DSP/BIOS II Timing Benchmarks on the TMS320C6000 DSP

Shawn Dirksen *Digital Signal Processing Solutions*

ABSTRACT

This document describes each of the DSP/BIOS II performance benchmarks and the accompanying results, followed by a technique used for calculating overall system performance or overhead. To help designers better analyze their system performance, we have detailed the methodology used for obtaining each of the benchmarks along with the number of CPU cycles to execute each of the DSP/BIOS II functions. The designers can then compute the sum of these components and the frequency of occurrence to determine the total system performance for their application.

Contents

List of Figures

List of Tables

1 DSP/BIOS II Timing Benchmarks

DSP/BIOS II functions have been described along with the approach taken to measure each performance benchmark.

1.1 LOG – Log Benchmarks

LOG_event. This is the execution time of a LOG_event function call, which is used to append an unformatted message to an event log.

LOG_printf. This is the execution time of a LOG_printf function call, which is used to append a formatted message to an event log. The execution time of the function is not dependent on the number of arguments specified in the function call.

1.2 STS — Statistics Benchmarks

STS_add. This is the execution time of a STS_add function call, which is used to update the total, count, and max fields of a statistics object.

STS_delta. This is the execution time of a STS_delta function call, which is used to update a statistics object, using the difference between a provided value and a previous setpoint value.

STS_set. This is the execution time of a STS_set function call, which is used to set the previous value for a statistics object.

1.3 TSK — Task Yield Benchmarks

TSK_yield. This is a measurement of the elapsed time between a function call to TSK_yield(which causes preemption of the current thread yielding control of the processor), and the execution of the first instruction in a task of equal priority, as shown below.

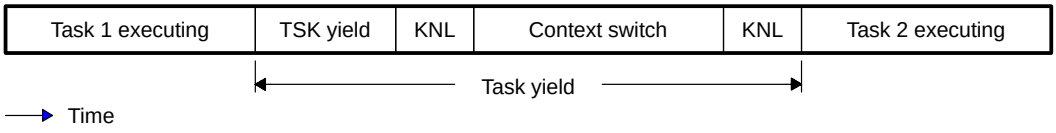

Figure 1. Task Yield Benchmarks

1.4 SEM — Semaphore Benchmarks

The semaphore benchmarks measure the time interval between the issuance of a post (SEM_post) or pend(SEM_pend) function call and the resumption of task execution, both with and without a context switch. The results are independent of task priority, an inherent characteristic of DSP/BIOS that makes it ideal for signal processing applications that require predictable, consistent real-time response.

Post of a Semaphore, no context switch. This is a measurement of a SEM_post function call, when the posted task is not higher priority that the currently running TSK, and no preemption occurs:

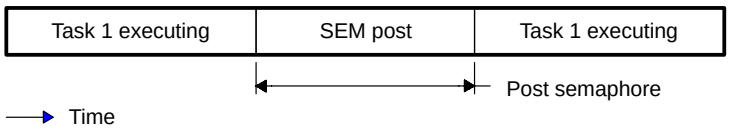

Figure 2. Semaphore Benchmarks

Post of a Semaphore, with context switch. This is a measurement of the elapsed time between a function call to SEM_post (which causes preemption of the current task), and the execution of the first instruction in the higher priority task, as shown below:

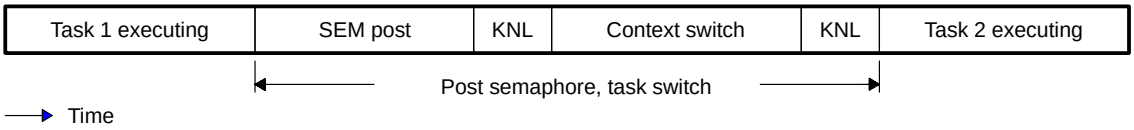

Figure 3. Post of Semaphore Task Switch

Pend of a Semaphore, no context switch. This is a measurement of a SEM_pend function call without a context switch.

Pend of a Semaphore, with context switch. This is a measurement of the elapsed time between a function call to SEM_pend(which causes preemption of the current task), and the execution of the higher priority task.

1.5 SWI — Software Interrupt Benchmarks

Post of Software Interrupt, no context switch. This is a measurement of a SWI_post function call, when the posted software interrupt is not higher priority than the currently running SWI, and no preemption occurs:

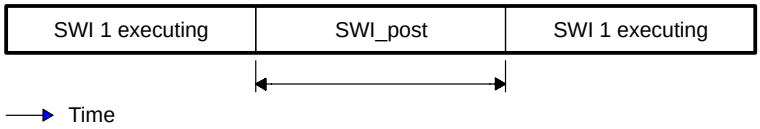

Figure 4. Software Interrupt Benchmarks

Post of Software Interrupt, with context switch. This is a measurement of the elapsed time between a function call to SWI_post (which causes preemption of the current thread), and the execution of the first instruction in the higher priority software interrupt, as shown below. The context switch for SWI 2 is performed within the SWI executive, and this time is included within the measurement.

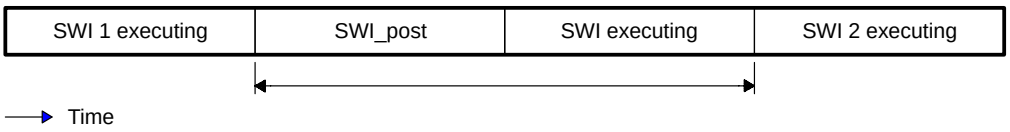

Figure 5. Post of Software Context Switch

1.6 PRD — Periodic Function Benchmarks

Timer Interrupt calling PRD_tick. This is a measurement of the elapsed time from the start to the completion of an ISR that calls PRD_tick to increase the system clock counter by one.

Timer Interrupt calling PRD_swi. This is a measurement of the elapsed time from the start of an ISR that calls PRD_tick and posts a software interrupt to the PRD_swi.

Timer Interrupt to Periodic Function. This is a measurement of the elapsed time from the start of an ISR that calls PRD_tick and posts a software interrupt, to the execution of the first instruction in the posted periodic function.

1.7 HWI — Hardware Interrupt Benchmarks

These benchmarks exhibit the interrupt latency typical of most interrupt processing applications independent a kernel being used. The interrupt latency provides a useful measure of worst-case interrupt response, but does not reflect the scheduling capability of the DSP/BIOS kernel (launching threads to perform background processing for the ISR). This is further demonstrated in the Hardware Interrupt to Software Interrupt and the Hardware Interrupt to Blocked Task benchmarks.

Interrupt prolog. This is a measurement of the execution time of an HWI_enter macro call. HWI_enter must be called in an ISR prior to any DSP/BIOS API calls that can post or affect a software interrupt (SWI). The execution time of the HWI_enter macro depends upon the list of registers to be saved for the ISR, as defined in masks specified by the user. This benchmark shows the minimum execution time for the prolog, with no registers saved.

Interrupt prolog for calling C function. This measurement is similar to the previous (interrupt prolog), but in this case the time shown in the data sheet corresponds to all C caller-preserved registers being saved, so that a C function can be called from the assembly stub.

Interrupt epilog. This is a measurement of the execution time of an HWI_exit macro call. HWI_exit must be the last statement of any ISR that calls HWI_enter. The execution time of HWI_exit depends upon the list of registers the user specifies to be restored. This benchmark shows the minimum execution time for the epilog, with no registers restored, and no higher priority SWIs posted in the ISR (i.e., following the ISR, execution resumes with the thread that was preempted by the hardware interrupt).

Interrupt epilog following C function call. This measurement is similar to the previous (Interrupt epilog), but in this case the time shown in the data sheet corresponds to all C caller-preserved registers being restored, with no higher priority SWIs posted in the ISR.

Hardware Interrupt to Blocked Task. This is a measurement of the elapsed time from the start of an ISR that posts a semaphore, to the execution of the blocked task:

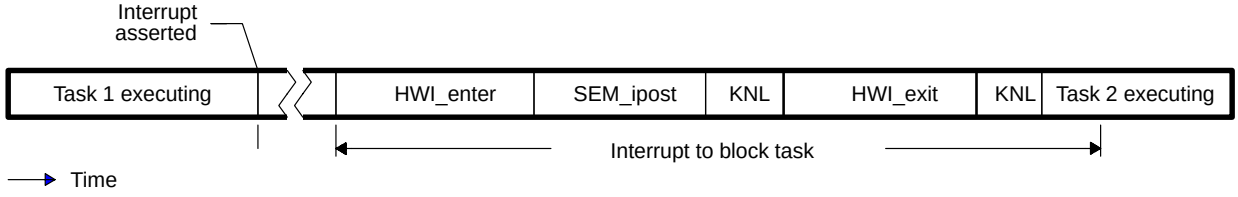

Figure 6. Hardware Interrupt to Blocked Task

Hardware Interrupt to Software Interrupt. This is a measurement of the elapsed time from the start of an ISR that posts a software interrupt, to the execution of the first instruction in the posted software interrupt:

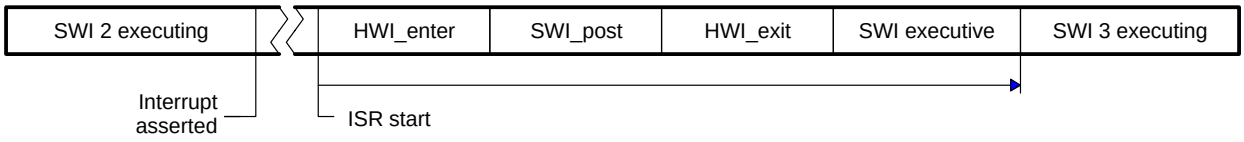

Figure 7. Hardware Interrupt to Software Interrupt

In the above example SWI 3 has a higher priority than SWI 2, so SWI 2 is preempted. The context switch for SWI 3 is performed within the SWI executive, and this time is included within the measurement. In this case, the registers saved/restored by HWI_enter/HWI_exit are only those modified by the SWI_post assembly macro.

Interrupt Latency. This is the maximum latency time during which DSP/BIOS disables maskable interrupts.

1.8 MBX — Mailbox Benchmarks

Post of a Mailbox, no context switch. This is a measurement of a MBX_post function call, when the posted mailbox copies a message into the unfilled mailbox, and no higher priority task is pending on the mailbox.

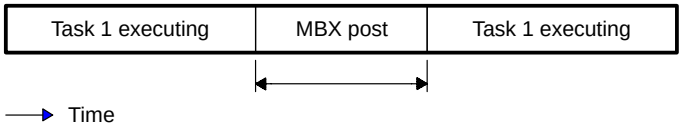

Figure 8. Mailbox Benchmarks

Post of a Mailbox, with context switch. This is a measurement of the elapsed time between a function call to MBX_post(which causes preemption of the current task), and a context switch to a higher priority task pending on the mailbox, as shown below:

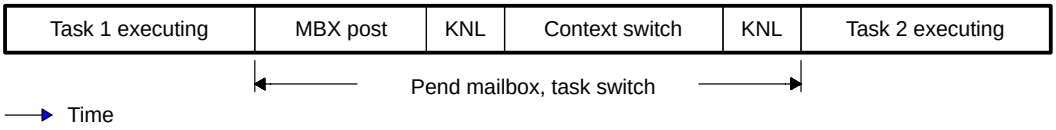

Figure 9. Post of a Mailbox with Context Switch

Pend of a Mailbox, no context switch. This is a measurement of a MBX_pend function call, when the unfilled pending mailbox copies a message, and no higher priority task is pending on the mailbox.

Pend of a Mailbox, with context switch. This is a measurement of the elapsed time between a function call to MBX_pend(which causes preemption of the current task) if the mailbox is empty or a higher priority task is blocked on a MBX_post.

1.9 PIP — Pipe Benchmarks

NOTE: Each of the following pipe benchmarks includes the execution time of a minimal notifyWriter (or notifyReader) C function call, i.e., a function that just does a return, but is considered to have modified all C caller–preserved registers.

PIP_alloc. This is the execution time of a PIP_alloc function call, which is used to allocate an empty frame from a pipe. PIP_free. This is the execution time of a PIP_free function call, which is used to recycle a frame back into a pipe.

PIP_get. This is the execution time of a PIP_get function call, which is used to get a full frame from a pipe.

PIP_put. This is the execution time of a PIP_put function call, which is used to put a full frame into a pipe.

2 DSP/BIOS II Timings

This data contains timing information for version 1.2 of DSP/BIOS II for the TMS320C6000 digital signal processors. These timings apply for the floating-point processor as well.

Environment Testing Platform: TMS320C6000 EVM, using TMS320C6201 internal memory for both code and data.

Software: DSP/BIOS version 4.00, built with TI Code Generation Tools, version 4.00.

2.1 Benchmark Results (C Language API)

Table 1. Benchmark Results

	Non-instrumented[i] CPU Cycles	Non-instrumented[i] Time (usec) @200Mhz[ii]	Instrumented CPU Cycles	Instrumented Time (usec) @200Mhz[ii]
Log Operations				
LOG_event	33	0.165	33	0.165
LOG_printf	36	0.18	36	0.18
Statistics Operations				
STS_set	14	0.07	14	0.07
STS_add	15	0.075	15	0.075
STS_delta	21	0.105	21	0.105
Task Yield				
TSK_yield	263	1.315	375	1.875
Semaphores				
Post semaphore, no task switch	182	0.91	269	1.345
Post semaphore, task switch	288	1.44	434	2.17
Pend semaphore, no task switch	152	0.76	205	1.025
Pend semaphore, task switch	278	1.39	389	1.945
Software Interrupts (SWIs)				
Post of Software Interrupt, no context switch	118	0.59	118	0.59
Post of Software Interrupt, with context switch	238	1.19	238	1.19
Periodic Functions				
Timer interrupt calling PRD_tick	113	0.565	113	0.565
Timer interrupt calling PRD_swi	360	1.80	360	1.80
Timer interrupt to Periodic function[v]	471	2.355	471	2.355
Hardware Interrupts[iii]				
Interrupt prolog (minimum)	32	0.16	32	0.16
Interrupt prolog for calling C function	41	0.205	41	0.205
Interrupt epilog (minimum)	52	0.26	52	0.26

Table 1. Benchmark Results *(Continued)*

	Non-instrumented[i] CPU Cycles	Non-instrumented[i] Time (usec) @200Mhz[ii]	Instrumented CPU Cycles	Instrumented Time (usec) @200Mhz[ii]
Interrupt epilog following C function call	65	0.325	65	0.325
Hardware Interrupt to Blocked Task	798	39.9	929	4.645
Hardware Interrupt to Software Interrupt	336	1.68	336	1.68
Interrupt Latency[iv]	72	0.36	72	0.36
Mailboxes				
Post mailbox, no task switch	431	2.155	571	2.855
Post mailbox, task switch	800	4.00	1086	5.43
Pend mailbox, no task switch	433	2.165	573	2.865
Pend mailbox, task switch	300	1.5	411	2.055
Pipe Operations				
PIP_alloc	98	0.49	98	0.49
PIP_free	93	0.465	93	0.465
PIP_get	96	0.48	96	0.48
PIP_put	95	0.475	95	0.475

[i] These timings where performed using the non–instrumented kernel. Refer to DSP/BIOS II Sizing Guidelines for the TMS320C62x DSP, literature number SPRA667, for details regarding scaling and code size of DSP/BIOS.

[ii] For a 200MHz C6201 processor the CPU cycle period is 5 nanoseconds.

[iii] These measurements relate to the DSP/BIOS assembly language API, not the C language API.

[iv] Longest Interrupt latency occurs during SWI scheduling.

[v] Event is a SWI_post to beginning of periodic SWI execution.

3 Calculating System Performance

We can estimate the amount of DSP/BIOS overhead in terms of CPU load in any application. This is possible since all DSP/BIOS operations are visible to the developer. That is, the developer specifies which DSP/BIOS components and function calls to include into the application, either in the Configuration Tool, or explicitly in the code. The developer needs only to compute the sum of the components and frequency of occurrence to determine the overhead analytically. By using the RTA tools in CCS, developers may also directly measure the overhead on their specific hardware platform.

To estimate the overhead in DSP/BIOS applications, the developer must first identify all the DSP/BIOS components and API calls within the application. In our sample application audio I/O example, the DSP/BIOS components are:

- one HWI object mapped to the Audio
- one SWI object to do the processing (copy) operation and,
- two Data Pipes; one for input, one for output.

The component overhead in instruction cycles may be taken from the DSP/BIOS II Timings as listed in Table 1. To process a single buffer of audio data requires the total overhead of 1106 cycles on a C6000. The processing period is 4 ms, so the frequency of occurrence is 250 times per second. Therefore, the total number of cycles in one second, attributed to DSP/BIOS overhead running the audio thread on a C6000 DSP is 276,500 or 0.276500 MIPS. On a 200 MHz C6000 DSP, this equates to a 0.14% CPU load. Further explanation of this calculation is demonstrted in the DSP/BIOS II Technical Overview, SPRA646.

To calculate the amount of memory consumed by DSP/BIOS, the developer again needs to identify the DSP/BIOS components and API calls in the program. By summing the components, the developer can estimate the memory usage, both data and program. By using the memory map from the application, the exact amount can be determined.

In a similar fashion, developers can analytically determine the overhead attributed to DSP/BIOS. However, since it is the nature of software to change over time, analytical calculation can be tedious. The real-time analysis tool provided by DSP/BIOS allow developers to measure the overhead directly. Finally, since developers can chose the amount of DSP/BIOS to use and include in their applications, they have full control over the overhead.

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.