RISC-V System-on-Chip description

Autor: Khabarov Sergey 5.1.2

MIPT

Mar 09, 2017

Contents

Chapter 1

RISC-V System-on-Chip VHDL IP library

Overview

The IP Library is an integrated set of reusable IP cores, designed for system-on-chip (SOC) development. The IP cores are centered around a common on-chip AMBA AXI system bus, and use a coherent method for simulation and synthesis. This library is vendor independent, with support for different CAD tools and target technologies. Inherited from gaisler GRLIB library plug&play method was further developed and used to configure and connect the IP cores without the need to modify any global resources.

Library organization

Open source repository with VHLD libraries, Debugger and SW examples is available at:

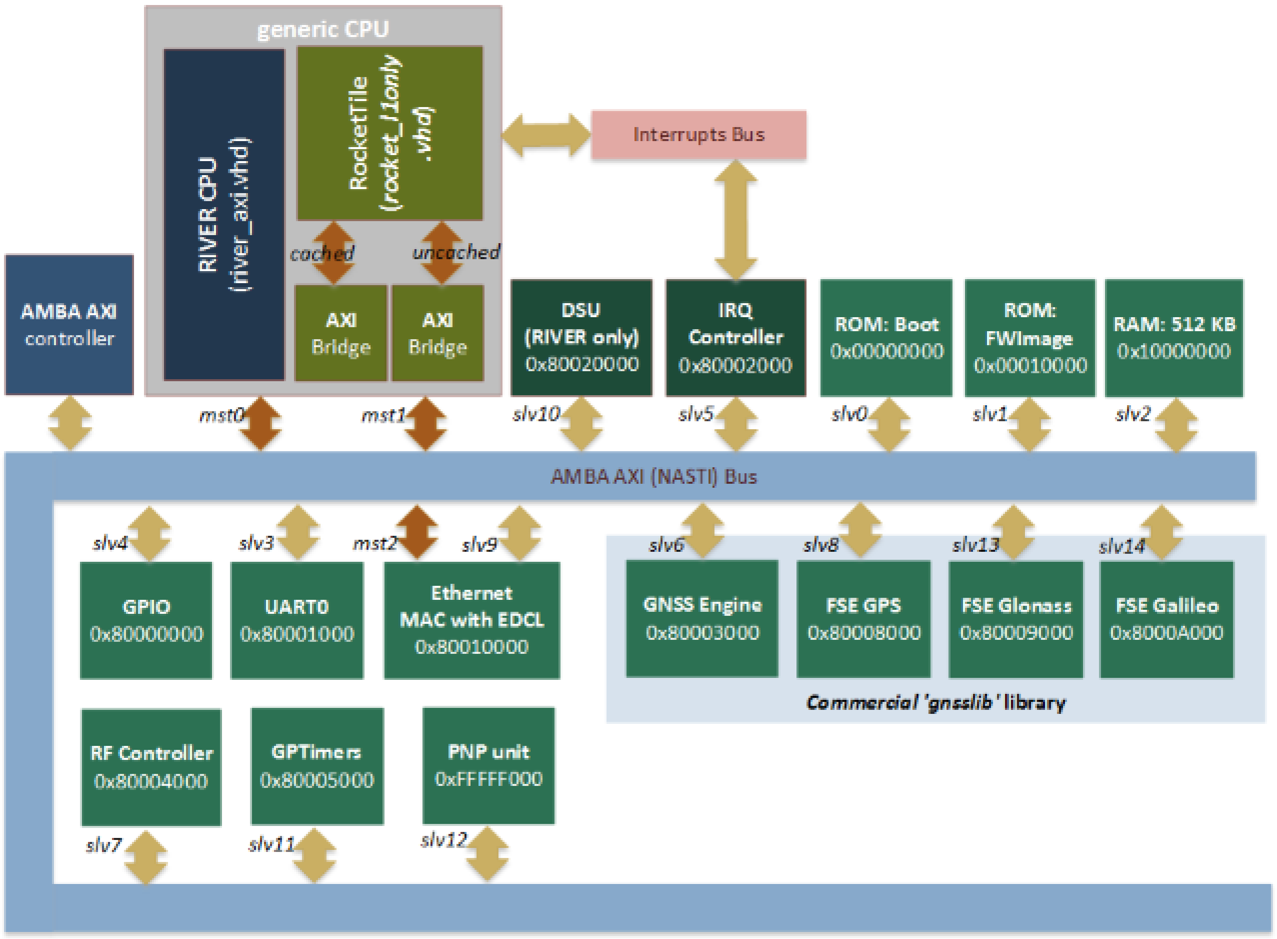
```
https://github.com/sergeykhbr/riscv_vhdl
```

This repository is organized around VHDL libraries, where each major IP is assigned a unique library name. Using separate libraries avoids name clashes between IP cores and hides unnecessary implementation details from the end user.

Satellite Navigation support

Hardware part of the satellite navigation functionality is fully implemented inside of the *gnsslib* library. This library is the commercial product of GNSS Sensor limited and in this shared repository you can find only↩
: modules declaration, configuration parameters and stub modules that provide enough functionality to use SOC as general purpose processor system based on RISC-V architecture. Netlists of the real GNSS IPs either as RF front-end for the FPGA development boards could be acquires via special request.

Common Top-level structure



Features

- Pre-generated single-core *"Rocket-chip"* core (RISC-V). This is 64-bits processor with I/D caches, MMU, branch predictor, 128-bits width data bus, FPU (if enabled) and etc.
- Custom 64-bits single-core CPU *"River"*(RISC-V).
- Set of common peripheries: UART, GPIO (LEDs), Interrupt controller, General Purpose timers and etc.
- Debugging via Ethernet using EDCL capability of the MAC. This capability allows to redirect UDP requests directly on system bus and allows to use external debugger from the Reset Vector.
- Debug Support Unit (DSU) for the RIVER CPU with full debugging functionality support: run/halt, breakpoints, stepping, registers/CSRs and memory access. Also it provides general SoC run-time information: Clock Per Instruction (CPI), Bus Utilisization for each master device and etc.
- Templates for the AXI slaves and master devices with DMA access
- Configuration parameters to enable/disable additional functionality, like: ***GNSS Engine***, ***Viterbi decoder***, *etc.*

Information about GNSS (*Satellite Navigation Engine*) you can find at www.gnss-sensor.com.

VHDL Generic Parameters

RTL Verification

RISC-V Processor

Peripheries

RISC-V debugger

Chapter 2

VHDL Generic Parameters

2.1 SoC configuration constants

Entities

- config_common package

  Techology independent configuration settings.

Libraries

- IEEE

  Standard library.
- techmap

  Technology definition library.

Use Clauses

- STD_LOGIC_1164

  Standard signal definitions.
- gencomp

  Generic IDs constants import.

Constants

- CFG_COMMON_RIVER_CPU_ENABLE **boolean:=true**

  Disable/Enable River CPU instance.
- CFG_SIM_BOOTROM_HEX **string:=" ../../fw_images/bootimage.hex "**

  HEX-image for the initialization of the Boot ROM.
- CFG_SIM_FWIMAGE_HEX **string:=" ../../fw_images/fwimage.hex "**

  HEX-image for the initialization of the FwImage ROM.
- CFG_HW_ID **std_logic_vector(31 downto 0):=X" 20170214 "**

  Hardware SoC Identificator.
- CFG_GNSSLIB_ENABLE **boolean:=false**

  *Disable/Enable usage of the **gnsslib library**.*
- CFG_GNSSLIB_GNSSENGINE_ENABLE **boolean:=false**

  Enable GNSS Engine module.
- CFG_GNSSLIB_FSEGPS_ENABLE **boolean:=false**

  Enable Fast Search Engine for the GPS signals.
- CFG_ETHERNET_ENABLE **boolean:=true**

  Enabling Ethernet MAC interface.
- CFG_DSU_ENABLE **boolean:=true**

  Enable/Disable Debug Unit.
- CFG_TESTMODE_ON **boolean:=true**

  Remove BUFGMUX from project and use internaly generate ADC clock.

2.1.1 Detailed Description

Target independible constants that are the same for FPGA, ASIC and behaviour simulation.

2.1.2 Variable Documentation

2.1.2.1 CFG_COMMON_RIVER_CPU_ENABLE

`CFG_COMMON_RIVER_CPU_ENABLE boolean:=true [Constant]`

Disable/Enable River CPU instance.

When enabled platform will instantiate processor named as "RIVER" entirely written on VHDL. Otherwise "Rocket" will be used (developed by Berkley team).

Warning

DSU available only for *"RIVER"* processor.

2.1.2.2 CFG_ETHERNET_ENABLE

`CFG_ETHERNET_ENABLE boolean:=true [Constant]`

Enabling Ethernet MAC interface.

By default MAC module enables support of the debug feature EDCL.

2.1.2.3 CFG_GNSSLIB_ENABLE

`CFG_GNSSLIB_ENABLE boolean:=false [Constant]`

Disable/Enable usage of the ***gnsslib library***.

This *'gnsslib'* is the property of the *"GNSS Sensor ltd"* (www.gnss-sensor.com) and it implements a lot of Navigation related peripheries, like:

- RF front-end synthezators controller;
- Multi-system GNSS Engine;
- Fast Search modules;
- Viterbi decoders;
- Self-test generators and so on.

Warning

This define enables RF front-end clock as a source of ADC clock.

2.1.2.4 CFG_HW_ID

`CFG_HW_ID std_logic_vector( 31 downto 0 ):=X" 20170214 " [Constant]`

Hardware SoC Identificator.

Read Only unique platform identificator that could be read by firmware from the Plug'n'Play support module.

2.1.2.5 CFG_SIM_BOOTROM_HEX

CFG_SIM_BOOTROM_HEX **string:=" ../../fw_images/bootimage.hex "** [Constant]

HEX-image for the initialization of the Boot ROM.

This file is used by *inferred* ROM implementation.

2.1.2.6 CFG_SIM_FWIMAGE_HEX

CFG_SIM_FWIMAGE_HEX **string:=" ../../fw_images/fwimage.hex "** [Constant]

HEX-image for the initialization of the FwImage ROM.

This file is used by *inferred* ROM implementation.

2.1.2.7 CFG_TESTMODE_ON

CFG_TESTMODE_ON **boolean:=true** [Constant]

Remove BUFGMUX from project and use internaly generate ADC clock.

We have some difficulties with Vivado + Kintex7 constrains, so to make test-mode stable working we use this temporary config parameter that hardcodes 'test_mode' is always enabled

2.2 AMBA AXI slaves generic IDs.

Constants

- CFG_NASTI_SLAVE_BOOTROM **integer:=** 0

  Configuration index of the Boot ROM module visible by the firmware.
- CFG_NASTI_SLAVE_ROMIMAGE **integer:=CFG_NASTI_SLAVE_BOOTROM +** 1

  Configuration index of the Firmware ROM Image module.
- CFG_NASTI_SLAVE_SRAM **integer:=CFG_NASTI_SLAVE_ROMIMAGE +** 1

  Configuration index of the SRAM module visible by the firmware.
- CFG_NASTI_SLAVE_UART1 **integer:=CFG_NASTI_SLAVE_SRAM +** 1

  Configuration index of the UART module.
- CFG_NASTI_SLAVE_GPIO **integer:=CFG_NASTI_SLAVE_UART1 +** 1

  Configuration index of the GPIO (General Purpose In/Out) module.
- CFG_NASTI_SLAVE_IRQCTRL **integer:=CFG_NASTI_SLAVE_GPIO +** 1

  Configuration index of the Interrupt Controller module.
- CFG_NASTI_SLAVE_ENGINE **integer:=CFG_NASTI_SLAVE_IRQCTRL +** 1

  Configuration index of the Satellite Navigation Engine.
- CFG_NASTI_SLAVE_RFCTRL **integer:=CFG_NASTI_SLAVE_ENGINE +** 1

  Configuration index of the RF front-end controller.
- CFG_NASTI_SLAVE_FSE_GPS **integer:=CFG_NASTI_SLAVE_RFCTRL +** 1

  Configuration index of the GPS-CA Fast Search Engine module.
- CFG_NASTI_SLAVE_ETHMAC **integer:=CFG_NASTI_SLAVE_FSE_GPS +** 1

  Configuration index of the Ethernet MAC module.
- CFG_NASTI_SLAVE_DSU **integer:=CFG_NASTI_SLAVE_ETHMAC +** 1

  Configuration index of the Debug Support Unit module.
- CFG_NASTI_SLAVE_GPTIMERS **integer:=CFG_NASTI_SLAVE_DSU +** 1

  Configuration index of the Debug Support Unit module.
- CFG_NASTI_SLAVE_PNP **integer:=CFG_NASTI_SLAVE_GPTIMERS +** 1

  Configuration index of the Plug-n-Play module.
- CFG_NASTI_SLAVES_TOTAL **integer:=CFG_NASTI_SLAVE_PNP +** 1

  Total number of the slaves devices.

2.2.1 Detailed Description

Each module in a SoC has to be indexed by unique identificator. In current implementation it is used sequential indexing for it. Indexes are used to specify a device bus item in a vectors.

2.3 AXI4 masters generic IDs.

Constants

- CFG_NASTI_MASTER_CACHED **integer:= 0**

  Cached TileLinkIO bus.
- CFG_NASTI_MASTER_UNCACHED **integer:=CFG_NASTI_MASTER_CACHED + 1**

  Uncached TileLinkIO bus.
- CFG_NASTI_MASTER_ETHMAC **integer:=CFG_NASTI_MASTER_UNCACHED + 1**

  Ethernet MAC master interface generic index.
- CFG_NASTI_MASTER_TOTAL **integer:=CFG_NASTI_MASTER_ETHMAC + 1**

  Total Number of master devices on system bus.

2.3.1 Detailed Description

Each master must be assigned to a specific ID that used as an index in the vector array of AXI master bus.

2.4 AXI4 interrupt generic IDs.

Constants

- CFG_IRQ_UNUSED **integer:= 0**

  Zero interrupt index must be unused.
- CFG_IRQ_UART1 **integer:=CFG_IRQ_UNUSED + 1**

  UART_A interrupt pin.
- CFG_IRQ_ETHMAC **integer:=CFG_IRQ_UART1 + 1**

  Ethernet MAC interrupt pin.
- CFG_IRQ_GPTIMERS **integer:=CFG_IRQ_ETHMAC + 1**

  GP Timers interrupt pin.
- CFG_IRQ_MISS_ACCESS **integer:=CFG_IRQ_GPTIMERS + 1**

  Memory miss access.
- CFG_IRQ_GNSSENGINE **integer:=CFG_IRQ_MISS_ACCESS + 1**

  GNSS Engine IRQ pin that generates 1 msec pulses.
- CFG_IRQ_TOTAL **integer:=CFG_IRQ_GNSSENGINE + 1**

  Total number of used interrupts in a system.

2.4.1 Detailed Description

Unique indentificator of the interrupt pin also used as an index in the interrupts bus.

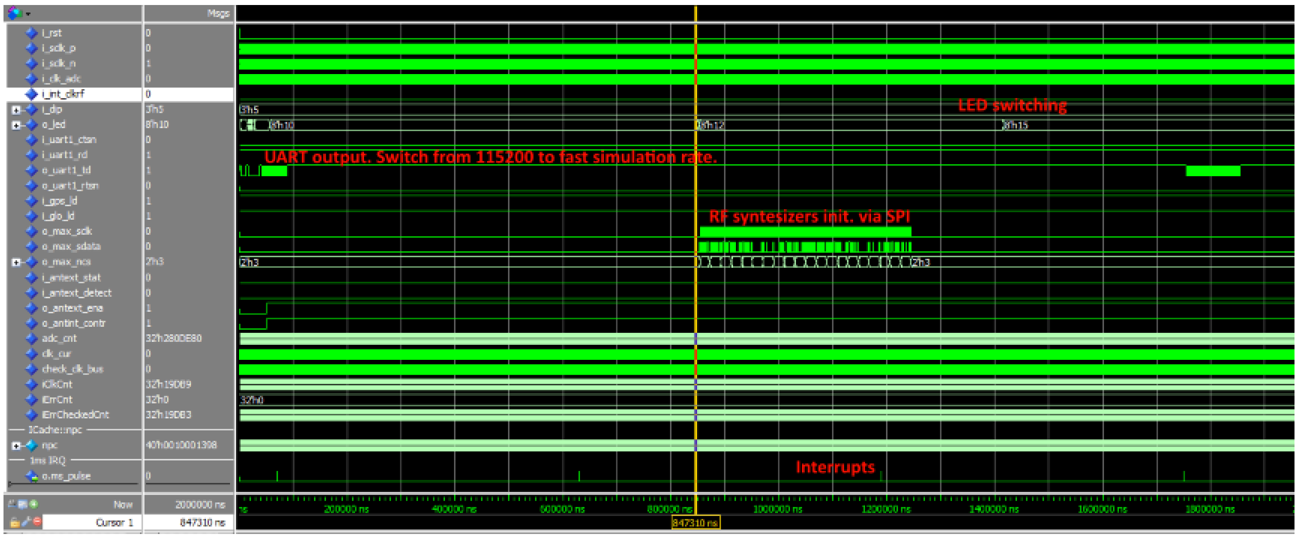
Chapter 3

RTL Verification

3.1 Top-level simulation

Test-bench example

Use file **work/tb/riscv_soc_tb.vhd** to run simulation scenario. You can get the following time diagram after simulation of 2 ms interval.



```
i_rst              0
i_sclk_p           0
i_sclk_n           1
i_clk_adc          0
i_int_clkrf        0
i_dip              3'h5
o_led              8'h10
i_uart1_ctsn       0
i_uart1_rd         1
o_uart1_td         1
o_uart1_rtsn       0
i_gps_ld           1
i_glo_ld           1
o_max_sclk         0
o_max_sdata        0
o_max_ncs          2'h3
i_antext_stat      0
i_antext_detect    0
o_antext_ena       1
o_antint_contr     1
adc_cnt            32'h280DE80
clk_cur            0
check_clk_bus      0
iClkCnt            32'h19DB9
iErrCnt            32'h0
iErrCheckedCnt     32'h19DB3
ICache::npc
npc                40'h0010001398
1ms IRQ
o.ms_pulse         0
Now                2000000 ns
Cursor 1           847310 ns
```

LED switching
UART output. Switch from 115200 to fast simulation rate.
RF syntesizers init. via SPI
Interrupts
847310 ns

Note

Simulation behaviour depends of current firmware image. It may significantly differs in a new releases either as Zephyr OS kernel image is absolutely different relative GNSS FW image.

Some FW versions can detect RTL simulation target by reading *'Target' Register* in PnP device that allows to speed-up simulation by removing some delays and changing Devices IO parameters (UART speed for example).

Running on FPGA

Supported FPGA:

- ML605 with Virtex6 FPGA using ISE 14.7 (default).
- KC705 with Kintex7 FPGA using Vivado 2015.4.

Warning

In a case of using GNSS FW without connected RF front-end don't forget to ***switch ON DIP[0] (i_int_clkrf) to enable Test Mode***. Otherwise there wouldn't be generated interrupts and, as result, no UART output.

3.2 VCD-files automatic comparision

3.2.1 Generating VCD-pattern form SystemC model

Edit the following attributes in SystemC target script *debugger/targets/sysc_river_gui.json* to enable vcd-file generation.

- ['InVcdFile','i_river','Non empty string enables generation of stimulus VCD file'].
- ['OutVcdFile','o_river','Non empty string enables VCD file with reference signals']

Files *i_river.vcd* and *o_river.vcd* will be generated. The first one will be used as a RTL simulation stimulus to generate input signals. The second one as a reference.

3.2.2 Compare RIVER SystemC model relative RTL

Run simulation in ModelSim with the following commands using correct pathes for your host:

```
vcd2wlf E:/Projects/GitProjects/riscv_vhdl/debugger/win32build/Debug/i_river.vcd -o e:/i_river.wlf
vcd2wlf E:/Projects/GitProjects/riscv_vhdl/debugger/win32build/Debug/o_river.vcd -o e:/o_river.wlf
wlf2vcd e:/i_river.wlf -o e:/i_river.vcd
vsim -t 1ps -vcdstim E:/i_river.vcd riverlib.RiverTop
vsim -view e:/o_river.wlf
add wave o_river:/SystemC/o_*
add wave sim:/rivertop/*
run 500us
compare start o_river sim
compare add -wave sim:/RiverTop/o_req_mem_valid o_river:/SystemC/o_req_mem_valid
compare add -wave sim:/RiverTop/o_req_mem_write o_river:/SystemC/o_req_mem_write
compare add -wave sim:/RiverTop/o_req_mem_addr o_river:/SystemC/o_req_mem_addr
compare add -wave sim:/RiverTop/o_req_mem_strob o_river:/SystemC/o_req_mem_strob
compare add -wave sim:/RiverTop/o_req_mem_data o_river:/SystemC/o_req_mem_data
compare add -wave sim:/RiverTop/o_dport_ready o_river:/SystemC/o_dport_ready
compare add -wave sim:/RiverTop/o_dport_rdata o_river:/SystemC/o_dport_rdata
compare run
```

Note

In this script I've used `vcd2wlf` and `wlf2vcd` utilities to form compatible with ModelSim VCD-file. Otherwise there're will be errors because ModelSim cannot parse std_logic_vector siganls (only std_logic).

Chapter 4

RISC-V Processor

4.1 Overview

Current repository supports two synthesizable processors: `Rocket` and `River`. Both of them implement open RISC-V ISA. To select what processor to use there's special generic parameter:

```
CFG_COMMON_RIVER_CPU_ENABLE
```

4.2 Rocket CPU

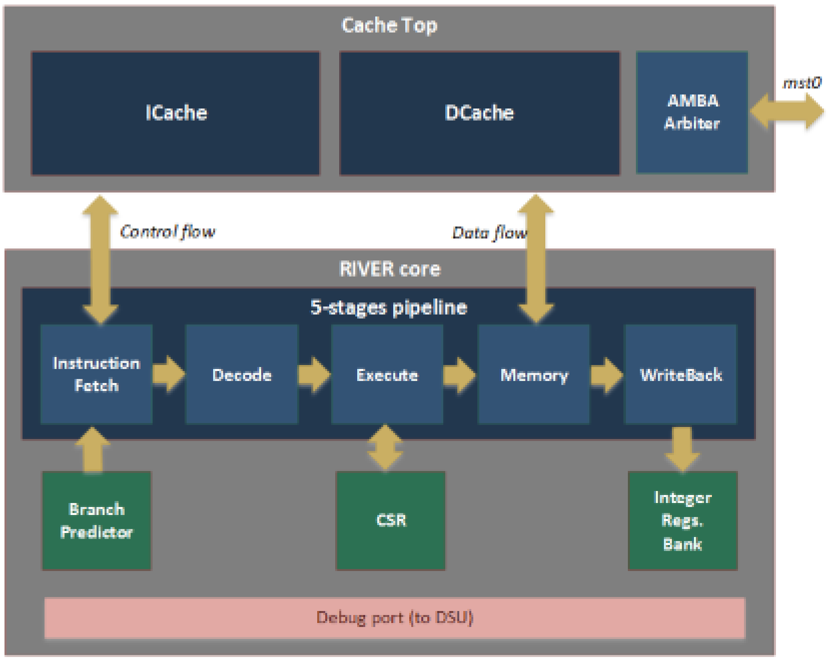
Rocket is the 64-bits single issue, in-order processor developed in Berkley and shared as the sources writen on SCALA language. It uses specally developed library `Chisel` to generate Verilog implementation from SCALA sources.

Rocket Core usually implements all features of the latest ISA specification, either as multi-core support with L2-cache implementation and many other. But it has a set of disadvantages: bad integration with other devices not writen on SCALA, not very-good integration with RTL simulators, no reference model. It shows worse performance than RIVER CPU (for now).

4.3 River CPU

River is my implementation of RISC-V ISA writen on VHDL either as all others parts of shared SoC implementation. There's also availabel precise SystemC model integrated into Simulator which is used as a stimulus during RTL simulation and garantee consistency of functional and SystemC models either as RTL.

River CPU is the 5-stage processor with the classical pipeline structure:


Cache Top
ICache
DCache
AMBA Arbiter
mst0
Control flow
Data flow
RIVER core
5-stages pipeline
Instruction Fetch
Decode
Execute
Memory
WriteBack
Branch Predictor
CSR
Integer Regs. Bank
Debug port (to DSU)

Chapter 5

Peripheries

5.1 Debug Support Unit (DSU)

5.1.1 Overview

Debug Support Unit (DSU) was developed to interact with "RIVER" CPU via its debug port interace. This bus provides access to all internal CPU registers and states and may be additionally extended by request. Run control functionality like 'run', 'halt', 'step' or 'breakpoints' imlemented using proprietary algorithms and intend to simplify integration with debugger application.

Set of general registers and control registers (CSR) are described in RISC-V privileged ISA specification and also available for read and write access via debug port.

Note

Take into account that CPU can have any number of platform specific CSRs that usually not entirely documented.

5.1.2 DSU registers mapping

DSU acts like a slave AMBA AXI4 device that is directly mapped into physical memory. Default address location for our implementation is 0x80020000. DSU directly transforms device offset address into one of regions of the debug port:

- **0x00000..0x08000 (Region 1):** CSR registers.
- **0x08000..0x10000 (Region 2):** General set of registers.
- **0x10000..0x18000 (Region 3):** Run control and debug support registers.
- **0x18000..0x20000 (Region 4):** Local DSU region that doesn't access CPU debug port.

Example:

Bus transaction at address *0x80023C10* will be redirected to Debug port with CSR index *0x782*.

5.1.2.1 CSR Region (32 KB)

User Exception Program Counter (0x00208). ISA offset 0x041.

Bits	Type	Reset	Name	Definition
64	RO	64h'0	uepc	**User mode exception program counter**. Instruction URET is used to return from traps in User Mode into specified instruction pointer. URET is only provided if user-mode traps are supported.

Machine Status Register (0x01800). ISA offset 0x300.

Bits	Type	Reset	Field Name	Bits	Description
1	RW	1b'0	SD	63	Bit summarizes whether either the FS field or XS field signals the presence of some dirty state that will require saving extended user context to memory
22	RW	22h'0	WPRI	62:20	Reserved
5	RW	5h'0	VM (WARL)	28:24	Virtual addressing enable
4	RW	4h'0	WPRI	23:20	Reserved
1	RW	1b'0	MXR	19	**Make eXecutable Readable**
1	RW	1b'0	PUM	18	**Protect User Memory** bit modifies the privilege with which loads access virtual memory
1	RW	1b'0	MPRV	17	Privilege level at which loads and stores execute
2	RW	2h'0	XS	16:15	Context switch reducing flags: 0=All Off; 1=None dirty or clean, some on; 2=None dirty, some clean; 3=Some dirty
2	RW	2h'0	FS	14:13	Context switch reducing flags: 0=Off; 1=Initial; 2=Clean; 3=Dirty
2	RW	2h'0	MPP	12:11	Priviledge mode on MRET
2	RW	2h'0	HPP	10:9	Priviledge mode on HRET
1	RW	1b'0	SPP	8	Priviledge mode on SRET
1	RW	1b'0	MPIE	7	MIE prior to the trap
1	RW	1b'0	HPIE	6	HIE prior to the trap
1	RW	1b'0	SPIE	5	SIE prior to the trap
1	RW	1b'0	UPIE	4	UIE prior to the trap
1	RW	1b'0	MIE	3	Machine interrupt enable bit
1	RW	1b'0	HIE	2	Hypervisor interrupt enable bit
1	RW	1b'0	SIE	1	Super-user interrupt enable bit
1	RW	1b'0	UIE	0	User interrupt enable bit

Machine Trap-Vector Base-Address Register (0x01828). ISA offset 0x305.

Bits	Type	Reset	Field Name	Definition
64	RW	64h'0	mtvec	**Trap-vector Base Address**. The mtvec register is an XLEN-bit read/write register that holds the base address of the M-mode trap vector.

Machine Exception Program Counter (0x01A08). ISA offset 0x341.

Bits	Type	Reset	Field Name	Definition
64	RW	64h'0	mepc	**Machine mode exception program counter**. Instruction MRET is used to return from traps in User Mode into specified instruction pointer. On implementations that do not support instruction-set extensions with 16-bit instruction alignment, the two low bits (mepc[1:0]) are always zero.

Machine Cause Register (0x01A10). ISA offset 0x342.

Bits	Type	Reset	Field Name	Bits	Definition
1	RW	1b'0	Interrupt	63	The Interrupt bit is set if the trap was caused by an interrupt.
63	RW	63h'0	Exception Code	62:0	**Exception code**. The Exception Code field contains a code identifying the last exception. Table 3.6 lists the possible machine-level exception codes.

Machine Cause Register (0x01A18). ISA offset 0x343.

Bits	Type	Reset	Field Name	Bits	Definition
64	RW	64h'0	mbadaddr	63:0	**Exception address**. When a hardware breakpoint is triggered, or an instruction-fetch, load, or store address-misaligned or access exception occurs, mbadaddr is written with the faulting address. mbadaddr is not modified for other exceptions.

Machine ISA Register (0x07880). ISA offset 0xf10.

Bits	Type	Reset	Field Name	Bits	Description
2	RO	2h'2	Base (WARL)	63:62	**Integer ISA width**: 1=32 bits; 2=64 bits; 3=128 bits.
34	RO	64h'0	WIRI	61:28	Reserved.
28	RO	28h'141181	Extension (WARL)	27:0	**Supported ISA extensions**. See priviledge-isa datasheet.

Machine Vendor ID (0x07888). ISA offset 0xf11.

Bits	Type	Reset	Field Name	Bits	Description
64	RO	64h'0	Vendor	63:0	**Vendor ID**. read-only register encoding the manufacturer of the part. This register must be readable in any implementation, but a value of 0 can be returned to indicate the field is not implemented or that this is a non-commercial implementation.

Machine Architecture ID Register (0x07890). ISA offset 0xf12.

Bits	Type	Reset	Field Name	Bits	Description
64	RO	64h'0	marchid	63:0	**Architecture ID**. Read-only register encoding the base microarchitecture of the hart. This register must be readable in any implementation, but a value of 0 can be returned to indicate the field is not implemented. The combination of mvendorid and marchid should uniquely identify the type of hart microarchitecture that is implemented.

Machine implementation ID Register (0x07898). ISA offset 0xf13.

Bits	Type	Reset	Field Name	Bits	Description
64	RO	64h'0	mimplid	63:0	**Implementation ID**. CSR provides a unique encoding of the version of the processor implementation. This register must be readable in any implementation, but a value of 0 can be returned to indicate that the field is not implemented.

Hart ID Register (0x078A0). ISA offset 0xf14.

Bits	Type	Reset	Field Name	Bits	Description
64	RO	64h'0	mhartid	63:0	**Integer ID of hardware thread**. Hart IDs mightnot necessarily be numbered contiguously in a multiprocessor system, but at least one hart musthave a hart ID of zero.

5.1.2.2 General CPU Registers Region (32 KB)

CPU integer registers (0x08000).

Offset	Bits	Type	Reset	Name	Definition
0x08000	64	RW	64h'0	zero	**x0**. CPU General Integer Register hardware connected to zero.
0x08008	64	RW	64h'0	ra	**x1**. Return address.
0x08010	64	RW	64h'0	sp	**x2**. Stack pointer.
0x08018	64	RW	64h'0	gp	**x3**. Global pointer.
0x08020	64	RW	64h'0	tp	**x4**. Thread pointer.
0x08028	64	RW	64h'0	t0	**x5**. Temporaries 0.
0x08030	64	RW	64h'0	t1	**x6**. Temporaries 1.
0x08038	64	RW	64h'0	t2	**x7**. Temporaries 2.
0x08040	64	RW	64h'0	s0/fp	**x8**. CPU General Integer Register 'Saved register 0/ Frame pointer'.
0x08048	64	RW	64h'0	s1	**x9**. Saved register 1.
0x08050	64	RW	64h'0	a0	**x10**. Function argument 0. It is also used to save return value.
0x08058	64	RW	64h'0	a1	**x11**. Function argument 1.
0x08060	64	RW	64h'0	a2	**x12**. Function argument 2.
0x08068	64	RW	64h'0	a3	**x13**. Function argument 3.
0x08070	64	RW	64h'0	a4	**x14**. Function argument 4.
0x08078	64	RW	64h'0	a5	**x15**. Function argument 5.
0x08080	64	RW	64h'0	a6	**x16**. Function argument 6.

Offset	Bits	Type	Reset	Name	Definition
0x08088	64	RW	64h'0	a7	**x17**. Function argument 7.
0x08090	64	RW	64h'0	s2	**x18**. Saved register 2.
0x08098	64	RW	64h'0	s3	**x19**. Saved register 3.
0x080a0	64	RW	64h'0	s4	**x20**. Saved register 4.
0x080a8	64	RW	64h'0	s5	**x21**. Saved register 5.
0x080b0	64	RW	64h'0	s6	**x22**. Saved register 6.
0x080b8	64	RW	64h'0	s7	**x23**. Saved register 7.
0x080c0	64	RW	64h'0	s8	**x24**. Saved register 8.
0x080c8	64	RW	64h'0	s9	**x25**. Saved register 9.
0x080d0	64	RW	64h'0	s10	**x26**. Saved register 10.
0x080d8	64	RW	64h'0	s11	**x27**. Saved register 11.
0x080e0	64	RW	64h'0	t3	**x28**. Temporaries 3.
0x080e8	64	RW	64h'0	t4	**x29**. Temporaries 4.
0x080f0	64	RW	64h'0	t5	**x30**. Temporaries 5.
0x080f8	64	RW	64h'0	t6	**x31**. Temporaries 6.
0x08100	64	RO	64h'0	pc	**Instruction pointer**. Cannot be modified because shows the latest executed instruction address
0x08108	64	RW	64h'0	npc	**Next Instruction Pointer**

5.1.2.3 Run Control and Debug support Region (32 KB)

Run control/status registers (0x10000).

Bits	Type	Reset	Field Name	Bits	Description
44	RW	61h'0	Reserved	63:6	Reserved.
16	RO	16h'0	core_id	15:4	**Core ID**.
1	RW	1b'0	Reserved	3	Reserved.
1	RO	1b'0	breakpoint	2	**Breakpoint**. Status bit is set when CPU was halted due the EBREAK instruction.
1	WO	1b'0	stepping_mode	1	**Stepping mode**. This bit enables stepping mode if the Register 'steps' is non zero.
1	RW	1b'0	halt	0	**Halt mode**. When this bit is set CPU pipeline is in the halted state. CPU can be halted at any time without impact on processing data.

Stepping mode Steps registers (0x10008).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	steps	63:0	**Step counter**. Total number of instructions that should execute CPU before halt. CPU is set into stepping using 'stepping mode' bit in Run Control register.

Clock counter registers (0x10010).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	clock_cnt	63:0	**Clock counter**. Clock counter is used for hardware computation of CPI rate. Clock counter isn't incrementing in Halt state.

Step counter registers (0x10018).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	executed_cnt	63:0	**Step counter**. Total number of executed instructions. Step counter is used for hardware computation of CPI rate.

Breakpoint Control registers (0x10020).

Bits	Type	Reset	Field Name	Bits	Description
63	RW	63h'0	Reserved	63:1	Reserved
1	RW	1b'0	trap_on_break	0	**Trap On Break**. Generate exception 'Breakpoint' on E↩ BRAK instruction if this bit is set or just Halt the pipeline otherwise.

Add hardware breakpoint registers (0x10028).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	add_break	63:0	**Add HW breakpoint address**. Add specified address into Hardware breakpoint stack. In case of matching Instruction Pointer (pc) and any HW breakpoint there's injected EBREAK instruction on hardware level.

Remove hardware breakpoint registers (0x10030).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	rem_break	63:0	**Remove HW breakpoint address**. Remove specified address from Hardware breakpoints stack.

Breakpoint Address Fetch registers (0x10038).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	br_address_fetch	63:0	**Breakpoint fetch address**. Specify address that will be ignored by Fetch stage and used Breakpoint Fetch Instruction value instead. This logic is used to avoid rewriting EBREAK into memory.

Breakpoint Instruction Fetch registers (0x10040).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	br_instr_fetch	63:0	**Breakpoint fetch instruction**. Specify instruction that should executed instead of fetched from memory in a case of matching Breapoint Address Fetch register and Instruction pointer (pc).

5.1.2.4 Local DSU Region (32 KB)

Soft Reset registers (0x18000).

Bits	Type	Reset	Field Name	Bits	Description
63	RW	63h'0	Reserved	63:1	Reserved.
1	RW	1b'0	soft_reset	0	**Soft Reset**. Status bit is set when CPU was halted due the EBREAK instruction.

Miss Access counter registers (0x18008).

Bits	Type	Reset	Field Name	Bits	Description
64	RO	64h'0	miss_access_cnt	63:0	**Miss Access counter**. This value as an additional debugging informantion provided by AXI Controller. It is possible to enable interrupt generation in Interrupt Controller on miss-access.

Miss Access Address registers (0x18010).

Bits	Type	Reset	Field Name	Bits	Description
64	RO	64h'0	miss_access_addr	63:0	**Miss Access address**. Address of the latest miss-accessed transaction. This information comes from AXI Controller.

Bus Utilization registers (0x18040 + n∗2∗sizeof(uint64_t)).

Offset	Bits	Type	Reset	Name	Definition
0x18040	64	RO	64h'0	w_cnt	**Write transactions counter for master 0**. Master 0 is the R↩ IVER CPU by default.
0x18048	64	RO	64h'0	r_cnt	**Read transactions counter for master 0**.
0x18050	64	RO	64h'0	w_cnt	**Write transactions counter for master 1**. Master 1 is unused in a case of configuration with RIVER CPU.
0x18058	64	RO	64h'0	r_cnt	**Read transactions counter for master 1**.
0x18060	64	RO	64h'0	w_cnt	**Write transactions counter for master 2**. Master 2 is the G↩ RETH by default (Ethernet Controller with master interface).
0x18068	64	RO	64h'0	r_cnt	**Read transactions counter for master 2**.

5.2 GPIO Controller

5.2.1 GPIO registers mapping

GPIO Controller acts like a slave AMBA AXI4 device that is directly mapped into physical memory. Default address location for our implementation is defined by 0x80000000. Memory size is 4 KB.

LED register (0x000).

Bits	Type	Reset	Field Name	Bits	Description
24	RW	24h'0	rsrv	24	Reserved
8	RW	8h'0	led	7:0	**LEDs**. Written value directly assigned on SoC output pins and can be used as test signals.

DIP register (0x004).

Bits	Type	Reset	Field Name	Bits	Description
28	RO	28h'0	rsrv	28	Reserved
4	RO	-	dip	3:0	**DIPs**. Input configuration pins value (Read-Only). Configuration pin meaning depends of the used FW.

Set of temporary registers (0x008).

Offset	Bits	Type	Reset	Name	Definition
0x008	32	RW	32h'0	reg32↩ _2	**Temporary register 2**. FW specific register used for debugging purposes.
0x00C	32	RW	32h'0	reg32↩ _3	**Temporary register 3**.
0x010	32	RW	32h'0	reg32↩ _4	**Temporary register 4**.
0x014	32	RW	32h'0	reg32↩ _5	**Temporary register 5**.
0x018	32	RW	32h'0	reg32↩ _6	**Temporary register 6**.

5.3 General Purpose Timers

5.3.1 GPTimers overview

This GPTimers implementation can be additionally configured using the following generic parameters.

Name	Default	Description
irqx	0	**Interrupt pin index** This value is used only as argument in output Plug'n'Play configuration.
tmr_total	2	**Total Number of Timers.** Each timer is the 64-bits counter that can be used for interrupt generation or without.

5.3.2 GPTimers registers mapping

GPTimers device acts like a slave AMBA AXI4 device that is directly mapped into physical memory. Default address location for our implementation is defined by 0x80005000. Memory size is 4 KB.

High Precision Timer register (Least Word) (0x000).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	highcnt	63:0	**High precision counter**. This counter isn't used as a source of interrupt and cannot be stopped from SW.

High Precision Timer register (Most Word) (0x004).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	highcnt	63:0	**High precision counter**. This counter isn't used as a source of interrupt and cannot be stopped from SW.

Pending Timer IRQ register (0x008).

Bits	Type	Reset	Field Name	Bits	Description
32-tmr_total	RW	0	reserved	31:tmr_total	Reserved.
tmr_total	RW	0	pending	tmr_total-1:0	**Pending Bit**. Each timer can be configured to generate interrupt. Simaltenously with interrupt is rising pending bit that has to be lowed by Software.

Timer[0] Control register (0x040).

Bits	Type	Reset	Field Name	Bits	Description
30	RW	30h'0	reserved	31:2	Reserved.
1	RW	1b'0	irq_ena	1	**Interrupt Enable**. Enable the interrupt generation when the timer reaches zero value.
0	RW	1b'0	count_ena	0	**Count Enable**. Enable/Disable counter.

Timer[0] Current Value register (0x048).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	value	63:0	**Timer Value**. Read/Write register with counter's value. When it equals to 0 the 'init_value' will be used to re-initialize counter.

Timer[0] Init Value register (0x050).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	init_value	63:0	**Timer Init Value**. Read/Write register is used for cycle timer re-initializtion. If init_value = 0 and value != 0 then the timer is used as a 'single shot' timer.

Timer[1] Control register (0x060 = 0x040 + Idx $*$ 32).

Bits	Type	Reset	Field Name	Bits	Description
30	RW	30h'0	reserved	31:2	Reserved.
1	RW	1b'0	irq_ena	1	**Interrupt Enable**. Enable the interrupt generation when the timer reaches zero value.
0	RW	1b'0	count_ena	0	**Count Enable**. Enable/Disable counter.

Timer[1] Current Value register (0x068 = 0x48 + Idx $*$ 32).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	value	63:0	**Timer Value**. Read/Write register with counter's value. When it equals to 0 the 'init_value' will be used to re-initialize counter.

Timer[1] Init Value register (0x070 = 0x050 + Idx $*$ 32).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64h'0	init_value	63:0	**Timer Init Value**. Read/Write register is used for cycle timer re-initializtion. If init_value = 0 and value != 0 then the timer is used as a 'single shot' timer.

5.4 Interrupt Controller

5.4.1 IRQ assignments

IRQ pins configuration is the part of generic constants defined in file *ambalib/types_amba4.vhd*. Number of interrupts and its indexes can changed in future releases.

Pin	Name	Description
0	Unused	**Zero** Interrupt pin is unsued and connected to Ground.
1	UART1	**Uart 1 IRQ**. UART device used this line to signal CPU via Interrupt Controller that new data is available or device ready to accept new Rx data.
2	ETHMAC	**Ethernet IRQ**.
3	GPTIMERS	**General Purpose Timers IRQ**.
4	MISS_ACCESS	**Memory Miss Access IRQ**. This interrupt is generated by AXI Controller in a case of access to unmapped memory region.
5	GNSSENGINE	**Gnss Engine IRQ**. Device Specific 1 msec interrupt that schedules critical Navigation Task.

5.4.2 IRQ Controller registers mapping

IRQ Controller acts like a slave AMBA AXI4 device that is directly mapped into physical memory. Default address location for our implementation is defined by 0x80002000. Memory size is 4 KB.

Interrupts Mask register (0x000).

Bits	Type	Reset	Field Name	Bits	Description
32-N	RW	h'0	reserved	31:N	Reserved
N	RW	all 1	mask	N-1:0	**IRQ mask**. 1 equals interrupt disabled; 0 is enabled.

Pending Interrupts register (0x004).

Bits	Type	Reset	Field Name	Bits	Description
32-N	RO	h'0	reserved	31:N	Reserved
N	RO	0	pending	N-1:0	**Pending Bits**. 1 signals rised interrupt. This bit is cleared by writing 1 into the register 'Clear IRQ' or writing 1 into 'Lock Register'.

Clear Interrupt Mask register (0x008).

Bits	Type	Reset	Field Name	Bits	Description
32-N	WO	h'0	reserved	31:N	Reserved
N	WO	0	clear_bit	N-1:0	**Clear IRQ line**. Clear Pending interrupt register bits that are marked with 1s.

Raise Interrupt Mask register (0x00C).

Bits	Type	Reset	Field Name	Bits	Description
32-N	WO	h'0	reserved	31:N	Reserved
N	WO	0	raise_irq	N-1:0	**Rise specified IRQ line manually**. This register can be used for test and debugging either as for 'system calls'.

ISR table address (low word) (0x010).

Bits	Type	Reset	Field Name	Bits	Description
32	WR	0	isr_table	31:0	**Interrupts table address LSB**. This register stores address where located ISR table. This value must be intialized be Software.

ISR table address (high word) (0x014).

Bits	Type	Reset	Field Name	Bits	Description
32	WR	0	isr_table	31:0	**Interrupts table address MSB**. This register stores address where located ISR table. This value must be intialized be Software.

ISR cause code (low word) (0x018).

Bits	Type	Reset	Field Name	Bits	Description
32	WR	0	dbg_cause	31:0	**Cause of te Interrupt LSB**. This register stores the latest cause of the interrupt. This value is optional and updates by ROM ISR handler in current implementation.

ISR cause code (high word) (0x01C).

Bits	Type	Reset	Field Name	Bits	Description
32	WR	0	dbg_cause	31:0	**Cause of the Interrupt MSB**. This register stores the latest cause of the interrupt. This value is optional and updates by ROM ISR handler in current implementation.

Instruction Pointer before trap (low word) (0x020).

Bits	Type	Reset	Field Name	Bits	Description
32	WR	0	dbg_epc	31:0	**npc[31:0] register value before trap** . This register stores copy of xEPC value. This value is optional and updates by ROM ISR handler in current implementation.

Instruction Pointer before trap (high word) (0x024).

Bits	Type	Reset	Field Name	Bits	Description
32	WR	0	dbg_epc	31:0	**npc[63:32] register value before trap**. This register stores copy of xEPC value. This value is optional and updates by ROM ISR handler in current implementation.

Lock interrupt register (0x028).

Bits	Type	Reset	Field Name	Bits	Description
31	WR	31h'0	reserved	31:1	Reserved
1	WR	1b'	lock	0	**Lock interrupts**. Disabled all interrupts when this bit is 1. All new interrupt request marked as postponed and will be raised when 'lock' signal will be cleared.

Lock interrupt register (0x02C).

Bits	Type	Reset	Field Name	Bits	Description
32	WR	0	irq_idx	31:0	**Interrupt Index**. This register stores current interrupt index while in ISR handler. This value is optional and updates by ROM ISR handler in current implementation.

5.5 UART

5.5.1 Overview

This UART implementation can be additionally configured using the following generic parameters.

Name	Default	Description
irqx	0	**Interrupt pin index** This value is used only as argument in output Plug'n'Play configuration.
fifosz	16	**FIFO size.** Size of the Tx and Rx FIFOs in bytes.

5.5.2 UART registers mapping

UART acts like a slave AMBA AXI4 device that is directly mapped into physical memory. Default address location for our implementation is defined by 0x80001000. Memory size is 4 KB.

Control Status register (0x000).

Bits	Type	Reset	Field Name	Bits	Description
16	RW	16h'0	Reserved	31:16	Reserved.
1	RW	1b'0	parity_bit	15	**Enable parity checking**. Serial port setting setup by SW.
1	RW	1b'0	tx_irq_ena	14	**Enable Tx Interrupt**. Generate interrupt when number of symbol in output FIFO less than defined in Tx Threshold register.
1	RW	1b'0	rx_irq_ena	13	**Enable Rx Interrupt**. Generate interrupt when number of available for reading symbol greater or equalt Rx Threshold register.
3	RW	3h'0	Reserved	12:10	Reserved.
1	RO	1b'0	err_stopbit	9	**Stop Bit Error**. This bit is set when the Stoping Bit has the wrnog value.
1	RO	1b'0	err_parity	8	**Parity Error**. This bit is set when the Parity error occurs. Will be automatically cleared by next received symbol if the parity OK.
2	RW	2h'0	Reserved	7:6	Reserved.
1	RO	1b'1	rx_fifo_empty	5	**Receive FIFO is Empty**.
1	RO	1b'0	rx_fifo_fifo	4	**Receive FIFO is Full**.
2	RW	2h'0	Reserved	3:2	Reserved.
1	RO	1b'1	tx_fifo_empty	1	**Transmit FIFO is Empty**.
1	RO	1'b0	tx_fifo_full	0	**Transmit FIFO is Full**.

Scaler register (0x004).

Bits	Type	Reset	Field Name	Bits	Description
32	RW	32h'0	scaler	31:16	**Scale threshold**. This register value is used to transform System Bus clock into port baudrate.

Data register (0x010).

Bits	Type	Reset	Field Name	Bits	Description
24	RW	28h'0	Reserved	31:8	Reserved.
8	RW	8h'0	data	7:0	**Data**. Access to Tx/Rx FIFO data. Writing into this register put data into Tx FIFO. Reading is accomplished from Rx F↩ IFO.

5.6 PNP support

5.6.1 PNP registers mapping

PNP module acts like a slave AMBA AXI4 device that is directly mapped into physical memory. Default address location for our implementation is defined as 0xFFFFF000. Memory size is 4 KB.

HW ID register (0x000).

Bits	Type	Reset	Field Name	Bits	Description
32	RO	CFG_HW_ID	hw_id	31:0	**HW ID**. Read only SoC identificator. Now it contains manually specified date in hex-format. Can be changed via CFG_HW_ID configuration parameter.

FW ID register (0x004).

Bits	Type	Reset	Field Name	Bits	Description
32	RW	32'h0	fw_id	31:0	**Firmware ID**. This value is modified by bootloader or user's firmware. Can be used to simplify firmware version tracking.

AXI Slots Configuration Register (0x008).

Bits	Type	Reset	Field Name	Bits	Description
8	RO	CFG_TECH	tech	7:0	**Technology ID**. Read Only value specifies the target configuration. Possible values: inferred, virtex6, kintex7. Other targets ID could be added in a future.

Bits	Type	Reset	Field Name	Bits	Description
8	RO	CFG_NASTI_SLAVES_TOTAL	slaves	15:8	**Total number of AXI slave slots**. This value specifies maximum number of slave devices connected to the system bus. If device wasn't connected the dummy signals must be applied to the slave interface otherwise SoC behaviour isn't defined.
8	RO	CFG_NASTI_MASTER_TOTAL	masters	23:16	**Total number of AXI master slots**. This value specifies maximum number of master devices connected to the system bus. Slot signals cannot be unconnected either.
8	RO	8'h0	adc_detect	31:24	**ADC clock detector**. This value is used by GNSS firmware to detect presence of the ADC clock frequency that allows to detect presence of the RF front-end board.

Debug IDT register (0x010).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64'h0	idt	63:0	**Debug IDT**. This is debug register used by GNSS firmware to store debug information.

Debug Memory Allocation Pointer register (0x018).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64'h0	malloc_addr	63:0	**Memory Allocation Pointer**. This is debug register used by GNSS firmware to store 'heap' pointer and allows to debug memory management.

Debug Memory Allocation Size register (0x020).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64'h0	malloc_size	63:0	**Memory Allocation size**. This is debug register used by G↩ NSS firmware to store total allocated memory size.

Debug Firmware1 register (0x028).

Bits	Type	Reset	Field Name	Bits	Description
64	RW	64'h0	fwdbg1	63:0	**Firmware debug1**. This is debug register used by GNSS firmware to store temporary information.

5.6.2 PNP Device descriptors

Our SoC implementaion provides capability to read in real-time information about mapped devices. Such information is packed into special device descriptors. Now we can provide 3 types of descriptors:

- Master device descriptor
- Slave device descriptor
- Custom device descriptor

All descriptors mapped sequentually starting from 0xFFFFF040. Each descriptor implements field 'size' in Bytes that specifies offset to the next mapped descriptor.

Master device descriptor

Bits	Description
[7:0]	**Descriptor Size.** Read Only value specifies size in Bytes of the current descriptor. This value should be used as offset to the next descriptor. Master descriptor size is hardwired to PNP_CFG_MASTE↩ R_DESCR_BYTES value (8'h08).
[9:8]	**Descriptor Type.** Master descriptor type is hardwired to PNP_CFG_TYPE_MASTER value (2'b01).
[31:10]	**Reserved.**
[47:32]	**Device ID.** Unique Master identificator.
[63:48]	**Vendor ID.** Unique Vendor identificator.

Slave device descriptor

Bits	Description
[7:0]	**Descriptor Size.** Read Only value specifies size in Bytes of the current descriptor. This value should be used as offset to the next descriptor. Slave descriptor size is hardwired to PNP_CFG_↩ SLAVE_DESCR_BYTES value (8'h10).
[9:8]	**Descriptor Type.** Slave descriptor type is hardwired to PNP_CFG_TYPE_SLAVE value (2'b10).
[15:10]	**Reserved.**
[23:16]	**IRQ ID.** Interrupt line index assigned to the device.
[31:24]	**Reserved.**
[47:32]	**Device ID.** Unique Master identificator.
[63:48]	**Vendor ID.** Unique Vendor identificator.
[75:64]	**zero.** Hardwired to X"000".
[95:76]	**Base Address Mask** specifies the memory region allocated for the device.
[107:96]	**zero.** Hardwired to X"000".
[127:108]	**Base Address** value of the device.

Chapter 6

RISC-V debugger

Overview

This debugger was specially developed as a software utility to interact with our SOC implementation in `riscv_soc` repository. The main purpose was to provide convinient way to develop and debug our Satellite Navigation firmware that can not be debugged by any other tool provided RISC-V community. Additionally, we would like to use the single unified application capable to work with Real and Simulated platforms without any modification of source code. Debugger provides base functionality such as: run control, read/write memory, registers and CSRs, breakpoints. It allows to reload FW image and reset target. Also we are developing own version of the CPU simulator (analog of `spike`) that can be extended with peripheries models to Full SOC simulator. These extensions for the debugger simplify porting procedure (Zephyr OS for an example) so that simulation doesn't require any hardware and allows to develop SW and HW simultaneously.

6.1 Project structure

General idea of the project is to develop one `Core` library providing API methods for registering `classes`, `services`, `attributes` and methods to interact with them. Each extension plugin registers one or several class services performing some usefull work. All plugins are built as an independent libraries that are opening by `Core` library at initialization stage with the call of method **plugin_init()**. All Core API methods start with `RISCV_`... prefix:

```
void RISCV_register_class(IFace *icls);

IFace *RISCV_create_service(IFace *iclass, const char *name,
                            AttributeType *args);

IFace *RISCV_get_service(const char *name);
...
```

Configuration of the debugger and plugins is fully described in JSON formatted configuration files **targets/target↩ _name.json**. These files store all instantiated services names, attributes values and interconnect among plugins. This configuration can be saved to/load from file at any time. By default command `exit` will save current debugger state into file (including full command history).

Note

You can manually add/change new Registers/CSRs names and indexes by modifying this config file without changing source code.

Folders description

1. **libdgb64g** - Core library (so/dll) that provides standard API methods defined in file `api_core.h`.
2. **appdbg64g** - Executable (exe) file implements functionality of the console debugger.
3. *Plugins:*
   (a) **simple_plugin** - Simple plugin (so/dll library) just for demonstration of the integration with debugger.
   (b) **cpu_fnc_plugin** - Functional model of the RISC-V CPU (so/dll library).
   (c) **cpu_sysc_plugin** - Precise SystemC model of RIVER CPU (so/dll library).
   (d) **socsim_plugin** - Functional models of the peripheries and assembled board (so/dll library). This plugin registers several classes: `UART`, `GPIO`, `SRAM`, `ROMs` and etc.

6.2 Ethernet setup

Overview

The Ethernet Media Access Controller (GRETH) provides an interface between an AMBA-AXI bus and Ethernet network. It supports 10/100 Mbit speed in both full- and half-duplex modes. Integrated EDCL submodule implements hardware decoding of UDP traffic and redirects EDCL request directly on AXI system bus. The AMBA interface consists of an AXI slave interface for configuration and control and an AXI master interface for transmit and receive data. There is one DMA engine for the transmitter and one for receiver. EDCL submodule and both DMA engines share the same AXI master interface.

To make development board visible in your local network your should properly specify connection properties. In this chapter I will show how to configure the host computer (Windows 7 or Linux) to communicate with the FPGA hardware over Ethernet.

Note

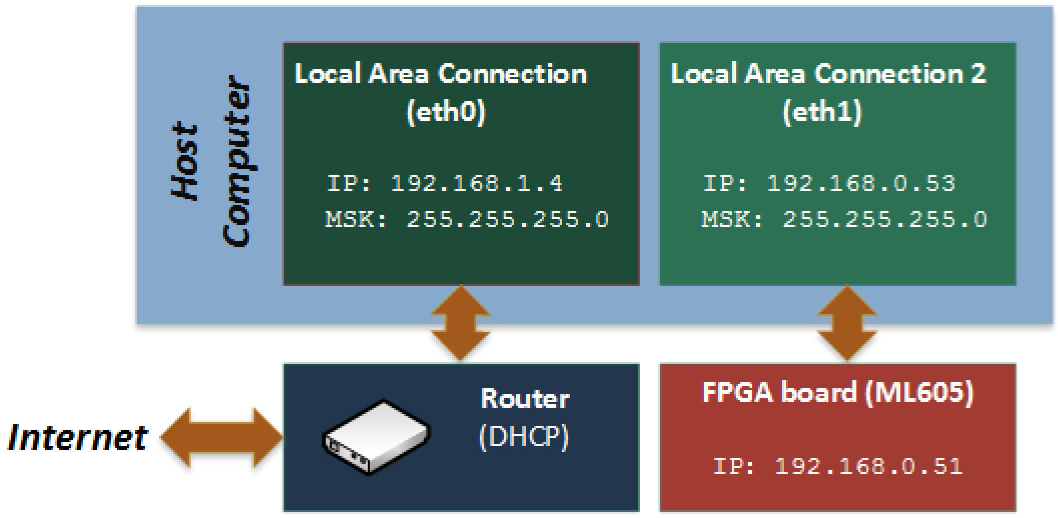
If you also want simultaneous Internet access your host computer requires a second Ethernet port. I couldn't find workable configuration via router.

Warning

I recommend you to make restore point before you start.

6.2.1 Configure Windows Host

Let's setup the following network configuration that allows to work with FPGA board and to be connected to Internet. I use different Ethernet ports and different subnets (192.168.0.x and 192.168.1.x accordingly).



Host IP and subnet definition:

1. Open `cmd` console.
2. Use `ipconfig` command to determine network settings.

```
ipconfig /all
```

3. Find your IP address (in my case it's 192.168.1.4)
4. Check and change if needed default IP address of SOC as follow.

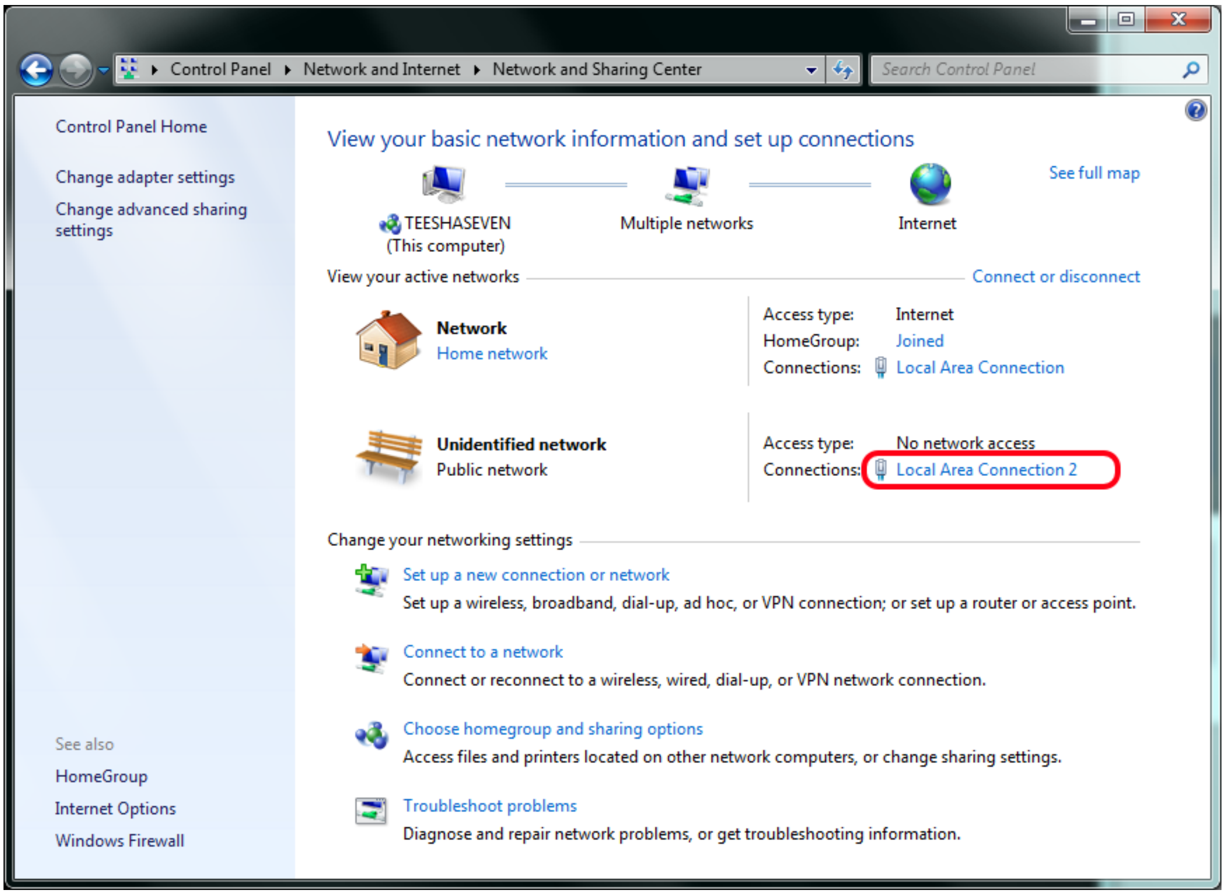
Setup hard-reset FPGA IP address:

1. Open in editor *rocket_soc.vhd*.
2. Find place where *grethaxi* module is instantiated.
3. Change generic **ipaddrh** and **ipaddrl** parameters so that they belonged another subnet (Default values: C0A8.0033 corresponding to 192.168.0.51) than Internet connection.

Configure the Ethernet card for your FPGA hardware

1. Load pre-built image file into FPGA board (located in *./rocket_soc/bit_files/* folder) or use your own one.

2. Open **Network and Sharing Center** via Control Panel


Control Panel ▸ Network and Internet ▸ Network and Sharing Center

Search Control Panel

Control Panel Home

Change adapter settings

Change advanced sharing settings

See also

HomeGroup

Internet Options

Windows Firewall

View your basic network information and set up connections

TEESHASEVEN (This computer) — Multiple networks — Internet

See full map

View your active networks

Connect or disconnect

Network
Home network

Access type: Internet
HomeGroup: Joined
Connections: Local Area Connection

Unidentified network
Public network

Access type: No network access
Connections: Local Area Connection 2

Change your networking settings

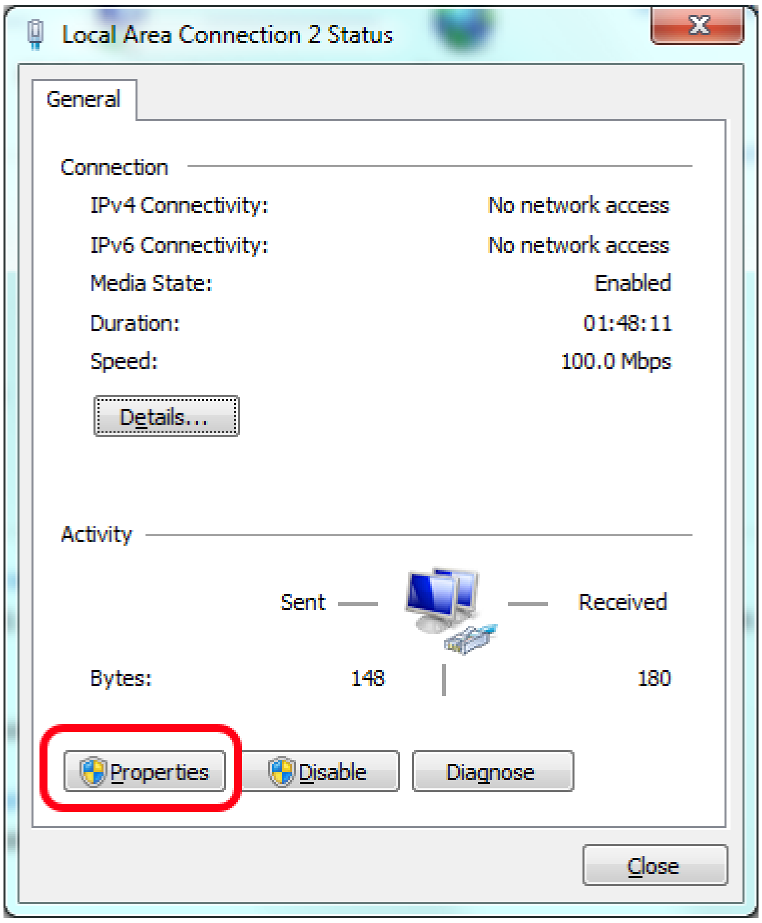
Set up a new connection or network
Set up a wireless, broadband, dial-up, ad hoc, or VPN connection; or set up a router or access point.

Connect to a network
Connect or reconnect to a wireless, wired, dial-up, or VPN network connection.

Choose homegroup and sharing options
Access files and printers located on other network computers, or change sharing settings.

Troubleshoot problems
Diagnose and repair network problems, or get troubleshooting information.

```
-# Click on <b>Local Area Connection 2</b> link
```



Local Area Connection 2 Status

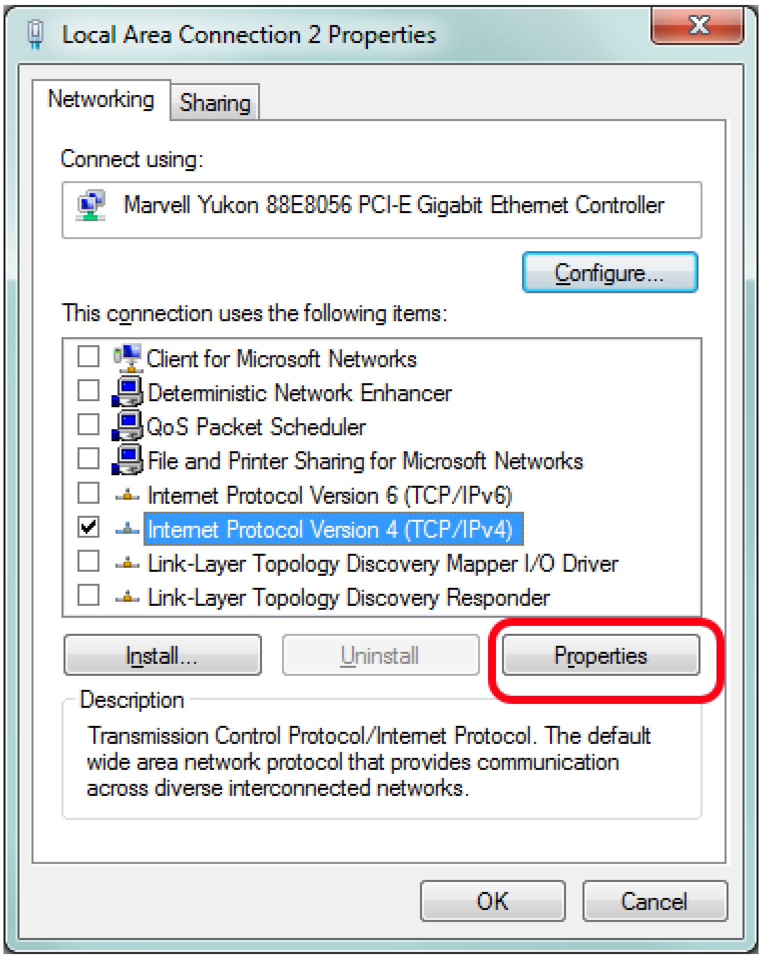
General

Connection

```
IPv4 Connectivity:        No network access
IPv6 Connectivity:        No network access
Media State:                        Enabled
Duration:                          01:48:11
Speed:                           100.0 Mbps
```

```
Bytes:        148        180
```

-# Click on <b>Properties</b> to open properties dialog.


Local Area Connection 2 Properties

Networking | Sharing

Connect using:

Marvell Yukon 88E8056 PCI-E Gigabit Ethernet Controller

Configure...

This connection uses the following items:

- [ ] Client for Microsoft Networks
- [ ] Deterministic Network Enhancer
- [ ] QoS Packet Scheduler
- [ ] File and Printer Sharing for Microsoft Networks
- [ ] Internet Protocol Version 6 (TCP/IPv6)
- [✓] Internet Protocol Version 4 (TCP/IPv4)
- [ ] Link-Layer Topology Discovery Mapper I/O Driver
- [ ] Link-Layer Topology Discovery Responder

Install... | Uninstall | Properties

Description

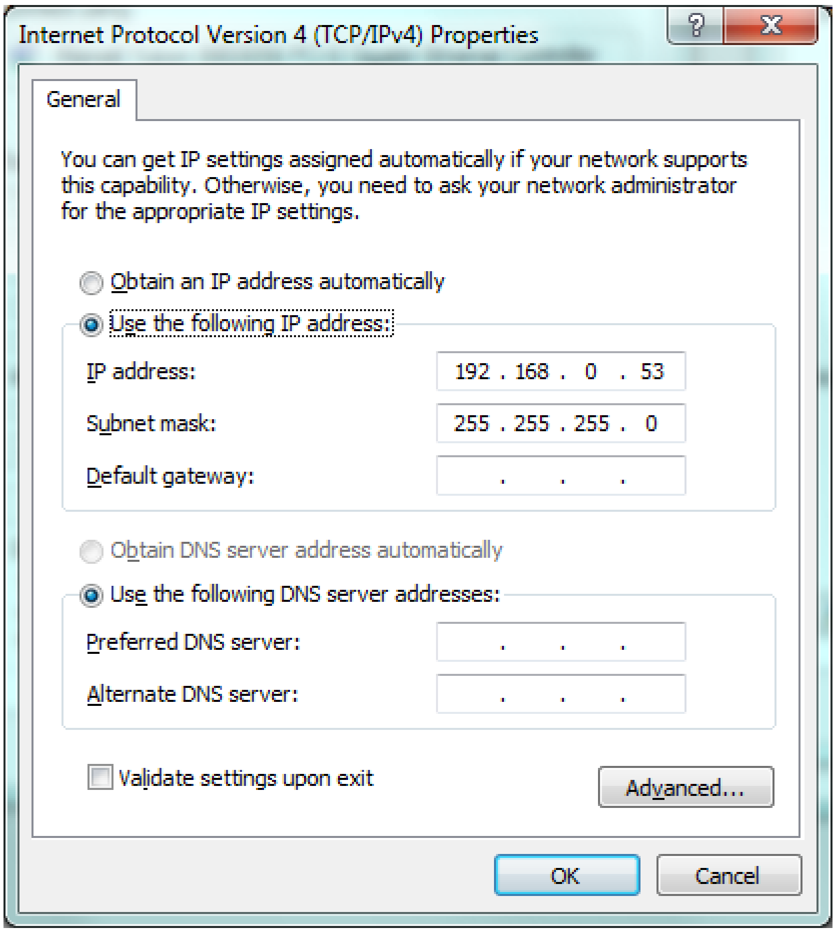
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

OK | Cancel

```
-# Disable all network services except <b>Internet Protocol Version 4</b>
```

as shown on figure above.

1. Select enabled service and click on **Properties** button.


Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

( ) Obtain an IP address automatically
[✓ Use the following IP address:]

IP address: 192 . 168 . 0 . 53
Subnet mask: 255 . 255 . 255 . 0
Default gateway: . . .

( ) Obtain DNS server address automatically
[✓ Use the following DNS server addresses:]

Preferred DNS server: . . .
Alternate DNS server: . . .

[ ] Validate settings upon exit

```
-# Specify unique IP as shown above so that FPGA and your Local
```

Connection were placed **in the same subnet**.

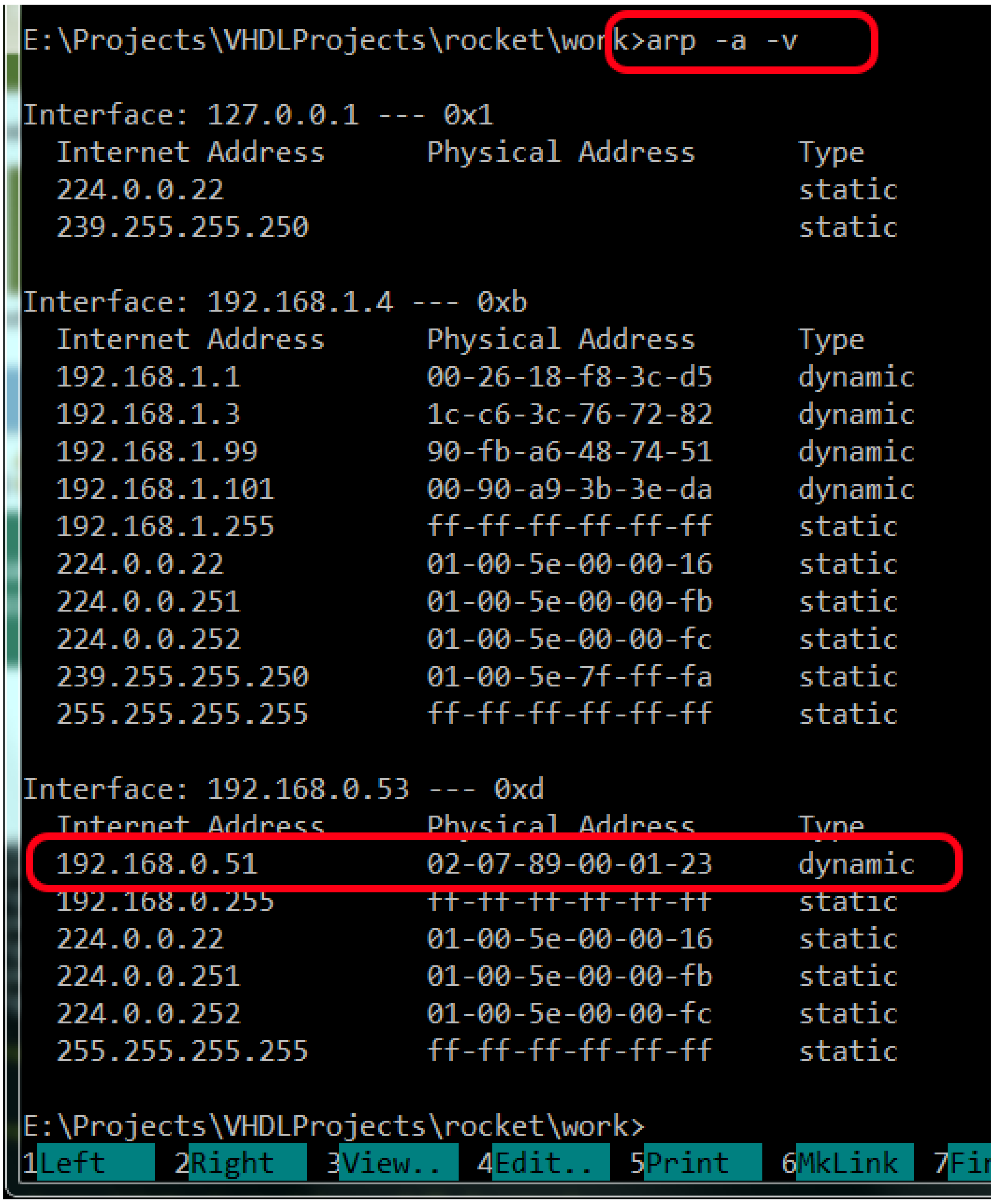
1. Leave the subnet mask set to the default value 255.255.255.0.

2. Click OK.

Check connection

1. Check presence of the Ethernet activity by blinking LEDs near the Ethernet connector on FPGA board
2. Run `arp` command to see arp table entries.

```
arp -a -v
```



```
E:\Projects\VHDLProjects\rocket\work>arp -a -v

Interface: 127.0.0.1 --- 0x1
  Internet Address      Physical Address      Type
  224.0.0.22                                  static
  239.255.255.250                             static

Interface: 192.168.1.4 --- 0xb
  Internet Address      Physical Address      Type
  192.168.1.1           00-26-18-f8-3c-d5     dynamic
  192.168.1.3           1c-c6-3c-76-72-82     dynamic
  192.168.1.99          90-fb-a6-48-74-51     dynamic
  192.168.1.101         00-90-a9-3b-3e-da     dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

Interface: 192.168.0.53 --- 0xd
  Internet Address      Physical Address      Type
  192.168.0.51          02-07-89-00-01-23     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

E:\Projects\VHDLProjects\rocket\work>
1Left    2Right    3View..   4Edit..   5Print    6MkLink   7Fir
```

`-# MAC supports only ARP and EDCL requests on hardware level and it cannot`

respond on others without properly installed software. By this reason ping won't work without running OS on FPGA target but it maybe usefull to ping FPGA target so that it can force updating of the ARP table or use the commands:

```
ipconfig /release
ipconfig /renew
```

6.2.2 Configure Linux Host

Let's setup the similar network configuration on Linux host.

1. Check **ipaddrh** and **ipaddrl** values that are hardcoded on top-level of SOC (default values: C0A8.0033 corresponding to 192.168.0.51).
2. Set host IP value in the same subnet using the `ifconfig` command. You might need to enter a password to use the `sudo` command.

   ```
   % sudo ifconfig eth0 192.168.0.53 netmask 255.255.255.0
   ```

3. Enter the following command in the shell to check that the changes took effect:

```
% ifconfig eth0
```

6.2.3 Run Application

Now your FPGA board is ready to interact with the host computer via Ethernet. You can find detailed information about MAC (GRETH) in `GRLIB IP Core User's Manual`.

There you can find:

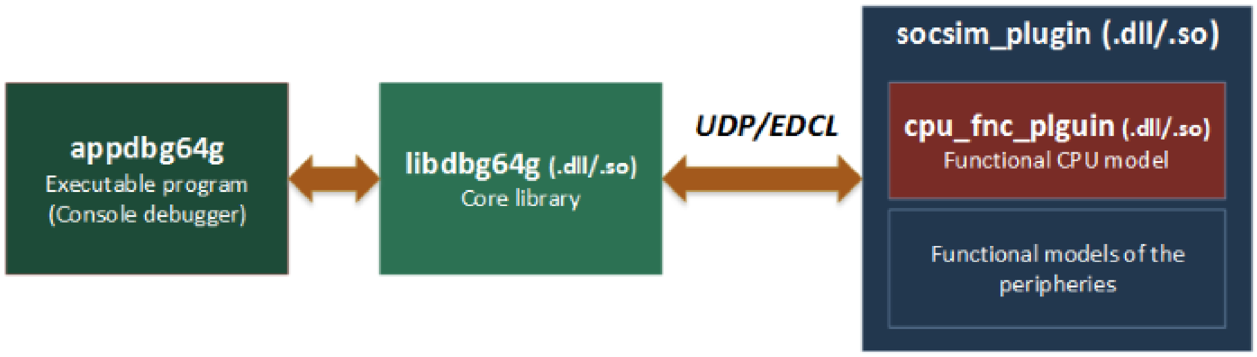
1. DMA Configuration registers description (Rx/Tx Descriptors tables and entries).
2. EDCL message format.
3. `GRLIB` itself includes C-example that configure MAC Rx/Tx queues and start transmission of the 1500 Mbyte of data to define Bitrate in Mbps.

We provide debugger functionality via Ethernet. See Debugger description page.

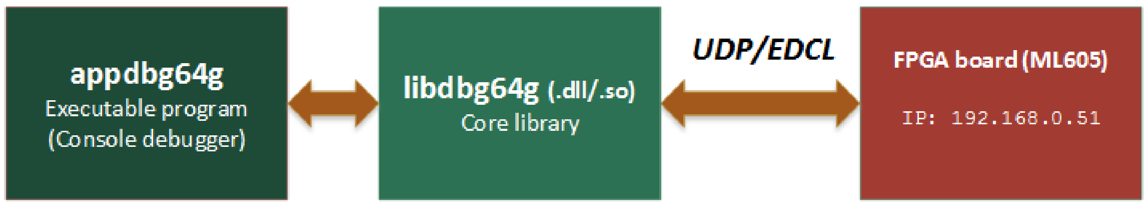
6.3 Debug session

6.3.1 Plugins interaction

Core library uses UDP protocol to communicate with all targets: FPGA or simulators. The general structure is looking like on the following figure:



or with real Hardware



GUI plugin uses QT-libraries and interacts with the core library using the text console input interface. GUI generates the same text commands that are available in debugger console for any who's using this debugger. That's why any presented in GUI widgets information can be achieved in console mode.

6.3.2 Start Debugger

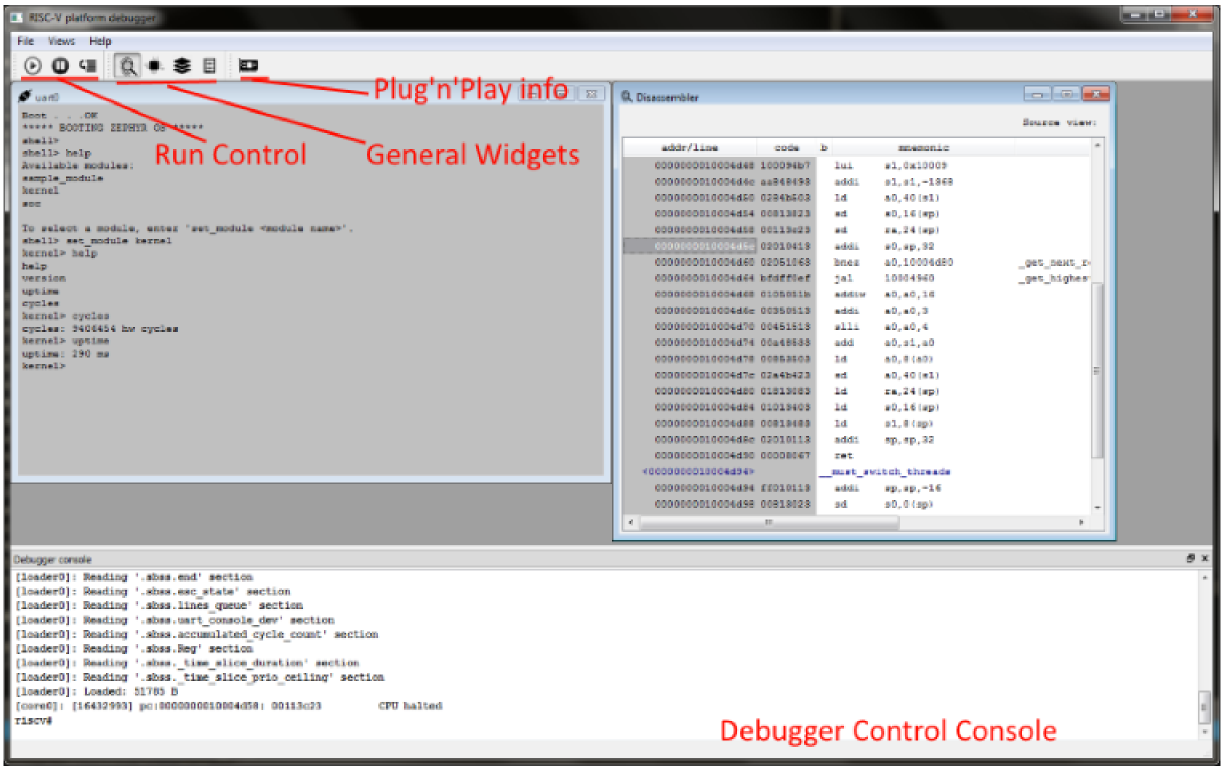
We provide several targets that can run software (bootloader, firmware or user specific application) without any source code modifications:

Start Configuration	Description
$ ./_run_functional_sim.sh[bat]	Functional RISC-V Full System Model
$ ./_run_systemc_sim.sh[bat]	Use SystemC Precise Model of RIVER CPU
$ ./_run_fpga_gui.sh[bat]	FPGA board. Default port 'COM3', TAP IP = 192.168.0.51

To run debugger with the real FPGA target connected via Ethernet do:

```
# cd rocket_soc/debugger/win32build/debug
# _run_functional_sim.bat
```

The result should look like on the picture below:



```
Boot . . . .OK
***** BOOTING ZEPHYR OS *****
shell>
shell> help
Available modules:
sample_module
kernel
soc

To select a module, enter 'set_module <module name>'.
shell> set_module kernel
kernel> help
help
version
uptime
cycles
kernel> cycles
cycles: 3406454 hw cycles
kernel> uptime
uptime: 290 ms
kernel>
```

```
addr/line         code      b  mnemonic
0000000010004d48  100094b7     lui     s1,0x10009
0000000010004d4c  aa848493     addi    s1,s1,-1368
0000000010004d50  0284b503     ld      a0,40(s1)
0000000010004d54  00813823     sd      s0,16(sp)
0000000010004d58  00113c23     sd      ra,24(sp)
0000000010004d5c  02010413     addi    s0,sp,32
0000000010004d60  02051063     bnez    a0,10004d80              _get_next_r
0000000010004d64  bddff0ef     jal     10004940                 _get_highes
0000000010004d68  0105051b     addiw   a0,a0,16
0000000010004d6c  00350513     addi    a0,a0,3
0000000010004d70  00451513     slli    a0,a0,4
0000000010004d74  00a48533     add     a0,s1,a0
0000000010004d78  00853503     ld      a0,8(a0)
0000000010004d7c  02a4b423     sd      a0,40(s1)
0000000010004d80  01813083     ld      ra,24(sp)
0000000010004d84  01013403     ld      s0,16(sp)
0000000010004d88  00813483     ld      s1,8(sp)
0000000010004d8c  02010113     addi    sp,sp,32
0000000010004d90  00008067     ret
<0000000010004d94>          __must_switch_threads
0000000010004d94  ff010113     addi    sp,sp,-16
0000000010004d98  00813023     sd      s0,0(sp)
```

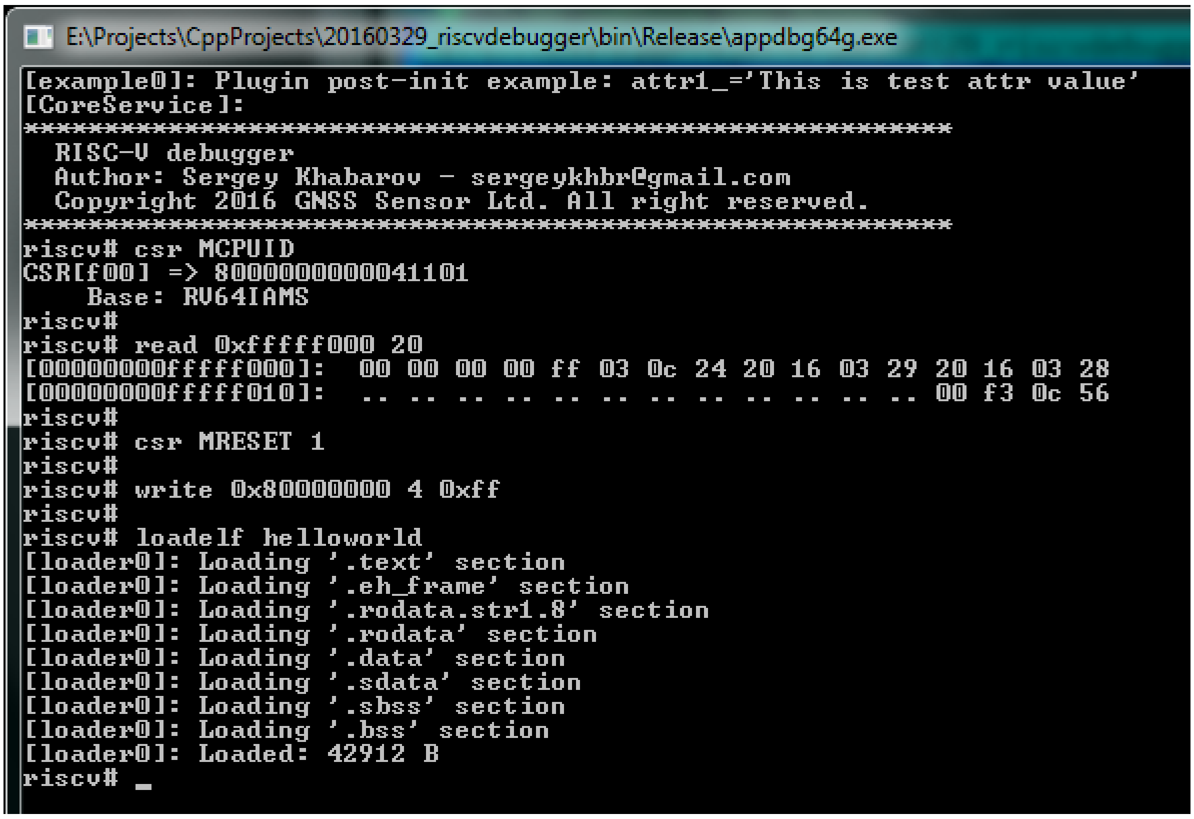
```
[loader0]: Reading '.sbss.end' section
[loader0]: Reading '.sbss.esc_state' section
[loader0]: Reading '.sbss.lines_queue' section
[loader0]: Reading '.sbss.uart_console_dev' section
[loader0]: Reading '.sbss.accumulated_cycle_count' section
[loader0]: Reading '.sbss.Reg' section
[loader0]: Reading '.sbss._time_slice_duration' section
[loader0]: Reading '.sbss._time_slice_prio_ceiling' section
[loader0]: Loaded: 31785 B
[core0]: [16432993] pc:0000000010004d58: 00113c23         CPU halted
riscv#
```

Example of the debug session

Switch ON all User LEDs on board:

```
riscv# help                        -- Print full list of commands
riscv# csr MCPUID                  -- Read supported ISA extensions
riscv# read 0xfffff000 20          -- Read 20 bytes from PNP module
riscv# write 0x80000000 4 0xff     -- Write into GPIO new LED value
riscv# loadelf helloworld          -- Load elf-file to board RAM and run
```

Console mode view


E:\Projects\CppProjects\20160329_riscvdebugger\bin\Release\appdbg64g.exe

```
[example0]: Plugin post-init example: attr1_='This is test attr value'
[CoreService]:
**********************************************************
  RISC-V debugger
  Author: Sergey Khabarov - sergeykhbr@gmail.com
  Copyright 2016 GNSS Sensor Ltd. All right reserved.
**********************************************************
riscv# csr MCPUID
CSR[f00] => 8000000000041101
    Base: RV64IAMS
riscv#
riscv# read 0xfffff000 20
[00000000fffff000]:  00 00 00 00 ff 03 0c 24 20 16 03 29 20 16 03 28
[00000000fffff010]:  .. .. .. .. .. .. .. .. .. .. .. .. 00 f3 0c 56
riscv#
riscv# csr MRESET 1
riscv#
riscv# write 0x80000000 4 0xff
riscv#
riscv# loadelf helloworld
[loader0]: Loading '.text' section
[loader0]: Loading '.eh_frame' section
[loader0]: Loading '.rodata.str1.8' section
[loader0]: Loading '.rodata' section
[loader0]: Loading '.data' section
[loader0]: Loading '.sdata' section
[loader0]: Loading '.sbss' section
[loader0]: Loading '.bss' section
[loader0]: Loaded: 42912 B
riscv# _
```

6.3.3 Debug Zephyr OS kernel with symbols

Build Zephyr kernel from scratch using our patches enabling 64-bits RISC-V architecture support:

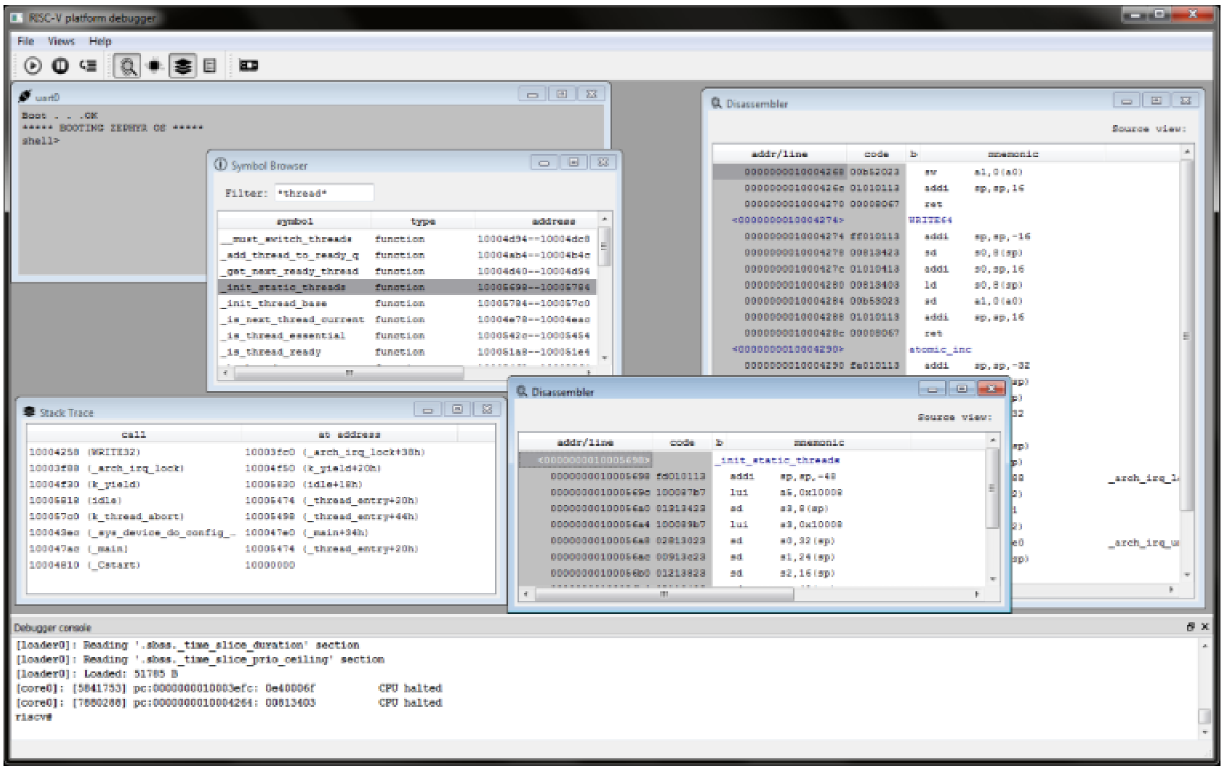
```
$ mkdir zephyr_160
$ cd zephyr_160
$ git clone https://gerrit.zephyrproject.org/r/zephyr
$ cd zephyr
$ git checkout tags/v1.6.0
$ cp ../../riscv_vhdl/zephyr/v1.6.0-riscv64-base.diff .
$ cp ../../riscv_vhdl/zephyr/v1.6.0-riscv64-exten.diff .
$ git apply v1.6.0-riscv64-base.diff
$ git apply v1.6.0-riscv64-exten.diff
```

Then build elf-file:

```
$ export ZEPHYR_BASE=/home/zephyr_160/zephyr
$ cd zephyr/samples/shell
$ make ARCH=riscv64 CROSS_COMPILE=/home/your_path/gnu-toolchain-rv64ima/bin/riscv64-unknown-elf- BOARD=
      riscv_gnss 2>&1
```

Load debug symbols from elf-file without target reprogramming (or with):

```
riscv# loadelf zephyr.elf
riscv# loadelf zephyr.elf nocode
```



```
Boot . . .OK
***** BOOTING ZEPHYR OS *****
shell>
```

Filter: *thread*

symbol	type	address
__must_switch_threads	function	10004d94--10004dc8
_add_thread_to_ready_q	function	10004ab4--10004b4c
_get_next_ready_thread	function	10004d40--10004d94
_init_static_threads	function	10005698--10005784
_init_thread_base	function	10005784--100057c0
_is_next_thread_current	function	10004e78--10004eac
_is_thread_essential	function	1000542c--10005454
_is_thread_ready	function	100051a8--100051e4

Source view:

addr/line	code	b	mnemonic
0000000010004268	00b52023		sw a1,0(a0)
000000001000426c	01010113		addi sp,sp,16
0000000010004270	00008067		ret
<0000000010004274>			WRITE64
0000000010004274	ff010113		addi sp,sp,-16
0000000010004278	00813423		sd s0,8(sp)
000000001000427c	01010413		addi s0,sp,16
0000000010004280	00813403		ld s0,8(sp)
0000000010004284	00b53023		sd a1,0(a0)
0000000010004288	01010113		addi sp,sp,16
000000001000428c	00008067		ret
<0000000010004290>			atomic_inc
0000000010004290	fe010113		addi sp,sp,-32

call	at address
10004258 (WRITE32)	10003fc0 (_arch_irq_lock+38h)
10003f88 (_arch_irq_lock)	10004f50 (k_yield+20h)
10004f30 (k_yield)	10005830 (idle+18h)
10005818 (idle)	10005474 (_thread_entry+20h)
100057c0 (k_thread_abort)	10005498 (_thread_entry+44h)
100043ec (_sys_device_do_config_…	100047e0 (_main+34h)
100047ac (_main)	10005474 (_thread_entry+20h)
10004810 (_Cstart)	10000000

Source view:

addr/line	code	b	mnemonic
<0000000010005698>			_init_static_threads
0000000010005698	fd010113		addi sp,sp,-48
000000001000569c	100087b7		lui a5,0x10008
00000000100056a0	01313423		sd s3,8(sp)
00000000100056a4	100089b7		lui s3,0x10008
00000000100056a8	02813023		sd s0,32(sp)
00000000100056ac	00913c23		sd s1,24(sp)
00000000100056b0	01213823		sd s2,16(sp)

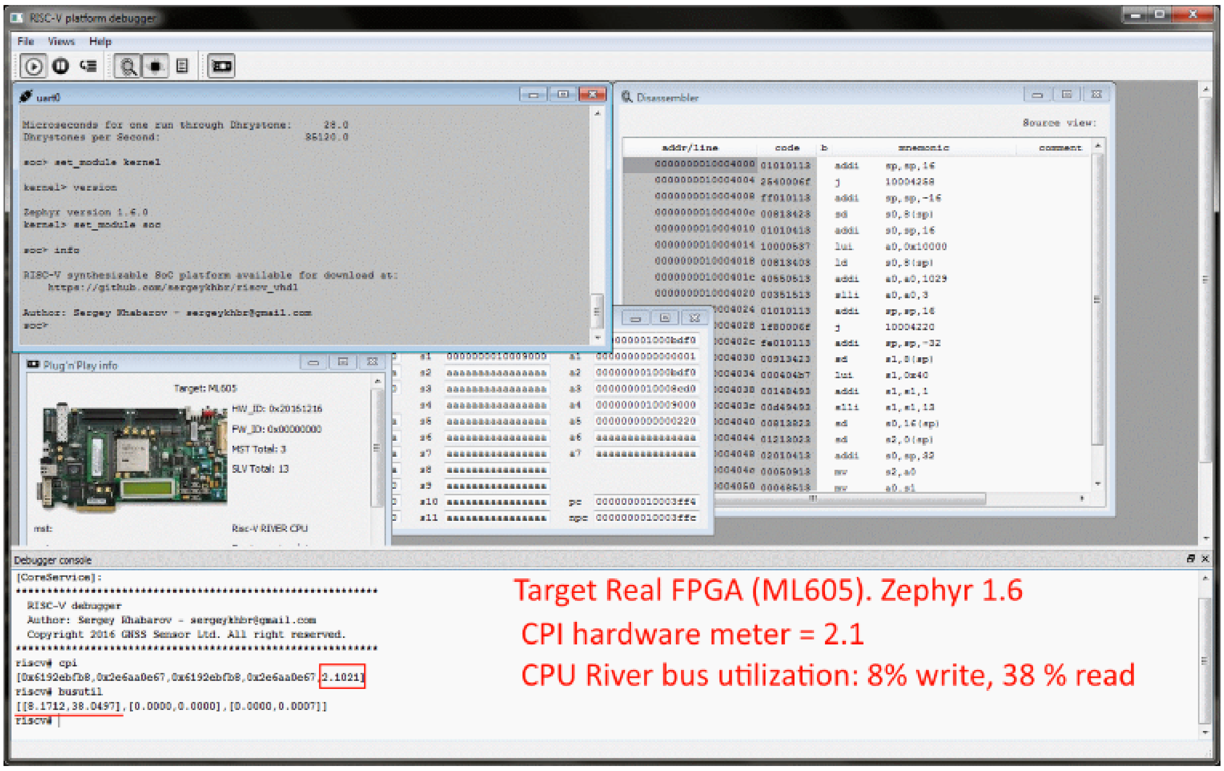
```
[loader0]: Reading '.sbss._time_slice_duration' section
[loader0]: Reading '.sbss._time_slice_prio_ceiling' section
[loader0]: Loaded: 51785 B
[core0]: [5841753] pc:000000001000 3efc: 0e40006f          CPU halted
[core0]: [7880288] pc:0000000010004264: 00813403          CPU halted
riscv#
```

Now becomes available the following features:

- Stack trace with function names
- Function names in Disassembler including additional information for branch and jump instructions in column `'comment'`.
- Symbol Browser with filter.
- Opening Disassembler and Memory Viewer widgets in a new window by name.

Debugger provides additional features that could simplify software development:

- Clock Per Instruction (CPI) hardware measure
- Bus utilization information
- Others. List of a new features is constantly increasing.



```
Microseconds for one run through Dhrystone:      28.0
Dhrystones per Second:                        35120.0

soc> set_module kernel

kernel> version

Zephyr version 1.6.0
kernel> set_module soc

soc> info

RISC-V synthesizable SoC platform available for download at:
    https://github.com/sergeykhbr/riscv_vhdl

Author: Sergey Khabarov - sergeykhbr@gmail.com
soc>
```

```
Target: ML605
HW_ID: 0x20161216
FW_ID: 0x00000000
MST Total: 3
SLV Total: 13
mst:          Risc-V RIVER CPU
```

```
s1   0000000010009000     a1   0000000000000001
s2   aaaaaaaaaaaaaaaa     a2   000000001000bdf0
s3   aaaaaaaaaaaaaaaa     a3   0000000010008cd0
s4   aaaaaaaaaaaaaaaa     a4   0000000010009000
s5   aaaaaaaaaaaaaaaa     a5   0000000000000220
s6   aaaaaaaaaaaaaaaa     a6   aaaaaaaaaaaaaaaa
s7   aaaaaaaaaaaaaaaa     a7   aaaaaaaaaaaaaaaa
s8   aaaaaaaaaaaaaaaa
s9   aaaaaaaaaaaaaaaa
s10  aaaaaaaaaaaaaaaa     pc   0000000010003ff4
s11  aaaaaaaaaaaaaaaa     npc  0000000010003ffc
```

```
                                                          Source view:
    addr/line            code      b    mnemonic               comment
0000000010004000    01010113       addi     sp,sp,16
0000000010004004    2540006f       j        10004258
0000000010004008    ff010113       addi     sp,sp,-16
000000001000400c    00813423       sd       s0,8(sp)
0000000010004010    01010413       addi     s0,sp,16
0000000010004014    10000537       lui      a0,0x10000
0000000010004018    00813403       ld       s0,8(sp)
000000001000401c    40550513       addi     a0,a0,1029
0000000010004020    00351513       slli     a0,a0,3
        ...0004024  01010113       addi     sp,sp,16
        ...0004028  1f80006f       j        10004220
        ...000402c  fe010113       addi     sp,sp,-32
        ...0004030  00913423       sd       s1,8(sp)
        ...0004034  000404b7       lui      s1,0x40
        ...0004038  00148493       addi     s1,s1,1
        ...000403c  00d49493       slli     s1,s1,13
        ...0004040  00813823       sd       s0,16(sp)
        ...0004044  01213023       sd       s2,0(sp)
        ...0004048  02010413       addi     s0,sp,32
        ...000404c  00050913       mv       s2,a0
        ...0004050  00048513       mv       a0,s1
```

```
[CoreService]:
***********************************************************
  RISC-V debugger
  Author: Sergey Khabarov - sergeykhbr@gmail.com
  Copyright 2016 GNSS Sensor Ltd. All right reserved.
***********************************************************
riscv# cpi
[0x6192ebfb8,0x2e6aa0e67,0x6192ebfb8,0x2e6aa0e67,2.1021]
riscv# busutil
[[8.1712,38.0497],[0.0000,0.0000],[0.0000,0.0007]]
riscv# 
```

Target Real FPGA (ML605). Zephyr 1.6
CPI hardware meter = 2.1
CPU River bus utilization: 8% write, 38 % read

6.4 Troubleshooting

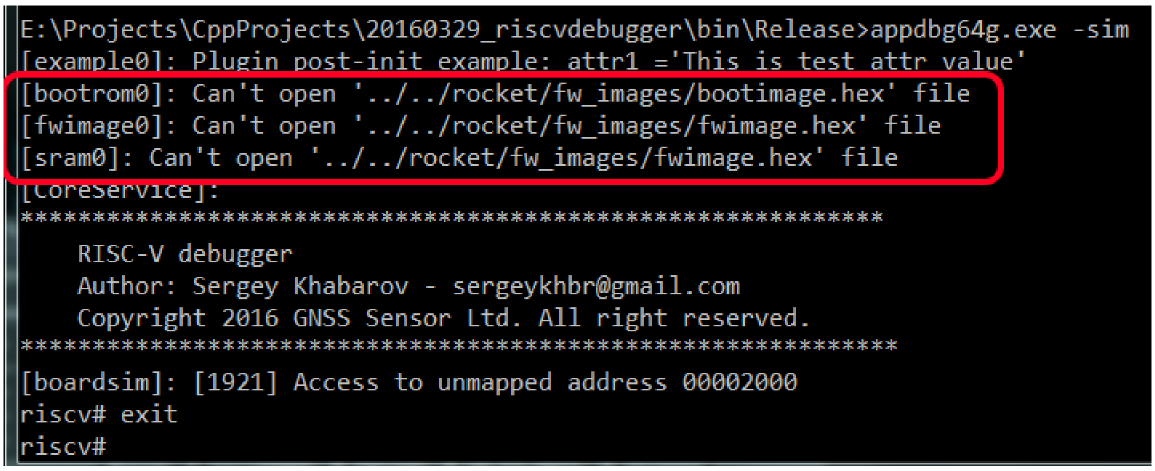
Image Files not found

Can't open COM3 when FPGA is used

EDCL: No response. Break read transaction

6.4.1 Image Files not found

If you'll get the error messages that image files not found

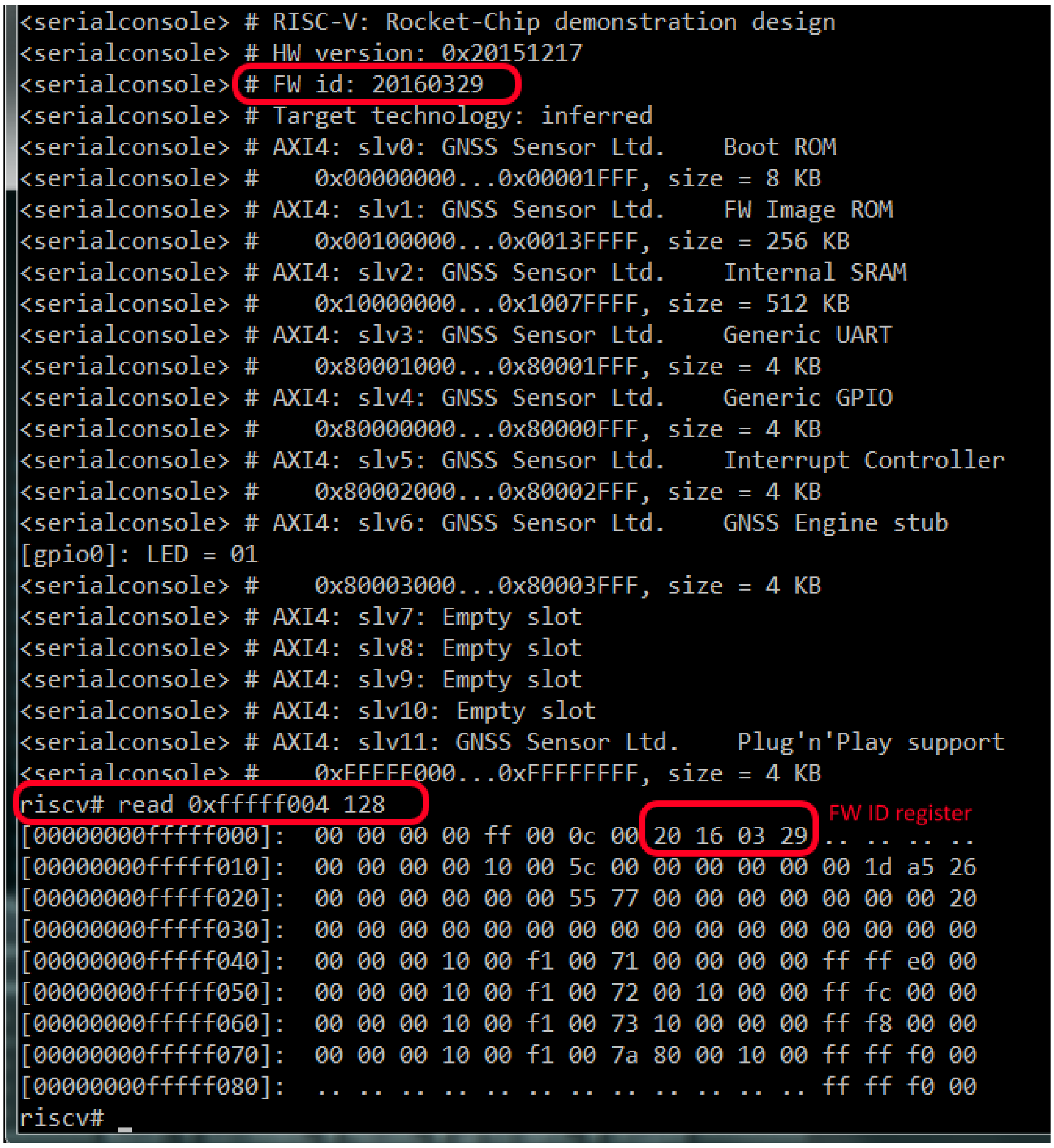


```
E:\Projects\CppProjects\20160329_riscvdebugger\bin\Release>appdbg64g.exe -sim
[example0]: Plugin post-init example: attr1 ='This is test attr value'
[bootrom0]: Can't open '../../rocket/fw_images/bootimage.hex' file
[fwimage0]: Can't open '../../rocket/fw_images/fwimage.hex' file
[sram0]: Can't open '../../rocket/fw_images/fwimage.hex' file
[CoreService]:
************************************************************
    RISC-V debugger
    Author: Sergey Khabarov - sergeykhbr@gmail.com
    Copyright 2016 GNSS Sensor Ltd. All right reserved.
*************************************************************
[boardsim]: [1921] Access to unmapped address 00002000
riscv# exit
riscv#
```

To fix this problem do the following steps:

1. Close debugger console using `exit` command.
2. Open *config_file_name.json* file in any editor.

3. Find strings that specify these paths and correct them. Simulator uses the same images as VHDL platform for ROMs intialization. You can find them in *'rocket_soc/fw_images'* directory. After that you should see something like follow:



```
<serialconsole> # RISC-V: Rocket-Chip demonstration design
<serialconsole> # HW version: 0x20151217
<serialconsole> # FW id: 20160329
<serialconsole> # Target technology: inferred
<serialconsole> # AXI4: slv0: GNSS Sensor Ltd.    Boot ROM
<serialconsole> #    0x00000000...0x00001FFF, size = 8 KB
<serialconsole> # AXI4: slv1: GNSS Sensor Ltd.    FW Image ROM
<serialconsole> #    0x00100000...0x0013FFFF, size = 256 KB
<serialconsole> # AXI4: slv2: GNSS Sensor Ltd.    Internal SRAM
<serialconsole> #    0x10000000...0x1007FFFF, size = 512 KB
<serialconsole> # AXI4: slv3: GNSS Sensor Ltd.    Generic UART
<serialconsole> #    0x80001000...0x80001FFF, size = 4 KB
<serialconsole> # AXI4: slv4: GNSS Sensor Ltd.    Generic GPIO
<serialconsole> #    0x80000000...0x80000FFF, size = 4 KB
<serialconsole> # AXI4: slv5: GNSS Sensor Ltd.    Interrupt Controller
<serialconsole> #    0x80002000...0x80002FFF, size = 4 KB
<serialconsole> # AXI4: slv6: GNSS Sensor Ltd.    GNSS Engine stub
[gpio0]: LED = 01
<serialconsole> #    0x80003000...0x80003FFF, size = 4 KB
<serialconsole> # AXI4: slv7: Empty slot
<serialconsole> # AXI4: slv8: Empty slot
<serialconsole> # AXI4: slv9: Empty slot
<serialconsole> # AXI4: slv10: Empty slot
<serialconsole> # AXI4: slv11: GNSS Sensor Ltd.    Plug'n'Play support
<serialconsole> #    0xFFFFF000...0xFFFFFFFF, size = 4 KB
riscv# read 0xfffff004 128
[00000000fffff000]:  00 00 00 00 ff 00 0c 00 20 16 03 29 .. .. .. ..
[00000000fffff010]:  00 00 00 00 10 00 5c 00 00 00 00 00 00 1d a5 26
[00000000fffff020]:  00 00 00 00 00 00 55 77 00 00 00 00 00 00 00 20
[00000000fffff030]:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[00000000fffff040]:  00 00 00 10 00 f1 00 71 00 00 00 00 ff ff e0 00
[00000000fffff050]:  00 00 00 10 00 f1 00 72 00 10 00 00 ff fc 00 00
[00000000fffff060]:  00 00 00 10 00 f1 00 73 10 00 00 00 ff f8 00 00
[00000000fffff070]:  00 00 00 10 00 f1 00 7a 80 00 10 00 ff ff f0 00
[00000000fffff080]:  .. .. .. .. .. .. .. .. .. .. .. .. ff ff f0 00
riscv# _
```

Debug your target. All commands that are available for Real Hardware absolutely valid for the Simulation. Users shouldn't see any difference between these targets this is our purpose.

6.4.2 Can't open COM3 when FPGA is used

1. Open *fpga_gui.json*
2. Change value **['ComPortName','COM3'],** on your one (for an example on `ttyUSB0`).

6.4.3 EDCL: No response. Break read transaction

This error means that host cannot locate board with specified IP address. Before you continue pass through the following checklist:

1. You should properly setup network connection and see FPGA board in ARP-table.

2. If you've changed default FPGA IP address:
   (a) Open *_run_fpga_gui.bat (∗.sh)*
   (b) Change value **['BoardIP','192.168.0.51']** on your one.
3. Run debugger

6.5 Core API methods

6.5.1 Detailed Description

Core methods that allow create, modify and delete base library objects such as: Attributes, Classes, Services and Interfaces

6.5.2 Function Documentation

6.5.2.1 RISCV_break_simulation()

```
void debugger::RISCV_break_simulation ( )
```

Break all threads that could be run by different services.

This method gracefully stops all threads and allows to avoid simulation hanging on library closing stage.

6.5.2.2 RISCV_cleanup()

```
void debugger::RISCV_cleanup ( )
```

Destroy and cleanup all dynamically allocated objects.

This method allows gracefully close library by stopping all running threads and free allocated resources.

6.5.2.3 RISCV_create_service()

```
IFace* debugger::RISCV_create_service (
             IFace * iclass,
             const char * name,
             AttributeType * args )
```

Create service of the specified class.

This method creates intstance of Service and assignes all registered attributes to its initial values.

6.5.2.4 RISCV_get_class()

```
IFace* debugger::RISCV_get_class (
             const char * name )
```

Get registred class interface by its name.

This method generally used to create instances of a specific service.

6.5.2.5 RISCV_get_clock_services()

```
void debugger::RISCV_get_clock_services (
             AttributeType * list )
```

Get list of all clock generators.

Clock generator must implement IClock (and usually IThread) interfaces. CPU is a most general clock generator.

6.5.2.6 RISCV_get_configuration()

```
const char* debugger::RISCV_get_configuration ( )
```

Read library configuration.

This method allows serialize library state and save configuration into the file in JSON format. Afterward configuration can be restored.

6.5.2.7 RISCV_get_global_settings()

```
const AttributeType* debugger::RISCV_get_global_settings ( )
```

Get current core configuration.

JSON configuration string implements special section `'Global'` that contains parameters not related to any specific service or class.

6.5.2.8 RISCV_get_service()

```
IFace* debugger::RISCV_get_service (
             const char * name )
```

Get IService interface by its name.

This method is used for interaction of different services in a system.

6.5.2.9 RISCV_get_service_iface()

```
IFace* debugger::RISCV_get_service_iface (
             const char * servname,
             const char * facename )
```

Get interface of the specified ervice.

This method can be used in runtime to implement dynamic connection of different services

```
...
IUdp *iudp1 = static_cast<IUdp *>
          (RISCV_get_service_iface("udpboard", IFACE_UDP));
...
```

6.5.2.10 RISCV_get_services_with_iface()

```
void debugger::RISCV_get_services_with_iface (
             const char * iname,
             AttributeType * list )
```

Get list of services implementing specific interface.

This method can return list of services of different classes and implementing different functionality.

6.5.2.11 RISCV_init()

```
int debugger::RISCV_init ( )
```

Library initialization.

This method must be called before any other from this library.

6.5.2.12 RISCV_is_active()

```
int debugger::RISCV_is_active ( )
```

State of the core library.

Core library is active while woudln't break by RISCV_break_simulation()

6.5.2.13 RISCV_register_class()

```
void debugger::RISCV_register_class (
             IFace * icls )
```

Registration of the class in the library kernel.

Registering interface pointer will be put into kernel list of classes. Any plugin can add its own class interfaces.

Parameters

in	*icls*	Pointer on new class interface.

6.5.2.14 RISCV_register_hap()

```
void debugger::RISCV_register_hap (
             IFace * ihap )
```

Registration of the system event (hap) listener.

Haps are used to synchronized different threads by a specific events in a system. Now there's used such haps as:

- ConfigDone
- Breakpoint

6.5.2.15 RISCV_set_configuration()

```
int debugger::RISCV_set_configuration (
             AttributeType * cfg )
```

Set core library configuration.

Configuration specify all instantiated services and interconnect among them.

Parameters

in	*cfg*	Configuration attribute.

6.5.2.16 RISCV_trigger_hap()

```
void debugger::RISCV_trigger_hap (
             IFace * isrc,
             int type,
            const char * descr )
```

Trigger system event (hap) from Service.

This method allows to call all registered listeneres of a specific event from running Service.

Index