Imperas Guide to using Virtual Platforms

Platform / Module Specific Information for imperas.ovpworld.org / Hetero_ARM_RISCV_NeuralNetwork

Imperas Software Limited

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

Author	Imperas Software Limited
Version	20211118.0
Filename	Imperas_Platform_User_Guide_Hetero_ARM_RISCV_NeuralNetwork.pdf
Created	31 December 2021
Status	OVP Standard Release

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: Hetero_ARM_RISCV_NeuralNetwork

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 Description

Platform for FreeRTOS bring

1.3 Licensing

Open Source Apache 2.0

1.4 Limitations

Created to demostrate specific Nerual network applications

1.5 Reference

Alexnet and Minst Neural Networks

1.6 Location

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

1.7 Module Simulation Attributes

Table 1. Module Simulation Attributes

Attribute	Value	Description
stoponctrlc	stoponctrlc	Stop on control-C

2.0 Processor [arm.ovpworld.org/processor/arm/1.0] instance: arm_cpu

2.1 Processor model type: 'arm' variant 'Cortex-A57MPx1' 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/arm.ovpworld.org/processor/arm/1.0/doc
- the OVP website: OVP_Model_Specific_Information_arm_Cortex-A57MPx1.pdf

2.1.1 Description

ARM Processor Model

2.1.2 Licensing

Usage of binary model under license governing simulator usage.
Note that for models of ARM CPUs the license includes the following terms:
Licensee is granted a non-exclusive, worldwide, non-transferable, revocable licence to:
If no source is being provided to the Licensee: use and copy only (no modifications rights are granted) the model for the sole purpose of designing, developing, analyzing, debugging, testing, verifying, validating and optimizing software which: (a) (i) is for ARM based systems; and (ii) does not incorporate the ARM Models or any part thereof; and (b) such ARM Models may not be used to emulate an ARM based system to run application software in a production or live environment.
If source code is being provided to the Licensee: use, copy and modify the model for the sole purpose of designing, developing, analyzing, debugging, testing, verifying, validating and optimizing software which: (a) (i) is for ARM based systems; and (ii) does not incorporate the ARM Models or any part thereof; and (b) such ARM Models may not be used to emulate an ARM based system to run application software in a production or live environment.
In the case of any Licensee who is either or both an academic or educational institution the purposes shall be limited to internal use.
Except to the extent that such activity is permitted by applicable law, Licensee shall not reverse engineer, decompile, or disassemble this model. If this model was provided to Licensee in Europe, Licensee shall not reverse engineer, decompile or disassemble the Model for the purposes of error correction.
The License agreement does not entitle Licensee to manufacture in silicon any product based on this model.
The License agreement does not entitle Licensee to use this model for evaluating the validity of any ARM patent.
Source of model available under separate Imperas Software License Agreement.

2.1.3 Limitations

Instruction pipelines are not modeled in any way. All instructions are assumed to complete immediately. This means that instruction barrier instructions (e.g. ISB, CP15ISB) are treated as NOPs, with the exception of any undefined instruction behavior, which is modeled. The model does not implement speculative fetch behavior. The branch cache is not 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 (as if the memory was of Strongly Ordered or Device-nGnRnE type). Data barrier instructions (e.g. DSB, CP15DSB) are treated as NOPs, with the exception of any undefined instruction behavior, which is modeled. Cache manipulation instructions are implemented as NOPs, with the exception of any undefined instruction behavior, which is modeled.
Real-world timing effects are not modeled: all instructions are assumed to complete in a single cycle.
Performance Monitors are implemented as a register interface only except for the cycle counter, which is implemented assuming one instruction per cycle.
TLBs are 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.
Debug registers are implemented but non-functional (which is sufficient to allow operating systems such as Linux to boot). Debug state is not implemented.

2.1.4 Verification

Models have been extensively tested by Imperas. ARM Cortex-A models have been successfully used by customers to simulate SMP Linux, Ubuntu Desktop, VxWorks and ThreadX on Xilinx Zynq virtual platforms.

2.1.5 Core Features

AArch64 is implemented at EL3, EL2, EL1 and EL0.
AArch32 is implemented at EL3, EL2, EL1 and EL0.

2.1.6 Memory System

Security extensions are implemented (also known as TrustZone). To make non-secure accesses visible externally, override ID_AA64MMFR0_EL1.PARange to specify the required physical bus size (32, 36, 40, 42, 44, 48 or 52 bits) and connect the processor to a bus one bit wider (33, 37, 41, 43, 45, 49 or 53 bits, respectively). The extra most-significant bit is the NS bit, indicating a non-secure access. If non-secure accesses are not required to be made visible externally, connect the processor to a bus of exactly the size implied by ID_AA64MMFR0_EL1.PARange.
VMSA EL1, EL2 and EL3 stage 1 address translation is implemented. VMSA stage 2 address translation is implemented.
LPA (large physical address extension) is implemented as standard in ARMv8.
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 (especially when 16-bit ASIDs are in use). 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

2.1.7 Advanced SIMD and Floating-Point Features

SIMD and VFP instructions are implemented.
The model implements trapped exceptions if FPTrap is set to 1 in MVFR0 (for AArch32) or MVFR0_EL1 (for AArch64). When floating point exception traps are taken, cumulative exception flags are not updated (in other words, cumulative flag state is always the same as prior to instruction execution, even for SIMD instructions). When multiple enabled exceptions are raised by a single floating point operation, the exception reported is the one in least-significant bit position in FPSCR (for AArch32) or FPCR (for AArch64). When multiple enabled exceptions are raised by different SIMD element computations, the exception reported is selected from the lowest-index-number SIMD operation. Contact Imperas if requirements for exception reporting differ from these.
Trapped exceptions not are implemented in this variant (FPTrap=0)

2.1.8 Generic Timer

Generic Timer is present. Use parameter "override_timerScaleFactor" to specify the counter rate as a fraction of the processor MIPS rate (e.g. 10 implies Generic Timer counters increment once every 10 processor instructions).

2.1.9 Generic Interrupt Controller

GIC block is implemented (GICv2, including security extensions). Accesses to GIC registers can be viewed externally by connecting to the 32-bit GICRegisters bus port. Secure register accesses are at offset 0x0 on this bus; for example, a secure access to GIC register GICD_CTLR can be observed by monitoring address 0x00001000. Non-secure accesses are at offset 0x80000000 on this bus; for example, a non-secure access to GIC register GICD_CTLR can be observed by monitoring address 0x80001000
The internal GIC block can be disabled by raising signal GICCDISABLE, in which case the GIC needs to be modeled using a platform component instead. Input signals vfiq_CPU<N> and virq_CPU<N> can be used by this component to raise virtual FIQ and IRQ interrupts on cores in the cluster if required.

2.1.10 Debug Mask

It is possible to enable model debug features in various categories. This can be done statically using the "override_debugMask" parameter, or dynamically using the "debugflags" command. Enabled debug features are specified using a bitmask value, as follows:
Value 0x004: enable debugging of MMU/MPU mappings.
Value 0x020: enable debugging of reads and writes of GIC block registers.
Value 0x040: enable debugging of exception routing via the GIC model component.
Value 0x080: enable debugging of all system register accesses.
Value 0x100: enable debugging of all traps of system register accesses.
Value 0x200: enable verbose debugging of other miscellaneous behavior (for example, the reason why a particular instruction is undefined).
Value 0x400: enable debugging of Performance Monitor timers
Value 0x800: enable dynamic validation of TLB entries against in-memory page table contents (finds some classes of error where page table entries are updated without a subsequent flush of affected TLB entries).
All other bits in the debug bitmask are reserved and must not be set to non-zero values.

2.1.11 AArch32 Unpredictable Behavior

Many AArch32 instruction behaviors are described in the ARM ARM as CONSTRAINED UNPREDICTABLE. This section describes how such situations are handled by this model.

2.1.12 Equal Target Registers

Some instructions allow the specification of two target registers (for example, double-width SMULL, or some VMOV variants), and such instructions are CONSTRAINED UNPREDICTABLE if the same target register is specified in both positions. In this model, such instructions are treated as UNDEFINED.

2.1.13 Floating Point Load/Store Multiple Lists

Instructions that load or store a list of floating point registers (e.g. VSTM, VLDM, VPUSH, VPOP) are CONSTRAINED UNPREDICTABLE if either the uppermost register in the specified range is greater than 32 or (for 64-bit registers) if more than 16 registers are specified. In this model, such instructions are treated as UNDEFINED.

2.1.14 Floating Point VLD[2-4]/VST[2-4] Range Overflow

Instructions that load or store a fixed number of floating point registers (e.g. VST2, VLD2) are

CONSTRAINED UNPREDICTABLE if the upper register bound exceeds the number of implemented floating point registers. In this model, these instructions load and store using modulo 32 indexing (consistent with AArch64 instructions with similar behavior).

2.1.15 If-Then (IT) Block Constraints

Where the behavior of an instruction in an if-then (IT) block is described as CONSTRAINED UNPREDICTABLE, this model treats that instruction as UNDEFINED.

2.1.16 Use of R13

In architecture variants before ARMv8, use of R13 was described as CONSTRAINED UNPREDICTABLE in many circumstances. From ARMv8, most of these situations are no longer considered unpredictable. This model allows R13 to be used like any other GPR, consistent with the ARMv8 specification.

2.1.17 Use of R15

Use of R15 is described as CONSTRAINED UNPREDICTABLE in many circumstances. This model allows such use to be configured using the parameter "unpredictableR15" as follows:
Value "undefined": any reference to R15 in such a situation is treated as UNDEFINED;
Value "nop": any reference to R15 in such a situation causes the instruction to be treated as a NOP;
Value "raz_wi": any reference to R15 in such a situation causes the instruction to be treated as a RAZ/WI (that is, R15 is read as zero and write-ignored);
Value "execute": any reference to R15 in such a situation is executed using the current value of R15 on read, and writes to R15 are allowed (but are not interworking).
Value "assert": any reference to R15 in such a situation causes the simulation to halt with an assertion message (allowing any such unpredictable uses to be easily identified).
In this variant, the default value of "unpredictableR15" is "undefined".

2.1.18 Unpredictable Instructions in Some Modes

Some instructions are described as CONSTRAINED UNPREDICTABLE in some modes only (for example, MSR accessing SPSR is CONSTRAINED UNPREDICTABLE in User and System modes). This model allows such use to be configured using the parameter "unpredictableModal", which can have values "undefined" or "nop". See the previous section for more information about the meaning of these values.
In this variant, the default value of "unpredictableModal" is "undefined".

2.1.19 Integration Support

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

2.1.20 Memory Transaction Query

Two registers are intended for use within memory callback functions to provide additional information about the current memory access. Register transactPL indicates the processor execution level of the current access (0-3). Note that for load/store translate instructions (e.g. LDRT, STRT) the reported execution level will be 0, indicating an EL0 access. Register transactAT indicates the type of memory access: 0 for a normal read or write; and 1 for a physical access resulting from a page table walk.

2.1.21 Page Table Walk Query

A banked set of registers provides information about the most recently completed page table walk. There are up to six banks of registers: bank 0 is for stage 1 walks, bank 1 is for stage 2 walks, and banks 2-5 are for stage 2 walks initiated by stage 1 level 0-3 entry lookups, respectively. Banks 1-5 are present only for processors with virtualization extensions. The currently active bank can be set using register PTWBankSelect. Register PTWBankValid is a bitmask indicating which banks contain valid data: for example, the value 0xb indicates that banks 0, 1 and 3 contain valid data.
Within each bank, there are registers that record addresses and values read during that page table walk. Register PTWBase records the table base address, register PTWInput contains the input address that starts a walk, register PTWOutput contains the result address and register PTWPgSize contains the page size (PTWOutput and PTWPgSize are valid only if the page table walk completes). Registers PTWAddressL0-PTWAddressL3 record the addresses of level 0 to level 3 entries read, respectively. Register PTWAddressValid is a bitmask indicating which address registers contain valid data: bits 0-3 indicate PTWAddressL0-PTWAddressL3, respectively, bit 4 indicates PTWBase, bit 5 indicates PTWInput, bit 6 indicates both PTWOutput and PTWPgSize. For example, the value 0x73 indicates that PTWBase, PTWInput, PTWOutput, PTWPgSize and PTWAddressL0-L1 are valid but PTWAddressL2-L3 are not. Register PTWAddressNS is a bitmask indicating whether an address is in non-secure memory: bits 0-3 indicate PTWAddressL0-PTWAddressL3, respectively, bit 4 indicates PTWBase, bit 6 indicates PTWOutput (PTWInput is a VA and thus has no secure/non-secure info). Registers PTWValueL0-PTWValueL3 contain page table entry values read at level 0 to level 3. Register PTWValueValid is a bitmask indicating which value registers contain valid data: bits 0-3 indicate PTWValueL0-PTWValueL3, respectively.

2.1.22 Artifact Page Table Walks

Registers are also available to enable a simulation environment to initiate an artifact page table walk (for example, to determine the ultimate PA corresponding to a given VA). Register PTWI_EL1S initiates a secure EL1 table walk for a fetch. Register PTWD_EL1S initiates a secure EL1 table walk for a load or store (note that current ARM processors have unified TLBs, so these registers are synonymous). Registers PTW[ID]_EL1NS initiate walks for non-secure EL1 accesses. Registers PTW[ID]_EL2 initiate EL2 walks. Registers PTW[ID]_S2 initiate stage 2 walks. Registers PTW[ID]_EL3 initiate AArch64 EL3 walks. Finally, registers PTW[ID]_current initiate current-mode walks (useful in a memory callback context). Each walk fills the query registers described above.

2.1.23 MMU and Page Table Walk Events

Two events are available that allow a simulation environment to be notified on MMU and page table walk actions. Event mmuEnable triggers when any MMU is enabled or disabled. Event pageTableWalk triggers on completion of any page table walk (including artifact walks).

2.1.24 Artifact Address Translations

A simulation environment can trigger an artifact address translation operation by writing to the architectural address translation registers (e.g. ATS1CPR). The results of such translations are written to an integration support register artifactPAR, instead of the architectural PAR register. This means that such artifact writes will not perturb architectural state.

2.1.25 TLB Invalidation

A simulation environment can cause TLB state for one or more address translation regimes in the processor to be flushed by writing to the artifact register ResetTLBs. The argument is a bitmask value, in which non-zero bits select the TLBs to be flushed, as follows:
Bit 0: EL0/EL1 stage 1 secure TLB
Bit 1: EL0/EL1 stage 1 non-secure TLB
Bit 2: EL2 stage 1 TLB
Bit 3: EL0/EL1 non-secure stage 2 TLB
Bit 4: EL3 stage 1 TLB

2.1.26 Halt Reason Introspection

An artifact register HaltReason can be read to determine the reason or reasons that a processor is halted. This register is a bitfield, with the following encoding: bit 0 indicates the processor has executed a wait-for-event (WFE) instruction; bit 1 indicates the processor has executed a wait-for-interrupt (WFI) instruction; and bit 2 indicates the processor is held in reset.

2.1.27 System Register Access Monitor

If parameter "enableSystemMonitorBus" is True, an artifact 32-bit bus "SystemMonitor" is enabled for each PE. Every system register read or write by that PE is then visible as a read or write on this artifact bus, and can therefore be monitored using callbacks installed in the client environment (use opBusReadMonitorAdd/opBusWriteMonitorAdd or icmAddBusReadCallback/icmAddBusWriteCallback, depending on the client API). The format of the address on the bus is as follows:
bits 31:26 - zero
bit 25 - 1 if AArch64 access, 0 if AArch32 access
bit 24 - 1 if non-secure access, 0 if secure access
bits 23:20 - CRm value
bits 19:16 - CRn value
bits 15:12 - op2 value
bits 11:8 - op1 value
bits 7:4 - op0 value (AArch64) or coprocessor number (AArch32)
bits 3:0 - zero
As an example, to view non-secure writes to writes to CNTFRQ_EL0 in AArch64 state, install a write monitor on address range 0x020e0330:0x020e0333.

2.1.28 System Register Implementation

If parameter "enableSystemBus" is True, an artifact 32-bit bus "System" is enabled for each PE. Slave callbacks installed on this bus can be used to implement modified system register behavior (use opBusSlaveNew or icmMapExternalMemory, depending on the client API). The format of the address on the bus is the same as for the system monitor bus, described above.

2.2 Instance Parameters

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

Table 2. Processor Instance 'arm_cpu' Parameters (Configurations)

Parameter	Value	Description
endian	little	Select processor endian (big or little)
simulateexceptions	simulateexceptions	Causes the processor simulate exceptions instead of halting
mips	50	The nominal MIPS for the processor

Table 3. Processor Instance 'arm_cpu' Parameters (Attributes)

Parameter Name	Value	Type
variant	Cortex-A57MPx1	enum
compatibility	ISA	enum
showHiddenRegs	0	bool

2.3 Memory Map for processor 'arm_cpu' bus: 'arm_bus'

Processor instance 'arm_cpu' is connected to bus 'arm_bus' using master port 'INSTRUCTION'.
Processor instance 'arm_cpu' is connected to bus 'arm_bus' using master port 'DATA'.

Table 4. Memory Map ('arm_cpu' / 'arm_bus' [width: 44])

Lo Address	Hi Address	Instance	Component
0x0	0xFFFFFFF	arm_ram	ram
remappable	remappable	fb	frameBuffer
0x10000000	0x10000FFF	fb	frameBuffer
0x20000000	0x3FFFFFFF	arm_bridge	bridge

Table 5. Bridged Memory Map ('arm_cpu' / 'arm_bridge' / 'shared_bus' [width: 38])

Lo Address	Hi Address	Instance	Component
0x20000000	0x2FFFFFFF	shared_ram	ram
0x30010000	0x3001001B	uart0	UART
0x30011000	0x3001101B	uart1	UART
0x30012000	0x3001201B	uart2	UART
0x30013000	0x3001301B	uart3	UART
0x30014000	0x3001401B	uart4	UART
0x30015000	0x3001501B	uart5	UART
0x30016000	0x3001601B	uart6	UART
0x30017000	0x3001701B	uart7	UART
0x30018000	0x3001801B	uart8	UART
0x30019000	0x3001901B	uart9	UART
0x3001A000	0x3001A01B	uart10	UART
0x3001B000	0x3001B01B	uart11	UART
0x3001C000	0x3001C01B	uart12	UART
0x3001D000	0x3001D01B	uart13	UART
0x3001E000	0x3001E01B	uart14	UART
0x3001F000	0x3001F01B	uart15	UART
0x30020000	0x3002001B	uart16	UART
0x30021000	0x3002101B	uart17	UART
0x30030000	0x300300FF	cpuload	ram
0x30040000	0x30040003	usecCount	usecCounter

0x30100000	0x3010BFFF	clint	CLINT

2.4 Net Connections to processor: 'arm_cpu'
There are no nets connected to this processor.

3.0 Processor [sifive.ovpworld.org/processor/riscv/1.0] instance: rv64_cpu

3.1 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

3.1.1 Description
RISC-V U54 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 corresponding 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. Parameter "sub_Extensions_mask" can be used to disable dynamic update of features in the same way.
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 Disabling Extensions
The following extensions are enabled by default in the model and can be disabled:
misa bit 0: extension A (atomic instructions)

misa bit 3: extension D (double-precision floating point)
misa bit 5: extension F (single-precision floating point)
misa bit 12: extension M (integer multiply/divide instructions)
To disable features that are enabled by default, use parameter "sub_Extensions". This is a string containing identification letters of features to disable; for example, value "DF" indicates that double-precision and single-precision floating point extensions should be disabled, if they are enabled by default on this variant.
To remove features from this list from the implicitly-enabled set (not visible in the misa register), use parameter "sub_implicit_Extensions". This is a string parameter in the same format as the "sub_Extensions" parameter described above.

3.1.5 Multicore Features

This is a multicore variant with 1 harts by default. The number of harts may be overridden with the "numHarts" parameter.

3.1.6 mtvec CSR

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 0x3ffffffffd. 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.
If parameter "mtvec_sext" is True, values written to "mtvec" are sign-extended from the most-significant writable bit. In this variant, "mtvec_sext" is False, indicating that "mtvec" is not sign-extended.
The initial value of "mtvec" is 0x0. A different value may be specified using parameter "mtvec" if required.

3.1.7 stvec CSR

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. 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.
If parameter "stvec_sext" is True, values written to "stvec" are sign-extended from the most-significant writable bit. In this variant, "stvec_sext" is False, indicating that "stvec" is not sign-extended.

3.1.8 Reset

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.

3.1.9 NMI

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. The cause reported on an NMI is 0x2 by default; a different cause may be specified using parameter "ecode_nmi" or applied using optional input port "nmi_cause" if required.
If parameter "rnmi_version" is not "none", resumable NMIs are supported, managed by additional CSRs

"mnscratch", "mnepc", "mncause" and "mnstatus", following the indicated version of the Resumable NMI extension proposal. In this variant, "rnmi_version" is "0.2.1".

3.1.10 WFI

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).

3.1.11 cycle CSR

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 cause Illegal Instruction traps.

3.1.12 time CSR

The "time" CSR is not implemented in this variant and reads of it will cause Illegal Instruction traps. Set parameter "time_undefined" to False to instead specify that "time" is implemented.

3.1.13 instret CSR

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 cause Illegal Instruction traps.

3.1.14 hpmcounter CSRs

"hpmcounter" CSRs are implemented in this variant. Set parameter "hpmcounter_undefined" to True to instead specify that "hpmcounter" CSRs are unimplemented and reads of them should cause Illegal Instruction traps.

3.1.15 Virtual Memory

This variant supports address translation modes 0 (bare) and 8 (Sv39). Use parameter "Sv_modes" to specify a bit mask of different implemented modes if required; for example, setting "Sv_modes" to (1<<0)+(1<<8) indicates that mode 0 (bare) and mode 8 (Sv39) are implemented. These indices correspond to writable values in the satp.MODE CSR field.
A 0-bit ASID is implemented. Use parameter "ASID_bits" to specify a different implemented ASID size 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.

3.1.16 Unaligned Accesses

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.

3.1.17 PMP

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.

3.1.18 LR/SC Granule

LR/SC instructions are implemented with a 64-byte reservation granule. A different granule size may be specified using parameter "lr_sc_grain".

3.1.19 Compressed Extension

Standard compressed instructions are present in this variant.
Parameter Zcea_version is used to specify the version of Zcea instructions present. By default, Zcea_version is set to "none" in this variant. Updates to this parameter require a commercial product license.
Parameter Zceb_version is used to specify the version of Zceb instructions present. By default, Zceb_version is set to "none" in this variant. Updates to this parameter require a commercial product license.
Parameter Zcee_version is used to specify the version of Zcee instructions present. By default, Zcee_version is set to "none" in this variant. Updates to this parameter require a commercial product license.

3.1.20 Floating Point Features

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".

3.1.21 Privileged Architecture

This variant implements the Privileged Architecture with version specified in the References section of this document. Note that parameter "priv_version" can be used to select the required architecture version; see the following sections for detailed information about differences between each supported version.

3.1.22 Legacy Version 1.10

1.10 version of May 7 2017.

3.1.23 Version 20190608

Stable 1.11 version of June 8 2019, with these changes compared to version 1.10:
- mcountinhibit CSR defined;
- pages are never executable in Supervisor mode if page table entry U bit is 1;
- mstatus.TW is writable if any lower-level privilege mode is implemented (previously, it was just if Supervisor mode was implemented);

3.1.24 Version master

Unstable master version corresponding to evolving 1.12 specification, with these changes compared to version 20190608:
- mstatush, mseccfg, mseccfgh, menvcfg, menvcfgh, senvcfg, henvcfg, henvcfgh and mconfigptr CSRs defined;
- xret instructions clear mstatus.MPRV when leaving Machine mode if new mode is less privileged than M-mode;
- maximum number of PMP registers increased to 64;
- data endian is now configurable.

3.1.25 Unprivileged Architecture

This variant implements the Unprivileged Architecture with version specified in the References section of this document. Note that parameter "user_version" can be used to select the required architecture version; see the following sections for detailed information about differences between each supported version.

3.1.26 Legacy Version 2.2

2.2 version of May 7 2017.

3.1.27 Version 20191213

Stable 20191213-Base-Ratified version of December 13 2019, with these changes compared to version 2.2:
- floating point fmin/fmax instruction behavior modified to comply with IEEE 754-201x.
- numerous other optional behaviors can be separately enabled using Z-prefixed parameters.

3.1.28 Other Extensions

Other extensions that can be configured are described in this section.

3.1.29 Zmmul

Parameter "Zmmul" is 0 on this variant, meaning that all multiply and divide instructions are implemented. if "Zmmul" is set to 1 then multiply instructions are implemented but divide and remainder instructions are not implemented.

3.1.30 Zicsr

Parameter "Zicsr" is 1 on this variant, meaning that standard CSRs and CSR access instructions are implemented. if "Zicsr" is set to 0 then standard CSRs and CSR access instructions are not implemented and an alternative scheme must be provided as a processor extension.

3.1.31 Zifencei

Parameter "Zifencei" is 1 on this variant, meaning that the fence.i instruction is implemented (but treated as a NOP by the model). if "Zifencei" is set to 0 then the fence.i instruction is not implemented.

3.1.32 Zicbom

Parameter "Zicbom" is 0 on this variant, meaning that code block management instructions are undefined. if "Zicbom" is set to 1 then code block management instructions cbo.clean, cbo.flush and cbo.inval are defined.
If Zicbom is present, the cache block size is given by parameter "cmomp_bytes". The instructions may cause traps if used illegally but otherwise are NOPs in this model.

3.1.33 Zicbop

Parameter "Zicbop" is 0 on this variant, meaning that prefetch instructions are undefined. if "Zicbop" is set to 1 then prefetch instructions prefetch.i, prefetch.r and prefetch.w are defined (but behave as NOPs in this model).

3.1.34 Zicboz

Parameter "Zicboz" is 0 on this variant, meaning that the cbo.zero instruction is undefined. if "Zicboz" is set to 1 then the cbo.zero instruction is defined.
If Zicboz is present, the cache block size is given by parameter "cmoz_bytes".

3.1.35 Svnapot

Parameter "Svnapot_page_mask" is 0x0 on this variant, meaning that NAPOT Translation Contiguity is not implemented. if "Svnapot_page_mask" is non-zero then NAPOT Translation Contiguity is enabled for page sizes indicated by that mask value when page table entry bit 63 is set.
If Svnapot is present, "Svnapot_page_mask" is a mask of page sizes for which contiguous pages can be created. For example, a value of 0x10000 implies that 64KiB contiguous pages are supported.

3.1.36 Svpbmt

Parameter "Svpbmt" is 0 on this variant, meaning that page-based memory types are not implemented. if "Svpbmt" is set to 1 then page-based memory types are indicated by page table entry bits 62:61.
Note that except for their effect on Page Faults, the encoded memory types do not alter the behavior of this model, which always implements strongly-ordered non-cacheable semantics.

3.1.37 Svinval

Parameter "Svinval" is 0 on this variant, meaning that fine-grained address-translation cache invalidation instructions are not implemented. if "Svinval" is set to 1 then fine-grained address-translation cache invalidation instructions sinval.vma, sfence.w.inval and sfence.inval.ir are implemented.

3.1.38 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.

3.1.39 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

3.1.40 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.

3.1.41 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.

3.1.42 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.

3.1.43 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.

3.1.44 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.

3.1.45 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.

3.1.46 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.

3.1.47 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).

3.1.48 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.

3.1.49 Integration Support

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

3.1.50 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.

3.1.51 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.

3.1.52 Page Table Walk Introspection

Artifact register "PTWStage" shows the active page table translation stage (0 if no stage active, 1 if HS-stage active, 2 if VS-stage active and 3 if G-stage active). This register is visibly non-zero only in a memory access callback triggered by a page table walk event.
Artifact register "PTWInputAddr" shows the input address of active page table translation. This register is visibly non-zero only in a memory access callback triggered by a page table walk event.
Artifact register "PTWLevel" shows the active level of page table translation (corresponding to index variable "i" in the algorithm described by Virtual Address Translation Process in the RISC-V Privileged Architecture specification). This register is visibly non-zero only in a memory access callback triggered by a page table walk event.

3.1.53 Artifact Register "fflags_i"

If parameter "enable_fflags_i" is True, an 8-bit artifact register "fflags_i" is added to the model. This register shows the floating point flags set by the current instruction (unlike the standard "fflags" CSR, in which the flag bits are sticky).

3.1.54 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.

3.1.55 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.

3.1.56 References

The Model details are based upon the following specifications:
RISC-V Instruction Set Manual, Volume I: User-Level ISA (User Architecture Version 20191213)
RISC-V Instruction Set Manual, Volume II: Privileged Architecture (Privileged Architecture Version Ratified-IMFDQC-and-Priv-v1.11)
SiFive U54-MC Core Complex Manual v1p0

3.1.57 SiFive-Specific Extensions

SiFive processors can add various custom extensions to the basic RISC-V architecture. This model implements the following:

3.1.58 SiFive-Specific CSRs

This section describes SiFive-specific CSRs implemented by this variant. Refer to SiFive reference documentation for more information.

3.1.59 bpm (csr num 0x7c0)

Reading and writing the SiFive custom M-Mode Branch Prediction Mode CSR is supported. Since this register controls only micro-architectural behavior, which is not modeled, the setting of this register has no effect.

3.1.60 featureDisable (csr num 0x7c1)

Reading and writing the SiFive custom M-Mode Feature Disable CSR is supported. Since this register controls only micro-architectural behavior, which is not modeled, the setting of this register has no effect, and all fields are hardwired to 0.

3.1.61 SiFive-Specific Instructions

This section describes SiFive-specific instructions implemented by this variant. Refer to SiFive reference documentation for more information.
CFLUSH.D.L1
CDISCARD.D.L1
CEASE

3.2 Instance Parameters

Several parameters can be specified when a processor is instanced in a platform. For this processor instance 'rv64_cpu' it has been instanced with the following parameters:

Table 6. Processor Instance 'rv64_cpu' Parameters (Configurations)

Parameter	Value	Description
simulateexceptions	simulateexceptions	Causes the processor simulate exceptions instead of halting
mips	200	The nominal MIPS for the processor

Table 7. Processor Instance 'rv64_cpu' Parameters (Attributes)

Parameter Name	Value	Type
variant	U54	enum
local_int_num	48	Uns32
numHarts	17	Uns32
wfi_is_nop	0	bool
mstatus_FS	1	Uns32

3.3 Memory Map for processor 'rv64_cpu' bus: 'rv64_bus'

Processor instance 'rv64_cpu' is connected to bus 'rv64_bus' using master port 'INSTRUCTION'.
Processor instance 'rv64_cpu' is connected to bus 'rv64_bus' using master port 'DATA'.

Table 8. Memory Map ('rv64_cpu' / 'rv64_bus' [width: 38])

Lo Address	Hi Address	Instance	Component
0x0	0xFFFFFFF	rv64_ram	ram
0x20000000	0x3FFFFFFF	rv64_bridge	bridge

Table 9. Bridged Memory Map ('rv64_cpu' / 'rv64_bridge' / 'shared_bus' [width: 38])

Lo Address	Hi Address	Instance	Component
0x20000000	0x2FFFFFFF	shared_ram	ram
0x30010000	0x3001001B	uart0	UART
0x30011000	0x3001101B	uart1	UART
0x30012000	0x3001201B	uart2	UART
0x30013000	0x3001301B	uart3	UART
0x30014000	0x3001401B	uart4	UART
0x30015000	0x3001501B	uart5	UART
0x30016000	0x3001601B	uart6	UART
0x30017000	0x3001701B	uart7	UART
0x30018000	0x3001801B	uart8	UART
0x30019000	0x3001901B	uart9	UART
0x3001A000	0x3001A01B	uart10	UART
0x3001B000	0x3001B01B	uart11	UART
0x3001C000	0x3001C01B	uart12	UART
0x3001D000	0x3001D01B	uart13	UART
0x3001E000	0x3001E01B	uart14	UART
0x3001F000	0x3001F01B	uart15	UART
0x30020000	0x3002001B	uart16	UART
0x30021000	0x3002101B	uart17	UART
0x30030000	0x300300FF	cpuload	ram
0x30040000	0x30040003	usecCount	usecCounter
0x30100000	0x3010BFFF	clint	CLINT

3.4 Net Connections to processor: 'rv64_cpu'

Table 10. Processor Net Connections ('rv64_cpu')

Net Port	Net	Instance	Component
hart0_MTimerInterrupt	MTimerInterrupt0	clint	CLINT
hart0_MSWInterrupt	MSWInterrupt0	clint	CLINT
hart1_MTimerInterrupt	MTimerInterrupt1	clint	CLINT
hart1_MSWInterrupt	MSWInterrupt1	clint	CLINT
hart2_MTimerInterrupt	MTimerInterrupt2	clint	CLINT
hart2_MSWInterrupt	MSWInterrupt2	clint	CLINT
hart3_MTimerInterrupt	MTimerInterrupt3	clint	CLINT
hart3_MSWInterrupt	MSWInterrupt3	clint	CLINT
hart4_MTimerInterrupt	MTimerInterrupt4	clint	CLINT
hart4_MSWInterrupt	MSWInterrupt4	clint	CLINT
hart5_MTimerInterrupt	MTimerInterrupt5	clint	CLINT
hart5_MSWInterrupt	MSWInterrupt5	clint	CLINT

hart6_MTimerInterrupt	MTimerInterrupt6	clint	CLINT
hart6_MSWInterrupt	MSWInterrupt6	clint	CLINT
hart7_MTimerInterrupt	MTimerInterrupt7	clint	CLINT
hart7_MSWInterrupt	MSWInterrupt7	clint	CLINT
hart8_MTimerInterrupt	MTimerInterrupt8	clint	CLINT
hart8_MSWInterrupt	MSWInterrupt8	clint	CLINT
hart9_MTimerInterrupt	MTimerInterrupt9	clint	CLINT
hart9_MSWInterrupt	MSWInterrupt9	clint	CLINT
hart10_MTimerInterrupt	MTimerInterrupt10	clint	CLINT
hart10_MSWInterrupt	MSWInterrupt10	clint	CLINT
hart11_MTimerInterrupt	MTimerInterrupt11	clint	CLINT
hart11_MSWInterrupt	MSWInterrupt11	clint	CLINT
hart12_MTimerInterrupt	MTimerInterrupt12	clint	CLINT
hart12_MSWInterrupt	MSWInterrupt12	clint	CLINT
hart13_MTimerInterrupt	MTimerInterrupt13	clint	CLINT
hart13_MSWInterrupt	MSWInterrupt13	clint	CLINT
hart14_MTimerInterrupt	MTimerInterrupt14	clint	CLINT
hart14_MSWInterrupt	MSWInterrupt14	clint	CLINT
hart15_MTimerInterrupt	MTimerInterrupt15	clint	CLINT
hart15_MSWInterrupt	MSWInterrupt15	clint	CLINT
hart16_MTimerInterrupt	MTimerInterrupt16	clint	CLINT
hart16_MSWInterrupt	MSWInterrupt16	clint	CLINT

4.0 Peripheral Instances

4.1 Peripheral [imperas.ovpworld.org/peripheral/frameBuffer/1.0] instance: fb

4.1.1 Licensing

Open Source Apache 2.0

4.1.2 Description

Provides Frame Buffer output.
Supports frame buffer formats used in Neural Network demo application i.e. RGB565 16-bit float for Alexnet application.

There are no configuration options set for this peripheral instance.

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

4.2.1 Licensing

Open Source Apache 2.0

4.2.2 Description

Sifive UART

4.2.3 Limitations

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

4.2.4 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 'uart0'

Attribute	Value	Type	Expression
ychars	30	uns32	
console	1	boolean	
finishOnDisconnect	1	boolean	

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

4.3.1 Licensing

Open Source Apache 2.0

4.3.2 Description

Sifive UART

4.3.3 Limitations

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

4.3.4 Reference

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 'uart1'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.4 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart2

4.4.1 Licensing

Open Source Apache 2.0

4.4.2 Description

Sifive UART

4.4.3 Limitations

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

4.4.4 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 'uart2'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.5 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart3

4.5.1 Licensing

Open Source Apache 2.0

4.5.2 Description

Sifive UART

4.5.3 Limitations

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

4.5.4 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 'uart3'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.6 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart4

4.6.1 Licensing

Open Source Apache 2.0

4.6.2 Description

Sifive UART

4.6.3 Limitations

When simulatebaud parameter is set to true baud rate delays are modeled for receive only, not transmit.

Data always sent immediately.

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

Table 15. Configuration options (attributes) set for instance 'uart4'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.7 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart5

4.7.1 Licensing
Open Source Apache 2.0

4.7.2 Description
Sifive UART

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

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

Table 16. Configuration options (attributes) set for instance 'uart5'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.8 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart6

4.8.1 Licensing
Open Source Apache 2.0

4.8.2 Description
Sifive UART

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

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

Table 17. Configuration options (attributes) set for instance 'uart6'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.9 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart7

4.9.1 Licensing
Open Source Apache 2.0

4.9.2 Description
Sifive UART

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

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

Table 18. Configuration options (attributes) set for instance 'uart7'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.10 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart8

4.10.1 Licensing
Open Source Apache 2.0

4.10.2 Description
Sifive UART

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

4.10.4 Reference
SiFive Freedom U540-C000 Manual FU540-C000-v1.0.pdf

(https://www.sifive.com/documentation/chips/freedom-u540-c000-manual)

Table 19. Configuration options (attributes) set for instance 'uart8'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.11 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart9

4.11.1 Licensing
Open Source Apache 2.0

4.11.2 Description
Sifive UART

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

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

Table 20. Configuration options (attributes) set for instance 'uart9'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.12 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart10

4.12.1 Licensing
Open Source Apache 2.0

4.12.2 Description
Sifive UART

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

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

Table 21. Configuration options (attributes) set for instance 'uart10'

Attribute	Value	Type	Expression
ychars	30	uns32	
console	1	boolean	
finishOnDisconnect	1	boolean	

4.13 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart11

4.13.1 Licensing
Open Source Apache 2.0

4.13.2 Description
Sifive UART

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

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

Table 22. Configuration options (attributes) set for instance 'uart11'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.14 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart12

4.14.1 Licensing
Open Source Apache 2.0

4.14.2 Description
Sifive UART

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

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

Table 23. Configuration options (attributes) set for instance 'uart12'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.15 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart13

4.15.1 Licensing
Open Source Apache 2.0

4.15.2 Description
Sifive UART

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

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

Table 24. Configuration options (attributes) set for instance 'uart13'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.16 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart14

4.16.1 Licensing
Open Source Apache 2.0

4.16.2 Description
Sifive UART

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

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

Table 25. Configuration options (attributes) set for instance 'uart14'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.17 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart15

4.17.1 Licensing

Open Source Apache 2.0

4.17.2 Description

Sifive UART

4.17.3 Limitations

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

4.17.4 Reference

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

Table 26. Configuration options (attributes) set for instance 'uart15'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.18 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart16

4.18.1 Licensing

Open Source Apache 2.0

4.18.2 Description

Sifive UART

4.18.3 Limitations

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

4.18.4 Reference

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

Table 27. Configuration options (attributes) set for instance 'uart16'

Attribute	Value	Type	Expression
ychars	7	uns32	

4.19 Peripheral [sifive.ovpworld.org/peripheral/UART/1.0] instance: uart17

4.19.1 Licensing
Open Source Apache 2.0

4.19.2 Description
Sifive UART

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

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

Table 28. Configuration options (attributes) set for instance 'uart17'

Attribute	Value	Type	Expression
ychars	20	uns32	
console	1	boolean	
finishOnDisconnect	1	boolean	

4.20 Peripheral [imperas.ovpworld.org/peripheral/usecCounter/1.0] instance: usecCount

4.20.1 Description
Provides a register with a Microsecond Counter

4.20.2 Limitations
Changes at simulator resolution

4.20.3 Reference
None

4.20.4 Licensing
Open Source Apache 2.0

There are no configuration options set for this peripheral instance.

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

4.21.1 Licensing
Open Source Apache 2.0

4.21.2 Description

SiFive-compatabile 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.

4.21.3 Reference

Various SiFive Core Complex manuals, e.g. SiFive U54 Core Complex Manual (https://sifive.cdn.prismic.io/sifive/a07d1a9a-2cb8-4cf5-bb75-5351888ea2e1_u54_core_complex_manual_21G2.pdf)

Table 29. Configuration options (attributes) set for instance 'clint'

Attribute	Value	Type	Expression
num_harts	17	Uns32	
clockMHz	1.0	double	

5.0 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.

6.0 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.

7.0 Simulating Software

7.1 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.

7.2 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.

7.3 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.

7.4 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.

7.5 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.

7.6 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.

7.7 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.

8.0 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.

8.1 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.

*8.2 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.

8.3 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.

*8.4 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.

8.5 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.

8.6 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.

9.0 Modifying the Platform / Module

9.1 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.

9.2 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.

9.3 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.

9.4 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.

9.5 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.

10.0 Available Virtual Platforms

Table 30. 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 31. 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 32. 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 33. 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

#