16. 최신 트렌드 및 응용 분야

16.1 AI 칩과 마이크로프로세서

🧠 왜 AI 칩이 필요한가?

기존 마이크로프로세서(CPU, MCU)는 일반적인 제어·계산에는 뛰어나지만,
딥러닝 연산(행렬 곱셈, Convolution, Activation 등)에서는 성능과 전력 효율이 낮다.

이유	설명
연산 패턴 특수	대부분 `행렬 곱`, `벡터 연산` 중심
병렬성 요구 ↑	GPU/TPU 수준의 massively parallel 처리 필요
대역폭 / 캐시 최적화	일반 CPU보다 고정된 데이터 흐름 구조

🧩 마이크로프로세서 vs AI 칩 구조 비교

항목	마이크로프로세서 (CPU/MCU)	AI 칩 (NPU, TPU 등)
구조	범용 제어 구조	행렬/벡터 처리 전용 아키텍처
ALU 수	소수	수천~수만 개 병렬 MAC
연산 단위	32/64비트 정밀도	8/16비트 저정밀 또는 BFloat16
제어 방식	프로그램 로직 중심	데이터 플로우 중심
활용	논리제어, OS	CNN, LSTM, Transformer 추론
소비 전력	낮음 (MCU) 또는 높음 (서버 CPU)	전력당 연산 효율 높음 (TOPS/W)

🔍 대표 AI 전용 칩 종류

종류	설계사	특징
TPU (Tensor Processing Unit)	Google	8x8~128x128 행렬 연산기, BFloat16
NPU (Neural Processing Unit)	ARM, Huawei, Apple 등	SoC에 내장, INT8 최적화
DLA (Deep Learning Accelerator)	NVIDIA	Jetson 시리즈에 내장
Edge TPU	Google Coral	초저전력 추론 칩 (3~5 TOPS)
Sipeed K210	RISC-V + KPU 내장 (MCU급 NPU)	

🧪 연산 포맷 차이

포맷	비트 수	설명
FP32	32bit 부동소수점	CPU/GPU 기본 연산
FP16 / BFloat16	저정밀 부동소수점	TPU, NPU
INT8	정수형 8bit	최적화된 추론용
INT4 / Binary	극저전력용 모델	

🧠 AI 연산 구조: MAC (Multiply-Accumulate)

> 모든 CNN, MLP, RNN은 결국 **MAC 연산의 반복**

Ex. Convolution 연산

```
Output = Σ (Weight × Input) + Bias
```

→ AI 칩은 **수천 개의 MAC 유닛**을 병렬로 동작시켜 CNN을 빠르게 수행

⚙️ AI 칩 내 구조 예 (TPU 스타일)

```
+------------------------+
| Matrix Multiply Unit   |  → (128×128) MAC Array
| ┌──────────┐           |
| | Weight   |           |
| | Activation|          |
| └──────────┘           |
|       ↓                |
| Accumulator + ReLU     |
| On-chip SRAM (4MB+)    |
+------------------------+
```

📦 AI 칩이 내장된 대표 SoC

SoC	NPU 탑재	활용 예
Apple M1/M2	16-core Neural Engine	iOS, macOS ML
HiSilicon Kirin 990	NPU 2-core	스마트폰 AI
NVIDIA Jetson Nano/Xavier	DLA + GPU	로봇/임베디드 추론
Sipeed K210	KPU (1TOPS)	초저가 AI MCU
Samsung Exynos	AI Engine 탑재	Bixby, 카메라 추론
Google Edge TPU	4TOPS@0.5W	TensorFlow Lite 추론

🤖 마이크로프로세서와 AI 칩의 통합 구조 예

```
[ MCU (STM32, ESP32) ]
        |
        | UART/SPI/I2C
        ▼
[ NPU (K210, Edge TPU, Coral) ]
        |
        └ 추론 처리 (이미지 분류, 얼굴 검출 등)
```

> MCU가 **센서 수집 + NPU에게 이미지 전달 → 결과 받아 처리**

💡 실습 가능한 구조 예

구성	설명
STM32 ↔ K210	UART 기반 추론 요청
Raspberry Pi + Coral	USB + TensorFlow Lite EdgeTPU 연동
Jetson Nano	Python + TensorRT 추론
Sipeed Maix Dock	RISC-V MCU + KPU로 독립 동작 (MaixPy, C SDK)

📌 요약 정리

항목	내용
AI 칩 목적	CNN, DNN 추론에 최적화된 병렬 연산 전용 하드웨어
MCU vs NPU	범용 vs AI 특화 / 제어 vs 병렬연산
MAC 연산	AI 연산의 핵심 / 병렬화 구조 필요
데이터 형식	FP32 → INT8, BFloat16 등 경량화
연동 방식	MCU ↔ NPU 통신 (UART, SPI, DMA 등)
주요 칩	TPU, NPU, EdgeTPU, DLA, KPU 등
활용	얼굴 인식, 물체 감지, 음성 인식, anomaly detection

16.2 임베디드 시스템에서의 MPU 응용

🧠 MPU란?

MPU (Memory Protection Unit)는
임베디드 MCU에서 **특정 메모리 영역의 접근 권한을 제어**하여
시스템의 안정성과 보안을 강화하는 하드웨어 모듈이다.

> 복잡한 가상메모리 없이도,
> **버퍼 오버플로우, 잘못된 포인터 접근, 태스크 간 침범 방지** 가능

🧩 MPU vs MMU 비교

항목	MPU	MMU
사용 대상	MCU (Cortex-M)	CPU (Cortex-A 등)
주소 변환	❌ 없음 (물리 주소만 사용)	✅ 가상 ↔ 물리 주소 매핑
권한 보호	✅ region 단위 read/write/execute 권한	✅ page 단위
설정 방식	소프트웨어적으로 직접 설정	OS 또는 하드웨어
RTOS 연동	✅ FreeRTOS, RTX, Zephyr 등	보통 OS 전용

🔐 MPU가 필요한 이유

목적	설명
안전성 확보	오류 접근 차단 (HardFault로 전환)
보안성 향상	코드 영역 쓰기 방지, 스택 보호
실시간성 보장	단순 구조로 오버헤드 없음
RTOS 태스크 보호	각 태스크의 Stack, Heap 영역 분리

🧠 MPU의 구성 요소 (Cortex-M 기준)

구성 요소	설명
Region	보호할 영역 (최대 8개 또는 16개)
Base Address	시작 주소
Size	32B ~ 4GB (2^n 크기만 가능)
Access Permission	no access / read-only / full access
Execute Never (XN)	코드 실행 금지 영역 설정
Subregion Disable	8등분 영역 일부만 마스킹 가능

✅ 실습 예: STM32에서 MPU 설정 (Cortex-M4)

1. 기본 설정 코드 (STM32CubeIDE 또는 레지스터 직접)

```
#include "stm32f4xx_hal.h"
void MPU_Config(void) {
    HAL_MPU_Disable();

    MPU_Region_InitTypeDef MPU_InitStruct = {0};

```

```
    MPU_InitStruct.Enable           = MPU_REGION_ENABLE;
    MPU_InitStruct.BaseAddress      = 0x20010000; // SRAM 일부
    MPU_InitStruct.Size             = MPU_REGION_SIZE_32KB;
    MPU_InitStruct.AccessPermission = MPU_REGION_NO_ACCESS;
    MPU_InitStruct.IsBufferable     = 0;
    MPU_InitStruct.IsCacheable      = 0;
    MPU_InitStruct.IsShareable      = 0;
    MPU_InitStruct.Number           = MPU_REGION_NUMBER0;
    MPU_InitStruct.TypeExtField     = MPU_TEX_LEVEL0;
    MPU_InitStruct.SubRegionDisable= 0x00;
    MPU_InitStruct.DisableExec      = MPU_INSTRUCTION_ACCESS_DISABLE;

    HAL_MPU_ConfigRegion(&MPU_InitStruct);
    HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);
}
```

2. main()에서 호출

```
int main(void) {
    HAL_Init();
    MPU_Config();  // 보호 활성화
    SystemClock_Config();
    ...
}
```

💥 예외 발생 예

```
volatile uint32_t* ptr = (uint32_t*)0x20010000;
*ptr = 0x12345678;  // 접근 시 HardFault 발생 (MPU 미허용)
```

→ 접근이 금지된 영역을 쓰거나 읽으려 할 경우,
→ **HardFault_Handler()로 진입** → 에러 처리 또는 리셋 가능

🧠 RTOS + MPU 연동 구조

FreeRTOS 예

```
xTaskCreateRestricted(&xTaskParameters, &xHandle);
```

- 태스크 별로 **MPU region 지정**
- **Stack overflow, Heap 침범 방지**
- **Unprivileged Task / Privileged Kernel** 분리

🔒 응용 사례: 실시간 시스템에서의 활용

분야	설명
자동차 (ASIL-D)	태스크 간 보호, 메모리 접근 방지

분야	설명
드론/로봇	센서/모터 인터페이스 분리 보호
의료기기	사용자 입력 영역과 제어 영역 분리
RTOS 기반 애플리케이션	User Task → Privileged System 침입 차단

📦 실전 구성 예

```
[ Region 0 ] Flash (RO, Executable)
[ Region 1 ] SRAM (RW)
[ Region 2 ] Heap (RW, No-Exec)
[ Region 3 ] Stack (RW, Subregion 0 = No Access)
[ Region 4 ] Peripherals (RW)
```

→ 특정 영역만 **Unprivileged Task에서 접근 가능**하게 설정

📌 요약 정리

항목	설명
MPU란	메모리 영역별 접근 권한/실행 제한 설정 장치
MMU와 차이	주소 변환 없음, 단순 권한 보호 전용
활용 목적	시스템 안정성, RTOS 보호, 스택 오버플로우 탐지
Cortex-M 적용	STM32F4, F7, H7 시리즈 등
실습 효과	접근 제한 오류 → HardFault 발생으로 즉시 감지 가능
RTOS 연계	Task마다 Stack 보호, Kernel 분리 실행

16.3 IoT 프로세서 설계 및 보안

🧠 IoT 시스템의 프로세서 요구사항

항목	요구 조건
저전력	배터리 기반 수년간 구동 가능
무선 통신	BLE, ZigBee, LoRa, WiFi 등 내장
보안	데이터 암호화, 키 관리, 부팅 보호
통합도	센서/무선/CPU를 단일 SoC로 통합
실시간성	RTOS 구동, 타이머/인터럽트 필수

🔧 대표적인 IoT용 프로세서 예

프로세서	설명
ESP32	WiFi + BLE, 듀얼코어 Xtensa, AES/RTC 내장
STM32WB55	BLE5 + Cortex-M4/M0+ 듀얼코어, Secure Boot 지원
Nordic nRF52840	BLE5.2 + USB, TrustZone-M, CryptoCell
RA4M2 (Renesas)	Arm Cortex-M33 + TrustZone, Secure Crypto
K210 (Sipeed)	RISC-V + KPU, No WiFi but Edge AI 지원

📦 IoT 보안 설계 핵심 요소

항목	설명
Secure Boot	부팅 시 펌웨어 서명 검증
Crypto Engine	AES, SHA, RSA, ECC 하드웨어 가속
TRNG (True RNG)	키 생성용 하드웨어 난수
Key Storage	OTP, Secure Flash, eFuse
TrustZone-M	Secure/Non-secure 영역 분할
Anti-Tamper	외부 개입 탐지 (전압, 주파수 등)

🧩 보안 아키텍처 예: STM32WB55

```
+-----------------------------------+
| Cortex-M0+ (Wireless Stack)       |
| Cortex-M4 (Application)           |
| ┌──────────────┐                  |
| | Secure Flash | ← TrustZone      |
| | Secure Boot  | ← Root of Trust  |
| | AES-256, RNG | ← HW Crypto      |
| └──────────────┘                  |
| OTA Update, BLE5 Stack            |
+-----------------------------------+
```

→ BLE 커넥션도 M0+에서 수행, 사용자 앱은 분리된 M4에서 실행

🧠 Secure Boot 개념

단계	설명
1️⃣ 부트로더 → SHA256 해시	
2️⃣ 공개키로 서명 검증 (RSA, ECC)	

단계	설명
3️⃣ OK → 실행 / FAIL → 무한 루프	
4️⃣ TrustZone 활성 시 Secure World 진입	

🔒 하드웨어 기반 보안 모듈

모듈	기능
eFuse	변경 불가한 OTP 설정, 키 저장
PUF (Physically Unclonable Function)	칩 고유의 난수 발생
Tamper Pins	침입 감지 시 초기화 트리거
Flash Read/Write Protection	특정 섹터 접근 금지 설정

🧪 OTA 보안 업데이트 절차

1. **OTA 서버로부터 서명된 펌웨어 다운로드**
2. **디바이스 내 Secure Boot에서 서명 검증**
3. **정상 시 Flash에 저장 후 재부팅**
4. **이중 영역 (A/B Bank) 방식으로 롤백 가능**

→ `MbedTLS`, `WolfSSL`, `TinyCrypt` 같은 경량 TLS 스택과 함께 사용

📶 통신 보안

계층	보호 방법
물리/링크	BLE pairing + AES-CCM
전송 계층	TLS 1.2/1.3 (mTLS 가능)
앱 계층	MQTT with TLS, CoAP with DTLS

→ 대부분 **TLS over TCP**, **DTLS over UDP** 방식

🔄 MCU ↔ NPU/모듈 연동 예시

```
[ MCU (STM32, ESP32) ]
    └─ SPI/UART
         ↓
[ Secure NPU / Radio ]
    └─ BLE 인증/암호화 수행
```

→ 민감 연산을 **MCU가 아닌 Crypto 블록이나 NPU로 위임**

📌 요약 정리

항목	설명
IoT 프로세서 조건	저전력 + 무선 통신 + 보안 하드웨어
보안 모듈	Secure Boot, Crypto Engine, TrustZone, eFuse
OTA 보안	서명 검증 + 이중 영역 펌웨어 구조
통신 보호	TLS, DTLS, BLE 인증, ECC 기반 키 교환
SoC 예시	ESP32, STM32WB, nRF52, Renesas RA4, K210

16.4 엣지 컴퓨팅과 RISC-V 확장

🧠 엣지 컴퓨팅이란?

> 클라우드에 의존하지 않고, 데이터를 생성하는 현장(엣지)에서 직접 처리

특징	설명
지연 최소화	센서 입력 → 로컬 추론 → 제어까지 수 ms 이내
네트워크 부담 감소	전체 데이터 업로드 대신 **요약 데이터**만 전송
실시간성 & 안정성	네트워크 불안정 상황에서도 독립적 판단 가능
보안 향상	민감 데이터가 로컬에서만 처리됨

🧩 엣지 프로세서가 갖춰야 할 조건

요소	설명
MCU 수준 제어 능력	GPIO, PWM, UART, SPI 등
AI 추론 능력	CNN, RNN, FFT, anomaly detection
보안 기능	Secure Boot, 암호화, 인증
저전력	배터리 기반 환경 대응
확장성	주변 장치 + 무선 통신 연계

🧠 RISC-V가 엣지에 적합한 이유

특징	설명
오픈 ISA	라이선스 비용 없음, 자유롭게 확장 가능
모듈화 설계	RV32/64, F/D, V, M, C 등 조합 가능

특징	설명
사용자 정의 명령어 지원	AI 가속, 보안 연산 등 전용 명령 삽입 가능
초경량 SoC 구현 가능	IoT, 웨어러블용 MCU에도 적합
자율 설계	TSMC, GF 등 파운드리와 직접 통합 가능

🛠️ RISC-V 명령어 확장 예

```
RV32IMAC + CustomX
```

- CustomX: 예를 들어 `dotmul.x a1, a2, a3`
  → 4-element vector dot product
  → RTL 수준에서 연산기 추가 + binutils 수정

→ AI용 MAC, DSP용 FFT, 보안용 AES 연산기를 직접 명령어로 통합 가능

📦 대표적인 RISC-V 엣지 SoC/보드

SoC	특징	활용 예
Sipeed K210	RV64 + KPU + FFT + AES	영상 추론, AIoT
ESP32-C3	RV32IMC + WiFi/BLE	저가 IoT 디바이스
Allwinner D1	RV64GCV + Linux	LoRa Gateway, 엣지 분석
SiFive E31/E51	Core IP, 커스텀 가능	자율차, 스마트센서
VisionFive 2	RV64 SoC + GPU + Linux	엣지 서버, 로봇 AI

🧪 엣지 노드 시스템 예

```
[ Sensor ]
    ↓ (SPI)
[ RISC-V MCU (e.g. ESP32-C3) ]
    ├─ GPIO / PWM 제어
    ├─ AES + ECC 보안 연산
    ├─ KPU (Edge AI 추론)
    └─ UART/WiFi ↔ Cloud

[ 특징 ]: 모든 연산을 local MCU에서 처리, 클라우드는 결과 업로드만
```

🔐 보안 처리 통합 예

항목	적용 방식
Secure Boot	SHA256 + ECC 검증
Flash 암호화	SoC 내부 키 기반 AES

항목	적용 방식
키 인증	PUF 기반 Key Derivation
통신 보호	TLS/DTLS with RISC-V crypto extension

🔄 RISC-V 개발 툴체인 & 생태계

도구	설명
GCC for RISC-V	기본 toolchain (`riscv64-unknown-elf-gcc`)
LLVM/Clang	Custom ISA 대응도 우수
Freedom E SDK	SiFive 기반 개발 SDK
OpenOCD	디버깅용
QEMU	시뮬레이션 (RV32, RV64 가능)
Verilator	명령어 검증, 확장 설계 가능

📊 성능 및 응용 분석

응용	요구	RISC-V 기반 실현성
영상 인식	CNN, ResNet	KPU, RVV 지원 가능
음성 인식	FFT, MFCC, LSTM	DSP 확장
진동 분석	anomaly detection	int8 추론용 TinyML
실시간 제어	타이머, 인터럽트	MCU Core 대응
OTA 보안	SHA256, ECC	RV32 + Crypto 확장으로 처리

📌 요약 정리

항목	내용
엣지 컴퓨팅	데이터를 현장에서 처리, 지연/보안 문제 해결
RISC-V 장점	오픈 ISA, 저전력/확장성, 명령어 커스터마이징
엣지용 SoC 예시	ESP32-C3, K210, D1, VisionFive, SiFive IP
보안 통합	Secure Boot, AES, ECC, Flash 암호화
툴체인	GCC, Clang, QEMU, OpenOCD, Verilator 등
응용 사례	IoT 센서 처리, TinyML 추론, 실시간 제어, 보안 통신

16.5 TrustZone 및 하드웨어 보안 기능

🧠 TrustZone이란?

ARM TrustZone은
하나의 SoC에서 **신뢰 가능한 영역(Secure World)**과
일반 영역(Non-Secure World)을 **하드웨어적으로 분리**해주는 기술이다.

구분	설명
Secure World	키, 인증, 보안 연산 수행
Non-Secure World	일반 앱, 드라이버, 네트워크 등 실행
전환 방식	SMC(System Management Call) 명령어 사용 (A 프로세서)
메모리 분리	Secure/NonSecure 영역을 **MPU/SAU/NIC**에서 강제 설정

🧩 TrustZone-M (Cortex-M 계열)

> ARMv8-M에서 도입된 **초경량 임베디드 보안 아키텍처**

특징	설명
적용 대상	Cortex-M23, M33, M35P 등
Secure / NS 코드 동시 실행	RTOS 지원 가능 (FreeRTOS NS + Secure RTOS)
SAU (Security Attribution Unit)	메모리 보안 속성 설정
NSC (Non-Secure Callable)	Secure → NS 함수 게이트 정의
인터럽트 분리	Secure / NS 벡터 테이블 분리

🔐 TrustZone-M의 메모리 구조 예

```
Flash         (Secure Boot + Key)
SRAM Secure   (Crypto, stack)
SRAM Non-Sec  (RTOS, 앱)
Peripherals:
  - Secure UART, RNG, AES
  - NS GPIO, UART
```

→ 메모리, 주변장치, 인터럽트를 **보안 경계로 나눔**

🛡️ 주요 하드웨어 보안 기능

기능	설명
Secure Boot	펌웨어 서명 검증 후 실행
eFuse / OTP	하드웨어 키 영구 저장

기능	설명
TRNG	인증/암호용 난수 생성기
PUF (Physically Unclonable Function)	칩 고유 ID 기반 키 생성
Tamper Detection	전압 조작, 클럭 위조 탐지
Memory Protection Unit (MPU)	실행/쓰기 보호
Flash Read Protection	디버거, 외부에서 접근 차단

✅ 실전 예: STM32L5 (Cortex-M33 TrustZone)

영역	설명
Secure Boot	OEM 키 기반 서명 검증 (OEM-iROT)
Secure Firmware Update (SFU)	OTA 업데이트 지원
SAU + IDAU	Secure/Non-Secure 주소 매핑
AES/HASH/PKA	하드웨어 가속기
SBSFU	ST 제공 Secure Boot + Firmware Update 예제

✅ 실전 예: nRF5340 (Nordic, Cortex-M33 듀얼)

- M33 Application Core (NS): BLE, App 실행
- M33 Network Core (S): Secure boot, Crypto 연산 전담
- Key storage: KMU (Key Management Unit)
- TrustZone + CryptoCell 통합

🔧 Secure Boot 절차 (일반화)

1. **ROM Stage**
    - Secure Boot 로더 실행 (RO 영역)
    - SHA256으로 펌웨어 해시 계산
    - RSA/ECC 공개키로 서명 검증
2. **Secure World 진입**
    - Secure OS 초기화 (옵션)
    - Non-Secure 앱에 제어 전달

📩 Secure ↔ Non-Secure 전환 예 (TrustZone-M)

```
// Secure function 선언
__attribute__((cmse_nonsecure_entry))
void SecurePrint(const char* msg) {
    printf("[SECURE]: %s\n", msg);
}

// Non-Secure에서 호출
typedef void (*SecurePrint_t)(const char* msg);
SecurePrint_t SecureCall = (SecurePrint_t)SECURE_ADDRESS;
SecureCall("Hello");
```

> 전환은 `cmse_nonsecure_call()` 경유 → 런타임에서 보안 검사

📐 보안 경계 설계 예 (IoT)

```
[ Secure World ]
  - Secure Boot
  - 암호화 엔진
  - 인증 키
  - OTA 검증

[ Non-Secure World ]
  - RTOS / MQTT
  - 센서 제어
  - 디스플레이 처리
```

→ Secure 영역은 외부에서 접근 불가

📌 요약 정리

항목	설명
TrustZone	SoC 내 Secure/Non-Secure 세계 분리
TrustZone-M	ARMv8-M 기반 Cortex-M33 전용 보안 기능
Secure Boot	펌웨어 위조 방지
Crypto Engine	HW 가속 (AES/SHA/RSA/ECC)
eFuse / OTP / PUF	보안 키 저장 및 파생
Tamper Detection	전압 조작 등 감지
응용 예	IoT, 의료기기, 산업 자동화, 드론 등

16.6 양자 내성 프로세서 아키텍처

🧠 왜 양자 내성(Post-Quantum)인가?

양자 컴퓨터는 **현재의 공개키 암호(PKC)** 체계인
RSA, ECC, DSA 등을 **Shor 알고리즘**으로 빠르게 해독 가능하다.

영향 받는 암호	상태
RSA, DH	💥 취약 (Shor's algorithm)
ECC	💥 취약
AES, SHA-2	✅ 대체로 안전 (단, Grover 알고리즘으로 키 길이 2배 필요)

→ Post-Quantum Cryptography(PQC)는 **양자 공격에도 안전한 암호 체계**

🔐 주요 PQC 알고리즘 유형

유형	예시	특징
Lattice 기반	Kyber, Dilithium	빠름, 효율적, 대부분 하드웨어 가속 대상
Code 기반	Classic McEliece	큰 키, 검증된 안전성
Multivariate	Rainbow	서명 특화 (실제선정 탈락)
Hash 기반	SPHINCS+	매우 안정적, 다소 느림

✅ NIST PQC 표준 선정 결과:

- **Kyber** (KEM): 선택됨
- **Dilithium** (Signature): 선택됨
- **SPHINCS+** (Hash-based Sig): 보조
- **Classic McEliece**: 장기 검증용

🧩 PQC가 기존 SoC와 다른 점

항목	기존 암호 엔진 (RSA/ECC)	PQC 가속 구조
연산 구조	모듈러 연산기, 곱셈기	폴리노미얼 곱, 격자 샘플링, NTT
키 크기	수백 바이트	수 KB (Kyber, McEliece는 수십 KB 이상)
연산 시간	~ms 단위	수 ms ~ 수백 ms
하드웨어 자원	작음	메모리/버스 대역폭 더 필요

🧠 PQC 하드웨어 가속기 구성 요소

```
+-----------------------+
|    PQC Accelerator    |
| ┌─────────────────┐   |
| | LWE Engine      | ← Kyber/Dilithium용 격자 기반 구조
| | NTT Unit        | ← Number Theoretic Transform
| | Sampler RNG     | ← Noise 생성 (TRNG/DRBG 연계)
| | Polynomial MUL  | ← PolyVec 연산 최적화
| └─────────────────┘   |
+-----------------------+
```

→ 키 생성 / 암호화 / 서명 연산을 하드웨어에서 실행 → 속도, 소비전력 개선

🛠️ 실리콘 구현 예

SoC / IP	설명
PQShield IP	Kyber/Dilithium/NTT 기반 하드웨어 블록
Infineon TRAVEO T2G	PQC 알고리즘 평가 내장
NXP LPC55S3x	PQC-ready Crypto 엔진 탑재
ARM CryptoCell	향후 PQC 확장 예정
FPGAs (Artix, Zynq 등)	Lattice 서명/암호화 가속기 시뮬레이션 구현 중

📦 PQC 연산 예: Kyber512 (KEM)

```
Alice (MCU):
    - Generate Keypair → Public key (1.6KB)
    - Send PK to Bob

Bob (Server):
    - Encapsulate → Ciphertext + Shared Secret
    - Send Ciphertext to Alice

Alice:
    - Decapsulate → Shared Secret

→ Shared Secret을 AES Key로 사용 (Session Key)
```

→ 이 과정을 PQC 가속기로 수행 시,
CPU 부하 감소 + 10배 이상 속도 향상 가능

🔐 PQC 통합 보안 아키텍처 예

```
[ MCU with TrustZone ]
    ├─ Secure Boot
    ├─ PQC Engine (Kyber/Dilithium)
    ├─ Legacy ECC/RSA Engine
    ├─ TRNG / PUF
    ├─ Flash 암호화 + Secure OTA
    └─ PQ-TLS Stack (mTLS with Kyber)
```

→ TLS 인증 시 RSA → Kyber, 서명 시 ECDSA → Dilithium으로 대체

📚 개발 툴과 오픈소스

도구	설명
liboqs (Open Quantum Safe)	PQC C 라이브러리 (Kyber, Dilithium 등)
mbedTLS + liboqs	TLS + PQC 혼합 지원
OpenSSL 3.x + PQC	실험적 Kyber 지원
pqcrypto-crystals	Kyber/Dilithium 구현체
RISC-V + PQC IP	OpenTitan, OpenHW 기반 확장 가능

📌 요약 정리

항목	내용
양자 내성 암호(PQC)	RSA, ECC 등 고전 암호의 대체 수단
표준 알고리즘	Kyber (KEM), Dilithium (Signature), SPHINCS+
하드웨어 설계 핵심	NTT, PolyMul, 샘플링 RNG
MCU 통합 예	PQC 엔진 + Secure Boot + Flash 암호화
활용 분야	IoT, 자동차, 정부 시스템, 장기 보관 데이터 보안 등