# Retargetable Compilation for Low Power

Wen-Tsong Shiue

Silicon Metrics Corporation
12710 Research Blvd. Suite 300
Austin, TX 78759
Phone: 1-(512)-651-1503
Email: shiue@siliconmetrics.com

## ABSTRACT

Most research to date on energy minimization in DSP processors has focuses on hardware solution. This paper examines the software-based factors affecting performance and energy consumption for architecture-aware compilation. In this paper, we focus on providing support for one architectural feature of DSPs that makes code generation difficult, namely the use of multiple data memory banks. This feature increases memory bandwidth by permitting multiple data memory accesses to occur in parallel when the referenced variables belong to different data memory banks and the registers involved conform to a strict set of conditions. We present novel instruction scheduling algorithms that attempt to maximize the performance, minimize the energy, and therefore, maximize the benefit of this architectural feature. Experimental results demonstrate that our algorithms generate high performance, low energy codes for the DPS architectural features with multiple data memory banks. Our algorithm led to improvements in performance and energy consumption of 48.3% and 66.6% respectively in our benchmark examples.

## Keywords

Architecture-aware compiler design, high performance and low power design, instruction scheduling, register allocation.

## 1. INTRODUCTION

Currently, there is a high demand for DSP processors with low power/energy in many areas such as telecommunications, information technology and automotive industries. This demand stems from the fact that low power consumption is important for reliability and low cost production as well as device portability and miniaturization.

In the last decade we have seen the proliferation of electronic equipment like never before. As these systems are becoming increasing portable, the minimization of power consumption has become an important criterion in system design. In order to design a system with low energy and high performance, it is important to analyze all the components of the system platform. Since a large portion of the functionality of today's system is in the form of software, it is important to estimate and minimize the software component of the energy cost and maximizes the software component of the performance cost [1][2].

Although dedicated hardware can provide significant speed and power consumption advantages for signal processing applications, extensive programmability is becoming an increasingly desirable feature of implementation platforms for VLSI signal processing. Increasingly shorter life cycles for consumer products have fueled the trend toward tighter time-to-market windows, which in turn, caused intense competition among DSP product vendors and forced the rapid evolution of embedded technology. As a consequence of these effects, designers are often forced to begin architecture design and system implementation before the specification of a product is fully completed. For example, a portable communication product is often designed before the signal transmission standards under which it will operate are finalized, or before the full range of standards that will be supported by the product is agree upon. In such an environment, late changes in the design cycle are mandatory. The need to quickly make such late changes requires the use of software.

Although the flexibility offered by software is critical in DSP applications, the implementation of production quality DSP software is an extremely complex task. The complexity arises from the diversity of critical constraints that must be satisfied. Typically these constraints involve stringent requirements on metrics such as latency, throughput, power consumption, code size, and data storage requirements [3].

DSPs are a special kind of processor that is primarily designed to implement signal-processing algorithms efficiently. Code generation for DSP is more involved than general-purpose processors. This is because DSP processors have non-homogeneous register sets, a number of specialized functional units, restricted connectivity, limited addressing, and highly irregular datapaths. It is a well-known fact that the quality of compilers for embedded DSP systems are generally unacceptable with respect to code density, performance, and power consumption. This is because the compilation techniques for general-purpose architectures being used do not adapt well to the irregularity of DSP architectures.

We address the problem of code generation for DSP systems on a chip. In such systems, the amount of silicon devoted to program ROM is limited, so the application software must be sufficiently dense. In addition, the software must be written to meet various high-performance and low energy constraints. Unfortunately, current compiler technologies are unable to generate high-quality code for DSPs, whose architectures are highly irregular. Thus, designers often resort to programming application software in assembly – a labor-intensive task.

In this paper, we present a novel instruction scheduling for one particular architectural feature, namely multiple data memory banks. This feature, increases memory bandwidth by permitting multiple data memory accesses to occur in parallel. This happens when the referenced variables belong to different banks and the register involved conforms to a strict set of conditions. Furthermore, the instruction set architecture (ISA) of these DSPs require the programmer to encode in a limited number of long instruction words, all the data memory accesses that are to be performed in parallel, thus assisting in the generation of dense code.

Instruction scheduling techniques that use a listed-based method has been around since the mid-1980s [4], and it is the most popular method of scheduling basic blocks. Trace scheduling is an optimization technique that selects a sequence of basic blocks as a trace, and schedules the operations from the trace together [5]. Percolation scheduling [6] looks at the whole program and tries to improve the parallelism of the code. The idea that register allocation can be viewed as a graph-coloring problem has been around since early 1970s, but Chaitin et. al. [7] were the first to actually implement it in a compiler. Briggs [8] came up with some modifications to Chaitin-style allocation, the most important idea being the optimization of variable selection for register spilling.

Most of the previous work on reducing power and energy consumption in DSP processors has focused on hardware solutions to the problem. However, embedded systems designers frequently have no control over the hardware aspects of the pre-designed processors with which they work and so, software-based power and/or energy minimization techniques play a useful role in meeting design constraints. Recently, new research directions in reducing power consumptions have begun to address the issues of arranging software at instruction-level to help reduce power consumption [10][11]. Previous improvements with software re-arrangements include the value locality of registers [10] and the swapping of operands for booth multiplier [11]. This new direction brings an interesting issue in the compiler participation in software re-arrangements for reducing power consumption for applications and systems.

The rest of the paper is organized as follows. Section 2 describes the DSP architectural features with multiple data memory banks. Section 3 describes the novel algorithms of instruction scheduling to reduce the number of cycles and the register pressure. Section 4 describes the examples that illustrate our algorithms. Section 5 describes the benchmark results. Section 6 concludes the paper.

# 2. DSP ARCHITECTURAL FEATURES WITH MULTIPLE DATA MEMORY BANKS

Our approach for increasing the packing efficiency has been tested on DSP architectural features with multiple data memory banks, which can be characterized as Dual-Load-Execute (DLE) architectures. Examples of DLE processors include Analog Devices' ADSP21xx family, NEC's u7701x family, Motorola, 56xxx family, and Fujitsu's Elixir family. These processors support parallel execution of an ALU operation and two data move (data load or data store) operations in the same cycle.

## 2.1 DSP Architectures

The DSP architectural units of interest are the data arithmetic logic unit (Data ALU), addressing generation unit (AGU) and X/Y data memory banks [9]. The unit of Data ALU contains hardware specialized for performing fast multiply-accumulate operations (MAC). The data ALU consists of FOUR 24-bit input registers named X0, X1, Y0, and Y1, and TWO 56-bit accumulators named A and B. The resource operands for all ALU operations must be input registers or accumulators, and the destination operand must always be an accumulator. TWO 24-bit buses named XDB and YDB permit two input registers or accumulators to be read or written in conjunction with the execution of an ALU operation. As a result, three operations may be executed simultaneously in one instruction cycle. The Address Generation Unit (AGU) contains TWO sets of 16-bit register files, one consisting of address registers R0, R1, R2, and R3 and offset registers N0, N1, N2, and N3, and the other consisting of address registers R4, R5, R6, and R7 and offset registers N4, N5, N6, and N7. The unit of X/Y Data Memory Banks contains two 512 words x 24 bits memory banks which allow a total of two data memory accesses to occur in parallel.

## 2.2 Instruction Set Architecture (ISA)

The ISA of above DSPs assists in the generation of dense, high-bandwidth code by requiring the programmer to encode all operations that are to execute in parallel during each instruction cycle in either one or two 24-bit instruction words. Specifically, up to two move operations and one Data ALU operation may be encoded in these words. A move in this case refers to a memory access (load or store), register transfer (moving of data from an input registers to an accumulator, or vice versa), or immediate load (loading of a 24-bit constant into an input register or accumulator). However, due to the nature of the M56000 micro-architecture, only the following pairs of move operations may be performed in parallel: (i) two memory accesses, (ii) a memory access and register transfer, and (iii) a register transfer and a load immediate.

Consider the following parallel move specification that simultaneously (i) loads a datum into X1 from the X memory bank at the address stored in R0 and (ii) loads a datum into Y1 from the Y memory bank at the address stored in R4. {MOVE X: (R0), X1 Y: (R4), Y1}.

# 3. INSTRUCTION SCHEDULING

Our instruction-scheduling algorithm based on list scheduling directly supports packing, because the above DSP supports simultaneous execution of multiple operations. Packing is efficient in terms of performance, because it always leads to a reduction in the cycle time of programs. Another important feature of packing is that it also tends to reduce the amount of energy consumed during program execution. In practice, packing has the potential to reduce energy consumption by more than half.

## 3.1 Instruction Level Energy Model

The average power P consumed by a processor while running a certain program is given by P = I*Vdd, where I is the average current and Vdd is the supply voltage. The energy consumed by a program, E, is given by E = P*T, where T is the execution time of the program. This, in turn, is given by T = N*delta, where N is the number of cycles and delta is the cycle period. Since a common application DSP embedded system is often in the portable space where power is stored in a battery, energy consumption is the focus of our attention. Now, Vdd and delta are known and fixed. Therefore, E is proportional to the product of I and N. Given the number of execution cycles, N, for a program, we only need to measure the average current, I, in order to calculate E. The product of I and N is, therefore, the measure used to compare the energy cost of programs in this analysis. The energy model as taken from

[2], is based on measuring the current consumed by individual instructions using an oscilloscope and estimating the energy consumed by a block of software through the calculation of a weighed sum.

## 3.2 Reduce the Register Pressure during the Scheduling Stage

List scheduling might give good results in terms of reducing the number of cycles required for execution, but it cannot guarantee an optimal scheduling of the code. If the required number of registers is very high, then register spills may decrease the performance of the final code. In addition to the notion of priority, our data dependence graph (DDG) also includes the lifetime of each variable so that register pressure can also be reduced in the scheduling.

# 4. EXAMPLES AND ALGORITHMS

## 4.1 Example One

Consider the C and corresponding uncompacted symbolic assembly code shown in Figure 1(a) and Figure 1(b). Figure 1(c) shows the DDG with two weighting factors on each node. The first weighting factor in the tuple is the value of depth, which is counted from the bottom node for each branch. The node with a higher value has higher priority. For instance, nodes V0 and V1 have the highest priority to be selected in the ready set shown in Figure 3.

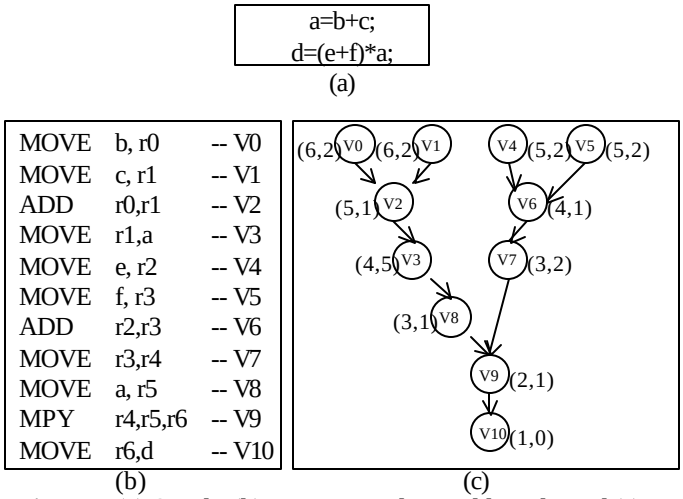



**Figure 1. (a) C code, (b) uncompacted assembly code, and (c) DDG with tuple of (depth,lifetime) on the node.**

Note that the boldface nodes in the ready set are ALU operations such as ADD and MPY. One ALU can be executed with one or two MOVE operation in the same cycle. Two ALU operation nodes, though, cannot be allowed to execute in the same cycle. The second weighting factor in the tuple is the lifetime of the register variables. For instance, node V0 is dependent upon register r0 that is alive during cycles 1 through 3 (see Figure 2); thus, the interval live time is 3-1 = 2. In Figure 2, the initial code is executed in 11 cycles and the number of registers required is 2 using the lifetime analysis. Next, we construct the ready set based on the as-soon-as-possible (ASAP) scheduling scheme for each node. The total cycles are now 6 cycles for the unscheduled nodes in ready set.

Figure 4 shows the nodes have been scheduled based on our algorithm to exploit the DSP architecture. The number of registers is 3. Note that the number of cycles and the number of registers have the tradeoff relationship. It is cost efficient to reduce the cycles instead of increasing the number of registers. However, for DSP processors, the number of registers are limited, so we need to develop the best scheduling algorithm to minimize the number of registers (i.e. reduce the register pressure) during the scheduling process. Figure 5 shows that for the random choice, the number of cycles is 7 and the number of registers is 4. This demonstrates that the algorithm developed is very important at this stage.

| Cycle | Instruction nodes | Registers and data | Live time Analysis |
|---|---|---|---|
| 1 | V0 | b,r0 | r0 r1 r2 r3 r4 r5 r6 |
| 2 | V1 | c,r1 | |
| 3 | V2 | r0,r1 | |
| 4 | V3 | r1,a | |
| 5 | V4 | e,r2 | |
| 6 | V5 | f,r3 | Need 2 registers |
| 7 | V6 | r2,r3 | |
| 8 | V7 | r3,r4 | |
| 9 | V8 | a,r5 | |
| 10 | V9 | r4,r5,r6 | |
| 11 | V10 | r6,d | |

**Figure 2. Before scheduling. (Need 11 cycles and 2 registers)**

| Cycle | Instruction nodes |
|---|---|
| 1 | V0, V1, V4, V5 |
| 2 | **V2**, **V6** |
| 3 | V3, V7 |
| 4 | V8 |
| 5 | **V9** |
| 6 | V10 |

**Figure 3. Unscheduled nodes in ready set. (Boldface nodes are ALU operations).**

We borrowed the energy model from [2] to count the energy consumption. In Figure 4, total current, I, is 780mA and total cycles, N, are 6. So, the energy cost is I*N = 780*6 = 4,680. In Figure 5, the scheduling based on random choice has an energy cost of I*N=850*7=5,950.

Before scheduling, in Figure 1, the total current is 1080mA and the number of cycles is 11. So, the energy cost is I*N= 11,880. After scheduling, Figure 4, based on our algorithm, the number of cycles is reduced from 11 to 6 (45% reduction in cycles) and the energy cost is reduced from 11,880 to 4,680 (60% reduction in energy consumption). This demonstrates that our scheduling algorithm has made a high performance, low energy code generation with minimum register pressure for the DSP processors.

| Cycle | Instruction nodes | Live time Analysis |
|---|---|---|
| 1 | V0, V1<br>(b,r0) (c,r1) | r0 r1 r2 r3 r4 r5 r6 |
| 2 | **V2**, V4, V5<br>(r0,r1) (e,r2) (f,r3) | Need 3 registers |
| 3 | V3, **V6**<br>(r1,a) (r2,r3) | |
| 4 | V7, V8<br>(r3,r4) (a,r5) | |
| 5 | **V9**<br>(r4,r5,r6) | |
| 6 | V10<br>(r6,d) | |

| Final Code | | | |
|---|---|---|---|
| MOVE | b,r0 | c,r1 | -- 120mA |
| ADD r0,r1 | e,r2 | f,r3 | -- 150mA |
| ADD r2,r3 | r1,a | | -- 140mA |
| MOVE | r3,r4 | a,r5 | -- 120mA |
| MPY r4,r5,r6 | | | -- 160mA |
| MOVE | r6,d | | -- 90mA |

**Figure 4. After scheduling based on our scheduling algorithm. (Need 6 cycles and 3 registers).**

| Cycle | Instruction nodes | Live time Analysis |
|---|---|---|
| 1 | V0, V4<br>(b,r0) (e,r2) | r0 r1 r2 r3 r4 r5 r6 |
| 2 | V1, V5<br>(c,r1) (f,r3) | Need 4 registers |
| 3 | **V2**,<br>(r0,r1) | |
| 4 | V3, **V6**<br>(r1,a) (r2,r3) | |
| 5 | V8 V7<br>(a,r5) (r3,r4) | |
| 6 | **V9**<br>(r4,r5,r6) | |
| 7 | V10<br>(r6,d) | |

| Final Code | | | |
|---|---|---|---|
| MOVE | b,r0 | e,r2 | -- 120mA |
| MOVE | c,r1 | f,r3 | -- 120mA |
| ADD r0,r1 | | | -- 100mA |
| ADD r2,r3 | r1,a | | -- 140mA |
| MOVE | r3,r4 | a,r5 | -- 120mA |
| MPY r4,r5,r6 | | | -- 160mA |
| MOVE | r6,d | | -- 90mA |

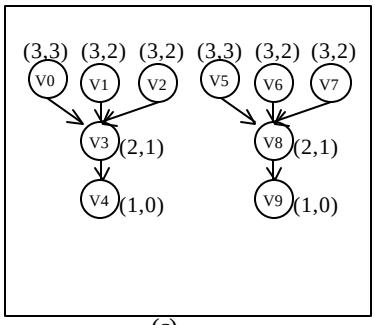
**Figure 5. After scheduling based on RANDOM choice. (Need 7 cycles and 4 registers).**

## 4.2 Example Two

Again, consider the C and corresponding uncompacted symbolic assembly code shown in Figure 6(a) and Figure 6(b). Figure 6(c) shows the DDG with two weighting factors on each node.

```
v=a*b+c;
w=d*e+f;
```
(a)

```
MOVE   a, r0     -- V0
MOVE   b, r1     -- V1
MOVE   c, r2     -- V2
MAC    r0,r1,r2  -- V3
MOVE   r2,v      -- V4
MOVE   d, r3     -- V5
MOVE   e, r4     -- V6
MOVE   f, r5     -- V7
MAC    r3,r4,r5  -- V8
MOVE   r5,w      -- V9
```
(b)



(c)

**Figure 6. (a) C code, (b) uncompacted assembly code, and (c) DDG with tuple of (depth,lifetime) on the node.**

| Cycle | Instruction nodes | Live time Analysis |
|---|---|---|
| 1 | V0, V5<br>(a,r0) (d,r3) | r0 r1 r2 r3 r4 r5 |
| 2 | V1, V2,<br>(b,r1) (c,r2) | Need 4 registers |
| 3 | **V3**, V6 V7<br>(r0,r1,r2)(e,r4)(f,r5) | |
| 4 | V4, **V8**<br>(r2,v) (r3,r4,r5) | |
| 5 | V9<br>(r5,w) | |

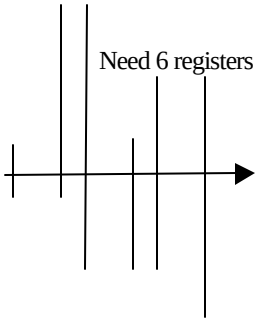
| Final Code | | | |
|---|---|---|---|
| MOVE | a,r0 | d,r3 | -- 120mA |
| MOVE | b,r1 | c,r2 | -- 120mA |
| MAC r0,r1,r2 | e,r4 | f,r5 | -- 180mA |
| MAC r3,r4,r5 | r2,v | | -- 170mA |
| MOVE | r5,w | | -- 90mA |

**Figure 7. After scheduling based on our scheduling algorithm. (Need 5 cycles and 4 registers).**

Our algorithm focuses on reducing the number of cycles and the register pressure at the same cycle. This helps to do register labeling in the next stage. Besides that, due to the code compaction, the number of cycles is minimized and further results in the reduction of the energy consumption.

Before scheduling, in Figure 6, the total current is 1040mA and the number of cycles is 10, giving an energy cost of I*N= 10,400. After scheduling, in Figure 7, based on our algorithm, the number of cycles is reduced from 10 to 5 (50% reduction in cycles) and the energy cost is reduced from 10,400 to 3,400 (67.3% reduction in energy consumption). The number of registers is only 4. If random choice, the number of cycles is increased from 5 to 6 and the number of registers is increased from 4 to 6 (see Figure 8). This implies that it degrades the performance and increases the energy consumption. Furthermore, it needs more registers. In Figure 7, total current, I, is 680mA and total cycles, N, are 5. So, the energy

cost is I*N = 680*5 = 3,400. In Figure 8, the scheduling based on random choice has an energy cost of I*N=780*6=4,680.

| Cycle | Instruction nodes | Live time Analysis |
|---|---|---|
| 1 | V1, V2<br>(b,r1) (c,r2) | r0 r1 r2 r3 r4 r5<br>  |
| 2 | V6, V7,<br>(e,r4) (f,r5) | |
| 3 | V0, V5<br>(a,r0) (d,r3) | |
| 4 | **V3**<br>(r0,r1,r2) | |
| 5 | V4, **V8**<br>(r2,v) (r3,r4,r5) | |
| 6 | V9<br>(r5,w) | |

| Final Code | | | |
|---|---|---|---|
| MOVE | b,r1 | c,r2 | -- 120mA |
| MOVE | e,r4 | f,r5 | -- 120mA |
| MOVE | a,r0 | d,r3 | -- 120mA |
| MAC r0,r1,r2 | | | -- 160mA |
| MAC r3,r4,r5 | r2,v | | -- 170mA |
| MOVE | r5,w | | -- 90mA |

**Figure 8. After scheduling based on RANDOM choice. (Need 6 cycles and 6 registers).**

## 4.3 Our Algorithms

Our algorithm is based on the list scheduling but the priority should be modified not only reduce the number of cycles (i.e. improve the performance) but also minimize register pressure. The following is our *algorithm.*

*Algorithm*

1. *Construct the Data Dependence Graph (DDG).*
2. *Make the Tuple (P,L), where P is the depth value of the node, and L is the lifetime of the node.*
3. *Find the initial ready set R (List Scheduling)*
4. *While ~isempty(R) do*
    - *The node with larger value of P has the higher priority. (reducing the number of cycles and energy cost)*
    - *If tie in value of P:*
        - *If there are pairs of nodes*
            - *The pair of nodes having the same lifetime value, L, has higher priority. (reducing the register pressure)*
        - *else*
            - *the node with lower lifetime value has higher priority. (reducing the register pressure)*
        - *If tie in values of P and L*
            - *The pair of nodes located in the same branch, has the higher priority. (reducing the energy cost). Note that the ALU nodes can be parallel processed with the other one or two MOVEs*

Before running the algorithm, we need to construct the data dependence graph (DDG) with nodes and calculate the depth value and lifetime value for each node. Next, the list scheduling starts to come to play. Before that, we need to construct the initial ready set R.

Our algorithm considering the first priority is the value of depth, the first weighting factor in tuple. If there are nodes in the same cycle of ready set, the node with higher depth value has higher priority. This ensures that the number of cycles is minimized. Remind that the number of cycles is the function of the energy cost. This helps reduce energy cost a lot.

Besides that the concept of reducing the number of cycle, the other important issue is how to reduce the number of registers. This is because the number of registers in DSP processor is limited. Hence, the next priority is to consider the second weighting factor in tuple. If the nodes have the same depth values, the pair of nodes having the same lifetime has higher priority. Otherwise, the node with lower lifetime value has higher priority. This is to reduce the number of overlap lifetimes among the nodes and hence reduce the number of registers.

If the nodes have the same depth values and the same lifetime values, the nodes in the same tree have the higher priority. This is to increase the chance of locating the ALU nodes with the MOVE nodes in the same cycle. Note that the ALU nodes can be parallel processed with the other one or two MOVEs. This helps reduce the value of current. For example, ADD with 2 parallel MOVEs only consumes the current of 150mA but the ADD with 2 serial MOVEs consumes the current of 280mA.

## 5. BENCHMARK RESULTS

In our benchmark results, the performance is improved in average 48.3% and the energy consumption is saved in average 66.6% (see Figure 9). Figure 9 shows the unscheduled assembly codes, scheduled assembly codes, and the total required current, the number of cycles, and the energy cost for both codes.

| Example 1: {a=b+c; d=(e+f)*a;} | |
|---|---|
| Unscheduled Assembly Codes | Scheduled Assembly Codes |
| MOVE b, r0 -- 90mA | MOVE b,r0 c,r1 -- 120mA |
| MOVE c, r1 -- 90mA | ADD r0,r1 e,r2 f,r3 -- 150mA |
| ADD r0,r1 -- 100mA | ADD r2,r3 r1,a -- 140mA |
| MOVE r1,a -- 90mA | MOVE r3,r4 a,r5 -- 120mA |
| MOVE e, r2 -- 90mA | MPY r4,r5,r6 -- 160mA |
| MOVE f, r3 -- 90mA | MOVE r6,d -- 90mA |
| ADD r2,r3 -- 100mA | |
| MOVE r3,r4 -- 90mA | |
| MOVE a, r5 -- 90mA | |
| MPY r4,r5,r6 --160mA | |
| MOVE r6,d -- 90mA | |
| Total Current: 1,080mA | Total Current: 780mA |
| Total Cycles: 11 cycles | Total Cycles: 6 cycles |
| Energy Cost: 11,880 | Energy Cost: 4,680 |

| Example 2: {v=a*b+c; w=d*e+f;} | |
|---|---|
| Unscheduled Assembly Codes | Scheduled Assembly Codes |
| MOVE a, r0 --90mA | MOVE a,r0 d,r3-- 120mA |
| MOVE b, r1 --90mA | MOVE b,r1 c,r2-- 120mA |
| MOVE c, r2 --90mA | MAC r0,r1,r2 e,r4 f,r5 --180mA |
| MAC r0,r1,r2 --160mA | MAC r3,r4,r5 r2,v -- 170mA |

| | |
|---|---|
| MOVE r2,v --90mA<br>MOVE d, r3 --90mA<br>MOVE e, r4 --90mA<br>MOVE f, r5 --90mA<br>MAC r3,r4,r5 --160mA<br>MOVE r5,w --90mA<br><br>Total Current: 1,040mA<br>Total Cycles: 10 cycles<br>Energy Cost: 10,400 | MOVE r5,w -- 90mA<br><br><br><br><br><br><br>Total Current: 680mA<br>Total Cycles: 5 cycles<br>Energy Cost: 10,400 |

| Example 3: {a=b+c; f=d+e;} | |
|---|---|
| Unscheduled Assembly Codes | Scheduled Assembly Codes |
| MOVE a, r0 -- 90mA<br>MOVE b, r1 -- 90mA<br>ADD r0,r1 -- 100mA<br>MOVE r1,c -- 90mA<br>MOVE d, r2 -- 90mA<br>MOVE e, r3 -- 90mA<br>ADD r2,r3 -- 100mA<br>MOVE r3,f -- 90mA<br><br>Total Current: 740mA<br>Total Cycles: 8 cycles<br>Energy Cost: 5,920 | MOVE a,r0 b,r1 -- 120mA<br>ADD r0,r1 d,r2 e,r3 -- 150mA<br>ADD r2,r3 r1,c -- 140mA<br>MOVE r3,f -- 90mA<br><br><br><br><br><br>Total Current: 500mA<br>Total Cycles: 4 cycles<br>Energy Cost: 2,000 |

| Bench-mark | Performance | | | Energy Cost | | |
|---|---|---|---|---|---|---|
| | Before | After | Improve | Before | After | Savings |
| Ex 1. | 11 cycles | 6 cycles | **45%** | 11,880 | 4,680 | **66.2%** |
| Ex 2. | 10 cycles | 5 cycles | **50%** | 10,400 | 3,400 | **67.3%** |
| Ex 3. | 8 cycles | 4 cycles | **50%** | 5,920 | 2,000 | **66.2%** |

**Figure 9. Benchmark Results.**

## 6. CONCLUSION

In recent years, power/energy consumption has become one of the primary constraints in the design of embedded applications. Current compiler technology is unable to take advantage of the potential increase in parallelism offered by multiple data memory banks. Consequently, compiler-generated code is far inferior to hand-written code. While optimizing compilers have proved effective for general purpose processors such as PowerPC CPU (RISC), Intel CPU, AMD CPU, and 680x0 CPU (CISC), and Intel CPU (EPIC), the irregular datapaths and small number of registers found in embedded processors, especially fixed-point DSPs with multiple data memory banks, remain a challenge to compilers.

In this paper, we have developed a high performance, low energy compiler design for DSPs. Such a compiler should not compromise on performance or code size when reducing energy consumption. Our benchmark shows that the performance is improved in average 48.3% and the energy consumption is saved in average 66.6%.

We assume that instruction selection is performed by another compiler and a sequence of instructions (unpacked) is given to our procedure. In the future work, we will develop an algorithm consisting of instruction scheduling, register allocation and memory assignment. This involves building a re-targetable compiler and simulator tool-kit which is extensible; developing program transformations for automatically exploiting all of the useful parallelisms in a given program, developing "architecture-aware" and "memory aware" optimizations for compiling, and finally exploring the interaction between compilers and architectures.

## 7. REFERENCES


[1] D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, S. Zhao, Spec C: specification language and methodology, in Kluwer Academic Publishers (March 2000).

[2] Sathishkumar Udayanarayanan, Energy -efficient code generation for DSP56000 family, MS. Thesis in Arizona State University (Aug. 2000).

[3] S. S. Bhattacharyya, R. Leupers, and P. Marwedel, Software synthesis and code generation for signal processing systems, IEEE Trans. On Circuit and Systems II: Analog and Digital Signal Processing.

[4] P. A. Gibbons and S. S. Muchnick, Efficient instruction scheduling for a pipelined processor, in Proc. of the SIGPLAN Symposium on Compiler Construction (July 1986), pp. 11-16.

[5] J. R. Ellis, Building: a compiler for VLIW architectures, The MIT Press (1985).

[6] A. Nicolau, A fine-grain parallelizing compiler, Technical Report (Dec. 1986), Department of Computer Science, Cornell University.

[7] Gregory J. Chaitin et. al., Register allocation via coloring, Computer language (Jan. 1981), 6(1): pp. 47-57.

[8] P. Briggs, K. D. Cooper, and L. Torczon, Improvement to graph coloring register allocation, ACM Trans. On Programming Languages and Systems (May 1994), 12(4): pp. 501-536.

[9] A. Sudarsanam and S. Malik, Simultaneous reference allocation in code generation for dual data memory bank ASIPs, IEEE International Conference on Computer-Aided Design (Nov. 1995), San Jose, CA.

[10] J.-M.Chang and M. Pedram, Register allocation and binding for low power, IEEE/ACM Design Automation Conference (June 1995), San Francisco, CA.

[11] M. T.-C. Lee, V. Tiwari, S. Malik, and M. Fujita, Power analysis and minimization technique for embedded DSP software, IEEE Transactions on VLSI Systems (March 1997) , vol. 5, no. 1, pp. 123-133.