DSP Performance Maximization with Multisample Technique

* Hosun Lee, ** Lawrence K.W. Law, *** Youngyearl Han
* Motorola Korea Inc, ** Motorola Semiconductors Hong Kong Ltd., *** Department of Electronic Communication Engineering, Hanyang University, Seoul, Korea.
R10394@email.mot.com

ABSTRACT

In this paper, we present multisample DSP coding technique for StarCore, SC140 DSP. The multisample programming is a pipelining technique that exploits operand reuse both coefficients and variables within kernel. A coefficient or operand is loaded once from memory and then the value may be used by multiple ALUs. It is possible to evaluate one intermediate product from each of four output sample calculations in parallel. Therefore, parallelization has been achieved by processing multiple samples in parallel rather than multiple intermediate products belonging to only one sample. The benefits of decreasing the number of memory moves per sample is to increase the algorithm performance. In this paper, the multisample technique has been implemented in FIR filter calculation using Motorola StarCore DSP development tool.

I. INTRODUCTION

The StarCore 140 in Figure 1 is a low cost, low power, high performance, high flexibility programmable general purpose fixed-point CMOS DSP core with the 3rd generation DSP Archnitecture that efficiently deploys a novel Variable Length Execution Set (VLES) execution model utilizing maximum parallelism by allowing multiple Data and Address ALUs to execute multiple operations in a single clock cycle. A Data Arithmetic Logic Unit (DALU) performs arithmetic and logical operations on data operands in the Star*Core 140 core. The Star*Core 140 has 4 Arithmetic & Logic units in the DALU. Four instances of a single-cycle Multiplier-Accumulator (MAC) Unit with automatic saturation capability and four instances of a Bit Field Unit (BFU), each with a 40-bit barrel shifter capable of executing a variety of single-bit and multi-bit logic and shift operations.

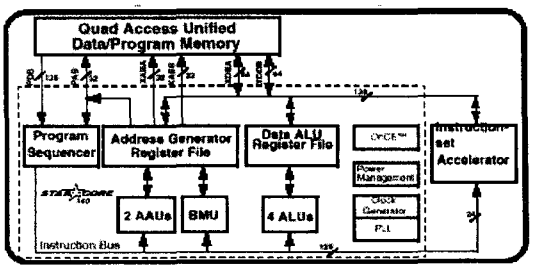
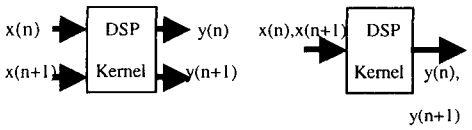


Figure 1. StarCore 140 Core Block Diagram

All the MAC and BFU Units can access all the DALU registers. Each register is partitioned into 3 portions: 2 sixteen bit registers (low and high portion of the register) and one 8 bit register (extension portion). The low and high parts of each register can be either inputs for the arithmetic operations, or as part of the 40 bit registers as output for the operation result.

II. MULTISAMPLE TECHNIQUE

Multisample programming is a pipelining technique that exploits operand reuse (both coefficients and variables) within the kernel. A coefficient or operand is loaded once from memory. The reuse of operands is similar to data caching and the register file acts as a data cache, allowing ALUs fast access to operands without accessing memory. The multisample programming technique can be applied to algorithms with alignment restrictions or bit-exact requirements. The multisample technique's reuse of operands relaxes the alignment requirements for loading operands. This allows simpler operand addressing, and effectively solves the problem of memory bus bandwidth, operand alignment, or limited algorithm parallelism when using multiple ALUs. Although not obvious, multisample algorithms provide the same bit-exact results as single-sample algorithms. This is possible because the algorithm performs the same exact operations, but with a different pipeline. This is important for algorithms requiring bit-exact compliance, such as speech coders. Multisample algorithms are ideal for block processing where data is buffered and processed in groups such as speech coders.



A.. Multiple Sample Algorithm B. Single Sample Algorithm

Figure 2. Multisample and Single Sample Kernels

III. IMPLEMENTATION

Implementing a DSP algorithm such as an FIR filter requires trade-offs between the number of samples processed simultaneously and the number of ALUs. As the Kernel computes more samples simultaneously, the number of memory loads decreases because data and coefficient values are being reused. However, to obtain this reuse, more intermediate results are required, which typically requires more registers in the processor architecture. It is theoretically possible to compute an algorithm faster by using more ALUs. To do this, some degree of parallelism is required in the algorithm to partition the computations. Although computing a single sample with multiple ALUs is theoretically possible, limitations in the DSP hardware may not allow this style of algorithm to be implemented. In particular, most processors typically require operands to be aligned in memory and multiple operand load and stores to be aligned. Let's consider the direct-form FIR filter for the example of common DSP kernel written in multisample form. Although the direct-form FIR filter is one of the simplest DSP kernels, it uses a majority of the features of the DSP architecture: input data samples and coefficients, a multiply-accumulate, and pointer arithmetic. To illustrate the point, consider the FIR filtering operation described by:

$$y(n) = \sum_{i=0}^{N-1} c(i)x(n-i), \qquad \text{for} \quad 0 \le n < L$$

Past input samples are multiplied by coefficients. The products are added together to form the output. In order to make use of the four ALUs, the operations can be grouped as illustrated in the following step.

Step1: Characterize the Algorithm

In characterizing the algorithm, four input samples are grouped together. Coefficients and delays are loaded and applied to all four input values to compute four output values. Using four ALUs reduces the execution

time of the filter to 25% of the execution time of a single ALU filter. The data flow for a quad-sample FIR filter is shown in Figure 3.

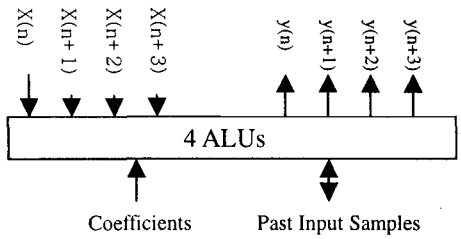


Figure 3. FIR Filter Data Flow

Step 2: Develop the Generic Kernel

This step is achieved by writing the equations for sample that is processed, and determining the reuse pattern. To develop the FIR filter equations for processing four samples simultaneously, the equations for the current sample y(n) and the next three output samples – y(n+1), y(n+2) and y(n+3) shown in Figure 4.

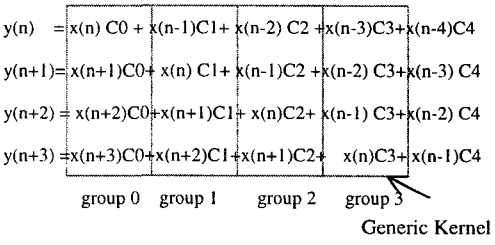


Figure 4. FIR Filter Equations for Four Samples

The generic kernel characteristics for the FIR filter include the following:

- The kernel utilizes four parallel MACs.
- One coefficient is loaded and used by all four MACs in the same generic kernel.
- One delay value is loaded, used by the generic kernel, and saved for the next three generic kernels.

Three delays are reused from the previous generic kernel. To develop the structure of the quad-ALU kernel, the filter operations are written in parallel and the loads are moved ahead of where they are first used.

Step 3: Determine the Requirements of the Generic Kernel

The generic kernel in this example requires four MACs and two parallel loads. In order to make use of the four ALUs, the operations can be grouped as illustrated in the following equation in Figure 4. The products and accumulations within each group are calculated in parallel but the groups themselves are evaluated in sequence. Therefore parallelization has been achieved by processing multiple samples in parallel rather than multiple intermediate products belonging to only one sample. Note also that when one group (e.g.Group 2) has been evaluated, only two words of data need to be loaded for the next group (Group 3): C3 and x(n-3). The other values needed for the calculations in Group 3 (x(n-2), x(n-1), and x(n)) should already exist in DSP registers from the calculation of Group 2. This results in reduced memory bandwidth requirements, thereby increasing the code efficiency.

Step 4: Rewrite the Generic Kernel

This step involves creating a basic kernel by rewriting the generic kernel to take maximum advantage of the resources of architecture. In the FIR filter example, assuming there are at at least four coefficients in the FIR filter, the generic kernel is replicated to create the basic kernel.

Step 5: Optimize the Basic Kernel

This step is achieved in the FIR example by folding the coefficient and delay loads. An important thing of this kernel is that only two data moves are required, yet all four ALUs maintain full operand bandwidth. The number if loop passes is reduced to 25% of the filter size to compensate for the generic kernel being

duplicated four times in the basic kernel. The total speed remains the same as the single instruction generic filter. The following is program example for single sample and multisample algorithm.

A. FIR Filter Program using single sample algorithm

```
main()
{long L_acc;   short m,n,i;
for(m=0,n=12; n<112; n++,m++)
  { L_acc = 0;
for(i=0; i<12; i++)
      {
L_acc += a[i] * x[n-i];              }
y[m] = (short)L_acc;
   }
}
```

B. FIR Filter Program using Multisample algorithm

```
main()
{long L_acc0=0,L_acc1=0,L_acc2=0, L_acc3=0;
 short m,n,i;
 short V_d0,V_d1,V_d2,V_d3;
for(m=0,n=12; n<112; n++,m+=4)
  {
V_d0 = x[n];V_d1 = x[n+1];
V_d2 = x[n+2];V_d3 = x[n+3];
for(i=0; i<12; i+=4)
 {
L_acc0 += a[i] * V_d0;   L_acc1 += a[i] * V_d1;
L_acc2 += a[i] * V_d2;L_acc3 += a[i] * V_d3;
 V_d3 = x[n-i];
L_acc0 += a[i+1] * V_d3;   L_acc1 += a[i+1] * V_d0;
L_acc2 += a[i+1] * V_d1;   L_acc3 += a[i+1] * V_d2;
 V_d2 = x[n-i+1];
L_acc0 += a[i+2] * V_d2; L_acc1 += a[i+2] * V_d3;
L_acc2+=a[i+2]*V_d0;   L_acc3   +=   a[i+2]*V_d1;
V_d1 = x[n-i+2];
L_acc0 += a[i+3] * V_d1;   L_acc1 += a[i+3] * V_d2;
L_acc2+=a[i+3]*V_d3;L_acc3+=a[i+3]*V_d0;
}
y[m] = (short)L_acc0;  y[m+1] = (short)L_acc1;
y[m+2] = (short)L_acc2; y[m+3] = (short)L_acc3;
            }
}
```

IV. CONCLUSION

As an advantage of the technique, it preserved the bit-exact output results with high level of parallelism which is better than the conventional split summation technique. It is possible to eliminate the register-to-register transfers by expanding the inner loop such that each group of four MAC instructions uses the data registers that already contain the required data values. This yields faster code but has greater code size. After simulating using Motorola StarCore DSP simulator, the summary of main multisample technique characteristic is in the table 1.

Table 1 Inner Loop Characteristic of Multi-sample technique

Characteristics	Single-sample Algorithm	Multi-sample Algorithm
Cycle count	N	N/4
Registers used	Fewer	More
Sample delay	1	4
Number of memory moves	2N	N/2
Code size	Small	Large

REFERENCES

1. Star*Core 140 Architecture Functional Specifications Rev.0.63
2. Star*Core Application Note: Multisample programming technique
3. SC140 DSP Core Reference Manual