FLASH Program

2010.06.17

Flash memory size

	Main memory block	Information block
low-density	4 Kbits × 64 bits	
medium-density	**16 Kbits × 64 bits**	**258 × 64 bits**
high-density	64 Kbits × 64 bits	
connectivity line	32 Kbits × 64 bits	2306 × 64 bits

```
/* Define the STM32F10x FLASH Page Size depen
ding on the used STM32 device */
#ifdef STM32F10X_LD
  #define FLASH_PAGE_SIZE    ((uint16_t)0x400)
#elif defined STM32F10X_MD
  #define FLASH_PAGE_SIZE    ((uint16_t)0x400)
#elif defined STM32F10X_HD
  #define FLASH_PAGE_SIZE    ((uint16_t)0x800)
#elif defined STM32F10X_CL
  #define FLASH_PAGE_SIZE    ((uint16_t)0x800)
#endif /* STM32F10X_LD */
```

- Flash memory의 크기는 최대 512 Kbytes
- STM32F103RB는 128 Kbytes
- Memory 구성은 Main memory block과 Information block으로 나뉜다.
- 각 크기는 디바이스 종류에 따라서 달라지게 된다

Address structure

Block	Name	Base addresses	Size (bytes)
Main memory	Page 0	0x0800 0000 - 0x0800 03FF	1 Kbyte
	Page 1	0x0800 0400 - 0x0800 07FF	1 Kbyte
	Page 2	0x0800 0800 - 0x0800 0BFF	1 Kbyte
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 Kbyte
	Page 4	0x0800 1000 - 0x0800 13FF	1 Kbyte
	⋮	⋮	⋮
	Page 127	0x0801 FC00 - 0x0801 FFFF	1 Kbyte
Information block	System memory	0x1FFF F000 - 0x1FFF F7FF	2 Kbytes
	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16

- Flash memory는 32 비트 폭 – code와 data를 저장
- 메모리맵 상에서 base address가 이미 지정되어 있다.
- information block은 System memory와 Option Bytes
  - System memory는 System memory boot mode로 디바이스 부팅용 미리 boot loader를 넣어두는 곳으로 사용하고 있다.
  - Option bytes는 사용자 데이터나 Read/Write Protection에 대한 제어 용도로 사용된다.

Erase Programming Time

Symbol	Parameter	Conditions	Min(1)	Typ	Max(1)	Unit
t_{prog}	16-bit programming time	T_A = –40 to +105 °C	40	52.5	70	µs
t_{ERASE}	Page (1 KB) erase time	T_A = –40 to +105 °C	20		40	ms
t_{ME}	Mass erase time	T_A = –40 to +105 °C	20		40	ms

- Flash memory에 대한 쓰기 작업 동안에는 읽기를 시도하는 것이 bus를 지연시키게 된다.
- 읽기 작업은 쓰기 작업이 완전히 종료한 상태에서 진행될 수 있다.
- **Flash memory**에 대한 쓰기나 지우는 작업이 진행되는 동안은 **code**나 **data**의 **fetch** 작업이 진행될 수 없다는 것을 의미

Read operation

- 내부 Flash는 일반적인 memory 공간과 마찬가지로 주소를 직접 주어서 접근이 가능하다. **(*(__IO uint32_t*) Address)** 와 같은 방식으로 직접 Flash의 값의 내용을 읽어볼 수 있다.

- I-Code interface – Instruction fetch
  - Prefetch buffer – 2개의 블록. 각 블록이 8 bytes로 구성
  - Prefetch block은 direct-mapped 되어 있다.
  - 한 블럭이 Flash memory에 대한 한번의 읽기로 완전히 replace 된다.
  - 블럭의 크기는 Flash memory의 bandwidth와 일치
  - Flash memory의 크기 16 Kbits × 64 bits – 여기서 64 비트가 의미하는 것이 Flash memory의 bandwidth를 의미
- D-Code interface – Data
  - D-code bus는 prefetch 접근보다 더 높은 priority를 갖는다.

Flash program & erase controller (FPEC)

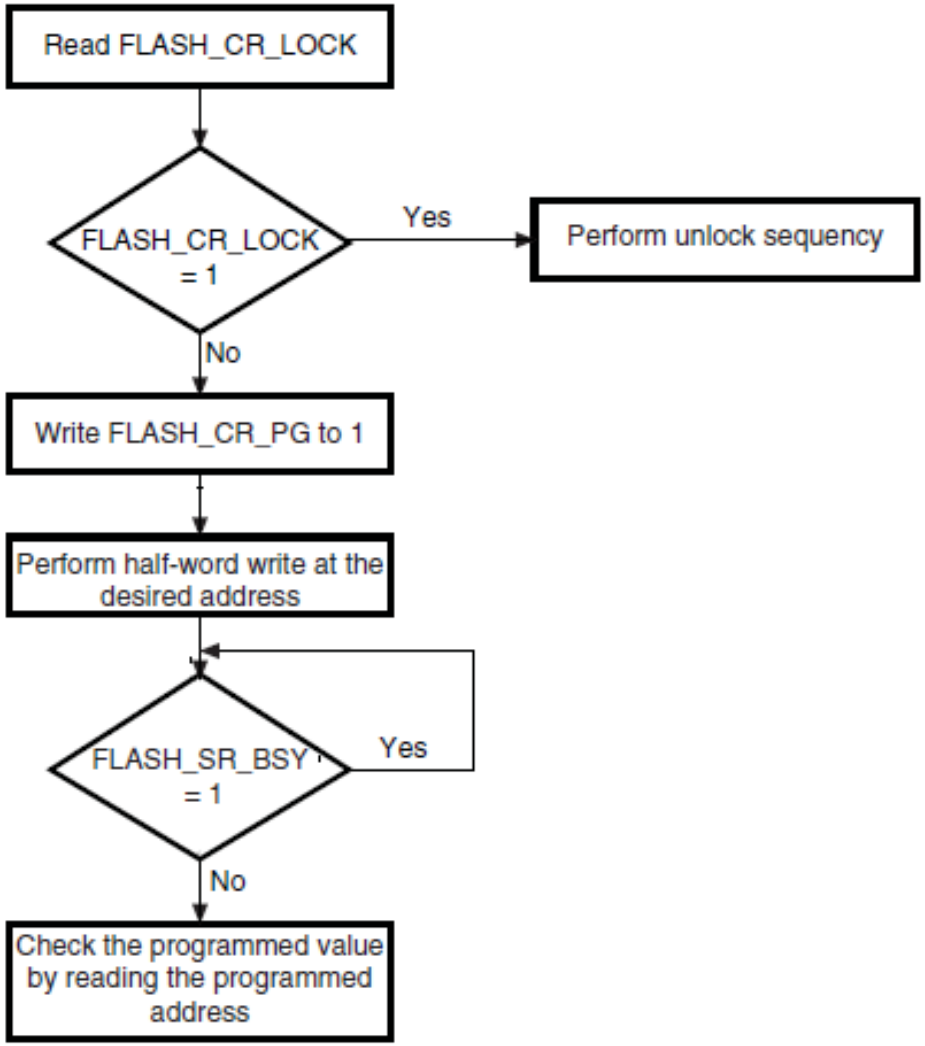
- FPEC block은 Flash memory에 대한 program과 erase 작업을 관장
- FPEC는 다음의 7개의 32 비트 레지스터들로 구성되어 있다.
- Flash 관련 레지스터 8개 중에서 FLASH_ACR 레지스터를 제외한 7개

- **FPEC key register (FLASH_KEYR)**
- **Option byte key register (FLASH_OPTKEYR)**
- **Flash control register (FLASH_CR)**
- **Flash status register (FLASH_SR)**
- **Flash address register (FLASH_AR)**
- **Option byte register (FLASH_OBR)**
- **Write protection register (FLASH_WRPR)**

Flash Unlocking

- 리셋이 된 이후 FPEC block은 protect 된다.
- FPEC block 사용을 위해서는 **unlocking sequence**를 수행해야 한다.
  - FLASH_KEYR register에 씀으로써 이루어진다.
  - unlocking sequence는 2개의 Key values를 FLASH_KEYR register에 차례로 적어 넣는 동작. KEY1 = 0x45670123, KEY2 = 0xCDEF89AB를 차례로 적어 넣는 것
- 반대로 FPEC block을 lock을 시키는 작업은 FLASH_CR register의 LOCK bit에 1을 쓰는 것
  - Flash control register (FLASH_CR) Bit 7 LOCK은 lock과 관련한 제어를 하는 비트.
  - 이 비트에 1을 쓰는 것은 FPEC와 FLASH_CR 레지스터를 lock시킨다는 것
  - 이 비트는 unlock sequence를 감지하면 하드웨어에 의해 0으로 reset
  - 만약 unlock sequence가 실패하게 되면 이 비트는 다음 리셋 때까지 계속 1 상태를 유지하게 된다.

Programming procedure



- main Flash memory는 한번에 16 bits를 program 할 수 있다.
- Program을 하기 전에 반드시 FLASH_CR register의 PG bit는 1로 set 되어 있어야 한다.
- program 과정은 CPU가 main Flash memory address로 half-word를 write 할 때 시작된다.
- program 과정이 시작되어 (BSY bit set) read/write operation이 시작되면 CPU는 Flash memory program이 끝날 때까지 멈춘다.
- PG 비트를 1로 설정한 이후에 쓰기 작업을 수행하고 BSY 비트가 1인가를 계속 검사해서 1이 아닐 때까지 기다린다.

FLASH_WaitForLastOperation

```
FLASH_Status FLASH_WaitForLastOperation(uint32_t Timeout)
{
  FLASH_Status status = FLASH_COMPLETE;

  /* Check for the Flash Status */
  status = FLASH_GetStatus();
  /* Wait for a Flash operation to complete or a TIMEOUT to occur */
  while((status == FLASH_BUSY) && (Timeout != 0x00))
  {
    delay();
    status = FLASH_GetStatus();
    Timeout--;
  }
  if(Timeout == 0x00 )
  {
    status = FLASH_TIMEOUT;
  }
  /* Return the operation status */
  return status;
} ? end FLASH_WaitForLastOperation ?
```

```
FLASH_Status FLASH_GetStatus(void)
{
  FLASH_Status flashstatus = FLASH_COMPLETE;

  if((FLASH->SR & FLASH_FLAG_BSY) == FLASH_FLAG_BSY)
  {
    flashstatus = FLASH_BUSY;
  } else {
    if((FLASH->SR & FLASH_FLAG_PGERR) != 0){
      flashstatus = FLASH_ERROR_PG;
    } else {
      if((FLASH->SR & FLASH_FLAG_WRPRTERR) != 0 ){
        flashstatus = FLASH_ERROR_WRP;
      } else {
        flashstatus = FLASH_COMPLETE;
      }
    }
  }
  /* Return the Flash Status */
  return flashstatus;
} ? end FLASH_GetStatus ?
```

FLASH_ProgramWord

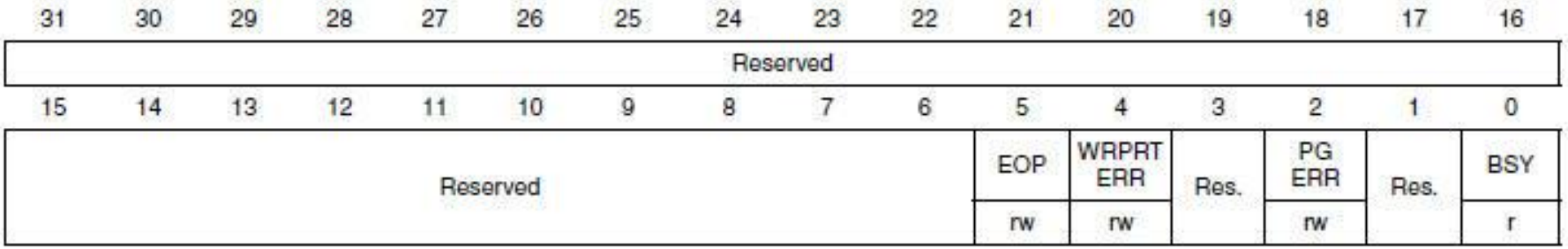
```
FLASH_Status FLASH_ProgramWord(uint32_t Address, uint32_t Data) {
  FLASH_Status status = FLASH_COMPLETE;
  __IO uint32_t tmp = 0;

  assert_param(IS_FLASH_ADDRESS(Address));  /* Check the parameters */
  /* Wait for last operation to be completed */
  status = FLASH_WaitForLastOperation(ProgramTimeout);

  if(status == FLASH_COMPLETE) {
    /* if the previous operation is completed, proceed to program the new first half word */
    FLASH->CR |= CR_PG_Set;

    *(__IO uint16_t*)Address = (uint16_t)Data;
    /* Wait for last operation to be completed */
    status = FLASH_WaitForLastOperation(ProgramTimeout);

    if(status == FLASH_COMPLETE) {
      /* if the previous operation is completed, proceed to program the new second half word */
      tmp = Address + 2;
      *(__IO uint16_t*) tmp = Data >> 16;

      /* Wait for last operation to be completed */
      status = FLASH_WaitForLastOperation(ProgramTimeout);

      if(status != FLASH_TIMEOUT) {
        FLASH->CR &= CR_PG_Reset; /* Disable the PG Bit */
      }
    } else {
      if (status != FLASH_TIMEOUT) {
        FLASH->CR &= CR_PG_Reset; /* Disable the PG Bit */
      }
     }
  } ? end if status==FLASH_COMPLETE ?
  /* Return the Program Status */
  return status;
} ? end FLASH_ProgramWord ?
```

Flash status register (FLASH_SR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										EOP	WRPRT ERR	Res.	PG ERR	Res.	BSY
										rw	rw		rw		r

- FLASH_ProgramWord는 Address와 data를 받아서 저장하는 기능
  - 32 비트를 받아서 16 비트씩 2번에 걸쳐서 write 작업을 수행
- FLASH_ProgramWord에서는 FLASH_WaitForLastOperation 함수를 3번 부르고 있다.
  - 이 함수는 Flash에 대한 이전 작업이 현재 어떤 상태인가를 검사하는 함수.
  - 첫 번째 호출은 FLASH_ProgramWord() 함수가 불리기 전에 어떤 Flash 작업이 있을 수 있기 때문에 그 작업이 정상적으로 끝날 때까지 기다리는 것이고, 두 번째 호출은 첫 16 비트를 저장한 후 종료 될 때까지 기다리는 것이고, 마지막 호출은 나머지 16 비트를 저장한 후 종료 될 때까지 기다리는 것이다.
- FLASH_WaitForLastOperation 함수는 결국 FLASH_GetStatus() 함수를 호출해서 그 결과를 알려주는 것이고, 결국은 위 레지스터의 비트 값들을 비교하는 것이다.

Flash status register (FLASH_SR)

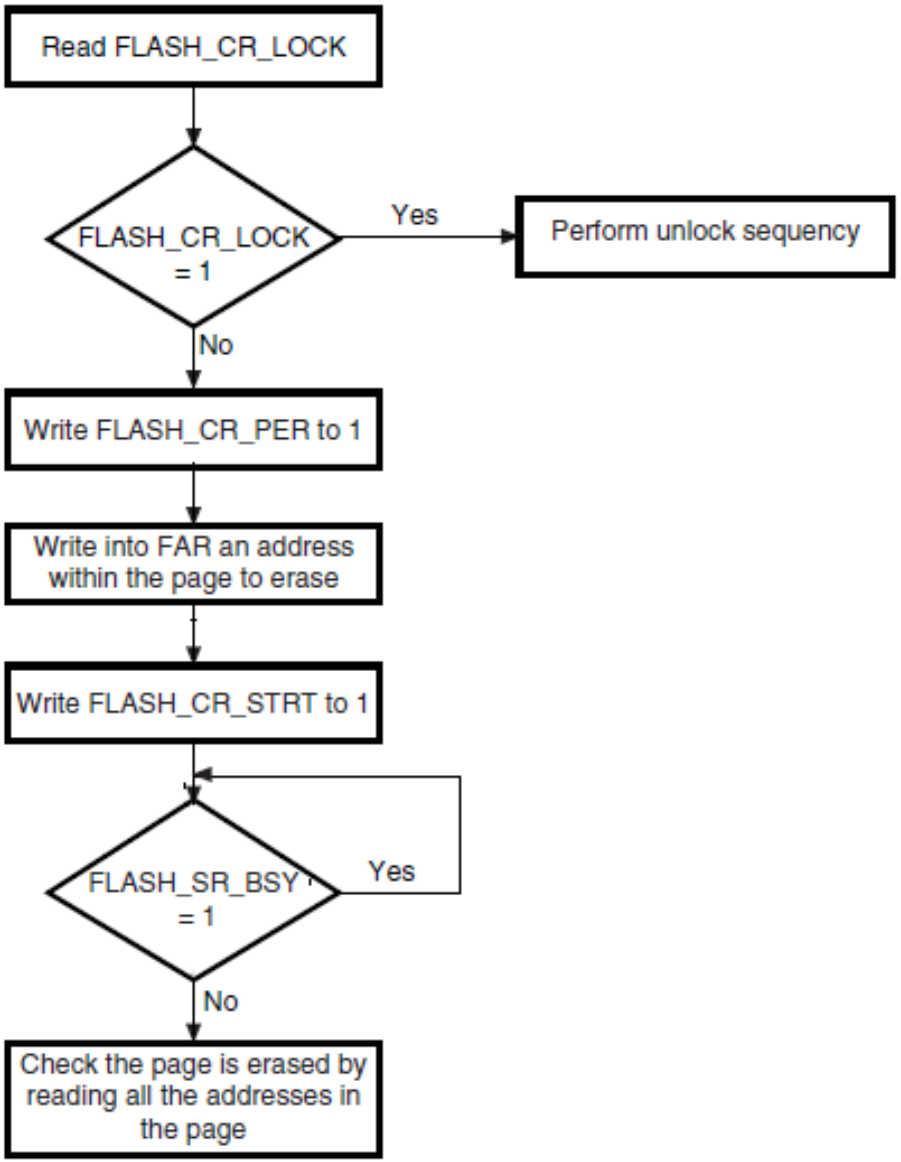


- Bit 0 BSY: Busy
  - 1은 Flash operation이 현재 이루어지고 있다는 것. Flash operation 시작 부분에서 1로 설정되고, Flash operation이 종료하고 나면 혹은 어떤 에러가 발생하게 되면 0으로 리셋. read only 비트.
- Bit 2 PGERR: Programming error
  - 쓰려는 영역이 **Erase**되어 있지 않으면 에러 발생
  - program되는 값이 16비트 값이 아닌 경우에 에러 발생
- Bit 4 WRPRTERR: Write protection error
  - program 하려는 주소값이 write-protected 되어있을 경우 에러 발생
- Bit 5 EOP: End of operation
  - Programming, Erase 같은 Flash operation이 종료하게 되면 1로 설정
- PGERR, WRPRTERR, EOP는 1을 write 함으로써 reset 시킬 수 있다.

Program 과정 정리

- FLASH_CR register의 PG bit를 1로 설정한 이후
- 16비트 half-word를 쓰게 된다.
- 이때 해당 영역이 erase되었는지를 검사하게 된다.
- 만약 erase 되어 있지 않다면 FLASH_SR register PGERR bit가 설정
- 그 영역이 write-protected되어 있으면 FLASH_SR register의 WRPRTERR bit가 설정
- program operation이 끝나면 FLASH_SR register EOP bit가 설정

Flash memory Erase – Page Erase

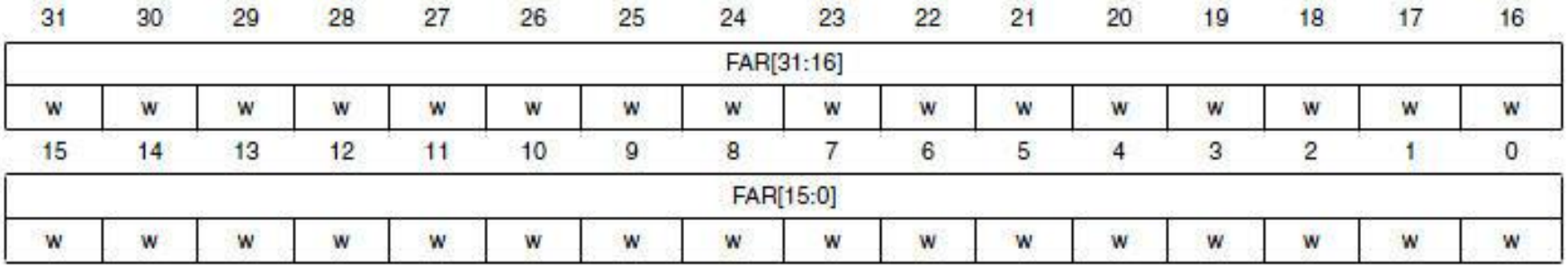


- Erase는 page 단위로 수행할 수도 있고, 전체를 한꺼번에 지울 수도 있다. 각각 **Page Erase**와 **Mass Erase**로 구분한다.
- Flash memory Lock 검사
- Flash memory operation이 동작하고 있지 않은가 검사
- **FLASH_CR register의 PER (Page erase) bit를 1로 설정**
- **FLASH_AR register에 page의 시작 주소값**을 적어 넣는다.
- **FLASH_CR register의 STRT bit를 1로 설정**
- 이후는 program 작업과 동일하게 BSY bit가 0이 될 때까지 기다리는 동작을 수행

Flash memory Erase – Page Erase

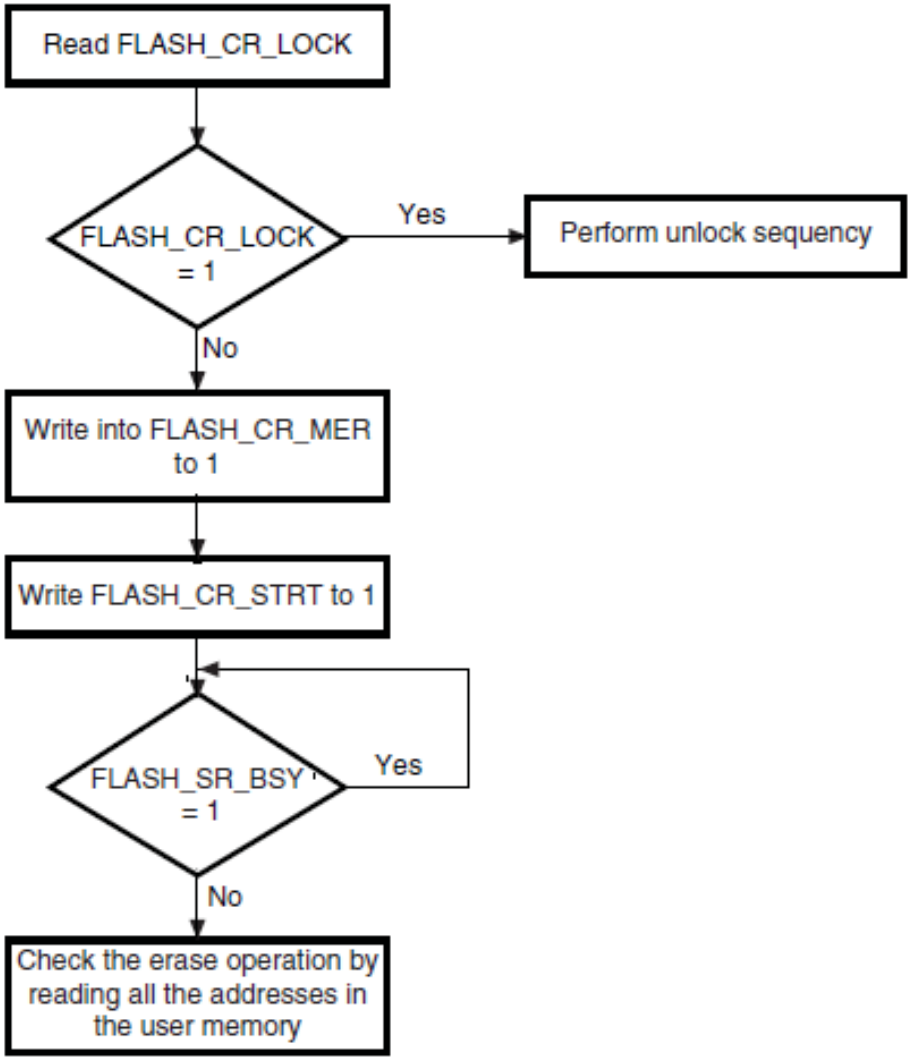
```
FLASH_Status FLASH_ErasePage(uint32_t Page_Address)
{
  FLASH_Status status = FLASH_COMPLETE;
  /* Check the parameters */
  assert_param(IS_FLASH_ADDRESS(Page_Address));
  /* Wait for last operation to be completed */
  status = FLASH_WaitForLastOperation(EraseTimeout);

  if(status == FLASH_COMPLETE)
  {
    /* if the previous operation is completed, proceed to erase the page */
    FLASH->CR|= CR_PER_Set;
    FLASH->AR = Page_Address;
    FLASH->CR|= CR_STRT_Set;

    /* Wait for last operation to be completed */
    status = FLASH_WaitForLastOperation(EraseTimeout);
    if(status != FLASH_TIMEOUT)
    {
      /* if the erase operation is completed, disable the PER Bit */
      FLASH->CR &= CR_PER_Reset;
    }
  }
  /* Return the Erase Status */
  return status;
} ? end FLASH_ErasePage ?
```

Flash address register (FLASH_AR)



- Bits 31:0 FAR: Flash Address
- 가장 최근에 사용된 혹은 현재 사용하고 있는 Flash의 주소값
- 하드웨어가 자동으로 이 레지스터에 값을 적어 놓는다.
- 다만 **Page Erase** 작업을 수행할 경우만 소프트웨어에서 주소값을 적을 수 있다.

Flash memory Erase – Mass Erase



- 전체 Flash를 지우는 동작 수행
- Page Erase과 달라지는 부분이 주소값 설정 – 주소 설정 필요 없음
- Flash memory Lock 검사
- Flash operation 동작 검사
- **FLASH_CR register의 MER bit를 1로 설정**
- **FLASH_CR register의 STRT bit를 1로 설정**
- BSY bit가 0이 될 때까지 기다림
- **Mass Erase가 information block을 지우지는 않는다**

Flash memory Erase – Mass Erase

```
FLASH_Status FLASH_EraseAllPages(void)
{
  FLASH_Status status = FLASH_COMPLETE;
  /* Wait for last operation to be completed */
  status = FLASH_WaitForLastOperation(EraseTimeout);

  if(status == FLASH_COMPLETE)
  {
    /* if the previous operation is completed, proceed to erase all pages */
    FLASH->CR |= CR_MER_Set;
    FLASH->CR |= CR_STRT_Set;

    /* Wait for last operation to be completed */
    status = FLASH_WaitForLastOperation(EraseTimeout);
    if(status != FLASH_TIMEOUT)
    {
      /* if the erase operation is completed, disable the MER Bit */
      FLASH->CR &= CR_MER_Reset;
    }
  }
  /* Return the Erase Status */
  return status;
} ? end FLASH_EraseAllPages ?
```

예제 프로그램 시나리오

- 최초 리셋이 되었을 때 Flash memory Program/Erase Controller 부분은 lock 되어 있다.
- 먼저 이 lock을 unlock 시킨다.

- 0x08008000 부터 0x0800A000까지의 Flash 영역을 Erase
- 모든 지운 영역에 0x15041979라는 임의의 값을 기록
- 기록을 모두 마친 후에 위의 영역을 모두 검사해서 0x15041979 값이 정확히 읽히는지 확인

Flash_Test

```
void Flash_Test(void) {
    FLASH_Unlock(); /* Unlock the Flash Program Erase controller */
    NbrOfPage = (EndAddr - StartAddr) / FLASH_PAGE_SIZE; /* Define the number of page to be erased */

    /* Clear All pending flags */
    FLASH_ClearFlag(FLASH_FLAG_BSY | FLASH_FLAG_EOP | FLASH_FLAG_PGERR | FLASH_FLAG_WRPRTERR);

    /* Erase the FLASH pages */
    for(EraseCounter = 0;
          (EraseCounter < NbrOfPage) && (FLASHStatus == FLASH_COMPLETE); EraseCounter++) {
        FLASHStatus = FLASH_ErasePage(StartAddr + (FLASH_PAGE_SIZE * EraseCounter));
    }

    /*  FLASH Word program of data 0x15041979 at addresses defined by StartAddr and EndAddr*/
    Address = StartAddr;

    while((Address < EndAddr) && (FLASHStatus == FLASH_COMPLETE)) {
        FLASHStatus = FLASH_ProgramWord(Address, Data);
        Address = Address + 4;
    }

    Address = StartAddr; /* Check the corectness of written data */
    while((Address < EndAddr) && (MemoryProgramStatus != FAILED)) {
        if((*(__IO uint32_t*) Address) != Data) {
            MemoryProgramStatus = FAILED;
            printf("[FAILED] Address: 0x%0X, Data: 0x%0X\n", Address, (*(__IO uint32_t*) Address));
        }
        Address += 4;
    }

    if(PASSED == MemoryProgramStatus) { printf("Memory check PASSED\n");
    } else {        printf("Memory check FAILED\n"); }

    while (1) { }
} ? end Flash_Test ?
```