Authenticated debug for PSOC™ Edge

About this document

Scope and purpose

This application note shows how to implement the authenticated debug on the PSOC™ Edge MCU. It includes the procedures for disabling the debug access ports, configuring the authenticated debug, and accessing them using the debug certificate. Additionally, it introduces the debug policies, debug access ports, how boot code enforces the debug policy, and how the debug ports can be opened by the Boot firmware using the certificates or by a trusted firmware. This application note also provides the best practices for configuring the debug access ports for in-field use cases.

Intended audience

This application note is intended for users already familiar with PSOC™ Edge secured boot who wish to gain insight into securing the debug ports of PSOC™ Edge. For an introduction to secured boot concepts, see AN237849 - Getting started with PSOC™ Edge security.

Table of contents

1 PSOC™ Edge debug overview

PSOC™ Edge is a multi-core MCU with a user-programmable Arm® Cortex® M33 (CM33) and Cortex® M55 (CM55) CPUs. Each of these CPUs is associated with a dedicated debug access port: CM33_AP and CM55_AP. These debug access ports communicate with the debug components via the Advanced Peripheral Bus Interconnect (APBIC) infrastructure, which further connects to one of the AHB access port. These ports give access to the internal debug components of the respective CPUs and the rest of the system through the respective MIMO. The debug components of CM33 can be accessed only via CM33_AP; similarly, debug components of CM55 can be accessed only via CM55_AP. This distinction ensures efficient and targeted debugging capabilities for each CPU, enhancing the overall debug experience.

PSOC™ Edge also exposes a SYS_AP in addition to the CM33_AP and CM55_AP. SYS_AP provides access to other critical system resources such as SRAM, RRAM, OTP, and MIMO regions. This additional access enhances the versatility of the debugging environment, allowing developers to assess a broader range of resources within the PSOC™ Edge MCU in the absence of CM33_AP and CM55_AP.

The Serial Wire/JTAG signals are linked to the processor's debug system through several stages. The hardware module responsible for managing the SWD or JTAG interface protocol is known as the Debug Access Port (DAP/AP). The AP offers a generic AHB bus interface, employing the same bus protocol utilized by the processor to access the memory system. PSOC™ Edge employs a configurable debug access port from Arm® CoreSight® SoC-600.

***Note**: Debugging the Secured Enclave is not possible and the access ports are not configurable. This document does not discuss the Secured Enclave debug ports as they are not accessible.*

1 PSOC™ Edge debug overview

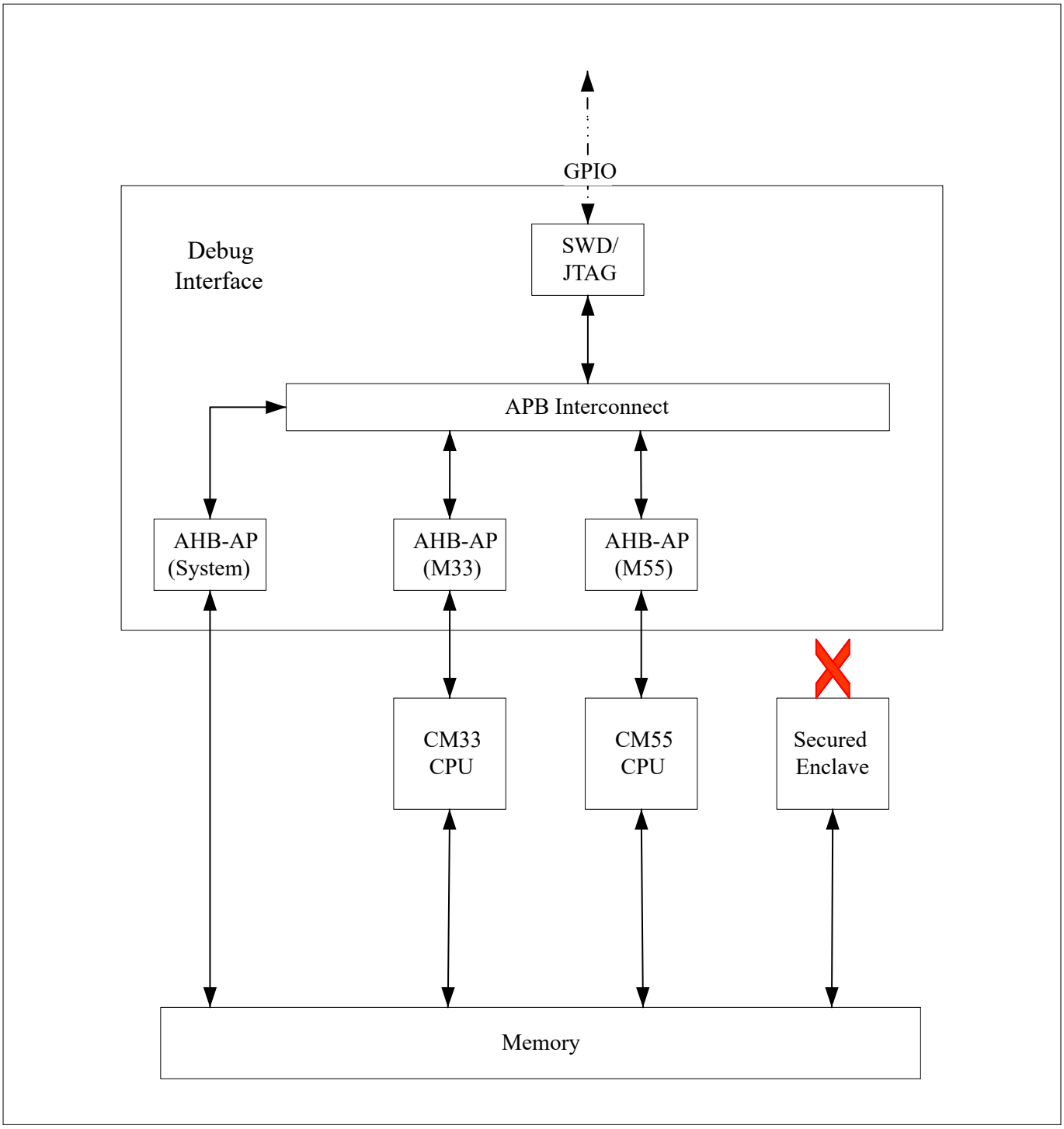

Figure 1 Debug architecture

2 Debug features

PSOC™ Edge provides the following debug features:

- Dedicated debug ports for CM33 and CM55 CPUs, along with an additional SYS_AP
- Ability to permanently activate/deactivate debugging
- Ability to enable debugging after authentication (debug authentication)
- Ability to enable debugging through a trusted firmware like TF-M
- Support for four debug states for CM33 debug access ports:
  - Non-invasive debug (niden)
  - Invasive debug (dbgen)
  - Secure privileged invasive debug (spiden)
  - Secure privileged non-invasive debug (spniden)
- Ability to disable/enable CM33 secure debug
- Support for two debug states for CM55 debug access ports:
  - Non-invasive debug (niden)
  - Invasive debug (dbgen)
- Resource protection using MPC and PPC when SYS_AP is accessible

Debugging involves various approaches and considerations, encompassing non-invasive, invasive, and secure debugging methodologies. CM33 CPU with TrustZone supports four debug control signals: niden, dbgen, spiden, and spniden. The CM55 CPU does not have TrustZone and supports only niden and dbgen. The debug access control signals do not apply to SYS_AP. Conversely, SYS_AP debug access control is managed by configuring the MPC and PPC for memories and MIMO registers, because SYS_AP is primarily intended for accessing the system resources. PSOC™ Edge provides mechanisms to enable or disable control over various memory regions such as RRAM, OTP, SRAM, and MIMO registers.

3 Debug authentication introduction

The PSOC™ Edge devices are shipped in the Development Lifecycle State (LCS), with a default debug policy that ensures that all the debug ports are open to allow seamless development and debugging out of the box. You can easily commence development on the device, and create, deploy, and debug applications with convenience.

Upon reset, the PSOC™ Edge boot starts from the Secured Enclave, which is the root of trust (RoT). Software running in the Secured Enclave is responsible for parsing the debug policies and applying them on every boot. The policy dictates the debug setups for CM33_AP and SYS_AP only. The behavior of CM55_AP depends on the CM33_AP policy. It is highly recommended to either fully close all the debug ports or activate the debug restrictions on these ports to uphold the highest security standards.

Within the Development LCS, the M33 debug policy can be explored, the debug restrictions can be enforced, and the debug ports can be enabled through the authenticated debug mechanism described in this document. To enforce a debug policy on CM33_AP, the first step is to obtain the device ownership via the Transfer of Ownership process as outlined in AN237849, "Getting started with PSOC™ Edge security". After the device ownership is transferred in Development LCS, the debug policy and debug parameters can be tailored as required and the device can be re-provisioned as many times as needed. Note that in the Development LCS, SYS_AP remains open regardless of the policy configurations, ensuring that the closed CM33 debug ports can be easily reopened via SYS_AP and the devices do not get accidentally locked.

It is recommended to ship the devices to the field in the Production LCS. In the Production LCS, the debug policy configuration and behaviors are identical to those of the Development LCS, with the distinction that the SYS_AP policies are implemented in this stage. This facilitates the option to secure both the CM33_AP and SYS_AP per application requirements.

For additional insights into the lifecycle state, see AN237849, "Getting started with PSOC™ Edge security", which provides comprehensive information on managing the device's lifecycle.

3.1 CM33_AP and SYS_AP access restrictions

Both CM33_AP and SYS_AP follow an identical model to enforce the access restrictions. The debug policy can be used to control the behavior and enforce the access restrictions on these debug access ports. This policy is a component of the OEM policy and is stored in the protected section of the RRAM. The default debug policy is provisioned by Infineon; however, it provides the flexibility to provision the new policy based on the specific requirements.

Each CPU has its own AP_CTL register where the debug features can be configured. Debug capabilities of CM33_AP and SYS_AP are controlled through SYSCPUSS_AP_CTL; those of CM55_AP are controlled through APPCPUSS_AP_CTL.

CM33_AP and SYS_AP feature three distinct states: Enabled, Disabled, and Allowed. The Allowed state is complemented by three additional configurations: Open, Firmware, and Certificate.

- **Enabled**: When the policy field is enabled, CPU debugging is enabled by the Boot ROM code running from the Secured Enclave to facilitate easy development. Applications or trusted firmware running on the device cannot modify the AP configuration. Having the debug ports open defeats the purpose of security. Note that both debug access ports (CM33_AP and SYS_AP) are enabled in the default policy
- **Disabled**: AP is permanently disabled during boot by the Boot ROM code running from the Secured Enclave. Applications or trusted firmware running on the device cannot enable debugging if it is permanently closed. Both CM33_AP and SYS_AP can be individually locked (permanently) using this value in the respective debug policy field
- **Allowed**: AP is disabled during boot by the Boot ROM code running from the Secured Enclave. CPU debugging can be enabled during run time by one of the following control options:
  - **Open**: The Boot ROM does not impose any debug restrictions. Consequently, any firmware, including non-secure user applications, can disable or enable debugging as necessary. Enabling both CM33_AP

and SYS_AP debug undermines all device security measures. If OEMs wish to use this option, it should only be considered after ensuring that all confidential data secrets are erased from the production silicon

- **Firmware**: The application running in secure mode of the CPU (such as TF-M or any secure projects) can directly access the control registers and make decisions on enabling or disabling debugging as required. The Boot ROM provides the necessary permission to secure the application to access the debug control registers and configure them as required. Trusted firmware can implement the necessary authentication and enable the debug access port for both CM33, M55, and SYS_AP. However, it is crucial that this option is used only when there is a well-implemented debug authorization in the trusted firmware. It can be implemented (DIY) by referring to the PSOC™ Edge registers manual
- **Certificate**: In this mode, a 'debug certificate' must be presented to the device. This debug certificate can be supplied either by the application or by SYS_AP (if it is accessible). To open an access-restricted debug port, place a debug certificate in an SRAM region accessible to the Secured Enclave, configure the BOOT_DLM_CTL register, and then reset the device. When the device comes out of reset, the Boot ROM detects the request through the BOOT_DLM_CTL register configuration, authenticates the certificate, and then applies the debug configurations requested in the certificate before initiating the next code from the CM33 CPU. Applications or trusted firmware running on the device cannot modify the AP configuration

As mentioned in the previous section, the M55_AP behavior depends on the M33_AP configuration. If the CM33_AP is set as "Open", the M55 debug port is also configured as Open by the Boot ROM code. If the CM33 debug port is configured to any state other than Open, that is "Disabled" or "Allowed", the M55 debug port remains closed and is not configured by the Boot ROM code. In such cases, the M55 debug port must be controlled by the secure application such as Trusted Firmware-M (TF-M). This behavior is consistent in both development and production LCS.

***Note**: SYS_AP can be used for loading a certificate into SRAM to open M33_AP. If SYS_AP is set to Allowed = Certificate or Allowed = Firmware, it remains initially closed. In such cases, the OEM application must implement a mechanism to enable SYS_AP before it can be used. It is the responsibility of the user firmware to open SYS_AP in these scenarios.*

3.1.1 Debug policy overview

The debug policy dictates the debug and test capabilities of CM33_AP and SYS_AP, allowing for customization of debug access port restrictions. The PSOC™ Edge debug policy is a simple JSON file, part of the provisioning policy, available in the `edgeprotecttools` initialization directory. The "debug": policy field indicates the start of the debug policy. This policy defines settings for debug access on two ports (CM33_AP and SYS_AP). It also includes a return merchandise authorization (RMA) policy and a placeholder to add debug keys (OEM_DEBUG_KEY Public) for the device, which are discussed in later sections.

3.1.1.1 CM33 debug policy (CM33_AP)

Table 1 CM33 debug policy (CM33_AP)

Policy Field	Description	Supported Values
cpu	CM33_AP debug restriction policy	**Disabled**: CM33_AP access is completely disabled during boot. All other fields of the CM33 debug policy are ignored. **Enabled**: CM33_AP is open for debugging. When CPU debugging is enabled, the rest of the fields in the policy are valid and applied by the Boot ROM on startup. **Allowed**: CM33_AP access is defined by the 'allowed' policy field. See the 'allowed' policy below.
allowed	Indicates who is allowed to access the AP. This option is applicable when the CPU policy is set to 'Allowed'.	**Open**: This indicates that CM33_AP is open for debug access. You can control the remaining debug control options such as dbgen, niden, spiden, spniden, and secure. **Firmware**: Trusted firmware can directly access CM33_AP control registers and determine whether to enable/disable debugging as required. This must be implemented by the user's secure application such as TF-M. **Certificate**: Firmware or SYS_AP must provide the authentication tokens to open the CM33_AP. Once the authentication tokens are received, the Secured Enclave kicks in on reset, applies the configurations per the debug token, and opens up the debug port.

(table continues...)

Table 1 (continued) CM33 debug policy (CM33_AP)

Policy Field	Description	Supported Values
dbgen	Enables invasive debug for the user application. If enabled, non-invasive debug is also enabled automatically. Invasive debugging includes actively modifying the system's state, often achieved by setting breakpoints, altering the computational state, and actively steering the flow of execution. This level of control enables developers to explore deeper into the system's operation, inspecting and manipulating critical variables and data structures, which is essential for diagnosing complex issues.	Supported values are 'true' for enabling and 'false' for disabling it. The ability to manage the dbgen field in the policy is determined by the values of the 'cpu' and 'allowed' fields in the policy, as follows: • 'cpu' is 'Disabled': The dbgen field in the policy is ignored • 'cpu' is 'Enabled': The dbgen field in the policy is applied by the Boot ROM as defined in the policy • 'cpu' is 'Allowed': Invasive debugging can still be activated by the user application (allowed = open) or trusted firmware (allowed = Firmware) through direct writes to the CPUSS_AP_CTL register. If authenticated debug is enabled (allowed = Certificate), the debug certificate can regulate this setting, where the Boot ROM controls this capability by applying the value defined in the debug certificate to the CPUSS_AP_CTL register
niden	Enables non-invasive debug for the user application. Non-invasive debugging involves passive monitoring such as generating a trace, providing developers with insights into system behavior without directly altering its state, etc.	Supported values are 'true' for enabling and 'false' for disabling. The ability to manage the niden field in the policy is determined by the values of the 'cpu' and 'allowed' fields in the policy, as described above under 'dbgen'.
spiden	Enables secure privileged invasive debug. If enabled, secure privileged non-invasive debug is also enabled automatically. This enables the debugging capability within the secure, privileged domain of the CPU, ensuring that critical operations can be monitored and modified.	Supported values are 'true' for enabling and 'false' for disabling. The ability to manage the spiden field in the policy is determined by the values of the 'cpu' and 'allowed' fields in the policy, as described above under 'dbgen'.
spniden	Enables secure privileged non-invasive debug. This enables non-intrusive debugging operations within the secure privileged domain of the processor, providing developers with the means to observe and analyze system behavior within the secure domain without disrupting its normal operation.	Supported values are 'true' for enabling and 'false' for disabling. The ability to manage the spiden field in the policy is determined by the values of the 'cpu' and 'allowed' fields in the policy, as described above under 'dbgen'.

(table continues...)

Table 1 (continued) CM33 debug policy (CM33_AP)

Policy Field	Description	Supported Values
secure	Enables secure debug of the CM33 CPU. Secure privileged invasive (spiden) and secure privileged non-invasive (spniden) debug policies are applicable only if secure policy is set to 'true'.	Supported values are 'true' for enabling and 'false' for disabling. The ability to manage the spiden field in the policy is determined by the values of the 'cpu' and 'allowed' fields in the policy, as described above under 'dbgen'.

The above table provides an overview of all the supported options in the CM33 debug policy. However, all the combinations of these configurations are not supported. See some of the following example scenarios for the usage of dbgen, niden, spiden, spniden, and secure.

- **Invasive debugging of CM33 non-secure**: For Invasive debugging of CM33 non-secure, set dbgen = true. You can leave the debugging of CM33 secure turned off by setting spiden = false and spniden = false to enable CM33 non-secure debugging. CM33 secure debugging is not enabled in this case
- **Invasive debugging of CM33 secure**: For Invasive debugging of CM33 secure, set secure = true, spiden = true, and debgen = true. Note that it is required to enable non-secure debugging to enable secure debugging

3.1.1.2 System debug policy (SYS_AP)

Table 2 System debug policy (SYS_AP)

Policy	Description	Supported Values
ap	SYS_AP debug restriction policy	**Disabled**: SYS_AP access is completely disabled during boot. All other fields of the SYS_AP debug policy are ignored. **Enabled**: SYS_AP is open for debugging. All other fields of the SYS_AP debug policy are ignored. **Allowed**: SYS_AP access is defined by the 'allowed' policy field. See 'allowed' policy below.
allowed	Indicates who is allowed to access the AP. This option is applicable when the AP policy is set to 'Allowed'	**Open**: This indicates that SYS_AP is open for debug access. **Firmware**: Trusted firmware can directly access SYS_AP control registers and determine whether to enable/disable debugging as required. This must be implemented by the user's secure application such as TF-M. **Certificate**: Firmware or CM33_AP must provide the authentication tokens to open the AP. Once the authentication tokens are received, the Secured Enclave kicks in on reset and opens up the debug port.

(table continues...)

Table 2 (continued) System debug policy (SYS_AP)

Policy	Description	Supported Values
mpc_ppc_enable	Indicates that the MPC/PPC restrictions on the system access port are programmed and locked according to the settings in the next four fields. If this policy is set to 'false', the remaining four policy fields in this table are ignored.	Valid only if 'ap' is set to 'Allowed' or 'Enabled'. Set to 'true' to enable and set to 'false' to disable.
rram_otp_enable	Indicates whether access is allowed to general-purpose OTP.	Valid only if 'ap' is set to 'Allowed' or 'Enabled' and 'mpc_ppc_enable' as true. Set to 'Entire Region' to enable access to the complete user-accessible general OTP region. Set to 'Nothing' to disable access to the general OTP. For details of the RRAM OTP region, see the PSOC™ Edge technical reference manual.
rram_nvm_enable	Indicates the portion of the user-programmable NVM accessible through SYS_AP. When a fractional value is selected, only a portion of the NVM starting at the bottom is exposed.	Valid only if 'ap' is set to 'Allowed' or 'Enabled' and 'mpc_ppc_enable' as true. Set to 'Entire Region' to enable access to the complete user-accessible RRAM region. Set to 'Nothing' to disable access to the RRAM. Set this to 7/8th, 3/4th, 1/2, 1/4th, 1/8th, or 1/16th to define the portion of the accessible memory area from the bottom of the RRAM.
ram_enable	Indicates the portion of the SRAM accessible through SYS_AP. When a fractional value is selected, only a portion of the SRAM starting at the bottom is exposed.	Valid only if 'ap' is set to 'Allowed' or 'Enabled'' and 'mpc_ppc_enable' as true. Set to 'Entire Region' to enable access to the full user-accessible SRAM region. Set to 'Nothing' to disable access to SRAM Set this to 7/8th, 3/4th, 1/2, 1/4th, 1/8th, 1/16th. These values defines the portion of the accessible memory area from the bottom of the SRAM.

(table continues...)

Table 2 (continued) System debug policy (SYS_AP)

Policy	Description	Supported Values
mmio_enable	Indicates the portion of the MMIO region accessible through the system access port.	Valid only if 'ap' is set to 'Allowed' or 'Enabled' and 'mpc_ppc_enable' as true. Set to 'All MMIO registers' to allow full access to MIMO register via SYS_AP. Set to 'Only IPC MMIO' to allow only SRSS.BOOT_STATUS MMIO register access. Set to 'RAM app only' to allow only RAM application launch request register access. Set to 'No MMIO access' to completely disable access to MIMO registers.

***Note**: CM33_AP and SYS_AP share the CPUSS_AP_CTL debug configuration register. If either AP is set to a mode that protects CPUSS_AP_CTL (requires Boot ROM only access), both APs must be configured for Boot ROM only access to avoid a security conflict. Do not configure one AP for Boot ROM only access while the other allows firmware access. Examples:*

- *Allowed: CM33_AP Allowed = Certificate; SYS_AP Allowed = Certificate or Enabled/Disabled*
- *Not allowed: CM33_AP Allowed = Certificate; SYS_AP Allowed = Firmware or Allowed = Open*

3.1.1.3 Debug keys

You can specify the debug key in the debug policy. This ECC P-256 key must be provisioned to the device for authenticated debug to function. Within this policy, you can set your own custom debug keys by indicating the path to the public key. A sample from the default policy is shown below:

```
"debug_key": {
  "description": "ECDSA public key, used to verify debug token. Use the path to the ECDSA P-256
public key or a 65-byte hex string",
  "value": ""
}
```

Note that if the default debug policy does not define the debug keys, Edge Protect Tools will use a zeroed key. In other words, if you provision the device without specifying a debug key, a zeroed key will be programmed to the device, if Authenticated debug is not enabled. However, once authenticated debug is enabled, it is essential to specify the debug key; failure to do so will result in an error being thrown by Edge Protect Tools.

3.2 CM55_AP access restrictions

CM55 resides in the high-performance domain (PD1) and Secured Enclave; CM33 CPU is in low-power domain (PD0). The access restriction of CM55_AP cannot be configured without turning on the high-performance domain (PD1). Configurations are not retained when the high-performance domain (PD1) is power cycled. In such scenarios, it is the responsibility of the secure application such as TF-M or any application in secure mode of the CPU to configure CM55 access ports. the CM55_AP should be configured by yourself by referring to the PSOC™ Edge registers manual. It is recommended to close CM55_AP for production devices.

3.3 Debug token and debug certificate

After implementing the debug port security measures, access is restricted. Depending on the policy configurations, either a trusted firmware or the Boot ROM can open the debug ports based on the request.

- **Secure firmware opening the debug ports**: If the debug policy allows the secure firmware to open the debug ports (Allowed = Firmware), the Boot ROM keeps the debug access ports disabled, configures the necessary permissions for the secure firmware such as TF-M to access the debug control registers, and boots to the next application. This implementation is completely user-defined and out of the scope of this document. Note that a non-secure user application cannot access the debug port registers in this case
- **Boot ROM opening the debug ports**: If the debug policy requires a certificate (Allowed = Certificate), the Boot ROM will apply the necessary restriction on the debug port registers such that it cannot be modified by user software. If debug port configuration is set as Allowed = Open, debug port is enabled by Boot ROM and debug port configuration can be modified by any user software. If debug policy requires a certificate, the Boot ROM process the debug certificates, and apply them on debug port to open the debug access as requested. Implementation of how to provide debug token is left to the user application. The document discusses both methods where SYS_AP and the user application provide debug certificates

Note that in the certificate method (Allowed = Certificate), it is the Boot ROM that applies the debug policy and processes the debug certificates. On the other hand, in the firmware method (Allowed = Firmware), the secure firmware (like TF-M) manages debug control. To avoid confusion, it is essential to understand that in the Certificate method (Allowed = Certificate), the debug certificate can be provided by either the application software or SYS_AP. Irrespective of whether the certificate is provided by SYS_AP or application software, Boot ROM validates and applies it.

3 Debug authentication introduction

The debug token is a simple JSON file with the collection of parameters to open the debug port. See the default `debug_token.json` file provided by Edge Protect Tools in packets folder where Edge Protect Tools was initialized. Table 3 lists the field and mapping to the debug certificate.

```
{
  "version": "1.0.0.0",
  "device_id": {
    "family_id": "0x0115",
    "si_revision_id": "0x11"
  },
  "silicon_id": "0xED84",
  "control_word": {
    "system": {
      "ap": "Enabled"
    },
    "cm33_0": {
      "ap": "Enabled",
      "dbgen": true,
      "niden": true,
      "spiden": true,
      "spniden": true,
      "secure": true
    }
  },
  "die_id": {
    "min": {
      "lot": 0,
      "wafer": 0,
      "xpos": 0,
      "ypos": 0,
      "sort": 0,
      "day": 0,
      "month": 0,
      "year": 0
    },
    "max": {
      "lot": 16777215,
      "wafer": 255,
      "xpos": 255,
      "ypos": 255,
      "sort": 255,
      "day": 255,
      "month": 255,
      "year": 255
    }
  }
}
```

In Authenticated debug using certificates, the Boot ROM applies the debug policy per the debug token. If the debug token and debug policy contain a common field, the value defined in the debug token takes precedence; the Boot ROM directly applies the value from the debug certificates.

The debug certificate is created using the `debug_token.json` file, signed using the debug private key. Edge Protect Tools provides a sample debug token and infrastructure to generate debug certificates from this token. Note that the debug token and debug certificate both follow a custom format described below.

3.3.1 Debug certificate format

Table 3 Debug certificate format

Offset	Size (in bytes)	Debug token field	Description
0x00	4	Version	Version of the debug certificate format. Example from `debug_token.json`: `"version": "1.0.0.0"`
0x04	3	device_id	*device_id* consists of the *family_id* (2 bytes) + *si_revision_id* (1 byte). If the Device ID in the token does not match the Device ID in the NVM, the token will be rejected. Example from `debug_token.json`: `"device_id": { "family_id": "0x0115", "si_revision_id": "0x11" },`
0x07	2	silicon_id	Provides the silicon ID of the device. If *silicon_id* in the token does not match the *silicon_id* in the NVM, the token will be rejected. Example from *debug_token.json*: `"silicon_id": "0xED84",`
0x09	3	Reserved	Reserved

(table continues...)

Table 3 (continued) Debug certificate format

Offset	Size (in bytes)	Debug token field	Description
0x0C	4	SYS_AP and CM33_AP restrictions	Provides an option to enable or disable the AP upon successful authentication of the debug token. It contains an explicit policy to control both SYS_AP and CM33_AP as shown below. SYS_AP can be either enabled or disabled using system AP policy. CM33 can be either enabled or disabled using the CM33_AP policy. Additionally, you can control all the debug access control signals discussed in Section 2 of this document. `"control_word": { "system": { "ap": "Enabled" }, "cm33_0": { "ap": "Enabled", "dbgen": true, "niden": true, "spiden": true, "spniden": true, "secure": true } },` Options outlined in this debug token serve as a request, while the prevailing debug policy which is provisioned to the device holds precedence over these specified options. For example, if the device has been configured with a policy that disables CM33_AP, it cannot be opened even if the debug token specifies it as open. The configuration for CM33_AP in this token will be considered valid only when the debug policy designates CM33_AP as 'Allowed'. This principle also applies to other fields in the token such as dbgen, niden, spiden, spniden, secure, and system AP.
0x11	4	Reserved	Reserved

(table continues...)

Table 3 (continued) Debug certificate format

Offset	Size (in bytes)	Debug token field	Description
0x15	20	die_id_min and die_id_max	*die_id* is 10-byte unique identifier of a device, which can be read using Edge Protect Tools. See the details on how to read the *die_id* in the "Getting started with secured debug" section. *die_id min* corresponds to the minimum value of die to constrain the token to a range of devices. *die_id max* corresponds to the maximum value of the die to constrain the token to a range of devices. The *die_id* field consists of the lot (3 bytes), wafer, xpos, ypos, sort, day, month, and year (1 byte each) placed in an array in the specified order. `"die_id": { "min": { "lot": 0, "wafer": 0, "xpos": 0, "ypos": 0, "sort": 0, "day": 0, "month": 0, "year": 0 }, "max": { "lot": 16777215, "wafer": 255, "xpos": 255, "ypos": 255, "sort": 255, "day": 255, "month": 255, "year": 255 } }` To limit the scope of this debug token to a specific PSOC™ Edge device or a range of PSOC™ Edge devices, you can adjust the *die_id min* and *die_id max* values. For example, by setting *die_id min* to 0x1A00010000000 and *die_id max* to 0xFF00000000000, the debug token will only be recognized by devices with *die_id* falling within the range of 0x1A00010000000 to 0xFF00000000000. Other devices will reject this debug token even if they are able to successfully validate the signature. Note that out of the 20 bytes, the first 10 bytes are reserved for *die_id min* and the next 10 bytes are for *die_id max*.

(table continues...)

Table 3 (continued) Debug certificate format

Offset	Size (in bytes)	Debug token field	Description
0x2A	64	Signature	ECC256 Signature. The debug token must be signed using the debug private key for a proper creation of the debug certificate.

4 Getting started with authenticated debug

This section describes the practical applications of AP, debug authentication, debug policy, and certificates to secure the debug ports and enable their access through authentication using debug certificates. This section will provide a step-by-step guide for this process.

Before starting this section, make sure that you have read AN237849, "Getting started with PSOC™ Edge security". To get started with debug authentication, ensure the following:

1. Get ownership of the device through the provisioning process described in Section 2.2.2 of AN237849, "Getting started with PSOC™ Edge security"
2. Create the "Hello World" code example (mtb-example-psoc-edge-hello-world), build, and program the device in ModusToolbox™
3. Generate the debug keys and incorporate them into the policy file
4. Tailor the debug policy to restrict access to the debug ports
5. Provision the device with the updated policy containing the debug key (OEM_DEBUG_KEY)
6. As a result of these actions, the debug ports on the device will be secured based on your policy configurations. You will notice that the debug ports are locked and inaccessible. Verify this by attempting to program/debug the application through ModusToolbox™. If the program/debug attempt fails, it indicates that debug ports are protected
7. To enable access to the debug ports for further programming and debugging, configure the debug token (`debug_token.json` file) and create signed debug certificates using Edge Protect Tools
8. Incorporate the generated debug token into the ModusToolbox™ launch configuration
9. Click on the "Program" option in your project within ModusToolbox™. This will open the debug access port, allowing you to program and debug your application

After completing these steps, you have secured the debug ports and opened the debug port using authentication. Detailed step-by-step instructions are provided in the subsequent sections.

4.1 Setting up the environment

This document assumes that you are already familiar with AN237849 "Getting started with PSOC™ Edge security". If you have not followed the "Getting started" instructions to transfer ownership and provision the device, start with Section 2.2.2, "Getting Started". If not done already, do the following:

1. Install ModusToolbox™ and make yourself familiar with the ModusToolbox™ ecosystem. See AN235935, "Getting started with PSOC™ Edge E84 on ModusToolbox™ software"
2. Initialize the application directory using Edge Protect Tools. Note down the path where your Edge Protect Tools is initialized. This document refers to this directory as *<EdgeProtectTools init dir>*
3. Configure the OpenOCD path for Edge Protect Tools
4. Generate the OEM keys
5. Generate the OEM certificate
6. Provision the device to obtain device ownership
7. Create the "Hello World" code example (mtb-example-psoc-edge-hello-world) in ModusToolbox™
8. Build and program the device by following the instructions in the README of this code example

4.2 Generating the debug keys

You need to create debug key pair (private and public key). The following command generates the two keys that will be placed in the *keys* directory. PSOC™ Edge supports 256-bit ECC keys, and the output is in the `.pem` format.

This is a common ASCII format compatible with many public domain tools. Execute the following command from <EdgeProtectTools init dir> directory.

```
edgeprotecttools create-key --key-type ECDSA-P256 --output ./keys/debug_priv_key.pem ./keys/
debug_pub_key.pem
```

4.3 Configuring the policy

After generating the debug keys, you should integrate the public key into the debug policy available in the `<EdgeProtectTools init dir>/policy/` directory.

1. Open the policy file named `policy_oem_provisioning.json`
2. Search for the "debug" policy and locate `debug_key` within this debug policy
3. Update the value field of the policy with the path to the debug key (public key)

```
  "debug_key": {
      "description": "ECDSA public key, used to verify debug token. Use the path to the ECDSA
P-256 public key or a 65-byte hex string",
      "value": "<edgeprotecttools init dir>/keys/debug_pub_key.pem"
    }
```

The default configuration sets both CM33_AP and SYS_AP as open. To secure your debug ports, you must configure the debug policy accordingly. Modify your debug policy for CM33_AP and SYS_AP as shown below to enable authenticated debug. Note that the following configurations have CM33_AP as 'Allowed', which indicates CM33 debugging is allowed. The next field 'allowed' is set to certificate, which indicates that a valid

debug certificate must be presented to the device to open the debug ports. Debug access control for other options such as dbgen and spiden must be defined in the debug token in this case.

```
  "debug": {
    "cm33": {
      "cpu": {
        "description": "CPU access port debug restriction",
        "applicable_conf": "Disabled, Enabled, Allowed",
        "value": "Allowed"
      },
      "allowed": {
        "description": "Indicates who is allowed to access the AP. The option is applicable
when CPU control is 'Allowed'",
        "applicable_conf": "Firmware, Certificate, Open",
        "value": "Certificate"
      },
      "dbgen": {
        "description": "Invasive Debug. It implies non-invasive is also enabled",
        "value": false
      },
      "niden": {
        "description": "Non-Invasive Debug (monitoring without control)",
        "value": false
       },
      "spiden": {
        "description": "Secure Privileged Invasive Debug. The control field is not applicable
in NORMAL_NO_SECURE LCS",
        "value": false
       },
      "spniden": {
        "description": "Secure Privileged Non-Invasive Debug. The control field is not
applicable in NORMAL_NO_SECURE LCS",
        "value": false
       },
      "secure": {
        "description": "Enables Secure debug of CM33 CPU. Secure invasive and non-invasive
debug policies are applicable only if this option is enabled"
        "value": true
       }
},
```

SYS_AP configuration is applied by Boot ROM only in Production LCS. SYS_AP is always in Enabled state in Development LCS. The following policy shows how to set SYS_AP to 'Allowed' (in Production LCS) indicating that the debug port is accessible with additional restrictions imposed by the policy. The subsequent policy sets 'allowed' to 'Certificate,' signifying that the Boot ROM will configure SYS_AP access restrictions in accordance with the debug policy, and user code will not have permission to modify them. Additionally, the 'mpc_ppc_enable' policy is enabled to ensure that the next four fields in the policy are effective. As the following JSON snippet shows, the SRAM is opened for access through SYS_AP, allowing SYS_AP to place data/

tokens in the SRAM. MIMO registers are opened for the RAM app so that the RAM app can access these registers to communicate with/from the Boot ROM.

```
"system": {
  "ap": {
    "description": "System access port (SYS-AP) restriction",
    "applicable_conf": "Disabled, Enabled, Allowed",
    "value": "Allowed"
  },
  "allowed": {
    "description": "Indicates who is allowed to access the AP. The option is applicable when AP
control is 'Allowed'",
    "applicable_conf": "Firmware, Certificate, Open",
    "value": "Certificate"
  },
  "mpc_ppc_enable": {
    "description": "Indicates that the MPC/PPC on the system access port must be programmed and
locked according to the settings in the next 4 fields.",
    "value": true
  },
  "rram_otp_enable": {
    "description": "This field indicates if access is allowed to general purpose OTP",
    "applicable_conf": "Entire region, Nothing",
    "value": "Nothing"
  },
  "rram_nvm_enable": {
    "description": "This field indicates what portion of RRAM MAIN_NVM (including reclaimed
area of PROTECTED_NVM but excluding MAIN_NVM Region 0 and Region 1)",
    "applicable_conf": "Entire region, 7/8th, 3/4th, 1/2, 1/4th, 1/8th, 1/16th, Nothing",
    "value": "Nothing"
  },
  "ram_enable": {
    "description": "This field indicates what portion of SRAM is accessible through the system
access port. Only a portion of SRAM starting at the bottom of the area is exposed, like the
'rram_nvm' option",
    "applicable_conf": "Entire region, 7/8th, 3/4th, 1/2, 1/4th, 1/8th, 1/16th, Nothing",
    "value": "Entire region"
  },
  "mmio_enable": {
    "description": "This field indicates what portion of the MMIO region is accessible through
the system access port",
    "applicable_conf": "All MMIO registers, Only IPC MMIO, RAM app only, No MMIO access",
    "value": "RAM app only"
  }
},
```

4.4 Provisioning the new debug policy to the device

When you have the policy ready, the next step is to provision the device with the new policy. To provision the device, you need the generated `debug_pub_key.pem` and the OEM certificate `ifx_oem_cert.bin` generated as described in AN237849, "Getting started with PSOC™ Edge security".

```
edgeprotecttools -t pse8xs2 provision-device -p policy/policy_oem_provisioning.json --key keys/
oem_private_key_0.pem --ifx-oem-cert packets/apps/prov_oem/oem_cert.bin
```

***Note**: Two targets are supported for PSOC™ Edge by Edge Protect Tools:*

- *pse8xs2*
- *pse8xs4*

This document uses pse8xs2 target for demonstration. User should select the target that matches the device being used for their application.

4.5 Accessing the debug port without authentication

After provisioning the device with new policy, the debug ports are locked down; you cannot access them normally. Do the following to confirm that debug ports are locked:

1. Launch ModusToolbox™ and go to the designated project (Hello World app). You should have already set up this project as outlined in the Setting up the environment section
   a. Select the PSOC_Edge_Hello_World project from Project Explorer
   b. Select the <Application Name> Program(KitProg3_MiniProg4) configuration in the Quick Panel or from the debug button to program the device as shown in the figure. For details, see the "Program and debug" section in the Eclipse IDE for ModusToolbox™ user guide
2. Observe the programming logs on ModusToolbox™ console. You will see an error which indicates that debug access ports are closed and you cannot program the device

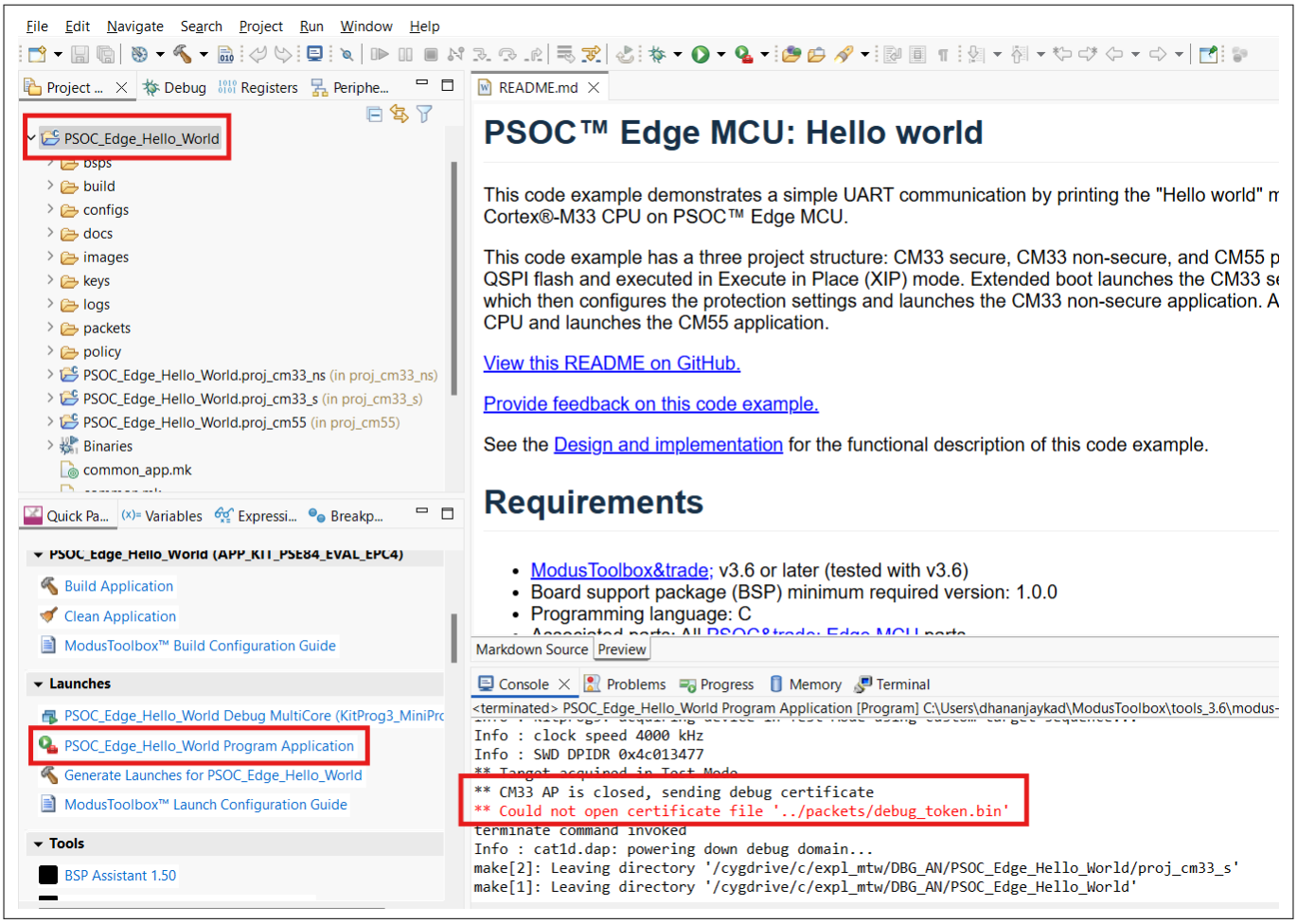

Figure 2 Verifying that debug ports are locked

4.6 Generating the debug certificate

To open debug ports, you need to provide the debug certificate. Generate the signed debug certificate using the following command:

```
edgeprotecttools -t pse8xs2 debug-token --template packets/debug_token.json --output ./packets/
debug_token.bin --key keys/debug_priv_key.pem
```

The default debug_token.json has Non-invasive debug (niden), invasive debug (dbgen), secure privileged invasive debug (spiden) and secure privileged non-invasive debug (spniden) enabled by default.

4.7 Opening debug ports using the debug token

You can open the debug ports using debug tokens with one of the following options:

- Load the debug certificate using SYS_AP
- Load the debug certificate using the user application

4.7.1 Loading the debug certificate using SYS_AP

This model requires SYS_AP to be accessible to a debugger. You must configure the policy accordingly and provisioned the device. To open debug ports, the debugger does the following:

1. Injects a signed debug certificate into the SRAM
2. Sets the BOOT_DLM_CTL register to indicate that the new debug certificate is loaded on to the SRAM
3. Sets the start address of the debug certificate in the BOOT_DLM_CTL2 register
4. Resets the device

Upon reset, the Boot ROM verifies the authenticity of the debug certificate and configures the AP_CTL register based on a combination of predefined debug policies and the information contained within the debug token.

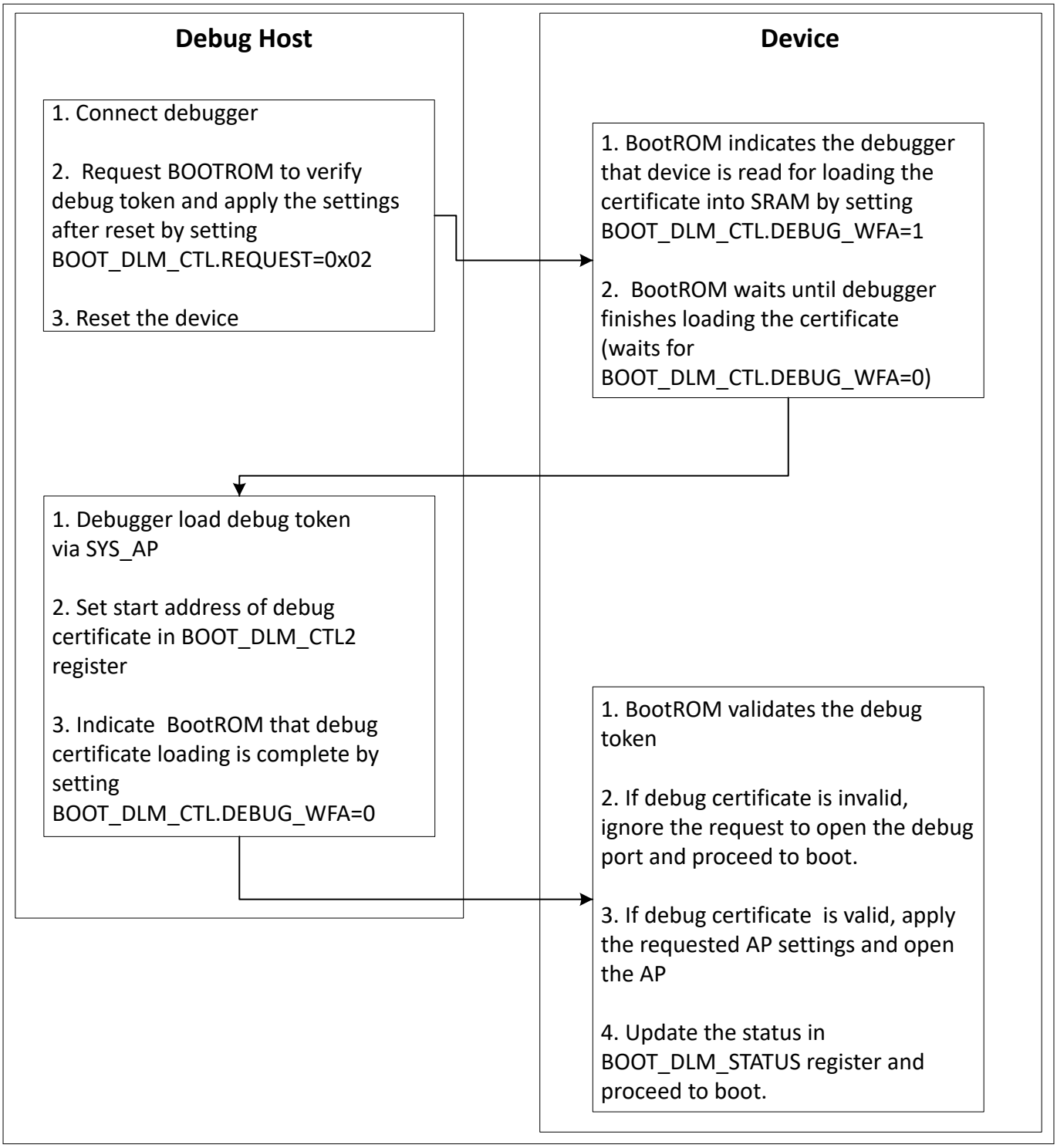

Figure 3 Loading the debug certificate using SYS_AP workflow

1. Ensure that debug certificate is generated and present in packets folder as instructed in previous section

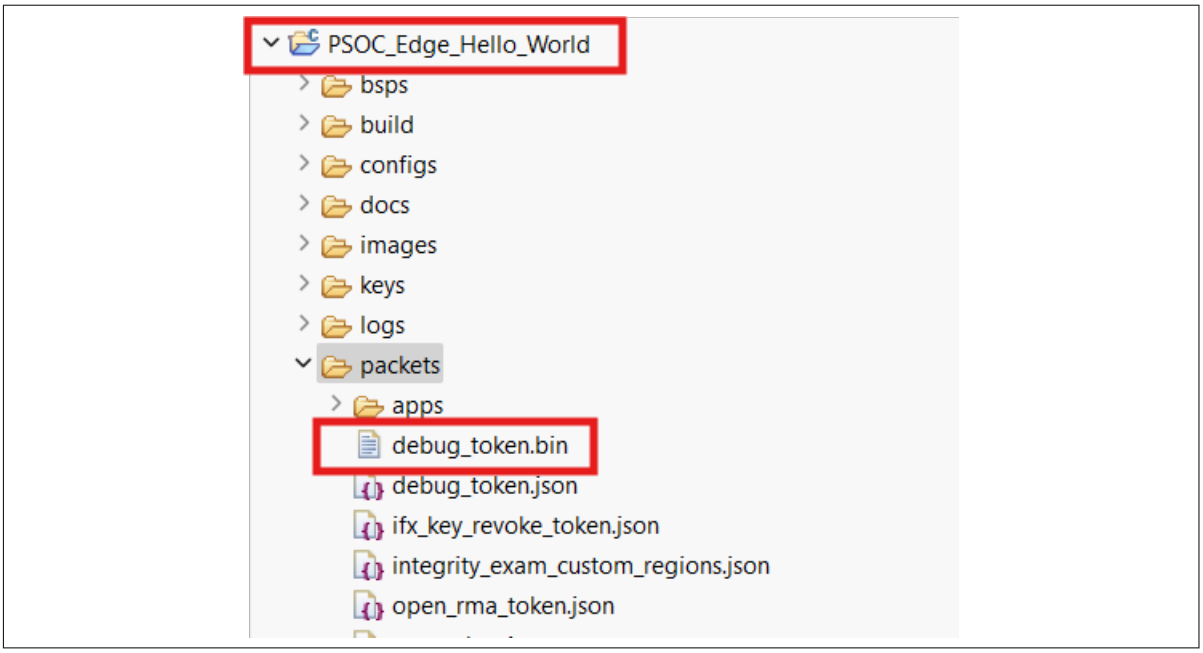

Figure 4 Debug certificate location

2. The default Program and Debug configurations uses the `packets/debug_token.bin` path to pick debug certificate. If required, the default path of debug certificate can be changed by providing new path to `CY_DBG_CERTIFICATE_PATH` variable in common.mk makefile of application followed by generating new Launches for the application. For example, if the debug certificate is placed at `PSOC_Edge_Hello_World/user_tokens/debug_token.bin` then the common.mk can be updated as shown below:

```
CY_DBG_CERTIFICATE_PATH=./user_tokens/debug_token.bin
```

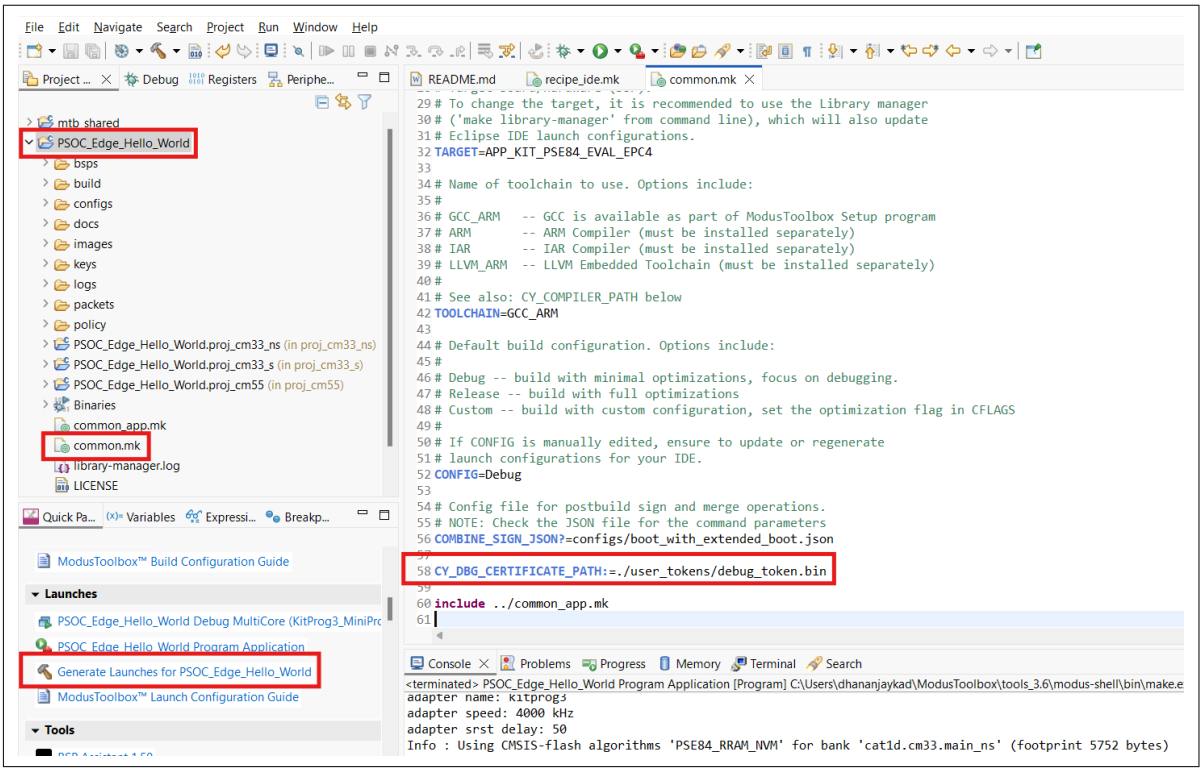

Figure 5 Change Debug Certificate path

Generating new Launches updates the program and debug configuration with new user defined debug certificate path. The following figure shows the updated debug certificate path for proj_cm33_s debug configuration.

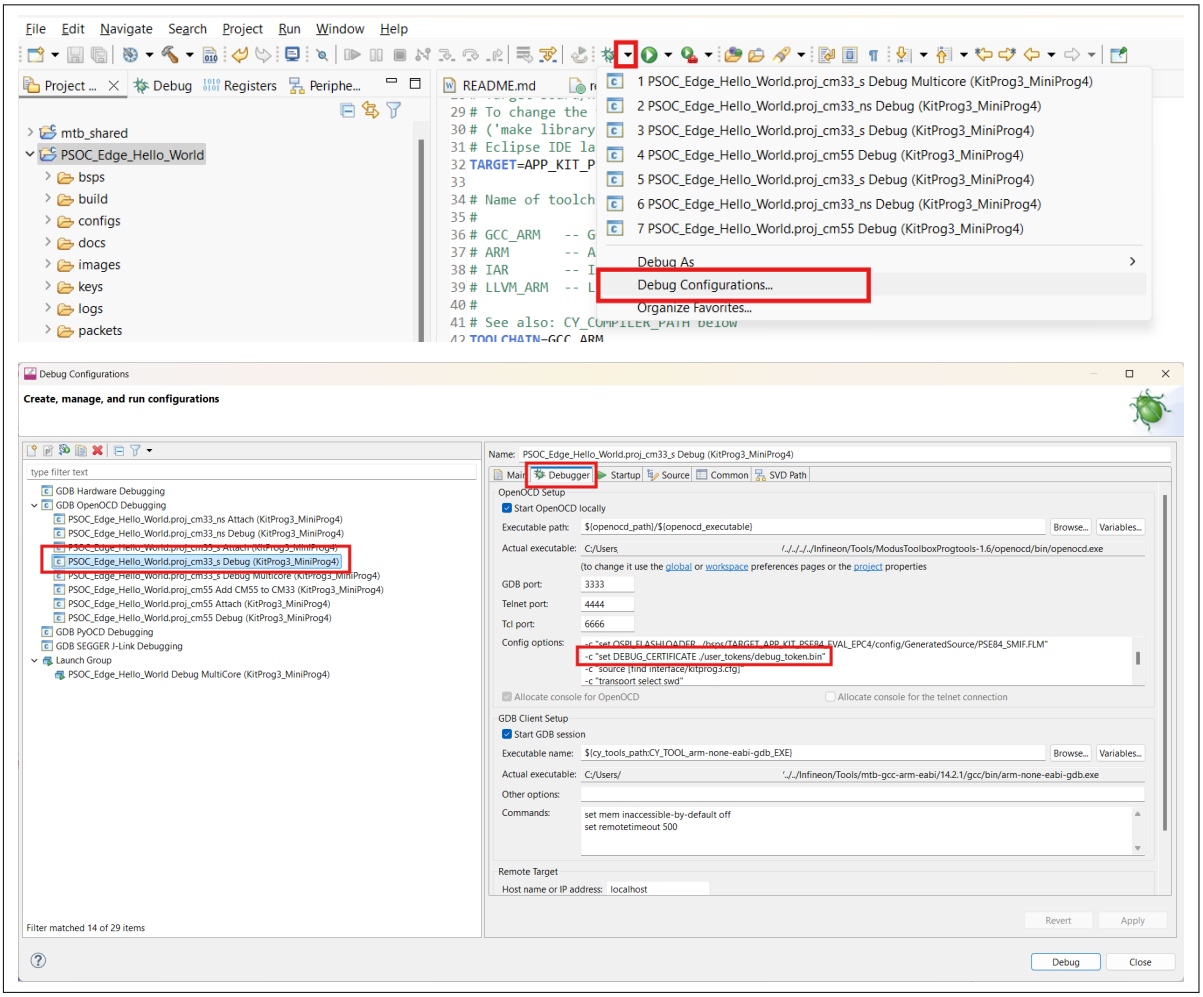

Figure 6 Updated Debug Configuration

3. Program the application. You should see debug ports are open and can be used for programming/ debugging as before.

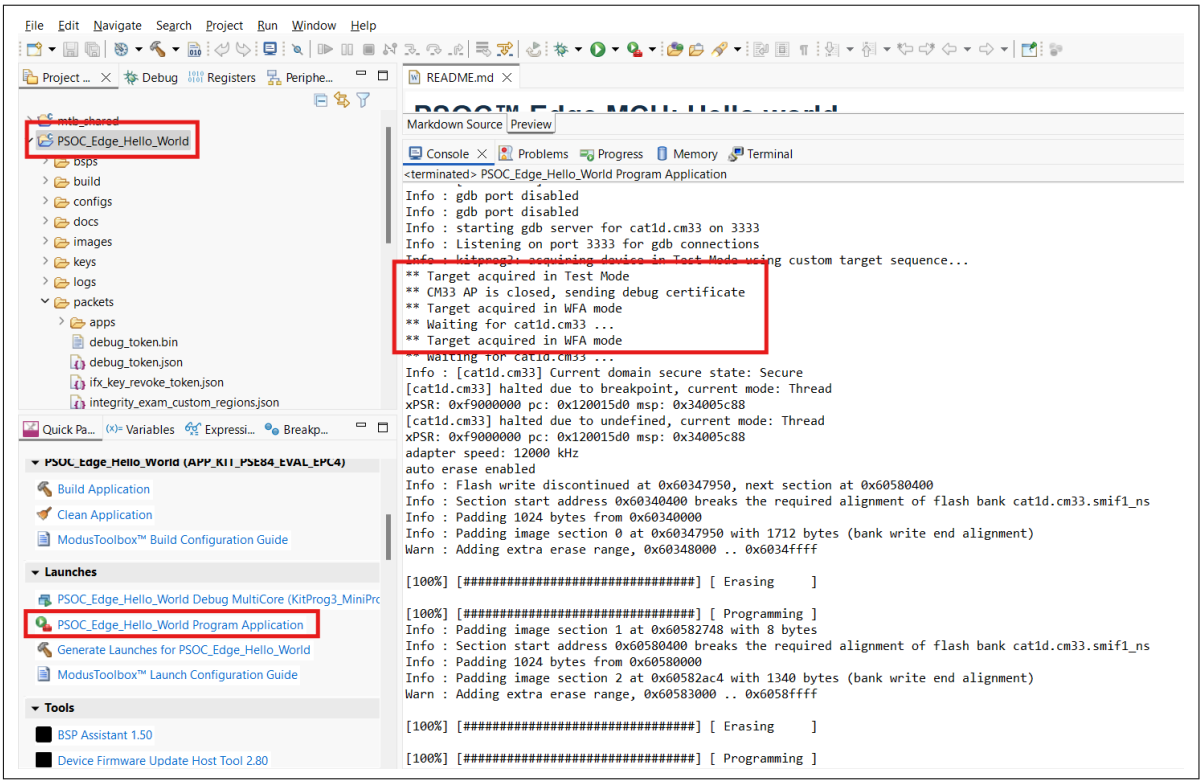

Figure 7 Programming the application

4.7.2 Loading the debug certificate using the user application

This is an alternative method to the earlier one to load the debug certificate. In this method, you may close down both CM33_AP and SYS_AP. Unlike the earlier case, it is not mandatory to keep SYS_AP accessible. However, the responsibility of receiving the debug certificate from the host is with the application running on the device. The application can receive the debug token from any interface of its choice, such as serial or wireless. The interface to use and the method to get the certificate on the device is application-specific, and outside the scope of this document.

Once the application receives the debug certificate, it must request the trusted firmware to place the debug token in the appropriate memory location, notify the Boot ROM, and initiate a reset to commence the execution of the Boot ROM. You need to implement the application yourself to get the debug certificate, place it in the SRAM, notify the Boot ROM using BOOT_DLM_CTL, and reset the device to transfer control to the Boot ROM for further process.

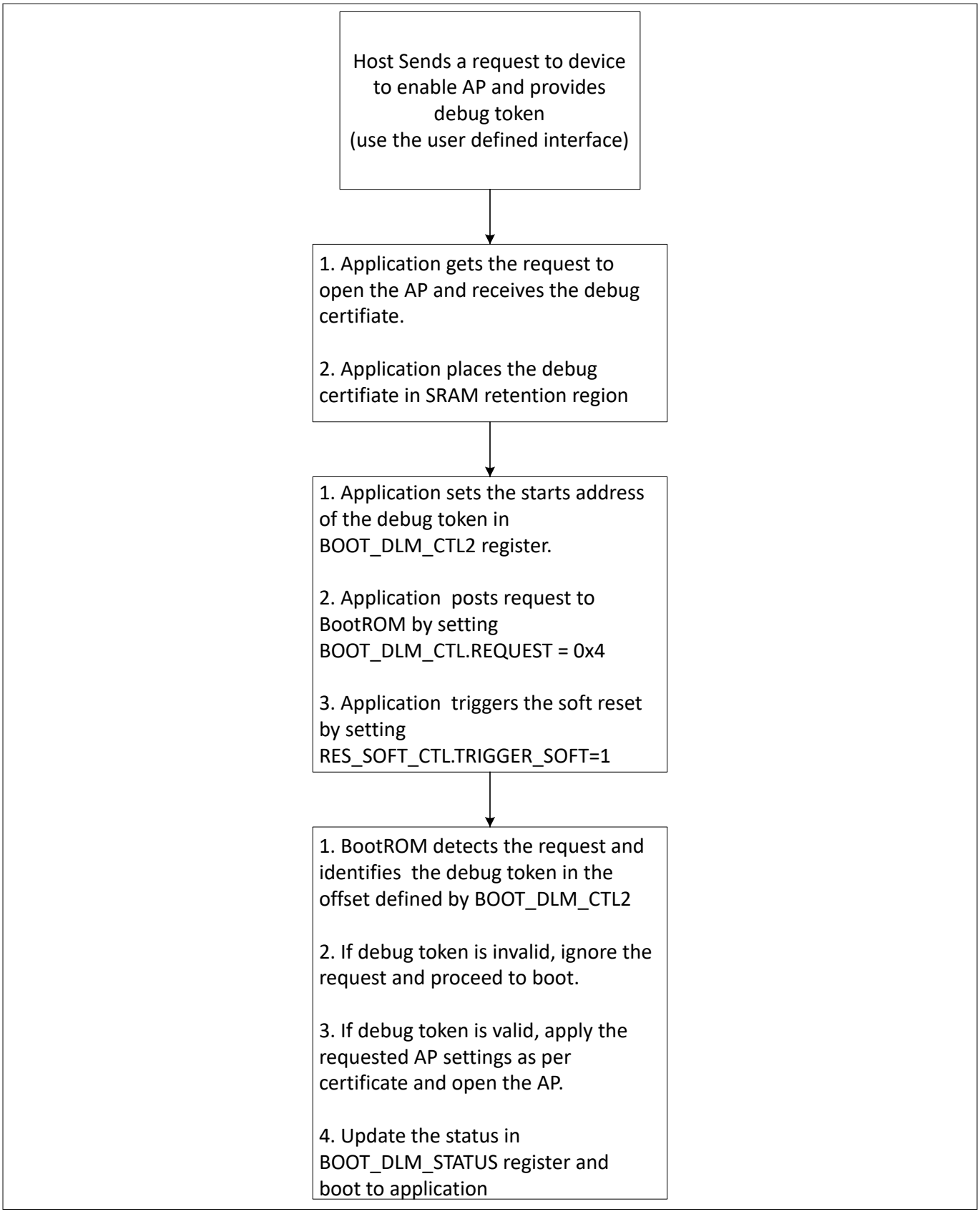

Figure 8 Loading the debug certificate using the user application

4.8 Securing CM55_AP

Earlier sections of this document described implementing security measures for both CM33_AP and SYS_AP and enabling CM33_AP with the authenticated debug method through SYS_AP. Note that CM55_AP has not been addressed because it is not governed by the debug policy. It remains open if CM33_AP is Enabled else it remains closed when CM33_AP is set as Allowed or Disabled. When CM55_AP is closed, it can be controlled by software. The following code snippet shows how CM55_AP can be enabled or disabled by software using Infineon's Peripheral Driver Library (PDL):

```
/* Note: PD1 must be ON before configuring DBG port */

/* Disable M55 DBG ports */
Cy_System_SetCM55DbgPort(APPCPUSS_DBG_DISABLE);

/* Enable M55 DBG ports */
Cy_System_SetCM55DbgPort(APPCPUSS_DBG_ENABLE_ALL);
```

An open CM55_AP presents a security risk, highlighting the need to secure CM55_AP. The CM55 debug access port can be controlled through the APPCPUSS_AP_CTL register. For more details, see PSOC™ Edge MCU registers. To secure CM55_AP, you need to protect the APPCPU_AP_CTL register by limiting its access only to trusted software. You can apply the protection on this register and limit the access to a specific Protection Context (for example, PC=2) such that only trusted software can enable the debug port.To learn more about how to apply protection settings, see the PSOC™ Edge technical reference manual.

5 Recommendation for production

It is recommended to move the device LCS to "Production" before shipping the devices out of the OEM. The following table lists the recommended policy configurations for some of the common use cases on the field:

1. Permanently close down all debug ports
2. Authenticated debug using certificates
   a. Using SYS_AP to load certificates
   b. Using user application to load certificates
3. Authenticated debug using trusted firmware

Table 4 Debug port configurations for Production LCS

Policy		Default value	Use case-specific recommendations			
Debug policy	**Policy field**		**Permanently close down all debug ports**	**Authenticated debug when Allowed=Certificates (SYS_AP loads debug certificates)**	**Authenticated debug when Allowed=Certificates (User application loads the debug token)**	**Debug using Allowed=Firmware**
CM33_AP	cpu	Enabled	Disabled	Allowed	Allowed	Allowed
	allowed	Open	x	Certificate	Certificate	Firmware
	dbgen	true	x	x	x	x
	niden	true	x	x	x	x
	spiden	true	x	x	x	x
	spniden	true	x	x	x	x
	secure	true	x	x	x	x
SYS_AP	ap	Enabled	Disabled	Allowed	Disabled	Allowed
	allowed	Open	x	Certificate	x	Firmware
	mpc_ppc_enable	true	x	true	x	true
	rram_otp_enable	Entire region	x	Nothing	x	Nothing
	rram_nvm_enable	Entire region	x	Nothing	x	Nothing
	ram_enable	Entire region	x	Entire region	x	Entire region
	mmio_enable	All MMIO registers	x	RAM app only	x	RAM app only
RMA	permission	true	false	false	false	false

"**x**" in this table indicates that these values are ignored by the Boot ROM.

5 Recommendation for production

***Note**: The RMA policy is consistently set to 'false' in all use case-specific recommendations. This prevents the device from being moved to the RMA LCS. If you anticipate requiring RMA support in the future, you should set the RMA policy value to 'true'. Note that the RMA field will be ignored if all debug ports have been permanently closed down. A discussion on RMA is out of the scope of this document.*

References

1. Infineon Technologies AG: *PSOC™ Edge MCU: Hello World*; Available online
2. Infineon Technologies AG: *Getting started with PSOC™ Edge E8 MCU on ModusToolbox™*; Available online
3. Infineon Technologies AG: *Getting started with PSOC™ Edge security*; Available online

Glossary

CM33 CPU

Low-power 32-bit Arm® Cortex®-M33 CPU in PSOC™ Edge

CM55 CPU

High-performance 32-bit Arm® Cortex® -M55 CPU in PSOC™ Edge

Debug Access Port (DAP/AP)

There may be multiple APs in this device that are accessible to the customer; System (SYS_AP), CM33 (CM33_AP), and CM55 (CM55_AP)

Debug certificate

A debug token is signed with the private debug key to generate a debug certificate. When this certificate is presented to the device, the Boot ROM validates and applies these configurations on the AP

Debug key

Pair of ECC256 keys dedicated to debug authentication. The public key must be provisioned to the device while the private key must be used to sign the debug certificate

Debug token

Debug token is a simple JSON file that is used create debug certificates

Debug_Policy

Part of the policy JSON file that defines the debug access control for CM33

Development LCS

Stage the device is in during development. In this stage, no permanent changes are made to the device, and it may be reverted back to its original state that was shipped from Infineon. You may transfer ownership as many times as needed while in this stage

EdgeProtectTools

EdgeProtectTools is a software application in ModusToolbox™ environment that provides security features for Infineon's edge computing products

Lifecycle stage (LCS)

The device is shipped from Infineon in the Development LCS and then will be advanced to the Production LCS when the customer ships the product

MIMO

Memory-mapped I/O

MPC

Memory protection controller of PSOC™ Edge

OEM

Original Equipment Manufacturer, Infineon's customers

PPC

Peripheral Protection Controller of PSOC™ Edge

Glossary

Provision

This is the process by which the OEM loads their custom software, keys, and boot parameters

RAM app

Executable code that is loaded into the SRAM. There are two types of RAM apps: SE RAM apps, which can only be written and signed by Infineon and only run on the M0s in the Secured Enclave. SE RAM apps are used for provisioning and to update the device resources such as extended boot and the SE runtime services. The second type of RAM apps is the CM33 RAM app that can be written and loaded by the OEM and executed by CM33

Return merchandise authorization (RMA)

LCS that is used if a device is suspected to be defective. The OEM places the device in this stage prior to shipment back to Infineon

RRAM

Resistive Random Access Memory

Secured boot

In the context of this document, this refers to when the first OEM code must be signed and then authenticated by the internal extended boot code

Serial Wire Debug (SWD)

A method used to interface debug and programming tools to the device

SRAM

Static random access memory

Transfer of ownership

This occurs when the default OEM_CERT from Infineon is replaced with an OEM_CERT from the actual OEM (sometimes referred to as extending ownership)

Trusted firmware

A secure software that runs in the secure processing environment of (SPE) of the CM33 CPU. TF-M offered by Arm® is an example of trusted firmware that is supported on PSOC™ Edge

Revision history

Document revision	Date	Description of changes
**	2024-05-30	Initial release
*A	2026-02-23	Published to web

Trademarks

PSOC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSOC™.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2026-02-23

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-fqk1711947946558

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.