Low Energy Memory and Register Allocation Using Network Flow

Catherine H. Gebotys
Department of Electrical & Computer Engineering
University of Waterloo,
Waterloo, Ontario. N2L 3G1 Canada

Abstract

This paper presents for the first time low energy simultaneous memory and register allocation. A minimum cost network flow approach is used to efficiently solve for minimum energy dissipation solutions in polynomial time. Results show that estimated energy improvements of 1.4 to 2.5 times over previous research are obtained. This research is important for industry since energy dissipation is minimized without requiring an increase in cost.

1.Introduction

Recently low power systems design has gained significant attention largely due to demands from the portable electronics industry. However system design for low power is also very important for other industries such as automotive, telecommunications, information technology, etc... This is due to the fact that low power designs can offer significant reductions in system packaging costs and improvements in system reliability[1]. For example in multimedia applications low power is believed to be crucial[2]. This includes audio and video algorithms which process large amounts of data, performing computations in real time. High level design issues, such as determining the number of external memory accesses, the number of accesses and sizes of internal register files and memories, the amount of precomputation, and the scheduling of operations to meet real time constraints, are seen as having a significant impact on the final system's cost, performance and power dissipation. Clearly there is a need to study high level techniques for designing low power, high performance systems.

Tools for low energy design need to be developed to support the mapping of applications into low energy system architectures. Although it is important to explore energy versus cost tradeoffs, for many cost sensitive markets, such as multimedia, new techniques need to be developed which provide low energy without an increase in cost or decrease in performance. The low energy system design problem involves several interdependent subproblems including scheduling and allocation. However it is critical to consider memory design, since it is well known that external memory accesses will dissipate significantly more energy than using on-chip memory or cache which in turn may dissipate more energy than performing some computation (such as addition) on the chip. Typical compiler techniques for map- ping variables to registers or variables to on-chip or off-chip memories have concentrated on fast compile times and performance [6,7]. There is a need to perform this mapping to optimize energy without any degradation of performance. It is known that these early design decisions have a significant impact on the final embedded system.

2.Problem Description and Related Work

The following problem, problem 1 given below, is an important part of the low energy system design problem that will be studied in this paper. For simplicity let us assume that an algorithm to implement the application has already been assigned based upon accuracy required, low energy implementation, etc.. The algorithm is given as a partially ordered list of code operations. For the problem definition below we assume that there is one chip with onchip register file, onchip memory and external (offchip) memory.

Problem 1.
Assume we are given an initial schedule of operations, represented by an ordered list of operations. Using the schedule, each data variable can be represented by a lifetime which is an interval of time, represented by two values of time. The first value denotes the time when the data variable is defined (the write time) and the second value is the time the data variable is used and not needed anymore (the read time). This lifetime may be broken up into a number of intervals, called split[7] lifetimes, (for example if the data variable is read more than once or the data variable is spilled into memory[7,10]). Assume we have a register file with *R* registers onchip and a (onchip and/or offchip) memory component allowing memory accesses every *c* control steps (or *c* clock cycles). For every *c* control steps, determine whether each data variable defined over this time is to be assigned to registers in the register file or remains in memory storage.
The objective is to minimize the total energy dissipation of the storage components.

One technique for reducing energy that has been previously researched is to reduce the average capacitance of the system being switched[9,3]. Another popular technique is to reduce energy by lowering the supply voltage, even though the the speed of the chip decreases. This technique, called voltage scaling, [15,13,1] increases the parallelism in the system (proportional to cost) so that time constraints are met and all computations can be performed with a lower supply voltage. The linear increase in capacitance (from added parallelism) is offset by the quadratic decrease in voltage, thus reducing the energy dissipated[3]. High level capacitance models[3,7,9,15] for various functional units (ie. multipliers, adders, etc), register files, interconnections, and controllers have also been researched.

Other studies identified that the energy dissipation of memory accesses and offchip transfers can be very significant[14]. For example a 16 bit multiplication, onchip memory read, onchip memory write, and offchip transfer required 4, 5, 10, and 11 times more energy dissipation respectively than a 16 bit addition operation in a CMOS library optimized for low energy[14]. Cache was found to be important in [2] reducing energy dissipation by reducing the number of offchip accesses and it was found that reordering memory accesses to reduce the switching activities of addresses sent offchip was also useful in reducing energy dissipated. Researchers in [16] found that accessing words in parallel from memory would decrease energy dissipation by reducing number of memory accesses.

Researchers in [20,21] optimize the number of external and internal memory accesses and the number of extra computations (or data regeneration) for tasks such that the total estimated energy dissipation is minimized. This approach applies a network flow technique to solve this problem in polynomial time. Some recent algorithms developed for memory allocation either have not addressed low energy [18,10,12], or are very application-specific[16]. The problem of register allocation which minimizes energy dissipated was researched in [8] using a maximum cost flow approach, however memory allocation was not considered. Researchers in [9] found significant reduction in energy could be obtained by avoiding memory operands and recommended the use of optimal register allocation. Researchers have found that by using multiple memory modules one could reduce energy dissipation by using parallel access instructions[15], by entering inactive memory modules into sleep modes[4], or by reducing the switching of memory address lines [19]. Mapping data variables to both registers and memory has been studied by standard compiler techniques in [6,7,8] for performance objectives, unfortunately energy or power dissipation was not considered. Researchers in [8] use register spilling in conjunction with loop unrolling techniques.

In this manuscript a new approach is presented to solve problem 1, low energy register/memory design for VLSI systems synthesis. Unlike previous research, a minimum cost network flow approach to finding minimum energy solutions is introduced to simultaneously partition data variables into memory and/or allocate them into registers. A globally optimal solution can be obtained in polynomial time using very efficient algorithms[17]. This technique is used to determine the optimal partitioning of data variables into registers and memory and simultaneous register allocation. Single port or multiport register files and memory are supported. Voltage scaling of internal or external memory modules and register spilling is supported. Unlike previous research[8,4,15,16,18], we partition data variables into memory and registers simultaneously with register allocation, account for memory read and write energy dissipation, can support split lifetimes and can support memory components that run at lower frequencies for saving power. The next section will outline the assumptions and terminology to be used in the rest of the paper.

3.Assumptions and Notation

The following terminology will be used in this paper:

$Energy_{msystem}$= energy dissipated by the memory components of a system.

E^{x}_{opn}= energy dissipated by performing the operation on data variable v in the x component, where x=m for memory or x=r for register file. and opn=$r(v)$ for reading variable v or opn=$w(v)$ for writing variable v. For example $E^{r}_{w(v)}$ is the energy dissipated by writing variable v to the register file, r.

Vx= the value of the scaled voltage[3,11] of a memory component (x=m,r).

C^{x}_{rw}= the average switched capacitance of module x, where x=r,m, performing read/write access to registers, rw.

$H(v1,v2)$= the hamming distance between data variables $v1$ and $v2$.

Both static[3] and activity based[13] energy models can be supported. The first equation below is the static energy model which has separate read and write terms for both memory and registers. The second equation supports the activity based energy model for the register file where the hamming distance or other researched measures[8] of successive data variables which share the same register are multiplied by a total capacitance. Energy models for memory component design, where $v\varepsilon M,R$ represent a data variable v mapped to memory M or registerfile R respectively, are given as:

$$Energy_{msystem} = \sum_{v,v\varepsilon M}[E^{m}_{w(v)}+E^{m}_{r(v)}] + \sum_{v,v\varepsilon R}[E^{r}_{w(v)}+E^{r}_{r(v)}] \qquad (1)$$

$$= \sum_{v,v\varepsilon M}[E^{m}_{w(v)}+E^{m}_{r(v)}] + \sum_{v1\to v2,v1,v2\varepsilon R} H(v1,v2)C^{r}_{rw}Vr^{2}] \qquad (2)$$

The model uses an activity based model as in (2) for register files. Although it does not account for switching of address lines it allocates a minimum number of locations in memory so that a minimum number of addresses can be used. These issues will be discussed further in section 7 of this paper. The energy model presented above will be used to estimate the relative power savings in the rest of this paper.

The next section will provide an introduction to minimum cost network flow. The following sections, will illustrate how the low energy memory design problem is mapped into a network flow problem and provide examples illustrating how memory components can also run at different frequencies. Finally examples will be presented to show that new energy optimized solutions are produced unlike approaches using previous research.

4. Introduction to Network Flow

The minimum cost network flow problem is defined on a directed acyclic graph, G, composed of nodes and arcs, G=(V,A), where each arc has a capacity associated with it. The capacity is a real valued quantity where arc $i\to j$, $i,j\varepsilon V$, has an associated capacity $c_{i\to j}$. Each arc has a cost which is also a real valued quantity where arc $i\to j$ has associated cost $e_{i\to j}$. Let the variable, $x_{i\to j}$, represent the flow in arc $i\to j$. We will call this the flow variable. For any node in the graph the flow into the node is equal to the flow out of the node (known as the conservation of flow [17]). The flow along any arc (in the same direction as the arc) must be positive valued but less than or equal to the capacity of that arc. There are two special nodes in this graph called node s and node t. The flow out of s is equal to the flow into node t. Arcs incident to s only leave node s and arcs incident to node t only are directed into node t. The maximum flow problem [17] is to find the maximum amount of flow from node s to node t through the network graph, such that the conservation of flow is maintained.

The minimum cost network flow problem[17] is, given a fixed amount of flow, F, from s to t, find the flow with the minimum cost such that the sum over all arcs of the multiplication of the flow in each arc and the cost of each arc is minimum. The following equations represent the formulation of the minimum cost network flow problem as a mathematical programming problem.

$$Minimize \sum_{i\to j} e_{i\to j}x_{i\to j}$$

Subject to

$$\sum_{i|i\to j} x_{i\to j} - \sum_{k|j\to k} x_{j\to k} = 0, \forall j, j\neq s, j\neq t, j\varepsilon V.$$

$$\sum_{i|i\to t} x_{i\to t} = F$$

$$\sum_{k|s \to k} x_{s \to k} = F$$

$$0 \le x_{i \to j} \le c_{i \to j},\ \forall (i \to j) \varepsilon A, i, j \varepsilon V.$$

In the problem above the flow F, capacities $c_{i \to j}, \forall (i \to j) \varepsilon A$ and costs $e_{i \to j}, \forall (i \to j) \varepsilon A$ are given. The problem is to solve for values of $x_{i \to j}$ that represent the flows in the network graph, G=(V,A), such that the objective function is minimum. As long as the capacities and the flow, F, are integer, we can be guaranteed of obtaining integer flows in the solution of this problem [17]. This problem can be solved in polynomial time using linear programming or more commonly by using faster and more efficient network algorithms[17].

5. Methodology and Modeling

This section will briefly describe the methodology for low energy systems synthesis followed by a description of how the minimum cost network flow formulation is used to solve problem 1. First an algorithm is selected for the application based upon cost, performance and energy dissipation requirements. The algorithm is represented as a task flow graph. Initially the task flow graph is scheduled to obtain an analysis of how many chips or what amount of parallelism is required to meet or exceed the real-time constraints. Transformations are performed within each task such as data regeneration[20,21], loop tiling, precomputation, etc.. to reduce energy dissipation. Each task is placed in an ordered list, and detailed scheduling of computations within each task is performed. Finally the minimum cost network flow approach is applied to each basic block in each task to obtain further reductions in energy dissipation by optimaly mapping variables at different times to registers or memory using the technique presented in this paper. The lifetimes of data variables assigned to memory are then used to form another network flow graph. The minimum cost network flow is then solved on this graph to reallocate memory using an activity based energy model. After this stage is completed detailed instruction mapping and data layout (for example adding loads and stores, or substituting in instructions with a memory operand etc) and system synthesis is completed. The two sections below will describe in detail how the minimum cost network flow is used to solve problem 1.

5.1. Mapping to Min-Cost Flow

To model problem 1 as a network flow problem, we first have to develop a graph. An interval graph is built and then transformed into a network flow graph. Next capacities and costs are assigned to each arc in the network flow graph and the minimum cost flow problem is solved.

In order to form the interval graph a time axis (representing control steps) from time 1 to time x is defined, where x is the maximum number of control steps it takes to perform the computations in the basic block. Each data variable is represented by an interval (or lifetime) that starts at the time it is defined (write time) and ends at the time it is last used (read time). For illustration purposes only we will assume each data variables is read only once. Extensions for data variables with multiple reads will be presented in section 5.2. Nodes s and t are added to this graph at times 0 and x+1 respectively. Regions of maximum lifetime density, or sections of time where a maximum number of data variable's lifetimes intersect, are identified as well. Inbetween adjacent regions of maximum lifetime density, several data variable lifetimes may end and other lifetimes may begin. A complete bipartite graph is formed between these nodes. This technique is also used between node s and the first region of maximum lifetime density. It is also used to connect the nodes in the last region of maximum lifetime density to node t. This guarantees that a minimum number of addresses will be used in memory allocation (see section 7 for further discussion). Alternatively one could also form edges between all nonintersecting lifetimes like in [8], however a minimum number of address locations in memory is no longer guaranteed (see section 6).

Figure 1a) shows an interval graph, where each edge is labeled with the data variables names a,b,c,d,e. The left hand side shows the control steps from 1 to 7, where each control step is illustrated with two dashed lines. Variables represented by edges which end/begin at the top/bottom dashed line are read/written during that control step. For example at control step three, variables a and b are read and variable d is written. In figure 1b) nodes s and t are added at times 0 and 8 (or as shown in diagram after the last time, 7). Variables d and c are read after time 7 by another task, so their lifetime intervals extend to after time 7. Dashed arcs in the network flow graph are also added inbetween regions of maximum variable lifetime densities. For example a region of maximum lifetime density is from time 2 to time 3 and another region is from time 5 to time 6.

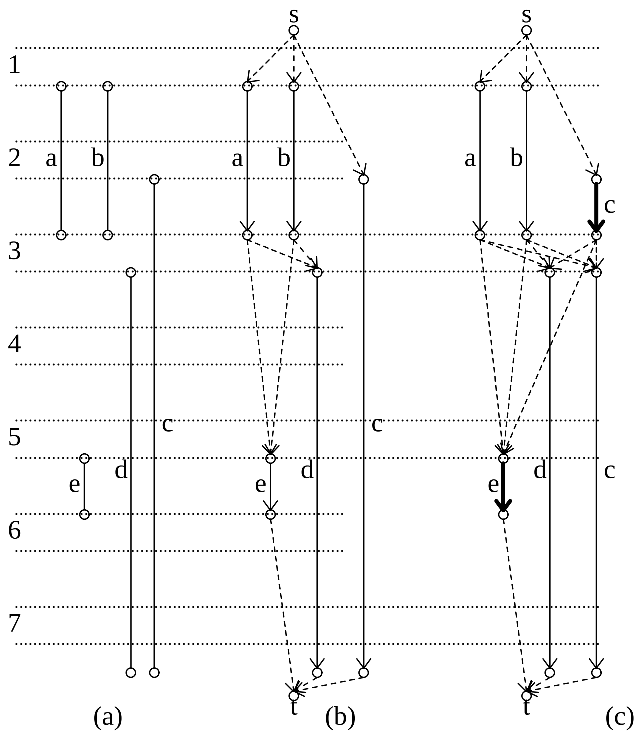

Figure 1. Example of interval graph for variables a,b,c,d,e in a) and it's network flow graph representation in b), and with restricted memory access times in c).

Inbetween these two adjacent regions lifetimes of data variables a and b end and lifetimes of data variables e and d begin. So a complete bipartite graph is formed between these regions by connecting edges from read times of data variables a and b to write times of data variables e and d shown with dashed arcs. This is also used to connect nodes s and nodes t to the variables lifetime edges. Now that the network graph is formed we next assign costs to each arc.

Given the network flow graph, G=(V,A), we use capacities along all arcs equal to one, and the flow is fixed at the total number of registers in our processor. The total flow in the network graph is equivalent to the number of registers, R, in the register file. The objective function to be minimized is the total estimated energy dissipated. We can formulate total energy dissipation as the sum of energy dissipated by writes and reads to/from memory and to/from registers. For data variables (v) where the flow into it's write node ($w(v)$) and out of it's read node ($r(v)$) is zero, energy is dissipated from a write and read to memory. Other data variables where the flow described above is one, dissipates energy from register file accesses. Arcs ($r(v1) \rightarrow w(v2)$) from the read time of one data variable ($v1$) into the write time of another data variable ($v2$) have costs assigned to them in order to support an activity based energy model. For example we can use previously researched activity models[8] for the cost of these arcs. The cost of each arc is calculated as follows:

$$e_{w(v) \rightarrow r(v)} = 0, \ \forall (w(v) \rightarrow r(v)) \varepsilon A, \ w(v), r(v) \varepsilon V. \tag{3}$$

$$e_{r(v1) \rightarrow w(v2)} = -E^m_{w(v2)} - E^m_{r(v1)} + E^r_{w(v2)} + E^r_{r(v1)} \tag{4}$$

$$= -E^m_{w(v2)} - E^m_{r(v1)} + H(v1,v2) C^r_{rw}, \tag{5}$$

$$\forall (r(v1) \rightarrow w(v2)) \varepsilon A, \ v1 \neq v2, \ r(v1), w(v2) \varepsilon V.$$

Now the objective function will be formulated using the flow variables $x_{r(v1) \rightarrow w(v2)}$. The first term below represents the energy dissipated by memory for each read and write of all data variables. The second term represents data variables which are mapped into register files, which have costs that subtract away the memory read/writes and adds the register file read/writes.

$$Minimize \ \sum_{v} [E^m_{w(v)} + E^m_{r(v)}]$$

$$+ \sum_{r(v1) \rightarrow w(v2)} [E^r_{w(v2)} + E^r_{r(v1)} - E^m_{w(v2)} - E^m_{r(v1)}] x_{r(v1) \rightarrow w(v2)}$$

Note that the first term is constant so we can remove it in our minimization problem. The energy dissipation for the registers will be formulated below with the activity based energy model. The final formulation of the minimum cost flow problem defined for $G=(V,A)$ as a mathematical programming problem is given below.

Minimize

$$\sum_{r(v1) \rightarrow w(v2)} [H(v1,v2) C^r_{rw} - E^m_{w(v2)} - E^m_{r(v1)}] x_{r(v1) \rightarrow w(v2)}$$

Subject to

$$\sum_{v | r(v) \rightarrow w(v1)} x_{r(v) \rightarrow w(v1)} - x_{w(v1) \rightarrow r(v1)} = 0,$$

$$\forall w(v1), w(v1) \neq s, w(v1) \neq t, w(v1) \varepsilon V.$$

$$x_{w(v1) \rightarrow r(v1)} - \sum_{v | r(v1) \rightarrow w(v)} x_{r(v1) \rightarrow w(v)} = 0,$$

$$\forall r(v1), r(v1) \neq s, r(v1) \neq t, r(v1) \varepsilon V.$$

$$\sum_{v | r(v) \rightarrow t} x_{r(v) \rightarrow t} = R$$

$$\sum_{v | s \rightarrow w(v)} x_{s \rightarrow w(v)} = R$$

$$0 \leq x_{w(v) \rightarrow r(v)} \leq 1, \ \forall (w(v) \rightarrow r(v)) \varepsilon A, w(v), r(v) \varepsilon V.$$

$$0 \leq x_{r(v1) \rightarrow w(v)} \leq 1, \ \forall (r(v1) \rightarrow w(v)) \varepsilon A, w(v), r(v1) \varepsilon V.$$

5.2 Extensions for Multiple Reads (Split Lifetimes)

In the previous sections we assumed that each data variable is read only once. For realistic applications data variables may be read several times. Furthermore memory access times may be restricted due to operating the memory module at a different frequency than the processor. In particular for low energy applications, memory modules may be operating at lower frequencies (and lower supply voltages to save energy[3]). Other reasons may also exist for using split lifetimes such as spilling of registers[7,10] or as will be shown in section 6 to minimize number of memory accesses. All of these cases can be supported with our model. Each data variable lifetime is divided into multiple lifetimes (or split lifetimes) by cutting the lifetime at memory access times and/or multiple read times. The multiple edges are connected to other edges in the network graph as if they would represent separate lifetimes using the procedure described earlier in section 5.1. However the edges costs and bounds on the flow variables are defined differently in these two cases, as will be explained below.

Figure 1c) shows an example of restricted memory access times, times 1,3,5, and the resulting network flow graph. The lifetime of data variable c intersects these memory access times so it becomes a split lifetime represented with two arcs. Any variables represented by lifetimes or split lifetimes which either begin and/or end inbetween the memory access times must be stored in the register files during these times. Therefore we place a lower bound on the flow in this arc to be 1. This extension to our problem can still be solved optimally in polynomial time as a minimum cost flow problem[17]. In figure 1c) these arcs are shown in bold for variable e and c (top arc of split lifetime). The arc costs must now support split lifetimes which may be both read/written from/to memory as well as register files (or spilled to memory from registers etc). Note we could have also split variables c and d into two segments, defined from control steps 3 to 5 and from 5 to 7. Figure 2 illustrates the different dashed arcs which may be defined with restricted memory access times. We use the following terminology to describe a split lifetime with multiple arcs $w1(v) \rightarrow r1(v)$, $w2(v) \rightarrow r2(v)$,..., $wlast(v) \rightarrow rlast(v)$ *or* $wi(v) \rightarrow ri(v)$ $\forall v, i = 1,2,...,rlast_v$. For example in figure 2c) dashed arc from $v1$ column to $v2$ column is $ri(v1) \rightarrow wi(v2)$ where $wi \neq w1(v2)$. Arcs are also placed inbetween these split lifetime arcs for the same data variable as shown in figure 2. For variables with multiple reads the objective function becomes:

$$Minimize \ \sum_{v} [E^m_{w(v)} + (rlast_v) E^m_{r(v)}]$$

$$+ \sum_{ri(vk) \rightarrow wj(vl)} [e_{ri(vk) \rightarrow wj(vl)}] x_{ri(vk) \rightarrow wj(vl)}$$

where the first term accounts for one write and $rlast_v$ reads of variable v. Again the first term is constant, so in the solution of the minimization problem, it can be deleted. This objective function is defined using equations (3,6,7,8,9,10) to define the coefficients. The general network flow constraints (defined at the end of the previous section 5.1) do not change, except their defined now over the new network flow graph which has more edges defining split lifetimes. Figure 2a) is the standard arc cost which we previously defined in equations (4) or (5), since the arc is connected from the last read node of $v1$ to the write node of $v2$, $rlast(v1) \rightarrow w1(v2)$, we have rewritten it in equation (10). Figure 2b),c),d) are examples for which arc costs are given below in equations (6),(7) and (8) respectively. In figures 2b) and 2c) variable $v1$ has two reads and in figures 2c) and 2d) variable $v2$ has two reads.

$$e_{ri(v1) \rightarrow w1(v2)} = -E^m_{r(v1)} - E^m_{w(v2)} + E^m_{w(v1)} + H(v1,v2) C^r_{rw}, \tag{6}$$

$$e_{ri(v1) \rightarrow wi(v2)} = E^m_{w(v1)} + H(v1,v2) C^r_{rw}, \tag{7}$$

$$e_{rlast(v1) \rightarrow wi(v2)} = -E^m_{r(v1)} + H(v1,v2) C^r_{rw}, \tag{8}$$

$$e_{ri(v) \rightarrow wi+1(v)} = -E^m_{r(v)} \tag{9}$$

$$e_{rlast(v1) \rightarrow w1(v2)} = -E^m_{w(v2)} - E^m_{r(v1)} + H(v1,v2) C^r_{rw}, \tag{10}$$

For example in figure 2b) if flow on arc $r1(v1) \rightarrow w1(v2)$ is equal to 1 on this dashed arc then data variable $v1$ was stored in the register file and now is being written back into memory while data variable $v2$ is replacing it by being written into the same register in the register file. So we have to account for subtracting out the energy dissipated for reading $v1$ from memory (first term), writing data variable $v2$ into memory (second term in equation (6)), and add in the energy dissipated by storing $v2$ into the same register as data variable $v1$ (fourth term in equation (6)), and finally writing data variable $v1$ into memory (third term in equation (6)). Equation (9) defines costs for arcs of the same data variable but defined inbetween split lifetimes, such as the dashed arc in figure 1c) from the node representing variable c at the top of the control step 3 to the node representing variable c at the bottom dashed line of control step 3. Here if two segments are stored in the register file, we must subtract the read of this variable from memory that would have taken place at that time.

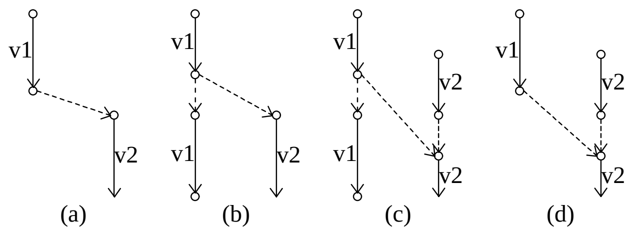

Figure 2. Different examples of arcs between data variables including split lifetimes for $v1,v2$.

6. Experimental Results

Several examples, along with a real industrial example (radar signal processing algorithm), are used to illustrate this methodology. A static energy model for onchip memory and register files was used with capacitances in [3] for onchip single port 256 X16 bit memory and single port 16X16bit register file. Although our model supports multiple port memory, energy models were not available, so single port memory models were used as an approximation. Activity based energy measures were also estimated using capacitance data in [3] along with models from [13]. Other energy measures previously researched[2,11,9,13,15] could not be used because for example they did not have both read/write register access energy values (separate from operations) and memory read/write access energy dissipation.

Figure 3a) shows a simple example of lifetimes of data variables a,b,c,d,e,f. The switching activities between each appropriate pair of data variables is given at the left hand side of the figure (as number of bits which change over total number of bits). The optimal solution for register allocation previously researched[8] would be figure 3a) which has a total switching activity of 2.4 (for illustration purposes we can assume that 0.5 of the bits change at time 0). If we then partition these symbolic registers into memory and registers we would ideally place the registers with highest switching activity in the register file (since average switched capacitance is smaller[3]). This is shown in the same diagram where the bold lifetimes represent data variables that are stored in the register file (which has capacity of one register in this simple example). By using the approach presented in this paper, simultaneously allocating registers and placing variables into memory to optimize energy, we obtain the solution shown in figure 2b). It has a lower switching activity in memory by 1.5 times and fewer memory accesses as well. The solution in figure 2b) has 1.4 times improvement in energy dissipation using a static energy model and 1.3 times using the activity based energy model.

A second example is shown in figure 4. In figure 4a) and 4b) we have used the network graph researched in [8], where all nonoverlapping lifetimes are connected by edges. Figure 4a) and b) show in bold the optimal solutions for partitioning after register allocation and the simultaneous approach respectively. Note that the solution in figure 4b), has a minimum number of memory accesses however uses more storage locations in memory. Using this type of network flow gpraph[8] we have no guarantee of using a minimum number of storage locations, unlike the use of the graph presented in this paper. Therefore in order to use this type of graph, one would have to account for energy dissipation in address lines as well which will be a part of future research. The solution in figure 4c) uses the network graph presented in this paper. The lifetime of data variable f is split to obtain a solution that has both a minimum number of storage locations in memory and a minimum number of memory accesses. It has 1.35 times improvement in energy dissipation than the solution in figure 4a).

arc	cost
$a \rightarrow b$	0.2
$a \rightarrow f$	0.5
$e \rightarrow b$	0.6
$e \rightarrow f$	0.3
$b \rightarrow c$	0.8
$d \rightarrow e$	0.1

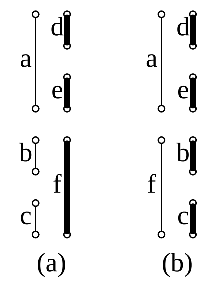

Figure 3. Memory partition after register allocation in (a), versus simultaneous partition and allocation in (b) respectively.

arc	cost
$a \rightarrow b$	0.2
$a \rightarrow f$	0.5
$e \rightarrow b$	0.6
$e \rightarrow f$	0.3
$b \rightarrow c$	0.8
$d \rightarrow e$	0.1
$f \rightarrow b$	0.5

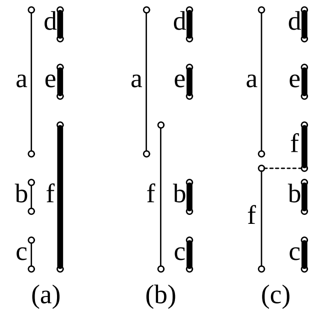

Figure 4. Fewer number of memory accesses than (a) using split lifetimes in (c) with fewer storage locations than (b).

A radar signal processing example was chosen to illustrate the minimum cost flow approach to simultaneous memory partitioning and register allocation with restricted memory access times. Table 1 outlines the design results of restricting memory access times thus allowing the memory module to operate at a lower frequency in a low power mode (using scaled supply voltage ranging from 5V to 2V). This example had a maximum density of variable lifetimes of 26. The memory module required one read/write port for solutions in rows 1 and 2, and required two read ports, one write port for the solution in the last row of table 1. Both static (E) and activity based (aE) energy estimates were used.

7. Discussions and Conclusions

In summary energy savings from 2.8 to 4.9 (see table 1) were attained with a real industrial signal processing example. Simultaneous memory partitioning and register allocation was found to have an improvement of 1.4 to 2.5 times for energy

Table 1. Energy Results for RSP application.

Memory	# Accesses		Relative	
Frequency	Mem	Reg	E	aE
f	6	12	4.9	2.8
f/2	7	11	2	1.6
f/4	8	10	1	1

dissipation over previously researched techniques.

The technique presented in this paper performs memory and register allocation. In order to simultaneously support an activity-based energy dissipation model for memory allocation a two-commodity flow problem would be required. Unfortunately the two-commodity flow problem is NP-complete[5]. After memory accesses, address circuitry is the next most significant source of energy dissipation[11]. This technique directly minimizes the number of memory accesses and by allocating a minimum number of storage locations in memory attempts to minimize the energy dissipation of address circuitry as well (see figure 4c) improvement over figure 4a)). The number of memory or register file ports is determined from the solution of our network flow problem, however it could be also specified as a constraint in our problem. For a fixed number of memory or register file ports the technique described in section 5.2 which sets certain arc flows to 1 can be used.

In contrast to previous research[2,7,10,16,15] which examined register allocation without memory partitioning, or did not consider low energy objectives, or performed memory partitioning without register allocation, we have examined the problem of simultaneous memory partitioning and register allocation. This problem is important since memory has been shown to be a crucial component of the total system energy dissipation[11] and low energy demands optimal register allocation[9]. We have introduced the application of network flow to this problem, which has been shown here for the first time to be beneficial in reducing estimated energy dissipation at negligible cost. Since large network flow problems have been solved with very efficient algorithms [17], extending this problem to very large basic blocks or beyond basic blocks should be a viable future research direction.

We have introduced a methodology for energy minimization of storage components in large realtime compute intensive applications. The methodology incorporates a new approach to low energy memory design which simultaneously performs memory partitioning and register allocation. For the first time network flow is applied to the problem of determining when and what data should be stored in memory and which data should share registers in order to minimize estimated energy dissipation. Restricted memory access times and split lifetimes are supported. The network flow approach finds globally optimal solutions in polynomial time. Results showed that 2.8 to 4.9 times improvement in energy dissipation is attainable with a real industrial radar signal processing application. Significantly larger savings in energy are expected when this network flow technique is applied to offchip memory, where energy dissipation of memory accesses is several orders of magnitude higher[2,19,14] than the onchip memory accesses. It is also interesting to note that this approach comes at no expense to performance or cost, unlike other techniques where cost may be traded for energy dissipation, such as voltage scaling. This research is important for industry since energy dissipation consideration during these early stages of design are critical to ensuring that the final product will be cost effective, competitive, and meet high performance requirements. This approach has recently been extended to solve the multiple offset assignment problem in software synthesis for DSP processors where performance, code size and power objective functions are supported. This research is supported in part by grants from NSERC and ITRC.

References

[1] K.Keutzer, "The Impact of CAD on the Design of Low Power Digital Circuits", IEEE Symposium on Low Power Electronics, 1994, p42-45.

[2] S.Wuytack,F.Catthoor,F.Franssen,L.Nachtergaele, H.DeMan, "Global Communication and Memory Optimizing Transformations For Low Power Systems", International Workshop on Low Power Design, 1994, p 203-208.

[3] A.Chandrakasan,et.al."Optimizing Power Using Transformations", IEEE Transactions on CAD, Jan 1995,Vol14,No.1, p12-31.

[4] A.Farrahi, G.Tellez,M.Sarrafzadeh, "Memory Segmentation to Exploit Sleep Mode Operation", Design Automation Conference, 1995.

[5] Garey and Johnson,**Computers and Intractability** New York Freeman and Co 1979.

[6] G.Chaitin, "Register Allocation & Spilling via Graph Coloring", ACM SIGPLAN Symp on Compiler Construction, 1982

[7] F.Chow,J.Hennessey, "The Priority-Based Coloring Approach to Register Allocation", ACM Transactions on Programming Languages and Systems, p501-536, Oct 1990.

[8] J.Chang, M.Pedram, "Register Allocation and Binding for Low Power", Design Automation Conference, 1995.

[9] V.Tiwari, S.Malik,A.Wolfe, "Power Analysis of Embedded Software ;A First Step Towards Software Power Minimization, IEEE Trans on VLSI, Vol. 2, No. 4, Dec 1994, p437-445,

[10] D.Kolson,A.Nicolau,N.Dutt,K.Kennedy, "Optimal Register Allocation to Loops for Embedded Code Generation", International Symposium on Systems Synthesis, p42-47, 1995.

[11] S.Wuytack,F.Catthoor,L.Nachtergaele,H.DeMan, "Power Exploration for Data Dominated Video Applications", ISLPED, p359-364, 1996.

[12] W.Cheng, Y-L.Lin, "A Transformation-Based Approach for Storage Optimization", ACM/IEEE Design Automation Conference, 1995.

[13] P.Landman, J.Rabaey, "Activity-Sensitive Architectural Power Analysis", IEEE Transactions on CAD,Vol.15,No.6, June 1996.

[14] P.A.Beerel, USC, Panel Presentation on "Where Does the Power Go?" at International Symposium on Low Power Design, April 1995.

[15] Lee,Tiwari, "A Memory Allocation Technique for Low-Energy Embedded DSP Software", Symposium on Low Power Electronics, Oct, 1995, p24-5.

[16] Lidsky,Rabaey, "Low-Power Design of Memory Intensive Functions", IEEE Symposium on Low Power Electronics, 1994, p16-7.

[17] Nemhauser, Wolsey, **Integer and Combinatorial Optimization**, New York Wiley Interscience, 1988.

[18] H.Schmitt,D.Thomas, "Array Mapping in Behavioral Synthesis ", Int'l Symp on Systems Synthesis, 1995.

[19] P.Panda, N.Dutt, "Low Power Mapping of Behavioral Arrays to Multiple Memories", Int'l Symp on Low Power Electronic Design, p289-292, 1996.

[20] C.Gebotys, "Low Energy Memory Component Design for Cost-Sensitive High Performance Embedded Systems", IEEE Custom Integrated Circuits Conference, p397-400, 1996.

[21] C.Gebotys,R.Gebotys, "Performance-Power Optimization of Memory Components for Complex Embedded Systems", IEEE, 30th Hawaii International Conference on System Sciences, ECCS-6, Jan. 1997.