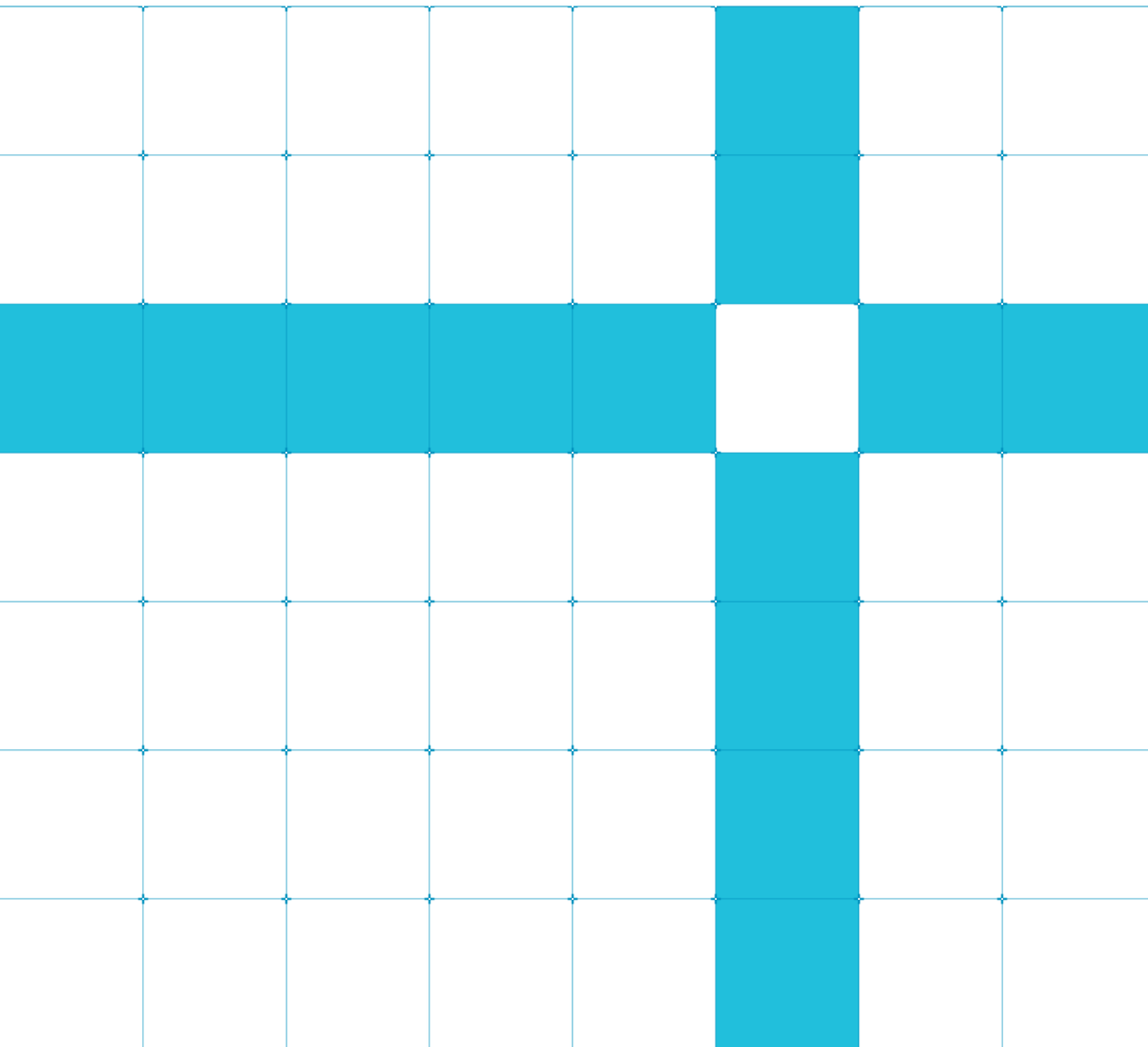




# Arm® Cortex®-A5 DesignStart™ Technical Overview

Version 1.0



# Contents

<b>1 Introduction.....</b>	<b>4</b>
1.1. Intended Audience.....	4
1.2. Documents provided with Arm IP.....	4
<b>2 Package contents .....</b>	<b>5</b>
2.1. Other IP required .....	6
2.2. Support .....	6
<b>3 Design flow .....</b>	<b>7</b>
<b>4 System overview.....</b>	<b>8</b>
4.1. Memory map guidelines .....	9
<b>5 Models available.....</b>	<b>9</b>
<b>6 Software .....</b>	<b>10</b>
<b>7 Components .....</b>	<b>10</b>
7.1. Cortex-A5 processor .....	10
1.7.1. Processor integration Layer .....	11
7.2. L2 Cache.....	11
7.3. NIC-400 .....	12
7.4. Debug and Trace .....	12
1.7.4. CoreSight integration and test .....	14
7.5. SRAM Controller .....	18
<b>8 RTL testing strategy .....</b>	<b>19</b>
<b>9 Socrates.....</b>	<b>20</b>
9.1. Add the DesignStart IP to Socrates .....	20
9.2. DesignStart IP Configuration.....	20

All brand names or product names are the property of their respective holders. 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 in good faith. All warranties implied or expressed, including but not limited to implied warranties of satisfactory quality or fitness for purpose are excluded. This document is intended only to provide information to the reader about the product. To the extent permitted by local laws Arm 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.

9.3. NIC-400 configuration .....	21
----------------------------------	----

# 1 Introduction

The Cortex®-A5 DesignStart™ package from Arm is designed to provide you with a starting point to build a custom SoC capable of providing full support for rich operating systems such as Linux. The package brings together a standard set of Arm processor, interconnect and peripheral IP which is needed by most SoC designs aimed at rich embedded and IoT applications.

The wide range of possible solutions can be overwhelming to someone coming to this particular design space for the first time, and this document provides an outline of some ways to simplify the process so you can make fast progress with your design. The individual IP elements within the package can still be fully customized, so you have the freedom to make different decisions where this is appropriate for your particular design.

## 1.1. Intended Audience

This document is aimed at software developers, front-end designers, implementation and validation engineers working with the Arm deliverables to develop a custom SoC. It should not be used to replace the detailed documentation which is introduced below.

Your use of the Arm deliverables will be covered by a license agreement, and you must follow the sign-off process described in your agreement with Arm

## 1.2. Documents provided with Arm IP

Arm documents each IP package as a stand-alone component, with typically several standard documents for each:

- **Release Note** – This short document will identify the structure of the component deliverables, the relevant installation process, and sometimes an ‘Out of Box’ test procedure.
- **Errata Documents** – These identify defects and workarounds for the component.
- **Technical Reference Manual** – This provides an overview of the component and describes the programmer visible specifics. You will need to refer to the relevant architecture reference manual at the same time as reading this document.
- **Integration Manual** – This describes each interface in detail, together with any relevant system level testbenches.

---

All brand names or product names are the property of their respective holders. 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 in good faith. All warranties implied or expressed, including but not limited to implied warranties of satisfactory quality or fitness for purpose are excluded. This document is intended only to provide information to the reader about the product. To the extent permitted by local laws Arm 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.

- **Configuration and Sign-off guide** – This document covers both implementation details and the testing which you are expected to perform for the individual IP. It may also describe some integration testing.

## 2 Package contents

The Cortex-A5 DesignStart Pro package provides an easy way to license and access several packages of Arm IP. For any SoC design, you must download and install IP from several different providers, then integrate them together to form a complete design. Parts from Arm are generally downloaded in bundles from Arm's deliverables server and are made available according to the licenses you have signed.

When you license Cortex-A5 DesignStart, you get access to a 'super-bundle' which contains several individual parts to ease the process of installing the Arm IP. These parts are listed in the following table (refer to the DesignStart release note within the bundle to check the revisions of each part).

Component	Arm Part Number	Details
Cortex A5 MP (+NEON)	MP008-BU-50000 AT552-BU-00000	The Cortex-A5 processor is a high-performance, low-power, ARM macrocell with an L1 cache subsystem that provides full virtual memory capabilities. Features include configurable number of cores (from one to four) and cache sizes.
NIC-400	PL401-BU-50000	Configurable AXI interconnect
Socrates	SYSOC	Tool to configure and build Arm IPs, required for configuring the NIC-400.
L2C-310	PL310-BU-00000	Optional L2 cache to trade latency for performance (depending on process)
CoreSight System for Cortex-A5 DesignStart	TM197-BU-50000	Support for debug and trace
CoreSight ETM-A5	TM955-BU-00000	Embedded trace Macrocell
BP140 SRAM Controller	BP140-BU-00000	Static memory offers low latency, at an area cost
Watchdog SP805	BP200-BU-00000	Standard APB peripheral included in design kit bundle

All brand names or product names are the property of their respective holders. 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 in good faith. All warranties implied or expressed, including but not limited to implied warranties of satisfactory quality or fitness for purpose are excluded. This document is intended only to provide information to the reader about the product. To the extent permitted by local laws Arm 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.

Timer SP804	BP200-BU-00000	Standard APB peripheral included in design kit bundle
UART PL011	PL011-BU-00000	Standard APB communication peripheral
GPIO PL061	PL061-BU-00000	General purpose I/O. This might be used to control on-chip logic.
RTC PL031	PL031-BU-00000	Real-time Clock

**Table 1** Components included in Cortex-A5 DesignStart

## 2.1. Other IP required

Depending on how you are partitioning your design work, you might require several other types of IP. Many of these are also available from Arm.

- Physical IP (standard cell libraries).
- Physical IP (foundry specific).
- Clock and power management IP.
- Analog IP.
- Peripherals.
- Dynamic memory controller.
- Memories.
- Radios.

The Cortex-A5 DesignStart Pro package is designed to be an ideal starting point for evaluation and building a custom SoC device. Depending on your application requirements, you can license additional IP from Arm to extend the system as necessary.

## 2.2. Support

When you license the Cortex-A5 DesignStart Pro package, you are entitled to direct access to our team of highly-qualified Arm experts via the web and email 24-hours a day. Our support team will provide responses to resolve any issues as quickly as possible, keeping delays to your project to a minimum. Track the status of your support questions with our online case-tracking tool accessible from <https://support.developer.arm.com>

## 3 Design flow

This chapter presents a simplified view of the initial steps that are required to start the process of building a full system on chip. For each step, you must refer to the documentation provided with each component.

Note:

When you are using mature IP, there might be aspects which need to be updated to work with modern design tools. For example, a modern compiler might be stricter with warnings and errors compared to tools that have been documented in the original IP documentation.

Some overlap is expected between the stages towards the end of the flow, and changes might need to be made in response to the results of later stages. You might need to repeat earlier stages in the flow to re-check the design.

The following steps are described in more detail in the individual IP documents, and presented in this document:

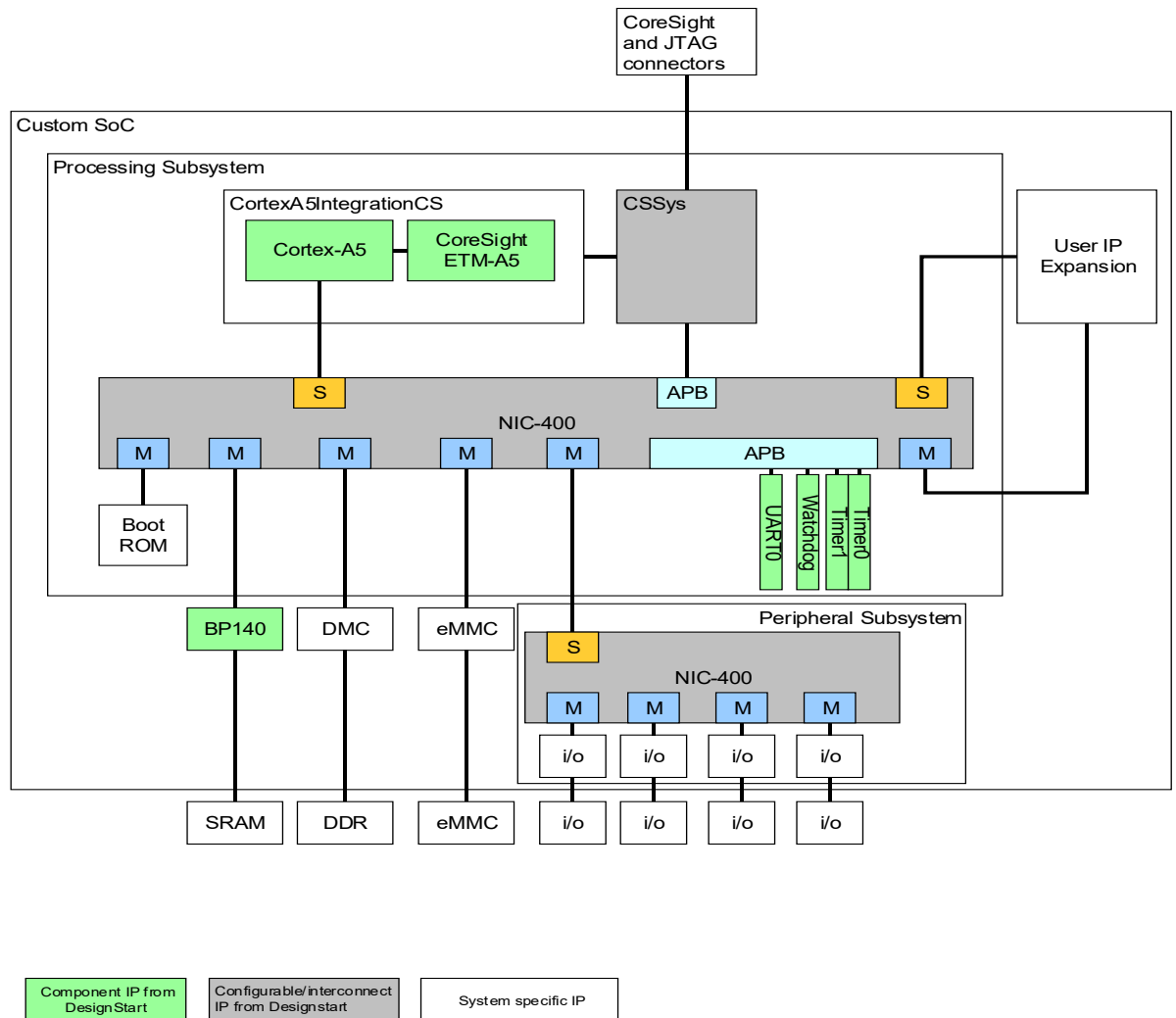
1. Unpacking and merging the deliverables
2. Configuring and generating the RTL for each configurable component. (processor, interconnect, CoreSight infrastructure). Socrates is used for this step.
3. Running the tests from the processor *Integration Kit* (IK) on the standalone processor. This also checks the configuration and uses behavioural RAM models.
4. Integrating RAMs with the processor and running the RAM integration tests.
5. Running the stand-alone testbenches for the remaining configured IP.
6. Integrating the processor, interconnect, CoreSight infrastructure and other peripherals.
7. Running the CoreSight integration tests for debug and trace.
8. Running integration tests for each peripheral, and any other relevant system level RTL simulations.
9. Following the peripheral specific implementation points, particularly relating to clock gating cells and clock domain crossing protection. This might require you to instantiate specific cells from your cell library.
10. Determining a strategy to partition your design for synthesis, based on your vendor's recommendations, the size of the full design, and your performance targets and the CSG recommendations.
11. Performing the implementation tasks.
12. Completing the sign-off requirements.

---

All brand names or product names are the property of their respective holders. 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 in good faith. All warranties implied or expressed, including but not limited to implied warranties of satisfactory quality or fitness for purpose are excluded. This document is intended only to provide information to the reader about the product. To the extent permitted by local laws Arm 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.

## 4 System overview

A simplified system block diagram is shown below as a guide for designing your system. It is assumed here that your design includes a custom IP block which has both master and slave AXI interfaces, however, in practice, you might have several of your own blocks to incorporate.





## 4.1. Memory map guidelines

There are very few restrictions on the memory map for a Cortex-A SoC. The processor boots from address 0x00000000, where the exception vectors are located. It is common to remap the vector table into fast RAM after boot. For the simplest path to porting Linux to your SoC, you can copy the Versatile Express memory map which is used by Arm models and FPGA platforms.

When designing your system, you should consider providing memories for the following:

- Boot flash memory (on-chip if security is important).
- Removable storage (bootable).
- DDR (system RAM).
- Code storage (eMMC or uSD).

# 5 Models available

Arm provides two different types of model, offering a trade-off between precision and operating speed. For best accuracy, you can use a Cycle model which is derived directly from the RTL of a representative system. Cycle models are licensed individually and can be downloaded from the Arm IP Exchange website, and several different example systems are available as pre-built *Cycle Performance Analysis Kits* (CPAK). Tools for customizing your virtual platform can be purchased separately.

For faster operating speed, Arm provides Fast Models. See <https://developer.arm.com/products/system-design/fixed-virtual-platforms> for more details. These are built as a virtual prototype platform and are better suited to software development or high-level modeling. Fast models are offered as a Fixed Virtual Platform, running at close to hardware speed. Like Cycle Models, tools for customizing your virtual platform can be purchased separately.

Both of these fixed virtual platforms are based around the Arm Versatile Express memory map. See the Fast Models Reference Manual for a full description of the VE memory map for Cortex®-A series.

The models are provided with a high level of instrumentation and support debug access by source level debuggers such as Arm DS-5 and a variety of others from Arm ecosystem partners.

## 6 Software

Cortex-A5 DesignStart Pro is intended to allow customers to develop a system capable of executing a Linux based software stack incorporating standard upstream components. A viable open Source software stack can be constructed using a variety of configurations. For example:

- Firmware: <https://www.trustedfirmware.org/>
- Bootloader: <https://www.denx.de/wiki/U-Boot>
- Linux Kernel: <https://www.kernel.org/>

The current release does not include any software deliverables demonstrating software integration for a Cortex-A5 DesignStart Pro based system. However, there is mature support upstream for Cortex-A5 (and associated Arm IP) in the Linux kernel and related open source software projects.

Arm is working towards providing evaluation platforms for a range of processor systems. When available, these will be supported by a reference software stack demonstrating relevant software applications (e.g. Cloud Clients) which customers can use to enable their own porting efforts.

## 7 Components

This section describes some of the design decisions and important points relating to each component in the design. It is intended to act as an introduction rather than to cover these components in detail.

### 7.1. Cortex-A5 processor

Up to four individual cores can be linked in a cache-coherent cluster, under the control of a *Snoop Control Unit* (SCU), that maintains L1 data cache coherency for memory marked as shared. The Cortex-A5 MPCore processor implements the Armv7-A architecture and runs 32-bit Arm instructions, and mixed 16-bit and 32-bit Thumb instructions. For general-purpose compute applications, Arm recommends that you configure the processor with the Arm NEON Media Processing Engine.

The processor can be configured with one or two AXI high-speed *Advanced Microprocessor Bus Architecture* (AMBA) L2 interfaces. The secondary port can be dedicated to a specific address range or used to increase memory bandwidth across the whole address map.

You can configure the processor with an *Acceleration Coherency Port* (ACP), an optional AXI 64-bit slave port that can be connected to a noncached peripheral such as a *Direct Memory Access* (DMA) engine.

The Cortex-A5 processor has several internal RAMs which need to be integrated as part of the implementation flow. You can simulate the core before integrating the RAMs using the models which are instantiated by default.

In the *Cortex-A5™ MPCore™ Configuration and Sign-off Guide* (ARM DII 0243), there are references to using vector replay for RTL validation. Arm now recommends that you use the *Integration Kit* (IK) which is described in the *Cortex-A5™ MPCore™ Integration manual* (ARM DIT 0015), as your lowest-level core testbench (this may also be described as an ‘execution testbench’ in other Arm documentation since it is the simplest environment which can execute code. For the trace and debug integration, you should now use the CoreSight SoC-400 flow which has been packaged as a stand-alone flow for Cortex-A5 DesignStart.

### 7.1.1 Processor integration Layer

The Cortex-A5 processor deliverables contain an integration layer (`cortexa5integration.v`) which was designed to integrate the processor and the tightly coupled debug components. This file has been superseded by the file `CORTEXA5INTEGRATIONCS.v` which is delivered with bundle TM197. This wrapper file should be used to instance the Cortex-A5 in your design. The Cortex-A5 configuration options are passed down as parameterized values when this module is instanced in the design, see the file `a5_system.v` in the TM197 bundle for an example of how to parameterize the `CORTEXA5INTEGRATIONCS` layer and the Cortex-A5. Note that the parameters for the `CORTEXA5INTEGRATIONCS` module include parameters for setting the cache sizes when using a DSM simulation model of the Cortex-A5 however when you are using the Cortex-A5 RTL the RAM sizes must be configured in the RTL for the Cortex-A5 RAM arrays as described in Chapter 4 Memory Integration of the *Cortex-A5™ MPCore™ Configuration and Sign-off Guide* (ARM DII 0243),

## 7.2. L2 Cache

An L2 cache is optional when using the Cortex-A5 processor. Depending on your process technology and clock frequencies, accesses to main memories can take many clock cycles. Where the main memories are off chip or constrained to low frequencies then using a Level 2 cache can provide a significant performance improvement. There is however a latency impact of using the L2 cache in your design, so it is possible that certain optimized designs using Cortex-A5 can achieve higher performance for specific tasks without using an L2 cache.

### 7.3. NIC-400

Refer to the *ARM® CoreLink™ NIC-400 Network Interconnect Technical Reference Manual* for more details and other references relevant to the NIC-400.

Cortex-A5 uses the AXI protocol for its memory interfaces. The desired system memory map must be constructed using an interconnect which maps each slave device to the correct address from the processor and any other bus masters in the design.

CoreLink NIC-400 is a highly configurable interconnect. It supports both AXI and AHB protocols on master and slave ports, as well as providing APB master ports for low-area peripherals.

The flow for configuring NIC-400 uses a tool to capture the required configuration (Socrates), and this tool controls the scripts for NIC-400. You can also render the configured RTL using command line scripting if this is necessary.

Depending on your system design and how you plan to develop variants in the future, it is often sensible to partition the system into more than one interconnect instance. Typically, the first level might serve the processor, main memories and any peripherals with very high bandwidth requirements. The other peripherals can then be connected as a separate block in the design hierarchy to partition the design task.

There are some compromises with the choice of how to partition the NIC and it can be simpler to treat the NIC for the whole system as one design which is split (within the configuration) into multiple switches. If the whole design is configured in one go, the memory map only needs to be entered once, and automated checking can be performed across the whole interconnect.

### 7.4. Debug and Trace

Debug and trace are important with a custom SoC, both for the initial bring-up process, to understand any problems with software development and to provide the best environment for software optimization.

The built-in debug functions of the Cortex-A5 processor include hardware breakpoints and watchpoints, single instruction step capabilities, and performance counters. Access to the control registers for this functionality is by the APB debug control interface.

Debug accesses to the Cortex-A5 are controlled through its Debug APB interface. For off-chip control of this interface uses the CoreSight debug architecture. This provides a single external interface (2 or 5 pins, using JTAG or the Serial Wire Debug protocol). This external serial interface is bridged to one or more on-chip interfaces using a *Debug Access Port* (DAP). The simplest DAP consists of a *Serial-Wire/JTAG Debug Port* (SWJ-DP) and an *APB Access Port* (APB-AP), this converts the JTAG or Serial Wire off-chip accesses into APB bus protocol accesses. All Cortex-A processors implement an APB slave port for debug control. The on-chip debug registers use memory mapped addressing.

---

All brand names or product names are the property of their respective holders. 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 in good faith. All warranties implied or expressed, including but not limited to implied warranties of satisfactory quality or fitness for purpose are excluded. This document is intended only to provide information to the reader about the product. To the extent permitted by local laws Arm 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.

The DAP provides an asynchronous bridge between the external interface and the internal buses. It also provides limited power and reset control which must be implemented correctly for the debug port to function. It is critical to verify proper function of the debug interface in simulation using the test methodology provided with CoreSight SoC-400 as part of your system validation.

Arm recommends that in addition to the basic debug, you also implement *Embedded Trace Macrocell* (ETM) trace with an *Embedded Trace Buffer* (ETB). This will require additional area budget (the ETB can be omitted if sufficient package pins can be dedicated to a trace port). The traceport size should be scaled according to the ratio of core clock to the interface clock, 400 MHz is a realistic maximum to aim for (assuming the appropriate pads are available).

To support streaming trace (not using the ETB), the following CPU frequency equivalent pins are required as a minimum:

Instruction trace	: 2 pins
Instruction with Cycle accurate trace	: 6 pins
Instruction plus data address	: 8 pins
Instruction, data address and data value	: 16 pins

Usually there is some filtering applied to limit the data transfers which need to be traced. In order to capture trace at higher port frequencies, dedicated trace capture hardware will be required. Trace can be read out of an ETB using the normal CoreSight debug interface.

A CoreSight system usually includes a connection from the main system memory map to the debug components (using the CoreSight subsystem to arbitrate accesses). This allows code running on the host to use the debug components for self-hosted debug.

### 7.4.1 CoreSight integration and test

The Arm CoreSight components are used to provide the debug access and trace infrastructure for the system. To simplify this design and integration, a pre-configured CoreSight subsystem and testbench is provided with Cortex-A5 DesignStart. For most designs that use one to four processors, this single pre-configured system provides all the debug and trace features which are typically implemented. When your system has fewer than four processors, the unused ATB (trace) ports should be tied off and any extra logic will be optimized away at synthesis time.

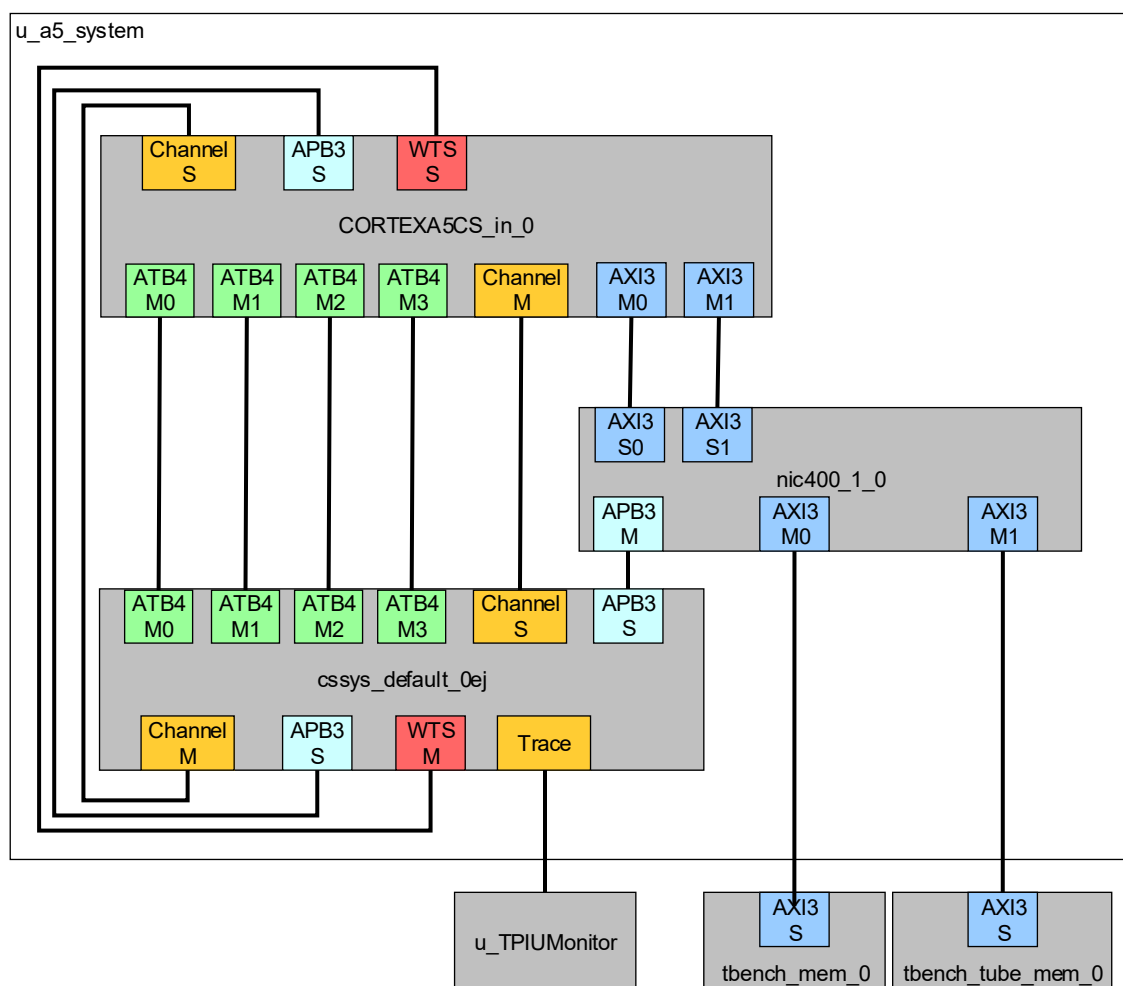
For more complex designs which incorporate additional trace sources, or which need to use an Embedded Trace Router to capture trace data to system memory, you can license the full CoreSight Soc-400 or SoC-600 IP Library from Arm. This also includes the AXI access port for direct debug access to memory, and JTAG access port to bridge from the CoreSight interface to internal scan chains.

The CoreSight Subsystem for Cortex-A5 DesignStart Pro provides:

- SWJ-DP (Debug port providing Serial Wire and JTAG interfaces over a common port).
- 4 32 bit ATB inputs to support up to 4 ETMs.
- Async ATB interface for each input port.
- ATB funnel (not configurable).
- ATB Replicator (programmable).
- Embedded Trace Buffer (ETB).
- Cross Trigger Interface (CTI).
- Cross Trigger Matrix (CTM).
- 32 bit Trace Port (TPIU).
- Timestamp generator connected directly to each trace source.
- ROM table (configurable for one to four processors).
- Testbench to support the maximal Cortex-A5 configuration.
  - Parameters to tie-off ports when used with other configurations.
- ARM test images as pre-built binaries
  - 'Discovery' test images for one to four processors

The top level of the CoreSight subsystem is `cssys_default_0ej`. It must be used with the `CORTEXA5INTEGRATIONCS.v` Cortex-A5 wrapper file to provide the correct debug control and capture capabilities for the Cortex-A5.

The example integration which is demonstrated by `a5_system.v` is shown in the diagram below.



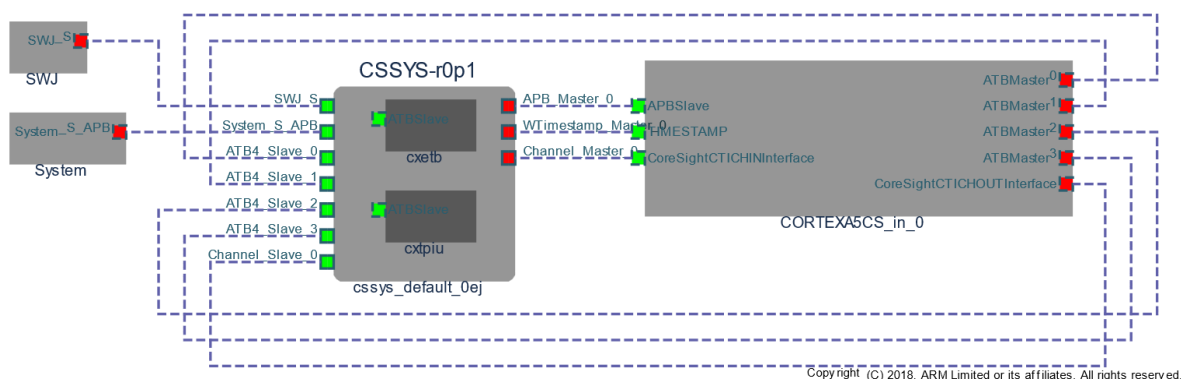
When incorporating the CoreSight subsystem into a design you must include an APB connection from the functional interconnect of the design into the APB slave port of the CoreSight subsystem. This port must be located at the physical address 0x40000000. This port is used by the Cortex-A5 processor to access the debug registers.

To run the CoreSight validation testcode on the subsystem in the design the testbench components `CXDT.v` and `TPIUMonitor.v` must be included in the testbench. Reference the `tbench.v` file for example connectivity of these components. The CoreSight SoC validation testcode also requires that a “Tube” component is included in the testbench at physical address 0xFC000000 so that the Cortex-A5 can report messages during the simulation. An example of how to connect this Tube component can be found in the file `a5_system.v`

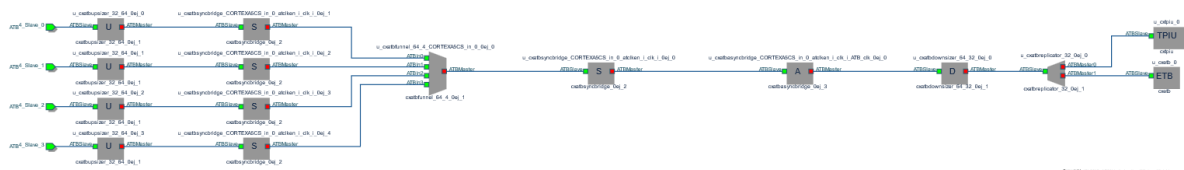
The testbench provided with the CoreSight system includes pre-compiled binary test executables for CoreSight integration. These tests will run on any Cortex-A5 system with between one and four cores, the only aspect which is not configured is the TARGETID value in the ROM Table. You must manually check that the TARGETID value reported in the logs matches the value that you have tied to u\_cxapbic\_APB\_MemMap0\_Oej\_0\_targetid and this value is consistent with your design intent. Guidance on how to tie off the TARGETID value can be found in the *ARM® CoreSight™ SoC-400 Integration Manual*. The testbench directly configures the NUM\_CPUs parameter for the processor, you must set this correctly to configure the design which is being tested.

The testbench connects the cssys\_default\_0ej CoreSight infrastructure subsystem to the CORTEXA5INTEGRATIONCS layer in the a5\_system.v file, the CORTEXA5INTEGRATIONCS instance is named CORTEXA5CS\_IN\_0.

The top-level connection between the Cortex-A5 and the CoreSight system is shown in the image below:



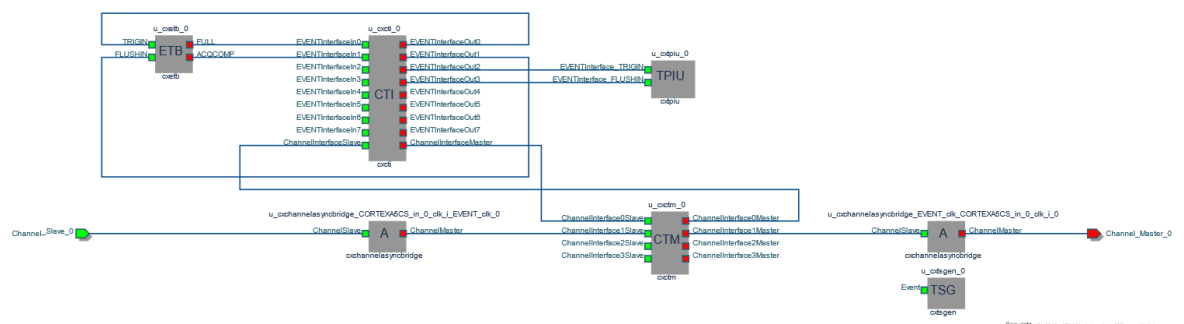
The detail of the trace interconnect can be found in the docs directory of the CoreSight package and is shown below for reference:



All brand names or product names are the property of their respective holders. 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 in good faith. All warranties implied or expressed, including but not limited to implied warranties of satisfactory quality or fitness for purpose are excluded. This document is intended only to provide information to the reader about the product. To the extent permitted by local laws Arm 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.



Within the CoreSight system, there is a Cross Trigger Interface and this has the following local trigger connections:



Once you have installed the processor, NEON, ETM and CoreSight system deliverables into a single working area, you can use the testbench from the <working area>/logical/testbench/validation directory. Start by configuring the test environment by running the setup script:

```
source ./set_env_vars.sh
```

This will configure your path for some of the validation scripts, and also set the following environment variables which are used to locate parts of the design and testbench. With the TM197 package for DesignStart, there is no additional installation or configuration of the CoreSight system to perform.

Variable	Purpose
CSSOC_INSTALL_DIR	CoreSight SoC testbench components
CORESIGHT_SUBSYSTEM_RTL	CoreSight Sub-system verilog
CORTXA5DS_RTL	Processor design 'logical' directory
CSSOC_HOME	Not used in this example
CSSOC_OVL_VERILOG	Assertion placeholder files

Before you run any tests, you must configure the NUM\_CPUs parameter in the <working area>/logical/testbench/verilog/testbench.v file. You must also edit <working area>/logical/testbench/validation/Makefile to set the correct CPU parameter.

For a single core Cortex-A5 set the parameter as:

```
CPU = 0
```

For a four core Cortex-A5 set the parameter as:

```
CPU = 0 1 2 3
```

You can modify the simulator that will be used by the tests by adjusting the common.mk file. Edit the line

```
SIMULATOR = IUS
```

The valid options are IUS, MTI, and VCS

Select the correct set of tests by linking the relevant test directory, for example for a two core design:

```
ln -s bin_2_cpu bin
```

You can now compile the design and run the first test using:

```
make rtl
```

```
make run TESTNAME=helloworld CXDT=1 CPU=0
```

If this is successful, you can run the full suite of tests with

```
make all
```

Note that in a multi-core system, some of the cpu tests (as opposed to the CXDT tests) will take a very long time to run.

Finally, check the results from all of the tests using

```
make report
```

These tests are described in detail in *ARM® CoreSight™ SoC-400 User Guide*.

## 7.5. SRAM Controller

The BP140 internal memory interface allows you to implement on-chip SRAM with an AXI interface. Although not strictly required for a linux platform, there will be many system applications which can take advantage of

a ~1MB region of relatively fast low latency RAM. Applications which use bus-master peripherals or accelerators may require dedicated RAM regions.

Critical code (such as exception handlers and vector tables) can also be located in the on-chip RAM.

## 8 RTL testing strategy

Arm recommends that for simulation testing is to focus in the integration between units of the design, testing both the correct operation (in a way that gives good top-level interface coverage), and also checking that invalid behavior generates the expected fault responses. For example, testing a memory interface should stress different access sizes and alignments, as well as out-of-range accesses to check that there is no aliasing and the correct exception handlers can be exercised. You are not expected to try and test (or verify) the internal functionality of any particular IP component since these will have already been thoroughly tested.

At the system level, it is important to check that any low-power (clock gated) states work correctly, and you might have some critical system level performance behavior which should be investigated using representative levels of interface activity.

Where you implement power gating, you are encouraged to perform power-aware simulations since the power-gated states will not otherwise be clearly observable.

Testing should be built up in stages so that if there are problems at the full chip level with a simulation, you can relate the behavior of each component to the activity seen when it was running in a simpler design. The starting point for this should be the processor integration testbench or the CoreSight creator testbench.

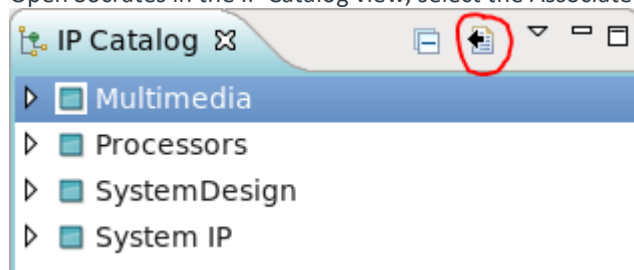
# 9 Socrates

Socrates is used to configure the DesignStart IPs and provides improved IP-XACT descriptions for these IPs.

## 9.1. Add the DesignStart IP to Socrates

The DesignStart IP must be added to Socrates as follows:

1. Download and unpack the DesignStart bundle.
2. Install the individual IP and Socrates using the instructions defined in the corresponding release notes.
3. Open Socrates in the IP Catalog view, select the Associate IP Bundle button.

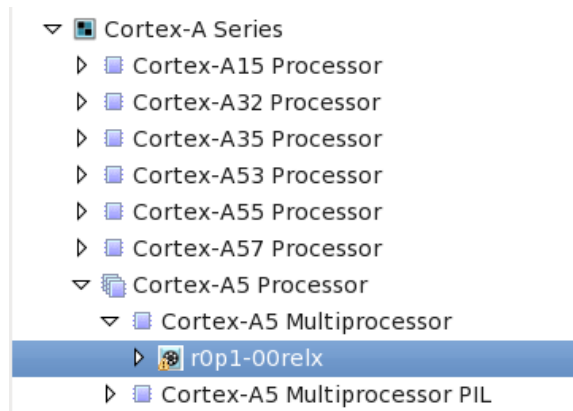


4. This brings up a dialog. In the field *Bundle IP Location*, specify the directory location where the DesignStart IP have been installed. Press Finish.
5. Socrates can now be used to configure and build all the DesignStart IP.

## 9.2. DesignStart IP Configuration

To configure an IP and generate the RTL:

1. Create a new project.
2. File -> New -> New Project
  - Select the location and name of the project.
3. Select the relevant IP in the IP Catalog view e.g. Cortex-A Series/Cortex-A5 Processor/Cortex-A5Multiprocessor/r0p1-00relx. (double-click/right-click "Add to Project")



4. This brings up a *Create Configured IP* dialog with configuration options. Enter values for the different fields and press *Finish* on the last page, this will validate the configuration and generate the RTL for the IP.
5. The generated RTL and other information can be found in the <project>/logical directory.

### 9.3. NIC-400 configuration

To configure the NIC-400, select the latest version of the NIC-400 in the IP Catalog view, this will allow you to select the project and provide a name.

Pressing *Finish* will create an empty NIC-400 configuration in the chosen project.

To update this newly created NIC-400, navigate to the project in the Project Explorer and select the NIC-400 configuration.

This brings up an editor which allows for making changes to the NIC-400 configuration. More information can be found in the Socrates and NIC-400 Creation user-guides. These are accessible in Socrates from Help -> Welcome.