arm

Arm Flexible Access program
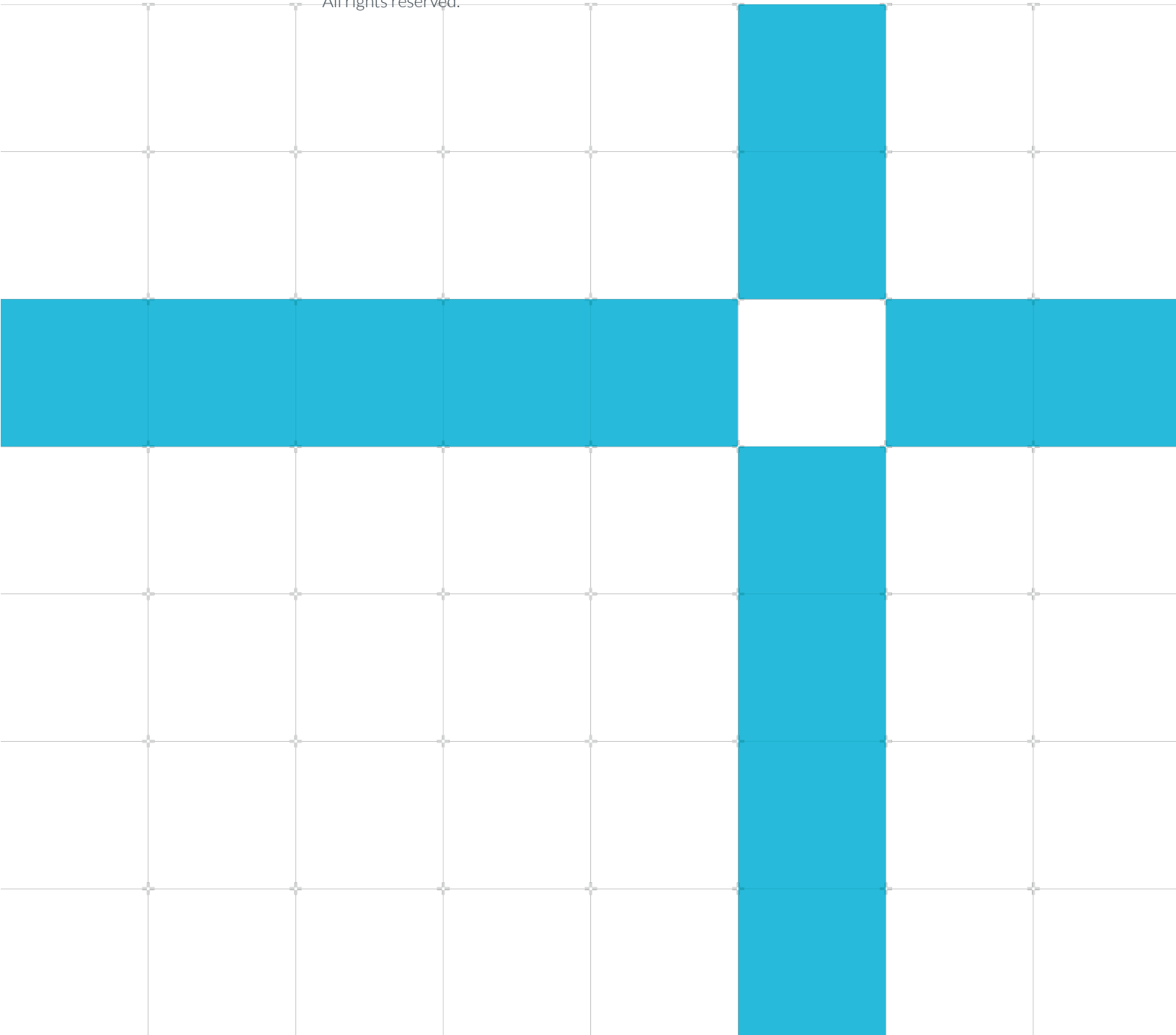
Creating a System for a Secure IoT Device

Issue 01

Non-Confidential

101892

Arm Flexible Access program

Creating a System for a Secure IoT Device

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

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	30 Apr 2020	Non-confidential	First release.

Non-Confidential Proprietary Notice

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.

Product Status

The information in this document is Final, that is relating to developed pieces IP.

Web Address

http://www.arm.com

Contents

1 Overview

This guide is for a system designer, possibly with access to Arm Flexible Access. We assume that you want to develop a System on Chip (SoC) for a secure IoT device, and that you intend the SoC to be used in a smart coffee maker. However, the guide could be relevant for any connected IoT device that provides a basic user interface through a small display. For example, a connected vending machine could present a touchscreen to the user. This type of vending machine could also create and send its own stock orders through its online capabilities.

For purposes of this guide, we assume that you want to take practical steps towards building a system. Taking the coffee machine as an example, the aim is to show you where you can find the required IP, and its related documentation. The aim is not to repeat information but instead to guide you to the correct place to find it and show you how to apply it.

This guide uses IP from Arm Flexible Access. The quickest way to create an SoC solution for a secure IoT device is to use the Corstone-201 foundation IP. Therefore, this guide can also be used if you are getting started with the Corstone-201.

2 The goals of a secure IoT device

Let's begin by thinking about the goals of a secure IoT device. Many of those goals aim to provide a superior experience to the customer. At the same time, the device must install confidence in relation to security. Achieving both aims requires attention at the software level, the firmware level, and in the design of the System on Chip (SoC) powering the device. This guide focuses on the design of the SoC and demonstrates how hardware functionality helps with security.

Imagine that you have the job of designing an SoC that powers a connected coffee machine. Let us start with the functionality this coffee machine is expected to achieve. First, the device must identify itself and communicate securely to both a cloud server and mobile devices. Communication over the Internet requires the use of cryptography.

In addition, the coffee machine must be able to:

• Securely perform firmware updates that are downloaded from the cloud server. During these updates, the coffee machine must decrypt and authenticate the firmware image.

• Monitor itself. The coffee machine must be able to report any service or refill requirements to either the service company or the owner

• Store user details securely including the preferences of the user

• Send usage data, like coffee selection statistics, to the manufacturers for market research

• Download brewing programs for new ranges of coffee beans

• Display advertising for new coffee blends on the screen of the coffee machine

Although this guide uses a connected coffee machine as an example, the functionality that the device requires could apply to many connected devices. For example, most connected devices are expected to send and receive data over the Internet. They also allow a user to interact with them through an app and can securely update their firmware. In this sense, there would be many similarities in the SoC design journey for any secure IoT device.

The final consideration for any secure IoT device is cost. The secure solution that the end user is going to enjoy must be cost effective for the manufacturer to produce.

3 How can I secure my device?

Ensuring that your device is secure involves four things:

- Platform Security Architecture (PSA). A framework offering a step-by-step guide to building in the right level of security for securing connected devices.

- Arm TrustZone. A Security Extension that facilitates hardware-supported isolation.

- CryptoCell and CryptoIsland. Technologies offering various cryptography and platform related security services.

- Mitigating against physical attacks

Whether you must consider the last two items in the list depends on the security level that you require. In the following sections, we explore how each item affects your hardware and software development.

3.1 Platform Security Architecture

The Platform Security Architecture offers a framework for securing connected devices. The PSA provides a step-by-step guide to building in the right level of security for a device. We recommend familiarizing yourself with the framework before you begin any SoC design for a secure IoT device.

The PSA is broken down into four stages as a project progresses:

- An analysis stage where device assets are analyzed and threats are assessed. This process defines the security requirements.

- An architecture stage where the device is designed based on the security requirements that are identified in the first stage. The design process includes the SoC, firmware, and software. This guide helps you to complete this stage by concentrating on the IP that a secure IoT device needs. This guide also explains how the selected IP are put together.

- An implementation stage that involves the implementation of software that meets the security requirement. The software must also work with the hardware that is defined during the architecture stage specifications.

- A certification stage where checks are made, through a PSA Certified scheme, that the product adheres to the security requirements.

We recommend learning more about the four stages in the Platform Security Architecture Overview Whitepaper. The whitepaper describes the Trusted Base System Architecture (TBSA), which is a set of SoC requirements for Armv6-M, Armv7-M, and Armv8-M processors.

The PSA Trusted Base System Architecture for M (TBSA-M) uses the Trusted Base System Architecture as a theoretical foundation. Putting together a secure IoT system involves selecting pieces of Arm IP based on this foundation.

The PSA Whitepaper provides example Threat Models and Security Analyses for three common IoT use cases:

- Asset tracker

- Smart water meter

- Network camera

The examples provide some use cases that you can compare against the secure IoT coffee maker use case. Each example contains a set of security objectives that mitigates one or more of the identified threats. Security Model (PSA-SM) defines a security architecture that is designed to address a generic set of threats.

To allow your firmware and software development to start before the hardware platform is available, use the following specifications and guidelines:

- A PSA Firmware Framework for M (PSA-FF-M) specification, which defines a standard programming environment and firmware interfaces. These definitions are for implementing and accessing security services within the Root of Trust for a device.

- The requirements for a Trusted Boot and Firmware Update (PSA-TBFU). The Trusted Boot requirements describe how to validate whether an image is authorized before booting it. The firmware update requirements describe how to validate an update before storing it to flash memory.

- Trusted Firmware-M (TF-M), which is an open-source firmware implementation for Cortex-M series processors.

3.2 Arm TrustZone technology

TrustZone for Armv8-M is a Security Extension for the Armv8-M architecture. Before TrustZone was introduced, SoC integrators still had to ensure the authenticity of secure assets. Integrators often created physically segregated Secure and Non-secure worlds with controlled access between the two worlds.

TrustZone provides the segregation that is necessary to support confidentiality. Because TrustZone is already defined in the processor and system architecture, the implementation of TrustZone is at the hardware level. In other words, TrustZone facilitates hardware-supported isolation.

TrustZone works by enabling regions in memory to be marked as Secure or Non-secure, which gives a Secure and a Non-secure memory world within TrustZone. Data and instructions that are stored in the Secure memory world are marked as Secure. Data and instructions that are stored in the Non-secure world are marked as Non-secure.

On a processor that supports TrustZone, there are two separate processor states: Secure state and Non-secure state. Code running in the Secure state can execute Secure instructions, and can also access Secure and Non-secure data. When code is running in a Non-secure state, the code can only execute Non-secure instructions and access Non-secure data. The processor can transition from Secure state to Non-secure state and back again. Transitioning from Non-secure code execution to Secure code execution is a controlled process. It is not possible to branch from Non-secure code to code at a random Secure address.

Two types of unit are responsible for defining the memory regions:

- An Implementation Defined Attribution Unit (IDAU). These units are implemented externally to the processor during SoC integration and allows the security partitioning of the memory map to be hardwired at integration. This approach reduces hardware and software overheads that are involved when defining security mapping for memory regions. You can think of an IDAU as defining default regions.

- A Security Attribution Unit (SAU). Unlike an IDAU, you can define SAU regions at runtime using registers. This enables a SAU to override IDAU default regions.

For TrustZone to function, either an IDAU, or an IDAU and a SAU, is required in the processor implementation. Otherwise memory regions cannot be defined.

You can also partition each security world into further subregions using the Memory Protection Units (MPUs) within each world. One set of MPUs defines the regions within Secure memory, and the other MPU defines the regions within Non-secure memory. You can use the

MPUs to create regions where the data is read-only, non-executable. An MPU also allows a processor in privileged mode to define the memory accessibility when next running in non-privileged, user application, mode. Effectively, the processor manages user applications by ensuring that the memory region of each application is protected against access by another application.

It is the responsibility of the software team to create secure firmware that carries out initialization tasks. If either the SAU or the MPUs are available, one of these tasks is to partition memory into suitable regions.

PSA Trusted Base System Architecture for M (TBSA-M) recommends that, where possible, the system integrator implements a TrustZone-based system. To meet this recommendation, you must, where applicable, make sure that the IP you choose can utilize TrustZone technology. This topic is explored further in The processor and The Advanced Microcontroller Bus Architecture components.

Bus components that are based on AMBA 5 can identify TrustZone transactions and differentiate between Secure and Non-secure transactions. This feature allows the Security state of each transaction to be propagated from the CPU to the interconnect. The transactions then pass on, with their Security state, to other components in the system that require security awareness. A security flag (NS) on the bus identifies Secure TrustZone transactions.

TrustZone is optional in some IPs. Remember to enable TrustZone when configuring a piece of IP.

3.3 CryptoCell and CryptoIsland technologies

CryptoCell and CryptoIsland technologies complement TrustZone and offer the following:

- Asymmetric and symmetric cryptography

- True random number generation

- Device lifecycle state management

- A hardware-enforced Root of Trust policy

- A Root of Trust model allowing multiple owners

- Secure boot technology with software image validation and decryption available at boot time

- Validation of software source updates

- Secure debugging

- Keys and assets provisioning, management, and isolation in persistent trusted storage

CryptoCell technologies are engines that require a CPU and, sometimes, infrastructure on the SoC to perform the preceding functionality. CryptoIsland integrates a subsystem around a CryptoCell-312 and includes its own processor. This design means that a CryptoIsland can perform more of the preceding functionality on its own.

3.3.1 CryptoCell

There are two families of CryptoCell, the CryptoCell-300 family and the CryptoCell-700 family. The CryptoCell-700 family has a higher performance than the CryptoCell-300 family and is intended for content intensive applications, for example higher-end smartphones and set-top boxes.

The CryptoCell-312 is aimed at SoCs that are powered by either Cortex-M series or Cortex-R series processors. The CryptoCell-312 fits well in a design that is optimized for low-power usage and a low area.

3.3.2 CryptoIsland

CryptoIsland executes a full software stack inside itself, which allows you to isolate software from the host system. For example, if a SIM is kept inside the CryptoIsland, the SIM has as much protection as a detachable SIM card. In terms of functionality, the CryptoIsland includes a CryptoCell-312. CryptoIsland is also able to mitigate against physical attacks, which is explored further in Mitigating against physical attacks.

3.4 Mitigating against physical attacks

CryptoIsland provides sophisticated defense against different kinds of low-level attacks. These defenses include:

- Side-channel attack protection. These types of attacks include power and electromagnetic emission analysis by the attacker.

- Perturbation attack protection. These types of attacks include clock or voltage manipulation.

- Anti-tampering alarm response

- Hardware-based execution environment isolation

Whether you use a CryptoIsland to protect your system depends on how sensitive the data that is held on the system is. You must be realistic about how damaging an attack can be. For an IoT coffee maker, investing in the security that is provided by the CryptoIsland might be unnecessary. Arm provides the CryptoIsland for:

- A secure IoT device that requires mitigation against physical attacks

- A security enclave that provides software isolation for secure services

3.4.1 An example of a secure IoT device requiring high security

Imagine a device that controls entry to a luxury apartment building, with apartments that are rented out on Airbnb. The owner of the building does not live in the country and, as Airbnb bookings come in, must remotely control entry to the apartments.

Access to the main door is possible through the mobile of a guest or manually, through a temporary access code that is issued to a guest. The main door IoT device:

- Provides a touchscreen for user interaction. Guests can enter their access codes here.

- Maintains an online connection. Through this connection, it is possible to fully administrate the entrance rights that are assigned to the main door and each apartment.

- Tracks the check-in and the check-out of guests. If a guest does not show up or if a guest checks out, the device can alert the user.

- Tracks any unauthorized access to the doors in the building. The device can alert the owner immediately in response to this situation.

- Tracks any attempt to tamper with the SoC itself. Here, the threat could come from someone who has legitimate access to the building. Specifically, the anti-tampering alarm response of a CryptoIsland can mitigate against side-channel attacks on the SoC. These attacks intend to gain key information or affect SoC operation.

- Communicates with the individual doors of apartments. Microprocessors, containing for example a Cortex-M3, control the locking systems on these individual doors. However, the microprocessors are under the direction of the main device.

- Requires a security level to a standard that the CryptoIsland provides. If the main device is compromised, the result of unauthorized access could be costly.

3.5 Implementing additional security using software

Refer to Appendix A and Appendix B in the Threat Models and Security Analyses for three common IoT use cases. For each case, Appendix A details which of the required security functionality is covered by the CryptoIsland product. Appendix B details which of the required security functionality is covered by Arm TrustZone technology. Try to see where your own secure IoT device has the same requirements and identify what technology would cover it.

In cases where CryptoIsland does not cover a security functionality, Trusted Firmware-M might provide the functionality. For example, CryptoIsland does not offer hardware support for maintaining an audit trail of security events, but the Trusted Firmware-M (TF-M) provides this feature.

4 What pieces of IP do I need to make a secure IoT device?

In the preceding sections, we have explored the device requirements of our secure IoT device and how Arm technologies could solve them. In this section of the guide, we examine the pieces of IP that make up the system.

4.1 The main components of the system

The following figure shows the main components of the system:
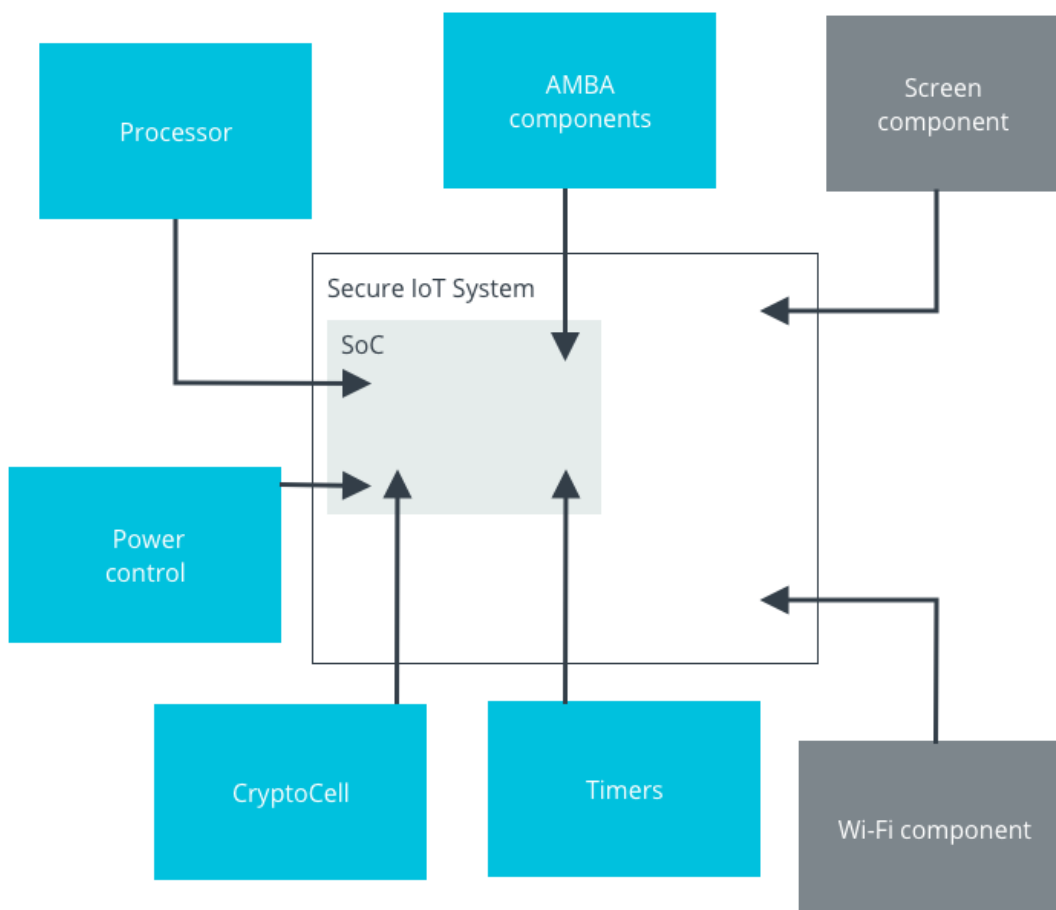
4.2 The processor

Arm offers Cortex-A, Cortex-R, and Cortex-M processors. The first thing to decide is which processor architecture to choose a processor from.

Cortex-A series processors are typically used for high-end devices, for example sophisticated smartphones. Cortex-A processors host rich Operating Systems and support multiple software applications.

Cortex-R series processors provide high performance in safety-critical environments and can meet real-time constraints. This feature means that these processors are typically used in the automotive industry and in storage devices. In these scenarios, responses to events must be guaranteed.

The Cortex-M series processors are the focus of the Trusted Base System Architecture. Microcontrollers are one of the primary markets for Cortex-M series processors. Microcontrollers are like SoCs but are less sophisticated. However, you can use the more powerful Cortex-M series processors in more demanding situations. In addition, Cortex-M series processors that implement the Armv8-M architecture include many features, for example TrustZone. As mentioned previously, TrustZone is essential to ensuring the device is secure. You can find a useful comparison of the Cortex-M series processors here.

The Cortex-M23 has enough features for a secure IoT coffee machine or similar device. Importantly, the Cortex-M23 has a TrustZone option. Depending on the requirements of a secure IoT device, you might need to use a newer, more expensive fab process to get extra performance. Getting the extra performance from a Cortex-M23 increases the area and static power usage of the SoC.

The Cortex-M33 is another candidate that you could consider if you need more compute performance than the Cortex-M23 offers. The Cortex-M33 has a DMIPS benchmark value of 1.5 compared with a value of 0.98 for the Cortex-M23. In addition to the extra performance that the Cortex-M33 offers, this processor also offers the following features:

- Floating-point unit (FPU)

- Digital Signal Processing (DSP)

- A coprocessor interface

If the software that is running on the device requires floating-point calculations, upgrading to the Cortex-M33 is a logical choice.

Because the Cortex-M23 and Cortex-M33 processors implement the Armv8-M architecture, including the TrustZone extension, the Trusted Base System Architecture applies to SoC designs that use either processor.

This guide assumes that you use a single Cortex-M23 processor for your SoC.
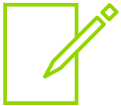
The Cortex-M23 processor and the Cortex-M33 processor are supplied with Arm Flexible Access.

4.2.1 Using a Cortex-A5 processor for a secure IoT SoC

The Cortex-A5 processor offers a possible solution for a secure IoT SoC. For example, if you need to run a version of the Linux operating system on your device, the Cortex-A5 is a good choice. The Cortex-A5 processor supports virtual memory. Through use of its L1 and L2 caches, a Cortex-A5 can achieve zero wait-state memory access for the most regularly used code and data.

However, it is worth comparing the PPA data of Cortex-M series processors with Cortex-A5 data. When comparing, select PPA data where a similar performance was achieved for both processors. A Cortex-A5 processor, when physically implemented, might have a higher static and dynamic power requirement, and have a larger area. However, if you need Cortex-A series capabilities and options, you might find that any increases in power usage and area are acceptable. The PPA Analysis Overview provides further information on comparing PPA data.

The Cortex-A5 processor is supplied with Arm Flexible Access.

4.3 The Advanced Microcontroller Bus Architecture components

The Advanced Microcontroller Bus Architecture (AMBA) is a specification that describes a range of bus protocols. Despite its name, the scope of AMBA is far beyond microcontroller devices.

AMBA bus protocols are used in bus interconnects in an SoC. These interconnects allow different IP blocks to talk to each other. For example, interconnects with protocols that are based on AMBA could enable a processor to communicate with a peripheral. Each version of the AMBA defines new interface signals and protocols. Legacy protocol support is also sometimes removed.

The CoreLink SIE-200 contains IP blocks that are compliant with the AMBA 5 Advanced High-performance Bus (AHB5) protocol. The focus of the SIE-200 is SoCs that are built around Cortex-M23 and Cortex-M33 processors. Importantly, the SIE-200 provides many different, highly configurable, AHB5-compliant IP blocks that allow you to build an interconnect that supports TrustZone.

This guide assumes that you will use the CoreLink SIE-200 to connect and secure the components in your SoC.

The CoreLink SIE-200 is part of the Corstone-201 Foundation IP that is supplied with Arm Flexible Access.

4.4 The CryptoCell

CryptoCell and CryptoIsland technologies explored the comprehensive security options that Arm provides. For a secure IoT coffee machine, the security that is provided by the CryptoCell-312 is adequate. Specifically, support from the CryptoCell-312 enables:

- The authentication of loaded firmware images before booting. This feature is based on a hardware root of trust.

- The authentication of firmware images before updating. This feature is based on a cryptographic signature with Public Key Infrastructure (PKI). Authentication failures are reported, and the firmware is rolled back to the last valid image.

- The authentication of remote servers. This feature is based on secure cryptographic and RNG support, which is used to support cryptographic protocols for communication.

- Integrity and confidentiality protection for exchanged assets. This feature is based on secure cryptographic and RNG support.

Software can handle the previous tasks. However, a CryptoCell increases the protection by hiding key assets and cryptographic processes from software.

In general, the CryptoCell-312 is a good match for an SoC based around a Cortex-M series processor. If physical attacks are a large concern for a secure IoT device, a CryptoIsland is the best choice.

This guide assumes that you will use the CryptoCell-312 to augment the security that is provided by TrustZone.

The CryptoCell-312 is supplied with Arm Flexible Access.

4.5 The power control

It is important to have a power and clock control infrastructure on an SoC. For this purpose, Arm created the PCK-600 Power Control Kit. The PCK-600 components use Arm Q-Channel and P-Channel Low-Power Interfaces to control the power and clock states of components on the SoC.

This guide assumes that you will use the PCK-600 for your SoC.
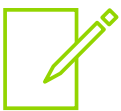
The PCK-600 is part of the Corstone-201 Foundation IP that is supplied with Arm Flexible Access.

4.6 The timers

It is important to have timers on an SoC. The Cortex-M System Design Kit (CMSDK) provides timers with varying functionality.

This guide assumes that you will use one or more of the CMSDK timers to fulfill the requirements of your SoC.
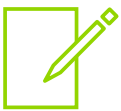
The CMSDK is part of the Corstone-201 Foundation IP that is supplied with Arm Flexible Access.

4.7 The Wi-Fi component

Arm does not currently offer any Wi-Fi IP. We recommend that you investigate IP that is offered by third parties. This guide assumes that you have Wi-Fi IP suitable for your IoT coffee machine and examines how you could integrate the Wi-Fi IP with the SoC.

4.8 The display

Arm does not currently offer any display IP targeting an AHB-based system. We recommend that you investigate IP that is offered by third parties. This guide assumes that you have display IP that is suitable for your IoT coffee machine and examines how you could integrate the display IP with the SoC.

5 How does the IP fit together?

This section of the guide focuses on how to fit together the pieces of IP that we discussed in What pieces of IP do I need to make a secure IoT device? The components that make up the CoreLink SIE-200 IP create the system interconnect. This interconnect allows signals and transactions to pass from the Cortex-M23 or Cortex-M33 processor to the peripherals in the system. These peripherals could be, for example, CryptoCell IP or Wi-Fi IP. This section uses theoretical knowledge from the Platform Security Architecture to take the first steps towards realizing that knowledge in an SoC.

Let's look at Arm IP components, and explore how to work with them when designing an SoC. Begin by downloading any of the following Arm IP that you currently have access to:

- Cortex-M23 with ETM or MTB

- CryptoCell-312 Security IP

- PL011 UART

- PL022 SPI Synchronous Serial Port

- Corstone-201 Foundation IP, which contains all the other Arm IP mentioned in this section

All the above IP that is listed in the preceding list is supplied with Arm Flexible Access.

Downloading the IP gives you access to the documentation and the RTL. Generally, IP is supplied with the following documentation:

Document	Description
Release note	Covers: • A list of the components, or products, that are delivered in the downloaded IP • The location of the components, including further component documentation, in the directory tree that is created for the downloaded IP • The installation procedure for the components • Any known issues with the downloaded IP
Technical Overview	For subsystems only. Describes the elements that make up a subsystem and the software that is available to use on it.
Configuration and Integration Manual	Covers: • Installation of a piece of IP that could be a set of components or a subsystem • Configuring the IP. For example, you can use Perl scripts, XML configuration files, and Verilog template files to generate custom Verilog RTL. • Performing Out of Box (OoB) testing • The guidelines for the functional integration of the IP into an SoC design • Implementing the IP
Technical Reference Manual	Includes low-level functional descriptions of IP components and a programmer's guide that provides register descriptions, memory maps, and interrupt maps.

Use this documentation to expand on the knowledge that is presented in the following sections of this guide.

The top-level directories in the Corstone-201 package are named according to product code. The README.txt file contains a key for the product codes to help you navigate the package.

5.1 Starting to develop an example TBSA-M system

Figure 5: Example TBSA-M system using TrustZone for Armv8-M from the PSA Trusted Base System Architecture for M (TBSA-M) document is an abstract depiction of a single core system that uses TrustZone for Armv8-M. The following diagram is a re-creation of Figure 5 from the document with the boxes filled out with real IP:
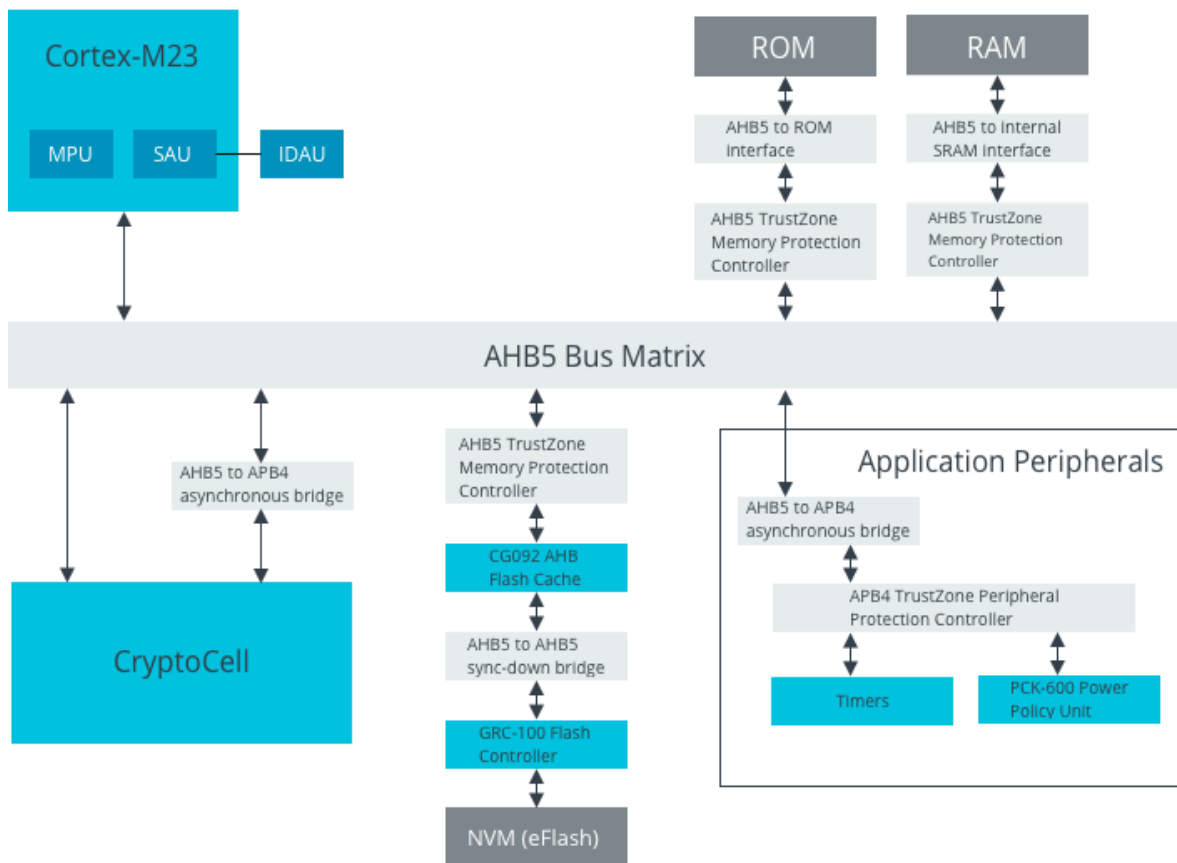
The timers and power control that are required for the SoC have been added. They are the only peripherals that the system currently has.

In What pieces of IP do I need to make a secure IoT device?, we decided to use a Cortex-M23 processor, AMBA components, and a CryptoCell-312 for our example design of a coffee maker. Now we will focus on the individual AMBA components, timers, and power components. The

following subsections explore the rationale behind the IP choices and their placement in the overall layout.

5.2 Select the individual AMBA components to link the IP together

Use most or all the following AMBA components from the CoreLink SIE-200 System IP for Embedded to link the other IP in your SoC together:

AMBA component	Description
AHB5 bus matrix	Connects other AHB5 components through its slave and master ports. This component can be thought of as the backbone of an SoC. You can configure this interconnect to add more interfaces based on your system needs. You can also add extra bus matrixes to these interfaces.
AHB5 Memory Protection Controller (MPC)	Monitors transactions to a memory interface. If a security violation occurs, this component gates the transaction. The MPC divides memory into blocks. A lookup table and registers determine whether the MPC can access an address. Security violations are reported through a dedicated interrupt.
APB4 Peripheral Protection Controller (PPC)	Monitors transactions to, and responses from, peripherals with APB4 interfaces. If a security violation occurs, this component gates the transaction. Security checking is defined for each APB peripheral that the controller hosts. External inputs configure the hosting. Supports up to 16 APB peripherals.
AHB5 to APB4 asynchronous bridge	Connects a low-bandwidth APB4 peripheral device to an AHB5 bus. This bridge creates a boundary between two clock or power domains and functions as an Access Control Gate.
AHB5 to internal SRAM interface	Enables on-chip synchronous RAM blocks to attach to an AHB5 interface. This component functions as a memory controller for the RAM.
AHB5 to ROM interface	Enables a simple ROM memory model to be attached to an AHB5 interface.
AHB5 to AHB5 sync-down bridge	Synchronizes AHB5 interfaces where the upstream side is faster than the downstream side.

AHB5 Access Control Gate (ACG)	Serves as a boundary between two clock or power domains. For example, the boundary could be between a base domain and a memory domain. If the receiving side of the transaction cannot accept the transfer or is explicitly asked not to, the transfer is blocked.
	For an SoC with multiple clock and power domains, an ACG is useful for handling clock and power domain crossings.
AHB5 Exclusive Access Monitor (EAM)	Monitors access to slaves that are downstream of it. The EAM is not a security component. Instead, this component implements AHB5 exclusive access support, which is required for handling semaphore passing. Semaphore passing is used to prevent multiple processors from accessing the same area of memory at the same time. An EAM supports TrustZone transactions, and you can place an EAM after an MPC.
	Unless your SoC has multiple processors, you do not need to use an EAM.

The CoreLink SIE-200 System IP for Embedded Release Note contains a complete list of all available components.

5.3 Connect to RAM

You must use a RAM controller for each bank of RAM in the system. The AHB5 to internal RAM interface provides this functionality. To enable TrustZone monitoring of the transactions to the RAM, an AHB5 MPC must be placed before the RAM controller. Connect the AHB5 MPC to the bus matrix. Then connect the AHB5 to RAM interface to the AHB5 MPC.

5.4 Connect to ROM

You must use a ROM controller for each bank of ROM. The AHB5 to ROM interface provides this functionality. To monitor the transactions to the ROM, an AHB5 MPC must be placed before the ROM controller. Connect the AHB5 MPC to the bus matrix. Then connect the AHB5 to ROM interface to the AHB5 MPC.

5.5 Connect to eFlash

For non-volatile memory, use eFlash. The GFC-100 Flash Controller is an ideal controller for our secure IoT system. To monitor the transactions to the eFlash, place an AHB5 MPC between the bus matrix and the controller. If you need AHB5 exclusive access support, you must also consider adding an EAM.

The Flash controller and flash memory can operate in a different clock domain to the main domain. In that case, you must use an AHB5 to AHB5 sync-down bridge to synchronize the two AHB5 interfaces.

We recommend that you add a cache between the bus interconnect and eFlash controller. This addition helps to reduce the impact of high eFlash access latency and low bandwidth from the CPU. You can use the CG092 AHB Flash Cache. Position the Flash Cache after the AHB5 MPC and before the AHB5 to AHB5 sync-down bridge.

Both the GFC-100 Flash Controller and CG092 AHB Flash Cache have an APB4 interface in addition to an AHB5 interface. Connections to both the GFC-100 Flash Controller and CG092 AHB Flash Cache APB4 interfaces must be after an APB4 PPC.

5.6 Connect the CryptoCell-312

The AHB5 bus matrix provides a slave and master interface for the CryptoCell-312 to connect to. The CryptoCell-312 connects as a slave to the AHB5 bus matrix through an AHB4 interface. It connects as a master to the AHB5 bus matrix through an AHB5 interface. An AHB5 to APB4 asynchronous bridge is needed between the AHB4 interface and the AHB5 bus matrix.

You must also implement One Time Programmable (OTP) memory and a persistent state storage block for the CryptoCell. The CryptoCell stores key state information in the persistent state storage block. If a cold reset occurs, the CryptoCell repopulates persistent values from the OTP memory. Implement the OTP memory using on die e-fuses or using private, protected eFlash locations.

5.7 Add power control to the SoC

The PCK-600 Power Control Kit provides several components for creating a clock and power infrastructure for an SoC. Use the components shown in the following table to provide the infrastructure for your SoC:

Component	Description
Low-Power Distributor Q-Channel (LPD-Q)	Distributes a Q-Channel from one Q-Channel controller to up to 32 Q-Channel devices. The LPD-Q allows the Q-Channel interfaces of the devices under control to be aggregated to a clock controller or a Power Policy Unit. This aggregation allows you to observe the current states and control the quiescent of the devices in relation to clock or power state.
Clock Controller (CLK-CTRL)	Provides high-level clock gating for devices in a clock domain that support Q-Channel Low-Power Interface (LPI) clock gating.
Power Policy Unit (PPU)	A configurable and programmable P-Channel or Q-Channel power domain controller. For controlling domain power modes in co-ordination with device quiescence, the PPU provides: • Technology-independent, low-power hardware interfaces • Software interfaces The PPU works alongside a Power Control State Machine (PCSM). The PCSM is a technology-dependent state machine for the sequencing of power switch chains and retention controls, which can include RAM and register retention. The PCSM executes power mode changes under PPU direction. The interface between the PPU and the PCSM is a P-Channel.

The Arm CoreLink PCK-600 Power Control Kit Release Note contains a complete list of all available components.

Connect the PPU through its APB interface to the APB4 PPC. Use the LPD-Q connect to other components whose power state the PPU must control.

5.8 Add the timers as peripherals

The CMSDK provides three timers that can be incorporated into an SoC as APB4 peripherals, as you can see in the following table:

Timer	Description
Timer	A 32-bit down-counter that generates an interrupt when the counter reaches 0

Dual-input timers	A module containing two programmable 32-bit or 16-bit down-counters that can generate interrupts when they reach 0. The timers can run in one of the following modes: free-running, periodic, and one-shot
Watchdog	A module containing 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 watch dog times out and result in the watchdog resetting the core. This mechanism provides a way to recover from software crashes.

The Arm Cortex-M0/M0+ System Design Kit Release Note contains a complete list of all available components, including other APB components.

Like all APB peripherals in a TrustZone-based system, an APB4 TrustZone Peripheral Protection Controller controls the timers.

You can use all the timers in your system. We strongly recommend that you include at least two watchdog timers in your system. Map one watchdog to the Secure world and map the other watchdog 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. We also recommend that you include a dual timer, or two more individual timers. Use one of the timers as a Secure timer and the other as a Non-secure timer.

The previous figures assume that the APB peripherals are in a separate clock or power domain. For this reason, the controller is connected to the AHB5 Bus Matrix through an AHB5 to APB4 Asynchronous Bridge. The bridge is not mandatory for TrustZone and is not part of the filter.

5.9 Add a theoretical Wi-Fi and display component

This section of the guide describes how to add a Wi-Fi or screen component to your system. In both cases, non-specific third-party IP is considered. Although the specifications of the IP are hypothetical, you can find comparable third-party products on the market. This information is designed to help you integrate any third-party IP into a secure IoT device system.

The previous sections of the guide dealt with combining the components in the SoC. Both the Wi-Fi and screen component are part of the system but not part of the SoC.

5.9.1 Wi-Fi component specifications

For a Wi-Fi component, we assume that you are using an off-chip Wi-Fi component. The component has a dedicated processor and its own memory, which contains the whole TCP/IP stack. These features mean that you do not require as much memory on your SoC. This guide assumes that the Wi-Fi component supports at least some of the following communication interfaces:

- UART

- SPI

- I^2C

How to connect these interfaces to the SoC is explained in Attach external components to an SoC.

The data throughput varies depending on the communication interface that is used. It is reasonable to expect 4Mbit/s data throughput over UART and 10Mbit/s data throughput over SPI.
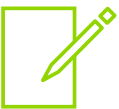
If the SoC will be produced at high volume levels, consider including the Wi-Fi module as part of the SoC. An on-chip Wi-Fi component can connect to a bus matrix through a compatible bus matrix interface.

5.9.2 Display component specifications

For a screen component, we assume that you are using an LCD touchscreen about 3 inches in size, with 320x480 pixels. The display connects to the SoC using either an I^2C or SPI interface. We also assume that the display has its own built-in controllers and display buffers.

5.9.3 Attach external components to an SoC

The previous sections of the guide identify communication interfaces that the third-party Wi-Fi or display component are likely to support. Other third-party components are also likely to support these interfaces. To connect these kinds of third-party components to your SoC, you

must provide the appropriate outward facing communication interfaces on your SoC. The following table describes how you can provide these interfaces:

Communication interface	Methodology
UART	Include an APB-compatible peripheral that provides an external UART interface. The CMSDK includes a basic UART. Alternatively, you could use the more advanced PL011 UART.
SPI	Include an APB-compatible peripheral that provides an external SPI interface. Arm supplies the PL022 SSP for this purpose.
I^{2}C	Include an APB-compatible peripheral that provides an external I^{2}C interface. Arm does not currently offer any I^{2}C connector IP. We recommend you investigate IP offered by third parties.

In the example SoC for this guide, connect APB-compatible peripherals to the APB4 PPC.

5.10 Integrate the IP into the SoC

We now have a list of IP components to put into the SoC. Next, we need to complete a series of steps before we test the system as a complete entity. This process is known as the integration phase of design and involves the following:

- Configuring the features that each piece of IP supports

- Configuring memory

- Setting up power and clock domains

- Connecting up SoC IP through master and slave interfaces

- Creating the IDAU for the Cortex-M23 processor

The following sections expand further on these integration steps:

5.10.1 Configure the features that each piece of IP supports

There are three ways to configure the IPs, as you can see in the following table:

IP configuration	Description
Render-time	Set the configuration options in a data file, for example an XML file. To then produce the RTL from the configuration, you run a script. This script also configures the testbench for the RTL and any related software. For example, a testbench for RAM might need to know the size that the memory was set to. Knowing the specified options enables the testbench to configure the tests accordingly.
Instantiation-time	When the RTL of a piece of IP has been rendered, the next step is to instantiate it. During instantiation of the RTL, you can configure each instance of the IP using Verilog parameters. For example, an SoC could have two processors. During instantiation, the FPU could be enabled for one processor and not the other by changing a Verilog parameter.
Using pin tiles	Pin tiles are used to configure features after rendering. However, if the pin tiles exclude certain options altogether, the RTL for those options are optimized out.

You can also configure IP at runtime using registers. However, physical implementations of the options that the registers offer must exist. Therefore, the RTL is always included for any optional features. In other words, a feature must be developed on the die ready to be switched on or off by the related registers.

5.10.2 Configure memory and replace generic Arm memory models

Configuring memory normally just involves setting the size of the memory. You must also replace the generic Arm memory model with a real memory library macro. Real memory libraries are fab process-specific. For example, you could use TSCM 28nm physical RAM. Although the size of the memory is defined within the IP configuration options, for the memory macro, you must still define:

- The size and number of memory banks

- Whether the memory is a square or rectangle

You must also consider these factors when arranging the floorplans of the die.

5.10.3 Set up power and clock domains

By defining a power or clock domain, you allow different domains to power down or stop their clock while other domains are still running.

A Power Policy Unit (PPU), which connects to the bus matrix through an APB4 interface, controls a power domain. The PPU provides a standardized software view for power control of a domain. The Power Control State Machine (PCSM) is responsible for switching components on or off. You must implement the PCSM for your system by writing the RTL for it.

A Clock Controller controls a clock domain.

Q-Channels or P-Channels allow:

- Components on an SoC to communicate their power state and whether they are idle. This information allows a PPU or a Clock Controller to decide whether to power down a domain or stop the clock for a domain.

- A PPU to make a request that a component change its power state. In response, a component can accept or refuse.

Q-Channels can only communicate two possible states. Clock domains only use a Q-Channel because the clock can only ever be on or off. P-Channels allow you to communicate multiple states to a component. A component can have more than two power states.

If you have one domain that is communicating through an AHB5 bus interface to another domain, you require an AHB5 Access Control Gate (ACG). The ACG stops a transaction entering a domain that is off and can also wake up a sleeping domain. Be aware that implementing power domains has overheads, in complexity and even performance. You must consider these overheads against the power-saving gains.

For example, imagine a smart temperature sensor that wakes up every ten seconds to take a reading. The sensor wakes up when used but spends much of its time asleep. The sensor also has a CryptoCell-312 for security. When the sensor wakes up, it wakes up everything including the CryptoCell-312. In some systems, the CryptoCell-312 is a candidate for putting in its own power domain. However, for the sensor, it does not make sense because the system is only awake for short periods of time.

The secure IoT coffee machine that we are designing might be simple enough to only include one clock and power domain. We might expect a coffee machine to be plugged into the wall plug and not to be running all the time. While the coffee machine is not running, the entire system can power down.

If you require multiple clock domains in your SoC, it is important to make sure that the timers are in the correct clock and power domain. If you do not want a timer to turn off, you must make sure that they are not in a gated domain.
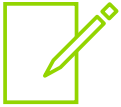
The AHB5 to APB4 asynchronous bridge can also function as an ACG. However, you can still use an AHB5 to APB4 asynchronous bridge within a single domain.

5.10.4 Connect up SoC IP through master and slave interfaces

When you configure an AHB5 bus matrix, you can define how many slave and master interfaces it has. Configure each interface to meet the requirements of the IP that is connecting to it. Be aware that the data width of the bus matrix interfaces is fixed. Therefore, if an interface of a piece of SoC IP has a different width, you must use a converter block. Sometimes, you might be able to configure the SoC IP interfaces so that they match the bus matrix interface width.

You must map each slave interface on a bus matrix to an area in the memory map. You must also decide which masters can access these interfaces.

5.10.5 Create the IDAU for the Cortex-M23

The IDAU must be implemented for each system. This process involves defining the IDAU regions in a look-up table for the Cortex-M23. Each entry in a table contains the following information about a region:

- Starting and ending address

- TrustZone Security state

- Region ID

5.11 Add your own registers for the SoC

During system integration, consider adding extra registers for control purposes or to provide system information. You can do this by adding a register bank that is accessible through the APB4 PPC in your system. These registers connect to the APB4 PPC along with some or all the peripherals that they control.

5.12 Arm subsystems

The process of integrating IP together and then validating the integration is complex. This complexity is reflected in the work hours that a successful integration requires. How can you save time in this area? The answer is in selecting the optimal Arm subsystems for your design. Exploring the SSE-123 Example Subsystem looks at an existing subsystem available in Arm Flexible Access. This subsystem fulfills the requirements for developing the secure IoT system we are exploring in this guide. The SSE-123 uses a Cortex-M23 processor, components from CoreLink SIE-200, and implements the core of a secure IoT system.

If you decide that the SSE-123 is a good fit for your SoC, there are time savings to be made. Alternatively, if you decide to build your own SoC from the start, we recommend exploring how the SSE-123 is put together at the RTL-level. This is because the SSE-123 provides a working example to kickstart your knowledge. The SSE-123 shows you how to configure and integrate the IP that we identified and discussed in the previous sections of this guide. Although it is a subsystem, it can provide a strong foundation for a secure IoT SoC.
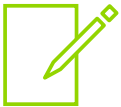
The SSE-123 is part of the Corstone-201 Foundation IP, which also contains the CoreLink SIE-200 IP that is used by the SSE-123.

6 Exploring the SSE-123 Example Subsystem

If you have already downloaded the Corstone-201 Foundation IP, you already have access to the SSE-123 Example Subsystem. The SSE-123 is separated into an inner SSE-123 Subsystem and an outer SSE-123 Integration. The integration expands on the subsystem and demonstrates the integration of more Arm IP.
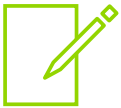
In this section of the guide, we use the term SSE-123 to refer to the entire subsystem.

The IP choices made for the SSE-123 are very similar to the recommendations that we made in How does the IP fit together? The recommendations and the SSE-123 are both based on theoretical knowledge from the PSA.

The SSE-123 does not contain a CryptoCell-312. Without this component, a system is not compliant with the PSA specification. To achieve compliance, you must add a CryptoCell-312 and then implement One Time Programmable (OTP) memory and a persistent state storage block for the CryptoCell-312. Adding a CryptoCell to the subsystem contains more information.

The SSE-123 subsystem shows how to:

- Use a PPC component

- Use an MPC component

- Implement an IDAU

- Perform a power integration

- Perform a debug integration

Let's review the features of the SSE-123.

6.1 SSE-123 features

The SSE-123 features:

- A Cortex-M23 processor, including Armv8-M Security Extensions and an Embedded Trace Macrocell (ETM) debug component

- An Implementation Defined Attribution Unit (IDAU) to define Secure and Non-secure regions of memory

- AMBA components from the CoreLink SIE-200 System IP for Embedded

- CoreLink PCK-600 Power Control Kit components

- System timers and watchdogs

- A single bank of SRAM

- eFlash memory

- System control registers and security control registers that are specific to the subsystem

- Interfaces:

    o   Serial Wire or JTAG

    o   AHB5 RAM Expansion

    o   AHB5 Peripheral Extension

    o   Power and Clock Q-Channel Device

    o   AHB5 Expansion Slave

6.2 Working with the SSE-123

The SSE-123 allows you to apply configuration options before rendering. These options are for individual pieces of IP, for example the Cortex-M23 processor, and also relate to the subsystem itself. For example, you can:

- Define the level of included debug functionality

- Include an interface for the Cortex-M23 single-cycle I/O port

- Include a Flash subsystem

The options are stored in `*.yaml`, `*.conf`, and *.xml files, and the IP is rendered using a Perl script.

The SSE-123 Subsystem also allows you to override some render configuration options using System Verilog parameters during instantiation of the module.
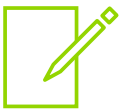
Chapter 2 of the Arm SSE-123 Example Subsystem Configuration and Integration Manual contains more detailed information on installing and configuring the SSE-123 product. Figure 2-1: Configuration options block diagram provides a useful visual guide to the components that are included or excluded depending on main configuration option settings.
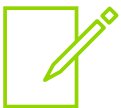
Chapter 4 of the Arm SSE-123 Example Subsystem Configuration and Integration Manual contains information on integrating the SSE-123 Subsystem into a larger system.

6.3 SSE-123 timers

The SSE-123 contains the following timers:

- Two system timers. You can map, by software, one timer for Secure mode and one timer for Non-secure mode.

- Two watchdog timers. You can map, by software, one watchdog for Secure mode and one watchdog for Non-secure mode.

- A system counter. This unit provides the timestamp that is used by all watchdog timer and system timers.

The timestamp-based timers are 64-bit and were developed for the SSE-123 and future IoT subsystems. The timers from the CMSDK, which were covered in Adding the timers as peripherals, are 32-bit and can also be added if necessary. The SSE-123 system and watchdog timers function in the same way as the watchdog that is described in Adding the timers as peripherals. The only exception is that these timers all need a shared reference timestamp input from which to derive their time. The system counter is part of the SSE-123 Integration and provides a 64-bit timestamp value that compatible components across the SoC can share.
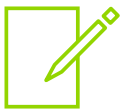
> More information on the SSE-123 counters is available in Appendix B of the Arm SSE Example Subsystem Technical Reference Manual.

6.4 SSE-123 registers

The SSE-123 allows run-time configuration through registers. If you use the SSE-123 as the basis of your own system, you could extend the registers. Use these registers to provide run-time control over any additional components that you add.

The SSE-123 Subsystem registers are defined in the SSE-123 Subsystem memory map. The register blocks that you can see in the following table are available:

Register block	Description
Subsystem information	Provides software with system information including the system version and how the system is configured.
Subsystem control	Provides registers for power, clocks, resets, and other general system control.
Secure privilege control	Allows software to control security gating units, which relate to the Secure state of the system. These registers are Secure privilege access only. For example, one of the registers permits software to configure Secure privilege access permissions for APB peripherals.
Non-secure privilege control	Allows software to control security gating units, which relate to the Non-secure state of the system. For example, one of the registers permits software to configure Non-secure privilege access permissions for external APB peripherals.
System timers	Allows software to control the security access control of the subsystem timers. Both timers can be given privileged or non-privileged access.

Watchdog timers	Allows software to control whether the refresh frame of each watchdog has privileged or non-privileged access.
Power Policy Unit	Allows software to control the Power Policy Control Unit.
SRAM memory protection control	Allows software to control the SRAM MPC registers.
Cortex-M23 processor Private Peripheral Bus (PPB)	Allows software to control the Cortex-M23 PPB.

6.5 Debugging with the SSE-123

The SSE-123 Integration provides a JTAG interface and a trace port. This debug integration consists of the following IP:

- M23 Debug Access Port (DAP)

- M23 Trace Port Interface Unit (TPIU)

- SoC-400 Cross Trigger Interface (CTI)

- SoC-400 Timestamp Generator

The former two pieces of IP come from the Cortex-M23 IP download. The latter two pieces of IP come from the CoreSight SoC-400 IP, which is a solution for the debug and trace of complex SoCs. The following table describes these pieces of IP:

IP	Description
Cortex-M23 DAP	Supports the JTAG interface and provides access to the core debug system
Cortex-M23 TPIU	Integrates with the Cortex-M23 ETM and supports instruction tracing
SoC-400 CTI	Allows events to be broadcast to other components. Events are signaled to the CTI through trigger inputs. Trigger outputs are used to signal events to other components. For example, in the SSE-123 Integration, the CTI triggers the system counter.

IP	Description
SoC-400 Timestamp Generator	Generates a debug timestamp value that provides a consistent view of time for multiple debug related IP blocks in an SoC. The Timestamp Generator is a necessary component when the system has enabled debugging with trace.

If you want to have similar capabilities in your SoC, you can render the SSE-123 with the debug option enabled. Then you can examine the rendered RTL and integrate the IP into your own system.

The SoC-400 IP is part of the Corstone-201 Foundation IP, which Arm Flexible Access supplies.

6.6 The SSE-123 I/O port

The Cortex-M23 IP includes an optional General Purpose Input/Output (GPIO) port. If you want to add an I/O port to your SoC, render the SSE-123 with the I/O option enabled. Examine the rendered RTL and then integrate the Cortex I/O port into your own system.

6.7 Ideas for extending the SSE-123

This section covers three ideas for extending and modifying the SSE-123:

- Replace the Cortex-M23 processor with a Cortex-M33 processor

- Add a CryptoCell-312 to the subsystem

- Add 32-bit timers to the subsystem

The SSE-200 Subsystem for Embedded is more advanced than the SSE-123. When making these extensions, we recommend using the SSE-200 as an example of how you can integrate the extra IP.

The SSE-200 contains the SSE-123 and is part of the Corstone-201 Foundation IP, which Arm Flexible Access supplies.

6.7.1 Replace the Cortex-M23 processor with a Cortex-M33 processor

The processor explores whether to use a Cortex-M23 processor or a Cortex-M33 processor in your SoC. The conclusion was that a Cortex-M23 processor would meet the performance requirements of your secure IoT device SoC. However, the possibility that more performance might be required was also accepted.

The SSE-123 uses a Cortex-M23 processor, which mirrors the choice of processor made previously. However, if you need more performance, or you need an FPU or DSP, you could upgrade the processor to a Cortex-M33. The SSE-200 contains two Cortex-M33 processors. If both processors are included, the usual setup is to have one processor running the operating system. The other processor acts as a coprocessor running occasionally and much faster.

If you choose to replace the Coretx-M23 processor in the SSE-123 with a Cortex-M33 processor, start by rendering the SSE-200 and looking at the RTL produced. When comparing the SSE-200 RTL with RTL rendered for the SSE-123, look at how the Cortex-M33 and Cortex-M23 processors integrate with the other components. These RTL examples provide insight into replacing a Cortex-M23 processor with a Cortex-M33.

Section A2.2 of the Arm CoreLink SSE-200 Subsystem for Embedded Configuration and Integration Manual describes the render options available for the SSE-200. The information includes the option for including or excluding either of the Cortex-M33 processors.

6.7.2 Add a CryptoCell to the subsystem

The **CryptoCell** section explores how using a CryptoCell-312 in your SoC complements Arm TrustZone and fortifies the security of the device. The SSE-123 does not include a CrytoCell-312. However, the SSE-200 provides an option to include a CryptoCell-312.

Figure 2-1: Top-level element interconnections of the Arm CoreLink SSE-200 Subsystem for Embedded Technical Overview shows the CryptoCell placed in its own domain. The figure also shows use of two AHB5 Access Control Gates to form the boundary between the CryptoCell domain and the main domain. The master interface on the AHB5 bus matrix also connects to the CryptoCell-312 through an AHB5 to APB4 asynchronous bridge. This bridge has a secondary function as an ACG.

If you want to add a CryptoCell-312 to the SSE-123, render the SSE-200 with the CryptoCell option enabled. The example RTL provides insight into adding the CryptoCell.

If the CryptoCell is positioned in the same clock and power domains of the main bus matrix, you do not require an ACG between the CryptoCell and the bus matrix.

6.7.3 Add 32-bit timers to the SSE-123 Subsystem

Figure 2-1: Top-level element interconnections of the Arm CoreLink SSE-200 Subsystem for Embedded Technical Overview shows a system control element. This element features an always-on timer and a secure watchdog that run on a slow 32KHz clock. Both components come from the CMSDK.

SSE-123 timers explores the 64-bit timers that are employed in the SSE-123. If you require CMSDK timers, you can add them to the system. You might add these timers in a situation where you require more timers that have different clock inputs. However, be aware that these timers do not share a timestamp with the system timers already in the system.

7 Related information

Here are some resources that are related to material in this guide:

Security on a secure IoT device :

- **Platform Security Architecture (PSA)**. A framework offering a step-by-step guide to building in the right level of security for securing connected devices.

- **Arm TrustZone**. A Security Extension that facilitates hardware-supported isolation.

- **CryptoCell and CryptoIsland**. Technologies offering various cryptography and platform related security services.

Platform Security Architecture:

- **Platform Security Architecture Overview Whitepaper**

- **PSA Trusted Base System Architecture for M (TBSA-M)**

- **Threat Model and Security Analysis for an asset tracker use case**

- **Threat Model and Security Analysis for a smart water meter use case**

- **Threat Model and Security Analysis for a network camera use case**

- **Security Model (PSA-SM)**

- **PSA Firmware Framework for M (PSA-FF-M)**

- **Trusted Boot and Firmware Update (PSA-TBFU)**

- **Trusted Firmware-M (TF-M)**

Specific Arm IP:

- **Arm CoreLink SSE-200 Subsystem for Embedded Technical Overview**

- The following documents are only available with an Arm Flexible Access license:

o   CoreLink SIE-200 System IP for Embedded Release Note

o   Arm CoreLink PCK-600 Power Control Kit Release Note

o   Arm Cortex-M0/M0+ System Design Kit Release Note

o   Arm SSE-123 Example Subsystem Configuration and Integration Manual

o   Arm SSE-123 Example Subsystem Technical Reference Manual

o   Arm CoreLink SSE-200 Subsystem for Embedded Configuration and
      Integration Manual

Other topics:

- **Real-time constraints**: R-series processors guarantee responses within specified time constraints.

- **Cortex-M series processor comparison**

- **DMIPS**: A common representation of the Dhrystone benchmark

- **Arm Q-Channel and P-Channel Low-Power Interfaces**

8 Next steps

This guide provides the essential knowledge that you must have to design and build an SoC for a secure IoT device. The IP discussed in this guide, including the subsystems, is all available through Arm Flexible Access. You can review the options for Arm Flexible Access here.

After you have familiarized yourself with the PSA, you can carry out a threat and security analysis for your use case. The analysis will help you decide on the IP you require to meet the security requirements of your SoC. You might also want to review the software and firmware requirements of your system at this point.

The next step is to explore the SSE-123 subsystem and decide whether you can use it as the basis for your SoC. You will find that the SSE-123 is designed using the same IP introduced in this guide. If the SSE-123 fits your vision, you will gain considerable time savings.