USER´S MANUAL:

WISHBONE BUILDER

The WISHBONE System-On-Chip (SoC) Interconnection Architecture for Portable IP Cores is a flexible design methodology for use with semiconductor IP cores. Its purpose is to foster design reuse by alleviating system-on-chip integration problems. This is accomplished by creating a common interface between IP cores. This improves the portability and reliability of the system, and results in faster time-to-market for the end user.

Table of Contents

Wishbone - overall description

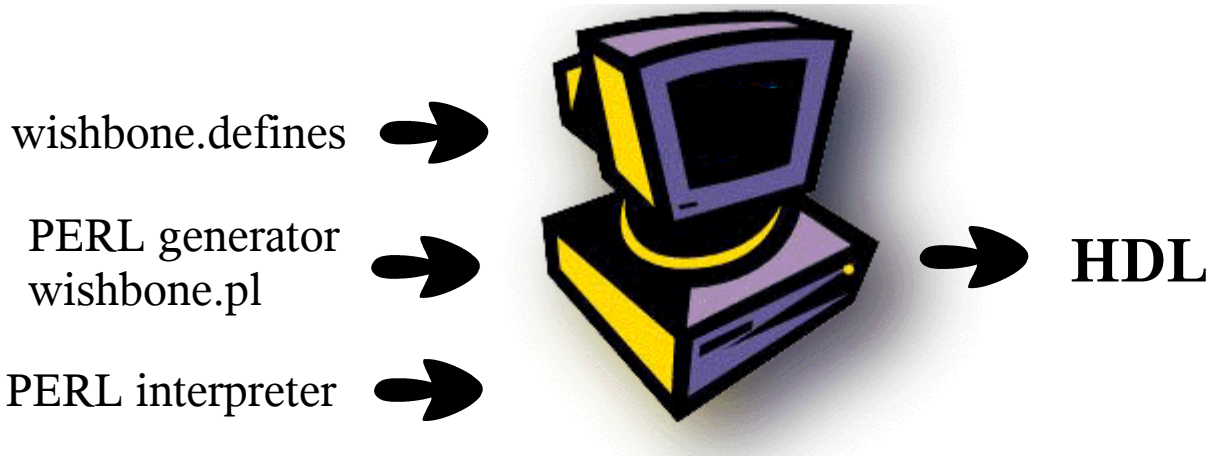
The WISHBONE system-on-chip (SoC) architecture is a portable interface for use with semiconductor IP cores. It is intended to be used as an internal bus for SoC applications. Its purpose is to foster design reuse by alleviating system-on-a-chip integration problems. This is accomplished by creating a common interface between IP cores. This improves the portability and reliability of the system, and results in faster time-to-market for the end user. WISHBONE itself is not an IP core...it is a specification for creating IP cores.

The WISHBONE standard is not copyrighted, and is in the public domain. It may be freely copied and distributed by any means. Furthermore, it may be used for the design and production of integrated circuit components without royalty or other financial obligation.

The specification for wishbone can be found at:

http://www.opencores.org/wishbone/specs/wbspec_b3.pdf

Generator – overall description



Generator flow

The wishbone generator is written in PERL.

PERL Facts

- Perl is a stable, cross platform programming language.
- It is used for mission critical projects in the public and private sectors.
- Perl is Open Source software, licensed under its Artistic License, or the GNU General Public License.
- Perl was created by Larry Wall.
- Perl 1.0 was released to usenet's alt.comp.sources in 1987
- *PC Magazine* named Perl a finalist for its 1998 Technical Excellence Award in the Development Tool category.
- Perl is listed in the Oxford English Dictionary.

PERL can be found at:

http://www.perl.org/get.html

The configuration of the wishbone arbiter for your applications is typed into a define file, typically called "*wishbone.defines*". The format and syntax of this file will be described later in this document. The generator is called "*wishbone.pl*". The generator is invoked by typing the following in a shell / at the command prompt.

perl wishbone.pl [-nogui] [wishbone.defines]

The option -nogui runs the generator quitely. Otherwise a graphical GUI is used for the tailoring of the arbiter. Independently of the GUI a define file is always generated.

Interface specification

Syscon signals

CLK_O	The system clock output [CLK_O] is generated by the SYSCON module. It coordinates all activities for the internal logic within the WISHBONE interconnect. The INTERCON module connects the [CLK_O] output to the [CLK_I] input on MASTER and SLAVE interfaces.
RST_O	The reset output [RST_O] is generated by the SYSCON module. It forces all WISHBONE interfaces to restart. All internal self-starting state machines are forced into an initial state. The INTERCON connects the [RST_O] output to the [RST_I] input on MASTER and SLAVE interfaces.

INTERCON Signals

Signals Common to MASTER and SLAVE Interfaces

CLK_I	The clock input [CLK_I] coordinates all activities for the internal logic within the WISHBONE interconnect. All WISHBONE output signals are registered at the rising edge of [CLK_I]. All WISHBONE input signals must be stable before the rising edge of [CLK_I].
RST_I	The reset input [RST_I] forces the WISHBONE interface to restart. Furthermore, all internal self-starting state machines will be forced into an initial state.
DAT_I()	The data input array [DAT_I()] is used to pass binary data. The array boundaries are determined by the port size. Also see the [DAT_O()] and [SEL_O()] signal descriptions.
TGD_I()	Data tag type [TGD_I()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data input array [DAT_I()], and is qualified by signal [STB_I]. For example, parity protection, error correction and time stamp information can be attached to the data bus. These tag bits simplify the task of defining new signals because their timing (in rela tion to every bus cycle) is pre-defined by this specification. The name and operation of a data tag must be defined in the WISHBONE DATASHEET.
DAT_O()	The data output array [DAT_O()] is used to pass binary data. The array boundaries are determined by the port size. Also see the [DAT_I()] and [SEL_O()] signal descriptions.
TGD_O()	Data tag type [TGD_O()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data output array [DAT_O()], and is qualified by signal [STB_O]. For example, parity protection, error correction and time stamp information can be attached to the data bus. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification. The name and operation of a data tag must be defined in the WISHBONE DATASHEET.

MASTER Signals

ACK_I	The acknowledge input [ACK_I], when asserted, indicates the termination of a normal bus cycle. Also see the [ERR_I] and [RTY_I] signal descriptions.
ADR_O()	The address output array [ADR_O(63..0)] is used to pass a binary address, with the most signifi-cant address bit at the higher numbered end of the signal array. The lower array boundary is specific to the data port size. The higher array boundary is core-specific. In some cases (such as FIFO interfaces) the array may not be present on the interface.
CYC_O	The cycle output [CYC_O], when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The [CYC_O] signal is asserted during the first data transfer, and remains asserted until the last data transfer. The [CYC_O] signal is useful for interfaces with multi-port interfaces (such as dual port memories). In these cases, the [CYC_O] signal re-quests use of a common bus from an arbiter. Once the arbiter grants the bus to the MASTER, it is held until [CYC_O] is negated.
ERR_I	The error input [ERR_I] indicates an abnormal cycle termination.
LOCK_O	The lock output [LOCK_O] when asserted, indicates that the current bus cycle is uninterruptible. Lock is asserted to request complete ownership of the bus. Once the transfer has started, the INTERCON does not grant the bus to any other MASTER, until the current MASTER negates [LOCK_O] or [CYC_O].
RTY_I	The retry input [RTY_I] indicates that the indicates that the interface is not ready to accept or send data, and that the cycle should be retried.
SEL_O()	The select output array [SEL_O()] indicates where valid data is expected on the [DAT_I()] signal array during READ cycles, and where it is placed on the [DAT_O()] signal array during WRITE cycles. Also see the [DAT_I()], [DAT_O()] and [STB_O] signal descriptions.
STB_O	The strobe output [STB_O] indicates a valid data transfer cycle. It is used to qualify various other signals on the interface such as [SEL_O()]. The SLAVE must assert either the [ACK_I], [ERR_I] or [RTY_I] signals in response to every assertion of the [STB_O] signal.
TGA_O()	Address tag type [TGA_O()] contains information associated with address lines [ADR_O()], and is qualified by signal [STB_O]. For example, address size (24-bit, 32-bit etc.) and memory management (protected vs. unprotected) information can be attached to an address. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) isdefined by this specification. The name and operation of an address tag must be defined in the WISHBONE DATASHEET.
TGC_O()	Cycle tag type [TGC_O()] contains information associated with bus cycles, and is qualified by signal [CYC_O]. For example, data transfer, interrupt acknowledge and cache control cycles can be uniquely identified with the cycle tag. They can also be used to discriminate between WISHBONE SINGLE, BLOCK and RMW cycles. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is defined by this specification. The name and operation of a cycle tag must be defined in the WISHBONE DATASHEET.

ACK_I	The acknowledge input [ACK_I], when asserted, indicates the termination of a normal bus cycle. Also see the [ERR_I] and [RTY_I] signal descriptions.
WE_O	The write enable output [WE_O] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.

SLAVE Signals

ACK_O	The acknowledge output [ACK_O], when asserted, indicates the termination of a normal bus cycle.
ADR_I()	The address input array [ADR_I()] is used to pass a binary address, with the most significant address bit at the higher numbered end of the signal array. The lower array boundary is specific to the data port size. The higher array boundary is core-specific. In some cases (such as FIFO interfaces) the array may not be present on the interface.
CYC_I	The cycle input [CYC_I], when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The [CYC_I] signal is asserted during the first data transfer, and remains asserted until the last data transfer.
ERR_O	The error output [ERR_O] indicates an abnormal cycle termination. The source of the error, and the response generated by the MASTER is defined by the IP core supplier. Also see the [ACK_O] and [RTY_O] signal descriptions.
LOCK_I	The lock input [LOCK_I], when asserted, indicates that the current bus cycle is uninterruptible. A SLAVE that receives the LOCK [LOCK_I] signal is accessed by a single MASTER only, until either [LOCK_I] or [CYC_I] is negated.
RTY_O	The retry output [RTY_O] indicates that the indicates that the interface is not ready to accept or send data, and that the cycle should be retried. When and how the cycle is retried is defined by the IP core supplier. Also see the [ERR_O] and [RTY_O] signal descriptions.
SEL_I()	The select input array [SEL_I()] indicates where valid data is placed on the [DAT_I()] signal array during WRITE cycles, and where it should be present on the [DAT_O()] signal array during READ cycles. The array boundaries are determined by the granularity of a port. For example, if 8-bit granularity is used on a 64-bit port, then there would be an array of eight select signals with boundaries of [SEL_I(7..0)]. Each individual select signal correlates to one of eight active bytes on the 64-bit data port. For more information about [SEL_I()], please refer to the data organization section in Chapter 3 of this specification. Also see the [DAT_I(63..0)], [DAT_O(63..0)] an [STB_I] signal descriptions.
DAT_I()	The data input array [DAT_I()] is used to pass binary data. The array boundaries are determined by the port size. Also see the [DAT_O()] and [SEL_O()] signal descriptions.
DAT_O()	The data output array [DAT_O()] is used to pass binary data. The array boundaries are determined by the port size. Also see the [DAT_I()] and [SEL_O()] signal descriptions.
ERR_O	The error output [ERR_O] indicates an abnormal cycle termination.
RTY_O	The retry output [RTY_O] indicates that the indicates that the interface is not ready to accept or send data, and that the cycle should be retried.

ACK_O	The acknowledge output [ACK_O], when asserted, indicates the termination of a normal bus cycle.
SEL_I()	The select input array [SEL_I()] indicates where valid data is placed on the [DAT_I()] signal array during WRITE cycles, and where it should be present on the [DAT_O()] signal array during READ cycles. Also see the [DAT_I()], [DAT_O()] and [STB_I] signal descriptions.
STB_I	The strobe input [STB_I] indicates a valid data transfer cycle. It is used to qualify various other signals on the interface such as [SEL_I(7..0)]. The SLAVE must assert either the [ACK_O], [ERR_O] or [RTY_O] signals in response to every assertion of the [STB_I] signal.
WE_I	The write enable input [WE_I] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.
TGA_I()	Address tag type [TGA_I()] contains information associated with address lines [ADR_I()], and is qualified by signal [STB_I]. For example, address size (24-bit, 32-bit etc.) and memory management (protected vs. unprotected) information can be attached to an address. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification. The name and operation of an address tag must be defined in the WISHBONE DATASHEET.
TGC_I()	Cycle tag type [TGC_I()] contains information associated with bus cycles, and is qualified by signal [CYC_I]. For example, data transfer, interrupt acknowledge and cache control cycles can be uniquely identified with the cycle tag. They can also be used to discriminate between WISHBONE SINGLE, BLOCK and RMW cycles. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification. The name and operation of a cycle tag must be defined in the WISHBONE DATASHEET.

Tag types

The WISHBONE interface can be modified with user defined signals. This is done with a technique known as tagging. Tags are a well known concept in the microcomputer bus industry.
They allow user defined information to be associated with an address, a data word or a bus cycle. All tag signals must conform to set of guidelines known as TAG TYPEs. Table below lists all of the defined TAG TYPEs along with their associated data set and signal waveform. When a tag is added to an interface it is assigned a TAG TYPE from the table. This explicitly defines how the tag operates. This information must also be included in the WISHBONE DATASHEET.

	MASTER		SLAVE	
DESCRIPTION	**TAG TYPE**	**Associated with**	**TAG TYPE**	**Associated with**
Address tag	TGA_O()	ADR_O()	TGA_I()	ADR_I()
Data tag, input	TGD_I()	DAT_I()	TGD_I()	DAT_I()
Data tag, output	TGD_O()	DAT_O()	TGD_O()	DAT_I()
Cycle tag	TGC_O()	Bus cycle	TGC_I()	Bus cycle

Address tag

Burst Type Extension, BTE	Bit pattern
Linear	00
Wrap-4	01
Wrap-8	10

Cycle tag

Cycle Type Identifier, CTI	Bit pattern
Classic	000
Constant address burst	001
Incrementing address burst	010
End of burst	111

Configuration options

All configuration options resides in config file, "*wishbone.defines*" or similar.

All lines starting with # are comments.

Global configurations

Module name

The module/entity name is configurable.

Parameter	function	Valid values	Default value
syscon	Defines top module/entity name for system controller	any	syscon
intercon	Defines top module/entity name for interconnection	any	intercon
filename	Defines file name	any	wb_arbiter

Output type

The wishbone core generator can generate a functional description in either VHDL92 with IEEE1164 type support or in Verilog HDL. Set parameter *hdl* to *vhdl* or *verilog*.

Parameter	function	Valid values	Default value
hdl	Defines output language	vhdl, verilog, perlilog	vhdl
signal_groups	Defines if signal groups should be used	0,1	0

If *signal_groups* is defined top level entity will be using VHDL records. Type definitions will be put in a package called *intercon*_types.

Target technology

For optimal performance/area efficiency target dependable optimasation is used. Target technology is defined by parameter target_family.

Parameter	function	Valid values	Default value
target	Defines target technology	generic xilinx altera	generic

Data bus size

The number of bits in data and adress buses is selectable.

Parameter	function	Valid values	Default value
dat_size	Defines data bus width	8, 16, 32, 64	32
adr_size	Defines address bus width	any	32

Note that any specific master or slave can have a narrower bus interface.

Tag fields

The user defined tag fields tga, tgc and tgd can be defined to any width and the signals can be renamed to better suit a specific application. Width set to zero means that particular signal should not be presented in design.

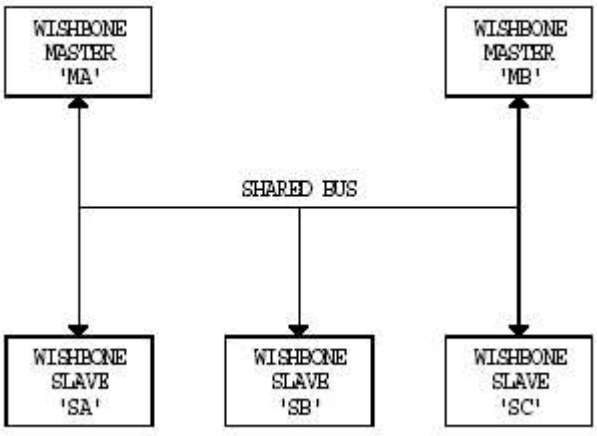
Parameter	function	Valid values	Default value
rename_tgc	Renames tgc signals on top level	any	cti
tgc	Defines tag signal width	any	3
endofburst	Defines tag bit pattern indicating last phase of burst transfer	any	111
rename_tga	Renames tgc signals on top level	any	bte
tga	Defines tag signal width	any	2
rename_tgd	Renames tgc signals on top level	any	tgd
tgd	Defines tag signal width	any	0

Interconnection type

The wishbone generator supports two types of interconnection types defined in the wishbone specification:

1. shared bus
2. crossbar switch

Shared bus



Shared bus interconnection

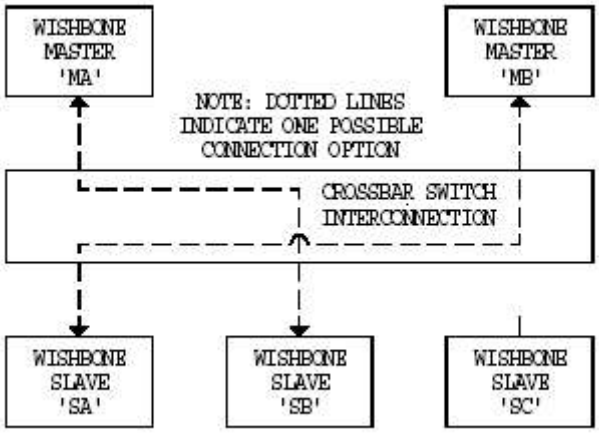
The shared bus interconnection is useful for connecting two or more MASTERs with one or more SLAVEs. A block diagram is shown in figure. In this topology a MASTER initiates a bus cycle to a target SLAVE. The target SLAVE then participates in one or more bus cycles with the MASTER.

An arbiter (not shown in the Figure) determines when a MASTER may gain access to the shared bus. The arbiter acts like a ‘traffic cop’ to determine when and how each MASTER accesses the shared resource. Also, the type of arbiter is completely defined by the system integrator. For

example, the shared bus can use a priority or a round robin arbiter. These grant the shared bus on a priority or equal basis, respectively.

The main advantage to this technique is that shared interconnection systems are relatively compact. Generally, it requires fewer logic gates and routing resources than other configurations, especially the crossbar switch. Its main disadvantage is that MASTERs may have to wait before gaining access to the interconnection. This degrades the overall speed at which a MASTER may transfer data.

Crossbar switch



Crossbar switch interconnection

The crossbar switch interconnection is used when connecting two or more WISHBONE MASTERs together so that each can access two or more SLAVEs. A block diagram is shown in figure. In the crossbar interconnection, a MASTER initiates an addressable bus cycle to a target SLAVE. An arbiter (not shown in the diagram) determines when each MASTER may gain access to the indicated SLAVE. Unlike the shared bus interconnection, the crossbar switch allows more than one MASTER to use the interconnection (as long as two MASTERs don't access the same SLAVE at the same time).

Under this method, each master arbitrates for a 'channel' on the switch. Once this is established, data is transferred between the MASTER and the SLAVE over a private communication link. The Figure shows two possible channels that may appear on the switch. The first connects MASTER 'MA' to SLAVE 'SB'. The second connects MASTER 'MB' to SLAVE 'SA'.

The overall data transfer rate of the crossbar switch is higher than shared bus mechanisms. For example, the figure shows two MASTER/SLAVE pairs interconnected at the same time. If each communication channel supports a data rate of 100 Mbyte/sec, then the two data pairs would operate in parallel at 200 Mbyte/sec. This scheme can be expanded to support extremely high data transfer rates.

One disadvantage of the crossbar switch is that it requires more interconnection logic and routing resources than shared bus systems. As a rule-of-thumb, a crossbar switch with two MASTERs and two SLAVEs takes twice as much interconnection logic as a similar shared bus system (with two MASTERs and two SLAVEs).

Interconnect specification

To specify interconnect implementation it is sufficient to sepcify number of concurrent paths. A value equal to one specifies a shared bus topology, a number greater than one specifies a crossbar switch with the given number of paths.

Parameter	function	Valid values	Default value
interconnect	Specifies number of concurrent paths	Sharedbus, crossbarswitch	sharedbus

Multiplexor implementation

The multiplexors used in the design can be implemented in three ways:

1. with tristate buses
2. as an and-or structure
3. as multiplexors

Best choice is dependent on target technology and expected performance.

Parameter	function	Valid values	Default value
mux_type	Defines implementation style of multiplexors	andor, tristate, mux	andor

Optimize

The design can be optimized for are or speed. When optimizing for area some intermidiate signals are being preserved and used for more purposes

Parameter	function	Valid values	Default value
optimize	Optimizes for speed or area	speed, area	speed

Master port(s) configuration

The actual number of master and slave devices connected to the wishbone system bus must be defined. Implemenation support any number of master and slaves. The performance will degrade with increasing number.

For each master port a section defines which signals should be present.

Parameter	function	Valid values	Default value
dat_size	Defines dat_i bus width	0, 8, 16, 24, 32, 64	dat_size
type	Defines read and/or write functionallity	ro, wo, rw	rw
adr_o	Defines address bus width	0-63	adr_size
lock_o	Defines if lock output is present	0,1	1
err_i	Defines if error input is supported	0,1	1
rty_i	Defines if retry is supported	0,1	1
tga_o	Defines address tag output	0,1	0
tgc_o	Defines cycle tag output	0,1	0
priority	Defines bus access for sharedbus systems	any	1
priority_%slave	Defines bus access for crossbarswitch systems. A value of zero indicates that slave is not used by current master.	any	None, must be defined

A wishbone bus master can be of three types:

1. ro – read only. The following signals are not present; we_o, dat_o
2. wo – write only. The following signal is not present; dat_i
3. rw – read/write. All signals are present.

Depending on priority each master is guaranteed a number of bus cycles according to its priority over a period of Σpriority bus cycles. If no priority is defined priority equal to one is assumed. That means that over a period equal to the sum of wishbone bus masters each master is guaranteed one bus cycle.

For crossbarswitch systems prioritties are defined and handled per slave.

Master port configuration example:

```
master or32_i
    priority=4
    type=ro
    lock_o=1
    err_i=1
    rty_i=1
    tga_o=1
    tgc_o=1
end master or32_i
```

Namning conventions; or32_i_dat_o or or32_i_o.dat_o depending on definition of *signal_groups*.

Slave port(s) configuration

For every slave unit the following must be defined:

Parameter	function	Valid values	Default value
dat_size	Databus width	0, 8, 16, 24, 32, 64	dat_size
type	Defines read and/or write functionallity	ro, wo, rw	rw
adr_i_hi	Addressbus width, upper (left) limit	0-31	31
adr_i_lo	Addressbus width, lower (right) limit	0-31	2
lock_i	Defines support for lock input	0,1	0
tga_i	Defines address tag present	0,1	0
tgc_i	Defines cycle tag present	0,1	0
err_o	Indicates an abnormal cycle termination	0,1	0
rty_o	Indicates that the interface is not ready and that the cycle should be retried	0,1	0
baseaddr[1]	Base address(es) of module	any	--
size[2]	Memory size used by module	any	0x0010_0000

For every wishbone slave a section of the define file is used. Each section starts and ends with reserved words. Se example below:

slave *uart16550*
  adr_i_hi=4
  adr_i_lo=2
  baseaddr=0x9000_0000
  size=0x0100_0000
end slave *uart16550*

Slave_name must be unique.

Slave signals corresponding to wishbone slave will have the following namning conventions: uart16550_dat_o or uart16550_o.dat_o depending on definition of *signal_groups*

1 Additional base adresses can be defined by adding baseaddr1 etc.
2 Additional size definition can be added by adding size1 etc.

OpenCore defined memory map

OpenRISC Reference Platform (ORP) Address Space

Start adr	End adr	cached	Size (Mb)	Content	
0xf000_0000	0xffff_ffff	Cached	256	ROM	
0xc000_0000	0xefff_ffff	Cached	768	Reserved	
0xb800_0000	0xbfff_ffff	Uncached	128	Reserved for custom devices	
0xa600_0000	0xb7ff_ffff	Uncached	288	Reserved	
0xa500_0000	0xa5ff_ffff	Uncached	16	Debug 0-15	
0xa400_0000	0xa4ff_ffff	Uncached	16	Digital Camera Controller 0-15	
0xa300_0000	0xa3ff_ffff	Uncached	16	I2C Controller 0-15	
0xa200_0000	0xa2ff_ffff	Uncached	16	TDM Controller 0-15	
0xa100_0000	0xa1ff_ffff	Uncached	16	HDLC Controller 0-15	
0xa000_0000	0xa0ff_ffff	Uncached	16	Real-Time Clock 0-15	
0x9f00_0000	0x9fff_ffff	Uncached	16	Firewire Controller 0-15	
0x9e00_0000	0x9eff_ffff	Uncached	16	IDE Controller 0-15	
0x9d00_0000	0x9dff_ffff	Uncached	16	Audio Controller 0-15	
0x9c00_0000	0x9cff_ffff	Uncached	16	USB Host Controller 0-15	
0x9b00_0000	0x9bff_ffff	Uncached	16	USB Func Controller 0-15	
0x9a00_0000	0x9aff_ffff	Uncached	16	General-Purpose DMA 0-15	
0x9900_0000	0x99ff_ffff	Uncached	16	PCI Controller 0-15	
0x9800_0000	0x98ff_ffff	Uncached	16	IrDA Controller 0-15	
0x9700_0000	0x97ff_ffff	Uncached	16	Graphics Controller 0-15	
0x9600_0000	0x96ff_ffff	Uncached	16	PWM/Timer/Counter Controller 0-15	
0x9500_0000	0x95ff_ffff	Uncached	16	Traffic COP 0-15	
0x9400_0000	0x94ff_ffff	Uncached	16	PS/2 Controller 0-15	
0x9300_0000	0x93ff_ffff	Uncached	16	Memory Controller 0-15	
0x9200_0000	0x92ff_ffff	Uncached	16	Ethernet Controller 0-15	
0x9100_0000	0x91ff_ffff	Uncached	16	General-Purpose I/O 0-15	
0x9000_0000	0x90ff_ffff	Uncached	16	UART16550 Controller 0-15	
0x8000_0000	0x8fff_ffff	Uncached	256	PCI I/O	
0x4000_0000	0x7fff_ffff	Uncached	1024	Reserved	
0x0000_0000	0x3fff_ffff	Cached	1024	RAM	