**International Journal of Software Engineering and Technology (IJSET)**
Vol. 1, No. 1, April 2016, pp. 41~58
ISSN: 2032-4038


# Embedded Software Engineering Approach to Implement BCM5354 Processor Performance

**Varuna Eswer*, Sanket Dessai****
*Founder &CEO, Eudaemonic Systems
** Department of Computer Enginering, M.S.Ramaiah School of Advanced Studies, Bengaluru, India

**Article Info**

***Article history:***

Received Jun 12th, 2015
Revised Aug 20th, 2015
Accepted Aug 26th, 2015

***Keyword:***

BCM5364
MIPS32
OS
TLB
Software Engineering

**ABSTRACT**

Efficiency of a processor is a critical factor for an embedded system. One of the deciding factors for efficiency is the functioning of the L1 cache and Translation Lookaside Buffer (TLB). Certain processors have the L1 cache and TLB managed by the operating system, MIPS32 is one such processor. The performance of the L1 cache and TLB necessitates a detailed study to understand its management during varied load on the processor. This paper presents an implementation to analyse the performance of the MIPS32 processor L1 cache and TLB management by the operating system (OS) using software engineering approach. Software engineering providing better clearity for the system developemt and its performance analysis.In the initial stage if the requirement analysis for the performance measurment sort very clearly,the methodologies for the implementation becomes very economical without any ambigunity.In this paper a implementation is proposed to determine the processor performance metrics using a software engineering approach considering the counting of the respective cache and TLB management instruction execution, which is an event that is measurable with the use of dedicated counters. The lack of hardware counters in the MIPS32 processor results in the usage of software based event counters that are defined in the kernel. This paper implements a subset of MIPS32 processor performance measurement metrics using software based counters. Techniques were developed to overcome the challenges posed by the kernel source code. To facilitate better understanding of the implementation procedure of the software based processor performance counters; use-case analysis diagram, flow charts, screen shots, and knowledge nuggets are supplemented along with histograms of the cache and TLB events data generated by the proposed implementation. Twenty-seven metrics have been identified and implemented to provide data related to the events of the L1 cache and TLB on the MIPS32 processor. The generated data can be used in tuning of compiler, OS memory management design, system benchmarking, scalability, analysing architectural issues, address space analysis, understanding bus communication, kernel profiling, and workload characterisation.



***Corresponding Author:***

Varuna Eswer,
Mysore University,
India.
Email: sanketdessai0808@gmail.com

## 1. INTRODUCTION

As Performance measurement is to arrive at the count of the desired event occurring in the system during execution. Processor performance measurement is about measuring the states and events related to the quantum of work done over a period by the processor. The workload on the processor involves computation

and movement of data in a defined sequence, and the sequence involves the usage of subsystems such as cache, pipeline, memory, peripherals and so on. The performance measurement can be achieved with the use of either the hardware or the software counters or both. The hardware counters utilises the physical counters provided by the processor designer, and is loaded with the counter values for measurement from the executing process (operating system or an application) [1]. The software counter on the other hand necessitates code modifications in the executing process that counts the occurrence of an event associated with the performance measurement. The advantage of hardware counters is that it is generally non-intrusive on the instruction execution cycles, while the software counters does require additional instruction cycle for counting the event. The disadvantage with the hardware counters is that the physical counters are limited in number, while the software counters are not. The focus of the hardware counters is fine-tuning either the operating system or the executing process in the identified bottlenecks, while the software counters can be utilised either for general-purpose performance measurement or for specifically measure a bottleneck. To measure the appropriate performance criteria's it is necessary to understand and analyse the requirement analysis properly. It has to be very clear what and how exactly to be measured. In this paper it had been attempted to analyse the requirement analysis for the processor performance measurement.

## 2. PROCESSOR AND PERFORMANCE MEASUREMENT

The study of a concept would necessitate the understanding of the context associated with the MIPS32 processor architecture, and this chapter develops the context of the processor pipeline and cache, performance measurement, data IO methods from the processor implementation board and the setup of the development system.

### 2.1. Pipeline and Cache in MIPS32 Architecture

The MIPS architecture are primarily RISC based processors are available in 32, and 64-bit addressing and operations mode [2]. The processors find its usage in range of environment: workstations to embedded systems, executing on a host of OS that are proprietary or OS based. In order to speedup the execution of an instruction, the MIPS processor depends on the pipelines and caches. A pipeline is a technique that is utilised to divide a work (instruction or data) into an ordered sequence enabling quicker turnaround time for executing the work. Generally the work is pre-fetched to reduce any latency of loading the processor with the work, and the advantages of MIPS architecture is that the work for execution are pipelined, and do not interlock the stages of executing a work. The pipeline in MIPS architecture is five stages as seen in Figure 1.

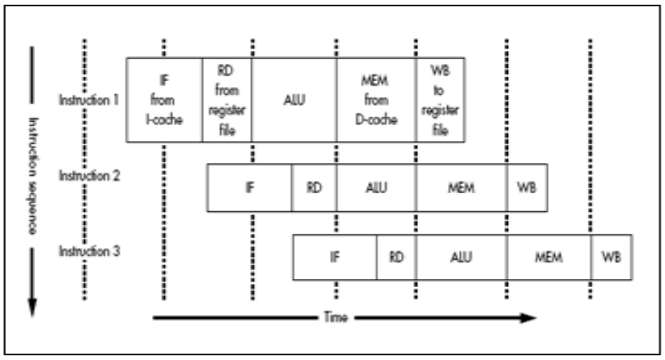


Figure 1. MIPS Five-stage Pipeline [7]

A cache is a fast access memory between the processor and the memory, and hold copies of instruction and data residing in the memory [2]. The cache provides the data for the CPU as required, and an occurrence of a request by the CPU for data that does not exist in the cache, then, the cache is refreshed by invalidating the set of data.There will be situations where the CPU updates the contents of the cache, thus, marking it as write back to enable the cache update the corresponding memory location associated with the data. Seen in Figure 1, the Icache is the instruction brought from the instruction cache, while the Dcache is the data from the cache for the corresponding instruction provided to the CPU for execution. The cache has a

critical role in ensuring that the CPU is not starved of either instruction or data, thus contributing to the efficiency of the processor. The MIPS architecture is designed to have separate Icache and Dcache to enable faster movement of data requested by the CPU [2].

### 2.2. Performance Measurement

Processor performance measurement is about gathering data of the events, state transitions, and data movement occurring in the processor. The event/data movement results with the execution of the instructions; hence, the responses of the cache and TLB on the data request by the CPU takes a slightly higher priority than the other operations of the CPU. The performance measurement can be achieved with the use of hardware or software based counters. Indicated in the Figure 2 is an illustration of hit-miss software counter [4]. The classification of metrics is broadly segregated into two-categories: base and derived. The base metrics are the direct or raw counting of the monitored events, while the derived metrics are those that are arrived at by a combination of two or more metrics that either could be of the category base or derived or a combination of both.

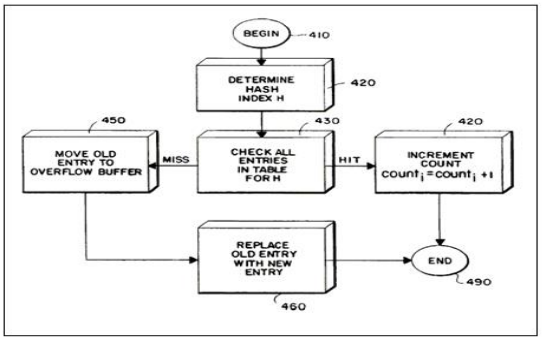


Figure 2. Software Counter Implementation [12]

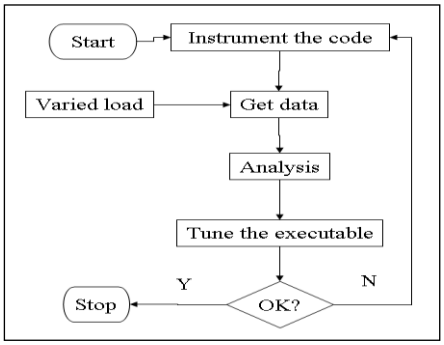


Figure 3. Flow Chart for Performance Measurement

The data generated by the counters can either be archived for long or short-term duration. The extent of archival depends on the quantum of data for analysis. The analysis begins with the source code modification for metric collection through either software or hardware based counters. The design of the performance measurement framework will necessitate a high amount of metric data from the system, generated using varied load. The analysis on the metric data provides the direction for further code instrumentation or reduction in the code instrumentation. This forms the basis for arriving at the measurement framework. The code instrumentation, metric data collection and the analysis is iterative in process; as indicated in Figure 3; until the framework is frozen. This sets the stage for performance metric measurement over a period for achieving the stated objectives in the defined performance measurement framework.

## 3. REQUIREMENT SPECIFICATIONS

The functional requirements are:

1. Data collected to be available in the ASCII format accessible for data correlation applications
2. Counter values should be available under the */proc* file system in the file *cpuinfo*
3. Categories of performance measurement is preferred in the area of the hit-miss-refresh cycle related to the:
   a. Dcache
   b. Icache
   c. Scache
   d. TLB
4. All operations defined for the cache, and the TLB are to be covered
5. Counters values can overflow; hence, the use of a separate variable to count the overflows for each metric will be essential
6. Data structure to be placed in the architecture specific *.../srclinux/linux//include/asm-mips* directory
7. Metric update routines to be placed in the architecture *.../src/linux/linux/arch/mips/kernel* directory
8. Modification to the source code should ensure minimal change in the firmware footprint size
9. The ability to compile the kernel without the metric collection has to be provided

10. Efficiency of the introduced code is *not* the goal as the focus is to get as much as data from the cache and TLB management routines

## 4. SYSTEM ANALYSIS

The major operations on the cache are either *write-back* or *invalidate*. The write-back operation is used when cache has been updated by the CPU; hence, necessitates the corresponding memory update. The invalidate operation is chosen to access a fresh set of data from the memory. The write-back and invalidate operations is applied on the Icache, Dcache and Scache lines. The flow of cache initialisation is first done on the Icache, followed by the Dcache. Analysing the source code, the cache operations for the BCM5354 processor are as defined by [20] are:

```
#define Index_Invalidate_I      0x00
#define Index_Writeback_Inv_D   0x01
#define Index_Writeback_Inv_SD  0x03
#define Hit_Invalidate_I    0x10
#define Hit_Invalidate_D    0x11
#define Hit_Invalidate_SD   0x13
#define Hit_Writeback_Inv_D 0x15
#define Hit_Writeback_Inv_SD    0x17
#define Hit_Writeback_I     0x18
#define Hit_Writeback_D     0x19
#define Hit_Writeback_SD    0x1b
```

The Figure 4 indicates the use case developed is in reference to the source code to determine the processor performance [20]. The actors that are hardware based are the *Dcache*, *Scache*, *Icache*, and the *TLB*. The actor *Kernel* is software based managing the on-processor caches [4], [5]. The association between the actor *Kernel* and the actors *Dcache*, *Icache*, *Scache*, and the *TLB* are unidirectional as the *Kernel* is waiting on the new set of instruction or data or the virtual address mapping is available in the respective segment and/or line for execution of the scheduled task.

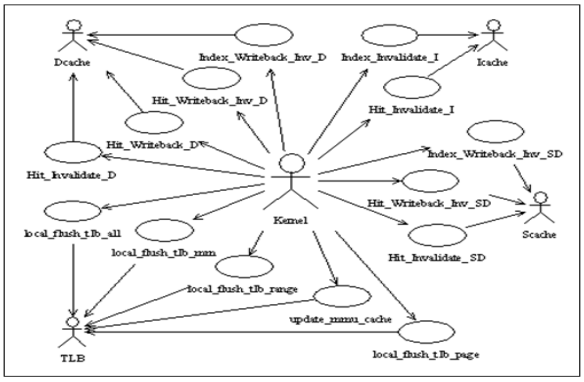


Figure 4. Use cases developed to determine the Processor Performance

a. *Use case name*: Hit_Invalidate_D
*Actors*: Kernel, Dcache
*Purpose*: Invalidate the data cache line.
*Overview*:The data provided by the cache is stale; hence, the next fetch will bring in a new set of data from the virtual address space for the *Kernel* to continue execution of an instruction.

b. *Use case name*: Hit_Writeback_D
*Actors*: Kernel, Dcache
*Purpose*: Update the memory location with the updated data in the cache.
*Overview*: The *Kernel* has modified the data available in the *Dcache*; hence, the next operation will update the corresponding virtual memory location for the data

c. *Use case name*: Hit_Writeback_Inv_D
*Actors*: Kernel, Dcache
*Purpose*: Update the memory location with the data in the *Dcache*, and invalidate the cache.
*Overview*:The *Kernel* has modified the data in the *Dcache*; hence, necessitates an update of the corresponding virtual memory location. Post update of memory location, the entire cache is invalidated so that the next cycle ensures a fresh set of data for the instruction execution.

d. *Use case name*: Index_Writeback_Inv_D
*Actors*: Kernel, Dcache
*Purpose*: Invalidate all *Dcache* lines after updating the respective memory locations.
*Overview*:The *Kernel* requests the data in all the *Dcache* lines to be updated in the corresponding memory locations prior to invalidating the lines. The *Kernel* requires a new set of data in the cache lines to continue execution.

e. *Use case name*: Hit_Invalidate_I
*Actors*: Kernel, Icache
*Purpose*: Invalidate the *Icache*.
*Overview*:The *Kernel* requires a new set of instruction to continue execution. This can occur during a context switch, or during execution of a process, that has a branch condition execution.

f. *Use case name*: Index_Invalidate_I
*Actors*:Kernel, Icache
*Purpose*: Invalidate the entire *Icache* line to receive a new set of instruction.
*Overview*: The *Kernel* is in a situation that has necessitated a new set of instructions on the entire cache line; hence, the next instruction fetch will cause the entire cache line to be filled with instruction fetched from the memory. The contents of the *Icache* line is not updated in the corresponding memory locations.

g. *Use case name*: Hit_Invalidate_SD
*Actors*: Kernel, Scache
*Purpose*: Invalidate the *Scache*.
*Overview*: The requested data by the *Kernel* was not available either the primary or the secondary cache, and has not been marked dirty. The next fetch of data will cause a new set of data will be brought from the corresponding memory location.

h. *Use case name*: Hit_Writeback_Inv_SD
*Actors*: Kernel, Scache
*Purpose*: Update the data in the *Scache* lines prior to invalidate of the lines.
*Overview*: The *Kernel* has modified the contents of the single *Scache* that necessitates an update in the corresponding memory locations prior to invalidating the line for ensuring new set of data is available on the next data fetch cycle.

i. *Use case name*: Index_Writeback_Inv_SD
*Actors*: Kernel, Scache
*Purpose*: Update the data in the *Scache* lines prior to invalidating all the *Scache* lines.
*Overview*: The *Kernel* has modified the contents of the *Scache* lines; hence, prior to invalidating the lines update the contents of the lines in the corresponding memory locations.

j. *Use case name*: local_flush_tlb_page
*Actors*: Kernel, TLB
*Purpose*: Flush the page associated with the virtual memory for the current context.
*Overview*: The *Kernel* is in a situation where the requested virtual address map for the current context is not available in the *TLB* page; hence, the page is rebuilt with the correct virtual address mapping.

k. *Use case name*: update_mmu_cache
*Actors*: Kernel, TLB
*Purpose*: Update the *TLB* with the correct PTE in the current context.
*Overview*:The MM is in a situation where the earlier *TLB* entries viewed by the *Kernel* have changed due to a possible context switch, resulting in a necessity to remap the PTE for the translations in the *TLB*.

l. *Use case name*: local_flush_tlb_range
*Actors*:Kernel, TLB
*Purpose*: Refresh the *TLB* entries.
*Overview*:The MM in the *Kernel* was not able to find the appropriate PTE; hence, the entire *TLB* is remapped.

m. *Use case name*: local_flush_tlb_mm
*Actors*: Kernel, TLB

*Purpose*: Recreate a new MM context for the *TLB*.

*Overview*: The current context of the *TLB* has changed; hence, a new context of the memory map is created for the entire *TLB* by recreating a new ASID.

n. *Use case name*: local_flush_tlb_all

*Actors*: Kernel and TLB

*Purpose*: Reorder the *TLB*

*Overview*: The MM will ensure that the ordering of the *TLB* will occur to aid the *Kernel* to get a new context in the KSeg0.

## 5. SOFTWARE DESIGN CONSIDERATION

The listed cache operations [20] are utilised by multiple routines on the caches to flush either the lines or the KSeg0; hence, it is essential to trace the function that used the defined cache operations using unique metrics. The arrived list of metrics is from the perspective of generating the complete picture of the cache operations for the MIPS32 processor. Analysing the source code; [5], [16], [20]; for the defined function calls for the caches; I, D and S; along with the TLB operations for the MIPS32 implementation is categorised in the Table 1. The cache operations as visualised from Table 1, the operations can be repeated on the lines or the ways or on KSeg0.

The base metric requires a roll over counter as it helps to track the metric data over an extended period. A high rate of activity of the processor will cause a rollover of the metric. The choice of the data type for the metrics; base and the roll over; are *unsigned int*. The data type of the base and the rollover metrics will necessitate a change to an *unsigned long* if need be, with the decision based on the activity of the processor. The designed base metrics data structure is available in Appendix-A.

The metric update is organised on the defined cache operations [20]; hence, the metric computation is switch statement based on the operations. The routines to update the metrics are defined as:

*void update_cache_metric (int type, int cm_ops);*
*void update_tlb_metric (int tlb_fn);*

The parameter *cm_ops* is the operation defined for the cache, while *type* indicates the operation of the cache on either the line, or the way, or the Kseg0 in the function *update_cache_metrics*. The parameter *tlb_fn* indicates the operation on the TLB. The parameters *type* and *tlb_fn* are defined with a unique value in the file *.../src/linux/linux/include/asm/cache_perf_mips32.h*, and is indicated as follows:

```
#define unroll_c    0xe0    /* cache unroll */
#define line_c      0xe1    /* cache line */
#define kseg0_c     0xe2    /* kseg0 address */
#define ways_c      0xe3    /* mip cache ways */
#define page_c      0xe4    /* cache page */
#define pline_c     0xe5    /* protected cache line */
#define Fill_Icache_line    0xe6    /* fill icache line */
#define lf_tlb_all  0xf0    /* local tlb flush all */
#define lf_tlb_mm   0xf1    /* local flush tlb mm struct */
#define lf_tlb_rng  0xf2    /* local flush tlb range */
#define lf_tlb_pg   0xf3    /* local flush tlb page */
#define up_tlb_mmu  0xf4    /* update tlb mmu */
```

An example of the method of function call for metric update from the functions indicated under the first column of Table is:

*update_cache_metric (line_c, Hit_Invalidate_I);*
*update_tlb_metric (lf_tlb_mm);*

The metric data for further analysis will be available in the OS provided file */proc/cpuinfo*, individually categorised for the Dcache, Icache, Scache, and TLB. The metric data can be read from the */proc/cpuinfo* file as required, each read provides the current data of the metrics. The format of the data in the */proc/cpuinfo* is indicated below, with the sequence of the metric display is indicated in Table 2:

```
dcache metrics: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
icache metrics: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
scache metrics: 0,0,0,0,0,0,0,0,0,0,0,0
tlb metrics: 0,0,0,0,0,0,0,0,0,0
```

The Appendix-B has the ASCII based screen capture of the file */proc/cpuinfo* from the router indicating the collected metric values.

The Figure 5 indicates a method of utilising the generated performance metric data to for analysis purpose. The metric data read from the */proc/cpuinfo* file is extracted into a structure and is placed in the

respective metric variables as indicated in Table 2. The difference between the current and the previous interval metric data is provided to the data analysis application. The duration between the current and the previous interval is controlled through the *Sleep* routine.

Likewise, the data analysis application can be developed to provide a combination of the base and derived metrics; as indicated in Equation 1; to arrive at a set of data suitable for a deep dive analysis of the processor performance. Figures 9 through 12 provides a set of histogram for the Dcache, Icache, Scache, and TLB generated using base metrics data over a period of one-hundred and ten seconds at an interval of five seconds indicates the use case developed is in reference to the source code to determine the processor performance [20]. The actors that are hardware based are the *Dcache*, *Scache*, *Icache*, and the *TLB*. The actor *Kernel* is software based managing the on-processor caches [4], [5]. The association between the actor *Kernel* and the actors *Dcache*, *Icache*, *Scache*, and the *TLB* are unidirectional as the *Kernel* is waiting on the new set of instruction or data or the virtual address mapping is available in the respective segment and / or line for execution of the scheduled task.

Table 1. Mapping of Function Calls Operation and Metrics

| Listed function name | Cache operation | Metrics to be updated |
|---|---|---|
| flush_icache_line_indexed | Index_Invalidate_I on ways | i_way |
| blast_icache | Index_Invalidate_I and cache unroll of kseg0 | i_unroll_kseg0 |
| blast_icache_page_indexed | Index_Invalidate_I and cache unroll on ways | i_unroll_way |
| flush_icache_line | Hit_Invalidate_I of line | i_line_flush |
| protected_flush_icache_line | Hit_Invalidate_I of line | i_pline_flush |
| blast_icache_page | Hit_Invalidate_I and cache unroll on page | i_unroll_page |
| flush_dcache_line_indexed | Index_Writeback_Inv_D on ways | d_way |
| blast_dcache_page_indexed | Index_Writeback_Inv_D and cache unroll of ways | d_unroll_way |
| blast_dcache | Index_Writeback_Inv_D and cache unroll of kseg0 | d_unroll_kseg0 |
| flush_dcache_line | Hit_Writeback_Inv_D on line | d_line_flush |
| blast_dcache_page | Hit_Writeback_Inv_D and cache unroll on page | d_unroll_page |
| invalidate_dcache_line | Hit_Invalidate_D on line | d_invld_ln |
| protected_writeback_dcache_line | Hit_Writeback_D on line | d_writeback |
| flush_scache_line_indexed | Index_Writeback_Inv_SD on ways | s_way |
| blast_scache_page_indexed | Index_Writeback_Inv_SD and cache unroll on page and ways | s_unroll_pg_ways |
| blast_scache | Index_Writeback_Inv_SD and cache unroll on kseg0 | s_unroll_kseg0 |
| invalidate_scache_line | Hit_Invalidate_SD | s_invld_ln |
| flush_scache_line | Hit_Writeback_Inv_SD on line | s_line_flush |
| blast_scache_page | Hit_Writeback_Inv_SD and cache unroll on page | s_unroll_page |
| fill_icache_line | Fill_Icache_line | i_fill |
| local_flush_tlb_all | | tlb_lflush_all |
| local_flush_tlb_mm | | tlb_lflush_mm |
| local_flush_tlb_range | | tlb_lflush_rng |
| update_mmu_cache | | tlb_updt_mmu |
| local_flush_tlb_page | | tlb_lflush_pg |

Table 2. Metric Display Sequence in cpu/proc/info

| Cache Type | Metric Sequence |
|---|---|
| dcache metrics | d_way, d_way_roll, d_invld_ln, d_invln_roll, d_writeback, d_wb_roll, d_blast, d_blast_roll, d_unroll_kseg0, d_ukseg_roll, d_unroll_way, d_uw_roll, d_line_flush, d_lf_roll, d_unroll_page, d_up_roll |
| icache metrics | i_way, i_way_roll, i_blast, i_blast_roll, i_unroll_way, i_uw_roll, i_unroll_kseg0, i_ukseg_roll, i_line_flush, i_lf_roll, i_pline_flush, i_plf_roll, i_unroll_page, i_up_roll, i_fill, i_fill_roll |
| scache metrics | s_invld_ln, s_invln_roll, s_way, s_way_roll, s_unroll_kseg0, s_ukseg_roll, s_unroll_pg_ways, s_upw_roll, s_line_flush, s_lf_roll, s_unroll_page, s_up_roll |
| tlb metrics | tlb_lflush_all, tlb_lfa_roll, tlb_lflush_mm, tlb_lfmm_roll, tlb_lflush_rng, tlb_lflrng_roll, tlb_lflush_pg, tlb_lfpg_roll, tlb_updt_mmu, tlb_upmmu_roll |

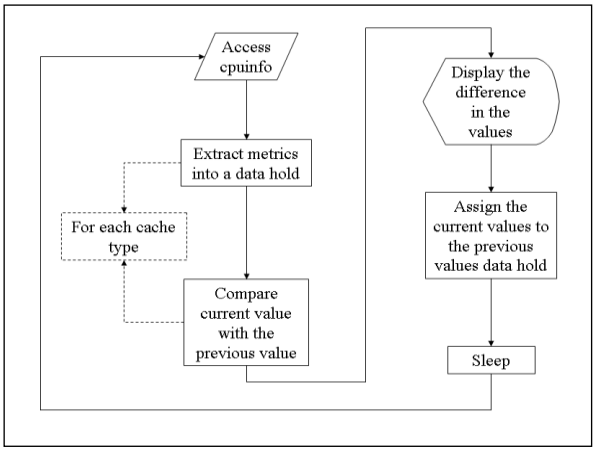


Figure 5. Flow Chart for Metric Analysis Applications

The Linux source code is split into two major sections: architecture independent and architecture dependent. The NETGEAR bundled source code has a third section that is router board specific. The build process involves compiling the architecture specific code, followed by Linux specific; architecture independent; code compile that results in the kernel image, then followed by compiling the router board specific that results in the firmware image for the NETGEAR WGR614v9 router. The procedure to setup and build the firmware image for the NETGEAR WGR614v9 router is:

1. Download and install the *libstdc++.so.5* OS development library.
2. Download the cross compiler tool-chain TOOLSOURCE_2004_03_31.tgz from *ftp://ftp.gpl-devices.org/pub/vendors/Belkin* [12] and install the tool-chain in the home directory of the user.
3. Create a symbolic link */opt/brcm* to the tool-chain directory *..../org/tools/brcm* that is available in the user home directory.
4. Download from the internet and install the utility *trx* in the directories */opt/brcm/hndtools-mipsel-linux-3.2.3/bin*, and */opt/brcm/hndtools-mipsel-uclibc-3.2.3/bin*. Ensure the utility *trx* has the execute permission for the owner, group, and the user.
5. Ensure the path to */opt/brcm/hndtools-mipsel-linux-3.2.3/bin* and */opt/brcm/hndtools-mipsel-uclibc-3.2.3/bin* directories are available in the shell environment variable *PATH*.
6. Download the NETGEAR WGR614v9 source code;
   example *WGR614v9-V1.2.6_18.0.17WW_src.tar.bz2.zip*; available at the website *ftp://downloads.netgear.com/files/GPL* [28], and install the code in a suitable directory under the home directory of the user.
7. Clean the existing object files, and the kernel image *vmlinux* under the Linux and the router sections of the source code tree using the followings indicated steps:
   a. *cd .../src/router*
   b. *make clean*

c. *make router-clean*
d. *cd ../linux/linux*
e. *make clean*

8. Build the Linux kernel image from the source code directory *.../src/linux/linux* using the following steps to generate the MIPS32 kernel image *vmlinux* as indicated in Figure 6:
   a. *make dep*
   b. *make*
9. Build the router code in the directory *.../src/router* using the following steps:
   a. *make*
   b. *make install*
10. The WGR614v9 router firmware upgrade image file will be created in the directory *.../src/router/mipsel-uclibc* in the file beginning with the name *WGR614v9*, and ending with the extension *chk* as indicated in Figure 7. An example of the firmware file name is *WGR614v9-12051706.chk* as seen in Figure 7.

The hardware setup is a router board with an implementation of MIPS32 core by BROADCOM processor BCM5354, and is indicated in Figure 8.

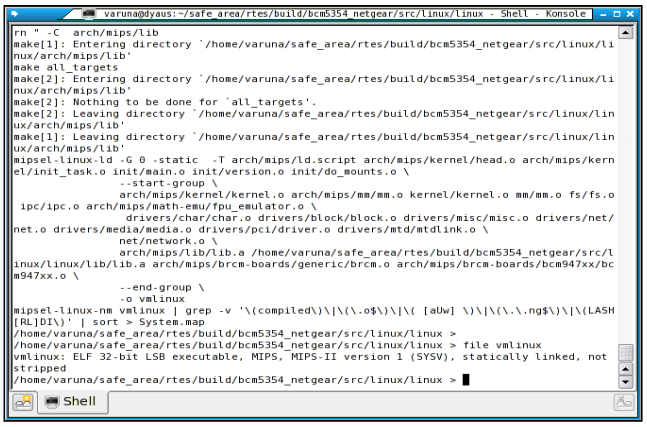


Figure 6. Screenshot of MIPS32 kernel build

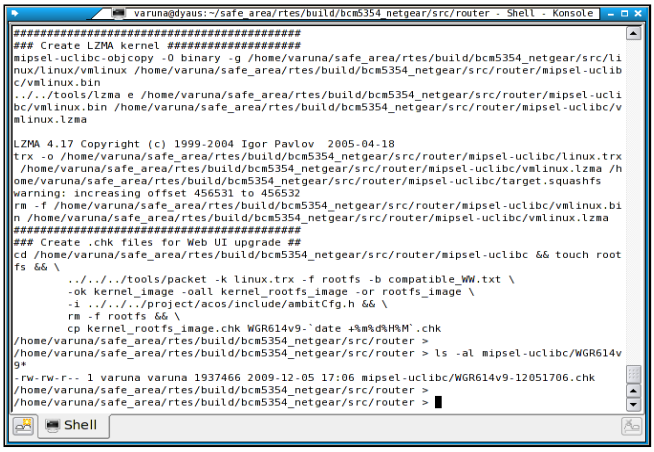


Figure 7. Screenshot of NETGEAR router image build

## 6. PSEUDO-CODE AND IMPLEMENTATION PROCEDURE

a. The performance metric data collection is implemented under the architecture specific memory management routines available in the locations *.../src/linux/linux/arch/mips/mm*, and the *.../src/linux/linux/include/asm-mips* directories, and the functions listed in Table 1 is available in the indicated directories. The pseudo-code for collecting the performance metrics is indicated in the following steps 1 through 5:

b. Call the function for metric update from the function calls associated with the cache and TLB management, along with the parameters of cache/TLB operation and the type of the operation; either on the ways, or the line, or the KSeg0.
c. Update the associated metric for the functions listed in Table 1.
d. If the metric counter overflows; wraps to the value zero; then increment the corresponding rollover metric counter.
e. Print the values of all the metrics in the */proc/cpuinfo* file.
f. Repeat the steps 1 through 4 for each of the function call listed in Table 1.
g. The method of implementation of the pseudo-code is indicated below:
h. Locate the section in the source code handling cache and TLB function calls from Linux kernel and memory management routines available in the directory .../*src/linux/linux//kernel* and .../*src/linux/linux/mm* directories.
i. Locate the section under the architecture specific source code handling the kernel and the memory management, and identify the section handling the cache and TLB management located under the directories *.../src/linux/linux//arch/mips/mm* and *.../src/linux/linux/arch/mips/kernel*. The associated architecture specific header file is located under the directory *.../src/linux/linux/include/asm-mips*. Specific files that will be used are:
j. *.../src/linux/linux/include/asm-mips/mips32_cache.h*
k. *.../src/linux/linux/arch/mips/kernel/Makefile*
l. *.../src/linux/linux/arch/mips/kernel/proc.c*
m. *.../src/linux/linux/arch/mips/mm/tlb-r4k.c*
n. Define the header file listing the data structure; as seen in Appendix-A; in a file under the .../*src/linux/linux/include/asm-mips* directory. Example: *cache_perf_mips32.h*
o. Define the routines to update and display the metric counters; as listed in Section 5; in a file under the .../*src/linux/linux./arch/mips/kernel* directory, example: *cache_perf_proc.c*. Ensure to initialise the metrics data structure to zero.
p. Modify the .../*src/linux/linux/arch/mips/kernel/Makefile* to include the resulting object file generated in step 4].
q. Modify the architecture specific listed cache operations; listed in Table 1; in the source code file *.../src/linux/linux//include/asm-mips/mips32_cache.h* and *.../src/linux/linux/arch/mips/mm/tlb-r4k.c* to call the metric update functions; defined in step 4]; along with the necessary parameters. The parameters are listed in Table 1.
r. Call the metric display function; created in step 4]; from the file *.../src/ Linux/linux//arch/mips/kernel/proc.c* to display the data in the */proc/cpuinfo* file.
s. Validate the changes on the hardware by building the firmware image for the NETGEAR WGR614v9 router based MIPS32 processor implementation along with the changes in the source code.

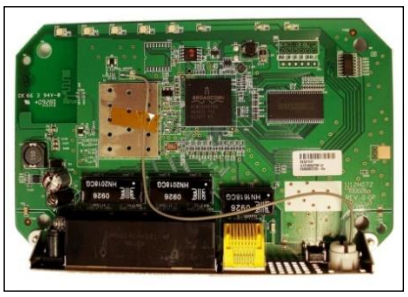

Figure 8. NETGEAR WRR614 Router Board

## 7. RESULTS

The result of processor performance measurement involves in understanding the metrics and makes its interpretations and concludes the results associated with these interpretations. Based on these interpretations wherever required modify the code and analyse the performance to understand the performance measurement.

### 7.1. Metric Interpretation

Consider an example of the cache operation *Index_Invalidate_I*, and as seen in Table 1, with the operations performed on the Icache line, Icache ways and on the KSeg0. In order to get an exact figure of the number of times the operation Index_Invalidate_I was performed, the method is:
*Number of Index_Invalidate_I =*

$$i\_way + i\_unroll\_kseg0 + i\_unroll\_way \quad (1)$$

Likewise, for each defined operation of the cache and TLB, the corresponding metrics indicated in Table 1 are to be added as indicated in Equation 1. The combination of metrics yields a derived metric, while the value of the individual metrics is termed as the base metric. The Column 3 of the Table 1 indicates the base metrics. The data provided by the metrics; base and derived; can be utilised to draw a histogram tracing the processor cache activity over a period.

The data generated by the metric analysis tool zmet has been transformed into histograms; Figures 9, 10, 11 and 12; for the chosen set of metrics for the Dcache, Icache, Scache, and the TLB. The y-axis for the histograms indicates the number of events, while the x-axis is the time in seconds. The data collected for the histogram has been every five seconds, over a period of one-hundred and ten seconds. The commands that were executed for data collection are standard OS commands that for displaying the contents of the directory and files under the */proc* file system, etc. In order to get a better perspective of the effect of the commands, consider an example of the command: *cat /proc/cpuinfo*. The execution of the command has the following indicative steps:

1. The shell spawns a new process by creating a process table entry, and copies the file descriptors. The process will be associated with the cat application.
2. Scheduler places the created process for execution.
3. The code for cat is accessed, brought into the memory from the file system.
4. Executing the code, the file name is parsed. In this case, the file is /proc/cpuinfo.
5. The file table entry is accessed to check if the file is a directory, if yes, exit.
6. Open the file for reading.
7. Read the line until end of line mark and store in the buffer.
8. Access the character device driver for console terminal.
9. Open the device for writing.
10. Get back to the file read operation, and now call the print routine to output data to the character device file.
11. Repeat the Steps 7 through 10 until end of file.
12. Release the system resources occupied for reading the file /proc/cpuinfo.
13. Release the system resources occupied by the cat application.
14. Release the system resources associated with process table entry.

The Steps 1 through 14 has multiple instructions each that either can be in the memory or on the flash file system of the router. The possibility of the instruction being in the memory either can be due to an earlier execution of the same instruction or has been pre-fetched. The indicated steps when viewed for the perspective of the cache and TLB usage, non-availability of the instruction in the memory will cause a flush of the Icache. This operation will cause a Dcache flush, and if need be, a flush of the TLB. Referring to the histograms in Figure 9, 10, 11 and 12, the first five-seconds of data collection has seen a high activity of the Dcache and Icache when compared to the TLB and Scache. This corresponds to the pre-fetch of the instruction into the memory, then to the Icache as seen in Figure 1.

The execution of the instruction will cause a fetch of the data from the memory into the Dcache as seen in Figure 3. Prior to loading of the Icache or Dcache, the entire KSeg0, or the ways or the lines are flushed, with the simultaneous update of the TLB. The each activity on the Icache, Dcache, Scache and TLB is a measurable event; hence, the corresponding metrics are updated. The flush activity either can be a writeback or invalidate of the respective caches. The four histograms; Figures 9,10,11 and 12 have to be visualised simultaneously in order to arrive at the cache and TLB activity on the processor.

### 7.2. Analysis of the Code Modifications

The lines of code that were added into the stock source code [3] have been listed in the Table 3. The files *mips32_cache.h*, *Makefile*, *proc.c*, *tlb-r4k.c*, *Config.h*, *applets.h* and the *usage.h* were part of the source code tree, while the files *cache_perf_proc.c*, *cache_perf_mips32.h* and *zmet.c* were added into the source code tree as part of the implementation of the cache and TLB performance measurement metrics for MIPS32 architecture. The corresponding location of modification and introduction of the new files has been indicated

in Table 3. The total number lines that were added into the source code tree has been 1009 (one-thousand and nine), counted without the comments or introduced blank lines for formatting of the code.

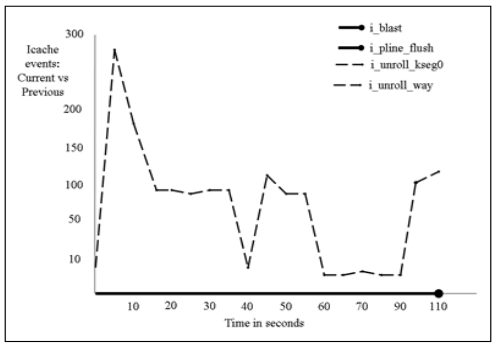


Figure 10. Histogram of Icache events

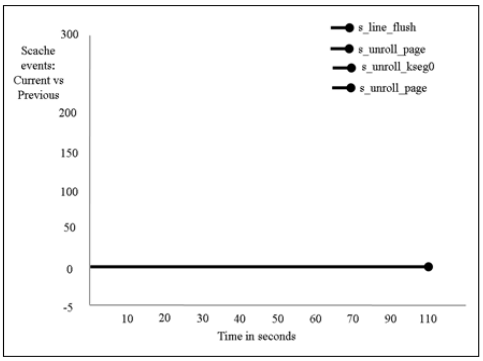


Figure 11. Histogram of Scache events

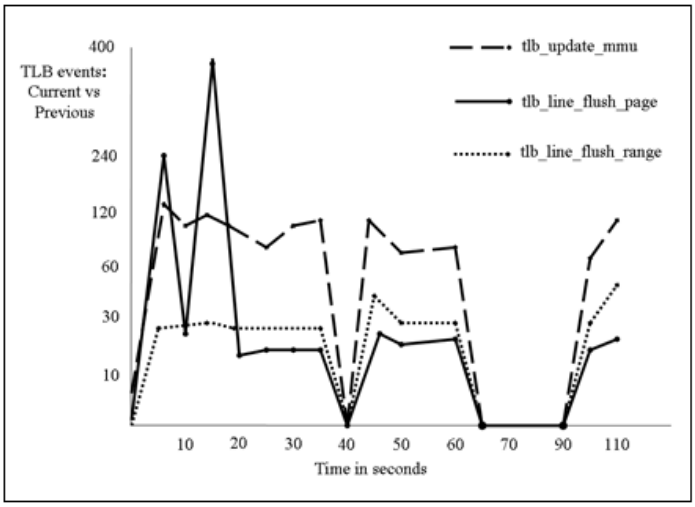


Figure 12. Histogram of TLB events

Table 3. Lines of Code added without comments and blank lines.

| File name | Line of code added |
|---|---|
| $linux/include/asm/mips32_cache.h | 69 |
| $linux/arch/mips/kernel/Makefile | 1 |
| $linux/arch/mips/kernel/proc.c | ,2 |
| $linux/arch/mips/mm/tlb-r4k.c | 20 |
| $busybox/{Config.h, applets.h, usage.h} | 1, 3, 4 |
| $linux/arch/mips/kernel/cache_perf_proc.c | 499 |
| $linux/include/asm/cache_perf_mips32.h | 81 |
| $busybox/zmet.c | 329 |

The *$linux* and *$busybox* used in Table 3 indicates the source code location *.../src/linux/linux*, and *.../src/router/busybox* respectively. The Table 4 lists the comparison of the file size; in bytes; listed under Rows 2 through 5, 7 through 10, 12 through 15, 17 through 20, and 22 through 25; for combinations of router firmware image built on the development system indicated in Rows 1, 6, 11, 16, and 21 respectively. The Column 3 lists the file sizes for the corresponding file type indicated in Column 2. The focus of the modifications has been to ensure the size of the vmlinux; busybox, target.squashfs and the firmware image do not exceed the respective size indicated in for Row 1 Vanilla. The target.squashfs is the file system that will be programmed into the router, while the firmware image is the result of the Step j of Section 5.

Referring to Table 3, the additional code added to the kernel will increase the kernel size, and this increase is seen in the Rows 7, 12, 17 and 22 of Table 4, for vmlinux. The Rows 11 through 25 of Table 4 indicates the usage of trimmed html pages, and was achieved for additional space on the router flash for the files under the *.../project/acos/www/html* directory. Likewise, the default utilities; cp, ping, etc., bundled in busybox were trimmed. The corresponding reduction in the size of busybox is seen in Row 13 Column 3 of Table 4. The metric analysis tool zmet has been integrated into busybox; linked into the directory /usr/bin; has increased the size of busybox as seen in Rows 18 and 23 Column 3 of Table 4. Although the size of the kernel vmlinux and busybox has increased with the integration of the performance measurement metrics; Column 3 Rows 7, 12, 17, and 22 of Table 4, the overall size of the target. Squashfs is less than the corresponding size indicated under Row 4 Column 3 under Table 4.

Table 4. Listing of files sizes.

| | | |
|---|---|---|
| 1 | Vanilla (water mark) | |
| 2 | vmlinux | 1,864,769 |
| 3 | busybox | 242,644 |
| 4 | target.squashfs | 1,409,024 |
| 5 | firmware image | 1,867,834 |
| 6 | With printk & metrics | |
| 7 | vmlinux | 1,869,878 |
| 8 | busybox | 242,644 |
| 9 | target.squashfs | 1,404,928 |
| 10 | firmware image | 1,863,738 |
| 11 | With printk & metrics, trimmed html pages & busybox | |
| 12 | vmlinux | 1,869,878 |
| 13 | busybox | 238,180 |
| 14 | target.squashfs | 1,388,544 |
| 15 | firmware image | 1,847,354 |
| 16 | With printk & metrics, trimmed html & busybox, zmet application | |
| 17 | vmlinux | 1,869,878 |
| 18 | busybox | 242,788 |
| 19 | target.squashfs | 1,392,640 |
| 20 | firmware image | 1,851,450 |
| 21 | Without printk, with metrics, trimmed html & busybox, zmet application | |
| 22 | vmlinux | 1,865,782 |
| 23 | busybox | 242,788 |
| 24 | target.squashfs | 1,392,640 |
| 25 | firmware image | 1,851,450 |

## 8. CONCLUSIONS

Hardware counters are not available on the MIPS32 architecture; hence, performance measurements necessitate the usage of software counters. Software counters are defined in the kernel to track the events being measured; hence, there is a possibility of the counters overflowing. Usage of overflow counters has been incorporated L1 Cache and TLB management is external to the hardware; hence, the OS manages the cache and TLB. Twenty-seven base metrics has been defined based on software engineering approach for measuring the L1 cache and TLB events on the MIPS32 architecture implementation, with an equal number of overflow counters. Linux Kernel 2.4.20 has been instrumented for metric data collection on NETGEAR WGR614v9 router having an implementation of MIPS32 processor from BROADCOM.

The generated data aids to monitor, optimise, tune, model and benchmark the system that comprise of the architecture, its subsystems, and the executing processes; be it the operating system or the applications using the operating system. A metric data extraction application *zmet* has been developed to aid analysis of the activities of the L1 cache and TLB through validation.

The challenges that were offered by the source code and the NETGEAR WGR614v9 router during the implementation phase were:

a. Space on router for additional code 9717 bytes of binary code (metrics + zmet)
b. Source code compatibility with loadable modules
c. Documentation support with the source code
d. Command line parameters processing
e. Proc file writing

The challenges were addressed by using the software engineering approach as follows:

a. Trimming the html files to gain 80k of code space
b. Trimming the *busybox* utilities by removing ping, cp etc.

c. Developing a technique to introduce modules into the kernel with the by figuring out the location to introduce the code
d. How-to knowledge nuggets were developed for module adding, ftp, space saving techniques, and telnet methods

## 9. APPENDIX A. DATA STRUCTURE FOR PERFORMANCE METRICS

Data structure for performance metrics

```
/*
 * Structure containing the cache performance metrics for mips32 processor.
 * Can consider to change the roll over counter from __u32 to
 * __u64, and depends on the size of the kernel & the activity
 * noticed with respect to the roll over counters.
 */
struct perf_m
{
/* dcache related metrics */
__u32 d_way;            /* ways updates */
__u32 d_way_roll;       /* ways roll */
__u32 d_invld_ln;       /* invalidate line */
__u32 d_invln_roll;     /* invalidate line roll over */
__u32 d_writeback;      /* writeback counter */
__u32 d_wb_roll;        /* writeback counter roll over */
__u32 d_blast;          /* blast counter */
__u32 d_blast_roll;     /* blast counter roll over */
__u32 d_unroll_kseg0; /* cache unroll for kseg0 */
__u32 d_ukseg_roll;   /* cache unroll for kseg0 roll over */
__u32 d_unroll_way; /* cache unroll on ways */
__u32 d_uw_roll;       /* cache unroll on ways roll over */
__u32 d_line_flush;   /* cache line flush */
__u32 d_lf_roll;        /* cache line flush roll over */
__u32 d_unroll_page;/* cache unroll */
__u32 d_up_roll;       /* cache unroll roll over counter */
/* icache related metrics */
__u32 i_way;            /* ways updates */
__u32 i_way_roll;       /* ways update roll */
__u32 i_blast;          /* blast counter */
__u32 i_blast_roll;     /* blast counter roll over */
__u32 i_unroll_way;   /* cache unroll on ways */
__u32 i_uw_roll;        /* cache unroll on ways roll over */
__u32 i_unroll_kseg0; /* cache unroll for kseg0 */
__u32 i_ukseg_roll;     /* cache unroll for kseg0 roll over */
__u32 i_line_flush;     /* cache line flush */
__u32 i_lf_roll;        /* cache line flush roll over */
__u32 i_pline_flush;    /* cache line flush */
__u32 i_plf_roll;       /* cache line flush roll over */
__u32 i_unroll_page;   /* cache unroll */
__u32 i_up_roll;        /* cache unroll roll over counter */
__u32 i_fill;           /* cache line fill */
__u32 i_fill_roll;      /* cache line fill roll over counter */
/* scache related metrics */
__u32 s_invld_ln;       /* invalidate line */
__u32 s_invln_roll;     /* invalidate line roll over */
__u32 s_way;            /* ways update */
```

```
__u32 s_way_roll;       /* ways update roll over */
__u32 s_unroll_kseg0;      /* cache unroll for kseg0 roll over */
__u32 s_ukseg_roll;        /* cache unroll for kseg0 roll over */
__u32 s_unroll_pg_ways; /* cache unroll, page and ways */
__u32 s_upw_roll;           /* cache unroll, page and ways roll over */
__u32 s_line_flush;        /* cache line flush */
__u32 s_lf_roll;           /* cache line flush roll over */
__u32 s_unroll_page;       /* cache unroll */
__u32 s_up_roll;            /* cache unroll roll over counter */
/* tlb related metrics        */
__u32 tlb_lflush_all;      /* tlb local flush all */
__u32 tlb_lfa_roll;         /* tlb local flush all roll */
__u32 tlb_lflush_mm;       /* tlb local flush of mm */
__u32 tlb_lfmm_roll;       /* tlb local flush of mm roll over */
__u32 tlb_lflush_rng;      /* tlb local flush of a range */
__u32 tlb_lflrng_roll;     /* tlb local flush of a range roll over */
__u32 tlb_lflush_pg;       /* tlb local flush of page */
__u32 tlb_lfpg_roll;       /* tlb local flush of page roll over */
__u32 tlb_updt_mmu;    /* tlb update mmu */
__u32 tlb_upmmu_roll;  /* tlb update mmu roll over */
};
```

## 10. APPENDIX B. METRIC DATA DISPLAY

```
Metric data display in /proc/cpuinfo
Trying 192.168.98.4...
Connected to 192.168.98.4.
Escape character is '^]'.
BusyBox v0.60.0 (2010.03.04-08:34+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.
# cat /proc/loadavg
0.04 0.01 0.00 2/17 116
### cat /proc/cpuinfo
system type   : Broadcom BCM5354 chip rev 3
processor   : 0
cpu model  : BCM3302 V2.9
BogoMIPS : 237.56
wait instruction : no
microsecond timers  : yes
tlb_entries  : 32
extra interrupt vector  : no
hardware watchpoint   : no
VCED exceptions  : not available
VCEI exceptions  : not available
unaligned_instructions  : 0
dcache metrics: 0,0,0,0,41,0,0,0,10002,0,10002,0,115440,0,13953,0
icache metrics: 0,0,0,0,10002,0,10002,0,0,0,41,0,0,0,0,0
scache metrics: 0,0,0,0,0,0,0,0,0,0,0,0
tlb metrics: 6,0,296,0,3915,0,1493,0,8074,0
### cat /proc/cpuinfo
system type  : Broadcom BCM5354 chip rev 3
processor : 0
cpu model  : BCM3302 V2.9
```

BogoMIPS : 237.56
wait instruction : no
microsecond timers : yes
tlb_entries : 32
extra interrupt vector : no
hardware watchpoint : no
VCED exceptions : not available
VCEI exceptions : not available
unaligned_instructions : 0
dcache metrics: 0,0,0,0,51,0,0,0,10280,0,10280,0,131393,0,14396,0
icache metrics: 0,0,0,0,10280,0,10280,0,0,0,51,0,0,0,0,0
scache metrics: 0,0,0,0,0,0,0,0,0,0,0,0
tlb metrics: 6,0,304,0,3991,0,1550,0,8327,0
### cat /proc/loadavg
0.02 0.01 0.00 2/18 121
#
# cat /proc/cpuinfo
system type : Broadcom BCM5354 chip rev 3
processor : 0
cpu model : BCM3302 V2.9
BogoMIPS : 237.56
wait instruction : no
microsecond timers : yes
tlb_entries : 32
extra interrupt vector : no
hardware watchpoint : no
VCED exceptions : not available
VCEI exceptions : not available
unaligned_instructions : 0
dcache metrics: 0,0,0,0,64,0,0,0,10760,0,10760,0,155797,0,15133,0
icache metrics: 0,0,0,0,10746,0,10746,0,0,0,64,0,0,0,0,0
scache metrics: 0,0,0,0,0,0,0,0,0,0,0,0
tlb metrics: 6,0,320,0,4147,0,1653,0,8739,0
##
/* end of cache performance metrics structure */

**REFERENCES**

[1] Broadcom Corporation, "*BCM5354 Product Brief – Optimised 802.11G Router with Broadrange*," Broadcom Corporation, 5354-PB01-R, 11 Sept 2007.
[2] BELKIN Inc., "Vendor toolchains, " ftp://ftp.gpl-devices.org/pub/vendors/Belkin, Sept 2009.
[3] Brinkley Sprunt, "The Basics of Performance Monitoring Hardware," *0272-1732/02/2002 IEEE,* 2002.
[4] Don Anderson, "Universal Serial Bus System Architecture," *Second Edition, Addison-Wesley Developer's Press,* ISBN 0-201-46137-4, 2001.
[5] David S. Miller, "Cache and TLB Flushing Under Linux," *Linux Documentation,* …/ Documentation/cachetlb.txt, 2001.
[6] David S. Miller and Ralf Baechle, "arch/mips/mm/tlb-r4k.c," *Linux Kernel 2.4.20 source code,* 1997
[7] Dominic Sweetman, "See MIPS Run," *Second Edition, Morgan Kaufmann Publishers, ISBN 13: 978-0-12-088421-6,* 2007.
[8] Guy G. F. Lemieux, "Hardware Performance Monitoring in Multiprocessors," *Masters degree thesis - University of Toronto ,* 1996.
[9] Gregory S. Freeland, Joel L. Gross, and Jose A. Laboy, "Tuneable Processor Performance Benchmarking," *USA Patent 20070136726 A1,* http://www.freepatentsonline.com/20070136726.html, June 14 2007.
[10] IEEE, "Software Engineering Body of Knowledge," *IEEE, 2004,* pp 2-1 to 2-10, http://www.swebok.org.

[11] Jack Dongarra, Kevin London, Shirley Moore, Phil Mucci, and Dan Terpstra, "Using PAPI for hardware performance monitoring on Linux systems," *Innovative Computing Laboratory, University of Tennessee,* http://icl.cs.utk.edu/publications/pub-papers/2001/papi-linuxrev1.pdf, Sept, 2009.
[12] Lance M. Berc, Sanjay Ghemawat, Moniika H. Henzinger, Richard L. Sites, Carl A. Waldspurger, and William E. Weihl, "High Frequency Sampling of Processor Performance Counters," *USA Patent 5796939,* http://freepatentsonline.com/5796939.html, Aug 18 '98, Oct, 2009.
[13] M. Bolado, H. Posadas1, J. Castillo, P. Huerta1, P. Sánchez, C. Sánchez, H. Fouren, and F. Blasco, "*Platform based on Open-Source Cores for Industrial Applications,"* Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 1530-1591/04 IEEE, 2004.
[14] Marco Zagha, Brond Larson, Steve Turner, and Marty Itzkowitz, *"Performance Analysis using the MIPS R10000 Performance Counters*," Proceedings of the 1996 ACM/IEEE Conference on Supercomputing (SC'96) , 0-89791-854-1/96, 1996.
[15] MIPS Technologies, "MIPS32 ® 4Kc ™ Processor Core Datasheet – Revision 01.03," *MIPS Technologies Inc., MD00247,* June 2000.
[16] MIPS Technologies, "…/include/asm-mips/mips32_cache.h," *Linux Kernel 2.4.20 source code,* 1996.
[17] M. Warner Losh, "An overview of FreeBSD/mips," *AsiaBSDCon 2009, February 2009,* http://2009.asiabsdcon.org/papers/abc2009-P4B-paper.pdf, Aug 2009.
[18] NETGEAR Inc., "Wireless-G Router WGR614v9 Reference Manual," *NETGEAR Inc., 202-10308-01,* May 2008.
[19] Paul J. Drongowski, "Basic Performance Measurements for AMD Athlon 64, AMD Opteron and AMD Phenom Processors," *Advanced Micro Devices, Inc,* http://developer.amd.com/Assets/Basic_Performance_Measurements.pdf, September 25 2008, (Nov 2009)
[20] Ralf Baechle, "Cache operations for the cache instruction," *Linux kernel 2.4.20, .../include/asm-mips/cacheops.h,* 2002.
[21] Roger S. Pressman., "Software engineering: a practitioner's approach," - *5th ed, McGraw-Hill, ISBN 0-07-365578-3.*
[22] S. Browne, J Dongarra, N. Garner, G. Ho, P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors," *Computer Science Department, University of Tennessee, and Oak Ridge National Laboratory,* http://icl.cs.utk.edu/publications/pub-papers/2000/papi-journal-final.pdf, Sept 2009.
[23] Shirley Moore, Patricia Teller, and Michael Maxwelll, "Efficiency and Accuracy Issues for Sampling vs. Counting Modes of Performance Monitoring Hardware," *University of Tennessee-Knoxville and University of Texas-El Paso.*
[24] Tsuyoshi Nagao and Hitoshi Suzuki, "Processor System and Performance Measurement Method for Processor System," *USA Patent 20070277178A1, Nov 29 2007,* http://www.freepatentsonline.com/20070277178.html.
[25] Warner Losh, "A Brief history of FreeBSD/MIPS," *BSDCan 2008 Canada,* http://www.freebsd.org/~imp/bsdcan2008.pdf, Aug 2009.
[26] Wendy Korn, Patricia J. Teller, and Gilbert Castillo, "Just how accurate are performance counters," *University of Texas as El Paso,* http://www.cs.utep.edu/pcat/papers/IPCCC2001paper.pdf, Aug 2009
[27] Wiplove Mathur, and Jeanine Cook, "Improved Estimation for Software Multiplexing of Performance Counters," http://www.ece.nmsu.edu/~jecook/pubs/jcook_multiplexing.pdf, Sept 2009
[28] WGR614v9 router source code, ftp://downloads.netgear.com/files/GPL, Sept 2009
[29] Xiao Zhang, Sandhya Dwarkadas, Girts Folkmanis, and Kai Shen, "Processor Hardware Counter Statistics As A First-Class System Resource," *Department of Computer Science, University of Rochester, 2007,* http://www.cs.rochester.edu/u/sandhya/papers/hotos07.pdf, Aug 2009
[30] FreeBSD/MIPS Project, http://www.freebsd.org/platforms/mips.html, Aug 2009
[31] Toolchains, http://www.linux-mips.org/wiki/Toolchains, Aug 2009
[32] Vendor toolchains, ftp://ftp.gpl-devices.org/pub/vendors/Belkin, Sept 2009

## BIOGRAPHIES OF AUTHORS

Varuna Eswer received his B.Tech in Computer Science and Engineering and MSc [Engg] in Real-Time Embedded Systems from Mysore University, India and Coventry University, UK.
He is founder and CEO Eudaemonic Systems. His research interests include the field of Open BSD, Operating Systems, Embedded Systems and System on Chip Design.

Sanket Dessai received his BSc, MSc degree in Physics and MSc [Engg] in Real-Time Embedded Systems from Goa University, India and Coventry University, UK.He is in Academic position at Assistant Professor. He is also hardcore Consultant, Researcher, and Tranier for many MNC. His research interests include the field of System on Chip Design, Embedded Systems, MEMS/NEMS Engineering, Nanophysics and Nanotechnology, Solid State Physics and Engineering and Photonics.