Engineer-to-Engineer Note EE-324

ANALOG DEVICES

Technical notes on using Analog Devices DSPs, processors and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

System Optimization Techniques for Blackfin® Processors

Contributed by Kaushal Sanghai

Rev 1 – July 10, 2007

Introduction

Efficient system resource utilization is critical for developing applications that demand high bandwidth on an embedded platform. Systems can often run out of bandwidth, even if the throughput requirements are within the limits of the system. The critical factors that result in lower than expected throughput are, more often than not, external memory access latencies and inefficient utilization of system resources. In order to fully exploit the capabilities of an embedded processor, it is important to understand its system architecture and the available system optimization techniques. This EE-Note serves as a quick reference to Blackfin® processor memory hierarchy and its system architecture. It also provides guidelines for using several optimization techniques to efficiently utilize the available system resources and discusses benchmark studies to evaluate and quantify the suggested optimization techniques.

Blackfin Processor Architecture

In this section, Blackfin memory hierarchy and system architecture is discussed. Each section begins with a description of a resource (memory, system bus, DMA controllers, etc.), followed by recommendations to use that resource more efficiently. The architecture is described mostly from a performance perspective, thus ignoring other details. Please refer to the appropriate *Hardware Reference* manual[1, 2, 3] for more details.

Memory Hierarchy

This section discusses the Blackfin processor's memory hierarchy (Figure 1) and the relative tradeoffs between on-chip (L1 and L2) memory and off-chip (external) memory. Guidelines are also provided to efficiently map code and data into the memory hierarchy to achieve minimal memory access latencies.

L1 Memory Description

Blackfin processors provide separate instruction and data L1 memory spaces. L1 data memory is further divided into data bank A and data bank B. For higher performance, L1 memory is implemented as single-ported sub-banks to allow for simultaneous access to multiple requesting elements (core, DMA, etc.). L1 memory also provides SRAM and cache memory configurability to take advantage of application-specific workload characteristics.

L1 Code and Data Memory Sub-Banking

L1 code memory is implemented as single-ported sub-banks, effectively making them dual-ported. Therefore, the core and DMA or system buses can access the L1 code memory simultaneously, as long as the accesses are not to the same sub-bank. Similarly, L1 data memory forms multi-ported sub-banks, allowing the following in a single core clock cycle:

- Two 32-bit data address generator (DAG) loads
- One pipelined 32-bit DAG store
- One 64-bit DMA I/O
- One 64-bit cache fill/victim access

L1 SRAM/Cache Configuration

By default, all of L1 memory is SRAM memory that provides direct access to the core. SRAM memory guarantees single-cycle access to an instruction or data item and is not subject to a cache miss. However, the limitation is the size of the available memory space. For applications where code and data is greater than the L1 memory space, part of L1 can be configured as cache to achieve lower memory access latencies.

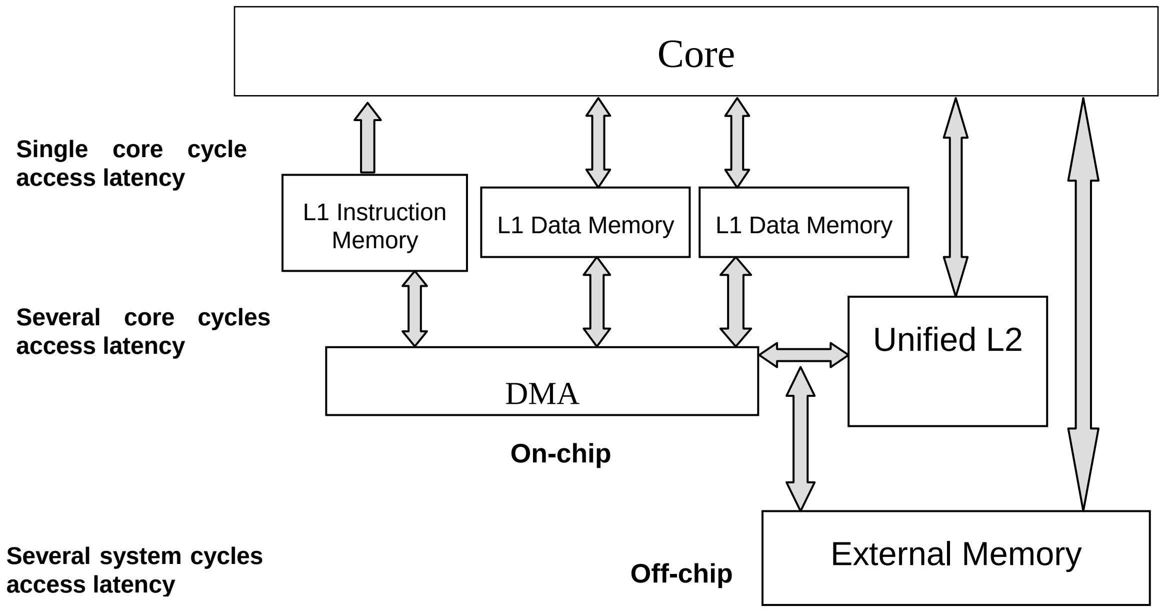
Core
Single core cycle access latency
L1 Instruction Memory
L1 Data Memory
L1 Data Memory
Several core cycles access latency
Unified L2
DMA
On-chip
Several system cycles access latency
Off-chip
External Memory

Figure 1. Blackfin processor memory hierarchy

Cached memory can provide significant benefits for execution of code and data mapped to L2 or external memory. Cache performance depends on the temporal and spatial characteristics of the application. The disadvantage of cache memory is that it suffers from cache miss penalties, which increases memory access latencies, thus increasing external memory bandwidth requirements. Also, for streaming data, cache lines must be invalidated when new data is transferred in external memory. Invalidating cache lines is expensive and can significantly decrease performance.

Guidelines

Taking Advantage of Multi-Ported Sub-Banks

As discussed above, memory contention occurs only if core and DMA accesses fall into the same sub-bank in memory. Figure 2 illustrates a scenario where potential memory contentions can be avoided by efficiently mapping data objects in L1 memory.

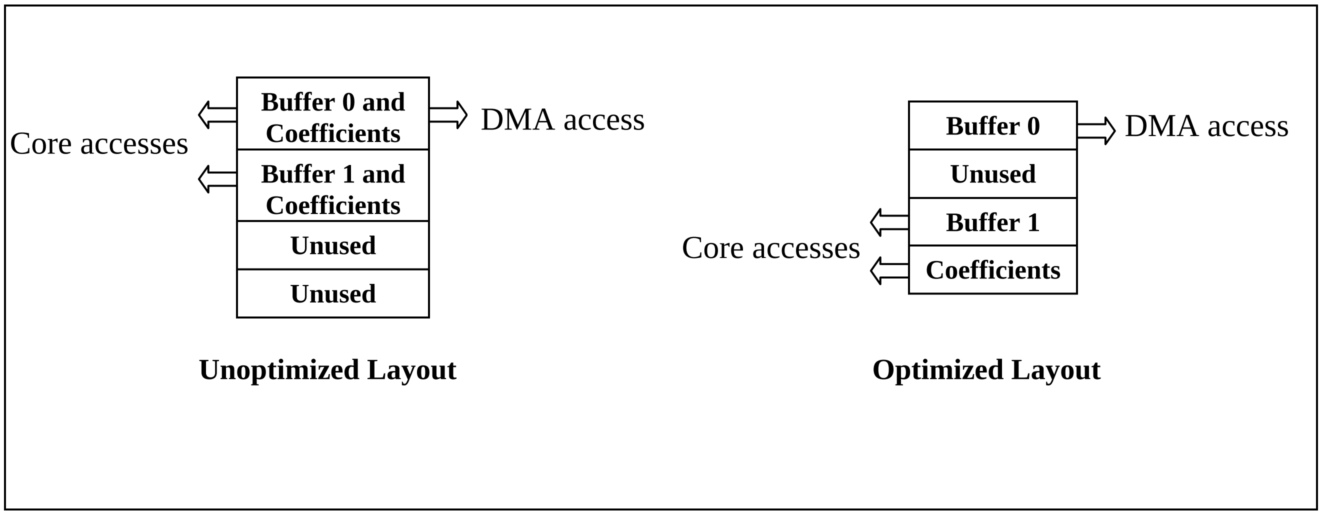

Figure 2. Efficient use of multi-ported sub-bank internal memory architecture

Using L1 SRAM Only

Fitting code and data in L1 SRAM provides the lowest memory access latency. If the total required memory for code and data is more than that available in L1 SRAM, map only the most frequently executed code and most frequently accessed data to L1 memory. For mapping code efficiently in L1 SRAM, use the automated PGO linker tool, described in *PGO Linker - A Code Layout Tool for the Blackfin Processors (EE-306)*[8]. Data layout should still be handled by the programmer, although some techniques are suggested in the next few sections.

L1 SRAM memory can also be used to map code and data items with real-time criticality, as SRAM memory provides guaranteed single-cycle access for all requests. Cache memory suffers from cache miss penalties; therefore, there is no deterministic access time to a requested code or data item. This can sometimes prove critical in meeting the real-time requirements of a system.

Using L1 SRAM and Cache

If the application code size is greater than L1 instruction SRAM space, using instruction cache will improve the average memory access latency. In the VisualDSP++ 4.5 tools, cache can be turned on using the `Project Options` dialog box or by appropriately defining the `_cplb_ctrl` constant in a project source file.

Refer to *Using Cache Memory on Blackfin Processors (EE-271)*[9] for details regarding the use of cache memory on Blackfin processors. An example project within the VisualDSP++ installation folder

(`VisualDSP++ 4.5/Blackfin/Examples/ADSP-BF533 EZ-Kit Lite/Cache/`) also provides implementation details for enabling cache on Blackfin processors.

Blackfin processors also provide alternate ways to move code and data more efficiently within the memory hierarchy. A DMA engine can be used to manage code and data instead of using the cache mechanism. By using DMA, code and data can be brought in ahead of time, consequently avoiding cache miss penalties and also saving any cycles lost due to cache line invalidation. DMA can also do memory transfers in the background while the core is processing, saving valuable core cycles. However, using DMA increases software development time and can be difficult for applications with irregular or random code and data access patterns.

Using DMA to Avoid Cache Miss Penalties

To use DMA effectively, it is important to know program behavior and data access patterns. Managing code and data is entirely the programmer's responsibility; therefore, software development time is expected to increase. To understand the feasibility of using DMA for managing data objects and obtain a more detailed explanation of the advantages over using cache, refer to *Video Templates for Developing Multimedia Applications on Blackfin Processors (EE-301)*[7]. For further details regarding the tradeoffs involved with cache vs. DMA, refer to *The best way to move multimedia data*[10].

L2 SRAM Description

L2 memory on Blackfin processors can only be used as SRAM memory. L2 access times are longer than L1 but provide better performance than accessing off-chip SDRAM. Note that L2 memory is available only on the ADSP-BF561 and ADSP-BF54x Blackfin processors. Table 1 shows L2 memory performance in terms of core clock cycles (`CCLK`) and/or system clock cycles (`SCLK`).

Access Type	**Number of Cycles**	**Comments**
Direct core access instruction fetch (64-bit)	9 CCLKs	
Direct core access data fetch	9 CCLKs (1st 32-bit Fetch) 2 CCLKs (2nd 32-bit Fetch)	Can be 8-, 16-, or 32-bit access
Cache line fill request Instruction and data (32-byte)	15 CCLKs	First 8 bytes available after 9 CCLKs, and 2 CCLKs for each successive 8 bytes
L2 to L1 memory DMA transfers (8-byte)	2 CCLKs	
L2 system read (e.g., L2 to external memory transfer)	1 SCLK + 2 CCLKs	
L2 system write (e.g., external memory to L2 transfer)	1 SCLK	

Table 1. L2 memory performance

Guidelines

L2 SRAM can be used to map code and data that does not fit into L1 SRAM memory. If using the ADSP-BF561 dual-core processor, L2 memory space can be used to map shared data objects between two cores.

L2 memory is also designed as multi-ported sub-banks. This enables simultaneous core and DMA accesses to L2 memory, provided that the accesses are to separate sub-banks.

External Memory Description

For ADSP-BF561 Blackfin processors, the SDRAM controller (SDC) supports 16- or 32-bit SDRAM memory bank widths; for ADSP-BF53x and ADSP-BF52x processors, the SDC supports a 16-bit SDRAM memory bank width. The SDRAM memory can transfer data at a maximum SCLK frequency of 133 MHz. The SDC provides a glueless interface to standard SDRAM memory devices with four internal memory banks and allows interleaved bank memory accesses. Arbitration logic is required for simultaneous core and DMA access to external memory. In this section, the SDRAM memory banking and its performance is discussed in detail.

SDRAM Banking

Standard SDRAM memory devices have internal memory banks. To take advantage of the internal banks of SDRAM, the SDC supports interleaved memory bank accesses. The SDC interface supports up to four internal memory banks.

The SDRAM internal bank addresses consist of row addresses. The internal banks are divided into a set of memory pages, which are configured with the help of the SDC control registers. The number of pages is determined by the SDRAM configuration settings and the internal memory bank size. The SDC can only have one open page at a time in each of the internal banks. Opening a closed page in the internal memory bank requires pre-charge and activation commands.

An access to an open page in memory is termed an *on-page* access. An on-page access does not require pre-charge or activation commands. If an access is to a closed page, it is termed an *off-page* access, which incurs an additional latency of the cycles required for the pre-charge and activation commands. To increase SDRAM performance, it is essential to minimize off-page accesses.

SDRAM Performance

Table 2 gives the SDRAM performance for on-page accesses.

Access Type	**Number of SCLK Cycles for a Word* Access**
Instruction/data cache line fill request	1.1
Direct core instruction fetch	1.1
DAG read access	8
DAG write access	1
MemDMA write access. (e.g., L1 to SDRAM memory transfer)	1
MemDMA read access (e.g., SDRAM to L1 memory transfer)	1.1

Table 2. External memory performance for on-page word accesses (32-bit for ADSP-BF56x processors, 16-bit for ADSP-BF53x and ADSP-BF52x processors.)*

Table 3 shows the performance of external memory accesses for off-page accesses.

Access Type	Number of SCLK Cycles for a Word* Access
Read	$t_{RP} + t_{RCD} + CL$
Write	$t_{WR} + t_{RP} + t_{RCD}$

Table 3. External memory performance for off-page word accesses (32-bit for ADSP-BF56x processors, 16-bit for ADSP-BF53x and ADSP-BF52x processors.)*

In Table 3:

t_{RP} – Delay between a pre-charge and an activation command (1-7 SCLK cycles)

t_{RCD} – Delay between an activation and a first read/write command (1-7 SCLK cycles)

t_{WR} – Delay between a write and a pre-charge command (1-2 SCLK cycles)

CL (CAS latency) – Delay between a read command and availability of data off-chip (2-3 SCLK cycles)

Guidelines

Minimizing Off-Page Accesses

Off-page access latency is higher when code and data with high spatial locality are mapped across page boundaries of an internal memory bank. To minimize off-page accesses, map blocks of highly spatial code and data to the same page of an internal memory bank. To further take advantage of the fact that up to four memory pages can be open in the SDRAM memory space at a time, it is best to create spatial blocks of code and data four times the size of a page and map them across four different pages in each internal bank.

If the spatial characteristics of the application are hard to determine, an application developer can evaluate the following approaches relative to the trade-offs specific to an application

1. Measuring and utilizing the spatial characteristics for a mix of instructions and data is even more difficult. To ease the programming effort, in certain cases L2 and external memory can be used as separate code and data memory space. For example, if the application code (or the most frequently accessed code) can be mapped entirely in L1 or L2 memory, the entire external memory is available for mapping data objects. Mapping data objects only to external memory provides greater control and predictability for accesses to external memory. However, note that the tradeoff here is the increased data access latency. Figure 3 shows an example memory mapping.

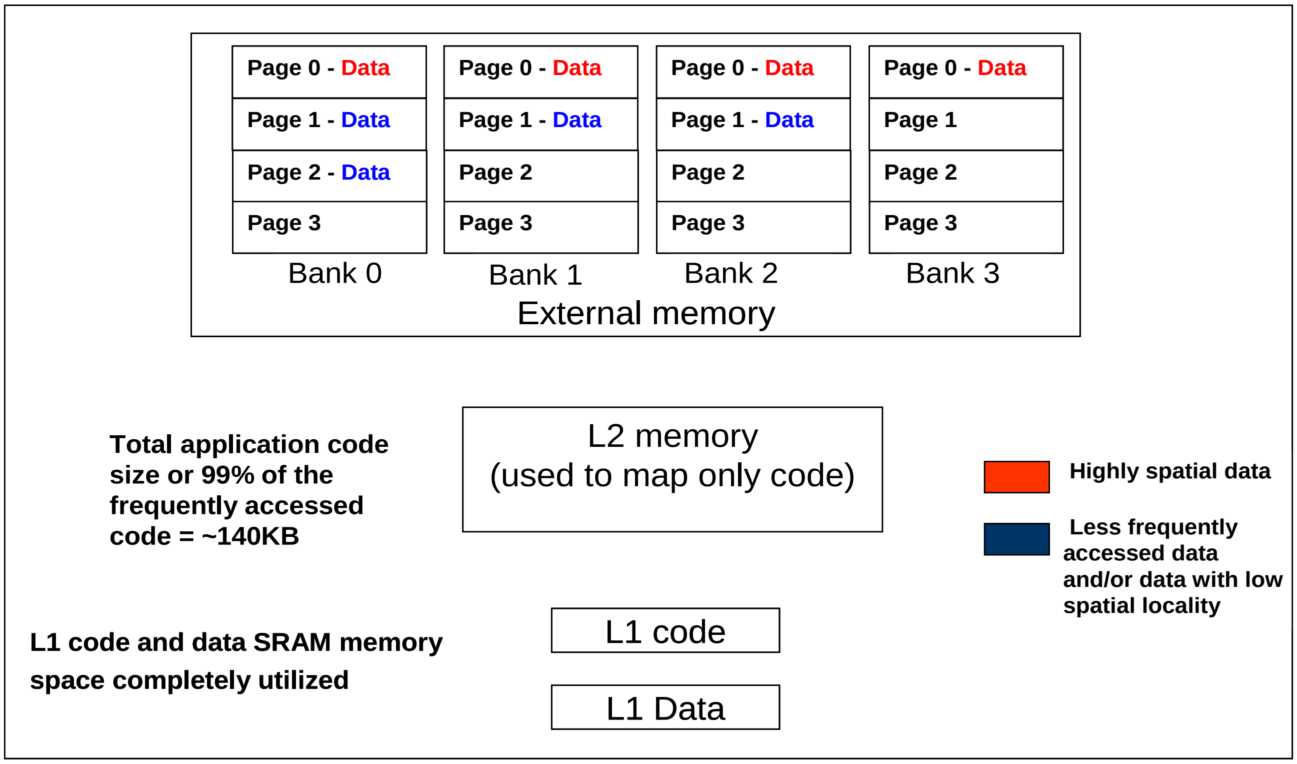
Page 0 - Data
Page 1 - Data
Page 2 - Data
Page 3
Bank 0
Page 0 - Data
Page 1 - Data
Page 2
Page 3
Bank 1
Page 0 - Data
Page 1 - Data
Page 2
Page 3
Bank 2
Page 0 - Data
Page 1
Page 2
Page 3
Bank 3
External memory
Total application code size or 99% of the frequently accessed code = ~140KB
L2 memory (used to map only code)
Highly spatial data
Less frequently accessed data and/or data with low spatial locality
L1 code and data SRAM memory space completely utilized
L1 code
L1 Data

Figure 3. Using external memory for data only

2. External memory can be divided such that code/data is mapped to separate internal memory banks. This is helpful if the spilled code from L1 and L2 memory can be mapped such that highly spatial code does not cross internal bank page boundaries. Figure 4 shows an example memory map.

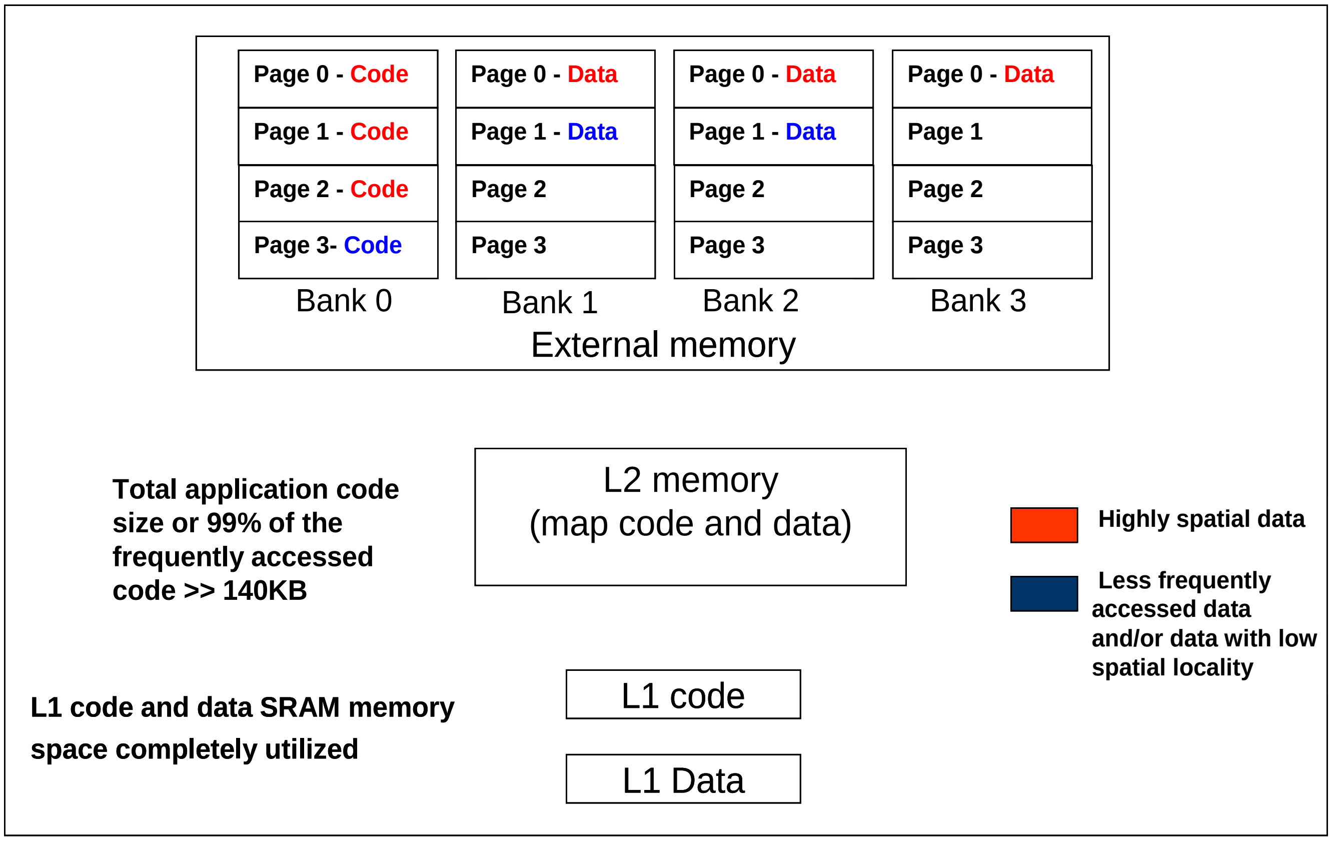

Figure 4. Placing code in only one SDRAM internal memory bank page

3. Multiple buffers are typically maintained for peripheral data. These buffers can be mapped to separate internal banks. Figure 5 shows an example memory map. Also refer to *Video Templates for Developing Multimedia Applications on Blackfin Processors (EE-301)*[7] for alternate ways to manage multiple data buffers.

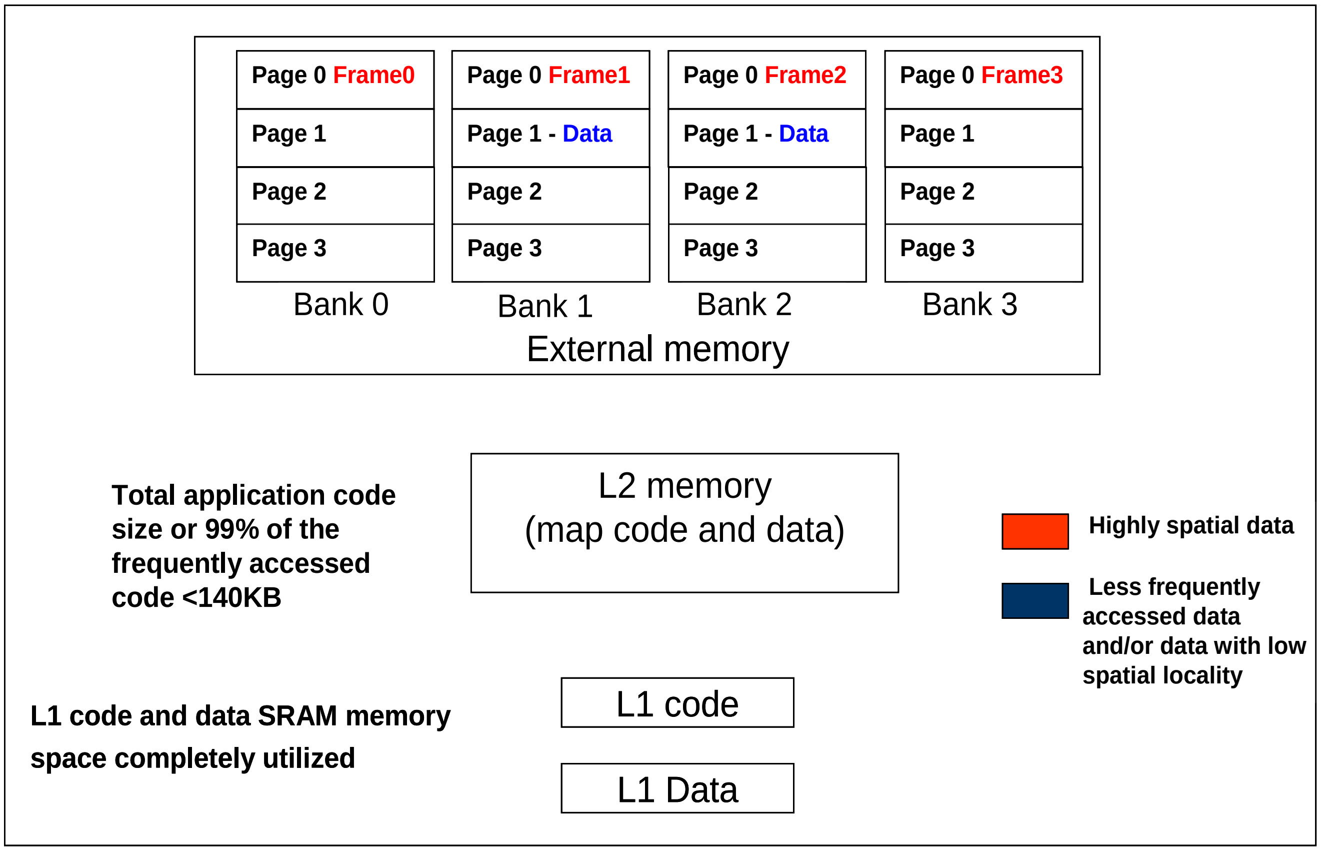
Page 0 Frame0
Page 1
Page 2
Page 3
Bank 0
Page 0 Frame1
Page 1 - Data
Page 2
Page 3
Bank 1
Page 0 Frame2
Page 1 - Data
Page 2
Page 3
Bank 2
Page 0 Frame3
Page 1
Page 2
Page 3
Bank 3
External memory
Total application code size or 99% of the frequently accessed code <140KB
L2 memory (map code and data)
Highly spatial data
Less frequently accessed data and/or data with low spatial locality
L1 code and data SRAM memory space completely utilized
L1 code
L1 Data

Figure 5. Placing frame buffers in four different SDRAM internal banks

Other Suggestions

Typically, frequently accessed code and data objects have high spatial locality. Thus, one can form sections of frequently accessed code and data equal to four times the size of a page in an internal bank and map them across four separate pages in each of the internal banks.

Another possibility is to try to avoid direct core accesses to external memory. A DAG access read, as noted in Table 2, takes eight `SCLK` cycles, which can result in significant bandwidth consumption for even small amounts of externally mapped data accesses. Thus, use cache or DMA to access data objects mapped to external memory.

System Architecture

This section discusses the Blackfin processor's system architecture, which includes the system buses, DMA controllers, peripherals, and external bus arbiter, and provides guidelines for optimizing their usage. Note that the discussion is only from a performance standpoint and ignores other details.

System Buses

The following system buses are connected to the DMA channels, peripherals, and core.

Peripheral Access Bus (PAB)

The PAB bus connects all off-core peripherals with the system MMR registers. Register read and write accesses on the PAB bus have a latency of three and two `SCLK` cycles, respectively.

DMA Buses

For the ADSP-BF561 processor, the following DMA buses connect the peripherals and the memory hierarchy.

- DMA Access Buses (32-bit DAB1 and 16-bit DAB2) provide DMA data transfers to and from the peripherals.
- DMA Core Buses (32-bit DCB1, DCB2, DCB3, and DCB4) provide DMA data transfer to/from the core (A and B) L1 memory, or DMA data transfer between L1 and L2 memory.
- DMA External Bus (DEB) provides DMA data transfer to/from external memory.

The transfer latency between peripherals and memory is two `SCLK` cycles for up to 32 bits per access, although any peripheral can access the bus every `SCLK` cycle to fully utilize the available bandwidth. Transfers between different levels of memory can be achieved at a latency of one `SCLK` cycle for every 32-bit word. For transfers between L1 and L2 memory, 32-bit words can be transferred every `CCLK` cycle. Table 4 lists the performance for a memory-to-memory DMA transfer between different levels of the memory hierarchy.

Source	**Destination**	**Approximate SCLKs for n Words** (from start of DMA to interrupt upon completion)
32-bit SDRAM	L1 Data memory	n+14
L1 Data memory	32-bit SDRAM	n+11
32-bit ASYNC memory	L1 Data memory	xn+12 x is the number of wait states + setup/hold SCLK cycles (minimum x =2)
L1 Data memory	32-bit ASYNC memory	xn+9 x is the number of wait states + setup/hold SCLK cycles (minimum x =2)
32-bit SDRAM	32-bit SDRAM	10+(17n/7)
32-bit ASYNC memory	32-bit ASYNC memory	10+2xn x is the number of wait states + setup/hold SCLK cycles (minimum x =2)
L1 Data memory	L1 Data memory	2n+12

Table 4. Performance of the DMA buses for memory-to-memory transfers

For ADSP-BF53x and ADSP-BF52x processors, the following DMA buses connect the peripherals and the memory hierarchy.

- DMA Access Buses (16-bit DAB) provide DMA data transfer to and from peripherals.
- DMA Core Buses (16-bit DCB) provide DMA data transfer to and from L1 memory, or DMA data transfer between L1 and external memory.
- DMA External Bus (DEB) provides DMA data transfer to/from external memory.

The performance and transfer latency for the ADSP-BF53x processors remains the same as for the ADSP-BF561 processor, but the maximum word size is 16 bits for every `SCLK` cycle.

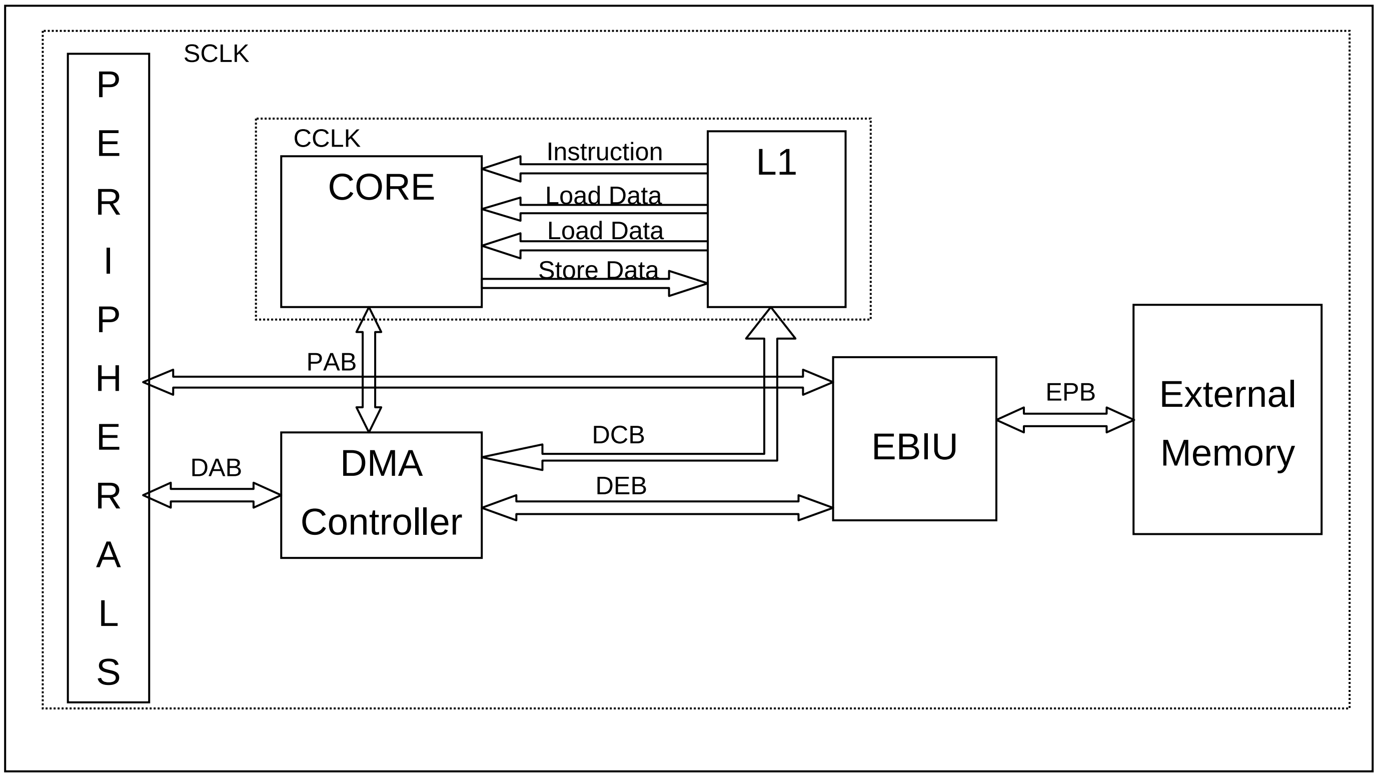

Figure 6. Internal bus structure for ADSP-BF53x/ADSP-BF52x Blackfin processors. For the ADSP-BF561 processor, there are two cores, two DMA controllers, and a block of L2 memory.

External Access Bus (EAB)

The EAB allows the core to access off-chip memory directly. For ADSP-BF561 dual-core processors, transfers of 8-, 16-, or 32-bit words can be performed each `SCLK` cycle. For ADSP-BF53x processors, transfers of 8- or 16-bit words can be performed each `SCLK` cycle. The EAB runs at a maximum frequency of 133 MHz.

Guidelines

Utilizing Maximum Bus Width and Packing on Peripherals

The system throughput can be greatly increased by using the maximum bus width for every transfer. Using 32-bit DMA access for ADSP-BF561 processors and 16-bit DMA access for ADSP-BF53x/ADSP-

BF52x processors combined with packing (if available on the peripheral interface) can free up the system buses for other activities, thereby greatly increasing the throughput of the system. For example, the PPI provides 32-bit packing for ADSP-BF561 processors and 16-bit packing for ADSP-BF53x/ADSP-BF52x processors.

DMA Traffic Control

Blackfin processors provide traffic control on all the system buses. If the traffic on the bus is switching directions too often, the result will be increased latencies due to bank turnaround times. Using the traffic control registers is one of the best ways to optimize the system bus traffic, consequently improving bandwidth utilization. The traffic period for each of the DMA buses can be specified to group transfers in one direction, thereby minimizing bank turnaround times. Figure 7 illustrates an optimized traffic pattern over the DAB bus.

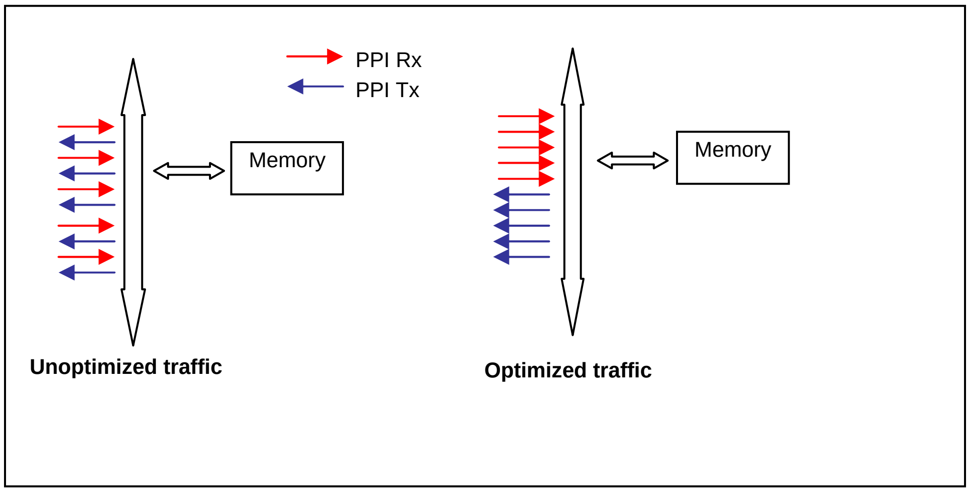

Figure 7. Optimizing DMA traffic over the system buses

Figure 8 shows the DMA traffic control register. Each of the buses can be programmed to group up to 16 transfers in one direction. Also, the memory DMA channels can be programmed for round-robin priorities using the traffic control register.

The values for the traffic period should be evaluated on a per-application basis, depending on the number of peripherals and the amount of bus traffic within the system. As a rule of thumb, a traffic period value close to three is optimal when four or more peripherals are on a DMA controller, and a value close to seven is optimal in cases where three peripherals or less are used on a DMA controller. Also, note that the traffic period value for the DEB has the most significant impact on system performance; therefore, it should be evaluated more thoroughly.

Also, a smaller value for memory DMA round-robin period will lower the throughput of DMA transfers, as the controller is switching between DMA channels every few words. In general, for most applications, a higher value balances the performance with equal sharing of resources.

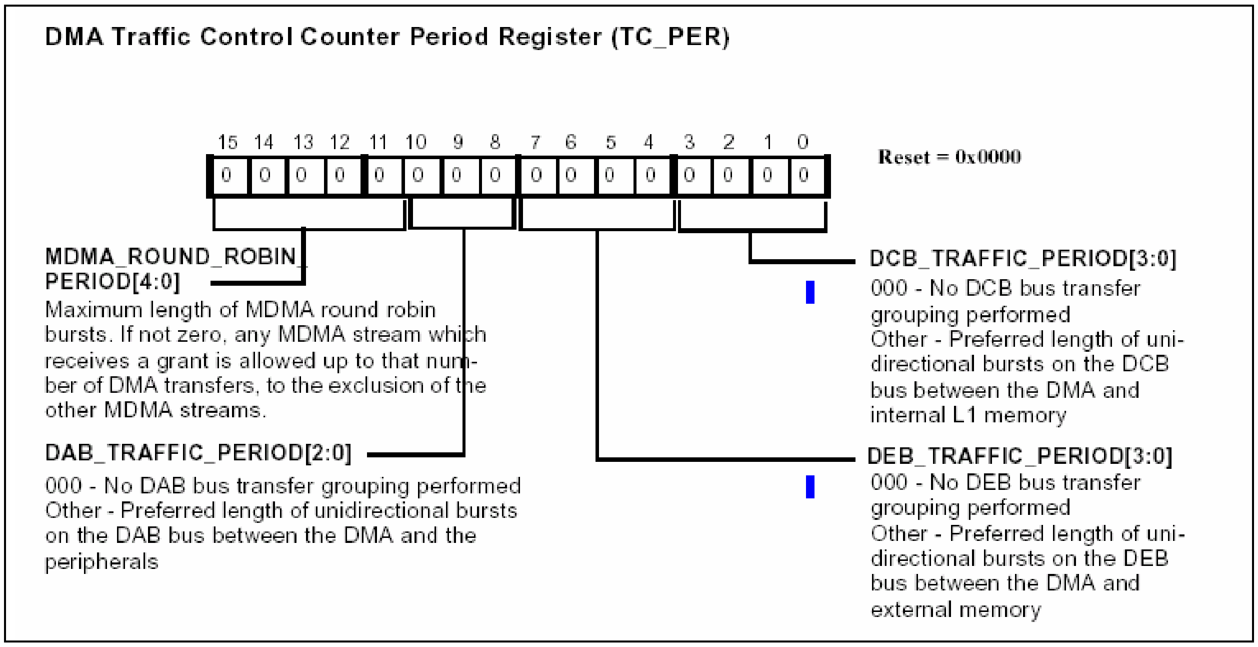
DMA Traffic Control Counter Period Register (TC_PER)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Reset = 0x0000
MDMA_ROUND_ROBIN_PERIOD[4:0]
Maximum length of MDMA round robin bursts. If not zero, any MDMA stream which receives a grant is allowed up to that number of DMA transfers, to the exclusion of the other MDMA streams.
DAB_TRAFFIC_PERIOD[2:0]
000 - No DAB bus transfer grouping performed
Other - Preferred length of unidirectional bursts on the DAB bus between the DMA and the peripherals
DCB_TRAFFIC_PERIOD[3:0]
000 - No DCB bus transfer grouping performed
Other - Preferred length of unidirectional bursts on the DCB bus between the DMA and internal L1 memory
DEB_TRAFFIC_PERIOD[3:0]
000 - No DEB bus transfer grouping performed
Other - Preferred length of unidirectional bursts on the DEB bus between the DMA and external memory

Figure 8. DMA traffic control register

CCLK/SCLK Ratio

The performance of the buses is affected by core-to-system clock frequency ratios. At ratios below about 2.5:1, synchronization and pipeline latencies result in lower bus utilization in the system clock domain. At a clock ratio of 2:1, for example, DMA typically runs at 2/3 of the system clock rate. Full bandwidth can be utilized by implementing higher core-to-system clock ratios.

DMA Architecture

The DMA controller manages DMA activity between the peripheral interface and the on-chip and off-chip memories. Multiple peripheral DMA channels and memory DMA channels compete for memory resources. The access latency is improved by separating the buses on the peripheral DMA channel (DAB bus) and the memory-to-core accesses (DCB bus). Also, each peripheral maintains its own FIFO buffer, which further reduces the latency as compared to a fully pipelined peripheral-to-memory DMA transfer.

A DMA transfer can be initiated in several modes: stop, auto-buffer, descriptor, etc. Refer to the DMA chapter in the processor's *Hardware Reference* manual for more information. In descriptor mode, the DMA controller fetches the contents to be loaded into the DMA channel registers from a descriptor object located in data memory space. A descriptor or descriptor array can hide delays due to MMR writes to the DMA registers.

ADSP-BF561 processors have two DMA controllers, whereas ADSP-BF53x processors have only one. For processors with more than one DMA controller, each DMA controller is connected to two separate buses; therefore, transfers can be spread across DMA controllers to improve throughput.

Guidelines

1. Prioritizing peripherals with high data rates improves throughput. Change the default assignment of the peripherals assigned to the DMA controllers using the DMA peripheral map register.

2. If more than one DMA controller is available, spreading transfers across DMA controllers may reduce bandwidth requirements of the system.
3. Using auto-buffer mode or descriptor mode can save system clock cycles.

Bus Arbiter

The EBIU arbitrates between core and DMA accesses to external memory. Before the arbitration policy for accessing external memory is addressed, it is important to note when a DMA request is termed *urgent* in the system.

Urgent DMA

Consider the following scenario where a peripheral channel is receiving data from an external device. On a peripheral receive (memory write), if the peripheral FIFO is full and DMA cannot access the bus to write the FIFO contents to memory, a buffer overflow condition is likely. In this situation, an *urgent DMA request* will be issued.

Similarly, to protect against an underflow condition, an urgent DMA request will be issued on a peripheral transmit (memory read) if the peripheral FIFO is empty and DMA cannot get access to the bus to read from memory to fill the FIFO.

For both situations above, an urgent DMA request is issued automatically by the DMA controller, thus no user intervention is required. On the ADSP-BF534/BF536/BF537 processors, if a DMA request becomes urgent, all pending DMA requests in the system are marked urgent. For the ADSP-BF531/BF532/BF533 and ADSP-BF561 processors, pending requests are not turned urgent when an urgent request is raised. To turn all pending requests urgent on these processors, the `CDPRIO` bit in the `EBIU_AMGCTL` register can be set, which is described in more detail in the next section.

Arbitration Scheme

For external memory accesses, urgent DMA requests have the highest priority by default, followed by core requests and then by DMA requests. This priority scheme is programmable for external memory accesses as will be discussed in the guidelines section.

For L1 and L2 memory, DMA and core requests follow a fixed arbitration. For internal memory accesses, arbitration is required only if the DMA and core accesses are to the same sub-bank of memory. If accessing the same sub-bank, DMA wins over core by default. For locked core accesses, DMA has to wait until the locked accesses are complete.

Guidelines

For external memory access, core requests have higher priority than DMA requests. This can severely impact throughput if there are several core accesses to external memory. This can be avoided by elevating all DMA accesses to urgent DMA requests, thus giving all DMA accesses higher priority than core accesses. This can be done setting the `CDPRIO` bit or the `DMAPRIO` bit in the `EBIU_AMGCTL` register.

Other General Guidelines

Theoretical estimation of the total bandwidth required by the application can provide a good starting point to study the feasibility of the system from a bandwidth perspective. In cases where system resources are inefficiently utilized, one may encounter buffer underflow or overflow, even if less than 20% of the total maximum theoretical bandwidth is utilized.

Often, buffer underflow/overflow errors go unnoticed during debugging efforts. By using the error interrupts supported by Blackfin processors, one can determine if the system is running out of bandwidth. The peripheral status registers indicate overflow or underflow errors. Note that the error interrupts are not turned on by default. On all Blackfin processors, error interrupts are mapped to IVG7 by default. It is the user's responsibility to determine which peripheral is causing the error within the interrupt service routine.

Error interrupts can be enabled like any other interrupt vector in an application. The error handler should specifically check the status registers of all the peripherals within the system to determine the exact cause for the error.

System Optimization Techniques

To summarize, the previously discussed techniques are listed below:

1. Code and data layout techniques
2. Packing and using the full bus width
3. Efficient bank placement
4. Optimizing bus traffic using the DMA traffic control
5. Maintaining higher CCLK/SCLK ratios
6. Spreading data transfers across DMA controllers, when applicable
7. Using programmable bus priority schemes

These techniques are evaluated in this section, and each is quantified by considering specific test cases. The techniques are progressively added to a test case, and the throughput of the system is evaluated at each step. Note that code layout and data layouts are described in *PGO Linker - A Code Layout Tool for the Blackfin Processors (EE-306)*[8] and *Video Templates for Developing Multimedia Applications on Blackfin Processors (EE-301)*[7], respectively, and are not included in the analysis in this EE-Note.

Evaluation Methodology

To evaluate system bandwidth, two benchmarks are synthesized. The first benchmark is a simple program that performs only memory DMA reads and writes to external memory; no other activity is enabled on the system bus for this benchmark. The second benchmark is a more complex program with multiple memory DMA channels, peripheral DMA channels, and core activities within the system; this benchmark is used to demonstrate the benefits of the discussed optimization techniques. In the later part of this analysis, the effects of setting the traffic control register and `CCLK`/`SCLK` ratio on the bandwidth utilization is quantified separately.

The goal of the analysis is to show how progressively the system performance can be improved using the various optimization techniques. To quantify the effects, the average throughput was chosen as a metric to evaluate the optimization techniques. The average system throughput was measured as follows:

Average system throughput = (Number of data read or writes to external memory)/sec

The time interval for the system bus activity is configured using the internal core timer. The timer is set to generate an interrupt after one second has elapsed. The timer is started just before the peripherals/MDMA channels are enabled, and then the peripherals/MDMA channels are disabled within the core timer ISR. The amount of data transferred is measured with the respective counters in the interrupt service routines of the peripherals. An interrupt is generated on every buffer transfer, and the counter is incremented every time a peripheral/MDMA ISR is invoked. As all peripherals and MDMA channels are running in auto-buffer mode, peripheral interrupt latency does not need to be accounted for in the final calculations for throughput. Note that the timer interrupt latency is ignored in the calculations.

The benchmarks were evaluated on the ADSP-BF561 processor, but the inferences drawn from the results are applicable to all Blackfin processors unless otherwise stated.

System Bandwidth Analysis

Two benchmarks are discussed in this section: a simple memory DMA example and a realistic embedded application featuring video and audio input/output and file sharing.

Example 1 - Memory DMA

In this benchmark, buffers are transferred between L1 memory and external memory using a single pair of memory DMA channels. The DMA buffer is 8 KB, and the `CCLK` and `SCLK` are set to 600 MHz and 120 MHz, respectively. Table 5 shows the throughput analysis for SDRAM accesses. Note that there is only one memory DMA channel running in the system, and no peripheral or core accesses are made to external memory.

DMA Operation	Packet Size	Buffers Transferred Per Second	Average Throughput (MB/s)	% of Max Theoretical Throughput (of 480 MB/s)
Write access L1 to external memory	8 KB	57303	469	98%
Read access External memory to L1	8 KB	51165	419	87%

Table 5. Throughput for DMA read and write accesses to the external memory

The memory DMA channel is used in auto-buffer mode running uninterrupted until the core timer expires. In this benchmark, none of the system optimization techniques are used. However, as shown in Table 5, the throughput is close to the maximum theoretical bandwidth available. This is because there is no other activity on the external bus except for memory DMA channel 1 read/write accesses. Also, the accesses are to the same page in external memory, which avoids page miss penalties. The read from external memory is affected due to the CAS latency on every read cycle. Refer to Table 2 for external memory latencies for DMA read and write accesses.

Example 2 - Audio/Video with File Sharing

For the second benchmark, a more likely application is analyzed, which includes the peripheral interface for video in/out, audio in/out, and some file sharing through a network or USB interface. This benchmark basically consists of additional interfaces to the established single memory DMA benchmark. The following peripheral assignments are used to interface with the external devices.

External Device	Peripherals Assigned	DMA Controller (DMAC)	Comments
Video encoder	PPI0	DMAC 1	ITU-656 format video input
Video decoder	PPI1	DMAC 1	
Audio in	SPORT0 RX	DMAC 2	
Audio out	SPORT0 TX	DMAC 2	
File write to PC (USB to ASYNC memory transfers)	MDMA1_1	DMAC 1	Write to ASYNC memory
Internal memory DMA transfers	MDMA1_0	DMAC 1	To move data from external memory to L1 memory

Table 6. Peripheral/DMA assignment for external devices interfaced to the system

To evaluate the optimization techniques and suggested guidelines, the following four application scenarios are considered:

Scenario 1: The scenario is set up as shown in Figure 9. This is the baseline scenario model, where none of the system optimization techniques have been used.

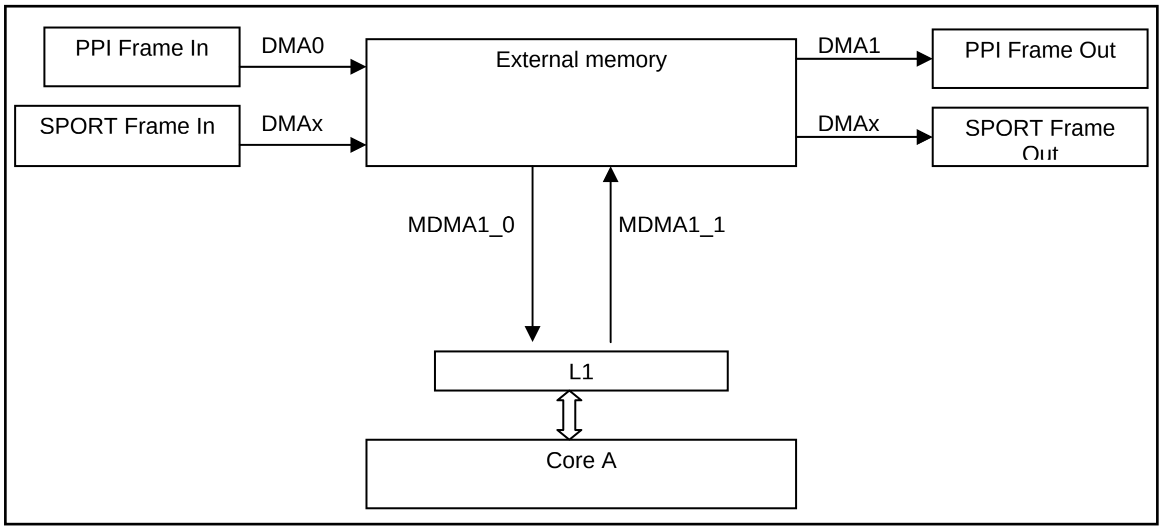

Figure 9: Example scenario 1 (L2 memory not shown)

Scenario 2: Memory DMA transfers (MDMA1_0 MDMA1_1) are moved to DMA controller 2 (MDMA2_0 MDMA2_1).

Scenario 3: Both DMA controllers are used for memory DMA transfers.

Scenario 4: A tight core access loop is added to external memory.

Note that MDMA channels can transfer 32-bit words to external memory each `SCLK` cycle, which results in maximum bus utilization. The MDMA channels run uninterrupted in auto-buffer mode, thereby utilizing any free bandwidth between peripheral DMA channel transfers.

Peripheral	**Packet Size**	**Packets Transferred Per Second**	**MB/s**
PPI – In (30 MHz-NTSC video in)	1716 * 525 = 900900	16	14
PPI – Out (30 MHz - NTSC video out)	1716 * 525 = 900900	9	8
MDMA	8192	14	1
SPORT_RX/TX (4 MHz)	32	14649	0.1
TOTAL			23

Table 7: Throughput calculation for baseline scenario 1

Optimization Technique	**Scenario 1**		**Scenario 2** **(Spreading to 2 DMA Controllers)**		**Scenario 3** **(Using Both DMA Controllers)**	
	MB/s	**Improvement**	**MB/s**	**Improvement**	**MB/s**	**Improvement**
Baseline	23	1x	23	1x	23	1x
Bus width and PPI packing	54	2.3x	148	6.4x	148	6.4x
Efficient bank placement	96	4.2x	348	15.2x	351	15.3x
DMA traffic control	230	10.0x	330	15.0x	342	15.5x

Table 8. Throughput improvements with various optimization techniques for three different application scenarios.

As can be seen from Table 8, using the full bus width and enabling packing can improve the performance by up to 2.3x for the first scenario and 6.4x in the second and third scenarios. Efficient bank placement can give an additional 8x (15.2x - 6.4x) improvement. By spreading the data transfers across two DMA controllers, the throughput is increased by a factor of five times the baseline throughput. DMA traffic control improves performance when traffic is scheduled on one DMA controller (demonstrated in the first scenario). By using two DMA controllers, the bandwidth utilization is maximized with only bus width packing and efficient bank placement; therefore, adding traffic control does not increase the throughput as much (scenarios 2 and 3).

A 4th scenario is also considered, which adds a tight core access to external memory, as shown in Table 9.

Optimization	**Scenario 1**		**Scenario 4**	
	MB/s	**Improvement**	**MB/s**	**Improvement**
Baseline	23	1x	21	1x
Bus width and PPI packing	54	2.3x	52	2.5x
Efficient bank placement	96	4.2x	54	2.6x
DMA traffic control	230	10.0x	55	2.6x
Set CDPRIO	230	10.0x	195	9.3x

Table 9: Throughput improvements by setting the CDPRIO bit when the core accesses external memory

As can be seen from Table 9, a tight core access can virtually lock the system bus to not give any access to the DMA channels. None of the techniques would be effective if the DMA does not get access to the bus. By setting the `CDPRIO` bit, the throughput increases.

Evaluating the Traffic Control Register

As shown above, using DMA traffic control can greatly improve system throughput, but larger traffic period values can make other peripherals starve for data for longer periods of time. For example, consider when the PPI0 and PPI1 interfaces are used in an application, with PPI1 in receive mode and PPI1 in transmit mode. If the DEB traffic period is programmed to 16, either of the PPIs might have to wait for 16 transfers before it can gain access to the bus again. This may cause the FIFO of the waiting PPI to underflow or overflow.

To demonstrate this, the test case in the associated ZIP file[11] was used. In the project, two PPI channels and two MDMA channels on DMA controller 1 are used. In the test case, when traffic control is off, the PPI1 status register shows an underflow error. By setting the traffic control to 0x0777, the underflow error on PPI1 is eliminated because of the reduced bank turnaround times. In order to increase the throughput further, setting a more aggressive value of 0x07ff will again cause an underflow error on PPI1. This is because holding the PPI1 for a longer time causes its FIFO to starve for data, eventually resulting in an underflow.

As discussed in the previous section, a general rule of thumb is to use a traffic period value of 0x7 for less than three peripherals on a DMA controller and a traffic period of 0x3 when four or more peripherals are running on a DMA controller. Also of interest is the fact that the DEB traffic period is more significant to the throughput performance than the DCB or the DAB bus traffic period, as the bank turnaround times on the external memory have higher penalties than turnarounds on L1 SRAM memories.

The five most significant bits of the traffic control register control the MDMA round-robin period. Round-robin allows equal sharing among the MDMA channels, such that a higher priority MDMA channel does not block a lower priority channel. However, introducing round-robin lowers the DMA performance due to the additional latency in switching between the channels. Figure 10 shows the throughput performance versus the MDMA round-robin count for two MDMA channels running uninterrupted on DMA controller 1. The throughput is measured in the same way as the previous experiments.

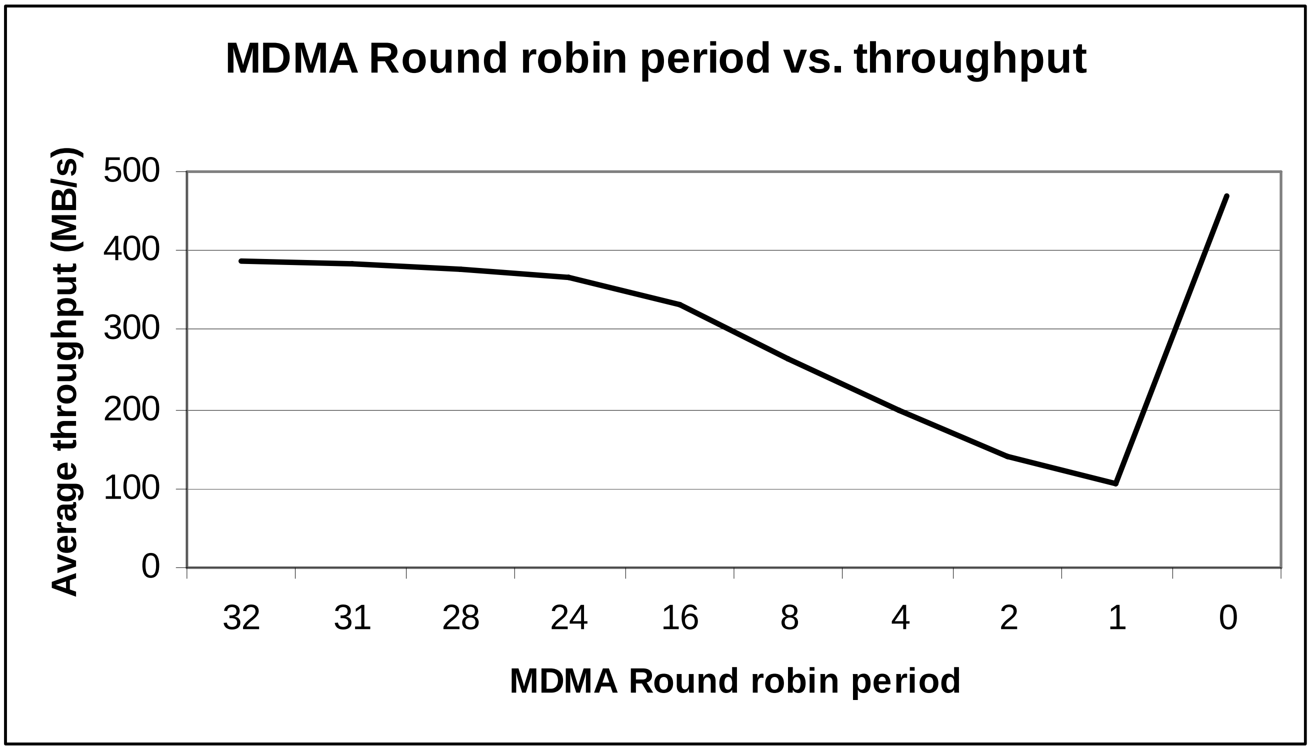

Figure 10. MDMA round-robin period vs. throughput

As can be seen, as the round-robin count is decreased, the achievable throughput of the system subsequently decreases. Programming the round-robin period to zero results in maximum system bus utilization; however, in this case, the higher priority channel (MDMA1) will not allow the lower priority channel (MDMA2) to gain access to the bus until its current transaction is complete.

Evaluating CCLK/SCLK Ratio

The `CCLK` to `SCLK` ratio also affects the system throughput because, at lower core clocks, synchronization and increased latencies of the core buses result in reduced system bus utilization. Table 10 shows the effects of the `CCLK/SCLK` ratio on the system throughput. The test case used is similar to the single MDMA channel to transfer an 8-KB DMA buffer between L1 and external memory, and the experiment is repeated for different `CCLK/SCLK` ratios for a read and a write access to external memory.

DMA Access	CCLK/SCLK Ratio	Transfers Per Second	Average Throughput (Transfers/sec * buffer size)	% of Theoretical Throughput (of 480 MB/s)
Write	5	57303	469	97.71
	4	46265	379	78.96
	3	46265	379	78.96
	2	33301	272	56.67
	1	23378	191	39.79
Read	5	51165	419	87.29
	4	51165	419	87.29
	3	51165	419	87.29
	2	38757	317	66.04
	1	29196	239	49.79

Table 10: Effects of CCLK/SCLK ratio on system throughput

As can be seen, the performance drops considerably below the 2:1 CCLK/SCLK ratio. The read is less sensitive, as the core bus latencies are hidden due to the additional CAS latency involved with a read access to external memory. The reduced bus utilization is due to DCB latencies, but this would be hidden if there were additional peripheral activities on the system bus, such that the DCB was not the bottleneck.

Conclusion

Blackfin processors provide several optimization techniques that can help to fully utilize the available bandwidth over the EBIU. The memory and system optimization techniques discussed in this application note will help produce efficient code/data layouts and optimize system performance.

References

[1] *ADSP-BF533 Blackfin Processor Hardware Reference*. Rev 3.2, July 2006. Analog Devices, Inc.

[2] *ADSP-BF561 Blackfin Processor Hardware Reference*. Rev 1.0, July 2005. Analog Devices, Inc.

[3] *ADSP-BF537 Blackfin Processor Hardware Reference*. Rev 2.0, December 2005. Analog Devices, Inc.

[4] *Embedded Media Processing*. David Katz and Rick Gentile. Newnes Publishers., Burlington, MA, USA, 2005.

[5] *Video Framework Considerations for Image Processing on Blackfin Processors (EE-276)*. Rev 1, September 2005. Analog Devices Inc.

[6] *VisualDSP++ 4.5 Device Drivers and System Services Manual for Blackfin Processors*. Rev 2.0, March 2006. Analog Devices, Inc.

[7] *Video Templates for Developing Multimedia Applications on Blackfin Processors (EE-301)*. Rev 1, September 2006. Analog Devices Inc.

[8] *PGO Linker - A Code Layout Tool for the Blackfin Processors (EE-306)*. Rev 1, December 2006. Analog Devices Inc.

[9] *Using Cache Memory on Blackfin Processors (EE-271*). Rev 1, June 2005. Analog Devices Inc.

[10] *The Best way to move multimedia data.* David Katz and Rick Gentile. http://www.embedded.com/showArticle.jhtml?articleID=16700107. December 2003. Embedded.com.

[11] Associated ZIP File. Rev 1, May 2007. Analog Devices, Inc.

Document History

Revision	**Description**
Rev 1 – July 10, 2007 *by Kaushal Sanghai*	Initial Release