Associated with document Ref. Ares(2019)5370161

Dependable Real-time Infrastructure for Safety-critical Computer

Project number: 869945

Project acronym: De-RISC

http://www.derisc-project.eu/

D2.2	De-RISC SoC Verification Report		
Work Package	WP2	**Lead Beneficiary**	CG
Type	Report	**Dissemination level**	Public
Due Date	31/03/2021	**Version**	1.0

Brief description

This report describes the verification effort and verification results of the system-on-chip verification of the De-RISC SoC design.

Document control page

Written by

Name	Beneficiary	Date
Stefano Ribes	CG	26/03/2021
Fabio Malatesta	CG	29/03/2021
Nils-Johan Wessman	CG	29/03/2021
Jaume Abella	BSC	30/03/2021

Reviewed by

Name	Beneficiary	Date
Sergi Alcaide	BSC	31/03/2021

Approved by

Name	Beneficiary	Date
Jan Andersson	CG	31/03/2021

Change log

Version	Date	Name	Beneficiary	Comments
1.0	31/03/2021	Jan Andersson	CG	First public release

Disclaimer

The De-RISC Consortium comprises the following partners:

#	Partner legal name	Short name	Acronym	Country
1	FENT INNOVATIVE SOFTWARE SOLUTIONS SL	fentISS	FEN	Spain
2	BARCELONA SUPERCOMPUTING CENTER - CENTRO NACIONAL DE SUPERCOMPUTACIÓN	BSC	BSC	Spain
3	THALES SA	THALES	TRT	France
4	COBHAM GAISLER AB	COBHAM GAISLER	CG	Sweden

Table of contents

Index of tables

Index of illustrations

1. Introduction

This document describes the verification effort undertaken for the De-RISC Soc Design. The verification objective from the work within the De-RISC project can be split into two main categories:

- **Functional System-on-chip Verification:** The De-RISC SoC designs builds on extensive re-use of IP core building blocks from Cobham Gaisler's GRLIB IP library. The verification effort has focused on integration tests, verification of new functionality that has been implemented following identified needs during the project's platform requirements definition work, and verifying that functional properties of modified building blocks have not unintentionally been altered during the work to add new functionality.

  The functional SoC verification work has employed standard approaches, RTL simulations and FPGA emulation.

- **Non-functional System-on-chip Features Verification:** Non-functional features considered in this report include those related to timing and to security. Oppositely to functional features, which can generally be verified explicitly, non-functional features are better suited for validation activities where test campaigns based on stress workloads provide evidence of a lack of failures to meet specific non-functional features. This relates, in part, to the fact that non-functional behavior emanates mostly from the integration and allowed use of the different parts of the platform, rather than from the individual parts themselves.

  In the verification side, limited activities to verify features related to real-time support and security concerns could be performed. They are described next. Note, however, that features verified are regarded as sufficient to meet real-time and security constraints on top of them, but this does not preclude that real-time and secure execution could not be achieved without those features. Hence, those features must be understood as sufficient but not necessary.

The document is structured following these categories where functional and non-functional feature verification are presented under separate headings in this report.

2. Applicable and reference documents

2.1. Applicable documents

Reference	Document information
[DoA2019]	De-RISC Consortium: GRANT AGREEMENT NUMBER 869945 — De-RISC, 2019.

2.2. Reference documents

Reference	Document information
[GRIP2021]	Cobham Gaisler AB: GRLIB IP Core User’s Manual, 2021.

3. Terms, definitions and acronyms

3.1. Terms and definitions

Term	Definition
Description of Action	Also known as “Technical Annex” or “Annex I to the Grant Agreement.” It is an annex to the contract (aka “Grant Agreement”) signed between the Project Consortium and the European Commission describing the technical content of the work to be carried out by the beneficiaries of the funding under the European Union's Horizon 2020 Programme.
Horizon 2020	It is the biggest EU Research and Innovation Programme ever with nearly €80 billion of funding available over 7 years (2014 to 2020) – in addition to the private investment that this money will attract. The purpose of Horizon 2020 is to foster the growth of breakthrough technologies, inventions and advanced developments by the promotion of scientific ideas from the laboratories to the market.
RISC-V	Pronounced "risk-five". It is an open-source hardware instruction set architecture based on established reduced instruction set computer principles.

3.2. Acronyms

Acronym	Definition
AppSW	Application Software
BSC	Barcelona Supercomputing Center
CG	Cobham Gaisler
CPU	Central Processing Unit
CSR	Control Status Register
DoA	Description of Action
DUT	Device Under Test
EC	European Commission
EU	European Union
ESA	European Space Agency
FDIR	Failure Detection, Isolation and Recovery
FEN	fentISS
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GA	Grant Agreement
GPP	General Purpose Processor

H2020	Horizon 2020
HM	Health Monitor
HW	Hardware
IRQ	Interrupt Request
ISA	Instruction Set Architecture
LTCF	LithOS Configuration File
MAF	Major Frame
MCU	Micro-Controller Unit
MPSoC	Multi-Processor (Multi-Core) SoC
NoC	Networks-on-Chip
PCB	Printed Circuit Board
PLIC	Platform-Level Interrupt Controller Specification
PUS	Packet Utilization Standard
PVEE	Partition virtual execution environment
RISC	Reduced Instruction Set Computer
RISC-V	Reduced Instruction Set Computer Five
RTOS	Real Time Operating System
SoC	System-on-Chip
SW	Software
TC	Telecommand
TM	Telemetry
TRL	Technology Readiness Levels
TRT	Thales Research & Technology
WP	Work Package
WCET	Worst-Ccase Execution Time
XCF	XNG Configuration File
XML	Extensible Markup Language
XNG	XtratuM NextGeneration

4. Functional SoC Verification Report

4.1. Verification Objective

The verification objective in this project is to verify integration of re-used building blocks, verify extensions made to existing building blocks.

Verification of the De-RISC SoC design is performed in several steps:

1. **RTL simulation:** to verify the VHDL implementation of all cores in the system. The RTL simulation step consists of two main parts
    1. Top-level simulation where the complete De-RISC SoC is tested with the help of test software running on one or several of the processors. The system is also tested by generating stimuli in the top-level test bench.
    2. Standalone testbenches. The IP cores used in the De-RISC SoC design have standalone IP core test benches. These test benches are run as part of normal work and have, to the extent that it is supported by the test benches, been run with configuration (VHDL generics) settings set to the same values as in the De-RISC SoC design.
2. **FPGA prototyping:** FPGA prototyping tests several, possibly downsized, configuration permutations of the De-RISC SoC design. This step verifies both the software development part and the hardware development part. This step entails:
    1. Running different operating systems and accompanying drivers on FPGA prototypes
    2. Performance evaluation using benchmark software
    3. Sanity checks using XNG software provided by project partner FentISS
3. **Verification of NOEL-V processor extensions:** Verification of the NOEL-V processor covers both RTL simulation and FPGA prototyping. It is described as a separate step in this report.

4.2. Verification Environment

The following tools have been used during the verification:

- VHDL simulator Siemens/Mentor Questa Prime, version 10.6 and later
- VHDL simulator Aldec Riviera-PRO, version 2018.02 and later
- Xilinx Vivado FPGA toolchain, 2018.3
- GRMON3 debug monitor
- VxWorks 7 SR0650 operating system
- NOEL-V buildroot 2020.08 distribution with Linux kernel and OpenSBI
- NOEL-V RTEMS5 Software Development Environment
- NCC C/C++ bare-metal toolchain

The boards used for FPGA prototyping are:

- Digilent Art-A7 T100
- Microsemi PolarFire Splash kit
- Xilinx KCU105
- Xilinx VC707
- Xilinx VCU118

The Xilinx KCU105 development kit has a Xilinx Kintex Ultrascale FPGA, which is the same product line of FPGAs that will be used on the De-RISC board.

4.3. Verification Report – RTL simulation

4.3.1. Strategy

As described in section 4.1, the RTL simulation step consists of the following parts:

1. Top-level simulation where the complete De-RISC SoC is tested with the help of test software running on one or several of the processors. The system is also tested by generating stimuli in the top-level test bench.
2. Standalone testbenches. The IP cores used in the De-RISC SoC design have standalone OP core test benches. These test benches are run as part of normal work and have, to the extent that it is supported by the test benches, been run with configuration (VHDL generics) settings set to the same values as in the De-RISC SoC design. Individual test cases for the standalone test benches have not been included in this report.

Section 4.3.3.1 describes the test software used when simulating the full De-RISC design.

The standalone VHDL test benches provide more in-depth coverage of individual cores and are not described in this report for cores re-used from the GRLIB IP library.

4.3.2. Limitations

The system test software covers the top-level design and verifies basic functionality of each core. It is not feasible to boot large programs etc. by means of VHDL simulation due to prohibitively long simulation times. In general, it is not feasible to simulate dataflow, bandwidth and throughput aspects for the same reason. These aspects will instead be covered during FPGA prototyping.

4.3.3. Top-level simulations

4.3.3.1. GRLIB system test

The GRLIB VHDL IP Library [GRIP2021] contains system test software that is run on one or several of the processors in simulation. The test software is written in C and compiled with NCC. The simulated processor cores execute binary code from simulated external memory. The tests performed cover the proper interfacing of IP cores, ensure that the cores' data and configuration registers are accessible, and perform elementary short transactions with the interface IP cores.

Parts of the system, such as the bus fabric, arbiters, and bridges, are implicitly verified by the system test software.

4.3.3.1.1. 8-bit UART (APBUART) test

Software:	software/noelv/systest/apbuart.c
Function call:	apbuart_test()
Requirements:	None
Description:	The APBUART test performs the following operations: 1. Initialize the scaler to 1 to speed up the test 2. Initialize the receiver holding registers to speed up simulation 3. Test transmitter and transmit status bits 4. Test receiver and receive status bits in loop back mode 5. Disable the UART core.
Result:	Test is self-checking

4.3.3.1.2. General purpose timer unit (GPTIMER) test

Software:	software/leon3/gptimer.c
Function call:	gptimer_test(0xf0008000, 1);
	gptimer_test(0xf0009000, 6);
	gptimer_test(0xf000a000, 7);
	gptimer_test(0xf000b000, 8);
	gptimer_test(0xf000c000, 9);

Requirements:	None
Description:	The GPTIMER test performs the following operations: • Enables timer and test that scaler starts counting • Tests basic timer functions for each timer, including assertion and clearing of interrupt pending status bit • Checks timer chaining function • IRQ assertion.
Result:	Test is self-checking

4.3.3.1.3. Gigabit Ethernet MAC (GRETH_GBIT) test

Software:	software/noelv/systest/greth.c
Function call:	int greth_test(int apbaddr); int apbaddr shall be set to the base APB address of the core. The return value is always 0 which indicates a successful run. The test might stop earlier if an error is detected. The test is stopped using the fail() function which is documented elsewhere.
Requirements:	The core inputs and outputs should be connected to a PHY simulation model. The PHY model is provided in the sim package in GRLIB.
Description:	First the greth_init routine is called which initializes the PHY and core. Then a 256 B packet is transmitted from the tx dma channel which is received by the receiver since the PHY model maps the transmit output signals to the receiver. It is checked that the packet is received correctly (length and received data is checked). If the EDCL is detected to be present in the core a sequence of EDCL packets are then transmitted. First an ARP packet to acquire the MAC address of the EDCL, a dummy zero length read to acquire the current sequence number, a 72 B write to memory and lastly a 72 B read to see that the written data can be read back correctly. All the the above is done using the transmitter and receiver dma channels.
Result:	The test is self-checking. If the test returns 0 it has been correctly executed.

4.3.3.1.4. L2 cache (L2C) test

Software:	software/noelv/systest/l2c.c
Function call:	l2c_enable(unsigned int regaddr, unsigned int ften)

Description:	If the L2 cache is disabled the function invalidates all cache lines and enabled the cache.
Result:	The test performs no checks. The L2 cache is implicitly verified by the other tests as they are run from main memory that is behind the L2 cache.

4.3.3.1.5. AHB/AHB bridge with protection functionality (GRIOMMU) test

Software:	software/noelv/systest/griommu.c
Function call:	griommu_test();
Description:	The test performs the following operations on the GRIOMMU core: • Basic tests (read and write operations) on core's register interface • Cache flush operation • Diagnostic cache accesses • Fault-tolerance
Result:	Test is self-checking

4.3.3.1.5.1.1. Result

All tests pass.

4.3.3.1.6. SPI memory controller (SPIMCTRL) test

Software:	software/noelv/systest/spimctrl.c
Function call:	spimctrl_test();
Description:	The test shifts out 0x5A and verifies that the received data is 0xFF. In simulation, the SPI_DI port should be connected to the SPI_DO (i.e. in loopback).
Result:	Test is self-checking

4.3.3.1.6.1.1. Result

All tests pass.

4.3.3.1.7. SPI controller (SPICTRL) test

Software:	software/noelv/systest/spictrl.c

Function call:	spictrl_test();
Description:	Performs an internal loopback test and communicates with a spi_flash simulation module. A model should be instantiated in the testbench with the following parameters: ```vhdl spi_flash ftype => 3, debug => Don't care, fname => Don't care, readcmd => 16#0B#, dummybyte => 0, dualoutput => 0 ``` The test requires that generic `slvselen` is 1 and that slave select 0 is connected to the spi_flash model. It also requires that the SPISEL input is HIGH.
Result:	Test is self-checking

4.3.3.1.7.1.1. Result

All tests pass.

4.3.3.2. Bootstrap verification

The SoC allows the selection of the boot memory interface via bootstrap signals. The verification of such feature has been achieved by the inspection of the results of RTL test benches when varying the value of the bootstrap inputs at reset.

4.3.4. Standalone test benches

4.3.4.1. Overview

The IP cores included in the De-RISC have stand-alone VHDL test benches. The subsections below describe the tests performed by standalone test benches that have been applied within the project due to updates being made to the corresponding IP.

In addition to this, standalone test benches exist for the other IP cores applied in the De-RISC design and these are used as part of normal work with the IP library.

Core	Section	Comment
AHB2AHB	4.3.4.2	Uni-directional AHB/AHB bridge
L2C	4.3.4.3	Level-2 cache
FTADDR	Error: Reference source not found	Fault-tolerant DDR2/3 SDRAM controller
GRIOMMU	4.3.4.4	IO MMU
GRSPWROUTER	4.3.4.5	SpaceWire router
GRETH_GBIT	4.3.4.6	10/100/1000 Mbit Ethernet MAC

Table 1: Standalone testbenches

4.3.4.2. Uni-directional AHB/AHB bridge (AHB2AHB)

4.3.4.2.1. Overview

The standalone VHDL test bench for the AHB2AHB core instantiates two AHB buses connected with one AHB2AHB in each direction (the test bench has a total of two AHB2AHB cores instantiated). Each bus has ten AHB test masters and one AHB test slave that are controlled from the main test bench process.

The test bench performs the following operations:

- Single accesses
- Burst accesses
- Trigger deadlock detection (not applicable for De-RISC since the bridge is not used in a bi-directional configuration)
- Locked accesses
- Instruction bursts
- Interrupt forwarding
- Bursts longer than the configured maximum burst length
- Test of pre-fetch logic
- Test that pre-fetch logic respects AMBA 1k boundary
- SPLIT/RETRY responses
- Test bridge bus release after locked access
- Test pre-fetch abort
- Test that core respects burst boundary
- Test that interfaces can be disabled
- Test for starvation scenarios
- Test saved responses
- Test first-come, first-served ordering (FCFS)
- Test back-to-back transfers with FCFS
- Test pre-fetch operation with FCFS
- Test collision in bidirectional configuration (not applicable for NGMP, the bridge is not used in a bi-directional configuration)

- Test back-to-back writes
- Test long write bursts
- Test read combining and interrupted accesses
- Test early burst termination
- Test read-modify-write
- Simulate polling scenario
- Simulate lock spin
- Simulate additional polling scenario

The tests performing single and burst accesses perform accesses of widths up to and including the maximum bus width supported by the bridge.

4.3.4.2.2. Verification matrix

The verification matrix for the core is provided in the table below.

Test case	Description
singleaccdown32	32-bit single accesses HSB → LSB
singleaccdown64	64-bit single accesses HSB → LSB
singleaccdown128	128-bit single accesses HSB → LSB
singleaccup32	32-bit single accesses LSB → HSB
singleaccup64	64-bit single accesses LSB → HSB
singleaccup128	128-bit single accesses LSB → HSB
singleaccboth32	Concurrent 32-bit single accesses in both directions, 32-bit
singleaccboth64	Concurrent 64-bit single accesses in both directions, 64-bit
singleaccboth128	Concurrent 128-bit single accesses in both directions, 128-bit
seqburstaccdown32	32-bit sequential burst accesses HSB → LSB
seqburstaccdown64	64-bit sequential burst accesses HSB → LSB
seqburstaccdown128	128-bit sequential burst accesses HSB → LSB
seqburstaccup32	32-bit sequential burst accesses LSB → HSB
seqburstaccup64	64-bit sequential burst accesses LSB → HSB
seqburstaccup128	128-bit sequential burst accesses LSB → HSB
seqburstaccboth32	Concurrent 32-bit sequential burst accesses in both directions, 32-bit
seqburstaccboth64	Concurrent 64-bit sequential burst accesses in both directions, 64-bit
seqburstaccboth128	Concurrent 128-bit sequential burst accesses in both directions, 128-bit
locked	Perform locked accesses when a SPLIT response has been issued for an earlier access
instburst	Test instruction bursts
irqforward	Test interrupt forwarding
rburst	Test bursts longer than value of rburst VHDL generic
prefetchlogic	Tests behaviour in pfread state. Reads out less than whole FIFO and then performs a transfer, locked transfer etc. Also checks ERROR responses during prefetch.
prefetch1kboundary	Tests that prefetch logic respects AMBA 1k address boundary limitation
splitretry	Tests AMBA SPLIT/RETRY responses
busrelease	Tests that bridge releases bus after performing locked access

Test case	Description
pfabort	Tests that prefetch operation can be aborted due to incoming locked access
boundbrst	Tests that core respects burst boundary
ifctrl	Test that interfaces can be disabled
starve	Test for starvation issue
saved	Test saved responses
fcfs	Test First-come, first-served
fcfs2	Test Back-to-back transfers with FCFS
pffcfs	Test PF operation with FCFS
sp2brel	Test collision in bidirectional configuration
b2bwrites	Test back2back writes
wrburst	Test long write bursts
rdcabort	Test read combining and interrupted accesses
ebterm	Test early burst termination
rmw	Test read-modify-write
poll	Simulate polling scenario
poll2	Simulate lock spin
poll3	Yet another polling scenario

4.3.4.2.3. Results

All tests pass.

4.3.4.3. L2 cache (L2C)

4.3.4.3.1. Overview

The standalone test bench for the Level 2 cache performs the following operations:

- Different combinations of read and write accesses of differing sizes
- Extended test with read operations triggering cache hits and misses
- Extended test with write operations triggering cache hits and misses
- Tests of diagnostic accesses
- Tests of cache flush functionality
- Tests of cache HPROT functionality
- Tests of write-through functionality
- Tests of LRU replacement policy
- Tests of random replacement policy

4.3.4.3.2. Verification matrix

The verification matrix for the core is provided in the table below.

Test case	Description
access_combo	Different combinations of read and write accesses of differing sizes
read_test	Extended test with read operations triggering cache hits and misses
dia_test	Extended test with write operations triggering cache hits and misses
flush_test	Tests of diagnostic accesses
hprot_test	Tests of cache flush functionality
write_through_test	Tests of cache HPROT functionality
lru_test	Tests of write-through functionality
random_test	Tests of random replacement policy

4.3.4.3.3. Results

All tests pass.

4.3.4.4. IO Memory Management Unit (GRIOMMU)

4.3.4.4.1. Overview

The standalone VHDL test bench for the GRIOMMU core instantiates two AHB buses connected with one GRIOMMU in each direction (the test bench has a total of two GRIOMMU cores instantiated). Each bus has ten AHB test masters and one AHB test slave that are controlled from the test bench process.

The test bench performs the following operations:

- Register interface test
- Access Protection Vector test
- Access Protection Vector cache test
- Interrupt generation and Status and Failing access registers with APV
- Test of statistics outputs
- Fault-tolerance of APV cache
- Loopback test
- Group-set addressing test
- Race conditions from bridge's internal denied state
- Test of general IOMMU functionality
- IOMMU ITR test
- IOMMU TLB test
- All of the tests below are then tested with the following settings: protection disabled, APV protection enabled, APV protection with APV cache enabled, IOMMU protection, IOMMU protection with TLB enabled:
  - Single accesses
  - Burst accesses
  - Trigger deadlock detection (not applicable for De-RISC SoC since the bridge is not used in a bi-directional configuration)
  - Locked accesses
  - Instruction bursts
  - Interrupt forwarding

- Bursts longer than the configured maximum burst length
- Test of pre-fetch logic
- Test that pre-fetch logic respects AMBA 1k boundary
- SPLIT/RETRY responses
- Test bridge bus release after locked access
- Test pre-fetch abort
- Test that core respects burst boundary
- Test that interfaces can be disabled
- Test for starvation scenarios
- Test saved responses
- Test first-come, first-served ordering (FCFS)
- Test back-to-back transfers with FCFS
- Test pre-fetch operation with FCFS
- Test collision in bidirectional configuration (not applicable for NGMP since the bridge is not used in a bi-directional configuration)
- Test back-to-back writes
- Test long write bursts
- Test read combining and interrupted accesses
- Test early burst termination
- Test read-modify-write

If the instantiated GRIOMMU cores have been configured with dynamic support for AMBA SPLIT, the majority of the tests are run first with SPLIT support disabled and then with SPLIT support enabled. The tests performing single and burst accesses perform accesses of widths up to and including the maximum bus width supported by the bridge.

4.3.4.4.2. Verification matrix

The verification matrix for the core is provided in the table below.

Test case	Description
reg	Register test
apv	Access Protection Vector test
apvc	Access Protection Vector cache test
apvistat	Interrupt generation and Status and Failing access registers with APV
stat	Statistics
apvcft	Fault tolerance of APV cache
loopback	Loopback test
apvgseta	Group-set addressing
deniedrace	Denied state race conditions
iommu	IOMMU test
iommuitr	IOMMU ITR test
iommutlb	IOMMU TLB test
The remaining tests are run: • Without any protection (IOMMU/APV functionality disabled) • With APV protection • With APV protection and APV cache enabled • With IOMMU protection • With IOMMU protection and TLB enabled	
singleaccdown32	32-bit single accesses HSB → LSB
singleaccdown64	64-bit single accesses HSB → LSB
singleaccdown128	128-bit single accesses HSB → LSB
singleaccup32	32-bit single accesses LSB → HSB
singleaccup64	64-bit single accesses LSB → HSB
singleaccup128	128-bit single accesses LSB → HSB
singleaccboth32	Concurrent 32-bit single accesses in both directions, 32-bit
singleaccboth64	Concurrent 64-bit single accesses in both directions, 64-bit
singleaccboth128	Concurrent 128-bit single accesses in both directions, 128-bit
seqburstaccdown32	32-bit sequential burst accesses HSB → LSB

Test case	Description
seqburstaccdown64	64-bit sequential burst accesses HSB → LSB
seqburstaccdown128	128-bit sequential burst accesses HSB → LSB
seqburstaccup32	32-bit sequential burst accesses LSB → HSB
seqburstaccup64	64-bit sequential burst accesses LSB → HSB
seqburstaccup128	128-bit sequential burst accesses LSB → HSB
seqburstaccboth32	Concurrent 32-bit sequential burst accesses in both directions, 32-bit
seqburstaccboth64	Concurrent 64-bit sequential burst accesses in both directions, 64-bit
seqburstaccboth128	Concurrent 128-bit sequential burst accesses in both directions, 128-bit
locked	Perform locked accesses when a SPLIT response has been issued for an earlier access
instburst	Test instruction bursts
irqforward	Test interrupt forwarding
rburst	Test bursts longer than value of rburst VHDL generic
prefetchlogic	Tests behaviour in pfread state. Reads out less than whole FIFO and then performs a transfer, locked transfer etc. Also checks ERROR responses during prefetch.
prefetch1kboundary	Tests that prefetch logic respects AMBA 1k address boundary limitation
splitretry	Tests AMBA SPLIT/RETRY responses
busrelease	Tests that bridge releases bus after performing locked access
pfabort	Tests that prefetch operation can be aborted due to incoming locked access
boundbrst	Tests that core respects burst boundary
ifctrl	Test that interfaces can be disabled
starve	Test for starvation issue
saved	Test saved responses
fcfs	Test First-come, first-served
fcfs2	Test Back-to-back transfers with FCFS
pffcfs	Test PF operation with FCFS
sp2brel	Test collision in bidirectional configuration
b2bwrites	Test back2back writes
wrburst	Test long write bursts
rdcabort	Test read combining and interrupted accesses
ebterm	Test early burst termination

Test case	Description
rmw	Test read-modify-write

4.3.4.4.3. Results

All tests pass.

4.3.4.5. SpaceWire router (GRSPWROUTER)

4.3.4.5.1. Overview

The standalone VHDL test bench for the GRSPWROUTER core instantiates one GRSPWROUTER core with one GRSPW2 PHY for each SpaceWire port. The number of SpaceWire, AMBA and FIFO ports is configurable. The test bench also instantiates the complete AMBA infrastructure needed to interact with the core together with AHB masters and slaves that are controlled by the test bench process.

4.3.4.5.2. Verification matrix

The verification matrix for the router is provided in the table below.

Test case	Description
treset	Resets the router core and test framework. All input signals are given deterministic values. Some are used as reset values for internal registers.
tinitlinks	Enables SpaceWire ports' codecs and sets up DMA descriptor tables for AMBA ports. The reset values for all ports' control registers are checked. One packet per link is sent to initialize some status register fields which would otherwise disturb other register reads.
tregisterresetvalues	Check reset values for all registers.
tconfigportnonrmappackets	Check that non RMAP packets sent to the configuration port are discarded correctly.
trmapunusedcommands	Check that RMAP commands of unused types are replied with the correct error code if reply is requested or discarded if no reply is requested.
trmapunauthorizedcommands	Check that RMAP commands with unauthorized settings are either replied with the correct error code if reply is requested or discarded otherwise.
trmaperrorcodeorder	Test that RMAP errors are detected in the correct order and that the error code in the reply status field is set accordingly.
tinvaddr	Test that all cases of invalid destination port addresses for incoming packets and that the invalid address bit is set for the corresponding port.
tspwtimecodeenabledwithsignal	Test the global external time-code enable signal.
tspwtimecodeenabledcontrolflagfiltering	Test control flag filtering function for time-codes.
tspwtimecodedisabledperportnofiltering	Test port specific time-code enable function with time-code filtering disabled.
tspwtimecodedisabledperportfiltering	Test port specific time-code enable function with time-code filtering enabled.
tspwtimecodedisabledglobally	Test global time-code enable function. Similar to tspwtimecodeenabledwithsignal.
tspwtimecodedisabledgloballyregister	Test global time-code enable function using configuration port register.
tzerolengthpackets	Check that zero length packets are handled correctly (EOP/EEP followed by another EOP/EEP).
tphysicalportnumberrouting	Test packet routing using path(physical addresses).
tlogicaladdressrouting	Test packet routing using logical addresses. All logical addresses are used covering the complete

Test case	Description
	routing table.
tallactiveportaddressing	Test all ports active simultaneously using path (physical) addressing.
tallactivelogicaladdressing	Test all ports active simultaneously using logical addressing.
tconfigurationportwriteread	Test RMAP write and read commands to the configuration port.
tconfigurationportrmw	Test RMAP rmw commands to the configuration port.
tconfigurationportdisable	Test port specific enable bit for accesses to the configuration port.
tconfigurationportwritedisabledaddresses	Test the write enable register for the configuration port.
tselfaddrenphysicalsingle	Test self addressing enable bit using path (physical) addresses without group adaptive routing or packet distribution.
tselfaddrenphysicalgroup	Test self addressing enable bit using path (physical) addresses with group adaptive routing.
tselfaddrenpacketdistributionphysical	Test self addressing enable bit using path (physical) addresses with packet distribution.
tselfaddrenlogicalsingle	Test self addressing enable bit using logical addresses without group adaptive routing or packet distribution.
tselfaddrenlogicalgroup	Test self addressing enable bit using logical addresses with group adaptive routing.
tselfaddrenpacketdistributionlogical	Test self addressing enable bit using logical addresses with packet distribution.
tconfigurationportlocksignal	Test configuration port lock signal.
tgroupadaptivephysicalbusy	Test group adaptive routing with busy ports (packet transfer active on some ports) using path (physical) addresses.
tgroupadaptivephysicaldisabled	Test group adaptive routing with disabled ports using path (physical) addresses.
tgroupadaptivelogicaldisabled	Test group adaptive routing with disabled ports using logical addresses
tpriority	Test that ports are allowed accesses in a round-robin order.
tpacketdistributionphysical	Test packet distribution using path (physical) addresses.
tpacketdistributionlogical	Test packet distribution using logical addresses.
tpacketdistributionphysicaldifferentspeed	Test packet distribution using path (physical) addresses and different link speed on the ports.
tpacketdistributionphysicaldifferentspeedblocked	Test packet distribution using path (physical) addresses and different link speed on the ports with some ports blocked by other transfers.
tpacketdistributionmultiple	Test multiple packet distribution transfers active simultaneously.
ttimer	Test the port timers for single transfers, group adaptive routing transfers and packet distribution.
tthroughput	Check throughput with all ports active using large packets.
tthroughputsmallpackets	Check throughput with all ports active using small packets.
tambaportrmaptarget	Test basic read write commands to the RMAP targets in the AMBA ports.

Test case	Description
tambaportburstlength	Check burst lengths for AMBA ports.
tahbslaveambatests	Test different AMBA AHB accesses to the AHB slave interface to the configuration port.
tsimultaneousahbslavermapandpackets	Test simultaneous packet transfers, RMAP accesses to configuration port and AHB slave interface accesses to the configuration port.
tahbslaveinterface	Check accesses to all registers in the configuration port through the AMBA AHB slave interface.

In addition to the GRSPWROUTER test bench, the SpaceWire codec used within the router has a separate standalone testbench. The verification matrix for the codec is provided below.

Test case	Description
treset	Reset the core and test environment and set input signals to deterministic values.
tlinkstartup	Check that the SpaceWire link startup mechanism is followed according to the standard.
tautostart	Check the SpaceWire autostart feature.
tinitclockdivision	Test clock divisor settings for initialization.
trunclockdivision	Test clock divisor settings for run state.
terrorsinerrorwait	Test error conditions in the error-wait state.
terrorsinready	Test error conditions in the ready state.
terrorsinstarted	Test error conditions in the started state.
terrorsinconnecting	Test error conditions in the connecting state.
terrorsinrun	Test error conditions in the run state.
treception	Test reception of packets.
ttransmission	Test transmission of packets.
ttimereception	Test time-code reception.
ttimetransmission	Test time-code transmission.
tcredits	Test credit counter handling
teeprunstatelinkerror	Check that EEP is inserted when link error occurs in run-state.
tparityerror	Test parity errors.
tcrediterror	Test credits errors.
tescapeerror	Test escape errors.
tdisconnecterror	Test disconnect errors.
ttxflush	Test transmitter flush feature.

Test case	Description
tdualportreception	Test reception with dual ports enabled.

4.3.4.5.3. Results

All tests pass.

4.3.4.6. 10/100/1000 Mbit Ethernet MAC (GRETH_GBIT)

4.3.4.6.1. Overview

The standalone VHDL test bench for the GRETH_GBIT core instantiates AMBA infrastructure that connects the GRETH_GBIT core to a AHB slave memory and to two AHB test masters. The GRETH_BIT is connected to a simulation model of an Ethernet transceiver. The test bench tests verifies core operation in 10/100/1000 Mbit full-duplex and 1000 Mbit half-duplex mode.

4.3.4.6.2. Verification matrix

The verification matrix for the Ethernet MAC is provided in the table below.

Test case	Description
tInitial	Default power up for APB registers values
tRegister	APB register access tests
tSoftWareReset	Test Reset ctrl bit
tSwInitial	Default power up values for APB registers.
tRegister	APB register access tests
tMulticast	Test multicast mode.
tEdclAddr	Test that EDCL addr can be changed using the edcladdr input and the APB register
tBasicReceive	Basic Receive test, Packets with length 0 - 1500.
tBasicTransmit	Basic Transmit test. Packets with length 0 – 1514 including address, type and crc fields.
tUnalignedTransmit	Transmit test with unaligned AHB buffer addresses.
tStart100	Start 100 Mbit Mode. Previous tests have been run in 10 Mbit mode.
tBasicReceive	Basic Receive test, Packets with length 0 - 1500.
tBasicReceiveUnaligned	Basic Receive test, Packets with length 0 to 40 stored to buffers with different alignments
tBasicTransmit	Basic Transmit test. Packets with length 0 – 1514 including address, type and crc fields.
tUnalignedTransmit	Transmit test with unaligned AHB buffer addresses.
tScatterGatherTransmit	Scatter Gather Transmit test
tDribbleNibble	Dribble Nibble Error receive (packet which is not an integer number of octets in length).
tEdclReceiveWithErrors	Edcl Receive test with errors
tStart1000	Start 1000 Mbit
tLengthError	Test Length Error detection
tSpecialCases3	Jumbo Frame test (packets larger than maximum size).

Test case	Description
tSpecialCases2	ICMP test 8192 B
tInterrupts	Test transmitter interrupts. Test that interrupt is not generated when TI and IE are clear
tSpecialCases	Cover special cases (packet types normally seen in typical network traffic).
tReceiverChecksumOffloadIpFragment	Test receiver checksum offloading for IP fragments
tReceiverChecksumOffload	Test receiver checksum offloading
tScatterGatherTransmitWithChecksumOffload	Scatter Gather Transmit test with checksum offload
tRetrySplitWaitStates	Split, Retry and waitstates on AHB bus
tTransmitChecksumOffloadFragmentedPackets	Transmit test with checksum offload for fragmented packets
tDmaTransmitWithAHBErrors	Transmit test with AHB errors
tEdclTransmitterAHBError	Edcl test with AHB errors
tTransmitMaximumSizeScatterGather	Test transmission of packets larger than maximum size for Scatter Gather transmissions
tTransmitMaximumSize	Test transmission of packets larger than maximum size
tBasicReceive	Basic Receive test, Packets with length 0 - 1500.
tBasicReceiveUnaligned	Basic Receive test, Packets with length 0 to 40 stored to buffers with different alignments
tBasicTransmit	Basic Transmit test
tUnalignedTransmit	Unaligned Transmit test
tScatterGatherTransmit	Scatter Gather Transmit test
tLengthError2	Test Length Error detection 2
tReceiverOverrun	Receiver Overrun (RAM buffer full).
tEdclReceiveSpeedTest	Edcl Receive Speed test
tDmaReceiveWithAHBErrors	Receive test with AHB error
tIntermixedPacket	Test ARP, IP, normal (Ethernet type field indicates length)and junk packets intermixed
tEdclReceiveWithErrors	Edcl Receive test with errors
tGeneralArp	Test ARP with different destination addresses
tEdclIPRegister	Edcl Receive test with changed IP
tPromiscousMode	Test Promiscous Mode for DMA Channel
tCtrlPacketReceive	Ctrl Packet Receive test
tReceiveMaximumSize	Test reception of packets larger than maximum size

Test case	Description
tCRCErrorReceive	CRC Error receive
tEdclArp	Edcl ARP test
tEdclReceive	Edcl Receive test
tStart1000HalfDuplex	Start 1000 Mbit Mode HalfDuplex
tTransmitBurstLimit	Test transmit burst counter
tTransmitBurst	Burst Transmissions in halfduplex
tEdclTransmissionWithCollission2	Send several edcl packets which are put in queue. Check how collissions are handled in this case
tEdclTransmissionWithCollission	Test Edcl Transmissions with collisions
tTransmissionWithLateCollission	Test Transmissions with late collisions
tEdclRetrySplit	Split, Retry on AHB bus accesses for Edcl transmitter
tTransmissionWithCollission	Test Transmissions with collisions
tTransmissionAttemptLimit	Test Transmission attempt limit
tReceiveMaximumSize	Test reception of packets larger than maximum size
tBasicReceiveHalfDuplex	Special receive testcases for 1000 Mbit Halfduplex
tBasicReceive	Basic Receive test, Packets with length 0 - 1500.
tBasicReceiveUnaligned	Basic Receive test, Packets with length 0 to 40 stored to buffers with different alignments
tCheckMdioError	Check that no illegal Mdio accesses have been made during the whole test suite.
tCheckJamSizeError	Check that no JamSize errors have been detected during the whole test suite.

4.3.4.6.3. Results

All tests pass.

4.4. Verification Report – FPGA prototyping

4.4.1. Strategy

The FPGA board used for this effort is an off-the-shelf products from Xilinx.

A goal of FPGA prototyping, in contrast to the verification, is to look at the design in a user perspective and to have the device running in as many different end user functional modes or configurations as possible. Thus trying to cover how the end user will use the device. The reason for this approach is to ensure that not only individual functional requirements are covered, but also that the device is sensible from a user perspective.

The FPGA prototyping approach is processor driven, based on the NOEL-V software environment. This facilitates re-use of existing resources and ensures that the overall objective of a processor-controlled device is achieved. The test development time is reduced since existing design concepts are re-used.

The final target for the De-RISC SoC design is to be implemented in an FPGA and this verification effort will be followed by a hardware validation effort. Because of this, what is done under the FPGA prototyping heading here is less than what would be reported for a FPGA prototyping effort in case the end target would have been an ASIC implementation.

The majority of the tests are self-checking go/no-go tests, not requiring any expected responses to be analysed. The number of test cases requiring analysis will be minimized.

4.4.2. Limitations

The tests described in this report have been run on the Xilinx KCU105 development board. This board does not have the same feature set as the planned De-RISC board that will host the full De-RISC SoC design and makes use of DDR4 SDRAM memory as the primary memory interface. The FTADDR memory controller that is the primary memory controller used for the De-RISC SoC does not support DDR4 SDRAM and instead a memory controller, without fault-tolerance features, from Xilinx has been used for the KCU105 FPGA prototyping.

4.4.3. Linux

4.4.3.1. Tools

The considered Linux images are built using the NOEL-V buildroot 2020.08 distribution with Linux kernel and OpenSBI. The GRMON hardware debugger has been used to control the core.

4.4.3.2. Procedure

The RISC-V target software needs a Flattened Device Tree (FDT) which describes the hardware system for the OS to find devices and adapt to other system settings. Therefore, A Device Tree Binary (DTB) is loaded into main memory by the GRMON hardware debugger and a pointer to the DTB is given to the OS at boot. The linux image is then loaded in memory and, once the load operation is complete, the booting process is started.

4.4.3.3. Tests

Verify that the booting process proceeds flawlessly and perform simple operations of file management in the operative system.

4.4.3.4. Results

All tests pass.

4.5. Verification Report – NOEL-V extensions

4.5.1. Strategy

The verification of a superscalar processor is a very complex task. There are countless different scenarios in which a small variation on the stimulus patterns can lead to a corner case which was not pictured by the designers. Moreover the number of necessary tests to unsure functional correctness is too large to envision even for a very large team of engineers.

Such a huge verification space is generally considered too hard to tackle using manual testing and therefore some form of automatic testing is generally employed. This is the case also for the verification of the NOEL-V processor, for which purpose a Random Instructions Generator has been developed.

The concept is simple:

- generate random sequence of Assembler instructions
- compile the sequence and run it on both the DUT and an Instruction Set Simulator
- compare the DUT behavior to the expected behavior and investigate the differences.

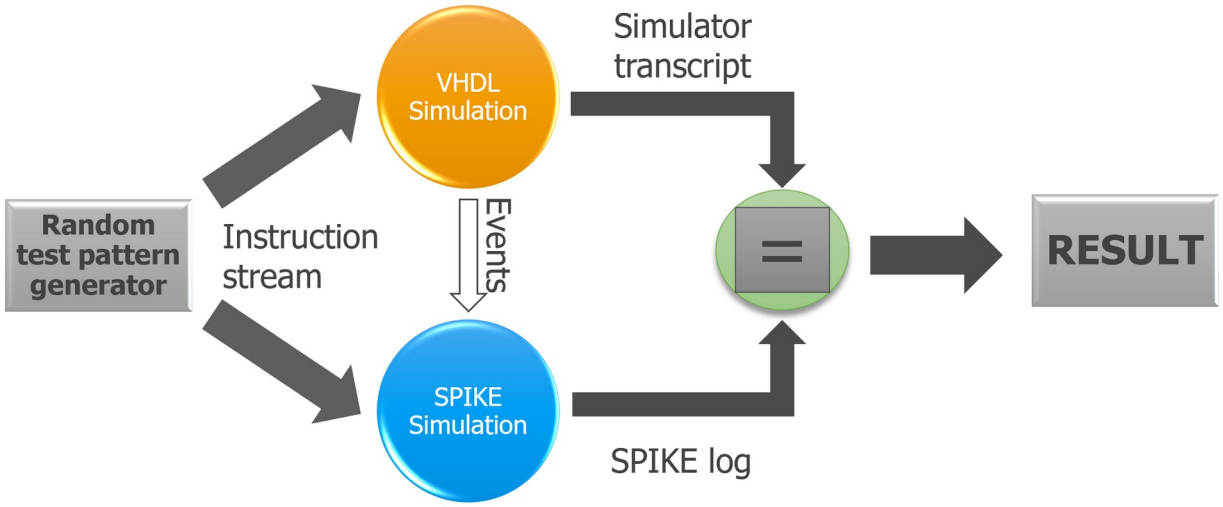

Illustration 1: Model comparison

The cycle of sequence generation, compilation, run and comparison can be automated and therefore a huge number of test cases can be run without human intervention. Any test case showing a difference between the behavior of the processor under test and the ISA simulator is stored in order to be investigated and added to a regression run.

In the generated sequence the randomness is applied to instruction ordering, program structure, privileged mode setting, exceptions. The produced software can be bare metal or also enable the MMU and produce page tables. The random sequence includes instructions belonging to all the RISC-V extensions implemented in the processor.

The comparison between the DUT behavior and the Instruction Set simulator has also been applied to verify the functional correctness of the processor during the booting process of complex operative systems such as Linux.

Directed tests that verify a specific feature are also employed and a set of these tests are also described.

4.5.2. Limitations

The random instruction sequence generator randomizes most of the aspects that define the test programs. However, some limitations are necessary in order to prevent the generation of infinite loops and of instruction sequences whose expected behavior is not strictly defined by the RISC-V standard. The adopted limitations are hereby described:

- The page tables used to access virtual memory are static and are defined before the start of the random execution.
- To simplify memory accesses and exception handling two general-purpose registers are reserved and are not overwritten during the execution of the random instruction sequences.
- Due to differences in the implemented behavior between the NOEL-V core and the adopted ISA simulator no LR/SC (Load reserved/Store conditional) sequences are generated. Note that both the implementations are compliant to the RISC-V standard.

4.5.3. Random instruction generation

4.5.3.1. Overview

The random instructions generator can be configured to target several different areas of the processor functionalities. Some examples of these areas are:

- branch prediction
- late ALU
- PMP (Physical Memory Protection)
- FPU (Floating Point Unit)
- compressed instructions

In particular, the verification of the H (hypervisor) extension is most relevant for this project. For this purpose the Random Instruction Generator has been extended to stimulate the features included in the extension, such as the virtualization mode, two stage address translation, handling of guest software exceptions and the use of the Hypervisor instructions.

The random instruction generator has also the capability to assign different probability to the generation of different types of instructions. This also allows to tailor the random generation to produce test cases which address different parts of the processor. For example, increasing the probability of load and store instructions would lead to a test case which would focus more on the cache controller behavior, while instead increasing the probability of branch instructions would produce a test case more focused on the control flow of the processor.

4.5.3.2. Test cases

Given the automatic nature of the test strategy, a countless number of test cases have been generated and run. Moreover, the random nature of the tests makes hard to pinpoint which features are tested by a single random sequence. In fact, in a single test case many different features may be stimulated. Therefore, no description of a specific test case is reported here.

4.5.3.3. Results

Whenever a discrepancy between the behavior NOEL-V and the ISA simulator is found, the difference is investigated and resolved. The test cases produced currently report no differences, as all the ones previously reported have been solved.

4.5.4. Complex software model comparison

The comparison between the behavior of the processor and the ISA simulator can also be applied to more complex software flows, such as the booting processes of operative systems. This gives the

possibility to monitor the correctness of the core while running real software and at the same time gives a level observability much higher than the one achievable by running the same software in an FPGA prototype, thereby improving the possibility to easily identify and correct potential issues. This strategy has been used simulate different operative systems, such as Linux and RTEMS. The developed test infrastructure allows also to read different device tree files and therefore simulate the behavior of the processor with different memory maps and peripherals.

4.5.5. Direct feature tests

4.5.5.1. Overview

To complement the generated random instruction test, some direct test has been constructed for specific features. This is especial important when new features has been implemented (in this case the H extension). These test can be used to verify a specific feature is implemented correctly. It can be the first step to verify the basic functionality before it is included in random generated test scenarios. This can also be useful for test cases which is unpractical or impossible to generate randomly.

4.5.5.2. Test cases

Test has been constructed for the following features

- Access control of new CSRs
- Operation mode transitions (including all privilege levels and virtualization mode)
- Functionality of specific CSR configuration bits.

4.5.5.3. Results

Most test cases has been constructed to be self checking, but also designed to run in an Instruction Set Simulator. This way the correct result can easily be checked and also verified against a reference implementation.

5. Non-Functional SoC Features Verification Report

Non-functional features considered in this report include those related to timing and to security. Oppositely to functional features, which can generally be verified explicitly, non-functional features are better suited for validation activities where test campaigns based on stress workloads provide evidence of a lack of failures to meet specific non-functional features. This relates, in part, to the fact that non-functional behavior emanates mostly from the integration and allowed use of the different parts of the platform, rather than from the individual parts themselves.

In the verification side, limited activities to verify features related to real-time support and security concerns could be performed. They are described next. Note, however, that features verified are regarded as sufficient to meet real-time and security constraints on top of them, but this does not preclude that real-time and secure execution could not be achieved without those features. Hence, those features must be understood as sufficient but not necessary.

5.1. Real-time features

Non-functional features related to real-time relate to the ability to limit interference across cores in the access to shared resources. The involved elements to be verified are the following:

- The AHB bus implements a fair arbitration policy.
- Non-shared L1 caches.
- The L2 cache implements cache partitioning.
- The L2 cache serves requests from different cores fairly.
- The memory controller serves requests from different cores fairly.

5.1.1. AHB bus arbitration policy

The AHB bus implements round-robin arbitration across all masters. This feature avoids starvation of any master (and hence, the cores) by construction. Moreover, it guarantees a minimum frequency at which requests from any particular core are granted access to the bus.

The effectiveness and degree of fairness of the AHB bus arbiter will need to undergo appropriate validation tests in the integrated platform.

5.1.2. Non-shared L1 caches

Each core implements its L1 instruction and data caches. They are not shared, hence meaning that only the program running in the corresponding core can fetch and evict data from those L1 caches.

L1 data cache is inclusive with the L2 cache. Hence, L2 evictions could produce cascade L1 evictions. Therefore, whether the L1 cache adheres to the expected behavior to meet real-time needs depends on whether other cores can cause L2 evictions of any data in the L1 data cache of the core being considered.

5.1.3. L2 cache partitioning

The L2 cache implements cache partitioning. In particular, it can be configured to allocate one of its 4 ways to each of the 4 cores. This feature, by construction, impedes that one core evicts data fetched from another core. Therefore, L2 cache contents under the partitioned configuration are solely determined by the core owning the corresponding L2 partition, thus avoiding interference in the L2 cache contents.

As indicated before, whether L1 data cache evictions could be caused by another core depends on whether L2 cache evictions can be caused by another core. The use of L2 cache partitioning prevents that type of evictions from happening, thus avoiding both L2 contents interference and L1 data cache evictions caused by other cores.

Specific multicore validation tests are required to further assess this feature.

5.1.4. L2 cache fair request processing

The L2 cache does not implement itself any feature to segregate requests from different cores and hence, they are processed in a FCFS basis. This makes that whether the L2 cache behaves fairly with respect to the order to process requests depends on whether requests arrive in a fair order.

L2 requests arrive from the AHB bus and from the memory controller (in the form of responses in the latter case). Since the AHB bus provides fairness implementing round-robin arbitration, L2 requests from cores arrive in a fair order and hence, the FCFS policy of the L2 cache inherits the round-robin behavior of the AHB bus, thus effectively serving core requests in a round-robin fashion.

Regarding memory controller responses, since the memory controller can only respond to the requests sent by the L2 cache, and the memory controller implements some form of age control limiting the out-of-order processing of L2 requests, responses arrive to the L2 cache in an order still inheriting some degree of fairness, thus making L2 FCFS policy preserve such fairness.

As for the L2 cache partitioning feature, this real-time feature needs being properly assessed with appropriate validation tests.

5.1.5. Memory controller fair request processing

The DDR2/3 memory controller processes requests following specific policies intended to maximize throughput by reordering requests so that the cumulative latency to serve all of them is

the lowest possible. However, this is done with some limitations related not to starve any request. Hence, any requests can be delayed by up to a fixed number of requests.

Requests arrive from the L2 cache in a fair manner, given that the L2 cache receives requests in a round-robin fashion and sends them in a FCFS basis. Hence, requests from different cores arrive in a fair order to the memory controller. This implies that any requests from any core can be delayed by up to a fixed number of requests from other cores. While this may cause some non-negligible delay to serve some requests, it allows bounding how much each request will take to be served, and hence, allows building real-time guarantees on top.

Overall, the memory controller provides sufficient support to achieve real-time behavior on top, but such behavior can only be fully assessed by means of appropriate validation tests.

5.2. Security features

Non-functional features related to security relate to the ability to avoid clogging of resources and to limit information leaked across tasks. The involved elements to be verified are the following:

- Shared resources cannot be clogged by any core.
- Information leaked across tasks running in different cores is explicitly limited.

5.2.1. Avoiding resource clogging

A type of attacks to be prevented is that based on Denial of Service (DoS). DoS attacks rely on a process clogging a specific hardware resource so that it cannot provide service to the other cores.

DoS attacks can be prevented at hardware level by implementing fair arbitration policies in shared resources so that, despite the amount of load that a core can pose on them, some non-negligible bandwidth is guaranteed for the other cores.

By implementing round-robin arbitration in the AHB bus, which propagates to the slave devices attached to it (e.g. L2 cache and memory controller), resource clogging can be achieved neither in the AHB bus, nor in any slave device attached to it.

Moreover, note that, despite a core could generate transactions of arbitrarily large length (e.g. through generating requests to/from peripherals), the AHB bus implements a maximum burst length, thus effectively limiting the duration of a transaction, and therefore ensuring that requests from all cores are arbitrated frequently.

In any case, since the impact of arbitrarily large and frequent AHB requests on resource utilization needs being validated with appropriate tests to determine whether DoS attacks could be, at least, partially effective.

5.2.2. Information leakage

Side-channel attacks exploit any opportunity to gain information on its victim. In a multicore platform where tasks are executed in parallel, an effective use of resources makes that varying amounts of load cause variations in the use of shared resources. Hence, any attacker loading a shared resource may learn about its victim by measuring how often its own requests are served when performed concurrently with the victim ones, and thus learn how much the victim is using the corresponding shared resource. The only simple way to prevent such information leak relates to providing time-isolation (e.g. implementing TDMA arbitration), hence allocating time slots to each core and not allowing any other core to use them, so that one cannot learn about other tasks by sensing how much they use shared resources. However, such a solution is against efficiency.

An even more critical way to learn about a victim relates to the use of shared caches where an attacker can evict contents from selected cache sets and assess the impact this has on the victim. This way, the attacker can learn about the memory addresses used by the victim and infer security-sensitive information from there. This problem is avoided by construction by implementing L2 cache partitioning and L1 private caches since any attacker cannot generate evictions on the victim.