



## OVP Guide to Using Processor Models

### Model Specific Information for variant ARM\_ARM1020E

#### Imperas Software Limited

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



Author	Imperas Software Limited
Version	0.5
Filename	OVP_Model_Specific_Information_arm_ARM1020E.pdf
Created	27 February 2018
Status	OVP Standard Release

## **Copyright Notice**

All rights reserved. This software and documentation contain information that is the property of Imperas Software Limited. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Imperas Software Limited, or as expressly provided by the license agreement.

## **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 reader's 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](http://OVPworld.org).

Table of Contents

1 Overview.....	5
1.1 Description.....	5
1.2 Licensing.....	5
1.3 Limitations.....	5
1.4 Verification.....	6
1.5 Features.....	6
1.5.1 Core Features.....	6
1.5.2 Memory System.....	6
1.6 Debug Mask.....	6
1.7 AArch32 Unpredictable Behavior.....	6
1.7.1 Equal Target Registers.....	6
1.7.2 Floating Point Load/Store Multiple Lists.....	7
1.7.3 Floating Point VLD[2-4]/VST[2-4] Range Overflow.....	7
1.7.4 If-Then (IT) Block Constraints.....	7
1.7.5 Use of R13.....	7
1.7.6 Use of R15.....	7
1.8 Integration Support.....	7
1.8.1 Memory Transaction Query.....	7
1.8.2 Page Table Walk Query.....	7
1.8.3 Artifact Page Table Walks.....	8
1.8.4 MMU and Page Table Walk Events.....	8
1.8.5 Artifact Address Translations.....	8
1.8.6 Halt Reason Introspection.....	8
1.8.7 System Register Access Monitor.....	8
1.8.8 System Register Implementation.....	9
2 Configuration.....	9
2.1 Location.....	9
2.2 GDB Path.....	9
2.3 Semi-Host Library.....	9
2.4 Processor Endian-ness.....	9
2.5 QuantumLeap Support.....	9
2.6 Processor ELF Code.....	9
3 Other Variants in this Model.....	9
4 Bus Ports.....	12
5 Net Ports.....	12
6 FIFO Ports.....	12
7 Parameters.....	12
8 Execution Modes.....	13
9 Exceptions.....	14
10 Hierarchy of the model.....	15
10.1 Level 1: CPU.....	15
11 Model Commands.....	16

11.1 Level 1: CPU.....	16
11.1.1 debugflags.....	16
11.1.2 dumpTLB.....	16
11.1.3 isync.....	16
11.1.4 itrace.....	16
11.1.5 validateTLB.....	17
12 Registers.....	17
12.1 Level 1: CPU.....	17
12.1.1 Core.....	17
12.1.2 Control.....	17
12.1.3 User.....	18
12.1.4 FIQ.....	18
12.1.5 IRQ.....	18
12.1.6 Supervisor.....	18
12.1.7 Undefined.....	19
12.1.8 Abort.....	19
12.1.9 Coprocessor_32_bit.....	19
12.1.10 Integration_support.....	20

## 1 Overview

This document provides the details of an OVP Fast Processor Model variant.

OVP Fast Processor Models are written in C and provide a C API for use in C based platforms. The models also provide a native interface for use in SystemC TLM2 platforms.

The models are written using the OVP VMI API that provides a Virtual Machine Interface 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.

Most 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 model families are validated using technology provided by the processor IP owners.

There is a companion document (OVP Guide to Using Processor Models) which explains the general concepts of OVP Fast Processor Models and their use. It is downloadable from the OVPworld website documentation pages.

### 1.1 Description

ARM Processor Model

### 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.

### 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.

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.

### ***1.4 Verification***

Models have been extensively tested by Imperas. ARM9 models have been successfully used by customers to simulate Linux and Nucleus on ArmIntegrator virtual platforms.

### ***1.5 Features***

#### ***1.5.1 Core Features***

Thumb instructions are supported.

#### ***1.5.2 Memory System***

FCSE extension is implemented.

VMSA address translation is implemented.

### ***1.6 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 0x004: enable debugging of MMU/MPU mappings

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).

All other bits in the debug bitmask are reserved and must not be set to non-zero values.

### ***1.7 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.

#### ***1.7.1 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.

### ***1.7.2 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**.

### ***1.7.3 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).

### ***1.7.4 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**.

### ***1.7.5 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.

### ***1.7.6 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 "unpredictable" 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 is "execute".

## ***1.8 Integration Support***

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

### ***1.8.1 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.

### ***1.8.2 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. Registers PTWAddressL0-PTWAddressL3 record the addresses of level 0 to level 3 entries read, respectively, and register PTWAddressValid is a bitmask indicating which address registers contain valid data: for example, the value 0xe indicates that PTWAddressL1-PTWAddressL3 are valid but PTWAddressL0 is not. Registers PTWValueL0-PTWValueL3 contain entry values read at level 0 to level 3. Register PTWInput contains the input address that starts a walk and Register PTWOutput contains the result address (valid only if the page table walk completes). Register PTWValueValid is a bitmask indicating which value registers contain valid data: bits 0-3 indicate PTWValueL0-PTWValueL3, respectively, bit 4 indicates PTWBase, bit 5 indicates PTWInput and bit 6 indicates PTWOutput.

### ***1.8.3 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.

### ***1.8.4 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).

### ***1.8.5 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.

### ***1.8.6 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.

### ***1.8.7 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`.

### ***1.8.8 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 Configuration**

### ***2.1 Location***

The model source and object file is found in the VLNV tree at:

[arm.ovpworld.org/processor/arm/1.0](http://arm.ovpworld.org/processor/arm/1.0)

### ***2.2 GDB Path***

The default GDB for this model is found at:

`$IMPERAS_HOME/lib/$IMPERAS_ARCH/gdb/arm-none-eabi-gdb`

### ***2.3 Semi-Host Library***

The default semi-host library file is found in the VLNV tree at :

[arm.ovpworld.org/semihosting/armNewlib/1.0](http://arm.ovpworld.org/semihosting/armNewlib/1.0)

### ***2.4 Processor Endian-ness***

This model can be set to either endian-ness (normally by a pin, or the ELF code).

### ***2.5 QuantumLeap Support***

This processor is qualified to run in a QuantumLeap enabled simulator.

### ***2.6 Processor ELF Code***

The ELF code supported by this model is: `0x28`

## **3 Other Variants in this Model**

Table 1. All variants in this model

<b>Variant</b>
ARMv4T
ARMv4xM
ARMv4
ARMv4TxM
ARMv5xM
ARMv5
ARMv5TxM
ARMv5T
ARMv5TExP
ARMv5TE
ARMv5TEJ
ARMv6
ARMv6K
ARMv6T2
ARMv6KZ
ARMv7
ARM7TDMI
ARM7EJ-S
ARM720T
ARM920T
ARM922T
ARM926EJ-S
ARM940T
ARM946E
ARM966E
ARM968E-S
ARM1020E
ARM1022E
ARM1026EJ-S
ARM1136J-S
ARM1156T2-S
ARM1176JZ-S
Cortex-R4
Cortex-R4F
Cortex-A5UP
Cortex-A5MPx1
Cortex-A5MPx2
Cortex-A5MPx3
Cortex-A5MPx4
Cortex-A8

Cortex-A9UP
Cortex-A9MPx1
Cortex-A9MPx2
Cortex-A9MPx3
Cortex-A9MPx4
Cortex-A7UP
Cortex-A7MPx1
Cortex-A7MPx2
Cortex-A7MPx3
Cortex-A7MPx4
Cortex-A15UP
Cortex-A15MPx1
Cortex-A15MPx2
Cortex-A15MPx3
Cortex-A15MPx4
Cortex-A17MPx1
Cortex-A17MPx2
Cortex-A17MPx3
Cortex-A17MPx4
AArch32
AArch64
Cortex-A32MPx1
Cortex-A32MPx2
Cortex-A32MPx3
Cortex-A32MPx4
Cortex-A35MPx1
Cortex-A35MPx2
Cortex-A35MPx3
Cortex-A35MPx4
Cortex-A53MPx1
Cortex-A53MPx2
Cortex-A53MPx3
Cortex-A53MPx4
Cortex-A55MPx1
Cortex-A55MPx2
Cortex-A55MPx3
Cortex-A55MPx4
Cortex-A57MPx1
Cortex-A57MPx2
Cortex-A57MPx3
Cortex-A57MPx4
Cortex-A72MPx1

Cortex-A72MPx2
Cortex-A72MPx3
Cortex-A72MPx4
Cortex-A73MPx1
Cortex-A73MPx2
Cortex-A73MPx3
Cortex-A73MPx4
Cortex-A75MPx1
Cortex-A75MPx2
Cortex-A75MPx3
Cortex-A75MPx4
MultiCluster

## 4 Bus Ports

Table 2. Bus Ports

Type	Name	min	max
master (initiator)	INSTRUCTION	32	32
master (initiator)	DATA	32	32

## 5 Net Ports

Table 3. Net Ports

Name	Type	Description
reset	input	Processor reset, active high
fiq	input	FIQ interrupt, active high (negation of nFIQ)
irq	input	IRQ interrupt, active high (negation of nIRQ)
sei	input	System error interrupt, active high

## 6 FIFO Ports

No FIFO Ports in this model.

## 7 Parameters

Table 4. Parameters that can be set in the model, type: CPU

Name	Type	Description
verbose	Boolean	Specify verbosity of output
showHiddenRegs	Boolean	Show hidden registers during register tracing
UAL	Boolean	Disassemble using UAL syntax
enableSystemBus	Boolean	Add 32-bit artifact System bus port, allowing system registers to be externally implemented
enableSystemMonitorBus	Boolean	Add 32-bit artifact SystemMonitor bus port, allowing system register accesses to be externally monitored

compatibility	Enumeration	Specify compatibility mode ISA=0 gdb=1 nopSVC=2
unpredictable	Enumeration	Specify unpredictable instruction behavior (undefined, nop, raz_wi, execute or assert) undefined=0 nop=1 raz_wi=2 execute=3 assert=4
override_debugMask	Uns32	Specifies debug mask, enabling debug output for model components
override_fcsePresent	Boolean	Specifies that FCSE is present (if true)
override_SCTLR_V	Boolean	Override SCTLR.V with the passed value (enables high vectors)
override_SCTLR_CP15BEN_Present	Boolean	Enable ARMv7 SCTLR.CP15BEN bit (CP15 barrier enable)
override_MIDR	Uns32	Override MIDR/MIDR_EL1 register
override_CTR	Uns32	Override CTR/CTR_EL0 register
override_TLBTR	Uns32	Override TLBTR register
override_CLIDR	Uns32	Override CLIDR/CLIDR_EL1 register
override_AIDR	Uns32	Override AIDR/AIDR_EL1 register
override_ERG	Uns32	Specifies exclusive reservation granule
override_STRoffsetPC12	Boolean	Specifies that STR/STR of PC should do so with 12:byte offset from the current instruction (if true), otherwise an 8:byte offset is used
override_fcseRequiresMMU	Boolean	Specifies that FCSE is active only when MMU is enabled (if true)
override_ignoreBadCp15	Boolean	Specifies whether invalid coprocessor 15 access should be ignored (if true) or cause Invalid Instruction exceptions (if false)
override_SGIDisable	Boolean	Override whether GIC SGIs may be disabled (if true) or are permanently enabled (if false)
override_condUndefined	Boolean	Force undefined instructions to take Undefined Instruction exception even if they are conditional
override_deviceStrongAligned	Boolean	Force accesses to Device and Strongly Ordered regions to be aligned
override_Control_V	Boolean	Override SCTLR.V with the passed value (deprecated, use override_SCTLR_V)
override_MainId	Uns32	Override MIDR register (deprecated, use override_MIDR)
override_CacheType	Uns32	Override CTR register (deprecated, use override_CTR)
override_TLBType	Uns32	Override TLBTR register (deprecated, use override_TLBTR)

## 8 Execution Modes

Table 5. CPU modes implemented in the model, type: CPU

<b>Name</b>	<b>Code</b>
User	16
FIQ	17
IRQ	18
Supervisor	19
Abort	23
Undefined	27
System	31

## 9 Exceptions

Table 6. Exceptions handled by the model, type: CPU

<b>Name</b>	<b>Code</b>
Reset	0
Undefined	1
SupervisorCall	2
PrefetchAbort	5
DataAbort	6
IRQ	8
FIQ	9

## 10 Hierarchy of the model

A CPU core may allow the user to configure it to instance many processors of a Symmetrical Multi Processor (SMP). A CPU core may also have sub elements within a processor, for example hardware threading blocks.

OVP processor models can be written to include SMP blocks and to have many levels of hierarchy.

Some OVP CPU models may have a fixed hierarchy, and some may be configured by settings in a configuration register. Please see the register definitions of this model.

This model documentation shows the settings and hierarchy of the default settings for this model variant.

### 10.1 Level 1: CPU

This level in the model hierarchy has 5 commands.

This level in the model hierarchy has 10 register groups:

Table 7. Register groups

Group name	Registers
Core	16
Control	3
User	7
FIQ	8
IRQ	3
Supervisor	3
Undefined	3
Abort	3
Coprocessor_32_bit	33
Integration_support	22

This level in the model hierarchy has no children.

## 11 Model Commands

### 11.1 Level 1: CPU

#### 11.1.1 *debugflags*

show or modify the processor debug flags

Table 8. debugflags command arguments

Argument	Type	Description
-get	Boolean	print current processor flags value
-set	Int32	new processor flags (only flags 0x000003e4 can be modified)

#### 11.1.2 *dumpTLB*

report TLB contents

Table 9. dumpTLB command arguments

Argument	Type	Description
-all	Boolean	show the contents of all TLBs (if False, show just the current TLB)

#### 11.1.3 *isync*

specify instruction address range for synchronous execution

Table 10. isync command arguments

Argument	Type	Description
-addresshi	Uns64	end address of synchronous execution range
-addresslo	Uns64	start address of synchronous execution range

#### 11.1.4 *itrace*

enable or disable instruction tracing

Table 11. itrace command arguments

Argument	Type	Description
-after	Uns64	apply after this many instructions
-enable	Boolean	enable instruction tracing
-instructioncount	Boolean	include the instruction number in each trace
-off	Boolean	disable instruction tracing
-on	Boolean	enable instruction tracing
-registerchange	Boolean	show registers changed by this instruction
-registers	Boolean	show registers after each trace



### 11.1.5 *validateTLB*

check TLB contents against page tables in memory and report incoherent entries

Table 12. *validateTLB* command arguments

Argument	Type	Description
-all	Boolean	check all TLBs (if False, validate just the current TLB)
-verbose	Boolean	show all TLB entries (if False, show only incoherent entries)

## 12 Registers

### 12.1 Level 1: CPU

#### 12.1.1 Core

Table 13. Registers at level 1, type: CPU, register group: 'Core'

Name	Bits	Initial value (Hex)		Description
r0	32	0	rw	
r1	32	0	rw	
r2	32	0	rw	
r3	32	0	rw	
r4	32	0	rw	
r5	32	0	rw	
r6	32	0	rw	
r7	32	0	rw	
r8	32	0	rw	
r9	32	0	rw	
r10	32	0	rw	
r11	32	0	rw	frame pointer
r12	32	0	rw	
sp	32	0	rw	stack pointer
lr	32	0	rw	
pc	32	0	rw	program counter

#### 12.1.2 Control

Table 14. Registers at level 1, type: CPU, register group: 'Control'

Name	Bits	Initial value (Hex)		Description
fps	32	0	rw	archaic FPSCR view (for gdb)
cpsr	32	d3	rw	
spsr	32	0	rw	

### 12.1.3 User

Table 15. Registers at level 1, type: CPU, register group: 'User'

Name	Bits	Initial value (Hex)		Description
r8_usr	32	0	rw	
r9_usr	32	0	rw	
r10_usr	32	0	rw	
r11_usr	32	0	rw	
r12_usr	32	0	rw	
sp_usr	32	0	rw	
lr_usr	32	0	rw	

### 12.1.4 FIQ

Table 16. Registers at level 1, type: CPU, register group: 'FIQ'

Name	Bits	Initial value (Hex)		Description
r8_fiq	32	0	rw	
r9_fiq	32	0	rw	
r10_fiq	32	0	rw	
r11_fiq	32	0	rw	
r12_fiq	32	0	rw	
sp_fiq	32	0	rw	
lr_fiq	32	0	rw	
spsr_fiq	32	0	rw	

### 12.1.5 IRQ

Table 17. Registers at level 1, type: CPU, register group: 'IRQ'

Name	Bits	Initial value (Hex)		Description
sp_irq	32	0	rw	
lr_irq	32	0	rw	
spsr_irq	32	0	rw	

### 12.1.6 Supervisor

Table 18. Registers at level 1, type: CPU, register group: 'Supervisor'

Name	Bits	Initial value (Hex)		Description
------	------	---------------------	--	-------------

sp_svc	32	0	rw	
lr_svc	32	0	rw	
spsr_svc	32	0	rw	

### 12.1.7 Undefined

Table 19. Registers at level 1, type: CPU, register group: 'Undefined'

Name	Bits	Initial value (Hex)		Description
sp_undef	32	0	rw	
lr_undef	32	0	rw	
spsr_undef	32	0	rw	

### 12.1.8 Abort

Table 20. Registers at level 1, type: CPU, register group: 'Abort'

Name	Bits	Initial value (Hex)		Description
sp_abt	32	0	rw	
lr_abt	32	0	rw	
spsr_abt	32	0	rw	

### 12.1.9 Coprocessor\_32\_bit

Table 21. Registers at level 1, type: CPU, register group: 'Coprocessor\_32\_bit'

Name	Bits	Initial value (Hex)		Description
CONTEXTIDR	32	0	rw	Context ID
CTR	32	d1b21b2	r-	Cache Type
CleanDCacheLineMVA	32	-	-w	Data Cache Line Clean by VA
CleanDCacheLineSW	32	-	-w	Data Cache Line Clean by Set/Way
CleanInvalDCacheLineMVA	32	-	-w	Data Cache Line Clean and Invalidate by VA
CleanInvalDCacheLineSW	32	-	-w	Data Cache Line Clean and Invalidate by Set/Way
DACR	32	0	rw	Domain Access Control
DCLR	32	fff0	rw	Data Cache Lockdown
DFAR	32	0	rw	Data Fault Address
DFSR	32	0	rw	Data Fault Status
DTLBIALL	32	-	-w	Invalidate Entire Data TLB
DTLBIMVA	32	-	-w	Invalidate Data TLB by VA
DTLBLR	32	0	rw	TLB Lockdown
DataSynchBarrier	32	-	-w	Data Synchronization Barrier

FCSEIDR	32	0	rw	FCSE Process ID
ICLR	32	fff0	rw	Instruction Cache Lockdown
IFAR	32	0	rw	Instruction Fault Address
IFSR	32	0	rw	Instruction Fault Status
ITLBIALL	32	-	-w	Invalidate Entire Instruction TLB
ITLBIMVA	32	-	-w	Invalidate Instruction TLB by VA
ITLBLR	32	0	rw	Instruction TLB Lockdown
InvalDCache	32	-	-w	Invalidate Data Cache
InvalDCacheLineMVA	32	-	-w	Invalidate Data Cache Line by VA
InvalICache	32	-	-w	Invalidate Instruction Cache
InvalICacheLineMVA	32	-	-w	Invalidate Instruction Cache Line by VA
InvalUnified	32	-	-w	Invalidate Unified Cache
MIDR	32	4115a200	r-	Main ID
PrefetchICacheLine	32	-	-w	Prefetch Instruction Cache Line
SCTLR	32	78	rw	System Control
TLBIALL	32	-	-w	Invalidate Entire Unified TLB
TTBR	32	0	rw	Translation Table Base
WaitForInterrupt	32	-	-w	Wait For Interrupt
WaitForInterrupt2	32	-	-w	Wait For Interrupt

### 12.1.10 Integration\_support

Table 22. Registers at level 1, type: CPU, register group: 'Integration\_support'

Name	Bits	Initial value (Hex)		Description
transactPL	32	1	r-	privilege level of current memory transaction
transactAT	32	0	r-	current memory transaction type: PA=1, VA=0
artifactPAR	64	0	r-	result of address translation for artifact write to ATS1CPR etc
PTWBankValid	8	0	r-	bitmask of valid banks (0x01 is stage 1, 0x02 is stage 2, 0x04-0x20 are stage 2 walks initiated by stage 1 level 0-3 entry lookups, respectively)
PTWAddressValid	8	0	r-	bitmask of valid bits for each of PTWAddressL0...PTWAddressL3, PTWBase, PTWInput and PTWOutput in current bank
PTWValueValid	8	0	r-	bitmask of valid bits for each of PTWValueL0...PTWValueL3 in current bank
PTWAddressL0	64	0	r-	current bank PTW address, level 0
PTWAddressL1	64	0	r-	current bank PTW address, level 1
PTWAddressL2	64	0	r-	current bank PTW address, level 2
PTWAddressL3	64	0	r-	current bank PTW address, level 3
PTWValueL0	64	0	r-	current bank PTW value, level 0
PTWValueL1	64	0	r-	current bank PTW value, level 1

PTWValueL2	64	0	r-	current bank PTW value, level 2
PTWValueL3	64	0	r-	current bank PTW value, level 3
PTWBase	64	0	r-	current bank PTW table base address
PTWInput	64	0	r-	current bank PTW input address
PTWOutput	64	0	r-	current bank PTW output address
PTWI_EL1NS	64	-	-w	perform EL1(NS) stage 1 page table walk for fetch, filling PTW query registers
PTWD_EL1NS	64	-	-w	perform EL1(NS) stage 1 page table walk for load/store, filling PTW query registers
PTWI_current	64	-	-w	perform current mode page table walk for fetch, filling PTW query registers
PTWD_current	64	-	-w	perform current mode page table walk for load/store, filling PTW query registers
HaltReason	8	0	r-	bit field indicating halt reason

#