Freescale Semiconductor
Application Note

Document Number: AN3519
Rev. 0, 9/2007

Optimizing Performance for the MPC5500 Family

by: Alistair Robertson
Powertrain Systems EKB

1 Introduction

The MPC5500 family of highly integrated microcontrollers boasts a host of new features, including a crossbar switch (XBAR), an enhanced Direct Memory Access (eDMA), and a Memory Management Unit (MMU).

These features are initialized upon negation of reset by a software program called the Boot Assist Module (BAM). This main purpose of the BAM is to initialize the device to an acceptable level, then locate and execute the user's application code.

Although the BAM performs basic initialization, it does not necessarily provide the optimum settings for any given application.

This application note provides guidance for users to fully optimize their application to achieve the highest possible performance from the MPC5500 family. It provides a description of the areas that should be focused on when optimizing an application, and provides benchmark data

Contents

to indicate the potential benefits. It focuses on hardware configurations and certain aspects of the application software, such as compiler settings and optimizations.

This paper does not provide software examples and it is intended to supplement application note AN2789, “MPC5500 Configuration and Initialization”.

This information in this document applies to the following devices.

MPC5553, MPC5554, MPC5533, MPC5544, MPC5561, MPC5565, MPC5566 and MPC5567.

2 Hardware Optimizations

2.1 Branch Target Buffer

To resolve branch instructions and improve the accuracy of branch predictions, the e200z6 and e200z3 Zen cores implement a dynamic branch prediction mechanism using a branch target buffer (BTB), a fully associative address cache of branch target addresses. Its purpose is to accelerate the execution of software loops with some potential change of flow within the loop body.

By default, this feature is disabled following negation of reset and execution of the BAM. It is controlled by the Branch Unit Control and Status Register (BUCSR). The BTB’s contents should be flushed and invalidated by writing BUSCR[BBFI]=1, and it may be enabled by writing BUSCR[BPEN]=1.

Generally, the BTB has a bigger impact on cacheless devices, due to the increased average cycle count for accessing the Flash memory.

Figure 1 details the potential performance gains achievable by enabling the BTB.

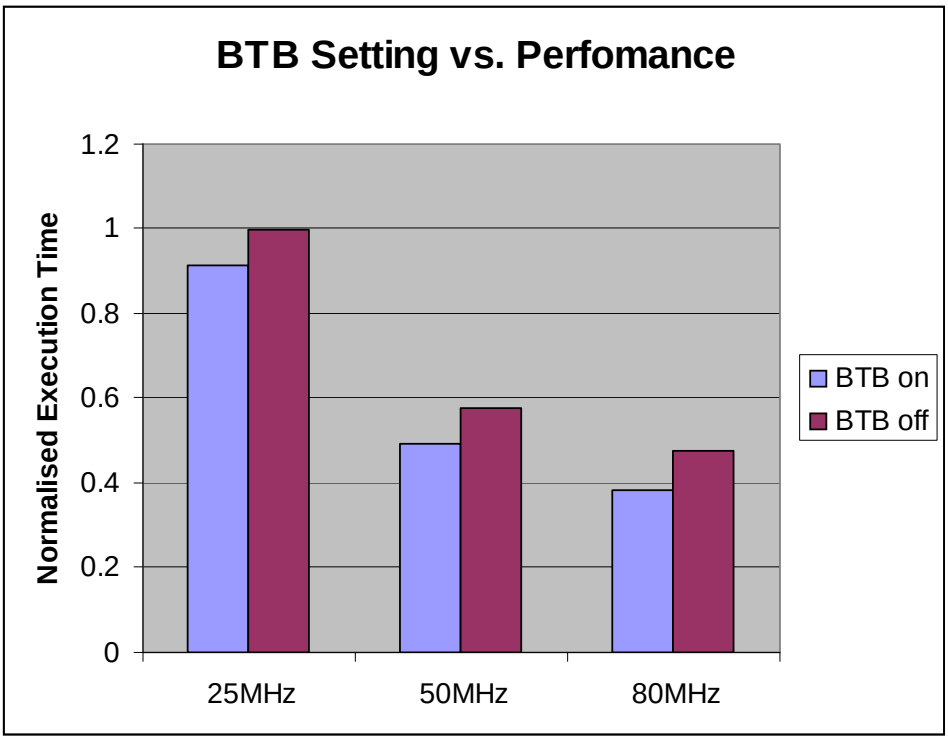

Figure 1. Performance impact of Branch Target Buffer

NOTE

Data for Figure 1 collected running Dhrystone version 2.1, compiled with Greenhills Multi Version 5.0 (Beta), with no optimizations running on MPC5534 Rev. A.

The Hardware Implementation Dependent Register 0 (HID0) register is a core implementation dependent register that is used for various configuration and control functions. Additional control of the BTB is available by using the HID0[BPRED] field. This controls whether forward or backward branches (or both) are candidates for entry into the BTB, and thus for branch prediction. The default setting is BPRED = 0b00, which enables forward and backward branch prediction, but the optimum setting should be assessed. It is recommended to not disable branch prediction.

2.2 Flash Bus Interface Unit Settings

The Flash Bus Interface Unit (FBIU) interfaces the system bus to the Flash memory array controller. The Flash BIU contains prefetch buffers and a prefetch controller which, if enabled, prefetches sequential lines of data from the Flash array into the buffer. Prefetch buffer hits allow zero-wait state responses.

The Flash Bus Interface Unit Control Register (BIUCR) register controls access to the internal Flash array. Its settings define the number of cycles required to access the array, access times, and how the prefetch buffering scheme operates.

Following negation of reset and execution of the BAM, the instruction and data prefetching is disabled, and the number of cycles required to access the internal Flash array is set to its maximum value of seven additional wait states.

As the operating frequency of the device is set by configuring the FMPLL (see Section 2.3, “Frequency Modulated Phase Locked Loop”), the number of cycles required to access the internal array should be configured accordingly. Note that the Flash BIUCR cannot be altered by code executing from the Flash array. Code for configuring the Flash should be copied to and executed from system RAM.

The reference manual for each MPC5500 device contains tables dictating the required Flash BIUCR settings for a given operating frequency. The reference manuals also provide recommendations for the prefetch buffer settings. Note that the BIUCR settings vary across the MPC5500 family and may vary between revisions of a particular device.

Table 1 shows the recommended setting for the Flash BIUCR register for specific frequency ranges for the MPC5534. For other devices, refer to the Flash chapter of the device reference manual for specific settings.

Table 1. BIUCR settings for MPC5534 Rev A

Frequency	MnPFE	APC	WWSC	RWSC	DPFEN	IPFEN	PFLIM	BFEN
Up to 25 MHz[1]	0xF	0x0	0x1	0x0	0b1	0b1	0x6	0b1
Up to 50 MHz	0xF	0x1	0x1	0x1	0b1	0b1	0x6	0b1
Up to 75 MHz	0xF	0x2	0x1	0x2	0b1	0b1	0x6	0b1
Up to 82 MHz	0xF	0x3	0x1	0x3	0b1	0b1	0x6	0b1

1 Requires that Pipelined Reads be disabled in the Flash Module Configuration register (Flash_MCR[PRD]=0b1).

MnPFE: Master Prefetch Enable
APC: Address Pipelining Control
RWSC: Read Wait State Control
WWSC: Write Wait State Control
DPFEN: Data prefetch Enable
IPFEN: Instruction Prefetch Enable
PFLIM: Prefetch Limit
BFEN: FBIU line read buffers Enable

The MPC5534/3 has another Flash bus interface control register, BIUCR2. This register is not implemented on the other MPC5500 devices described in this application note. This register contains a Line Buffer Configuration field, BIUCR2[LBCFG]. This may be used to determine how many of the prefetch buffers are allocated to instruction fetches, and how many are allocated to data accesses. The default value is copied from a location in the shadow row of the Flash array following reset. This optimum setting for this register is application dependent, but in most cases the value of 0b00 (which allocates all buffers to any Flash access) will provide the best performance.

The data in Figure 2 illustrates the impact of BIUCR settings.

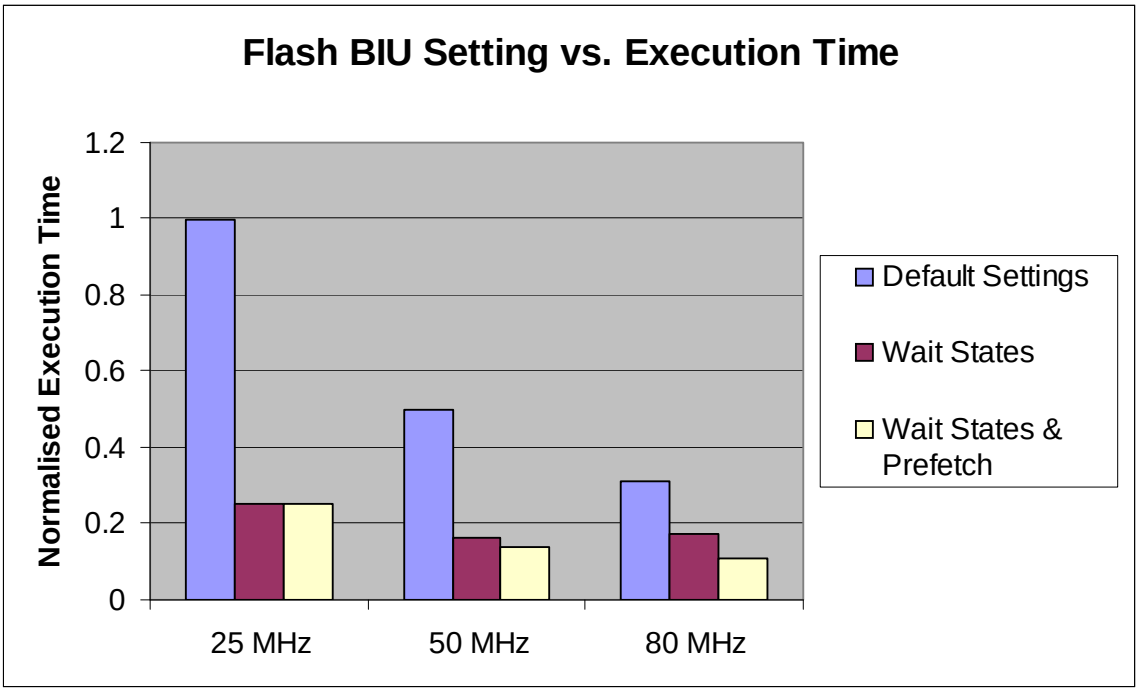

Figure 2. Performance Impact of Flash BIU Settings

NOTE

Data for Figure 2 collected running Dhrystone version 2.1, compiled with Greenhills Multi Version 5.0 (Beta), with no compiler optimizations running on an MPC5534 Rev A.

2.3 Frequency Modulated Phase Locked Loop

The default operating frequency of the MPC55xx device is 1.5 times the crystal reference frequency. Typically, the system frequency is increased shortly after reset negates, to provide acceptable performance. Take care to ensure that the correct internal and/or external Flash configuration is chosen for the selected system frequency (refer to Section 2.2, “Flash Bus Interface Unit Settings”). Application Note AN2789, section 4.2.4 “Initialize the PLL”, provides details on how the frequency modulated phase locked loop (FMPLL) should be initialized in an application.

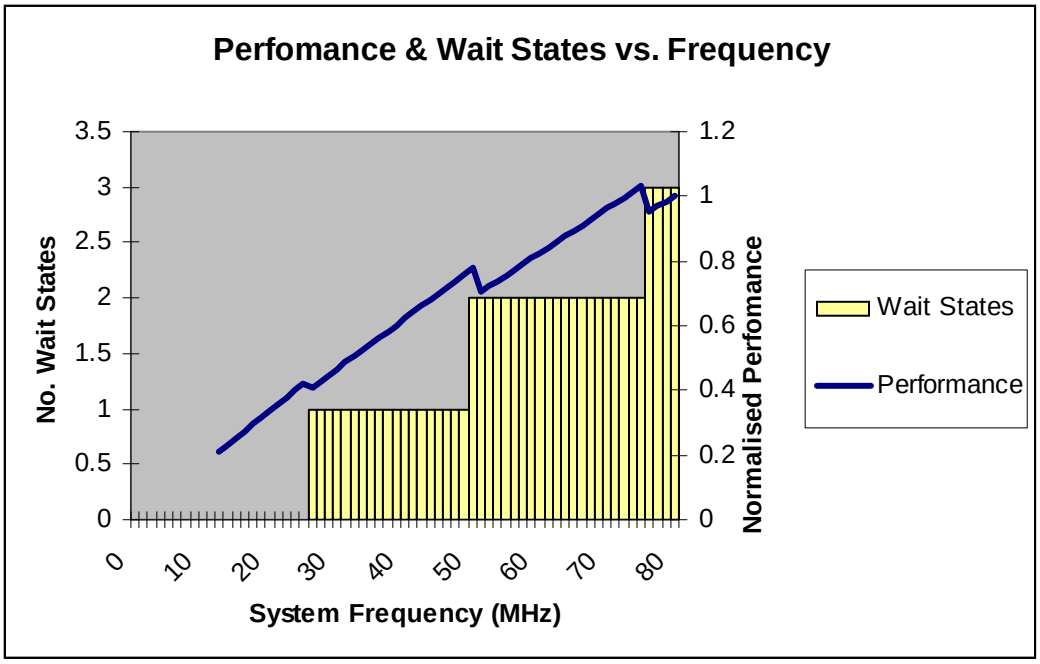

Figure 3. System Performance Versus Operating Frequency

NOTE

Data for Figure 3 collected running Dhrystone version 2.1, compiled with Greenhills Multi Version 5.0 (Beta), with no optimizations running on an MPC5534 Rev A.

System performance cannot be linearly extrapolated with system frequency, as is often the expectation. It is due to the insertion of additional Flash wait states as system frequency increases that system performance does not scale linearly, as shown in Figure 3. For example, an MPC5534 operating at 50 MHz provides greater performance with lower power consumption than when operating at 56 MHz.

2.4 Crossbar Switch

This multi-port crossbar switch (XBAR) supports simultaneous connections between master ports and slave ports. The XBAR allows for concurrent transactions to occur from any master port to any slave port. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions. By default, requesting masters are granted access based on a fixed priority. A round-robin priority mode also is available.

The e200z6 based MPC555x and MPC556x devices have Von Neumann architectures: they have a single unified bus for instruction fetches and data accesses. The default XBAR configuration sets the CPU bus as the highest priority master on the XBAR switch with all slaves parked on this master. This default configuration for the MPC555x/6x crossbar is typically the optimum configuration.

The e200z3 based MPC5534/3 has a Harvard architecture: it has independent buses for instruction fetches and data accesses. However, by default, the BAM applies the same configuration to the XBAR registers as for the unified bus of the MPC5554. This default setting provides less than optimum performance.

Optimal XBAR settings are application dependent, but in e200z3 based devices assigning the CPU data bus to have highest priority and parking the slave port associated with system RAM on this master generally provides the best overall performance.

Figure 4 shows the performance improvement possible on the MPC5534 by reconfiguring the XBAR settings.

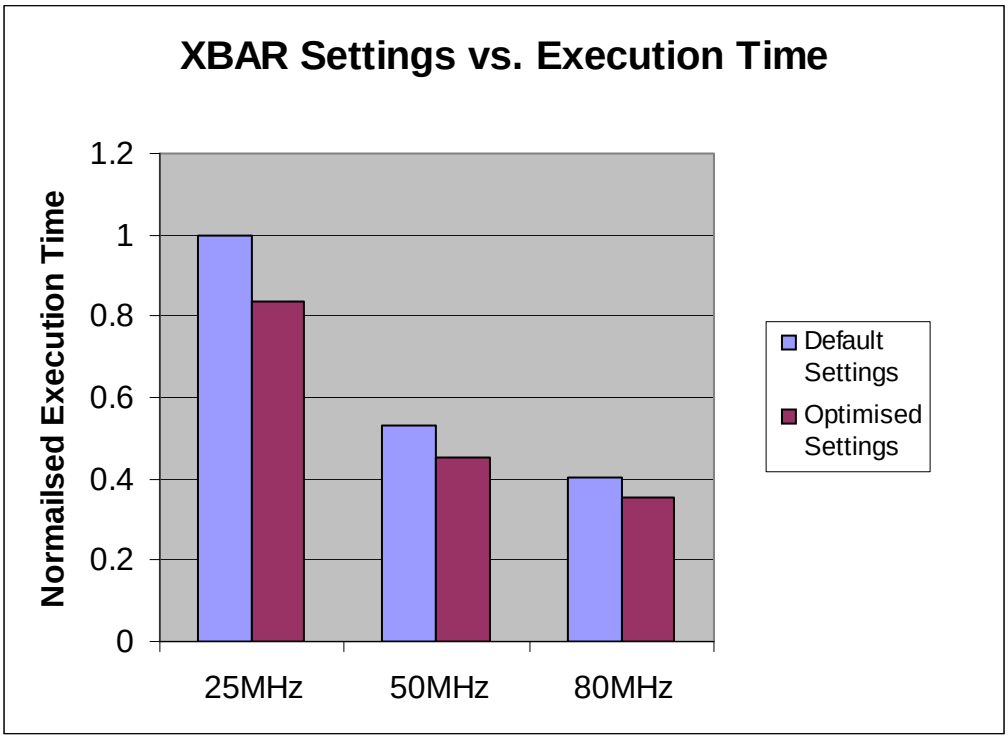

Figure 4. Improving MPC5534 performance by reconfiguring the XBAR

NOTE

Data for Figure 4 collected running Dhrystone version 2.1, compiled with Greenhills Multi Version 5.0 (Beta), with no optimizations running on an MPC5534 Rev A.

To reconfigure the XBAR on the MPC5534, write the following resisters:

1. XBAR_SGPCR3 = 0x0000_0004. *This parks the slave 3 (internal SRAM) on master port 4 (CPU data bus)*
2. Write XBAR_MPR0 = 0x0000_0321. *This sets slave port 0 (Flash) to give the master port 4 (CPU data bus) highest priority.*

2.5 Cache and MMU

The MPC5554 provides an 32-kilobyte, 8-way set-associative, unified (instruction and data) cache with a 32-byte line size. The MPC5553 provides an 8-Kbyte, 2-way set associative unified cache with a 32-byte line size. The MPC5534/3 has no cache. In all cases the Cache is disabled by default.

The cache improves system performance by providing low-latency instructions and data to the e200z6 instruction and data pipelines, which decouples processor performance from system memory performance. There are several stages to enabling the cache. Not only does the cache itself have to be invalidated then enabled, but memory regions upon which it can operate must be configured in the MMU to permit cache access.

General rules for using cache to improve performance:

- Enable cache for all memories being executed from.
- Enable cache for data memories that do not change.
- Reserve part of cache for stack usage.
- Copyback mode in the cache generally uses fewer system resources.
- Avoid using cache copyback mode where another master (for example, eDMA) can access the same memory as the core.
- Consider locking critical performance routines in cache.

Application note AN2789 provides software examples for enabling the cache and locking the stack. Figure 5 shows an example of the performance benefits on using the cache in an MPC5554 and MPC5553 system.

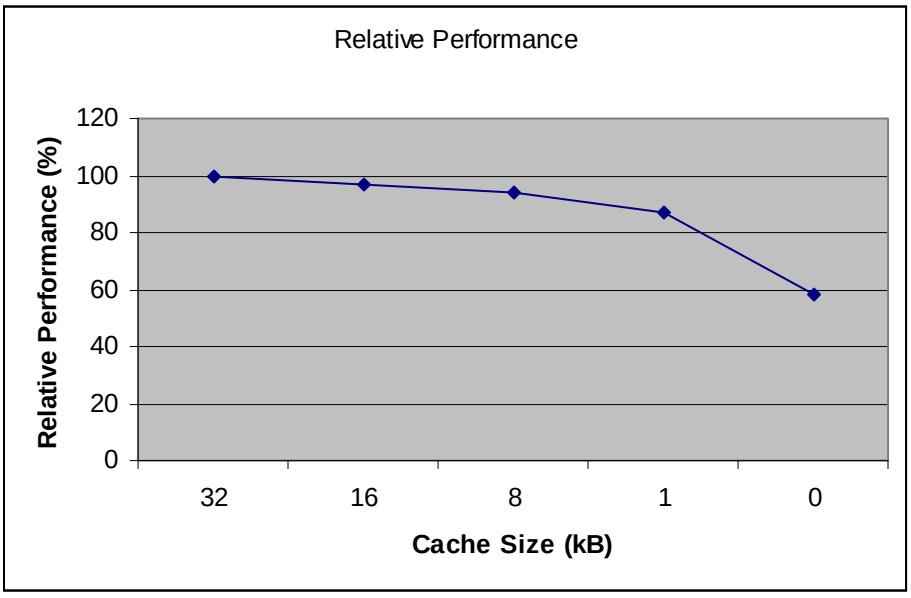

Figure 5. Performance Enhancements for MPC555x and MPC556x Caches

NOTE

Cache performance figures generated with Architectural Modeling Environment simulator, running Powertrain benchmark.

Note that the default configuration after the BAM has executed is for the entire system RAM to remain cache inhibited.

3 Application Software

3.1 Compiler Optimizations

The most significant opportunity for influencing the performance of a given application is by compiler and linker optimizations. Optimizing is a trade off between code size and performance. The higher the performance of the application, the greater the size of the code generated. Compilers use a host of features, such as loop unrolling, function inlining and application profile feedback to make the desired trade-offs between enhanced performance and minimized code size.

The data in Figure 6 shows the effects of compiler optimization on a simple application. In this case, the Dhrystone benchmark was run under three conditions:

- Optimized for small code size
- Optimized for high performance
- No optimizations (a trade-off between code size and performance.)

Although this is an extreme example, it highlights how significant the role of the compiler and linker is in determining the overall performance of an application.

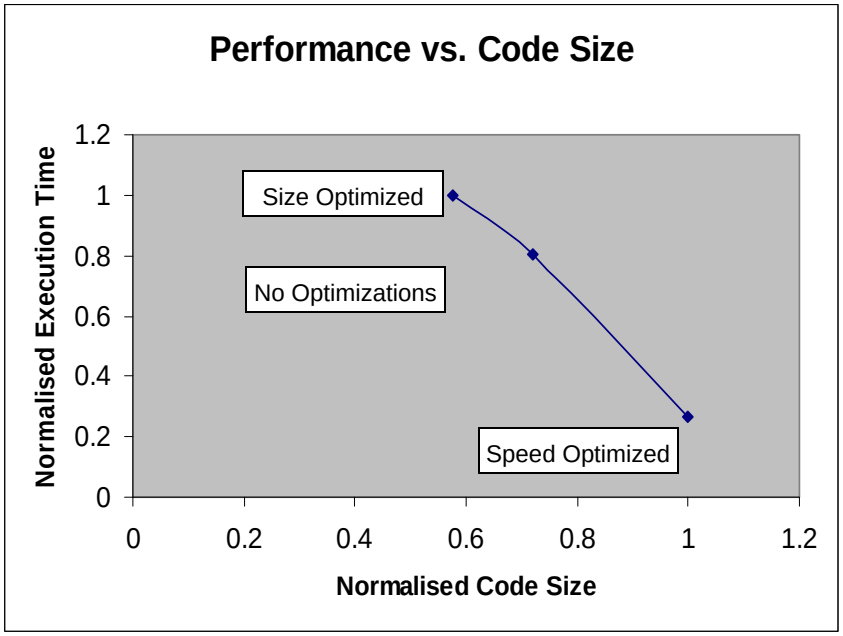

Figure 6. Influence of Compiler Settings on Application Performance and Code Size

NOTE

Dhrystone version 2.1 run on MPC5534 Rev A. Compiled using Greenhills Multi version 5.0 (Beta). Speed optimizations were –OL, -OI and -OB. Code Size optimizations were –OS.

The compiler optimizations do not necessarily have to be applied to the entire application. Analysis of an application can identify time critical functions that may subsequently be targeted for performance optimization, without incurring the impact of optimizing the entire application.

There are several other aspects of the compiler and linker that should be considered. In particular, the use of Small Data Areas (SDAs, sometimes referred to as Special Data Areas) can make a significant performance improvement. Generally, most compilers, by default, do not use these.

SDAs provide a means to address data quickly. A core general purpose register can act as an address anchor. Data can be accessed quickly by executing load/store instructions to memory addressed at an offset from the anchor. The range of memory that is addressable by this offset is referred to as the Small Data Area. The Power Architecture Embedded Application Binary Interface (EABI) specifies up to three SDAs as standard, although some compilers allow the user to add additional SDAs as required.

3.2 Signal Processing Extension

To further optimize time critical functions, the Signal Processing Extension Auxiliary Processing Unit (SPE-APU) may be used. The SPE-APU provides a set of Single Instruction Multiple Data (SIMD) instructions. These SIMD instructions typically involve performing the same operation on multiple data elements stored within a single 64-bit register. Through the implementation of SIMD instructions, including vector multiply and accumulate (MAC) instructions, the SPE APU provides Digital Signal Processing (DSP) functionality. This can be used to accelerate signal processing routines, such as Finite Impulse Response (FIR), Infinite Impulse Response (IIR) and Discrete Fourier Transforms (DFT).

The data in Table 2 provide an example of the performance improvements that can be made for typically DSP functions. For a full set of performance figures refer to the “MPC5500 DSP Function Library” and the “MPC5500 DSP Function Library 2” user manuals.

Table 2. Example Performance Improvements Achieved Using SPE-Assembly Versus C-Code

Function	Description	Configuration	Cycle Count	Improvement to C-Function
FIR	Real 32TAP FIR filter with symmetrical impulse response	256 entry output vector	9447	18.5
IIR	Direct form 2nd order IIR filter	256 entry output vector	2371	3.7
FFT	256-point complex FFT, single precision floating point	Vector of twiddle factors of length 256 in internal Flash	12465	5.2

NOTE

The “Improvement to C function“ column in Table 2 shows performance increase comparing the assembly code to a respective C function. The value is calculated as the ratio of the number_of_clock_cycles_of_C_function to the number_of_clock_cycles_of_optimized_library_function.

3.3 Hardware Single Precision Floating Point

The SPE-APU also supports 32-bit IEEE®-754 single-precision floating-point formats, and supports vector and scalar single-precision floating-point operations. Most compiler vendors include libraries that can emulate floating point functionality. However, by specifying the correct compiler options, the single precision floating point instructions may be used.

Experiments on small sections of floating point C-code have demonstrated performance increase in excess of an order of magnitude, when enabling hardware floating point over software floating point emulation.

3.4 Variable Length Encoding

In addition to the base Power Architecture Book E instruction set support, the Zen cores also implement the VLE (variable-length encoding) APU, providing improved code density. The VLE-APU can be viewed as a supplement to the existing Power Architecture instruction set that can be conditionally applied to a portion of, or an entire application for which improved code density is desired.

Using it is straightforward:

1. Select the appropriate compiler target and option to generate VLE code.
2. Configure the Memory Management Unit (MMU) to specify VLE attributes for the relevant MMU pages.

VLE-enabled cores run both Book E and VLE instruction encodings on a page by page basis, with pages defined by the MMU. The performance impact of implementing VLE is small. Generally, there is less than 3% performance impact, and, more typically for Powertrain code, this is closer to 1%.

Table 3 provides benchmark examples of the code size improvement that may be achieved with VLE for a range of applications.

Table 3. VLE Code Size Reductions

Benchmark	Code Size Reduction (%)
Freescale General Purpose Code	30.4
EEMBC	32.2
SpecINT95	29.2
Freescale Powertrain Code	29.4
X Powertrain Code	28.6
Y Powertrain code	25.6-28.9
Z Powertrain Code	30.5

Notes:
1. Data collected using VLE with Green Hills Compiler (MULTI v4.0.7 compared to MULTI v4.0.4 non-VLE)
2. The VLE APU is further documented in “PowerPC VLE APU Definition, Version 1.00", a separate document.
3. The MPC5554 and MPC5553 do not support VLE.

4 Peripherals and General Application Guidelines

Optimizing the device configuration and compiler setup is only one part of optimizing an entire application. Correct use of the peripherals can also dramatically improve overall system performance. In particular, use of the interrupt controller, the enhanced Direct Memory Access (eDMA), and intelligent peripherals such as the Enhanced Timer Processing unit (eTPU), can off-load significant work from the CPU.

For example, the eDMA may be used to shift data to avoid unnecessary CPU loading. Most peripheral modules can generate eDMA requests to trigger data transfers. A typical application example would be using the eDMA to pass conversion commands to the analog to digital converter (ADC) and maintaining circular buffers of the ADC results in the system RAM, with no core intervention.

The Performance Optimization Checklist in the next section provides several system level examples of how to optimize an application.

5 Performance Optimization Checklist

1. Hardware Configuration		
Description	**Register(s)**	**Details**
Branch Target Buffer	Flush with BUSCR[BBFI] Enable with BUSCR[BPEN]	Flush and enable to improve accuracy of branch predictions
Branch Prediction	HID0[BPRED]	Consider fine tuning of BTB operation for specific applications.
System Frequency	FMPLL_SYNCR	Select desired frequency taking into account performance impacts of additional wait states.
Flash Wait States	BIUCR[APC, WW, RWSC]	Refer to device reference manuals for BIUCR settings for FMPLL frequency ranges.
Flash Prefetching	BIUCR[DPFEN, IPFEN, PFLIM, BFEN]	Enable prefetching for data and instructions.
Flash Prefetch Algorithm	BIUCR2[LBCFG]	Allocate buffers to data and/or instructions. Fine tune for specific applications (MPC5534 and MPC5533 only).
Crossbar Switch	Park slave SRAM on master port with XBAR_SGPCR3. Set Flash slave port to highest priority with XBAR_MPR0.	For e200z3 based devices reconfigure to optimize for Harvard architecture
Cache	Invalidate cache with L1CSR0[CINV] Enable cache with L1CSR0[CE]	Invalidate and the enable the cache for instructions. Assess in application best configuration for using cache with data. Make application dependent decision on copyback operation and store/push buffer configuration.
Memory Management Unit	TLB_MAS2[VLE, I]	Configure relevant pages for cache and VLE by setting MMU TLB attributes.

2. Software Configuration		
Description	**Registers**	**Details**
Compiler optimization	N/A	Use the features of the compiler to select the optimum trade off between code size and performance improvements. Enable usage of small data areas.
Hardware floating point	Enable with MSR[SPE]	Set compiler switches to specify using hardware single precision floating point as opposed to software emulation.
Signal processing extensions	Enable with MSR[SPE]	Take advantage of the SPE-APU to encode time critical functions in SPE assembly code.
Variable Length Encoding	Enabled with TLB_MAS2[VLE]	Set compiler switches and configure the MMU to take advantage of the VLE-APU.

3. Peripherals and General Application Guidelines
• Use the eDMA rather than the core to move data where possible. Most peripherals can generate eDMA requests to shift data. — Use the eDMA to control movement of commands and results from ADC and to maintain circular buffers in system memory. — Create circular buffers of ADC results can exist in RAM with zero core overhead.
• Shift loading from the CPU to the eTPU whenever possible. — Each eTPU provides ~1MIPs performance. — eTPU can trigger ADC directly – no need for CPU interruption.
• Avoid software polling and allow peripherals to request interrupts or eDMA servicing. — Use hardware instead of software vectored interrupts to reduce latency. — Trigger eDMA requests rather than interrupting the CPU to move data/results.
• Avoid external memories unless absolutely necessary. — Place time critical functions in internal memory. — Enable bursting on the external bus. — Reduce external bus wait states from default maximum settings

6 References

1. AN2789 Version 1.1, 10/2004.
2. MPC5534 Reference Manual Addendum Version 1, 6/2006
3. MPC5554/3 Reference manual version 3.1, 11/2005
4. MPC5500 DSP Function Library users manual.
5. MPC5500 DSP Function Library 2 users manuals.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3519
Rev. 0
9/2007

freescale™
semiconductor