Freescale Semiconductor
Application Note

DSP56800WP1
Rev. 1, 7/2005

Novel Digital Signal Processing Architecture with Microcontroller Features

JOSEPH P. GERGEN
DSP Consumer Design Manager
PHIL HOANG
DSP Consumer Section Manager
EPHREM A. CHEMALY Ph.D .
DSP Applications Manager

1. Abstract

Traditional Digital Signal Processors (DSPs) were designed to execute signal processing algorithms efficiently. This led to some serious compromises between developing a good DSP architecture and a good microprocessor architecture. For this, as well as other reasons, most DSP applications used a DSP and a microcontroller. This paper presents a new 16-bit DSP architecture from Freescale that maintains the performance of the DSP, while adding microcontroller functionality.

2. Introduction

2.1 Overview

DSPs are dedicated processors, designed to execute signal processing algorithms efficiently. Even though DSPs are specialized microprocessors, they need to execute many kinds of DSP algorithms. In addition, these DSPs are often called upon to execute traditional microcontroller code. To resolve this problem, designers use a DSP and a microcontroller in their system. This adds to the material cost of their products. This paper will present a new architecture, that is well suited for general purpose DSP algorithms, as well as efficient microcontroller code and compiler performance.

Contents

3. Background

3.1 Overview

In the early 1980’s general purpose DSPs made their entrance into the merchant market. The first generation DSPs were expensive, and designers were trying to find applications suited for them. The applications development process was long and required specialized skills. The developers needed to have both digital signal processing background, as well as understanding the idiosyncrasies of programming the DSPs. The tools used for development were primitive, and almost all the code was written in assembly language. By the end of the ‘80s, DSPs were well established in specific markets and were making inroads into traditional microcontroller markets. The rules of engagement were changed, and customers were looking for a more mature product.In response to the needs of the customer, Freescale has developed a new architecture suited not only for efficient DSP processing, but also for high performance control. This paper will describe the new low cost processor family-the DSP56800.

4. Introduction tothe 56F800 Family

The DSP56800 family is a group of chips built around the DSP56800 16-bit fixed point DSP microcontroller Central Processing Unit (CPU) core. This core is designed for both efficient DSP and controller operations. Its instruction set efficiency as a DSP microprocessor is on par with the best general purpose DSP architectures, and it also has been designed for efficient, straightforward coding of controller-type tasks. The general purpose MCU-style instruction set, with its powerful addressing modes and bit manipulation instructions enables a user to begin writing code immediately without having to worry about the complexities associated with former DSP microprocessors. A true software stack allows for unlimited interrupt and subroutine nesting, as well as support for passed parameters and local variables. The experienced DSP programmer sees a powerful DSP instruction set with many different arithmetic operations and flexible single and dual memory moves that can occur in parallel with an arithmetic operation. Compilers are efficiently implemented on the DSP56800 architecture due to the general purpose nature of its instruction set.

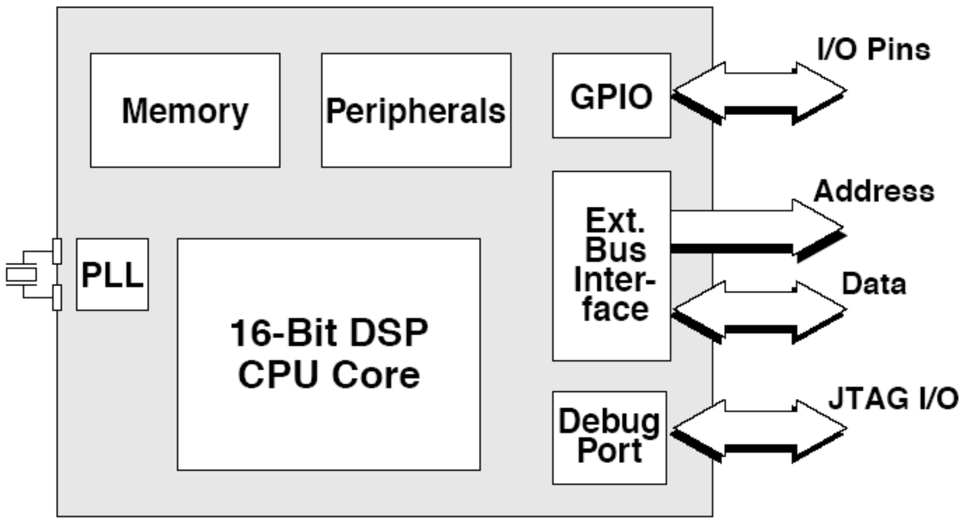


Figure 4-1. DSP56800 Based DSP Microcontroller Chip

Different memory configurations can be built around the DSP56800 core. Likewise, a variety of standard peripherals can be added around the DSP56800 core, (Figure 1) such as serial ports, general purpose timers, realtime and watchdog timers, and General Purpose Input/Output (GPIO) ports. Each peripheral interfaces to the DSP core through a standard Peripheral Interface Bus. This bus allows easy hookup of standard or custom designed peripherals. On-Chip Emulation (OnCE[a]) capability is provided through a debug port conforming to the JTAG standard. This provides realtime embedded system debug with OnCE capability through the 5-pin JTAG interface, allowing for hardware and software breakpoints, display and modification of registers and memory locations, and single stepping or step through multiple instructions in an application.

The high performance DSP features, flexible parallel moves, multiple internal buses, an external bus interface, on-chip program and data memories, standard peripherals, and a JTAG debug port make the DSP56800 family an excellent solution for realtime embedded control tasks. It becomes an excellent fit for wireless or wireline DSP applications, digital control, and controller applications in need of more processing power.

4.1 DSP56L811 16-bit Chip Architecture

The first chip available in the DSP56800 family is the DSP56L811. In addition to peripherals useful for signal processing, it also includes a complement of peripherals, GPIO pins, and timers useful for controlling an application. Its features include the following:

56800 Features

- 56800 core
- Phase Lock Loop (PLL)
- 1 K ´ 16-bit Program RAM
- Three General 16-bit Timers
- 64 ´ 16-bit bootstrap ROM
- Realtime Timer
- 2 K ´ 16-bit X-data RAM
- Computer Operating Properly (COP) Timer
- External Bus Interface
- Two SPIs for MCU interfacing
- JTAG/OnCE debug port
- Synchronous Serial Interface (SSI) for codecs
- Two external interrupts
- Sixteen dedicated GPIO pins
- Programs can run out of X-memory
- Sixteen additional multiplexed GPIO pins
- Five Low Power modes
- Interrupt available on eight GPIO pins
- 2.7 V to 3.6 V operation
- 100-pin QFP (0.5 mm)

A block diagram of the DSP56L811 is shown on the following page in **Figure 4-2**.

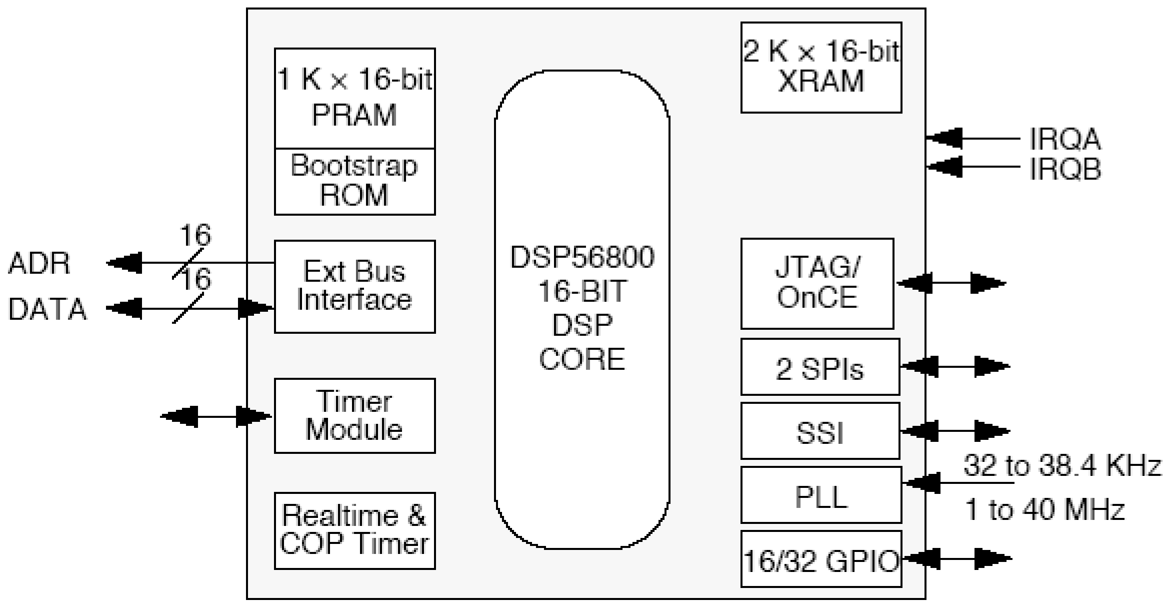


Figure 4-2. DSP56L811 Functional Block Diagram

5. 56800 16-BIT DSC Core Architecture

The DSP56800 core is a programmable CMOS 16-bit DSP designed for efficient real-time digital signal processing and general purpose computing. The DSP56800 core is composed of four functional units that operate in parallel to increase throughput of the machine. The functional blocks-the program controller, Address Generation Unit (AGU), Data Arithmetic Logic Unit (Data ALU), and bit manipulation unit-each contain their own register set and control logic so that they may operate independently and in parallel with the other three. Each functional unit interfaces with other units, with memory, and with memory-mapped peripherals over the core's internal address and data buses. Thus, it is possible for the Program Controller to be fetching a first instruction, the Address Generation Unit to generate up to two addresses for a second instruction, and the Data ALU to perform a multiply in a third instruction. Alternatively, it is possible for the bit manipulation unit to perform an operation in the third instruction described above in place of an operation in the Data ALU. The architecture is pipelined to take advantage of the parallel units and significantly decrease the execution time of each instruction.The major components of the DSP56800 core, shown in **Figure 5-1**, are the following:

- Data ALU
- Address Generation Unit (AGU)
- Program controller and hardware looping unit
- Bit manipulation unit
- Three internal address buses
- Four internal data buses
- OnCE debug port
- Clock generation circuitry

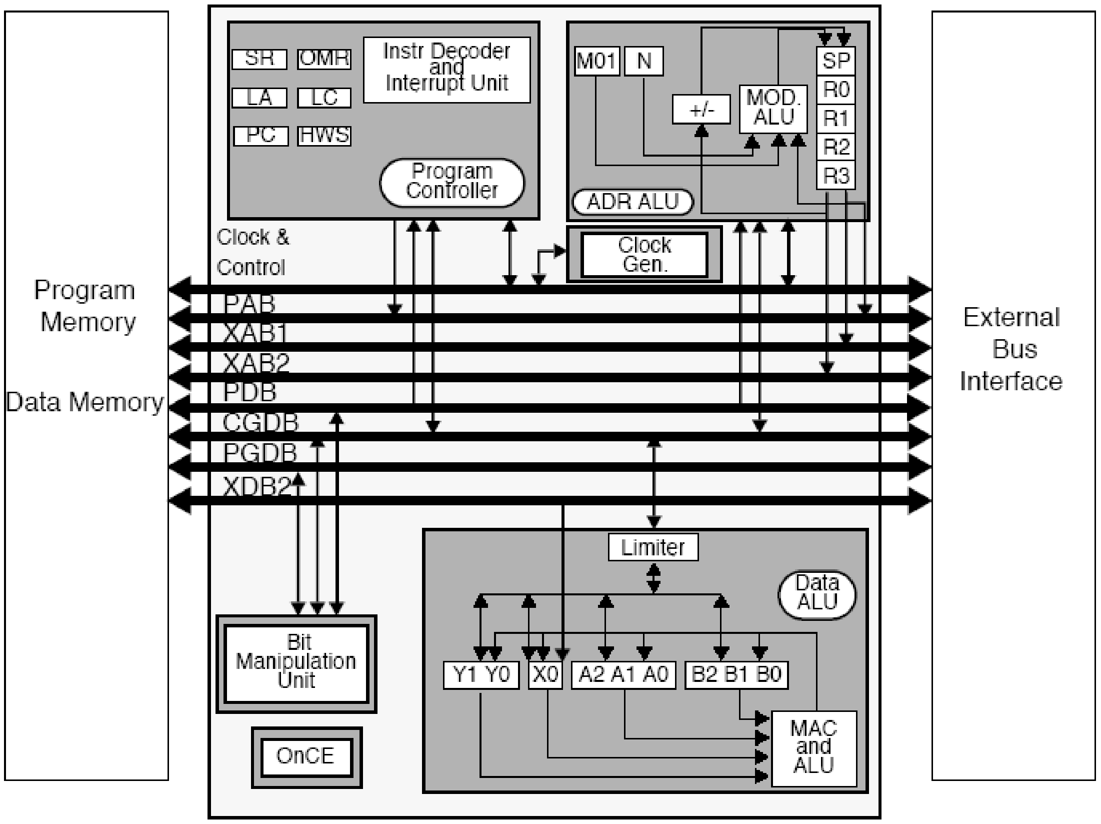


Figure 5-1. DSP56800 16-bit DSP Core Functional Block Diagram

The architecture of the DSP56800 core has been streamlined and tuned for efficient DSP processing, compact DSP and controller code size, and excellent compiler performance. Several of the high performance signal processing features are described the next section. The bulleted list on the following page lists the features of the DSP core.

- 30 Million Instructions Per Second (MIPS) with a 60 MHz clock at4.57 V-5.5 V
- 20 Million Instructions Per Second (MIPS) with a 40 MHz clock at 2.7 V-3.6V
- Parallel instruction set with useful DSP addressing modes
- Single-cycle 16 ´ 16-bit parallel Multiplier-Accumulator (MAC)
- 2 ´ 36-bit accumulators, including extension bits
- Single-cycle 16-bit barrel shifter
- Hardware DO and REP loops
- Three 16-bit internal core data buses and three 16-bit internal address buses
- One 16-bit Peripheral Interface Data Bus
- Instruction set supports both DSP and controller functions
- Controller style addressing modes and instructions for smaller code size
- Efficient ‘C’ Compiler and local variable support
- Hooks on core for 1 Mbyte program address space
- Software subroutine and interrupt stack with unlimited depth

6. High Performance DSP Features on a Low Cost Architecture

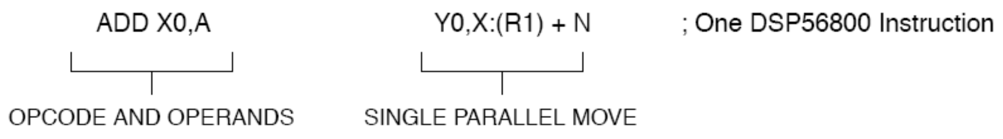
There are four key attributes of a powerful digital signal processing engine:

- High bandwidth parallel memory transfer capability
- An AGU that adequately supports the parallel memory transfers and provides DSP addressing modes
- Powerful computation unit with an adequate register set for fast algorithm calculation
- Hardware looping mechanisms for looping with no penalty in performance

The DSP56800 architecture is strong in all four key attributes. Each is presented below.

6.1 DSP56800 Family Parallel Moves

For any high performance computation engine, such as a Digital Signal Processor, it is critical that data is fed to and from the computation unit at a high bandwidth so that the computation unit is kept busy and the data transfers in and out of the unit are not a bottleneck. This processing bottleneck can be avoided with a flexible set of parallel move instructions' instructions that allow memory accesses to occur in parallel with operations in the computation unit. Two types of parallel moves are permitted' the single parallel move and the dual parallel read. Both of these are extremely powerful for DSP algorithms and numeric computation. All DSP56800 instructions with parallel moves execute in one instruction cycle and occupy one word of program memory.The single parallel move allows an arithmetic operation and one memory move (read or write) to be completed with one instruction in one instruction cycle. For example, it is possible to execute in one instruction an addition of two numbers while writing a value from a Data ALU register to memory:

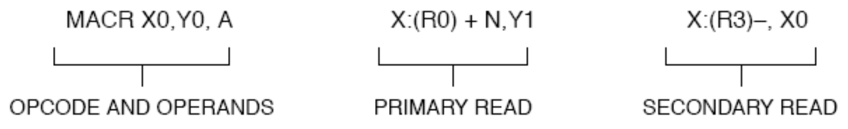


Note that an address calculation is also simultaneously performed in the Address Generation Unit. Below are some examples of single parallel move instructions.

```
; Examples of instructions with single parallel moves (1 program word, 1 instruction cycle):

      MPYR   A1,Y0,BX:(R0)+,X0   ;      Multiply w/ Rounding & read from memory
      MAC    -Y0,Y1,AY0,X:(R1)+  ;      Mult-Acc w/ Inversion of product & write to memory
      ADD    A,B   X:(R2)+N,X0   ;      Add & read from memory, post updating by register N
      TFR    Y1,A  A,X:(R3)+     ;      Move into A while writing previous contents to memory
      INCW   B     X:(R0)+,A1    ;      Increment accumulator and read from memory
      ASL    A     X:(R1)+,B     ;      Shift accumulator and read from memory
```

The dual parallel read allows an arithmetic operation to occur and two values to be read from X-data memory with one instruction in one instruction cycle. For example, it is possible to execute in one instruction a multiplication of two numbers and accumulation with a third with rounding of the result while reading two values from X-data memory to two of the Data ALU registers:



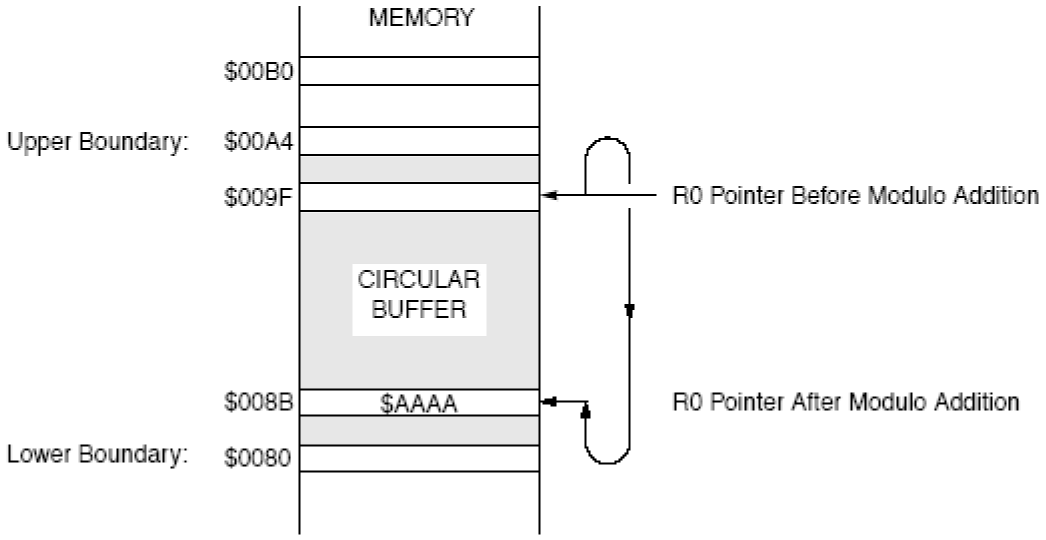
Note that two address calculations are simultaneously performed in the Address Generation Unit. Below are some examples of dual parallel read instructions.

```
; Examples of instructions with dual parallel reads (1 program word, 1 instruction cycle):
       MPYR   X0,Y1,A     X:(R0)+,Y0  X:(R3)+,X0    ;  MPY w/ Rounding w/ dual reads
       MAC    Y0,Y1,A     X:(R1)+,Y1  X:(R3)-,X0    ;  Mult-Acc w/ dual reads
       ADD    X0,B        X:(R0)+,Y0  X:(R3)+,X0    ;  Add w/ dual reads
       SUB    Y1,A        X:(R0)+,Y1  X:(R3)-,X0    ;  Subtract w/ dual reads
       MOVE               X:(R1)+,Y0  X:(R3)+,X0    ;  Read two values from memory
```

The parallel move capability of the DSP56800 architecture is good at providing the memory bandwidth required for the Data ALU because it allows up to two memory accesses in parallel with a single cycle computation. Its use is not restricted to single instruction loops, as found on other low cost architectures. Instead, any of the parallel moves can be used in hardware DO loops, hardware REP loops, and even outside of loops, if desired. This is because the DSP56800 architecture is capable of three memory accesses in a single cycle-one to fetch an instruction from program memory and two to access data memory.

6.2 56F800 Family Address Generation (AGU)

The Address Generation Unit (AGU) of the DSP56800 is the block where all address calculations are performed. It contains two arithmetic units and its own register set so that up to two addresses can be provided to data memory with two address updates in a single cycle. In the cases where the AGU generates two addresses to access X data memory, the program controller generates a third address used to concurrently fetch the next instruction.When an arithmetic operation is performed in the AGU, it can be performed using either linear or modulo arithmetic. Linear arithmetic is important for general purpose address computation, and modulo arithmetic allows the creation of data structures in memory such as FIFOs (queues), delay lines, circular buffers, and stacks' data is manipulated by updating address registers (pointers) rather than moving large blocks of data.Linear arithmetic is the case where address arithmetic is performed using normal 16-bit two's complement linear arithmetic. Modulo arithmetic is used when it is necessary to set up and step through a circular buffer in memory. Modulo arithmetic is similar to linear arithmetic, but if the result of an effective address calculation would be larger than the last address in a buffer, then an automatic wraparound is performed in the calculation. Similarly for the case where the result of an effective address calculation calculates an address that would be smaller than the first address in a buffer again, automatic wraparound is performed in the address calculation. An example of the modulo arithmetic capability of the DSP56800 family is shown in Figure 4 below. Note that the DSP56800 correctly wraps the address calculation even if the calculation does not land right on the upper or lower boundary of the circular buffer.



```
MOVE        #$24,M01    ; M01 register sets up a circular buffer size of 37
MOVE        #$9F,R0     ; R0 pointer located near upper boundary of buffer
MOVE        #$11,N      ; N goes several locations past upper boundary
MOVE        #$AAAA,X0   ; Value to be written to Wrapped Address
MOVE        X0,X:(R0+N) ; R0 pointer correctly wraps around to $008B
```

Figure 6-1. Figure 4 DSP56800 Modulo Arithmetic Example

The DSP56800 AGU provides the capabilities needed for high performance signal processing calculations by providing up to two data memory addresses per cycle in parallel with a third program memory address generated by the program controller, and by providing the types of addressing, such as modulo arithmetic needed for signal processing.

6.3 DSP56800 Family Computation - the Data ALU Unit

When examining the computation unit of a processor, it is important to examine two different aspects-the manner in which operands are accessed and stored by the unit, and the computation capabilities of the unit.Previous DSP architectures are accumulator based. This means that operands are provided from one or two different sources, but the results of an operation are always stored in an accumulator. Operations are performed so that one operand must always be an accumulator, except in the case of multiplication, where an accumulator is not allowed as one of the multiplier inputs. Some low cost DSP architectures provide a single accumulator, others provide two.

The DSP56800’s Data ALU unit, on the other hand, provides more registers and is organized in a more orthogonal nature. This permits the results of arithmetic operations to be written back to any of the Data ALUÕs five registers. In addition, the Data ALU input allows for immediate value operands. This significantly increases the power of the register set, because the orthogonal nature now allows the storage of frequently accessed variables in an algorithm to reside directly in registers, while still providing other registers for arithmetic computations. Accumulators may be used as inputs to the multiplier, in addition to being used for accumulation. Memory accesses are also reduced using this technique because intermediate results do not need to be temporarily stored to memory.

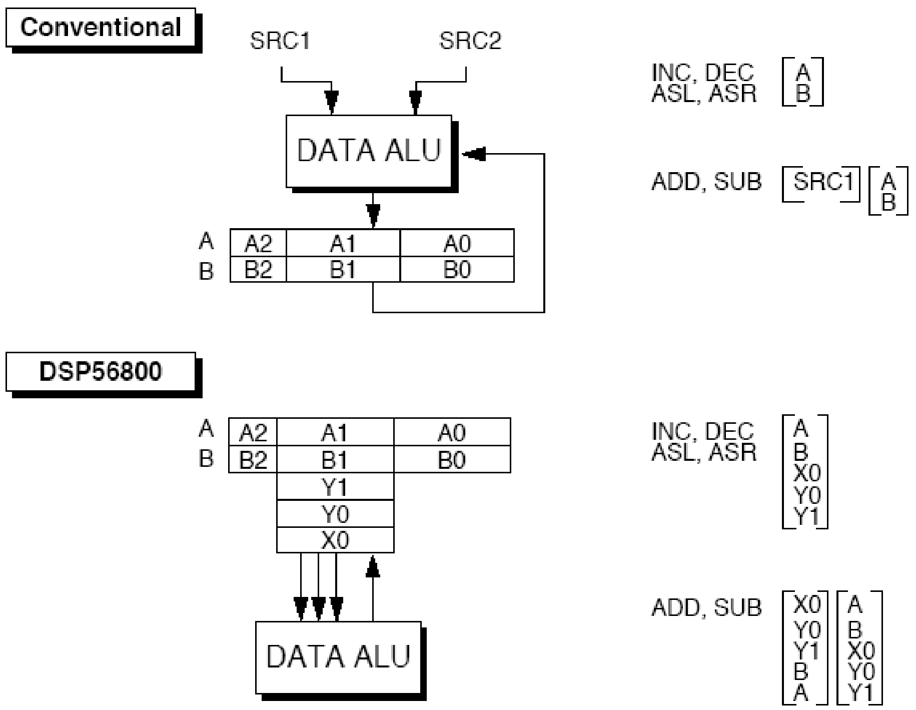


Figure 6-2. Comparison of Different DSP Computation Unit Structures

A comparison of the two techniques is shown in Figure 5 . The conventional technique, for example, can only increment one or two accumulators, whereas the technique allowed on the DSP56800 allows increments on any register. Note that it is also possible to increment and decrement memory locations directly on the DSP56800 architecture.In addition to its powerful register set, the Data ALU is very powerful for algorithm calculation because it can perform so many single cycle operations. The Data ALU can be used to perform fractional and integer arithmetic, as well as signed, unsigned, and multiprecision arithmetic. Multiple buses are located in the Data ALU so that complex arithmetic operations such as a multiply-accumulate can be performed in parallel with up to two memory transfers.

The Data ALU can perform any of the following operations in a single instruction cycle:

- Multiplication (with or without rounding)
- Multiplication with inverted product (with or without rounding)
- Multiplication and accumulation (with or without rounding)
- Multiplication and accumulation with inverted product (with or without rounding)
- Addition and subtraction
- AND, OR, and Exclusive-OR
- Compares
- Arithmetic and logical shifts
- Increments and decrements
- 16-bit barrel shifting

- Arithmetic shifts
- Rotates
- Two’s complement (negation)
- One’s complement
- Rounding
- Conditional register moves (T_{cc})
- Absolute value
- Saturation (limiting)
- Division iteration
- Normalization iteration

The Multiplier-Accumulator of the DSP56800 is not pipelined, so that a result of a multiplication or multiply-accumulate is available after one instruction cycle instead of two, as found on DSP architectures with pipelined MAC units. Note that any of the above multiplications or multiply-accumulates can be performed on two operands, or can be used to ‘square’ the value of a single operand.

6.4 DSP56800 Family Looping Mechanisms

Programs for DSP or numeric computation are often such that much of the processor execution is spent in small, numerically intensive computation loops with a lot of memory traffic. For this reason, it becomes apparent why it is necessary to have a powerful computation unit supported by a powerful register set and flexible set of parallel moves.

In addition, it is also critical that the execution time due to looping itself is minimized. The optimal solution for a DSP architecture is to have a hardware looping mechanism that automatically performs the looping without adding any extra computation time, referred to as ‘no overhead looping’.

Traditional low cost DSP architectures have provided no overhead looping on a single instruction, referred to as a ‘REPEAT’ loop. This is useful for the simplest DSP algorithms, such as an FIR filter algorithm, but is not well suited to more complex algorithms, such as speech coders, or digital filters, such adaptive filters, IIR filters, or PID controllers. In addition, these are more difficult to use because they are not interruptible, significantly increasing interrupt latency. Other DSP architectures have provided hardware ‘DO’ loops, which provide looping on up to 15 instructions, but again are non-interruptible. For this type of loop, instructions may take additional cycles to execute the first time through the loop or if executed outside of a hardware loop.

The DSP56800 architecture provides the most flexible hardware looping mechanism by providing a hardware ‘DO’ looping mechanism, which can loop on any number of instructions without adding any execution time, and is interruptible. An immediate value or register value can be used to specify the loop count. In addition, the DSP56800 also provides a REPEAT loop capability, which can be efficiently nested within the DO loop mechanism. The DSP56800’s looping mechanism provides the fastest interrupt latency, and instructions do not take additional cycles on the first pass through a loop or if executed outside of the loop.

An example of the DSP56800 code for a cascaded set of IIR filters is shown on the following page. This example demonstrates the parallel move and hardware looping capabilities of the DSP56800. It uses twelve words of program memory, and executes in 6N + 7 instructions, where N is the number of cascaded filters.

```
; N Cascaded Biquad IIR Filters (Direct Form II)
;
;          Many digital filter design packages generate coefficients for Direct Form II
;          IIR filters. Often, these coefficients are greater in magnitude than 1.0.
;          This implementation is suitable for IIR filters with coefficients greater in
;          magnitude than 1.0, because it allows the user to simply divide all coefficients
;          generated by 2.

;          w(n)/2 = x(n)/2 - (a1/2) * w(n-1) - (a2/2) * w(n-2)
;          y(n)/2 = w(n)/2 + (b1/2) * w(n-1) + (b2/2) * w(n-2)

;          D High Memory Order - w(n-2)1,w(n-1)1,w(n-2)2,w(n-1)2,...
;          D Low Memory Order - (a2/2)1,(a1/2)1,(b2/2)1,(b1/2)1,(a2/2)2,...

           MOVE       #-1,N                            ;1         1
           MOVE                  X:INPUT,A             ;1         1
           ASR        A                     X:(R3)+,X0 ;1         1          X0=a2/2
           MOVE                  X:(R0)+,Y0            ;1         1          Y0=wn-2
           DO         #N,LABEL                         ;2         3
           MAC        Y0,X0,A    X:(R0)+N,Y1X:(R3)+,X0 ;1         1          y1=wn-1
           MAC        Y1,X0,A    Y1,X:(R0)+            ;1         1
           ASL        A                     X:(R3)+,X0 ;1         1          X0= b2/2
           ASR        A          A,X:(R0)+             ;1         1          X0=b1/2
           MAC        Y0,X0,A               X:(R3)+,X0 ;1         1
           MAC        Y1,X0,A    X:(R0)+,Y0 X:(R3)+,X0 ;1         1
LABEL
;                                                      ;_________________
;                                           Total:     12         6N+7
```

7. General Purpose Computing-Ease of Programming

The DSP56800 architecture is significantly easier to program than any other previous DSP architecture. This is because it was designed from the ground up not only as an efficient signal processing engine, but also as an efficient, easy to program controller. The general purpose features that make it easy to program in assembly code will also allow for a very efficient DSP56800 compiler with excellent code density.

Several factors contribute to the programming ease and efficiently of the DSP56800:

- A programming model with a generous and flexible set of registers
- An instruction set with immediate data, memory, and register-oriented instructions
- A complete and orthogonal set of MOVE instructions with a full set of addressing modes, many of which have never been available on a DSP architecture
- Efficient hardware and software looping techniques
- A software stack with a true stack pointer
- Efficient support of structured programming techniques

7.1 DSP56800 Programming Model

The programming model for the DSP56800 core, shown in Figure 6 , is separated into three different sets of registers corresponding to the three functional units within the DSP core. Each functional unit has a full set of registers for performing its tasks.

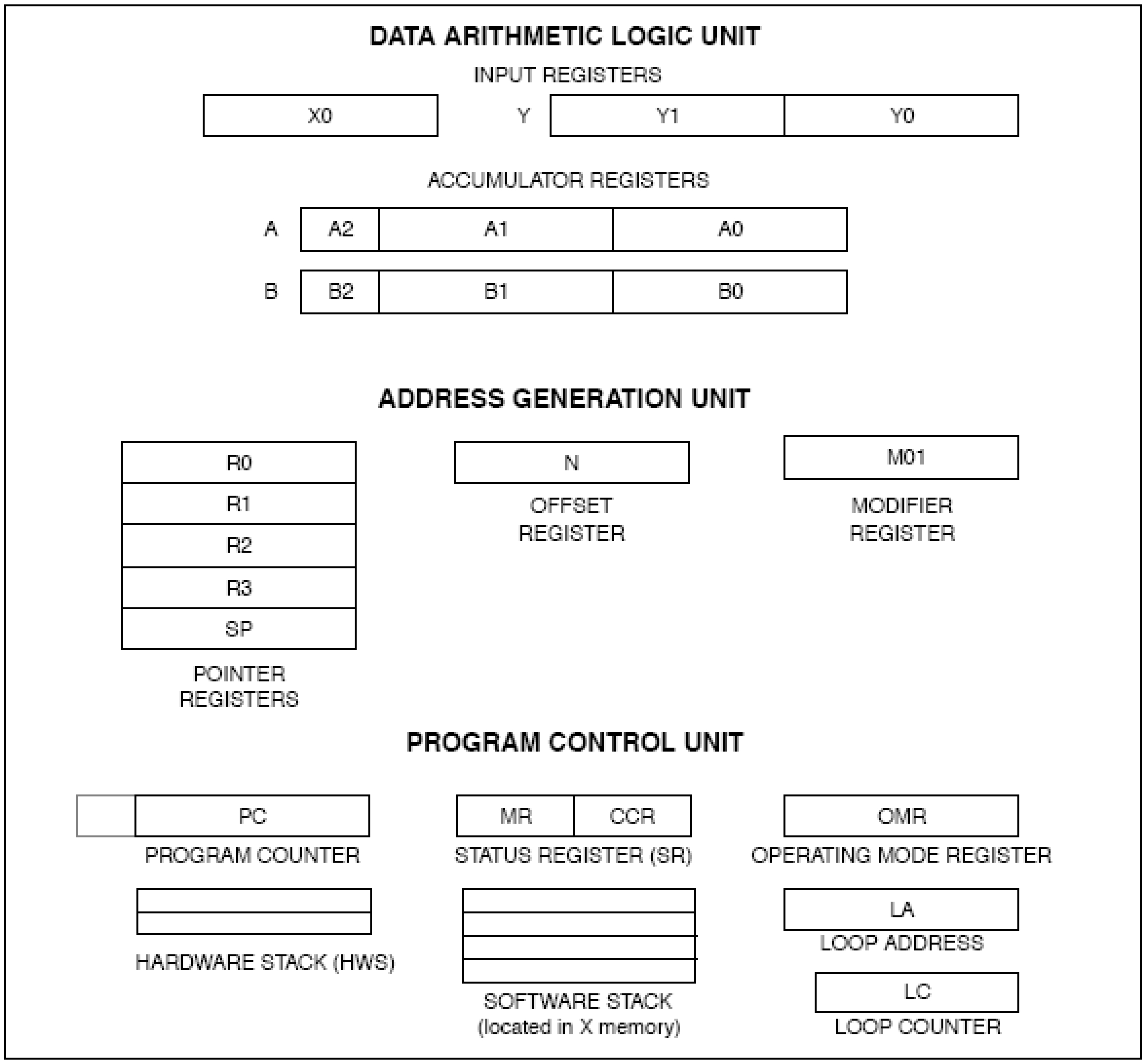


Figure 7-1. Figure 6 DSP56800 Core Programming Model

7.2 Instructions That Operate on Registers, Immediate Data, or Memory

In addition to a complete register set, the DSP56800 architecture is further enhanced by an instruction set that is not only register oriented, but can also operate directly with immediate data or on memory. The use of immediate data, for example, helps to decrease register usage because arithmetic operations can be performed directly with immediate data, allowing the registers to store important variables and intermediate results instead. The following code shows all the different addition instructions available on the DSP56800 architecture.

```
ADD    <register>,<register>     ; Register Oriented
ADD    #xx,<register>            ; Short Immediate Data
ADD    #xxxx,<register>          ; Long Immediate Data
ADD    X:<aa>,<register>         ; Direct Addressed Memory Location
ADD    X:xxxx,<register>         ; Absolute Addressed Memory Location
ADD    X:(SP-xx),<register>      ; Location addressed on the stack
```

The DSP56800 provides a full set of logical and arithmetic instructions, complemented by a 16-bit barrel shifter and unsigned arithmetic support. All instructions are directly accessible-there is no need to set any mode bits to modify the operation of an instruction.

7.3 The DSP56800's MOVE Instruction and Addressing Modes

A powerful set of MOVE instructions is found on the DSP56800 core, the most general set of MOVE instructions and addressing modes found on any DSP architecture. This not only eases the task of programming the DSP, but also decreases the program code size and improves the efficiency. The MOVE instructions available on the DSP chip include:

```
MOVE   <any_DSPcore_register>,<any_DSPcore_register>

MOVE   <any_DSPcore_register>,<X_Data_Memory>
MOVE   <any_DSPcore_register>,<On_chip_peripheral_register>
MOVE   <X_Data_Memory>,<any_DSPcore_register>
MOVE   <On_chip_peripheral_register>,<any_DSPcore_register>

MOVE   <immediate_value>,<any_DSPcore_register>
MOVE<immediate_value>,<X_Data_Memory>
MOVE   <immediate_value>,<On_chip_peripheral_register>
```

The DSP56800 has a complete set of addressing modes never before found on DSP architectures. For any MOVE instruction accessing X data memory or an on-chip memory mapped peripheral register, eight different addressing modes are supported. Additional addressing modes are available on a subset of frequently accessed DSP core registers, including the registers in the Data ALU, and all the pointers in the Address Generation Unit. The addressing modes include:

Table 7-1. Addressing Modes

• Indirect-No Update	• Immediate Data (16 bits)
• Indirect-Post Increment	• Short Immediate Data (7 bits)
• Indirect-Post Decrement	• Absolute Addressing (16 bits, Extended)
• Indirect-Post updated by a register	• Absolute Short Addressing (6 bits, Direct)
• Indirect-Indexed by 16-bit offset	• Peripheral Short (6 bits, Direct)
• Indirect-Indexed by 6-bit offset	• Register direct
• Indirect-Indexed by a register	• Inherent

7.4 Looping on the DSP56800 Architecture

In addition to the DSP56800 hardware DO looping capability previously discussed, which is not only interruptible, but also allows a nested REPEAT loop, software looping is also efficiently supported on the DSP56800 architecture.Previous DSP architectures have supported software looping using registers, which are often in short supply for an entire application. This was necessary because of the lack of a powerful hardware looping mechanism. The DSP56800 can implement software loops using registers in either the AGU or Data ALU for the loop counter. More useful, however, is the ability of the DSP56800 to implement loops using a memory location for the loop count. This is perhaps the most useful setup because the full register set is then available for algorithm computation. An example of a software loop using a memory location is shown below.

```
; Software Looping - Memory Location (one of first 64 XRAM locations) Used for Loop Count
      MOVE   #3,X:$7       ; Load loop count to execute the loop 3 times
LABEL                      ; Enters loop at least once
        •
        •
        •


      DECW   X:$7
      BGT    LABEL         ; Back to Top-of-Loop if positive and not zero
```

7.5 DSP56800 Structured Programming and Software Stack

Instead of the hardware stack found on most DSP architectures, the DSP56800 implements its stack using a true stack pointer in memory. Not only does this allow for unlimited nesting of subroutines and interrupts, but this also supports structured programming techniques typically found only on high end controllers, such as parameter passing to subroutines and local variables. The software stack, a dedicated stack pointer, and the new addressing modes of the DSP56800 allow for an efficient implementation. These techniques can be used for both assembly language programming, as well as high level language compilers.

Parameters can be passed to a subroutine by placing these variables on the software stack immediately before performing a JSR to the subroutine. Placing these variables on the stack is referred to as building a “stack frame.” These passed parameters are then accessed in the called subroutines using the stack addressing modes available on the DSP56800. This is demonstrated in the code example, which builds the stack frame shown in Figure 7 on page 22.

In a similar manner, it is also possible to allocate space and to access variables that are locally used by a subroutine, referred to as local variables. This is done by reserving stack locations above the location that stores the return address stacked by the JSR instruction. These locations are then accessed using the DSP56800’s stack addressing modes, as demonstrated on the previous page. For the case of local variables, the value of the stack pointer is updated to accommodate the local variables. For example, if five local variables are to be allocated, then the stack pointer is increased by the value of five to allocate space on the stack for these local variables. When a large numbers of variables are allocated on the stack, it is often more efficient to use the (SP) + N addressing mode.

It is possible to support passed parameters and local variables for a subroutine at the same time. In this case, the program first pushes all passed parameters onto the stack. The JSR instruction is then executed, which pushes the return address and the status register onto the stack. Upon entering the subroutine, the first thing the subroutine does is to allocate space for local variables by updating the SP, ensuring that any writes to the SP register are always with even values. Then, both passed parameters and local variables can be accessed with the stack addressing modes.

```
;
; Example of Subroutine Call With Passed Parameters
;
      MOVE                X:$35,N          ; pointer variable to be passed to subroutine
      LEA     (SP)+       ; pre-increment stack pointer
      MOVE    N,X:(SP)+   ; push variables onto stack
      MOVE    X:$21,N     ; 1st data variable to be passed to subroutine
      MOVE    N,X:(SP)+   ; push onto stack
      MOVE    X:$47,N     ; 2nd data variable to be passed to subroutine
      MOVE    N,X:(SP)    ; push onto stack
      JSR     ROUTINE1
      POP                 ;remove the three passed parameters from stack when done
      POP
      POP


ROUTINE1
      MOVE    #5,N        ; allocate room for Local Variables
      LEA     (SP)+N
       •
       •
       •

      MOVE    X:(SP-9),R0 ; get pointer variable
      MOVE    X:(SP-7),B  ; get 2nd data variable
      MOVE    X:(R0),X0   ; get data pointed to by pointer variable
      ADD     X0,B
      MOVE    B,X:(SP-8)  ; store sum in 1st data variable
       •
       •
       •
      MOVE    #-5,N
      LEA     (SP)+N
      RTS
```

The software stack is also useful for providing a temporary variable, such as when swapping two registers, for saves and restores of registers before entering critical loops, in addition to the structured programming techniques and unlimited nesting previously described.

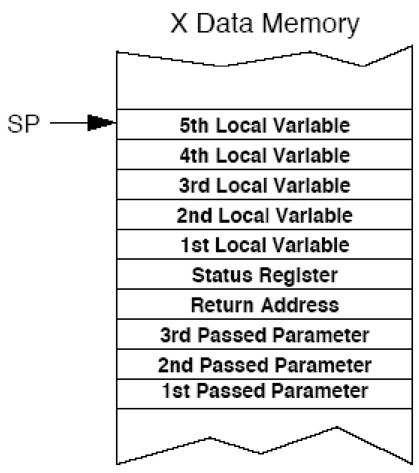


Figure 7-2. Example of a DSP56800 Stack Frame

7.6 Benefits in Program Code Size

Many features of the DSP56800 architecture contribute to a significant decrease in overall code size, not just for DSP programs, but for controller code as well. The ability for instructions to work directly on memory locations, as found with the bit manipulation instructions, the efficient looping mechanisms, the orthogonal set of move instructions and complete set of addressing modes, and the ability to load immediate values directly into memory locations, all contribute to reductions in program code size for general purpose computing. Code size is reduced for DSP algorithms due to the efficient set of parallel move instructions, the complete register sets of the DSP, and the ability to write results back to any of the Data ALU registers. Likewise, the ability to perform arithmetic operations directly with immediate data or memory locations also improves the code density.

8. Interrupt Processing

The interrupt unit on the DSP56800 uses a vectored interrupt scheme, which allows for fast servicing of interrupts, and is expandable to support future DSP56800 core-based designs. It currently supports thirteen different interrupt sources-seven interrupt channels for seven different on-chip peripherals, two external interrupts, and four interrupt sources from the DSP core. From these interrupt sources, execution can be vectored to any of up to sixty-four different interrupt vectors. Interrupt servicing automatically stacks and unstacks the Program Counter and Status Register, and nested interrupts can be supported. Each maskable interrupt source can be individually masked, or all maskable interrupts can be masked in the status register.

9. True Core-Based Design

The DSP56800 has been designed from its inception as a true CPU core from which many derivatives can be created. Its internal architecture incorporates many features necessary for efficient core-based design. As previously discussed, the DSP56800’s interrupt machine has been designed to support multiple on-chip peripherals on seven general purpose interrupt channels, in addition to the two external interrupts and the DSP core interrupts. Each peripheral may have more than one vectored interrupt. A DSP chip may have up to 128 on-chip peripheral registers. Each peripheral interfaces to the DSP56800 core on a standard Peripheral Interface Bus, which not only has a standard bus interface, but also a standard interrupt interface.The DSP56800 has been designed with industry standard design tools to enable users to later develop their own peripherals for connection to the DSP56800 core.

10. Achieving Low Power Designs

The DSP56800 core has been designed from its inception for low power consumption. Both architectural and circuit techniques are used to provide intelligent power management as the DSP is operating. The power management scheme automatically turns off unused blocks of the DSP.When considering power consumption, it is also important to remember that much of the processing actually occurs in tight numeric processing loops. The actual calculations required for the algorithms are where most of the execution time occurs, and as a result, where most of the power is consumed. From this observation, it becomes obvious that an architecture that is efficient at processing at numeric algorithms will burn considerably less power because significantly fewer instructions will be executed to complete a task. Lowest power consumption for an application not only requires an architecture designed for low power consumption, but also an architecture that is very efficient at performing DSP calculations. The high performance features on the DSP56800 architecture and its low power consumption give the DSP56800 excellent low power performance.The clocking on the DSP56800-based chips has also been carefully designed to reduce power consumption. It is possible to dynamically change the

DSP core’s clocking frequency with a Phase Lock Loop. An output clock pin can optionally be turned off if desired. The DSP also supports multiple low power Stop and Wait modes for significant reductions in power consumption while waiting for an event to occur. The five low power modes on the DSP56L811 chip include:

- All internal clocks gated off
- Oscillator is running, but PLL and DSP core and all peripherals are off
- Oscillator and timers are running, but all remaining circuitry is off
- Oscillator and PLL are running, but DSP core and peripherals are off
- Wait mode, where DSP core is off, but all peripherals continue functioning

11. Ease of Development

Development is straightforward with DSP56800-based chips. The external bus supports execution and debug of applications programs from external memory. It is possible to locate both data and programs externally-the chip simply inserts an additional cycle if an external program and external data access occur simultaneously. Programmable wait states may be individually programmed for external program and data memory to support the operation of slower memories.The assembly language for the DSP56800 is straightforward and very easy to support due to its general purpose nature. Likewise, the efficiency of the C Compiler now enables a user to develop significant portions of an application in C, while still leaving the numerically intensive computation routines in assembly language. Initial results show reductions of a third to a half in program code size compared to compilers for existing DSP architectures.

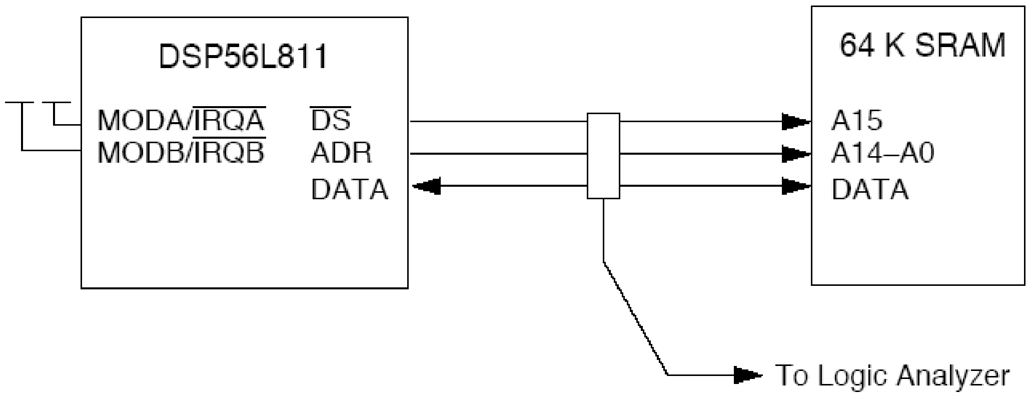


Figure 11-1. Example of Code Development with Visibility on All Memory Accesses

An on-chip debug port gives emulation capability on the chip even in a user’s target system through a 5-pin JTAG interface. Through this port, it is possible to set breakpoints, examine and modify register and memory locations, and perform other actions useful for debugging real systems. Freescale offers a full line of software and hardware Digital Signal Processor (DSP) development tools for rapid development and debugging of user applications and algorithms on the DSP56800 family. The development tools include the following:

- Relocatable Macro Cross Assembler
- Linker and Librarian
- C/C++ Cross Compiler

- Clock-by-Clock Multiple Chip Instruction Simulator
- Hardware Development using the Application Development System (ADS)
- Graphical User Interface (GUI) and Symbolic Debugger

These tools can be ordered to operate on: ISA-BUS™ IBM PCs™, SBUS™ SUN-4 Workstations™, or HP 7xx Computers, except the Compiler, which is not available for the HP platform. Freescale's DSP development tools can be obtained through a local Freescale Semiconductor Sales Office or authorized distributor.

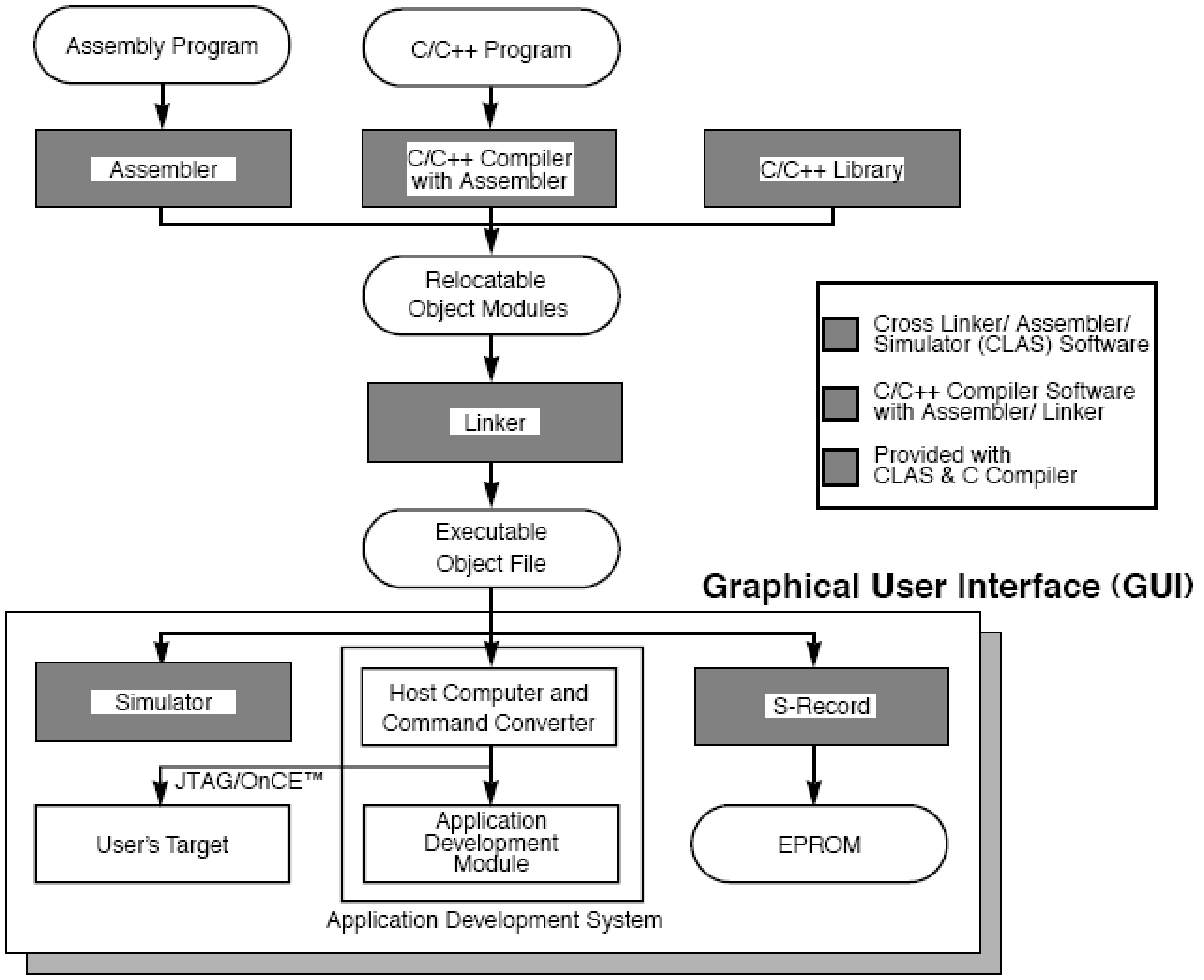

Assembly Program
C/C++ Program
Assembler
C/C++ Compiler with Assembler
C/C++ Library
Relocatable Object Modules
Cross Linker/ Assembler/ Simulator (CLAS) Software
C/C++ Compiler Software with Assembler/ Linker
Provided with CLAS & C Compiler
Linker
Executable Object File
Graphical User Interface (GUI)
Simulator
Host Computer and Command Converter
S-Record
JTAG/OnCE™
User's Target
Application Development Module
EPROM
Application Development System

Figure 11-2. Development Flow

The software development environment provides the programmer with a flexible, modular programming environment, giving full utilization to the DSP chips, a variety of data storage definitions, relocatability of generated code, symbolic debugging, and flexible linkages of object files. A library facility is included for creating archives of the final applications code.

12. Applications

The DSP56800 is targeted for cost sensitive applications. This DSP is well suited for consumer type applications that require low cost with moderate performance. This includes:

- Wireline and wireless modem
- Digital wireless messaging
- Digital answering machine/feature phones
- Servo and AC motor control
- Digital cameras

13. Results/Summary

The DSP56800 is a new DSP core architecture that provides not only efficient DSP processing, but is also powerful for controller applications. Its high performance DSP features and general purpose instruction set make this architecture a leader in the areas of low cost DSP performance, low power consumption, program code density that further reduces system costs by decreasing the amount of on-chip program memory required, and Compiler code density and performance.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

DSP56800WP1

Rev. 1
7/2005