1 SIS version 3.0.1 manual

NAME

sis - SPARC instruction set simulator version 3.0.1

SYNOPSYS

sis, sis64 [**-c** *file*] [**-nfp**] [**-ift**] [**-wrp**] [**-rom8**] [**-sparclite**] [**-uart1** *device*] [**-uart2** *device*] [**-fast_uart**] [**-uben**] [**-ram** *ram_size*] [**-rom** *rom_size*] [**-freq** *system_clock*] [**-wdfreq** *watchdog_clock*] *input_files*

sparc-rtems-gdb *input_files*
sparc-rtems-gdb64 *input_files*

DESCRIPTION

The SIS is a SPARC V7 architecture simulator configured to emulates an ERC32 system with up to 32 Mbyte ram and 4 Mbyte rom. Two versions of the simulator is provided; sis and sis64. The standard version (**sis**) uses 32-bit time and has a simulation time limit of 2^{32} clock ticks (= 5 minutes at 14 MHz). **sis64** uses 64-bit time and has virtually unlimited simulation time, it runs however about 20% slower than **sis**.

COMMAND LINE OPTIONS

-c *file* Reads commands from *file* instead of stdin. If the file .sisrc exists in the home directory, it will be automatically be executed.

-nfp Disables the FPU to emulate system without FP hardware. Each FP instruction will generate an FP-disabled trap.

-wrp Sets the prom to be writable (ROMWRT input on the MEC).

-rom8 By default, the prom area is considered to be 32-bit. Specifying -rom8 will simulate a 8-bit prom. The only visible difference is in the instruction timing.

-sparclite

Enables two sparclite instructions, SMUL and DIVSCC.

-uart[1,2] *device*

By default, UARTA is connected to stdin/stdout and UARTB is disconnected. This switch can be used to connect the uarts to other devices. E.g., ‘-uart1 /dev/ptypc’ will attach UART A to pseudo-device ptypc. Use ‘tip /dev/ttypc’ to connect to it

-uben Reverses the default UART setting. i.e UARTB is connected to stdin/stdout while UARTA is disconnected. Often required when simulating programs compiled with Aonix (Alsys) Ada.

-fast_uart

Run UARTS at infinite speed, rather than with correct (slow) timing.

-freq *system_clock*

Sets the simulated system clock (MHz). Default is 14.

-wdfreq *watchdog_clock*

Sets the simulated watchdog clock (MHz). Default is the same value as the system clock.

-ram *ram_size*

Sets the amount of simulated RAM (Kbyte). Default is 4096, maximum is 32768.

-rom *rom_size*

Sets the amount of simulated ROM (Kbyte). Default is 2048, maximum is 16384.

input_files

Each input file is loaded into the emulated memory according to the entry point for each segment. Recognized formats are aout, srecords and tektronix hex.

COMMANDS

Below is description of commands that are recognized by the simulator. The command-line is parsed using GNU readline. A command history of 64 commands is maintained. Use the up/down arrows to recall previous commands. For more details, see the readline documentation.

batch *file* Execute a batch file of SIS commands.

+**bp** *address* Adds an breakpoint at *address*.

bp Prints all breakpoints

-bp *num* Deletes breakpoint *num*.

cont [*count*]

tcont *[time]* Continue execution at present position, optionally for *count* instructions or for *time* time.

dis [*addr*] [*count*]

Disassemble [*count*] instructions at address [*addr*]. Default values for count is 16 and *addr* is the program counter address.

echo *string* Print <string> to the simulator window.

float Prints the FPU registers

go [*address*] [*count*]

tgo [*address*] [*time*]

The **go** command will set pc to *address* and npc to *address* + 4, and resume execution. No other initialisation will be done. If *count* is specified, execution will stop after the specified number of instructions. If address is not given, the default load address will be assumed. The **tgo** command will optionally resume execution until *time* is reached. See **tlim** on how to specify the time.

help Print a small help menu for the SIS commands.

hist [*length*] Enable the instruction trace buffer. The *length* last executed instructions will be placed in the trace buffer. A **hist** command without *length* will display the trace buffer. Specifying a zero trace length will disable the trace buffer.

load *files* Load *files* into simulator memory.

mec Display MEC registers. Some write-only registers are also displayed, these are marked with an asterix ('*').

mem [*addr*] [*count*]

Display memory at *addr* for *count* bytes. Same default values as for **dis**. The **mem** command can also be used to display MEC registers (address 0x1f80000). Unimplemented registers are displayed as zero.

quit Exits the simulator.

perf [*reset*] The **perf** command will display various execution statistics. A 'perf reset' command will reset the statistics. This can be used if statistics shall be calculated only over a part of the program. The **run** and **reset** command also resets the statistic information.

reg [*reg_name value*]

Prints and sets the IU registers in the current register window. **reg** without parameters prints the IU registers. **reg** *reg_name value* sets the corresponding register to *value*. Valid register names are psr, tbr, wim, y, g1-g7, o0-o7 and l0-l7. To view the other register windows, use **reg** *w*n, where n is 0 - 7.

reset Performs a power-on reset. This command is equal to **run** 0.

run [*count*]

trun [*time*] Resets the simulator and starts execution from address 0. If an instruction *count* is given, the simulator will stop after the specified number of instructions. The event queue is emptied but any set breakpoints remain. **trun** command will execute until *time* is reached. See **tlim** on how to specify the time.

step Equal to **trace** 1.

tlim <*time*> Limit the simulated time. Will stop a running simulator after *time*. The time parameter is relative to the current time. The time is given in micro-seconds, but can also be given in milli-seconds, seconds or minutes by adding 'ms', 's' or 'min' to the time expression. Example: tlim 400 ms.

tra [*count*] Starts the simulator at the present position and prints each instruction it executes. If an *count* is given, the simulator will stop after the specified number of instructions.

wmem <*address*> <*value*>

Write simulated memory. Only full 32-bit words can be written.

Typing a 'Ctrl-C' will interrupt a running simulator. Short forms of the commands are allowed, e.g **c**, **co**, or **con**, are all interpreted as **cont**.

TIMING

The SIS emulates the behaviour of the TSC691E and TSC692E SPARC IU and FPU from Temic/MHS. These are roughly equivalent to the Cypress 7C601 and 7C602. The simulator is cycle true, i.e a simulator time is maintained and incremented according the IU and FPU instruction timing. The parallel execution between the IU and FPU is modelled, as well as stalls due to operand dependencies (IU & FPU). Tracing using the **trace** command will display the current simulator time in the left column. This time indicates when the instruction is fetched. If a dependency is detected, the following fetch will be delayed until the conflict is resolved. Below is a table describing the instruction timing with no resource dependencies:

Instruction	Cycles	Instruction	Cycles
jmpl, rett	2	sqrts	37
load	2	fsqrtd	65
store	3	fsubs	4
load double	3	fsubd	4
store double	4	fdtoi	7
other integer inst	1	fdots	3
fabs	2	fitos	6
fadds	4	fitod	6
faddd	4	fstod	2
fcmps	4		
fcmpd	4		
fdivs	20		
fdivd	35		
fmovs	2		
fmuls	5		
fmuld	9		
fnegs	2		

FPU

The simulator maps floating-point operations on the hosts floating point capabilities. This means that accuracy and generation of IEEE exceptions is host dependent. The FPU instruction timing above indicates average cycle counts, the simulator implements (to some extent) data-dependant execution timing as in the real FPU.

MEC EMULATION

The following list outlines the implemented MEC registers:

Register	Address	Status
MEC control register	0x01f80000	implemented
Software reset register	0x01f80004	implemented
Power-down register	0x01f80008	implemented
Memory configuration register	0x01f80010	partly implemented
IO configuration register	0x01f80014	implemented
Waitstate configuration register	0x01f80018	implemented
Access protection base register 1	0x01f80020	implemented

Access protection end register 1	0x01f80024	implemented
Access protection base register 2	0x01f80028	implemented
Access protection end register 2	0x01f8002c	implemented
Interrupt shape register	0x01f80044	implemented
Interrupt pending register	0x01f80048	implemented
Interrupt mask register	0x01f8004c	implemented
Interrupt clear register	0x01f80050	implemented
Interrupt force register	0x01f80054	implemented
Watchdog acknowledge register	0x01f80060	implemented
Watchdog trap door register	0x01f80064	implemented
RTC counter register	0x01f80080	implemented
RTC counter program register	0x01f80080	implemented
RTC scaler register	0x01f80084	implemented
RTC scaler program register	0x01f80084	implemented
GPT counter register	0x01f80088	implemented
GPT counter program register	0x01f80088	implemented
GPT scaler register	0x01f8008c	implemented
GPT scaler program register	0x01f8008c	implemented
Timer control register	0x01f80098	implemented
System fault status register	0x01f800A0	implemented
First failing address register	0x01f800A4	implemented
Error and reset status register	0x01f800B0	implemented
Test control register	0x01f800D0	implemented
UART A RX/TX register	0x01f800E0	implemented
UART B RX/TX register	0x01f800E4	implemented
UART status register	0x01f800E8	implemented

The MEC registers can be displayed with the **mec** command, or using **mem** (‘mem 0x1f80000 256’). The registers can also be written using **wmem** (e.g. ‘wmem 0x1f80000 0x1234’). When written, care has to be taken not to write an unimplemented register bit with ‘1’, or a MEC parity error will occur.

UARTS

The UARTS operate with correct timing as defined in the MEC control register when the baudrate is programmed in the MEC control register. If the baudrate is left at the default value, or if the -fast_uart switch was used, the UARTS operate at infinite speed. This means that the transmitter holding register always is empty and a transmitter empty interrupt is generated directly after each write to the transmitter data register. The receivers can never overflow or generate errors.

Note that with correct UART timing, it is possible that the last character of a program is not displayed on the console. This can happen if the program forces the IU in error mode, there by terminating the simulation, before the last character has been shifted out from the transmitter shift register. To avoid this, an application should poll the UART status register and not force the IU in error mode before the transmitter shift registers are empty. The real hardware does not exhibit this problem since the UARTs continue to operate even when the IU is halted.

INTERRUPT CONTROLLER

External interrupts are not implemented, so the interrupt shape register has no function. Internal interrupts are generated as defined in the MEC specification. All 15 interrupts can be tested via the interrupt force register.

WATCHDOG

The watchdog timer operate as defined in the MEC specification. The frequency of the watchdog clock can be specified using the **-wdfreq** switch.

POWER-DOWN MODE

The power-down register (0x01f800008) is implemented as in the specification. A Ctrl-C in the simulator window will exit the power-down mode. The simulator runs at least 10 times faster in power-down mode.

MEMORY EMULATION

The amount of simulated memory is configured through the **-ram** and **-rom** switches. The ram size can be between 256K and 32 M, the rom size between 128K and 4M. Access to unimplemented MEC registers or non-existing memory will result in a memory exception trap.

The memory configuration register is used to decode the simulated memory. The fields RSIZ and PSIZ are used to set RAM and ROM size, the remaining fields are not used. NOTE: after reset, the MEC is set to decode 128 Kbyte of ROM and 256 Kbyte of RAM. The memory configuration register has to be updated to reflect the available memory.

The waitstate configuration register is used to generate waitstates. This register must also be updated with the correct configuration after reset.

I/O AREAS

To allow testing of I/O drivers, eac of the four I/O areas have 1 Kbyte of RAM attached to them. To access the I/O areas, the I/O configuration register must be setup accordingly. If an I/O area is set to be larger the 1 Kbyte, access above the 1 Kbyte limit will simply wrap-around.

GDB-INTEGRATED SIS

To use the GDB-integrated simulator (**gdb** or **gdb64**), use the ‘target sim’ command at the gdb prompt. The only valid options for gdb are **-rom8**, **-nfp**, **-freq**, **-v**, **-sparclite**, **-uben,** -f**ast_uart, -ram, -rom** and **-nogdb**. GDB inserts breakpoints in the form of the ‘ta 1’ instruction. The GDB-integrated simulator will therefore recognize the breakpoint instruction and return control to GDB. If the application uses ‘ta 1’, the breakpoint detection can be disabled with the -nogdb switch. In this case however, GDB breakpoints will not work.

Before control is left to GDB, all register windows are flushed out to the stack. Starting after the invalid window, flushing all windows up to, and including the current window. This allows GDB to do backtraces and look at local variables for frames that are still in the register windows. Note that strictly speaking, this behaviour is **wrong** for several reasons. First, it doesn’t use the window overflow handlers. It therefore assumes standard frame layouts and window handling policies. Second, it changes system state behind the back of the target program. Typically, this will only create problems when debugging trap handlers. The ‘-nogdb’ switch disables the register flushing as well.