Imperas Guide to using Virtual Platforms

Platform / Module Specific Information for sifive.ovpworld.org / FU540

Imperas Software Limited

Imperas Buildings, North Weston
Thame, Oxfordshire, OX9 2HA, U.K.
docs@imperas.com.

Author	Imperas Software Limited
Version	20210408.0
Filename	Imperas_Platform_User_Guide_FU540.pdf
Created	05 May 2021
Status	OVP Standard Release

Copyright Notice

Right to Copy Documentation

The license agreement with Imperas permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the readers responsibility to determine the applicable regulations and to comply with them.

Disclaimer

IMPERAS SOFTWARE LIMITED, AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Model Release Status

This model is released as part of OVP releases and is included in OVPworld packages. Please visit OVPworld.org.

Table Of Contents

1.0 Platform / Module: FU540

This document provides the details of the usage of an Imperas OVP Virtual Platform / Module. The first half of the document covers specifics of this particular component. For more information about Imperas OVP virtual platforms, how they are built and used, please see the later sections in this document.

1.1 Virtual Platform / Module Type

Hardware described using OVP can either be a platform, module, processor, or peripheral.
This hardware component is described as being a module. A module is a component that is used in other modules, platforms, or test harnesses. It is normally used to encapsulate a layer in a hierarchical system.

1.2 Licensing

Open Source Apache 2.0

1.3 Description

SiFive FU540-C000 SoC module.

On start up or reset, the reset code at 0x1004 will jump to a jump table entry indexed by the MSEL register at address 0x1000 (default initial MSEL value is 0xf which will cause a jump to address 0x10000). Use the msel peripheral's MSEL parameter to change the initial value of this register.

To run a bare metal application use the --program command line option to specify an elf file to be loaded. It must be linked to use addresses corresponding to the implemented memory regions. The --program option will override the initial pc with the ELF file's start address.

To facilitate booting Linux an OVP SmartLoader psuedo-peripheral has been included that provides the functionality of the ZSBL/FSBL. The SmartLoader's dtb parameter should be used to specify the device tree blob file to load, and the bbl elf file should be loaded using the --objfilenoentry command line option.

1.4 Reference

SiFive Freedom U540-C000 Manual FU540-C000-v1.0.pdf
(https://www.sifive.com/documentation/chips/freedom-u540-c000-manual)

1.5 Limitations

Caches and the Cache Controller are not modeled.

The Instruction Tightly Integrated Memory (ITIM) is implemented simply as RAM. Deallocation by writing to the byte immediately following the memory is a NOP.

The L2 Loosely Integrated Memory (L2-LIM) is implemented simply as RAM. It is always available, since the Cache Controller is not modeled.

The L2 Scratchpad memory is not modeled.

The Platform DMA Engine (PDMA) is not modeled.

The Pulse Width Modulator (PWM) is not modeled.

The Inter-Integrated Circuit (I2C) Master Interface is not modeled.

The Serial Peripheral Interface (SPI) is not modeled. Instead a Virtio Block MMIO device has been added at reserved address 0x1f000000, using interrupt 54.

The General Purpose Input/Output Controller (GPIO) is not modeled.

The One-Time Programmable Memory Interface (OTP) is not modeled.

DDR controller is not modeled. DDR memory is modeled as RAM.

The Debug Interface is not modeled.

1.6 Location

The FU540 virtual platform / module is located in an Imperas/OVP installation at the VLNV: sifive.ovpworld.org / module / FU540 / 1.0.

1.7 Module Simulation Attributes

Table 1. Module Simulation Attributes

Attribute	Value	Description
stoponctrlc	stoponctrlc	Stop on control-C

2.0 External Ports for Module FU540

Table 2. External Ports

Port Type	Port Name	Internal Connection
netport	gi1	gi1
netport	gi2	gi2
netport	gi3	gi3
netport	gi4	gi4
netport	gi5	gi5
netport	gi6	gi6
netport	gi7	gi7
netport	gi8	gi8
netport	gi9	gi9
netport	gi10	gi10
netport	gi11	gi11

netport	gi12	gi12
netport	gi13	gi13
netport	gi14	gi14
netport	gi15	gi15
netport	gi16	gi16
netport	gi17	gi17
netport	gi18	gi18
netport	gi19	gi19
netport	gi20	gi20
netport	gi21	gi21
netport	gi22	gi22
netport	gi23	gi23
netport	gi24	gi24
netport	gi25	gi25
netport	gi26	gi26
netport	gi27	gi27
netport	gi28	gi28
netport	gi29	gi29
netport	gi30	gi30
netport	gi31	gi31
netport	gi32	gi32
netport	gi33	gi33
netport	gi34	gi34
netport	gi35	gi35
netport	gi36	gi36
netport	gi37	gi37
netport	gi38	gi38
netport	gi39	gi39
netport	gi40	gi40
netport	gi41	gi41
netport	gi42	gi42
netport	gi43	gi43
netport	gi44	gi44
netport	gi45	gi45
netport	gi46	gi46
netport	gi47	gi47
netport	gi48	gi48
netport	gi49	gi49
netport	gi50	gi50
netport	gi51	gi51
netport	gi52	gi52
netport	gi53	gi53

3.0 Processor [sifive.ovpworld.org/processor/riscv/1.0] instance: E51

3.1 Processor model type: 'riscv' variant 'E51' definition

Imperas OVP processor models support multiple variants and details of the variants implemented in this model can be found in:

- the Imperas installation located at ImperasLib/source/sifive.ovpworld.org/processor/riscv/1.0/doc
- the OVP website: OVP_Model_Specific_Information_sifive_riscv_E51.pdf

3.1.1 Description

RISC-V E51 64-bit processor model

3.1.2 Licensing

This Model is released under the Open Source Apache 2.0

3.1.3 Extensions Enabled by Default

The model has the following architectural extensions enabled, and the following bits in the misa CSR Extensions field will be set upon reset:
misa bit 0: extension A (atomic instructions)
misa bit 2: extension C (compressed instructions)
misa bit 8: RV32I/RV64I/RV128I base integer instruction set
misa bit 12: extension M (integer multiply/divide instructions)
misa bit 20: extension U (User mode)
To specify features that can be dynamically enabled or disabled by writes to the misa register in addition to those listed above, use parameter "add_Extensions_mask". This is a string parameter containing the feature letters to add; for example, value "DV" indicates that double-precision floating point and the Vector Extension can be enabled or disabled by writes to the misa register, if supported on this variant.
Legacy parameter "misa_Extensions_mask" can also be used. This Uns32-valued parameter specifies all writable bits in the misa Extensions field, replacing any permitted bits defined in the base variant.
Note that any features that are indicated as present in the misa mask but absent in the misa will be ignored. See the next section.

3.1.4 Available Extensions Not Enabled by Default

The following extensions are supported by the model, but not enabled by default in this variant:
misa bit 3: extension D (double-precision floating point)
misa bit 5: extension F (single-precision floating point)
To add features from this list to the base variant, use parameter "add_Extensions". This is a string parameter containing the feature letters to add; for example, value "DV" indicates that double-precision floating point and the Vector Extension should be enabled, if they are currently absent and are available on this variant.
Legacy parameter "misa_Extensions" can also be used. This Uns32-valued parameter specifies the reset value for the misa CSR Extensions field, replacing any permitted bits defined in the base variant.

3.1.5 General Features

This is a multicore variant with 1 cores by default. The number of cores may be overridden with the "numHarts" parameter.
On this variant, the Machine trap-vector base-address register (mtvec) is writable. It can instead be configured as read-only using parameter "mtvec_is_ro".
Values written to "mtvec" are masked using the value 0x3fffffffd. A different mask of writable bits may be specified using parameter "mtvec_mask" if required. In addition, when Vectored interrupt mode is enabled,

parameter "tvec_align" may be used to specify additional hardware-enforced base address alignment. In this variant, "tvec_align" defaults to 64.
The initial value of "mtvec" is 0x0. A different value may be specified using parameter "mtvec" if required.
On reset, the model will restart at address 0x0. A different reset address may be specified using parameter "reset_address" or applied using optional input port "reset_addr" if required.
On an NMI, the model will restart at address 0x0. A different NMI address may be specified using parameter "nmi_address" or applied using optional input port "nmi_addr" if required.
WFI will halt the processor until an interrupt occurs. It can instead be configured as a NOP using parameter "wfi_is_nop". WFI timeout wait is implemented with a time limit of 0 (i.e. WFI causes an Illegal Instruction trap in Supervisor mode when mstatus.TW=1).
The "cycle" CSR is implemented in this variant. Set parameter "cycle_undefined" to True to instead specify that "cycle" is unimplemented and reads of it should trap to Machine mode.
The "time" CSR is not implemented in this variant and reads of it will require emulation in Machine mode. Set parameter "time_undefined" to False to instead specify that "time" is implemented.
The "instret" CSR is implemented in this variant. Set parameter "instret_undefined" to True to instead specify that "instret" is unimplemented and reads of it should trap to Machine mode.
Unaligned memory accesses are not supported by this variant. Set parameter "unaligned" to "T" to enable such accesses.
Unaligned memory accesses are not supported for AMO instructions by this variant. Set parameter "unalignedAMO" to "T" to enable such accesses.
8 PMP entries are implemented by this variant. Use parameter "PMP_registers" to specify a different number of PMP entries; set the parameter to 0 to disable the PMP unit. The PMP grain size (G) is 0, meaning that PMP regions as small as 4 bytes are implemented. Use parameter "PMP_grain" to specify a different grain size if required. Unaligned PMP accesses are not decomposed into separate aligned accesses; use parameter "PMP_decompose" to modify this behavior if required.
LR/SC instructions are implemented with a 64-byte reservation granule. A different granule size may be specified using parameter "lr_sc_grain".

3.1.6 CLIC

The model can be configured to implement a Core Local Interrupt Controller (CLIC) using parameter "CLICLEVELS"; when non-zero, the CLIC is present with the specified number of interrupt levels (2-256), as described in the RISC-V Core-Local Interrupt Controller specification, and further parameters are made available to configure other aspects of the CLIC. "CLICLEVELS" is zero in this variant, indicating that a CLIC is not implemented.

3.1.7 Load-Reserved/Store-Conditional Locking

By default, LR/SC locking is implemented automatically by the model and simulator, with a reservation granule defined by the "lr_sc_grain" parameter. It is also possible to implement locking externally to the model in a platform component, using the "LR_address", "SC_address" and "SC_valid" net ports, as described below.
The "LR_address" output net port is written by the model with the address used by a load-reserved instruction as it executes. This port should be connected as an input to the external lock management component, which should record the address, and also that an LR/SC transaction is active.

The "SC_address" output net port is written by the model with the address used by a store-conditional instruction as it executes. This should be connected as an input to the external lock management component, which should compare the address with the previously-recorded load-reserved address, and determine from this (and other implementation-specific constraints) whether the store should succeed. It should then immediately write the Boolean success/fail code to the "SC_valid" input net port of the model. Finally, it should update state to indicate that an LR/SC transaction is no longer active.
It is also possible to write zero to the "SC_valid" input net port at any time outside the context of a store-conditional instruction, which will mark any active LR/SC transaction as invalid.
Irrespective of whether LR/SC locking is implemented internally or externally, taking any exception or interrupt or executing exception-return instructions (e.g. MRET) will always mark any active LR/SC transaction as invalid.

3.1.8 Active Atomic Operation Indication

The "AMO_active" output net port is written by the model with a code indicating any current atomic memory operation while the instruction is active. The written codes are:
0: no atomic instruction active
1: AMOMIN active
2: AMOMAX active
3: AMOMINU active
4: AMOMAXU active
5: AMOADD active
6: AMOXOR active
7: AMOOR active
8: AMOAND active
9: AMOSWAP active
10: LR active
11: SC active

3.1.9 Interrupts

The "reset" port is an active-high reset input. The processor is halted when "reset" goes high and resumes execution from the reset address specified using the "reset_address" parameter or "reset_addr" port when the signal goes low. The "mcause" register is cleared to zero.
The "nmi" port is an active-high NMI input. The processor resumes execution from the address specified using the "nmi_address" parameter or "nmi_addr" port when the NMI signal goes high. The "mcause" register is cleared to zero.
All other interrupt ports are active high. For each implemented privileged execution level, there are by default input ports for software interrupt, timer interrupt and external interrupt; for example, for Machine mode, these are called "MSWInterrupt", "MTimerInterrupt" and "MExternalInterrupt", respectively. When the N extension is implemented, ports are also present for User mode. Parameter "unimp_int_mask" allows the default behavior to be changed to exclude certain interrupt ports. The parameter value is a mask in the same format as the "mip" CSR; any interrupt corresponding to a non-zero bit in this mask will be removed from the processor and read as zero in "mip", "mie" and "mideleg" CSRs (and Supervisor and User mode equivalents if implemented).

Parameter "external_int_id" can be used to enable extra interrupt ID input ports on each hart. If the parameter is True then when an external interrupt is applied the value on the ID port is sampled and used to fill the Exception Code field in the "mcause" CSR (or the equivalent CSR for other execution levels). For Machine mode, the extra interrupt ID port is called "MExternalInterruptID".
The "deferint" port is an active-high artifact input that, when written to 1, prevents any pending-and-enabled interrupt being taken (normally, such an interrupt would be taken on the next instruction after it becomes pending-and-enabled). The purpose of this signal is to enable alignment with hardware models in step-and-compare usage.

3.1.10 Debug Mode

The model can be configured to implement Debug mode using parameter "debug_mode". This implements features described in Chapter 4 of the RISC-V External Debug Support specification with version specified by parameter "debug_version" (see References). Some aspects of this mode are not defined in the specification because they are implementation-specific; the model provides infrastructure to allow implementation of a Debug Module using a custom harness. Features added are described below.
Parameter "debug_mode" can be used to specify three different behaviors, as follows:
1. If set to value "vector", then operations that would cause entry to Debug mode result in the processor jumping to the address specified by the "debug_address" parameter. It will execute at this address, in Debug mode, until a "dret" instruction causes return to non-Debug mode. Any exception generated during this execution will cause a jump to the address specified by the "dexc_address" parameter.
2. If set to value "interrupt", then operations that would cause entry to Debug mode result in the processor simulation call (e.g. opProcessorSimulate) returning, with a stop reason of OP_SR_INTERRUPT. In this usage scenario, the Debug Module is implemented in the simulation harness.
3. If set to value "halt", then operations that would cause entry to Debug mode result in the processor halting. Depending on the simulation environment, this might cause a return from the simulation call with a stop reason of OP_SR_HALT, or debug mode might be implemented by another platform component which then restarts the debugged processor again.

3.1.11 Debug State Entry

The specification does not define how Debug mode is implemented. In this model, Debug mode is enabled by a Boolean pseudo-register, "DM". When "DM" is True, the processor is in Debug mode. When "DM" is False, mode is defined by "mstatus" in the usual way.
Entry to Debug mode can be performed in any of these ways:
1. By writing True to register "DM" (e.g. using opProcessorRegWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate), dcsr cause will be reported as trigger;
2. By writing a 1 then 0 to net "haltreq" (using opNetWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate);
3. By writing a 1 to net "resethaltreq" (using opNetWrite) while the "reset" signal undergoes a negedge transition, followed by simulation of at least one cycle (e.g. using opProcessorSimulate);
4. By executing an "ebreak" instruction when Debug mode entry for the current processor mode is enabled by dcsr.ebreakm, dcsr.ebreaks or dcsr.ebreaku.
In all cases, the processor will save required state in "dpc" and "dcsr" and then perform actions described above, depending in the value of the "debug_mode" parameter.

3.1.12 Debug State Exit

Exit from Debug mode can be performed in any of these ways:
1. By writing False to register "DM" (e.g. using opProcessorRegWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate);
2. By executing an "dret" instruction when Debug mode.
In both cases, the processor will perform the steps described in section 4.6 (Resume) of the Debug specification.

3.1.13 Debug Registers

When Debug mode is enabled, registers "dcsr", "dpc", "dscratch0" and "dscratch1" are implemented as described in the specification. These may be manipulated externally by a Debug Module using opProcessorRegRead or opProcessorRegWrite; for example, the Debug Module could write "dcsr" to enable "ebreak" instruction behavior as described above, or read and write "dpc" to emulate stepping over an "ebreak" instruction prior to resumption from Debug mode.

3.1.14 Debug Mode Execution

The specification allows execution of code fragments in Debug mode. A Debug Module implementation can cause execution in Debug mode by the following steps:
1. Write the address of a Program Buffer to the program counter using opProcessorPCSet;
2. If "debug_mode" is set to "halt", write 0 to pseudo-register "DMStall" (to leave halted state);
3. If entry to Debug mode was handled by exiting the simulation callback, call opProcessorSimulate or opRootModuleSimulate to resume simulation.
Debug mode will be re-entered in these cases:
1. By execution of an "ebreak" instruction; or:
2. By execution of an instruction that causes an exception.
In both cases, the processor will either jump to the debug exception address, or return control immediately to the harness, with stopReason of OP_SR_INTERRUPT, or perform a halt, depending on the value of the "debug_mode" parameter.

3.1.15 Debug Single Step

When in Debug mode, the processor or harness can cause a single instruction to be executed on return from that mode by setting dcsr.step. After one non-Debug-mode instruction has been executed, control will be returned to the harness. The processor will remain in single-step mode until dcsr.step is cleared.

3.1.16 Debug Ports

Port "DM" is an output signal that indicates whether the processor is in Debug mode
Port "haltreq" is a rising-edge-triggered signal that triggers entry to Debug mode (see above).
Port "resethaltreq" is a level-sensitive signal that triggers entry to Debug mode after reset (see above).

3.1.17 Debug Mask

It is possible to enable model debug messages in various categories. This can be done statically using the "override_debugMask" parameter, or dynamically using the "debugflags" command. Enabled messages are specified using a bitmask value, as follows:

Value 0x002: enable debugging of PMP and virtual memory state;
Value 0x004: enable debugging of interrupt state.
All other bits in the debug bitmask are reserved and must not be set to non-zero values.

3.1.18 Integration Support

This model implements a number of non-architectural pseudo-registers and other features to facilitate integration.

3.1.19 CSR Register External Implementation

If parameter "enable_CSR_bus" is True, an artifact 16-bit bus "CSR" is enabled. Slave callbacks installed on this bus can be used to implement modified CSR behavior (use opBusSlaveNew or icmMapExternalMemory, depending on the client API). A CSR with index 0xABC is mapped on the bus at address 0xABC0; as a concrete example, implementing CSR "time" (number 0xC01) externally requires installation of callbacks at address 0xC010 on the CSR bus.

3.1.20 LR/SC Active Address

Artifact register "LRSCAddress" shows the active LR/SC lock address. The register holds all-ones if there is no LR/SC operation active or if LR/SC locking is implemented externally as described above.

3.1.21 Limitations

Instruction pipelines are not modeled in any way. All instructions are assumed to complete immediately. This means that instruction barrier instructions (e.g. fence.i) are treated as NOPs, with the exception of any Illegal Instruction behavior, which is modeled.
Caches and write buffers are not modeled in any way. All loads, fetches and stores complete immediately and in order, and are fully synchronous. Data barrier instructions (e.g. fence) are treated as NOPs, with the exception of any Illegal Instruction behavior, which is modeled.
Real-world timing effects are not modeled: all instructions are assumed to complete in a single cycle.
Hardware Performance Monitor registers are not implemented and hardwired to zero.

3.1.22 Verification

All instructions have been extensively tested by Imperas, using tests generated specifically for this model and also reference tests from https://github.com/riscv/riscv-tests.
Also reference tests have been used from various sources including:
https://github.com/riscv/riscv-tests
https://github.com/ucb-bar/riscv-torture
The Imperas OVPsim RISC-V models are used in the RISC-V Foundation Compliance Framework as a functional Golden Reference:
https://github.com/riscv/riscv-compliance
where the simulated model is used to provide the reference signatures for compliance testing. The Imperas OVPsim RISC-V models are used as reference in both open source and commercial instruction stream test generators for hardware design verification, for example:
http://valtrix.in/sting from Valtrix
https://github.com/google/riscv-dv from Google

The Imperas OVPsim RISC-V models are also used by commercial and open source RISC-V Core RTL developers as a reference to ensure correct functionality of their IP.

3.1.23 References

The Model details are based upon the following specifications:
RISC-V Instruction Set Manual, Volume I: User-Level ISA (User Architecture Version 20190305-Base-Ratification)
RISC-V Instruction Set Manual, Volume II: Privileged Architecture (Privileged Architecture Version 20190405-Priv-MSU-Ratification)
SiFive E51 Core Complex Manual v1p2

3.2 Instance Parameters

Several parameters can be specified when a processor is instanced in a platform. For this processor instance 'E51' it has been instanced with the following parameters:

Table 3. Processor Instance 'E51' Parameters (Configurations)

Parameter	Value	Description
simulateexceptions	simulateexceptions	Causes the processor simulate exceptions instead of halting
mips	1000	The nominal MIPS for the processor

Table 4. Processor Instance 'E51' Parameters (Attributes)

Parameter Name	Value	Type
mhartid	0	Uns64
local_int_num	48	Uns32
reset_address	0x1004	Uns64
variant	E51	enum

3.3 Memory Map for processor 'E51' bus: 'bus0'

Processor instance 'E51' is connected to bus 'bus0' using master port 'INSTRUCTION'.
Processor instance 'E51' is connected to bus 'bus0' using master port 'DATA'.

Table 5. Memory Map ('E51' / 'bus0' [width: 38])

Lo Address	Hi Address	Instance	Component
0x0	0xFF	safe0addr	ram
remappable	remappable	emgmt	trap
0x1000	0x1FFF	msel	MSEL
0x10000	0x17FFF	maskROM	ram
0x1000000	0x1001FFF	hart0DTIM	ram
0x1800000	0x1802000	hart0ITIM	ram
0x1808000	0x180F000	hart1ITIM	ram
0x1810000	0x1817000	hart2ITIM	ram
0x1818000	0x181F000	hart3ITIM	ram
0x1820000	0x1827000	hart4ITIM	ram
0x2000000	0x200BFFF	clint	CLINT
0x8000000	0x9FFFFFF	l2LIM	ram

0xC000000	0xFFFFFFF	plic	PLIC
0x10000000	0x10000FFF	prci	PRCI
0x10010000	0x1001001B	uart0	UART
0x10011000	0x1001101B	uart1	UART
0x10090000	0x10090FFF	emac	gem
0x1F000000	0x1F0001FF	vbd0	VirtioBlkMMIO
0x80000000	0xBFFFFFFF	mem1	ram

3.4 Net Connections to processor: 'E51'

Table 6. Processor Net Connections ('E51')

Net Port	Net	Instance	Component
hart0_MTimerInterrupt	MTimerInterrupt0	clint	CLINT
hart0_MSWInterrupt	MSWInterrupt0	clint	CLINT
hart0_MExternalInterrupt	irqT0	plic	PLIC

4.0 Processor [sifive.ovpworld.org/processor/riscv/1.0] instance: U54

4.1 Processor model type: 'riscv' variant 'U54' definition

Imperas OVP processor models support multiple variants and details of the variants implemented in this model can be found in:
- the Imperas installation located at ImperasLib/source/sifive.ovpworld.org/processor/riscv/1.0/doc
- the OVP website: OVP_Model_Specific_Information_sifive_riscv_U54.pdf

4.1.1 Description

RISC-V U54 64-bit processor model

4.1.2 Licensing

This Model is released under the Open Source Apache 2.0

4.1.3 Extensions Enabled by Default

The model has the following architectural extensions enabled, and the following bits in the misa CSR Extensions field will be set upon reset:
misa bit 0: extension A (atomic instructions)
misa bit 2: extension C (compressed instructions)
misa bit 3: extension D (double-precision floating point)
misa bit 5: extension F (single-precision floating point)
misa bit 8: RV32I/RV64I/RV128I base integer instruction set
misa bit 12: extension M (integer multiply/divide instructions)
misa bit 18: extension S (Supervisor mode)
misa bit 20: extension U (User mode)
To specify features that can be dynamically enabled or disabled by writes to the misa register in addition to those listed above, use parameter "add_Extensions_mask". This is a string parameter containing the feature

letters to add; for example, value "DV" indicates that double-precision floating point and the Vector Extension can be enabled or disabled by writes to the misa register, if supported on this variant.
Legacy parameter "misa_Extensions_mask" can also be used. This Uns32-valued parameter specifies all writable bits in the misa Extensions field, replacing any permitted bits defined in the base variant.
Note that any features that are indicated as present in the misa mask but absent in the misa will be ignored. See the next section.

4.1.4 General Features

This is a multicore variant with 1 cores by default. The number of cores may be overridden with the "numHarts" parameter.
On this variant, the Machine trap-vector base-address register (mtvec) is writable. It can instead be configured as read-only using parameter "mtvec_is_ro".
Values written to "mtvec" are masked using the value 0x3fffffffd. A different mask of writable bits may be specified using parameter "mtvec_mask" if required. In addition, when Vectored interrupt mode is enabled, parameter "tvec_align" may be used to specify additional hardware-enforced base address alignment. In this variant, "tvec_align" defaults to 64.
The initial value of "mtvec" is 0x0. A different value may be specified using parameter "mtvec" if required.
Values written to "stvec" are masked using the value 0xfffffffffffffffd. A different mask of writable bits may be specified using parameter "stvec_mask" if required. parameter "tvec_align" may be used to specify additional hardware-enforced base address alignment in the same manner as for the "mtvec" register, described above.
On reset, the model will restart at address 0x0. A different reset address may be specified using parameter "reset_address" or applied using optional input port "reset_addr" if required.
On an NMI, the model will restart at address 0x0. A different NMI address may be specified using parameter "nmi_address" or applied using optional input port "nmi_addr" if required.
WFI will halt the processor until an interrupt occurs. It can instead be configured as a NOP using parameter "wfi_is_nop". WFI timeout wait is implemented with a time limit of 0 (i.e. WFI causes an Illegal Instruction trap in Supervisor mode when mstatus.TW=1).
The "cycle" CSR is implemented in this variant. Set parameter "cycle_undefined" to True to instead specify that "cycle" is unimplemented and reads of it should trap to Machine mode.
The "time" CSR is not implemented in this variant and reads of it will require emulation in Machine mode. Set parameter "time_undefined" to False to instead specify that "time" is implemented.
The "instret" CSR is implemented in this variant. Set parameter "instret_undefined" to True to instead specify that "instret" is unimplemented and reads of it should trap to Machine mode.
A 0-bit ASID is implemented. Use parameter "ASID_bits" to specify a different implemented ASID size if required.
This variant supports address translation modes 0 and 8. Use parameter "Sv_modes" to specify a bit mask of different modes if required.
TLB behavior is controlled by parameter "ASIDCacheSize". If this parameter is 0, then an unlimited number of TLB entries will be maintained concurrently. If this parameter is non-zero, then only TLB entries for up to "ASIDCacheSize" different ASIDs will be maintained concurrently initially; as new ASIDs are used, TLB entries for less-recently used ASIDs are deleted, which improves model performance in some cases. If the model detects that the TLB entry cache is too small (entry ejections are very frequent), it will

increase the cache size automatically. In this variant, "ASIDCacheSize" is 8
Unaligned memory accesses are not supported by this variant. Set parameter "unaligned" to "T" to enable such accesses.
Unaligned memory accesses are not supported for AMO instructions by this variant. Set parameter "unalignedAMO" to "T" to enable such accesses.
8 PMP entries are implemented by this variant. Use parameter "PMP_registers" to specify a different number of PMP entries; set the parameter to 0 to disable the PMP unit. The PMP grain size (G) is 0, meaning that PMP regions as small as 4 bytes are implemented. Use parameter "PMP_grain" to specify a different grain size if required. Unaligned PMP accesses are not decomposed into separate aligned accesses; use parameter "PMP_decompose" to modify this behavior if required.
LR/SC instructions are implemented with a 64-byte reservation granule. A different granule size may be specified using parameter "lr_sc_grain".

4.1.5 Floating Point Features

The D extension is enabled in this variant only if the F extension is also enabled. Set parameter "d_requires_f"to "F" to allow D and F to be independently enabled.
Half precision floating point is not implemented. Use parameter "Zfh" to enable this if required.
By default, the processor starts with floating-point instructions disabled (mstatus.FS=0). Use parameter "mstatus_FS" to force mstatus.FS to a non-zero value for floating-point to be enabled from the start.
The specification is imprecise regarding the conditions under which mstatus.FS is set to Dirty state (3). Parameter "mstatus_fs_mode" can be used to specify the required behavior in this model, as described below.
If "mstatus_fs_mode" is set to "always_dirty" then the model implements a simplified floating point status view in which mstatus.FS holds values 0 (Off) and 3 (Dirty) only; any write of values 1 (Initial) or 2 (Clean) from privileged code behave as if value 3 was written.
If "mstatus_fs_mode" is set to "write_1" then mstatus.FS will be set to 3 (Dirty) by any explicit write to the fflags, frm or fcsr control registers, or by any executed instruction that writes an FPR, or by any executed floating point compare or conversion to integer/unsigned that signals a floating point exception. Floating point compare or conversion to integer/unsigned instructions that do not signal an exception will not set mstatus.FS.
If "mstatus_fs_mode" is set to "write_any" then mstatus.FS will be set to 3 (Dirty) by any explicit write to the fflags, frm or fcsr control registers, or by any executed instruction that writes an FPR, or by any executed floating point compare or conversion even if those instructions do not signal a floating point exception.
In this variant, "mstatus_fs_mode" is set to "always_dirty".

4.1.6 CLIC

The model can be configured to implement a Core Local Interrupt Controller (CLIC) using parameter "CLICLEVELS"; when non-zero, the CLIC is present with the specified number of interrupt levels (2-256), as described in the RISC-V Core-Local Interrupt Controller specification, and further parameters are made available to configure other aspects of the CLIC. "CLICLEVELS" is zero in this variant, indicating that a CLIC is not implemented.

4.1.7 Load-Reserved/Store-Conditional Locking

By default, LR/SC locking is implemented automatically by the model and simulator, with a reservation granule defined by the "lr_sc_grain" parameter. It is also possible to implement locking externally to the model in a platform component, using the "LR_address", "SC_address" and "SC_valid" net ports, as described below.

The "LR_address" output net port is written by the model with the address used by a load-reserved instruction as it executes. This port should be connected as an input to the external lock management component, which should record the address, and also that an LR/SC transaction is active.

The "SC_address" output net port is written by the model with the address used by a store-conditional instruction as it executes. This should be connected as an input to the external lock management component, which should compare the address with the previously-recorded load-reserved address, and determine from this (and other implementation-specific constraints) whether the store should succeed. It should then immediately write the Boolean success/fail code to the "SC_valid" input net port of the model. Finally, it should update state to indicate that an LR/SC transaction is no longer active.

It is also possible to write zero to the "SC_valid" input net port at any time outside the context of a store-conditional instruction, which will mark any active LR/SC transaction as invalid.

Irrespective of whether LR/SC locking is implemented internally or externally, taking any exception or interrupt or executing exception-return instructions (e.g. MRET) will always mark any active LR/SC transaction as invalid.

4.1.8 Active Atomic Operation Indication

The "AMO_active" output net port is written by the model with a code indicating any current atomic memory operation while the instruction is active. The written codes are:

0: no atomic instruction active
1: AMOMIN active
2: AMOMAX active
3: AMOMINU active
4: AMOMAXU active
5: AMOADD active
6: AMOXOR active
7: AMOOR active
8: AMOAND active
9: AMOSWAP active
10: LR active
11: SC active

4.1.9 Interrupts

The "reset" port is an active-high reset input. The processor is halted when "reset" goes high and resumes execution from the reset address specified using the "reset_address" parameter or "reset_addr" port when the signal goes low. The "mcause" register is cleared to zero.

The "nmi" port is an active-high NMI input. The processor resumes execution from the address specified using the "nmi_address" parameter or "nmi_addr" port when the NMI signal goes high. The "mcause" register is cleared to zero.

All other interrupt ports are active high. For each implemented privileged execution level, there are by default input ports for software interrupt, timer interrupt and external interrupt; for example, for Machine mode, these are called "MSWInterrupt", "MTimerInterrupt" and "MExternalInterrupt", respectively. When the N extension is implemented, ports are also present for User mode. Parameter "unimp_int_mask" allows the default behavior to be changed to exclude certain interrupt ports. The parameter value is a mask in the same format as the "mip" CSR; any interrupt corresponding to a non-zero bit in this mask will be removed from the processor and read as zero in "mip", "mie" and "mideleg" CSRs (and Supervisor and User mode equivalents if implemented).
Parameter "external_int_id" can be used to enable extra interrupt ID input ports on each hart. If the parameter is True then when an external interrupt is applied the value on the ID port is sampled and used to fill the Exception Code field in the "mcause" CSR (or the equivalent CSR for other execution levels). For Machine mode, the extra interrupt ID port is called "MExternalInterruptID".
The "deferint" port is an active-high artifact input that, when written to 1, prevents any pending-and-enabled interrupt being taken (normally, such an interrupt would be taken on the next instruction after it becomes pending-and-enabled). The purpose of this signal is to enable alignment with hardware models in step-and-compare usage.

4.1.10 Debug Mode

The model can be configured to implement Debug mode using parameter "debug_mode". This implements features described in Chapter 4 of the RISC-V External Debug Support specification with version specified by parameter "debug_version" (see References). Some aspects of this mode are not defined in the specification because they are implementation-specific; the model provides infrastructure to allow implementation of a Debug Module using a custom harness. Features added are described below.
Parameter "debug_mode" can be used to specify three different behaviors, as follows:
1. If set to value "vector", then operations that would cause entry to Debug mode result in the processor jumping to the address specified by the "debug_address" parameter. It will execute at this address, in Debug mode, until a "dret" instruction causes return to non-Debug mode. Any exception generated during this execution will cause a jump to the address specified by the "dexc_address" parameter.
2. If set to value "interrupt", then operations that would cause entry to Debug mode result in the processor simulation call (e.g. opProcessorSimulate) returning, with a stop reason of OP_SR_INTERRUPT. In this usage scenario, the Debug Module is implemented in the simulation harness.
3. If set to value "halt", then operations that would cause entry to Debug mode result in the processor halting. Depending on the simulation environment, this might cause a return from the simulation call with a stop reason of OP_SR_HALT, or debug mode might be implemented by another platform component which then restarts the debugged processor again.

4.1.11 Debug State Entry

The specification does not define how Debug mode is implemented. In this model, Debug mode is enabled by a Boolean pseudo-register, "DM". When "DM" is True, the processor is in Debug mode. When "DM" is False, mode is defined by "mstatus" in the usual way.
Entry to Debug mode can be performed in any of these ways:
1. By writing True to register "DM" (e.g. using opProcessorRegWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate), dcsr cause will be reported as trigger;

2. By writing a 1 then 0 to net "haltreq" (using opNetWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate);
3. By writing a 1 to net "resethaltreq" (using opNetWrite) while the "reset" signal undergoes a negedge transition, followed by simulation of at least one cycle (e.g. using opProcessorSimulate);
4. By executing an "ebreak" instruction when Debug mode entry for the current processor mode is enabled by dcsr.ebreakm, dcsr.ebreaks or dcsr.ebreaku.
In all cases, the processor will save required state in "dpc" and "dcsr" and then perform actions described above, depending in the value of the "debug_mode" parameter.

4.1.12 Debug State Exit

Exit from Debug mode can be performed in any of these ways:
1. By writing False to register "DM" (e.g. using opProcessorRegWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate);
2. By executing an "dret" instruction when Debug mode.
In both cases, the processor will perform the steps described in section 4.6 (Resume) of the Debug specification.

4.1.13 Debug Registers

When Debug mode is enabled, registers "dcsr", "dpc", "dscratch0" and "dscratch1" are implemented as described in the specification. These may be manipulated externally by a Debug Module using opProcessorRegRead or opProcessorRegWrite; for example, the Debug Module could write "dcsr" to enable "ebreak" instruction behavior as described above, or read and write "dpc" to emulate stepping over an "ebreak" instruction prior to resumption from Debug mode.

4.1.14 Debug Mode Execution

The specification allows execution of code fragments in Debug mode. A Debug Module implementation can cause execution in Debug mode by the following steps:
1. Write the address of a Program Buffer to the program counter using opProcessorPCSet;
2. If "debug_mode" is set to "halt", write 0 to pseudo-register "DMStall" (to leave halted state);
3. If entry to Debug mode was handled by exiting the simulation callback, call opProcessorSimulate or opRootModuleSimulate to resume simulation.
Debug mode will be re-entered in these cases:
1. By execution of an "ebreak" instruction; or:
2. By execution of an instruction that causes an exception.
In both cases, the processor will either jump to the debug exception address, or return control immediately to the harness, with stopReason of OP_SR_INTERRUPT, or perform a halt, depending on the value of the "debug_mode" parameter.

4.1.15 Debug Single Step

When in Debug mode, the processor or harness can cause a single instruction to be executed on return from that mode by setting dcsr.step. After one non-Debug-mode instruction has been executed, control will be returned to the harness. The processor will remain in single-step mode until dcsr.step is cleared.

4.1.16 Debug Ports

Port "DM" is an output signal that indicates whether the processor is in Debug mode
Port "haltreq" is a rising-edge-triggered signal that triggers entry to Debug mode (see above).
Port "resethaltreq" is a level-sensitive signal that triggers entry to Debug mode after reset (see above).

4.1.17 Debug Mask

It is possible to enable model debug messages in various categories. This can be done statically using the "override_debugMask" parameter, or dynamically using the "debugflags" command. Enabled messages are specified using a bitmask value, as follows:
Value 0x002: enable debugging of PMP and virtual memory state;
Value 0x004: enable debugging of interrupt state.
All other bits in the debug bitmask are reserved and must not be set to non-zero values.

4.1.18 Integration Support

This model implements a number of non-architectural pseudo-registers and other features to facilitate integration.

4.1.19 CSR Register External Implementation

If parameter "enable_CSR_bus" is True, an artifact 16-bit bus "CSR" is enabled. Slave callbacks installed on this bus can be used to implement modified CSR behavior (use opBusSlaveNew or icmMapExternalMemory, depending on the client API). A CSR with index 0xABC is mapped on the bus at address 0xABC0; as a concrete example, implementing CSR "time" (number 0xC01) externally requires installation of callbacks at address 0xC010 on the CSR bus.

4.1.20 LR/SC Active Address

Artifact register "LRSCAddress" shows the active LR/SC lock address. The register holds all-ones if there is no LR/SC operation active or if LR/SC locking is implemented externally as described above.

4.1.21 Limitations

Instruction pipelines are not modeled in any way. All instructions are assumed to complete immediately. This means that instruction barrier instructions (e.g. fence.i) are treated as NOPs, with the exception of any Illegal Instruction behavior, which is modeled.
Caches and write buffers are not modeled in any way. All loads, fetches and stores complete immediately and in order, and are fully synchronous. Data barrier instructions (e.g. fence) are treated as NOPs, with the exception of any Illegal Instruction behavior, which is modeled.
Real-world timing effects are not modeled: all instructions are assumed to complete in a single cycle.
Hardware Performance Monitor registers are not implemented and hardwired to zero.
The TLB is architecturally-accurate but not device accurate. This means that all TLB maintenance and address translation operations are fully implemented but the cache is larger than in the real device.

4.1.22 Verification

All instructions have been extensively tested by Imperas, using tests generated specifically for this model and also reference tests from https://github.com/riscv/riscv-tests.

Also reference tests have been used from various sources including:
https://github.com/riscv/riscv-tests
https://github.com/ucb-bar/riscv-torture
The Imperas OVPsim RISC-V models are used in the RISC-V Foundation Compliance Framework as a functional Golden Reference:
https://github.com/riscv/riscv-compliance
where the simulated model is used to provide the reference signatures for compliance testing. The Imperas OVPsim RISC-V models are used as reference in both open source and commercial instruction stream test generators for hardware design verification, for example:
http://valtrix.in/sting from Valtrix
https://github.com/google/riscv-dv from Google
The Imperas OVPsim RISC-V models are also used by commercial and open source RISC-V Core RTL developers as a reference to ensure correct functionality of their IP.

4.1.23 References

The Model details are based upon the following specifications:
RISC-V Instruction Set Manual, Volume I: User-Level ISA (User Architecture Version 20190305-Base-Ratification)
RISC-V Instruction Set Manual, Volume II: Privileged Architecture (Privileged Architecture Version 20190405-Priv-MSU-Ratification)
SiFive U54-MC Core Complex Manual v1p0

4.2 Instance Parameters

Several parameters can be specified when a processor is instanced in a platform. For this processor instance 'U54' it has been instanced with the following parameters:

Table 7. Processor Instance 'U54' Parameters (Configurations)

Parameter	Value	Description
simulateexceptions	simulateexceptions	Causes the processor simulate exceptions instead of halting
mips	1000	The nominal MIPS for the processor

Table 8. Processor Instance 'U54' Parameters (Attributes)

Parameter Name	Value	Type
mhartid	1	Uns64
local_int_num	48	Uns32
reset_address	0x1004	Uns64
numHarts	4	Uns32
variant	U54	enum

4.3 Memory Map for processor 'U54' bus: 'bus0'

Processor instance 'U54' is connected to bus 'bus0' using master port 'INSTRUCTION'.
Processor instance 'U54' is connected to bus 'bus0' using master port 'DATA'.

Table 9. Memory Map ('U54' / 'bus0' [width: 38])

Lo Address	Hi Address	Instance	Component
0x0	0xFF	safe0addr	ram
remappable	remappable	emgmt	trap
0x1000	0x1FFF	msel	MSEL
0x10000	0x17FFF	maskROM	ram
0x1000000	0x1001FFF	hart0DTIM	ram
0x1800000	0x1802000	hart0ITIM	ram
0x1808000	0x180F000	hart1ITIM	ram
0x1810000	0x1817000	hart2ITIM	ram
0x1818000	0x181F000	hart3ITIM	ram
0x1820000	0x1827000	hart4ITIM	ram
0x2000000	0x200BFFF	clint	CLINT
0x8000000	0x9FFFFFF	l2LIM	ram
0xC000000	0xFFFFFFF	plic	PLIC
0x10000000	0x10000FFF	prci	PRCI
0x10010000	0x1001001B	uart0	UART
0x10011000	0x1001101B	uart1	UART
0x10090000	0x10090FFF	emac	gem
0x1F000000	0x1F0001FF	vbd0	VirtioBlkMMIO
0x80000000	0xBFFFFFFF	mem1	ram

4.4 Net Connections to processor: 'U54'

Table 10. Processor Net Connections ('U54')

Net Port	Net	Instance	Component
hart1_MTimerInterrupt	MTimerInterrupt1	clint	CLINT
hart1_MSWInterrupt	MSWInterrupt1	clint	CLINT
hart1_MExternalInterrupt	irqT1	plic	PLIC
hart1_SExternalInterrupt	irqT2	plic	PLIC
hart2_MTimerInterrupt	MTimerInterrupt2	clint	CLINT
hart2_MSWInterrupt	MSWInterrupt2	clint	CLINT
hart2_MExternalInterrupt	irqT3	plic	PLIC
hart2_SExternalInterrupt	irqT4	plic	PLIC
hart3_MTimerInterrupt	MTimerInterrupt3	clint	CLINT
hart3_MSWInterrupt	MSWInterrupt3	clint	CLINT
hart3_MExternalInterrupt	irqT5	plic	PLIC
hart3_SExternalInterrupt	irqT6	plic	PLIC
hart4_MTimerInterrupt	MTimerInterrupt4	clint	CLINT
hart4_MSWInterrupt	MSWInterrupt4	clint	CLINT
hart4_MExternalInterrupt	irqT7	plic	PLIC
hart4_SExternalInterrupt	irqT8	plic	PLIC

5.0 Peripheral Instances

5.1 Peripheral [sifive.ovpworld.org/peripheral/MSEL/1.0] instance: msel

5.1.1 Description

Mode Select reset module. Entered on reset and calls boot code based on MSEL pin state. Override the MSEL parameter to specify the initial value for the MSEL pin state (default 0xf). From application code or debugger write to the MSEL register at offset 0 to change the MSEL pin state.

5.1.2 Limitations

None

5.1.3 Licensing

Open Source Apache 2.0

5.1.4 Reference

SiFive Freedom U540-C000 Manual FU540-C000-v1.0.pdf
(https://www.sifive.com/documentation/chips/freedom-u540-c000-manual)

There are no configuration options set for this peripheral instance.

5.2 Peripheral [riscv.ovpworld.org/peripheral/CLINT/1.0] instance: clint

5.2.1 Licensing

Open Source Apache 2.0

5.2.2 Description

Risc-V Core Local Interruptor (CLINT). Use the num_harts parameter to specify the number of harts suported (default 1). For each supported hart there will be an MTimerInterruptN and MSWInterruptN net port, plus msipN and mtimecmpN registers implemented, where N is a value from 0..num_harts-1 There is also a single mtime register.

5.2.3 Limitations

Writes to mtime register are not supported

5.2.4 Reference

SiFive Freedom U540-C000 Manual FU540-C000-v1.0.pdf
(https://www.sifive.com/documentation/chips/freedom-u540-c000-manual)

Table 11. Configuration options (attributes) set for instance 'clint'

Attribute	Value	Type	Expression
num_harts	5	Uns32	
clockMHz	1.0	double	

5.3 Peripheral [riscv.ovpworld.org/peripheral/PLIC/1.0] instance: plic

5.3.1 Licensing

Open Source Apache 2.0

5.3.2 Limitations
None

5.3.3 Description
PLIC Interrupt Controller
Use parameters to configure specific implementation.
Default model is based on SiFive PLIC implementation details - other variations are available (e.g. Andes NCEPLIC100).

5.3.4 Reference
The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.10 (https://riscv.org/specifications/privileged-isa)
SiFive Freedom U540-C000 Manual FU540-C000-v1.0.pdf (https://www.sifive.com/documentation/chips/freedom-u540-c000-manual)

Table 12. Configuration options (attributes) set for instance 'plic'

Attribute	Value	Type	Expression
num_targets	9	Uns32	
num_sources	53	Uns32	

5.4 Peripheral [sifive.ovpworld.org/peripheral/PRCI/1.0] instance: prci

5.4.1 Description
Power Reset Clocking Interrupt (PRCI) block for SiFive FU540 chip

5.4.2 Limitations
None
Register only model. Reset values based on typical post-ZSBL configuration (1GHz coreclk, 500MHz tlclk).

5.4.3 Licensing
Open Source Apache 2.0

5.4.4 Reference
SiFive Freedom U540-C000 Manual FU540-C000-v1.0.pdf (https://www.sifive.com/documentation/chips/freedom-u540-c000-manual)

There are no configuration options set for this peripheral instance.

5.5 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart0

5.5.1 Licensing
Open Source Apache 2.0

5.5.2 Description
Sifive UART

5.5.3 Limitations
When simulatebaud parameter is set to true baud rate delays are modeled for receive only, not transmit. Data always sent immediately.

5.5.4 Reference
SiFive Freedom U540-C000 Manual FU540-C000-v1.0.pdf
(https://www.sifive.com/documentation/chips/freedom-u540-c000-manual)

Table 13. Configuration options (attributes) set for instance 'uart0'

Attribute	Value	Type	Expression
refClkFreq	500000000	uns32	

5.6 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart1

5.6.1 Licensing
Open Source Apache 2.0

5.6.2 Description
Sifive UART

5.6.3 Limitations
When simulatebaud parameter is set to true baud rate delays are modeled for receive only, not transmit. Data always sent immediately.

5.6.4 Reference
SiFive Freedom U540-C000 Manual FU540-C000-v1.0.pdf
(https://www.sifive.com/documentation/chips/freedom-u540-c000-manual)

Table 14. Configuration options (attributes) set for instance 'uart1'

Attribute	Value	Type	Expression
refClkFreq	500000000	uns32	

5.7 Peripheral [cadence.ovpworld.org/peripheral/gem/1.0] instance: emac

5.7.1 Description
Model of Cadence Gigabit Ethernet Controller (GEM). For further details please consult README-

EMAC.txt
This model is based upon the data and use in the Xilinx Zynq
Basic network Tx/Rx functionality tested using Xilinx Linux Kernel using wget and other similar tools
Tested with Xilinx SDK Example driver.

5.7.2 Licensing
Open Source Apache 2.0

5.7.3 Limitations
This model is based upon the data from the Xilinx Zynq platform, other registers may not be included.
Does not implement: VLAN, pause frames, filtering or timestamps.

5.7.4 Reference
Zynq-7000 TRM
(https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf)

There are no configuration options set for this peripheral instance.

5.8 Peripheral [ovpworld.org/peripheral/trap/1.0] instance: emgmt

5.8.1 Description
Open a port and allocate a region that is defined by parameters.
The region can be configured to act as standard memory or can report read/write accesses.

5.8.2 Licensing
Open Source Apache 2.0

5.8.3 Limitations
This peripheral cannot be used in a hardware description used to generate a TLM platform.

5.8.4 Reference
This is not based upon the operation of a real device but is intended to be used for bring up and development of new virtual platforms.

Table 15. Configuration options (attributes) set for instance 'emgmt'

Attribute	Value	Type	Expression
portAddress	0x100a0000	uns32	

5.9 Peripheral [ovpworld.org/peripheral/VirtioBlkMMIO/1.0] instance: vbd0

5.9.1 Description
VIRTIO version 1 mmio block device This model implements a VIRTIO MMIO block device as

described in: http://docs.oasis-open.org/virtio/virtio/v1.0/virtio-v1.0.pdf. Use the VB_DRIVE parameter to specify the disk image file to use. Set the VB_DRIVE_DELTA parameter to 1 to prevent writes to disk during simulation from changing the image file.

5.9.2 Limitations

Only supports the Legacy (Device Version 1) interface. Only little endian guests are supported.

5.9.3 Licensing

Open Source Apache 2.0

5.9.4 Reference

http://docs.oasis-open.org/virtio/virtio/v1.0/virtio-v1.0.pdf

There are no configuration options set for this peripheral instance.

5.10 Peripheral [riscv.ovpworld.org/peripheral/SmartLoaderRV64Linux/1.0] instance: smartLoader

5.10.1 Licensing

Open Source Apache 2.0

5.10.2 Description

Psuedo-peripheral to insert boot code for a Riscv 64-bit Linux kernel boot. Loads simulated memory with a device tree blob file and boot code to set regs and jump to a Risc-v Linux Kernel.

5.10.3 Limitations

Only supports little endian

5.10.4 Reference

RISC-V Linux Kernel development

Table 16. Configuration options (attributes) set for instance 'smartLoader'

Attribute	Value	Type	Expression
bootaddr	0x80000000	address	
slbootaddr	0x10000	address	
membase	0x80000000	address	
memsize	0x40000000	address	

6.0 Overview of Imperas OVP Virtual Platforms

This document provides the details of the usage of an Imperas OVP Virtual Platform / Module. The first half of the document covers specifics of this particular virtual platform / module.

This second part of the document, includes information about Imperas OVP virtual platforms and modules, how they are built and used.

The Imperas virtual platforms are designed to provide a base for you to run high-speed software simulations of CPU-based SoCs and platforms on any suitable PC. They are typically based on the functionality of vendors fixed or evaluation platforms, enabling you to simulate software on these reference platforms. Typically virtual platforms are fixed and require the vendor to modify or extend them. Imperas virtual platforms are different in that they enable you to extend the functionality of the virtual platform, to closer reflect your own platform, by adding more component models, running different operating systems or adding additional applications.

Imperas virtual platforms are created using the Imperas iGen technology, allowing them to be used with Imperas OVP based simulators and also with Accellera/OSCI compliant SystemC simulators and commercial EDA System Design environments that use SystemC.

Virtual platforms include simulation models of the target devices, including the processor model(s) for the target device plus enough peripheral models to boot an operating system or run bare metal applications. The platform and the peripheral models used in most of the virtual platforms are open source, so that you can easily add new models to the platform as well as modify the existing models. Some models are only provided as binary, normally because the IP owner has restricted the release of the model source. In this case, please contact Imperas for more information.

There are typically several generic flavors of the virtual platforms for specific processor families, some targeting full operating systems, such as Linux, and some which focus on Real Time Operating Systems (RTOS) such as Mentor Nucleus or freeRTOS. OVP models of the processor cores are included in the virtual platforms, and for those processors which support mulitple cores SMP Linux is often supported for that virtual platform. For all of these virtual platforms, many of the peripheral components of the platform are modeled, often including the Ethernet and USB components. The semi-hosting capability of the Imperas virtual platform simulator products enables connection via the Ethernet and USB components from the virtual platform to the real world via the x86 host machine.

The Imperas OVP CPU models are written using the OVP Virtual Machine Interface (VMI) API that defines the behavior of the processor. The VMI API makes a clear line between model and simulator allowing very good optimization and world class high speed performance. The processor models are Instruction Accurate and do not model the detailed cycle timing of the processor and they implement functionality at the level of a Programmers View of the processor and peripherals and the software running on them does not know it is not running on hardware. Many models are provided as a binary shared object and also as source. This allows the download and use of the model binary or the use of the source to explore

and modify the model. The models are run through an extensive QA and regression testing process and most processor model families are validated using technology provided by the processor IP owners. All the models in this platform are developed with the Open Virtual Platforms APIs and are implemented in C. A platform can be modeled as different levels of hierarchy using separately describable and compilable modules.

More information on modeling and APIs can be found on the www.OVPworld.org site.

7.0 Getting Started with Imperas OVP Virtual Platforms

Virtual platforms are downloadable from the OVPworld website OVPworld.org/downloads. You need to browse and look for '<platform processor name> Examples'. You do need to be registered and logged in on the OVP site to download. OVPworld currently provides 32 bit host versions of packages containing virtual platforms.

When downloading, choose, Linux or Windows host. 32 bit packages can be installed and executed on 32 bit or 64 bit hosts. If you require a 64 bit host version please contact Imperas.

For example, for the ARM Versatile Express platform booting Linux on Cortex-A15MP Single, Dual, and Quad core procesors, you would want the download package: 'OVPsim_demo_Linux_ArmVersatileExpress_arm_Cortex-A15MP'.

Most virtual platform packages contain the platform and all the processor and peripheral models needed. You will need to download a simulator to run the platform. You can use OVPsim, downloadable from OVPworld.org/downloads, or you can use one of the Imperas simulators (imperas.com/products) available commercially from Imperas.

8.0 Simulating Software

8.1 Getting a license key to run

After you have downloaded you will need a runtime license key before the simulators will run. For OVPsim please visit OVPworld.org/likey and provide the required information and an evaluation/demo license key will be automatically sent to you. If you are using Imperas, then please contact Imperas for a license key.

8.2 Normal runs

To run a platform, read the section below on command line control of the platform and the section on setting command line arguments.

8.3 Loading Software

For most virtual platforms the platform is already configured to run the default software application/program and there is normally a script to run that sets some arguments. You can then copy/edit this script to select your own applications etc.

The example application programs are typically .elf format files and are provided pre-compiled. There are normally makefiles and associated scripts to recompile the example applications.

To find more information about compiling and loading software, the following document should be looked at: Imperas_Installation_and_Getting_Started.pdf.

8.4 Semihosting

In a virtual platform, semihosting is not normally used as there is normally hardware that implements the appropriate functionality - for example I/O will be handled by UARTs etc.

8.5 Using a terminal (UART)

If the platform includes one or more UARTs you will need to connect a terminal program to it so that you can see output and type into the simulated program. Review the list of peripherals below and see what configuration options it has been set with. In most cases there is an option to set to instruct the simulator to 'pop up' a terminal window connected to the simulated UART.

8.6 Interacting with the simulation (keyboard and mouse)

If the platform has a simulated UART you can normally set a command to get the simulator to pop up a terminal window allowing you to see output from the simulated UART and also allowing you to type characters into the UART that can be processed by the simulated software.

If your simulated platform has an LCD device then you can often configure it to recognize mouse movements and mouse clicks - allowing full interaction.

To see these interactions in action, have a look at some of the available videos available at OVPworld.org/demosandvideos.

8.7 More Information (Documentation) on Simulation

To find more information about running simulations and more of the options the simulators provide, the following documents should be looked at:
Imperas_Installation_and_Getting_Started.pdf
Simulation_Control_of_Platforms_and_Modules_User_Guide.pdf
Advanced_Simulation_Control_of_Platforms_and_Modules_User_Guide.pdf
OVP_Control_File_User_Guide.pdf

A full list of the currently available OVP documentation is available: OVPworld.org/documentation.

9.0 Debugging Software running on an Imperas OVP Virtual Platform

The Imperas and OVP simulators have several different interfaces to debuggers. These include several proprietary formats and also the standard GNU RSP format is supported allowing many compatible debuggers to be used. Below are some examples that Imperas directly support.

9.1 Debugging with GDB

A GNU debugger (GDB) can be connected to a processor in a platform using the RSP protocol. This allows the application program running on a processor to be debugged using a specific GDB for the processor selected. When using the Imperas Professional products many connections can be made allowing a GDB to be connected to all the processors in the platform.

The use of GDB is documented: OVPsim_Debugging_Applications_with_GDB_User_Guide.pdf.

*9.2 Debugging with Imperas M*DBG*

The Imperas multi-processor debugger can be connected to a platform and through this connection you can debug application programs running on all of the processors instanced within the platform. It is also capable, within this single unified environment, to debug peripheral model behavioral code in conjunction with the processor application programs.

For more information please see the Imperas M*DBG user guide.

The Imperas multi-processor debugger is also capable of controlling the Imperas Verification Analysis aand Profiling (VAP) tools in real time, making them invaluable to application program development, debugging and analysis.

For more information please see the Imperas VAP tools user guide.

9.3 Debugging with the Imperas eGui and GDB

Imperas eGui gives a GUI front end to the use of the GDB debugger. It allows use of all the features of GDB including source level application program debugging on processors.

*9.4 Debugging with the Imperas eGui and M*DBG*

Imperas eGui gives a GUI front end to the Imperas multi-processor debugger. It provides all the features of this debugger but does so with source level application program debugging on processors and source level debugging of the behavioral code on peripheral components in the platform. A context view shows all the processor and peripheral components within the platform and allows switching between them to examine the state of each at the event at which the simulation was stopped

Imperas eGui provides a menu from which the Imperas VAP tools can be controlled.

9.5 Debugging with Imperas eGui and Eclipse

Imperas provide a GUI based on Eclipse called eGui. This provides a GUI front end to use with a standard GDB or the Imperas MPD (Multi-Processor Debugger).

The use of eGui is documented: eGui_Eclipse_User_Guide.pdf.

A standard Eclipse CDT development environment can be connected to one or more processors in a

platform (multiple processors require an Imperas professional product). The simulation platform is started remotely or using the external tool feature in Eclipse, opens a debug port and awaits the connection with Eclipse. All features provided by the Eclipse CDT development environment are available to be used to debug software applications executing on the processors in the platform.

The use of Eclipse is documented: OVPsim_Debugging_Applications_with_Eclipse_User_Guide.pdf.

9.6 Debugging applications running under a simulated operating system

If the simulated platform is running an Operating System and the platform has a UART or Ethernet etc connection then it is often possible to connect an external debugger and debug the applications running under the simulated operating system.

An example would be a simulated platform running the Linux operating system, such as the MIPS Malta, or ARM Versatile Express. Within the simulated Linux you can start a gdbserver that connects from within the simulation through a UART out to the host PC via a port. Within the host PC you start a terminal program and connect to the port with a debugger such as GDB and can then debug the simulated user application.

10.0 Modifying the Platform / Module

10.1 Platforms / Modules use C/C++ and OVP APIs

The Imperas and OVP simulators execute a platform / module that is written in C/C++ and that makes function calls into the simulator's APIs. Thus the virtual platform / module is compiled from C/C++ into a binary shared object that the simulator loads and runs. OVP provides the definition and documentation that defines the C APIs for modeling the platforms, modules, the peripherals, and the processors. You can find more information about these APIs on the OVP website and in the OVP API documentation.

10.2 Platforms/Modules/Peripherals can be easily built with iGen from Imperas

Imperas provides a product 'iGen' that takes an input script file and creates the C/C++ files needed for platforms, modules, and peripherals - it creates the C/C++ file that is compiled into the platform, module or peripheral that is needed as an object file by the simulator. iGen creates the C/C++ files, you then need to add any necessary behaviors or further details etc. For platforms iGen creates either a C platform or a C++ SystemC TLM2 platform. For peripherals or modules iGen creates the C files and also provides a native C++ SystemC TLM2 interface to allow the peripheral/module to be instantiated in SystemC TLM2 platforms.

Information on iGen is available from: imperas.com/products.

10.3 Re-configuring the platform

There will nornmally be several configuration options that you can set when running the platform without the neeed to change any source. Refer to the section above on command line arguments. If these do not allow you to make the changes you need, then you may need to edit and recompile the source of the platform.

The source of the platform, modules, and the source of the peripherals will be installed as part of the packages you are using. The sources are located in the Imperas/OVP installation VLNV source tree. The VLNV term refers to: Vendor (eg arm.ovpworld.org), Library (eg platform), Name, (eg ArmVersatileExpress-CA15), and Version (eg 1.0). To modify the platform, locate the platform source files.

If you are an Imperas user and have access to iGen, we recommend you modify the source script files and regenerate and recompile the C that makes up the platform. Refer to the Imperas iGen model generator guide and the Imperas platform generator guide.

If you are using the C or SystemC TLM2 platforms with OVPsim, then you can edit the C/C++ files, recompile the source directly using the supplied makefiles, and the run the simulator directly with the resultant shared object.

10.4 Replacing peripherals components

If you need to replace peripherals, find the appropriate place in the source of the platform, make the change you need, and recompile etc. Look in the library for documentation on available peripherals and their configuration options.

10.5 Adding new peripherals components

If you need to add peripherals, find the appropriate place in the source, make the additions you need, and recompile etc. Look in the library for documentation on available peripherals and their configuration options.

If you need to create new peripheral components then use iGen to very quickly create the necessary C/C++ files that get you started. With iGen you can create peripherals with register/memory state in a few lines of iGen source. When adding behavior to the peripherals refer to the OVP API documentation.

11.0 Available Virtual Platforms

Table 17. Imperas / OVP Extendable Platform Kits (13 available)

Name	Vendor
AlteraCycloneIII_3c120	altera.ovpworld.org
AlteraCycloneV_HPS	altera.ovpworld.org
ArmIntegratorCP	arm.ovpworld.org
ArmVersatileExpress	arm.ovpworld.org
ArmVersatileExpress-CA15	arm.ovpworld.org
ArmVersatileExpress-CA9	arm.ovpworld.org
AtmelAT91SAM7	atmel.ovpworld.org
FreescaleKinetis60	freescale.ovpworld.org
FreescaleKinetis64	freescale.ovpworld.org
FreescaleVybridVFxx	freescale.ovpworld.org
MipsMalta	mips.ovpworld.org
RenesasUPD70F3441	renesas.ovpworld.org
XilinxML505	xilinx.ovpworld.org

Table 18. Imperas General Virtual Platforms (6 available)

Name	Vendor
arm-ti-eabi	arm.imperas.com
armm-ti-coff	arm.imperas.com
armm-ti-eabi	arm.imperas.com
HeteroAlteraCycloneV_HPS_CycloneIII_3c120	imperas.ovpworld.org
HeteroArmNucleusMIPSLinux	imperas.ovpworld.org
SiFiveFU540	imperas.ovpworld.org

Table 19. Imperas Modules (component of other platforms) (55 available)

Name	Vendor
AlteraCycloneIII_3c120	altera.ovpworld.org
AlteraCycloneV_HPS	altera.ovpworld.org
AE350	andes.ovpworld.org
ARMv8-A-FMv1	arm.ovpworld.org
ArmIntegratorCP	arm.ovpworld.org
ArmVersatileExpress	arm.ovpworld.org
ArmVersatileExpress-CA15	arm.ovpworld.org
ArmVersatileExpress-CA9	arm.ovpworld.org
AtmelAT91SAM7	atmel.ovpworld.org
ArmCortexMFreeRTOS	imperas.ovpworld.org
ArmCortexMuCOS-II	imperas.ovpworld.org
ArmuKernel	imperas.ovpworld.org
ArmuKernelDual	imperas.ovpworld.org
BareMetalMIPS	imperas.ovpworld.org
Dual_ARMv8-A-FMv1_VLAN	imperas.ovpworld.org
Hetero_1xArm_3xMips32	imperas.ovpworld.org
Hetero_ARM_RISCV_NeuralNetwork	imperas.ovpworld.org

Hetero_ARMv8-A-FMv1_Cortex-M3	imperas.ovpworld.org
Hetero_ARMv8-A-FMv1_MIPS_microAptiv	imperas.ovpworld.org
Hetero_AlteraCycloneV_HPS_AlteraCycloneIII_3c120	imperas.ovpworld.org
Hetero_ArmIntegratorCP_XilinxMicroBlaze	imperas.ovpworld.org
Hetero_ArmVersatileExpress_MipsMalta	imperas.ovpworld.org
Hetero_ArmVersatileExpress_XilinxMicroBlaze	imperas.ovpworld.org
Quad_ArmVersatileExpress-CA15	imperas.ovpworld.org
RiscvRV32FreeRTOS	imperas.ovpworld.org
MipsMalta	mips.ovpworld.org
iMX6S	nxp.ovpworld.org
RenesasUPD70F3441	renesas.ovpworld.org
ghs-multi	renesas.ovpworld.org
virtio	riscv.ovpworld.org
FaultInjection	safepower.ovpworld.org
PublicDemonstrator	safepower.ovpworld.org
Zynq_PL_DualMicroblaze	safepower.ovpworld.org
Zynq_PL_NoC	safepower.ovpworld.org
Zynq_PL_NoC_node	safepower.ovpworld.org
Zynq_PL_NostrumNoC	safepower.ovpworld.org
Zynq_PL_NostrumNoC_node	safepower.ovpworld.org
Zynq_PL_RO	safepower.ovpworld.org
Zynq_PL_SingleMicroblaze	safepower.ovpworld.org
Zynq_PL_TTELNoC	safepower.ovpworld.org
Zynq_PL_TTELNoC_node	safepower.ovpworld.org
Zynq_PL_TTELNoC_processing_node_public_demonstrator	safepower.ovpworld.org
Zynq_PL_TTELNoC_public_demonstrator	safepower.ovpworld.org
Zynq_PL_TTELNoC_sensor_actor_node_public_demonstrator	safepower.ovpworld.org
FU540	sifive.ovpworld.org
S51CC	sifive.ovpworld.org
coreip-s51-arty	sifive.ovpworld.org
coreip-s51-rtl	sifive.ovpworld.org
dualFifo	vendor.com
XilinxML505	xilinx.ovpworld.org
Zynq	xilinx.ovpworld.org
Zynq_PL_Default	xilinx.ovpworld.org
Zynq_PS	xilinx.ovpworld.org
zc702	xilinx.ovpworld.org
zc706	xilinx.ovpworld.org

Table 20. Imperas / OVP Bare Metal Virtual Platforms (22 available)

Name	Vendor
BareMetalNios_IISingle	altera.ovpworld.org
BareMetalArcSingle	arc.ovpworld.org
BareMetalArm7Single	arm.ovpworld.org
BareMetalArmCortexADual	arm.ovpworld.org
BareMetalArmCortexASingle	arm.ovpworld.org
BareMetalArmCortexASingleAngelTrap	arm.ovpworld.org
BareMetalArmCortexMSingle	arm.ovpworld.org

ArmCortexMFreeRTOS	imperas.ovpworld.org
ArmCortexMuCOS-II	imperas.ovpworld.org
BareMetalArmx1Mips32x3	imperas.ovpworld.org
Or1kUclinux	imperas.ovpworld.org
BareMetalM14KSingle	mips.ovpworld.org
BareMetalMips32Dual	mips.ovpworld.org
BareMetalMips32Single	mips.ovpworld.org
BareMetalMips64Single	mips.ovpworld.org
BareMetalMipsDual	mips.ovpworld.org
BareMetalMipsSingle	mips.ovpworld.org
BareMetalOr1kSingle	ovpworld.org
BareMetalM16cSingle	posedgesoft.ovpworld.org
BareMetalPowerPc32Single	power.ovpworld.org
BareMetalV850Single	renesas.ovpworld.org
ghs-multi	renesas.ovpworld.org

#