AMBA University Kit

User Guide

ARM

ARM DUI 0167A

AMBA University Kit
User Guide

Release information

Change history

Date	Issue	Change
September 2001	A	First issue

Proprietary notice

Document confidentiality status

This document is Open Access. This document has no restriction on distribution.

Product status

The information in this document is Final (information on a developed product).

ARM web address

`http://www.arm.com`

Contents
AMBA University Kit User Guide

Preface

This preface introduces the ARM *AMBA University Kit* (AUK) and its reference documentation. It contains the following sections:

About this document

This document is the user guide for the AUK.

Intended audience

This document has been written for *System-on-Chip* (SoC) designers and system architects, and provides a description of components within the AUK architecture.

Using this manual

This document is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for details of the structure of the AUK.

Chapter 2 *Initialization*

Read this chapter for information on how to initialize the AUK.

Chapter 3 *Simulation*

Read this chapter for details of TICTalk testing, using the TBTic test bench, and for ARM software testing, using the TBEasy test bench.

Chapter 4 *Synthesis*

Read this chapter for details of how to synthesize the AUK.

Appendix A *HDL Libraries and Dependencies*

Read this chapter for relationships between the modules, and libraries required for VHDL and Verilog files.

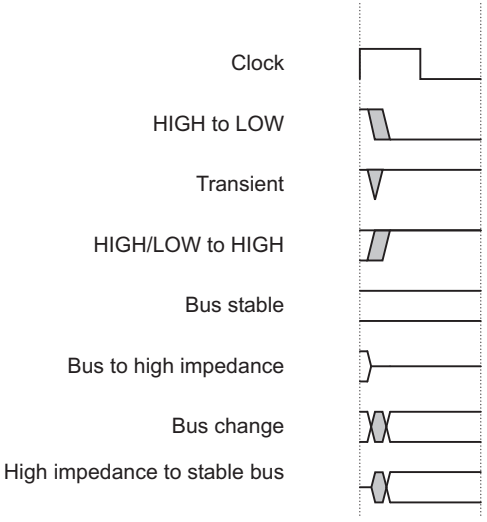
Typographical conventions

The following typographical conventions are used in this document:

bold	Highlights ARM processor signal names, and interface elements such as menu names. Also used for terms in descriptive lists, where appropriate.
italic	Highlights special terminology, cross-references, and citations.
`typewriter`	Denotes text that can be entered at the keyboard, such as commands, file names and program names, and source code.
`typewriter`	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`typewriter italic`	Denotes arguments to commands or functions where the argument is to be replaced by a specific value.
`typewriter bold`	Denotes language keywords when used outside example code.

Timing diagram conventions

This manual contains one or more timing diagrams. The following key explains the components used in these diagrams. Any variations are clearly labeled when they occur. Therefore, no additional meaning must be attached unless specifically stated.



Key to timing diagram conventions

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

Further reading

This section lists publications by ARM Limited, and by third parties.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda.

See also the ARM Frequently Asked Questions list at:
`http://www.arm.com/DevSupp/Sales+Support/faq.html`

ARM publications

This document contains information that is specific to the AUK. Refer to the following documents for other relevant information:

- *AMBA Specification (Rev 2.0)* (ARM IHI 0011)
- *ARM Architecture Reference Manual* (ARM DDI 0100)
- *ARM7TDMI Data Sheet* (ARM DDI 0029)
- *AUK User Guide* (ARM DUI 0167).
- *Micropack AHB CPU Wrappers Technical Reference Manual* (ARM DDI 0169).
- *Reference Peripherals Specification* (ARM DDI 0062).

Other publications

- *IEEE 1149.1 JTAG standard*.

Feedback

ARM Limited welcomes feedback both on the AUK, and on the documentation.

Feedback on the AUK

If you have any comments or suggestions about this product, contact your supplier giving:

- the product name
- a concise explanation of your comments.

Feedback on this document

If you have any comments on about this document, send an email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments refer
- a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter provides an introduction to the *AMBA University Kit* (AUK). The chapter gives information about the overall structure and individual modules that constitute the AUK:

- *Structure of the EASY system* on page 1-2
- *General system structure* on page 1-5.

1.1 Structure of the EASY system

The EASY microcontroller comprises the building blocks required to create an example system based on the low-power, generic design methodology of the *Advanced Microcontroller Bus Architecture* (AMBA).

The EASY microcontroller:

- enables custom devices to be developed in very short design cycles
- allows the resulting sub-components to be re-used easily in future designs.

1.1.1 EASY system blocks

The example design provides all the system modules required to manage an AMBA system:

- reset controller
- arbiter
- decoder.

These system modules control the various aspects of the main system bus.

1.1.2 EASY components

The example design contains:

- Two bus masters:
  - the ARM processor to allow execution of the ARM code
  - the *Test Interface Controller* (TIC), to enable external control of the system bus during system test.
- A minimum set of basic microcontroller peripherals which are implemented as low-power designs on the AMBA *Advanced Peripheral Bus* (APB), including:
  - an interrupt controller
  - a remap and pause controller
  - a 16-bit timer module.
- The example *Static Memory Interface* (SMI). This demonstrates the minimum requirements for an *External Bus Interface* (EBI).
- A 1KB block of internal memory.

Note

The ARM processor module is only a model used for simulation and testing of the system.

Figure 1-3 on page 1-8 illustrates the structure of the AHB system.

Note

The processor module is not synthesizable in its entirety. This document does not tell you how to integrate the ARM processor macrocell into the synthesizable AMBA bus master logic veneer. See the documentation for the chosen ARM processor core type for more information.

1.1.3 AMBA bus master logic

You can connect the EASY microcontroller to any ARM processor core or cached macrocell. However, different processors require different AMBA bus master logic to control them.

Many ARM processor cached macrocells are supplied with an AMBA interface. However, the ARM7TDMI requires the AMBA wrapper supplied with the example system.

The AMBA ARM7TDMI bus master wrapper with test supports debug, using:

- **COMMTX** and **COMMRX** (connected internally in EASY)
- the JTAG pins for debug communications (not serial test or board level test).

  For more information, see the *Example AMBA System Micropack v2.0 Technical Reference Manual*.

For more information on variants, please contact ARM Limited.

1.1.4 Standard EASY component modifications

A typical design flow using EASY requires replacement or removal of the following components:

SMI Replace with system-specific memory controllers and an EBI.

Internal memory Remove or replace with a real SRAM component.

Provision is also made for the optional addition of:

- APB low-power peripherals, such as a UART or parallel port
- AHB slaves, such as a PC card (PCMCIA) interface or video controller
- AHB masters, such as a DMA engine or graphics processor.

These additions require minor modifications to the AMBA decoder, arbiter, or APB bridge interface, where appropriate. The *Example AMBA System Technical Reference Manual* contains details of all modules in the system, and describes the modifications required in each case.

Note

If you are using EASY as the basis for new microcontroller designs, ensure that your designs conform fully to the *AMBA Specification*, the *Reference Peripherals Specification*, and the base block designs of those presented in the EASY microcontroller. If you make minor changes to bus or state machine operation, you might produce a design that does not enable the ease of re-use, modification, or addition provided by the *AMBA Specification*.

1.2 General system structure

There are three main structures associated with the EASY system:

- *The file structure*
- *The system hierarchy* on page 1-7
- *The system connection layout* on page 1-8.

1.2.1 The file structure

Figure 1-1 illustrates the file structure of the HDL files used in an AHB-based system, and the directories associated with the files.

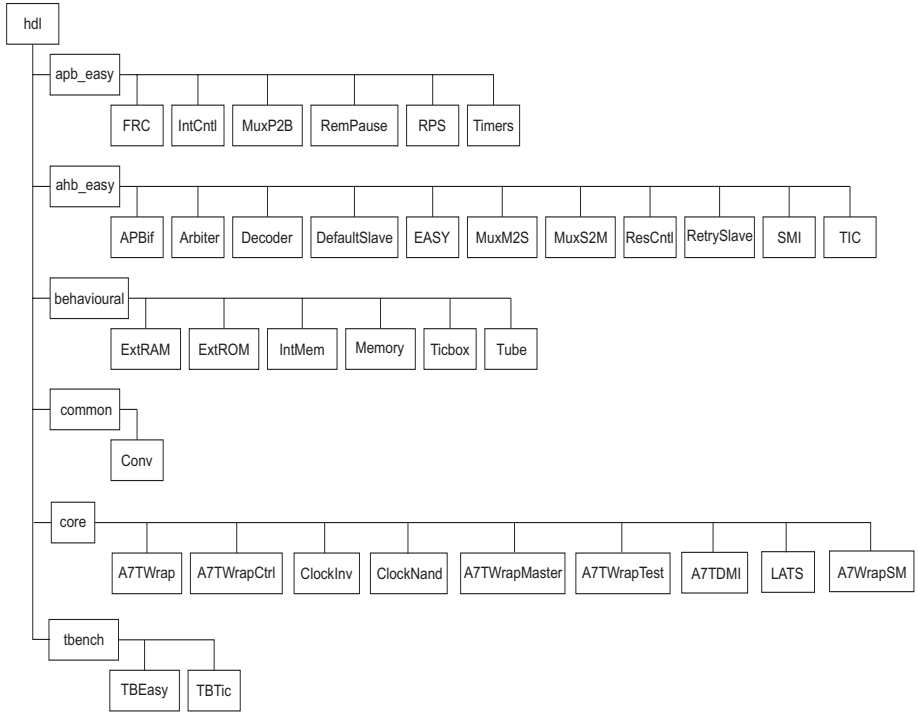


Figure 1-1 The AHB EASY file structure

At the top level of the file structure of Figure 1-1 on page 1-5 is the HDL directory. This directory is either `vhdl` or `verilog` depending on the chosen language. The directory contains seven sub-directories:

- `apb_easy` contains the standard APB peripherals and the RPS top-level structural file.
- `ahb_easy` contains all standard synthesizable modules connected to the system bus:
  - APB bridge
  - arbiter
  - decoder
  - top-level EASY structural file
  - reset controller
  - external bus interface
  - test interface controller
  - control multiplexors from masters to slaves and slaves to masters
  - example retry slave
  - default slave.
- `behavioural` contains all non-synthesizable modules provided with the example system:
  - the internal and external memories
  - ticbox
  - TUBE model.
- `core` contains the AMBA wrapper for the ARM7TDMI core used in the default system. The files in this directory must be changed if a different core is used.
- `common` contains the `conv` package which contains the conversion function not found in the standard IEEE `vhdl` packages.
- `tbench` contains the top-level simulation parts of the system, the test benches. It is also the directory from which simulations are run.

1.2.2 The system hierarchy

The system hierarchy is the way in which the HDL files connect together for simulation or synthesis. Figure 1-2 illustrates the system hierarchy for the AHB EASY system.

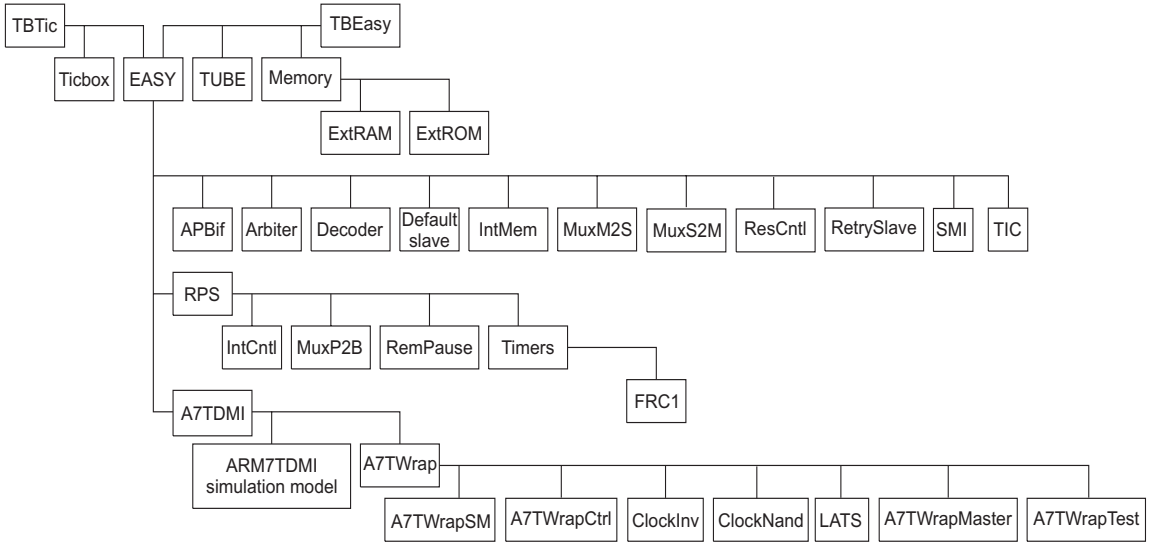


Figure 1-2 AHB EASY system hierarchy

The system hierarchy is the structure present in an HDL simulation of the EASY system, with the TBTic or TBEasy test bench at the top, and all other modules instantiated from it.

1.2.3 The system connection layout

This section shows the system connection layout for an AHB EASY system.

The system connection layout illustrates the relationship of the external, system, and peripheral buses to the system modules. For simplicity the diagrams show only one main bus and do not show all module-to-module and module-to-bus connections.

Figure 1-3 shows the system connection layout for the AHB EASY system.

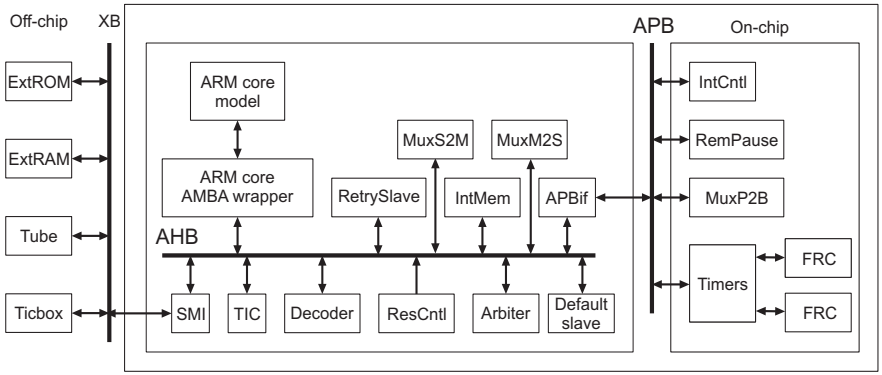


Figure 1-3 AHB EASY system connection

Chapter 2
Initialization

This chapter provides information on how to initialize the *AMBA University Kit* (AUK). It contains the following section:

- *System setup* on page 2-2.

2.1 System setup

Initialization consists of copying the EASY system files from the CD supplied. The following restrictions must be applied:

1. Use the default directory setup for your directory structure.
2. Do not set up symbolic links.
3. Do not make environment settings.

When this procedure has been completed, you are ready to install, set up, and run a simulation as described in Chapter 3.

Chapter 3
Simulation

This chapter provides information on the two simulations that you can run on the *AMBA University Kit* (AUK):

- TICTalk testing using the TBTic test bench
- ARM software test using the TBEasy test bench.

The main parts of the system are the same for both simulations, but the actual stimulus and input files are different. The following sections describe the simulation processes:

3.1 Simulation setup

When you have copied the system files as described in Chapter 2, the EASY system is ready for use with a simulation tool.

If you are using an ARM core, install and set up a simulation model of the required type for the specific simulator and operating system. See the documentation provided with the model for information about how to install and use the model.

To simulate the ARM core, it has to be compiled. This can be done by creating a link to the HDL source file from the ARM core installation in the `/core` directory:

- for VHDL, go into the `vhdl/core` directory and use the command:
  ```
  Ln -s $ARM7TDMI_HOME/ARM7TDMI/ARM7TDMI.vhd
  ```
- for Verilog, use the command:
  ```
  Ln -s $ARM7TDMI_HOME/ARM7TDMI/ARM7TDMI.v
  ```

3.2 System compilation

Before you can perform simulation, you must compile the system HDL files. This process is simulator-specific. Table 3-1 shows the HDL file compilation order to follow (also found in the README file in the top-level EASY_System/hdl directories).

Note

The TICBOX requires an infile.sim file to be present at compilation when using Verilog. See *Inputting data to the TBTic test bench* on page 3-5 for details of how to generate this file.

Table 3-1 HDL file compilation order

Directory	AHB HDL file
Common	Conv (VHDL only)
core	ClockNand ClockInv LATS A7TWrapCtrl A7TWrapTest A7TWrapMaster A7WrapSM A7TWrap A7TDMI
behavioural	ExtRAM ExtROM IntMem Memory Ticbox Tube
apb_easy	FRC IntCntl MuxP2B RemPause Timers RPS

Table 3-1 HDL file compilation order (continued)

Directory	AHB HDL file
ahb_easy	APBif Arbiter Decoder DefaultSlave MuxM2S MuxS2M ResCntl RetrySlave SMI TIC EASY
tbench	TBEasy TBTic

The EASY simulation world requires the use of extra HDL files that are not synthesizable, such as the test benches and memories. Example versions of these come with the system in the `behavioural` directory.

Perform all simulation from the `tbench` directory. This directory contains the top-level test bench HDL files for both the VHDL and Verilog versions of the EASY system, including the `easy.f` linking file for use with VerilogXL.

The `tbench` directory also contains the input data files, or symbolic links to the data files, used during the simulation, as explained in *TBTic test bench* on page 3-5 and *TBEasy test bench* on page 3-8.

3.3 TBTic test bench

The TBTic test bench uses the ticbox to drive the *Test Interface Controller* (TIC) to apply test vectors to the EASY components. Figure 3-1 shows the basic layout of the TBTic system.

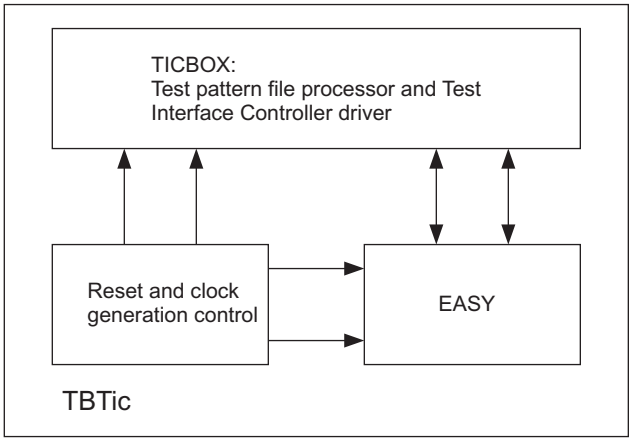


Figure 3-1 Block diagram of TBTic

3.3.1 Setting the TBTic clock frequency

To alter the clock frequency used in the TBTic test bench, change the delay mechanism in the reset and clock generation control module. The default is a 20ns period, or 50MHz for the AHB system.

At the start of the system operation, the system reset signal that is fed to the reset and clock generation control module is generated based on set numbers of clock cycles for its high and low phases.

3.3.2 Inputting data to the TBTic test bench

The *TIC Interface Format* (`TIF`) file contains the test-pattern commands.

The TICBOX reads the `TIF` file and translates the test-pattern commands into their respective TIC signal sequences. The TIC signal sequences are then applied to the EASY components.

The TIC signals control the system:

- reads
- writes
- address changes.

These commands are taken from the input file and displayed in the simulation window. An example of this for VHDL is shown in Example 3-1 on page 3-7 (the Verilog trace shows similar transfer type information).

The HDL version of the EASY system you are running determines the type of input files used to input data to the test bench.

VHDL

When you are using a VHDL version of the TBTic test bench, the data comes from `infile.tif` file in the `tbench` directory. The `infile.tif` file is a `TIF` file that is generated from TICTalk test patterns found in the `EASY_System/tictests` directory. (You can find precompiled `TIF` files for the APB peripherals in the `tictests/invec` directory.) The `infile.tif` can be one of the following:

- a copy of a compiled `TIF` file
- a symbolic link to a `TIF` file in the `invec` directory.

For more information on the file formats, conversion processes, and generation of new test vector files, see the *Test Interface Driver* section of the *Example AMBA System (Micropack v2.0) Technical Reference Manual*.

Note

Messages starting with a semi-colon (;) are comments from the `TIF` file broadcast by the TICBOX.

The warnings shown in Example 3-1 on page 3-7 indicate that the test on the read value has failed. You can modify the TICBOX to generate Failure assertions when test errors are detected. See the *Example AMBA System Technical Reference Manual*, Chapter 5 *Test interface driver*, for more details about read vector assertions. When the test is complete, TBTic halts the simulation with a Failure assertion.

The system applies the read vectors in a pipelined fashion, so the value read back from a read vector is checked several cycles after the `Reading :Expected...` message.

Example 3-1 TBTic simulation output example

```
# ** Note: ; Writing data 00000002
# Time: xxxxxx ps Iteration: 1 Instance:/u_ticbox
# ** Note: ; Addressing location 84000024
# Time: xxxxxx ps Iteration: 1 Instance:/u_ticbox
# ** Note: ; Reading. Expected: 12345678. Mask 0000FFFF
# Time: xxxxxx ps Iteration: 1 Instance:/u_ticbox
# ** Note: ; Looping for 2 cycles
# Time: xxxxxx ps Iteration: 1 Instance:/u_ticbox
# ** Warning: Error on vector read. Expected: 12345678 Actual: 12345677 Mask: 0000FFFF
# Time: xxxxxx ps Iteration: 1 Instance:/u_ticbox
# ** Failure: Test sequence completed
# Time: xxxxxx ps Iteration: 1 Instance:/
```

Verilog

If you are using a Verilog version of the TBTic test bench, the data comes from the `infile.sim` file in the `tbench` directory, and is read by the TICBOX. You can generate the file `infile.sim` from a `TIF` file using the conversion script `tif2sim` found in the `tictests` and `tbench` directories.

You can modify the verilog ticbox to generate a warning on a read vector error, or to stop the simulator. See the *Example AMBA System Technical Reference Manual*, Chapter 5 *Test interface driver*, for more details about controlling the simulator operation on read vector errors.

Cadence Verilog

If you are running a Cadence Verilog simulation, use the executable file `runit-tic` with a `tbench/Verilog` executable containing the ARM core model. File `runit-tic` converts the specified `TIF` file to `.sim` format automatically before starting the simulation. See the documentation supplied with the core for more information about generating this Verilog executable.

3.4 TBEasy test bench

The TBEasy test bench uses an external memory model to simulate running code on the ARM core in the EASY system. Figure 3-2 illustrates the basic layout of the TBEasy system.

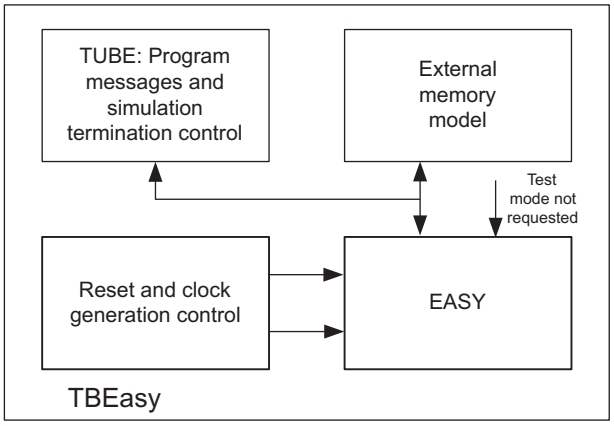


Figure 3-2 Block diagram of TBEasy

3.4.1 External memory module

The external memory model is the same for VHDL and Verilog systems. The *Example AMBA System (Micropack v2.0) Technical Reference Manual* describes the external memory model used by the TBEasy test bench.

3.4.2 Setting the TBEasy clock frequency

To alter the clock frequency used in the TBEasy test bench, alter the delay mechanism in the reset and clock generation control module. The default is a 100ns period, or 10MHz.

The system reset signal that is fed to the reset and clock generation control module is also generated based on set numbers of clock cycles for its high and low phases at the start of the system operation.

3.4.3 Program message and simulation termination control

In the TBEasy test bench, program message and simulation termination control is performed by the TUBE. This module acts as a one-way communication port through which the program can pass ASCII information.

Messages are written, one byte at a time, to location `0x20000000` in external memory. This location is used by the TUBE model, which buffers received bytes until a terminating control character is recognized and the message is printed by the simulator, for example:

```
# ** Note: TUBE: Hard Reset
# Time: xxxxxxx ps Iteration: 0 Instance:/
```

In this example the message `Hard Reset` has been passed to the TUBE. The program running on the microcontroller can also terminate simulation by writing a control character to the TUBE with no message to produce the following assertion:

```
# ** Failure: TUBE: Program exit
# Time: xxxxxxx ps Iteration: 0 Instance:/
```

The control codes passed to the TUBE are also output to the file `tbench/tube`.

3.4.4 Inputting data to the TBEasy test bench

The input data required to run the simulation comes from the four memory data files, `rom0.dat` to `rom3.dat`, that are located in the `tbench` directory. The external ROM models load the memory data files. The ARM core reads the data from ROM when the simulation is running.

The following sections describe the process:

- *Creating memory data files*
- *Reducing simulation runtimes* on page 3-9
- *Cadence simulation* on page 3-10.

Creating memory data files

The format of the memory data files is the same for both VHDL and Verilog memory models. The original code is compiled in the `EASY_System/code` directory.

To create the memory data files:

1. Write the necessary files in either C or ARM assembly language.
2. Compile and convert the source files into `rom(n).dat` files.

Test programs on page 3-13 describes this complete process in more detail.

Reducing simulation runtimes

You can reduce the simulation runtime by loading the internal memory or external SRAM models with data. This enables access to user data during the execution of programs, avoiding the use of test programs for manual data storage.

The tbench/intram.dat file is loaded by the 1KB internal memory, and must contain data in 8-character hexadecimal $readmemh Verilog format (for both the VHDL and Verilog format models).

Note

Loading takes place with or without addresses. When no addresses are used, loading starts from 0.

If you want to load the external SRAM with data you must use ram.dat files that have the same format as the rom.dat files for the external ROM models.

Cadence simulation

To input data to the TBEasy test bench for a Cadence simulation, use the runit-easy executable with a tbench/verilog simulator executable containing the model of the ARM core.

3.5 Simulation messages

During simulation, some of the non-synthesizable models generate messages in the simulation window. These simulation messages show either that the simulation is proceeding as normal, or that there is an error in the system.

This section details most of the possible messages that can be generated during a simulation run. These include:

- *Startup messages*
- *ARM core messages* on page 3-12
- *TICTalk messages* on page 3-12
- *Netlist messages* on page 3-12
- *Other messages* on page 3-12.

Note

The examples given are for the Modeltech VSIM simulator running a VHDL EASY model. Other simulators might produce slightly different message formats.

3.5.1 Startup messages

The ARM core model generates messages when the system first starts to run, during the *Power On Reset* (POR) stage. ARM generates messages for the commands that are being performed by the core.

A typical simulation start is shown below:

```
# 0ps
# ### ARM7TDMI Model Revision 1.4 (ModelGen 2.1.0beta) ###
#
# 0ps
# Tracer Module: trace all
#
# 0ps
# Instruction & Data tracing turned on.
#
# 0ps
#
# === MODE CHANGE ==> USR26
#
#
# 0ps
# Illegal MUX setup in PSR
#
# >>> Interrupt sequence started <<<
# # 0ps - /u_easy/u_proc/u_core/times/inout0/core: Warning:
```

```
# Width violation on signal TCK
#
# 2000ps - /u_easy/u_proc/u_core/times/inout0/core: Warning:
# Hold violation on signal BIGEND, ref signal MCLK
#
# 2000ps - /u_easy/u_proc/u_core/times/inout0/core: Warning:
# Low Phase violation on signal BIGEND, ref signal MCLK
#
# 1098000ps
# 00000000 : EA000032 : B  0x000000D0
#
# 1644000ps
# 000000D0 : EB0007E8 : BL 0x00002078
```

3.5.2 ARM core messages

When you use TBEasy to run ARM code, the simulation window displays the instruction trace.

See the documentation included with the ARM core simulation model for more information about simulation messages generated by the core.

3.5.3 TICTalk messages

When a TICTalk test is being run, it displays each TIC command. Bear in mind that TIC commands correspond to combinations of TIC signals. When the read data is incorrect, it gives rise to an error message.

3.5.4 Netlist messages

When you are simulating netlist files, timing differences from the original behavioral HDL, or incorrect setup times on register cells, might generate extra messages.

3.5.5 Other messages

If you receive any other messages that are not described in this section, or are not from the ARM core simulation model and are not standard simulator messages, use the following to help you track down their cause:

- the assertion time
- simulation waveform display
- the ARM instruction trace.

3.6 Test programs

The default system contains two example test programs that you can run on the system to generate ARM-based code:

`easy_c.c`	C test program, found in `EASY_System/code/easy_c`
`easy_asm.s`	an ARM assembly language test program, found in `EASY_System/code/easy_asm`.

3.6.1 easy_c.c

The `easy_c.c` example C program is compiled with the embedded C library using the `armcc` compiler from the *ARM Development Suite* (ADS).

This program:

- exercises the basic RPS functions
- checks the operation of the counters
- reports the results of the checks using the TUBE.

The advantages of using C rather than an assembly language test program are that:

- you do not have to be familiar with ARM assembly language
- C is a high-level programming language.

The disadvantage of using C is that you cannot directly control the ARM instruction sequence produced by the compiler.

When you use `easy_c.c` under simulation, it is quite difficult to establish which section of the C program is currently executing by looking at the instruction trace. You can use the TUBE program to output messages during simulation to avoid this problem.

The `easy_c.c` test program is described in:

- *Compiling and converting easy_c.c*
- *Files used by easy_c.c* on page 3-14
- *Components not tested when using easy_c.c* on page 3-14
- *easy_c.c test program messages* on page 3-14.

Compiling and converting easy_c.c

To use `easy_c.c` with the external ROM model in the EASY system, you must compile the test and convert it into a memory file format (`.dat`). To do this, use the makefile supplied and type `make`.

The ADS commands `armcc`, `armasm`, and `armlink` must be available in your path for use.

Files used by easy_c.c

During compilation the `easy_c.c` test program uses the following files:

- `easy_c.c`
- `init.s`
- `irq.s`
- `support.c`
- `reset.s`
- `easy_init.s`
- `arm.h`
- `peripherals.h`

The `EASY_System/code {easy_c, common, include}` directories hold these files. For details, see the `README` file in the `code/easy_c` directory.

Components not tested when using easy_c.c

Although the `easy_c.c` program and the tests it performs are quite basic, running the program on the EASY microcontroller model exercises most components within the system. The only sections that are not well exercised by these tests are the:

- test veneers
- *Test Interface Controller* (TIC)
- arbiter (though this is exercised partially by the reset and pause sequences).

easy_c.c test program messages

In normal execution on the EASY microcontroller, the test program sends the following messages using the TUBE. The messages, and the related conditions and events that initiate the messages, are tabulated below:

`Easy C test:`	Initial message.
`Power on reset`	This message is sent if the *Power On Reset* (POR) condition is detected in the *reset status register* of the *remap and pause controller*. The message `Soft Reset` is sent if the condition is not detected.
`Display ID`	Reads ID information from the *remap and pause controller*.

`No ID information`

The current implementation of the *remap and pause controller* does not have a valid ID value. If the ID register is implemented, the ID value is displayed in hexadecimal.

`Memory test` A simple memory test.

The following message is printed if the memory test fails: `Error: Memory test failed.`

`Software interrupt test`

A software interrupt test.

The following message is printed if the software interrupt test fails: `Error: Software interrupt test failed.`

`Counter tests` Counter timer tests.

`Counter test 1 complete`

This message is displayed when the test has completed.

`Counter test 2 complete`

This message is displayed when the test has completed.

`Test Passed` If all the tests pass, this message is displayed.

3.6.2 easy_asm.s

The `easy_asm.s` example test program is an ARM assembly language program.

Note

`easy_asm.s` is only a *template* test program designed to be expanded to test other logic in the system. It cannot be used as a complete validation test solution. Therefore, you must be familiar with ARM assembly language to use it effectively.

The advantages of using `easy_asm.s` are:

- You can more easily use the ARM core instruction trace to follow the execution of the test program.
- You can achieve greater control of the operation of the ARM core.

Test sequence of easy_asm.s

The `easy_asm.s` test program runs through the following simple sequence:

1. Initialize the processor.
2. Clear the Remap bit.
3. Writes `Easy_asm start` to the TUBE.
4. Start counter 1 at 30.
5. Write `Interrupt received` to the TUBE on receipt of the interrupt from counter 1.
6. Write `**TEST PASSED OK**` to the TUBE.
7. Exit.

Assembling and converting easy_asm.s

To use `easy_asm.s` with the external ROM model in the EASY system you must assemble it and convert it into a memory file format (`.dat`). To do this, use the makefile supplied, and type `make`.

The ADS commands `armcc`, `armasm` and `armlink` must be available in your path for use.

3.7 Setup to simulate a synthesized EASY system

Before you can simulate a synthesized EASY system you must:

1. Set up a working cell library for the simulator.
2. Compile the system files.

These tasks are described below.

3.7.1 Setting up a cell library

When setting up your working cell library for simulation, ensure it is compiled and appropriately referenced from within the netlist files.

If you compile a netlist file in a different directory from the directory containing the compiled cells, do one of the following:

- For VHDL netlist simulation add an extra `use` clause for a VHDL netlist.
- For Cadence Verilog netlist simulation either:
  - Add an extra command line option to the `verilog` command in `runit-easy/runit-tic` (such as `-y ../cells +libext+.ismvmd`).
  - Add an extra line in the `easy.f` file (such as `../common/udps.vmd`).

Note

A netlist simulation must have the correct timing to be able to run properly. Ensure the cell library contains timing for the cell, or that SDF annotation is used.

3.7.2 Simulating the system files

To simulate the system files:

1. Compile the system files following the instructions for the simulator to be used.
2. Run the simulation.

Note

When you run the simulation you might get extra warning or error messages. These can result from incorrect setup times or incorrect hold times on the inputs to registers. They can also be due to the initialization states of the simulation cell models.

Messages are generated in the first few nanoseconds of simulation while in the reset phase, but they are not normally displayed during the rest of the simulation.

Chapter 4
Synthesis

This chapter explains how to synthesize the AHB EASY system using the Synopsys synthesis tool with the script files supplied. It contains the following sections:

4.1 Synthesizing the EASY system

ARM supports synthesis of the EASY system using only the Synopsys synthesis tool with the script files supplied. ARM does not support other synthesis tools, but you can use the information found in the Synopsys synthesis scripts to generate scripts for other synthesis tools.

Note

You must already possess a working knowledge of the synthesis process before you attempt to synthesize the EASY system.

Figure 4-1 shows the default structure of the synthesis directory.

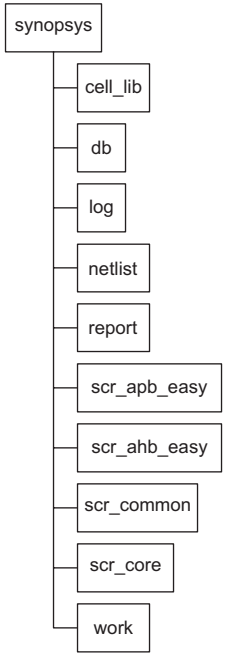


Figure 4-1 The default synthesis directory structure

Note

To synthesize the wrapper, a synopsys `db` file is required. This file is not supplied with the AUK, because it is dependent on the target technology.

4.2 EASY synthesis process

The following subsections describe the EASY synthesis process:

- *Requirements for the default EASY system for synthesis*
- *Purpose of synthesis scripts*
- *System synthesis scripts* on page 4-4
- *Synthesis timing constraints* on page 4-6
- *Synthesis results* on page 4-7
- *Using a different technology cell library* on page 4-7.

4.2.1 Requirements for the default EASY system for synthesis

The system must have access to the standard Synopsys synthesis tools version 1999.05 or later. To use Synopsys versions earlier than this, some commands in the script files might have to be modified because of command changes made in version 1999.05:

1. Ensure that the system directory structure is the same as the default system, so that the synthesis script files can find the correct HDL source files.
2. Choose the HDL RTL input and netlist output settings found in the `synopsys/HDL.csh` file.

When the system is set up correctly, you can synthesize the EASY modules.

4.2.2 Purpose of synthesis scripts

The standard synthesis process requires a script to:

- read in the RTL HDL source file
- set up the general system parameters
- constrain the port timing
- synthesize the module
- generate area, cell, timing and violation reports
- write out the generated netlist files.

This is done using a single master compile script, `scr_common/amba.cmd`, which can be used to synthesize all EASY modules. Environmental variables are used to pass settings to this script to select the module to synthesize.

The final area and gate count that is achieved is dependent on the target cell library use, and synthesis timing constraints and command options. The default synthesis settings are not optimized for speed or area, but for ease of re-use with different technology libraries and clock frequencies.

4.2.3 System synthesis scripts

The synthesis scripts that are used to synthesize each module in the AHB EASY system are:

`amba.cmd` The main synthesis command file contains the dc_shell command list used to synthesize the module. This file is cell library and module independent, only using variables and environmental settings that are passed to it.

`amba_params.scr` The AMBA timing parameters file contains the timing settings for all input and output ports used by all EASY modules, and the system clock frequency (default 50MHz = 20ns clock period). This contains generic timing values for all ports, and can be altered to use more accurate values when the system clock frequency and cell library used are defined.

`<module>.scr` The synthesis constraints file contains the timing constraints for the module input and output ports that are not included in the standard master and slave `.scr` files. Also contains any path specific settings that are required by the module, such as false or multicycle paths.

`apb_master.scr`, `apb_slave.scr`, `ahb_master.scr` and `ahb_slave.scr`
These four scripts contain generic port timing constraints for the standard APB and AHB signals (such as **HRDATA** and **PRDATA**) for APB and AHB masters and slaves (peripherals).

`cpu_wrapper.scr` This script file is used to set the extra options required when synthesizing an AHB CPU wrapper, including the top-level name of the wrapper module, and the synopsys timing library names for the CPU.

`dont_use.scr` The cells in the target library that are not to be used are specified in this script. This file must be modified for the chosen technology library.

`gobal.scr` This uses parameters from the `settings.scr` script to define the library worst and best case condition names, and the default cell drive and load strength. This file must be modified for the chosen technology library.

`name_rules.scr` The name rules used are defined in this file, and are used to ensure that SDF and netlist files are written out with matching names, no invalid characters, and restricted name lengths. This file must be modified for the chosen technology library.

`settings.scr` Defines the technology library settings used. This file must be modified for the chosen technology library.

`setup.scr` The setup script defines the target, link and symbol libraries, the UNIX directory path for synopsys read and write caches, and general system settings used to control synthesis for all system designs. It also includes the `settings.scr`, `dont_use.scr` and `name_rules.scr` script files.

Two shell scripts are used to control the synthesis of AMBA modules:

- `HDL.csh` is used to set the RTL source HDL used (VHDL or Verilog), and the HDL of the output netlist generated (VHDL, Verilog or both).
- `run_A7TDMI_all.csh` contains the system settings used to control the synthesis of all AMBA modules. This is used to specify the module names and HDL source filenames used during the synthesis of all system modules. The commands found in this script file can be used to synthesize individual AMBA modules.

The following environment variables are used during synthesis:

`HDL_IN` sets the RTL input HDL that is read.

`HDL_OUT` sets the netlist output HDL that is written.

`HDL_COMP` sets the `-checkout` option of `dc_shell` to use either VHDL-Compiler or HDL-Compiler.

`TOP_NAME` sets the top-level name of the module to be synthesized.

`OTHER_NAMES` sets the names of the modules instantiated by `TOP_NAME`.

`CPU_TOP_NAME` sets the top-level name of the CPU wrapper.

`CPU_LIB_MAX` sets the maximum CPU timing library file.

`CPU_LIB_MIN` sets the minimum CPU timing library file.

`MODULE_TYPE` defines the type of module to be synthesized, which can be any one of `APB_Master`, `APB_Slave`, `AHB_Master`, `AHB_Slave`, and `Other`.

4.2.4 Synthesis timing constraints

All synthesis timing values are defined in the `amba_params.scr` script, and are set as multiples of the system clock period, enabling all values to be scaled automatically if the clock frequency used is modified.

The default timing constraint values are:

- Point to point signals:
  - 40% output valid
  - 50% input setup
  - 10% spare slack
- Multiplexed signals:
  - 40% output valid
  - 20% multiplexer
  - 30% input setup
  - 10% spare slack

These values are chosen to enable most system ports to use the same timing values, but some signals are valid later than others, so there are exceptions to these timings such as slow CPU outputs and the slave select line which is a decoded version of the address.

The default constraints can be replaced with more accurate values when the system clock frequency and technology library are chosen, enabling more tightly constrained synthesis of the system.

Clock skew has to be set explicitly as minus (worst case) and plus (best case) uncertainty. The following default values have been used as preliminary values prior to layout:

- minus uncertainty is 5% of the clock period
- plus uncertainty is 40% of the minus uncertainty, which is 2% of the clock period.

The plus uncertainty derives from best case conditions and is less than the minus uncertainty value which derives from worst case conditions.

4.2.5 Synthesis results

Synthesizing a module creates the following files:

db/<module>_<hdl>.db	Synopsys format netlist file containing the synthesized module.
log/<module>_<hdl>.log	Log showing the output of the synthesis run.
netlist/<module>_net.v(hd)	VHDL or Verilog format netlist file containing the synthesized module.
netlist/<module>_<hdl>.sdf21	SDF 2.1 post-synthesis timing information file for the generated VHDL or Verilog netlist.
report/<module>.max	Maximum timing report for module.
report/<module>.min	Minimum timing report for module.
report/<module>.rpt	General information report for module.
report/<module>.vio	Synthesis violations for module.
report/<module>_area.rpt	Area report for module.
report/<module>_timing.rpt	Timing report for all module output ports.

Two additional log files (log/errors_<hdl>.log and log/violated_<hdl>.log) are generated when the run_all.csh script is run, which check the synthesis logs for occurrences of the words Error and VIOLATED. If either of these words are found it indicates a problem with the synthesis process. The violated log always contains at least one line for each module, because of the command set_max_area 0, which always creates an area related VIOLATED message. If modules are synthesized individually, the synthesis log and reports must be manually checked for errors and violations.

4.2.6 Using a different technology cell library

The default synthesis settings use the Avant! CB25 Passport set of cell libraries. If a different cell library is to be used, the following changes must be made to the synthesis scripts:

1. Ensure that a valid Synopsys format .db cell library is available from the synopsys/cell_lib directory.
2. Update the technology specific settings in dont_use.scr.
3. Update the default cell drive and load strength values in global.scr.
4. Update the synthesis naming rules used in name_rules.scr.
5. Update the technology library parameters in settings.scr.

4.3 Synthesizing new AHB modules

Follow these steps to synthesize new AHB modules:

1. Create a new entry in the `run_all.csh` shell script. Standard modules just require `TOP_NAME` and `MODULE_TYPE` setting, along with a `dc_shell` command line. Modules with hierarchy also require the `OTHER_NAMES` setting to be configured, and CPU wrappers require all standard CPU wrapper settings. Always use the command `unsetenv OTHER_NAMES` after the synthesis of a module with hierarchy, because this avoids subsequently synthesized modules from reading in these extra files.

2. Create a `<module>.scr` synthesis script file, which constrains the non-standard module ports, and sets any path specific constraints. Even if this file is blank, it must still be created to avoid an error message in the synthesis log.

3. For complex modules with many non-standard synthesis scripts, it might be worth creating a new `scr_module` subdirectory to avoid cluttering of the other `scr_*` directories. If this is done, a new entry is required in the search path setting of the `amba.cmd` main synthesis command file to enable the module synthesis scripts to be found.

Appendix A
HDL Libraries and Dependencies

This appendix shows the relationships between all the modules, and the libraries they require for use in the following sections:

A.1 AHB VHDL files

Table A-1 shows VHDL file dependencies for the AHB.

Table A-1 AHB VHDL file dependencies

Library	File	Type	Dependencies
apb_easy	FRC.vhd	Entity/architecture	ieee.std_logic_1164 ieee.std_logic_unsigned."-"
	IntCntl.vhd	Entity/architecture	ieee.std_logic_1164
	MuxP2B.vhd	Entity/architecture	ieee.std_logic_1164
	RemPause.vhd	Entity/architecture	ieee.std_logic_1164
	RPS.vhd	Entity/architecture	ieee.std_logic_1164 apb_easy.IntCntl apb_easy.MuxP2B apb_easy.RemPause apb_easy.Timers
	Timers.vhd	Entity/architecture	ieee.std_logic_1164 ieee.std_logic_unsigned."-" apb_easy.FRC
ahb_easy	APBif.vhd	Entity/architecture	ieee.std_logic_1164
	Arbiter.vhd	Entity/architecture	ieee.std_logic_1164 ieee.std_logic_unsigned."-"
	Decoder.vhd	Entity/architecture	ieee.std_logic_1164
	DefaultSlave.vhd	Entity/architecture	ieee.std_logic_1164
	EASY.vhd	Entity/architecture	ieee.std_logic_1164 apb_easy.RPS ahb_easy.APBif ahb_easy.Arbiter ahb_easy.Decoder ahb_easy.DefaultSlave ahb_easy.MuxS2M ahb_easy.MuxM2S ahb_easy.ResCntl ahb_easy.RetrySlave ahb_easy.SMI ahb_easy.TIC behavioural.IntMem core.A7TDMI

Table A-1 AHB VHDL file dependencies (continued)

Library	File	Type	Dependencies
	MuxM2S.vhd	Entity/architecture	ieee.std_logic_1164
	MuxS2M.vhd	Entity/architecture	ieee.std_logic_1164
	ResCntl.vhd	Entity/architecture	ieee.std_logic_1164
	RetrySlave.vhd	Entity/architecture	ieee.std_logic_1164, ieee.std_logic_unsigned."-"
	SMI.vhd	Entity/architecture	ieee.std_logic_1164, ieee.std_logic_unsigned."-"
	TIC.vhd	Entity/architecture	ieee.std_logic_1164 ieee.std_logic_unsigned."+"
behavioural	ExtRAM.vhd	Entity/architecture	ieee.std_logic_1164 std.textio common.Conv
	ExtROM.vhd	Entity/architecture	ieee.std_logic_1164 std.textio common.Conv
	IntMem.vhd	Entity/architecture	ieee.std_logic_1164 std.textio common.Conv
	Memory.vhd	Entity/architecture	ieee.std_logic_1164 behavioural.ExtRAM behavioural.ExtROM
	Ticbox.vhd	Entity/architecture	ieee.std_logic_1164 std.textio common.Conv
	Tube.vhd	Entity/architecture	ieee.std_logic_1164 std.textio common.Conv
common	Conv.vhd	Package	ieee.std_logic_1164

Table A-1 AHB VHDL file dependencies (continued)

Library	File	Type	Dependencies
core	A7TWrap.vhd	Entity/architecture	ieee.std_logic_1164 core.- A7TWrapSM A7TWrapCtrl ClockNand ClockInv LATS A7TWrapMaster A7TWrapTest
	A7TWrapSM.vhd	Entity/architecture	ieee.std_logic_1164 ieee.std_logic_unsigned."+"
	A7TWrapCtrl.vhd	Entity/architecture	ieee.std_logic_1164
	ClockNand.vhd	Entity/architecture	ieee.std_logic_1164
	ClockInv.vhd	Entity/architecture	ieee.std_logic_1164
	LATS.vhd	Entity/architecture	ieee.std_logic_1164
	A7TWrapMaster.vhd	Entity/architecture	ieee.std_logic_1164
	A7TWrapTest.vhd	Entity/architecture	ieee.std_logic_1164
	A7TDMI.vhd	Entity/architecture	ieee.std_logic_1164 core.A7TWrap Core simulation model library
tbench	TBEasy.vhd	Entity/architecture	ieee.std_logic_1164 ahb_easy.EASY behavioural.Memory behavioural.Tube
	TBTic.vhd	Entity/architecture	ieee.std_logic_1164 ahb_easy.EASY behavioural.Ticbox

A.2 AHB Verilog file

Table A-2 shows Verilog file dependencies for the AHB.

Table A-2 AHB Verilog file dependencies

Library	File	Dependencies
apb_easy	FRC.v	None
	IntCntl.v	None
	MuxP2B.v	None
	RemPause.v	None
	RPS.v	apb_easy.IntCntl apb_easy.RemPause apb_easy.MuxP2B apb_easy.Timers
	Timers.v	apb_easy.FRC
ahb_easy	APBif.v	None
	Arbiter.v	None
	Decoder.v	None
	DefaultSlave.v	None
	EASY.v	apb_easy.RPS ahb_easy.APBif ahb_easy.Arbiter ahb_easy.Decoder ahb_easy.DefaultSlave ahb_easy.MuxS2M ahb_easy.MuxM2S ahb_easy.ResCntl ahb_easy.RetrySlave ahb_easy.SMI ahb_easy.TIC behavioural.IntMem core.A7TDMI
	MuxM2S.v	None
	MuxS2M.v	None
	ResCntl.v	None

Table A-2 AHB Verilog file dependencies (continued)

Library	File	Dependencies
	SMI.v	None
	TIC.v	None
behavioural	ExtRAM.v	None
	ExtROM.v	None
	IntMem.v	None
	Memory.v	behavioural.ExtRAM behavioural.ExtROM
	Ticbox.v	None
	Tube.v	None
core	A7TWrap.v	core.A7TWrapSM core.A7TWrapCtrl core.ClockNand core.ClockInv core.A7TWrapMaster core.A7TWrapTest
	A7TWrapSM.v	None
	A7TWrapCtrl.v	None
	ClockNand.v	None
	ClockInv.v	None
	LATS.v	None
	A7TWrapMaster.v	None
	A7TWrapTest.v	None
	A7TDMI.v	core.A7TWrap Core simulation model library
tbench	TBEasy.v	ahb_easy.EASY behavioural.Memory behavioural.Tube
	TBTic.v	ahb_easy.EASY behavioural.Ticbox

Index

The items in this index are listed in alphabetic order. The references given are to page numbers.

D

E

F

H

I

L

M

N

P

R

S

T

V

W