IMPROVING Z893X1 DSP FAMILY MEMORY READ AND WRITE

A SIMPLE CHANGE IN THE BUS CONNECTION CAN DRAMATICALLY AFFECT THE PERFORMANCE OF THE Z893X1 DSP CHIP—EVEN WITHOUT A BARREL SHIFTER!

INTRODUCTION

The Barrel Shifter Problem

In applications requiring reading and writing to memory, cost constraints often dictate that a 16-bit DSP or 16-bit microcontroller will be connected to 8-bit-wide memory instead of the expected 16-bit-wide memory. Specifically, 8-bit-wide SRAM or 8-bit FLASH Memory prove to be cost effective and are widely used instead of 16-bit-wide SRAM or 16-bit FLASH Memory or two 8-bit-wide half-size SRAM or two 8-bit-wide half-size FLASH Memory.

Zilog's Z89321 and Z89371 chips (Z893X1 DSP family) provide for the favored system architecture where a 16-bit DSP or microcontroller connect to 8-bit memory. There is, however, one apparent drawback: the Z893X1 does not have a flexible Barrel Shifter. Without the Barrel Shifter, variable-length left shift cannot be executed in one instruction (one cycle). Each time data is stored (written) to and later retrieved (read) from memory, a conversion from word (16 bits) to two bytes (8 bits) and from two bytes to word must be performed. What is needed is an 8-bit logical left-shift operation; however, the Z893X1 DSP core does not have this instruction. The Z893X1 performs 8 one-bit, shift-left instructions, resulting in too many instructions and less than optimal performance in some cases.

Improving Performance without a Barrel Shifter

This app. note describes a new system architecture that dramatically improves the performance of reading (conversion from bytes to word) and writing (conversion from word to bytes) to memory for the Z893X1 DSP family—even though it does not have a Barrel Shifter. In this solution, one shift-left instruction replaces the need for 8 shift-left instructions, and the process of reading from the SRAM (or FLASH) and writing to the SRAM (or FLASH) takes only 14 instructions (cycles), instead of the usual 23 instructions—a dramatic 64 percent increase in performance.

A Real-Life Competition Scenario

A Zilog Field Area Engineer (FAE) recently had a customer with an application demanding very critical word-to-bytes (writing) and bytes-to-word conversions (reading). This particular customer was in a critical loop to determine the performance (operation frequency) of their system, and the customer was using this critical loop to determine DSP performance of Zilog's Z893X1 and the competition.

After evaluating their application and price range requirements, the decision came down to two chips: the Zilog Z89321 and one other chip from a competing DSP vendor. As usual, there are advantages and disadvantages in any competitive situation: the disadvantage of the Zilog Z89321 chip compared to the competitor's chip is the lack of a Barrel Shifter. The advantage of the Zilog chip is higher instruction rate (operation speed)—20 MIPS for the Z89321 or 16 MIPS for the Z89371 (OTP version of the Z89321), compared to a much lower 12 MIPS for the low-cost version of the competitor's chip.

The "Straightforward" Z893X1 Solution

In the straightforward customer solution using the Z89321, the process of reading from the SRAM and writing to the SRAM took 23 instructions (cycles). (Refer to the "Code_1: The Customer Original Z893X1 Assembler" code listing in the Source Code Listings section at the conclusion of this app. note.) Each A-D sample requires 20 passes through this code. The Z89371 works at 16 MHz, therefore each sample requires 28.75 µsec. There is a requirement of 31 kHz sampling rate; therefore, the period between interrupts is 32.26 µsec. This leaves only 10.875 percent of the time for the background process, which is not acceptable.

In the customer solution using Zilog's competition, the process of reading from the SRAM and writing to the SRAM takes only 12 instructions. Each A-D sample requires 20 passes through this code. The low-cost version of the chip works at 12 MHz; therefore, each sample requires 20 µsec. The 31 kHz sampling rate requirement leaves 38 percent of the time for the background process, which is better than the straightforward customer solution with the Z89371

Z893X1 New Solution

In the new solution suggested in this app. note, one shift-left instruction replaces the need for 8 shift-left instructions, and the process of reading from the SRAM (or FLASH) and writing to the SRAM (or FLASH) takes only 14 instructions (cycles), instead of 23 instructions without improvements. Each A-D sample requires 20 passes through this code. The Z89371 works at 16 Mhz; therefore, each sample requires 17.5 µsec. This leaves 45.75 percent of the time for the background process—compared to the measly 10.875 percent achieved with straightforward Z89321 and the 38 percent posted by the competing chip. (Refer to "Code_2: The New Z893X1 Assembler Code" in the Source Code section at the conclusion of this app. note.)

Z893X1 Architecture Trade-offs

Many architecture trade-offs between higher performance and lower chip price were considered during the design of the Z893X1 core (Z89C00 or Z89S00). The Z893X1 DSP family targets the low-cost, high-volume market, so it is very important to keep the price low. This was accomplished, largely, by reducing the die size.

All the blocks of the Z893X1 went through a rigorous process of determining which specific features for Z893X1 applications were really required and which could be eliminated. The "hard look" evaluation included:

- 32-Bit-Wide Multiplier-Accumulator Output Result
- 32-Bit-Wide Accumulator
- Context Switching
- Additional ALU
- Full Hardware Looping Support
- Full-Range Barrel Shifter

A full-range Barrel Shifter—from 16-bit left to 16-bit right—is very expensive and consumes a great deal of silicon. Since the Z893X1 applications are very cost sensitive, the full-range Barrel Shifter was not supported. Instead, only single-bit left shift, single-bit right shift and 3-bits right shift of the multiplier output are supported.

Z893X1-Supported Shift Instructions

The Z893X1 supports the following shift instructions:

SLL– Shift Left Logical

Instruction Description: All 24 bits of the Accumulator are shifted left through the carry bit. The Most Significant Bit (MSB), bit 23, passes through the carry bit before being discarded.

The Least Significant Bit (LSB), bit 0, is filled with a zero; subsequent shifts will result in additional zeros shifted in.

SRA–Shift Right Arithmetic

Instruction Description: All 24 bits of the Accumulator are shifted right, with sign extension, through the carry bit. The MSB, bit 23, is replicated into vacated bits. Bit 0 is passed through the carry before being discarded.

Sign Extension

It is also possible to shift right 3 bits with Sign Extension in the Multiplier output (P register) by writing "1" to bit 9 of the Status Register. This is often used to prevent overflow conditions with the Accumulator.

Variable Length Shift

Variable Length Shift Right into P register using the Multiplier.

Straightforward Z89371 System Architecture

As previously shown, when the straightforward (eight 1-bit left shifts) system implementation of Z89371 is used, then the Zilog competitor has the advantage. But when the new system implementation of Z89371 is used (detailed in this app. note), the Z89371 achieves better performance than the Zilog competitor.

In the straightforward system architecture, the 8-bit data bus of the SRAM can be connected to the low byte of EXT bus (EXT[7:0]), or (as originally suggested by the customer) the 8-bit data bus of the SRAM is connected to the high byte of EXT bus shifted right by 1 (EXT[14:7]) as shown in Figure 1, which follows. Shift right by 1 is necessary because Multiply and Multiply-Add operations are used in the bytes-to-word conversion program as part of the critical word-to-bytes and bytes-to-word conversions program that influences the application baud rate (performance) to combine the two bytes. Bit EXT15 is mapped into the sign bit of the Accumulator. Therefore, inserting data bit into this bit will corrupt the combined result. Bits EXT15 and EXT[6:0] are connected through a pull down to logic 0 and will be read as 0. Data written to bits EXT15 and EXT[6:0] will be lost.

EXT1 register is used as address register for the SRAM address. The control decoder block in Figure 1 includes EXT1 register write decoder and 14-bit latch to latch to the SRAM address. EXT2 is a virtual register for the low byte of EXT data bus, and EXT3 is a virtual register for the high byte of EXT data bus. Any write operation to EXT2 register or EXT3 register generates write pulse on the SRAM /WR input. Any read operation to EXT2 register or EXT3 register generates read pulse on the SRAM /RD input. Address bit 0 of the SRAM is connected to External Data bit 0 of the DSP (EA[0]). For each address that was written to EXT1 register, it is possible to write or to read two bytes. Writing or reading with EXT2 register always use the even addresses of the SRAM, and writing or reading with EXT3 register always use the odd addresses of the SRAM.

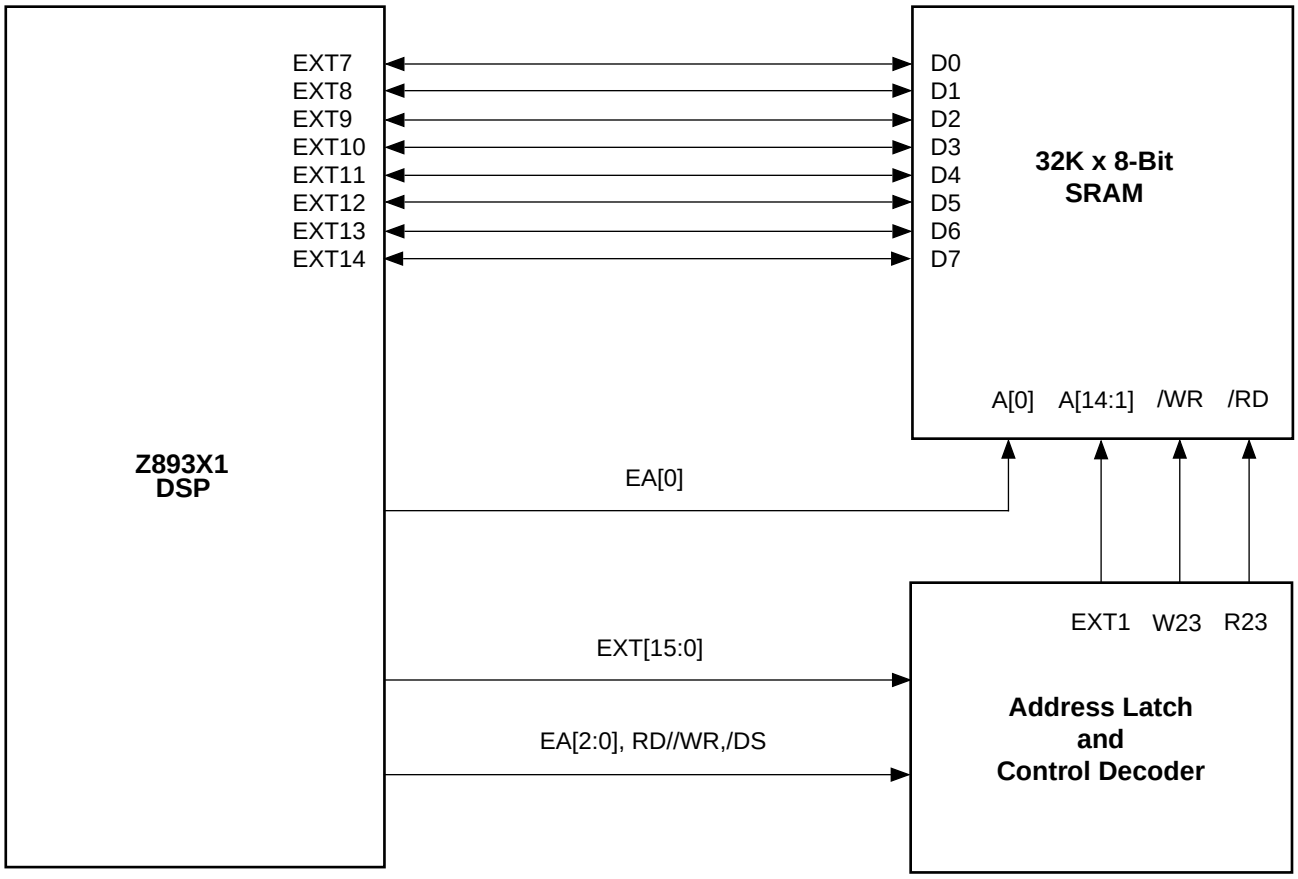

Figure 1. System Architecture for the Straightforward Z893X1 Solution

Straightforward Code Explanation

The first part of the code is initialization code.

Byte_Word

The next part of the code is the bytes-to-word conversion code. First, the current SRAM address is latched into EXT1. Later, the LSB is being read from the SRAM, and using the multiplier, the data is shifted 8 bits right into the P register. The MSB is being read from the SRAM and loaded into the A register, then the MSB and the LSB are added (combined) together and the result is shifted left by one. (The data read from the SRAM is shifted right by one bit because the SRAM is connected to EXT[14:7] instead of EXT[15:8].) The final word result is stored in the memory.

Address_Update

The SRAM address is incremented by one for the next cycle of bytes-to-word and word-to-bytes conversion.

Word_Bytes

The next part of the code is the word-to-bytes conversion code. First, the word is fetched from memory, then it is shifted right by one. Next, the MSB is written into the SRAM. Later, the Accumulator is shifted left 8 bits, and the LSB is written into the SRAM. The word-to-bytes conversion is now completed.

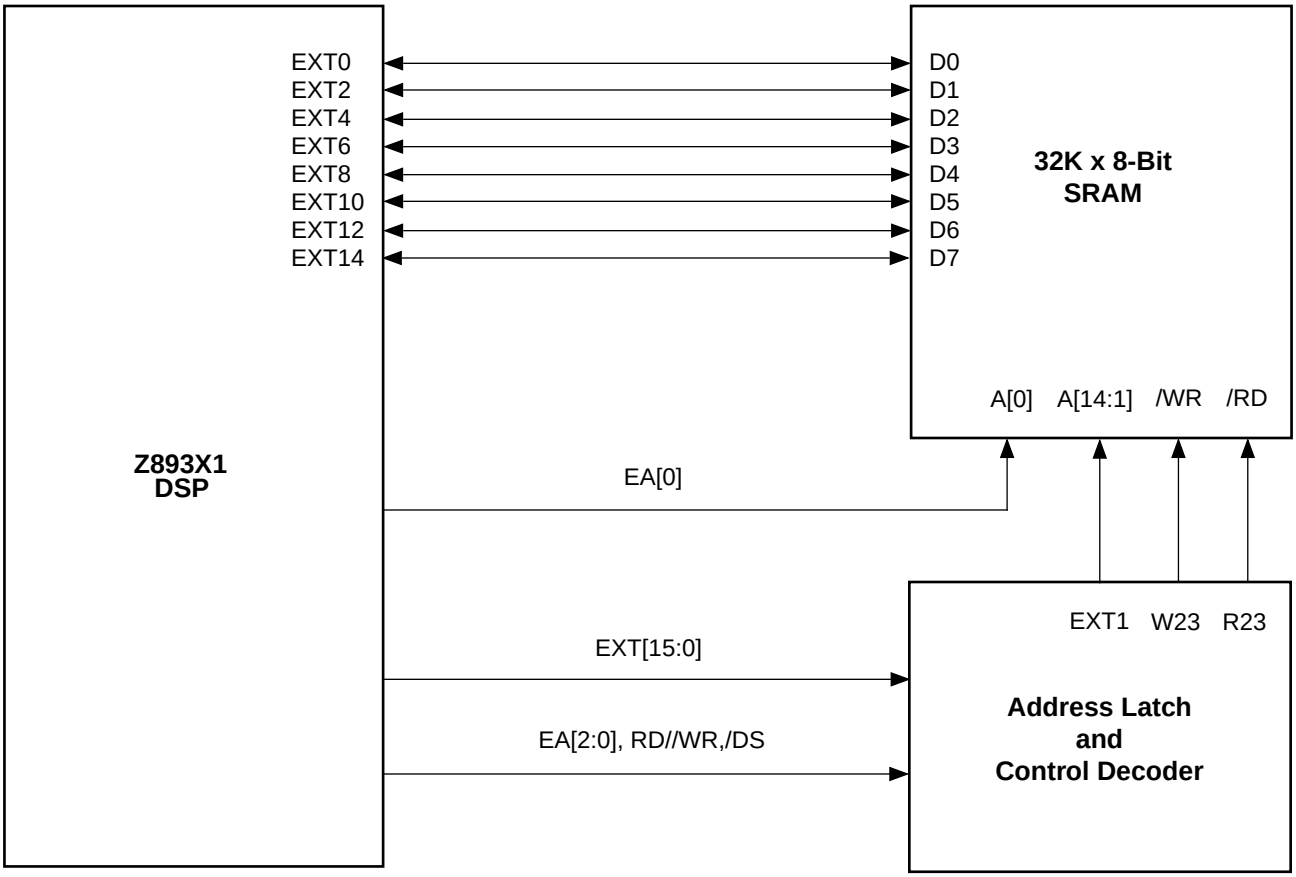

Figure 2. System Architecture for the New Z893X1 Solution

The main problem with the straightforward solution is the length of the word-to-bytes conversion code. The Z893X1 family does not have a flexible barrel shifter and therefore 8 single-bit, shift-left instructions are needed in the word-to-bytes conversion code instead of one 8-bit, shift-left instruction.

The Z893X1 Multiplier-Accumulator uses fractional arithmetic representation of numbers; therefore, it is easy to do any variable shift right in one instruction, but it is impossible to do any shift left—except the one shift left—in one instruction.

The problem is overcome by changing the way that the 8-bit-wide memory is connected to the 16-bit-wide DSP.

Instead of connecting the 8 memory data bus bits to EXT[14:7], the even bits of the EXT bus bits EXT[14,12,10,8,6,4,2,0] are connected. This special way of connection enables switching between the two bytes—with 1-bit shift right or with 1-bit shift left. All the odd bits EXT[15,13,11,9,7,5,3,1] are connected through a pull down to logic 0 and will be read as 0. Data written to bits EXT[15,13,11,9,7,5,3,1] will be lost.

The other logic is the same as in the original design: EXT1 register is used as address register for the SRAM address. The control decoder block in Figure 2 includes EXT1 register write decoder and 14-bit latch to latch to the SRAM address. EXT2 is a virtual register for the low byte of EXT data bus, and EXT3 is a virtual register for the high byte of EXT data bus. Any write operation to EXT2 register or EXT3 register generates write pulse on the SRAM /WR input. Any read operation to EXT2 register or EXT3 register generates read pulse on the SRAM /RD input. Address bit 0 of the SRAM is connected to External Address bit 0 of the DSP (EA[0]); therefore, for each address that was written to EXT1 register, it possible to write or to read two bytes. Writing or reading with EXT2 register always use the even addresses of the SRAM, and writing or reading with EXT3 register always use the odd addresses of the SRAM.

New System Architecture Code Explanation

The first part of the code is initialization code.

Byte_Word

The next part of the code is the bytes-to-word conversion code. First, the current SRAM address is latched into EXT1. Later, the odd bits are read from the SRAM. Next, they are shifted left to their position, then the even bits are read from the SRAM and combined (OR operation) with the odd bits in the Accumulator. The word result is then stored into the table in the memory.

Address_Update

The SRAM address is incremented by one for next cycle of bytes-to-word and word-to-bytes conversion.

Word_Bytes

The next part of the code is the word-to-bytes conversion code. First, the word is fetched from memory, then the even bits are written into the SRAM. Later, the odd bits are shifted one bit right, and the odd bits are written into the SRAM. The word-to-bytes conversion is now completed.

Conclusion

Lacking a full-range barrel shifter is not a fatal blow to Z893X1 applications. Not having a very expensive and high-consumption barrel shifter can be overcome by changing the system architecture connection of the 8-bit-wide memory to the 16-bit data bus of the Z893X1 DSP. In the actual case described in this app. note, a simple modification of the bus connection improved the performance of the Z893X1 DSP by a whopping 64 percent. The simple change in bus connection also enabled the Z89321/Z89371 to win the competition.

SOURCE CODE

Code_1: The Customer Original Z893X1 Assembler Code

```
p0:0 = Delay Length Table
p1:0 = I/O Table
d0:0 = External Buffer Pointer
Segment  b
NumberDelays       EQU12
DelayTable:   ds   NumberDelays+1
DelayIO:      ds   2*NumberDelays
DelayPointer: ds   1
One:          ds   1
Shiftright8:  ds   1
     segment code
     org     %0
     ld      p0:0,#DelayTable
     ld      p1:0,#DelayIO
     ld      p2:0,#Shiftright8
     ld      @p2:0,#%80      ;shift right 8 places
     ld      d0:0,#%8000     ;Delay Starting Address
     ld     a,#1
     ld     One,a

Bytes_Word:
     ld     EXT1,d0:0        ;output delay end address
     mld    EXT2,@p2:0,on    ;fetch LSB and shift right 8 places
     ld     a,EXT3           ;fetch MSB
     mpya   @p0:1,@p0:0      ;combine
     sll    a                ;one final shift
     ld     @p1:0+,a         ;store result into table
```

```
Address_update:
     ld       a,d0:0          ;fetch buffer address
     add      a,@p0:0+        ;add in delay length to get to start of buffer
     ld       EXT1,a          ;output start address
     add      a,one           ;compute end address for next buffer
     ld       d0:0,a          ;save updated address

Word_Bytes:
     ld       a,@p0:0+        ;fetch delay in value
     sra      a
     ld       EXT3,a          ;send out MSB
     sll      a
     sll      a
     sll      a
     sll      a
     sll      a
     sll      a
     sll      a
     sll      a
     ld       EXT2,a
     end
```

Code_2: The New Z893X1 Assembler Code

```
p0:0 = Delay Length Table
p1:0 = I/O Table
d0:0 = External Buffer Pointer
Segment       b_new
NumberDelays  EQU      12
DelayTable:   ds       NumberDelays+1
DelayIO:      ds       2*NumberDelays
DelayPointer: ds       1
One:          ds       1
Shiftright8:  ds       1

    segment code
    org       %0
    ld        p0:0,#DelayTable
    ld        p1:0,#DelayIO
    ld        p2:0,#Shiftright8
    ld        @p2:0,#%80      ;shift right 8 places
    ld        d0:0,#%8000     ;Delay Starting Address
    ld        a,#1
    ld        One,a

Bytes_Word:
   ld         EXT1,d0:0       ;output delay end address
   ld         a,EXT3          ;reading the odd bits
   sll        a               ;shifting the odd bits to their position
   or         a,EXT2          ;reading the even bits and combining the two bytes
   ld         @p1:0+,a        ;store result into table
Address_update:
  ld          a,d0:0          ;fetch buffer address
  add         a,@p0:0+        ;add in delay length to get to start of buffer
  ld          EXT1,a          ;output start address
  add         a,one           ;compute end address for next buffer
  ld          d0:0,a          ;save updated address

Word_Bytes:
  ld          a,@p0:0+        ;fetch delay in value
  ld          EXT2,a          ;writing the even bits
  sra         a               ;shifting the odd bits to the even bits
  ld          EXT3,a          ;writing the odd bits
  end
```