OMAP5910 DSP External Memory Performance

Jim Patterson *North America DSP Applications*

ABSTRACT

The DSP subsystem of the OMAP5910 can access the 192K bytes of OMAP5910 internal RAM and memory external to the OMAP5910 device, in addition to the 80K x 16 bits of DSP RAM and 16K x 16 bits of DSP ROM. The DSP will achieve the highest performance when the internal DSP RAM is used, but some applications will require the use of internal system RAM or external memory. This report analyzes the DSP performance when external memory is used for program and data storage, and suggests strategies to help achieve high performance when using external memory.

Contents

List of Figures

List of Tables

1 Introduction

In order to achieve optimal performance with the OMAP5910 DSP subsystem, it is necessary to understand the types of memory available to the DSP and to recognize the performance that can be achieved when different memory types are used for program and data storage.

This report describes the procedures required to allow the DSP to access memory external to the DSP subsystem and documents the performance of the DSP subsystem when using external memory.

2 Overview of DSP Subsystem Memory

The TMS320C55x™ DSP subsystem uses a single 16M byte linear address space for both program and data. The internal memory of the DSP subsystem comprises 160K bytes of RAM and 32K bytes of masked ROM. The remainder of the memory space can be mapped within the 4G bytes OMAP5910 system address space by way of the DSP memory management unit (MMU). The system memory that the DSP can access includes 192K bytes of internal SRAM and external memory connected to either of the OMAP device's two external memory interfaces (EMIF).

There is also a 24K byte instruction cache that can be used in conjunction with external memory accesses.

The DSP memory map is shown in Appendix A.

2.1 Internal DSP Memory

The internal memory of the DSP subsystem comprises three memory types: single access RAM (SARAM), dual access RAM (DARAM), and program/data ROM (PDROM). This memory is connected directly to the DSP core, and provides the fastest access for the DSP. Table 1 summarizes the organization and bandwidth for each of these memory types.

Table 1. DSP Internal Memories

Memory Type	Amount	Organization	Bandwidth
SARAM	96K bytes	12 blocks of 8K bytes	1 access/block/cycle
DARAM	64K bytes	8 blocks of 8K bytes	2 accesses/block/cycle
PDROM	32K bytes	1 block of 32K bytes	1 access/cycle

2.2 DSP Instruction Cache

The DSP instruction cache (I-cache) comprises three 8-kilobyte banks. Two of the banks consist of 512 lines of 16 bytes, and can be configured as an 8-kilobyte direct-mapped cache, or as a 16-kilobyte, two-way set associative cache. The third bank is divided into two 4-kilobyte blocks, each of which can be configured as a RAM set cache, in which there is only one tag address associated with the entire block.

The I-cache is only used in conjunction with external memory accesses. Accesses to the internal DSP memory are never cached because the cache would provide no performance advantage.

2.3 External DSP Memory

External DSP memory includes any memory connected to one of the OMAP EMIFs, and the 192K bytes of OMAP internal SRAM. While the SRAM is internal to the OMAP device, it is considered as external memory for the DSP core.

2.3.1 DSP External Memory Data Path

All DSP access to memory that is external to the DSP subsystem is through the system traffic controller (TC). The traffic controller manages all accesses by the ARM9, DSP, system DMA controller, and the peripheral local bus to the OMAP5910 system memory resources, which consist of internal SRAM and external memory devices.

2.3.1.1 DSP Memory Management Unit

The DSP memory map size is 16M bytes, while the OMAP5910 memory map is 4G bytes. A memory management unit (MMU) translates the 24-bit DSP addresses to 32-bit system addresses, allowing the DSP memory to be mapped within the system memory map.

The DSP MMU is controlled by the ARM9. At system reset, the DSP MMU is disabled, and the DSP external memory space is mapped to the first 16M bytes of the system address space.

A detailed description of the DSP MMU is beyond the scope of this report, but a high-level comprehension of the MMU is needed to understand how the DSP accesses external memory.

The operation of the MMU is somewhat like the operation of a cache system. The heart of the MMU is a 32-entry content addressable memory (CAM) and a 32-entry RAM. The contents of the CAM and RAM are collectively referred to as a translation look-aside buffer (TLB). Each TLB CAM entry comprises a tag address, preserved and valid bits, and a page-size specification. The index or address points to an entry in the MMU RAM that contains the most significant bits of the physical address and access permission bits that mark the page for no access, read-only access, or read-write access.

To configure the DSP MMU, the ARM9 must write the most significant bits of the virtual address and page size information into the CAM, and write the corresponding most significant bits of the physical address and the access permissions into the RAM. After the TLB entries have been initialized, the ARM9 can enable the DSP MMU.

When the DSP MMU is enabled, the virtual address from the DSP is presented to the MMU. If the upper bits of the MMU match a valid entry in the CAM, the index of that entry is presented to the MMU RAM, which provides the corresponding most significant bits of the physical address along with the access permission. If the requested access is allowed by the access permission, the physical address is presented to the traffic controller, which performs the access to the physical memory.

If the virtual address is not present in the TLB, or if the access permission does not allow the requested access, the MMU generates a page fault or permission fault and asserts an interrupt to the ARM9 interrupt controller. The ARM9 can determine the cause of the fault by interrogating the MMU's registers.

Detailed information on the OMAP5910 MMU can be found in *OMAP5910 Dual-Core Processor Technical Reference Manual* (SPRU602).

2.3.1.2 OMAP5910 Traffic Controller

The traffic controller (TC) manages accesses to OMAP5910 memory other than the DSP module's internal memory. The TC receives access requests from the DSP, the ARM9, the system DMA, and the internal local bus, which supports accesses from internal peripherals with bus master capability. On the OMAP5910, the USB host controller is the only peripheral with bus master capability.

The TC connects to one internal and two external memory interfaces:

- The internal memory interface (IMIF)
- The external memory interface fast (EMIFF)
- The external memory interface slow (EMIFS)

2.3.1.3 OMAP5910 Memory Interfaces

The IMIF is a 32-bit wide interface that connects the TC to the 192K byte internal SRAM.

The EMIFS is a 16-bit wide interface that supports asynchronous memory types such as flash EPROM, SRAM, or external peripheral devices. The timing for EMIFS accesses is programmable, and is derived from the clock provided to the traffic controller.

The EMIFF is a 16-bit wide interface that supports only synchronous DRAM (SDRAM) memory. The EMIFF timing is programmable and is also derived from the TC clock.

2.4 Configuring the DSP MMU for External Memory Access

At device reset, the DSP MMU is disabled, and any DSP memory accesses that are outside of the DSP's internal memory are mapped directly to the first 16M bytes of the OMAP5910 system memory. This is typically not a useful configuration, so before the DSP is allowed to access external memory, the ARM9 should first configure and then enable the DSP MMU. The configuration is usually done while the DSP is held in reset, although this is not a restriction.

Configuring the DSP MMU consists of loading a TLB entry for each page of DSP virtual memory to be used. The page size can be selected as 1M byte, 64K byte, 4K byte, or 1K byte. The MMU_Write_TLB_Entry function shown in Figure 1 illustrates the configuration of a TLB entry. This is a general function that can be called to initialize any of the OMAP5910 MMUs. The segment that is applicable to the DSP MMU is in the DSP_MMU case.

```
//-------------------------------------------------------------------------
// NAME        : MMU_Write_TLB_Entry
// DESCRIPTION : To load one item in the TLB, 5 consecutive register
//               accesses are required.
//               This entry is loaded at the address pointed by the lock
//              counter register.
//               1. Write CAM msb in CAM_REG_H register
//               2. Write CAM lsb in CAM_REG_L register
//               3. Write RAM msb in RAM_REG_H register
//               4. Write RAM lsb in RAM_REG_L register
//               5. Update Lock Counter register
//               6. Write 1 in LD_TLB_REG register
// PARAMETERS  : mmu_name could be DSP_MMU
//
//               physical_address
//               virtual_address
//               slst could be SECTION
//              LARGE_PAGE
//              SMALL_PAGE
//              TINY_PAGE
//     AP_bits could be NOT_ACCESSIBLE
//            READ_ONLY
//            FULL_ACCESS
//     locked_base_value
//     current_entry
//     p_bit could be ENTRY_NOT_PRESERVED
```

Figure 1. OMAP5910 MMU Initialization Function

```
//            ENTRY_PRESERVED
// RETURN VALUE: None.
// LIMITATIONS : It is possible to load an entry in the TLB only if
//               the WTL is DISABLED.
//-----------------------------------------------------------------------------
void MMU_Write_TLB_Entry(MMU_NAME_t  mmu_name,
           UWORD32     physical_address,
           UWORD32     virtual_address,
           SLST_t      slst_bit,
           AP_t        ap_bits,
          UWORD8      locked_base_value, // between 0 and 31
          UWORD8      current_entry,     // between base_value and 31
          PRESERVED_t p_bit)
{
UWORD16 VA_tag_I2, VA_tag_I1;
  VA_tag_I1 = ((virtual_address & 0xFFF00000) >> 20);
  VA_tag_I2 = ((virtual_address & 0x000FFC00) >> 10);
    switch(mmu_name)
    {


    case DSP_MMU :
      {
       // Write CAM msb and lsb into CAM_REG_H and CAM_REG_L registers
       DSPMMU_CAM_H_REG = (VA_tag_I1 >> 2);
       DSPMMU_CAM_L_REG = (slst_bit | (p_bit << 3) | (VA_tag_I2 << 4) | \
                          ((VA_tag_I1 & 0x0003) << 14) );

       // Write RAM msb and lsb into RAM_REG_H and RAM_REG_L registers
       DSPMMU_RAM_H_REG = (physical_address >> 16);
       DSPMMU_RAM_L_REG = ((ap_bits << 8) | (physical_address & 0x0000FC00));
       DSPMMU_LOCK_REG = ((current_entry << 4) | (locked_base_value << 10));
       // write 1 into LD_TLB_REG register
       DSPMMU_LD_TLB_REG = 1;
       break;
      }
    }
```

Figure 1. OMAP5910 MMU Initialization Function (Continued)

When the DSP MMU configuration is complete, the ARM9 can enable the MMU, which will then map the 24-bit addresses from the DSP core to the 32-bit OMAP5910 system address space.

CAUTION:

The DSP MMU provides no inherent protection against mapping the same physical memory to both the DSP and the ARM9. Sharing memory between the processor cores may be advantageous in many applications. It is the responsibility of the programmer to ensure that ARM accesses do not corrupt memory mapped to the DSP, and vice versa.

2.5 Configuring the EMIFs

The ARM9 should also configure the EMIFF and EMIFS for best performance based on the TC clock speed and the timing characteristics of the external memory. At device reset, the EMIFS is set to a default configuration that provides the maximum access time possible. This provides compatibility with most asynchronous devices, but will not give the best performance. The EMIFF must be configured and initialized for the particular size and configuration of the attached SDRAM.

3 Memory Performance Testing

Tests were performed to measure the performance that the DSP could achieve when using external memory for both program and data accesses.

3.1 Test Setup

The equipment used for the memory performance testing included the following items:

- A Texas Instruments Incorporated test board with a POMAP5910CGZG device and external SDRAM, flash, and SRAM. The specific RAM devices used on the test board are:
  - Samsung K6T4015U3C-TB70 70ns 512Kb SRAM, organized as 32K x 16 bits
  - Samsung K4S561632B-TL75 256 Mbit 133 MHz SDRAM organized as 4M x 16-bit x 4 banks

  Data sheets for these devices are available from Samsung Electronics at www.samsungelectronics.com
- Windows™ 98 PC with Code Composer Studio version 2.10 for OMAP
- Texas Instruments XDS510 JTAG Emulator
- Tektronix 640A Digitizing Oscilloscope

3.2 Test Procedure

3.2.1 ARM9 Initialization

In the OMAP5910 architecture, the DSP operates as a slave to the ARM9, meaning that the ARM9 not only controls resetting the DSP, but it must also configure the programmable clock domains, the memory interfaces, and the DSP MMU. The Code Composer Studio debugger was used to load the setup code into RAM and run through the initialization code. The ARM operation was stopped by a software breakpoint following the DSP and system initialization, ensuring no conflicts between ARM and DSP accesses to system resources.

The ARM initialization code performed the following steps:

- Set up the ARM and DSP clock domains as shown in Table 2.

Table 2. Clock Configuration for External Memory Performance Tests

Clock Domain	Frequency
ARM9	150 MHz
DSP	150 MHz
Traffic Controller	75 MHz
EMIFF Clock	75 MHz
EMIFS Clock	75 MHz

- Initialize the DSP MMU as shown in Table 3.

Table 3. DSP MMU Configuration

DSP Base Address		OMAP System Base Address	Memory Type
Byte	Word		
0x80:0000	0x40:0000	0x1000:0000	SDRAM
0x90:0000	0x48:0000	0x0C40:0000	SRAM
0xA0:0000	0x50:0000	0x2000:0000	Internal SRAM
0xB0:0000	0x58:0000	0x1030:0000	SDRAM

- Configure ARM, DSP and TC clocking as follows:
  - PLL Clock Generator configured for 150 MHz operation
  - Synchronous scalable operation – ARM, DSP, and TC operate at fixed divisors of the PLL clock
  - ARM9 and DSP clocks set to 150 MHz, TC clock set to 75 MHz
  - EMIFS configured to provide 6 TC clock read cycles and 7 TC clock write cycles. The write cycle consists of 1 TC clock address setup before $\overline{WE}$, a $\overline{WE}$ active duration of 5 TC clocks, and a 1 TC clock address and data hold after $\overline{WE}$. With the TC clock frequency set at 75 MHz, the TC clock period is 13.33ns. This EMIFS configuration meets the SRAM timing requirements, as shown in Table 4.

Table 4. EMIFS Configuration for Throughput Tests

Parameter	Symbol	Requirement	EMIFS Configuration
Read cycle time	t_{RC}	70 ns	80 ns
Chip select to output	t_{CO}	70 ns	80 ns
Address access time	t_{AA}	70 ns	80 ns
Write cycle time	t_{WC}	70 ns	93 ns
Address valid to end of write	t_{AW}	60 ns	66.7 ns
Write pulse width	t_{WP}	55 ns	66.7 ns

– Configure EMIFF to provide the SDRAM timing shown in Table 5. Note that while the EMIF configuration is much slower than what is required by the SDRAM, this is the only one of the four possible SDRAM configurations (EMIFF_SDRAM_CONFIG.SDRAM_FREQUENCY = SDF0) that will satisfy all of the SDRAM timing requirements.

Table 5. EMIFF Configuration for Throughput Tests

Parameter	Symbol	Requirement	EMIFF Configuration
Row cycle time	t_{RC}(minimum)	65 ns	120 ns
Row active time	t_{RAS}(minimum)	45 ns	67 ns
Row precharge time	t_{RP}(minimum)	20ns	40 ns
$\overline{\text{RAS}}$ to $\overline{\text{CAS}}$ delay	t_{RCD}(minimum)	20 ns	26 ns
Row active to row active delay	t_{RRD} (minimum)	15 ns	26 ns
$\overline{\text{CAS}}$ latency	CL	3 clocks	3 clocks

– Configure GPIO12 to be controlled by the DSP core. GPIO12 was used to measure the duration of the memory accesses on an oscilloscope.

3.2.2 DSP Test Procedure

The DSP test code performed a series of data moves between internal SARAM and DSP external memory to measure the average throughput available through each memory interface. Next, the test code performed a series of instruction fetches from each memory external to the DSP, first with instruction cache disabled, then with instruction cache enabled.

For each type of memory access, the duration of the access was measured in two ways.

1. The execution time was measured with the Code Composer Studio profiler clock, with compensation for function call and return overhead.
2. A GPIO signal (GPIO12) was used to frame the external memory access by setting the signal high at the beginning of the access and setting it low following the access. The active-high duration of the signal was measured using an oscilloscope, and the measurement was compensated for measurement overhead.

Because the profiler clock and GPIO measurement provided an aggregate time for a number of accesses, the oscilloscope was also used to measure the duration of the SDRAM and SRAM accesses, which are actually external to the OMAP5910 device. Measuring the duration of the individual accesses allows separating the EMIF access time from the aggregate transfer time, so that the TC latency can be inferred.

During all the DSP tests, the ARM9 was halted under control of the Code Composer Studio debugger so that there were no ARM accesses to generate any contention for the system memory. The DSP test code and data were also always in separate memory blocks so that there were no data/program access conflicts generated by the DSP itself.

The test code combined unoptimized C code for the test setup along with assembly coded functions to perform the memory accesses that were tested.

3.2.2.1 Data Accesses

To measure the data access performance, the test code allocated 16K word (16-bit) buffers in each of the DSP external memories: SDRAM, SRAM, and internal RAM. The test code then used single-cycle instructions to transfer from the external memory to internal SARAM, and from internal SARAM to the external memory. The assembly language test code used is shown in Figure 2 and Figure 3.

```
; C55xx memory to memory move function
;
; Description:  Moves a block of data memory
;
; prototype:   void        mem_move( int* source, int* destination, int* count );
;
;              Jim Patterson
;              Texas Instruments
;              Copyright 2002
;
               .mmregs
               .sect ”test_code”
               .global     _mem_move
_mem_move:     sub         #1, T0          ;adjust count for N+1 operation of RPTB
               mov         T0, CSR         ;put count into CSR
               amov        #0xF002, AR3    ;set GPIO 12
               mov         port(*AR3), T1
               or          #0x1000, T1
               mov         T1, port(*AR3)
               rpt         CSR
               mov         *AR0+, *AR1+    ;copy 1 word from source to destination
               and         #0xEFFF, T1
               mov         T1, port(*AR3);clear GPIO 12
               ret
```

Figure 2. 16-Bit Data Access Function

```
; C55xx 32–bit memory to memory move function
;
; Description:  Moves a block of data memory using 32–bit loads and stores
;
; prototype:   void     mem_move32( int* source, int* destination, int* count );
;
; T0 contains count
; AR0 contains the source address
; AR1 contains the destination address
;
;
;              Jim Patterson
;              Texas Instruments
;              Copyright 2002
;
               .mmregs
               .sect ”test_code”
               .global     _mem_move32
_mem_move32:
               sfts        T0, #–1        ;divide block size by 2
               sub         #1, T0         ;adjust count for N+1 operation of RPTB
               mov         T0, CSR        ;put count into CSR
               amov        #0xF002, AR3   ;set GPIO 12
               mov         port(*AR3), T1
               or          #0x1000, T1
               mov         T1, port(*AR3)
               rpt         CSR
               mov         dbl(*AR0+), dbl(*AR1+)  ;move double word
               and         #0xEFFF, T1
               mov         T1, port(*AR3)          ;clear GPIO 12
               ret
```

Figure 3. 32-Bit Data Access Function

The overhead for each data access was measured by timing the execution of a function identical to the test function but without the data moves. The function overhead was subtracted from the aggregate time for the transfer, and the average transfer time was calculated using both the profiler clock measurement and the GPIO duration measurement. The data access test results are shown in Table 6.

Table 6. Data Access Throughput Measurements

Operation	Memory	Words	DSP Clocks	Time	Net DSP Clocks	DSP Clocks per Word	Net Time	Net TC Clocks	TC Clocks per Word
16-bit read	SDRAM	16384	333546	2.22E–03 sec	333522	20.36	2.22E–03 sec	166813	10.18
16-bit write	SDRAM	16384	134702	8.98E–04 sec	134678	8.22	8.98E–04 sec	67348	4.11
32-bit read	SDRAM	16384	167131	1.11E–03 sec	167107	10.20	1.11E–03 sec	83548	5.10
32-bit write	SDRAM	16384	67573	4.50E–04 sec	67549	4.12	4.50E–04 sec	33748	2.06
16-bit read	IRAM	16384	131106	8.74E–04 sec	131082	8.00	8.74E–04 sec	65548	4.00
16-bit write	IRAM	16384	114721	7.65E–04 sec	114697	7.00	7.65E–04 sec	57373	3.50
32-bit read	IRAM	16384	65571	4.37E–04 sec	65547	4.00	4.37E–04 sec	32766	2.00
32-bit write	IRAM	16384	57378	3.83E–04 sec	57354	3.50	3.82E–04 sec	28686	1.75
16-bit read	SRAM	16384	360482	2.40E–03 sec	360458	22.00	2.40E–03 sec	180148	11.00
16-bit write	SRAM	16384	360467	2.40E–03 sec	360443	22.00	2.40E–03 sec	180163	11.00
32-bit read	SRAM	16384	278563	1.86E–03 sec	278539	17.00	1.86E–03 sec	139198	8.50
32-bit write	SRAM	16384	294917	1.97E–03 sec	294893	18.00	1.97E–03 sec	147448	9.00

3.2.2.2 Program Access Tests

To measure the performance for execution from each type of memory, the test code called a series of assembly language functions consisting of a sequence of 2048 4-byte instructions that made no memory reference and executed in a single cycle. The functions were called first with the instruction cache disabled, then with the instruction cache enabled and cleared, so that each instruction was initially fetched from external memory before execution. A fragment of the code for the instruction test function is shown in Figure 4. The particular code in Figure 4 is for the function that was linked to SDRAM. The other test functions were identical except for the name of the entry point and the section name, which was changed to allow linking each function to the appropriate type of memory.

```
;
; This function is designed to measure instruction fetch throughput
;
; It consists of 2048 in–line 32–bit instructions
;
; Upon entry, T0 will contains a 16–bit integer.
; T0 will be decremented 2048 times with the result left in T0
;
               .global     _SDRAM_fetch32
               .sect ”I_$_SDRAM”
_SDRAM_fetch32:
               amov        #0xF002, AR3            ;set GPIO 12
               mov         port(*AR3), T1
               or          #0x1000, T1
               mov         T1, port(*AR3)
               sub         #1, T0, T0              ;1st instruction
               .
               .           ;this instruction was fetched and executed 2048 times
               .
               sub #1, T0, T0                      ;2048th instruction
               and         #0xEFFF, T1
               mov         T1, port(*AR3)          ;clear GPIO 12
done2048:      ret
```

Figure 4. Program Access Test Code

The program access test results are shown in Table 7.

Table 7. Program Access Measurements

Operation	Source	Packets	DSP Cycles	DSP Cycles per Packet	Net Time	TC Cycles	TC Cycles per Packet
I-fetch	SDRAM	1024	20939	20.4	1.39E–04 sec	10435.1	10.2
I-fetch	SRAM	1024	34989	34.2	2.32E–04 sec	17393.6	17.0
I-fetch	IRAM	1024	8224	8.0	5.45E–05 sec	4091.2	4.0
Cache fill	SDRAM	1024	10984	10.7	7.28E–05 sec	5459.5	5.3
Cache fill	SRAM	1024	29923	29.2	1.98E–04 sec	14878.0	14.5
Cache fill	IRAM	1024	6189	6.0	4.10E–05 sec	3076.0	3.0

Individual accesses were also observed and recorded using the oscilloscope. By observing the duration of the individual memory accesses and subtracting the duration of the individual accesses from the average access duration for that memory type, the TC overhead can be inferred.

3.2.2.3 16-Bit Data Read from SDRAM

Figure 5 shows a 16-bit data read from SDRAM as recorded on the oscilloscope.

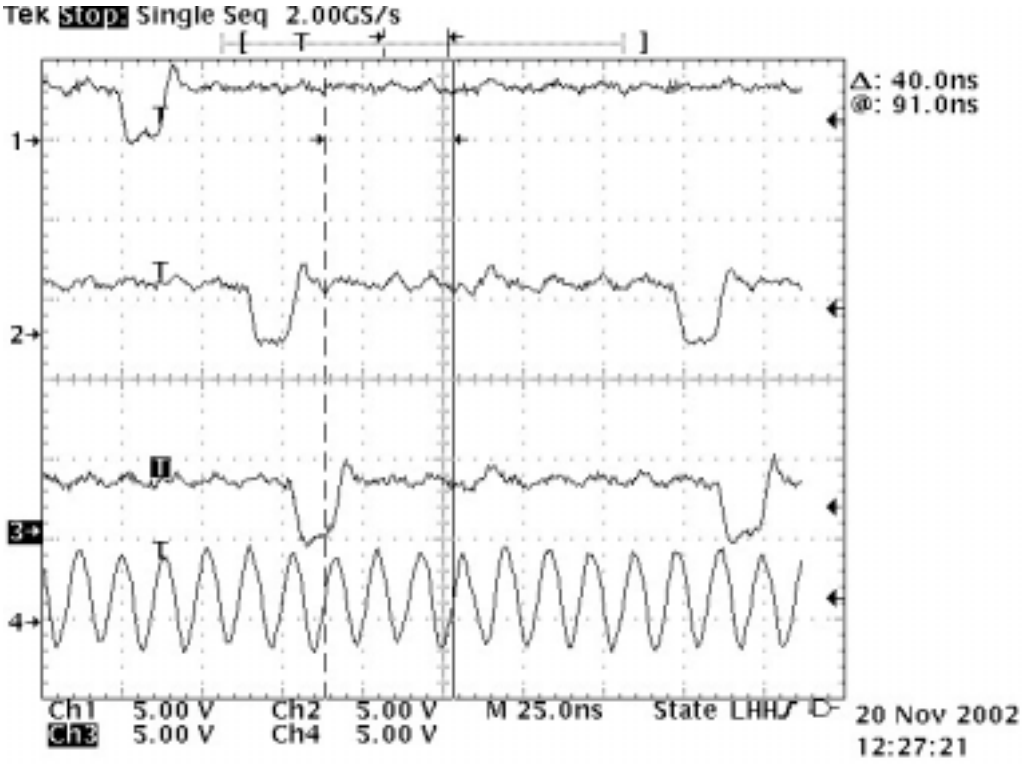

Figure 5. 16-bit Data Read From SDRAM

In this and subsequent oscilloscope pictures, trace 1 is the SDRAM $\overline{\text{RAS}}$ signal, trace 2 is $\overline{\text{CAS}}$, trace 3 is $\overline{\text{WE}}$, and trace 4 is the SDRAM clock, which is at the same frequency as the TC clock.

The read sequence starts with a bank activate command at the point marked by **T** on the screen. Three clocks later a burst read access is initiated when $\overline{\text{CAS}}$ goes active. On the next clock the burst is terminated when $\overline{\text{WE}}$ goes active. The data from the read is latched on the third clock after $\overline{\text{CAS}}$, or two clocks after the burst stop command. The duration of the individual read is four cycles, beginning with $\overline{\text{CAS}}$ and ending three cycles later when the data are latched. The next read command is issued six TC clocks after the data are latched. From the data shown in Table 6, a 16-bit read can be completed every 10 TC clock cycles. Since the actual SDRAM read only takes four cycles, the remaining six cycles must be incurred in the TC and EMIFF, indicating that these two blocks in combination incur a six TC clock cycle latency for reads.

In addition, the first read from SDRAM, or any read which does not access the same row as the previous read, incurs an additional three TC penalty to set the row address.

3.2.2.4 16-Bit Writes to SDRAM

Figure 6 shows 16-bit writes to SDRAM. As in the 16-bit read, the SDRAM operation starts when $\overline{RAS}$ goes active, followed three TC clock cycles later by $\overline{CAS}$ and $\overline{WE}$, indicating a write command. The data is written to the SDRAM during this cycle, because there is no write data latency for the SDRAM. The $\overline{WE}$ signal remains active for the next TC clock cycle to stop the burst operation. The 16-bit write operation takes only two TC clock cycles, but there are an additional two TC clock cycles before the next write command is issued. This corroborates the data in Table 6, which shows four TC clock cycles for a 16-bit write. The four cycles include two cycles for the SDRAM write plus two cycles latency due to the TC and EMIFF. Just like the 16-bit read operation, the first write on an SDRAM row incurs an additional three TC clock cycle delay.

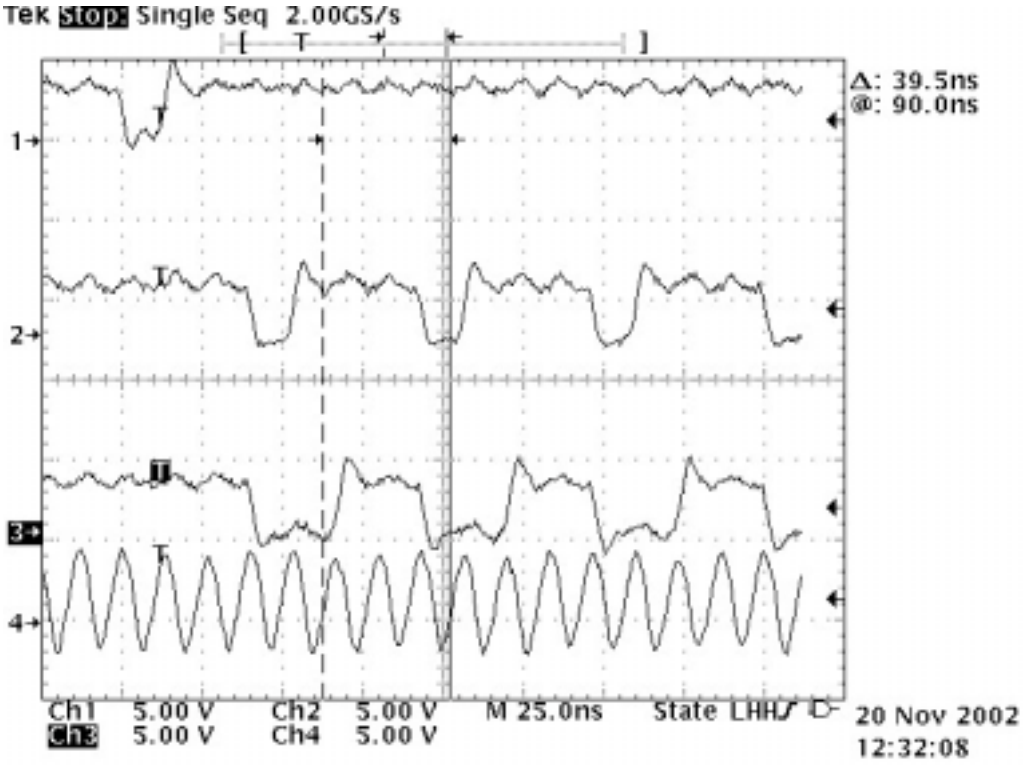

Figure 6. 16-Bit Writes to SDRAM

3.2.2.5 32-Bit Data Read from SDRAM

Figure 7 shows 32-bit data reads from SDRAM.

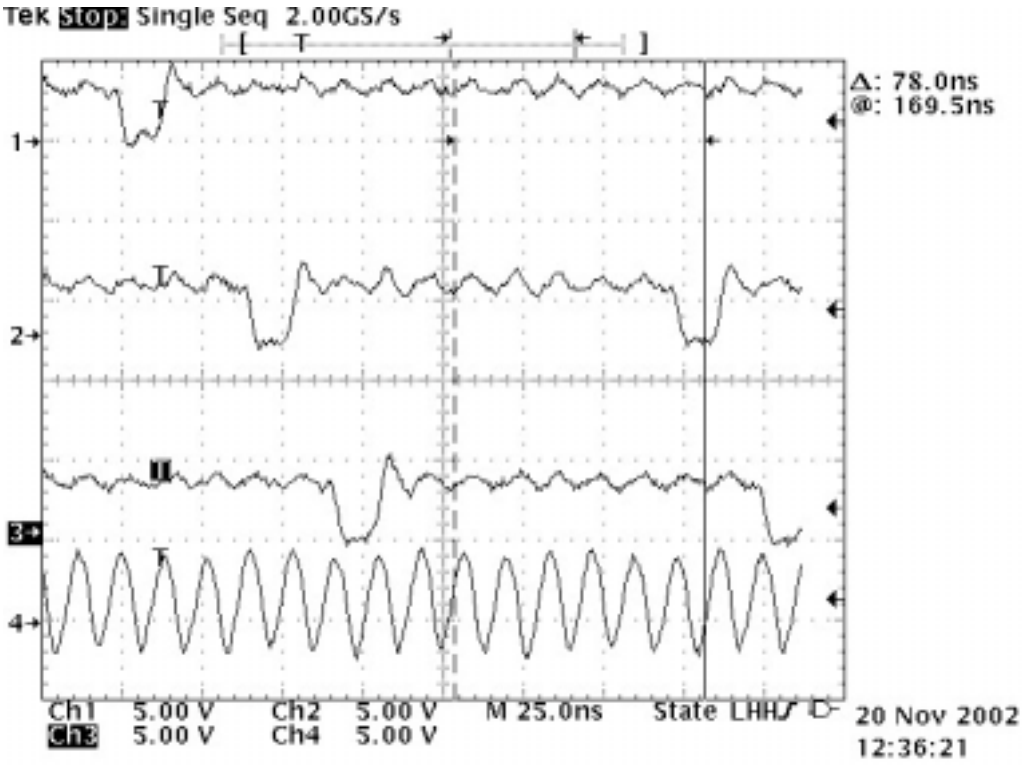

Figure 7. 32-Bit Data Read from SDRAM

The 32-bit read differs from the 16-bit read in that the burst stop command is issued on the second TC clock cycle following the read command, rather than on the first cycle. The actual end of the read is four TC clock cycles after the read command, when the second 16-bit word is latched. The next read command is issued six TC clock cycles after the end of the current read command. The DSP can complete a 32-bit data read every 10 TC clock cycles. The 10 cycles include four cycles for the read operation plus six cycles of TC and EMIFF latency for read operations. This agrees with the average throughput of one 16-bit word per five TC clock cycles measured using the Code Composer Studio profiler.

3.2.2.6 32-Bit Data Writes to SDRAM

A series of 32-bit writes to SDRAM are shown in Figure 8. For the 32-bit data write, the two 16-bit words are written to the SDRAM during the first two TC clock cycles of the access. The burst stop command is issued on the third TC clock cycle, and the next write command is issued two clocks after the burst stop command, giving a throughput of two 16-bit words per four TC clock cycles.

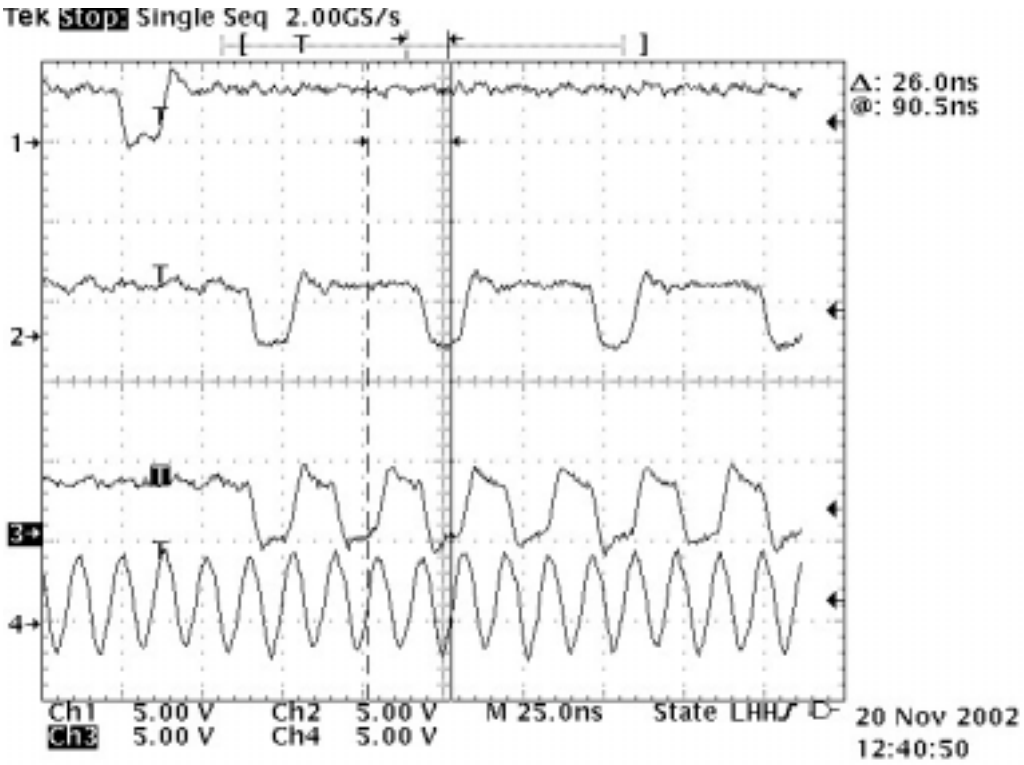

Figure 8. 32-Bit Data Write to SDRAM

3.2.2.7 Instruction Fetches from SDRAM

Figure 9 shows instruction fetches from SDRAM with the instruction cache disabled. In this example, the DSP is fetching and executing 16-bit, single cycle instructions. The access is identical to a 32-bit data read, as described in section 3.2.2.5. The operation begins with the read command initiated by $\overline{\text{CAS}}$, and completes 6 TC clock cycles later when the second word of the fetch packet is latched. There is a 4 TC clock cycle delay between the end of the current fetch and the issuance of the read command for the next fetch. Thus, the average throughput for instruction fetches from SDRAM is 1 fetch packet per 10 TC clock cycles.

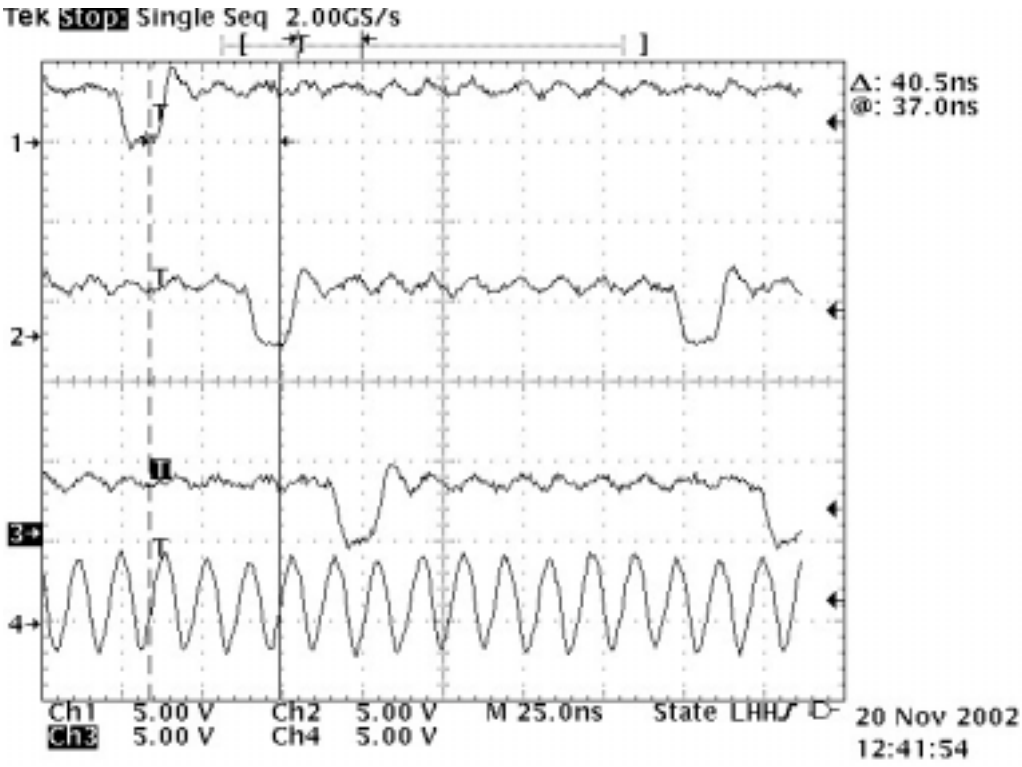

Figure 9. Instruction Fetches from SRAM with Instruction Cache Disabled

3.2.2.8 Instruction Cache Line Fill from SDRAM

Figure 10 shows a series of instruction fetches from SDRAM due to an instruction cache miss. Since a cache miss will always result in fetching 16 bytes to fill a cache line, the EMIFF performs an eight-word burst access. The eight 16-bit words are latched on the third through tenth clocks following the read command. The burst stop command is issued on the eighth clock following the read command. After the 8-word burst read completes there is a nine-clock delay before the next read command is issued. If the DSP is retiring instructions faster than they can be fetched via the EMIF, the EMIF issues a burst read command every 20 TC clock cycles. The average EMIFF throughput for instruction cache fills is eight 16-bit words, or four 32-bit fetch packets per 20 TC clock cycles. This can also be expressed as five TC clock cycles per fetch packet.

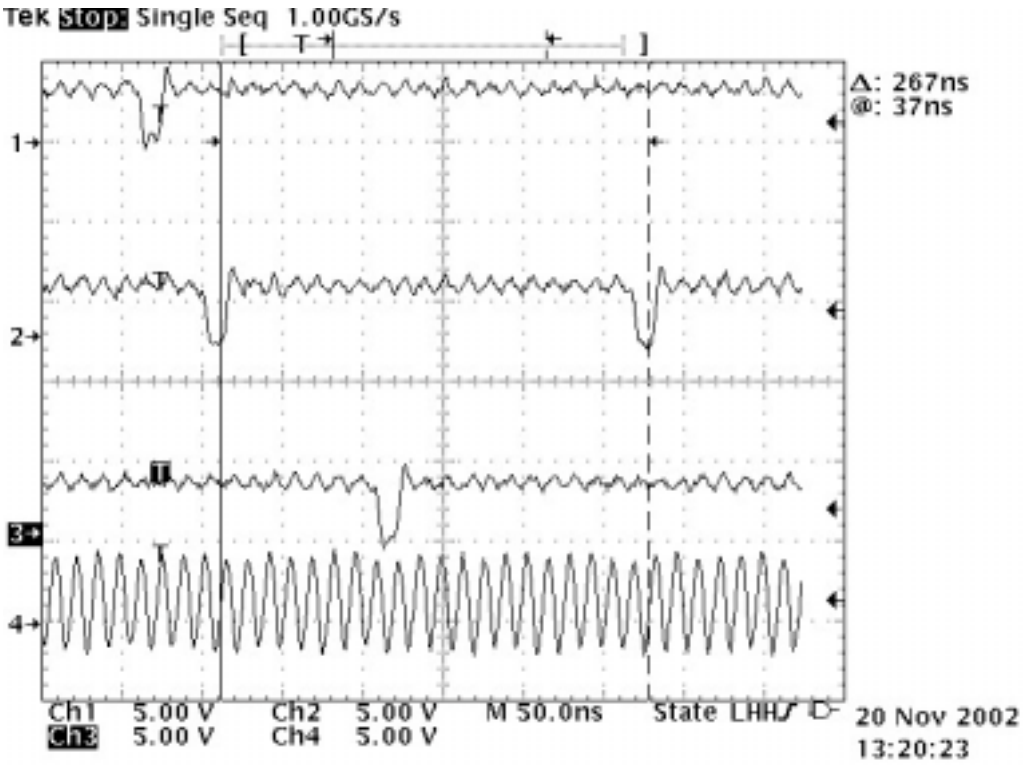

Figure 10. Instruction Cache Line Fill from SDRAM

3.2.2.9 16-Bit Data Reads from SRAM

In all of the oscilloscope traces of SRAM accesses, the upper trace is the SRAM $\overline{CS}$ signal, and the lower trace is the SRAM $\overline{WE}$ signal.

Figure 11 shows 16-bit data reads from SRAM. The EMIFS configuration set up by the ARM9 MCU code set the EMIFS clock to 75 MHz, yielding a 13.33ns EMIFS clock cycle time. The $\overline{CS3}$ active time was set to six EMIFS clock periods, for a $\overline{CS3}$ duration of 80ns. $\overline{CS}$ is seen to go active for 80 ns, or six EMIFS clocks. $\overline{CS}$ then goes inactive for 67 ns, or six EMIFS clocks before the start of the next read. This shows that for a series of 16-bit reads from SRAM, the EMIFS starts a read access every 11 EMIFS cycles, which is the same performance that was previously measured and is shown in Table 6.

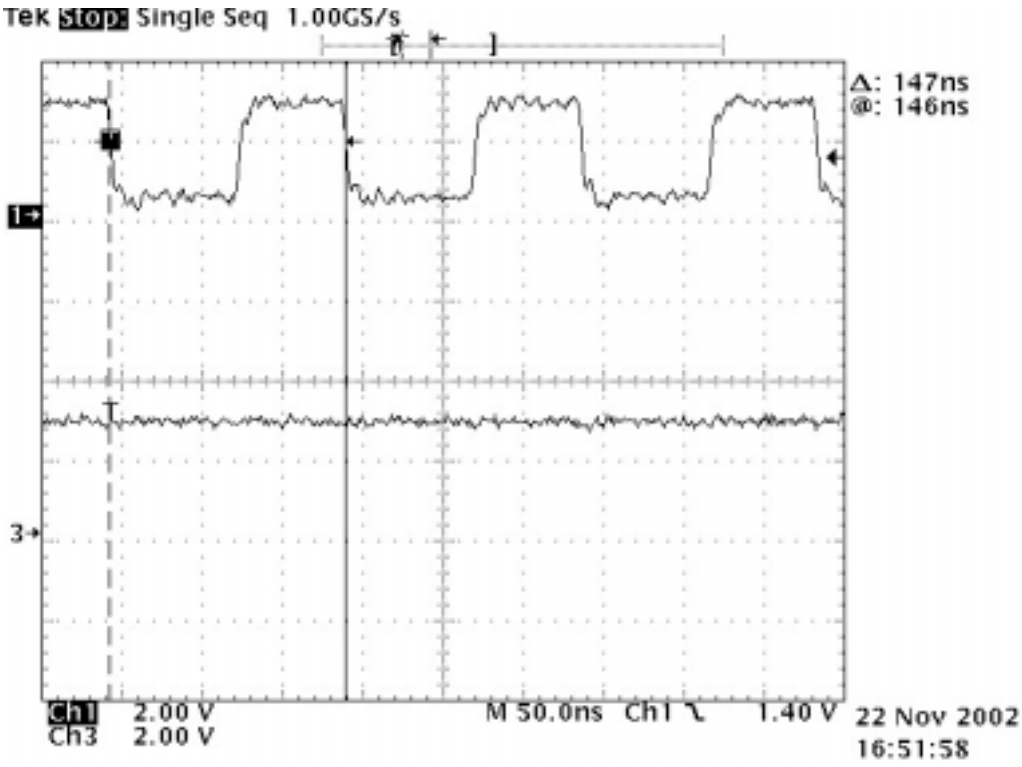

Figure 11. 16-Bit Data Reads from SRAM

3.2.2.10 16-Bit Data Writes to SRAM

Figure 12 shows a series of 16-bit writes to SRAM. The write cycle starts when $\overline{CS}$ goes active, followed one EMIFS clock later by $\overline{WE}$ going active. As programmed by the ARM9, $\overline{WE}$ stays active for five EMIFS clocks. $\overline{CS}$ returns to the inactive state one clock cycle later, for a total write cycle length of seven EMIFS clocks. $\overline{CS}$ remains inactive for four clock cycles before the next write starts. With the programmed configuration the EMIFS can complete one 16-bit read cycle every 11 EMIFS clock cycles, as shown in Table 6.

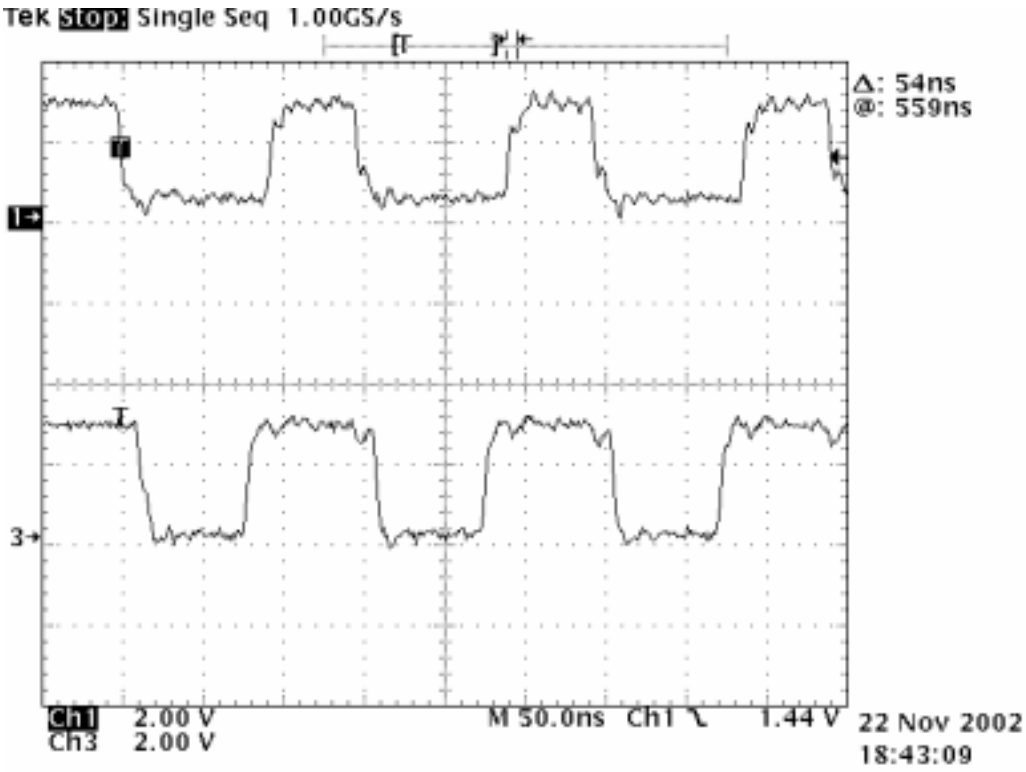

Figure 12. 16-Bit Data Writes to SRAM

3.2.2.11 32-Bit Data Reads from SRAM

When the EMIFS performs a 32-bit read, the $\overline{CE}$ stays active for the duration of both reads. Since the EMIFS is programmed for a read cycle of six clocks, the strobe stays active for 12 clocks for the double-word read, and only the address changes between read accesses. Following the read, $\overline{CE}$ returns inactive for five clock cycles. When performing 32-bit reads, the EMIFS as configured can return two 16-bit words every 17 EMIFS clocks. Figure 13 illustrates two 32-bit data reads from SRAM.

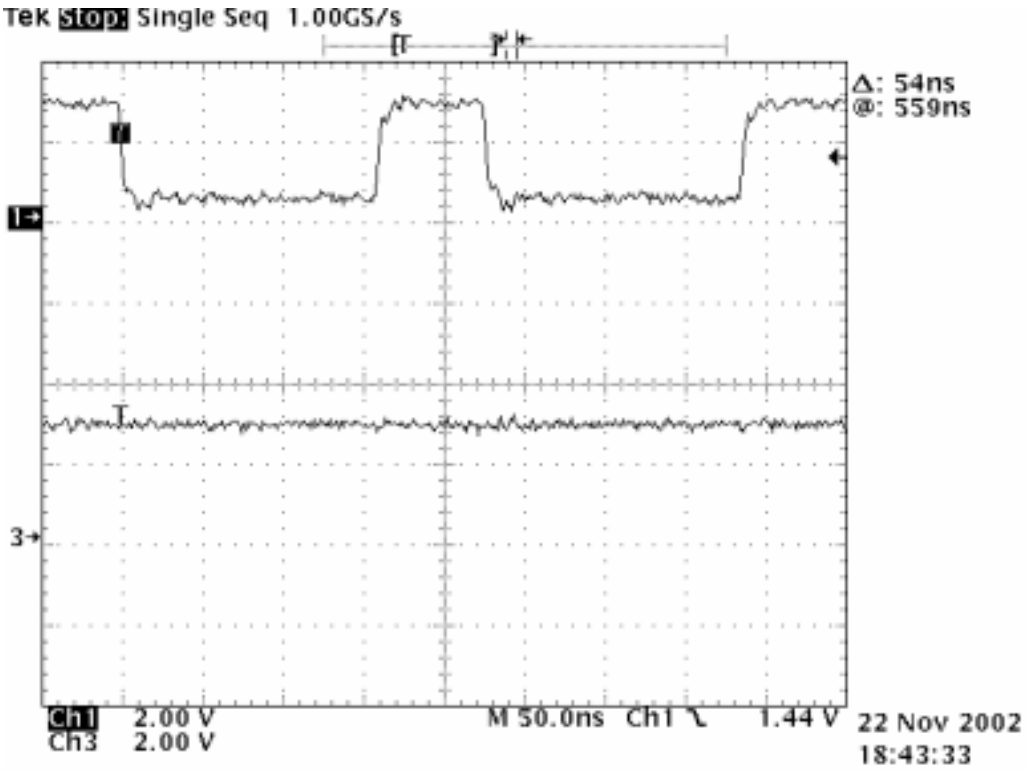

Figure 13. 32-Bit Data Reads from SRAM

3.2.2.12 32-Bit Data Writes to SRAM

Two consecutive 32-bit writes to SRAM are shown in Figure 14. Just as in the case of the 32-bit write, the $\overline{CE}$ signal stays active throughout the double word write. The write cycle begins when $\overline{OE}$ goes low, followed by $\overline{WE}$ one EMIFS clock cycle later. The $\overline{WE}$ signal pulses low for five EMIFS clock cycles, returns high for two clock cycles, then pulses low again for five clock cycles for the second 16-bit transfer of the 32-bit write. One cycle after $\overline{WE}$ returns inactive, $\overline{CE}$ goes inactive to end the write cycle, and remains inactive for four cycles before starting the next double-word write. The EMIFS can complete one write cycle of a series of double-word writes in 18 EMIFS clock cycles.

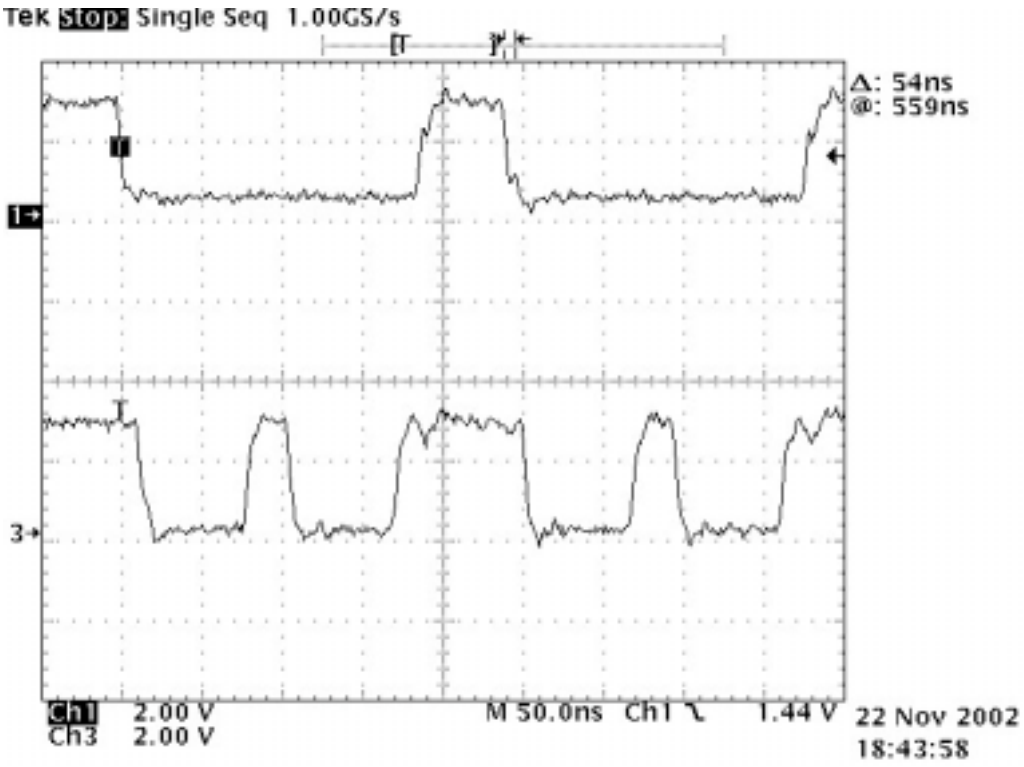

Figure 14. 32-Bit Data Write to SRAM

3.2.2.13 Instruction Fetches from SRAM

An instruction fetch for the TMS320C5x is always a 32-bit read operation. Figure 15 shows two consecutive instruction fetches from SRAM with the instruction cache disabled. The timing is indistinguishable from a 32-bit data read. The EMIFS can return one 32-bit fetch packet from the SRAM every 17 EMIFS clock cycles.

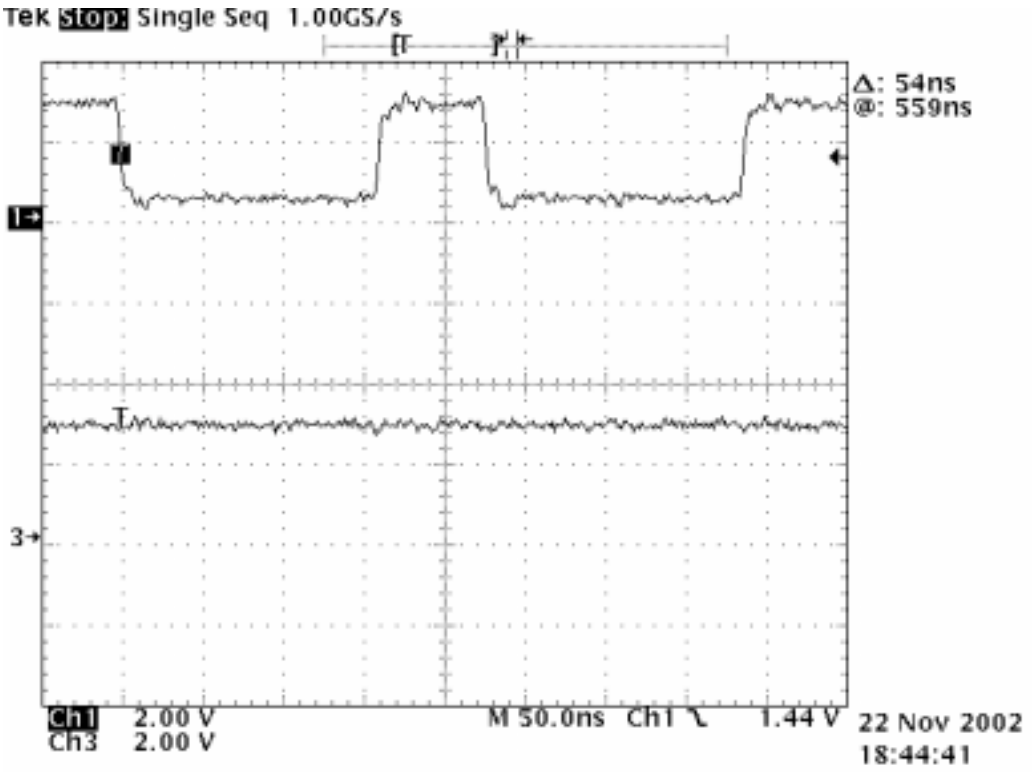

Figure 15. Instruction Fetches from SRAM with Instruction Cache Disabled

3.2.2.14 Instruction Cache Line Fill from SRAM

When the instruction cache is enabled, a cache miss will cause the cache controller to read one cache line of 16 bytes from memory. At the EMIFS, this generates a series of eight 16-bit reads. Figure 16 shows three consecutive line-fill program accesses.

The eight-word read starts when $\overline{CE}$ goes low. It remains low for 8 x 6 = 48 EMIFS clock cycles while the eight consecutive addresses for the line fill are driven onto the address bus. At the end of the multi-word read $\overline{CE}$ goes inactive and remains high for 10 clock cycles. In this configuration the EMIFS can return 16 bytes, or four 4-byte fetch packets, every 58 EMIFS clocks, for a throughput of 14.5 EMIFS clocks per fetch packet.

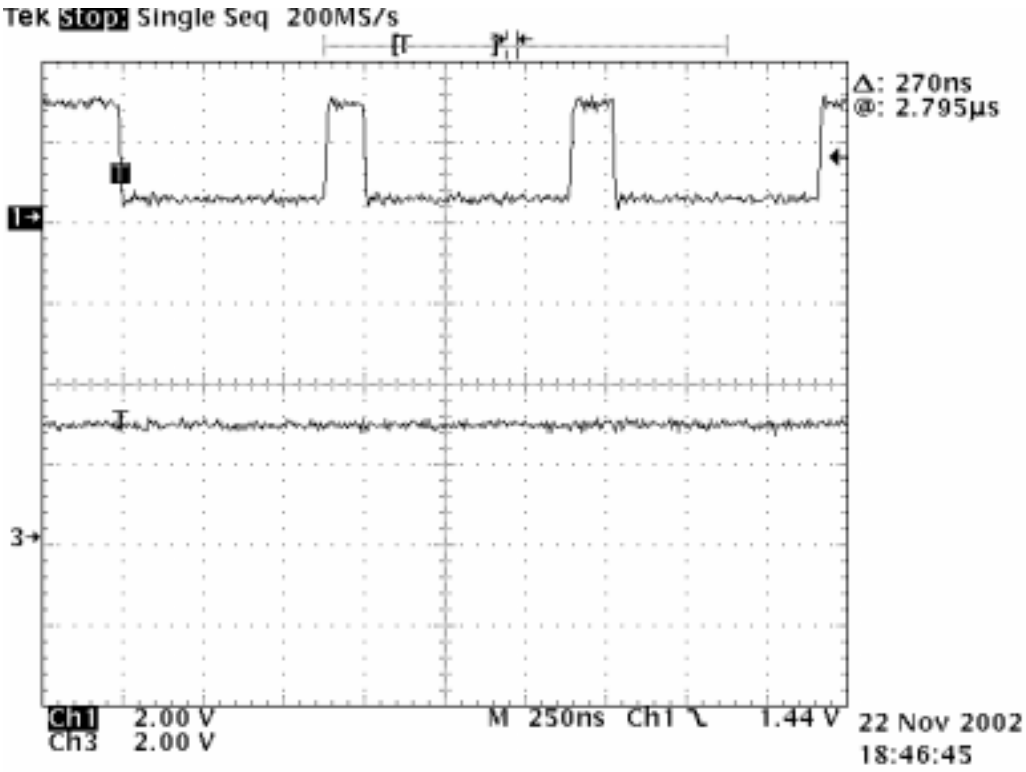

Figure 16. Instruction Cache line Fills from SRAM

4 Conclusions and Recommendations

While the DSP can access any memory within the OMAP5910 memory map, there is a substantial performance penalty for accessing memory outside of the DSP core. The internal SRAM can be accessed with the smallest penalty, as might be expected, followed by SDRAM, with external SRAM requiring the longest accesses.

If the DSP core memory is not sufficient to support an application, the internal memory should be allocated to data in preference to program. Program fetches by the TMS320C55x DSP are always made in 32-bit fetch packets. The OMAP5910 TC and memory interfaces provide better throughput for double-word accesses than for single-word accesses. This is especially true of the IMIF, which provides a 32-bit data path. If DSP code must be fetched from off-chip, the instruction cache should be enabled for best performance. The I-cache improves performance two ways. Once an instruction has been cached, it can be fetched from the cache with no performance penalty. Even when a cache miss occurs, the performance penalty is not as severe for the cache fill as for an isolated instruction fetch from external memory, because when the I-cache accesses external memory to fill a cache line, it always results in an 8-word burst transfer rather than a series of individual accesses. This not only reduces the total time for the external accesses, it uses less TC and memory interface bandwidth, making more available for access by the ARM9 MPU.

Data accesses to external memory are particularly expensive. For data accesses, the internal SRAM again provides the lowest penalty. Isolated random reads from SDRAM are particularly costly because each access incurs the overhead to initiate an SDRAM burst transfer, including the $\overline{\text{RAS}}$ cycle for a single data item. If external data accesses are unavoidable, the penalty can be reduced somewhat by performing double-word accesses rather than single word accesses, but only if the transfer of the second word is useful. Instead of allowing the DSP to make external data accesses directly, it might be preferable to use the DSP DMA controller to transfer data between DSP internal and external memory.

The system designer should take special care in selecting SDRAM to be used with the OMAP5910 device so that one of the four available SDRAM timing options is a close match to the memory device requirements. For example, if the SDRAM device used in the tests performed for this report were replaced with a higher performance part it would be possible to use a faster timing configuration and reduce the fetch packet access time by one TC clock cycle, which could have a significant impact on system performance.

5 References

1. *OMAP5910 Dual-Core Processor Technical Reference Manual* (SPRU602)
2. *OMAP5910 Dual-Core Processor Data Manual* (SPRS197)
3. *K4S561632 256Mbit SDRAM Data Sheet*, Samsung Electronics
4. *K6T4016V3C 256K x 16 bit Low Power and Low Voltage CMOS Static RAM Data Sheet*, Samsung Electronics

Appendix A DSP Memory Map

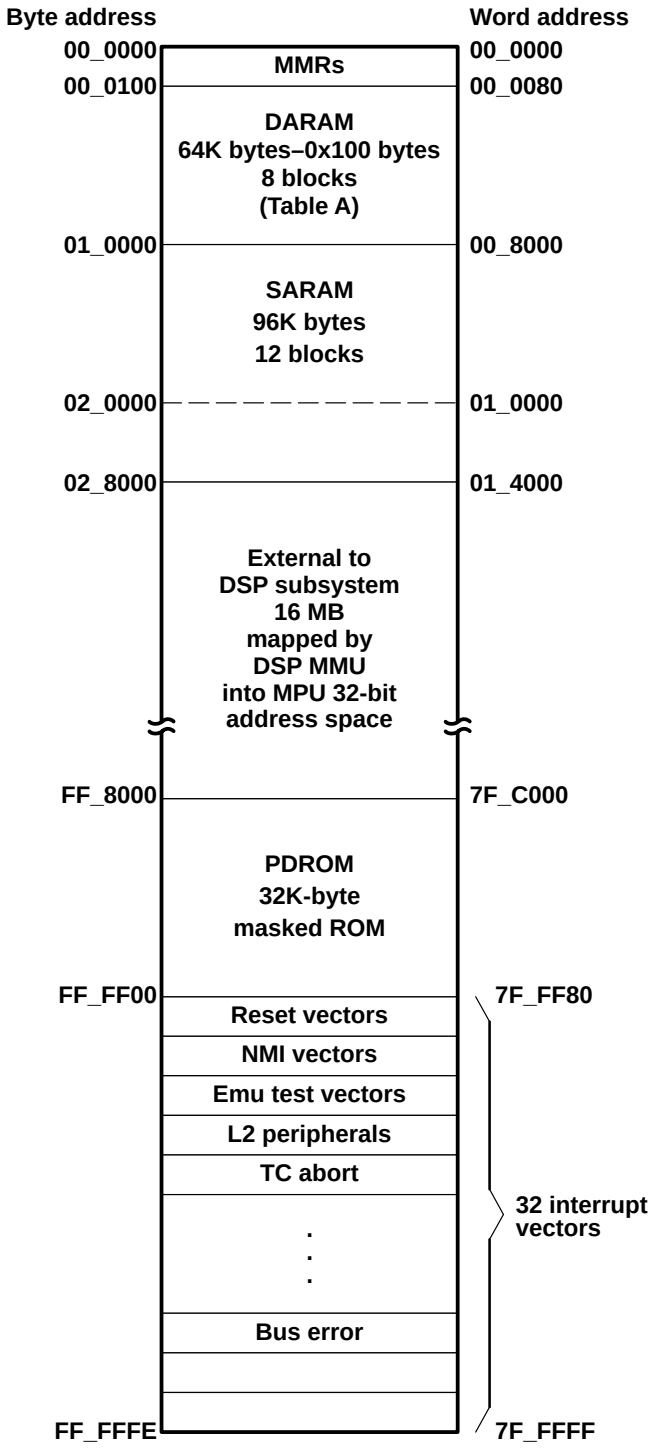

Figure A–1. DSP Memory Map

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265