MOTOROLA

Sim_G4 v1.1.2 User's Guide

System Performance Modeling and Simulation
Networking and Computing Systems Group

Document 1998-007-0
Version 1.1.2
JR, KG

1 Overview

The Sim_G4 performance model was developed specifically to model the Motorola G4 microprocessor. Its main purpose is to model the cycle-by-cycle timing of the processor using only an instruction trace file as input. By not modeling data flow, large gains in execution speed are achieved, allowing Sim_G4 to process millions of traced instructions quickly.

1.1 Scope

The purpose of this document is to describe how to run the Sim_G4 model, and also how to customize its behavior through run time switches and parameters.

Section 2 describes how to invoke the Sim_G4 simulator with supported command line options. Customization of Sim_G4 using runtime parameters is covered in Section 3. Section 4 describes how to invoke the trace utilities built into Sim_G4 through command line options. Using the Macintosh GUI for Sim_G4 is detailed in Section 5. The last section, Section 6, describes the input instruction trace expectation.

2 Running Sim_G4

2.1 Usage

The Sim_G4 performance model's executable is called "`sim_g4`". Sim_G4 supports several command line parameters. Sim_G4 expects as input a binary trace file in TT6 format (the TT6 format is described in the document "TT6 Trace Format" available from the System Performance Modeling and Simulation group). Sim_G4 always reads its input from standard input (stdin). For example:

```
% gunzip -c mytrace.tt6.gz | sim_g4
% sim_g4 < mytrace.tt6
```

As shown in the example above, the first line is a unix program `gunzip` is used to uncompress the gzip'ed file `mytrace.tt6.gz`. The output of the `gunzip` program becomes the input of `sim_g4`. In the second line, the input file `mytrace.tt6` is used directly by `sim_g4`.

Both of the above examples are valid invocations of Sim_G4. Although the simulator is highly customizable through command line options (Section 2.2 "Command Line Options") and runtime parameters (Section 3 "Customizing Sim_G4 using Runtime Parameters"), it can still be run without any options.

By default, Sim_G4 outputs the statistical report when no options are selected. More detailed output can be generated by the use of command line options, discussed in the next section.

2.2 Command Line Options

Command line options control the behavior of the output from Sim_G4. To change the configuration of the processor being simulated, one can use runtime parameters found in Section 3 "Customizing Sim_G4 using Runtime Parameters".

Below is a listing of all command line parameters. Arguments for parameters are always shown in italics. Additionally, arguments for parameters that should be numbers will be followed by a '#' sign.

-h	Display all command line options, and exit
-v	Display current sim_g4 version number, and exit
-dp *clk#*	Display progress message every *clk#* number of clocks.
-oe *filename*	Specifies that *filename* will be used for error messages (default is stdout)

Table 1: General Options

The options below allow the specification of runtime parameters. For a listing of available runtime parameters, see Section 3.2 “Available Runtime Parameters”

-irf *filename*	Specifies that the file *filename* contains runtime parameters
-r *param=val*	Specifies the runtime parameter *param* with the value *val* (NOTE: There can be no spaces between the *param*, =, and *val*)
-rp	Print all supported run-time parameters, and exit

Table 2: Runtime Parameters

Sim_G4 also provides details about the completion cycles of each instruction. Table 4 “Completion Report Options” lists the options. Further details can be found in Section 2.2.2 "Completion Cycle Report"

In addition to the above methods to display the output of Sim_G4, three other pipeline views are available. The three pipeline views are horizontal, vertical, and wide vertical scrollpipe view. A complete list of all options for the scrollpipe output can be found in Table 3 “Scrollpipe Output Options” .

-sp *fname*	Enable scrollpipe output to file *fname*. To print to stdout enter a dash (‘-’) for *fname*.
-st *type*	Set scrollpipe *type* (0=Horizontal, 1=Vertical, 2=Wide Vertical)
-sw #	Set scrollpipe display to # characters wide (Horizontal only)
-sh	Print scrollpipe format and then exit.

Table 3: Scrollpipe Output Options

Further details can be found in Section 2.2.3 "Scrollpipe Output".

-oc *filename*	Generates a binary file called *filename* containing the completion cycle of each instruction.
-cf	Expands the output format to include the instruction address and opcode in the following order: instruction address, opcode, completion cycle
-ca	Indicates the output file specified by the **-oc** options should be an ASCII file rather than a binary file
-cfa	Same as using both **-cf** and **-ca**.

Table 4: Completion Report Options

-os *filename*	Specified that the file *filename* is the destination for all statistical output (default is stdout)

Table 5: Statistical Report Option

These options are discussed in Section 4 "Sim_G4 Built-In Trace Utilities".

-ltrace	Generates only a trace listing (simulation does not run)
-lo	Causes trace generated to include opcodes in the output (simulation does not run)
-ifreq	Report the frequency of instructions in the trace (simulation does not run)

Table 6: Trace Utility Options

Note: The following options only apply toScrollpipe Output and Instruction Trace Listing.

-pd *clk#*	Delays pipeline output for *clk#* clock cycles
-pdi *inst#*	Delays pipeline output for *inst#* instructions
-pdr *ret#*	Delays pipeline output for *ret#* retired instructions
-pe *clk#*	Ends pipeline output *clk#* clocks after simulation begins
-pei *inst#*	Ends pipeline output *inst#* instructions after simulation begins
-per *ret#*	Ends pipeline output after *ret#* retired instructions.
-pt	Terminate simulation upon reaching end of pipeline output

Table 7: Output Control Options

2.2.1 Bus Output

Detailed bus activity can be viewed using the **-b** flag. Because the bus output always prints to standard output, it will conflict with the pipeline output (if the pipeline display output is enabled). This can be resolved by redirecting the pipeline output using the **-op** option. For example,

```
% sim_g4 -b -op pipeline.out < mytrace.tt6
```

will redirect the bus activity output from `stdout` to the file `pipeline.out`.

An example of the bus output is shown below:

`% sim_g4 -b < mytrace.tt6`									
System Clock Number	*Address and type (Read/Write) of next Address tenure*			*Address currently on the address bus*		*Address of data for current data tenure*		*Address of valid data for current data tenure*	
85:	ARB:	....	R	ADDR:	....	DBUS:	....	DIN:	....
89:	ARB:	00001000	W	ADDR:	....	DBUS:	....	DIN:	....
93:	ARB:	00000120	R	ADDR:	00001000	DBUS:	....	DIN:	....
97:	ARB:	00000120	R	ADDR:	00001000	DBUS:	00001000	DIN:	....
101:	ARB:	....	R	ADDR:	00000120	DBUS:	00001000	DIN:	....
105:	ARB:	....	R	ADDR:	00000120	DBUS:	00001000	DIN:	....
109:	ARB:	....	R	ADDR:	....	DBUS:	00001000	DIN:	....
113:	ARB:	....	R	ADDR:	....	DBUS:	00001000	DIN:	....
117:	ARB:	....	R	ADDR:	....	DBUS:	00001000	DIN:	00001000
121:	ARB:	....	R	ADDR:	....	DBUS:	00000120	DIN:	00001000
125:	ARB:	00001300	W	ADDR:	....	DBUS:	00000120	DIN:	....
129:	ARB:	....	W	ADDR:	00001300	DBUS:	00000120	DIN:	00000120
133:	ARB:	....	W	ADDR:	00001300	DBUS:	00001300	DIN:	00000120
137:	ARB:	....	W	ADDR:	....	DBUS:	00001300	DIN:	....
%									

2.2.2 Completion Cycle Report

The completion cycle report is enabled with the **`-oc`** flag. This report contains the clock cycles during which instructions are completed. The default format is a binary format consisting of 4 bytes for each instruction completion cycle. The instruction order in the output file is identical to that of the order of the input instruction trace.

A more detailed format is also available (using the **`-cf`** option) that includes the instruction address, opcode and completion cycle.

Outputting the report in ASCII text is done by adding the **-ca** option. Using the ASCII output also reports the original instruction mnemonic. Using both the **-cf** and **-ca** options (or by using the **-cfa** option) would produce a file of the following format.

```
% sim_g4 -cfa -oc - < mytrace.tt6

Inst No.   Inst. Addr     Opcode          Mnemonic       Cycle Count
-----------------------------------------------------------------------
   0       0x06029528   0x60000000   [ori     R0,R0,0x0   ]    72
   1       0x0602952c   0x80010048   [lwz     R0,0x48(R1) ]   143
   2       0x06029530   0x38210040   [addi    R1,R1,0x40  ]   143
   3       0x06029534   0x7c0803a6   [mtspr   LR,R0       ]   144
   4       0x06029538   0x4e800020   [bclr+   20,0        ]    84
                                  .
                                  .
                                  .
%
```

2.2.3 Scrollpipe Output

One of three different scrolling pipelines can be chosen, horizontal, vertical, or wide vertical. The default type is horizontal. To specify a vertical scrollpipe output add "**-st 1**" to the command-line invocation of Sim_G4. Add "**-st 2**" to the command-line invocation to specify the wide vertical scrollpipe output.

2.2.4 Horizontal Scrollpipe

When enabled, the horizontal scrollpipe output will print the pipeline activity for each instruction executed. The output looks like:

```
10:bl      0x50            |   166 | FFFFFFFFFFFFR\\IIIIIIIIIIIIII|   254
11:mfspr   R0,LR           |   243 | ..IDEEEEEEEEER.............. |   255
12:mfcr    R12             |   243 | ..IIIIIIIIIIDR.............. |   256
+  +--------+-----------+     +---+  +-------------+--------------+   +-+-+
|           |                  |                  |                    |
|           |                  |                  |  Retire/Fold Cycle +
|           |                  |                  |
|           |                  |    Pipeline output+
|           |                  |
|           |                   +- Fetch Cycle
|           |
|           +- Instruction Mnemonic
|
+- Instruction number
```

Each character in the "Pipeline Output" represents one clock cycle. A period ('.') indicates the instruction was not active. A backslash ('\') indicates that the instruction was active for more clock cycles than the pipe display is wide, and the beginning cycles have been overwritten by later cycles. The text indicates what stage of the pipe the instruction was in. The following abbreviations are used:

I - Fetched - held in IB	D - Dispatched	E - Executing
F - Finished	R - Retired	@ - Folded

The width of the scrollpipe (number of characters) can be changed using the "**-sw #**" command-line option. Its default value is 30, with a minimum value of 10 and a maximum value of 300.

2.2.5 Vertical Scrollpipe

The vertical scroll pipe prints out one line every clock cycle (as long as instructions are in the Completion Buffer (CB)). It prints the status of each instruction using the same abbreviations listed above. As instructions are entered into the CB, they are printed to the right of the pipe, as well as the status of the dispatched instructions for that cycle:. For example:

```
1072 D -                  97:( 1071) vadduhm   V21,V16,V18   MAX
1072 - D                  98:( 1071) vmrghh    V17,V19,V2
1073 R R                                                    IB
1073
1074                                                        IB
1074
1075     D                99:( 1074) vmrglh    V18,V19,V2    VPUF
1075     -
1076     R D             100:( 1074) vmrghh    V16,V20,V1    IB
+--+ + + + +             +-+  +-+-+  +---+----------------+  +-+-+
 |   | | | |              |     |        |                     |
 |   | | | |              |     |        + Mnemonic            |
 |   | | | |              |     |            Dispatch Status* -+
 |   | | | |              |     +- Cycle Fetched
 |   | | | |              +- Instruction Number
 |   | | | |
 |   | | | +- Status of instruction #100 dispatched on clock 1076
 |   | | +- Status of instruction #99 dispatched on clock 1075
 |   | +- Status of instruction #98 dispatched on clock 1073
 |   +- Status of instruction #97, dispatched on clock 1072
 +- Clock Cycle
```

The dispatch status is an abbreviation with the following meanings:

IB - Insufficient IB Insts
DS - Dispatch Serialization
CB - Completion Buffer Full
GPR - Maximum Rename GRP's Reached
FPR - Maximum Rename FPR's Reached
VAUB - VAU Unit Busy
VAUF - VAU Nonemtpy
DLS - Double Load/Store
LSUB - LSU Busy
MAX - Maximum Allowed Dispatched
MD - Mispredict Drain
TS - Tail Serialization
SPEC - Maximum Speculation Reached
FXUB - Fixed Point Unit Busy
FPUB - FPU Unit Busy
VR - Maximum Rename VR's Reached
VPUF - VPU Nonempty
LSUF - LSU Full
SUB - System Unit Busy

It is possible to have the maximum allowed dispatch and only see one instruction actually dispatched. This is the case when either a branch or no-op instruction was folded out of one of the bottom two instruction dispatch buffers.

2.2.6 Wide Vertical Scrollpipe

The wide vertical scroll pipe has the same information as the regular vertical scroll pipe. The difference is that instead of printing two lines per clock, the vertical scroll pipe prints only one line per clock cycle. This causes the lines printed to be very wide (almost 110 characters wide).

3 Customizing Sim_G4 using Runtime Parameters

3.1 Overview

Sim_G4 can be customized using runtime parameters. Runtime parameters can be specified one at a time on the command line (using the **-r** option), or can be placed in a file and read at execution time (using the **-irf** option). The **-irf** option can only be used once, but the **-r** option can be used to specifiy multiple runtime parameters.

3.2 Available Runtime Parameters

The available runtime parameters can be displayed by invoking Sim_G4 with the **-rp** option. These options are summarized in the table below:

	Name	Default	Description
1	bus_mode[a]	1	External Bus Mode (1 => native bus - enhanced 60x Bus, 0 => 60x Bus)
2	g4_bus_fraction_ numerator[b]	4	Number of internal clocks per 1 or 2 bus clock
3	g4_bus_fraction_ denominator[b]	1	1 => numerator == full processor clocks, 2 => half-processor clocks
4	l2_bus_fraction_ numerator[c]	2	Number of internal clocks per 1 or 2 L2 bus clock
5	l2_bus_fraction_ denominator[c]	1	1 => numerator == full processor clocks, 2 => half-processor clocks
6	l2_size	1024	Size of the L2 (in K bytes). Valid choices are 512, 1024, or 2048.
7	l2_disable	0	1=> Disables the L2 Cache, 0=> enables the L2 Cache
8	l2_sram_latency	3	Latency for the first beat of an access to L2 SRAMs (max: 10)
9	mem_controller[d]	2	External Memory Controller (1=MPC106,2=Hypothetical)
10	dram_type[d]	2	Type of DRAM (1=EDO, 2=SDRAM)
11	dram_row_bits[d]	12	Number address bits assigned to DRAM Row address
12	dram_col_bits[d]	10	Number address bits assigned to DRAM Column address
13	sdram_bank_bits[d]	2	Number of address bits used to index SDRAM device banks
14	sdram_close_latency[d]	2	Latency associated with the closing of a SDRAM device bank
15	sdram_hit_latency[d]	5	Latency caused by a hit to an open SDRAM device bank
16	sdram_miss_latency[d]	11	Latency caused by a miss to a closed SDRAM device bank
17	sdram_burst_latency[d]	1	Latency for burst portions of a data transaction
18	sdram_min_latency[d]	2	Minimum latency between back to back pipelined data transactions

Table 8: Sim_G4 Runtime Parameters

	Name	Default	Description
19	sdram_max_open_pages[d]	2	Maximum number of SDRAM pages that can be kept open by the Memory Controller
20	edo_hit_latency[d]	4	Latency caused by a hit to an EDO memory device
21	edo_miss_latency[d]	8	Latency caused by a miss to an EDO memory device
22	edo_burst_latency[d]	3	Latency for burst portions of a data transaction
23	vscr_nj	1	AltiVec Floating Point Mode (0 => do not use java mode for AltiVec floating point operations, 1 => use java mode for AltiVec floating point ops).
24	ibat0_enabled	0	Set to "1" to enable Instruction Block Address Translation (IBAT) 0
25	ibat0_size	003fffff	Set IBAT0 Block Size
26	ibat0_addr	40000000	Set IBAT0 Block Address
27	ibat0_ci	0	Set IBAT0 to Cache Inhibited
28	ibat1_enabled	0	Set to "1" to enable Instruction Block Address Translation (IBAT) 1
29	ibat1_size	00000000	Set IBAT1 Block Size
30	ibat1_addr	00000000	Set IBAT1 Block Address
31	ibat1_ci	0	Set IBAT1 to Cache Inhibited
32	ibat2_enabled	0	Set to "1" to enable Instruction Block Address Translation (IBAT) 2
33	ibat2_size	00000000	Set IBAT2 Block Size
34	ibat2_addr	00000000	Set IBAT2 Block Address
35	ibat2_ci	0	Set IBAT2 to Cache Inhibited
36	ibat3_enabled	0	Set to "1" to enable Instruction Block Address Translation (IBAT) 3
37	ibat3_size	00000000	Set IBAT0 Block Size
38	ibat3_addr	00000000	Set IBAT0 Block Address
39	ibat3_ci	0	Set IBAT0 to Cache Inhibited
40	dbat0_enabled	0	Set to "1" to enable Data Block Address Translation (DBAT) 0
41	dbat0_size	003fffff	Set DBAT0 Block Size
42	dbat0_addr	40000000	Set DBAT0 Block Address
43	dbat0_ci	0	Set DBAT0 to Cache Inhibited
44	dbat0_wt	0	Set DBAT0 to Write Through
45	dbat1_enabled	0	Set to "1" to enable Data Block Address Translation (DBAT) 1
46	dbat1_size	ffffffff	Set DBAT1 Block Size
47	dbat1_addr	00000000	Set DBAT1 Block Address
48	dbat1_ci	0	Set DBAT1 to Cache Inhibited
49	dbat1_wt	1	Set DBAT1 to Write Through

Table 8: Sim_G4 Runtime Parameters

	Name	Default	Description
50	dbat2_enabled	0	Set to "1" to enable Data Block Address Translation (DBAT) 2
51	dbat2_size	00000000	Set DBAT2 Block Size
52	dbat2_addr	00000000	Set DBAT2 Block Address
53	dbat2_ci	0	Set DBAT2 to Cache Inhibited
54	dbat2_wt	0	Set DBAT2 to Write Through
55	dbat3_enabled	0	Set to "1" to enable Data Block Address Translation (DBAT) 3
56	dbat3_size	00000000	Set DBAT0 Block Size
57	dbat3_addr	00000000	Set DBAT0 Block Address
58	dbat3_ci	0	Set DBAT0 to Cache Inhibited
59	dbat3_wt	0	Set DBAT0 to Write Through
60	warmup[e]	0	1 => L1 and L2 Caches and TLBs warmed up, 0 => Caches and TLBs not warmed up
61	warmup_l1[e]	0	1=>Instruction and Data L1 Cache and TLBs warmed up, 0 => Instruction and Data L1 Cache and TLBs not warmed up
62	warmup_Il1[e]	0	1=> Instruction L1 Cache and TLBs warmed up, 0 => Instruction L1 Cache and TLBs not warmed up
63	warmup_Dl1[e]	0	1=> Data L1 Cache and TLBs warmed up, 0 =>Data L1 Cache and TLBs not warmed up
64	warmup_l2[e]	0	1=> L2 Cache warmed up, 0 => L2 Cache not warmed up

Table 8: Sim_G4 Runtime Parameters

a. The native bus mode allows back-to-back address tenures to occur, whereas the 60x bus mode requires an idle cycle in between address tenures. In addition, the native bus also supports data streaming, whereby back-to-back data tenures of the same type (back-to-back reads or back-to-back writes) can occur. The 60x bus requires an idle cycle between back-to-back data tenures.

b. To specify a processor-to-bus ratio of X:Y use *g4_bus_fraction_numerator* to specify X and *g4_bus_fraction_denominator* to specify Y.

c. The parameters *l2_bus_fraction_numerator* and *l2_bus_fraction_denomiator* are used to specify the processor-to-l2 clock ratio in the same manner that *g4_bus_fraction_numerator* and *g4_bus_fraction_denominator* are used to specify the processor-to-60xbus clock ratio.

d. See Section 3.4 "Memory Related Runtime Parameters".

e. When a TLB and/or Cache is warmed up, all accesses to invalid entries will be simulated as if the entry was valid, and the entry will become valid. Hits or collisions to valid entries are unaffected by warmup.

3.3 Specifying Runtime Parameters

To specify a parameter via the command line, the **-r** option is used. For example,

```
% sim_g4 [....] -r mem_controller=1 -r warmup_l2=0 [....]
```

will set the `mem_controller` parameter to a value of '1' (this sets the memory controller to be the MPC 106 controller) and set the `warmup_l2` parameter to a value of "0" (this means L2 Cache not warmed up).

To specify a group of runtime parameters in a file, the **-irf** option is used. Each line in the file is used to set one parameter. The format for each line is:

```
PARAMETER    VALUE
```

Blank lines in the file are ignored, as are lines beginning with a '#' (there cannot be any spaces between the start of the line and the '#' character). There can be any amount of whitespace between the '`PARAMETER`' and its assigned '`VALUE`', but there is not an '=' sign as there was for the command line options. An example file is shown below:

```
# Specify the memory controller - 1 = MPC 106, 2 = Hypothetical
mem_controller 1

# Specify and setup DRAM
# DRAM type 1 = EDO, 2 = SDRAM
dram_type 2

# Number address bits assigned to DRAM Row address
dram_row_bits 12

# Number address bits assigned to DRAM Column address
dram_col_bits 10
```

Note: Runtime Parameter Ordering

Sim_G4 processes command line parameters in the order they are read. Therefore, if one runtime parameter overrides another runtime parameter, the parameter which came later (either on the command line or from a configuration file) will be used. No warning message is generated when this happens, and it is up to the user to ensure that conflicting parameters are not specified.

3.4 Memory Related Runtime Parameters

Sim_G4 models two memory controllers for its interface to DRAM: MPC 106 (known as Grackle) and a hypothetical controller (see Section 3.4.4 "Differences between MPC 106 and hypothetical memory controllers" for a discussion of the different memory controller models). The controller type is set using the `mem_controller` runtime parameter. For the MPC 106, use a value of "1", and for the hypothetical controller, use a value of "2". The default configuration uses MPC 106.

Two types of DRAM are supported: Extended Data Output (EDO), and Synchronous DRAM (SDRAM). The memory type is changed using the runtime parameter `dram_type`. Specifying a `dram_type` of "1" specifies EDO memory, while "2" corresponds to SDRAM memory.

Sim_G4 allows the setting of the memory device sizes. This is specified in terms of the number of row and column bits, and in the case of SDRAMs also the number of bank bits. For all memory types, setting the number of row and column bits are done with the `dram_row_bits` and `dram_col_bits` parameters, respectively. For instance, for 4M x 4 EDO DRAMs, the following settings should be used:

```
dram_row_bits 12
dram_col_bits 10
```

Note: For SDRAM models, the value of `dram_col_bits` is calculated in Sim_G4 and may be modified as noted in Section 3.4.3.

3.4.1 EDO Specific Parameters

It is up to the user to specify the correct latencies characteristic for the EDO memory type using the following set of parameters:

Parameter	Description
edo_miss_latency	This is the latency (in clocks) for accesses that do not hit in the currently open page within a particular memory bank. This occurs when the row address of the current access does not match the row address of the previous access to this memory bank.
edo_hit_latency	This is the latency (in clocks) for accesses that hit in the currently open page within a particular memory bank.
edo_burst_latency	This is the latency (in clocks) between beats after the first beat.

Table 9: EDO Memory Parameters

3.4.2 SDRAM Specific Parameters

SDRAMs employ independent internal memory array banks, allowing address decode to one bank in parallel while reading from another bank. SDRAMs come in both 2 bank and 4 bank designs. Each internal bank works independently, thus can store its own row decode such that subsequent access to this same row (known as a "page hit") will have a much lower latency. If an

access to a particular internal bank is not to the same row than the previous access to this bank, it is considered a "page miss". In addition, since the bank is currently selecting a different row, the memory controller must send a command to pre-charge the bank so the new address can be applied. This extra latency is known as the "page close" latency. The only page miss not subject to page close is the very first access to a bank (in a real system pages are automatically closed during refresh cycles, but refresh is not modeled in Sim_G4). Memory controllers also have a limit on how many pages can be kept open at any time. This limit can be modified using the parameter `sdram_max_open_pages`. Sim_G4 uses a LRU page replacement policy to select a victim SDRAM page to be closed if the limit has been reached.

The following parameters are specific to SDRAMs:

Parameter	Description
sdram_bank_bits	Number of address bits used to address internal SDRAM banks.
sdram_close_latency	This is the latency (in clocks) caused by having to pre-charge a bank within the SDRAM before re-accessing it with a new row address.
sdram_miss_latency	This is the latency (in clocks) incurred when accessing a bank with a different row address than was previously used.
sdram_hit_latency	This is the latency (in clocks) for accessing a bank with the same row address as was previously used.
sdram_burst_latency	This is the latency (in clocks) between beats after the first beat.
sdram_max_open_pages	This is the maximim number of pages that can be kept open at any given time.

Table 10: SDRAM Memory Parameters

3.4.3 Important Notes on how Sim_G4 models the memory system

By default, memory space is assumed to be fully populated to 4GB of DRAM. You can choose the type of DRAM, but all 4GB must be made up of the same type. However, for SDRAMs, this number can change depending upon the type of memory controller being used. When using MPC106 in the Sim_G4 model, only the lowest 1GB of physical memory can be addressed; addresses over 1GB wrap around to the lowest 1GB. The hypothetical memory controller modeled in Sim_G4 can address the lowest 2GB of memory; addresses over 2GB wrap around to the lowest 2GB.

When attached to a 64-bit bus that can transfer 8 bytes of data in a single beat, both memory controllers use the bottom 3 bits as the byte offset.

For EDO systems, Sim-G4 uses this page offset and the column bits to determine the page size for page hits and misses. Additionally, the number of memory banks is not set, but rather it varies based on the size of the memory chips used (that is, its based on the values of `dram_row_bits`, `dram_column_bits`. The number of memory banks is computed as follows: memory banks = 4GBytes / bytes per bank. In most cases, the number of memory banks modeled will exceed that which is supported by MPC106 or the hypothetical controller (eight memory banks). The assumption is that no single application's memory access will span more than eight memory banks.

For SDRAMs, Sim_G4 assumes that the memory is populated using (`dram_row_bits * dram_col_bits * sdram_bank_bits`) chips (e.g., 12x10x2) that are 8 bits wide. Sim_G4 assumes a fixed 2KB page size (lowest 11 bits, using the byte offset and some column bits) for MPC106, and a variable page size for the hypothetical memory controller where the column bits and the byte offset give us the page size. The number of memory banks is set to 8, which is the maximum that is supported by MPC106 or the hypothetical controller.

3.4.4 Differences between MPC 106 and hypothetical memory controllers

In Sim_G4, a 32-bit address that is sent to the MPC106 breaks down as follows for SDRAM:

0 1	2 4	5 9	10 20	21 28	29 31
00	Memory Bank	Spill-over row and column bits	Bank Sel Bits	Row Bits	Column Bits	Byte Offset

Sim_G4 assumes that bits 10 through 20 are used for the lowest row bits. Because the page size is limited to 2KB, only the lowest 8 column bits are used to determine page hits and misses. The remaining row and column bits spill to the left of the bank select bits.

The following table gives the valid values for the above bits for MPC 106. Although Sim_G4 will support configurations not listed in the table, the table does describes the actual hardware supported ranges.

Address Bits	Corresponding Sim_G4 Parameter	Valid Range for EDO Memory	Valid Range for SDRAM Memory	Supported Sim_G4 Range
Byte Offset	N/A	3[a]	3[a]	N/A
Column Bits	dram_col_bits	9-12	9-12	1-15
Row Bits	dram_row_bits	9-13	12[a]	1-15
Bank Sel Bits	sdram_bank_bits	N/A	1-2[a]	1-2

Table 11: Valid Memory Parameters for the MPC 106 Memory Controller

a. When using the MPC 106, sdram_min_latency is forced to 2, g4_bus_width is forced to 64, and g4_bus_mode is forced to 0 (60xbus).

Sim_G4 breaks down a 32-bit SDRAM address that is sent to the hypothetical memory controller as follows:

0	1	4			31
0	Memory Bank	Row Bits	Bank Sel Bits	Column Bits	Byte Offset

Sim_G4 assumes that the bank select bits fall between the row and the column bits. It does not force any page size, which can vary depending upon the byte offset (i.e., the bus width) and the size of SDRAM chips chosen by the user.

The following table gives the valid values for the above bits for the hypothetical memory controller. Although Sim_G4 will support configurations not listed in the table, the table does describes the actual hardware supported ranges.

Address Bits	Corresponding Sim_G4 Parameter	Valid Range for EDO Memory	Valid Range for SDRAM Memory	Supported Sim_G4 Range
Byte Offset	N/A	3	3	N/A
Column Bits	dram_col_bits	8-13	7-11	1-15
Row Bits	dram_row_bits	10-14	11-13	1-15
Bank Sel Bits	sdram_bank_bits	N/A	1-2	0-2

Table 12: Valid Memory Parameters for the Hypothetical Memory Controller

4 Sim_G4 Built-In Trace Utilities

4.1 Overview

This section describes how to invoke the trace utilities built into Sim_G4. These utilities are accessed through command line options.

4.2 Usage

For the Instruction Trace Lister Utility:

```
% sim_g4 -ltrace [options] < input_trace.tt6
```

For the Instruction Frequency Count Utility:

```
% sim_g4 -ifreq [options] < input_trace.tt6
```

Where: *options* are described below
Trace files are expected through standard input in TT6 format.
Report listings are sent to standard output.

The *options* supported by the traces utilities are found in Section 2.2 "Command Line Options".

4.3 Instruction Trace Lister

This utility (invoked with the **-ltrace** command) simply converts a binary trace file into a readable disassembled form. An example of the output is given below:

```
Inst #  Inst. Address       Mnemonic            Data Address
   0    [0x100001C8]     lwz     R18,0x0(R2)      0x30030774
   1    [0x100001CC]     lwz     R7,0x8(R18)      0x30013BD0
   2    [0x100001D0]     addi    R8,R0,0x0
   3    [0x100001D4]     stw     R8,0x0(R7)       0x30030C4C
   4    [0x100001D8]     lwz     R9,0x0(R18)      0x30013BC8
```

Note: An alternate method to invoke this utility is to create a link of the name "`ltrace`" to Sim_G4. Using this method, the **-ltrace** option is not necessary. For example:

```
% ln -s sim_g4 ltraces
% ltraces -lo < mytrace.tt6
```

4.4 Instruction Frequency Counter

This utility goes through the entire trace file counting the number of occurrences for each instruction. At the end it outputs the results. An example of the output report is given below:

```
Primary Op  Extend Op          Mnemonic              # of Occurrences
   PO: 14                      addi    R0,R0,0x0     Frequency: 367273
   PO: 15                      addis   R0,R0,0x0     Frequency: 3918
   PO: 16                      bc-     0,0,0x0       Frequency: 1203458
   PO: 18                      b       0x0           Frequency: 402796
   PO: 19     XO: 0    Rc=0    mcrf    0,0           Frequency: 3
   PO: 19     XO: 16   Rc=0    bclr-   0,0           Frequency: 324551
```

Any unrecognized mnemonics will be displayed as a hexadecimal number in the Mnemonic column.

Note: An alternate method to invoke this utility is to create a link of the name "`ifreq`" to Sim_G4. Using this method, the **-ifreq** option is not necessary. For example:

```
% ln -s sim_g4 ifreq
% ifreq < mytrace.tt6
```

5 Using Sim_G4 on the Macintosh

Although Sim_G4 was originally designed to use the command line interface described in the previous sections, a GUI version of the tool has been generated that runs on Macintosh Platforms under MacOS 8.x. This version allows the same configurations as the command line version.

The Configuration Menu (shown below) can be used to specify either command line options (Section 2.2 "Command Line Options") or simulation runtime parameters (Section 3 "Customizing Sim_G4 using Runtime Parameters"). The following three graphics detail the Configuration pulldown menu, the command line options dialog box, and the simulator parameters dialog box.

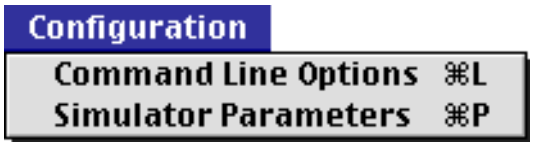

Figure 1: Configuration Pulldown Menu

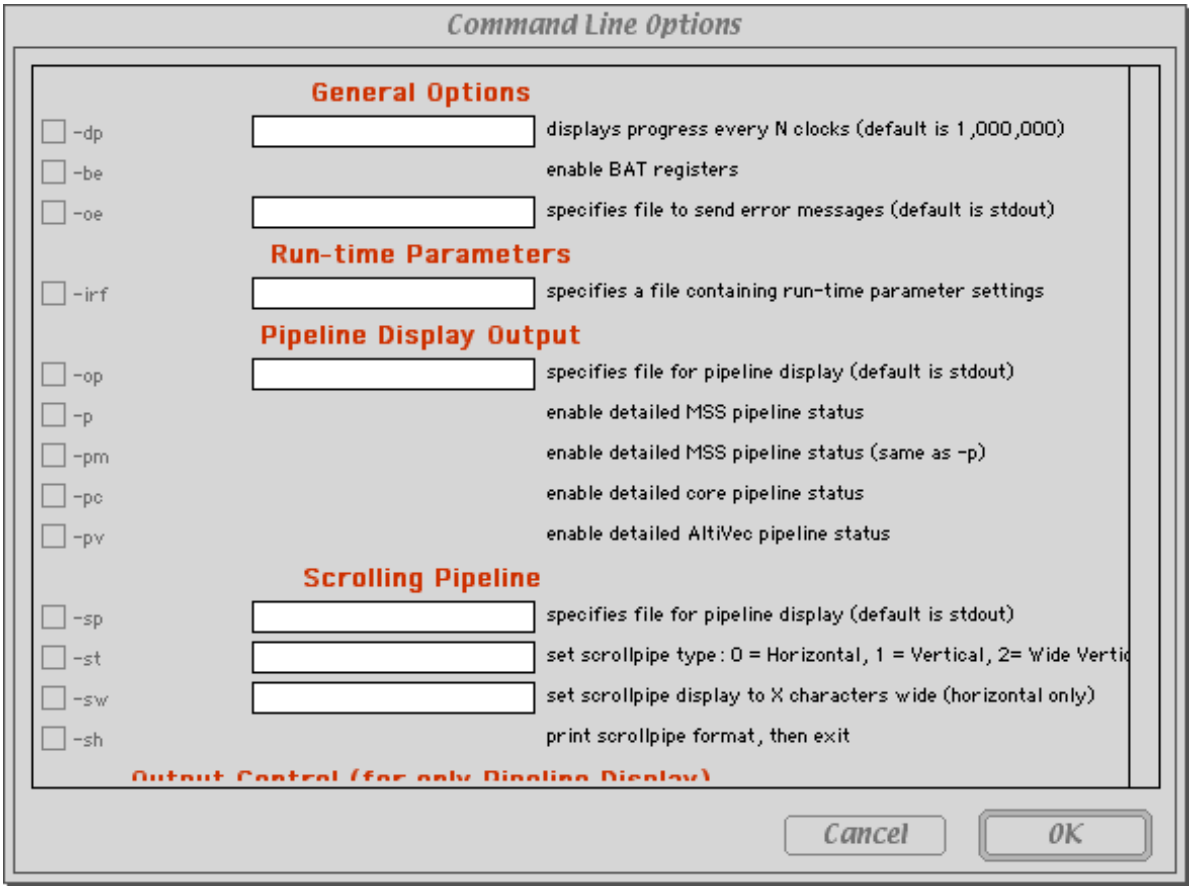

Figure 2: Command Line Option Dialog Box

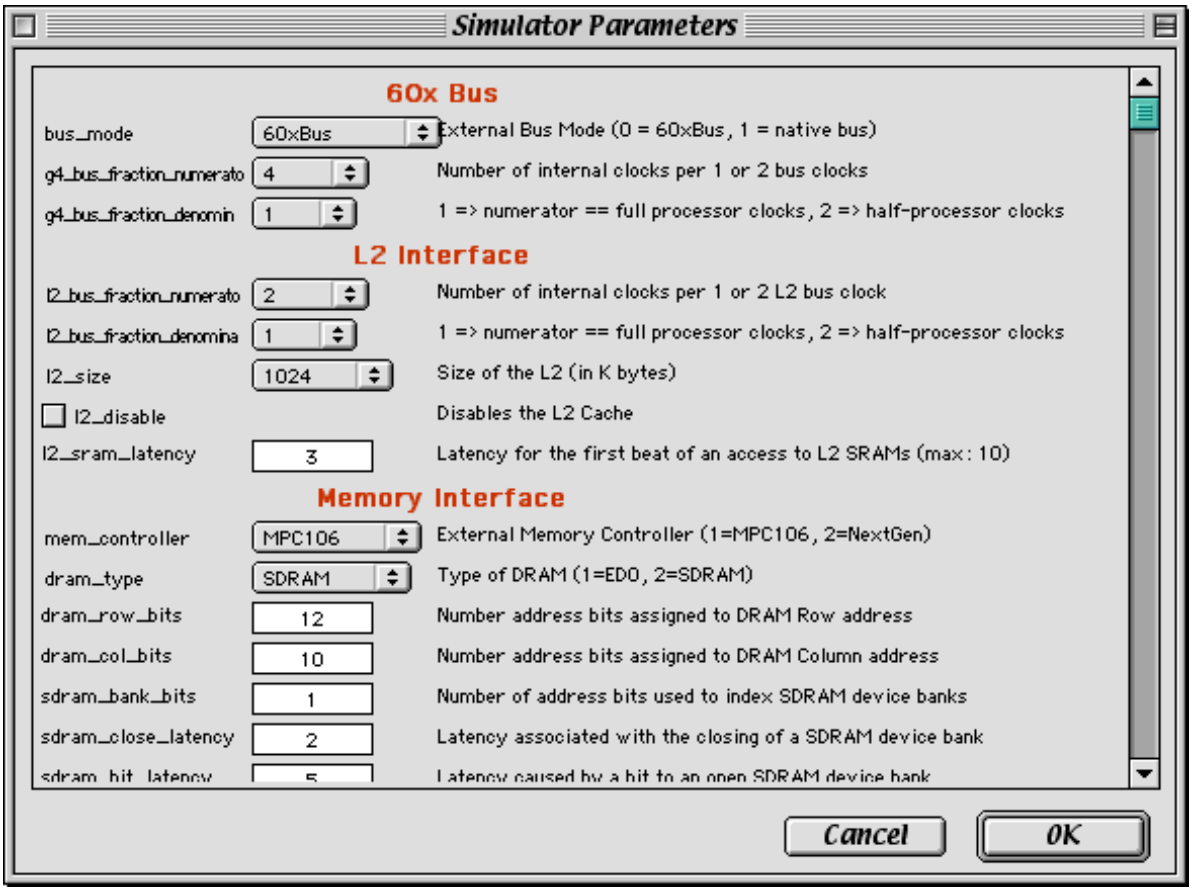

Figure 3: Simulation Parameters Dialog Box

After specifying the desired command line options and runtime parameters the File pulldown menu is used to choose an input tt6 file. It can also be used to specify an Apple Event Stream, but use of this stream is beyond the scope of this document. The following picture details the File pulldown menu.

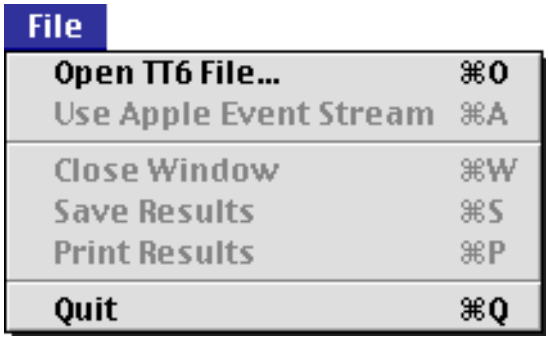

Figure 4: File Pulldown Menu

Upon selecting the input source, a progress bar (shown below) is displayed indicating the percentage of the input source that has been processed.

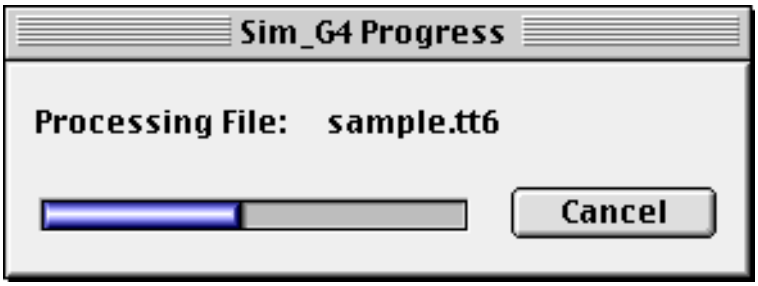

Figure 5: Sim_G4 Progress Bar

When the simulation is complete, the results are displayed in a text window. The actual results displayed will vary according to the chosen command line options. One such example is shown below.

sample.tt6.out

```
Clocks: 1510  Retired: 77  Folded: 5  IPC=  0.0543

Instruction Flow Stats:
  Fetched: 82  Dispatched: 79  Retired: 77  Branches Folded: 5


Dispatch Stalls (in evaluation order):
  Drain:          0.00% (0)
  IB_empty:      90.53% (1367)
  CB_full:        0.00% (0)
  GPR_rename:     0.00% (0)
  FPR_rename:     0.00% (0)
  VR_rename:      0.00% (0)
  Unit_busy:      3.84% (58)
    FXU1:                    0.07% (1)
    FXU2:                    0.00% (0)
    FPU:                     0.00% (0)
    VAUS:                    0.00% (0)
    VAUC:                    0.00% (0)
    VAUF:                    0.00% (0)
    VPU:                     3.77% (57)
    SYS:                     0.00% (0)
    LSU:                     0.00% (0)
  Disp_serial:    0.00% (0)
  Tail_serial:    0.00% (0)

Execution Unit Stats/Stalls:
  FXU1: idle:  98.81%  dispatch:        17  depend_stall:         0  ser_stall:         0
  FXU2: idle:  99.87%  dispatch:         1  depend_stall:         0  ser_stall:         0
  FPU:  idle:  99.93%  dispatch:         0  depend_stall:         0  ser_stall:         0
cb1_stall:          0
  VAU:  double dispatch attempts:         1
  VAUS: idle:  99.87%  dispatch:         1  depend_stall:         0  ser_stall:         0
  VAUC: idle:  99.93%  dispatch:         0  depend_stall:         0  ser_stall:         0
```

Figure 6: Sample output

6 Instruction Trace Format (TT6)

The expected input to the Sim_G4 model is an instruction trace format known as TT6. This section does not cover the complete TT6 format, but only the subset that Sim_G4 recognizes.

In brief, a TT6 trace file contains a stream of **non-speculative** instructions in the order they were retired. Some instructions also include additional words such as: operand address for load/store instructions, number of bytes in the case of certain load/store string instructions; block size and stride in the case of data stream touch instructions; and target address for branch instructions). All trace items are 4 bytes long (1 word) and are stored in a binary form (to make the file more compact).