Application Report
SPRA891 – January 2003

OMAP5910 ARM Program Throughput Analysis

Jon Hunter *Associate Technical Staff, DSP Applications*

ABSTRACT

The OMAP™ device is built upon a dual-core architecture that consists of a TIARM925T MPU and a C55x™ DSP device. Both cores have access to internal memory via an internal memory interface (IMIF) as well as external memory via two external memory interfaces, namely the EMIF Slow (EMIFS) and EMIF Fast (EMIFF). This application report focuses on the ARM side of the device and investigates the performance that can be achieved for executing program code that is located in internal or external memory. Examples will be given for instances where program code is placed in external memories such as SDRAM and SRAM.

Contents

List of Figures

List of Tables

1 Introduction

The DSP Catalog OMAP device is built upon a dual-core architecture that consists of a TIARM925T MPU and a C55x DSP. Both cores have access to internal memory via an internal memory interface (IMIF) as well as external memory via two external memory interfaces, namely the EMIF Slow (EMIFS) and EMIF Fast (EMIFF).

This application report focuses on the ARM side of the device and investigates the performance that can be achieved for executing program code that is located in internal or external memory. This document will also provide a brief description of the MPU subsystem, data paths for internal and external memory, and will show how to achieve the best performance for MPU program accesses.

2 Overview of the MPU Subsystem

The main features of the MPU subsystem are:

- MPU core
- 16KB 2-way set associative instruction-cache with 16 byte line size
- 8KB 2-way set associative data-cache with 16 byte line size
- 32-bit x17 deep Write Buffer
- Memory Management Unit (MMU)

The OMAP5910 MPU core, namely the TIARM925T, is a 32-bit reduced instruction set (RISC) processor. The MPU has the ability to perform 32-bit or 16-bit instructions and process 32-bit, 16-bit, and 8-bit data. The 16-bit instruction format is an extension to the ARM architecture and provides a subset of the original 32-bit instructions. The 16-bit instruction set is also known as the “thumb” instruction set and has been developed to reduce code size. Thumb instructions are compressed versions of some of the original 32-bit instructions. Hence, prior to the execution, a thumb instruction is decompressed by the MPU. The thumb instruction set does have its limitations by reducing the number of instructions available and limiting the number of general-purpose core registers available. However, the ability to switch between 32-bit instructions and 16-bit instructions during the program flow can allow significant savings in code size.

The TIARM925T MPU has a five-stage pipeline architecture that allows different phases of different instructions to be performed simultaneously. For example, the MPU five-stage pipeline consists of a fetch, decode, execute, data memory access, and register write phase. Therefore, while the first instruction is being decoded, the second can be fetched.

Since the MPU is a 32-bit processor, it has a 32-bit addressing range and hence, provides a 4GB address space. Specific details of the MPU memory map can be found in the *OMAP5910 Dual-Core Technical Reference Manual* (SPRU602).

The MMU provides the ability to use virtual space addressing and permission checking within the system. In other words, the MMU performs virtual to physical address translations and checks whether the process wishing to access a particular memory region has permission to do so.

The instruction-cache (I-cache), data-cache (D-cache), write-buffer, and MMU can be enabled or disabled within the system as the user wishes. However, it should be noted that the I-cache can be used independently of the MMU, whereas the data-cache and write-buffer cannot. Therefore, the use of I-cache does not require enabling the MMU, whereas the use of the data-cache and write buffer do.

A detailed description of the MPU subsystem can be found in *OMAP5910 Dual-Core Technical Reference Manual* (SPRU602).

3 Overview of the OMAP5910 Traffic Controller and Memory Interfaces

The OMAP5910 can support up to three shared memories, via three memory interfaces, namely the internal memory interface (IMIF), external memory interface slow (EMIFS) and external memory interface fast (EMIFF). The IMIF provides a 32-bit interface to the internal 192KB SRAM on the OMAP5910. Unlike the IMIF, the EMIFS and EMIFF are user-configurable and provide a 16-bit interface to a variety external memory. The EMIFS supports memories such as FLASH, ROM, SRAM and synchronous burst FLASH, whereas the EMIFF only supports SDRAM.

The on-chip devices that have access to the three memory-interfaces are the MPU, DSP, System DMA, DSP DMA and USB Host Controller. With up to five devices being able to access any of the shared-memories within the system, there is an important need to manage the “traffic” within the device. Hence, the task of the OMAP5910 traffic controller is to manage all accesses made by the five on-chip devices to shared-memory.

4 MPU Program Performance Analysis

4.1 System Setup

The following were used in the course of creating this application report:

- Test board with:
  - 256Mbit Infineon SDRAM (HYB39S256160AT-8)
  - 512KB Samsung SRAM (KM616U4000CLT-7L)
- Tektronix TLA704 logic analyzer
- POMAP5910CGZG

- Spectrum Digital XDS510PP_Plus emulator
- PC running Windows™ 2000 OS
- Code Composer Studio™ v2.1 for OMAP

4.2 OMAP5910 Configuration

Prior to evaluating the program performance of the MPU, the OMAP5910 was configured by running an initialization routine. The initialization routine configured the OMAP5910 in the following manner:

4.2.1 OMAP5910 Clock Domains

The OMAP5910 clock domains were configured as shown in Table 1.

Table 1. Clock Domain Settings

Clock Domain	Frequency
MPU	150MHz
DSP	150MHz
Traffic Controller	75MHz
Peripheral Clocks	75MHz
LCD	75MHz
DSP MMU	75MHz

The above clock domains are derived from the OMAP5910 digital phase locked loop (DPLL), which was configured for 150MHz clock generation, in synchronous scalable mode. Note that the output of the OMAP5910 DPLL is itself derived from an external 12MHz clock.

Other than the MPU and traffic controller (TC) clock domains, all other clock domains are not relevant for these tests. However, in order to set the MPU and TC domains all had to be configured, and so they were set as shown above. Note that the settings of these other timer domains will not have any direct effect on the results.

4.2.2 EMIFS Configuration

The EMIFS is configured by specifying:

- For a write operation:

  The number of wait states between write operations (WRWST)

  The number of wait states for which the write enable signal is asserted (WELEN)
- For a read operation, the number of wait states between the assertion of chip select and output being valid (RDWST).

The EMIFS has a total of four chip selects and for each chip select there is a separate EMIFS configuration register. Therefore, to configure the EMIFS for a particular memory, the appropriate configuration register must be programmed with the required WRWST, WELEN, and RDWST value. The EMIFS internal clock reference for each chip select is derived from the traffic controller clock, which can be divided down as required by setting the FCLKDIV value in the appropriate EMIFS configuration register. Table 2 shows how the values WRWST, WELEN, and RDWST are related to FCLKDIV and can be used to configure read/write cycle times for the EMIFS.

Table 2. EMIFS $\overline{CS}$ Active Widths for Asynchronous Reads/Write

FCLKDIV	$\overline{CS}$ Active Width Read (TC cycles)	$\overline{CS}$ Active Width Write (TC Cycles)
/1	1*(RDWST+1)+1	1*(WRWST+WELEN+1)+2
/2	2*(RDWST+1)+2	2*(WRWST+WELEN+1)+4
/4	4*(RDWST+1)+4	4*(WRWST+WELEN+1)+8
/6	6*(RDWST+1)+6	6*(WRWST+WELEN+1)+12

The EMIFS was connected to a 512KB SRAM, which had a minimum access period of 70ns. For this application, the EMIFS was configured using the following setup:

FCLKDIV = 1, RDWST = 4, WRWST = 1, WELEN = 3

Table 3 shows the minimum timing requirements for the SRAM and actual timing that the EMIFS was configured to ensure correct operation. Note that even though the write cycle was not optimal, this was not a concern as only read accesses were being analyzed.

Table 3. EMIFS Timing

Parameter	Symbol	SRAM Timing Requirement	EMIFS Configuration
Read Cycle Time	t_{RC}	70ns (min)	80ns
Write Cycle Time	t_{WC}	70ns (min)	93.3ns

4.2.3 EMIFF Configuration

The EMIFF timing is configured by specifying the following parameters:

- SDRAM Frequency Range: This parameter is used to group SDRAMs into four categories based on operating speed alone. Therefore, the read/write timing for an SDRAM is determined by specifying the appropriate SDRAM Frequency Range category.
- SDRAM Type: This parameter is used to indicate the SDRAM’s size, width and number of banks.
- Refresh Mode and Refresh Period
- CAS Latency

The EMIFF was connected to a 256Mb SDRAM, which had a maximum operational speed of 125MHz. For this application, the EMIFF was configured using the following setup:

- SDRAM Frequency Range: SDF0
- SDRAM Type: 256Mb, 16-bit width, 4 banks
- Refresh Mode and Refresh Period: Auto-refresh @ 585 TC cycles
- CAS Latency = 2

Table 4 shows the minimum timing requirements for the SDRAM and actual timing that the EMIFF was configured to use to ensure correct operation. Note that the below timing may not appear that optimal, but this was the only configuration available that would meet the SDRAM timing specification.

Table 4. EMIFF Timing

Parameter	Symbol	SDRAM Timing Requirement	EMIFF Configuration
Row cycle time	t_{RC}	70 ns (min)	120 ns
Row active time	t_{RAS}	48 ns (min)	67 ns
Row precharge time	t_{RP}	20 ns (min)	40 ns
RAS to CAS delay	t_{RCD}	20 ns (min)	26 ns
Row active to row active delay	t_{RRD}	16 ns (min)	26 ns
CAS latency	CASL	2 clocks	2 clocks

4.2.4 Traffic Controller Configuration

The traffic controller manages all accesses between the three memory interfaces, namely the IMIF, EMIFS and EMIFF and the requestors to these interfaces such as the MPU, DSP, DMAs and USB host controller. At any one time there can be potentially more than one requestor wishing to access the same memory interface. By default no requestor has priority over another, so in this case the traffic controller will make sure that the interface is shared fairly between requestors. However, the traffic controller has three priority registers that allow the user to set the priority of each requestor for each memory interface accordingly. A requestor of higher priority will be allowed to retain access to a memory interface for a longer duration before having to share it with another requestor of lower priority.

For this particular investigation, the memory interface priority registers were not reconfigured and their default setting was used. In other words, no requestor had priority over another. However, when only program accesses are being performed by the MPU alone, there is only one requestor, and so setting these registers with different priority configurations would not have made a difference.

4.2.5 MPU MMU Configuration

During the evaluation of the MPU program performance, tests were carried out with MPU MMU enabled and with the MPU MMU disabled.

For the tests where the MMU was enabled, it was configured as follows:

- 4096 1Mb sections
- Translation Table located in external SDRAM
- Full read/write permissions to all sections

All sections are cacheable apart from:

- MPU and DSP peripheral address space
- MPU MMU translation table

4.2.6 DSP Configuration

For evaluating the MPU's program performance, it was not necessary for the DSP to be operational and so the DSP was held in reset for the duration of the analysis. Note that despite configuring the DSP clock domain to operate at 150MHz (as shown above) the DSP clock was never actually enabled.

4.3 MPU Program Performance Examples

The program performance examples provide analysis of the throughput that can be achieved when program code is located in internal SRAM, external SDRAM, and external SRAM. These examples also investigate the effects of the I-cache, MMU and thumb instruction set on the MPU program performance.

It should be noted that the instructions executed during these tests by the MPU were simply a series of MOVE operations that had the same source and destination registers. In other words, these MOVE operations are essentially dummy commands and thus, are referred to as NOPs (no operation).

The program performance of the MPU was evaluated by:

- Using a 6MHz internal timer on the OMAP5910.
- Using a GPIO to trigger a logic analyzer and capture the activity at the external memory interfaces. Note that when executing program code located in internal SRAM, it was not possible to capture the bus activity using a logic analyzer.

With the I-cache enabled, tests were produced to evaluate 100% cache-misses and 100% cache-hit cases, to illustrate the worst and best case scenarios. Note that in order to generate a 100% miss scenario, the I-cache was flushed prior running the test; for a 100% hit scenario, the I-cache was filled prior to running the test with the test code.

However, given that the I-cache is only 16KB, only a total of 4096 32-bit or 8192 16-bit instructions can reside in the cache. Therefore, when executing a test case with more instructions than could be accommodated within the I-cache, it was necessary to configure the test code as a loop of *N* instructions that could reside within the cache. Note that the number of times the loop was executed was dependent on the total number of instructions that were to be executed in the test itself. It was also found that filling the whole cache would not generate the most reliable data, because prior to executing the test routine after filling the cache, intermediate instructions for setting up a timer or GPIO would thrash regions of the I-cache. Therefore, a 100% cache-hit scenario would not be produced. To overcome this issue, it was decided that a 100% cache-hit scenario could be produced by filling only half of the cache and taking advantage of the 2-way set associative configuration of the I-cache.

In the following subsections the terms are used:

- **Latency between reads** describes the period between latching the instruction at a memory interface, to the start of the next read operation seen at a memory interface. Hence, this latency accounts for the time taken for the instruction to propagate through the TC to the MPU, and then for the request from the MPU to make the next read access to propagate back through the TC to the memory interface.
- **MPU cycles per instruction read** describes the average number of MPU cycles for each instruction read. This value was calculated by using the 6MHz timer to measure the time taken for the MPU to execute 40000 NOP instructions and then dividing the result by 40000, to get the average number of cycles per instruction. This value was also verified for instruction fetches from the external memory, by viewing the activity on the external memory interfaces with a logic analyzer.
- **MPU cycles per cache line fill** describes the average number of MPU cycles for each cache line fill. This value was calculated by using the “MPU cycles per instruction read” and multiplying this value by the number of instructions in a cache line. This value was also verified for external memories, by viewing the activity on the external memory interfaces during a cache miss with a logic analyzer.
- **ARM** and **thumb** refer to where the MPU uses the 32-bit instruction set and 16-bit instruction set, respectively.

4.3.1 External SRAM Examples

Table 5 summarizes the MPU program throughput achieved, when the program code was located in external SRAM.

Table 5. External SRAM Performance

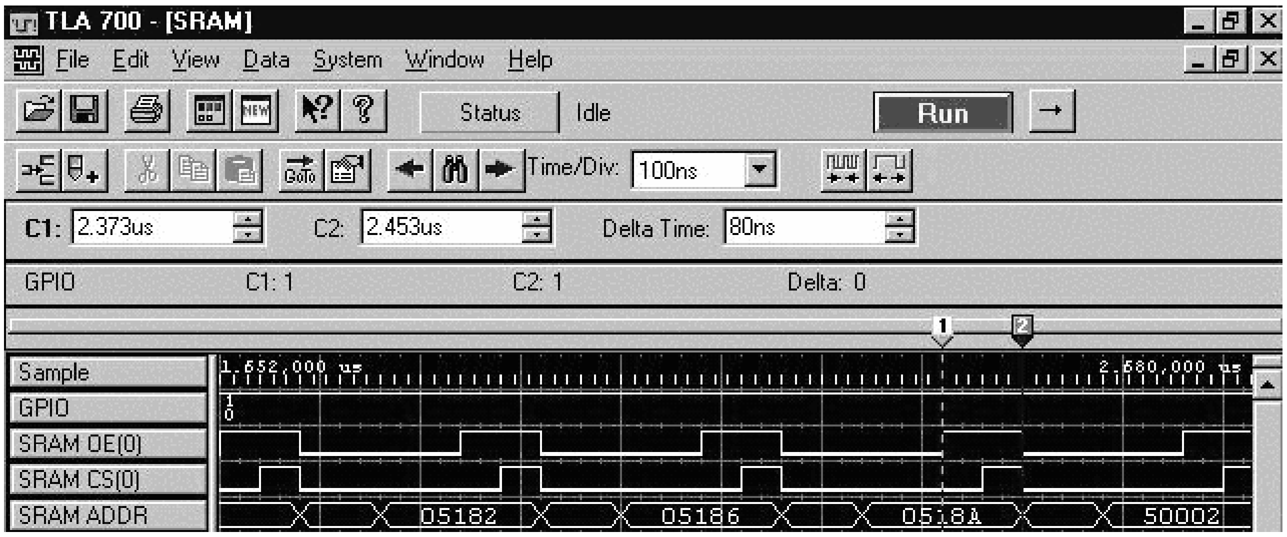
MPU Mode	Instruction Cache	MMU	Number of NOPs	Latency Between Reads (MPU Cycles)	MPU Cycles Per Instruction Read	MPU Cycles Per Cache Line Fill
ARM	OFF	OFF	40000	12	36	–
ARM	OFF	ON	40000	See Note 1	36	–
ARM	Miss	OFF	40000	12	27	108
ARM	Miss	ON	40000	See Note 1	27	108
ARM	Hit	OFF	40000	See Note 2	1	–
ARM	Hit	ON	40000	See Note 2	1	–
Thumb	OFF	OFF	40000	12	24	–
Thumb	OFF	ON	40000	See Note 1	24	–
Thumb	Miss	OFF	40000	12	13.75	110
Thumb	Miss	ON	40000	See Note 1	13.75	110
Thumb	Hit	OFF	40000	See Note 2	1	–
Thumb	Hit	ON	40000	See Note 2	1	–

NOTES: 1. The MMU performs virtual to physical address translation, by using a translation table that maps virtual memory sections to physical memory sections. The location of the translation table is user-defined and so can be placed in any memory location the MPU can access. When the MMU is enabled, to avoid having to always reference the translation table when performing a virtual to physical address mapping, the MMU has translation look-aside buffers (TLBs) that are used to store the addresses of most recent memory sections that have been accessed. Therefore, with respect to the above table, when the MMU is enabled, the latency for the first read will be dependent on whether the address falls into a memory section whose physical address is already located in the TLBs. If it is located in the TLBs, then the latency of the first read will be the same as for the cases where the MMU is disabled. Otherwise, the latency of the first read will be delayed by the time taken to fetch the address from the memory where the translation table is located

2. When the I-cache is enabled and a cache hit occurs, because there is no activity on the EMIFS, it is not possible to measure the latency of the first read.

Table 5 shows that when the I-cache is disabled, the average instruction read time is greater than when the I-cache is enabled and a miss occurs. Therefore, performance through the EMIFS with the I-cache disabled, does not reflect the performance with the I-cache enabled during a cache-miss. The reason for this can be explained by referring to the following two figures. Figures 1 and 2 show the activity observed on the EMIFS address bus, output-enable (OE) signal, and chip-select (CS) signal during a sequence of external SRAM read accesses made by the MPU, with the I-cache disabled and enabled, respectively. Note that in these waveforms each address seen on the SRAM ADDR bus represents a 16-bit read.

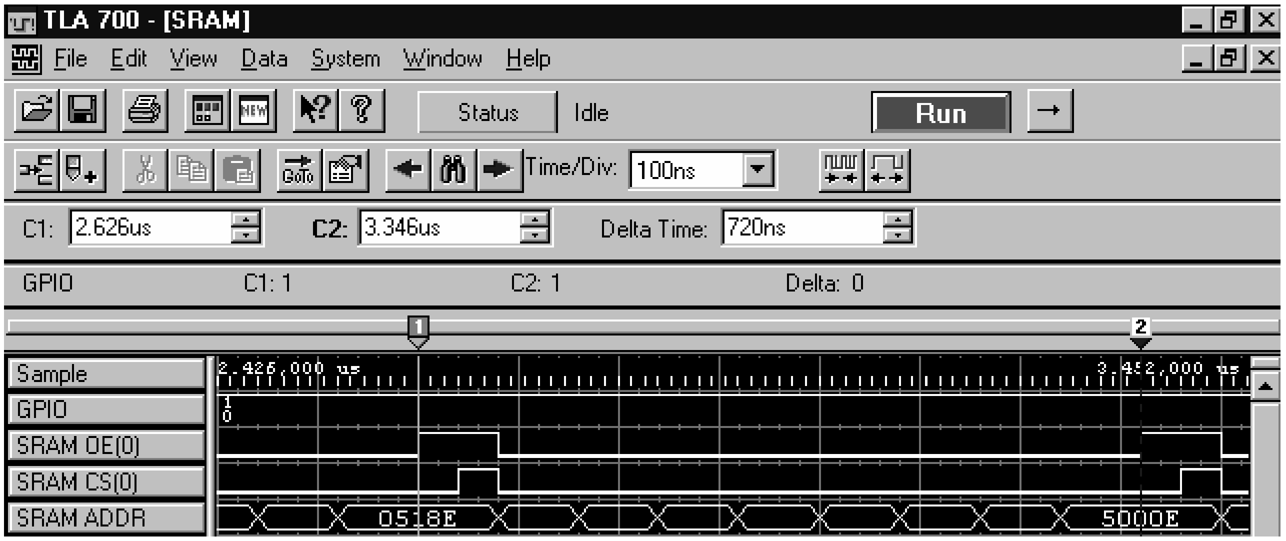
Figure 1 shows that two 16-bit words are being read per chip-select, because in this case, the MPU is using the 32-bit instruction set and so bursts two 16-bit reads back-to-back. However, between 32-bit instruction reads, there is a delay of 80ns (12 MPU cycles) that occur. Note that this delay is the latency between reads that is shown in table 5 and accounts for the propagation delay through the EMIFS and TC.



Idle
Time/Div: 100ns
C1: 2.373us C2: 2.453us Delta Time: 80ns
GPIO C1: 1 C2: 1 Delta: 0

Figure 1. External SRAM Read Accesses with I-Cache Disabled (ARM Mode)

Figure 2 illustrates that with the I-cache enabled, eight consecutive 16-bit reads occur per chip-select, which is equivalent to an I-cache line fill. It also shows that a delay of 80ns (12 MPU cycles) only occurs between cache line fills, as opposed to between every instruction read when the I-cache is disabled. For this reason we see an improvement in the average time between instruction reads with the I-cache enabled.

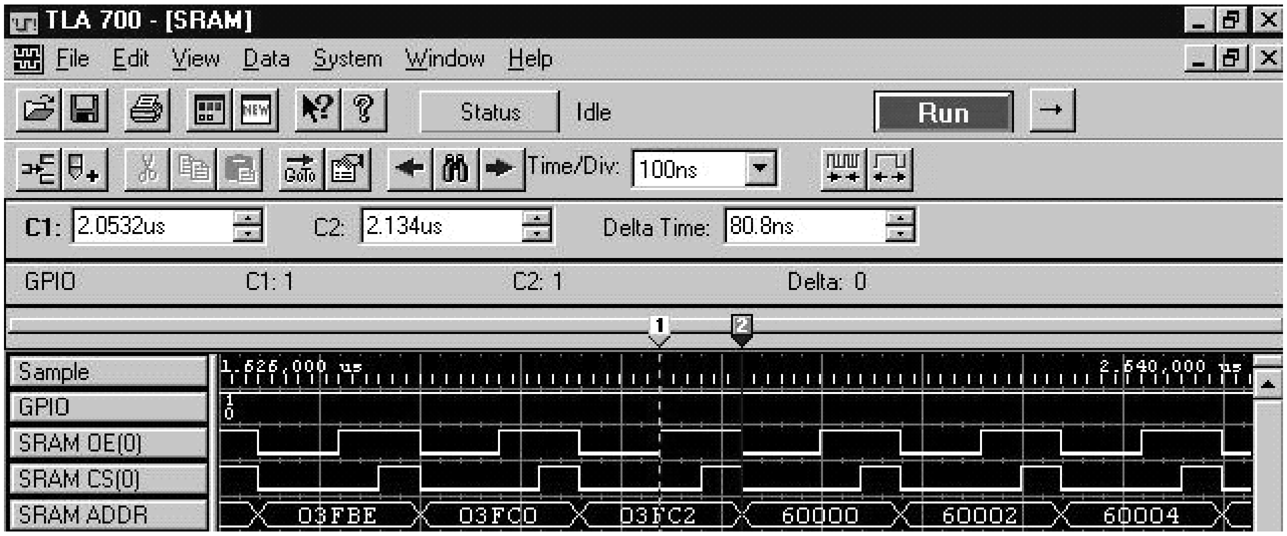


Idle
Time/Div: 100ns
C1: 2.626us C2: 3.346us Delta Time: 720ns
GPIO C1: 1 C2: 1 Delta: 0

Figure 2. External SRAM Read Accesses with I-Cache Enabled (ARM Mode)

When the I-cache is enabled during a cache-hit, the MPU achieves an average time of one MPU cycle per instruction read. When the I-cache is disabled, the average time for a 16-bit instruction (thumb) fetch is less efficient in terms of MPU cycles per bit than the average time for a 32-bit instruction (ARM) fetch. The reason for this is explained in Figure 1 and Figure 3. They show the activity observed on the EMIFS address bus, output-enable (OE) signal and chip-select (CS) signal during a sequence of external SRAM read accesses made by the MPU, in ARM and thumb modes, respectively. Note that in these waveforms each address seen on the SRAM ADDR bus represents a 16-bit read.

Figure 1 shows that two 16-bit words are being read per chip-select, because in this case the MPU is using the 32-bit instruction set and so bursts two 16-bit reads back-to-back. This figure shows that between 32-bit instruction reads that a delay of 80ns (12 MPU cycles) occurs. Note that this delay is the latency between reads that is shown in Table 5 and accounts for the propagation delay through the EMIFS and TC.



Time/Div: 100ns

C1: 2.0532us C2: 2.134us Delta Time: 80.8ns

GPIO C1: 1 C2: 1 Delta: 0

Figure 3. External SRAM Read Accesses with I-Cache Disabled (Thumb Mode)

Figure 3 shows that one 16-bit word is being read per chip-select, due to the fact that in this case the MPU is using the 16-bit instruction set. This figure also shows that that a delay of 80ns (12 MPU cycles) occurs between every 16-bit instruction read, as opposed to between every 32-bit instruction read when using the 32-bit instruction set. Hence, there is an improved efficiency in terms of MPU cycles per bit when using the 32-bit instruction set with the I-cache disabled.

When the I-cache is enabled, 16-bit and 32-bit instruction formats have nearly the same efficiency in terms of MPU cycle per bit. This would be expected because the time taken to fill a cache line should be the same irrespective of instruction size. However, an I-cache line fill takes two MPU cycles longer when using the 16-bit instruction set.

Enabling the MMU, regardless of whether the I-cache was enabled or disabled, did not have any noticeable effect on the program performance.

4.3.2 External SDRAM Examples

Table 6 summarizes the MPU program throughput achieved, when the program code was located in external SDRAM.

Table 6. External SDRAM Performance

MPU Mode	Instruction Cache	MMU	Number of NOPs	Latency Between Reads (MPU Cycles)	MPU Cycles Per Instruction Read	MPU Cycles Per Cache Line Fill
ARM	OFF	OFF	40000	10	18	–
ARM	OFF	ON	40000	See Note 1	18	–
ARM	Miss	OFF	40000	10	7.5	30
ARM	Miss	ON	40000	See Note 1	7.5	30
ARM	Hit	OFF	40000	See Note 2	1	–
ARM	Hit	ON	40000	See Note 2	1	–
Thumb	OFF	OFF	40000	10	18	–
Thumb	OFF	ON	40000	See Note 1	18	–
Thumb	Miss	OFF	40000	10	4	32
Thumb	Miss	ON	40000	See Note 1	4	32
Thumb	Hit	OFF	40000	See Note 2	1	–
Thumb	Hit	ON	40000	See Note 2	1	–

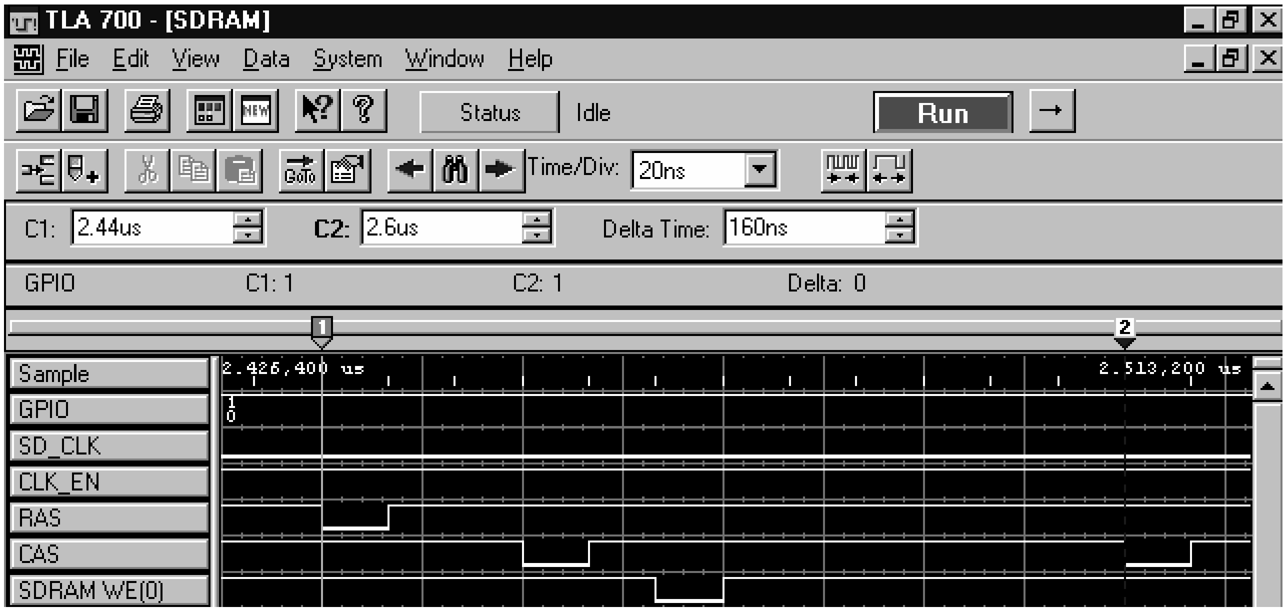
NOTES: 1. The MMU performs virtual to physical address translation, by using a translation table that maps virtual memory sections to physical memory sections. The location of the translation table is user-defined and so can be placed in any memory location the MPU can access. When the MMU is enabled, to avoid having to always reference the translation table when performing a virtual to physical address mapping, the MMU has translation look-aside buffers (TLBs) that are used to store the addresses of most recent memory sections that have been accessed. Therefore with respect to the above table, when the MMU is enabled the latency for the first read, will be dependent on whether the address falls into a memory section whose physical address is already located in the TLBs. If it is located in the TLBs, then the latency of the 1st read will be the same as for the cases where the MMU is disabled. Otherwise the latency of the first read will be delayed by the time taken to fetch the address from the memory where the translation table is located

2. When the I-cache is enabled and a cache hit occurs, then because there is no activity on the EMIFF, it was not possible to measure the latency of the first read.

Table 6 indicates that the average instruction read time when the I-cache is disabled is more than twice as much than when the I-cache is enabled and a cache miss occurs. Therefore, the performance through the EMIFF with the I-cache disabled, does not reflect the performance with the I-cache enabled during a cache-miss. The reason for this can be explained by referring to Figure 4 and Figure 5. They show the activity observed on the EMIFF row access strobe (RAS), column access strobe (CAS) and write enable (WE) signals during a sequence of external SDRAM read accesses made by the MPU, with the I-cache disabled and enabled, respectively. Note that in these figures, the TC cycles are illustrated by the top waveform labeled “sample”.

Figure 4 shows that the RAS goes low first to activate an SDRAM row. Three TC cycles after the RAS, the CAS goes low to start the read burst. Note that because the SDRAM has a CAS latency of 2, the first data word in the burst will appear on the EMIFF data bus two cycles after the assertion of the CAS.

Another two TC cycles after the CAS, the WE signal is activated to terminate the read burst at the same time the first 16-bit word appears on the EMIFF data bus. Again with a CAS latency of 2, the burst will not be terminated for another two cycles. Once a read burst has started, new data will appear on the EMIFF data bus every TC cycle. Therefore, by the time the burst has been terminated, two 16-bit words will have been read. This is to be expected because in this case the MPU is using the 32-bit instruction set and so will burst two 16-bit reads back-to-back. The latching of the second 16-bit word would occur two cycles after the assertion of the WE signal. Following the latching of the second 16-bit word there are 5 more TC cycles (10 MPU cycles) that occur before the next burst read command (CAS) is seen. This delay is the latency between reads that is shown in table 6 and accounts for the propagation delay through the EMIFS and TC.



Status: Idle

Time/Div: 20ns

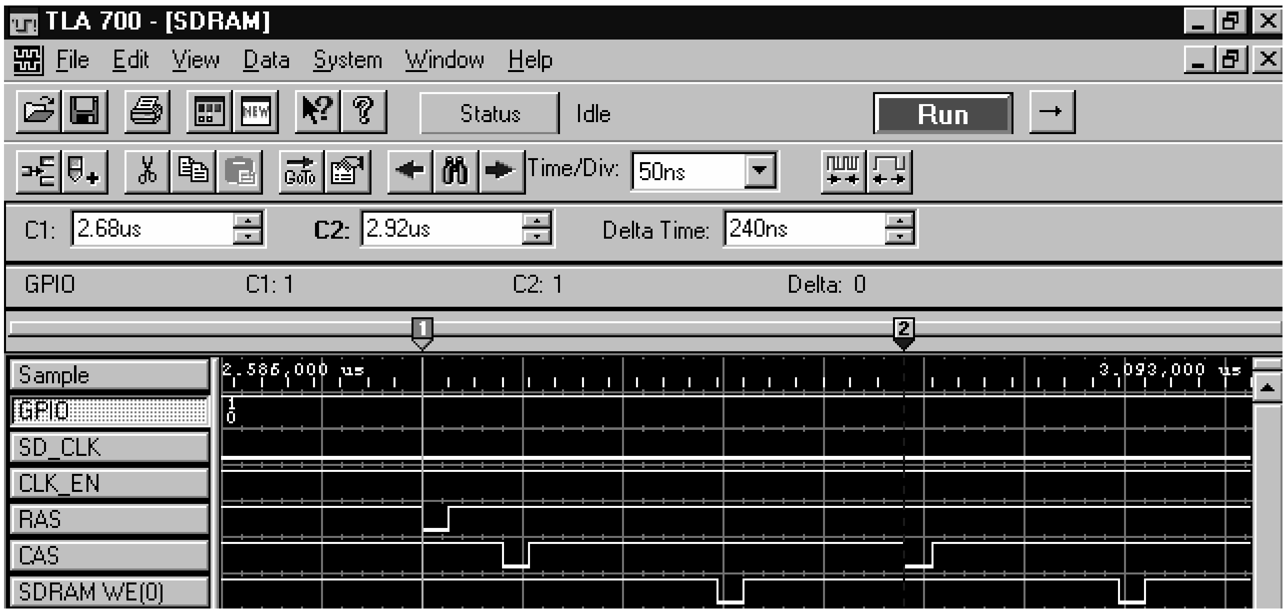
C1: 2.44us C2: 2.6us Delta Time: 160ns

GPIO C1: 1 C2: 1 Delta: 0

Figure 4. External SDRAM Read Accesses with I-Cache Disabled (ARM Mode)

Figure 5 shows that the RAS goes low first to activate an SDRAM row. Three TC cycles after the RAS, the CAS goes low to start the read burst. Note that because the SDRAM has a CAS latency of 2, the first data word in the burst will appear on the EMIFF data bus two cycles after the assertion of the CAS signal. Once a read burst has started, new data will appear on the EMIFF data bus every TC cycle until such a time when the burst is terminated.

Another eight TC cycles after the CAS, the WE signal is activated to terminate the read burst. Again with a CAS latency of 2, the burst will not be terminated for another two cycles. Therefore, by the time the burst has been terminated eight 16-bit words will have been read, which is equivalent to an I-cache line fill. The latching of the eighth 16-bit word will occur two cycles after the assertion of the WE signal. Following the latching of the eighth 16-bit word, there are five more TC cycles (10 MPU cycles) that occur before the next burst read command (CAS) is seen. This delay is the latency between reads that is shown in Table 6 and accounts for the propagation delay through the EMIFS and TC.



C1: 2.68us C2: 2.92us Delta Time: 240ns

Time/Div: 50ns

GPIO C1: 1 C2: 1 Delta: 0

Figure 5. External SDRAM Read Accesses with I-Cache Enabled (ARM Mode)

Figure 5 illustrates that with the I-cache enabled, a delay of 10 MPU cycles only occurs between cache line fills. However, Figure 4 shows that when the I-cache is disabled, a delay of 10 MPU cycles is seen between every instruction read. For this reason we see an improvement in the average time between instruction reads with the I-cache enabled.

When the I-cache is enabled during a cache-hit the MPU achieves an average time of one MPU cycle per instruction read.

When the I-cache is disabled the average time for a 16-bit instruction (thumb) fetch is less efficient in terms of MPU cycles per bit than the average time for a 32-bit instruction (ARM) fetch. The reason for this can be explained by referring to Figure 4 and Figure 6. These figures show the activity observed on the EMIFF row access strobe (RAS), column access strobe (CAS) and write enable (WE) signals during a sequence of external SDRAM read accesses made by the MPU, in ARM and thumb modes, respectively.

Figure 4 shows that between 32-bit instruction reads, a delay of 10 MPU cycles occurs. Note that this delay is the latency between reads that is shown in Table 6 and accounts for the propagation delay through the EMIFS and TC.

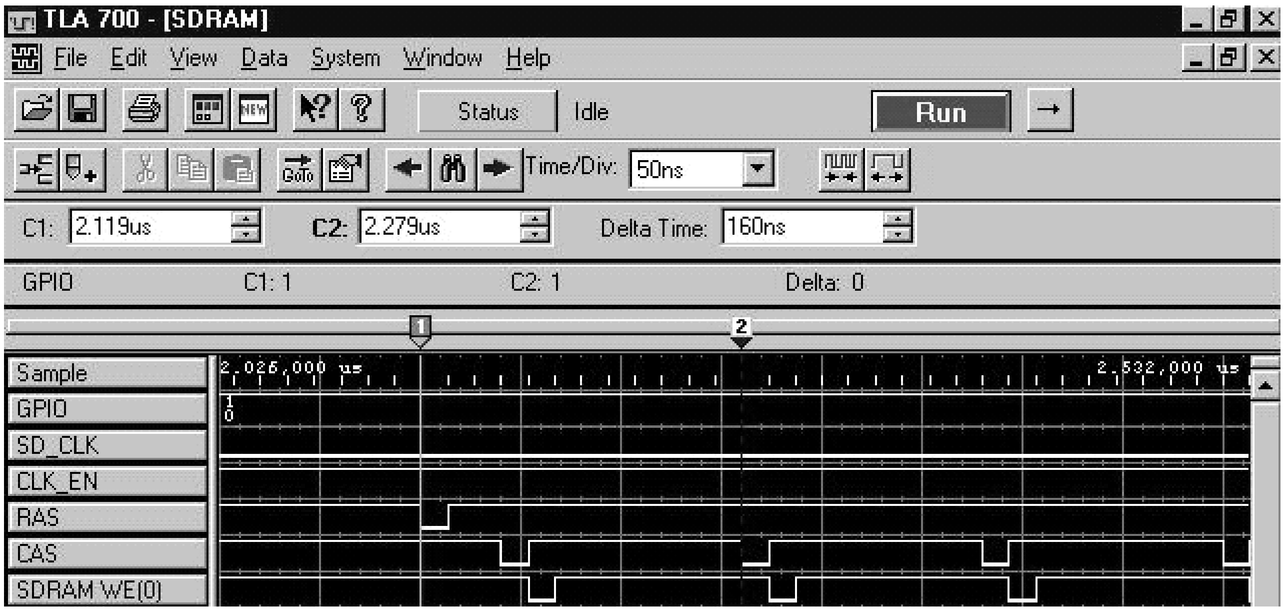


Figure 6. External SDRAM Read Accesses with I-Cache Disabled (Thumb Mode)

Figure 6 illustrates that the RAS signal goes low first to activate an SDRAM row. Three TC cycles after the RAS, the CAS goes low to start the read burst. Note that because the SDRAM has a CAS latency of 2, the first data word in the burst will appear on the EMIFF data bus two cycles after the assertion of the CAS.

Another TC cycle after the CAS, the WE signal is activated to terminate the read burst. Again with a CAS latency of 2, the burst will not be terminated for another two cycles. The first 16-bit word appears on the EMIFF data bus, one TC cycle after the WE is asserted. Hence, by the time the burst has been terminated, only one 16-bit word would have been read. This is to be expected because in this case, the MPU is using the 16-bit instruction set, and so will only read one 16-bit word at a time. The latching of the 16-bit word occurs two cycles after the assertion of the WE signal. Following the latching of the 16-bit word there are six more TC cycles (12 MPU cycles) that occur before the next burst read command (CAS) is seen. Note that this delay is the latency between reads that is shown in Table 6 and accounts for the propagation delay through the EMIFS and TC.

Figure 6 shows that when using the thumb instruction set, there is a delay of 12 MPU cycles between every 16-bit read. However, Figure 4 illustrates that there was only a delay of 10 MPU cycles for every 32-bit instruction that was read. Therefore, there is an improved efficiency in terms of MPU cycles per bit when using the 32-bit instruction set as opposed to the 16-bit instruction set when the I-cache is disabled.

When the I-cache is enabled, 16-bit and 32-bit instruction formats have nearly the same efficiency in terms of MPU cycle per bit. This would be expected because the time taken to fill a cache line should be the same irrespective of instruction size. However, it should be noted that an I-cache line fill, takes two MPU cycles longer when using the 16-bit instruction set.

Enabling the MMU, regardless of whether the I-cache was enabled or disabled, did not have any noticeable effect on the program performance.

4.3.3 Internal SRAM Examples

Table 7 summarizes the MPU program throughput achieved, when the program code was located in external SRAM.

Table 7. Internal SRAM Performance

MPU Mode	Instruction Cache	MMU	Number of NOPs	Latency Between Reads (MPU Cycles)	MPU Cycles Per Instruction Read	MPU Cycles Per Cache Line Fill
ARM	OFF	OFF	40000	See Note 1	8	–
ARM	OFF	ON	40000	See Note 1	8	–
ARM	Miss	OFF	40000	See Note 1	3.5	14
ARM	Miss	ON	40000	See Note 1	3.5	14
ARM	Hit	OFF	40000	See Note 1	1	–
ARM	Hit	ON	40000	See Note 1	1	–
Thumb	OFF	OFF	40000	See Note 1	8	–
Thumb	OFF	ON	40000	See Note 1	8	–
Thumb	Miss	OFF	40000	See Note 1	2	16
Thumb	Miss	ON	40000	See Note 1	2	16
Thumb	Hit	OFF	40000	See Note 1	1	–
Thumb	Hit	ON	40000	See Note 1	1	–

NOTE 1: The latency of the 1st read was not possible to measure for all internal SRAM benchmarks, simply because it is not feasible to monitor the activity of the internal buses within the device.

Table 7 indicates that when the I-cache is disabled, the average time of an instruction read is greater than when the I-cache is enabled and a miss occurs. Therefore, it should be noted that the performance through the IMIF with the I-cache disabled, does not reflect the performance with the I-cache enabled during a cache-miss. When an I-cache miss occurs, a complete cache line, consisting of 16 bytes, will be fetched in a single burst access, across the IMIF and through the TC. However, when the I-cache is disabled, each instruction is read individually over the IMIF and through the TC. Given that there is a fixed initial latency for each access that is made, the more data that is read per access will have the effect of reducing the total overhead. Hence, it is more efficient to read instructions in bursts with the I-cache enabled, than individually when I-cache is disabled.

When the I-cache is enabled during a cache-hit, the MPU achieves an average time of one MPU cycle per instruction read. When the I-cache is disabled the average time of a 16-bit instruction (thumb) fetch is less efficient in terms of MPU cycles per bit than the average time of a 32-bit instruction (ARM) fetch. In fact, the average number of MPU instructions per instruction read is the same for both 16-bit and 32-bit instructions. The reason for this is that the OMAP5910 has 32-bit buses running between the internal SRAM and MPU. Therefore, whether the MPU is reading 16-bit or 32-bit instructions the latency for fetching an individual instruction with the I-cache disabled will be the same.

When the I-cache is enabled, 16-bit and 32-bit instruction formats have nearly the same efficiency in terms of MPU cycle per bit. This would be expected because the time taken to fill a cache line should be the same irrespective of instruction size. However, it should be noted that an I-cache line fill, takes two MPU cycles longer when using the 16-bit instruction set.

Enabling the MMU, regardless of whether the I-cache was enabled or disabled, did not have any noticeable effect on the program performance.

5 Conclusions

In general, to ensure the most optimum MPU program performance, the I-cache should always be enabled. Disabling the I-cache will not allow the multiple instructions to be fetched in a single burst and hence, will have a considerable effect on the system performance. The 16-bit and 32-bit instruction sets have a comparable performance in terms of instructions per cycle, depending on memory, when the I-cache is disabled. However, when the I-cache is enabled, the 16-bit instruction set has nearly twice the performance of the 32-bit instruction set. This is due to twice the number of instructions being included per line fill. In addition to this, enabling the MMU had no noticeable effect on the system performance for this particular case. Though, it should be understood that the program code was located in a contiguous block of memory. So, it did not thrash the TLBs as much as some code may.

6 References

1. *OMAP5910 Dual-Core Processor Data Manual* (SPRS197)
2. *OMAP5910 Dual-Core Processor Technical Reference Manual* (SPRU602)
3. Samsung Electronics http://www.samsungelectronics.com
4. Infineon Technologies http://www.infineon.com

Appendix A OMAP Block Diagram

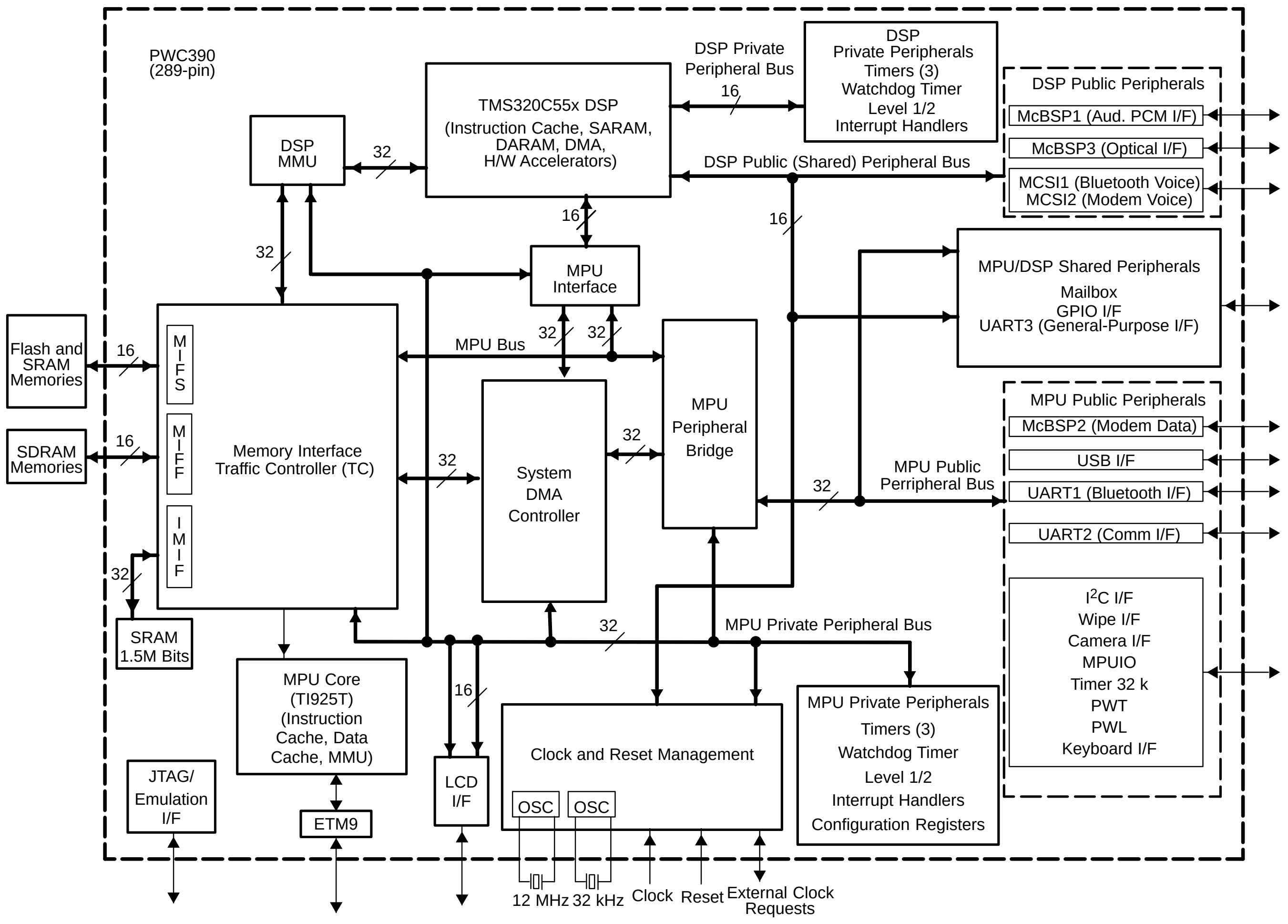


Figure A–1. OMAP Block Diagram

Appendix B MPU Memory Map

Table B–1. MPU Memory Map

Address Range	On-Chip	External Interface
0x0000 0000 0x01FF FFFF		FLASH CS0 32M bytes
0x0200 0000 0x03FF FFFF	reserved	
0x0400 0000 0x05FF FFFF		FLASH CS1 32M bytes
0x0600 0000 0x07FF FFFF	reserved	
0x0800 0000 0x09FF FFFF		FLASH CS2 32M bytes
0x0A00 0000 0x0BFF FFFF	reserved	
0x0C00 0000 0x0DFF FFFF		FLASH CS3 32M bytes
0x0E00 0000 0x0FFF FFFF	reserved	
0x1000 0000 0x13FF FFFF		SDRAM 64M bytes
0x1400 0000 0x1FFF FFFF	reserved	
0x2000 0000 0x2002 FFFF	Internal SRAM 192K bytes	
0x2003 0000 0x2FFF FFFF	reserved	
0x3000 0000 0x7FFF FFFF		Local bus space for USB host
0x8000 0000 0xDFFF FFFF	reserved	
0xE000 0000 0xE0FF FFFF	DSP public memory space (accessible by MPUI) 16M bytes	
0xE100 0000 0xEFFF FFFF	DSP public peripherals (accessible by MPUI)	
0xFFFB 0000 0xFFFC FFFF	MPU public, MPU/DSP shared peripherals	
0xFFFD 0000 0xFFFE FFFF	MPU private peripherals	
0xFFFF 0000 0xFFFF FFFF	reserved	