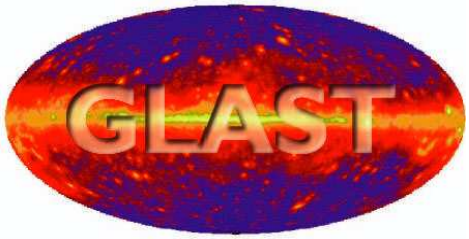

LAT Flight Software

CPU Monitor (MON) User Manual

Type:	User Manual
Version:	V0-0-0
Author:	Don May
Created:	16 February 2005
Updated:	10 March 2005
Printed:	10 March 2005

A description of how to use the LAT flight software CPU monitor functions found within the MON package.

Contents

Tables

0 Introduction

The MON package contains functions and tasks that monitor and report the performance of the RAD750 CPU. The types of performance metrics that the MON package can monitor include CPU loading, cache miss ratios, instructions executed per second, etc.

0.0 CPU Loading

At a rate of once per second, the MON package measures the loading of the RAD750 CPU and maintains a history of these loading measurements for the previous 60 seconds. MON provides a function that extracts and reports CPU loading statistics from this history data. These statistics include items such as average loading during the previous second and previous minute, as well as minimum and maximum per-second average loading during the previous minute.

0.1 Performance Monitor

The MON package utilizes the RAD750's performance monitor facility to measure the performance characteristics of the CPU. This facility includes registers within the CPU that count events such as processor clocks, cache misses, instruction dispatches, mispredicted branches, etc. MON provides a function that configures the performance monitor facility, allows it to run for a specified period of time, and reports the results via a user-defined callback function. Since only four events can be counted by the CPU at any one time, the configuration parameters supplied to this MON function include the types of events that should be counted.

1 Control

The MON package provides four functions that control its initialization and shutdown. This is similar to the interface provided by most flight software packages. These functions are listed here in the order in which they typically should be called:

- MON_initialize() Allocates resources and sets default configuration values.
- MON_start() Starts the tasks associated with the MON package.
- MON_stop() Stops the tasks associated with the MON package.
- MON_shutdown() Frees resources allocated by MON_initialize().

1.0 MON_initialize()

The MON_initialize() function allocates resources used by the MON package and sets configuration values to their default state. It requires no parameters and returns a MSG status code.

1.1 MON_start()

The MON_start() function starts two tasks which support the operation of the MON package. The user must call MON_initialize() to configure the MON package before calling MON_start().

The tasks started by this function include:

- IDLE task This task runs in the background with the lowest possible priority. Its purpose is to consume all unused CPU cycles and report the number of cycles that it has consumed.
- POLL task This task periodically samples the activity of the IDLE task and maintains a history of the percentage of CPU cycles consumed by the IDLE task. This task also handles requests for performance monitor statistics by configuring the RAD750 to count events and then reporting the results.

MON_start() requires two parameters, which are pointers to the *TASK_attr* attribute structures for the IDLE and POLL tasks. For proper operation of the IDLE task, these attributes should specify that the IDLE task run at the lowest possible priority. If a NULL pointer is passed for a task, a default set of attributes will be used for it.

Upon completion, MON_start() returns a MSG status code.

1.2 MON_stop()

The MON_stop() function stops the tasks started by MON_start(). It requires no parameters and returns a MSG status code.

1.3 MON_shutdown()

The MON_shutdown() function frees resources allocated by the MON package. If MON_start() has been called, however, the user must call MON_stop() before calling MON_shutdown(). This function requires no parameters and returns a MSG status code.

2 CPU Loading

The MON package provides a function that reports CPU loading statistics for the previous 60 seconds. This function acts as the interface to the statistics collected by the IDLE task and maintained by the POLL task.

2.0 MON_get_loading()

The MON_get_loading() function is the interface to the CPU loading statistics gathered by the MON package. It requires a single parameter, which is a pointer to a structure into which it should store the statistics. Upon completion, MON_get_loading() returns a MSG status code.

MON_get_loading() supplies CPU loading statistics within a structure that contains the following elements:

- avg_prev_sec — The average CPU loading during the previous second, as a percentage between 0 and 100.
- avg_prev_min — The average CPU loading during the previous minute, as a percentage between 0 and 100.
- min_prev_min — The minimum value of *avg_prev_sec* during the previous minute.
- max_prev_min — The maximum value of *avg_prev_sec* during the previous minute.

2.1 IDLE Task

The IDLE task consumes all CPU cycles that are not used by higher priority tasks. The task consists of two loops – one nested within the other. The inner loop simply performs 10,000 iterations of a register copy operation. The outer loop continuously runs the inner loop and increments a counter each time the inner loop completes its operation.

When started, the IDLE task runs the outer loop twice with interrupts disabled. The task records the amount of time needed to run the second pass of the outer loop as the baseline CPU time required for execution of the outer loop on an unloaded CPU. (The first pass of the outer loop execution is ignored since a portion of the time needed to execute it is spent loading the instruction cache).

After recording the baseline CPU time required for execution of its outer loop, the IDLE task continuously runs the outer loop until told to stop by the MON_stop() function. Once stopped, the IDLE task exits.

2.2 POLL Task and CPU Loading

The POLL task samples the IDLE task once per second to determine the number of times it has completed its outer loop. By comparing loop count values from two samples, the POLL task determines how many loop iterations were completed during the interval between the samples. The task multiplies this value by the baseline CPU time required for a single iteration of the idle outer loop to determine the amount of time that the RAD750 was idle during the sampling interval. The idle time is then divided by the length of the sampling interval, multiplied by 100, and subtracted from 100 to calculate the percentage of time the CPU was loaded during the interval.

3 Performance Monitor

The MON package provides functions that collect and report information about the performance of the RAD750. These functions act as the interface to the CPU's performance monitor facility.

3.0 MON_start_perf_mon()

The MON_start_perf_mon() function instructs the MON package to start counting CPU events. It requires a single parameter, which is a pointer to a structure that defines the types of events that should be counted. This function returns a MSG status code.

The CPU events that can be counted are defined by the RAD750 performance monitor facility. They are divided into four sets, where one event from each set can be counted at any given time.

The following events are common to all four sets:

- Number of processor cycles.
- Number of instructions that have completed (not including folded branches).
- Number of transitions from 0 to 1 of certain bits in the time base lower register.
- Number of instructions dispatched (0, 1, or 2 instructions per cycle).

Set 1 includes the following events:

- Number of **eieio** instructions completed.
- Number of cycles spent performing table search operations for the ITLB.
- Number of accesses that hit the L2.
- Number of valid instruction EAs delivered to the memory subsystem.
- Number of times an instruction address matches the address in the IABR.
- Number of loads that miss the L1 with latencies that exceed a threshold value.
- Number of branches that are unresolved when processed.
- Number of cycles the dispatcher stalls due to a second unresolved branch in the instruction stream.

Set 2 includes:

- Number of L1 instruction cache misses.
- Number of ITLB misses.
- Number of L2 instruction misses.
- Number of branches predicted or resolved not taken.
- Number of MSR[PR] bit toggles.
- Number of times reserved load operations completed.
- Number of completed load and store instructions.
- Number of snoops to the L1 and the L2.
- Number of L1 cast-outs to the L2.
- Number of completed system unit instructions.
- Number of instruction fetch misses in the L1.
- Number of branches allowing out-of-order execution that resolved correctly.

Set 3 includes:

- Number of L1 data cache misses.
- Number of DTLB misses.
- Number of L2 data misses.
- Number of taken branches, including predicted branches.
- Number of transitions between marked and unmarked processes while in user mode (i.e. the number of MSR[PM] toggles while the processor is in user mode).
- Number of store conditional instructions completed.
- Number of instructions completed from the FPU.
- Number of L2 castouts caused by snoops to modified lines.
- Number of cache operations that hit in the L2 cache.
- Number of cycles generated by L1 load misses.
- Number of branches in the second speculative stream that resolve correctly.
- Number of cycles the BPU stalls due to LR or CR unresolved dependencies.

Set 4 includes:

- Number of L2 castouts.
- Number of cycles spent performing table searches for DTLB accesses.
- Number of mispredicted branches.
- Number of transitions between marked and unmarked processes while in supervisor mode (i.e. the number of MSR[PM] toggles while the processor is in supervisor mode).
- Number of store conditional instructions completed with reservation intact.
- Number of completed **sync** instructions.
- Number of snoop request retries.
- Number of completed integer operations.
- Number of cycles the BPU cannot process new branches due to having two unresolved branches.

To measure, for example, the CPU instruction cache miss ratio, MON_start_perf_mon() would be configured to count the “number of valid instruction EAs delivered to the memory subsystem” from set 1 and the “number of L1 instruction cache misses” from set 2. The ratio of these two values would then be calculated to determine the instruction cache miss ratio during the collection period.

3.1 MON_stop_perf_mon()

The MON_stop_perf_mon() function instructs the MON package to stop counting CPU events. It requires a single parameter, which is a pointer to a structure into which the collected event counter values should be stored. This function returns a MSG status code.

3.2 MON_collect_perf_stats()

The MON_collect_perf_stats() function instructs the MON package to count CPU events for a specified period of time. It requires five parameters and returns a MSG status code. The parameters for this function specify:

- the types of events to count,
- the length of time to count the events,
- a pointer to a structure into which the collected event counter values should be stored,
- a pointer to a user-defined function to call when the collection is complete, and
- a user-defined value to pass to this callback function.

4 Error Reporting

The MON package functions support the MSG status reporting system. The following message codes are defined.

Table 1 – MON Package MSG Codes

MSG Code	Description	Parameters (if any)
SUCCESS	Function succeeded.	
ALLOCMEM	Unable to allocate memory.	• Size of allocation request, in bytes. • String describing the purpose of the allocation.
ALLOCMTX	Unable to allocate a mutex.	
ALLOCWUT	Unable to allocate a wake-up timer.	
BADSTATE	The requested operation is not valid in the current state of the MON package.	• Bit field identifying the valid states for the operation. • Current MON state.
FREEMEM	Unable to free memory.	
FREEMTX	Unable to destroy a mutex.	
FREEWUT	Unable to destroy a wake-up timer.	Status value returned by WUT_destroy().
LOCKMTX	Unable to lock a mutex.	
NOTRDY	Unable to perform the requested operation, probably because the operation is not valid in the MON package's current state.	
POLSLEEP	A task is unable to sleep.	
PRFRUN	Unable to start collecting performance statistics because the MON task is already collecting statistics.	

TSKJOIN	A task couldn’t be stopped.	String identifying the task.
TSKSPAWN	A task couldn’t be started.	String identifying the task.
LOCKMTX	Unable to unlock a mutex.	