An energy-conscious algorithm for memory port allocation

Preeti Ranjan Panda
Dept. of Computer Science and Engineering
Indian Institute of Technology, Delhi
Hauz Khas, New Delhi 110016, India

Lakshmikantam Chitturi
Teradyne, Inc.
880 Fox Lane, San Jose, CA 95131, U.S.A.

ABSTRACT

Multiport memories are extensively used in modern system designs because of the performance advantages they offer. The increased memory access throughput could lead to significantly faster schedules in behavioral synthesis. However, they also have an associated area and energy penalty. We describe a technique for mapping data accesses to multiport memories during behavioral synthesis that results in significantly better energy characteristics than an unoptimized multiport design. The technique consists of an initial colouring of the array access nodes in the data flow graph based on spatial locality, followed by attempts to consecutively access memory locations with the same colour on the same port. Our experiments on several applications indicate a significant reduction in address bus switching activity, leading to an overall energy reduction over an unoptimized design, while still maintaining a performance advantage over a single-port solution.

1. INTRODUCTION

Power-aware optimization techniques at the system level form an essential feature of modern low-power embedded systems. Very often, there exists an important trade-off between a performance-optimized and a power-optimized design; the two respective optimal design points are not always the same. A good example of this behavior is observed in systems where the clock is slowed down to decrease power dissipation. However, power and energy awareness can also be explicitly built into the system-level synthesis algorithms used to generate design implementations. In this paper we study the impact of multiport memories and their associated port allocation strategy on the performance and energy of synthesized designs. Although multiport memories generally lead to better performance, they typically incur a significant area and energy overhead (up to 100% and 75% for area and power respectively for the technology we studied). However, with an energy-conscious memory port allocation algorithm, it is possible to minimize the energy overhead while still retaining the performance advantage over a single-port memory solution.

As demonstrated in works such as [1], design considerations such as power and energy can be tightly integrated into the inner loop of typical high-level synthesis tasks such as scheduling. A design optimization problem involving the performance and energy coordinates can be phrased in one of two forms:

1. **Optimize for performance** – and while retaining this level of performance, minimize the energy.

2. **Optimize for energy** – and while retaining this level of energy, maximize the performance

In this paper, we present algorithms for solving both forms of the optimization problem.

The memory subsystem has long been recognized as a serious bottleneck in terms of performance, area, and power dissipation in embedded systems [2, 3]. Memories tend to be significant sources of power dissipation because they are associated with long, high capacitance wires, both inside the memory module (in the form of long word-lines and bit-lines) as well as outside the module (in the form of address and data buses). Consequently, many optimization efforts at reducing memory power have targeted the transition count on the memory address and data buses – specifically, address buses since the pattern of addresses accessed is usually known in advance [4, 5]. Approaches to address bus switching reduction include data placement [4] and bus encoding [3].

Minimizing memory power is also closely related to performance optimizations performed by compilers [6]. Standard optimizations such as induction variable elimination, loop fusion, loop interchange, etc., that result in fewer memory access also reduce memory power as a direct consequence. Loop optimizations that improve cache performance also reduce power since cache misses impact not only performance, but also power. Since the actual cache configuration can be customized in embedded systems, a number of research efforts have addressed the problem of determining an application-specific memory hierarchy that optimizes performance and power [7, 8].

Multiport memories have been incorporated into traditional behavioral synthesis algorithms by treating the ports as independent schedulable resources [9, 10]. These algorithms focus on performance alone and do not study the energy implications. In [11], power optimization on multiport memories is applied to a limited set of applications where the data can be divided into tiles. In this paper, we outline memory port assignment algorithms that can be tightly integrated into the scheduling phase of behavioral synthesis.

2. ILLUSTRATIVE EXAMPLES

Consider the following simple section of code to be synthesized into hardware:

```
int a[100];
⋮
for i = 0 to 99
   a[i] = a[i] + n
```

where a is to be stored in memory; n is a variable; memory reads and writes require one clock cycle; addition and comparison require one cycle. An example 3-cycle schedule (Schedule A) of the loop body is shown in Figure 1(a). A single port memory is sufficient for this implementation. A loop pipelining transformation shown in Figure 1(b) can optimize the schedule to execute in only 2 cycles (Schedule B) – here, the addition on the current item proceeds in parallel with reading the next element, leading to a 33% shorter schedule. However, a further improvement is possible if we use a dual-port memory, as shown in Figure 1(c). Since there are 2 ports, the computation on the current data can proceed in parallel with writing the previous element and reading the next element, effectively requiring only one cycle, leading to a 66% performance improvement in the steady state (Schedule C).

However, a study of the memory address bus switching activity of the three schedules yields different results. In Schedule A, the sequence of addresses on the memory address bus is:
0,0,1,1,2,2,3,3,4,4,...,99,99

0-7803-7607-2/02/$17.00

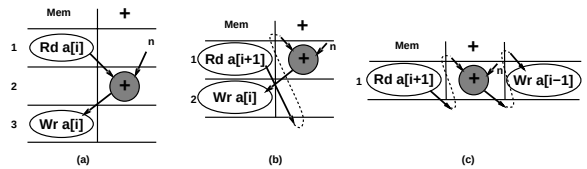

Figure 1: (a) Example Schedule – A (b) Pipelined Schedule – B (c) Schedule using Multiport Memory – C

In Schedule B, the sequence of addresses is:
0,{1,0,2,1,3,2,4,3,5,4,...,98,97,99,98},99 where braces denote accesses in the pipelined loop. This is a more expensive alternative in terms of energy dissipation because the additional switching activity on the memory address bus amounts to almost 100%.

Finally, in Schedule C, the address sequences are:
Port 1: 0,1,2,3,...,99
Port 2: 0,1,2,3,...,99
Clearly, this leads to twice the number of address bits transitioning compared to Schedule A, since now the address buses of both ports are switching, as opposed to only one. When synthesis for low power/energy is an important design objective, the aggressively performance-oriented optimizations may actually result in inferior power characteristics. However, the unoptimized design from a performance point of view is not always power-optimal. The following example illustrates this point.

```
for i = 0 to 99
   a[i] = b[i] + n
```

The unoptimized, pipelined, and multiport/pipelined schedules are shown in Figure 2. In this case, array a is located at addresses 0...99 and b occupies 100...199. We have the sequence of addresses as follows.
Schedule A: 100, 0, 101, 1, 102, 2,...,199,99
Schedule B: 100,{101, 0, 102, 1, 103, 2,...,199,98},99
Schedule C: Port 1: {0,1,2,3,...,97},98,99
Schedule C: Port 2: 100,101,{102,103,...,199}

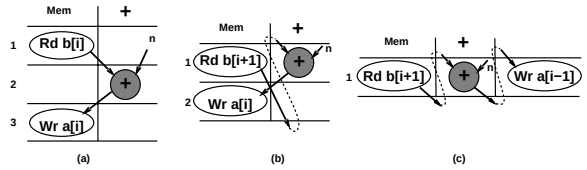

Figure 2: (a) Example Schedule – A (b) Pipelined Schedule – B (c) Schedule using Multiport Memory – C

In this case, Schedule B results in 9 % more switching than Schedule A, but Schedule C results in 57 % less switching than Schedule A, possibly making Schedule C a viable candidate from both the performance as well as energy points of view. The dual-port memory configuration actually led to minimum address bus switching because spatial locality could be exploited resulting in sequential accesses. Moreover, addresses being sequential in Schedule C means that we can use appropriate encoding techniques such as Gray code, T0, etc. [3] to further reduce power dissipation both on the memory interface and in the memory module itself.

The measured reduction in memory address bus switching does not translate to an equivalent reduction in the total system energy. In order to determine the actual energy dissipation figures for the different designs represented by the three schedules of Figure 2(a), (b), and (c), we synthesized them using the commercial synthesis tool Synopsys SystemC Compiler and a 0.18μ IBM ASIC library of components. We used the Synopsys Design Power utility to measure the power dissipation of the resulting circuit. In order to understand the actual impact of address bus switching, we divided the total system energy into the following components:

Interconnect Energy – energy dissipated due to switching of the (relatively high capacitance) data and address buses, and other nets in the design.

Memory Internal energy – energy dissipated inside the memory module during the READ and WRITE accesses. This includes the dissipation at the address decoders, word-lines, bit-lines, address latches, etc.

Datapath and FSM energy – energy dissipated in the cells of the datapath and finite state machine generated from synthesizing the application.

The energy dissipated in each of the above components for the three schedules of Figure 2, A, B, and C, is indicated in Figure 3. The difference in the interconnect energy is mainly due to the varying switching activity on address buses reported earlier.

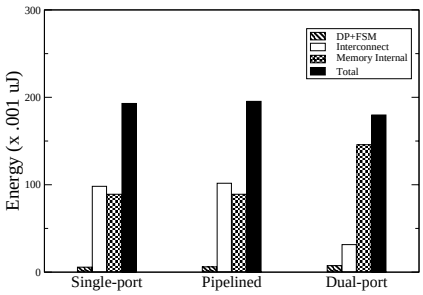

Figure 3: Energy dissipation comparison

The internal memory energy of dual port memories is, in general, larger than that of a single port memory because each memory cell drives a larger capacitive load; the address decodes, latches, and other circuitry are duplicated; etc. In the 0.18μ IBM ASIC library, the dual port memory had a 75% power dissipation overhead compared to a single port memory of the same size.

The energy dissipated in the datapath and FSM cells is minimum for case A, and is 21% and 31% more in cases B and C. This is expected since the pipelining leads to slightly more complex control and address generation circuitry. The energy dissipated in the Datapath and FSM cells is a relatively smaller fraction of the total energy (about 6%). This is obvious in a small example with minimal computation, but the trend of computation related energy begin dwarfed by memory related energy is also observed in the wider class of data-intensive applications, and is independently reported in other studies too. The interconnect and internal memory are the more significant contributors to the total system energy. This is a very important observation and forms the motivation for our research. Dual-port memories incur a higher internal memory energy, but it may be possible to reduce, or even (in some circumstances such as this example) completely negate the overhead by a more efficient addressing mechanism. More importantly, if a dual-port memory has already been chosen for a design due to performance considerations, the techniques presented in the next section help achieve an energy-efficient allocation of memory ports to data so that superior energy characteristics can be obtained for the same or similar performance levels.

3. ENERGY-AWARE SYNTHESIS

The memory energy-aware synthesis problem involves the generation of a schedule for a behavioral specification that reduces energy by minimizing switching activity on the memory address bus. The primary optimization criteria may be performance or energy. We address both cases, which require different solution approaches.

3.1 Colouring of Memory Access Nodes

The scheduling of memory accesses is preceded by a *colouring* phase where we identify those accesses in a loop body that have spatial locality and are likely to cause only a small number of address bit transitions when accessed consecutively, e.g., $a[i]$ and $a[i+1]$. Prediction of spatial locality is done by examining whether the array indices differ by a small constant (constrained to be ≤ 4). Standard array index analysis techniques are used for this purpose. An example of colouring memory accesses in a loop is shown in Figure 4. $a[i][j]$ and $a[i][j+1]$ are the same colour because they are spatially close; $a[i][j]$ and $a[j][i]$ are coloured different.

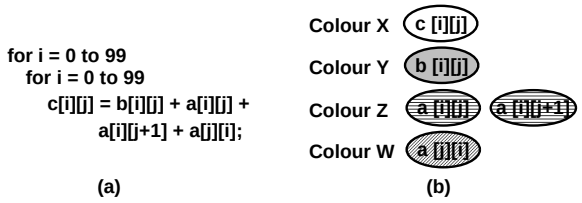

Figure 4: (a) Example loop (b) Colouring

3.2 Scheduling Memory Accesses Primarily for Performance

When the primary objective is performance, we employ energy-optimization as a post-processing step that modifies the generated schedule by assigning the ports to memory accesses in a energy-efficient way without changing the schedule length. At every cycle in the schedule, we attempt to assign each memory access to that port whose previous access had the same colour. A simple example with an initial schedule is shown in Figure 5(a). The memory accesses are grouped into two colours black and white. The white node is reassigned to port P2 because a white node was accessed in the previous access to P2. In the third control step, the black node is reassigned to P1 (Figure 5(b)). This results in a energy-optimal schedule where successive memory accesses on each port have spatial locality and hence results in the minimum number of address bit transitions. The schedule length remains unchanged because the swaps are always *horizontal*, never *vertical*. We omit the detailed algorithm due to lack of space.

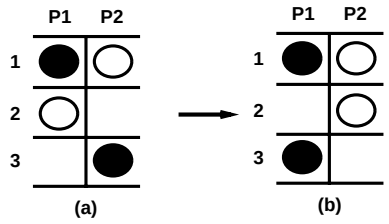

Figure 5: (a) Initial schedule (b) After port reassignment

3.3 Scheduling for Low Energy

If the schedule length is allowed to be modified, i.e., performance is allowed to be sacrificed at the expense of energy, then more aggressive energy-optimized schedules are possible. Our strategy in this case, is to perform the port assignment up front as a *pre-processing* step instead. We first assign the ports to memory accesses using the colour information to ensure that each port is assigned memory accesses of the same colour as far as possible, resulting in spatial locality being preserved to the maximum extent. Algorithm PREASSIGN_PORTS outlines the strategy.

Algorithm PREASSIGN_PORTS (G: DFG, n: #ports)
for all loop bodies L
  Let colours 0..m-1 be used in this loop body
  Sort the colours in decreasing order of their access frequency in L into array c[0]..c[m-1]
  for ports i = 0..n-2
    Assign colour c[i] to port i
  Assign remaining colours c[n-1]..c[m-1] to port n-1

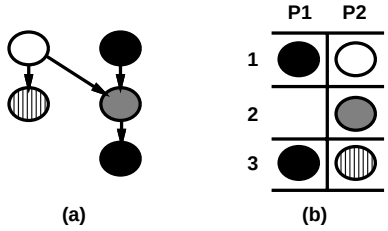

Figure 6: (a) DFG (b) Port assignment and schedule

For each loop, we attempt to assign only one colour to each port for all but one port, considering each colour in decreasing order of its access frequency in the loop body. We assign the remaining colours to the final port. This configuration is energy-efficient because spatial locality is violated the minimum number of times. The violation takes place when colours change on any port. By assigning one colour to each of m-1 ports, we ensure that colours never change for those ports. The assignment is illustrated in Figure 6. Since the black coloured memory access occurs more frequently (twice), it is assigned to port P1; others are assigned to P2. This ensures spatial locality (no colour change through all loop iterations) in P1, and the violations are restricted to P2 (3 per loop iteration). Note that if, instead, two colours each were to be assigned to P1 and P2 respectively, then there would be at least 4 violations per iteration. The assignment strategy may have a negative impact on the schedule length. However, the strategy helps identify important design points on the performance-energy trade-off curve. The final selection decision can be made by the designer.

The port resources are now bound to the memory accesses and scheduling can begin. List scheduling works by invoking a *priority function* to determine the next node to be scheduled among the set of *schedulable nodes* in the current cycle. A common priority function is the *mobility* of operations, but this targets a performance-optimized design. Our modification to the priority function that attempts to schedule consecutive nodes of the same colour to a port is summarized in function NEXT_NODE. *PrevColour[p]* keeps track of the colour of the previous node scheduled on port p. If a schedulable memory access node with the same colour as *PrevColour[p]* is found for any p, then we select that node. If no such node is found and there is a schedulable non-memory operation ($Y \neq \phi$), we use the mobility of that operation to determine the selected node. This serves to defer any unfavourable memory port assignment until absolutely necessary. But if $Y = \phi$ we must switch colours on one port (port n-1, as indicated in PREASSIGN_PORTS). Ideally, we would like to select a node such that future memory access nodes

of the same colour would get clustered together, but this can be computationally expensive. To prune the search space, we select the node for which the next DFG node with the same colour is at a maximum depth, to allow for the possibility of clustering of other colours later on. Function NEXT_NODE omits some details (initial condition, update of *PrevColour*) due to lack of space.

Function NEXT_NODE
X = Set of schedulable memory nodes
Y = Set of schedulable non-memory nodes
for all ports p = $0 \ldots n-1$
  for all $x \in X$
    if Colour(x) == PrevColour[p]
      return x; // matching colour found
if $Y \neq \phi$
  return node $y \in Y$ with minimum mobility
else // forced to switch colour on port n-1
  return $x \in X$ for which depth of DFG node with
    Colour(x) is maximum

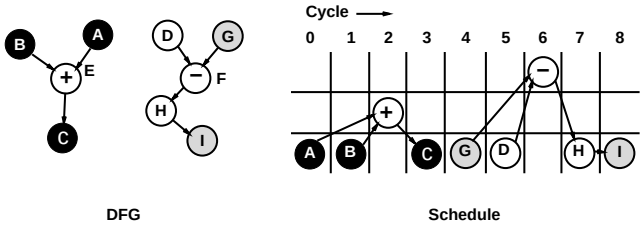

Figure 7: Operation of NEXT_NODE

Figure 7 shows an example of how NEXT_NODE selects candidate nodes for scheduling. Sets of memory access nodes of the same colour are: $\{A, B, C\}$, $\{D, H\}$, and $\{G, I\}$. Suppose the last node scheduled on the single port memory is A. We have $X = \{B, D, G\}$. B is selected because it is the same colour as A. Now, we have $X = \{D, G\}, Y = \{E\}$. Since there is a colour mismatch at the port with nodes D and G, we select E. Now we have $X = \{C, D, G\}$. C is chosen to match the colour at the port. In the next cycle, we have $X = \{D, G\}$. Note that node I (same colour as G) is at a greater depth than H (same colour as D). Thus, NEXT_NODE returns node G to allow for the possibility of D and H being consecutive on the port (selecting D here would lead to an additional colour switch).

3.4 Loop unrolling and pipelining

Loop transformations such as unrolling tend to increase the number of consecutive memory accesses of the same colour due to the concatenation of array accesses of different iterations. Similarly, loop pipelining improves the throughput by keeping more resources (memory ports) busy in each cycle. The post-processing and preprocessing steps, and the new priority function discussed earlier are directly incorporated into scheduling techniques that involve these loop transformations, since these power optimizations are independent and always applicable.

4. EXPERIMENTS

We studied the effect of our memory port assignment algorithms by performing experiments on several loop- and data-intensive applications involving array accesses and computations. Since most practical systems use either single port (SPRAM) or dual-port (DPRAM) memories, we conducted our experiments on these two port types, although the algorithms themselves are general enough to handle a larger port count. We studied the following 3 cases:

Case A: Performance-optimized schedule (no power optimization)

Case B: Performance-optimized schedule with power optimizations applied as a post-processing step (Section 3.2)

Case C: Power optimizations applied as a pre-processing step before and during scheduling (Section 3.3)

For each of the above three cases, we performed experiments on both single- and dual-port memories. We used the 0.18μ IBM ASIC library in our experiments. The procedure consisted of the following steps for each design example: (1) Behavioral and logic synthesis using the Synopsys SystemC Compiler and Design Compier; (2) Logic simulation of the gate-level netlist with Cadence NC-Verilog simulator; and (3) Energy calculation from the resulting *Activity file* and the ASIC library using the Synopsys Design Power simulator.

4.1 Detailed Example

We discuss in detail the experimental results for one important example, the Fast Fourier Transform (*FFT*) algorithm, which is a popular routine used in several Digital Signal Processing applications. We assume that memory accesses, additions, and subtractions require one cycle, and multiplication requires 2 cycles. In Case A (performance-optimized) the dual-port memory causes a significant reduction in the schedule length, resulting in 30% better performance. Figure 8 shows the schedules for single- and multiport memories for case B, when power optimizations are applied as a post processing step. The single-port schedule is the same as case A, but the dual-port schedule is modified by interchanging the port assignments of two memory accesses. The memory access nodes are grouped into two colours, and it is desirable to assign the same colour to successive accesses from the same port. $x[i].re$ and $x[i].im$ are grouped into the same colour because the fields of the struct have spatial locality.

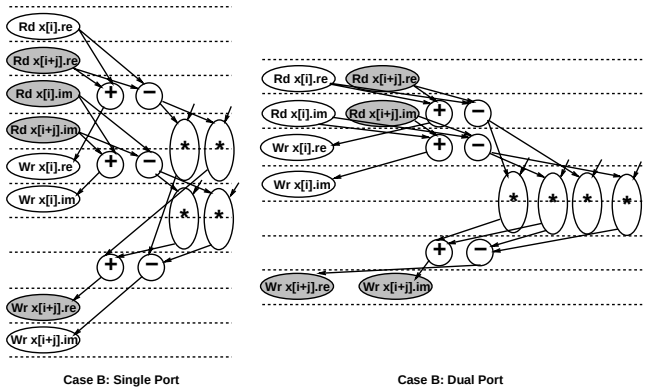

Figure 8: Case B: Schedules for single and multiport memory

Figure 9 shows the schedules generated by case C. Note that the single-port schedule is longer, but the $x[i]$ elements (coloured white) are clustered together, which results in reduced transitions on the address bus. Finally, the case C dual-port schedule is 8 cycles long; energy considerations caused our port assignment algorithm to assign each colour to a different port. Overall, the power-optimized dual-port memory configuration results in a 20% better performance than the unoptimized SPRAM-based design.

The energy dissipation characteristics of the six schedules discussed above are summarized in Figure 10. For each configuration, we have indicated: (1) the energy spent in datapath and FSM cells; (2) interconnect energy; (3) memory internal energy; and (4) total

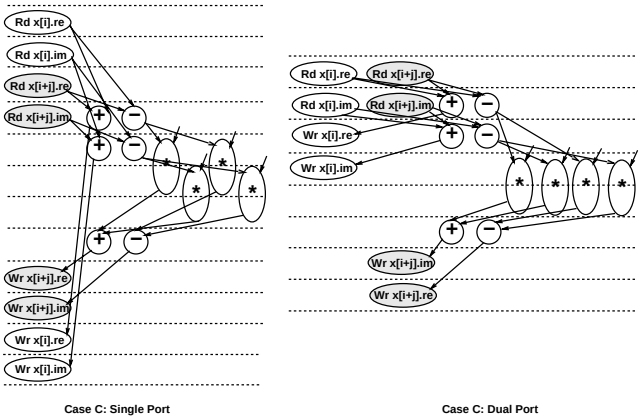

Figure 9: Case C: Schedules for single and multiport memory

energy. For the three single port configurations, the energy dissipation in the various components are almost the same.

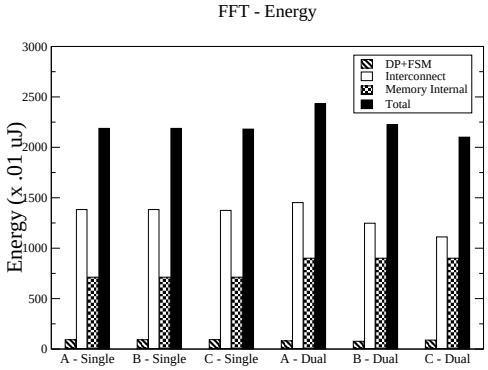

Figure 10: Energy dissipation comparison for *FFT*

The DPRAM offers an interesting energy comparison. Case A results in higher interconnect energy as well as higher internal memory energy due to the DPRAM. The total energy is 11% higher than the SPRAM-based design. However, our energy optimization, when applied to the DPRAM based design, results in lower interconnect energy, which offsets the increased internal memory energy of the DPRAM. Cases B and C result in 9% and 14% less energy than case A (unoptimized) for the DPRAM. In fact, case C actually results in marginally *lower* energy than even the SPRAM-based design.

4.2 Summary of Results

We report the performance and energy results on three other examples: *SOR*, *Dprod*, and *Planckian*. *SOR* is the successive over-relaxation algorithm often used in image processing; *Dprod* is the dot product example from DSPStone benchmark set; and *Planckian* is a scientific computing benchmark from the Livermore Loop set. For *SOR*, the optimized DPRAM-based design (C-Dual) results in 42% performance improvement over the SPRAM design, while maintaining comparable energy dissipation. For *Planckian*, the dual-port memory improves performance by 40%, while the energy-optimization in C-Dual causes a 7% overhead. In the *Dprod* example, we observe a 33% performance improvement from using DPRAMs, with the total energy overhead being 3%.

4.3 Discussion

The most important observation from our experiments is that an efficient assignment of memory ports to data accesses usually ensures a reduction in energy dissipation of synthesized designs based on multiport memories. It is important to note that this work does not attempt to demonstrate that multiport memory-based design can always yield lower overall energy than single port-based ones. The single port numbers were presented as a reference/baseline to perform comparisons. Our experiments show that the energy of the optimized dual-port configurations (C-Dual) was comparable to that of SPRAM-based ones. However, performance considerations might lead designers to choose DPRAMs in system designs. Once this decision is made, our port assignment algorithms help reducing the energy dissipation significantly, as the A-Dual vs. C-Dual numbers clearly show.

The energy consumed in the datapath and FSM cells is a small fraction of the interconnect-related energy, confirming our motivation for this work. Note that it is not necessary to have multiport memories to benefit from the algorithms in Section 3.3. The energy optimization strategy can be useful even with single-port memories. In such cases, the problem is not assignment of data to ports, but the appropriate re-ordering of data accesses to reduce address bus switching (this is incorporated into the NEXT_NODE function. Finally, The area overhead of the multiport memories will usually lead to larger overall design area (e.g., area overhead was 37% in *FFT*).

5. CONCLUSION

We presented algorithms to reduce memory address bus switching energy by an efficient allocation of memory ports to behavioral array accesses combined with the re-ordering of data accesses by incorporating energy optimizations into the scheduling phase of behavioral synthesis. The optimizations apply to both single- and multiport memories. Frequently, multiport memories can offer significant performance advantages in system designs because of the increased data throughput, but lead to overheads in area and energy. We have shown that our strategy can help reduce some of this overhead in the overall energy dissipation of the system, while still retaining most of the performance advantages offered by multiport memories.

6. REFERENCES

[1] E. Musoll and J. Cortadella, “High-level synthesis techniques for reducing the activity of functional units,” *ISLPD*, 1995

[2] P. R. Panda et al., Data and memory optimization techniques for embedded systems, *TODAES*, Apr. 2001.

[3] L. Benini and G. De Micheli, “System level power optimization: Techniques and tools,” *TODAES*, Apr. 2000.

[4] P. R. Panda and N. D. Dutt, “Low-power memory mapping through reducing address bus activity,” *TVLSI*, Sept. 1999.

[5] L. Benini et al., “Power optimization of core-based systems by address bus encoding,” *TVLSI*, Dec. 1998.

[6] M. Kandemir et al., “Influence of Compiler Optimizations on System Power,” *DAC*, June 2000.

[7] P. R. Panda et al., “Local memory exploration and optimization in embedded systems,” *IEEE TCAD*, Jan. 1999.

[8] W.-T. Shiue and C. Chakrabarti, “Memory exploration for low power embedded systems,” *DAC*, June 1999

[9] M. Balakrishnan et al., “Allocation of multiport memories in data path synthesis,” *IEEE TCAD*, Apr. 1988.

[10] T. Kim and C. L. Liu, “Utilization of multiport memories in data path synthesis,” *DAC*, June 1993.

[11] P. R. Panda and N. D. Dutt, “Behavioral array mapping into multiport memories targeting low power,” *Intl. Conf. on VLSI Design*, Jan. 1997.