S R I S C

Simply RISC S1 Core Specification

- version 0.1 -

WISHBONE COMPATIBLE

Simply RISC S1 Core Summary

This is the summary for the S1 Core (codename "Sirocco");
all the informations you need are contained in the text files
that you can find in the "docs" subdirectory:

- README.txt	Summary (this file)
- INSTALL.txt	Quick Installation Guide
- REQUIREMENTS.txt	System Requirements
- SIMULATION.txt	Simulation Environment
- SYNTHESIS.txt	Synthesis Environment
- SPEC.txt	Functional Specification
- SUPPORT.txt	Support and References
- LICENSE.txt	License for Design and Documentation
- TODO.txt	To Do List

Probably now you just have to read the docs/INSTALL.txt file.

Please note that this PDF file you downloaded from the Simply RISC
site http://www.srisc.com is just a collection of the text files
bundled with the design and listed above.

Simply RISC S1 Core - Quick Installation Guide

To install the package just extract it:

```
  tar zxvf s1.tar.gz
```

then edit the top-level "sourceme" file to reflect the locations of the S1 design (we call "S1 root directory" the one containing the "hdl" and "tools" subdirectories) and the root of the T1 design: the first one is mandatory, the second path is needed only if you want to update the SPARC Core source file bundled with this tarball with an updated version of the T1 design released by the OpenSPARC community (using the provided update_sparccore script file).

After that just use on your GNU/Linux or Unix box a bash shell to source this file:

```
  source sourceme
```

Then you have to update the list of files to be used, to do so just run the script:

```
  update_filelist
```

Now you are ready to try the S1 environment to run simulations or synthesis.

Simply RISC S1 Core - System Requirements

You can run simulation and synthesis of the S1 Core almost on any machine: all you need is a Unix-like machine with the following programs installed:

- bash shell;
- sed stream editor;
- for simulations: Icarus Verilog (free software) or Synopsys VCS MX (commercial);
- for synthesis: Icarus Verilog (free software) or Synopsys Design Compiler (commercial).

As you can easily understand, whatever GNU/Linux or Unix machine should be suitable for your purposes; we haven't tried on Windows with Cygwin but we suspect that it could work (please let us know your experience at support@srisc.com and we'll list it here).

Infact since the only tool you need for simulation and synthesis is Icarus Verilog, and since it is free software, you can download its source code and compile it for your platform.

On some GNU/Linux distributions there's even no need to compile it since you can install it from Internet with one command:

- on Debian and similar distros like Ubuntu to install Icarus Verilog just use the command:

  ```
  apt-get install verilog
  ```

- if you are a Gentoo maniac you can use the command:

  ```
  emerge iverilog
  ```

In both cases you will need an Internet connection and root privileges to perform the installation (otherwise go to the official site and compile it from the sources).

Please note that we have been using Icarus version 0.8 without any trouble, but some user reported some compiling error using the latest version 0.8.2.

Another requirement is related with the SPARC v9 compiler: there's an x86 to sparc64 GCC cross-compiler available on the web so you should be able to compile test programs for the S1 Core using not only a SPARC machine but whatever GNU/Linux x86 PC; please check on the Download Area of the Simply RISC website at http://www.srisc.com .

Simply RISC S1 Core - Simulation Environment

To run a simulation using the free software Icarus Verilog simulator use the following commands:

```
  build_icarus
  run_icarus
```

If you want to use a commercial tool such as Synopsys VCS then set up your PATH enviroment variable so that you are able to find the "vcs" executable, and then type in the following commands:

```
  build_vcs
  run_vcs
```

Within this design the only visible difference between Icarus and VCS is the speed: the commercial tool could be hundreds of times faster than its FLOSS counterpart; but with Icarus if you have time to wait for some minutes you will obtain exactly the same results just using free software.

At the end of the simulation you can look at the logfile and at the waveforms placed at the following paths:

```
  run/sim/icarus/sim.log
  run/sim/icarus/trace.vcd

  run/sim/vcs/sim.log
  run/sim/vcs/trace.vcd
```

Obviously if you do not have access to a commercial tool you can use GTKWave to look at the waveforms: for instance from the top-level directory just type in the following command:

```
  gtkwave run/sim/icarus/trace.vcd &
```

and then from "File|Read Save File" choose the file named "tools/src/gtkwave.sav".

Simply RISC S1 Core - Synthesis Environment

The scripts to run synthesis are similar to the ones used for simulations, you can still use the free Icarus Verilog software (that will target an FPGA application) or a commercial Design Compiler tool from Synopsys (that will be used for ASIC).

With Icarus you will use the "fpga" target, to do so just run:

```
  build_fpga
```

If you want to use Synopsys Design Compiler instead you have to use:

```
  build_dc
```

Please note that the commercial tools are NOT supported, and they will probably not work unless you fix all the required parameters properly (we are focusing on free software since we want to build up a community of developers around the S1).

The results for these two kinds of scripts are in the directories:

```
  run/synth/fpga/
```

and

```
  run/synth/dc/
```

Simply RISC S1 Core - Functional Specification

Preface

The S1 is a CPU core that makes use of a single SPARC Core
extracted from the OpenSPARC T1, with the addition of a
Wishbone Bridge, a Reset Controller and an Interrupt
Controller.

```
         ___________________________________
        |        Simply RISC S1 Core        |
        | _______  _____ ________  ________ |
        ||       ||     ||       ||        ||
        || Reset || Int || SPARC ||Wishbone||
        || Ctrl. ||Ctrl.|| Core  || Bridge ||
        ||       ||     ||       ||        ||
        | -------  -----  -------  -------- |
         -----------------------------------
```

Instruction Set Architecture

The CPU inside the S1 Core is the SPARC Core of the
OpenSPARC T1 microprocessor, so you can read OpenSPARC
documents for the specification of this CPU.
Basically it follows the SPARC v9 64-bit ISA, specified
in the documents freely available on the opensparc.net
website; for the full documentation you also have to read
the datasheet and other specs included in the download
file of the T1 design.

Software Support

The SPARC v9 ISA is obviously supported by the GCC compiler;
also GNU/Linux is supported and the latest versions of the
kernel are ready for the T1.
There's also a complete GNU/Linux distribution, Ubuntu,
that comes ready for the SPARC Core of the T1 and could be
used in a seamless way also for Simply RISC S1 based micros.

S1 Memory Map

The S1 Core has 64-bit wide Data Bus and Address Bus.
Each bit of the Address Bus has a different meaning:
- bits [63:59] specify the one-hot encoding for the T1 target region number;
- bits [58:40] are 19 bits always hardwired to zero;
- bits [39:0] are the 40-bit Physical Address as it comes from the SPARC Core.

For most applications the lowest 40-bit address coming from the SPARC Core
should be enough, so it should be safe to ignore the other bits (unless
you suspect that for instance Physical Address zero in region 0 is
different from address zero in region 1 and so forth).

S1 Physical Address [63:56]

the first byte specifies the T1 target region number (one-hot encoded
on the most significant 5 bits, the 3 least significant bits are
hardwired to zero):

- 0x08	RAM Bank 0 (bit 59 set)
- 0x10	RAM Bank 1 (bit 60 set)
- 0x20	RAM Bank 2 (bit 61 set)
- 0x40	RAM Bank 3 (bit 62 set)

- 0x80	I/O Bridge (bit 63 set)

As you can see one of the five most significant bits of the address must be always set depending on the target region.

S1 Physical Address [39:32]

In the T1 bit 39 is zero for all memory addresses and 1 for I/O addresses. Then the T1, from which the S1 is derived, uses the following memory map for bits [39:32] (please note that this mapping is NOT used by the S1):

- 0x00-0x7F	RAM
- 0x80	JBus
- 0x81-0x95	<Reserved>
- 0x96	Clock Unit
- 0x97	RAM Controller
- 0x98	I/O Bridge Management Block
- 0x99	TAP Unit
- 0x9A-0x9D	<Reserved>
- 0x9E	TAP2ASI
- 0x9F	I/O Bridge Interrupt Table
- 0xA0-0xBF	L2 Control Registers
- 0xC0-0xFE	JBus
- 0xFF	Boot ROM

For the S1 Core the memory map is different and the following table applies:

- 0x00-0x7F	RAM
- 0x80-0x95	<Reserved>
- 0x96	Real Time Clock
- 0x97	RAM Controller
- 0x98	Wishbone Interconnect Arbiter
- 0x99	DMA Controller
- 0x9A-0x9D	<Reserved>
- 0x9E	General Purpose I/O
- 0x9F	Interrupt Controller
- 0xA0-0xFE	<Reserved>
- 0xFF	Boot ROM

If you intend to use in your system not only the least significant 40 bits generated by the SPARC Core, take into account that for accesses not directed to RAM but to configuration registers it should be an access to the I/O region (bit 63 set), so for instance the full 64-bit base address for the Interrupt Controller should be 0x8000_009F_0000_0000.

S1 Physical Address [2:0]

The S1 Core implements a Wishbone Master interface with a data bus of 64 bits and a granularity of 1 byte.
Then the address generated by the S1 is always aligned on 64-bit boundaries, i.e. bits [2:0] are always zero; the information about the bytes that have to be accessed during a Wishbone bus cycle is obviously encoded on the SEL signals described in the Wishbone spec.

Reset Controller

The boot sequence of the T1 is quite complex; we have then written a reset controller for the S1 that just takes one reset signal and generates all the signals required by the SPARC Core to boot up properly.

Interrupt Controller

Early versions of the S1 Core feature a very basic Interrupt Controller that latches interrupt requests arriving to the core that are then signaled to the SPARC Core with the proper CPX packet.

Currently this block is untested and it will be enhanced in the future.

Wishbone Bridge

The eight SPARC Cores inside the OpenSPARC T1 microprocessor make use of a proprietary protocol to communicate with the rest of the chip; this protocol is often referred to as PCX/CPX protocol, where PCX stands for "Processor-to-Cache Xbar" and is used for the requests outgoing the SPARC Cores and CPX stands for "Cache-to-Processor Xbar" and is used for incoming packets.

The main block designed specifically for the S1 Core is the "SPARC Core to Wishbone Master interface bridge" that translates the requests and return packets of the SPARC Core into the Wishbone protocol.

You can find the full specification of the Wishbone protocol on the OpenCores site at http://www.opencores.org

Interface Details

These are the features of the bus interface of the S1 Core (they can also be referred to as "Wishbone Datasheet"):

- Wishbone Master interface that follows revision B.3;
- standard signals names identified by leading "wbm_" chars;
- no ERR/RTY support;
- 64-bit Address Bus (with some bits unused, see above);
- 64-bit Data Bus supporting 8, 16, 32 and 64 bit accesses;
- data transfer ordering is Big Endian;
- supports Single Read/Write Cycles.

Simply RISC S1 Core - Support and References

The S1 Core has been developed by Simply RISC and released as Free Hardware Design under the GPL license version 2. You have all the freedom granted by this license, but to avoid legal problems we have to state clearly that:

1) all the files bundled with this package come WITHOUT WARRANTY, so USE THEM AT YOUR OWN RISK;

2) you do NOT have the right to pretend the support you need from Simply RISC.

Anyway we will try to provide all the free support that we can, and now we try to list how to ask for it.

Basically since the S1 Core is based upon the OpenSPARC T1 microprocessor released by Sun Microsystems, for issues related with the SPARC Core inside the S1 you can refer to the OpenSPARC website:

  http://www.opensparc.net

In the documentation section of this site you will find several documents about the SPARC v9 64-bit Instruction Set Architecture (you should be able to read them without registration).

For the specification of the T1 implementation of the 64-bit SPARC ISA you need a free registration to download the full design (it's a very large file) that contains also the full specification of the T1 microprocessor.

On the OpenSPARC website listed above there are also public forums online where you can ask for free support.

If you need support about the Wishbone Interconnect, the URL you need is:

  http://www.opencores.org

Here you will find also several other cores that can be connected to the S1 seamlessly; again, there are mailing lists available from this site when you can discuss with real experts about your problems with the Wishbone interconnect protocol.

If you still need support or you want to take part in the development of the S1 Core, then you can contact us at

  support@srisc.com

and we will try to help you if possible. If you find something that can be corrected and/or improved in the design or in the documentation please let us know and we will fix it in the next release.

To see if there is a new release available just check
from time to time on the Simply RISC website:

```
  http://www.srisc.com
```

Simply RISC S1 Core - To Do List

=================================

This is the list of the higher priority tasks:
- sim problem: dirty requests after a while
- synth problem: Icarus assertion failed

Lower priority:
- support for PCX/CPX packets other than IMISS/LD/ST
- full test-suite for verification
- harness and interrupt handlers to verify INT_CTRL