Engineer-to-Engineer Note EE-301

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.

Video Templates for Developing Multimedia Applications on Blackfin® Processors

Contributed by Kaushal Sanghai

Rev 1 – September 28, 2006

Introduction

Blackfin® processors enable a variety of ways to efficiently manage data for high-performance applications. This document describes a set of "templates" that can be used to efficiently manage multimedia data on Blackfin processors.

The templates provided with this EE-Note can help users get started in implementing customized applications, as they can be modified to fit specific application needs. The first part of the EE-Note discusses the types of templates available, and the second part describes specific examples for each template.

Background

The size of data buffers involved in multimedia applications exceeds the processor's internal memory space. To take advantage of the low-latency access of the processor's on-chip memory (L1/L2), some example templates have been created that can be used as a starting point to move peripheral data directly into L1 or L2 memory. The tradeoff to consider is between the size of the memory and the processing granularity of the image block within an application. The templates also exploit the predictable data access pattern inherent in multimedia applications to minimize the transfers required to move data between different levels of memory, thus improving resource utilization.

Video Templates Overview

A video sequence can be regarded as a three-dimensional (3-D) signal, comprising two dimensions in the spatial domain (i.e., a single 2-D frame) and one dimension in the temporal domain (i.e., a temporal sequence of 2-D frames). The data flow of the video stream can thus be partitioned in one of the two dimensions: 1) a temporal dimension, or 2) a spatial dimension. In the temporal dimension, the data flow can be managed at the granularity of a group of pictures (GOP) level or at the frame level. In the spatial dimension, the data movement can be managed at the granularity of a line or a macro-block. These various levels of granularity in a video signal are called *sub-processing blocks*. Note that the granularity of the sub-processing decreases from GOP level to line level.

The templates discussed in this document exploit the temporal and spatial characteristics inherent in video signals described above. The templates take advantage of the predictable data access pattern to hide the memory latencies and reduce the number of memory transfers between different levels of memory.

In most image processing algorithms, the data access pattern of the sub-processing block is predictable. For example, in JPEG compression, the macro-blocks of an image are accessed sequentially across the rows. Table 1 shows examples in which the access pattern of the sub-processing block is predictable.

Example Application	Comments
Color Conversion (e.g., YUV to RGB)	The image is accessed line by line
Histogram Analysis	Image data is accessed line by line
Edge Detection	Macro-blocks are accessed sequentially row wise
JPEG/Motion JPEG	Macro-blocks are accessed sequentially row wise
Motion Detection	Macro-blocks are accessed sequentially row wise; dependent on past frame
MPEG2/MPEG4	Motion window sequentially across macro-blocks; dependent on past and future frames

Table 1. Predictable Access Patterns

If the data access pattern of the image is known in advance, the data can be transferred to L1 memory before it is requested by the core. This avoids cycles consumed due to the core being held off for a memory request. Also, to hide the latency of the memory transfer, DMA can be effectively used in the background for any data transfer. Using DMA instead of cache helps to save core cycles consumed by cache misses.

To further save system resources, the templates utilize the smaller and faster L1/L2 memory spaces by storing image data from the peripheral into on-chip memory, thus eliminating the delays that would be incurred during accesses to external SDRAM. This applies to algorithms that operate on finer sub-processing block granularities (i.e., a line or a macro-block).

The templates are designed such that once the application's *sub-processing block* is identified and the real time constrains are met, the image processing algorithm can be dropped into one of the templates to obtain an efficient data layout. This is demonstrated by taking specific examples for each of the templates.

In certain algorithms, data dependency can exist between the sub-processing blocks in the temporal domain (i.e., current frame of reference and past or future frames). In these cases, managing data is more complicated. One such scenario is discussed in the Inter-Frame Processing template.

The proposed video templates are discussed in the next few sections, and example applications are provided to demonstrate the recommended approach to using the templates.

Proposed Video Templates

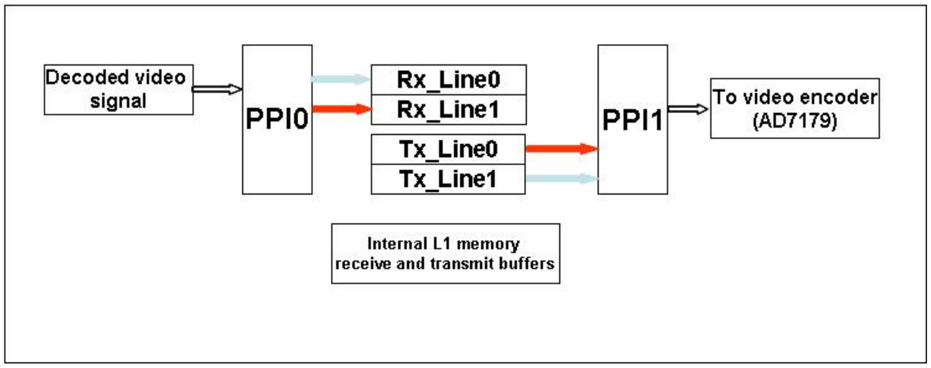
In this section, three different templates, based on the granularity of the sub-processing block, are broadly described. They include the sub-processing blocks in the spatial domain (line and macro-blocks), as well as templates for applications where data dependencies exist in the temporal domain.

Line Processing

For line processing, one approach is to collect one frame at a time in external memory from the peripheral. After each frame is collected, a line-by-line memory transfer can then be performed to L1 data memory, using either cache or DMA. A more efficient approach involves transferring a line directly from the peripheral into L1 data memory and subsequently transferring the processed line directly from L1 to a peripheral interface or external memory[1]. This decreases the number of memory transfers required and saves valuable external bus bandwidth.

[1] On ADSP-BF56x processors, two PPIs can be used for I/O video. On ADSP-BF53x processors, one PPI interface and one other peripheral interface (i.e., SPORT, Ethernet etc.) can be used for I/O. For sending/reading compressed images, the USB or LAN interface can be used on ADSP-BF56x or ADSP-BF53x processors.

Figure 1 shows the data flow implemented within the line-based template. DMA is used to transfer the image directly to L1 data memory from the peripheral. The image is then processed and sent out to the peripheral from L1 via DMA. Double-buffering is maintained in L1 memory to allow concurrent DMA and core accesses, in addition to avoiding L1 sub-bank contention.



Decoded video signal
PPI0
Rx_Line0
Rx_Line1
Tx_Line0
Tx_Line1
PPI1
To video encoder (AD7179)
Internal L1 memory receive and transmit buffers

Figure 1. Data Flow Diagram for Line Processing Template

Macro-Block Processing Template

For macro-block processing, the frame is processed in sections of dimension (n x m), where n is the height and m is the width of the macro-block. The L1 data memory would be insufficient to place the entire n rows of an image at a time, so either L2[2] or external memory must be used. However, to conserve the external bus bandwidth, using *only* L2 memory will help.

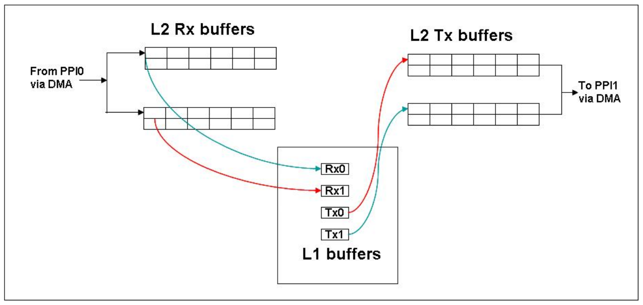


Figure 2. Data Flow Diagram for Macro-Block Processing Template

[2] L2 memory is available on ADSP-BF56x processors. On ADSP-BF53x processors, external memory can be used for placing buffers.

L2 memory would be insufficient to place an entire image, so only *n* rows of an image are transferred from the Parallel Peripheral Interface (PPI) to L2 memory at a time. Macro-blocks are then transferred from L2 to L1 data memory. Double-buffering is used in L1 and L2 memories, and DMA channels are used for all memory transfers. The dataflow diagram for this template is shown in Figure 2. Also, in this template, external memory can be used for an input or output frame buffer to ensure more uniform use of resources. In Figure 2, either the Tx or Rx buffers can be placed in external memory.

Inter-Frame Processing Template

This template can be used for applications in which dependencies exist between sub-processing blocks in the temporal dimension (i.e., between past and future frames). The L1 and L2 memory spaces are insufficient to place the dependent frames in memory. Thus, the external SDRAM memory is used to map the dependent frames.

Sub-processing blocks of the dependent frames are then transferred from external memory to L1 memory. If L1 memory space is insufficient, L2 memory can be used for placing internal buffers. The data flow diagram in Figure 3 shows an application in which dependencies exist on a macro-block basis over a past frame.

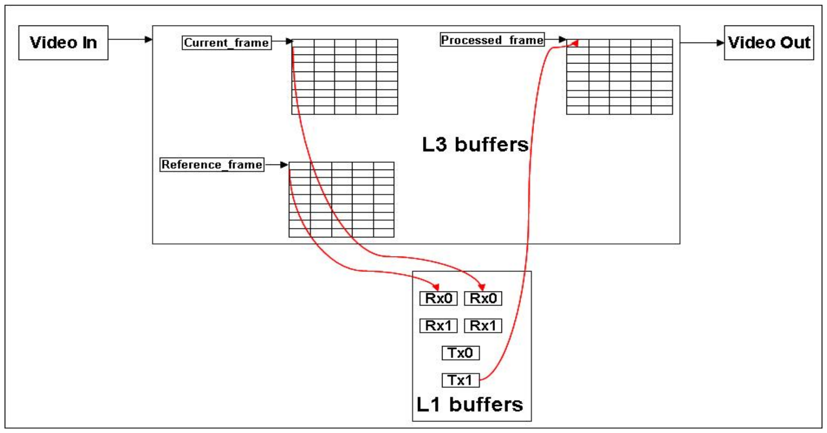


Figure 3. Data Flow Diagram for Inter-Frame Processing Template

In Figure 3, the `current_frame` and `reference_frame` pointers are interchanged every other frame. For applications where dependencies exist on a line basis, line transfers can be made between L1 and external memory.

Dependencies can also exist between several sub-processing blocks. In these cases, a slice of a frame (set of lines) can be transferred to internal memory.

Optimizations Specific to Blackfin Processors

To further improve performance and system bandwidth utilization, several optimizations within the templates have been adopted. The following optimizations are included in the templates:

- **16/32-bit transfers**: Maximum bus width is used for all peripheral DMA and memory DMA transfers. 16-bit transfers are initiated for ADSP-BF53x processors, and 32-bit transfers are initiated for ADSP-BF56x processors.
- **Efficient use of DMA channels**: No two simultaneous memory transfers are initiated on the same DMA channel.
- **SDRAM bank partitioning**: The SDRAM is partitioned into four banks to ensure simultaneous access to multiple frame buffers and minimal turn-around times. For example, the ADSP-BF561 EZ-KIT Lite® board has 64 Mbytes of SDRAM, which can be configured as four 16-Mbyte internal SDRAM banks. To take advantage of the bank structure, no two frame buffers that are accessed simultaneously are mapped to the same SDRAM bank.
- **DMA traffic control**: DMA traffic control registers are used for efficient system bandwidth utilization.

Refer to *Embedded Media Processing*[3] and *Video Framework Considerations for Image Processing on Blackfin Processors (EE-276)*[5] to learn more about developing optimized multimedia applications on Blackfin processors.

Using the Templates

To use the templates for specific applications, first identify the following items:

- The granularity of the sub-processing block in the image processing algorithm. This is required to choose a specific template for the application.

- The available L1 and L2 data memory, as required by the specific templates.
- An estimate for the computation cycles required per sub-processing block. This will indicate whether the application meets the real-time constraints within the template.
- The spatial and temporal dependencies between the sub-processing blocks. If dependencies exist, modify the templates to account for data dependencies.

Table 2 lists the core cycles available per sub-processing block for each of the templates. The table lists the cycles for use with Debug and Release modes of VisualDSP++® builds. Fewer core cycles are available in macro-block processing because it requires additional pointer manipulation. For inter-frame processing, multiple transfers are required between dependent frames, reducing the available core cycles as compared to line or macro-block processing. Note that the cycles shown for inter-frame processing are only for a transfer of a slice of a frame (set of lines) to internal memory.

Template	**Approx. CCLK/Pixel (Debug)**	**Approx. CCLK/Pixel (Release)**	**L1 Data Memory Required**	**L2 Data Memory Required**	**Comments**
Line Processing	36	42	2 * Line Size	N/A	Buffer chaining with loopback Double-buffering in L1
Macro-Block Processing	30	36	2 * Macro-Block Size Macro-Block Size = n * m	Macro-Block Height * 2 * Line Size (Size of Frame Slice)	Buffer chaining with loopback Double-buffering in L1 and L2
Inter-Frame Processing	30	35	Sub-Processing Block Size * # of Dependent Blocks	Sub-Processing Block Size * # of Dependent Blocks	Only L1 or L2 can be used Buffer chaining with loopback Double-buffering in L1 or L2

Table 2. Specification for Each Template

Since most data accesses are managed from L1 memory, the number of cycles lost due to memory latency is minimized. The cycles shown can therefore be entirely budgeted for the core computation cycles required by an algorithm.

The core processing cycles required for an algorithm should be computed theoretically or obtained by using the cycle counters available on Blackfin processors. A simulator or emulator session in VisualDSP++ can be used to obtain this information.

Table 2 also shows the required data memory space for L1 and L2 memories and the buffer chaining mechanism used within the templates. For more information on buffer chaining, refer to the DMA drivers in the *VisualDSP++ Device Drivers and System Services Manual for Blackfin Processors*[6].

Example Applications

Each of the templates is evaluated with an example application. Table 3 lists the examples provided for each of the templates.

Template	Application
Line Processing	Color conversion, YUV4:2:2 to RGB
Macro-Block Processing	Edge detection
Inter-Frame Processing	Motion detection

Table 3. Example Applications for Each Template

The templates and the examples for each are included in the associated ZIP file [7] for this EE-Note. The examples are located in the directory:

`<template_name>/BF561/examples`

These examples were tested on the ADSP-BF561 EZ-KIT Lite evaluation board with an input camera source and output video configured for ITU-R-656 frame format. For details on ITU-R-656 format, refer to the appropriate *Hardware Reference Manual* [1,2]. The example programs operate on a D1-sized (720x480) image.

Combining Templates

Some applications are likely to have more than one image-processing algorithm with different sub-processing block granularities or data dependencies between the sub-processing blocks. In that case, multiple templates can be combined into one project. This section discusses the combination of templates for such applications.

Multiple Sub-Processing Blocks

Consider an application that may involve some pre-processing on a line basis, along with macro-block processing. In this case, the line processing template and the macro-block processing template can be combined into one framework. Line processing should be modified to store the processed image to L2 or external memory, and then the macro-block template should be invoked to process the stored image.

Multiple Access to a Sub-Processing Block

Consider another example, such as histogram equalization, where the line processing template is invoked twice. In this case, the image is accessed twice—once to compute the cumulative grayscale values, and a second time to apply the equalization. For this application, one line at a time from L1 can be processed, stored back to external memory, and then brought back into L1.

Data Dependency in the Spatial Dimension

Data dependencies may exist between several lines or macro-blocks of an image, and access to multiple sub-processing blocks at one time may be required. An example use for this template is the motion estimation algorithm, which accesses several macro-blocks.

Dual-Core/Multiprocessor Applications

The templates can be combined to develop applications for the ADSP-BF561 dual-core processor in a multiprocessor system. Based on the dual-core programming model, different templates can be executed on either core.

Conclusions

Managing data efficiently is critical to increase performance and improve system bandwidth utilization on resource-constraining embedded platforms. To help address this, efficient data management techniques have been incorporated within a set of templates for developing multimedia applications on Blackfin processors. The templates will help to produce optimized data layouts and reduce development time.

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

[4] *Digital Video and HDTV*. Charles Poynton. Morgan Kaufmann Publishers Inc., San Francisco, CA,USA, 2003.

[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] *Associated ZIP File*. Rev 1, September 2006. Analog Devices, Inc.

Readings

[8] *ADSP-BF53x/ADSP-BF56x Programming Reference*. Rev 1. May 2005. Analog Devices, Inc.

Document History

Revision	**Description**
Rev 1 – September 28, 2006 by Kaushal Sanghai	Initial Release