OVP Guide to Using Processor Models

Model Specific Information for variant ARM_ARMv6T2

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_ARMv6T2.pdf
Created	23 February 2018
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 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.

1.4 Verification

Models have been extensively tested by Imperas.

1.5 Features

1.5.1 Core Features

Thumb-2 instructions are supported.

Trivial Jazelle extension is implemented.

1.5.2 Memory System

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 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 "undefined".

1.8 Integration Support

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

1.8.1 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.

1.8.2 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.

1.8.3 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

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

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

Variant
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
AXI_SLVERR	input	AXI external abort type (DECERR=0, SLVERR=1)

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_CLIDR	Uns32	Override CLIDR/CLIDR_EL1 register
override_AIDR	Uns32	Override AIDR/AIDR_EL1 register
override_PFR0	Uns32	Override ID_PFR0/ID_PFR0_EL1 register
override_PFR1	Uns32	Override ID_PFR1/ID_PFR1_EL1 register
override_DFR0	Uns32	Override ID_DFR0/ID_DFR0_EL1 register
override_AFR0	Uns32	Override ID_AFR0/ID_AFR0_EL1 register
override_MMFR0	Uns32	Override ID_MMFR0/ID_MMFR0_EL1 register
override_MMFR1	Uns32	Override ID_MMFR1/ID_MMFR1_EL1 register
override_MMFR2	Uns32	Override ID_MMFR2/ID_MMFR2_EL1 register
override_MMFR3	Uns32	Override ID_MMFR3/ID_MMFR3_EL1 register
override_ISAR0	Uns32	Override ID_ISAR0/ID_ISAR0_EL1 register
override_ISAR1	Uns32	Override ID_ISAR1/ID_ISAR1_EL1 register
override_ISAR2	Uns32	Override ID_ISAR2/ID_ISAR2_EL1 register
override_ISAR3	Uns32	Override ID_ISAR3/ID_ISAR3_EL1 register
override_ISAR4	Uns32	Override ID_ISAR4/ID_ISAR4_EL1 register
override_ISAR5	Uns32	Override ID_ISAR5/ID_ISAR5_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_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_InstructionAttributes0	Uns32	Override ID_ISAR0 register (deprecated, use override_ISAR0)
override_InstructionAttributes1	Uns32	Override ID_ISAR1 register (deprecated, use override_ISAR1)
override_InstructionAttributes2	Uns32	Override ID_ISAR2 register (deprecated, use override_ISAR2)
override_InstructionAttributes3	Uns32	Override ID_ISAR3 register (deprecated, use override_ISAR3)
override_InstructionAttributes4	Uns32	Override ID_ISAR4 register (deprecated, use override_ISAR4)
override_InstructionAttributes5	Uns32	Override ID_ISAR5 register (deprecated, use override_ISAR5)

8 Execution Modes

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

Name	**Code**
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

Name	**Code**
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 3 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	23
Integration_support	3

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 isync

specify instruction address range for synchronous execution

Table 9. isync command arguments

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

11.1.3 itrace

enable or disable instruction tracing

Table 10. 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

12 Registers

12.1 Level 1: CPU

12.1.1 Core

Table 11. 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 12. 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	1d3	rw	
spsr	32	0	rw	

12.1.3 User

Table 13. 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 14. 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 15. 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 16. 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 17. 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 18. 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 19. Registers at level 1, type: CPU, register group: 'Coprocessor_32_bit'

Name	Bits	Initial value (Hex)		Description
CPACR	32	0	rw	Coprocessor Access Control
DBGDIDR	32	0	r-	Debug ID
ID_AFR0	32	0	r-	Auxiliary Feature 0
ID_DFR0	32	0	r-	Debug Feature 0
ID_ISAR0	32	0	r-	Instruction Set Attribute 0
ID_ISAR1	32	0	r-	Instruction Set Attribute 1
ID_ISAR2	32	0	r-	Instruction Set Attribute 2
ID_ISAR3	32	0	r-	Instruction Set Attribute 3
ID_ISAR4	32	0	r-	Instruction Set Attribute 4
ID_ISAR5	32	0	r-	Instruction Set Attribute 5
ID_MMFR0	32	0	r-	Memory Model Feature 0
ID_MMFR1	32	0	r-	Memory Model Feature 1
ID_MMFR2	32	0	r-	Memory Model Feature 2
ID_MMFR3	32	0	r-	Memory Model Feature 3
ID_PFR0	32	0	r-	Processor Feature 0
ID_PFR1	32	0	r-	Processor Feature 1
JIDR	32	0	rw	Jazelle ID
JMCR	32	0	rw	Jazelle Main Configuration
JOSCR	32	0	rw	Jazelle OS Control
MIDR	32	0	r-	Main ID
SCTLR	32	0	rw	System Control
TCMTR	32	0	r-	TCM Type
WFAR	32	0	rw	Watchpoint Fault Address

12.1.10 Integration_support

Table 20. 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
HaltReason	8	0	r-	bit field indicating halt reason

#