following observations are noteworthy:

Observation 1 – If the tile moves *row-first* then all m ports incur a maximal transition in every iteration when column-major mapping used. For instance, port P_0 (Figure 3(b)) incurs a maximal transition when accessing e_2 in one iteration, followed by e_0 in the next iteration. Therefore, column-major mapping is preferred for an access pattern with two columns, only if the pattern moves *column-first* (otherwise, tile-based mapping performs better). A similar argument requires us to use row-major mapping only if the access pattern moves *row-first*.

Observation 2 – Note that in Figure 3(b), a better port assignment that causes no maximal transitions in the inner loop, is – $P_0 = \{e_0, e_1, e_2\}, P_1 = \{e_3, e_4\}$ and $P_2 = \{e_5, e_6\}$, assuming column-major mapping is used, and the access pattern moves column-first. This is because all three ports access consecutive elements, and hence, incur only minimal transitions.

Procedure *PortAssignCOLUMN* below assigns elements of the extracted tile to ports P_0 to P_{m-1} in a column-major mapping, taking the above observations into account. We begin by assigning blocks of T contiguous elements in each column. This uses up D_1 and D_2 ports in columns 1 and 2 respectively in procedure *PortAssignCOLUMN*. If more than T elements now remain to be assigned ($C_1 + C_2 > T$, i.e., all remaining $C_1 + C_2$ elements cannot be accomodated into the same port), we make the assignment that does not require the same port to access elements from different columns. Procedure *PortAssignROW* is defined symmetrically to procedure *PortAssignCOLUMN*.

procedure *PortAssignCOLUMN*
Let S_1 and S_2 be the no. of elements in Columns 1 and 2
Let $T = \lceil (S_1 + S_2)/m \rceil$
$--T$ is the max #elements assigned to each port
Let $C_1 = S_1 \bmod T$ and $C_2 = S_2 \bmod T$
Let $D_1 = \lfloor S_1/T \rfloor$ and $D_2 = \lfloor S_2/T \rfloor$
Assign first $D_1 \times T$ elements in col 1 to the first D_1 ports (i)
Assign first $D_2 \times T$ elements in col 2 to the next D_2 ports (ii)
If $C_1 + C_2 > T$, (iii)
  $--$ 2 ports, P_{m-2} and P_{m-1} are free
  Assign the last C_1 elements in Column 1 to port P_{m-2}
  Assign the last C_2 elements in Column 2 to port P_{m-1}
else *$--$ $C_1 + C_2$ elements fit into one port P_{m-1}*
  Assign the last C_1 elements in Column 1 and
  the last C_2 elements in Column 2 to port P_{m-1}

Condition (iii) handles the problem raised in *Observation 2*. For the example of Figure 3(b), we have: $S_1 = 5; S_2 = 2; T = \lceil (5+2)/3 \rceil = 3$, i.e., $C_1 = 5 \bmod 3 = 2; C_2 =$

2 mod 3 = 2; $D_1 = \lfloor 5/3 \rfloor = 1$; $D_2 = \lfloor 2/3 \rfloor = 0$. Therefore, the first $D_1 \times T$, i.e., $1 \times 3 = 3$ elements (e_0, e_1, and e_2) are assigned to port P_0 (statement (i) of procedure *PortAssignCOLUMN*). Since $D_2 = 0$, no assignment takes place in statement (ii). Now, $C_1 + C_2 = 2 + 2 = 4 > 3$, so we assign the last $C_1(= 2)$ elements of column 1 to port P_1 and the last $C_2(= 2)$ elements of column 2 to port P_2.

Row- or column-major mapping is also preferable if sufficient number of ports are available to access the elements in parallel. For the access pattern in Figure 2, if $m = 4$, then we could assign one element to each port and select row-major mapping. This ensures that there is no maximal transition during an inner loop iteration. Further, since the pattern moves row-first, each port incurs a minimal transition, since it always accesses an element located near the previous one, in the same row. Similarly, in Figure 3(b), if $m = 5$, we could have the assignment – $P_0 = \{e_0\}, P_1 = \{e_1, e_5\}, P_2 = \{e_2, e_6\}, P_3 = \{e_3\}, P_4 = \{e_4\}$ and select row-major mapping (if the access pattern moved row-first), which would lead to all minimal transitions (note that $\{e_1, e_5\}$ are in consecutive locations, and so are $\{e_2, e_6\}$).

The following computation enables us to check if we have sufficient number of ports available to match the situation described in the previous paragraph. If there is to be no maximal transition in an inner loop iteration, we need that each port be assigned elements within the same row (assuming we have selected row-major mapping). Therefore, if S_i is the number of elements accessed from row i, we need $\lceil S_i/T \rceil$ ports for this tile row. Thus, the total number of ports required $= \sum_i \lceil S_i/T \rceil$. That is, a zero maximal transition assignment is possible if $\sum_i \lceil S_i/T \rceil \leq m$. If the performance is to be *better than or equal to* tile-based mapping, we should have no more than 1 maximal transition, i.e., we can allow: $\sum_i \lceil S_i/T \rceil \leq m + 1$. [2] The function *SufficientPortsAvailable* below returns ROW (COLUMN) if sufficient number of ports are available for ensuring ≤ 2 maximal transitions per iteration for a row-major (column-major) mapping. If this is not possible, it returns TILE.

function *SufficientPortsAvailable*
-- Determining the Best Mapping Scheme
Let the tile dimensions be M rows $\times N$ columns
Let $S_1 \ldots S_M$ be the no. of elements accessed from rows $1 \ldots M$ of tile
Let $T = (\sum_{i=1}^{M} S_i)/m$
if $\sum_{i=1}^{M} \lceil S_i/T \rceil \leq m + 1$ **then**
    return ROW
Let $U_1 \ldots U_N$ be the no. of elements accessed from columns $1 \ldots N$ of tile
if $\sum_{i=1}^{N} \lceil U_i/T \rceil \leq m + 1$ **then**
    return COLUMN
else
    return TILE

On applying function *SufficientPortsAvailable* to the tile of dimensions 5×4 shown in Figure 3(a), with $m = 5$, we observe that $T = \lceil 17/5 \rceil = 4$; $\sum_{i=1}^{5} \lceil S_i/4 \rceil = 5 \leq 5 + 1$. Hence, row-major is selected as the mapping strategy. This is clearly advantageous, since each port would now be assigned elements from the same row, leading to all minimal transitions in any single inner loop iteration of the behavioral loop.

We now extend Heuristic 1 (Section 3) to select a mapping style and make the port assignments for mapping a single array into a single physical memory module with multiple ports. Heuristic 2 below incorporates the conditions discussed above, combines them with the original conditions for tile-based mapping (in Heuristic 1), and invokes the port assignment functions after selecting the mapping scheme.

Heuristic 2
-- Selection of Mapping Strategy and Port Assignment
Let the inner loop index be i and outer loop index be j
Extract the basic repeating shape from the access patterns
Determine the enclosing rectangle R.
Let i-dimension of R be L_i and j-dimension be L_j
Let A = TRUE if increment of $j = L_j$ else FALSE
if (not A) **and** ((*SufficientPortsAvailable* = ROW) **or** ($L_j \leq 2$)) **then**
    Select Row-major mapping
    PortAssignROW
else if (not A) **and** ((*SufficientPortsAvailable* = COLUMN) **or** ($L_i \leq 2$)) **then**
    Select Column-major mapping
    PortAssignCOLUMN
else
    Select Tile-based mapping
    PortAssignTILE

Condition ($L_j \leq 2$) in Heuristic 2 ensures that Row-major mapping is selected if the access pattern has two rows, and moves *row-first*, and the condition ($L_i \leq 2$) ensures that column-major mapping is selected only if the access pattern has two columns and moves *column-first*. The selection rule assumes that only one array is accessed. The function *SufficientPortsAvailable* will need slight modification to handle the case of multiple arrays. The summation $\sum_{i=1}^{M} S_i$ will be over elements of all arrays accessed in the loop iteration, instead of over one array.

We have thus formulated and solved the problems of selecting a mapping scheme for arrays into a multi-ported memory, and finding an efficient assignment strategy for mapping the array elements accessed in the inner loop of a behavioral specification into ports of the memory, with the objective of minimizing total number of transitions on all the address buses.

[2] As explained earlier, a second maximal transition is automatically incurred.

5. Experiments and Results

Experiments reported in [6] showed that Heuristic 1, which selects an appropriate array mapping scheme, results in a transition count reduction of 27–63% over the simple row-major mapping used by most compilers and synthesis tools. The experiments we report here confirm that the memory mapping and port assignment performed by Heuristic 2 results in significant power reduction through reduced cumulative transition count on the multiport memory buses. The transition count measurements were performed on several benchmarks from the Image Processing domain [5]. All the examples involve manipulation of two-dimensional arrays. Our experiments were performed using the value of 1000×1000 as the dimensions for the arrays. We present a comparison of the transition counts observed during the execution of the examples, using two configurations – (i) a single physical memory with 2 ports and (ii) a single physical memory with 3 ports.

Table 1 shows the experimental results for dual-port and 3-port memories, with read and write accesses possible on both ports. Column 1 shows the benchmark examples on which we performed our experiments. For a dual-port memory: Column 2 shows the off-chip transition counts incurred when a simple port assignment strategy that does not consider access patterns is used, and Column 3 shows the corresponding counts for our port mapping strategy. Column 4 shows the percentage reduction in transition count on the memory address buses. Columns 5, 6, and 7 show the corresponding results for a 3-port memory. The reductions

	2-Port Memory			3-Port Memory		
Example	Smpl	Our	%rd	Smpl	Our	%rd
Compress	13394	3992	70.2	5988	3992	33.3
GSR	35641	35575	18.8	32689	18310	44.0
Laplace	30356	28330	6.7	26332	22790	13.5
Linear	59.25	13.99	76.4	12.00	12.00	0
Lowpass	23986	16738	30.2	16594	7964	52.0
SOR	44121	9900	77.6	49657	14331	71.1
Wavelet	5.71	5.71	0	3.92	3.78	3.4

Table 1. Transition Count Comparison for Port Assignment Strategies ($\times 1000$)

observed in Columns 4 and 7 demonstrate show that our memory mapping and port assignment strategies result in significant power savings (transition count reduction of upto 77.6%) over a simple random port assignment. Moreover, the assignment has no overhead in terms of area or performance of the system. Note that we always ensure a port assignment that satisfies the maximum throughput in the memory accesses, so there is no delay overhead. The area (and on-chip transition count) overhead due to the tile-based mapping strategy were shown to be negligible in [6][3] . In both experiments, there are cases where there is no reduction in transition count (0% decrease was observed in one example each in Columns 4 and 7). These are the cases where the random assignment strategy performed the same port assignment as Heuristic 2.

6. Conclusions

We described a technique for mapping of behavioral arrays to memory location, and port assignment of memory accesses in multiport memories, targetting low power through transition count reduction on the memory address buses. Our experiments showed that the power dissipation during memory accesses, as measured through off-chip signal transition count on the memory address bus could reduce by upto 77% over a straightforward port assignment scheme. The concept of maximal and minimal transitions used in Section 3 has performance implications in DRAMs, apart from power. A minimal transition roughly corresponds to accesses in the same memory page (faster access), while a maximal transition indicates accesses from different pages (slower access). Thus, a reduction in maximal transition count also leads to reduced access time.

The mapping strategy we have described works only for array references in loops of the form $a[i \pm k]$, where i is the index and k is a constant. The formulation and solution of the problem to handle multi-dimensional arrays remains exactly the same.

References

[1] H.B. Bakoglu, "Circuits, Interconnections, and Packaging for VLSI," Addison-Wesley, 1988.

[2] A. Dasgupta and R. Karri, "Simultaneous Scheduling and Binding for Power Minimization During Microarchitecture Synthesis," *Intl. Symp. on Low Power Design*, Dana Point, CA, April 1995.

[3] R. S. Martin and J. P. Knight, "Optimizing Power in ASIC Behavioral Synthesis," *IEEE Design and Test of Computers*, June 1996.

[4] E. Musoll and J. Cortadella, "Scheduling and Resource Binding for Low Power," *International Symposium on System Synthesis*, Cannes, September 1995.

[5] P. R. Panda and N. D. Dutt, "1995 High Level Synthesis Design Repository," *International Symposium on System Synthesis*, Cannes, September 1995.

[6] P. R. Panda and N. D. Dutt, "Reducing Address Bus Transitions for Low Power Memory Mapping," *European Design and Test Conference*, Paris, March 1996.

[7] M. R. Stan and W. P. Burleson, "Limited-weight codes for low-power I/O," *IEEE Intnl. Workshop on Low Power Design*, Napa, CA, April 1994.

[8] S. Wuytack, F. Catthoor, F. Franssen, L. Nachtergaele and H. De Man, "Global communication and memory optimizing transformations for low power systems," *IEEE International Workshop on Low Power Design*, Napa, CA, April 1994.

[3] An example implementation of tile-based mapping is discussed in [6]