CryptoAuthentication™ Introduction and CryptoAuthLib usage of Harmony3

A Leading Provider of Smart, Connected and Secure Embedded Control Solutions

CAE Taiwan Team
Microchip Technology Inc.

2022 July

What’s major study in this course ?

- **Secure Embedded system and Cryptographic**
- **CryptoAuthentication™ production introduction**
- **Harmony Configuration**
  - CONSOLE system service
  - TIME System Service
  - COMMAND System Service
- **Usage of CryptoAuthLib**
- **CryptoAuthLib application with ECC608**
  - Get chip Revision and Unique Serial Number
  - Random and Nonce command
  - Chip Provision
  - MAC (Message Authentication Code) Symmetric Authentication
  - ECDSA Sign/Verify Asymmetric Authentication
  - Command and Response Package
  - Command Builder

Course Index (1/2)

- Secure Embedded Systems
- Cryptographic
- CryptoAuthentication™ ECCx08 Chip
- EVB introduction
- Harmony Configuration
  - 💻 Lab1 Project Compiler and Console Output Test
  - 💻 Lab2 TIME System Service – LED Toggle
  - 💻 Lab3 COMMAND System Service – Command Console
- Usage of CryptoAuthLib
  - 💻 Lab4 Initialize CryptoAuthLib and ECC608 Chip
  - 💻 Lab5 Get Chip Revision and Unique Serial Number
  - 💻 Lab6 Random and Nonce Command
  - 💻 Lab7 Chip Provision

Course Index (2/2)

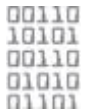
Secure Embedded Systems

Security: a mandatory pillar in every new embedded design

Today's weaknesses

Security by design : embedded security is now being **considered but hard to implement**

Lack of education : The **chain of trust** principle is not well understood, complex, hard to implement and consequently **incorrectly implemented**



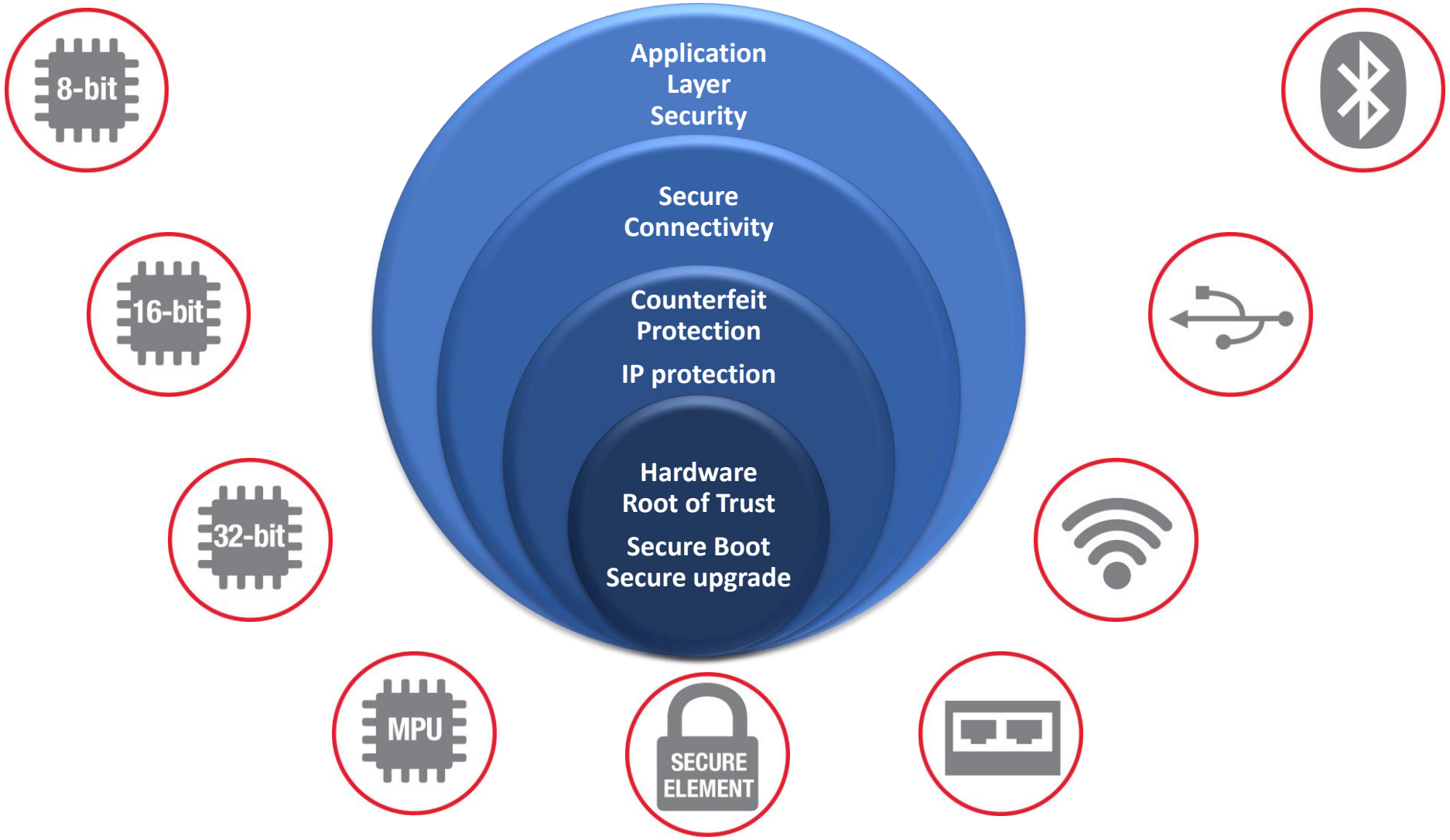
Keys/Certificates mishandling**: Private keys** are being handled by software at best and **left accessible in the clear** of the system memory

Backdoors are consequently left open to hackers – they attack the weakest point, in IoT, the **unsecure software** and exploit the **user habits**

Manufacturing is not trustable nor scalable, not secure and create scalability issues

MICROCHIP

Anatomy of a Secure Embedded System



All those features will require a
Crypto-ALGORITHM (the math) triggered by a KEY (the secret)

MICROCHIP

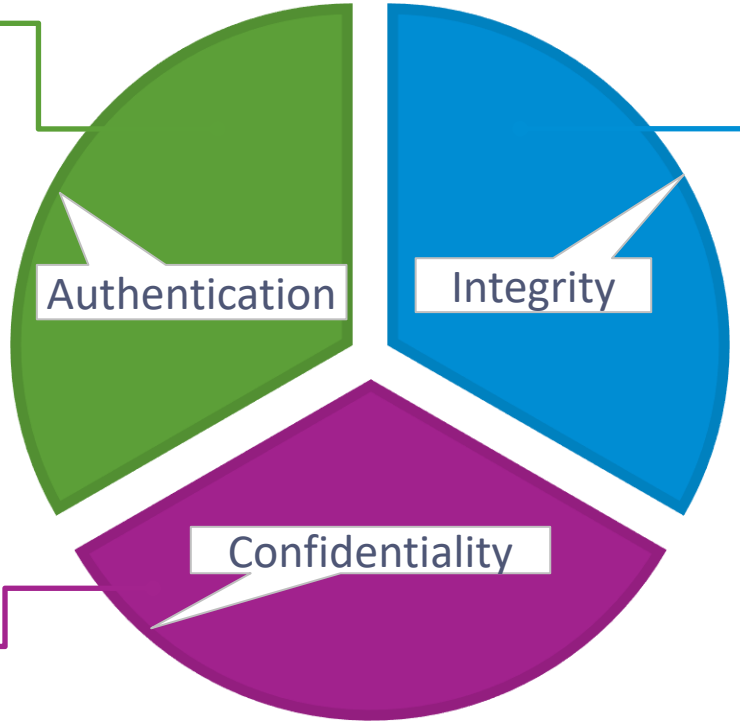
Secure Systems

Secure systems are built on 3 fundamentals

Authentication

Assures Someone/Something is what it is declared to be!

Simplified	*Technical Term*
Guranteed Identity	Asymmetric / Symmetric
Establish Trust	Attestation
Shared Secret	Signing
Genuineness	Digital Signature



Integrity

Ensures that the Content has not been manipulated!

Simplified	*Technical Term*
Seal	SHA256
Checksum of the Message	Hash
	Message Digest
	MAC

Confidentiality

Protects the Content from prying eyes!

Simplified	*Technical Term*
Password	Cipher
Obfuscate the Content	AES
	Encryption & Decryption

MICROCHIP

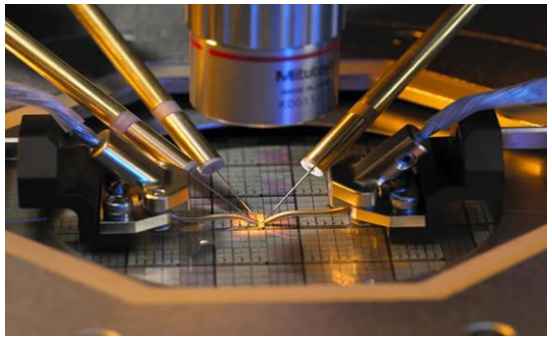
Already “CP=ON” or “SecureBit=Set” ?

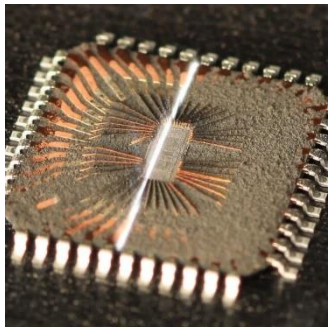
Output - 628config (Build, Load)

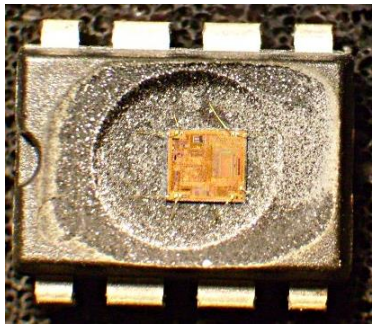
Address	Name	Value	Field	Option	Category
2007	CONFIG	FF10	FOSC	INTOSCIO	Oscillator Selection bits
			WDTE	OFF	Watchdog Timer Enable bit
			PWRTE	ON	Power-up Timer Enable bit
			MCLRE	OFF	RA5/MCLR/VPP Pin Function Select bit
			BOREN	OFF	Brown-out Detect Enable bit
			LVP	OFF	Low-Voltage Programming Enable bit
			CPD	OFF	Data EE Memory Code Protection bit
			CP	OFF	Flash Program Memory Code Protection bit

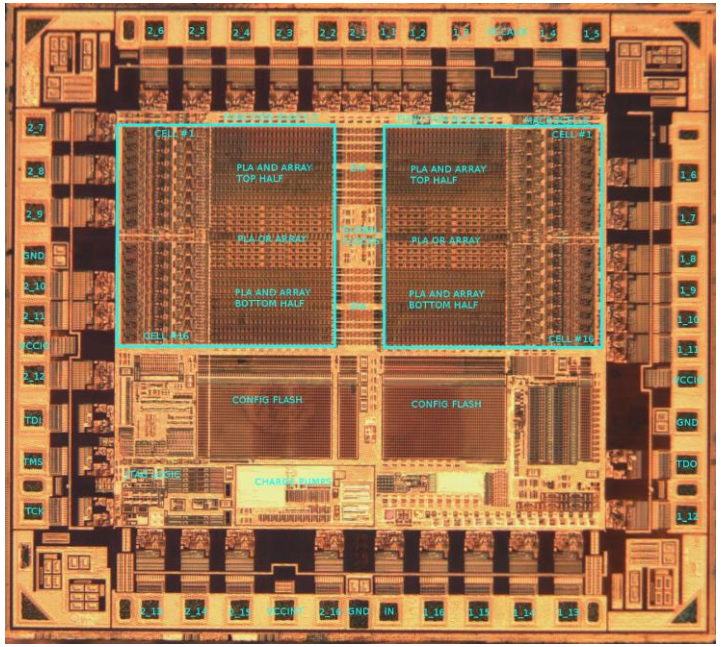
Configuration Bits

Setting
INTOSC oscillator: I/O function on RA6/OSC2
WDT disabled
PWRT enabled
RA5/MCLR/VPP pin function is digital input,
BOD disabled
RB4/PGM pin has digital I/O function, HV on
Data memory code protection off
Code protection off





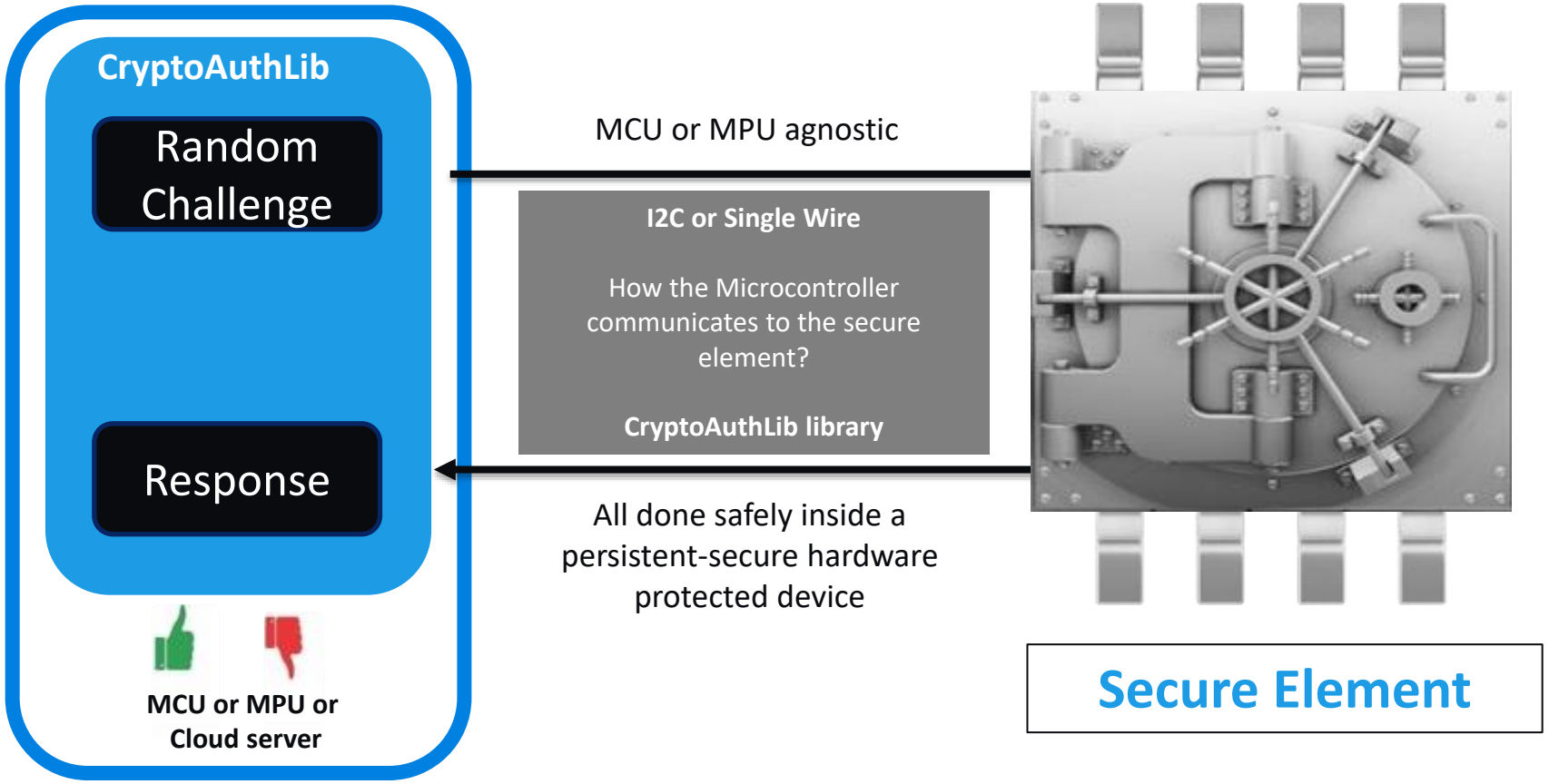




MICROCHIP

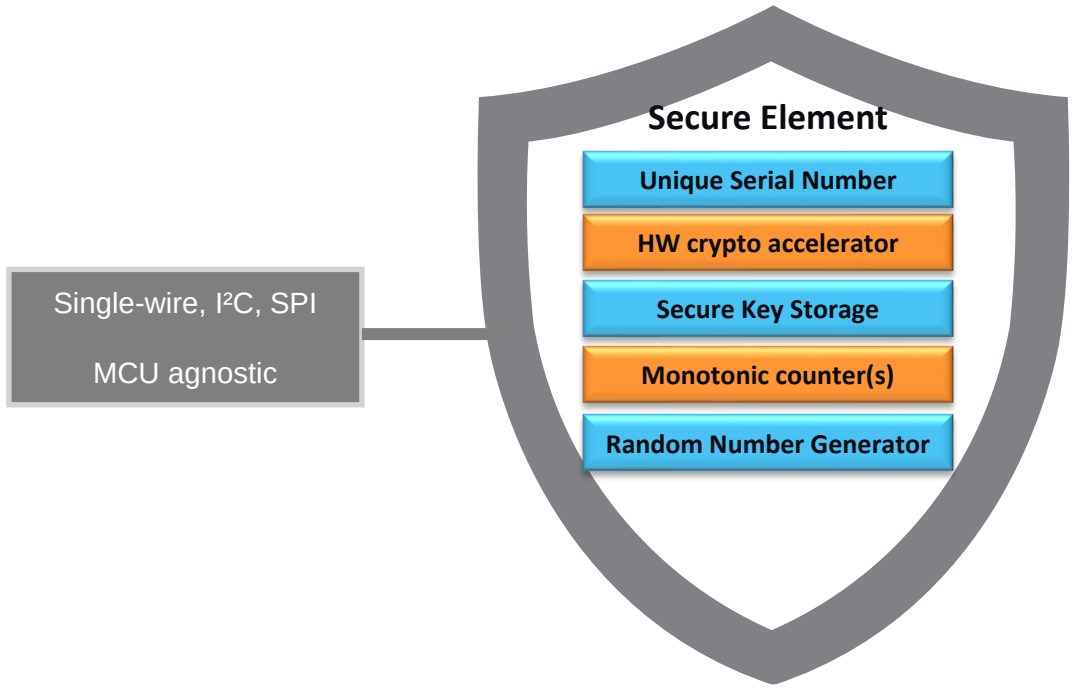
How to protect the keys ?

A secure element is a **vault** that **protects secrets**, it's a **companion** device to **any** microcontroller
Secrets are generated **inside** the vault **during manufacturing** - at Microchip secured factories
The secrets (keys, certificates) are **not exposed** and handled by Microchip **secure provisioning** service



Isolate keys in Hardware

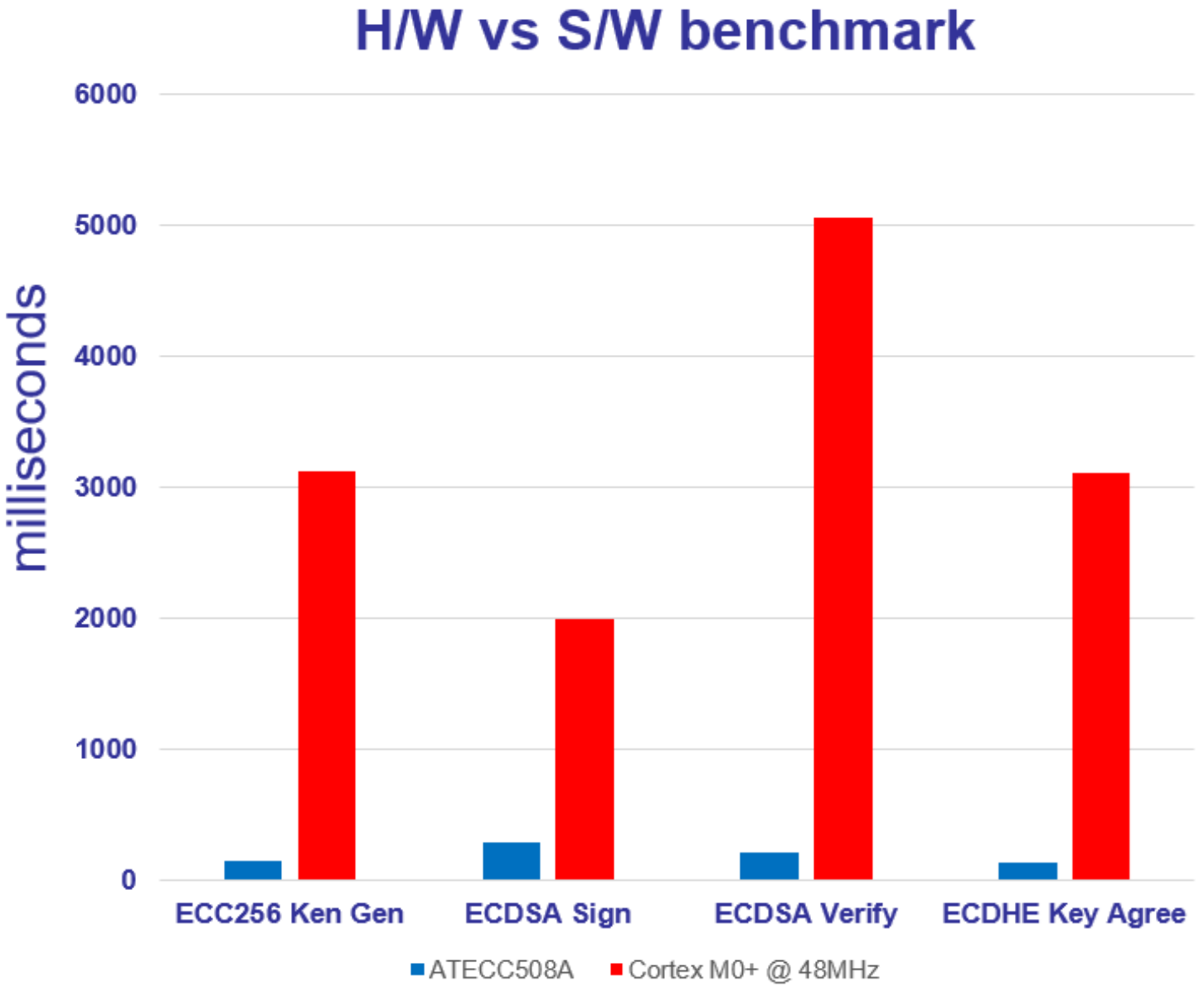
- The CryptoAuthentication ICs and CryptoAutomotive ICs securely store key material for the ***lifetime*** of the device.
- Keys are safely ***generated inside the device by the device*** using a “best in class” Random Number Generator (RNG) meeting NIST specifications
- Good security practices require all critical crypto-primitives are executed inside the device where the keys are **This keeps the keys, secrets and primitives *strictly segregated* from any vulnerable resources**



- Cryptographic acceleration performs primitives faster and with less power than MCU/MPU
- Information inside is uniquely scrambled and encrypted in EEPROM
- Each device has a unique serial number
- Monotonic counters for usage controls
- Multiple serial I/O options

MICROCHIP

Faster using Hardware Acceleration



MICROCHIP

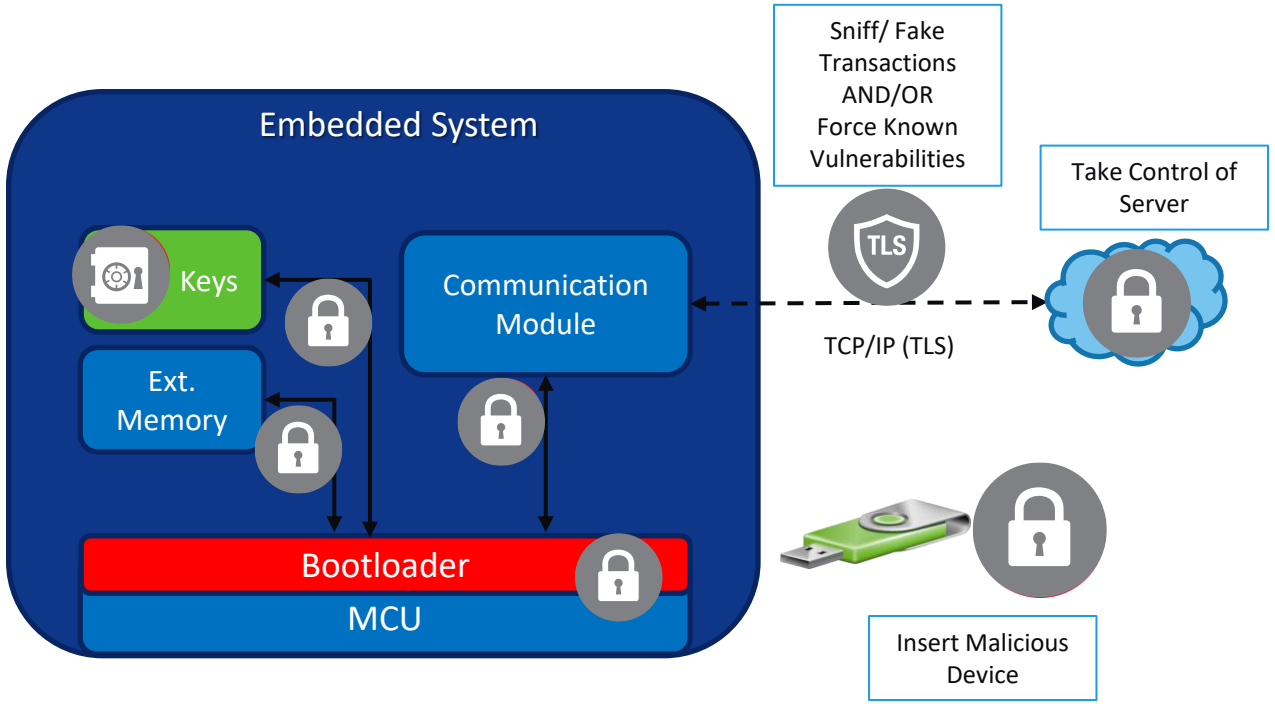
Threat Modelling

Developing Hack-resilient systems

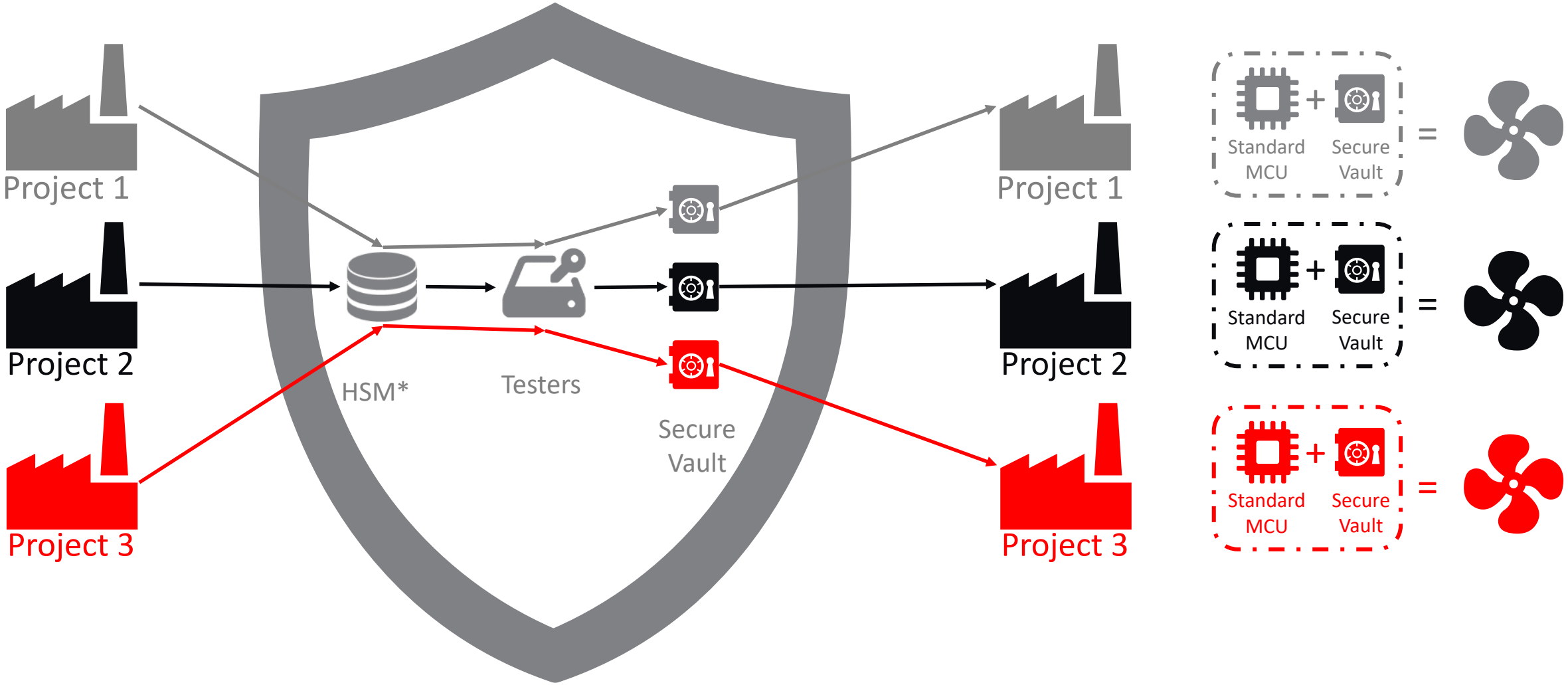
- **Define system requirements**
- **Risk Analysis**

→ Security Implementation

Next step : how to secure manufacturing ?



Secure Customization

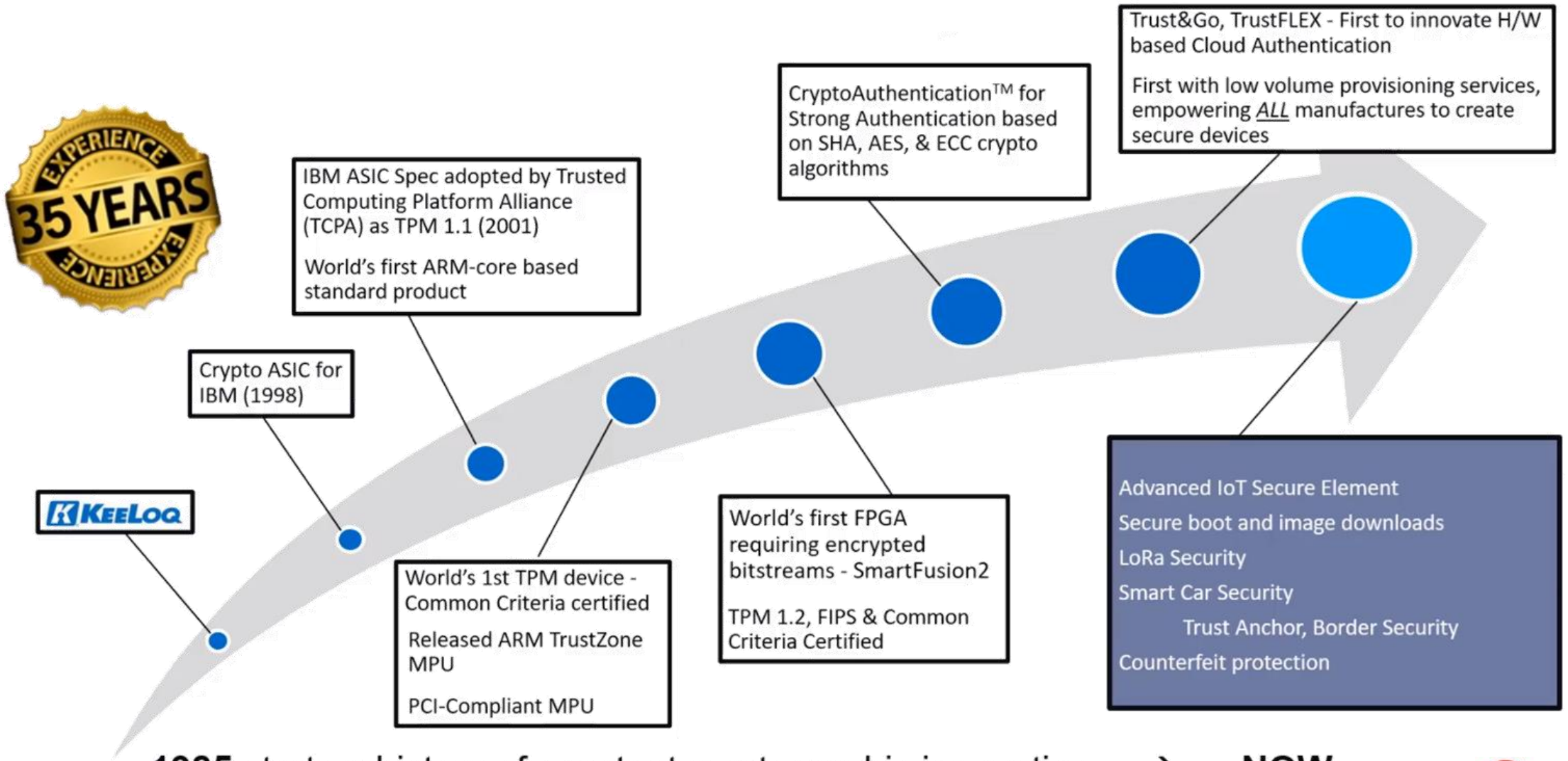


*HSM: Hardware security module in isolated network and dedicated secure rooms

MICROCHIP

A trusted and experienced team

Why Microchip's Security Products



1985 starts a history of constant cryptographic innovation → **NOW**

MICROCHIP

Secure Products & Application Segments

Accessories

Accessories or Disposable authentication
- USB type-C
- BLE
- Counterfeit protection of disposable goods
- Wireless Charging

WIRELESS POWER CONSORTIUM

USB Type-C

ATSHA204A
ATSHA206A
ATECC608

IoT

Connected Nodes
- Simple Security Upgrade to Any IoT Node
- Seamless Key & Certificate Provisioning
- Node Authentication
- Communication Encryption

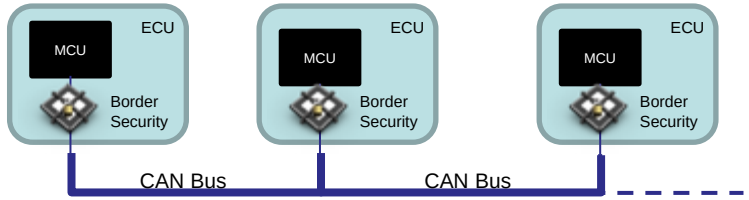
ATECC608

Automotive

In-Vehicle Network Security
- Secure Boot & FW Update
- Message Authentication
- WPC / TLS / HDCP / USB-C Security
- EV Battery Authentication

WIRELESS POWER CONSORTIUM

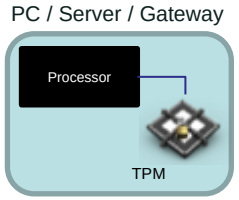
HDCP HIGH-BANDWIDTH DIGITAL CONTENT PROTECTION



TA100

Enterprise

Trusted Platform Module (TPM)
- TCG v1.2
- TCG v2.0



AT97SC

Our secure elements are supported by an in-house **secure provisioning service** that places the secrets and keys inside the device to isolate them from manufacturing, users, software

MICROCHIP

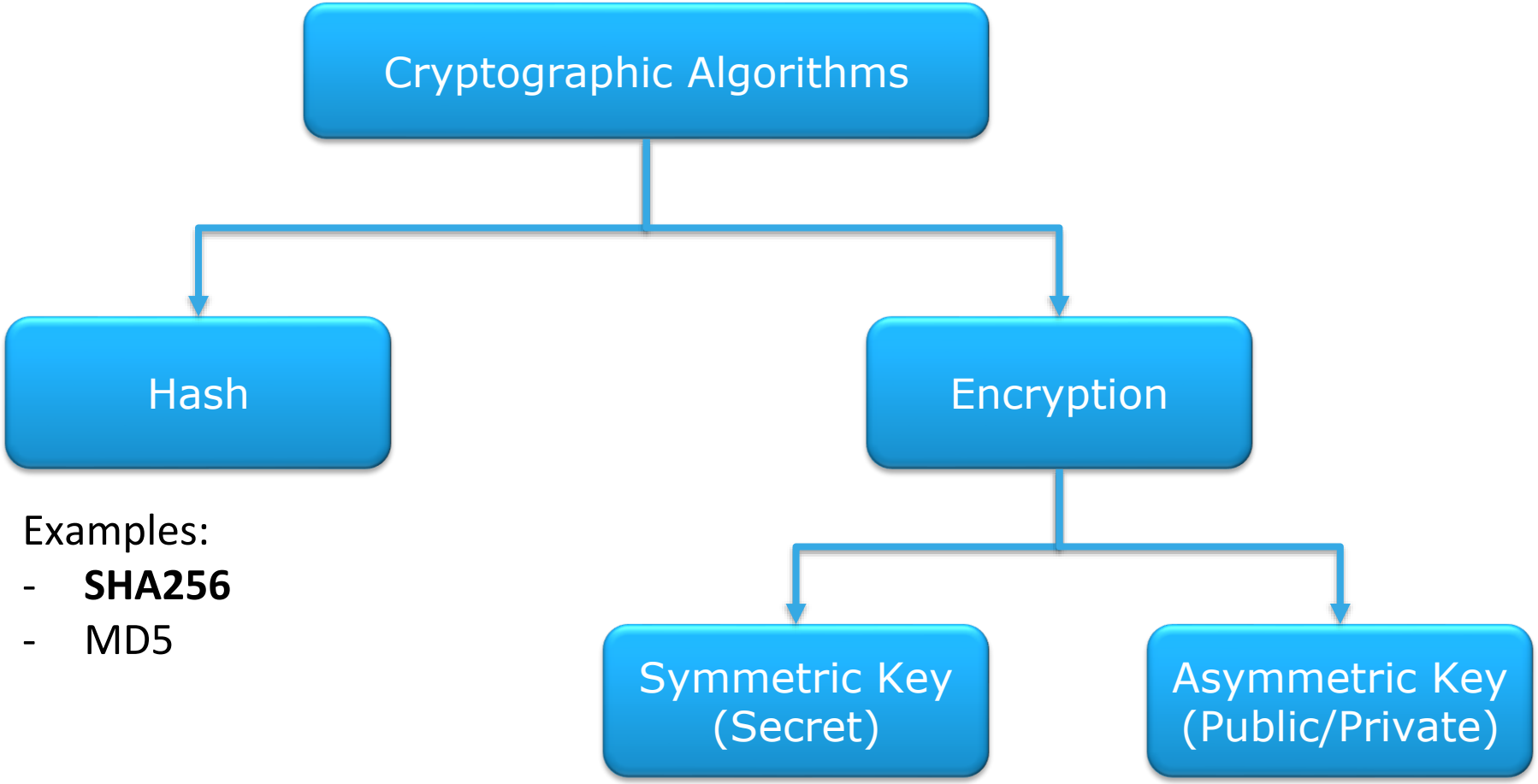
Trust Platform : pre-configuration

	TRUST&GO	TRUST FLEX	TRUST CUSTOM
Pre-configured	YES	YES	NO
Development Time	Lowest	Lower	Custom
Complexity	Lowest	Lower	Custom
Low MOQ Flow (<100ku EAU)	10 units MOQ	2000 units MOQ	4000 units MOQ
High Volume flow (>100ku EAU)	Starting 30 000 units MOQ	Starting 30 000 units MOQ	Starting 30 000 units MOQ
Secure Key Storage	JIL High	JIL High	JIL High
Use cases	TLS authentication LoRaWan authentication	TLS authentication LoRaWan authentication Token authentication Key rotation Firmware verification (OTA, Secure boot) IP protection Public key attestation (A-)symmetric Accessories authentication (A-)symmetric Disposable authentication	Fully customizable
Devices	ATECC608	ATECC608	ATECC608 ATSHA204A (w/o RBH)

MICROCHIP

Cryptographic

The Big Three



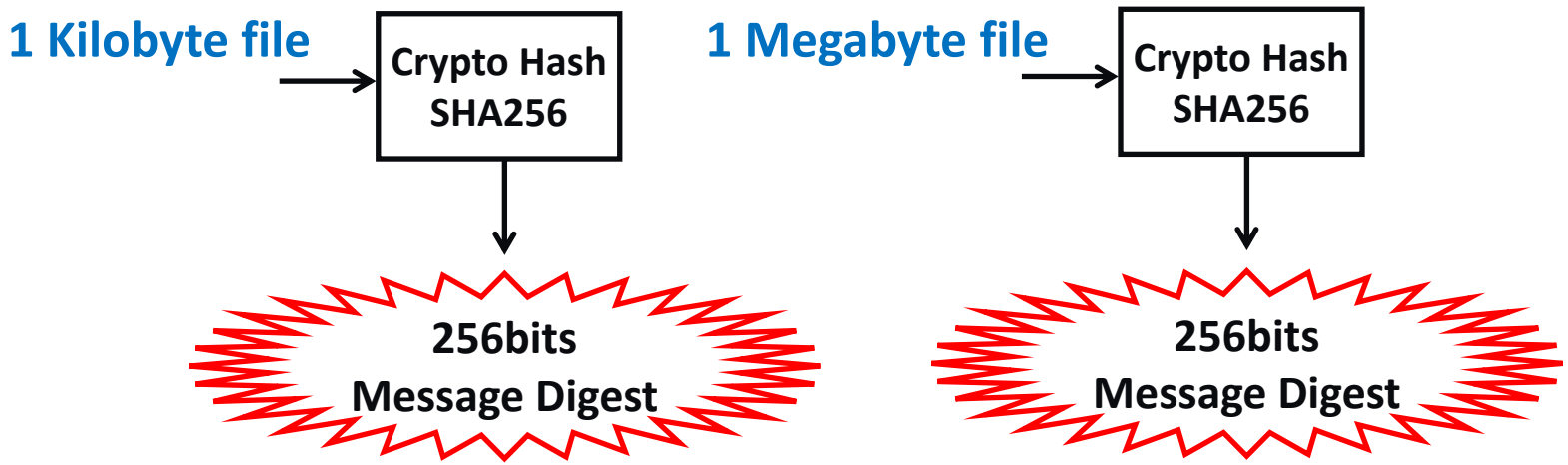
Examples:
- **SHA256**
- MD5

Examples:
- **AES 128**
- DES

Examples:
- **ECC P256 (Elliptic curve)**
- RSA

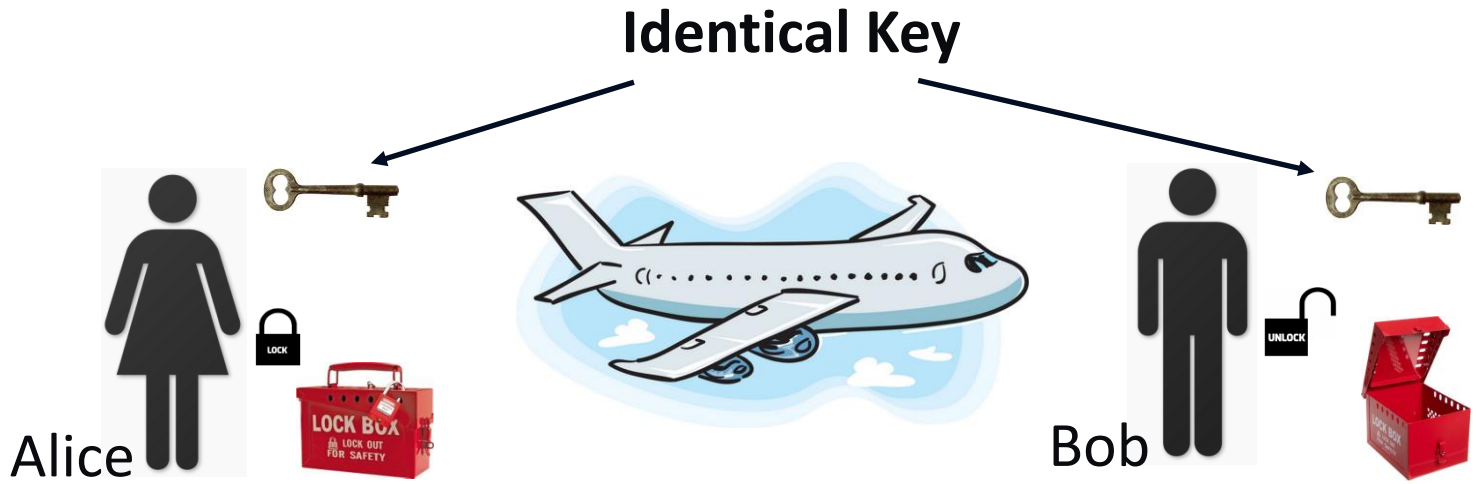
A HASH is a fundamental building block of Cryptography

- **A crypto hash uses a strong, irreversible, math transform**
  - Easy to compute the digest,
  - Infeasible to regenerate the original message
  - Infeasible to modify a message without changing the digest
  - Infeasible to find different messages with the same digest
- **Output is fixed length**
  - SHA-256 outputs a 256bits(32 Bytes) digest no matter the size of its input
- **Examples:**
  - SHA256, SHA-1, MD5, CRC



Symmetric Key Cryptography

- **Other common terms for symmetric key:**
  - Secret-key, Single-key, and Shared-key
- **Uses *identical* key(s) for both encryption and decryption**
  - Example: using identical keys to lock and unlock a lock-box
- **Examples**
  - AES
  - DES / 3DES
  - KEELOQ® Security ICs

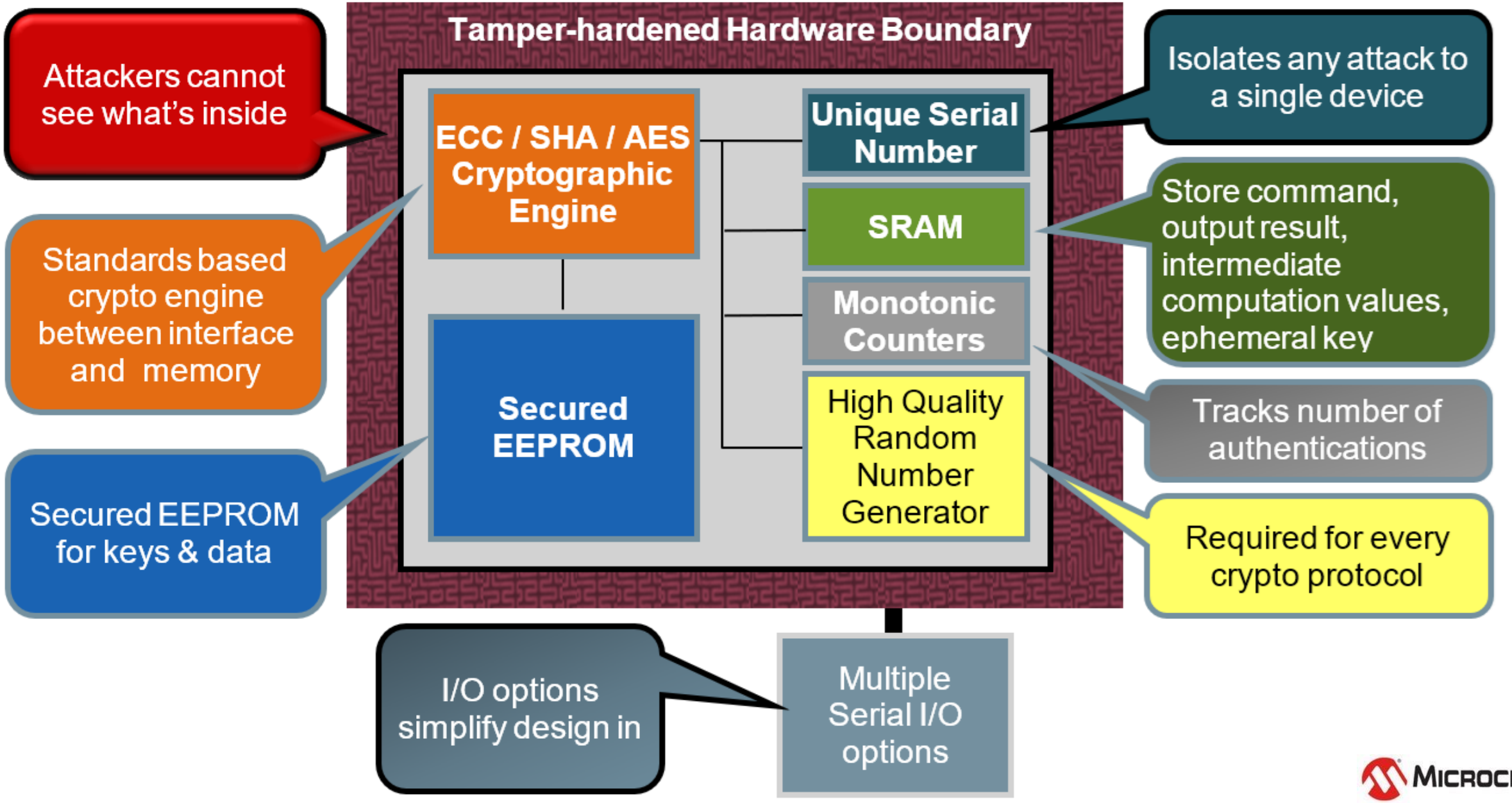


Public Key (Asymmetric) Cryptography

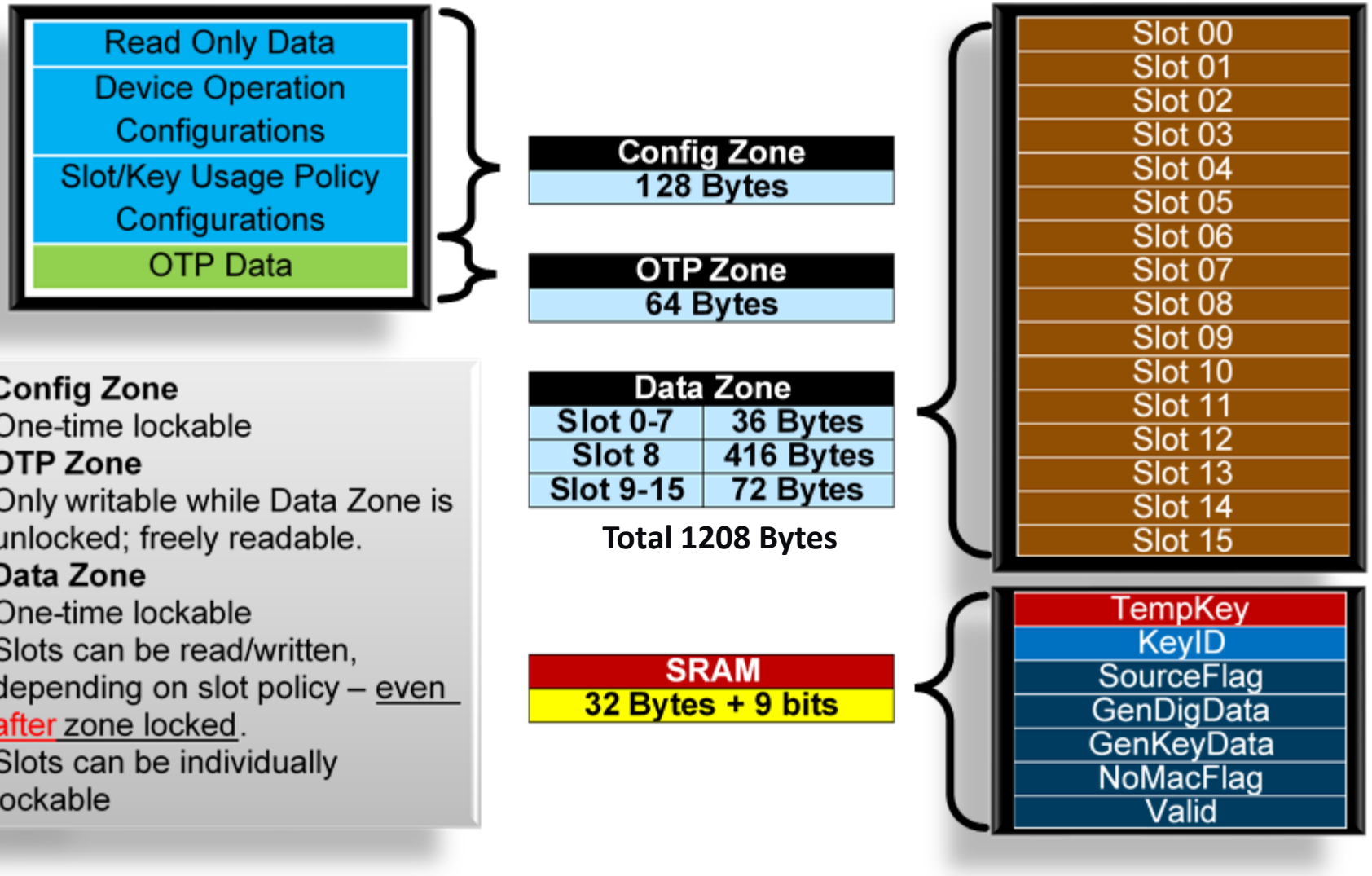
- **Uses two mathematically related keys**
- **Imagine having two different keys to the same lock box**
  - If the *first key* locked it, only the *second key* can unlock it
  - If the *second key* locked it, only the *first key* can unlock it
- **Key distribution is easy**
  - One of these keys can be shared publicly and is called the "**Public Key**"
  - The other is held very private and is called the "**Private Key**"
- ***The security of this key is critical***
- **Examples**
  - RSA
  - ECC(ECDSA/ECDH)

CryptoAuthentication™ ECCx08 Chip

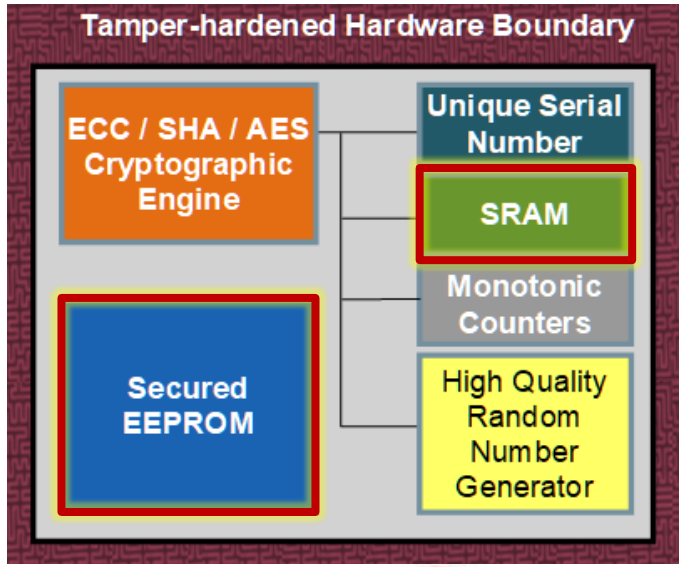
Block Diagram



Memory Organization



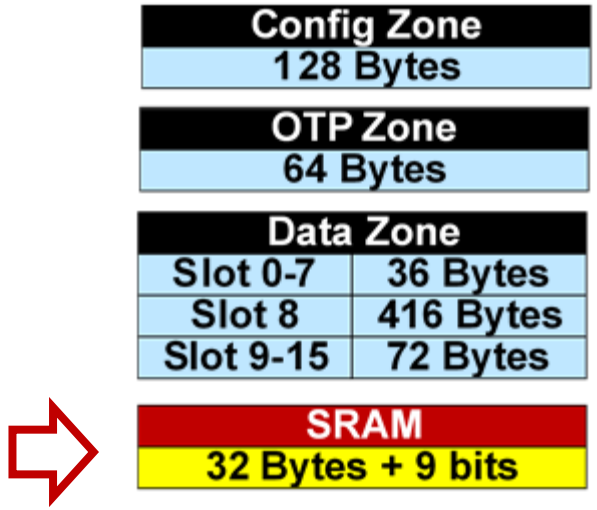
Config Zone
One-time lockable
OTP Zone
Only writable while Data Zone is unlocked; freely readable.
Data Zone
One-time lockable
Slots can be read/written, depending on slot policy – even after zone locked.
Slots can be individually lockable

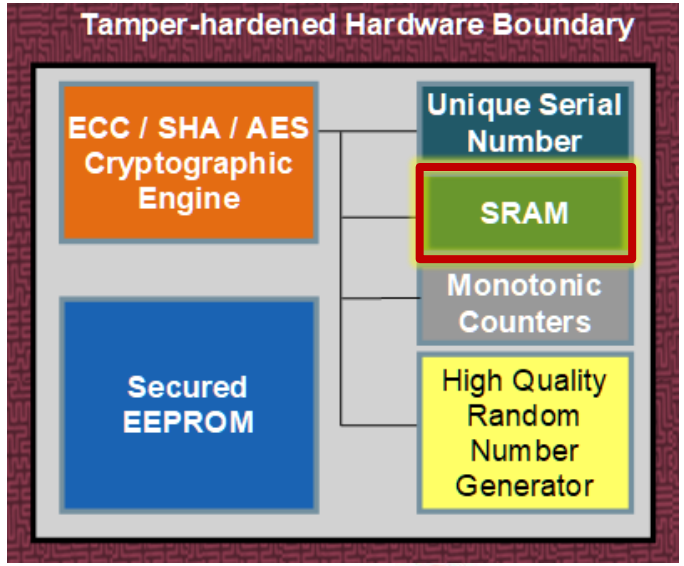


MICROCHIP

SRAM

Name	Length	Description
TempKey	256 bits (32 Bytes)	Nonce (from Nonce command) or digest (from GenDig or GenKey(Digest) commands).
KeyID	4 bits	If TempKey was generated by GenDig or GenKey, these bits indicate which key was used in its computation.
SourceFlag	1 bit	The source of the randomness in TempKey: 0 = Internally generated random number (Rand). 1 = Input seed only, no internal random generation (Input).
GenDigData	1 bit	0 = TempKey was not generated by GenDig. 1 = The contents of TempKey were generated by GenDig using one of the slots in the Data zone
GenKeyData	1 bit	0 = TempKey.KeyID was not generated by GenKey. 1 = The contents of TempKey were generated by GenKey using one of the slots in the Data zone
NoMacFlag	1 bit	0 = The contents of TempKey were generated and can be used with any of the MAC commands. 1 = The contents of TempKey were generated using the value in a slot for which SlotConfig.NoMac is one, and therefore cannot be used by the MAC and HMAC commands.
Vaild	1 bit	0 = The information in TempKey is invalid. 1 = The information in TempKey is valid.





MICROCHIP

Configuration Zone

Word	Byte	00	01	02	03
00	00	SN[0:1]		SN[2:3]	
01	04	RevNum			
02	08	SN[4:7]			
03	0C	SN[8]	Reserved	I2CEnable	Reserved
04	10	I2CAddress	Reserved	OTPmode	ChipMode
05	14	SlotConfig00		SlotConfig01	
06	18	SlotConfig02		SlotConfig03	
07	1C	SlotConfig04		SlotConfig05	
08	20	SlotConfig06		SlotConfig07	
09	24	SlotConfig08		SlotConfig09	
0A	28	SlotConfig0A		SlotConfig0B	
0B	2C	SlotConfig0C		SlotConfig0D	
0C	30	SlotConfig0E		SlotConfig0F	
0D	34	Counter0			
0E	38				
0F	3C	Counter1			
10	40				
11	44	LastKeyUse			
12	48				

Word	Byte	00	01	02	03
13	4C	LastKeyUse			
14	50				
15	54	UserExtra	Selector	LockValue	LockConfig
16	58	SlotLocked		Rfu90	
17	5C	X509Format[0:3]			
18	60	KeyConfig00		KeyConfig01	
19	64	KeyConfig02		KeyConfig03	
1A	68	KeyConfig04		KeyConfig05	
1B	6C	KeyConfig06		KeyConfig07	
1C	70	KeyConfig08		KeyConfig09	
1D	74	KeyConfig0A		KeyConfig0B	
1E	78	KeyConfig0C		KeyConfig0D	
1F	7C	KeyConfig0E		KeyConfig0F	

Note:

SN[0:1] always **0x01 0x23**

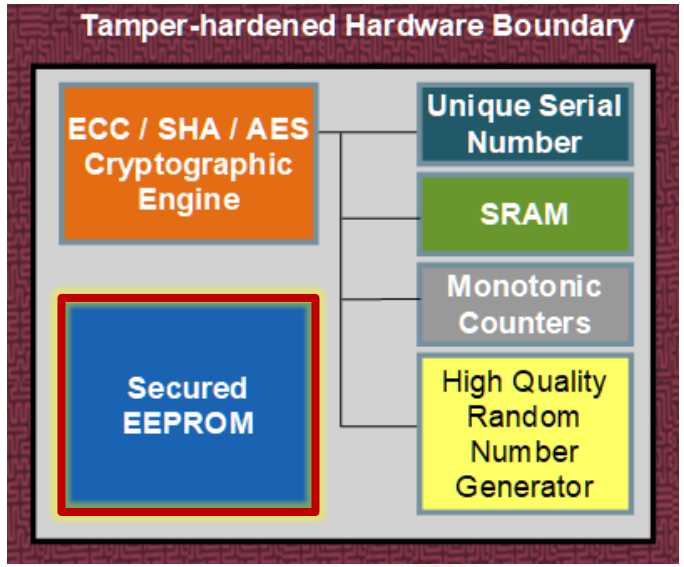
SN[8] always **0xEE**

Access of configure zone should apply it's WORD address. (one WORD equal four bytes)

Ex. SlotConfig00 Byte Address = 0x14

WORD address = 0x14 / 4 = **0x05**

Config Zone	
128 Bytes	
OTP Zone	
64 Bytes	
Data Zone	
Slot 0-7	36 Bytes
Slot 8	416 Bytes
Slot 9-15	72 Bytes
SRAM	
32 Bytes + 9 bits	



MICROCHIP

Configuration Zone

8	7	6	5	4	3	2	1	0
I2C Address (7 bits)								X

03	0C	SN[8]	Reserved	I2CEnable	Reserved
04	10	I2CAddress	Reserved	OTPmode	ChipMode
05	14	SlotConfig00		SlotConfig01	

Device	Default 7-bit I^2C Address	8-bit Programmed I^2C Address
ATECC608A-TNGTLS	0x35	0x6A
ATECC608A-TFLXTLS	0x36	0x6C
ATECC608A-MAHDA	0x60	0xC0

Configuration Zone

Byte #	Name	Description
14	I2C_Enable	Bit 0: 0 = The device operates in Single-Wire Interface mode. 1 = The device operates in I^2C interface mode.
16	I2C_Address	When I2C_Enable:0 is one, this field is the I^2C Address: Bit 0: Ignored. Bits 1-7: For I^2C interface parts the most significant seven bits of this byte for the device address value to which this device will respond.

MICROCHIP

Configuration Zone

Word	Byte	00	01	02	03
00	00	SN[0:1]		SN[2:3]	
01	04	RevNum			
02	08	SN[4:7]			
03	0C	SN[8]	Reserved	I2CEnable	Reserved
04	10	I2CAddress	Reserved	OTPmode	ChipMode
05	14	SlotConfig00		SlotConfig01	
06	18	SlotConfig02		SlotConfig03	
07	1C	SlotConfig04		SlotConfig05	
08	20	SlotConfig06		SlotConfig07	
09	24	SlotConfig08		SlotConfig09	
0A	28	SlotConfig0A		SlotConfig0B	
0B	2C	SlotConfig0C		SlotConfig0D	
0C	30	SlotConfig0E		SlotConfig0F	
0D	34	Counter0			
0E	38				
0F	3C	Counter1			
10	40				
11	44	LastKeyUse			
12	48				

Word	Byte	00	01	02	03
13	4C	LastKeyUse			
14	50				
15	54	UserExtra	Selector	LockValue	LockConfig
16	58	SlotLocked		Rfu90	
17	5C	X509Format[0:3]			
18	60	KeyConfig00		KeyConfig01	
19	64	KeyConfig02		KeyConfig03	
1A	68	KeyConfig04		KeyConfig05	
1B	6C	KeyConfig06		KeyConfig07	
1C	70	KeyConfig08		KeyConfig09	
1D	74	KeyConfig0A		KeyConfig0B	
1E	78	KeyConfig0C		KeyConfig0D	
1F	7C	KeyConfig0E		KeyConfig0F	

Note:

SN[0:1] always **0x01 0x23**

SN[8] always **0xEE**

Access of configure zone should apply it's WORD address. (one WORD equal four bytes)

Ex. SlotConfig00 Byte Address = 0x14

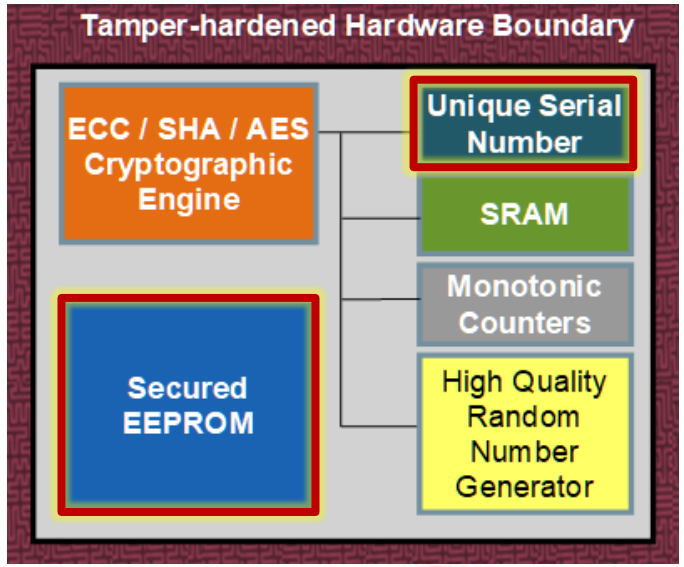
WORD address = 0x14 / 4 = **0x05**

Config Zone
128 Bytes

OTP Zone
64 Bytes

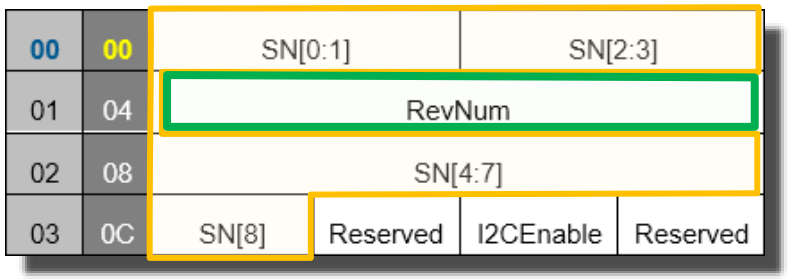
Data Zone	
Slot 0-7	36 Bytes
Slot 8	416 Bytes
Slot 9-15	72 Bytes

SRAM
32 Bytes + 9 bits



MICROCHIP

Configuration Zone



The revision number of ECC608A :

Byte 0	Byte 1	Byte 2	Byte 3
0x00	0x00	0x60	0x02

The serial number will be in the format of:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
0x01	0x23	xx	xx	xx	xx	xx	xx	0xEE

Bytes 0, 1 and 8 are set by Microchip at the factory. These bytes will be the same for all standard devices. These bytes can change when Microchip does volume custom programming of devices for a customer. Bytes 2 – 7 will be unique for all Crypto Authentication devices.

Configuration Zone

Word	Byte	00	01	02	03
00	00	SN[0:1]		SN[2:3]	
01	04	RevNum			
02	08	SN[4:7]			
03	0C	SN[8]	Reserved	I2CEnable	Reserved
04	10	I2CAddress	Reserved	OTPmode	ChipMode
05	14	SlotConfig00		SlotConfig01	
06	18	SlotConfig02		SlotConfig03	
07	1C	SlotConfig04		SlotConfig05	
08	20	SlotConfig06		SlotConfig07	
09	24	SlotConfig08		SlotConfig09	
0A	28	SlotConfig0A		SlotConfig0B	
0B	2C	SlotConfig0C		SlotConfig0D	
0C	30	SlotConfig0E		SlotConfig0F	
0D	34	Counter0			
0E	38				
0F	3C	Counter1			
10	40				
11	44	LastKeyUse			
12	48				

Word	Byte	00	01	02	03
13	4C	LastKeyUse			
14	50				
15	54	UserExtra	Selector	LockValue	LockConfig
16	58	SlotLocked		Rfu90	
17	5C	X509Format[0:3]			
18	60	KeyConfig00		KeyConfig01	
19	64	KeyConfig02		KeyConfig03	
1A	68	KeyConfig04		KeyConfig05	
1B	6C	KeyConfig06		KeyConfig07	
1C	70	KeyConfig08		KeyConfig09	
1D	74	KeyConfig0A		KeyConfig0B	
1E	78	KeyConfig0C		KeyConfig0D	
1F	7C	KeyConfig0E		KeyConfig0F	

Note:

SN[0:1] always **0x01 0x23**

SN[8] always **0xEE**

Access of configure zone should apply it's WORD address. (one WORD equal four bytes)

Ex. SlotConfig00 Byte Address = 0x14

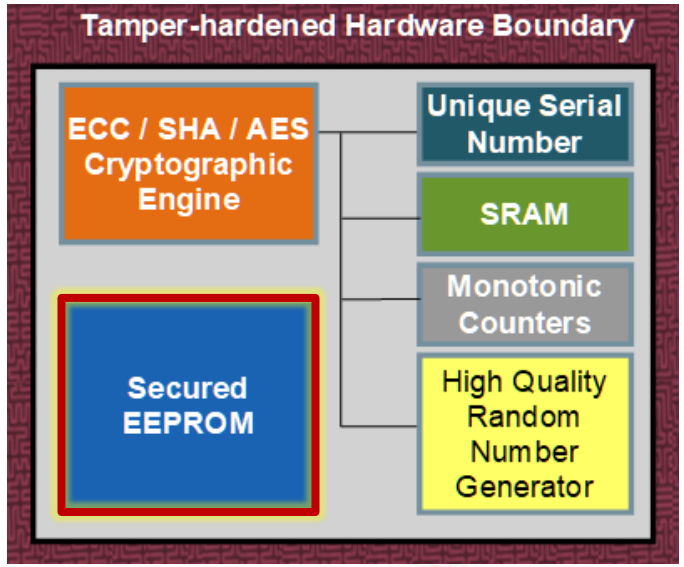
WORD address = 0x14 / 4 = **0x05**

Config Zone
128 Bytes

OTP Zone
64 Bytes

Data Zone	
Slot 0-7	36 Bytes
Slot 8	416 Bytes
Slot 9-15	72 Bytes

SRAM
32 Bytes + 9 bits



MICROCHIP

Configuration Zone

Prior Lock Config before write Data zone

Word	Byte	00	01	02	03
00	00	SN[0:1]		SN[2:3]	
01	04	RevNum			
02	08	SN[4:7]			
03	0C	SN[8]	Reserved	I2CEnable	Reserved
04	10	I2CAddress	Reserved	OTPmode	ChipMode
05	14	SlotConfig00		SlotConfig01	
06	18	SlotConfig02		SlotConfig03	
07	1C	SlotConfig04		SlotConfig05	
08	20	SlotConfig06		SlotConfig07	
09	24	SlotConfig08		SlotConfig09	
0A	28	SlotConfig0A		SlotConfig0B	
0B	2C	SlotConfig0C		SlotConfig0D	
0C	30	SlotConfig0E		SlotConfig0F	
0D	34	Counter0			
0E	38				
0F	3C	Counter1			
10	40				
11	44	LastKeyUse			
12	48				

Word	Byte	00	01	02	03
13	4C	LastKeyUse			
14	50				
15	54	UserExtra	Selector	LockValue	LockConfig
16	58	SlotLocked		Rfu90	
17	5C	X509Format[0:3]			
18	60	KeyConfig00		KeyConfig01	
19	64	KeyConfig02		KeyConfig03	
1A	68	KeyConfig04		KeyConfig05	
1B	6C	KeyConfig06		KeyConfig07	
1C	70	KeyConfig08		KeyConfig09	
1D	74	KeyConfig0A		KeyConfig0B	
1E	78	KeyConfig0C		KeyConfig0D	
1F	7C	KeyConfig0E		KeyConfig0F	

Note:

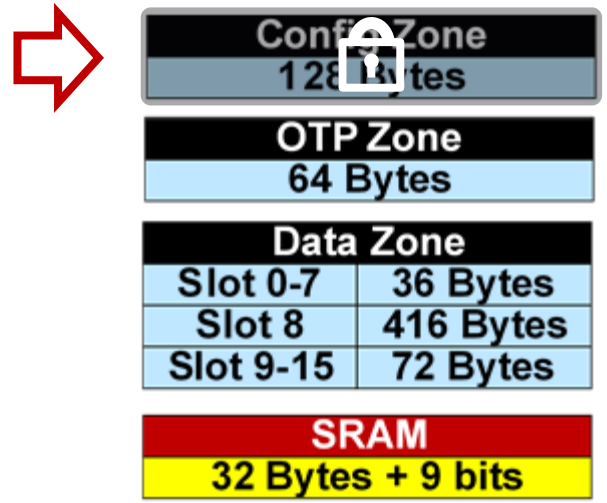
SN[0:1] always **0x01 0x23**

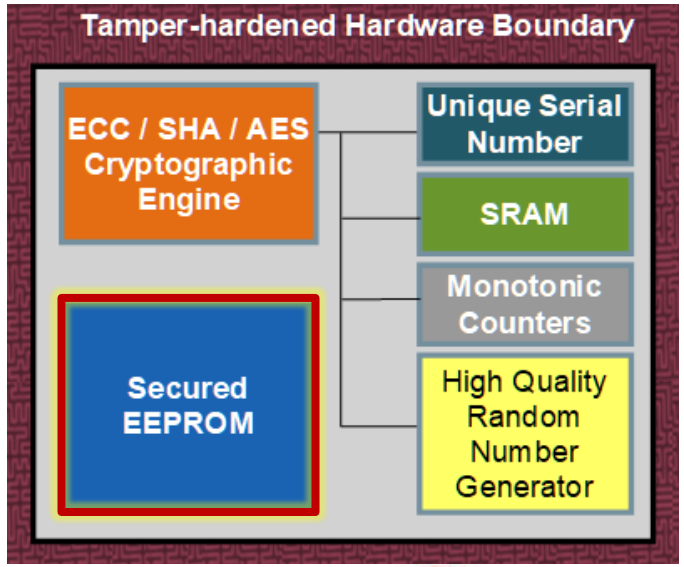
SN[8] always **0xEE**

Access of configure zone should apply it's WORD address. (one WORD equal four bytes)

Ex. SlotConfig00 Byte Address = 0x14

WORD address = 0x14 / 4 = **0x05**

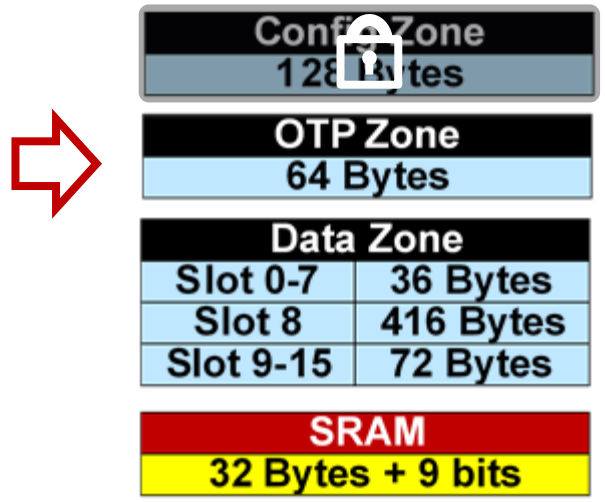


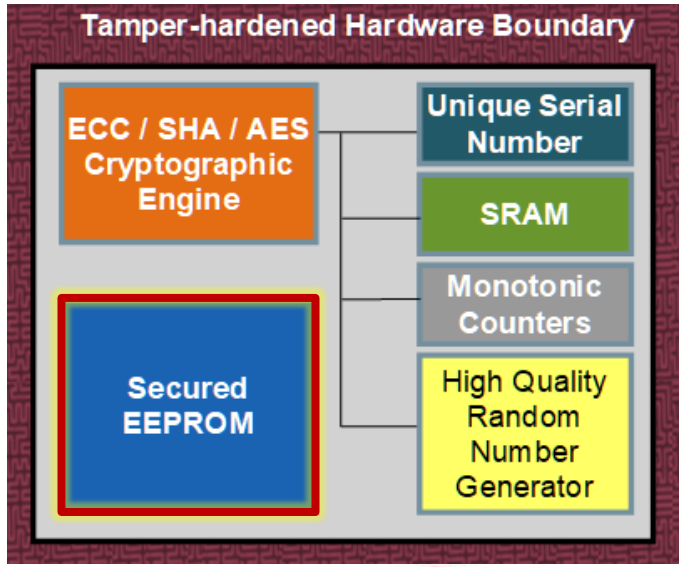


MICROCHIP

OTP Zone

	00	01	02	03
00	Total 64 Bytes (512 bits) (16 WORDs)			
04				
08				
0C				
.....				
30				
34				
38				
3C				





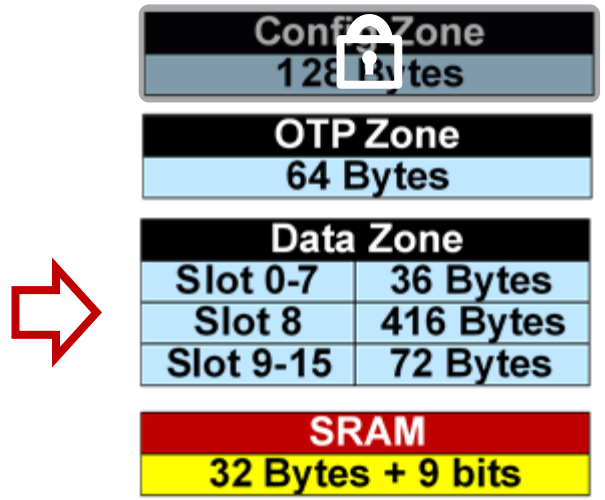
MICROCHIP

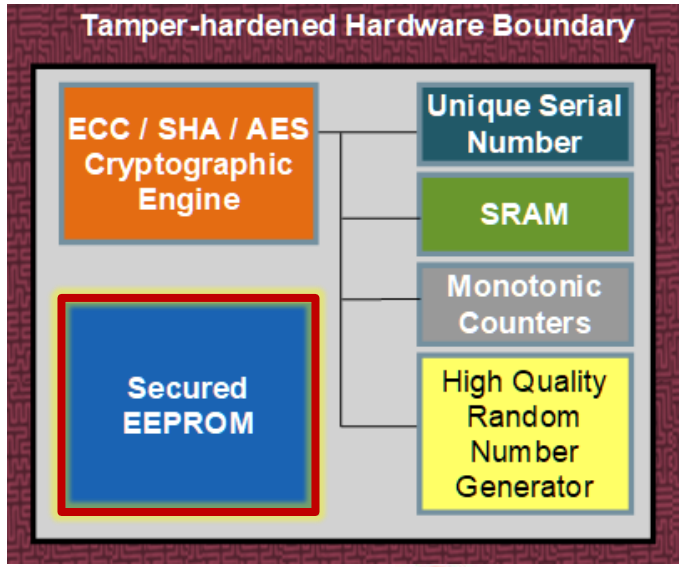
Data Zone

	Block0		Block1						
Slot	WORD Address	Size	WORD Address	Size	WORD Address	Size		Size	
0	0x0000	32	0x0100	4					36
1	0x0008	32	0x0108	4					36
2	0x0010	32	0x0110	4					36
3	0x0018	32	0x0118	4					36
4	0x0020	32	0x0120	4					36
5	0x0028	32	0x0128	4					36
6	0x0030	32	0x0130	4					36
7	0x0038	32	0x0138	4					36
8	0x0040	32	0x0140	32	0x0240	32	0x0340 ~ 0x0C40	32	416
9	0x0048	32	0x0148	32	0x0248	8			72
10	0x0050	32	0x0150	32	0x0250	8			72
11	0x0058	32	0x0158	32	0x0258	8			72
12	0x0060	32	0x0160	32	0x0260	8			72
13	0x0068	32	0x0168	32	0x0268	8			72
14	0x0070	32	0x0170	32	0x0270	8			72
15	0x0078	32	0x0178	32	0x0278	8			72

Word	Byte	00	01	02	03
13	4C	LastKeyUse			
14	50				
15	54	UserExtra	Selector	LockValue	LockConfig
16	58	SlotLocked		Rfu90	

Total 1208 Bytes





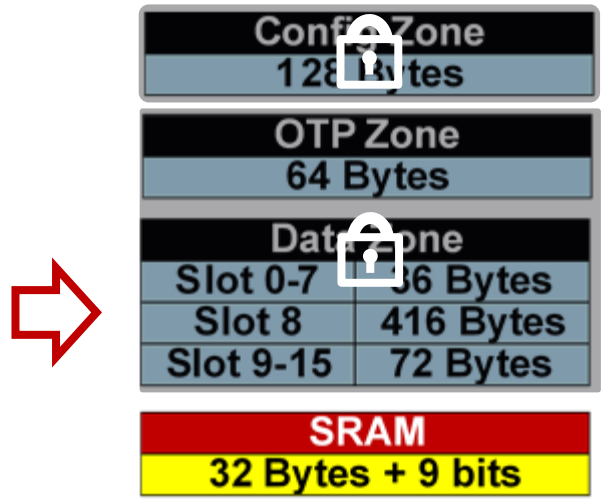
MICROCHIP

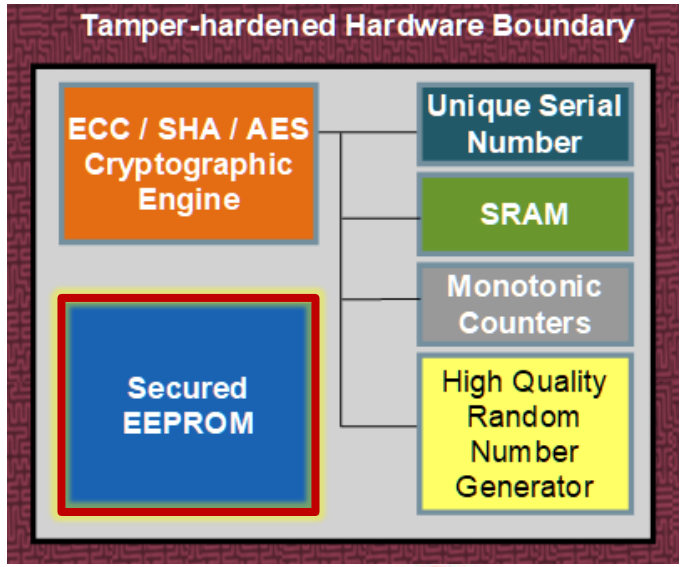
Data Zone

	Block0		Block1						
Slot	WORD Address	Size	WORD Address	Size	WORD Address	Size	WORD Address	Size	Size
0	0x0000	32	0x0100	4					36
1	0x0008	32	0x0108	4					36
2	0x0010	32	0x0110	4					36
3	0x0018	32	0x0118	4					36
4	0x0020	32	0x0120	4					36
5	0x0028	32	0x0128	4					36
6	0x0030	32	0x0130	4					36
7	0x0038	32	0x0138	4					36
8	0x0040	32	0x0140	32	0x0240	32	0x0340 ~ 0x0C40	32	416
9	0x0048	32	0x0148	32	0x0248	8			72
10	0x0050	32	0x0150	32	0x0250	8			72
11	0x0058	32	0x0158	32	0x0258	8			72
12	0x0060	32	0x0160	32	0x0260	8			72
13	0x0068	32	0x0168	32	0x0268	8			72
14	0x0070	32	0x0170	32	0x0270	8			72
15	0x0078	32	0x0178	32	0x0278	8			72

Word	Byte	00	01	02	03
13	4C	LastKeyUse			
14	50				
15	54	UserExtra	Selector	LockValue	LockConfig
16	58	SlotLocked		Rfu90	

Total 1208 Bytes

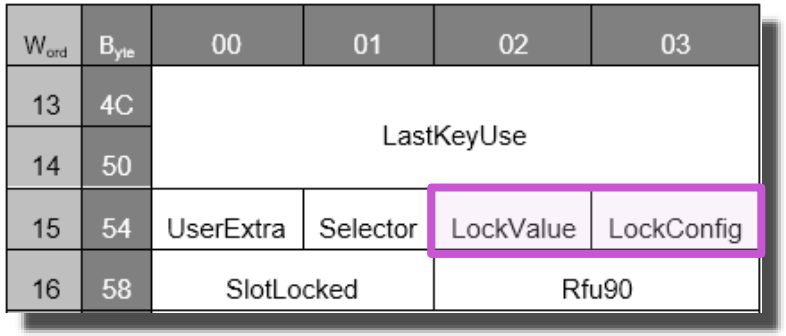




MICROCHIP

EEPROM Locking

Lock is OTP,
Many secure Functions activated after lock.

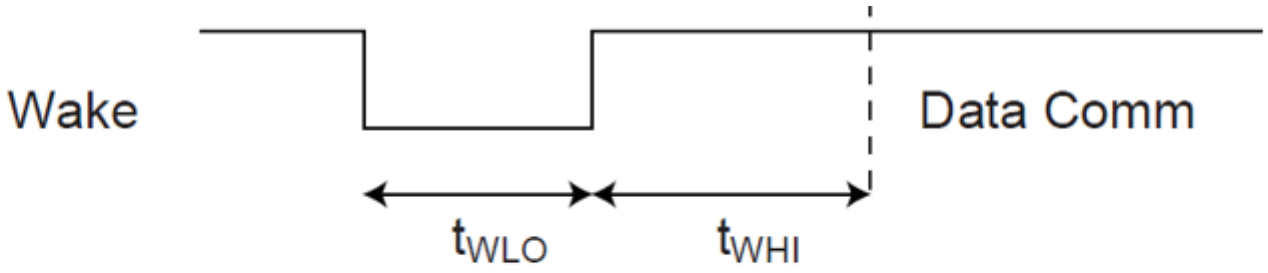


Word	Byte	00	01	02	03
13	4C	LastKeyUse			
14	50				
15	54	UserExtra	Selector	LockValue	LockConfig
16	58	SlotLocked		Rfu90	

		Read	Write 4 Bytes	Write 32 Bytes	Lock Config	Lock Data
Configuration		Y			Unlock	
Data/OTP		N				
Configuration		Y	N		Lock	Unlock
Data/OTP		N	N	Y		
Configuration		Y	N		Lock Note: Byte == 0x55 (unlocked) Byes != 0x55 (locked)	
Data		Config Depended				
OTP	Read-Only	Y	N			
	Consumption	Y	Y One way, bit 1 → 0			
	Legacy	N, Word 0-1 Y, Word 2-15	N	N		

MICROCHIP

Wake-up Token



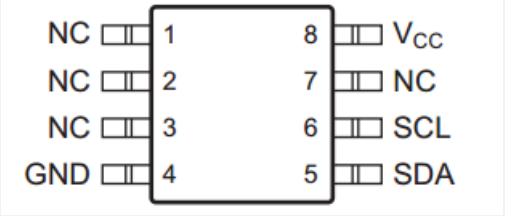
Symbol	Minimum duration	Description
t_{WLO}	60 µs	
t_{WHI}	1500 µs	SDA should be stable high for this entire duration.

Host will receive below Wake-up acknowledge from Crypto Chip.

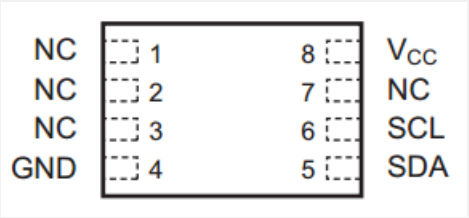
0x04 0x11 0x33 0x43

Package & Pinouts

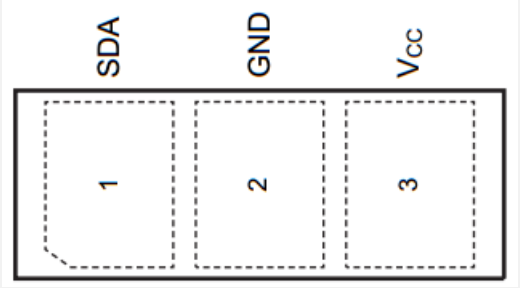
- **ECC508A**
  - 8-lead SOIC



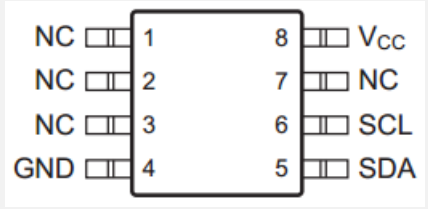
  - 8-pad UDFN



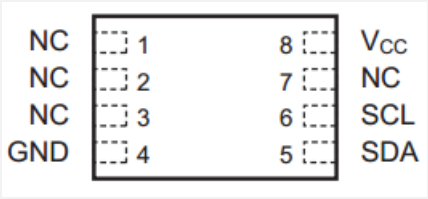
  - 3-lead Contact



- **ECC608A**
  - 8-lead SOIC

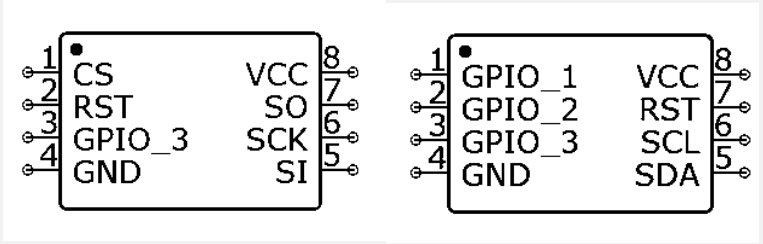


  - 8-pad UDFN

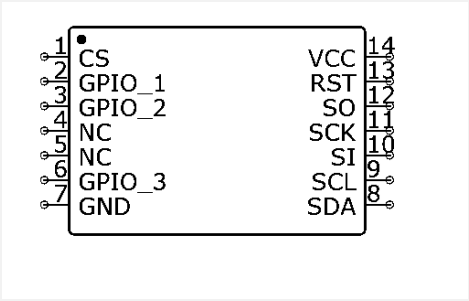


* Die-on-Tape and Reel for Qualified Customers

- **TA100**
  - 8-lead SOIC



  - 14-lead SOIC



Tools

- **ECC508A**
  - CryptoAuth-XPRO-B
  - AT88CKECCROOT
  - Crypto Authentication SOIC XPRO Starter Kit
  - 88CKECC-AWS-XSTK-B
  - Crypto Kit UDFN/SOIC Socketed XPRO Development Board
  - AT88CK590



...

- **ECC608A**
  - CryptoAuth-XPRO-B
  - AT88CKECCROOT
  - Crypto Authentication SOIC XPRO Starter Kit
  - 88CKECC-AWS-XSTK-B
  - Crypto Kit UDFN/SOIC Socketed XPRO Development Board
  - AVR-IOT WG Evaluation Board

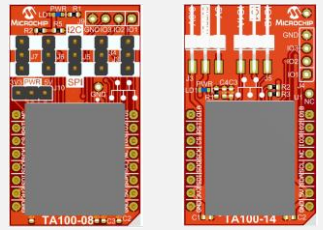


...

- **TA100**
  - CryptoAuto IVN TA/BSD Development Kit

  - DM320112 Dev Board

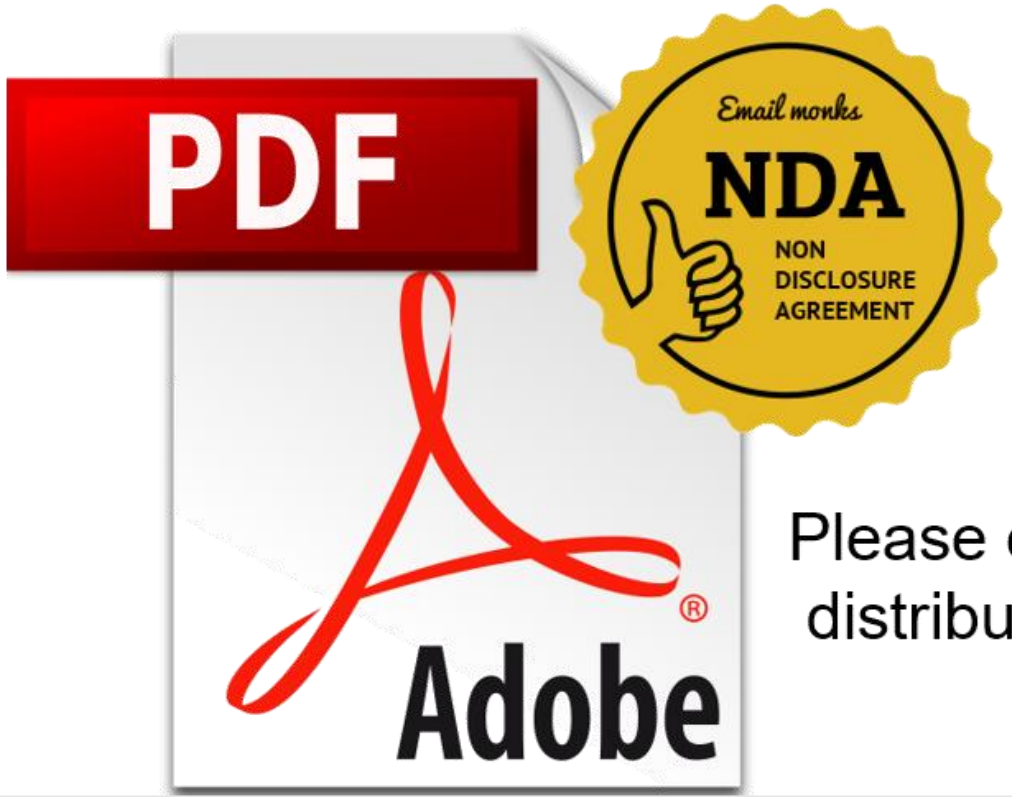
  - AC164166&AC164167 Socket Board



Full Version Datasheet ?

ECCx08 only provide summary Version datasheet

Documentation				
★ ATECC508A CryptoAuthentication Device Summary Data Sheet	Data Sheets	12/19/2017	332KB	
Application Notes				Download All
★ AN_42686 - AT15736: Smart Plug Getting Started Guide	AppNote	12/10/2016	1008KB	
★ AN_42688 - AT15735: Smart Plug Firmware User Guide	AppNote	12/10/2016	1126KB	

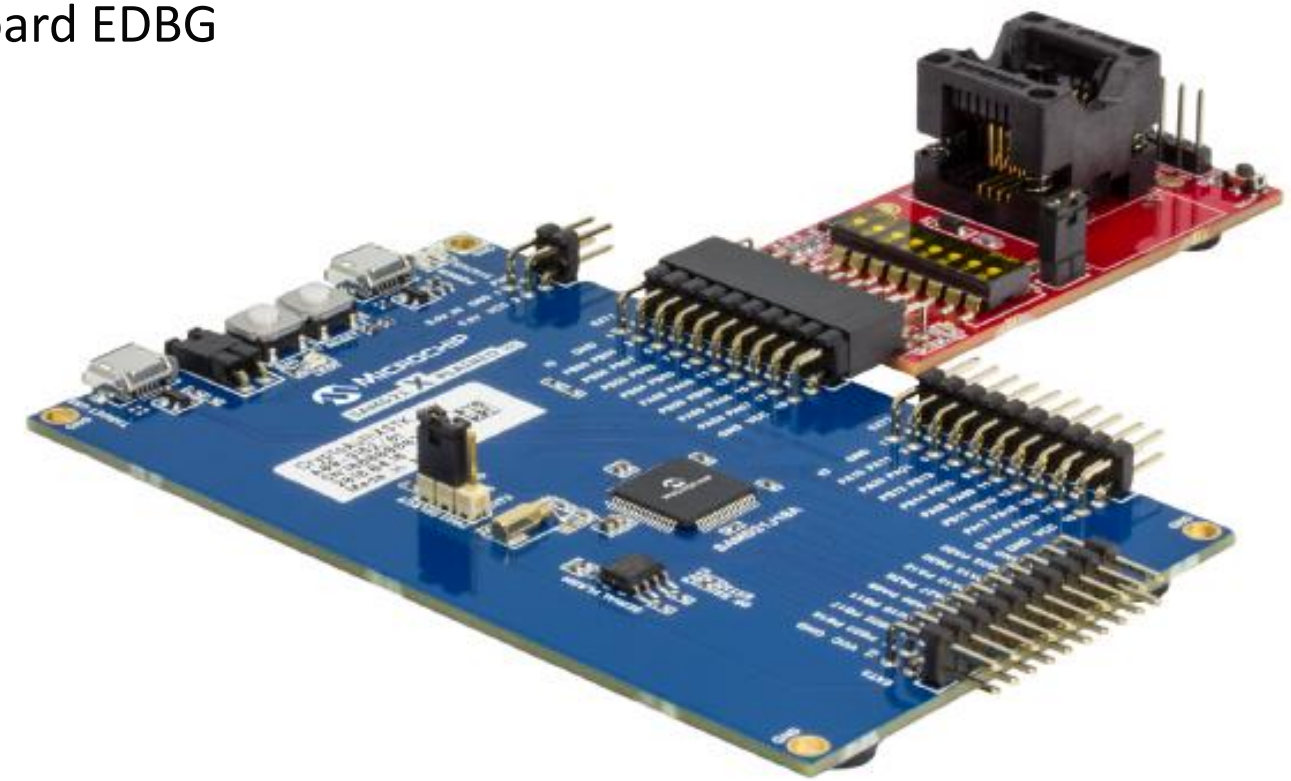


Please contact your Microchip sales or distributor window for NDA process.

EVB introduction

Hardware Introduction

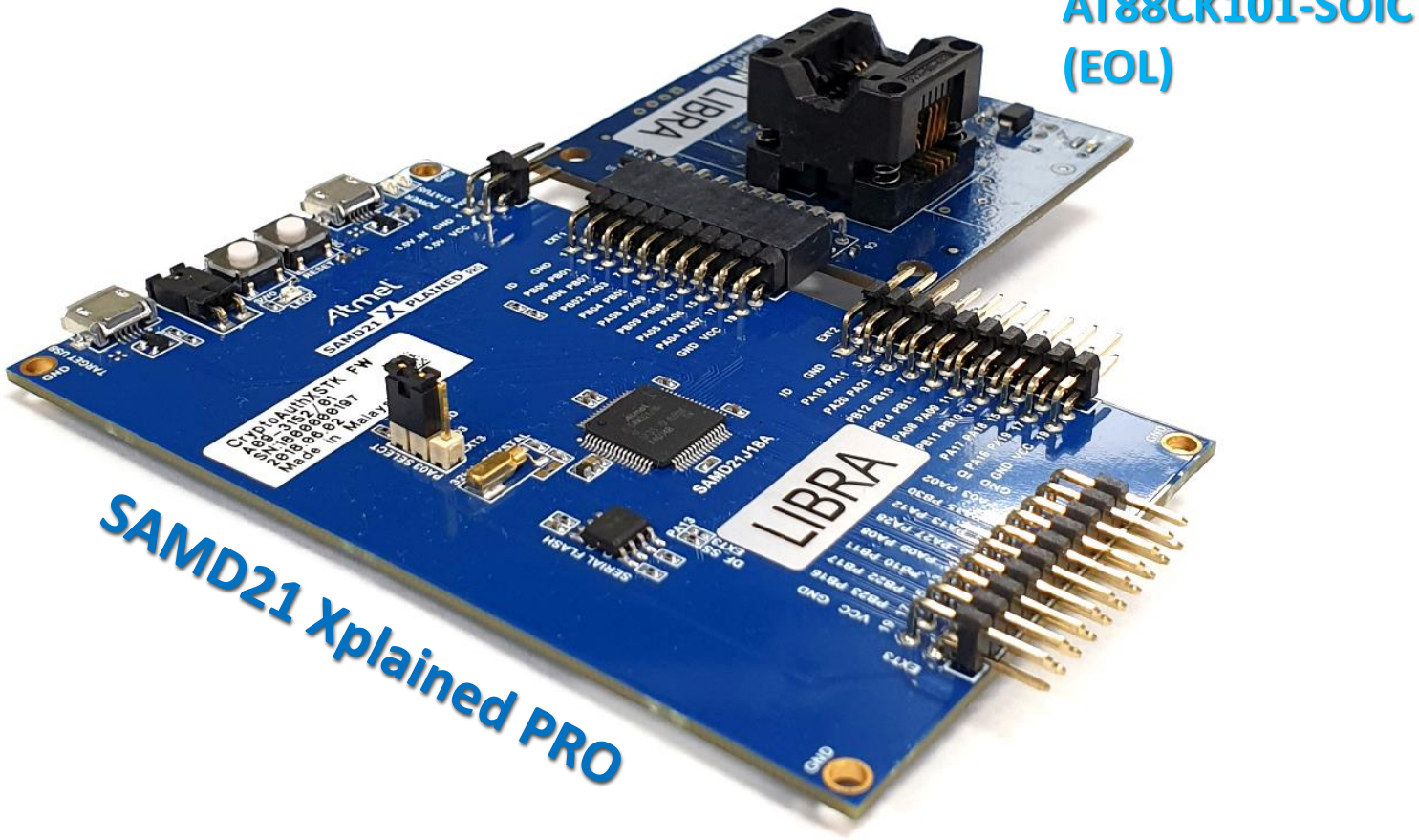
- **CryptoAuthentication™ SOIC XPRO Starter Kit (DM320109)**
  https://www.microchip.com/DevelopmentTools/ProductDetails/DM320109
  - MCU : SAMD21J18A (Cortex-M0+ 48MHz, 256KB Flash, 32KB SRAM)
  - Socket : AT88CKSCKTSOIC-XPRO
  - Debugger : Onboard EDBG



MICROCHIP

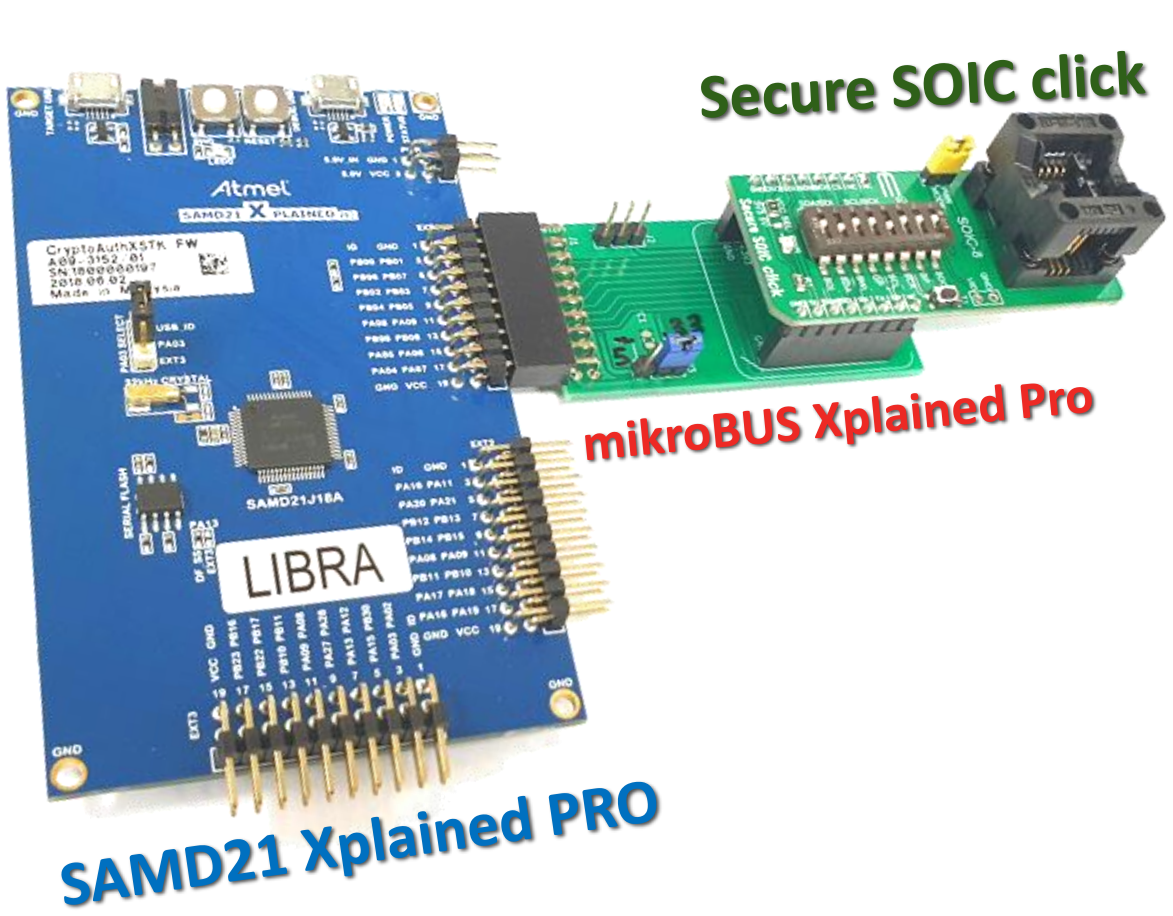
Hardware Alternate

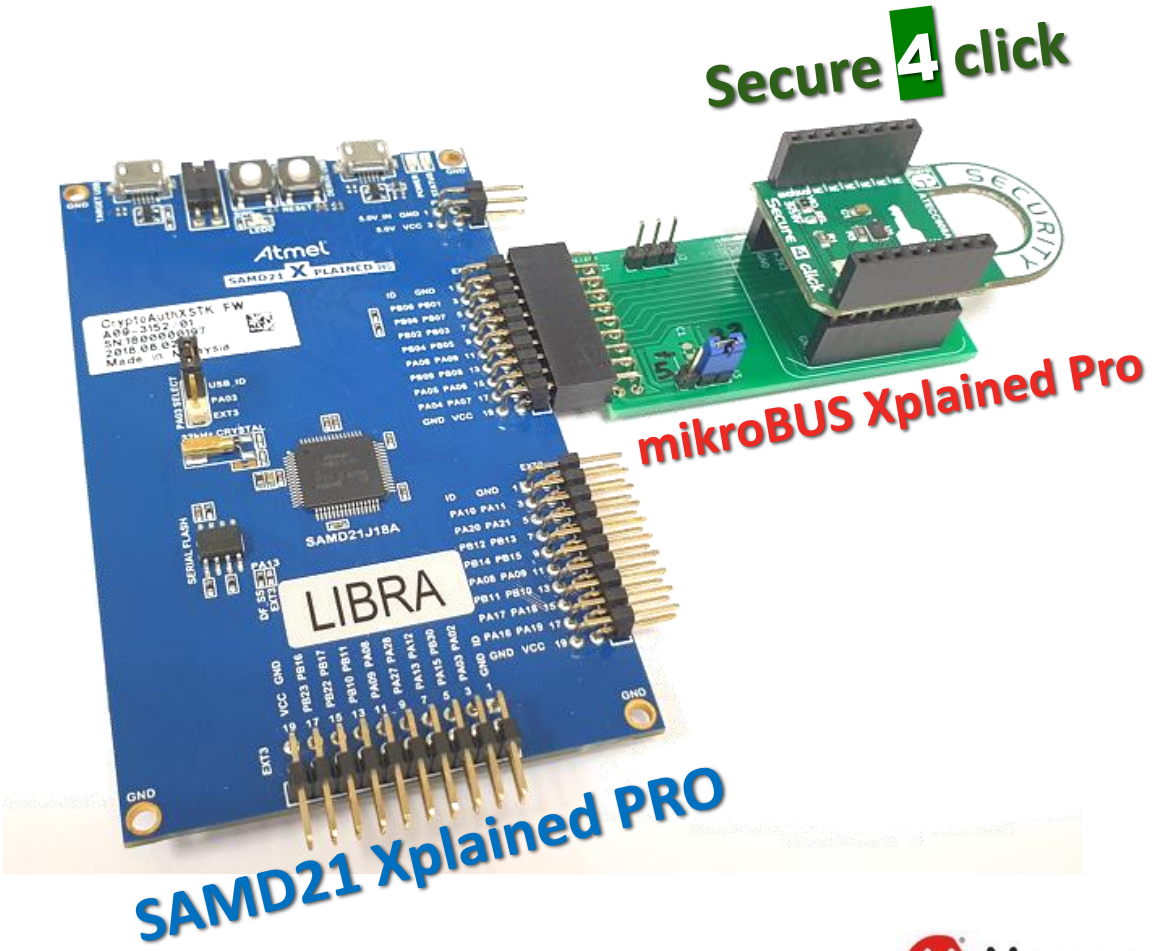
- SAMD21 Xplained PRO + AT88CK101-SOIC socket



Hardware Alternate

- **SAMD21 Xplained PRO + mikroBUS Click boards with mikroBUS Xplained Pro convert board**





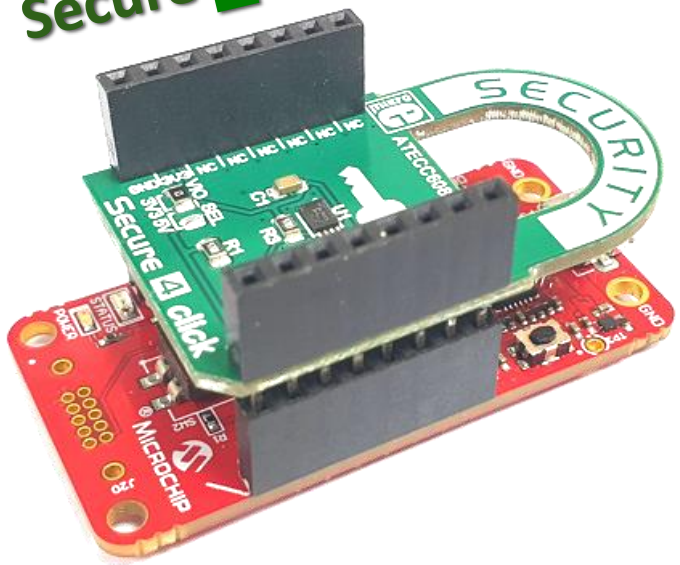
Hardware Alternate

- CryptoAuth Trust (DM320118) + mikroBUS Click boards

Secure SOIC click

CryptoAuth Trust

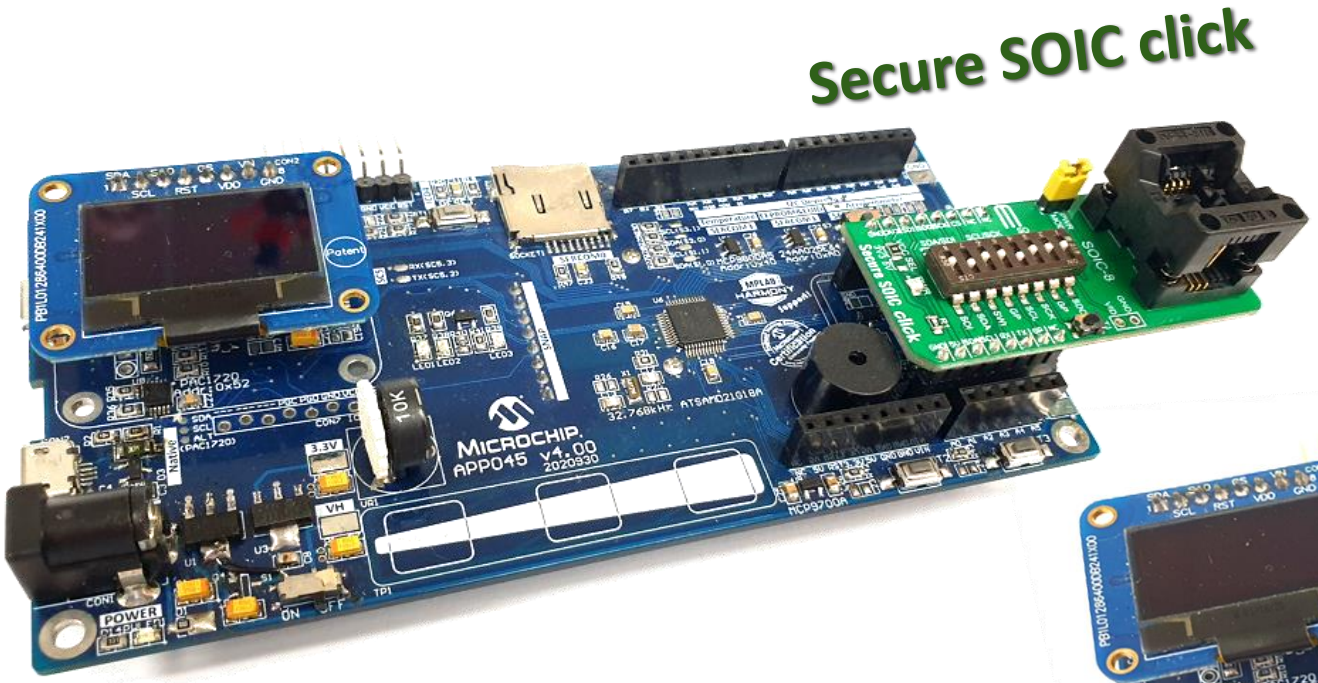
Secure 4 click

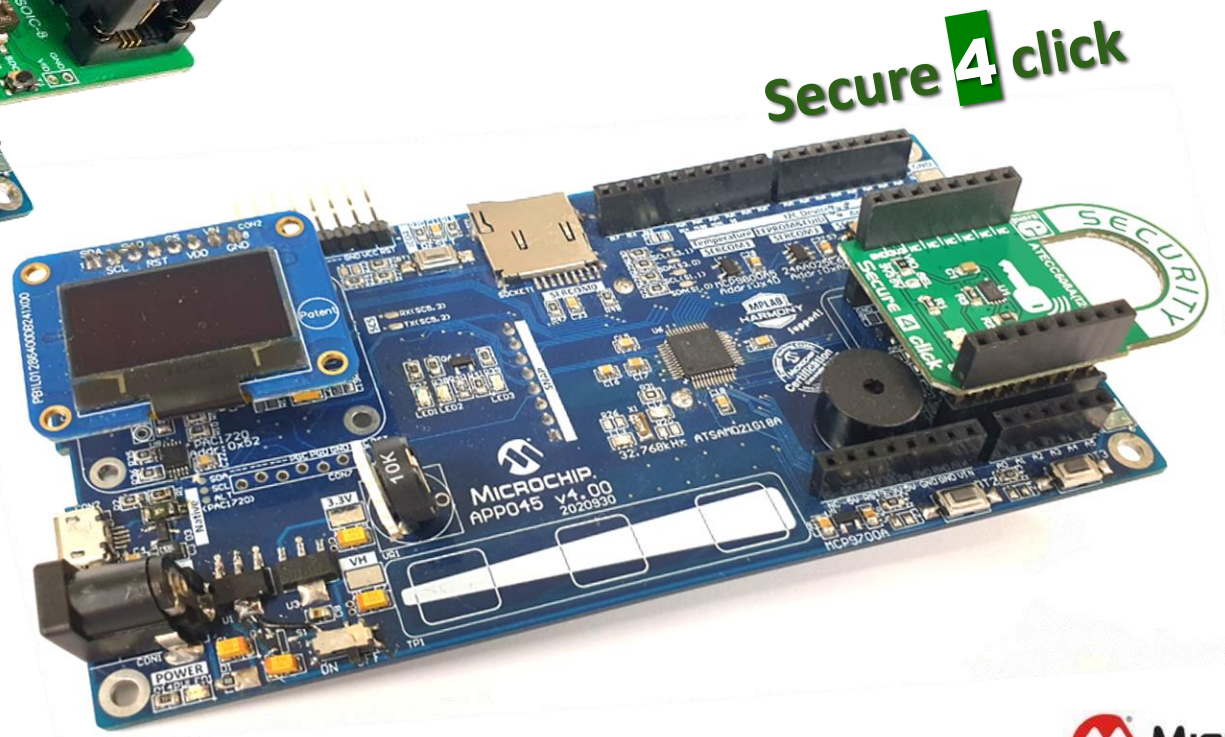


MICROCHIP

Hardware Alternate

- **APP045v4** + **mikroBUS Click boards**





Harmony Configuration

Step by step to finish Harmony Configuration

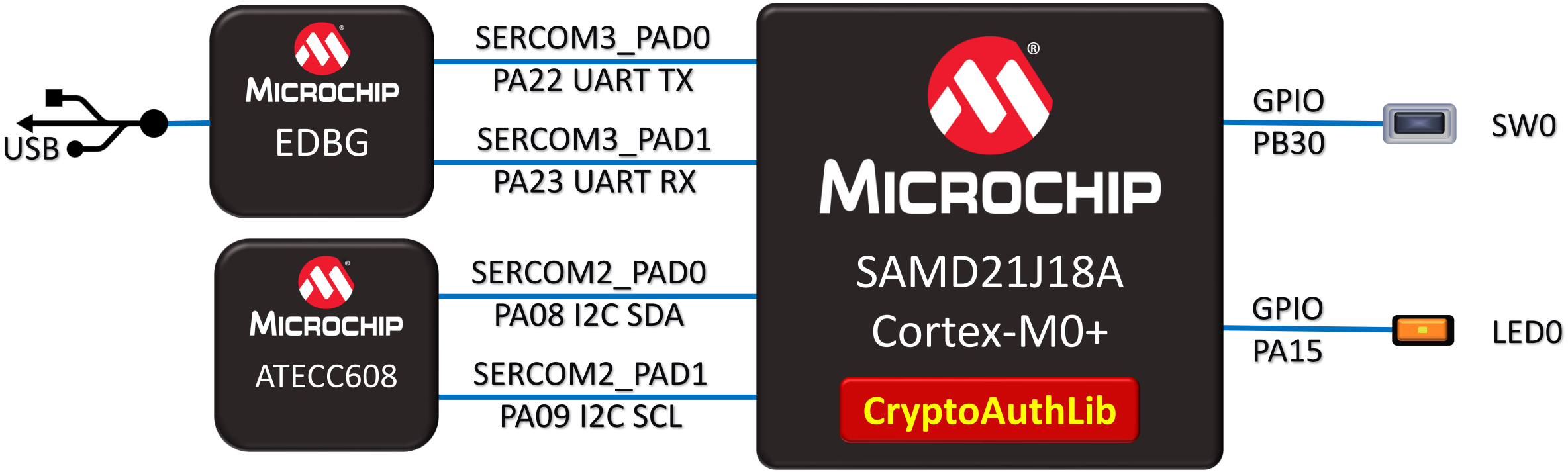
Software Request In this Course

- **MPLAB® X IDE 6.00** https://www.microchip.com/mplab/mplab-x-ide
- **MPLAB® XC32 v4.10** https://www.microchip.com/mplab/compilers
- **MPLAB® Harmony 3 Launcher v3.6.4 (From X IDE plugin)**
- **MPLAB® Harmony Framework (From X IDE Harmony 3 Content Manager)**
  - bsp v3.13.0
  - core v3.11.0
  - csp v3.13.0
  - mhc v3.8.5
  - dev_packs v3.13.0
  - **CryptoAuthLib v3.3.3**
- **SAMD21 DFP v3.6.144 (Device Firmware Pack) (From X IDE \Tools\Packs)**
- **CMSIS v5.8.0 (Common Microcontroller Software Interface Standard)**

(Lasted Update : 2022/07/06)

MICROCHIP

Block Diagram

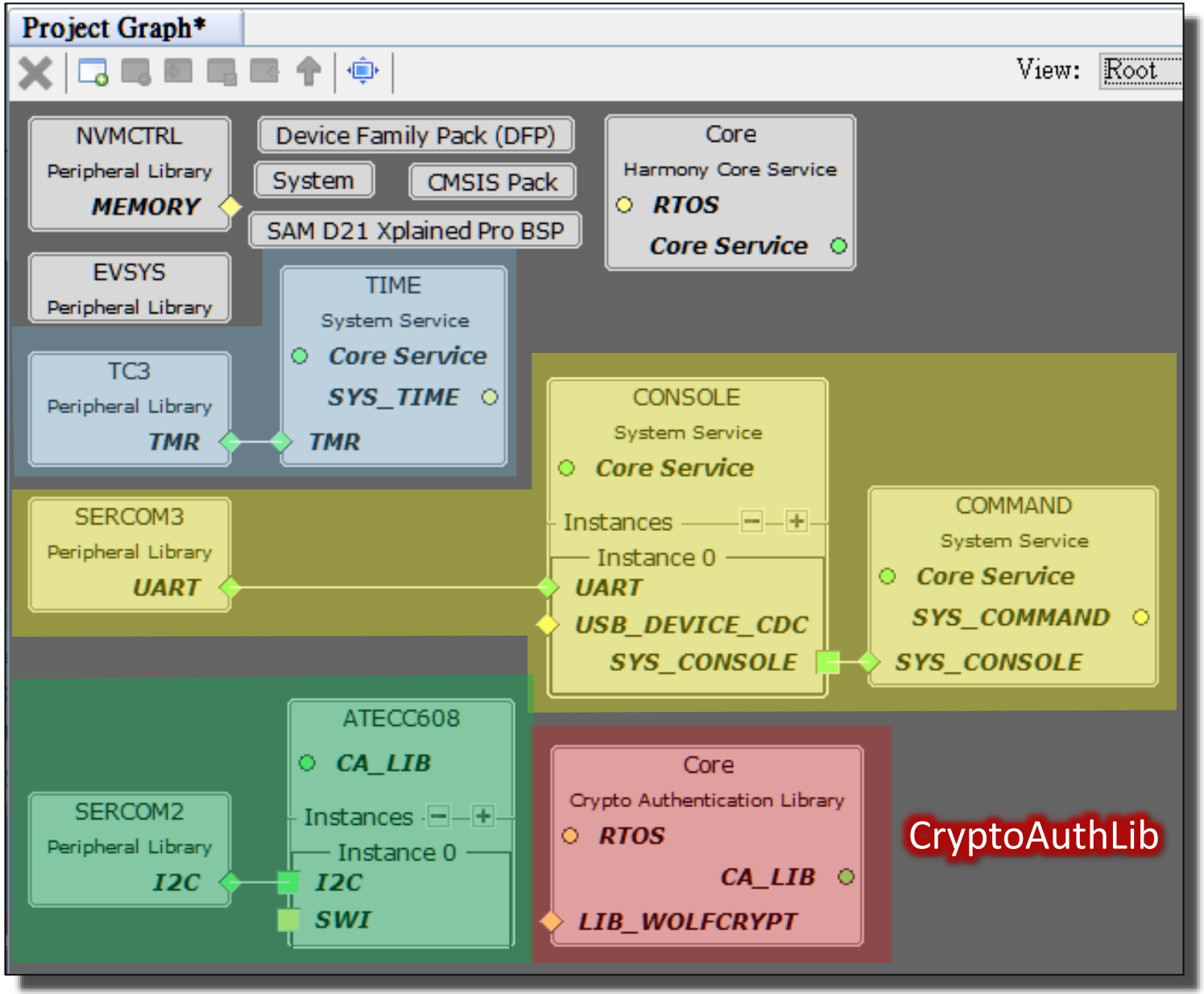


Harmony Block Diagram

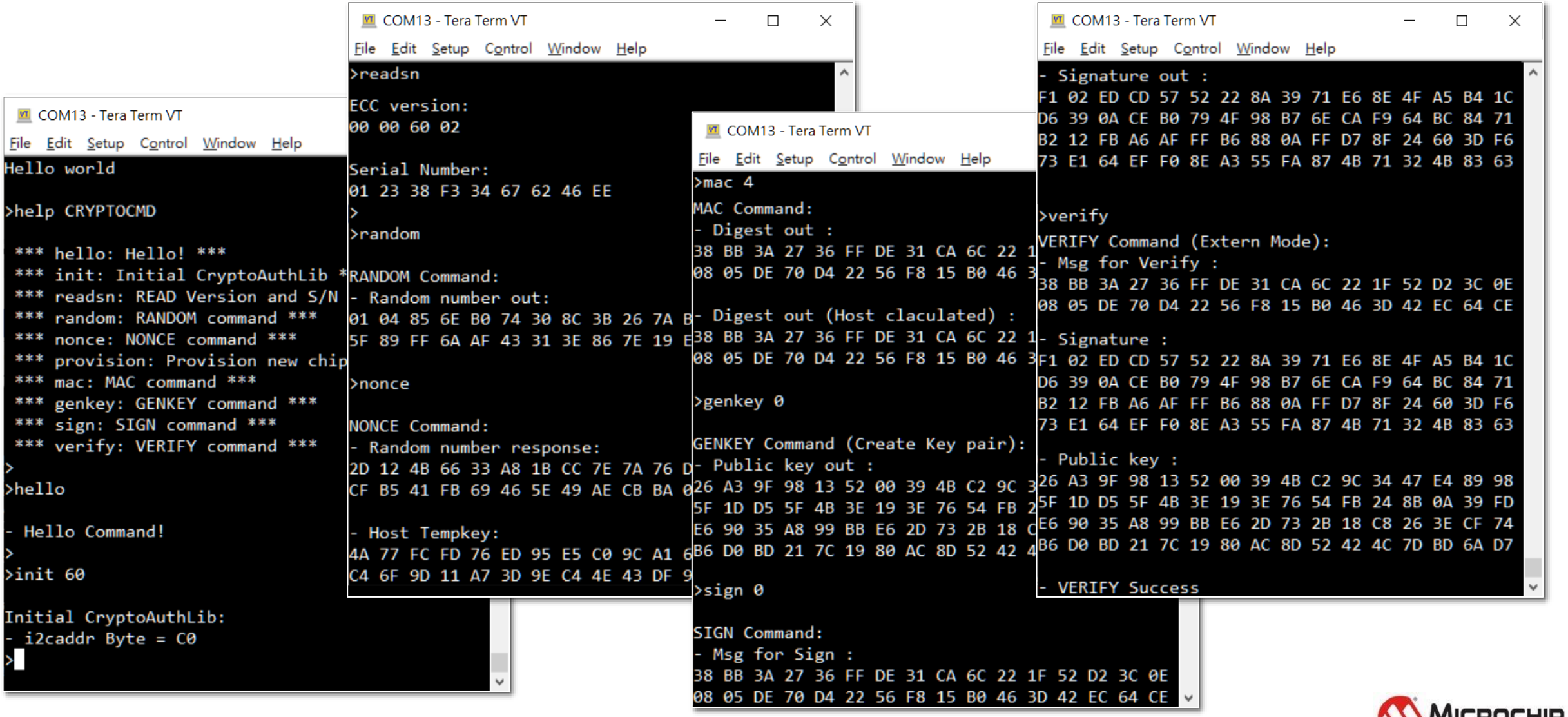
Timer/Delay

UART
Console/Command

I2C
ATECC608

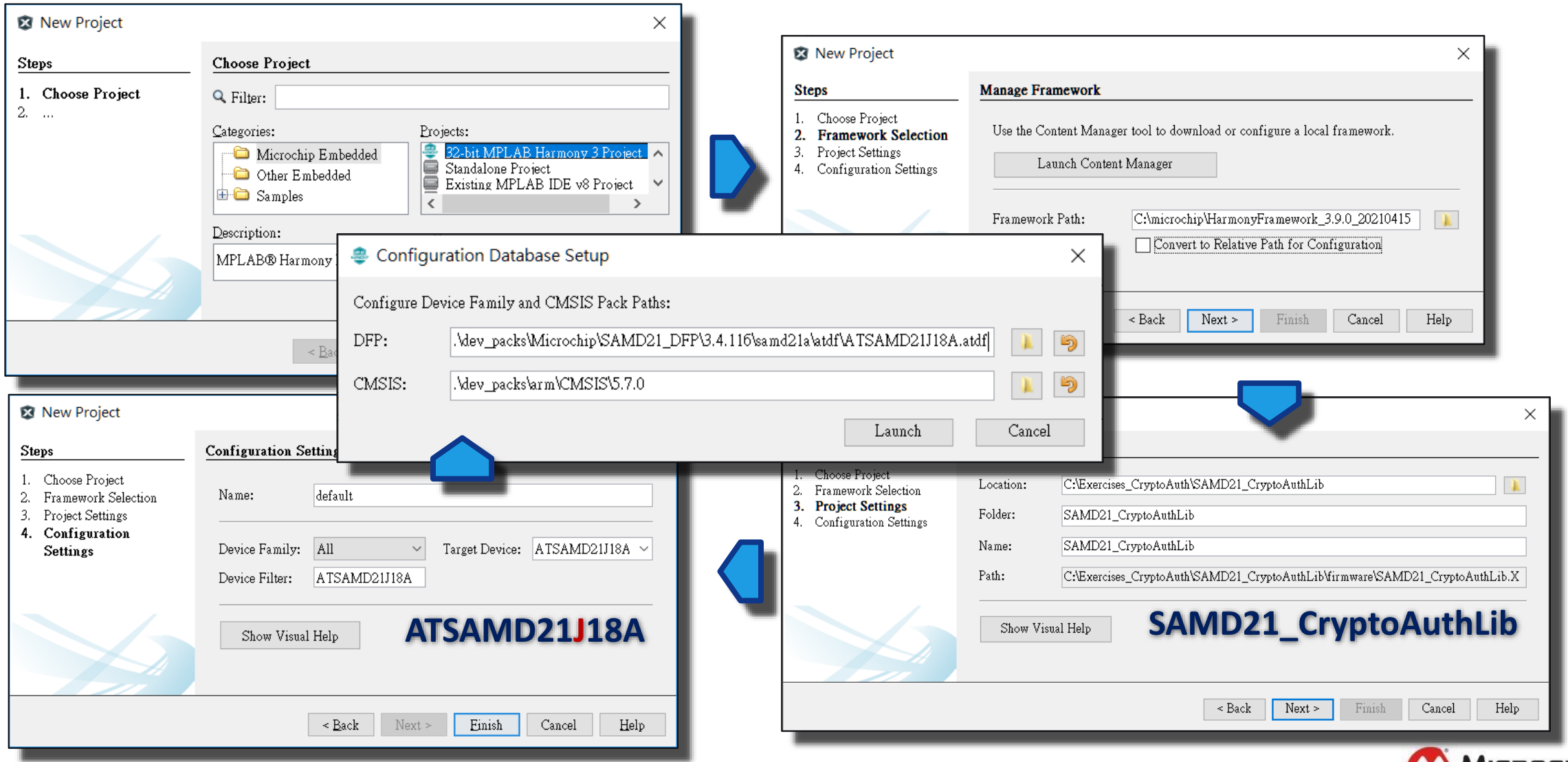


UART Command Console result after training



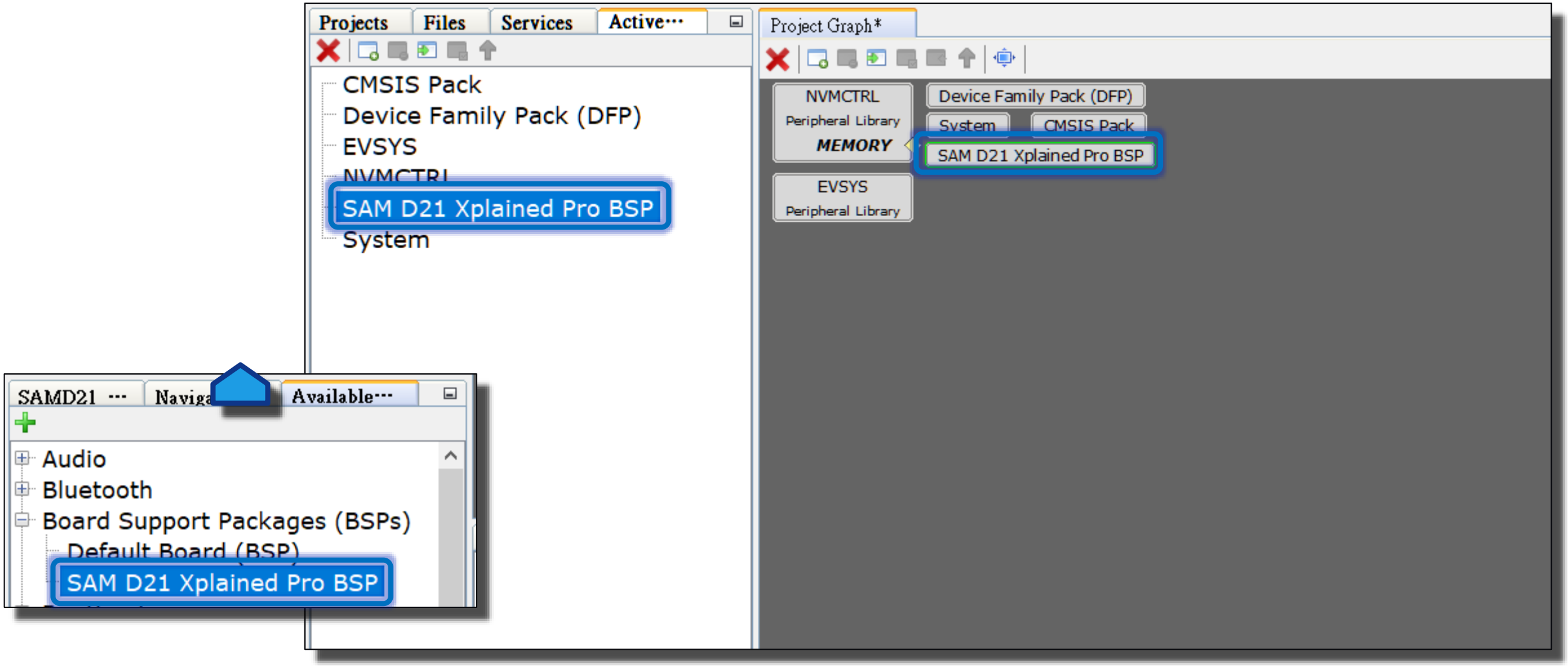
Harmony Configuration

New Project (ATSAMD21J18A)



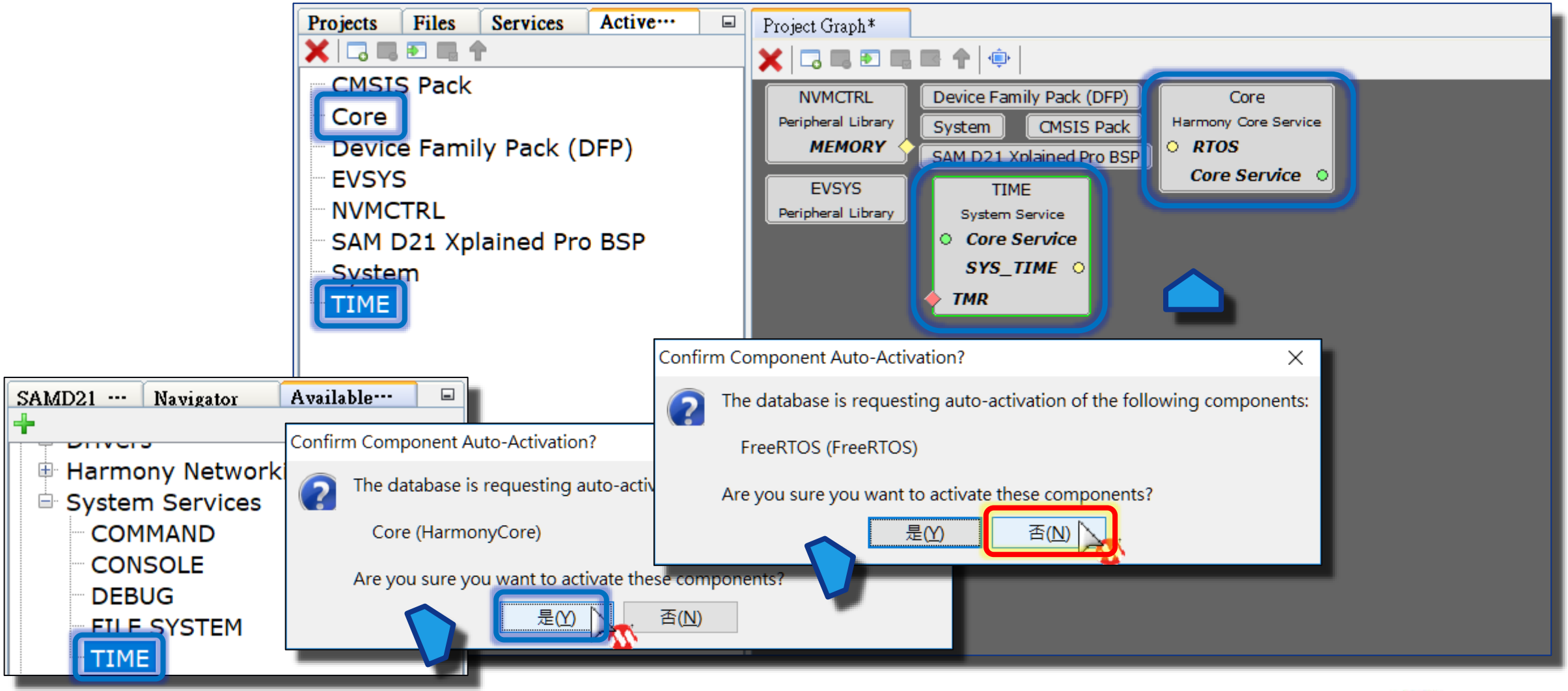
Harmony Configuration

Add SAMD21 Xplained Pro BSP (Board Support Package)



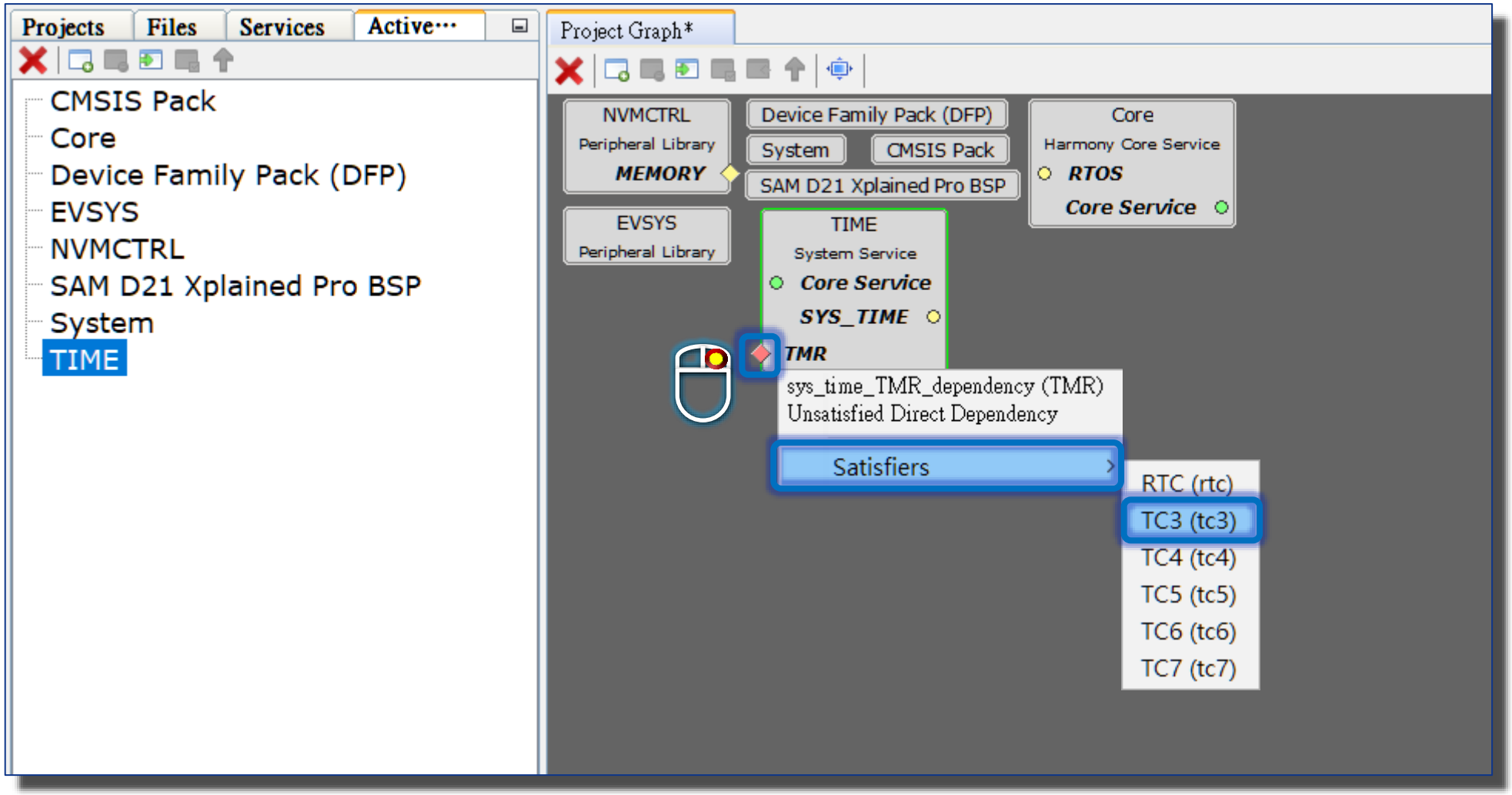
Harmony Configuration

Add TIME System Service and Harmony Core Service



Harmony Configuration

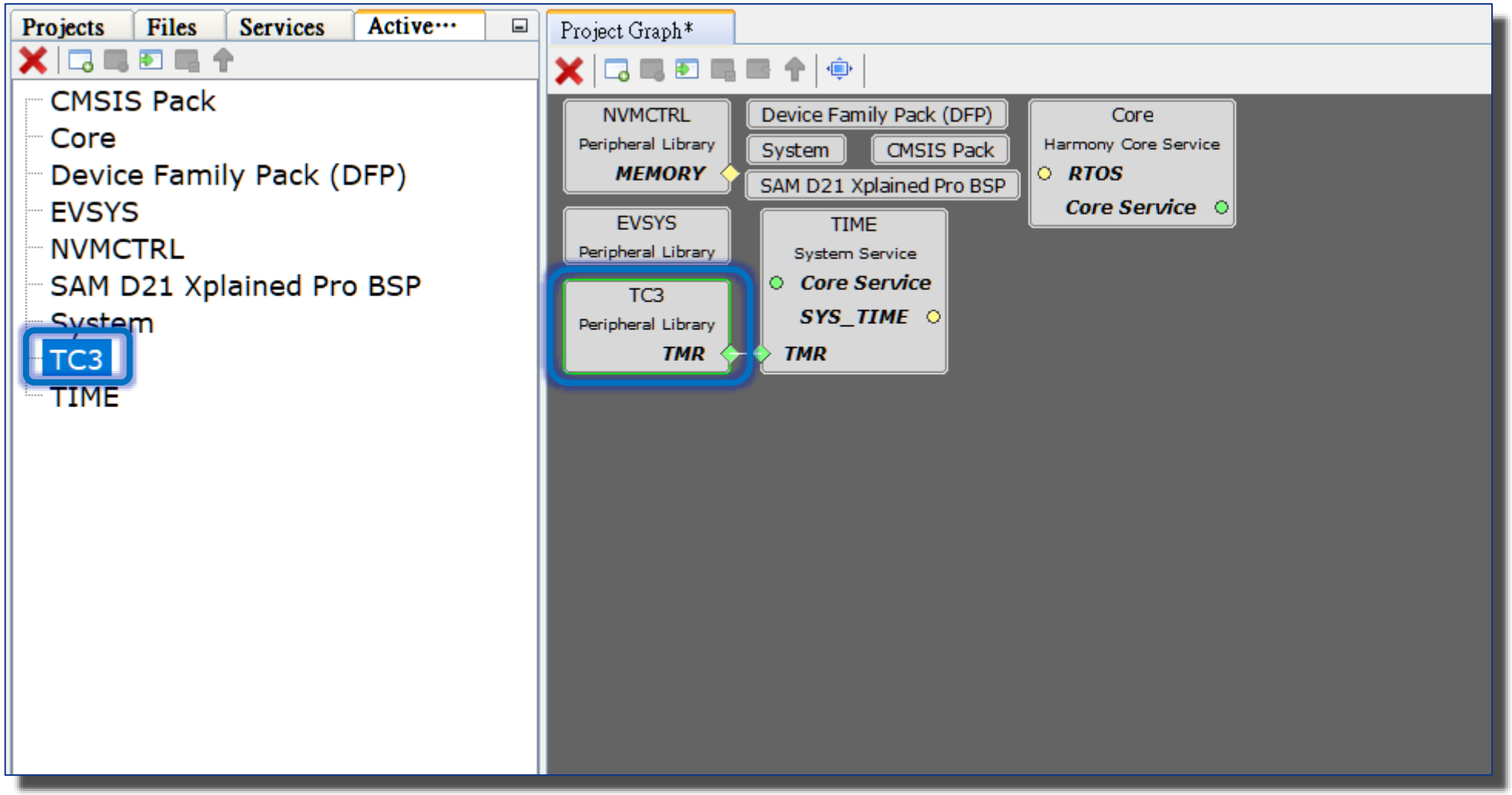
Add TC3 Peripheral Library for TIME System Service



MICROCHIP

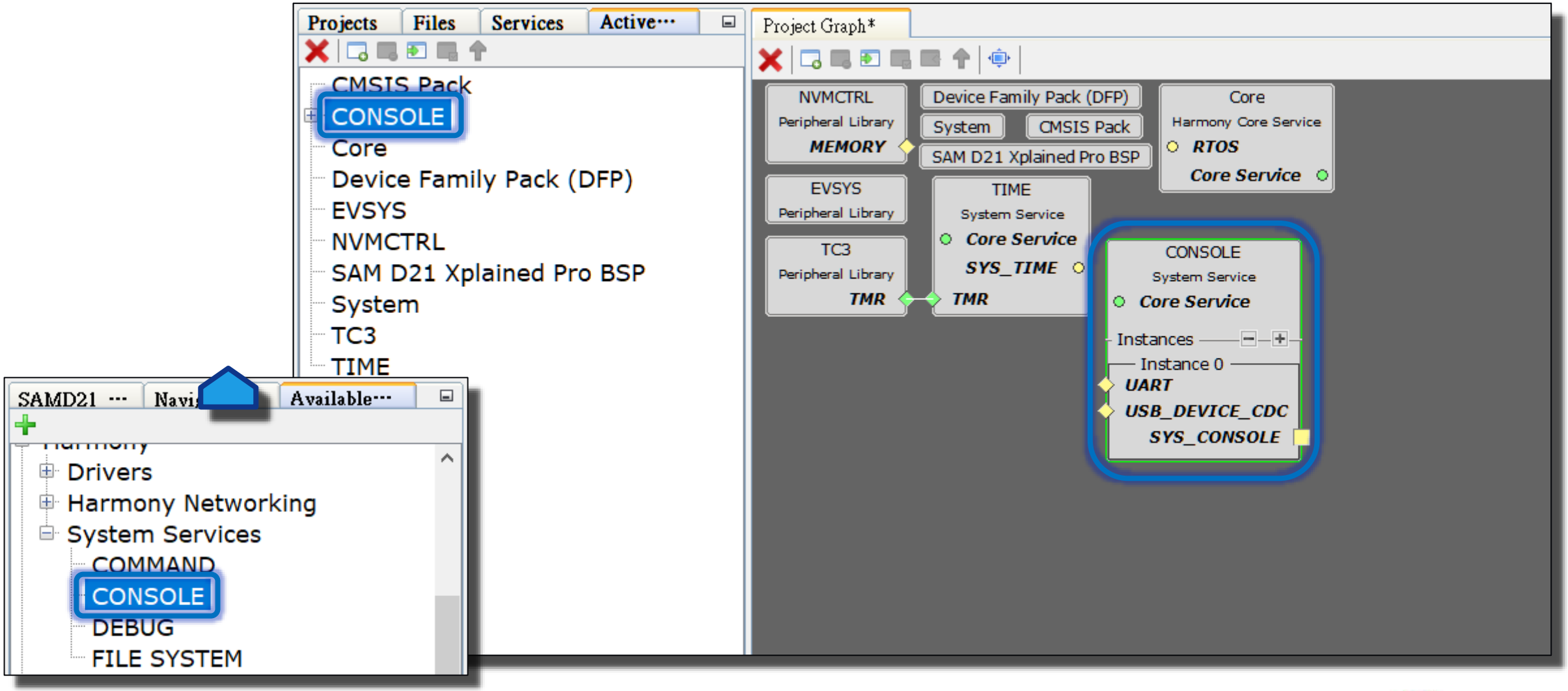
Harmony Configuration

Add TC3 Peripheral Library for TIME System Service



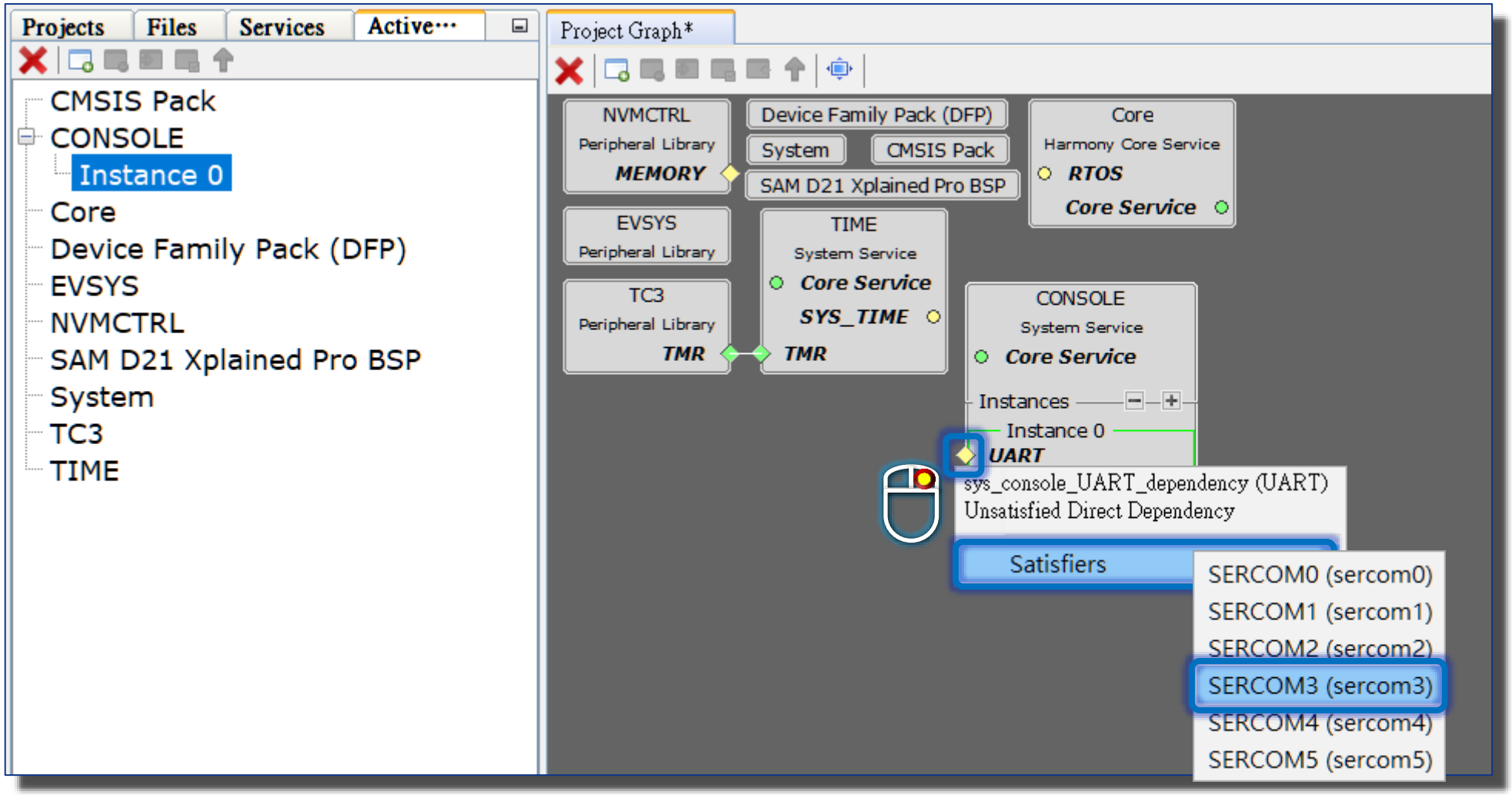
Harmony Configuration

Add CONSOLE System Service



Harmony Configuration

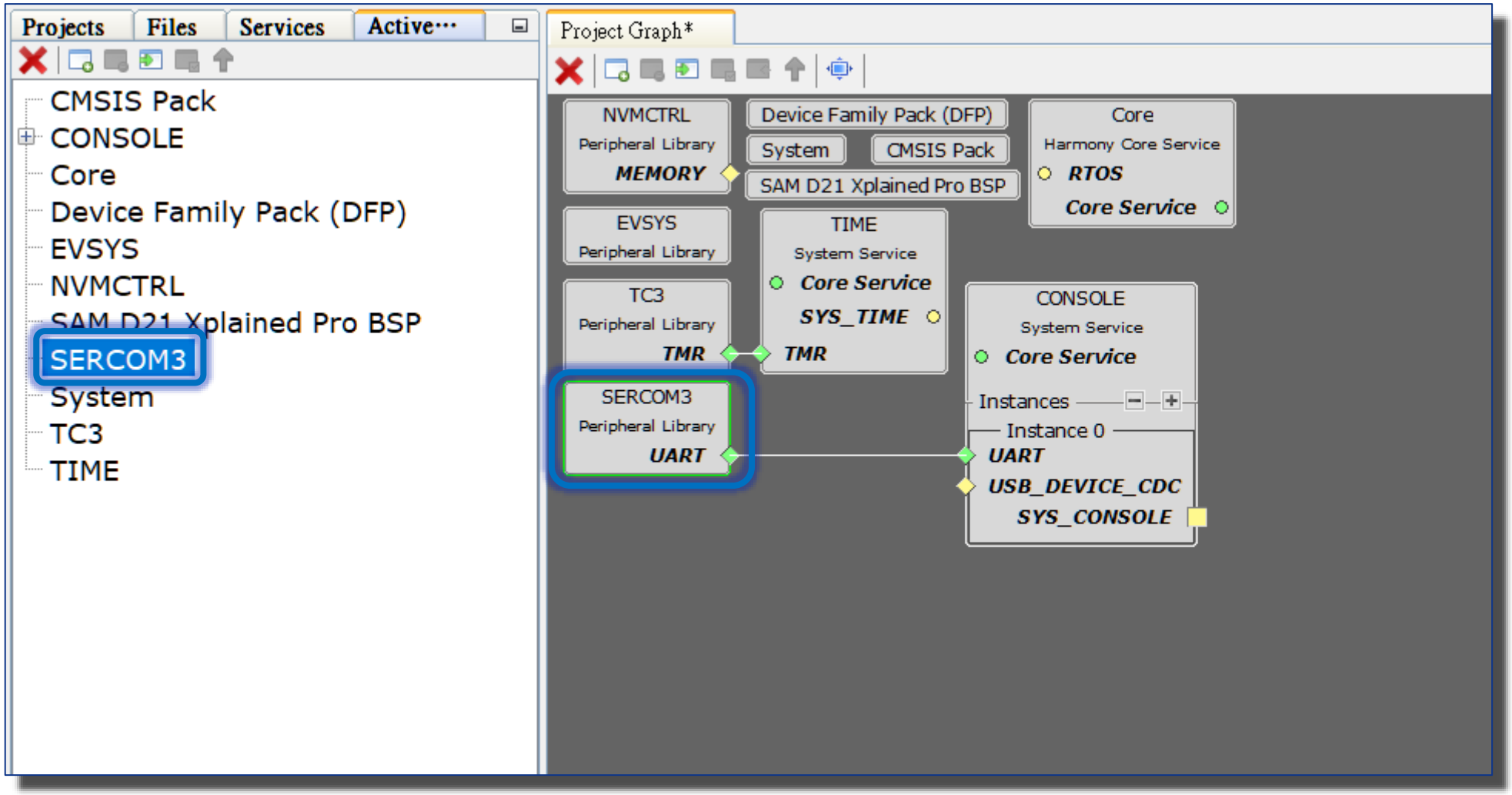
Add SERCOM3 Peripheral Library for CONSOLE System Service



MICROCHIP

Harmony Configuration

Add SERCOM3 Peripheral Library for CONSOLE System Service



Harmony Configuration

Configure Tx Ring Buffer Size to 1024 and Rx pin to SERCOM PAD[1]

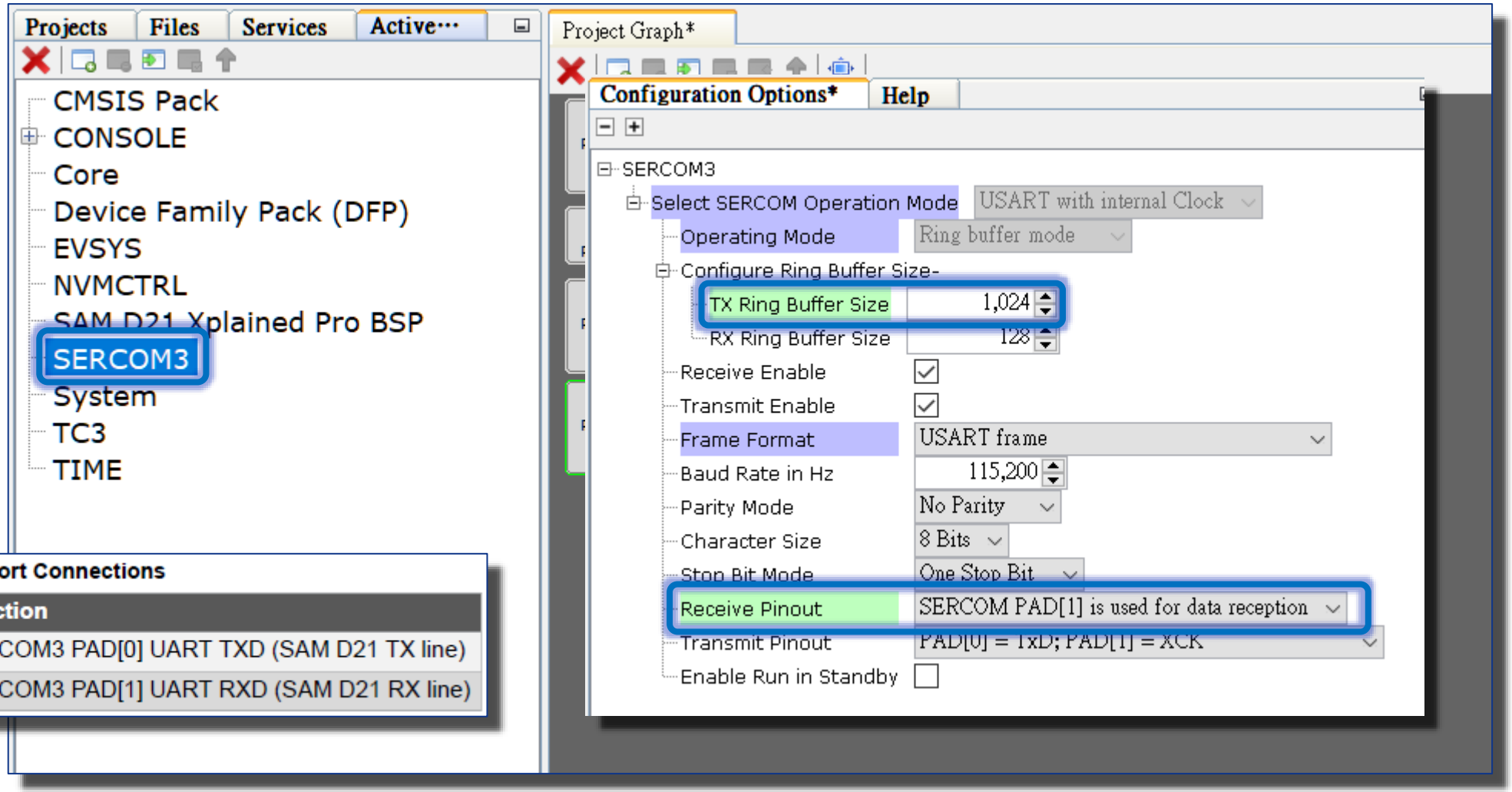
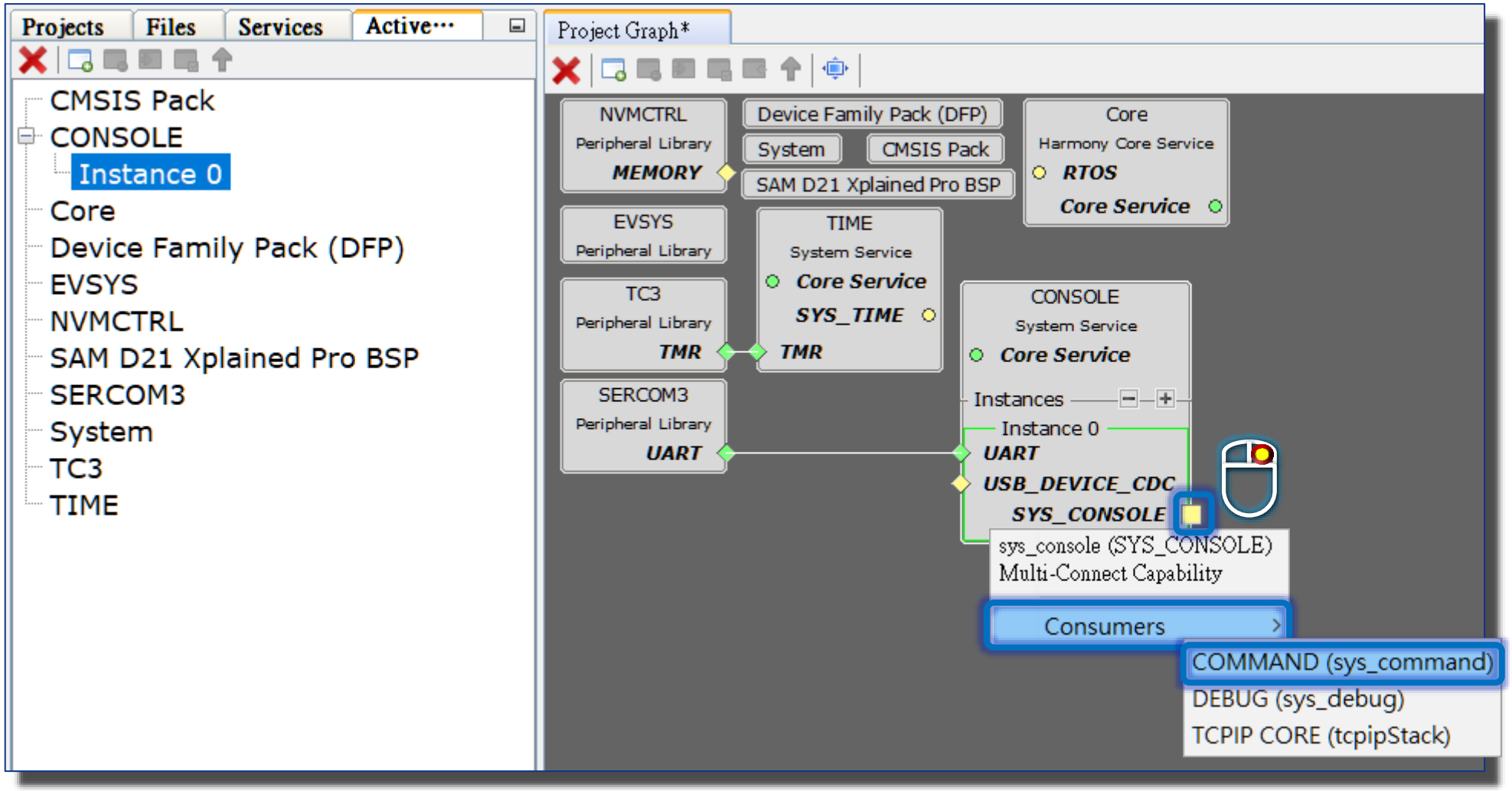


Table 4-10. Virtual COM Port Connections

Pin on SAM D21	Function
PA22	SERCOM3 PAD[0] UART TXD (SAM D21 TX line)
PA23	SERCOM3 PAD[1] UART RXD (SAM D21 RX line)

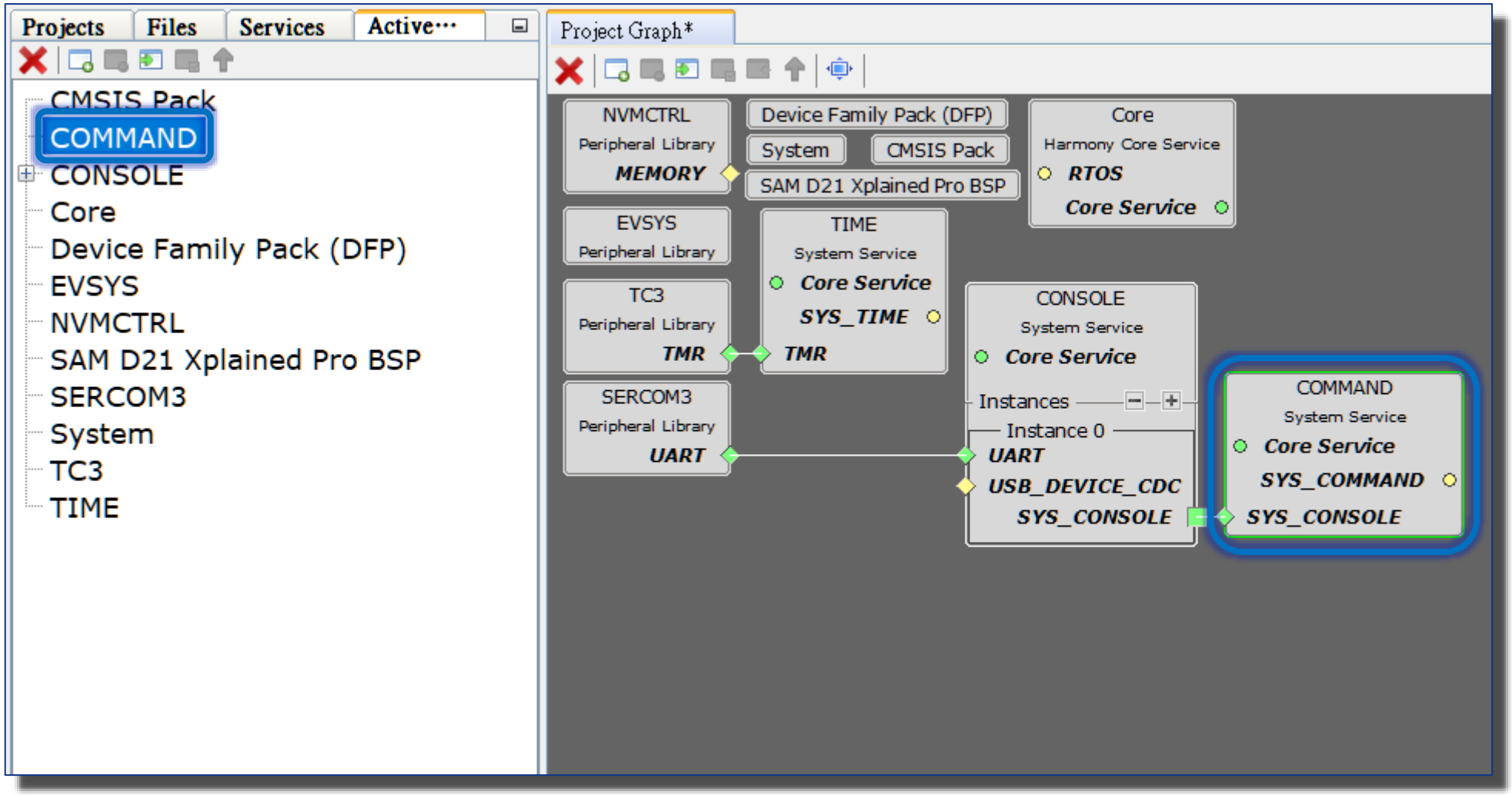
Harmony Configuration

Add COMMAND as Consumer of CONSOLE System Service



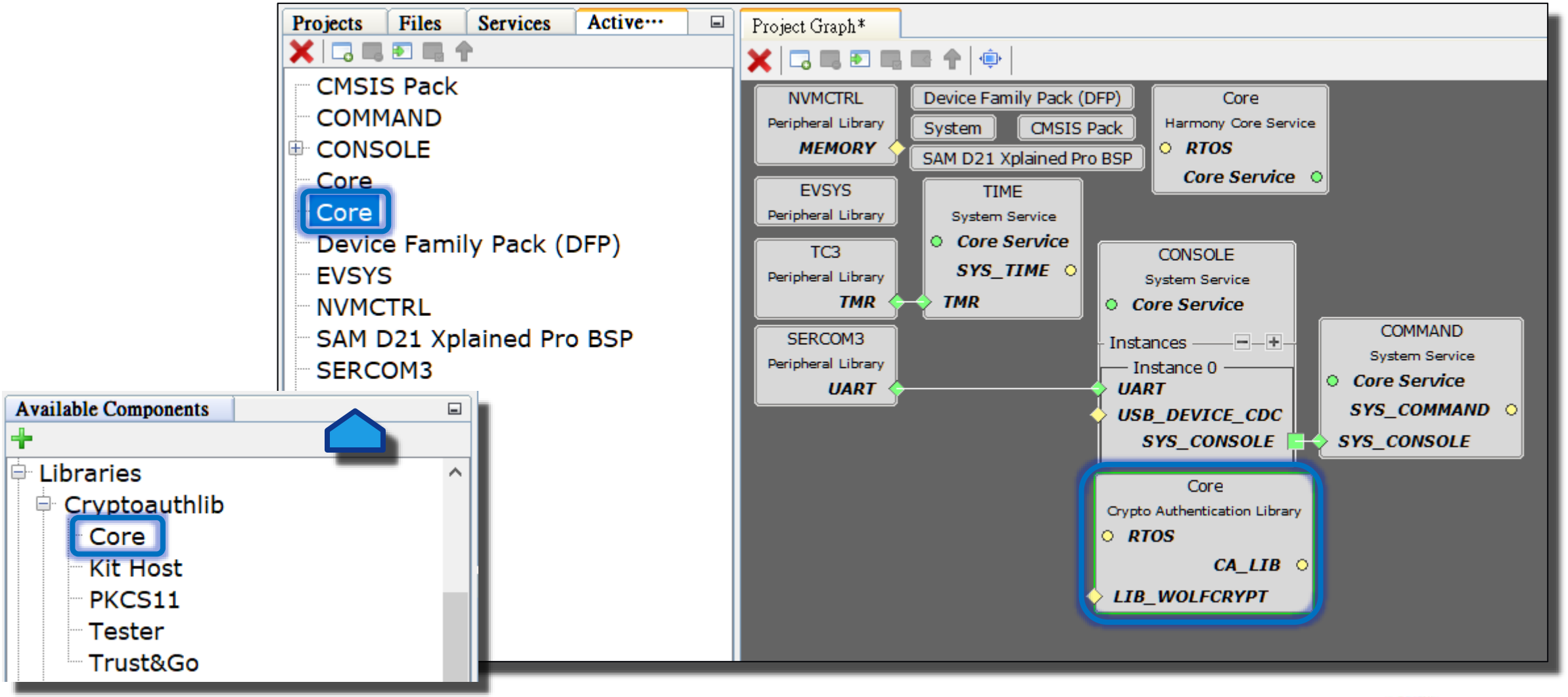
Harmony Configuration

Add COMMAND as Consumer of CONSOLE System Service



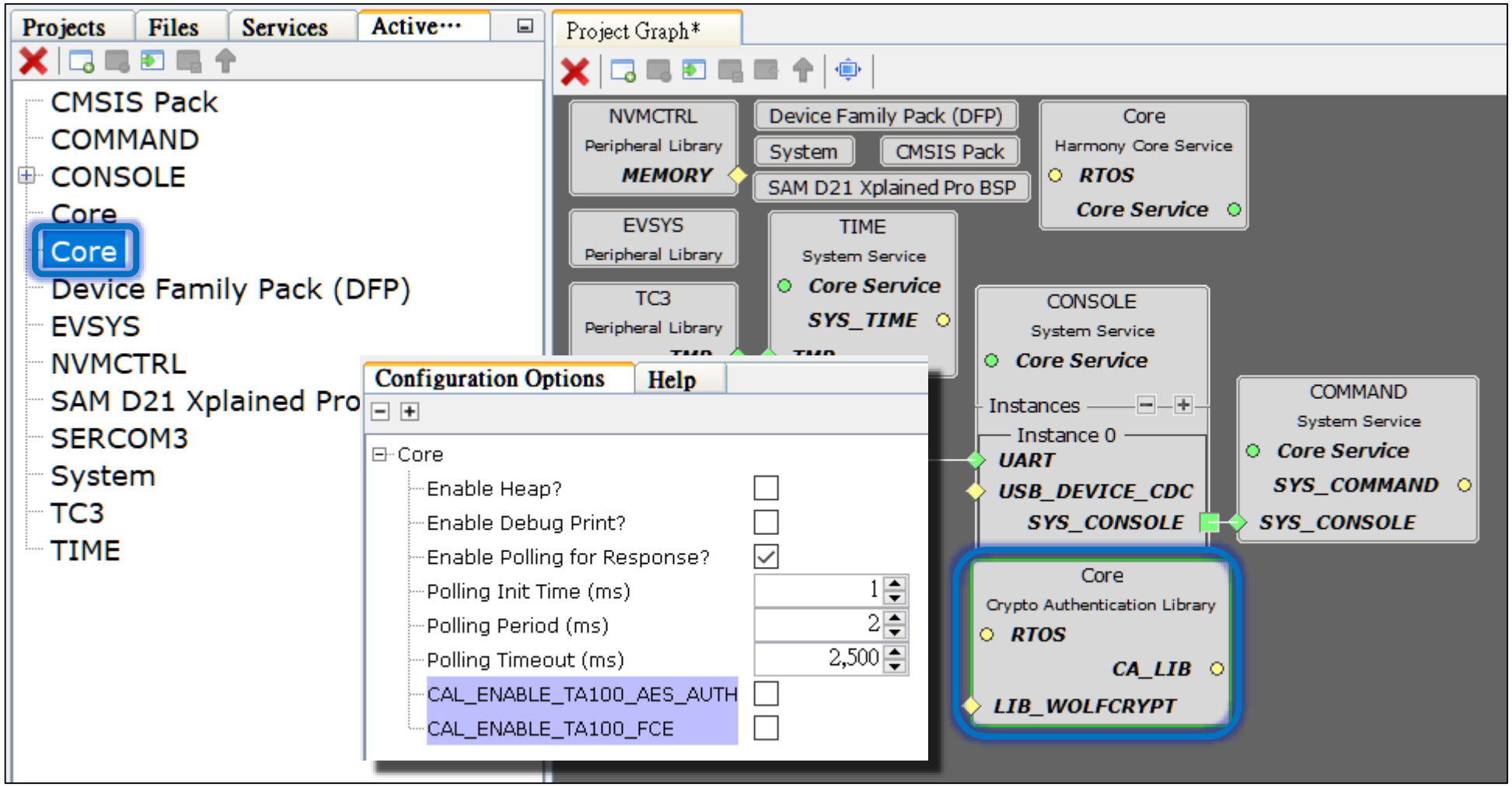
Harmony Configuration

Add CryptoAuthLib Core (Crypto Authentication Library)



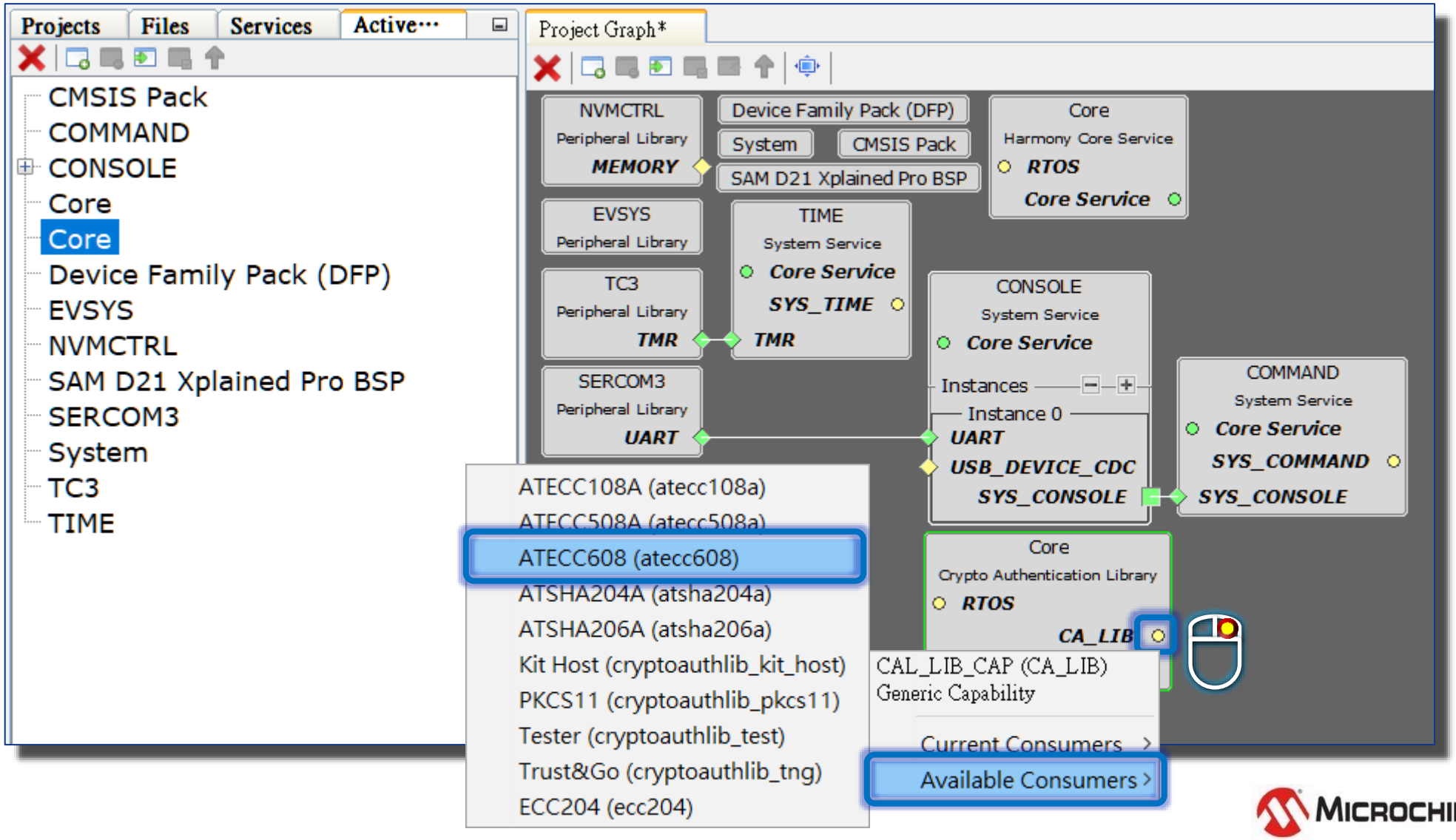
Harmony Configuration

Check CryptoAuthLib Core Configuration



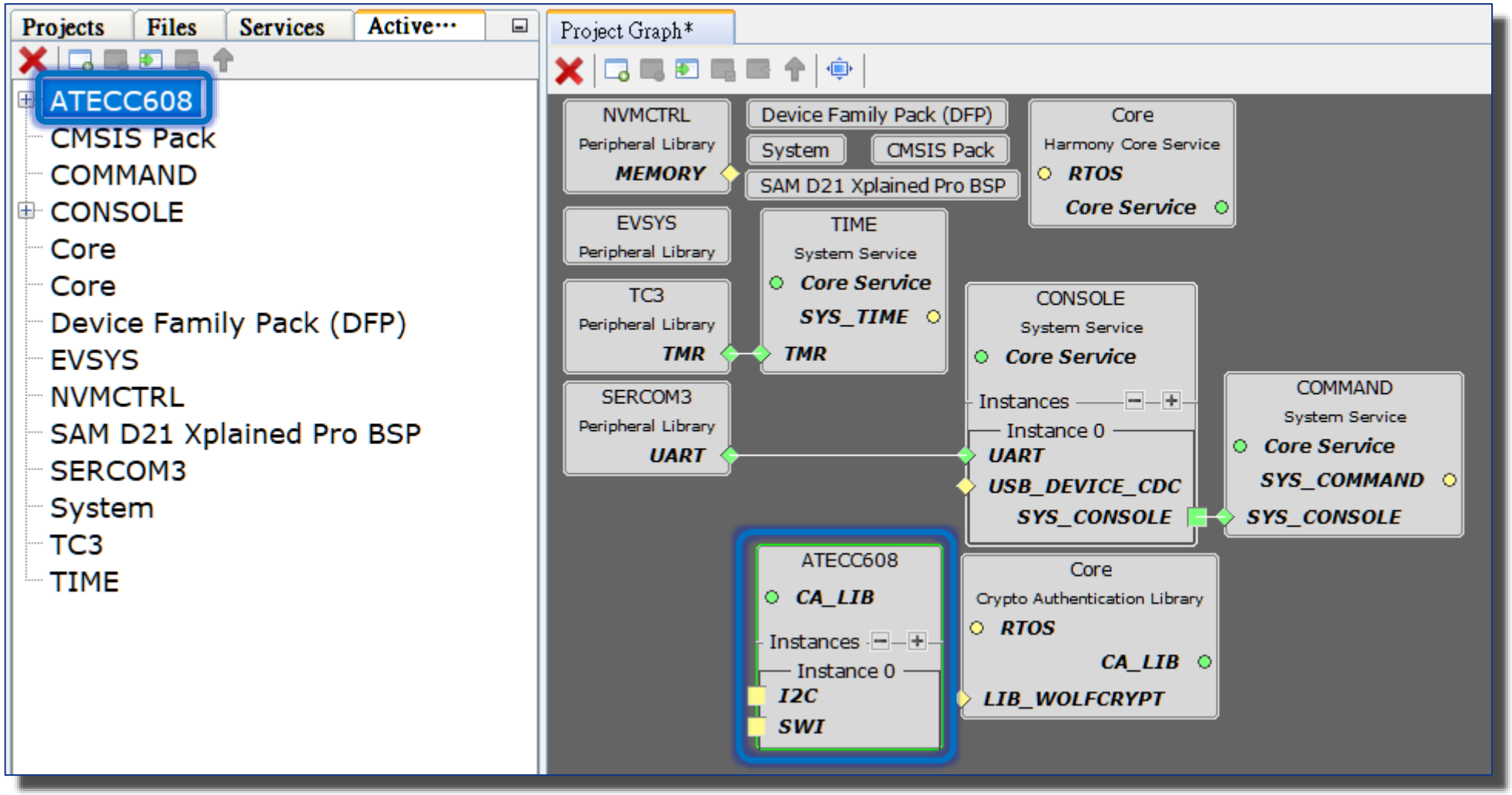
Harmony Configuration

Add ATECC608 Driver as Consumer of CryptoAuthLib Core



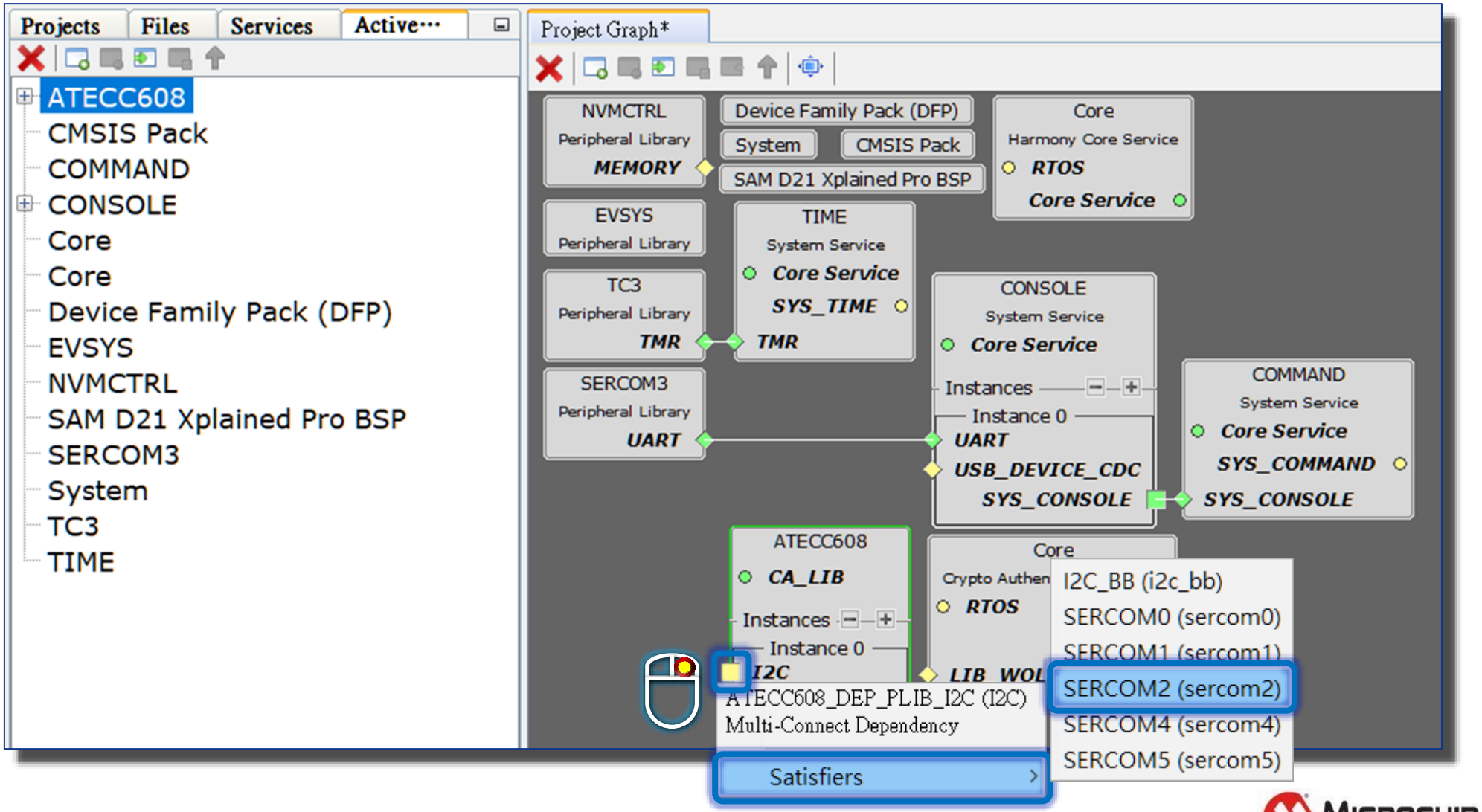
Harmony Configuration

Add ATECC608 Driver as Consumer of CryptoAuthLib Core



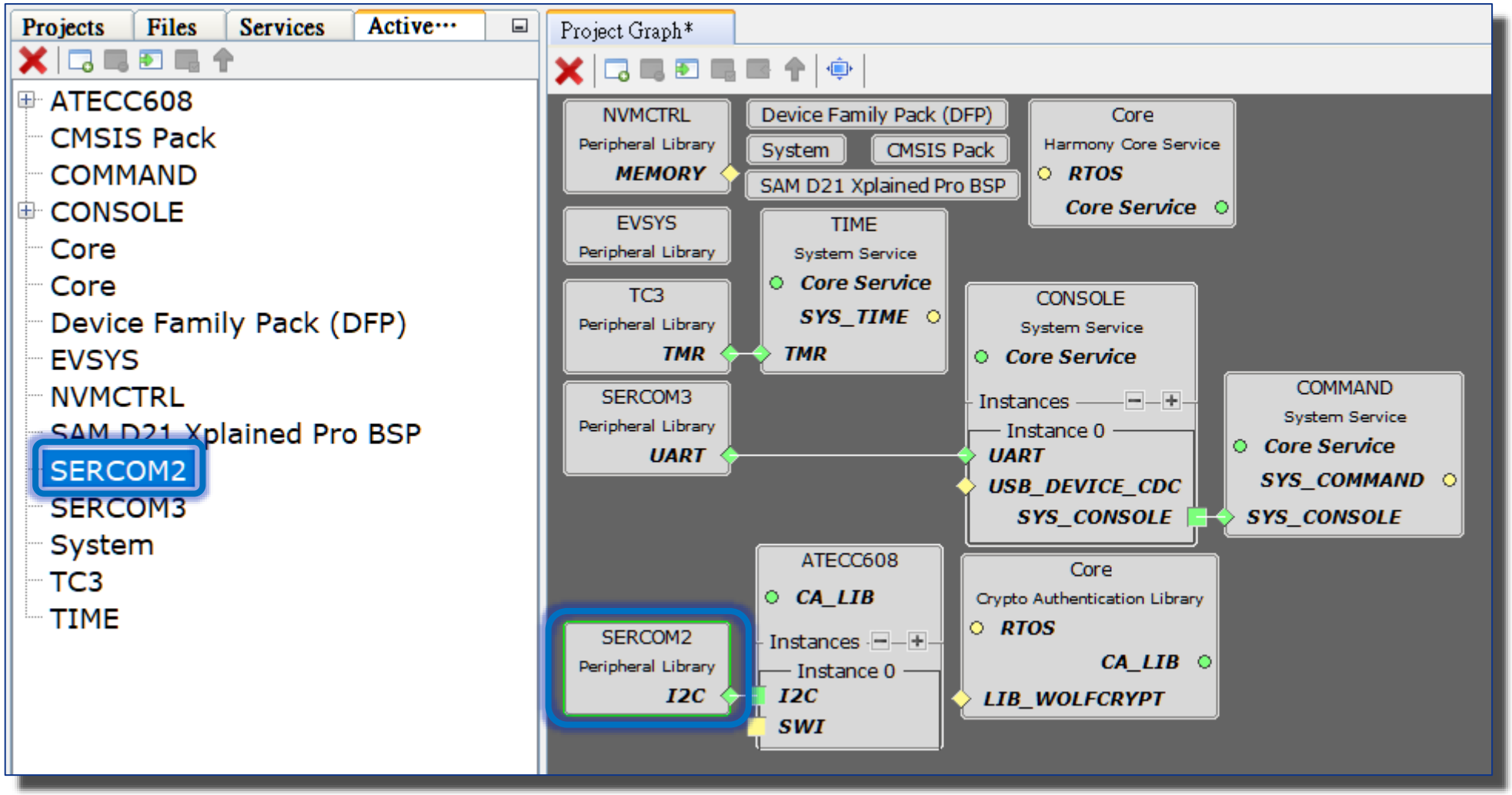
Harmony Configuration

Add SERCOM2 Peripheral Library for ATECC608 I2C Driver



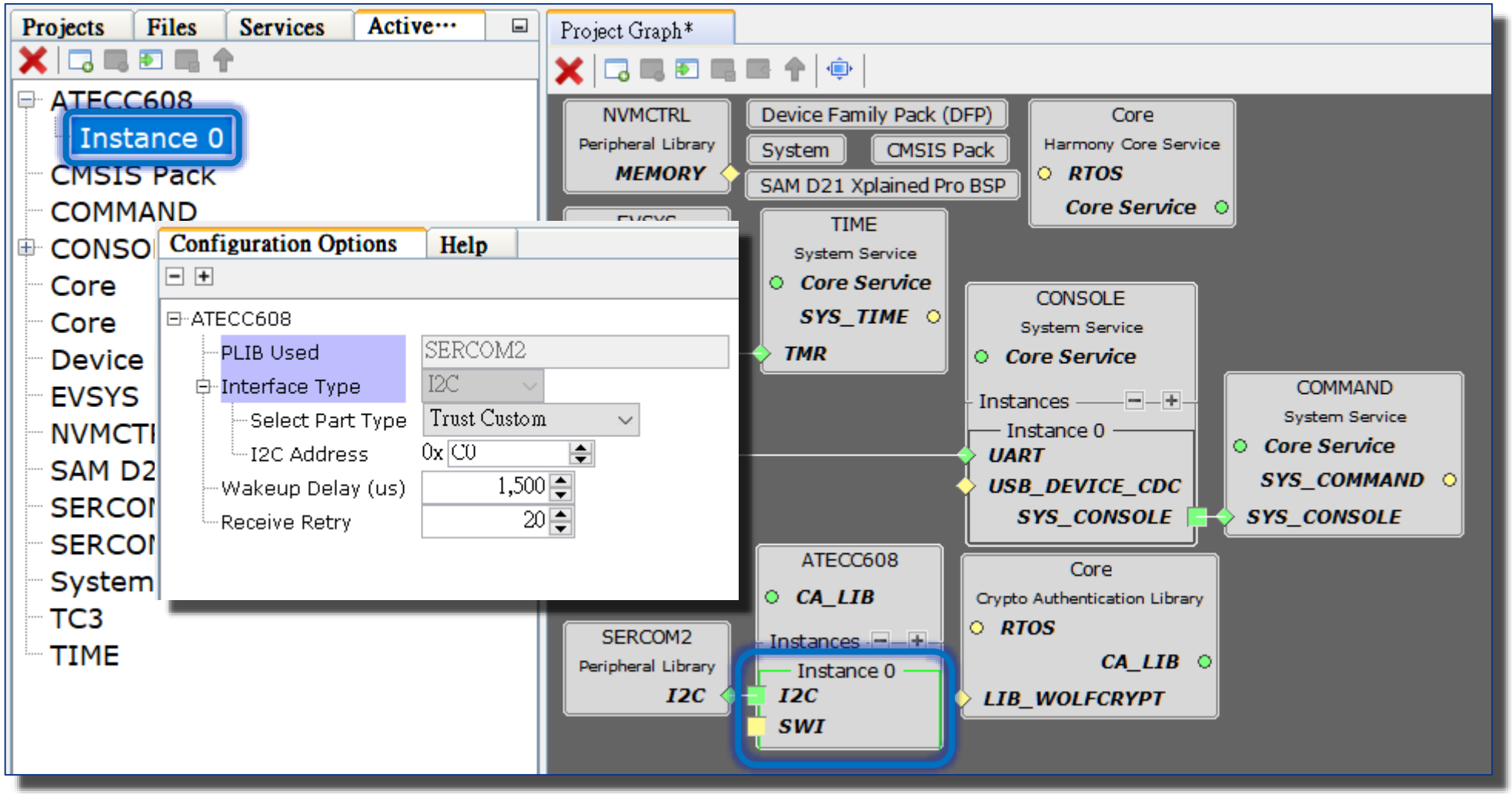
Harmony Configuration

Add SERCOM2 Peripheral Library for ATECC608 I2C Driver



Harmony Configuration

Check ATECC608 Driver Instance 0 Configuration



Harmony Configuration

Configure Pin of SERCOM2(PA08/PA09) and SERCOM3(PA22/PA23)

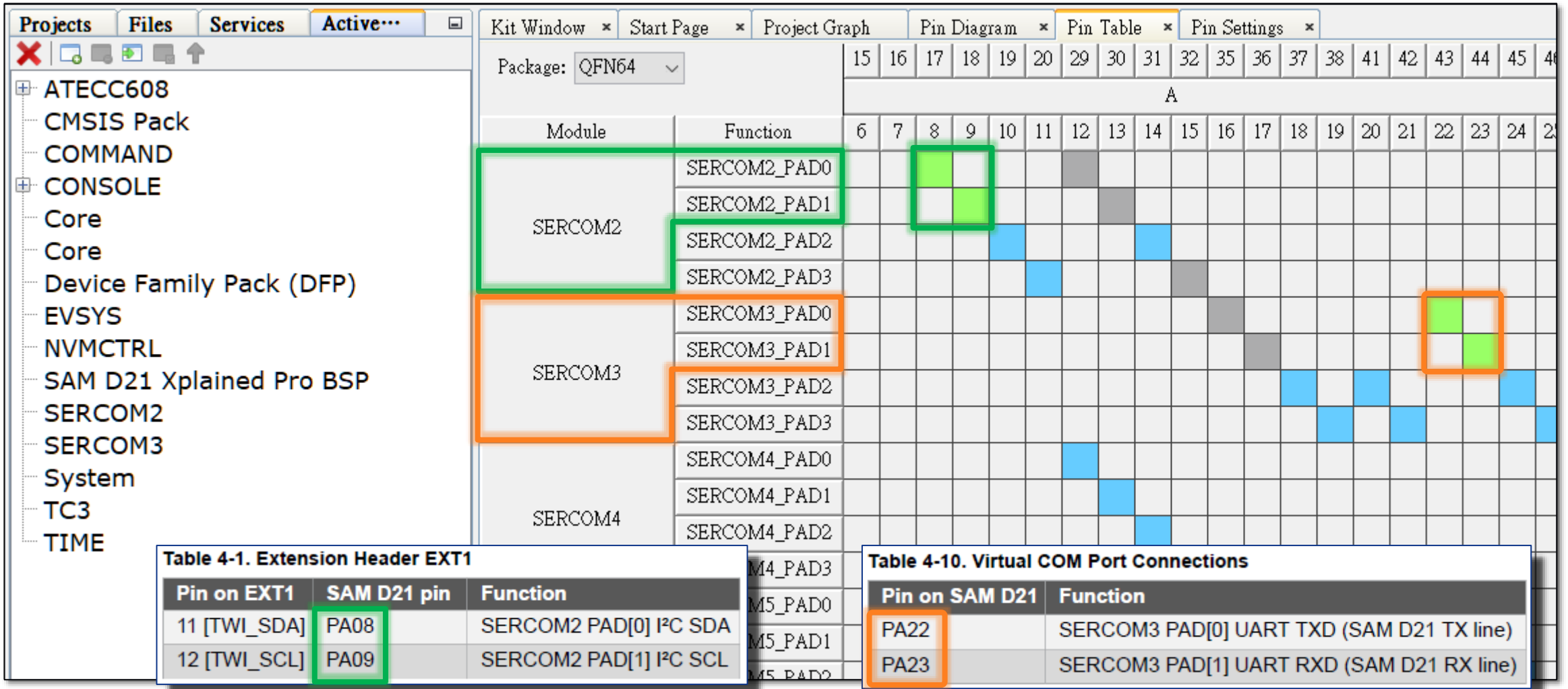


Table 4-1. Extension Header EXT1

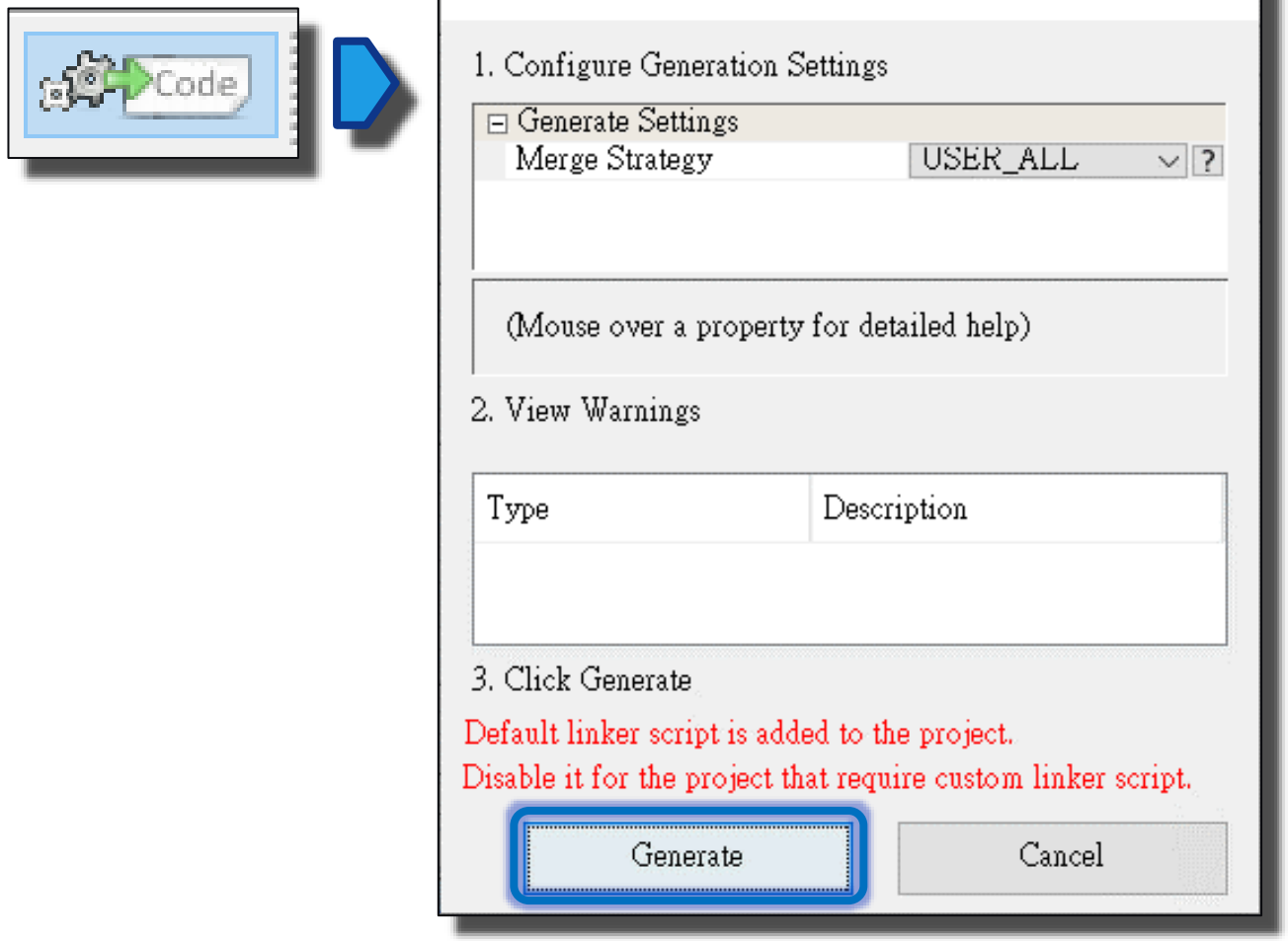
Pin on EXT1	SAM D21 pin	Function
11 [TWI_SDA]	PA08	SERCOM2 PAD[0] I²C SDA
12 [TWI_SCL]	PA09	SERCOM2 PAD[1] I²C SCL

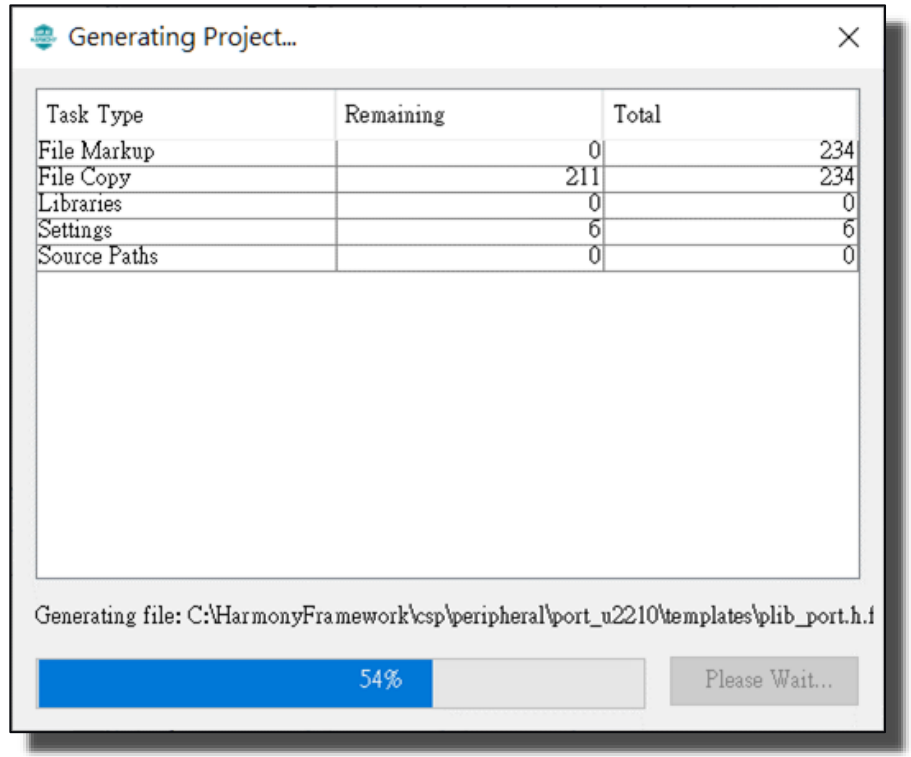
Table 4-10. Virtual COM Port Connections

Pin on SAM D21	Function
PA22	SERCOM3 PAD[0] UART TXD (SAM D21 TX line)
PA23	SERCOM3 PAD[1] UART RXD (SAM D21 RX line)

Harmony Configuration

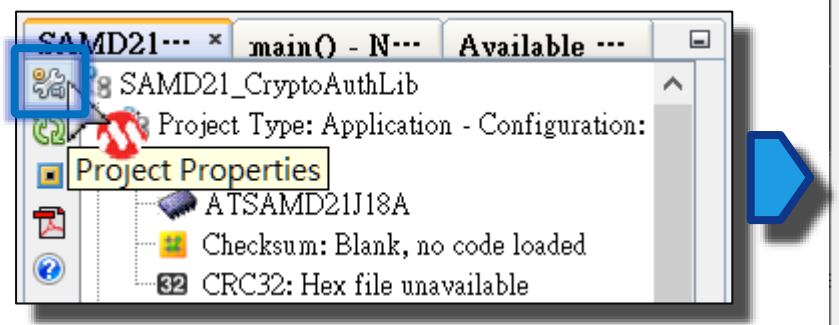
Generate Code and continue our APP implement

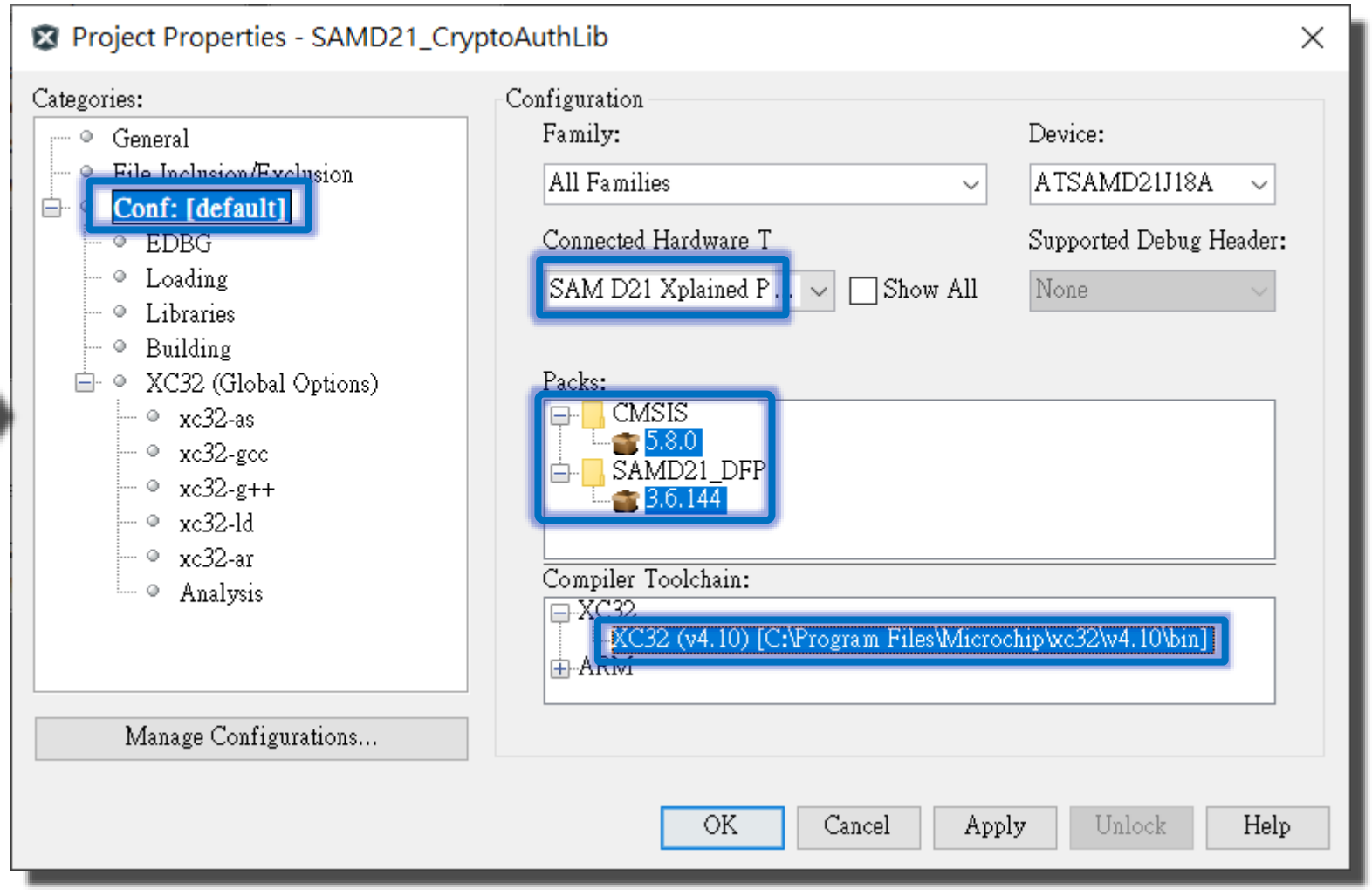




Harmony Configuration

Configure Project Properties (Debugger, DFP, CMSIS and Compiler)



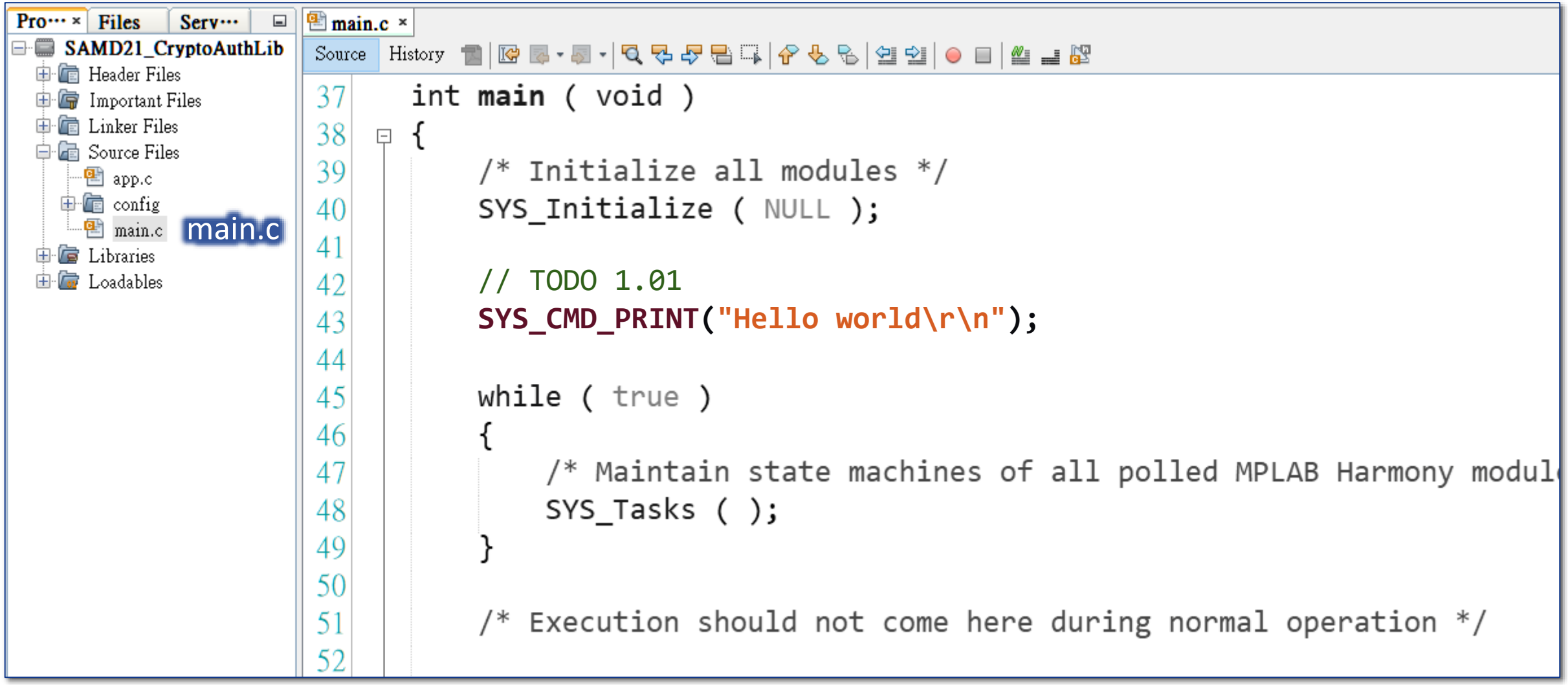


🛠 Lab1
Project Compiler and Console Output Test

Harmony Configuration

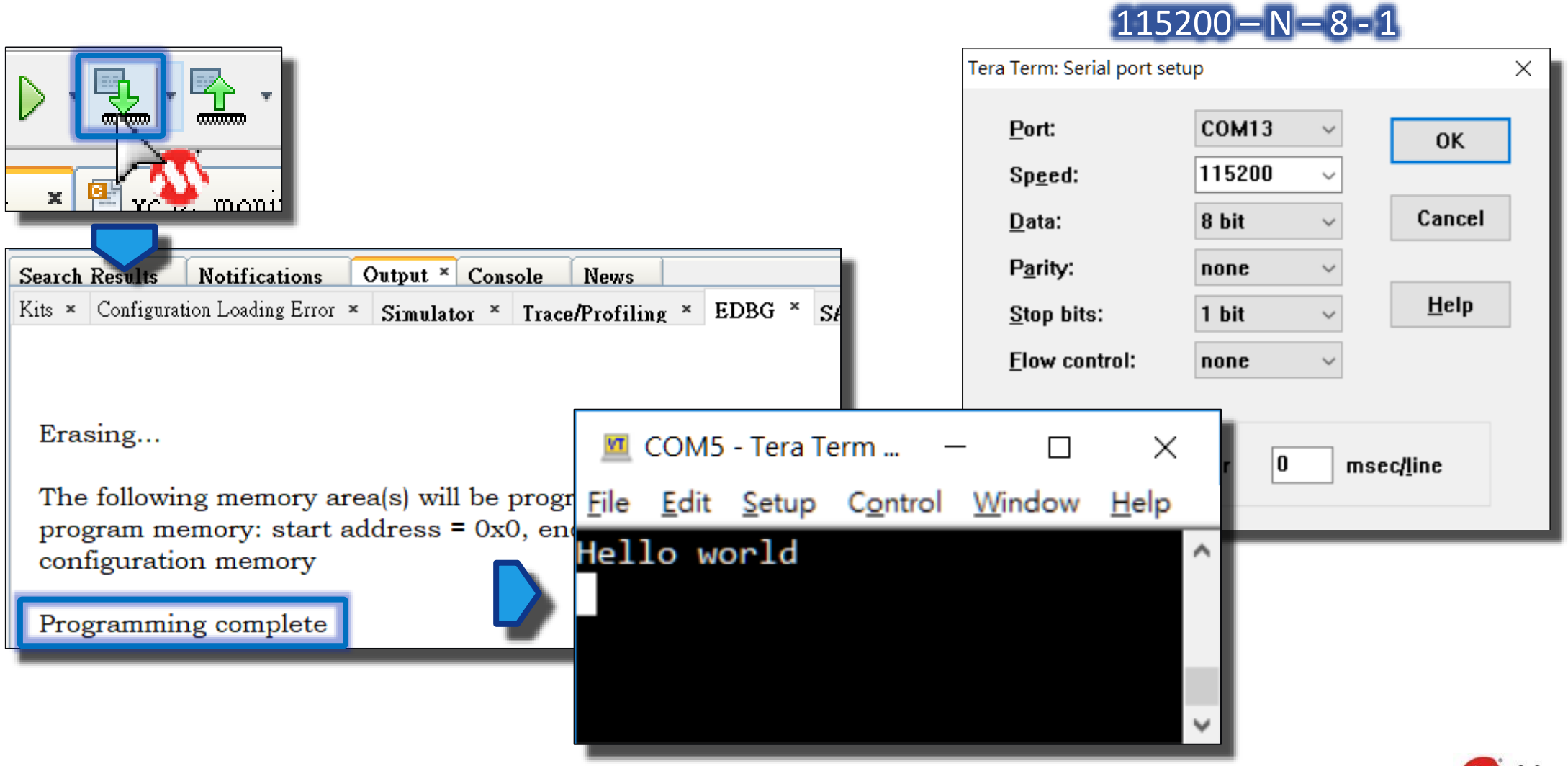
🛠 Lab1 Project Compiler and Console Output Test

Try your first “Hello World” UART output to serial console



🛠 Lab1 Project Compiler and Console Output Test

Compiler and Program your project to EVB.



🛠 Lab2
TIME System Service
LED Toggle

Harmony Configuration

MICROCHIP

Lab2 TIME System Service – LED Toggle

Use TIME System Service delay for LED blank

main.c

Source History

```
42      // TODO 1.01
43      SYS_CMD_PRINT("Hello world\r\n");
44
45      // TODO 2.01
46      SYS_TIME_HANDLE timer = SYS_TIME_HANDLE_INVALID;
47      SYS_TIME_DelayMS( 1000, &timer );
48
49      while ( true )
50      {
51          // TODO 2.02
52          if( SYS_TIME_DelayIsComplete(timer) )
53          {
54              LED_Toggle();
55              SYS_TIME_DelayMS( 1000, &timer );
56          }
57
58          /* Maintain state machines of all polled MPLAB Harmony modules. */
59          SYS_Tasks ( );
```

Handle of Timer

Create initial Delay Timer

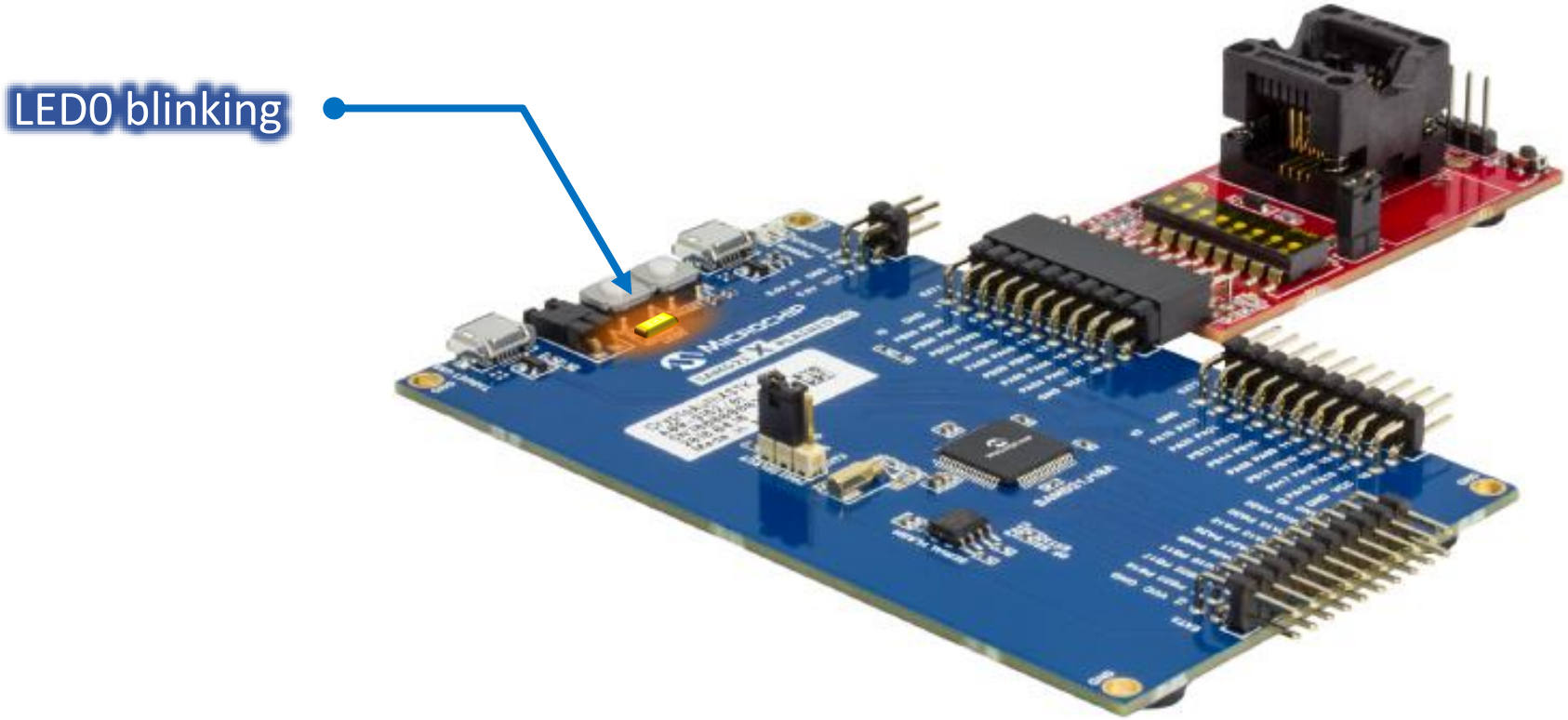
Check if Delay complete

Create next Delay Timer

MICROCHIP

🛠 Lab2 TIME System Service – LED Toggle

LED0 will blink in one second period



🛠 Lab3
COMMAND System Service Command Console

Harmony Configuration

MICROCHIP

Lab3 COMMAND System Service – Command Console

Declare a COMMAND table and related command functions

main.c

Source History

```
34  // ******************************************************************************
35  // ******************************************************************************
36  // TODO 3.01
37  void CommandHello(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
38  {
39      SYS_CMD_PRINT("\r\n- Hello Command!\r\n");
40  }
41
42  // TODO 3.02
43  static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
44  {
45      { "hello", CommandHello, ": Hello!" },
46  };
47
48  int main ( void )
49  {
```

Command

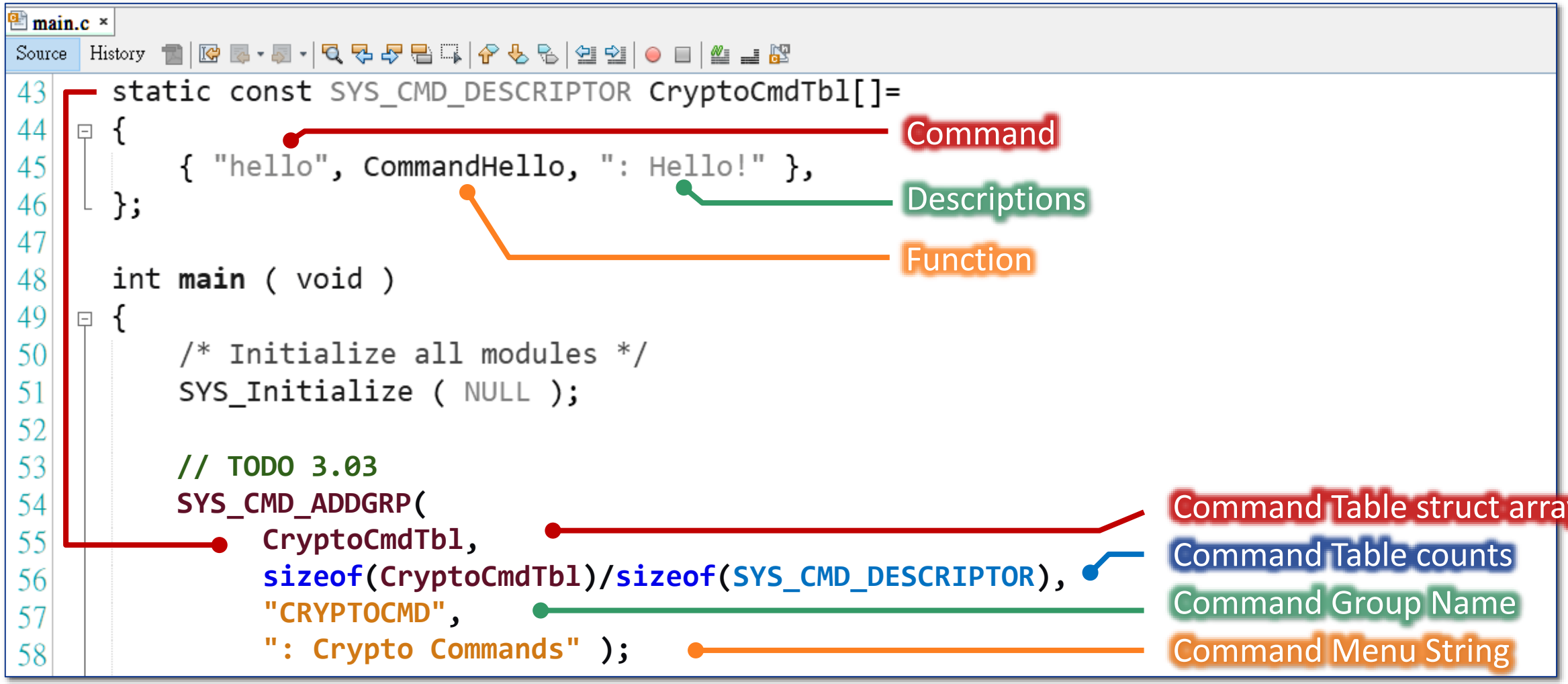
Descriptions

Function

MICROCHIP

Lab3 COMMAND System Service – Command Console

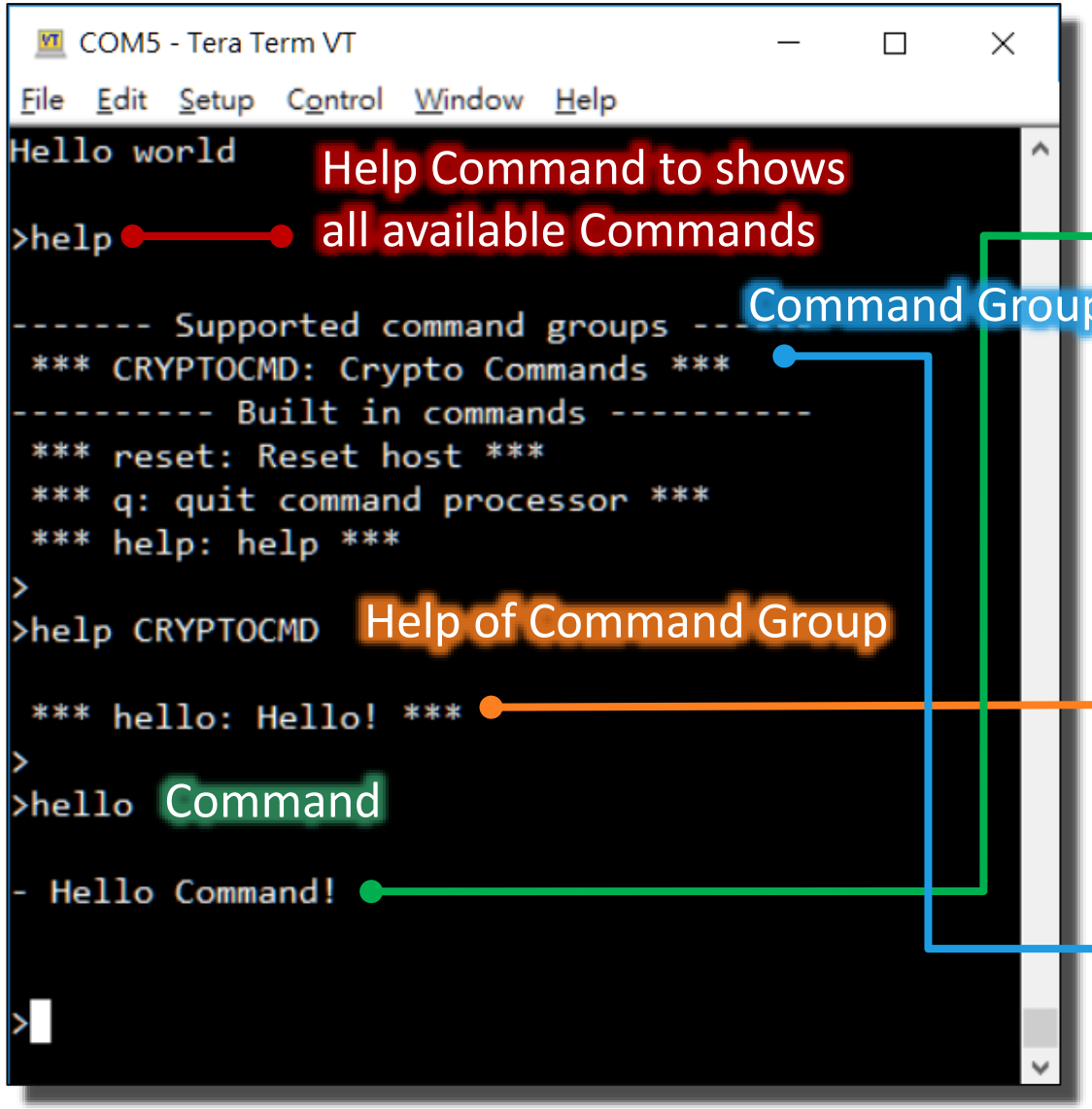
Add Custom Command Table as New Command Group



MICROCHIP

Lab3 COMMAND System Service – Command Console

Result of your first COMMAND System Service




```
// TODO 3.01
void CommandHello(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    SYS_CMD_PRINT("\r\n- Hello Command!\r\n");
}

// TODO 3.02
static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
{
    { "hello", CommandHello, ": Hello!" },
};

int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    // TODO 3.03
    SYS_CMD_ADDGRP(
        CryptoCmdTbl,
        sizeof(CryptoCmdTbl)/sizeof(SYS_CMD_DESCRIPTOR),
        "CRYPTOCMD",
        ": Crypto Commands" );
```

MICROCHIP

Lab3 COMMAND System Service – Command Console

Add parameter for your COMMAND

main.c

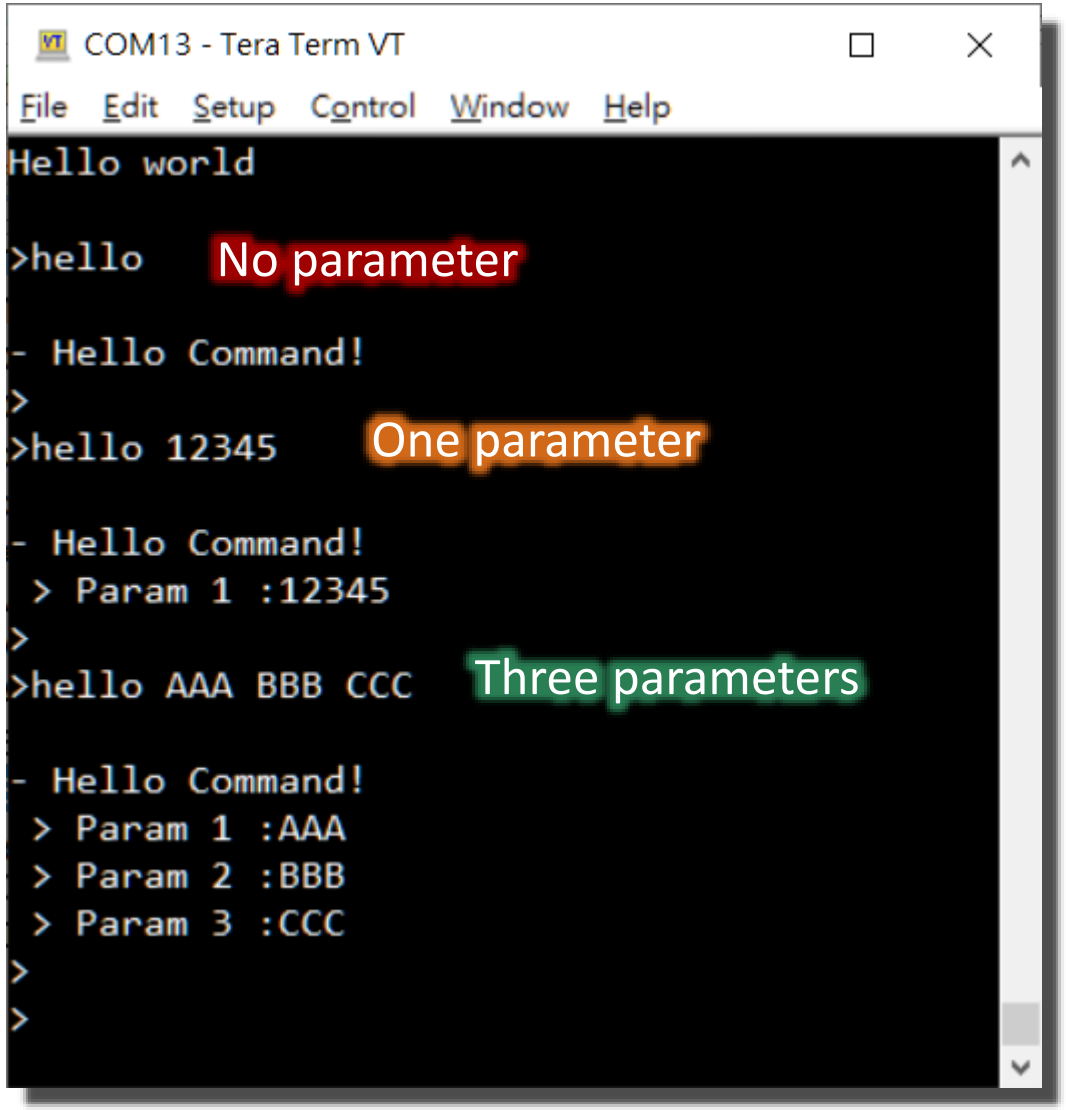
Source History

```
36  // TODO 3.01
37  void CommandHello(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
38  {
39      SYS_CMD_PRINT("\r\n- Hello Command!\r\n");
40
41      // TODO 3.04
42      if( argc>1 )
43      {
44        for( int i=1 ; i<argc ; i++ )
45        {
46          SYS_CMD_PRINT(" > Param %d :%s\r\n", i, argv[i] );
47        }
48      }
49  }
50
51  // TODO 3.02
```

Print all parameters

🛠 Lab3 COMMAND System Service – Command Console

Result of COMMAND parameters




```
// TODO 3.01
void CommandHello(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    SYS_CMD_PRINT("\r\n- Hello Command!\r\n");

    // TODO 3.04
    if( argc>1 )
    {
        for( int i=1 ; i<argc ; i++ )
        {
          SYS_CMD_PRINT(" > Param %d :%s\r\n", i, argv[i] );
        }
    }
}

// TODO 3.02
static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
{
    { "hello", CommandHello, ": Hello!" },
};
```

Parameter Count

Parameters

MICROCHIP

Usage of CryptoAuthLib

The library for Crypto Authentication family

MICROCHIP

Usage of CryptoAuthLib

Related resource

- **The CryptoAuthentication Library GitHub**
  - https://github.com/MicrochipTech/cryptoauthlib
- **CryptoAuthentication Library on-line document**
  - https://microchiptech.github.io/cryptoauthlib/
- **Classic CryptoAuthentication standalone library (Registration need)**
  - https://www.microchip.com/swlibraryweb/product.aspx?product=cryptoauthlib

Usage of CryptoAuthLib

Introduction

- **Host Device Support**
  - CryptoAuthLib will run on a variety of platforms from small micro-controllers to desktop host systems.
  - Rich OS Hosts:
    - Linux Kit Protocol over HID USB
    - Linux I2C
    - Linux SPI
    - Windows Kit Protocol over HID USB
  - Microcontrollers:
    - Microchip AVR, SAM, & PIC families.

- **CryptoAuthLib Architecture**
  - The library is structured to support portability to:
    - multiple hardware/microcontroller platforms
    - multiple environments including bare-metal, RTOS and Windows/Linux/MacOS
    - multiple chip communication protocols (I2C, SPI, and SWI)
  - All platform dependencies are contained within the HAL (hardware abstraction layer).

Usage of CryptoAuthLib

CryptoAuthLib directory Structure

cryptoauthlib-manual.pdf - Help Document

/app	- Application specific implementation of various use cases.
/lib	- Primary library source code.
/atcacert	- Certificate data and i/o methods.
/calib	- The Basic CryptoAuth API.
/crypto	- Software crypto implementations external crypto libraries support.
/hal	- Hardware abstraction layer code for supporting specific platforms.
/host	- Support functions for common Host-side calculations.
/jwt	- JSON web token functions.
/mbedtls	- Interfacing and wrapper functions to integrate mbedtls.
/openssl	- interfacing and wrapper functions to integrate openssl .
/pkcs11	- PKCS11 Library Utility Functions.
/wolfssl	- WolfSSL interface Functions.
/python	- Thin Python ctypes layer to evaluate the cryptoauthlib interface.
/test	- Integration test and examples.
/third_party	- 3rd party supports.

Usage of CryptoAuthLib

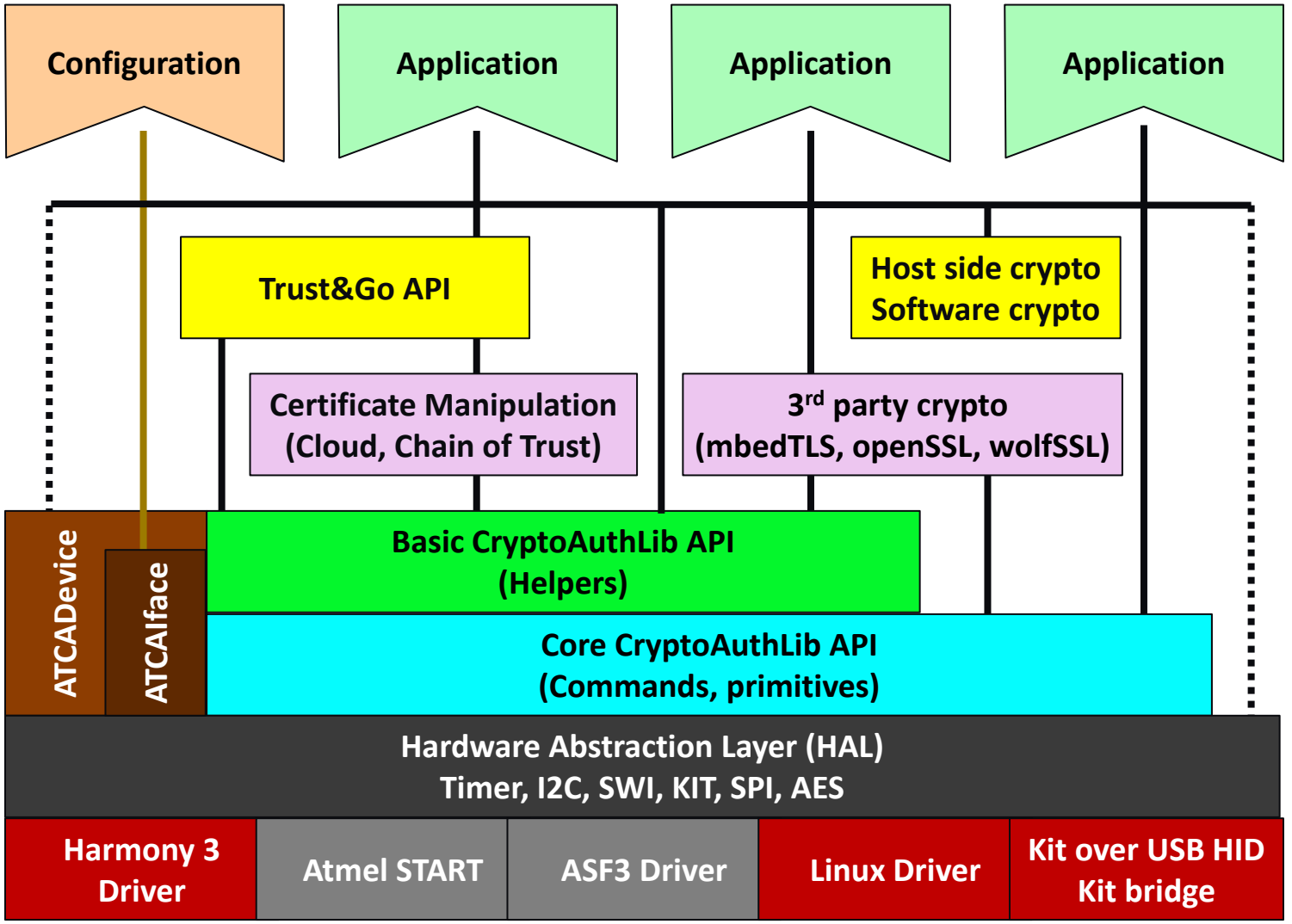
CryptoAuthLib Modules

Module Name	Function Prefix	Description
Basic Crypto API methods	**atcab_**	These methods provide the most convenient, simple API to CryptoAuth chips.
Helpers of Basic Crypto API	**atcab_**	Helpers to support the CryptoAuthLib Basic API methods.
Core Crypto API methods for CryptoAuth Devices	**calib_**	These methods provide a simple API to CryptoAuth chips.
ATCACommand	**atXxx**	Builds commands as an array of bytes to send to the CryptoAuthentication device.
ATCADevice	**xxxATCAxxx**	ATCADevice object - composite of command and interface objects.
Configuration	**cfg_**	Logical device configurations describe the CryptoAuth device type and logical interface
ATCAIface	**atxxx** **atca_iface** **xxxATCAIface**	Abstract interface to all CryptoAuth device types. This interface connects to the HAL implementation and abstracts the physical details of the device communication from all the upper layers of CryptoAuthLib.
Hardware abstraction layer	**hal_, kit_, swi_**	These methods define the hardware abstraction layer for communicating with a CryptoAuth device
Host side crypto methods	**atcah_**	The functions provide host-side cryptographic functionality for an ATECC client device. They are intended to accompany the CryptoAuthLib functions.
Software crypto methods	**atcac_**	These methods provide a software implementation of various crypto algorithms.
Certificate manipulation methods	**atcacert_**	These methods provide convenient ways to perform certification I/O with CryptoAuth chips and perform certificate manipulation in memory.
JSON Web Token (JWT) methods	**atca_jwt_**	Methods for signing and verifying JSON Web Token (JWT) tokens.
mbedTLS Wrapper methods	**atca_mbedtls_**	These methods are for interfacing cryptoauthlib to mbedtls.
TNG(Trust&Go) API	**tng_**	These methods provide some convenience functions (mostly around certificates) for TNG devices, which currently include ATECC608A-MAHTN-T.
Attributes	**pkcs11_, C_**	PKCS11 attributes support methods.

Usage of CryptoAuthLib

CryptoAuthLib Modules Hierarchy

Module Name	Function Prefix
Basic Crypto API methods	atcab_
Helpers of Basic Crypto API	atcab_
Core Crypto API methods for CryptoAuth Devices	calib_
ATCACommand	atXxx
ATCADevice	xxxATCAxxx
Configuration	cfg_
ATCAIface	atxxx atca_iface xxxATCAIface
Hardware abstraction layer	hal_, kit_, swi_
Host side crypto methods	atcah_
Software crypto methods	atcac_
Certificate manipulation methods	atcacert_
JSON Web Token (JWT) methods	atca_jwt_
mbedTLS Wrapper methods	atca_mbedtls_
TNG(Trust&Go) API	tng_
Attributes	pkcs11_, C_



Usage of CryptoAuthLib

Basic Crypto API methods (atcab_)

These methods provide the most convenient, simple API to CryptoAuth chips.

atcab_
version
init_ext
init
init_device
release_ext
release
get_device
get_device_type_ext
get_device_type
get_device_address
is_ca_device
is_ta_device
wakeup
idle
sleep
get_zone_size
_atcab_exit
aes_xxxx
checkmac
counter
counter_increment

counter_read
derivekey
ecdh_base
ecdh
ecdh_enc
ecdh_ioenc
ecdh_tempkey
ecdh_tempkey_ioenc
gendig
genkey_base
genkey
get_pubkey
get_pubkey_ext
hmac
info_base
Info
info_set_latch
info_get_latch
kdf
lock
lock_config_zone
lock_config_zone_crc

lock_data_zone
lock_data_zone_crc
lock_data_slot
mac
nonce_base
nonce
nonce_load
nonce_rand
challenge
challenge_seed_update
priv_write
random
random_ext
read_zone
is_locked
is_config_locked
is_data_locked
is_slot_locked
read_bytes_zone
read_serial_number
read_pubkey
read_sig

read_config_zone
cmp_config_zone
read_enc
secureboot
secureboot_mac
selftest
sha_base
sha_start
sha_update
sha_end
sha_read_context
sha_write_context
sha
hw_sha2_256
hw_sha2_256_init
hw_sha2_256_update
hw_sha2_256_finish
sha_hmac_init
sha_hmac_update
sha_hmac_finish
sha_hmac
sign_base

sign
sign_ext
sign_internal
updateextra
verify
verify_extern
verify_extern_ext
verify_extern_mac
verify_stored
verify_stored_ext
verify_stored_mac
verify_validate
verify_invalidate
write
write_zone
write_bytes_zone
write_pubkey
write_config_zone
write_enc
write_config_counter
execute_command (#)
get_addr (#)

Usage of CryptoAuthLib

Helpers to support the CryptoAuthLib Basic API (atcab_)

These methods provide the most convenient, simple API to CryptoAuth chips.

atcab_
printbin
printbin_sp
printbin_label
bin2hex
bin2hex_
hex2bin
hex2bin_
memset_s
reversal
base64decode
base64decode_
base64decode_block
base64encode
base64encode_

Others
packHex
isAlpha
isBase64
isBase64Digit
isBlankSpace
isDigit
isHex
isHexAlpha
isHexDigit
base64Char
base64Index

Usage of CryptoAuthLib

Basic Crypto API methods for CryptoAuth Devices (calib_)

These methods provide the most convenient, simple API to CryptoAuth chips.

calib_
wakeup
idle
sleep
_exit
get_addr
get_zone_size
ecc204_get_addr
aes
aes_encrypt
aes_decrypt
aes_gfm
checkmac
counter
counter_increment
counter_read
derivekey
ecdh_base
ecdh
ecdh_enc
ecdh_ioenc
ecdh_tempkey
ecdh_tempkey_ioenc

gendig
genkey_base
genkey
get_pubkey
genkey_mac
hmac
info_base
info
info_set_latch
info_get_latch
info_privkey_valid
info_lock_status
ecc204_is_locked
ecc204_is_data_locked
ecc204_is_config_locked
kdf
lock
lock_config_zone
lock_config_zone_crc
lock_data_zone
lock_data_zone_crc
lock_data_slot
ecc204_lock_config_slot

ecc204_lock_config_zone
ecc204_lock_data_slot
mac
nonce_base
nonce
nonce_load
nonce_rand
challenge
challenge_seed_update
nonce_gen_session_key
priv_write
random
read_zone
is_locked
is_slot_locked
read_bytes_zone
read_serial_number
read_pubkey
read_sig
read_config_zone
cmp_config_zone
ecc204_read_zone
ecc204_read_config_zone

ecc204_read_serial_number
ecc204_read_bytes_zone
ecc204_cmp_config_zone
read_enc
secureboot
secureboot_mac
selftest
sha_base
sha_start
sha_update
sha_end
sha_read_context
sha_write_context
sha
hw_sha2_256
hw_sha2_256_init
hw_sha2_256_update
hw_sha2_256_finish
sha_hmac_init
sha_hmac_update
sha_hmac_finish
sha_hmac
sign_base

sign
sign_internal
ecc204_sign
updateextra
verify
verify_extern
verify_extern_mac
verify_stored
verify_stored_mac
verify_validate
verify_invalidate
write
write_zone
write_bytes_zone
write_pubkey
write_config_zone
ecc204_write
ecc204_write_zone
ecc204_write_config_zone
ecc204_write_bytes_zone
write_enc
ecc204_write_enc
write_config_counter

Usage of CryptoAuthLib

ATCACommand (atXxx)

The ATCACommand is an object that builds commands as an array of bytes to send to the CryptoAuthentication device. ATCACommand does not send the command, it only builds the command. As part of that process, it accepts a packet, ATCAPacket, which contains the parameters for the command. From that information, it determines the expected length of the response packet and computes the CRC of the command to send.

atXxx
atAES
atCRC
atCalcCrc
atCheckMAC
atCounter
atDeriveKey
atECDH
atGenDig
atGenKey

atHMAC
atInfo
atIsECCFamily
atIsSHAFamily
atKDF
atLock
atMAC
atNonce
atPause
atPrivWrite

atRandom
atRead
atSHA
atSecureBoot
atSelfTest
atSign
atUpdateExtra
atVerify
atWrite
isATCAError

Usage of CryptoAuthLib

ATCADevice (xxxATCAxxx)

ATCADevice object - composite of command and interface objects.

xxxATCAxxx
newATCADevice
deleteATCADevice
initATCADevice
releaseATCADevice
atGetIFace

Usage of CryptoAuthLib

Configuration (cfg_)

Logical device configurations describe the CryptoAuth device type and logical interface.

Ex. The default configuration for an ECC608 device with I2C interface.

```
ATCAIfaceCfg cfg_ateccx08a_i2c_default =
{
    .iface_type                 = ATCA_I2C_IFACE,
    .devtype                    = ATECC608,
    {
        .atcai2c.address        = 0xC0,
        .atcai2c.bus            = 2,
        .atcai2c.baud           = 400000,
    },
    .wake_delay                 = 1500,
    .rx_retries                 = 20
};
```

Usage of CryptoAuthLib

ATCAIface (atxxx, atca_iface, xxxATCAIface)

Abstract interface to all CryptoAuth device types. This interface connects to the HAL implementation and abstracts the physical details of the device communication from all the upper layers of CryptoAuthLib.

atxxx
atinit
atsend
atreceive
atcontrol
atwake
atidle
atsleep
atgetifacecfg
atgetifacehaldat

atca_iface, xxxATCAIface
atca_iface_is_kit
atca_iface_get_retries
atca_iface_get_wake_delay
initATCAIface
newATCAIface
releaseATCAIface
deleteATCAIface

Usage of CryptoAuthLib

Hardware abstraction layer (hal_, kit_, swi_)

These methods define the hardware abstraction layer for communicating with a CryptoAuth device. These functions perform the transmit/receive of data for a given interface. These are split into sublayers:

- The HAL layer is the first hal layer that presents the interface expected by the higher-level library. When using a native driver and no further interpretation is required this layer is all that is required.
- The PHY layer if for hals that perform an interpretation or additional protocol logic. In this situation the HAL performs protocol interpretation while the phy performs the physical communication.

A hal or phy must have the following functions and their signatures:

ATCA_STATUS **hal_<name>_init**(ATCAIface iface, ATCAIfaceCfg *cfg)

ATCA_STATUS **hal_<name>_post_init**(ATCAIface iface)

ATCA_STATUS **hal_<name>_send**(ATCAIface iface, uint8_t address, uint8_t *txdata, int txlength);

ATCA_STATUS **hal_<name>_receive**(ATCAIface iface, uint8_t address, uint8_t *rxdata, uint16_t *rxlength)

ATCA_STATUS **hal_<name>_control**(ATCAIface iface, uint8_t option, void* param, size_t paramlen)

ATCA_STATUS **hal_<name>_release**(void *hal_data)

Usage of CryptoAuthLib

Hardware abstraction layer (hal_, kit_, swi_)

CryptoAuthLib Supported HAL Layers

Device Interface	Physical Interface	HAL	PHY
I2C	I2C	hal_i2c	
	GPIO	hal_i2c_gpio	hal_gpio
SPI	SPI	hal_spi	
SWI	UART	hal_swi	hal_uart
	GPIO	hal_swi_gpio	hal_gpio
Any	UART	kit	hal_uart
	HID	kit	hal_hid
	Any (user provided)	kit_bridge	

Usage of CryptoAuthLib

Hardware abstraction layer (hal_, kit_, swi_)

Implement of Supported HAL Layers

Framework/OS		Interface	Implement Files	API	Note
Harmony 3		I2C	hal_i2c_harmony.c	plib.h	For all Harmony 3 based projects
		SPI	hal_spi_harmony.c		
		UART	hal_uart_harmony.c		
OS Support	Linux	I2C	hal_linux_i2c_userspace.c/h	I2c-dev	
		SPI	hal_linux_spi_userspace.c/h	spi-dev	
	Linux/MacOS	---	hal_linux.c		For all Linux/MacOS projects
	Windows	---	hal_windows.c		For all Windows projects
	OS All	Kit-hid	hal_all_platforms_kit_hidapi.c/h	hidapi	Work for Windows, Linux, and MacOS
	freeRTOS	---	hal_freertos.c		freeRTOS common routines
Classic ASF	Atmel START	---	hal_timer_start.c	ASF4	Timer implementation
		I2C	hal_i2c_start.c/h		
		SWI	swi_uart_start.c/h		SWI using UART
	cortex-m0	I2C	hal_sam0_i2c_asf.c/h	ASF3	SAMD21, SAMB11, etc
	cortex-m3/4/7	I2C	hal_sam_i2c_asf.c/h		SAM4S, SAMG55, SAMV71, etc
	ASF3 All	---	hal_sam_timer_asf.c		Common timer hal for all platforms

Usage of CryptoAuthLib

Hardware abstraction layer (hal_, kit_, swi_)

These methods define the hardware abstraction layer for communicating with a CryptoAuth

hal_
iface_init
iface_register_hal
iface_release
check_wake
rtos_delay_ms
delay_ms
delay_us
delay_10us
create_mutex
destroy_mutex
lock_mutex
unlock_mutex
malloc
free
is_command_word
i2c_discover_buses
i2c_discover_devices
i2c_init
i2c_post_init
i2c_send
i2c_receive
i2c_control
i2c_release
i2c_wake
i2c_idle
i2c_sleep
kit_hid_init
kit_hid_post_init
kit_hid_send
kit_hid_receive
kit_hid_control
kit_hid_release
kit_attach_phy
kit_discover_buses
kit_discover_devices
kit_init
kit_post_init
kit_send
kit_receive
kit_control
kit_release
spi_discover_buses
spi_discover_devices
spi_init
spi_post_init
spi_select
spi_deselect
spi_send
spi_receive
spi_control
spi_release
swi_discover_buses
swi_discover_devices
swi_init
swi_post_init
swi_send
swi_receive
swi_wake
swi_idle
swi_sleep
swi_release
swi_control

kit_
id_from_devtype
interface_from_kittype
phy_send
phy_receive
Init
post_init
send
receive
wake
idle
sleep
wrap_cmd
parse_rsp
control
release

swi_
uart_init
uart_deinit
uart_setbaud
uart_mode
uart_discover_buses
uart_send_byte
uart_receive_byte

atca_
delay_10us
delay_ms
delay_us

others
change_i2c_speed

Usage of CryptoAuthLib

Host side crypto methods (atcah_)

The functions provide host-side cryptographic functionality for an ATECC client device. They are intended to accompany the CryptoAuthLib functions.

atcah_
nonce
mac
check_mac
hmac
gen_dig
gen_mac
write_auth_mac
privwrite_auth_mac
derive_key
derive_key_mac
decrypt

sha256
include_data
gen_key_msg
config_to_sign_internal
sign_internal_msg
verify_mac
secureboot_enc
secureboot_mac
encode_counter_match
io_decrypt
ecc204_write_auth_mac
gen_session_key

Usage of CryptoAuthLib

Software crypto methods (atcac_)

These methods provide a software implementation of various crypto algorithms.

atcac_
sw_ecdsa_verify_p256
sw_random
sw_sha1_init
sw_sha1_update
sw_sha1_finish
sw_sha1
sw_sha2_256_init

sw_sha2_256_update
sw_sha2_256_finish
sw_sha2_256
sha256_hmac_init
sha256_hmac_update
sha256_hmac_finish
sha256_hmac_counter

Usage of CryptoAuthLib

Certificate manipulation methods (atcacert_)

These methods provide convenient ways to perform certification I/O with CryptoAuth chips and perform certificate manipulation in memory.

atcacert_
read_device_loc
read_cert
write_cert
create_csr
create_csr_pem
get_response
read_subj_key_id
read_cert_size
date_enc
date_dec
date_enc_compcert
date_dec_compcert
date_get_max_date
date_enc_iso8601_sep
date_dec_iso8601_sep
date_enc_rfc5280_utc
date_dec_rfc5280_utc

date_enc_rfc5280_gen
date_dec_rfc5280_gen
date_enc_posix_uint32_be
date_dec_posix_uint32_be
date_enc_posix_uint32_le
date_dec_posix_uint32_le
get_device_locs
cert_build_start
cert_build_process
cert_build_finish
get_device_data
set_subj_public_key
get_subj_public_key
get_subj_key_id
set_signature
get_signature
set_issue_date
get_issue_date

set_expire_date
get_expire_date
set_signer_id
get_signer_id
set_cert_sn
gen_cert_sn
get_cert_sn
set_auth_key_id
set_auth_key_id_raw
get_auth_key_id
set_comp_cert
get_comp_cert
get_tbs
get_tbs_digest
set_cert_element
get_cert_element
get_key_id
merge_device_loc

is_device_loc_overlap
public_key_add_padding
public_key_remove_padding
transform_data
max_cert_size
der_enc_length
der_dec_length
der_adjust_length
der_enc_integer
der_dec_integer
der_enc_ecdsa_sig_value
der_dec_ecdsa_sig_value
verify_cert_hw
gen_challenge_hw
verify_response_hw
verify_cert_sw
gen_challenge_sw
verify_response_sw

Usage of CryptoAuthLib

JSON Web Token (JWT) methods (atca_jwt_)

These methods for signing and verifying JSON Web Token (JWT) tokens.

atca_jwt_
Init
add_claim_string
add_claim_numeric
Finalize
check_payload_start
verify

Usage of CryptoAuthLib

mbedTLS Wrapper methods (atca_mbedtls_)

These methods are for interfacing cryptoauthlib to mbedtls.

atca_mbedtls_
pk_init_ext
pk_init
cert_add
ecdh_slot_cb
ecdh_ioprot_cb

Usage of CryptoAuthLib

TNG(Trust&Go) API (tng_)

These methods provide some convenience functions (mostly around certificates) for TNG devices, which currently include ATECC608A-MAHTN-T.

tng_
map_get_device_cert_def
get_device_cert_def
get_device_pubkey
atcacert_max_device_cert_size
atcacert_read_device_cert
atcacert_device_public_key

atcacert_max_signer_cert_size
atcacert_read_signer_cert
atcacert_signer_public_key
atcacert_root_cert_size
atcacert_root_cert
atcacert_root_public_key

Usage of CryptoAuthLib

Attributes (pkcs11_, C_)

PKCS11 attributes support methods.

pkcs11_
attrib_fill
attrib_value
attrib_false
attrib_true
attrib_empty
cert_get_encoded
cert_get_type
cert_get_subject
cert_get_subject_key_id
cert_get_authority_key_id
cert_get_trusted_flag
cert_x509_write
config_init_private
config_init_public
config_init_cert
config_cert
config_key
config_remove_object
config_load_objects
config_load
find_init

find_continue
find_finish
find_get_attribute
get_lib_info
get_context
lock_context
unlock_context
init_check
init
deinit
key_write
key_generate_pair
key_derive
mech_get_list
mech_get_info
object_alloc
object_free
object_check
object_get_handle
object_get_name
object_get_class
object_get_type

object_get_destroyable
object_get_size
object_find
object_create
object_destroy
object_deinit
object_load_handle_info
os_create_mutex
os_destroy_mutex
os_lock_mutex
os_unlock_mutex
get_session_context
session_check
session_open
session_close
session_closeall
session_get_info
session_login
session_logout
signature_sign_init
signature_sign
signature_sign_continue

signature_sign_finish
signature_verify_init
signature_verify
signature_verify_continue
signature_verify_finish
slot_get_context
slot_initslots
slot_config
slot_init
slot_get_list
slot_get_info
token_init
token_get_access_type
token_get_writable
token_get_storage
token_get_info
token_random
token_convert_pin_to_key
token_set_pin
util_escape_string
util_convert_rv
util_memset

Usage of CryptoAuthLib

Attributes (pkcs11_, C_)

PKCS11 attributes support methods.

C_
Initialize
Finalize
GetInfo
GetFunctionList
GetSlotList
GetSlotInfo
GetTokenInfo
GetMechanismList
GetMechanismInfo
InitToken
InitPIN
SetPIN
OpenSession
CloseSession
CloseAllSessions
GetSessionInfo
GetOperationState

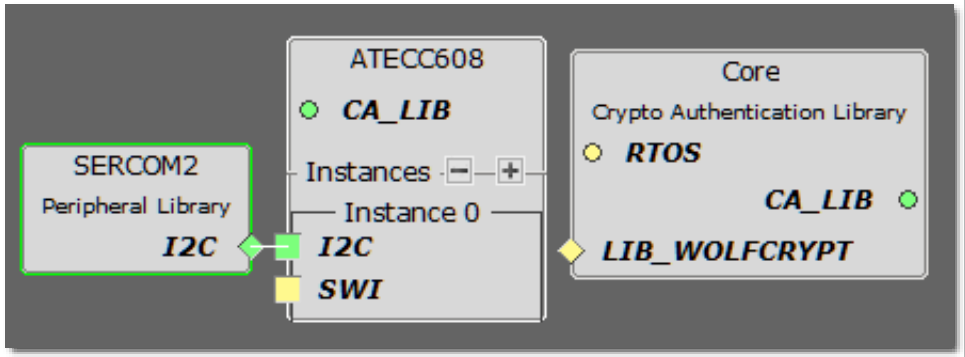
SetOperationState
Login
Logout
CreateObject
CopyObject
DestroyObject
GetObjectSize
GetAttributeValue
SetAttributeValue
FindObjectsInit
FindObjects
FindObjectsFinal
EncryptInit
Encrypt
EncryptUpdate
EncryptFinal
DecryptInit

Decrypt
DecryptUpdate
DecryptFinal
DigestInit
Digest
DigestUpdate
DigestKey
DigestFinal
SignInit
Sign
SignUpdate
SignFinal
SignRecoverInit
SignRecover
VerifyInit
Verify
VerifyUpdate

VerifyFinal
VerifyRecoverInit
VerifyRecover
DigestEncryptUpdate
DecryptDigestUpdate
SignEncryptUpdate
DecryptVerifyUpdate
GenerateKey
GenerateKeyPair
WrapKey
UnwrapKey
DeriveKey
SeedRandom
GenerateRandom
GetFunctionStatus
CancelFunction
WaitForSlotEvent

Usage of CryptoAuthLib

CryptoAuthLib HAL has completed by Harmony




```
hal_i2c_wait(atca_plib_i2c_api_t* plib, uint32_t rate, …)
hal_i2c_init(void *hal, ATCAIfaceCfg *cfg)
hal_i2c_post_init(ATCAIface iface)
hal_i2c_send(ATCAIface iface, uint8_t address, …)
hal_i2c_receive(ATCAIface iface, uint8_t address,…)
change_i2c_speed(ATCAIface iface, uint32_t speed)
hal_i2c_control(ATCAIface iface, uint8_t option, …)
hal_i2c_release(void *hal_data)
hal_i2c_discover_buses(int i2c_buses[], int max_buses)
hal_i2c_discover_devices(int bus_num, ATCAIfaceCfg cfg[], …)
```

hal_i2c_harmony.c

hal_harmony_init.c

ATECC608_0.c

```
ATCAIfaceCfg atecc608_0_init_data = {
    .iface_type             = ATCA_I2C_IFACE,
    .devtype                = ATECC608,
    .atcai2c.address        = 0xC0,
    .atcai2c.bus            = 0,
    .atcai2c.baud           = 100000,
    .wake_delay             = 1500,
    .rx_retries             = 20,
    .cfg_data               = &sercom2_plib_i2c_api
};
```

```
atca_plib_i2c_api_t sercom2_plib_i2c_api = {
    .read = SERCOM2_I2C_Read,
    .write = SERCOM2_I2C_Write,
    .is_busy = SERCOM2_I2C_IsBusy,
    .error_get = SERCOM2_I2C_ErrorGet,
    .transfer_setup = SERCOM2_I2C_TransferSetup
};
```

HAL will depend to what device you to connect in Harmony

MICROCHIP

🛠 Lab4
Initialize CryptoAuthLib and ECC chip

Usage of CryptoAuthLib

MICROCHIP

🛠 Lab4 Initialize CryptoAuthLib and ECC608 Chip

Implement your first application of CryptoAuthLib

main.c

Source History

```
27  #include <stdlib.h>                     // Defines EXIT_FAILURE
28  #include "definitions.h"                // SYS function prototypes
29  // TODO 4.01
30  #include "cryptoauthlib.h"
31
32  extern ATCAIfaceCfg atecc608_0_init_data;
33  ATCA_STATUS status;
34
35  // TODO 4.02
36  #define CHECK_STATUS( status )\
37  {\
38      if(status != ATCA_SUCCESS)\
39      {\
40          SYS_CMD_PRINT("Error: Line %d in %s\r\n", __LINE__, __FILE__);\
41      }\
42  }\
```

Define inline macro for Error Line number output.
The '\' must be added for every newline in macro.

MICROCHIP

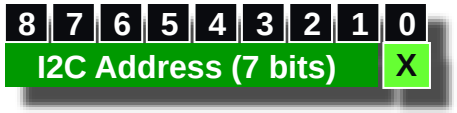
🛠 Lab4 Initialize CryptoAuthLib and ECC608 Chip

Implement your first application of CryptoAuthLib

main.c

Source | History

```
64      // TODO 4.03
65      void CommandInit(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
66      {
67          if( argc>1 )
68          {
69              unsigned char SlaveAddr;
70              SlaveAddr = (unsigned char)strtol(argv[1], NULL, 16)<<1;
71              atecc608_0_init_data.atcai2c.address = SlaveAddr;
72              SYS_CMD_PRINT("\r\nInitial CryptoAuthLib:\r\n");
73              SYS_CMD_PRINT("- i2caddr Byte = %X\r\n", SlaveAddr);
74              status = atcab_init(&atecc608_0_init_data);
75              CHECK_STATUS(status);
76          }
77          else
78          {
79              SYS_CMD_PRINT("\r\nNo target I2C addess of device !!\r\n");
80          }
81      }
82
83    // TODO 3.02
84    static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
```



MICROCHIP

🛠 Lab4 Initialize CryptoAuthLib and ECC608 Chip

Implement your first application of CryptoAuthLib

main.c ×

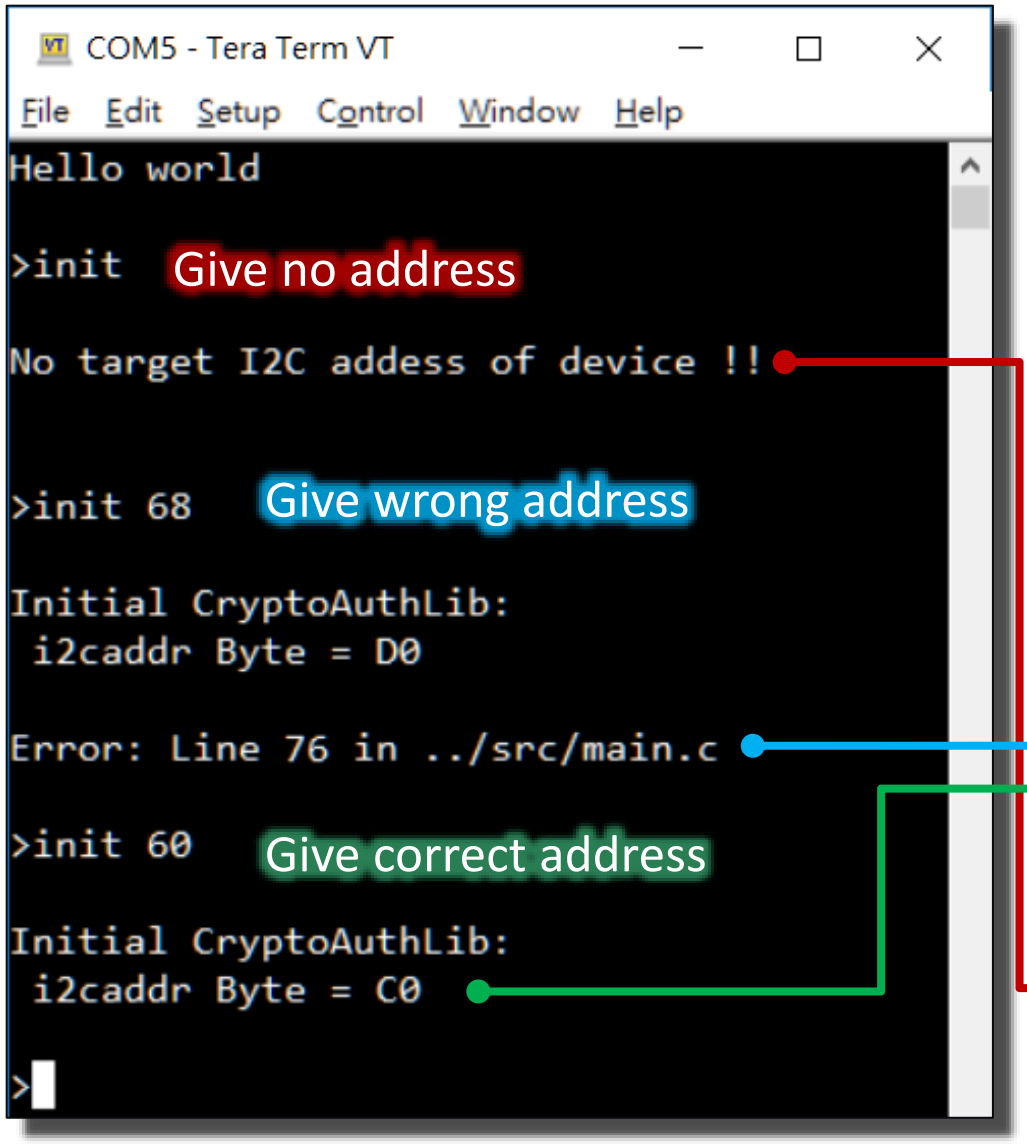
Source | History

```
63
64    // TODO 4.03
65    void CommandInit(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
66  ⊞ {...14 lines }
80
81    // TODO 3.02
82    static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
83    {
84        { "hello", CommandHello, ": Hello!" },
85    // TODO 4.04
86        { "init",  CommandInit, ": Initial CryptoAuthLib" },
87    };
88
89    int main ( void )
90    {
91        /* Initialize all modules */
```

MICROCHIP

🛠 Lab4 Initialize CryptoAuthLib and ECC608 Chip

Result of your first application of CryptoAuthLib




```
// TODO 4.02
#define CHECK_STATUS( status )\
{\
    if(status != ATCA_SUCCESS)\
    {\
        SYS_CMD_PRINT("Error: Line %d in %s\r\n", __LINE__, __FILE__);\
    }\
}
```

```
// TODO 4.03
void CommandInit(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    if( argc>1 )
    {
        unsigned char SlaveAddr;
        SlaveAddr = (unsigned char)strtol(argv[1], NULL, 16)<<1;
        atecc608_0_init_data.atcai2c.address = SlaveAddr;
        SYS_CMD_PRINT("\r\nInitial CryptoAuthLib:\r\n");
        SYS_CMD_PRINT("- i2caddr Byte = %X\r\n", SlaveAddr);
        status = atcab_init(&atecc608_0_init_data);
        CHECK_STATUS(status);
    }
    else
        SYS_CMD_PRINT("\r\nNo target I2C addess of device !!\r\n");
}
```

MICROCHIP

🛠 Lab5
Get Chip Revision and Unique Serial Number

Usage of CryptoAuthLib

MICROCHIP

🛠 Lab5 Get Chip Revision and Unique Serial Number

Read Chip Revision and Unique Serial Number from ECC608

main.c ×

Source History

```
29  // TODO 4.01
30  #include "cryptoauthlib.h"
31
32  extern ATCAIfaceCfg atecc608_0_init_data;
33  ATCA_STATUS status;
34  // TODO 5.01
35  uint8_t ecc_version[ATCA_WORD_SIZE];
36  uint8_t serial_number[ATCA_SERIAL_NUM_SIZE];
37
38  // TODO 4.02
39  #define CHECK_STATUS( status )\
40  {\
41      if(status != ATCA_SUCCESS )\
42      {\
43          SYS_CMD_PRINT("Error: Line %d in %s\r\n", __LINE__, __FILE__);\
44      }\
```

🛠 Lab5 Get Chip Revision and Unique Serial Number

Read Chip Revision and Unique Serial Number from ECC608

main.c

Source History

```
34  // TODO 5.01
35  uint8_t ecc_version[ATCA_WORD_SIZE];
36  uint8_t serial_number[ATCA_SERIAL_NUM_SIZE];
37
38  // TODO 5.02
39  void OutputData(char* label, uint8_t *buffer, size_t len)
40  {
41      SYS_CMD_PRINT("%s", label);
42      for( int i=0 ; i<len ; i++ )
43      {
44          SYS_CMD_PRINT("%02X ", buffer[i]);
45          if( (i+1)%16==0 )
46          {
47              SYS_CMD_PRINT("\r\n");
48          }
49      }
50      SYS_CMD_PRINT("\r\n");
51  }
52
```

Print Data Buffer with Label

MICROCHIP

🛠 Lab5 Get Chip Revision and Unique Serial Number

Read Chip Revision and Unique Serial Number from ECC608

main.c

Source History

```
100   // TODO 5.03
101   void CommandReadSN(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
102   {
103       status = atcab_info((uint8_t*)&ecc_version);
104       CHECK_STATUS(status);
105       OutputData( "\r\nECC version:\r\n", ecc_version, ATCA_WORD_SIZE);
106
107       status = atcab_read_serial_number((uint8_t*)&serial_number);
108       CHECK_STATUS(status);
109       OutputData( "\r\nSerial Number:\r\n", serial_number, ATCA_SERIAL_NUM_SIZE);
110   }
111
112   // TODO 3.02
113   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
114   {
115       { "hello", CommandHello, ": Hello!" },
```

MICROCHIP

🛠 Lab5 Get Chip Revision and Unique Serial Number

Read Chip Revision and Unique Serial Number from ECC608

main.c

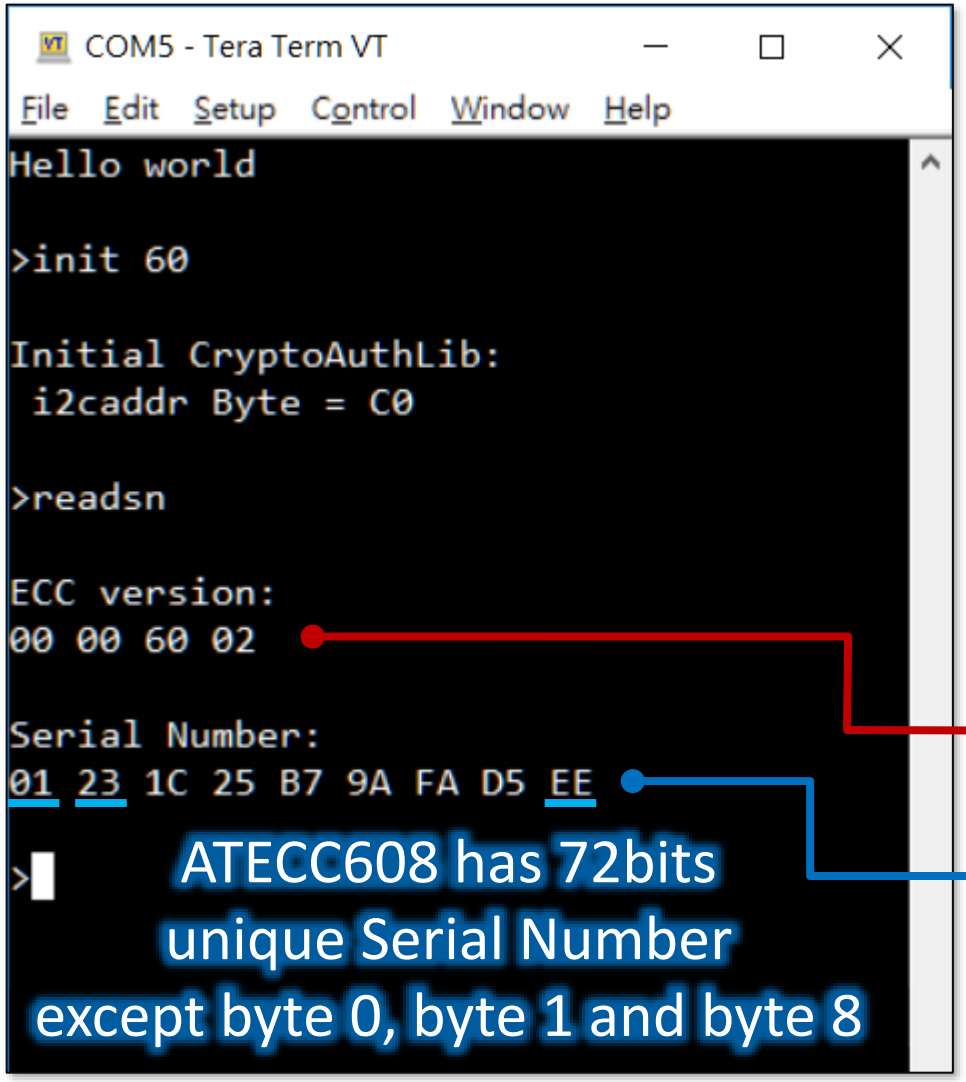
Source History

```
100   // TODO 5.03
101   void CommandReadSN(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
102   {...9 lines }
111
112   // TODO 3.02
113   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
114   {
115       { "hello", CommandHello, ": Hello!" },
116   // TODO 4.04
117       { "init",  CommandInit, ": Initial CryptoAuthLib" },
118   // TODO 5.04
119       { "readsn", CommandReadSN, ": READ Version and S/N number" },
120   };
121
122   int main ( void )
123   {
```

MICROCHIP

🛠 Lab5 Get Chip Revision and Unique Serial Number

Result of Chip Revision and Unique Serial Number from ECC608




```
// TODO 5.02
void OutputData(char* label, uint8_t* buffer, size_t len)
{
    SYS_CMD_PRINT("%s", label);
    for( int i=0 ; i<len ; i++ )
    {
        SYS_CMD_PRINT("%02X ", buffer[i]);
        if( (i+1)%16==0 )
        {
            SYS_CMD_PRINT("\r\n");
        }
    }
    SYS_CMD_PRINT("\r\n");
}

// TODO 5.03
void CommandReadSN(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    status = atcab_info((uint8_t*)&ecc_version);
    CHECK_STATUS(status);
    OutputData( "\r\nECC version:\r\n", ecc_version, ATCA_WORD_SIZE);

    status = atcab_read_serial_number((uint8_t*)&serial_number);
    CHECK_STATUS(status);
    OutputData( "\r\nSerial Number:\r\n", serial_number, ATCA_SERIAL_NUM_SIZE);
}
```

MICROCHIP

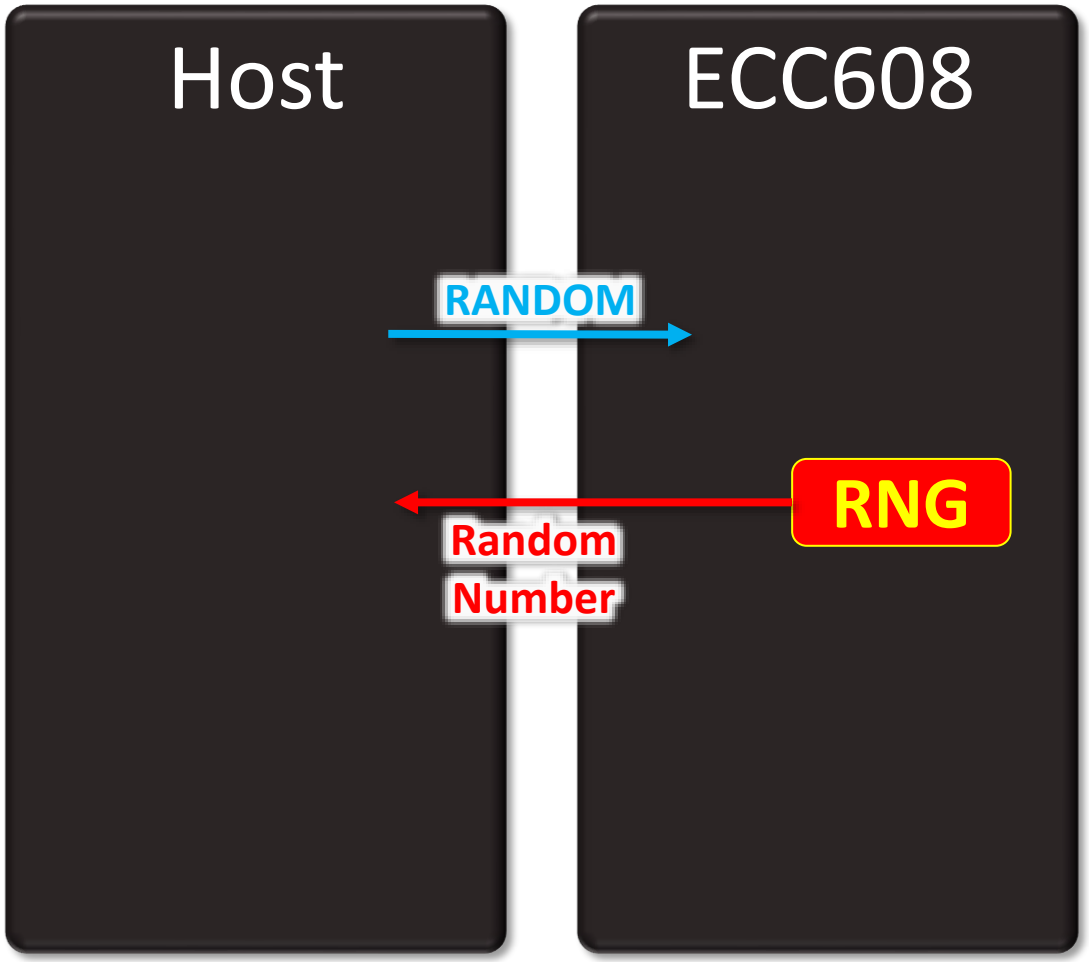
🛠 Lab6
Random and Nonce command

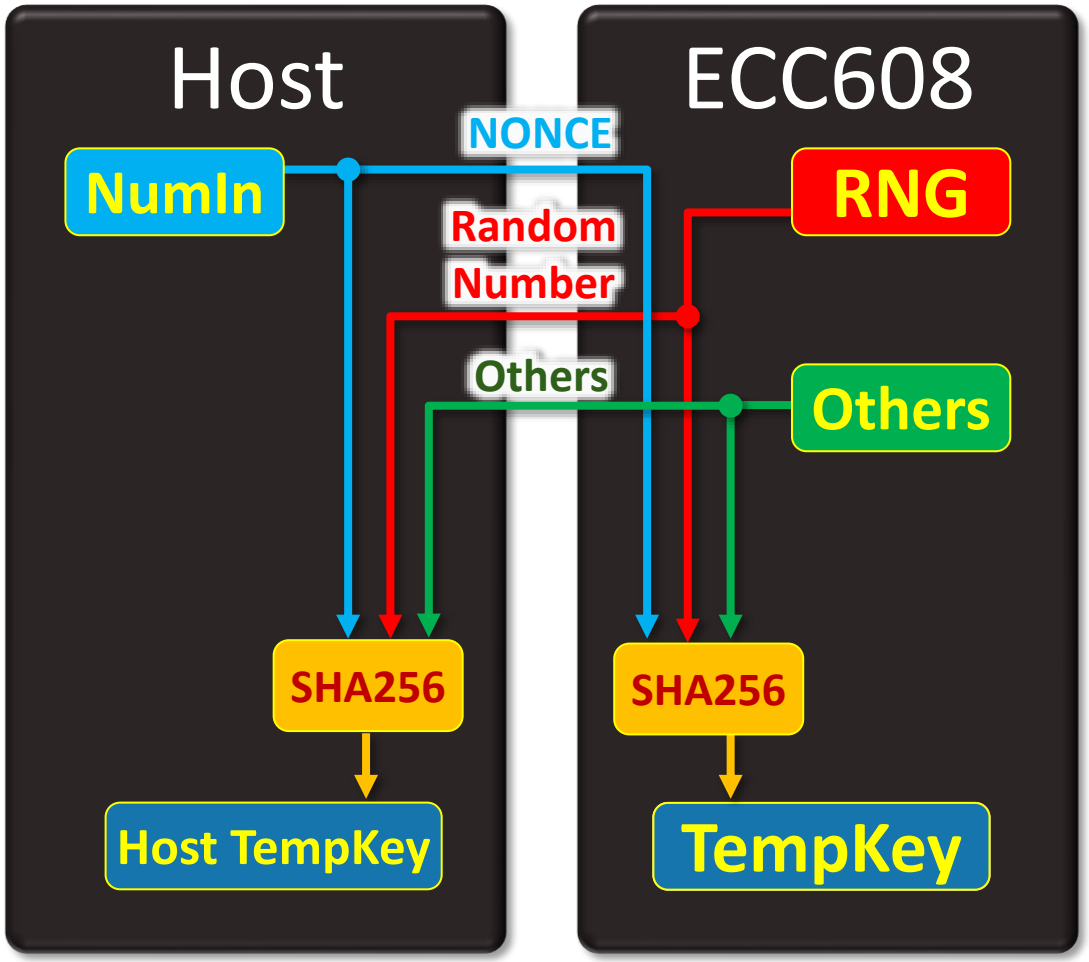
Usage of CryptoAuthLib

MICROCHIP

🛠 Lab6 Random and Nonce Command

Develop Random and Nonce command





🛠 Lab6 Random and Nonce Command

Develop Random and Nonce command

main.c ×

Source | History

```
29  // TODO 4.01
30  #include "cryptoauthlib.h"
31  // TODO 6.01
32  #include "host/atca_host.h"
33
34  extern ATCAIfaceCfg atecc608_0_init_data;
35  ATCA_STATUS status;
36  // TODO 5.01
37  uint8_t ecc_version[ATCA_WORD_SIZE];
38  uint8_t serial_number[ATCA_SERIAL_NUM_SIZE];
39  // TODO 6.02
40  uint8_t rand_out[32];
41  atca_nonce_in_out_t host_nonce;
42  atca_temp_key_t host_tempkey;
43  const uint8_t num_in[] = {
44        0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, \
45        0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x20 };
46
47  // TODO 5.02
```

MICROCHIP

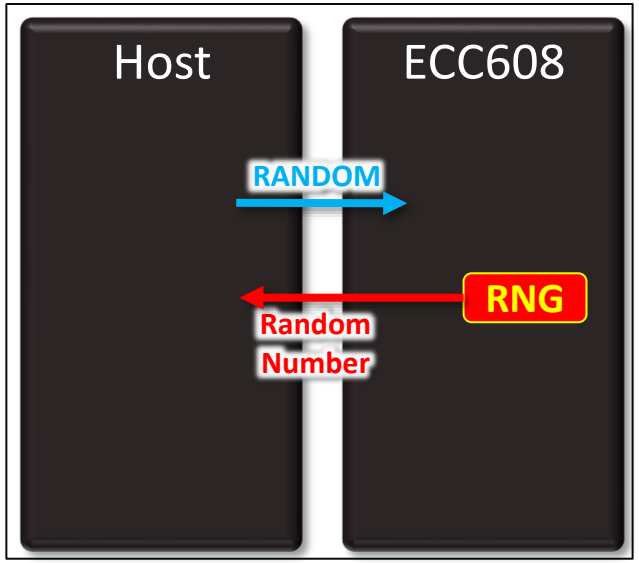
🛠 Lab6 Random and Nonce Command

Develop Random and Nonce command

main.c

Source | History

```
109  // TODO 5.03
110  void CommandReadSN(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
111  {...9 lines }
120
121  // TODO 6.03
122  void CommandRandom(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
123  {
124      status = atcab_random(rand_out);
125      CHECK_STATUS(status);
126      SYS_CMD_PRINT("\r\nRANDOM Command:\r\n");
127      OutputData("- Random number out:\r\n", rand_out, 32);
128  }
129
130  // TODO 3.02
131  static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
132  {
```



MICROCHIP

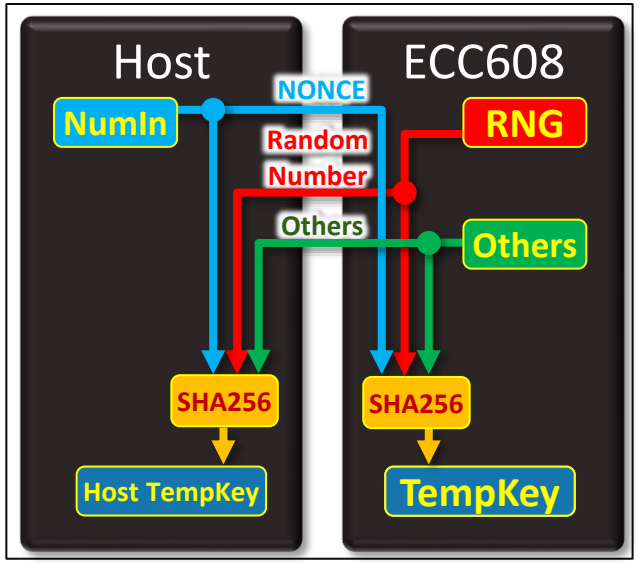
🛠 Lab6 Random and Nonce Command

Develop Random and Nonce command

main.c

Source | History

```
130   // TODO 6.04
131   void CommandNonce(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
132   {
133       status = atcab_nonce_base(NONCE_MODE_SEED_UPDATE, 0, num_in, rand_out);
134       CHECK_STATUS(status);
135       SYS_CMD_PRINT ("\r\nNONCE Command:\r\n");
136       OutputData("- Random number response:\r\n", rand_out, 32);
137
138       host_nonce.mode = NONCE_MODE_SEED_UPDATE;
139       host_nonce.zero = 0;
140       host_nonce.num_in = num_in;
141       host_nonce.rand_out = rand_out;
142       host_nonce.temp_key = &host_tempkey;
143       status = atcah_nonce(&host_nonce);
144       CHECK_STATUS(status);
145       OutputData("- Host Tempkey:\r\n", host_tempkey.value, 32);
146   }
147
148   // TODO 3.02
149   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
```



MICROCHIP

🛠 Lab6 Random and Nonce Command

Develop Random and Nonce command

main.c

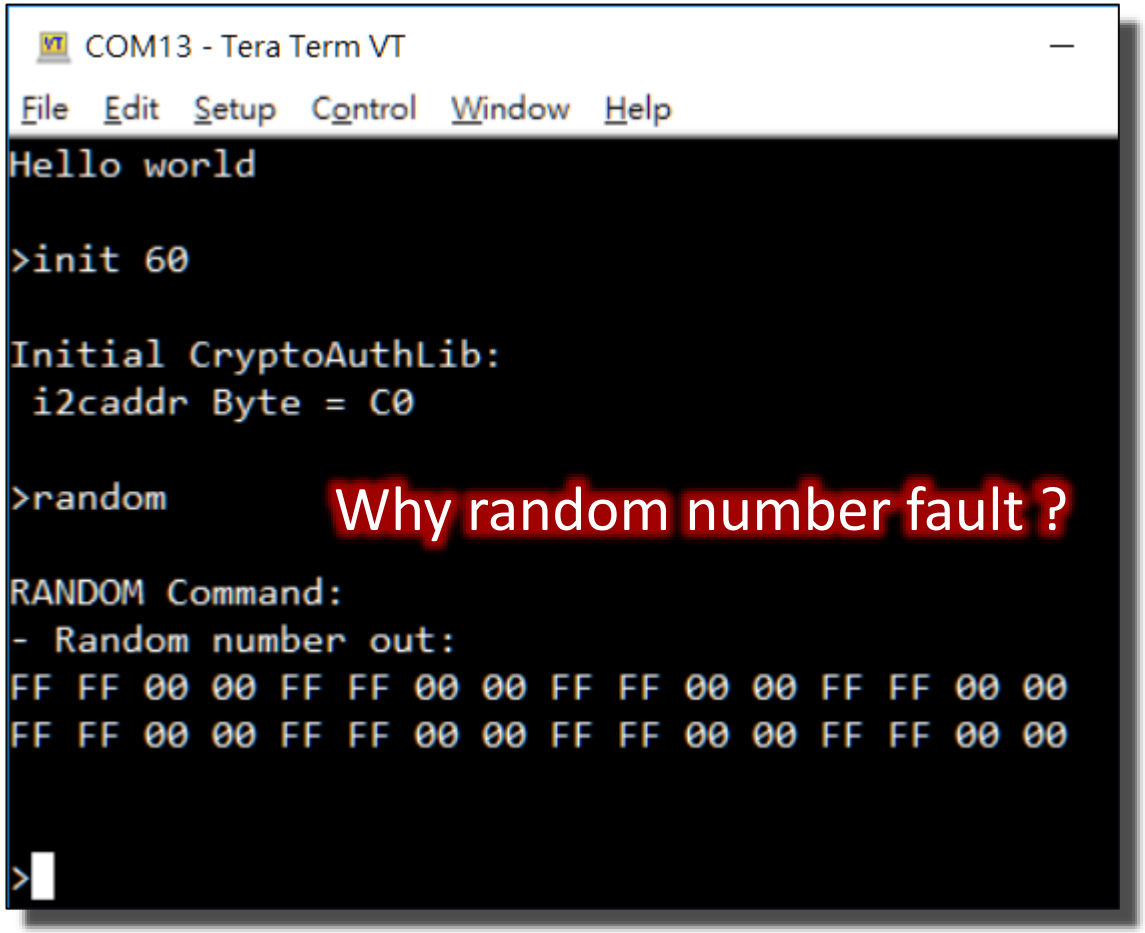
Source | History

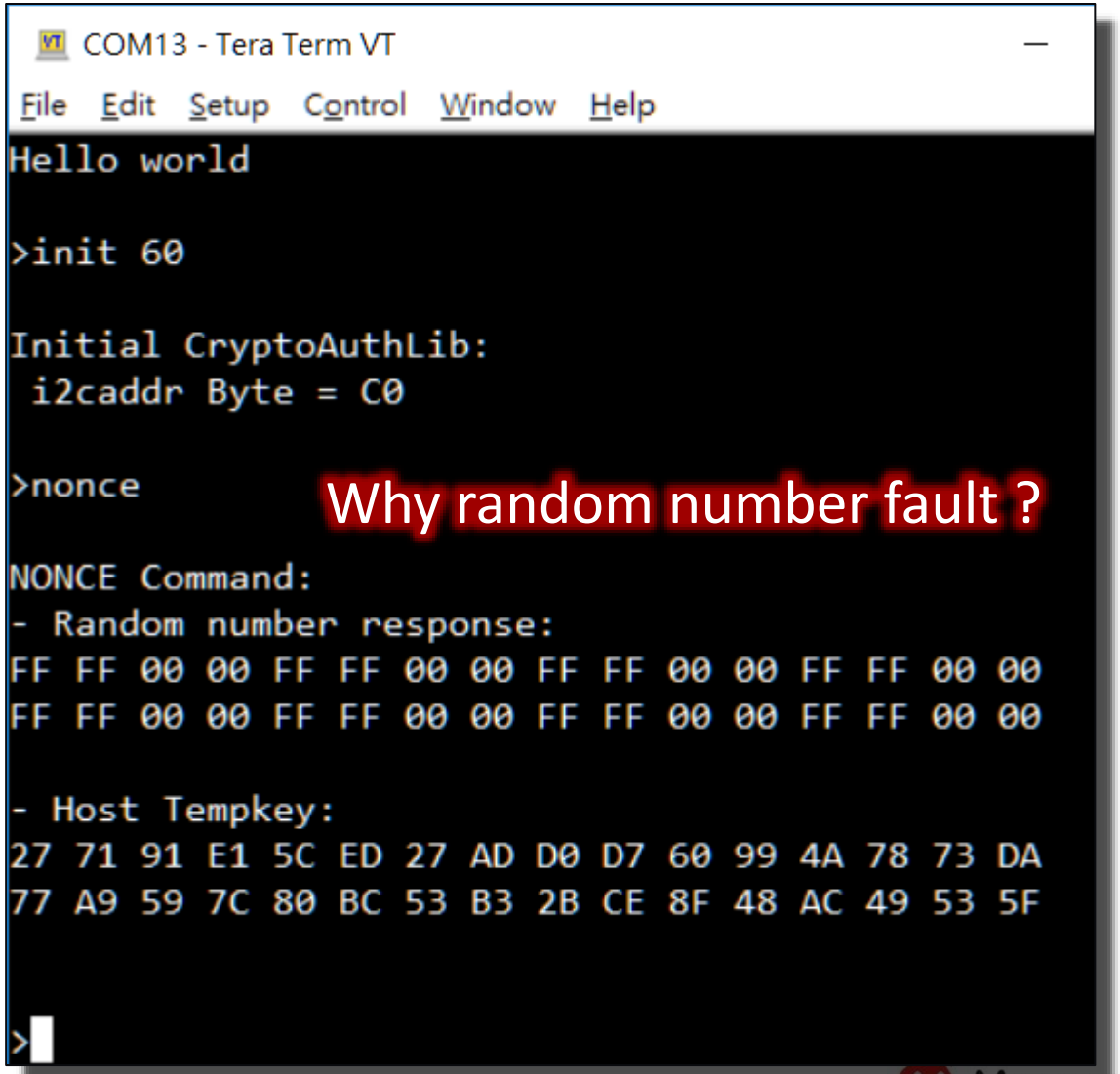
```
130  // TODO 6.04
131  void CommandNonce(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
132  {...15 lines }
147
148  // TODO 3.02
149  static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
150  {
151      { "hello", CommandHello, ": Hello!" },
152  // TODO 4.04
153      { "init",  CommandInit, ": Initial CryptoAuthLib" },
154  // TODO 5.04
155      { "readsn", CommandReadSN, ": READ Version and S/N number" },
156  // TODO 6.05
157      { "random", CommandRandom, ": RANDOM command" },
158      { "nonce", CommandNonce, ": NONCE command" },
159  };
```

MICROCHIP

🛠 Lab6 Random and Nonce Command

Develop Random and Nonce command





🛠 Lab7
Chip Provision

Usage of CryptoAuthLib

MICROCHIP

🛠 Lab7 Chip Provision

Develop a Provision command

main.c

Source | History

```
43  const uint8_t num_in[] = {
44      0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, \
45      0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x20 };
46  // TODO 7.01
47  uint8_t test_ecc608_configdata[ATCA_ECC_CONFIG_SIZE] = {
48      0x01, 0x23, 0x00, 0x00, 0x00, 0x00, 0x60, 0x00, 0x04, 0x05, 0x06, 0x07, 0xEE, 0x01, 0x01, 0x00,
49      0xC0, 0x00, 0xA1, 0x00, 0xAF, 0x2F, 0xC4, 0x44, 0x87, 0x20, 0xC4, 0xF4, 0x8F, 0x0F, 0x0F, 0x0F,
50      0x9F, 0x8F, 0x83, 0x64, 0xC4, 0x44, 0xC4, 0x64, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F,
51      0x0F, 0x0F, 0x0F, 0x0F, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF,
52      0x00, 0x00, 0x00, 0x00, 0xFF, 0x84, 0x03, 0xBC, 0x09, 0x69, 0x76, 0x00, 0x00, 0x00, 0x00, 0x00,
53      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x0E, 0x40, 0x00, 0x00, 0x00, 0x00,
54      0x33, 0x00, 0x1C, 0x00, 0x13, 0x00, 0x1C, 0x00, 0x3C, 0x00, 0x3E, 0x00, 0x1C, 0x00, 0x33, 0x00,
55      0x1C, 0x00, 0x1C, 0x00, 0x38, 0x10, 0x30, 0x00, 0x3C, 0x00, 0x3C, 0x00, 0x32, 0x00, 0x30, 0x00
56  };
57  const uint8_t g_slot4_key[] = {
58      0x37, 0x80, 0xe6, 0x3d, 0x49, 0x68, 0xad, 0xe5,
59      0xd8, 0x22, 0xc0, 0x13, 0xfc, 0xc3, 0x23, 0x84,
60      0x5d, 0x1b, 0x56, 0x9f, 0xe7, 0x05, 0xb6, 0x00,
61      0x06, 0xfe, 0xec, 0x14, 0x5a, 0x0d, 0xb1, 0xe3
62  };
```

🛠 Lab7 Chip Provision

Develop a Provision command

main.c

Source | History

```
165     // TODO 7.02
166     void CommandProvision(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
167     {
168             status = atcab_write_config_zone(test_ecc608_configdata);
169             CHECK_STATUS(status);
170
171             status = atcab_lock_config_zone();
172             CHECK_STATUS(status);
173
174            status = atcab_write_zone(ATCA_ZONE_DATA, 4, 0, 0, g_slot4_key, 32 );
175             CHECK_STATUS(status);
176
177             status = atcab_lock_data_zone();
178             CHECK_STATUS(status);
179
180             printf("\r\nProvision done!!\r\n");
181     }
182
183    // TODO 3.02
184    static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
```

Write Config

Lock Config Zone

Write Data

Lock Data Zone

MICROCHIP

🛠 Lab7 Chip Provision

Develop a Provision command

main.c

Source | History

```
165   // TODO 7.02
166   void CommandProvision(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
167   {...15 lines }
182
183   // TODO 3.02
184   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
185   {
186       { "hello", CommandHello, ": Hello!" },
187   // TODO 4.04
188       { "init",  CommandInit, ": Initial CryptoAuthLib" },
189   // TODO 5.04
190       { "readsn", CommandReadSN, ": READ Version and S/N number" },
191   // TODO 6.05
192       { "random", CommandRandom, ": RANDOM command" },
193       { "nonce", CommandNonce, ": NONCE command" },
194   // TODO 7.03
195       { "provision", CommandProvision, ": Provision new chip" },
196   };
```

MICROCHIP

🛠 Lab7 Chip Provision

Result of Provision command

COM13 - Tera Term VT

File Edit Setup Control Window Help

```
>init 60

Initial CryptoAut
 i2caddr Byte = C

>provision

Provision done!!

>
```

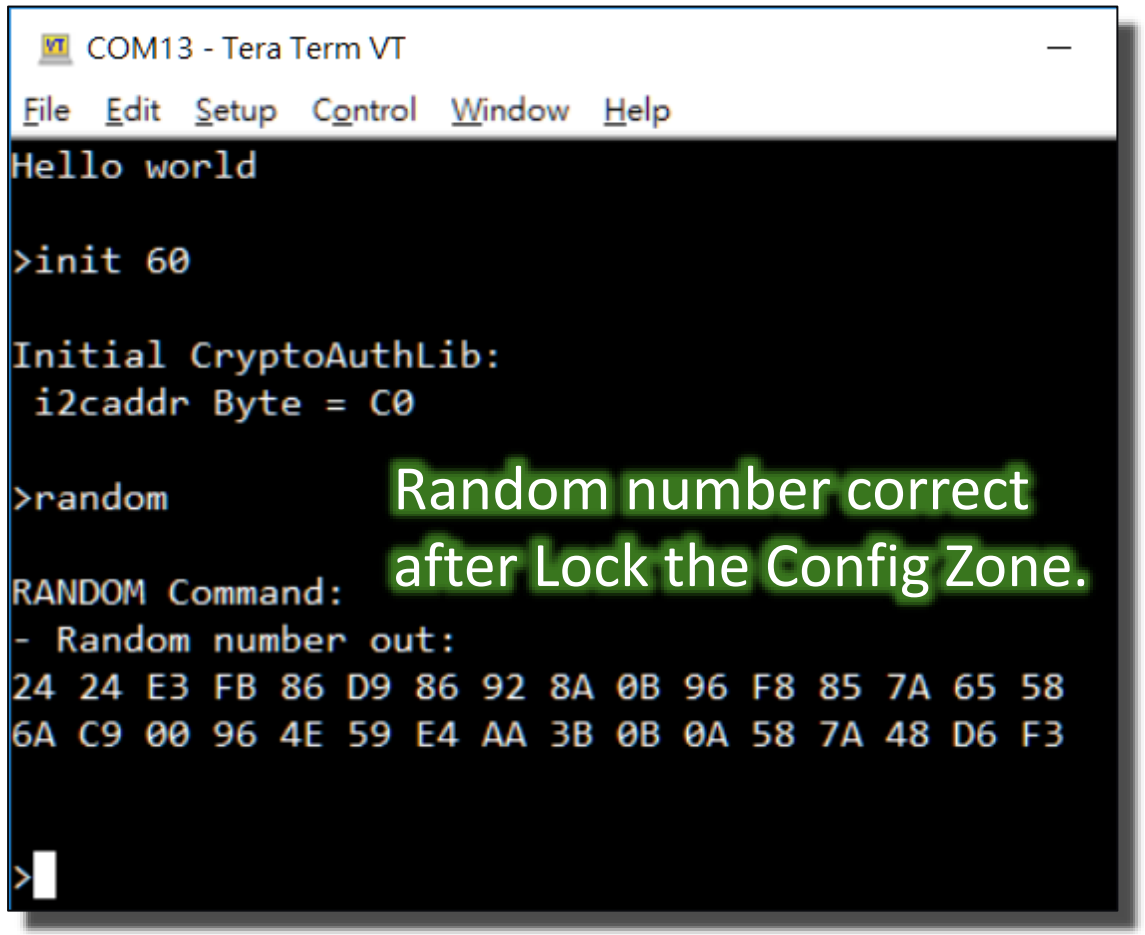
Slot	Description	Type	Lockable	ReqRandom	ReqAuth	AuthKey	PersistentDisable	NoMac	LimitedUse	GenPubKey	ExternalSigs	InternalSigs	ECDH	WriteMasterSecret	GenKey	PrivWrite	ReqValidate	X509id	ReadMode	ReadKey	WriteMode	WriteKey	DeriveKey	DeriveKeyParent	DeriveKeyMAC	SlotConfig	KeyConfig
0		ECC Private Key	☑	☐	☐	0	☐	☐	☑	☑	☑	☑	☑	☑	☑	☐	☑	0	Never	15	Never	15	☑	☐	☐	0x2FAF	0x0033
1		Data	☐	☐	☐	0	☐	☐	☐	☐	☐	☐	☑	☐	☐	☑	☐	0	Encrypted	4	Encrypt	4	☐	☐	☐	0x44C4	0x001C
2		ECC Private Key	☐	☐	☐	0	☐	☐	☐	☑	☑	☑	☑	☐	☑	☐	☑	0	Never	7	Never	0	☑	☐	☐	0x2087	0x0013
3		Data	☐	☐	☐	0	☐	☐	☐	☐	☐	☐	☑	☐	☐	☑	☐	0	Encrypted	4	Encrypt	4	☑	☑	☑	0xF4C4	0x001C
4		Data	☑	☐	☐	0	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐	☐	0	Never	15	Always	15	☐	☐	☐	0x0F8F	0x003C
5		Data	☑	☐	☐	0	☐	☐	☐	☑	☑	☑	☑	☐	☐	☑	☐	0	Clear	15	Always	15	☐	☐	☐	0x0F0F	0x003E
6		Data	☐	☐	☐	0	☐	☑	☐	☐	☑	☑	☑	☐	☐	☐	☐	0	Never	15	Never	15	☐	☐	☑	0x8F9F	0x001C
7		ECC Private Key	☑	☐	☐	0	☐	☐	☐	☑	☑	☑	☐	☐	☑	☑	☐	0	Never	3	Encrypt	4	☑	☐	☐	0x6483	0x0033
8		Data	☐	☐	☐	0	☐	☐	☐	☐	☐	☐	☑	☐	☐	☑	☐	0	Encrypted	4	Encrypt	4	☐	☐	☐	0x44C4	0x001C
9		Data	☐	☐	☐	0	☐	☐	☐	☐	☐	☐	☑	☐	☐	☑	☐	0	Encrypted	4	Encrypt	4	☑	☐	☐	0x64C4	0x001C
10		AES Key	☑	☐	☐	0	☑	☐	☐	☐	☑	☑	☑	☐	☐	☐	☐	0	Clear	15	Always	15	☐	☐	☐	0x0F0F	0x1038
11		ECC Public Key	☑	☐	☐	0	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐	☐	0	Clear	15	Always	15	☐	☐	☐	0x0F0F	0x0030
12		Data	☑	☐	☐	0	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐	☐	0	Clear	15	Always	15	☐	☐	☐	0x0F0F	0x003C
13		Data	☑	☐	☐	0	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐	☐	0	Clear	15	Always	15	☐	☐	☐	0x0F0F	0x003C
14		ECC Public Key	☑	☐	☐	0	☐	☐	☐	☑	☑	☑	☑	☑	☐	☐	☑	0	Clear	15	Always	15	☐	☐	☐	0x0F0F	0x0032
15		ECC Public Key	☑	☐	☐	0	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐	☐	0	Clear	15	Always	15	☐	☐	☐	0x0F0F	0x0030

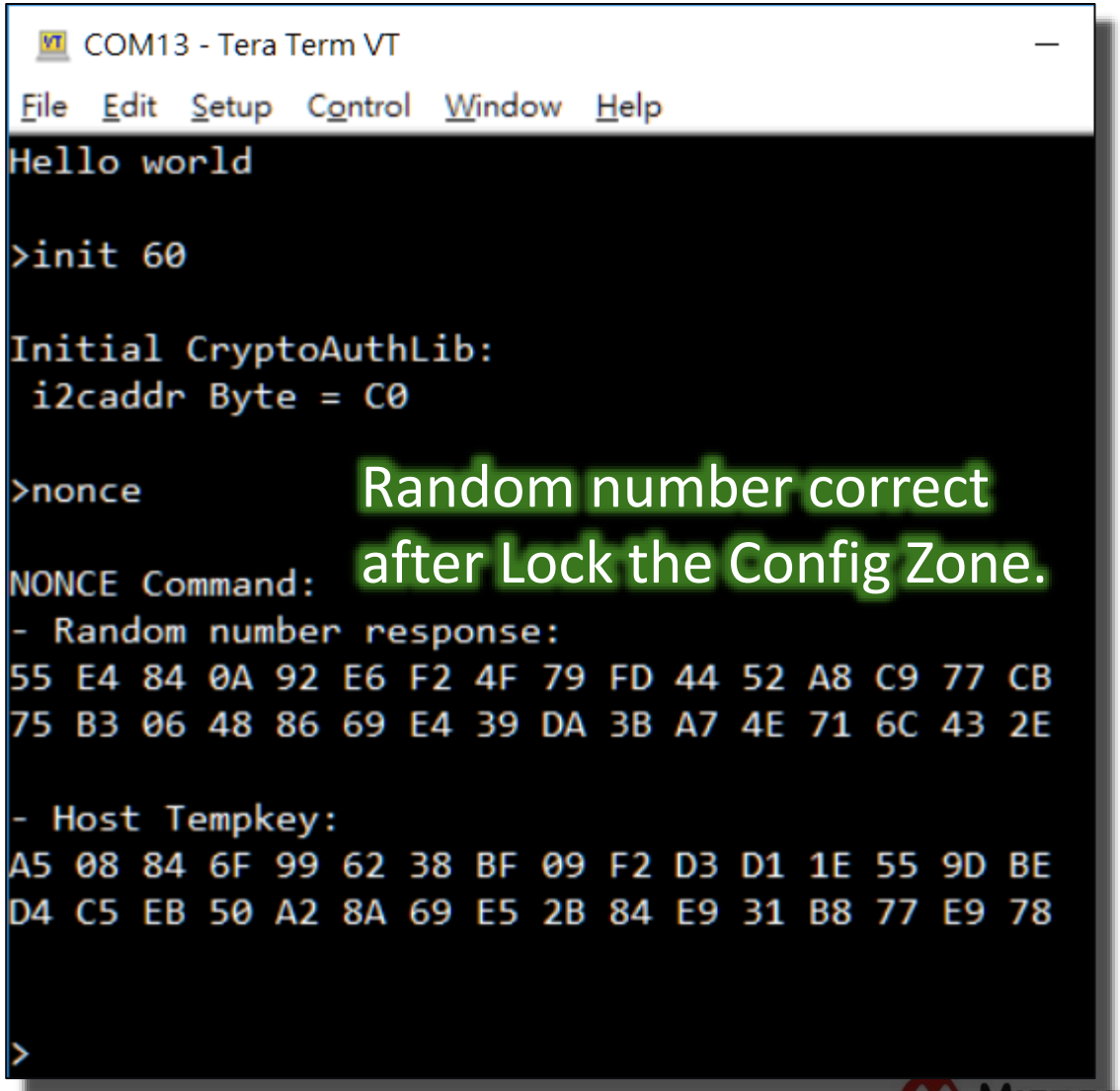
Config Value Format: ◉ Value (BE) ○ Data (LE)

MICROCHIP

🛠 Lab7 Chip Provision

Check Random and Nonce Command after Provision



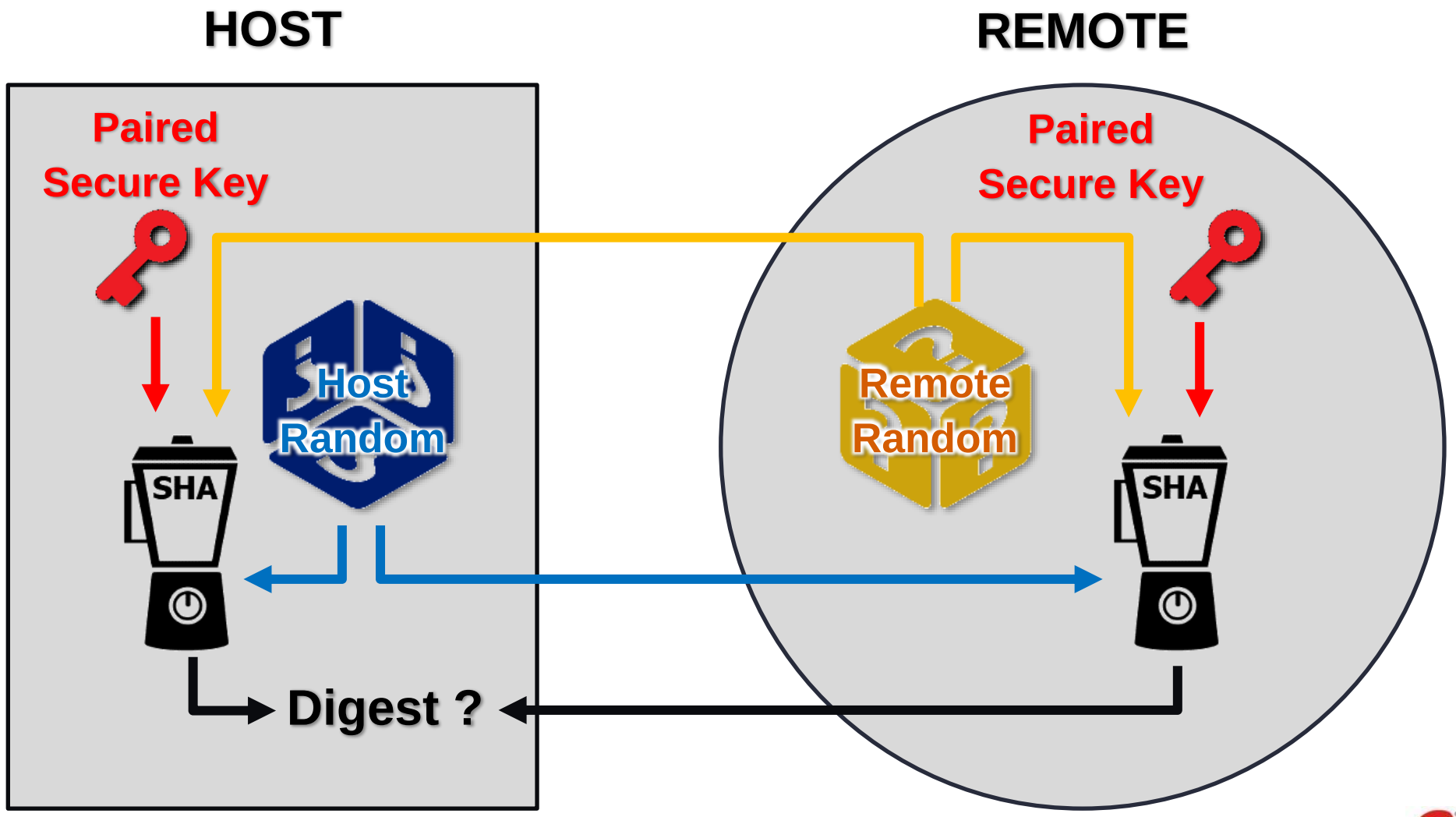


Symmetric Authentication

Paired key authentication

Symmetric Authentication

MAC (Message Authentication Code)



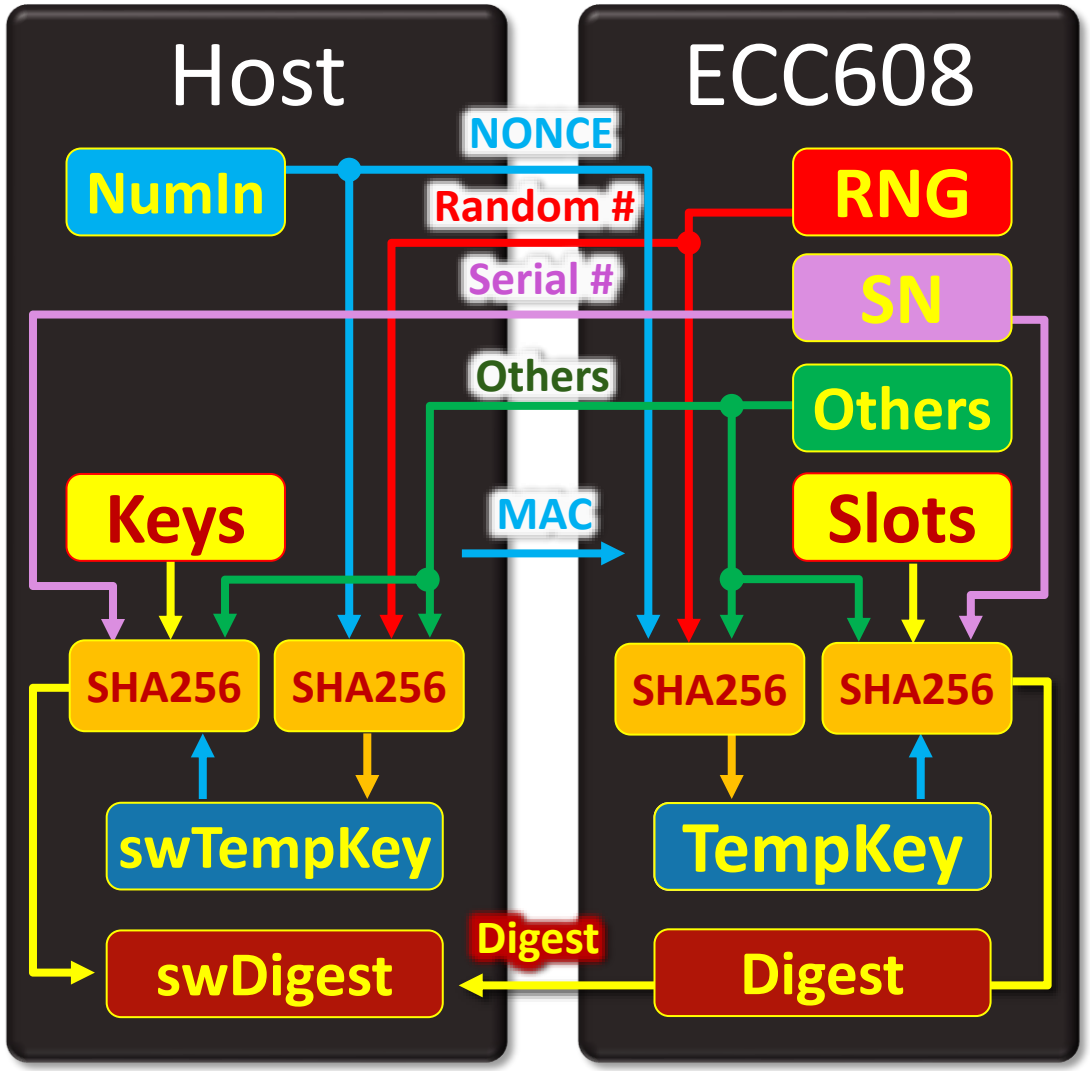
🛠 Lab8
MAC (Message Authentication Code)

Symmetric Authentication

MICROCHIP

🛠 Lab8 MAC (Message Authentication Code)

Develop MAC (Message Authentication Code)



MICROCHIP

🛠 Lab8 MAC (Message Authentication Code)

Develop MAC (Message Authentication Code)

main.c

Source History

```
57  const uint8_t g_slot4_key[] = {
58      0x37, 0x80, 0xe6, 0x3d, 0x49, 0x68, 0xad, 0xe5,
59      0xd8, 0x22, 0xc0, 0x13, 0xfc, 0xc3, 0x23, 0x84,
60      0x5d, 0x1b, 0x56, 0x9f, 0xe7, 0x05, 0xb6, 0x00,
61      0x06, 0xfe, 0xec, 0x14, 0x5a, 0x0d, 0xb1, 0xe3
62  };
63  // TODO 8.01
64  atca_mac_in_out_t host_mac;
65  uint16_t key_id;
66  uint8_t digest[32];
67  uint8_t response[32];
68
69  // TODO 5.02
70  void OutputData(char* label, uint8_t* buffer, size_t len)
71  {
72      SYS_CMD_PRINT("%s", label);
```

🛠 Lab8 MAC (Message Authentication Code)

Develop MAC (Message Authentication Code)

main.c

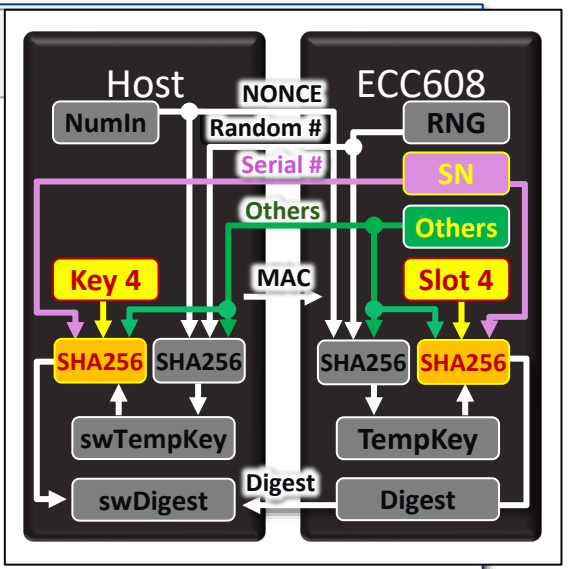
Source | History

```
170  // TODO 7.02
171  void CommandProvision(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
172  {...15 lines }
187
188  // TODO 8.02
189  void CommandMac(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
190  {
191      if( argc>1 )
192      {
193          key_id  = (unsigned char)strtol(argv[1], NULL, 16);
194          status = atcab_mac(MAC_MODE_BLOCK2_TEMPKEY|MAC_MODE_INCLUDE_SN, key_id, NULL, digest);
195          CHECK_STATUS(status);
196          OutputData("\r\nMAC Command:\r\n- Digest out :\r\n", digest, 32);
197          host_mac.mode = MAC_MODE_BLOCK2_TEMPKEY|MAC_MODE_INCLUDE_SN;
198          host_mac.key_id = key_id;
199          host_mac.challenge = NULL;
200          host_mac.key = g_slot4_key;
201          host_mac.otp = NULL;
```

SN include in MAC Digest Calculate

Host side Other Data

Host side Key 4



MICROCHIP

🛠 Lab8 MAC (Message Authentication Code)

Develop MAC (Message Authentication Code)

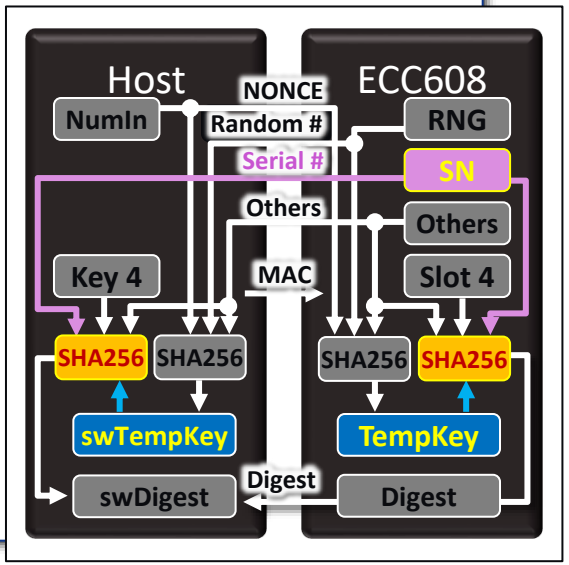
main.c

Source | History

```
200             host_mac.key = g_slot4_key;
201             host_mac.otp = NULL;
202             host_mac.sn = serial_number;
203             host_mac.response = response;
204             host_mac.temp_key = &host_tempkey;
205             status = atcah_mac(&host_mac);
206             CHECK_STATUS(status);
207             OutputData("- Digest out (Host claculated) :\r\n", response, 32);
208         }
209         else
210         {
211             SYS_CMD_PRINT("\r\nNo target key slot !!\r\n");
212         }
213     }
214
215     // TODO 3.02
216     static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
217     {
```

Host side SN include for Digest calculate

Host side TempKey got from Nonce



MICROCHIP

🛠 Lab8 MAC (Message Authentication Code)

Develop MAC (Message Authentication Code)

main.c

Source History

```
215   // TODO 3.02
216   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
217   {
218       { "hello", CommandHello, ": Hello!" },
219   // TODO 4.04
220       { "init",  CommandInit, ": Initial CryptoAuthLib" },
221   // TODO 5.04
222       { "readsn", CommandReadSN, ": READ Version and S/N number" },
223   // TODO 6.05
224       { "random", CommandRandom, ": RANDOM command" },
225       { "nonce", CommandNonce, ": NONCE command" },
226   // TODO 7.03
227       { "provision", CommandProvision, ": Provision new chip" },
228   // TODO 8.03
229       { "mac", CommandMac, ": MAC command" },
230   };
231
232   int main ( void )
```

MICROCHIP

🛠 Lab8 MAC (Message Authentication Code)

Result of MAC (Message Authentication Code)

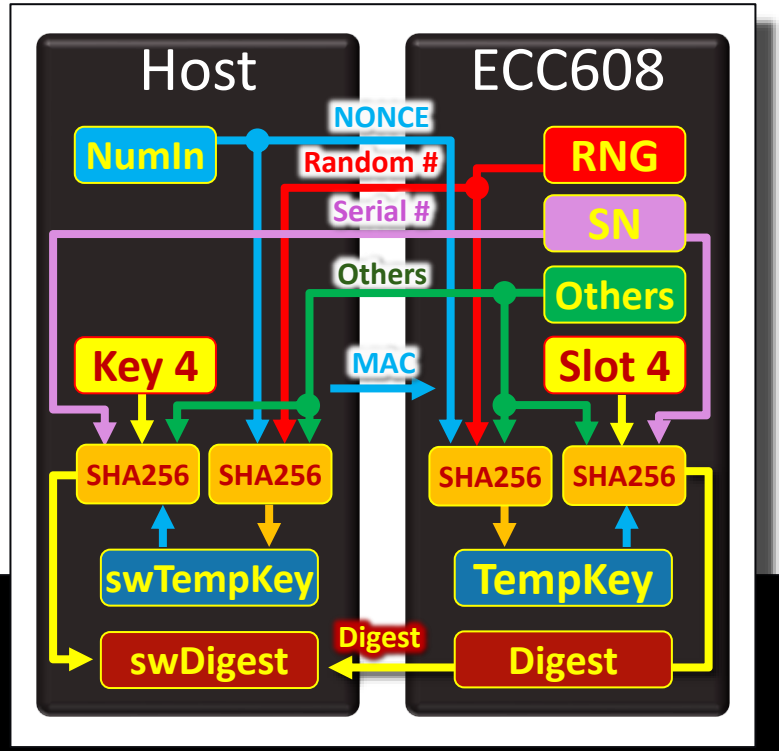
COM13 - Tera Term VT

File Edit Setup Control Window Help

```
>init 60

Initial CryptoAuthLib:
 i2caddr Byte = C0

>readsn

ECC version:
00 00 60 02

Serial Number:
01 23 E4 76 FD D4 FC 75 EE

>nonce

NONCE Command:
- Random number response:
3A 19 4A 43 AB D7 16 C2 5F CC 33 5F 1B 97 29 C6
03 3D 5D AB 61 89 0A 5E 26 C3 8A 48 41 F7 07 A2

- Host Tempkey:
3A 7B D6 93 23 9E 2C EB 72 33 DB D3 3B AC A6 85
8E 72 EB CD 22 AC 05 2C 77 32 8B 4A B4 10 30 F9
```

SN include in MAC Digest Calculate

NONCE is necessary to update TempKey for MAC




```
>mac 4

MAC Command:
- Digest out :
BF D9 88 88 D0 4E 1B B7 DA 87 33 60 D8 7C C4 5E
FE 76 D9 F2 A1 C4 96 37 83 D1 BC 26 56 54 F7 42

- Digest out (Host claculated) :
BF D9 88 88 D0 4E 1B B7 DA 87 33 60 D8 7C C4 5E
FE 76 D9 F2 A1 C4 96 37 83 D1 BC 26 56 54 F7 42
```

Same Digest Result

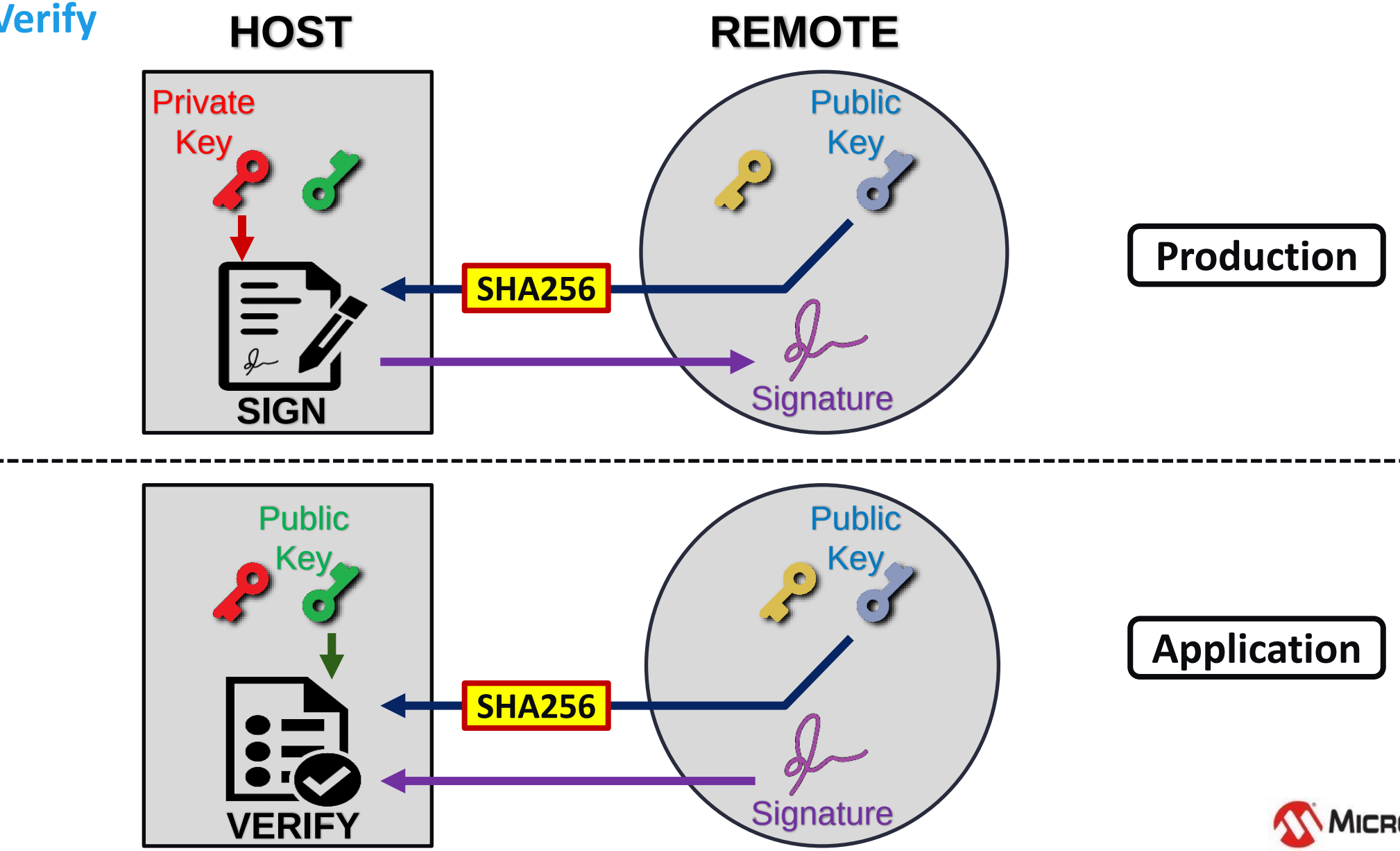
MICROCHIP

Asymmetric Authentication

Public and private key authentication

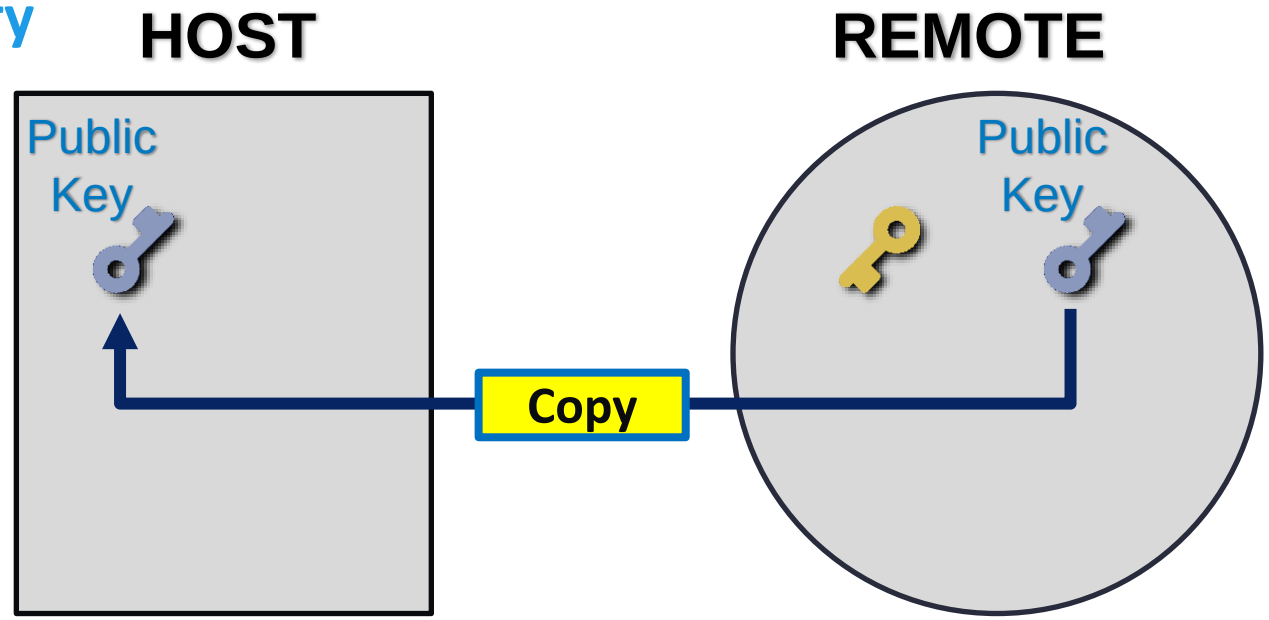
Asymmetric Authentication

Host Sign/Verify

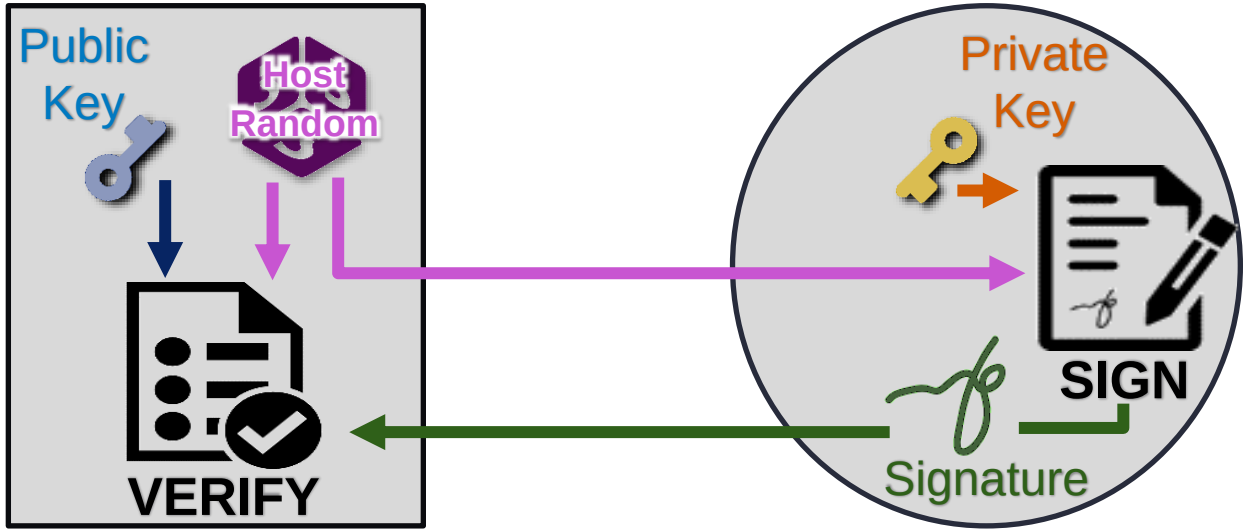


Asymmetric Authentication

Remote Sign/Verify



Production



Application

Asymmetric Authentication

Summary of Sign/Verify challenge

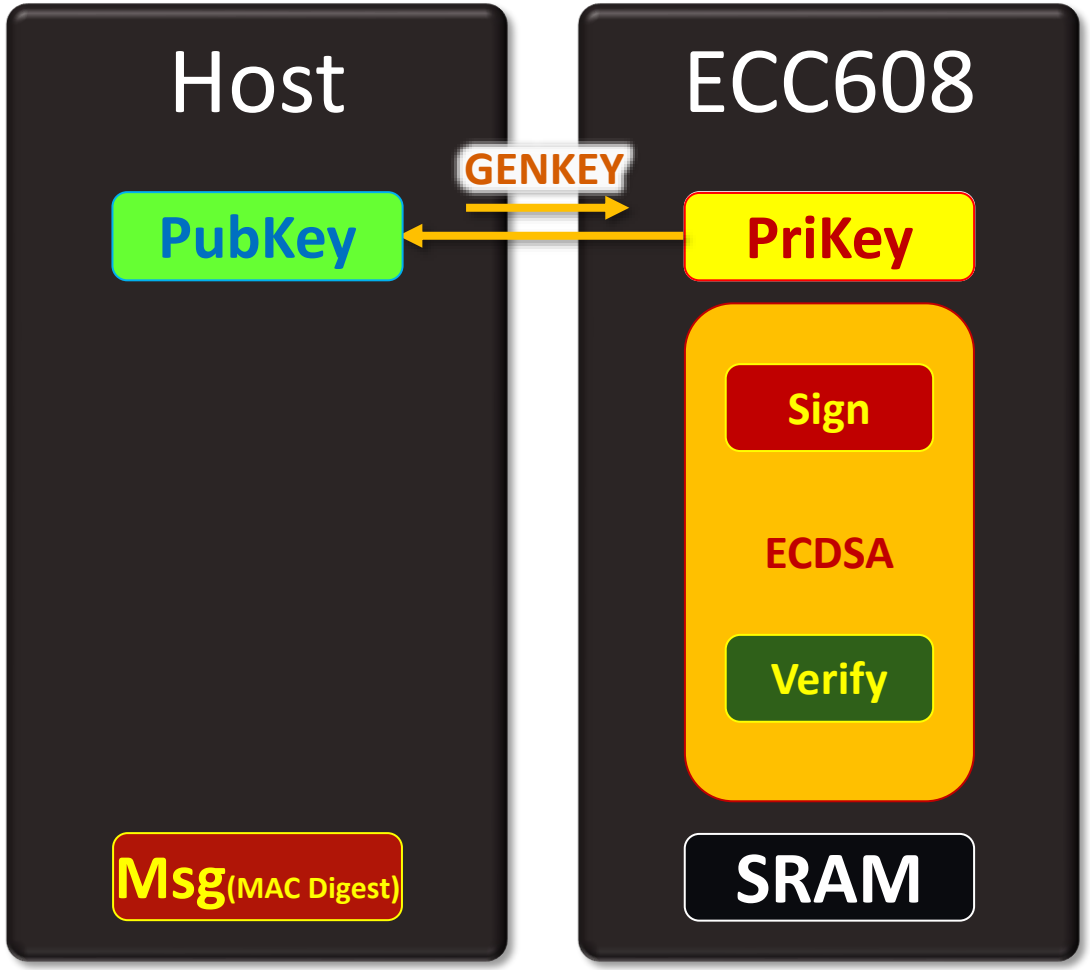
Remote Public Key	Remote Private Key	Remote Signature	Host Random	Host Signature	Remote Public Key Verify	Host Public Key Verify
True(Pair)		True(Sign)	Random	True(Sign)	Pass	Pass
True(Copy)	Fake	Fake(Sign)	Random	True(Copy)	Fail	Pass
Fake(Pair)		Fake(Sign)	Random	True(Copy)	Pass	Fail
True(Copy)	Fake	True(Sniffer)	Random	True(Copy)	Fail	Pass
True(Copy)	Fake	True(Sniffer)	No	True(Copy)	Pass	Pass

🛠 Lab9
GENKEY Command

Asymmetric Authentication

🛠 Lab9 GENKEY Command

Develop GENKEY command for ECC Keypair creation



🛠 Lab9 GENKEY Command

Develop GENKEY command for ECC Keypair creation

main.c ×

Source | History

```
60          0x5d, 0x1b, 0x56, 0x9f, 0xe7, 0x05, 0xb6, 0x00,
61          0x06, 0xfe, 0xec, 0x14, 0x5a, 0x0d, 0xb1, 0xe3
62      };
63      // TODO 8.01
64      atca_mac_in_out_t host_mac;
65      uint16_t key_id;
66      uint8_t digest[32];
67      uint8_t response[32];
68      // TODO 9.01
69      uint8_t pubkey[64];
70
71      // TODO 5.02
72      void OutputData(char* label, uint8_t* buffer, size_t len)
73      {
74          SYS_CMD_PRINT("%s", label);
75          for( int i=0 ; i<len ; i++ )
```

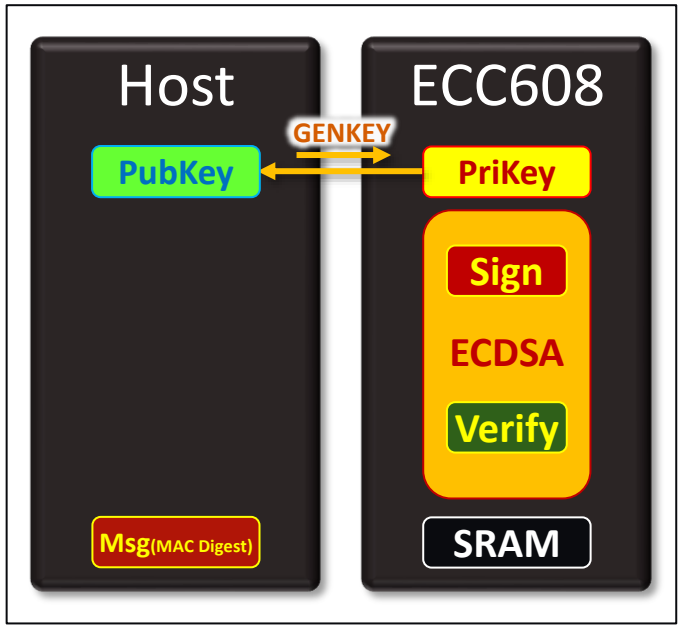
🛠 Lab9 GENKEY Command

Develop GENKEY command for ECC Keypair creation

main.c

Source History

```
217   // TODO 9.02
218   void CommandGenkey(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
219   {
220       if( argc>1 )
221       {
222           memset( pubkey, 0, sizeof(pubkey) );
223           key_id  = (unsigned char)strtol(argv[1], NULL, 16);
224           status = atcab_genkey(key_id, pubkey);
225           CHECK_STATUS(status);
226           SYS_CMD_PRINT("\r\nGENKEY Command (Create Key pair):\r\n");
227           OutputData("- Public key out :\r\n", pubkey, 64);
228       }
229       else
230       {
231           SYS_CMD_PRINT("\r\nNo target ECC Privet key slot !!\r\n");
232       }
233   }
234
235   // TODO 3.02
236   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
```



MICROCHIP

🛠 Lab9 GENKEY Command

Develop GENKEY command for ECC Keypair creation

main.c

Source | History

```
235    // TODO 3.02
236    static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
237    {
238        { "hello", CommandHello, ": Hello!" },
239    // TODO 4.04
240        { "init",  CommandInit, ": Initial CryptoAuthLib" },
241    // TODO 5.04
242        { "readsn", CommandReadSN, ": READ Version and S/N number" },
243    // TODO 6.05
244        { "random", CommandRandom, ": RANDOM command" },
245        { "nonce", CommandNonce, ": NONCE command" },
246    // TODO 7.03
247        { "provision", CommandProvision, ": Provision new chip" },
248    // TODO 8.03
249        { "mac", CommandMac, ": MAC command" },
250    // TODO 9.03
251        { "genkey", CommandGenkey, ": GENKEY command" },
252    };
```

MICROCHIP

Lab9 GENKEY Command

Result of GENKEY command for ECC Keypair creation

COM13 - Tera Term VT

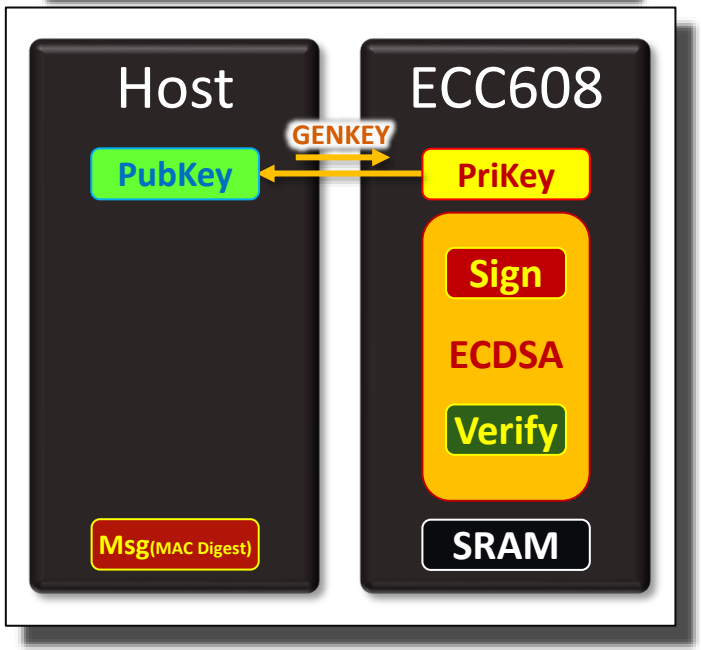
File Edit Setup Control Window Help

```
>init 60

Initial CryptoAuthLib:
- i2caddr Byte = C0
>
>genkey 4
Error: Line 93 in ../src/main.c

GENKEY Command (Create Key pair):
- Public key out :
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

>genkey 0

GENKEY Command (Create Key pair):
- Public key out :
13 A3 91 48 71 2C 1A 1E 9F 13 EC A9 2D 5D CD 3C
6D 5B 05 8D 2D A0 46 32 EB 3F 99 00 F3 AA 82 43
37 88 BB 48 A8 43 D4 44 CE 82 DC 7C C3 52 B9 77
96 FC 25 F0 87 6F A6 2C A7 86 00 B1 C5 4A 3D 47

>
```

Wrong target ECC Key slot 4

Correct target ECC key slot 0

Slot	Description	Type	Lockable	Req
0		ECC Private Key	☑	☐
1		Data	☐	☐
2		ECC Private Key	☐	☐
3		Data	☐	☐
4		Data	☑	☐
5		Data	☑	☐



MICROCHIP

The ECDSA (Elliptic Curve Digital Signature Algorithm)

Asymmetric Authentication

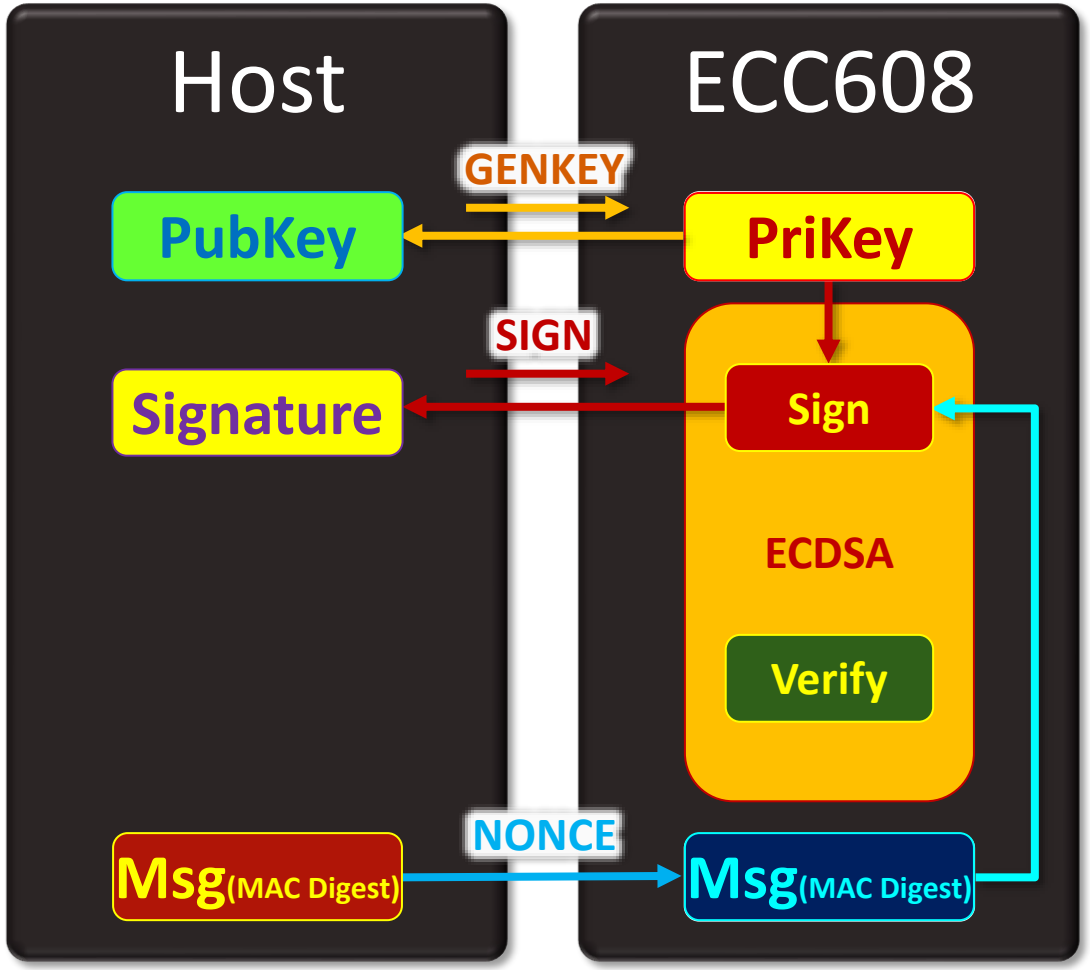
🛠 Lab10
SIGN Command

The ECDSA (Elliptic Curve Digital Signature Algorithm)

MICROCHIP

🛠 Lab10 SIGN Command

Develop SIGN command for ECDSA Sign operation



Lab10 SIGN Command

Develop SIGN command for ECDSA Sign operation

main.c

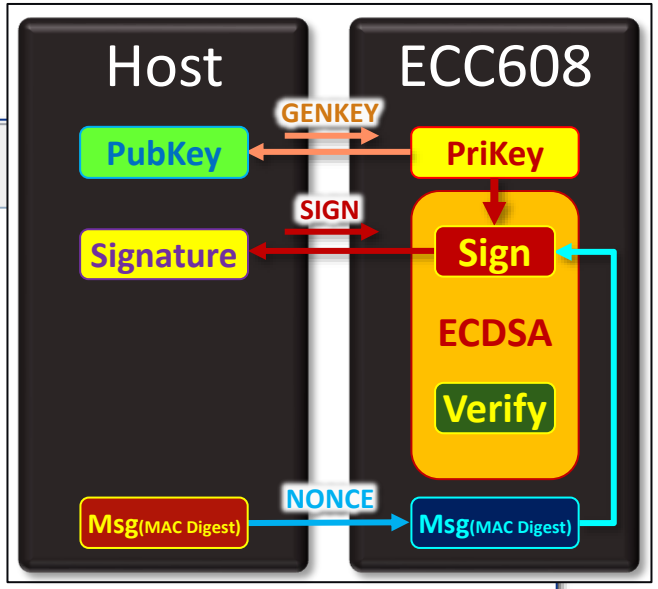
Source History

```
66    uint8_t digest[32];
67    uint8_t response[32];
68    // TODO 9.01
69    uint8_t pubkey[64];
70    // TODO 10.01
71    uint8_t signature[64];
72
73    // TODO 5.02
74    void OutputData(char* label, uint8_t* buffer, size_t len)
75    {
76        SYS_CMD_PRINT("%s", label);
77        for( int i=0 ; i<len ; i++ )
78        {
79            SYS_CMD_PRINT("%02X ", buffer[i]);
80            if( (i+1)%16==0 )
81            {
```

MICROCHIP

Lab10 SIGN Command

Develop SIGN command for ECDSA Sign operation



main.c

Source | History

```
237   // TODO 10.02
238   void CommandSign(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
239   {
240       if( argc>1 )
241       {
242           key_id  = (unsigned char)strtol(argv[1], NULL, 16);
243           status = atcab_sign(key_id, digest, signature);
244           CHECK_STATUS(status);
245           OutputData("\r\nSIGN Command\r\n- Msg for Sign :\r\n", digest, 32);
246           OutputData("- Signature out :\r\n", signature, 64);
247       }
248       else
249       {
250           SYS_CMD_PRINT("\r\nNo target ECC Privet key slot !!\r\n");
251       }
252   }
253
254   // TODO 3.02
255   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
```

MICROCHIP

🛠 Lab10 SIGN Command

Develop SIGN command for ECDSA Sign operation

main.c ×

Source History

```
255   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
256   {
257       { "hello", CommandHello, ": Hello!" },
258   // TODO 4.04
259       { "init",  CommandInit, ": Initial CryptoAuthLib" },
260   // TODO 5.04
261       { "readsn", CommandReadSN, ": READ Version and S/N number" },
262   // TODO 6.05
263       { "random", CommandRandom, ": RANDOM command" },
264       { "nonce", CommandNonce, ": NONCE command" },
265   // TODO 7.03
266       { "provision", CommandProvision, ": Provision new chip" },
267   // TODO 8.03
268       { "mac", CommandMac, ": MAC command" },
269   // TODO 9.03
270       { "genkey", CommandGenkey, ": GENKEY command" },
271   // TODO 10.03
272       { "sign", CommandSign, ": SIGN command" },
273   };
```

🛠 Lab10 SIGN Command

Result of SIGN command for ECDSA Sign operation

COM13 - Tera Term VT

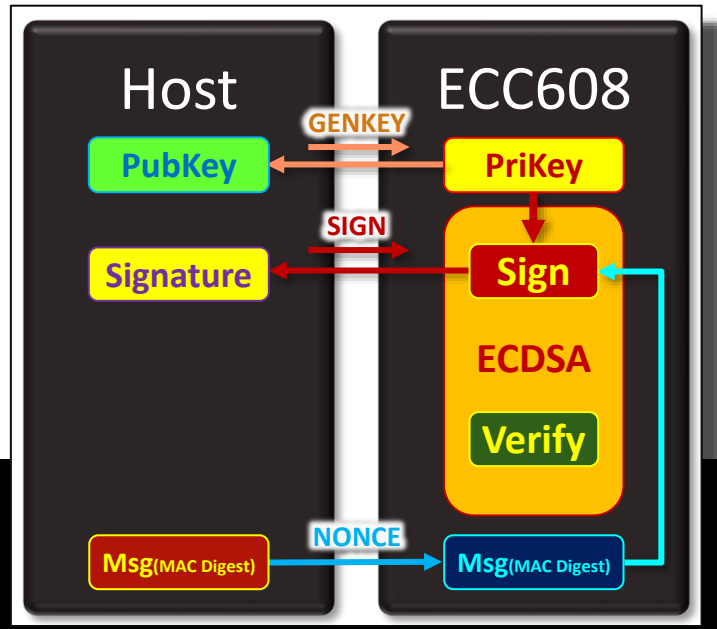
File Edit Setup Control Window Help

```
>mac 4

MAC Command:
- Digest out :
B8 4D 26 A2 78 F1 1D 0F D9 64 B1 D4 B2 33 15 9A
42 3F C9 E6 C7 89 D6 52 0B 69 32 0A 26 C1 F8 73

- Digest out (Host claculated) :
B8 4D 26 A2 78 F1 1D 0F D9 64 B1 D4 B2 33 15 9A
42 3F C9 E6 C7 89 D6 52 0B 69 32 0A 26 C1 F8 73

>genkey 0

GENKEY Command (Create Key pair):
- Public key out :
35 DF 34 96 A3 05 5D 89 C5 87 2F 92 F1 7D 43 9A
0C 7D EC 48 40 43 18 B3 A6 E1 11 59 81 3E E1 BB
70 C8 F3 C0 AF 35 DF ED 62 98 38 3C B6 82 53 BE
3E 26 C2 FD CA 3C 8B 4E 42 39 5B 37 4D 80 FA 43

>
```

Prepare MAC Digest as Msg for NONCE passthrough

GENKEY to Create ECC Keypair

ECC Public Key of key slot 0




```
>sign 0

SIGN Command:
- Msg for Sign :
B8 4D 26 A2 78 F1 1D 0F D9 64 B1 D4 B2 33 15 9A
42 3F C9 E6 C7 89 D6 52 0B 69 32 0A 26 C1 F8 73

- Signature out :
E2 8A F0 22 93 9A 82 27 51 CF 2D 2F 03 56 2F B9
C9 B7 14 6D 22 4D E8 B0 03 DC E8 F1 2D 3C 20 5E
B4 E9 F7 BC E5 F5 80 6C 47 4A F0 B0 12 6C CD 35
D1 A6 80 04 AD 3B A9 35 ED 93 EB 5C 30 10 75 C6
```

Signature output

MICROCHIP

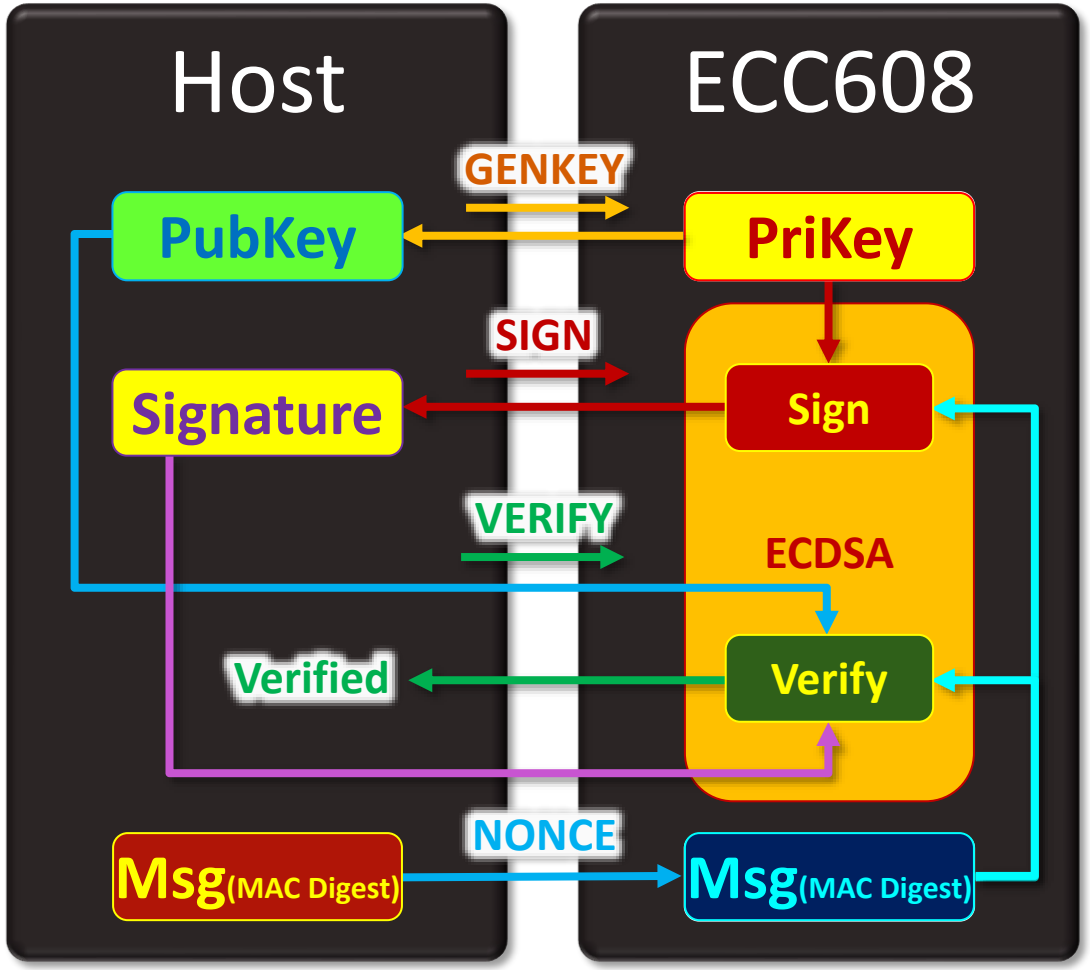
🛠 Lab11
VERIFY Command

The ECDSA (Elliptic Curve Digital Signature Algorithm)

MICROCHIP

🛠 Lab11 VERIFY Command

Develop VERIFY command for ECDSA Verify operation



MICROCHIP

🛠 Lab11 VERIFY Command

Develop VERIFY command for ECDSA Verify operation

main.c

Source History

```
66    uint8_t digest[32];
67    uint8_t response[32];
68    // TODO 9.01
69    uint8_t pubkey[64];
70    // TODO 10.01
71    uint8_t signature[64];
72    // TODO 11.01
73    bool verifyReslut;
74
75    // TODO 5.02
76    void OutputData(char* label, uint8_t* buffer, size_t len)
77    {
78        SYS_CMD_PRINT("%s", label);
79        for( int i=0 ; i<len ; i++ )
80        {
81            SYS_CMD_PRINT("%02X ", buffer[i]);
```

MICROCHIP

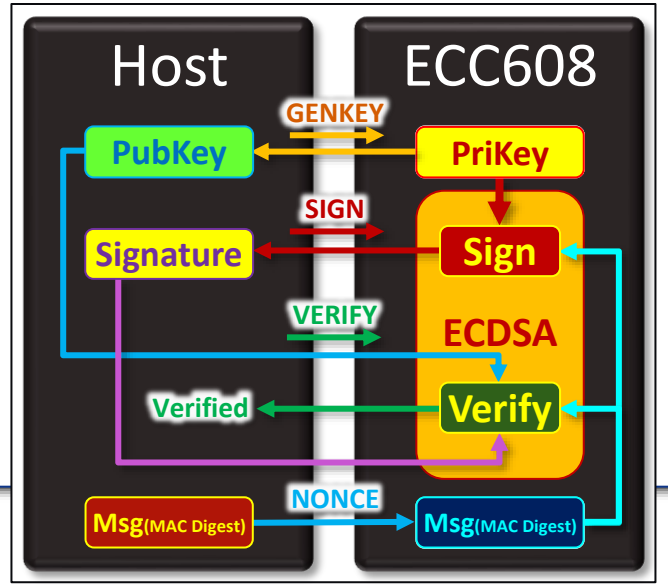
🛠 Lab11 VERIFY Command

Develop VERIFY command for ECDSA Verify operation

main.c

Source | History

```
256  // TODO 11.02
257  void CommandVerify(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
258  {
259      SYS_CMD_PRINT("\r\nVERIFY Command (Extern Mode):\r\n");
260      OutputData("- Msg for Verify :\r\n", digest, 32);
261      OutputData("- Signature :\r\n", signature, 64);
262      OutputData("- Public key :\r\n", pubkey, 64);
263      status = atcab_verify_extern(digest, signature, pubkey, &verifyReslut);
264      CHECK_STATUS(status);
265
266      if( verifyReslut == true )
267          SYS_CMD_PRINT("- VERIFY Success\r\n");
268      else
269          SYS_CMD_PRINT("- VERIFY Failure\r\n");
270  }
271
272  // TODO 3.02
273  static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
```



MICROCHIP

🛠 Lab11 VERIFY Command

Develop VERIFY command for ECDSA Verify operation

main.c

Source History

```
273   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
274   {
275       { "hello", CommandHello, ": Hello!" },
276   // TODO 4.04
277       { "init",  CommandInit, ": Initial CryptoAuthLib" },
278   // TODO 5.04
279       { "readsn", CommandReadSN, ": READ Version and S/N number" },
280   // TODO 6.05
281       { "random", CommandRandom, ": RANDOM command" },
282       { "nonce", CommandNonce, ": NONCE command" },
283   // TODO 7.03
284       { "provision", CommandProvision, ": Provision new chip" },
285   // TODO 8.03
286       { "mac", CommandMac, ": MAC command" },
287   // TODO 9.03
288       { "genkey", CommandGenkey, ": GENKEY command" },
289   // TODO 10.03
290       { "sign", CommandSign, ": SIGN command" },
291   // TODO 11.03
292       { "verify", CommandVerify, ": VERIFY command" },
293   };
```

MICROCHIP

🛠 Lab11 VERIFY Command

Result of VERIFY command for ECDSA Verify operation

COM13 - Tera Term VT

File Edit Setup Control Window Help

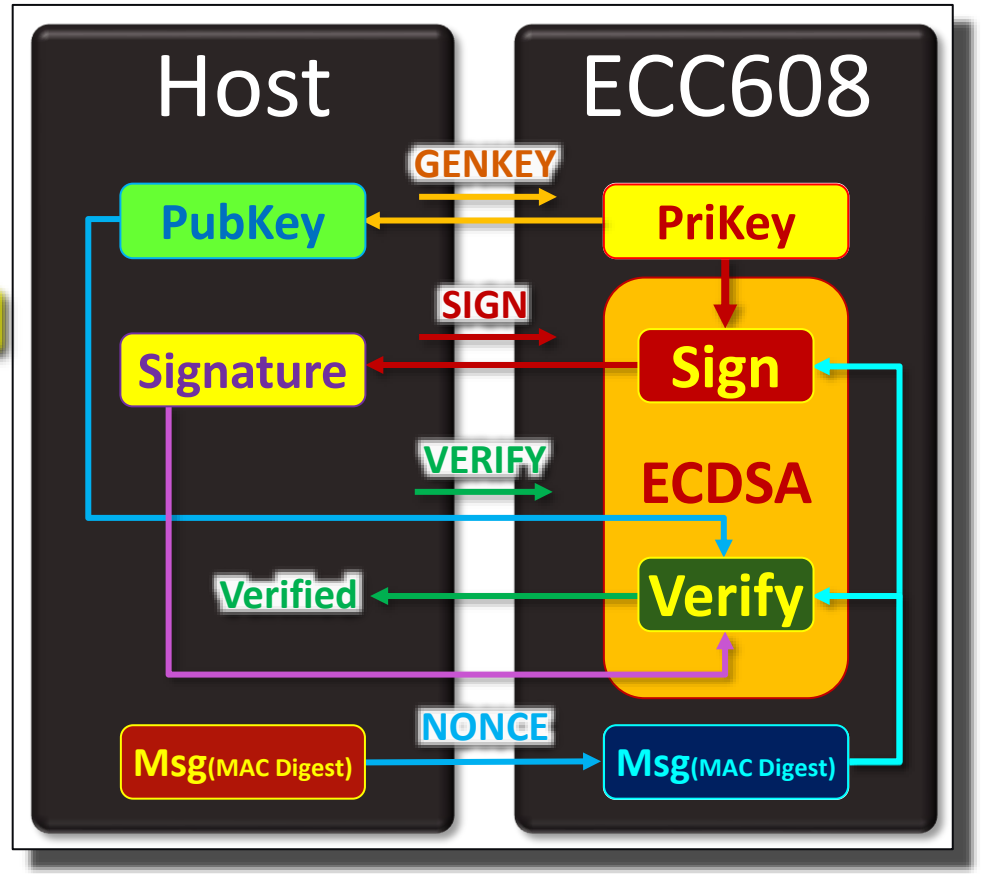
```
>verify

VERIFY Command (Extern Mode):
- Msg for Verify :
B8 4D 26 A2 78 F1 1D 0F D9 64 B1 D4 B2 33 15 9A
42 3F C9 E6 C7 89 D6 52 0B 69 32 0A 26 C1 F8 73

- Signature :
E2 8A F0 22 93 9A 82 27 51 CF 2D 2F 03 56 2F B9
C9 B7 14 6D 22 4D E8 B0 03 DC E8 F1 2D 3C 20 5E
B4 E9 F7 BC E5 F5 80 6C 47 4A F0 B0 12 6C CD 35
D1 A6 80 04 AD 3B A9 35 ED 93 EB 5C 30 10 75 C6

- Public key :
35 DF 34 96 A3 05 5D 89 C5 87 2F 92 F1 7D 43 9A
0C 7D EC 48 40 43 18 B3 A6 E1 11 59 81 3E E1 BB
70 C8 F3 C0 AF 35 DF ED 62 98 38 3C B6 82 53 BE
3E 26 C2 FD CA 3C 8B 4E 42 39 5B 37 4D 80 FA 43

- VERIFY Success
>
```

MAC Digest as Msg

Signature output of SIGN command

ECC Public Key of key slot 0

Verified



MICROCHIP

ECC Command Package

The package of ECC communication

ECC Command Package

Write command format

I2C Addr W	I2C Write Address.	(**One byte**)
Cmd Prefix	Command prefix byte. For I2C interface is **0x03** For SWI interface is **0x77**	(**One byte**)
Count	Package Length from ‘Count’ to ‘CRC16’.	(**One byte**)
Opcode	The Command operation code.	(**One byte**)
Param1	The first parameter; always present.	(**One byte**)
Param2	The second parameter; always present.	(**Two Bytes**)
Data	Optional remaining input data.	(**0 or N Bytes**)
CRC16	Checksum CRC-16 from ‘Count’ to ‘Data’.	(**Two bytes**)

ECC Command Package

Read response format

I2C Addr R	I2C Read Address.	(**One byte**)
Count	Package Length from ‘Count’ to ‘CRC16’.	(**One byte**)
Data / Status	Data, or Status.	(**1 or N Bytes**)
CRC16	Checksum CRC-16 from ‘Count’ to ‘Data’.	(**Two bytes**)

ECC Command Package

Read Revision and Serial Number raw command and response

- **Write raw command**

I2C Addr	W	Cmd Prefix	Count	Opcode	Param1	Param2	Data	CRC16
0xC0		0x03	0x07	0x02	0x80	0x00 0x00		0x09 0xAD

- **Read response**

I2C Addr	R	Count	Data / Status	CRC16
0xC1		0x23	0x01 0x23 0x1C 0x25 0x00 0x00 0x60 0x02 0xB7 0x9A 0xFA 0xD5 0xEE 0x01 0x21 0x00 0xC0 0x00 0x00 0x00 0x83 0x20 0x87 0x20 0x8F 0x20 0xC4 0x8F 0x8F 0x8F 0x8F 0x8F	0x2A 0x06

```
>readsn

ECC version:
00 00 60 02

Serial Number:
01 23 1C 25 B7 9A FA D5 EE
```

MICROCHIP

ECC Command Package

Description of raw command

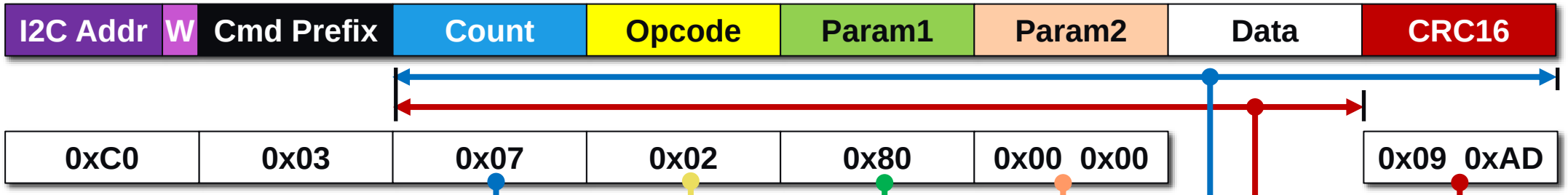


Table 11-38. Input Parameters

	Name	Size	Notes
Opcode	READ	1	0x02
Param1	Zone	1	Bit 7: 1 = 32 bytes are read; otherwise four bytes are read. Bits 2-6: Must be zero. Bits 0-1: Select among Configuration, OTP, or Data. See SectionAddress Encoding.
Param2	Address	2	Address of first word to be read within the zone. See SectionAddress Encoding.
Data	—	0	—

Table 10-6. Zone Encoding (Param1)

Zone	Param1
Config	0
OTP	1
Data	2

ECC Command Package

Description of response

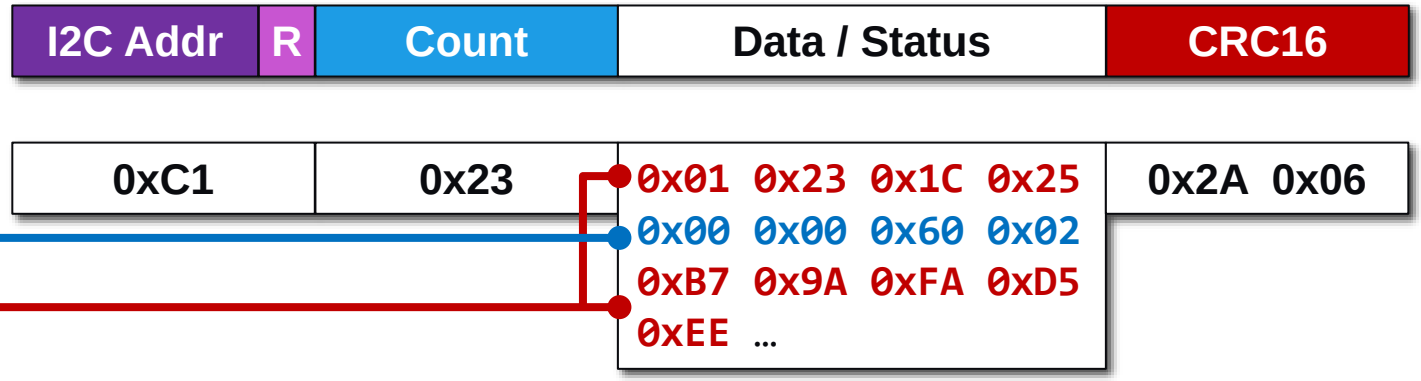


Table 2-5. Configuration Zone

Byte	Name	Description	Write	Read
0 → 3	SN<0:3>	Part of the serial number value. See Section Serial Number (Bytes 0-2 and 8-12)	Never	Always
4 → 7	RevNum	Device revision number. See Section Revision Number (Bytes 4-7)	Never	Always
8 → 12	SN<4:8>	Part of the serial number value. See Section Serial Number (Bytes 0-2 and 8-12)	Never	Always
.....	.....	.....		

🛠 Lab12
Command Builder

ECC Command Package

MICROCHIP

🛠 Lab12 Command Builder

Develop a Command Builder for raw command

main.c

Source History

```
70    // TODO 10.01
71    uint8_t signature[64];
72    // TODO 11.01
73    bool verifyReslut = false;
74    // TODO 12.01
75    uint8_t RawCommandData[192];
76
77    // TODO 5.02
78    void OutputData(char* label, uint8_t* buffer, size_t len)
79    {
80        SYS_CMD_PRINT("%s", label);
81        for( int i=0 ; i<len ; i++ )
82        {
83            SYS_CMD_PRINT("%02X ", buffer[i]);
84            if( (i+1)%16==0 )
85            {
```

MICROCHIP

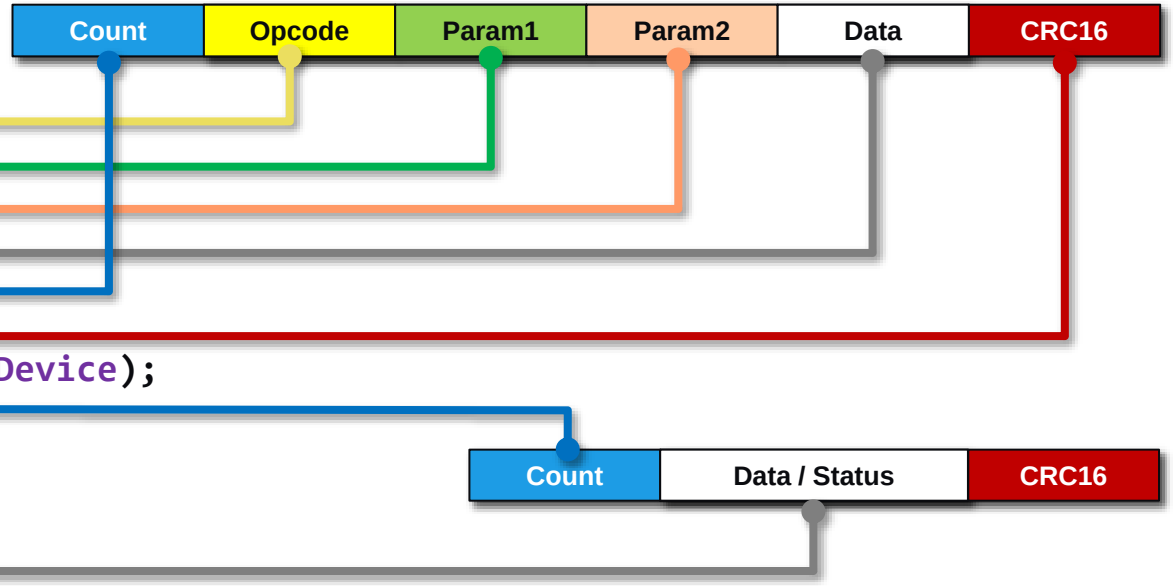
🛠 Lab12 Command Builder

Develop a Command Builder for raw command

main.c

Source | History

```
258  // TODO 11.02
259  void CommandVerify(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
260  {...13 lines }
273
274  // TODO 12.03
275  ATCA_STATUS atcab_rawcmd(uint8_t opcode, uint8_t param1, uint16_t param2, uint8_t *data, uint8_t *len)
276  {
277      ATCAPacket packet;
278
279      packet.opcode = opcode;
280      packet.param1 = param1;
281      packet.param2 = param2;
282      memcpy(&packet.data[*len], data, *len);
283      packet.txsize = 7 + *len;
284      atCalcCrc(&packet);
285      status = atca_execute_command(&packet, _gDevice);
286      *len = packet.data[0]-3;
287      memcpy(data, &packet.data[1], *len);
288
289      return status;
290  }
291
292  // TODO 12.02
293  void CommandRawCmd(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
```



Note! Put the code above of TODO 12.02

MICROCHIP

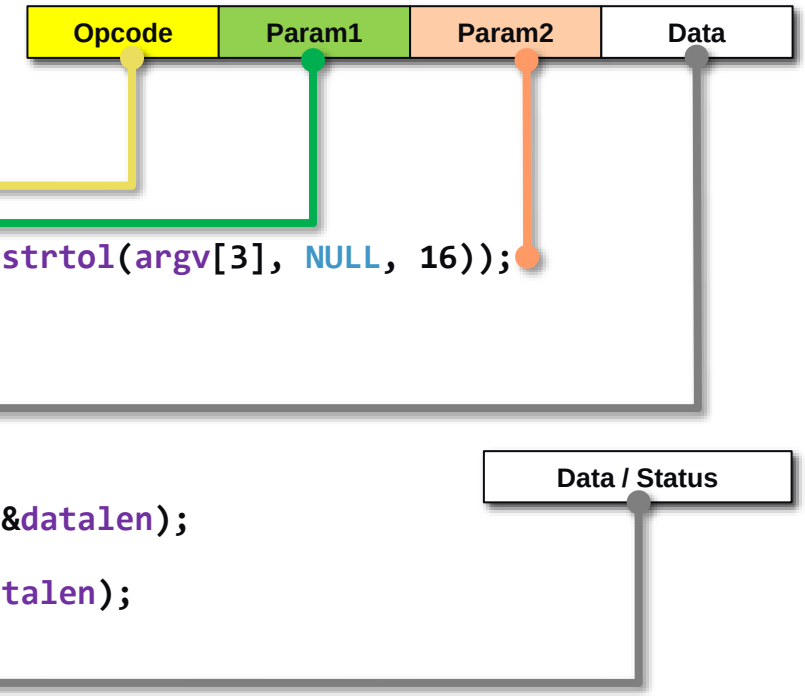
Lab12 Command Builder

Develop a Command Builder for raw command

main.c

Source History

```
274   // TODO 12.02
275   void CommandRawCmd(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
276   {
277       if( argc>1 )
278       {
279           uint8_t opcode, param1, *data, datalen;
280           uint16_t param2;
281
282           opcode = (uint8_t)strtol(argv[1], NULL, 16);
283           param1 = (uint8_t)strtol(argv[2], NULL, 16);
284           param2  = (uint16_t)(strtol(argv[4], NULL, 16)<<8 | strtol(argv[3], NULL, 16));
285           if( argc>5 )
286           {
287               datalen = atoi(argv[6]);
288               memcpy( RawCommandData, argv[5], datalen);
289               data = RawCommandData;
290           }
291           status = atcab_rawcmd(opcode, param1, param2, data, &datalen);
292           CHECK_STATUS(status);
293           OutputData("\r\nRaw Command Response:\r\n", data, datalen);
294       }
295   }
296
297   // TODO 3.02
298   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
```



MICROCHIP

🛠 Lab12 Command Builder

Develop a Command Builder for raw command

main.c

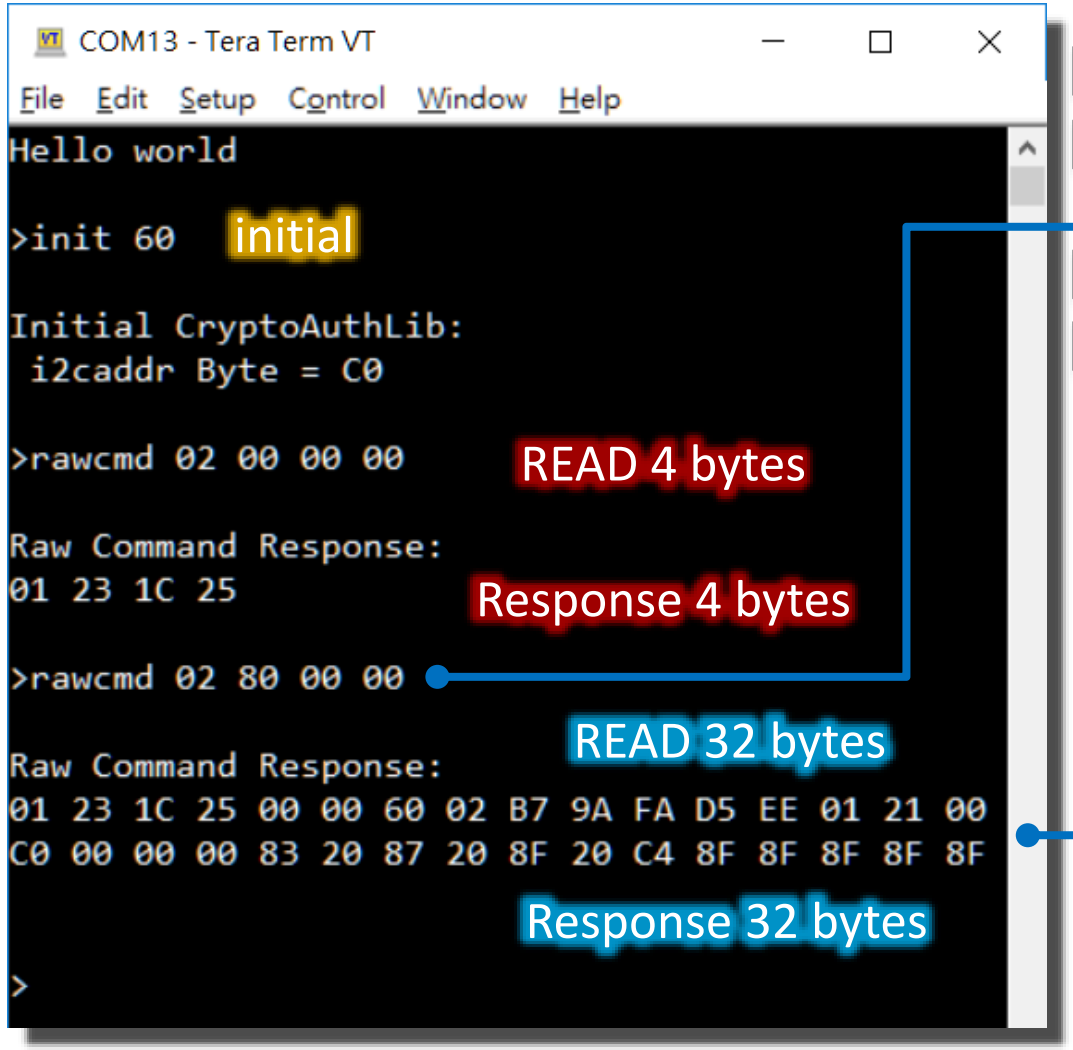
Source History

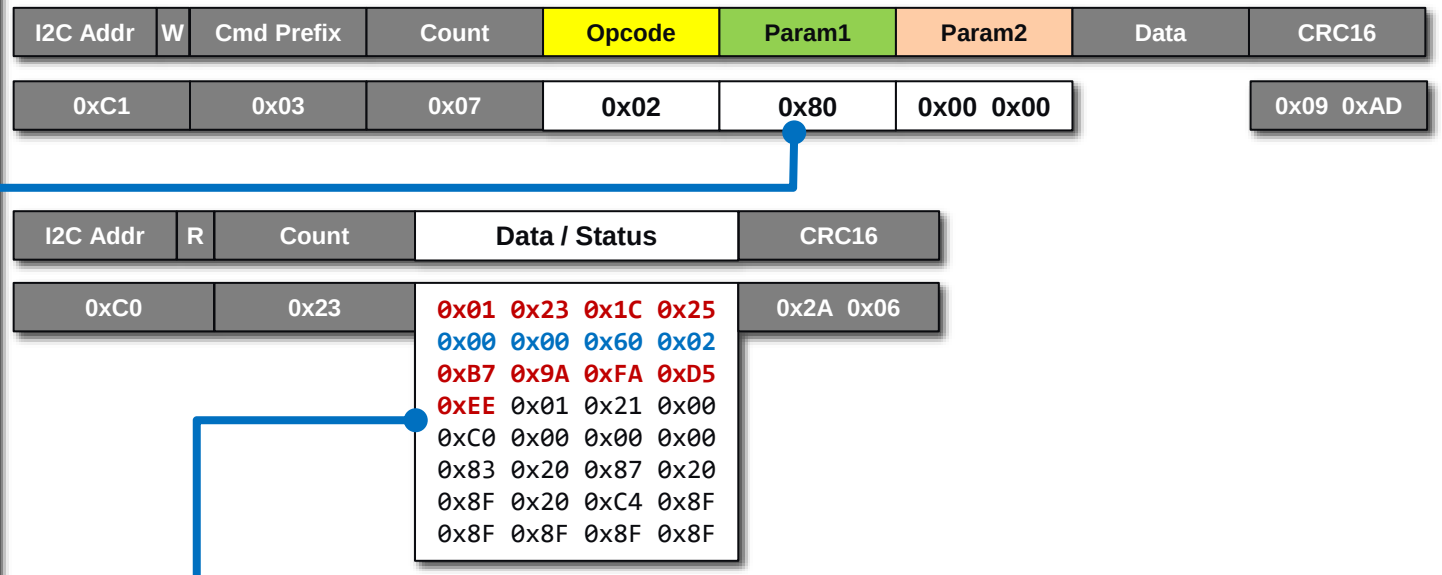
```
316   static const SYS_CMD_DESCRIPTOR CryptoCmdTbl[]=
317   {
318       { "hello", CommandHello, ": Hello!" },
319   // TODO 4.04
320       { "init",  CommandInit, ": Initial CryptoAuthLib" },
321   // TODO 5.04
322       { "readsn", CommandReadSN, ": READ Version and S/N number" },
323   // TODO 6.05
324       { "random", CommandRandom, ": RANDOM command" },
325       { "nonce", CommandNonce, ": NONCE command" },
326   // TODO 7.03
327       { "provision", CommandProvision, ": Provision new chip" },
328   // TODO 8.03
329       { "mac", CommandMac, ": MAC command" },
330   // TODO 9.03
331       { "genkey", CommandGenkey, ": GENKEY command" },
332   // TODO 10.03
333       { "sign", CommandSign, ": SIGN command" },
334   // TODO 11.03
335       { "verify", CommandVerify, ": VERIFY command" },
336   // TODO 12.04
337       { "rawcmd",    CommandRawCmd,    ": Raw command bytes" },
338   };
```

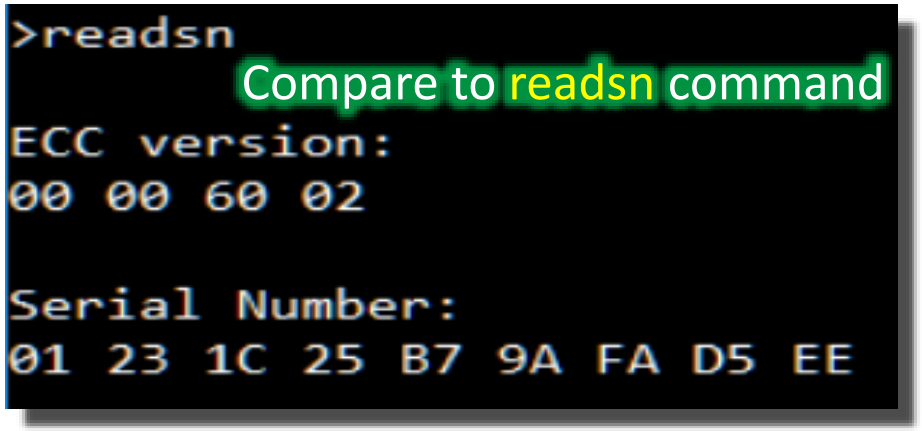
MICROCHIP

🛠 Lab12 Command Builder

Develop a Command Builder for raw command







MICROCHIP

Microchip Trademarks

Overview

Microchip Technology Incorporated's products are marketed and sold under one or more of the following trademarks belonging to Microchip or one of Microchip's subsidiaries. Questions regarding use of these trademarks should be addressed to the Microchip's Legal Department via email: legal.department@microchip.com.

The following are registered trademarks of Microchip in the U.S.A. and other countries:

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA

The following are registered trademarks of Microchip in the U.S.A:

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, WinPath, and ZL

The following are registered trademarks of Microchip in other countries: BeaconThings, GestIC, Silicon Storage Technology

The following are trademarks of Microchip in the U.S.A. and other countries:

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA

The following is a service mark of Microchip in the U.S.A.: SQTP

The following are logos of Microchip in the U.S.A. and other countries: The Adaptec logo, Frequency on Demand, Silicon Storage Technology, Symmcom, and Trusted Time

The following is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries: GestIC

MICROCHIP