

Application Note 52

The ARMulator Configuration File



Document number: ARM DAI 0052A

Issued: January 1998

Copyright Advanced RISC Machines Ltd (ARM) 1998

ENGLAND

Advanced RISC Machines Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN
UK
Telephone: +44 1223 400400
Facsimile: +44 1223 400410
Email: info@arm.com

JAPAN

Advanced RISC Machines K.K.
KSP West Bldg, 3F 300D, 3-2-1 Sakado
Takatsu-ku, Kawasaki-shi
Kanagawa
213 Japan
Telephone: +81 44 850 1301
Facsimile: +81 44 850 1308
Email: info@arm.com

GERMANY

Advanced RISC Machines Limited
Otto-Hahn Str. 13b
85521 Ottobrunn-Riemerling
Munich
Germany
Telephone: +49 89 608 75545
Facsimile: +49 89 608 75599
Email: info@arm.com

USA

ARM USA Incorporated
Suite 5
985 University Avenue
Los Gatos
CA 95030 USA
Telephone: +1 408 399 5199
Facsimile: +1 408 399 8854
Email: info@arm.com

World Wide Web address: <http://www.arm.com>



Proprietary Notice

ARM and the ARM Powered logo are trademarks of Advanced RISC Machines Ltd.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

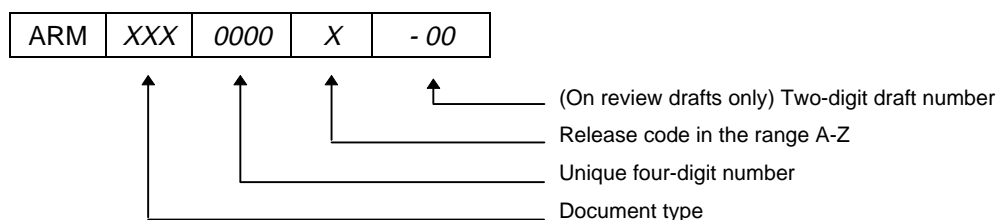
The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties or merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Ltd shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Key

Document Number

This document has a number which identifies it uniquely. The number is displayed on the front page and at the foot of each subsequent page.



Document Status

The document's status is displayed in a banner at the bottom of each page. This describes the document's confidentiality and its information status.

Confidentiality status is one of:

ARM Confidential	Distributable to ARM staff and NDA signatories only
Named Partner Confidential	Distributable to the above and to the staff of named partner companies only
Partner Confidential	Distributable within ARM and to staff of all partner companies
Open Access	No restriction on distribution

Information status is one of:

Advance	Information on a potential product
Preliminary	Current information on a product under development
Final	Complete information on a developed product

Change Log

Issue	Date	By	Change
A	January 1998	SKW	Released



Table of Contents

1 Introduction	2
2 Basics	3
2.1 Tree structure	3
2.2 Built-in types	3
2.3 Programming interface	4
3 File Format	5
3.1 Example	5
3.2 Specifying children	5
3.3 Pre-processing	6
3.4 Searching for ToolConf files	7
4 The armul.cnf ToolConf File	8
4.1 Location and layout	8
4.2 Predefined tags	9
4.3 Header	10
4.4 Operating system	10
4.5 Processors	11
4.6 Memories	14
4.7 Coprocessors	17
4.8 Basic models	17



1 Introduction

ARMulator is part of the ARM Debugger for Windows. For details, refer to the *Software Development Toolkit Reference Guide* (ARM DUI 0041).

ARMulator is a flexible program for emulating ARM-based systems. It can emulate a number of ARM processors, coprocessors and memory systems. The precise configuration of the system is controlled through a configuration file called `armul.cnf`.

A simple description of the configuration file, and what you can do with it, is given in the *Software Development Toolkit User Guide* (ARM DUI 0040). This Application Note gives a more detailed description of the file, its format and uses.

This Application Note describes the ToolConf database, and ToolConf files. `armul.cnf` is such a file. It is read by ARMulator at initialization (and reinitialization) and converted into a ToolConf database.

Note *Reinitialization allows you to change `armul.cnf` and restart the debugger via the "Configure Debugger" dialog, without having to kill and restart the debugger.*

2 Basics

This section describes the ToolConf database system.

`armul.cnf` is a ToolConf configuration file. ToolConf is a module inside ARMulator for dealing with generic configurations. Conceptually it is a tree-structured database consisting of tag/value pairs (“tags” and “values” are strings.) Given a ToolConf database, you can find a value associated with a tag, add or change a value, and so on.

Tags and values are, generally, case-insensitive.

2.1 Tree structure

Each tag can also have another ToolConf database associated with it, called its child. This gives the whole database a tree structure. When a tag lookup is performed on the database, the search can recurse up to the child’s parent, and up the tree until an entry is found.

2.2 Built-in types

The ToolConf module provides calls to look up the value associated with a tag, and to convert that value to a given type automatically. Although no type-checking is performed, either at this call or when the file is read (as no type information is contained in the ToolConf database), this typed interface is used by ARMulator.

The basic set of supported types is:

- String
- Unsigned integer
- Signed integer
- Boolean flag (the full set of permissible values for the “Bool” (or flag) type are shown in **Table 1: Boolean values**.)

True	False
True	False
On	Off
High	Low
Hi	Lo
1	0
T	F

Table 1: Boolean values

Some values, for example memory sizes, clock speeds, and so on, may also be specified in S.I. (*Système Internationale*, or standard metric prefix) notation. For example:

```
ClockSpeed=10MHz
MemorySize=2Gb
```

In these cases only the first letter of the suffix is examined to determine the scaling factor (k, M or G being the possible values, for “kilo”, “Mega” and “Giga”). Whether this is decimal (x1,000, x1,000,000 and x1,000,000,000) or binary (x1,024, x1,048,576 or x1,073,741,824) is context-dependent, and decided by the ARMulator.



2.3 Programming interface

The API for ToolConf is given in the *Software Development Toolkit User Guide* (ARM DUI 0040), and in the file `toolconf.h`, which is provided as part of the ARMulator rebuild kit:

- `armsd/source/toolconf.h` on UNIX systems. (This is located in, for example, `solaris/armsd.tar`)
- `Source/Win32/Armulate/toolconf.h` on Windows NT/95.

3 File Format

This section describes the layout of a ToolConf file, and the pre-processor it uses.

A ToolConf file holds a textual representation of a ToolConf database.

3.1 Example

This is a sample ToolConf file. It is a simple, flat, configuration, with no children.

```
;; This is an example ToolConf file.
; Lines starting ';' are comments. Comments must be on a line
; by themselves.
; All blank lines are ignored
Tag=Value
; This creates a tag in the ToolConf called 'Tag' with value
; 'Value'
Tag = Value
; Whitespace IS important. Creates a tag called 'Tag ' with
; value ' Value'. This is NOT the same tag as 'Tag'
OtherTag
; Creates a tag called 'OtherTag' with no value
Tag=NewValue
; This has no effect. We already have a tag called 'Tag' with
; a value.
OtherTag=OtherValue
; 'OtherTag' has no value, so this assigns 'OtherValue' to
; 'OtherTag'.
```

Note *The rules for white space in tags (and values) may change in future releases. Use of white space in tags should therefore be considered as unpredictable.*

3.2 Specifying children

There are two ways of specifying children in the configuration file. One is more suited to large children, one for smaller children. For example:

```
{ Tag=Value
; Creates a tag called 'Tag' with a child, and the value
; 'Value'. This is the child.
Tag=Value
; Creates a tag called 'Tag' in the child, with value 'Value'
}
OtherTag:Tag=OtherValue
; Creates a tag called 'OtherTag', with a child which has an
; entry called 'Tag' with value 'OtherValue'. 'OtherTag' (if it did
; not already exist) is created with no value.
```

If a child specifier appears more than once, the children are, effectively, merged.

Note *Note that releases prior to 2.11 did not support the ":" syntax. The above example would have been written as:*

```
{ OtherTag
Tag=OtherValue
}
```

The ":" syntax was introduced to remove the need for this lengthy construct.



3.3 Pre-processing

The configuration file can also contain C-like pre-processor directives:

- `#if ... #elif ... #else ... #endif;` and
- `#include`

3.3.1 Conditional

The full `#if ... #elif ... #else ... #endif` syntax is supported, for skipping regions of a ToolConf file. Expressions use tags from the file. For example, the C pre-processor sequence:

```
#define Control True
#if defined(Control) && Control==True
#define ControlIsTrue Yes
#endif
```

would map to the ToolConf sequence:

```
Control=True
#if Control && Control==True
ControlIsTrue=Yes
#endif
```

The “==” (and “!=”) operators perform a case-independent string comparison of the value of the tag and the string given.

The conditional is evaluated, left-to-right, on the contents of the configuration at that point.

Operators

Expressions can contain the standard boolean operators, which use eager-evaluation (that is, the right-hand-side of an expression is not evaluated if the left-hand-side determines its value, as in C).

The complete set of operators is given in **Table 2: Operators in pre-processor expressions**.

Operator	Example	Description
<code>==</code>	<code>Tag==Value</code>	String-equality test (case insensitive)
<code>!=</code>	<code>Tag!=Value</code>	String-inequality test (case insensitive)
<code>(<expression>)</code>	<code>(Tag==Value)</code>	Grouping
<code><no operator></code>	<code>Tag</code>	Defined
<code>!</code>	<code>!Tag</code>	Not defined
<code>&&</code>	<code>Tag==Value && Other==Value</code>	Boolean AND
<code> </code>	<code>Tag==Value Other==Value</code>	Boolean OR

Table 2: Operators in pre-processor expressions

3.3.2 File inclusion

A `#include` directive is also provided which includes another ToolConf file at the point where it is inserted. (The directive is ignored if it is being “skipped”.)

The file is searched for using the algorithm described in **3.4 Searching for ToolConf files** on page 7.

3.4 Searching for ToolConf files

The following algorithm is used to locate a ToolConf file:

- 1 The current working directory.
- Note** *On UNIX systems, this is the directory from which you ran `armsd`. Under Windows 95/NT, this is the directory from which you ran the debugger. If an `armul.cnf` file is not found in this directory, or, if it is invalid, it is loaded from another directory on the current path.*
- 2 The location of the executable (for example `C:\ARM211\BIN`)
 - 3 The path pointed to by the system variable `ARMCONF` (if set)
 - 4 The path pointed to by the system variable `ARMLIB` (if set)
 - 5 *UNIX only:* The path pointed to by the system variable `HOME` (if set)
 - 6 The path pointed to by the system variable `PATH` (if set)
 - 7 *UNIX only:* `/usr/local/lib/arm`
 - 8 *UNIX only:* `/usr/local/lib`
 - 9 *UNIX only:* `/usr/local/arm`
 - 10 *UNIX only:* `/usr/lib/arm`
 - 11 *UNIX only:* `/usr/lib`



4 The armul.cnf ToolConf File

This section describes the layout of the `armul.cnf` ToolConf file.

4.1 Location and layout

`armul.cnf` is located in the BIN directory of your Software Development Toolkit. This is dependent on your installation, but may be something like (for release 2.11):

- Windows 95/NT: `C:\ARM211\BIN`
- SunOS/Solaris/HP-UX: `/usr/local/arm211/bin` (use `which armsd`)

It is recommended that you take a copy of `armul.cnf` before modifying it. See **3.4 Searching for ToolConf files** on page 7.

`armul.cnf` is split into a number of areas.

- Header
- O/S Model
- Processors
- Memories
- Coprocessors
- Models

The order of these regions is not important, but this is the order they occur in `armul.cnf` as supplied.

Note *Future releases may have additional sections or some sections removed/changed.*

4.2 Predefined tags

Before the file is read the ARMulator creates a number of tags of its own, based on the settings you provide to the debugger. These are given in **Table 3: Tags predefined by ARMulator**.

Tag	Description
RDI_*	A tag starting RDI_ is created for each emulator in the DLL. Examples might be RDI_BASIC (the ARM6, ARM7, ARM8 emulator) and RDI_STRONG (the StrongARM emulator).
MEMORY_*	A tag starting MEMORY_ is created for each memory model declared in models.h. (See the <i>Software Development Toolkit User Guide</i> (ARM DUI 0040) for details of models.h.) For example, MEMORY_Flat.
COPROCESSOR_*	A tag starting COPROCESSOR_ is created for each coprocessor model declared in models.h. For example, COPROCESSOR_Validate.
OSMODEL_*	A tag starting OSMODEL_ is created for each operating-system model. For example, OSMODEL_Angel.
MODEL_*	A tag starting MODEL_ is created for each basic model. For example, MODEL_Profiler.
CPUSpeed	Set to the speed specified in the configuration window (or, for armsd, the -clock command-line option). For example, CPUSpeed=30MHz.
MCLK FCLK	Set to the same value as CPUSpeed, if that value is not zero. Otherwise not set at all.
ByteSex	Set to L or B if the debugger is forcing a bytesex on the ARMulator (not set otherwise).
FPE	Set to True or False according to whether the FPE is to be loaded.
MemConfigToLoad	Set to a .map filename, if there is to be one loaded. For example, MemConfigToLoad=C:\Project\Dhrystone\armsd.map.

Table 3: Tags predefined by ARMulator

These are used by pre-processing directives in the file to control the configuration. For example, the MemConfigToLoad tag is checked for, to decide whether to use a memory model that supports map files.



The armul.cnf ToolConf File

4.3 Header

The header exists at the root level in the configuration file, and contains sets of flags used later on by processing directives in the file. This allows changes in configuration which might affect several areas of the file to be controlled from a single place.

All the flags are boolean (see **2.2 Built-in types** on page 3), and are shown in **Table 4: Tags in the header section**.

Tag	Description
Verbose	Models check the <code>Verbose</code> flag to decide whether to report their full configuration to you during initialization. (It is recommended that user-supplied models also do this, to ease debugging.)
TraceMemory	The <code>TraceMemory</code> tag controls whether memory accesses are traced by the Tracer model. See 4.5.2 Plumbing on page 13.
Validate	<code>Validate</code> controls whether ARMulator initializes an ARM validation system. This provides a memory model and/or coprocessor which can generate exceptions, interrupts, and so on. It is used to validate some aspects of the ARMulator model.
WatchPoints	The ARMulator can directly support watchpoints, but to do so impacts on its performance. The <code>WatchPoints</code> switch allows you to make this choice. (It should be set to <code>FALSE</code> when benchmarking code, and <code>TRUE</code> when debugging code.)
UsePageTables	The <code>PageTables</code> model writes page tables to memory, and initializes the cache and MMU on cached processors. (See <i>Application Note 53: Configuring ARM Caches</i> (ARM DAI 0053)) This flag allows you to disable this feature.

Table 4: Tags in the header section

4.4 Operating system

This area is a single child called `OS`, containing all the configuration parameters for the operating-system model.

For example:

```
{ OS
  AngelSWIARM=0x123456
  AngelSWIThumb=0xab
}
```

Note *Future ARMulators will not use this area, as the operating-system model will be a `Basic Model`, so the configuration will exist in the `Models` area. (It is duplicated in both places at present.)*

4.5 Processors

The `Processors` area is a child ToolConf database which contains a full list of processors supported by the ARMulator. You might want to extend this list to package new memory models; for example, if you add a memory model to model some ASIC functions, this could be packaged up as a new processor. The processors in this list are the basis of the list of processors in the ARMulator configuration window (or the list of accepted arguments for the `-processor` option of `armsd`).

With the exception of `Default`, which names the default processor should one not be specified, each entry in the `Processors` region is the name of a processor. For example:

```
{ Processors
{ ARM7TDM
  Processor=ARM7TDM
  Core=ARM7
  ARMulator=BASIC
  Architecture=4T
  ARM7TDMI:Processor=ARM7TDM
}

ARM7TDMI=ARM7TDM
}
```

This declares a pair of processors: ARM7TDM and ARM7TDMI.

Notice that ARM7TDM has a child of its own. Inside this child are the various options for an ARM7TDM. For example, its name (`Processor`) is ARM7TDM, it is based on the ARM7 core, and so on.

It also contains the entry `ARM7TDMI:Processor=ARM7TDM`. This declares a child of ARM7TDM called ARM7TDMI. Because of the way lookups in ToolConf recurse up the tree structure, ARM7TDMI inherits all the features of ARM7TDM. The exception is that it has its own `Processor` tag; its name is ARM7TDMI. (Functionally the ARM7TDMI is identical to the ARM7TDM, so there are no other differences in the configuration.)

The final line (`ARM7TDMI=ARM7TDM`) declares the ARM7TDMI processor, but that it is an alias for ARM7TDM.

When a processor is selected, ARMulator needs to find a configuration for it. The algorithm it uses for this is:

- 1 Set the current section to the `Processors` section.
- 2 Look up the named processor in the current section.
- 3 If the tag has a child, that child is the configuration for the named processor.
- 4 Otherwise, if the tag has a value:
 - a) Look up the value in the current section.
 - b) If the tag has a child, set the current section to be that child, and return to step 2.
- 5 Otherwise not found.



The armul.cnf ToolConf File

For the example ARM7TDMI:

- 1 Find Processors.
- 2 Look up ARM7TDMI in Processors.
- 3 Tag has no child.
- 4 Tag has value ARM7TDM, so:
 - a) Look up ARM7TDM in Processors.
 - b) Return to stage 2 with resulting child.
- 2 Look up ARM7TDMI in Processors:ARM7DM.
- 3 This has a child—found the configuration for ARM7TDMI.

Note that this “aliasing” is done by ARMulator, not by ToolConf.

4.5.1 Adding a new processor model

Suppose you have created a memory model called `MyASIC`, designed to be combined with an ARM7TDMI processor core to make a new micro-controller, called ARM7TASIC. To allow you to select this as an available processor, add a section to the `Processors` list in the configuration file:

```
{ ARM7TDM
Processor=ARM7TDM
Core=ARM7
ARMulator=BASIC
Architecture=4T
ARM7TDMI:Processor=ARM7TDM
| ARM7TASIC:Processor=ARM7TASIC
| ARM7TASIC:Memory=MyASIC
}

ARM7TDMI=ARM7TDM
| ARM7TASIC=ARM7TDM
```

Three lines (denoted with “|”) have been added:

- `ARM7TASIC:Processor=ARM7TASIC`
This line is added *inside* the ARM7TDM child. This declares that you have a new processor, selectable as ARM7TASIC, which is derived from the ARM7TDM and is called ARM7TASIC.
- `ARM7TASIC:Memory=MyASIC`
The `Memory` tag in a processor’s configuration declares which memory model the processor uses. Usually it is not specified, and ARMulator uses the default memory model (see below). This line specifies that the ARM7TASIC processor uses the `MyASIC` memory model. Again, it goes *inside* the ARM7TDM region, as ARM7TASIC is derived from ARM7TDM.
- `ARM7TASIC=ARM7TDM`
This goes *outside* the ARM7TDM region, and declares that for ARM7TASIC, ARMulator should look in ARM7TDM (as for ARM7TDMI).



4.5.2 Plumbing

A complication exists in ARMulators up to and including release 2.11. Where other memory models may want to exist in between a processor core and the true memory model—for example, the *Tracer* bus-watching model—`armul.cnf` must include an amount of “plumbing” for these models for each and every processor.

For example, take a look at the plumbing for the ARM710. The ARM710 is an ARM7 with a cache memory model instead of the standard memory model. You would expect this to appear:

```
{ ARM7
Processor=ARM7
Core=ARM7

ARM710:Processor=ARM710
ARM710:Memory=ARM710
}

ARM710=ARM7
```

This is similar to the example in **4.5 Processors** and **4.5.1 Adding a new processor model**—ARM710 is an ARM7—but with a memory model called ARM710 (which is the Cache+MMU model: see **4.6 Memories**).

However, this is not what `armul.cnf` contains. Instead, it reads:

```
{ ARM7
Processor=ARM7
Core=ARM7

ARM710:Processor=ARM710
#if TraceMemory
ARM710:Memory=TARM710
#endif
#if WatchPoints==True
ARM710:Memory=WARM710
#endif
ARM710:Memory=ARM710
}
```

If `TraceMemory` or `Watchpoints` are set the memory model becomes `TARM710` (“Traced ARM710”) or `WARM710` (“Watched ARM710”) respectively.

The way this works on the memory side of the configuration file is explained in **4.6.2 Plumbing** on page 16.



4.6 Memories

The next major section is the `Memories` section. This contains all the memory models which may be referred to by a `Memory` tag in a processor's configuration.

Its structure is similar to the `Processors` section, in that it contains children and aliases.

It also contains a `Default` tag, which declares the memory model to be used at the bottom level; that is, the basic RAM model. This may be one of several things. For example, the `MemConfigToLoad` tag is checked to see whether to use the `MapFile` model, or the basic `Flat` model.

The same algorithm is followed for finding a memory model's configuration inside the `Memories` section as is used to locate processors in the `Processors` section (see **4.5 Processors** on page 11).

4.6.1 Hierarchy

The memory system inside ARMulator forms a hierarchy. This allows, for example, cache models, bus tracers and so on, to be inserted seamlessly into the emulated system. This hierarchy is largely controlled by the `armul.cnf` file.

The processor's configuration contains the tag `Memory`. The value of this tag locates the top-level of the memory system. For example, ARM710 has `Memory=ARM710`. The memories section contains:

```
{ Memories
{ MMUlator
{ ARM700
ARM710:NoCoprocessorInterface
ARM710:ChipNumber=0x710
}
ARM710=ARM700
Memory=Default
}
ARM710=MMUlator
}
```

Following the algorithm from **4.5 Processors** on page 11, the configuration for ARM710's memory model is:

- 1 Find Memories.
- 2 Look up ARM710 in Memories.
- 3 Tag has no child.
- 4 Tag has value MMUlator so:
 - a) Lookup MMUlator in Memories.
 - b) Return to stage 2 with resulting child.
- 2 Look up ARM710 in Memories:MMUlator.
- 3 Tag has no child.
- 4 Tag has value ARM700 so:
 - a) Look up ARM700 in Memories:MMUlator.
 - b) Return to stage 2 with resulting child.
- 2 Look up ARM710 in Memories:MMUlator:ARM700.
- 3 This has a child—found configuration for ARM710.

Hence ARM710 is derived from ARM700 (generic cached-ARM7), which is in turn derived from MMUlator (generic cached-ARM).

When the MMUlator cache model initializes it looks for the next level down in the memory hierarchy. It looks up the Memory tag in the ARM710 configuration, and finds Default (as the lookup recurses up the tree to the MMUlator area).

Hence the Default memory model is placed under the ARMUlator.

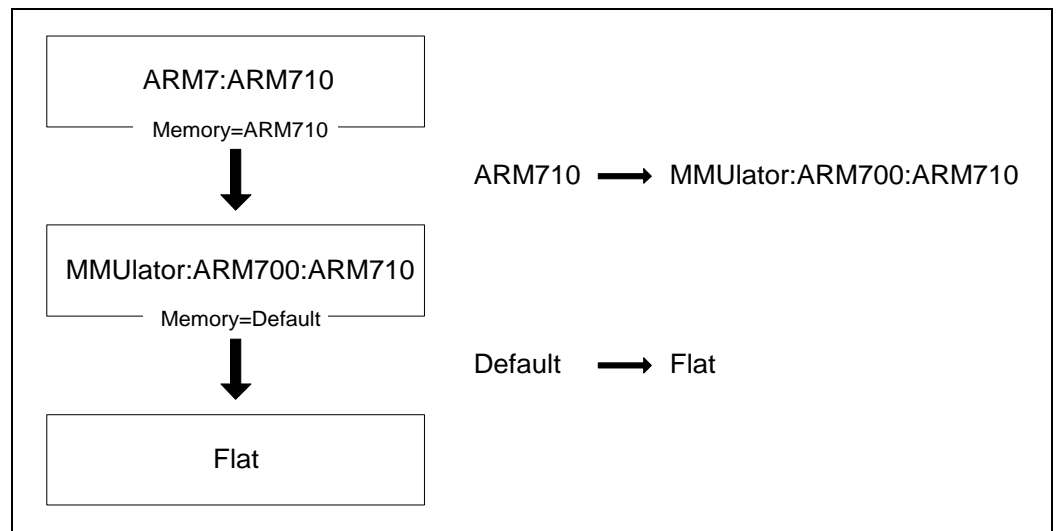


Figure 1: Memory hierarchy of a cached processor

4.6.2 Plumbing

As mentioned in **4.5.2 Plumbing** on page 13, releases up to and including release 2.11 require “plumbing” to support models such as the tracer.

If `TraceMemory` is defined, then the ARM710 processor instead uses `TARM710` as the memory model. This is defined in the `Memories` section as:

```
TARM710=Tracer
{ Tracer
  TARM710:Memory=ARM710
}
```

This inserts `Tracer` between the processor and the true ARM710 memory model.

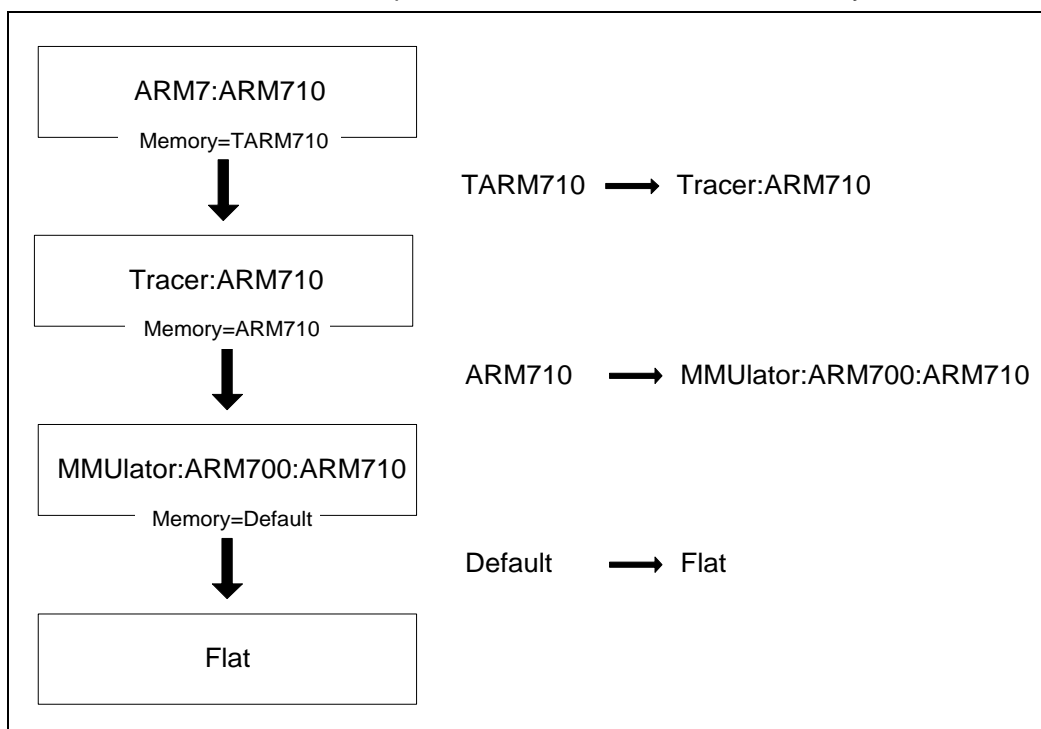


Figure 2: Memory hierarchy with tracer inserted

Similar plumbing is used for the `WatchPoints` model. The disadvantage of this method is that, for M real memory models, N processors and P veneer memory models, `armul.cnf` would need to have $N*M*P$ entries to deal with the possible combinations. (In fact, the current release does not support having both `WatchPoints` and `TraceMemory` in the hierarchy.)

Note *Future releases will remove this need, as models such as `Tracer` will be able to insert themselves into this hierarchy.*

4.7 Coprocessors

The `Coprocessors` section contains two types of entry.

Firstly, it contains configurations for all coprocessor models that need them. It does not, however, do so in the same complex manner as the `Processors` and `Memories` sections. For example, the `DummyMMU` model might have the simple entry:

```
{ Coprocessors
  DummyMMU:ChipID=0x12345678
}
```

Secondly, it contains a list of assignments of coprocessor numbers to coprocessor models, for example:

```
Coprocessor[15]=DummyMMU
```

which associates the `DummyMMU` coprocessor model with coprocessor number 15.

Note *The ARM supports 16 coprocessors, numbered 0 to 15. Entries for coprocessors outside this range are ignored.*

4.8 Basic models

The `Models` section contains configurations for the basic models. Basic models are those models which attach themselves to call-backs, write things to memory and so on, but do not supply functions to the ARMulator (as memory models do).

ARMulator goes through the list of models declared to it, and initializes all those for which it can find an entry in the `Models` child.

For example, the `Profiler` module's configuration is contained in the `Models` section:

```
{ Models
  { Profiler
    Type=Instruction
  }
}
```

This tells the ARMulator to start the model called `Profiler`. That model then sets itself up to profile instructions.



The armul.cnf ToolConf File

4.8.1 Disabling models

Since a model is only started if a configuration for it is found in the `Models` section, you can disable a model by selectively removing the configuration from the section.

For example, the `PageTables` model is controlled in this way. In the `Header` section, there is an entry:

```
UsePageTables=True
```

and in `Models`:

```
{ Models

  #if UsePageTables==True
  { PageTables
    MMU=Yes
    AlignFaults=No
    Cache=Yes
  }
  #endif
}
```

If `UsePageTables` is set to `False` in the header, the `#if UsePageTables==True` directive fails, and hence no configuration for `PageTables` is found, so the model is not enabled.