# Analysis and Optimization of Code with sim_G4plus on Genesi Pegasos II

*by* *Maurie Ommerman, Top Changwatchai, James Yang*
*CPD Applications*
*Freescale Semiconductor, Inc.*
*Austin, TX*

This application note is the tenth in a series describing the Genesi Pegasos II system and its various applications.

Contents



## 1 Introduction

This document describes analysis and optimization of code with the sim_G4plus simulator. While Freescale application note AN2749, *Using sim_G4plus on the Genesi Pegasos II*, describes the general use and methodology of sim_G4plus, this application note is intended to analyze a single problem, a bubble in the pipeline, and determine a programming solution to remove the bubble. Therefore, AN2749 is a prerequisite to this document. An overview of the sections included follows:

- Section 2, “Terminology,” defines the terminology used in this application note.
- Section 3, “Creating the Execution Pipeline Trace File,” describes the procedures for creating a pipeline trace file with the sim_G4plus simulator.
- Section 4, “Viewing the Pipeline Trace File,” describes how to display, read, and analyze the pipeline trace file.
- Section 5, “Conclusions,” describes an approach to determining a programming solution.
- Section 6, “References,” lists the other documents in the series and the reference material used in preparing this application note.
- Section 7, “Document Revision History,” describes the history of this application note.



freescale semiconductor

# 2 Terminology

The following terms are used in this document.

| | |
|---|---|
| Linux OS | Linux operating system |
| Bubble | A gap in the instruction flow, typically caused by a stall. |
| gcc | GNU compiler collections and GNU utilities |

# 3 Creating the Execution Pipeline Trace File

The goal of this paper is to demonstrate how to read the pipeline output, find a bubble in the pipeline, and correct the program to remove the bubble, and therefore speed up program execution. For the analysis described in Section 4, "Viewing the Pipeline Trace File," a pipeline trace file is generated by using the procedures in this section. This section describes how to do the following: load and compile the program, insert the trace start and stop markers to isolate the code of interest, and run the simulator to produce the pipeline trace file.

The program, sndfdemo-slow.c, used for this example is in the directory /home/guest/fae-training-04/library/matrix_mul/sndf-slow. The sndfdemo-slow.c program, which is not described here, uses AltiVec™ instructions and is a series of vector multiplies.

To begin, log in as `guest`, with the password `guest`. Then change to the directory, /home/guest/fae-training-04/library/matrix_mul/sndf-slow, using the following commands:

```
guest@debian:~$ cd fae-training-04/library/matrix_mul/

guest@debian:~/fae-training-04/library/matrix_mul$ ls

performance  sndfdemo-fast  sndfdemo-slow  sndfdemo.tar

guest@debian:~/fae-training-04/library/matrix_mul$

guest@debian:~/fae-training-04/library/matrix_mul$ cd sndfdemo-slow

guest@debian:~/fae-training-04/library/matrix_mul/sndfdemo-slow$ ls

,build.sh             sndfdemo-slow.execute.pipeout

,cleanup.sh           sndfdemo-slow.execute.stats

,execute-and-sim.sh   sndfdemo-slow.lptrace.pipeout

,trace-and-sim.sh     sndfdemo-slow.lptrace.stats

,viewpipe.execute.sh  sndfdemo-slow.traceme

,viewpipe.lptrace.sh  sndfdemo-slow.tte

sndfdemo-slow.c       ttt

guest@debian:~/fae-training-04/library/matrix_mul/sndfdemo-slow$
```

The shell script `,build.sh` compiles the program as shown in the example that follows. Note that all shell scripts in this directory begin with a comma.

```
guest@debian:~/fae-training-04/library/matrix_mul/sndfdemo-slow$ cat ,build.sh

#!/bin/sh

# remember: must link statically

gcc -O2 -maltivec -mcpu=7450 sndfdemo-slow.c -o sndfdemo-slow.traceme -static
```

A fragment of the C code from the program sndfdemo-slow.c is used as an example of how to use the special markers to isolate an instruction and monitor its progress in the pipeline. There is nothing new here; set the appropriate AltiVec flags and create the elf executable, sndfdemo-slow.traceme, as shown in the script above, with static libraries. The start and stop markers, 0x14000001 and 0x14000002, are inserted in lines 80 and 82 fragment around the function call, do_mult_vec, which is the code to be traced.

```
73  int main (void) {
    74          int j;  /* accumulates return values from multply functions to
    75                     prevent them from being optimized away */
    76
    77          print_vector_uc(&va);
    78          print_vector_uc(&vb);
    79
    80          asm (".long 0x14000001");       /* start tracing */
    81          j = do_mult_vec (&va, &vb);
    82          asm (".long 0x14000002");       /* stop tracing */
    83          print_vector_uc(&vvc);
```

The shell script `,trace-and-sim.sh` runs the simulator and produces the pipeout file, sndfdemo-slow.lptrace.pipeout, which this paper analyzes. The following command lists this script.

```
guest@debian:~/fae-training-04/library/matrix_mul/sndfdemo-slow$ cat ,trace-and-sim.sh
#!/bin/sh

echo "Running using lptrace..."
# Generate trace  (executable -> .tte)
lptrace -f sndfdemo-slow.tte -x ./sndfdemo-slow.traceme

if [ ! -x lptrace ]; then
    echo "ERROR: Expected sim_G4plus to be here: ../bin/sim_G4plus"
    #exit 1
fi

echo "Running on the simulator..."
# Run through simulator (.tte -> .pipeout) (.tte -> .stats)
sim_G4plus -p sndfdemo-slow.lptrace.pipeout sndfdemo-slow.tte > sndfdemo-slow.lptrace.stats


if [ ! -x sim_G4plus ]; then
    echo "ERROR: Expected sim_G4plus_vp to be here: ../bin/sim_G4plus_vp"
    #exit 1
fi
```

# 4 Viewing the Pipeline Trace File

The following sequence displays the pipeline file with the sim_G4plus_vp viewer.

```
guest@debian:~/fae-training-04/library/matrix_mul/sndfdemo-slow$ sim_G4plus_vp
sndfdemo-slow.execute.pipeout

Reading pipeline file `/usr/bin/babehspf WriteBack.Buffer
sndfdemo-slow.execute.pipeout|'opened via `/usr/bin/babehspf WriteBack.Buffer
sndfdemo-slow.execute.pipeout|'...

Reading SPF 2.x file

Updating data values...

File read complete.
```

Figure 2 on page 8 shows the pipeout display from the sim_G4plus_vp viewer. The figures that follow in Section 4.4, “Pipeline Analysis Examples,” illustrate the descriptions given below.

## 4.1 Pipeline View Output

Note that the snapshot output has many columns. The snapshot body consists of two parts: an instruction listing on the left and a pipeline view on the right. As shown in the screen shots in Section 4.4, “Pipeline Analysis Examples,” the far left of the display window lists instructions flowing through the pipeline; the pipeline stages and queues are to the right of the instruction information and clock cycle column. Each instruction is identified by a letter, shown on the right side as the instruction flows through the pipeline. Clock cycles are represented horizontally in the pipeline, with each row displaying what instructions are in the pipeline in the given cycle.

### 4.1.1 Instruction View

The instruction view on the left maps each instruction in the trace to a letter that is used to represent the instruction. The instructions are listed in the order that they are fetched from the trace. The uppercase and lowercase letters are assigned using the instruction number modulo 52; for example, instruction 0 appears as A and instruction 51 appears as z. Speculative path instructions are marked with the # character before the instruction address. The first instruction listed is the oldest instruction in the snapshot. The last instruction listed is the newest, which may not yet be visible in the pipeline view. If more than 52 instructions are in a snapshot, enumeration restarts at A. Careful inspection of the snapshot is needed to make sure the proper instruction is being observed.

### 4.1.2 Core Pipeline View

The pipeline view on the right side of the display shows the state of all of the queues in the machine in each cycle. The letters in the contents of each queue represent instructions, as indexed by the instruction listing on the left side of the snapshot. The header labels the columns in the snapshot body indicating the stage in each queue. Table 1 lists the header and other notation descriptions for the pipeline output.

**Table 1. Pipeline Output Legend**

| Column Name | Description |
|---|---|
| Fetch 0 | Fetch address, stage 0 |
| Fetch 1 | Fetch address, stage 1 |
| InstB | Instruction buffer (queue) or IQ |
| GIQ | General purpose issue queue |
| VIQ | Vector issue queue |
| FI | Floating-point issue queue |
| IU2/R/01 | Integer unit 2 reservation buffer |
| IU2MU/E/01 | Integer unit 2 multiply execution pipeline |
| IU2S/E/0 | Integer unit 2 system execution pipeline |
| IU1.1/R/0 | Integer unit 1.1 reservation buffer |
| IU1.2/R/0 | Integer unit 1.2 reservation buffer |
| IU1.3/R/0 | Integer unit 1.3 reservation buffer |
| VFPU/R/0 | Vector floating-point unit reservation buffer |
| VFPU/EEE/012 | Vector floating-point unit execution pipeline. Note: 3 cycles to perform the execution. |
| VPU/R/0 | Vector permute unit reservation buffer |
| VPU/E/0 | Vector permute unit execution pipeline |
| VIU1/R/0 | Vector integer unit 1 reservation buffer |
| VIU1/E/0 | Vector integer unit 1 execution pipeline |
| VIU2/R/0 | Vector integer unit 2 reservation buffer |
| VIU2/EEE/012 | Vector integer unit 2 execution pipeline. Note: 3 cycles to perform the execution. |
| FPU/RR/01 | Floating-point unit reservation buffer |
| FPU/EEEEE/01234 | Floating-point unit execution pipeline. Note: 5 cycles to perform the execution. |
| LSM/RR/10 | Load/store reservation buffer |
| LSM/EE/01 | Load/store writeback pipeline. Note: 2 cycles to perform the execution. |
| CB | Completion buffer (queue) or CQ |
| WB | Write-back buffer |
| P-A- | No prediction or direction status for branch |
| P-AT | No prediction for actually taken branch |
| PFAF | Predicted fall through, actually fall through branch |
| PTAT | Predicted taken, actually taken branch |
| CR | Completion redirect, the instruction completing out of the CQ redirects fetch unit to a new fetch address |
| FS | Fetch sequential, fetch the next sequential fetch group of instructions |

**Table 1. Pipeline Output Legend (continued)**

| Column Name | Description |
|---|---|
| BR | Branch prediction unit redirect, the fetch address is redirected due to a branch mispredict |
| IV | Invalid |
| IQ | Instruction queue full redirect |

## 4.2 Instruction Movement through the Stations

Figure 1 is an architectural diagram of the functional units of the MPC7447A processor.

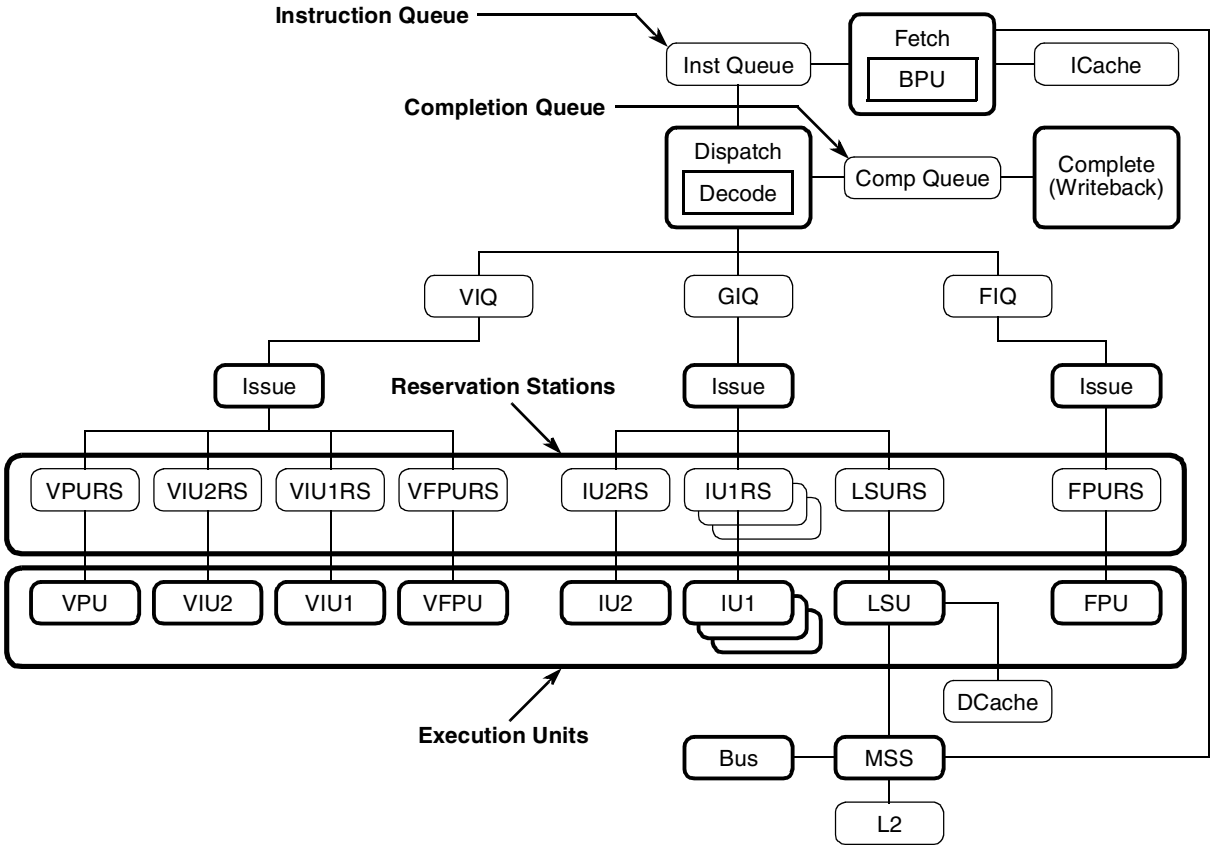


**Figure 1. MPC7447A Block Diagram**

## 4.3 Cycle Instructions

Look at the instruction fetch queues in Figure 2. Instructions are fetched sequentially until a branch is encountered. Instructions at address 0x10002cc, 0x10002d0, 0x10002d4, and 0x10002d8 are fetched, but at 0x10002d8 a branch instruction is allocated to the branch queue. In this case, it branched to address 0x1000404. Therefore instructions starting at 0x10002dc (the instruction following the branch) are not fetched. Thus instructions A, B, C, and D move from the Fetch queue into the InstB queue. This happens for cycle 1 and 3.

- In cycle 4, Instructions C, B, and A move to the GIQ for eventual dispatching. Instruction D is a taken branch so the E instruction comes from 0x1000404.

- In cycle 5, C moves to the IU2 queue, A moves to the IU1 queue, and B moves to the IU1 2 reservation queue.
- In cycle 6, H, G, F, and E enter the InstB queue.
- In cycle 7, G (an AltiVec instruction) enters VIQ.
- In cycle 8, G enters the VIU1 reservation queue.
- In cycle 9, G enters the VIU1 execution queue.
- The other cycles are left as an exercise for the user.

## 4.4 Pipeline Analysis Examples

Look at the following series of figures for an example of how a bubble will develop in the pipeline.

Figure 3 is the expanded view of the lines to be analyzed. Notice that the instructions of interest here are from lines O through U, with R, S, T, P, and O as the instructions of most interest.

Figure 4 and Figure 5 indicate that there is a bubble in the pipeline between R and T, because T depends on S for v10 to become available.

Figure 4 and Figure 6 indicate that T waits in the VIU2 reservation station 1 extra cycle, because S has not completed releasing v10.

Figure 4 indicates that T depends on the result in register v10 from S. S is delayed from entering VPU reservation station because it is blocked in VIQ by R. R waits for Q to set v10 and S waits for R to set v10 and therefore T waits for S to set v10.

Figure 7 indicates that R is held in VIQ for 1 extra cycle because the VIU2 reservation station is blocked by P. P is waiting because v0 is required from O before P can start. See instructions O and P.

| | | |
|---|---|---|
| (O) vperm | v0, v0, v0, v1 | (v0 is the result.) |
| (P) vmsumubm | v13, v10, v0, v9 | (v0 is needed.) |

Figure 8 indicates that P depends on the result in register v0 from O.

Figure 9 indicates that O stalls in the VPU reservation station because it depends on v1 from K.

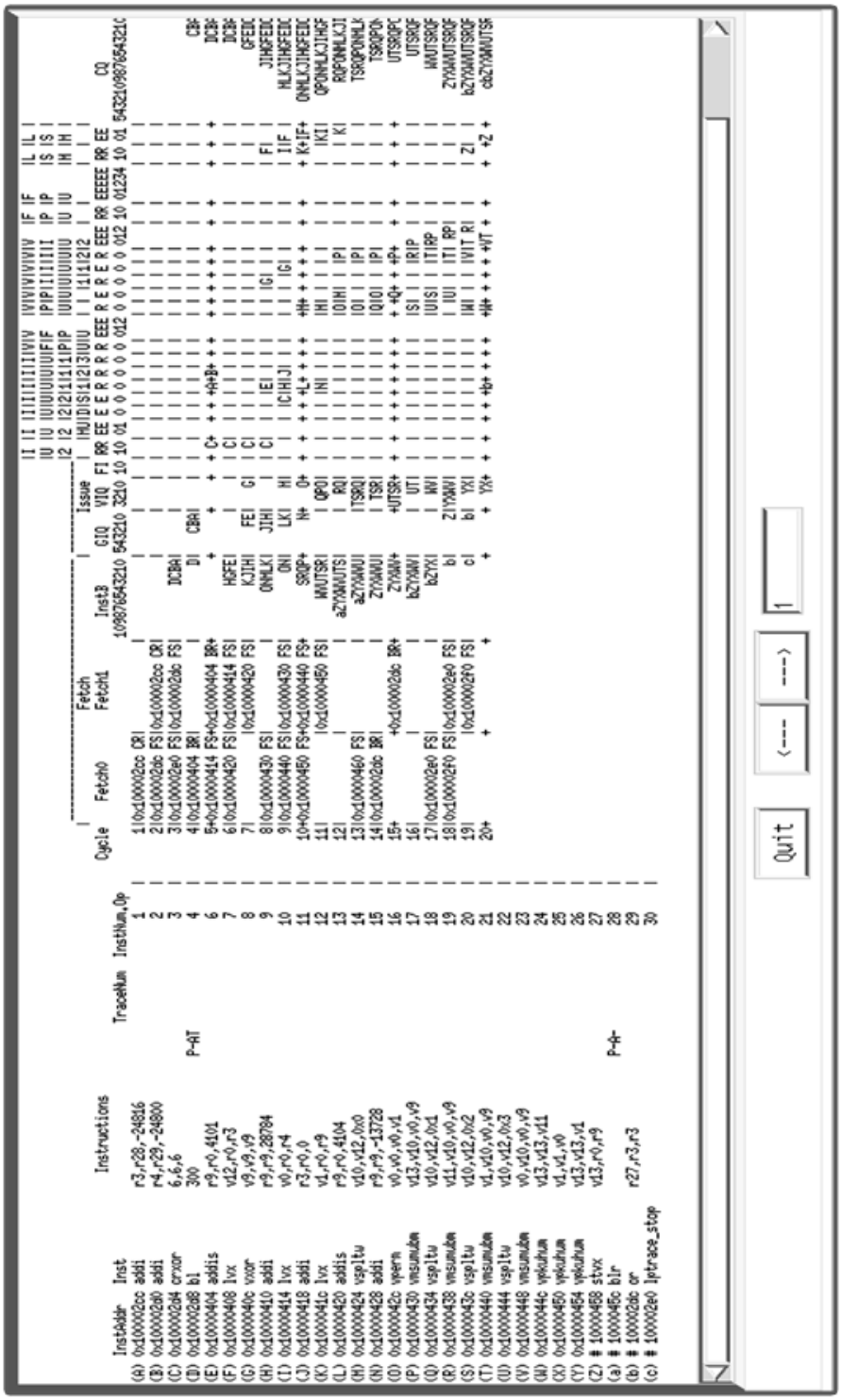


**Figure 2. Pipeline Output**

Note: plus signs '+' are used to indicate every 5 cycles.

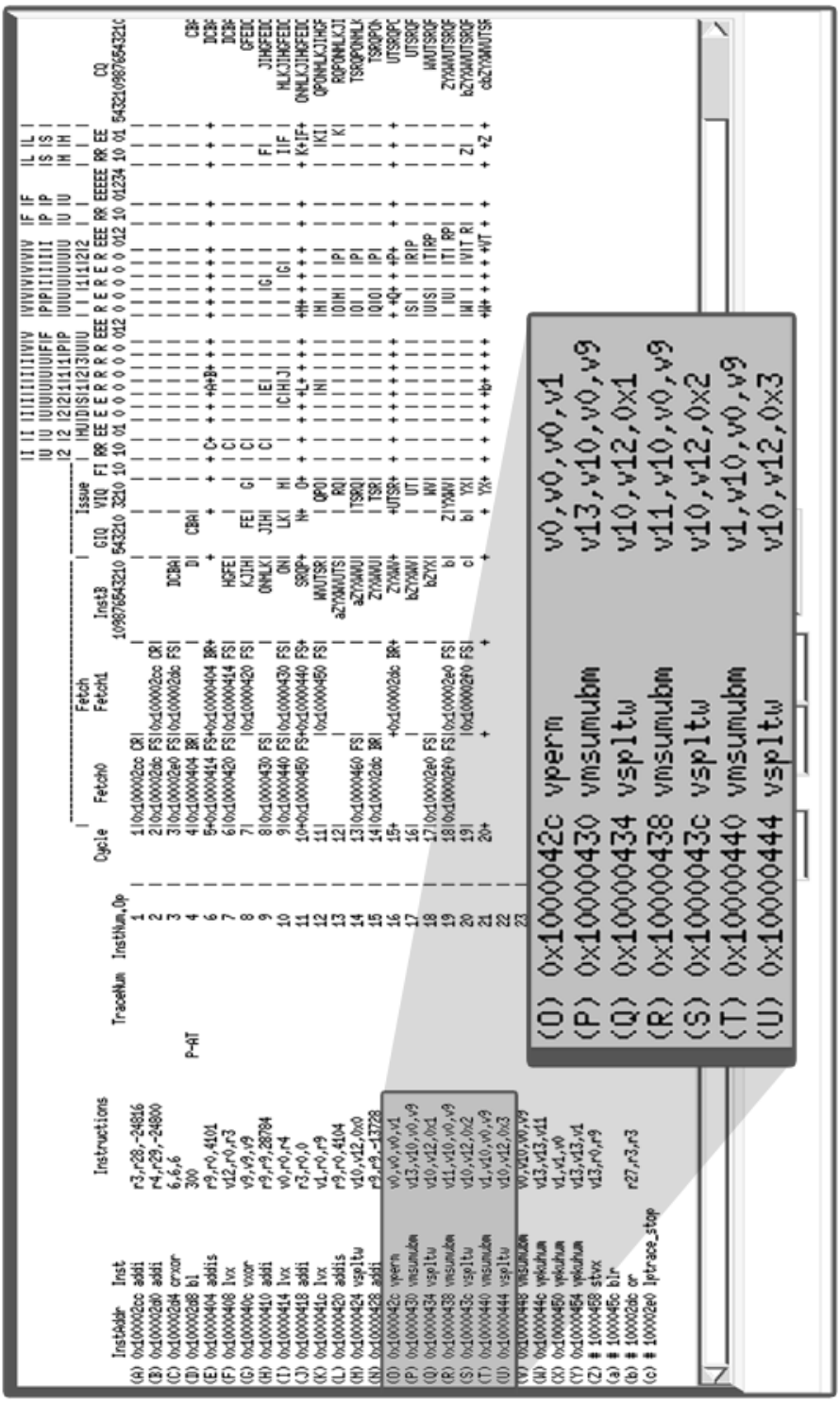


**Figure 3. Lines O through U Are of Interest**

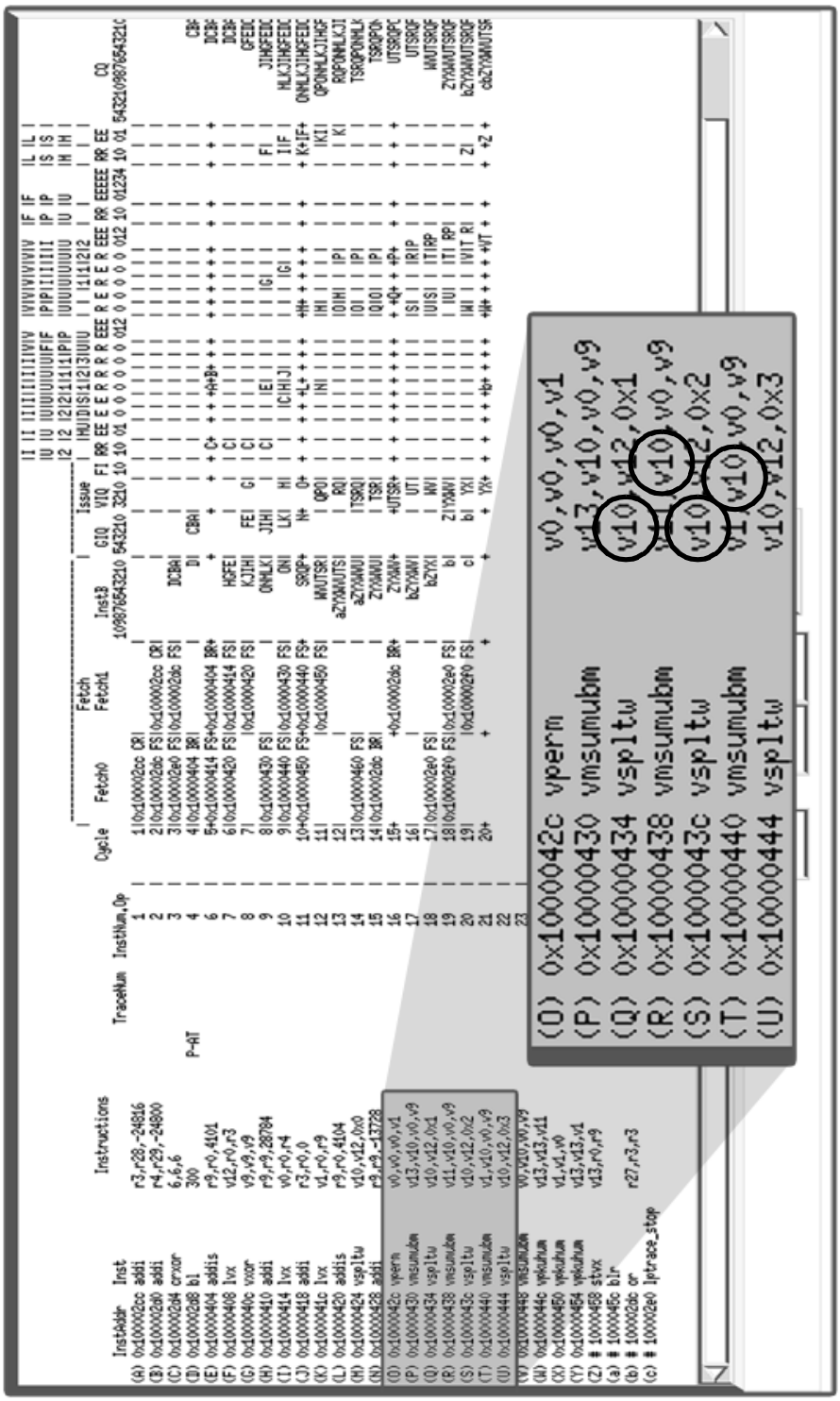


**Figure 4. R and S Wait for Q, T Waits on S for v10, and All to Have Access to v10**

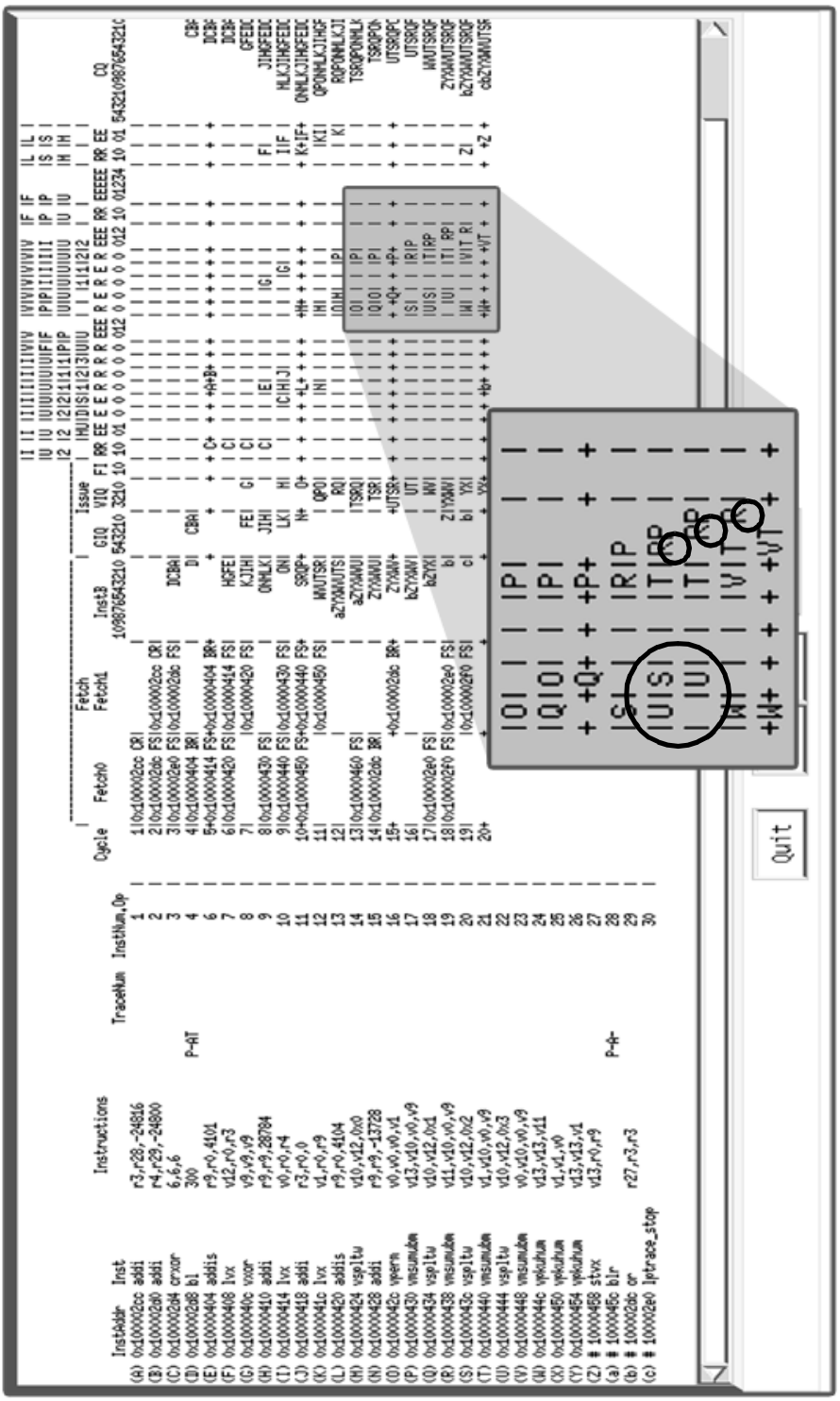


**Figure 5. One Bubble Exists for Three Cycles in the VIU2 R Station**

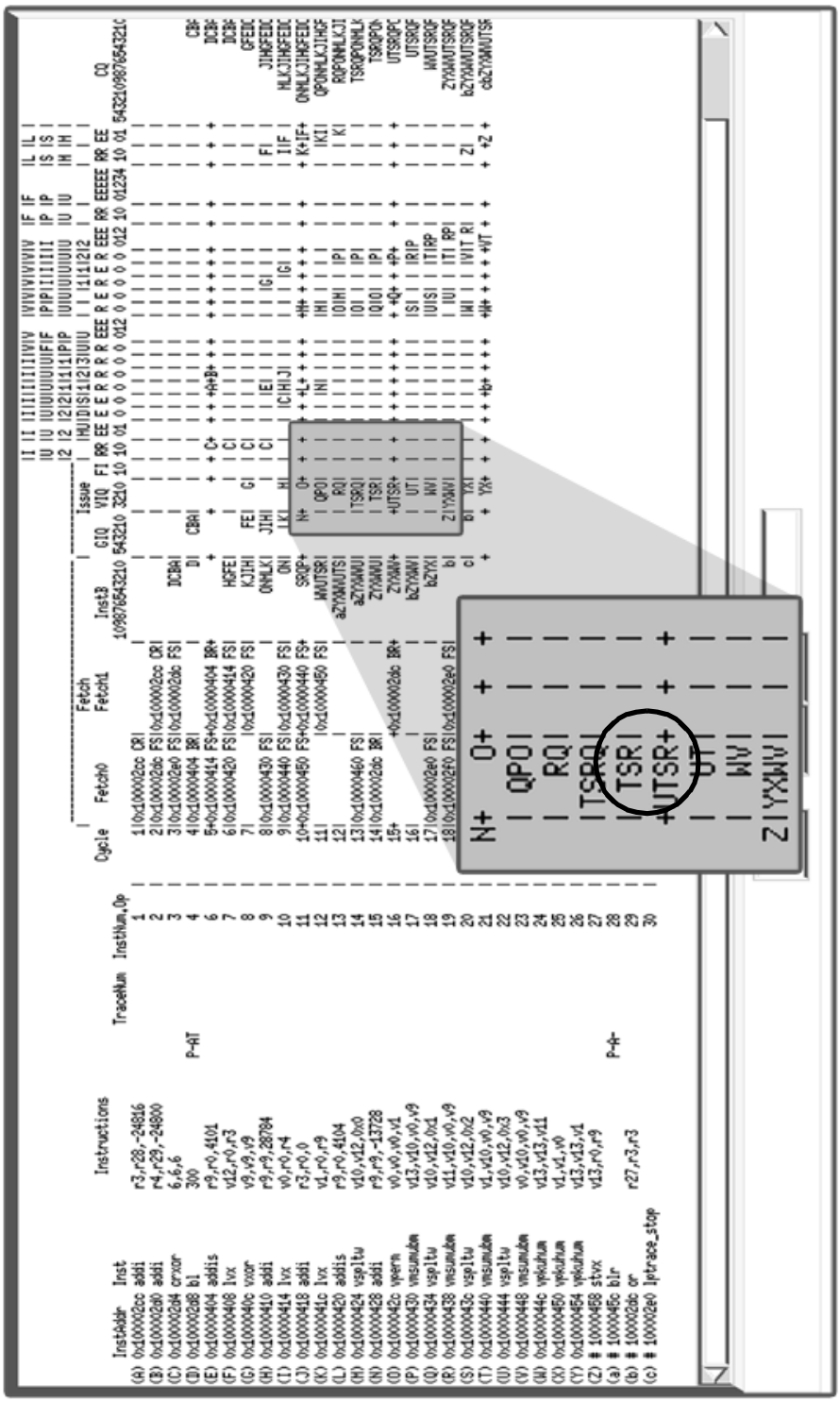

Figure 6. S Delayed Because R is Blocking

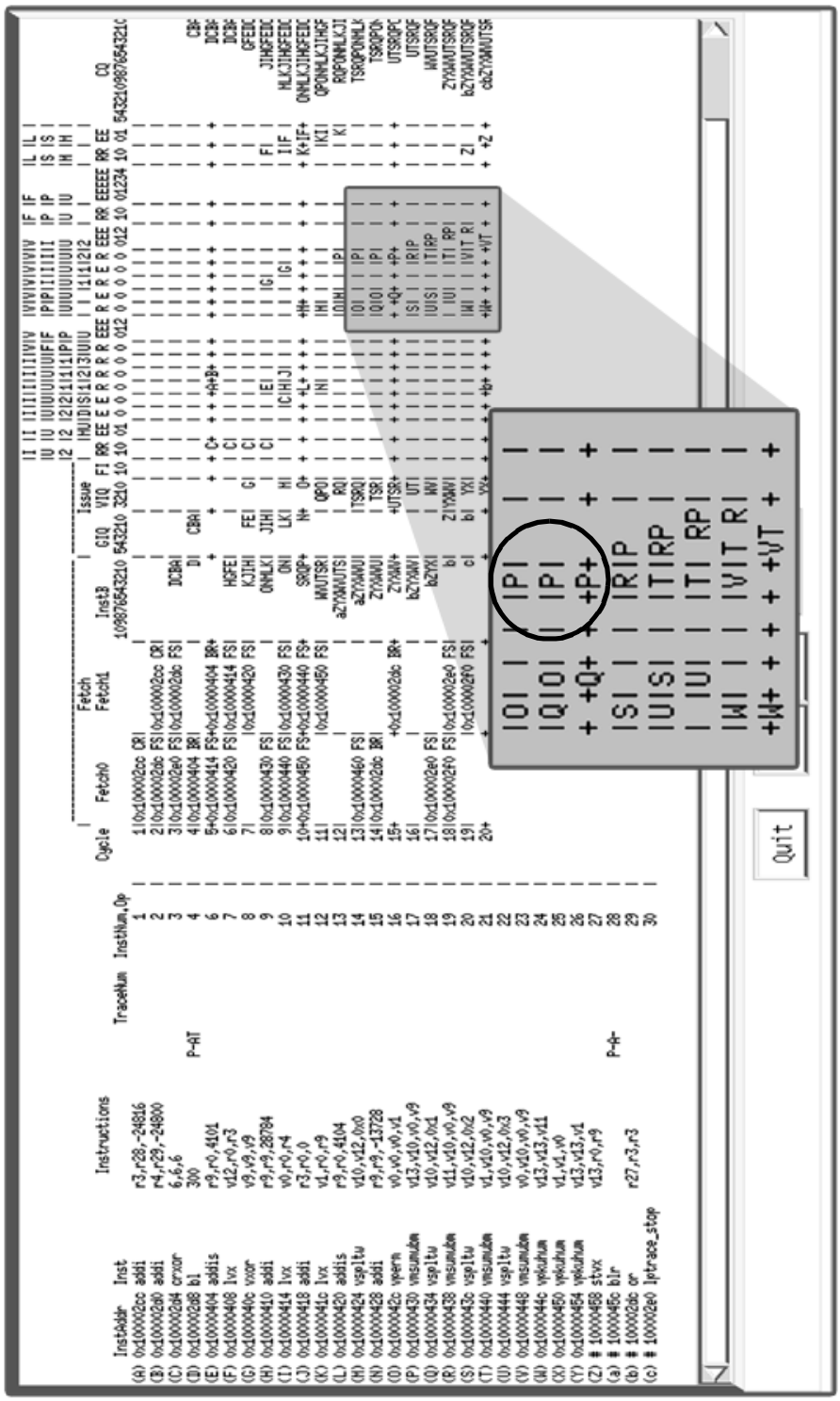


**Figure 7. P Waits Because v0 is Required from O**

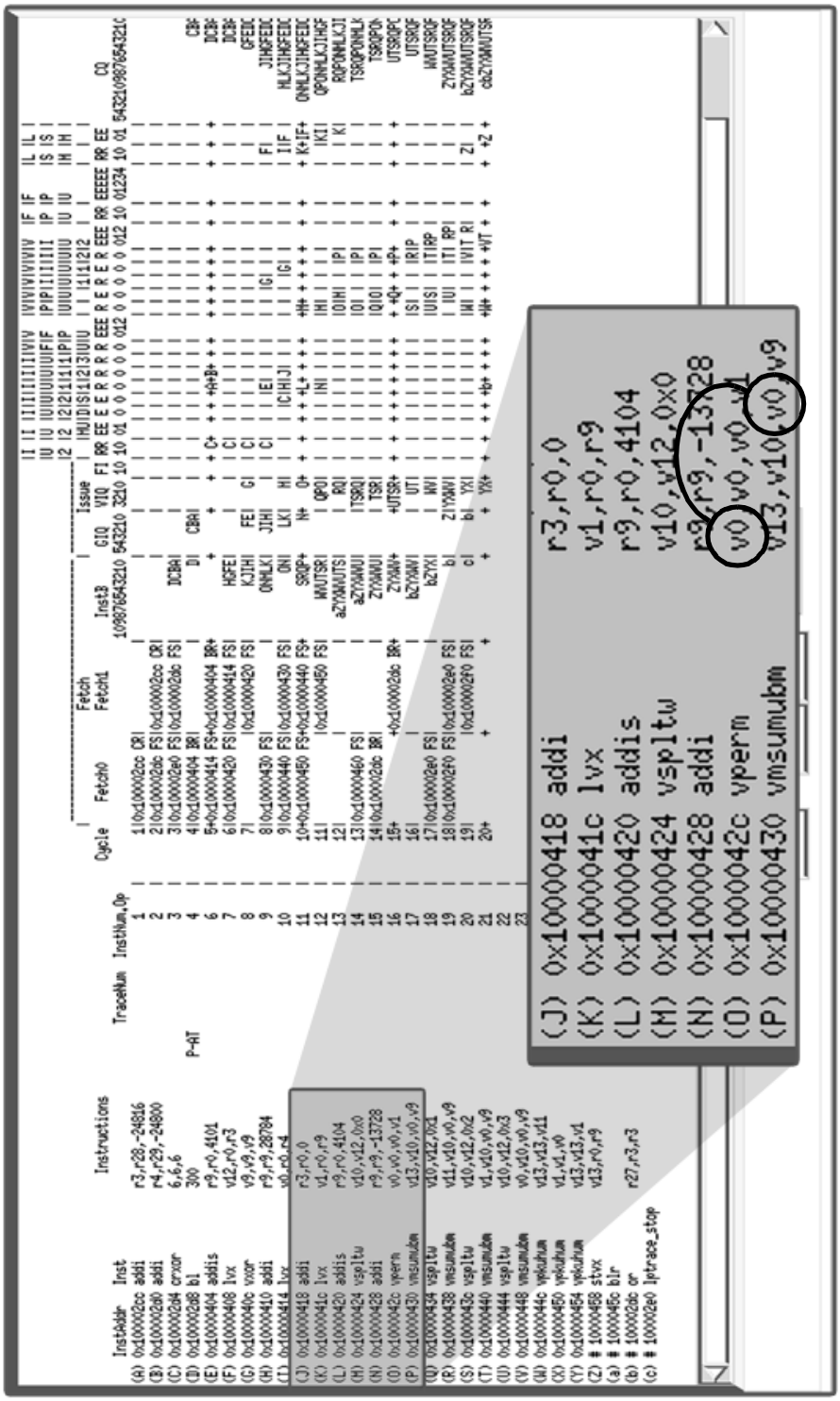


**Figure 8. P Waits for v0 from O and O Waits for v1 from K**

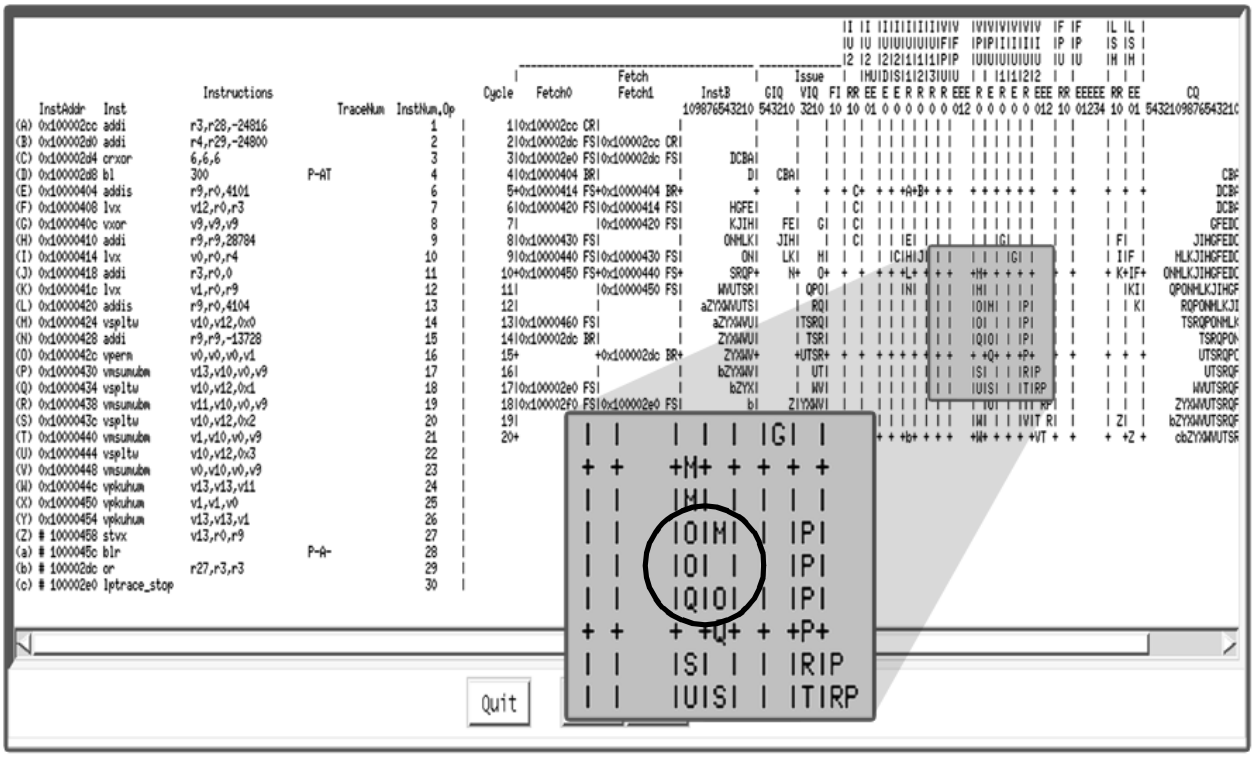


**Figure 9. O is Stalled**

# 5 Conclusions

If instructions are inserted between K and O, then O does not stall waiting for K. This is an easy task if the code is written in assembly. However, if it is a C program then look at the assembly code generated by the C compiler to find where these instructions reside and then determine how best to rearrange the assembly code to use the cycles in this stall.

This application note has presented the information needed to begin to understand analyzing the execution pipeline from the sim_G4plus simulator trace output.

# 6 References

The following documents describe the various applications of the Genesi Pegasos II system or are references for the systems.

- Freescale application notes in the Genesi Pegasos II series
  - AN2666, *Genesi Pegasos II Setup*
  - AN2736, *Genesi Pegasos II Boot Options*
  - AN2738, *Genesi Pegasos II Firmware*
  - AN2739, *Genesi Pegasos II Debian Linux*
  - AN2751, *Genesi Pegasos II Yellow Dog Linux*
  - AN2743, *Software Analysis on Genesi Pegasos II Using PMON and AltiVec*
  - AN2744, *PMON Module—An Example of Writing Kernel Module Code for Debian 2.6 on Genesi Pegasos II*
  - AN2748, *Genesi Pegasos II Kernel and NFS facility*
  - AN2749, *Using sim_G4plus on the Genesi Pegasos II*
- *sim_G4plus v0.7 Cycle-Accurate Simulator User's Guide*, Rev 1.5
- *RISC Microprocessor Family User's Manual* (MPC7450)

For assistance or answers to any question on the information that is presented in this document, send an e-mail to risc10@freescale.com.

# 7 Document Revision History

Table 2 provides a revision history for this application note.

**Table 2. Document Revision History**

| Revision Number | Date | Change(s) |
|---|---|---|
| 0 | 09/16/2004 | Initial release |

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

***How to Reach Us:***

**Home Page:**
www.freescale.com

**email:**
support@freescale.com

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

**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)
support@freescale.com

**Japan:**
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-0047 Japan
0120 191014
+81 3 3440 3569
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
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@
hibbertgroup.com

AN2750
Rev. 0
09/2004



freescale™
semiconductor