# MPU (Memory Protection Unit)

---

## 1. 개요

### 1.1. 목적

- 펌웨어가 저장된 메모리 영역을 권한이 없는 사용자로부터 쓰기 권한을 부여하여 공격자가 펌웨어, remote attestation을 구현한 영역을 변조하지 못하게 함

### 1.2. 주요 기능

- 메모리 영역 별 접근 권한 정의
    - 펌웨어가 저장되는 영역의 접근 권한을 privileged only read/write로 설정

### 1.3. Privileged & Unprivileged

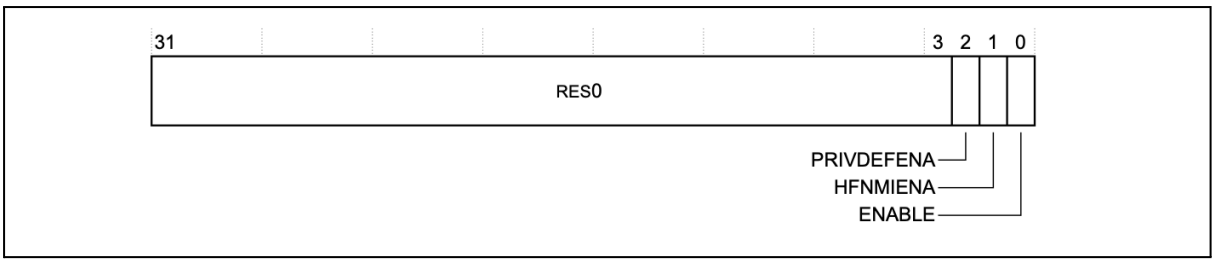
- Secure Extension(TrustZone)과 함께 사용하면 secure world와 non-secure world 각각에 대해 privileged와 unprivileged 권한에 대한 읽기/쓰기 권한 설정 가능

- privileged code와 unprivileged code 각각에 대한 읽기/쓰기 권한 설정 가능
  메모리 영역을 아래와 같이 4가지 경우로 설정 가능
    - privileged only : read/write 혹은 read-only
    - any privileged : read/write 혹은 read-only

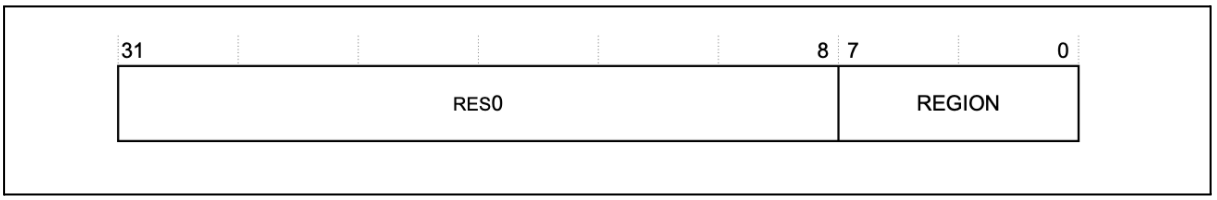
- ARM 아키텍처에서는 CONTROL 레지스터를 설정해 privileged/unprivileged 상태를 제어 가능

### 1.4. 레지스터

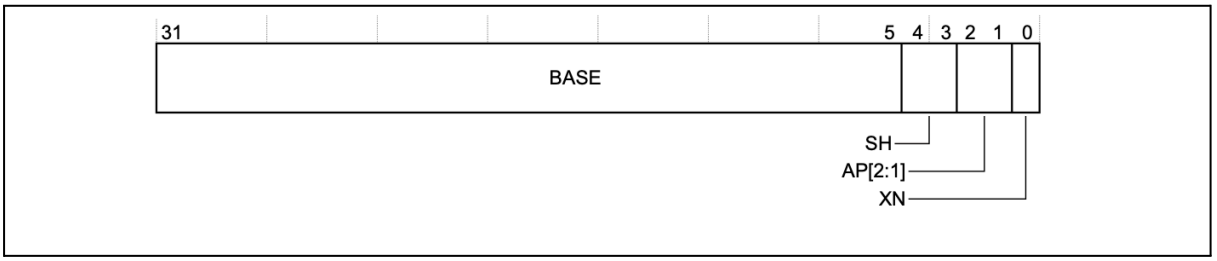
1. MPU_CTRL
    - 설명 : MPU 활성화/비활성화 조작
    - 주소 : 0xE000ED94
    - PRIVDEFENA
    - HFNMIENA :
    - ENABLE : MPU 활성화/비활성화



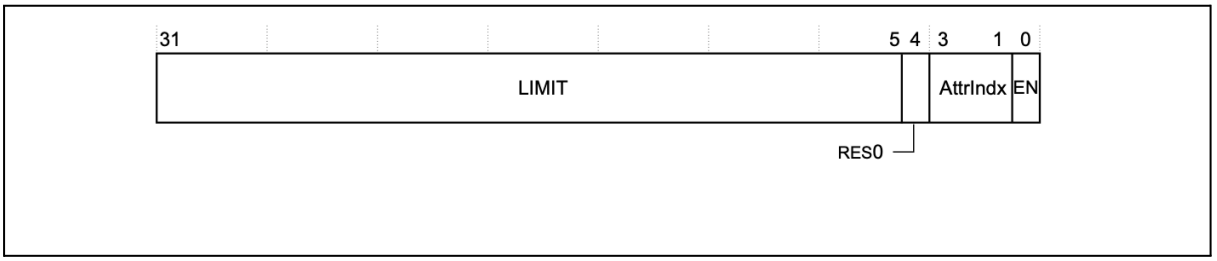

2. MPU_RNR
   - 설명 : MPU_RBAR, MPU_RLAR로 설정할 메모리 영역의 번호 지정
   - 주소 : 0xE000ED98
   - REGION : 메모리 영역 번호



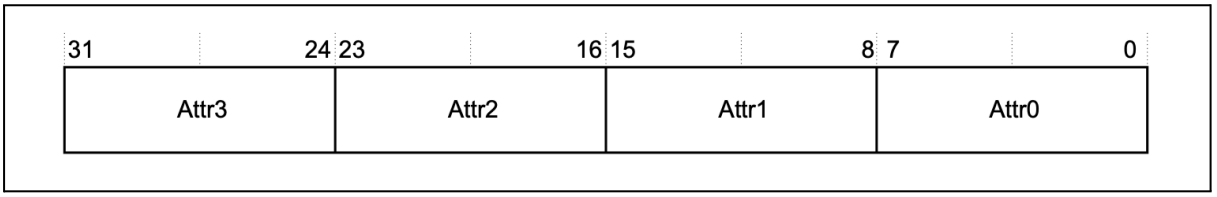

3. MPU_RBAR
   - 설명 : 설정할 메모리 영역의 시작 주소 정의
   - 주소 : 0xE000ED9C
   - BASE : 메모리 영역의 시작 주소
   - SH : shareability 설정
   - AP : 접근 권한 설정
   - XN : 실행 가능/불가능 설정




4. MPU_RLAR
   - 설명 : 설정할 메모리 영역의 끝 주소 정의
   - 주소 : 0xE000EDA0
   - LIMIT : 메모리 영역의 끝 주소
   - AttrIndx : 메모리 속성의 index
   - EN : 영역 활성화 / 비활성화



5. MPU_MAIR
    - 설명 : 메모리 속성를 encoding
    - 주소 : 0xE000EDC0




---

# 2. High-Level

## 2.1. MPU 설정 흐름

1. MPU 비활성화
2. 메모리 속성 설정 - MPU_MAIR0
3. 메모리 구역 번호 설정 - MPU_RNR
4. 설정할 메모리 구역 시작 주소 설정 - MPU_RBAR
5. 설정할 메모리 구역 끝 주소 설정 - MPU_RLAR
6. MPU 활성화

---

# 3. Module-Level

직접 레지스터를 제어하는 함수를 작성해 MPU 설정

## 3.1. void MPU_Enable (void)

MPU_CTRL 레지스터의 Enable 영역을 1으로 만들어 MPU를 활성화하는 함수

## 3.2. void MPU_Disable (void)

MPU_CTRL 레지스터의 Enable 영역을 0으로 만들어 MPU를 비활성화하는 함수

## 3.3. void MPU_Config (void)

2.1. 의 설정 흐름에 맞춰 각 레지스터를 설정해주는 함수

# 4. Source-Level

## 4.1. mpu_config.h

MPU 레지스터 설정에 필요한 매크로를 정의한 헤더 파일

```
// MPU Register Pointer
#define MPU_BASE    0xE000ED90UL

#define MPU_TYPE    (*(volatile uint32_t *)(MPU_BASE + 0x00))
#define MPU_CTRL    (*(volatile uint32_t *)(MPU_BASE + 0x04))
#define MPU_RNR     (*(volatile uint32_t *)(MPU_BASE + 0x08))
#define MPU_RBAR    (*(volatile uint32_t *)(MPU_BASE + 0x0C))
#define MPU_RLAR    (*(volatile uint32_t *)(MPU_BASE + 0x10))
#define MPU_MAIR0   (*(volatile uint32_t *)(MPU_BASE + 0x30))
#define MPU_MAIR1   (*(volatile uint32_t *)(MPU_BASE + 0x34))

// Set Bit
#define SET_BIT(REG, BIT)       ((REG) |= (BIT))
#define CLEAR_BIT(REG, BIT)     ((REG) &= ~(BIT))
#define READ_BIT(REG, BIT)      ((REG) & (BIT))
#define CLEAR_REG(REG)          ((REG) = 0x0)
#define WRITE_REG(REG, VAL)     ((REG) = (VAL))
#define READ_REG(REG)           ((REG))
#define MODIFY_REG(REG, CLEARMASK, SETMASK)  \
   WRITE_REG((REG), (((READ_REG(REG)) & (~(CLEARMASK))) | (SETMASK)))

// CTRL Register
#define MPU_CTRL_ENABLE_Pos     0U
#define MPU_CTRL_ENABLE_Msk     (0x1UL << MPU_CTRL_ENABLE_Pos)
#define MPU_CTRL_HFNMIENA_Pos   1U
#define MPU_CTRL_HFNMIENA_Msk   (0x1UL << MPU_CTRL_HFNMIENA_Pos)
#define MPU_CTRL_PRIVDEFENA_Pos 2U
#define MPU_CTRL_PRIVDEFENA_Msk (0x1UL << MPU_CTRL_PRIVDEFENA_Pos)

// RBAR Register
#define MPU_RBAR_BASE_Pos       5U
#define MPU_RBAR_BASE_Msk       (0x7FFFFFFUL << MPU_RBAR_BASE_Pos)
```

```
#define MPU_RBAR_SH_Pos          3U
#define MPU_RBAR_SH_Msk          (0x3UL << MPU_RBAR_SH_Pos)
#define MPU_RBAR_AP_Pos          1U
#define MPU_RBAR_AP_Msk          (0x3UL << MPU_RBAR_AP_Pos)
#define MPU_RBAR_XN_Pos          0U
#define MPU_RBAR_XN_Msk          (0x1UL << MPU_RBAR_XN_Pos)

// RLAR Register
#define MPU_RLAR_LIMIT_Pos       5U
#define MPU_RLAR_LIMIT_Msk       (0x7FFFFFFUL << MPU_RLAR_LIMIT_Pos)
#define MPU_RLAR_AttrIndx_Pos    1U
#define MPU_RLAR_AttrIndx_Msk    (0x7UL << MPU_RLAR_AttrIndx_Pos)
#define MPU_RLAR_EN_Pos          0U
#define MPU_RLAR_EN_Msk          (0x1UL << MPU_RLAR_EN_Pos)
```

## 4.2. void MPU_Enable (void)

WRITE_REG 매크로를 이용해 MPU_CTRL 레지스터의 enable 영역을 1로 설정하는 함ㅅ

```
void MPU_Enable(void)
{
   // MPU Enable
   uint32_t ctrl_value = 0;
   ctrl_value |= MPU_CTRL_ENABLE_Msk;
   WRITE_REG(MPU_CTRL, ctrl_value);
}
```

## 4.3. void MPU_Disable (void)

CLEAR_BIT 매크로를 이용해 MPU_CTRL 레지스터의 enable 영역을 0으로 설정하는 함수

```
void MPU_Disable(void)
{
   CLEAR_BIT(MPU_CTRL, MPU_CTRL_ENABLE_Msk);
}
```

## 4.4. void MPU_Config (void)

Attribute Index 0에 Normal memory, write-through, read allocate 속성을 적용
Memory 영역 번호 0에 code flash 부분을 할당
Non-shareable에 privileged 상태에서 읽기/쓰기 권한과 실행 가능으로 설정한 위에서 작성한 속성 인덱스 0을 적용

```
void MPU_Config(void)
{
   // MPU Disable
   MPU_Disable();

   // Configure Flash Region
   // Memory Attribute 0
   MPU_MAIR0 &= ~(0xFFUL << 0);  // 하위 8비트 클리어
   MPU_MAIR0 |= (0xAAUL << 0);   // Normal memory, Write-through, Read
allocate

   // Memory Region 0
   WRITE_REG(MPU_RNR, 0);

   // Flash Base Address
   uint32_t rbar_value = 0;
   rbar_value |= (0x08000000UL & MPU_RBAR_BASE_Msk);  // 시작 주소
   rbar_value |= (0x0UL << MPU_RBAR_SH_Pos);          // Non-shareable
   rbar_value |= (0x2UL << MPU_RBAR_AP_Pos);          // 읽기 권한
   rbar_value &= ~MPU_RBAR_XN_Msk;                    // 실행 가능
   WRITE_REG(MPU_RBAR, rbar_value);

   // Flash Limit Address
   uint32_t rlar_value = 0;
   rlar_value |= (0x0807FFFFUL & MPU_RLAR_LIMIT_Msk); // 끝 주소
   rlar_value |= (0x0UL << MPU_RLAR_AttrIndx_Pos);    // 속성 인덱스 0
   rlar_value |= MPU_RLAR_EN_Msk;                     // 영역 활성화
   WRITE_REG(MPU_RLAR, rlar_value);

   // MPU Enable
   MPU_Enable();
}
```

## 5. 참고자료


- ARM Developer Document
  https://developer.arm.com/documentation/107565/0101/Memory-protection?lang=en

- Introduction to memory protection unit management on STM32 MCUs
  https://www.st.com/resource/en/application_note/an4838-introduction-to-memory-protection-unit-management-on-stm32-mcus-stmicroelectronics.pdf

- STM32 Cortex-M33 MPU Programming Manual
  https://www.st.com/resource/en/programming_manual/pm0264-stm32-cortexm33-mcus-and-mpus-programming-manual-stmicroelectronics.pdf