



Creating a system for Machine Learning at the Edge

Version 1.0

guide

Non-Confidential

Copyright © 2020, 2024 Arm Limited (or its affiliates).
All rights reserved.

Issue 02

102076_0100_02_en



Creating a system for Machine Learning at the Edge guide

Copyright © 2020, 2024 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	3 April 2024	Non-Confidential	Image updates
0100-01	1 June 2020	Non-Confidential	First release

Proprietary Notice

This document is protected by copyright and other related rights and the use or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited ("Arm"). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document.

The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A

PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. Please follow Arm's trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on <https://support.developer.arm.com>

To provide feedback on the document, fill the following survey: <https://developer.arm.com/documentation-feedback-survey>.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Contents

1. Overview.....	7
2. The possibilities of a dual Cortex-A53 SoC with a Mali GPU.....	8
2.1 Machine Learning at the edge.....	8
2.2 Software support.....	9
2.3 Case study.....	9
2.4 Smart devices.....	10
3. System diagram.....	11
4. Configuring and connecting the Cortex-A53 processors.....	13
4.1 Configuration.....	13
4.2 Connections.....	13
5. Configuring and connecting the Mali-G52.....	15
5.1 Configuration.....	15
5.2 Connections.....	15
6. Using a CCI-500 in the SoC.....	16
6.1 Configuration.....	16
6.2 Connections.....	16
7. Using a NIC-400 in the SoC.....	17
7.1 Configuration.....	17
7.2 Connections.....	18
8. Configuring and connecting the GIC-500.....	19
8.1 Architectural overview.....	19
8.2 Configuration.....	19
8.3 Connections.....	20
9. Configuring and connecting the MMU-500.....	21
9.1 Functionality.....	21
9.2 Configuration.....	22

9.3 Connections.....	22
10. Configuring and connecting the TZC-400.....	23
10.1 Overview of the TZC-400.....	23
10.2 Configuration.....	24
10.3 Connections.....	24
11. Configuring and connecting the ADB-400.....	25
12. Using the SoC-400 to create a debug subsystem.....	26
12.1 Serial Wire JTAG Debug Port.....	28
12.2 Debug Access Port Bus Interconnect.....	28
12.3 APB Access Port.....	28
12.4 AXI Access Port.....	28
12.5 APB Interconnect.....	29
12.6 Cross Trigger Matrix.....	29
12.7 Cross Trigger Interconnect.....	29
12.8 Trace funnel.....	29
12.9 Trace replicator.....	30
12.10 Embedded Trace Buffer.....	30
12.11 Trace Port Interface Unit.....	31
13. Smaller IP.....	32
13.1 BP140 AXI Internal Memory Interface.....	32
13.2 PL011 UART Universal Asynchronous Receiver/Transmitter.....	32
13.3 PL061 General Purpose Input/Output.....	32
13.4 Dual timer.....	33
13.5 Watchdog timers.....	33
14. Clock and power management in an SoC.....	34
14.1 Low Power Interfaces.....	34
14.2 PCK-600 components.....	35
14.3 Usage example.....	37
15. Related information.....	38
16. Next steps.....	40

1. Overview

This guide is for a system designer, possibly with access to the Arm Flexible Access program. This guide will help you to develop a System on Chip (SoC) that can perform machine learning at the edge. The SoC that is presented in this guide can handle machine learning tasks related to image recognition. Image recognition is a reasonably complex machine learning task, which usually requires more performance than, for example, keyword recognition.

The guide is also relevant to system designers who want to create an SoC for a high-end smart device, for example a smartphone.

Specifically, the guide explains:

- Why specific pieces of IP were chosen for this SoC
- How to configure the pieces of IP to use them in this SoC
- How to connect the pieces of IP together

The aim is to provide a broad view of how the pieces of IP work together. Use the SoC presented in this guide as an example. It is expected that you customize the SoC to suit your exact requirements. This involves further understanding and configuration of each piece of IP, the possible removal of some IP, or the addition of extra IP.

This guide uses IP from the [Arm Flexible Access program](#). The Arm Flexible Access program provides low-cost access to a wide range of Arm IP, so that you can experiment and design with a complete IP portfolio. If you have a license for the Arm Flexible Access program, or hold licenses for the individual pieces of IP, use this guide as a starting point. You can then begin to take practical steps with the IP and tweak the design to suit your individual requirements.

If you do not have any licenses, this guide still provides you with a unique overview. You can then go deeper into the specifics of any individual piece of IP included in the SoC.

2. The possibilities of a dual Cortex-A53 SoC with a Mali GPU

This SoC design highlights the performance that is available in an SoC designed from IP within the Arm Flexible Access program. Before being included in the Arm Flexible Access program, the Cortex-A53 already had a reputation as a widely used, high-end processor with 64-bit capabilities. In addition to being high performing, the Cortex-A53 has low-power usage.

There are two Cortex-A53 processors in this SoC, which substantially increases the performance that is available. The combination of high-performance with low-power usage gives a range of scenarios under which you could use this SoC. Specifically, IoT and mobile devices could benefit from using a system based around this design. The next two sections explore a machine learning use case and a smart device use case.

2.1 Machine Learning at the edge

Machine Learning (ML) performs computational tasks by recognizing patterns and making inferences. An inference is a process of applying models, that are built using sample data, to accomplish a defined task. For example, the task could be image recognition in a frame that is received from a camera. Building the models involves a process that is called training. ML algorithms can continue to learn after the models have been built. Therefore, the algorithms can improve over time and adapt to changes.

ML is moving out of the cloud and into the devices that gather the data. This trend is called ML moving to the edge. The reasons for this trend include efficiency, speed, privacy, and security. The emergence of connected devices in new areas, like advanced autonomous cars, is also accelerating the process.

This SoC can support Machine Learning at the edge. This means that the analysis is done in the same place that the data is collected. This approach represents a marked alternative to sending the data to the cloud for analysis. Eliminating the delay involved in bouncing information to the cloud and back helps give the real-time responses that an end user requires. The solution also works when the cloud is unreachable.

2.2 Software support

Arm provides software platforms to complement a system that has the hardware capabilities to run a Neural Net (NN). The following table gives a brief description of two Neural Net software platforms Arm provides.

Software	Description
Arm NN	<p>An inference engine that bridges the gap between existing NN frameworks and underlying Arm IP, including the Cortex-A53 and Mali-G52. Arm NN works with models trained by existing neural net frameworks like:</p> <ul style="list-style-type: none">• Caffe• Tensorflow Lite• ONNX and PyTorch <p>The most recent version of Android is supported through the NNAPI. This API enables performance acceleration through Mali GPUs, Ethos-N NPU's, and Cortex-A CPUs.</p> <p>Importantly, Arm NN abstracts the details of the underlying Arm processor IP. This abstraction allows NN frameworks to use the latest hardware features without the need to port between platforms and generations. Execution of ML algorithms is optimized and can run on a multiprocessor.</p> <p>The Arm NN SDK is supplied as open-source software and enables ML workloads on Android and Linux edge devices.</p>
Arm Compute Library	<p>A convenient repository of low-level kernels that developers can use to accelerate their algorithms and applications. The functions have been implemented for:</p> <ul style="list-style-type: none">• The Arm Cortex-A family of CPUs• The Arm Mali family of GPUs

2.3 Case study

To be effective, inferences must be completed within time constraints. These constraints mean that the performance of a system determines what kind of inference can be completed on time. For example, keyword detection is less expensive than voice and image recognition. Autonomous driving is even more expensive than voice and image recognition.

In terms of machine learning, the inclusion of a six-core Mali-G52 in the SoC gives an advantage. The system is potentially capable of image recognition. Imagine a camera on a door that provides access when it recognizes the face of a person. For a workable solution, the response must be instant as soon as a human is perceived. The system must be able to make five inferences a second. This figure is the inference rate, which is also referred to as the frame rate. In other words, the inference must complete in 200ms. For face unlocking using SSD-Mobile Net v1, we estimate that each inference would take about 20ms. This figure is very usable, because the system can complete 50 inferences a second or run other workloads sequentially.

2.4 Smart devices

A high performance yet energy efficient SoC is ideal for powering a sophisticated smart device. Ultimately, a smart device must run an operating system, and this SoC can support this requirement.

The inclusion of a 6-core Mali-G52 processor allows the SoC to bring premium visual experiences to the end user.

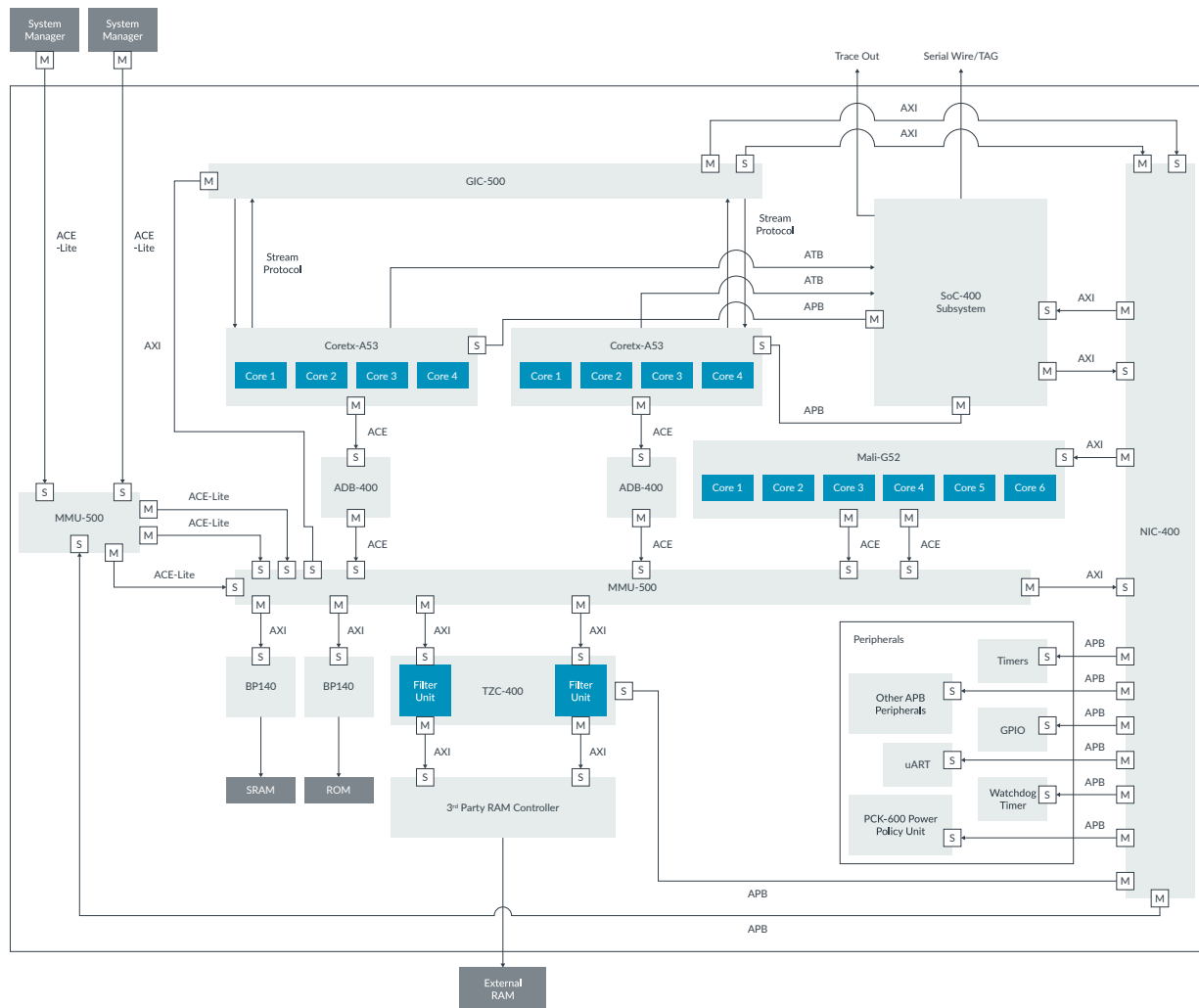
You could use this SoC design for any system that requires high-end graphics capabilities, including high-end IoT devices. For example, you could use this SoC for:

- A smartphone
- A fridge with a touch-screen interface
- A printer with a touch-screen interface

3. System diagram

This section contains a diagram showing how the different pieces of IP in the dual Cortex-A53 SoC connect to each other. The diagram also shows external connections to system masters and RAM.

Figure 3-1: System diagram



The following pieces of IP are used in the preceding figure:

- Cortex-A53 processor
- Mali-G52 graphics processor
- CoreLink CCI-500 Cache Coherent Interconnect
- CoreLink NIC-400 Cache Coherent Interconnect
- CoreLink GIC-500 Generic Interrupt Controller

- CoreLink MMU-500 System Memory Management Unit
 - CoreLink TZC-400 TrustZone Address Space Controller
 - CoreLink ADB-400 AMBA Domain Bridge
 - CoreSight SoC-400 Debug and Trace
 - BP140 AXI Internal Memory Interface
 - PL011 UART Universal Asynchronous Receiver/Transmitter
 - PL061 General Purpose Input/Output
 - Corstone-201 Foundation IP, which contains all the other Arm IP mentioned in this section
-



The Arm Flexible Access program supplies all the IP in the preceding list.

The Configuration sections of this guide explore how the pieces of IP in this SoC are configured. The Connections sections of the guide explore how these IP are connected to each other.



The sections in this guide that relate to connections assume that Q-Channel or P-Channel Low-Power Interfaces (LPIs) are available for each piece of IP. [Clock and Power Management in an SoC](#) explores LPIs in more detail.

4. Configuring and connecting the Cortex-A53 processors

This system is based around two Cortex-A53 clusters, which provide high performance coupled with low-power usage.

4.1 Configuration

In this SoC, the following configuration options were chosen for the Cortex-A53 clusters:

- Both clusters have the maximum of four cores. You can choose between one and four cores for each Cortex-A53 cluster.
- Each cluster has an AXI Coherency Extensions (ACE) interface that connects to a CCI-500. You can choose to have either a Coherent Hub Interface (CHI) or an ACE interface for each Cortex-A53 cluster.
- Each cluster is configured to integrate with either an external GICv3 or an external GICv4 distributor, in this case the GIC-500. You must enable the external Generic Interrupt Controller (GIC) interface for this option. The option to integrate with an external GICv2 distributor component is possible. If you use this option, the internal GIC must be disabled.
- Each cluster supports a Floating Point Unit (FPU) and Neon. We recommend including this functionality for a system on which you intend to run Linux.
- Each core has a L1 cache of 32KB. You can configure the L1 caches for a core to be between 8KB and 64KB.
- The Cortex-A53 clusters share an L2 cache of 1024KB. You can configure the L2 cache for a cluster to be between 128KB and 2048KB.
- All L1 and L2 caches have Error Correcting Code (ECC) included. This feature provides fault protection capability for the caches.

4.2 Connections

Each Cortex-A53 cluster includes the interfaces that are shown in the following table:

Interfaces	Description
ACE master	Each cluster has an ACE master interface that connects to the CCI-500. This interface allows the cores within the cluster to access memory, peripherals, and other components.
Stream protocol	Each cluster has a stream protocol interface that can connect to an external interrupt controller. In this case, the external interrupt controller is a GIC-500.
Debug APB3 slave	Each cluster has an APB3 debug slave interface. This interface enables a debug control subsystem, in this case a CoreSight SoC-400 subsystem, to access system resources. For example, the subsystem can use this interface to set watchpoints and breakpoints.

Interfaces	Description
ATB master	Each core has an AMBA Trace Bus (ATB) master interface. These interfaces transmit trace data from the Embedded Trace Macrocell (ETM) of each core. ETMs capture the execution of a program running on a core and this information is known as trace data. In this SoC, the trace data is sent to a CoreSight SoC-400 subsystem. Components in the subsystem store trace data on-chip and off-chip. External analyzers can decompress the data as required.
ACP	Each cluster has an Accelerator Coherency Port (ACP). This interface provides access for non-cached masters, such as a DMA controller. This interface only supports a subset of AXI transaction types and is not used in this SoC.
MBIST	Each cluster has a Memory Built-in Self-test (MBIST). This interface supports performing a manufacturing test of the memories that are embedded in the Cortex-A53 process.
DFT	Each cluster has a Design For Test (DFT) interface. This interface enables an industry standard Automatic Pattern Generation (ATPG) tool to test logic outside of the embedded memories.
Cross-trigger	Each cluster has a cross-trigger interface. This interface allows you to connect, through a Cross Trigger Matrix, to the cross-trigger channel interfaces of other clusters, and Cross Trigger Interface components in the system. A Cross Trigger Matrix provides: <ul style="list-style-type: none">• The connectivity that supports cross-cluster halting.• The connectivity to communicate a trace collection trigger signal for a trace subsystem.

5. Configuring and connecting the Mali-G52

Our example SoC benefits from the inclusion of a Mali-G52 graphics processor. This GPU is based on the Bifrost architecture and is designed to bring premium visual experiences to mainstream mobile devices. In addition, the Mali-G52 has a significant Machine Learning capability.

5.1 Configuration

In this SoC, the Mali-G52 has been configured to have six cores. This number of cores provides high performance when using the Mali-G52 for Machine Learning tasks.

You can choose to have one, two, three, four, or six cores in a Mali-G52. The choice of core affects the size options that are available for the L2 cache. For example, a two-core Mali-G52 GPU has the option of either a 128KB cache or a 256KB cache. Three, four, and six-core Mali-G52 clusters have the option of having two 256K L2 caches. For our SoC, we chose two 256K L2 caches for the Cortex-G52 cluster.

5.2 Connections

Each Mali-G52 includes the interfaces that are shown in the following table:

Interface	Description
ACE master	A Mali-G52 with L2 caches has two ACE master interfaces. In this SoC, these two interfaces connect to the CCI-500. These interfaces allow the GPU to access internal and external memory in the system.
AXI4 slave	A Mali-G52 has an AXI slave interface. In this SoC, this interface connects to the NIC-400. The interface provides access to the control and status registers for the GPU driver running on the CPU. The driver uses the GPU control registers to configure the graphics system to start and control multiple concurrent jobs running on the GPU.
Interrupt	A Mali-G52 can send interrupt requests. These requests are usually made to signal that a graphics job is complete, but also whenever a graphics job fails. In this SoC, the interrupt line connects to the GIC-500, which distributes them to the Cortex-A53 cores, so they can act on them.
Q-Channel	A Mali-G52 cluster has a Q-Channel port. This port allows a clock controller to query the GPU about whether the clock domain containing the GPU can be shut down.

6. Using a CCI-500 in the SoC

The SoC requires an interconnect from the CoreLink Cache Coherent Interconnect (CCI) family. These interconnects provide ACE interfaces that the Cortex-A53 and Mali-G52 must connect to. Because the CCI-400 only provides two ACE slave masters, a CCI-500 is required.

The CoreLink CCI-500 provides:

- Between one and four ACE slave interfaces
- Between zero and six ACE-Lite slave interfaces
- Between one and four AXI4 master memory interfaces
- Between one and two AXI4 master system interfaces

6.1 Configuration

In this SoC, the CoreLink CCI-500 is configured to have:

- Four ACE slave interfaces
- Two ACE-Lite slave interfaces
- Four AXI4 master memory interfaces
- Two AXI4 master system interfaces



If you need more than two AXI4 master system interfaces, and [striping](#) is not enabled, you can use memory interfaces as system interfaces.

6.2 Connections

The following table shows the IP that CCI-500 interfaces connect to in this SoC:

Interface	Number	Connection
ACE master	4	Connect to the Cortex-A53 clusters, through ADB-400s, and the Mali-G52
ACE-Lite slave	4	Connect to the MMU-500 and GIC-500. The GIC-500 AXI master interface can use an ACE-Lite interface.
Memory AXI4 master	4	Connect to two BP140 memory controllers and two third-party RAM controllers, through a TCZ-400
System AXI4 master	1	Connect to the NIC-400

7. Using a NIC-400 in the SoC

The CCI-500 does not provide all the interfaces that this SoC requires. The SoC needs more AXI interfaces and APB interfaces, which the CCI-500 does not support.

The solution requires a second interface that supports APB interfaces and a larger number of AXI slave and master interfaces. The CoreLink network interconnect family offers the NIC-400 and NIC-450 to help with this.

The CoreLink NIC-400 is:

- Highly configurable
- Offers [Network on Chip](#) (NoC)-like properties
- Provides fully configurable, hierarchical, low latency, and low-power connectivity for AXI, AHB-Lite, and APB interfaces
- Consists of up to 128 masters and 64 slaves of AMBA protocols. AXI and AHB-Lite protocols are supported for the master and slave interfaces. The master interfaces also support APB protocols.

7.1 Configuration

To configure an NIC-400, you must use Socrates. Socrates is available as part of the Arm Flexible Access program. The following list broadly covers the steps that you need to follow to configure an NIC-400:

1. Define the high-level specification, which covers the overall look and behavior of the NIC-400. The high-level specification includes clock domains, slave interfaces, and master interfaces. To define the interfaces, select the protocol and choose appropriate values for data width, ID width, and other parameters.
2. Create the memory map. This map enables a transaction to be sent to the correct interface. For each address, the NIC-400 can determine which port to use.
3. Set the paths. These paths determine which interfaces can talk to each other. The paths allow Socrates to optimize the NIC-400. Setting paths also increases the security of the NIC-400 by facilitating physical path removal between non-communicating interfaces.
4. Generate the microarchitecture. During this process, Socrates checks the configuration is valid and creates internal connections for the NIC-400. The internal blocks are connected using switches. The switches match the paths between the interface that you previously defined in step 3.
5. Edit the microarchitecture to obtain more performance. When generating microarchitecture, Socrates optimizes the result for area. This step gives you a chance to look at performance. In addition, you can also add buffering, add register slices, and resolve deadlocks.
6. Perform a full validation to check any changes that you made. Designs are run against the Design Rule Check (DRC). The DRC checks whether an NIC-400 design is valid.

7. Change the high-level specification as required and feed the result back into the microarchitecture. At this stage, changes like this are still possible and can also be validated. The design is now finalized and ready for the next step.
8. Generate the RTL. The top-level RTL contains the clock ports, and slave and master interfaces. The NIC-400 is now ready for connection to other IP.

In this SoC, the NIC contains five AXI interfaces and eight APB interfaces.

7.2 Connections

The following table shows the IP that NIC-400 interfaces connect to:

Interface	Description
ACE master	Connect to the: <ul style="list-style-type: none">• GIC-500• Mali-G52
AXI4 slave	Connect to the: <ul style="list-style-type: none">• GIC-500• CCI-500• SoC-400 subsystem
APB master	Connect to the: <ul style="list-style-type: none">• SoC-400 subsystem• TZC-400• MMU-500• Timers• GPIO• UART• Watchdog timer• PCK-600 Power Policy Unit



Any extra APB peripherals connect to APB master interfaces.

8. Configuring and connecting the GIC-500

Our example SoC requires an external Generic Interrupt Controller (GIC). Both the GIC-400 and GIC-500 are external. This SoC benefits from the inclusion of a GIC-500, which is based on a later architecture than the GIC-400 and provides more advanced interrupt options.

8.1 Architectural overview

The GIC-500 is based on the GICv3 architecture. The GIC-500 receives interrupts from, for example, a GPIO, UART, or a peripheral. Depending on how the GIC-500 is configured, the interrupts are managed and distributed to up to 32 clusters in an SoC. Each cluster can have up to eight cores. The GIC-500 interfaces with clusters rather than cores. Interrupts are, however, directed to specific cores, and the GIC-500 receives notification when a core activates them.

The GIC-500 can support 960 Shared Peripheral Interrupts (SPIs). These SPIs reach the GIC-500 from peripherals through physical inputs on standard interrupt lines.

The GIC-500 can receive up to 16 Software Generated Interrupts (SGIs). These SGIs are received through the programmable slave interface of the GIC-500.

The GIC-500 also supports Locality-specific Peripheral Interrupts (LPIs), which are typically used for peripherals that produce message-based interrupts. Compared with SPIs, you can have a far larger number of LPIs than SPIs for the same area on the silicon die. This is because LPIs are only cached and not stored in the GIC-500. LPIs are generated when peripherals write to the Interrupt Translation Service (ITS) through the programmable slave interface. The ITS also provides interrupt ID translation that allows the possibility of a virtual machine directly owning a peripheral.

8.2 Configuration

You can configure the following on a GIC-500:

- The number of supported clusters
- The number of supported cores within a cluster *The number of supported SPIs
- Whether LPIs are supported
- Whether legacy mode is supported

Therefore, you can limit the size of the GIC-500 in your SoC by excluding functionality that you do not require. For example, if you know that you only need 32 SPIs, you can save space by specifying this amount.

8.3 Connections

The following table describes the interfaces available on the GIC-500:

Interface	Description
Physical interrupt signals	An interface that allows physical interrupts to be received. The interrupts originate from peripherals and other IP on the SoC.
Stream protocol master	One of a pair of AXI-Stream interfaces. The GIC-500 uses these interfaces to send interrupts to a core and receive notifications from a core. Packets are sent through the master interface to a specific core within the cluster.
Stream protocol slave	One of a pair of AXI-Stream interfaces. The GIC-500 uses these interfaces to send interrupts to a core and receive notifications from a core. The slave interface receives notification when a core activates an interrupt.
AXI slave	A slave AXI interface allowing software to program and configure the GIC-500. Software can also generate SGIs using this interface. Peripherals can use it to generate LPIs.
Master	A master AXI interface allowing the GIC-500 to access the main memory on the SoC. This interface is available if the ITS is present and LPIs are supported. In these cases, the GIC-500 must access information in the form of tables that are held in memory. Because you can have thousands of LPIs, storing LPI-related information requires the use of the main memory.

9. Configuring and connecting the MMU-500

This SoC allows external system masters to connect to it, which requires the inclusion of an MMU-500. The MMU-500 adds a flexibility to the SoC, which can be used as required.

9.1 Functionality

The MMU-500 is a System MMU. For system masters, a System MMU performs a functionality similar to the functionality that CPU MMUs perform for CPU cores. This functionality involves the translation of virtual addresses into physical addresses. Both types of MMU use the same translation tables, which are defined during memory virtualization. These translation tables are stored in the main memory of the SoC.

The MMU-500 effectively enables one or more system coherent masters to operate in the same virtual address space as the Cortex-A53 clusters. The system masters could be a:

- DMA
- Display processor
- GPU
- Custom accelerator
- PCIe interface

The MMU-500 can perform multiple address translations from different system masters at the same time. Context switching is not required.

The MMU-500 is composed of two main blocks, the Translation Buffer Unit (TBU) and the Translation Control Unit (TCU). One MMU-500 instance can have up to 32 TBUs but always has a single TCU.

TBU master interfaces are paired with a slave interface. Each system master requires a TBU master and slave pair. The TBU receives system master transactions, which are tagged with a virtual address, on the allocated slave interface. After translating the addresses, the TBU accesses the memory using the corresponding master interface.

In an SoC, the TBU is expected to be physically near the connected system master. This requirement means that the TBU is in the same clock or power domain as the system master. The TBU also includes a Translation Lookaside Buffer (TLB) to cache translation table information. The purpose of the buffer is to speed up the translations by caching translation table descriptors.

The TCU is a central control block that is responsible for reading translation tables from memory through its main master interface. The translation information is sent to the TBU, which puts them in the TLB. The TCU also includes extra caches and implements the MMU-500 programmers view, which is a set of configurable registers.

9.2 Configuration

The main configuration parameters for each TBU are:

- The TLB depth. Each TLB can have up to 128 entries. One entry is used for each translation table descriptor.
- The write data buffer depth. For a memory write, this buffer holds the data to write.
- The number of transactions that can be handled in parallel. The maximum value is 16.

The main configuration parameters for the TCU are:

- The depth of the various caches
- The maximum number of translations (translation table walks) that can be handled in parallel

9.3 Connections

The following table describes the interfaces available on the MMU-500:

Interface	Description
TBU ACE-Lite slave	The MMU-500 has up to 32 ACE-Lite slave interfaces. Coherent masters connect to these interfaces and send virtual address requests.
TBU ACE-Lite master	The MMU-500 has up to 32 ACE-Lite master interfaces that pair with the slave interfaces. The master interfaces connect to the CCI-500 and can read and write to memory using the translated virtual addresses. Each coherent master uses one slave and one master interface on the MMU-500. In this way, the MMU-500 intercepts traffic driven to it by each coherent master and enables access to the correct physical address in memory.
TCU ACE-Lite master	The MMU-500 has a single ACE-Lite master interface that obtains translated addresses from the translation tables, which are stored in the main memory.
TCU ACE-Lite configuration	The MMU-500 has an AXI4 slave interface that enables software to configure the MMU-500. The configuration process involves setting registers.

10. Configuring and connecting the TZC-400

This SoC design anticipates that significant data, requiring extra access control, is in Non-secure memory. By including the TZC-400, the option is available to protect this data. The protection is slave side, which means that it comes after an interconnect master interface rather than, for example, a system master interface.

10.1 Overview of the TZC-400

The TZC-400 enables you to partition and protect up to eight regions of memory by marking them as read-only or write-only. To assist with this process when the IP is integrated into an SoC, the TZC-400 also has a concept known as a filter unit. Each individual memory port requires one filter unit. You can configure the TZC-400 to have one filter unit, or multiple filter units, depending on the requirements of the system memory. The eight definable regions apply to all filter units. Therefore, the filter view is the same when accessing any memory port through a specific TZC-400 filter unit. In this SoC, there are two memory ports.

Access to the eight regions is also controllable. There are two ways to achieve this control:

- Restricting access to a region according to the Non-secure Access ID (NSAID) of a master
- Marking any region as inaccessible for a filter unit. In other words, you can prevent access to a region through a specific port.

Because the Cortex-A53 MMUs and the MMU-500 give an SoC protection, the TZC-400 increases the level of protection that is available. While Cortex A-series processors support TrustZone, the TZC-400 allows you to add extra security permissions within memory that is marked as Non-secure by TrustZone.

A large video buffer is an example of where these permissions could be useful. It is not ideal to place a large memory requirement, like a video buffer, inside memory that is marked as Secure by TrustZone. A video codec can also be potentially buggy. TrustZone Secure memory is better for smaller data like cryptographic keys. In this SoC, TrustZone Secure memory is in the SRAM and ROM.

Although processor MMUs and the MMU-500 could protect a video buffer, this solution is slower than using a TZC-400. Sometimes, a master does not use an MMU either. In this system, the Mali-G52 does not access memory using an MMU. The TZC-400 ensures that memory protection is in place for all accessing masters.

10.2 Configuration

You can configure the TZC-400 to support one, two, or four filter units. Each memory port requires one unit. This SoC has two memory ports, so the TZC-400 is configured to have two units.

10.3 Connections

The following table describes the interfaces available on the TZC-400:

Interface	Description
ACE-Lite master	An ACE-Lite master interface exists for each filter unit configured. Masters in the system must access memory through these interfaces.
ACE-Lite slave	An ACE-Lite slave interface exists for each filter unit configured. These slave interfaces connect to memory controllers.
APB4 slave programming that i	An APB4 interface that enables software to program the registers of the TZC-400. To ensure integrity, the addresses of these registers must be in an area of memory s marked as Secure by TrustZone.
Interrupt	A physical interrupt line that allows the system to assert when an access attempt fails its security check.

11. Configuring and connecting the ADB-400

The ADB-400 is an asynchronous bridge for two components or systems that are in either a different power domain, different clock domains, or both. An ADB-400 consists of a slave domain and a master domain. The slave domain has a slave interface and the master domain has a master interface.

The system design anticipates that the Cortex-A53 clusters are in a separate power or clock domain to the CCI-500. The design includes an ADB-400 between each Cortex-A53 cluster and the CCI-500. In this SoC, the slave and master interfaces are specified to be ACE. You can configure the ADB-400 slave and master interfaces to use AXI3, AXI4, ACE, and ACE-Lite protocols.

A FIFO buffer is a key part of the logic structure that the ADB-400 uses to bridge between two clock domains. The ADB-400 receives transactions that are clocked with the slave side domain clock. The ADB-400 then releases them in a safe way for the master side domain clock. The depth of the FIFO buffer is configurable.

12. Using the SoC-400 to create a debug subsystem

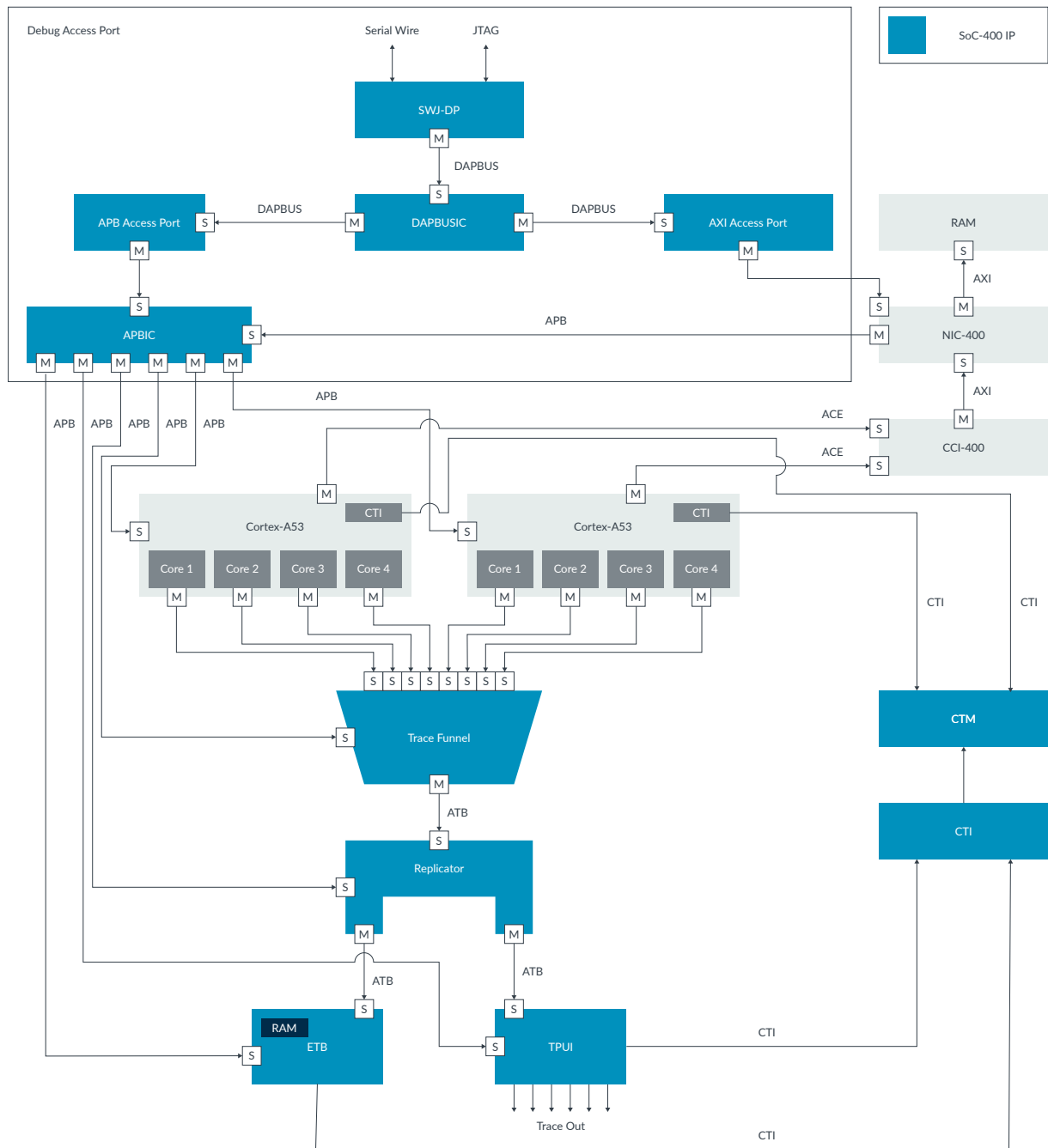
The SoC-400 differs from other pieces of IP shown in the example SoC because it is not a single piece of IP. The SoC-400 instead consists of several components. You can use these components to build an infrastructure to debug, monitor, and optimize an SoC design. These components include buses, control and access components, and trace links. Most of these components are configurable.

System designers use SoC-400 components to create a subsystem that is designed to meet the requirements of a specific SoC. The complexity of the subsystem depends on the SoC which the subsystem is part of. In this SoC, the subsystem collects trace output from the four cores of both Cortex-A53 clusters.



In this guide, the terms SoC and SoC-400 refer to different things. SoC refers to the example dual Cortex-A53 System on Chip, which is the subject of this guide. The SoC-400 is a piece of Arm IP that contains multiple components. The example SoC in this guide contains an SoC-400 subsystem, which is shown as a single entity in [System diagram](#).

The following diagram shows an SoC-400 subsystem that is suitable for our example SoC:

Figure 12-1: System diagram

Now let's explore the components in the SoC-400 subsystem that we have described, including their configuration options and interfaces:

12.1 Serial Wire JTAG Debug Port

A Serial Wire JTAG Debug Port (SWJ-DP) allows you to connect either a Serial Wire Debug (SWD) or JTAG probe to the SoC-400 subsystem, and therefore the SoC itself. This component is accessible from outside the SoC. In other words, the SWJ-DP forms the main entry point into an SoC for debugging it.

In addition to the external connection, the SWJ-DP contains a single DAPBUS master port interface. This interface connects to the Debug Access Port Bus interconnect (DAPBUSIC) in an SoC-400 subsystem.

12.2 Debug Access Port Bus Interconnect

A Debug Access Port Bus Interconnect (DAPBUSIC) allows the SWJ-DP to combine with Access Ports (APs). This provides a route from outside the SoC to the components within.

The DAPBUSIC has the following interfaces:

Interface	Description
DAPBUS slave	Each DAPBUSIC has one slave DAPBUS interface. The SWJ-DP connects to this interface.
DAPBUS master	Each DAPBUSIC has up to 32 master DAPBUS interfaces. These interfaces connect to individual APs providing AHB, APB, AXI, and JTAG interfaces.

12.3 APB Access Port

The APB Access Port provides a single APB master interface, and allows conversion from a DAPBUS interface to an APB interface.

Many SoC-400 components have APB programming interfaces. The APB Access Port and the APBIC allow you to control SoC-400 components from outside the SoC-400 subsystem.

12.4 AXI Access Port

The AXI Access Port provides a single AXI master interface and facilitates conversion from a DAPBUS interface to an AXI interface. AXI-AP is typically used to provide an access path to the SoC memory. You can also see the connection in the [System diagram](#). If the CPUs hang, the AXI-AP can provide another path to the system memory.

12.5 APB Interconnect

The APB Interconnect (APBIC) facilitates multiple master connections to APB interfaces on components within the SoC-400 subsystem. Connections from external CoreSight components in the SoC are also facilitated.

The APBIC has the following interfaces:

Interface	Description
APB slave	The APBIC can have from one to four slave APB interfaces. Only one of these interfaces can connect to an APB-AP component. Other slave interfaces might have connections from IPs that are outside the SoC-400 but are part of the SoC. For example, in the previous diagram, the Cortex-A53 clusters can control the SoC-400 components through the connection that the NIC-400 has with the APBIC. The APBIC connects like a master to the APB slave interfaces on the Cortex-A53 clusters. This design facilitates communication between the SoC-400 subsystem and the rest of the SoC.
APB master	The APBIC can have from 1-64 master APB interfaces. These interfaces connect to and control other SoC-400 components within the subsystem. They can also connect to IP that is external to the subsystem.

12.6 Cross Trigger Matrix

The Cross Trigger Matrix (CTM) combines the trigger requests that are generated from CTIs and broadcasts them to all CTIs as channel triggers. The two Cortex-A53 clusters in our example have an internal Cross Trigger Interconnect (CTI) and CTM. Triggering requests arising internally in a Cortex-A53 cluster are sent to the Cross Trigger Matrix of the SoC-400 subsystem.

12.7 Cross Trigger Interconnect

The CTI provides an interface that enables events broadcasting in the system.

12.8 Trace funnel

A trace funnel can combine up to eight trace sources into one. To combine more than eight trace sources, chain several trace funnels together. This strategy effectively increases the number of trace inputs that are available.



To avoid combinatorial timing loops, a register slice that is a forward, reverse, or full register slice must be instantiated between the cascaded funnels.

In this system, the trace output from all cores of both Cortex-A53 clusters are combined into one trace funnel.

A trace funnel has the following interfaces:

Interface	Description
ATB slave	You can configure the funnel to have two to eight slave ATB interfaces that receive trace data.
ATB master	A master ATB interface that outputs the combined trace data.
APB slave programming	You can configure the funnel to have an APB programming interface. If you do, an APB slave interface is added. You can use this interface to enable and disable slave ATB interfaces at runtime.

12.9 Trace replicator

The trace replicator enables an incoming trace stream to be passed to two trace sinks. A trace replicator has a slave ATB interface that receives the trace stream and two master ATB interfaces that output the combined sources.

A trace replicator has the following interfaces:

Interface	Description
ATB slave	An ATB slave interface receives the combined trace output.
ATB master	Two ATB master interfaces output the trace output.
APB slave programming	The Trace replicator can be configured to have an APB programming interface. If you do, an APB slave interface is added. You can use this interface to: <ul style="list-style-type: none"> Filter the trace passed to each master ATB interface according to the trace ID. Filter out higher bandwidth traces from trace sinks that only support a lower bandwidth such as a TPIU. This filtering allows a high-bandwidth trace sink, for example an ETB, to be enabled at the same time as a lower bandwidth trace. The filtering prevents the lower bandwidth trace sink from slowing down the output of both trace sinks. If the filtering is not used, all trace sinks slow down to match the slowest trace sink.

12.10 Embedded Trace Buffer

The Embedded Trace Buffer (ETB) provides on-chip storage of trace data in RAM. When designing the SoC-400 subsystem, you can configure the size of the RAM for ETB and then implement the required RAM.

The ETB contains a formatter that combines the source data and IDs into a single data stream before storing the data in RAM. The formatter operates in an identical manner to the formatter in the TPIU.

The ETB has the following interfaces:

Interface	Description
ATB slave	Receives the combined trace output to store.
APB master	Allows programming of the ETB through its registers. In addition, you can read the captured trace through this interface.
Cross trigger	Allows you to send and receive cross trigger events to and from a CTI.
MBIST	Enables the testing of the storage RAM.

12.11 Trace Port Interface Unit

The Trace Port Interface Unit (TPIU) formats the trace data that it receives and outputs the formatted data through the pins of the trace port. The Trace Port Analyzer (TPA) can then capture the data. In other words, the trace port in the TPIU provides a route for trace out of the SoC that the TPIU is part of. The TPIU can output patterns over the trace port so that a TPA can tune its capture logic to the trace port. This feature enables the maximization of the speed at which trace can be captured.

The TPIU inserts source ID information into the trace stream. The formatter operates in an identical manner to the formatter in the ETB.

The TPIU has the following interfaces:

Interface	Description
Trace out	Connects to the external trace port pins to facilitate trace exporting.
ATB slave	Receives the combined trace output to export out of the SoC.
APB slave	Allows programming of the TPIU through its registers.
Cross trigger	Allows you to send and receive cross trigger events to and from a CTI.

13. Smaller IP

This section of the guide covers the smaller pieces of IP in the system. Except for the BP140 AXI Internal Memory Interface, the IP in this section connects to the NIC-400 interconnect through an APB interface.

13.1 BP140 AXI Internal Memory Interface

This SoC includes SRAM and ROM. This memory is internal to the SoC and requires an appropriate controller. We chose two BP140 AXI Internal Memory Interfaces.

The BP140 is an internal memory interface that has:

- An AXI slave interface.
- A single-port memory interface that is configurable for SRAM and ROM.



If you require TrustZone protection for Secure memory regions on the SRAM or ROM, place a BP141 TrustZone AXI Memory Interface between the CCI-500 and the BP140. The BP141 is also available in the Arm Flexible Access program.

13.2 PL011 UART Universal Asynchronous Receiver/Transmitter

When configuring the PL011 UART Universal Asynchronous Receiver/ Transmitter, you must choose the Baud rate, stop bits, parity bits, and data bits.

The PL011 has an APB slave interface that is used both for configuration and for reading and writing data.

The clock signal must be free running for the UART to operate. The signal must never be gated. If an APB bridge is used to connect to the UART, the clock to the bridge can only be gated if the UART is disabled.

13.3 PL061 General Purpose Input/Output

The PL061 General Purpose Input/Output has eight I/O pins that you can configure:

- To be input or output
- To generate interrupts using edge or level detection on inputs

You can read and write from each of the pins using APB access.

13.4 Dual timer

In this SoC, the dual timer is configured to consist of two programmable 32-bit down counters, which generate interrupts when they reach 0. Providing a dual timer allows you to use one timer as a Secure timer and one timer as a Non-secure timer. The timers can run in one of the following modes: free-running, periodic, and one-shot.

The dual timer provides an APB slave interface.



The dual timer is a component from the Cortex-M0/M0+ System Design Kit (CMSDK). The CMSDK is part of the Corstone-201 Foundation IP.

13.5 Watchdog timers

In this SoC, there are two watchdog timers. Each watchdog consists of a 32-bit down counter that generates an interrupt, which is used for a reset event. The watchdog, when running, must be periodically reset to prevent it generating the reset event. If a core is locked up, the watchdog times out and results in the watchdog resetting the core. This mechanism provides a way to recover from software crashes.

One watchdog is mapped to the Secure world and the other watchdog is mapped to the Non-secure world. The Secure world watchdog timer can reset the system. However, the Non-secure world watchdog timer must normally not be allowed to reset the system directly. Instead, on a reset timeout, the Non-secure watchdog requests that the Secure world performs a system reset on its behalf.

The watchdog provides an APB slave interface.



The watchdog timer is a component from the CMSDK. The CMSDK is part of the Corstone-201 Foundation IP.

14. Clock and power management in an SoC

Depending on the complexity of a system, a system can have multiple clock and power domains.

A clock domain is defined as a group of IP components which run from a common clock. If all components are in a state of quiescence, the clock can be removed from the components. This process is also known as clock gating. Clock gating is one technique that you can use to reduce dynamic power consumption.

A power domain is defined as a group of components which can power up or down together. If all components in a power domain are in a state of quiescence, then they can be powered down together to save power.

In an SoC, clock domain and power domain management involve building an infrastructure to handle communication and decision making in these areas. The PCK-600 contains a set of components that you can use to build this infrastructure.



Information about specific clock and power domains for this SoC is beyond the scope of this guide. However, we anticipate that this SoC contains more than one clock domain and more than one power domain. Components from the PCK-600 help to create the control infrastructure for these domains. How these components can link together is explained in a non-specific way in Low-Power Interfaces, PCK-600 components and Usage example.



In Low-Power Interfaces, PCK-600 components and Usage example, the term device is synonymous with IP.

14.1 Low Power Interfaces

Low-Power Interfaces (LPIs) facilitate communication between controller and devices. Specifically, this communication enables a safe transition between different power states. There are two types of LPI:

- Q-Channel LPI
- P-Channel LPI

A Q-Channel interface can be used to control transition between two main states, which are run and stop. A P-Channel can be used to control transition between multiple states. For example, a device could have the following power states:

- On mode. The device is fully operational.
- Retention mode: The device logic is turned off. However, the RAM blocks are in retention, which preserves state.
- Off mode: The device is fully shut down, and the RAM retains no state.

Q-Channels can be used to control quiescence of clock domains and power domains. P-Channels are typically used for controlling power domains.



For clarity, LPIs are not shown in any of the previous figures or mentioned in the interface tables. However, you can assume the IP is managed in suitable clock and power domains for the SoC using LPIs and, potentially, PCK-600 components.

14.2 PCK-600 components

PCK-600 components:

Low-Power Distributor Q-Channel

The Low-Power Distributor Q-Channel (LPD-Q) allows a Q-Channel controller to control up to 32 Q-Channel interfaces present on the components of a system.

Controllers, like CLK-CTRL, distribute signals to multiple components in a domain using the LDP-Q. For example, a CLK-CTRL can broadcast a request to clock gate all devices in a clock domain. It is important to remember that communication is two way. The CLK-CTRL also receives information about the level of activity in the components so it can decide whether to gate the domain.

When communicating with components, requests can be broadcast in parallel or passed sequentially to each device Q-Channel. You can control how the broadcast is made through configuration parameters.

Low-Power Distributor P-Channel

The Low-Power Distributor P-Channel (LPD-P) allows a P-Channel controller to control up to eight P-Channel interfaces that are present on the components of a system.

Controllers, like the Power Policy Unit (PPU), distribute signals to multiple components in a domain using the LDP-P. For example, a PPU can broadcast a request to transition between power states to all components in a power domain. It is important to remember that communication is two-way. The PPU also receives information about the level of activity in the components so it can decide whether to broadcast a transition state request.

When communicating with components, requests can be broadcast in parallel or passed sequentially to each device P-Channel. You can control how the broadcast is made through configuration parameters.

Low-Power Combiner Q-Channel

The Low-Power Combiner Q-Channel (LPC-Q) allows two or more Q-Channel controllers to control up to 32 Q-Channel interfaces that are present on the components of a system.

The LPC-Q works in a similar way to the LPD-Q, but allows more than one controller to communicate with the devices.

P-Channel to Q-Channel Converter

The P-Channel to Q-Channel Converter (P2Q) converts a single P-Channel into a single Q-Channel.

Power Policy Unit

The Power Policy Unit (PPU) monitors activity information from connected components. Based on this information and a software-programmed power domain policy, the PPU decides whether to place the components of a domain in another power state.

The PPU, using a P-Channel, communicates with a Power Control State Machine (PCSM), which executes power state changes under PPU direction. This process is done while the PPU directly communicates with the components. This means that the PCSM is activated when the devices have transitioned to the correct power state. The scheme is:

- Software programs a policy into the PPU.
- The PPU observes device activity on a device interface. The interface can be either a Q-Channel or P-Channel.
- The PPU uses the device interface to transition device LPIs to the proper state as defined in the policy.
- The PPU controls the PCSM to activate the physical power control hooks, for example to power down the device.

The PCSM is a technology-dependent state machine that you are responsible for implementing. The PCSM handles the sequencing of power switch chains and retention controls. If necessary, you can implement RAM and registers dedicated to the PCSM. If a CLK-CTRL is present on the system, the PPU also communicates with it. This connection enables potential clock gating of the PPU itself. The PPU allows software control of its operations and provides an APB interface for this. For example, software can control the policy that the PPU is running. In System diagram, the PPU is shown as a peripheral.

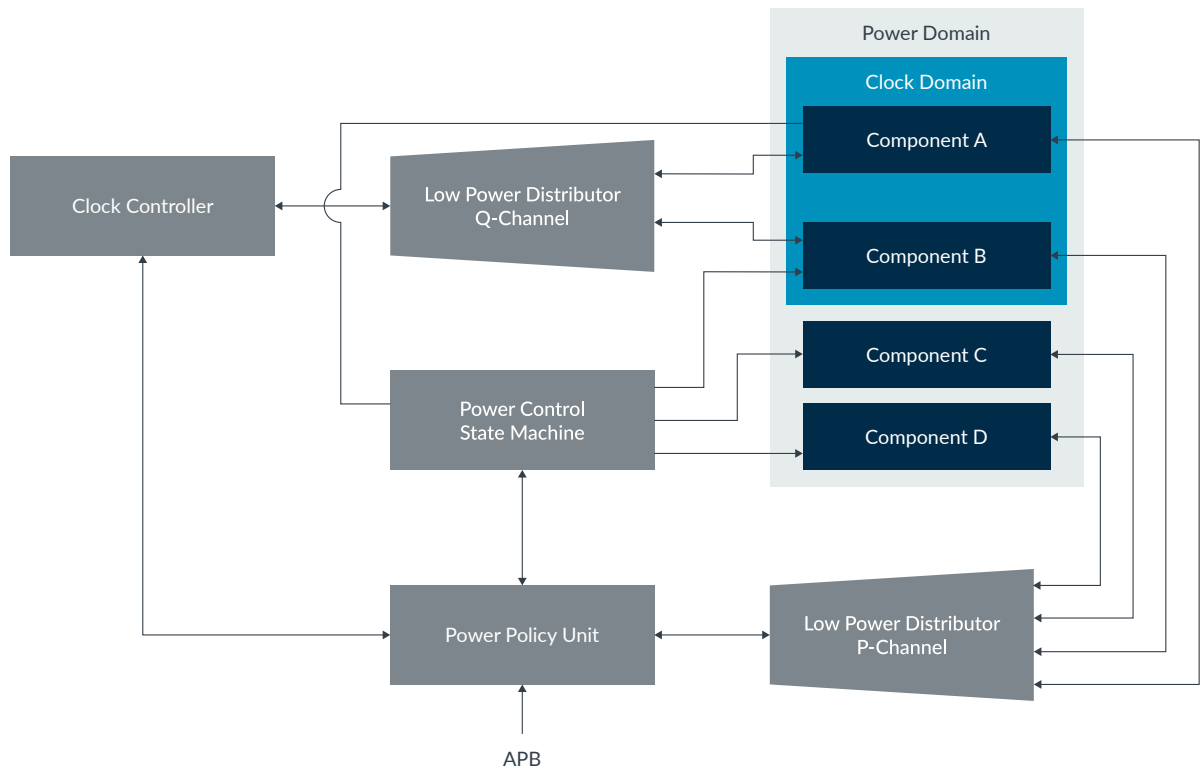
Clock Controller

The Clock Controller (CLK-CTRL) monitors activity information from components that are connected through a Q-Channel interface. Based on this information, the PPU decides whether to gate the components of a domain.

14.3 Usage example

The following figure shows an example of how you could use four of the PCK-600 components:

Figure 14-1: Example of PCK-600 components usage diagram



The components connect with the devices and each other to control a clock domain and a power domain in an SoC. In this example, the clock domain has two components in it. The power domain contains four components. Component C and Component D are also in a clock domain, but this domain is not shown in the figure.



In Arm-designed systems, the hierarchy typically includes power domains that contain one or more clock domains.

15. Related information

Here are some resources related to the different IP in this guide.



The following documents which do not have links are only available with an Arm Flexible Access program license.

CortexA-53 processor:

- [Cortex-A53 MPCore Processor Technical Reference Manual](#)
- Cortex-A53 MPCore Processor Configuration and Sign-off Guide
- Cortex-A53 MPCore Processor Integration Manual

Mali-G52 graphics processor:

- Mali-Gondul GPU Technical Reference Manual
- Mali-Gondul GPU Technical Overview
- Mali-Gondul GPU Configuration and Integration Manual

CCI-500:

- [CoreLink CCI-500 Cache Coherent Interconnect Technical Reference Manual](#)
- CoreLink CCI-500 Cache Coherent Interconnect Configuration and Sign-off Guide
- CoreLink CCI-500 Cache Coherent Interconnect Integration Manual

NIC-400:

- [CoreLink NIC-400 Network Interconnect Technical Reference Manual](#)
- CoreLink NIC-400 Network Interconnect Integration Manual
- CoreLink NIC-400 Network Interconnect Implementation Guide

GIC-500:

- [CoreLink GIC-500 Generic Interrupt Controller Technical Reference Guide](#)
- CoreLink GIC-500 Generic Interrupt Controller Implementation Guide
- CoreLink GIC-500 Generic Interrupt Integration Manual

MMU-500:

- [CoreLink MMU-500 System Memory Management Unit Technical Reference Manual](#)

TZC-400:

- [CoreLink TZC-400 TrustZone Address Space Controller Technical Reference Manual](#)

- CoreLink TZC-400 TrustZone Address Space Controller Implementation Guide
- CoreLink TZC-400 TrustZone Address Space Controller Integration Manual

ADB-400:

- CoreLink ADB-400 AMBA Domain Bridge User Guide

SoC-400:

- [CoreSight SoC-400 Technical Reference Manual](#)
- CoreSight SoC-400 System Design Kit
- CoreSight SoC-400 Implementation Guide
- CoreSight SoC-400 Integration Manual
- CoreSight SoC-400 User Guide

BP140:

- PrimeCell Infrastructure AMBA 3 AXI Internal Memory Interface (BP140) Design Manual
- PrimeCell Infrastructure AMBA 3 AXI Internal Memory Interface (BP140) Technical Overview
- PrimeCell Infrastructure AMBA 3 AXI TrustZone Memory Adapter (BP141) Design Manual
- PrimeCell Infrastructure AMBA 3 AXI TrustZone Memory Adapter (BP141) Technical Overview

UART PL011:

- [PrimeCell UART PL011 Cycle Model User Guide](#)
- PrimeCell UART PL011 Technical Reference Manual

GPIO PL061:

- [Arm PrimeCell General Purpose Input/Output \(PL061\) Technical Reference Manual](#)

PCK-400:

- [CoreSight PCK-400 Power Control Kit Technical Reference Manual](#)
- CoreSight PCK-400 Power Control Kit Configuration and Integration Manual

APB dual timers and watchdog timers:

- Cortex-M System Design Kit Technical Reference Manual

16. Next steps

This guide provides essential knowledge that you must have to design and build a powerful SoC suitable for machine learning at the edge. The IP that is covered in this guide is all available through the Arm Flexible Access program. You can [review the options](#) for the Arm Flexible Access program.

Because this guide only gives a high-level overview, you must further customize your own SoC to meet your requirements. This means that you will need to review all the configuration options for each piece of IP you use. However, customization might also involve removing certain pieces of IP, or adding different IP. You can customize the interfaces on the NIC-400, so that you can add extra IP to your SoC.

A useful next step might involve learning more about the configuration options available for each piece of IP. You could also learn how to integrate and implement the IP. The documents in [Related information](#) can help you do this.