Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

a. ARM Cortex-M4 Manual 분석하기

< arm 어셈블리 함수 >

* add r3,r1,r2
→ r3 = r1+ r2
* **subgt** r3,r1,r2
→ r3= r1-r2
* **rsble** r3,r1,r2 (reversee subtract)
→ r3 =r2 – r1
* **and** r3,r1,r2
→ r3= r1&r2
* **biceq** r3,r1,r2
→ r3=r1&~r2
* **orr** r3,r1,r2
→ r3 = r1|r2
***eors** r3,r1,r2 (exclusive or)
→ r3 = r1^r2
***mov** r1,r2
→ r1 = r2
***mvn** r1,r2
→ r1 = 0xffffffff^r2
***cmp** r1,r2
→ r1 – r2 하여 비교후 state flag 업데이트
***cmn** r1,r2 (cmp negative)
→ r1 + r2 하여 비교
***tsteq** r1,r2
→ r1 & r2 하여 비교 후 조건 flag 업데이트
***teq** r1,r2
→ r1 ^ r2 하여 비교 후 조건 flag 업데이트

산술 명령어 뒤에 접미사 gt ge lt le eq 의미 (status flag상태를 보고 실행할지 말지 결정한다)

gt : great than

ge : great equal //크거나 같다

lt : less than

le: less equal // 작거나 같다.

eq : equal // 같다.

ne : not equal

b: byte

산술 명령어 뒤에 접미사 s 가 붙을 경우 → 산술 처리 결과에 따라 flag 업데이트 한다.

*** ! 느낌표 들어간 문장 이해하기

arm 어셈블리 함수 오늘 추가 분

*mov r0, #0xff, 8 : lotation shift
→ arm은 32bit 단위이다. 0Xff = 0x00 00 00 ff → 우측 8비트 시프트 시킨다. 0x 00 00 00 00 //ff 돌아서 상위 비트로 물린다. 즉 0x ff 00 00 00 00

*r1, lsl r2 : logical shift leg
→ r1 * 2^r2

*r1, asr , r2
→ r1 / 2^r2

*mul r1,r2,r3
→ r1 = r2 * r2

*mla r1,r2,r3,r4 : dsp는 곱셈과 덧셈을 동시에 진행 할 수 있다. r2와 r3를 곱하고 거기에 r4를 더해라. 이 모두가 1 clock에 끝난다.
→ r1 = r2 * r3 + r4

*umull r0,r1,r2,r3 : Unsigned Long 곱하기, 32 비트 피연산자 및 64 비트 결과
→ r1 r0 = r2*r3

*umlal r0,r1,r2,r3
→ r1 r0 = r2*r3 + (r0, r1) → r1 = 상위비트 8bit+ r1, r0 = 하위8비트 + r0

*ldr r0,[r1,#0x4] : load 레지스터 (reg → mem)
→ r0 = *(r1 + 4byte)
*ldreqb r0,[r1,#0x5] : eq 는 z플래그 확인
→ r0 = *(r1 + 5 byte)

*ldr r0, [r1],r2 : r1은 주소이다. r2는 byte크기이다.
→ r0 = *r1 , r1 = r1 +r2

***str : store 레지스터 (mem → reg)**

***strb r0,[r1] : b는 byte**
→ *r1 = r0

***stmia r0!,{r1,r2,r3}** : store multiful , r0 memory 에 r1, r2, r3레지스터값을 r0 에 보내고 위치 값을 갱신
→ r0 = {r1,r2,r3}

***ldmia r0,{r4,r5,r6} : memory → reg**
→ r0가 가리키는 메모리 값을 r4, r5, r6 에 차례대로 넣음.

***mrs r0 cpsr : cpsr, sqrs의 값을 arm 의 범용 레지스터로 읽어 온다.**
→ r0= cpsr , 특수 인터럽트 플래그 비트 레지스터에 접근이 가능하다. msr은 그 반대

***msr cpsr r0 : arm의 값을 cpsr, sqrs 의 레지스터에 쓴다.**
→ cpsr = r0 , r0값을 cpsr reg로 전달

mrs 없으면 cpsr값에 접근할 수 없다. Cpsr레지를 mrs와 같은 별도 명령어로 관리를한다. → 키보드나 마우스를 사용도 인터럽트인데, 사용자가 임의로 cpsr레지를 건드리면 인터럽트 반응이 안되기 때문에 아무나 움직이지 못하도록 운영체제가 관리하는 별도의 instruction을 만들어 둔것.

***어셈블리 사용하는 이유??**
하나의 예로 인터럽트 껐다 켰다 하는 것은 운영체제가 한다. 이때 사용하는 프로그래밍은 어셈블리로만 가능하다.

***ldr , *str**
어셈블리어 레벨에서 볼 때 **load store register는** 핵심이다. **load store register 가** 존재하는 이유가 뭘까?
intel은 memory to memory 연산이 가능하다. arm은 불가능하기 때문에 memory에 있는 정보를 레지스터에 가져오고 다시 메모리에 집어 넣어야 한다.
→ load는 레지스터에 가져오는 오는 것이고, store는 레지스터에 때려 넣는 것이다. Ok?

Manual 보는법.

처음 메모리 0번지부터 시작한다. arm의 경우 인터럽트 100번 table에 리셋 핸들러가 걸려있었다.(arm -vector table _ reset handler가 동작)
Cortex M4 mcu들어가보는데, 리셋핸들러의 동작을 메뉴얼과 함께 확인해본다.

Cortex M4: 단순함. 12MHz
Cortex R : 상당히 복잡함. 멀티코어(cpu 2개). 400MHz → 내일모레함.
Atmega: 멀티코어가 없다.
아두이노: 8MHz
cortex A- 페이징하니까 레지스터 실시간 처리 불가, 대용량 메모리 처리할 때 사용.

arm info : http://infocenter.arm.com/help/index.jsp

Cortex M4 Technical Reference Manual
: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf

Cortex™ -M4 Devices Generic User Guide
: http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf

```
; Reset handler
Reset_Handler   PROC
                EXPORT  Reset_Handler             [WEAK]
        IMPORT  SystemInit
        IMPORT  __main

                ;FPU settings
                LDR     R0, =0xE000ED88          ; Enable CP10,CP11
                LDR     R1, [R0]
                ORR     R1,R1,#(0xF << 20)
                STR     R1, [R0]

                LDR     R0, =SystemInit
                BLX     R0
                LDR     R0, =__main
                BX      R0
                ENDP

; Dummy Exception Handlers (infinite loops which can be modified)

NMI_Handler     PROC
                EXPORT  NMI_Handler               [WEAK]
                B       .
                ENDP
HardFault_Handler₩
                PROC
                EXPORT  HardFault_Handler         [WEAK]
                B       .
```

그림 ①: Boot code 1

Address	Name	Type	Reset	Description
0xE000ED88	CPACR	RW	-	Coprocessor Access Control Register

********** Boot code **********

PROC 어셈블리 Reset_Handler 함수 시작쿠
EXPORT Reset_Handler 함수 외부에서 사용할 수 있도록 수출
IMPORT SystemInit, main 함수 불러오기

(*LDR = 메모리에서 레지스터로 가져오는 것)
LDR R0, = 0xE000ED88
→ R0 = 0xE000ED88 → <0xE000ED88 ARM 데이터시트 찾아보면 >

********** 1. Cortex M4 Technical Reference Manual **********

Table 4-1 System control registers : 프로세스의 불량 요소를 제어하는 레지스터를 조사한다.

→ CPCR이라는 것이 있네 확인해보자.

Coprocessor Access Control Register

The CPACR register specifies the access privileges for coprocessors. See the register summary in *Cortex-M4F floating-point system registers* for its attributes. The bit assignments are:

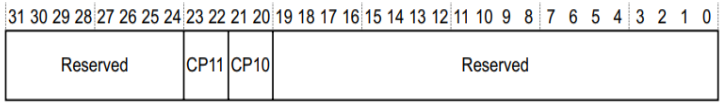

Table 4-50 CPACR register bit assignments

Bits	Name	Function
[31:24]	-	Reserved. Read as Zero, Write Ignore.
[2*n*+1:2*n*] for *n* values10 and 11	CP*n*	Access privileges for coprocessor *n*. The possible values of each field are: 0b00 = Access denied. Any attempted access generates a NOCP UsageFault. 0b01 = Privileged access only. An unprivileged access generates a NOCP fault. 0b10 = Reserved. The result of any access is Unpredictable. 0b11 = Full access.
[19:0]	-	Reserved. Read as Zero, Write Ignore.

4.6.6 Enabling the FPU

The FPU is disabled from reset. You must enable it before you can use any floating-point instructions. Example 4-1 shows an example code sequence for enabling the FPU in both privileged and user modes. The processor must be in privileged mode to read from and write to the CPACR.

Example 4-1 Enabling the FPU

```
; CPACR is located at address 0xE000ED88
LDR.W   R0, =0xE000ED88
; Read CPACR
LDR     R1, [R0]
; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR     R1, R1, #(0xF << 20)
; Write back the modified value to the CPACR
STR     R1, [R0]; wait for store to complete
DSB
;reset pipeline now the FPU is enabled
ISB
```

bootcode → 0XE000ED88

[31:24], [19:0] 예약되어있고,[23:20] 비트에는 cp11,cp10이라는 레지가 들어있다. 이 reg는 무엇일까?

arm 에는 mmu 같은 것을 제어하기 위해 별도의 하드웨어가 있다. cp라는 coprocessor가 존재한다.이 녀석은 cp0 – cl15 프로세스 동작을 서포터 해주기 위한 기능들을 가지고 있다. 그중의 cp11 과 cp10 레지스터가 있다.

CP11, CP10 기능 – floating point(부동소수점)를 사용할 수 있게 해준다.

Mcu → 쉽게 연산,control을 목적으로하는 cpu 즉, MMU 가 있고, 가상메모리 또한 사용한다.

0b00	Access denied → x
0b01	Privileged access only → 커널만 접근가능
0b10	Reserved → 예약, 예측 불가
0b11	Full access → 사용자, 커널 둘 모두 부동소수점 사용가능

0.1, 0.2 와같은 소수점을 이제 사용할 수 있는 것.

CPACR 을 사용하는 방법도 나와있다

FPU를 활성화하는 일련 과정을 보여줌. 커널과 사용자 모두 사용가능함

```
; Reset handler
Reset_Handler   PROC
                EXPORT  Reset_Handler             [WEAK]
        IMPORT  SystemInit
        IMPORT  __main

                ;FPU settings
                LDR     R0, =0xE000ED88           ; Enable CP10,CP11
                LDR     R1,[R0]
                ORR     R1,R1,#(0xF << 20)
                STR     R1,[R0]

                LDR     R0, =SystemInit
                BLX     R0
                LDR     R0, =__main
                BX      R0
                ENDP

; Dummy Exception Handlers (infinite loops which can be modified)

NMI_Handler     PROC
                EXPORT  NMI_Handler               [WEAK]
                B       .
                ENDP
HardFault_Handler₩
                PROC
                EXPORT  HardFault_Handler         [WEAK]
                B       .
```

********** Boot code **********

LDR R1,[R0]
→ R1 = *R0 (포인터연산)

ORR R1,R1,#(0xF << 20)
→ R1 = R1 or 0xf00000
→ 20~23 bit (CP10, CP11 reg) 를 1로 set

STR R1,[R0] : (*STR =레지스터를 메모리로 다시 내보내기)
→ *R0 = R1

LDR R0, =SystemInit : 함수이기 때문에 주소이다.
→ R0에 함수 SystemInit 함수 주소 값 넣기

BLX R0 : 함수들어가기
→ 복귀주소를 저장하고, R0주소의 함수로 넘어간다.

그림 1에서 LDR R0, =SystemInit SystemInit함수, 아래 그림2를 들여다 보자.

```
/**
  * @brief  Micro Controller System을 설정한다.
  *         Embedded Flash Interface, PLL을 초기화하고 SystemFrequency 변수를 갱신한다.
  * @param  None
  * @retval None
  */
void SystemInit(void)
{
  /* RCC Clock 구성을 Default Reset State로 reset(재설정)한다. */
  /* Set HSION bit */
  RCC->CR |= (uint32_t)0x00000001;

  /* Reset CFGR register */
  RCC->CFGR = 0x00000000;

  /* Reset HSEON, CSSON and PLLON bits */
  RCC->CR &= (uint32_t)0xFEF6FFFF;

  /* Reset PLLCFGR register */
  RCC->PLLCFGR = 0x24003010;

  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFF;

  /* 모든 Interrupt를 비활성화한다. */
  RCC->CIR = 0x00000000;

#ifdef DATA_IN_ExtSRAM
  SystemInit_ExtMemCtl();
#endif /* DATA_IN_ExtSRAM */

  /* System Clock Source, PLL 곱셈기, 나눗셈기, AHB/APBx Prescalers와 Flash 설정을 구성한다. */
  SetSysClock();

  /* Offset Address를 더한 Vector Table 위치를 구성한다. */
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* 내부 SRAM에 Vector Table 재배치 */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* 내부 FLASH(NAND)에 Vector Table 재배치 */
#endif
}
```

그림 ②: boot code2 .SystemInit함수

********** SystemInit Func **********

SystemInit 함수: MCL 설정, flash 메모리, PLL 초기화, 동작주파수 설정

RCC → CR | = (uint32_t)0x00000001 로 설정하고있다.
RCC가 뭘까?
RCC 살펴보니 구조체로 되어있다. 이 구조체는 Peripheral 제어함.

```
typedef struct
{
__IO uint32_t CR; /*!< RCC clock control register, Address offset: 0x00 */
__IO uint32_t PLLCFGR; /*!< RCC PLL configuration register, Address offset: 0x04 */
__IO uint32_t CFGR; /*!< RCC clock configuration register, Address offset: 0x08 */
__IO uint32_t CIR; /*!< RCC clock interrupt register, Address offset: 0x0C */
__IO uint32_t AHB1RSTR; /*!< RCC AHB1 peripheral reset register, Address offset: 0x10 */
__IO uint32_t AHB2RSTR; /*!< RCC AHB2 peripheral reset register, Address offset: 0x14 */
__IO uint32_t AHB3RSTR; /*!< RCC AHB3 peripheral reset register, Address offset: 0x18 */
uint32_t RESERVED0; /*!< Reserved, 0x1C */
__IO uint32_t APB1RSTR; /*!< RCC APB1 peripheral reset register, Address offset: 0x20 */
__IO uint32_t APB2RSTR; /*!< RCC APB2 peripheral reset register, Address offset: 0x24 */
uint32_t RESERVED1[2]; /*!< Reserved, 0x28-0x2C */
__IO uint32_t AHB1ENR; /*!< RCC AHB1 peripheral clock register, Address offset: 0x30 */
__IO uint32_t AHB2ENR; /*!< RCC AHB2 peripheral clock register, Address offset: 0x34 */
__IO uint32_t AHB3ENR; /*!< RCC AHB3 peripheral clock register, Address offset: 0x38 */
uint32_t RESERVED2; /*!< Reserved, 0x3C */
__IO uint32_t APB1ENR; /*!< RCC APB1 peripheral clock enable register, Address offset: 0x40 */
__IO uint32_t APB2ENR; /*!< RCC APB2 peripheral clock enable register, Address offset: 0x44 */
uint32_t RESERVED3[2]; /*!< Reserved, 0x48-0x4C */
__IO uint32_t AHB1LPENR; /*!< RCC AHB1 peripheral clock enable in low power mode register, Address offset: 0x50 */
__IO uint32_t AHB2LPENR; /*!< RCC AHB2 peripheral clock enable in low power mode register, Address offset: 0x54 */
__IO uint32_t AHB3LPENR; /*!< RCC AHB3 peripheral clock enable in low power mode register, Address offset: 0x58 */
uint32_t RESERVED4; /*!< Reserved, 0x5C */
__IO uint32_t APB1LPENR; /*!< RCC APB1 peripheral clock enable in low power mode register, Address offset: 0x60 */
__IO uint32_t APB2LPENR; /*!< RCC APB2 peripheral clock enable in low power mode register, Address offset: 0x64 */
uint32_t RESERVED5[2]; /*!< Reserved, 0x68-0x6C */
__IO uint32_t BDCR; /*!< RCC Backup domain control register, Address offset: 0x70 */
__IO uint32_t CSR; /*!< RCC clock control & status register, Address offset: 0x74 */
uint32_t RESERVED6[2]; /*!< Reserved, 0x78-0x7C */
__IO uint32_t SSCGR; /*!< RCC spread spectrum clock generation register, Address offset: 0x80 */
__IO uint32_t PLLI2SCFGR; /*!< RCC PLLI2S configuration register, Address offset: 0x84 */
} RCC_TypeDef;
```

: 이 핀에 어떤 신호가 들어가느냐에 따라 어떤출력이나올지, or Peripheral 장치들이 다르게 동작한다.

1. RCC

```
RCC → CR | = (uint32_t)0x00000001
```

→ HSION 비트를 1로 셋팅하여 HIS ocilator on

6.3 RCC registers

Refer to *Section 1.1: List of abbreviations for registers* for a list of abbreviations used in register descriptions.

6.3.1 RCC clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		PLLSAIRDY	PLLSAION	PLLI2SRDY	PLLI2SON	PLLRDY	PLLON	Reserved				CSSON	HSEBYP	HSERDY	HSEON
		r	rw	r	rw	r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSIRDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Bit 0 **HSION**: Internal high-speed clock enable
Set and cleared by software.
Set by hardware to force the HSI oscillator ON when leaving the Stop or Standby mode or in case of a failure of the HSE oscillator used directly or indirectly as the system clock. This bit cannot be cleared if the HSI is used directly or indirectly as the system clock.
0: HSI oscillator OFF
1: HSI oscillator ON

ocilator(발진기, 크리스탈 집어넣으면 주파수가 나오기 시작함)
저항과 콘덴서를 사용한 RC 발진기라고 부른다.

2.3 Memory map

See the datasheet corresponding to your device for a comprehensive diagram of the memory map. *Table 1* gives the boundary addresses of the peripherals available in all STM32F4xx devices.

Table 1. STM32F4xx register boundary addresses

Boundary address	Peripheral	Bus	Register map
0xA000 0000 - 0xA000 0FFF	FSMC control register (STM32F405xx/07xx and STM32F415xx/17xx)/ FMC control register (STM32F42xxx and STM32F43xxx)	AHB3	*Section 36.6.9: FSMC register map on page 1601* *Section 37.8: FMC register map on page 1681*
0x4002 3800 - 0x4002 3BFF	RCC	AHB1	*Section 7.3.24: RCC register map on page 266*
0x4002 3000 - 0x4002 33FF	CRC		*Section 4.4.4: CRC register map on page 116*
0x4002 2800 - 0x4002 2BFF	GPIOK		*Section 8.4.11: GPIO register map on page 288*
0x4002 2400 - 0x4002 27FF	GPIOJ		
0x4002 2000 - 0x4002 23FF	GPIOI		

2. memory map와 같이 보기.
이주소에 해당하는 주소를 쓰면 여기 걸려있는 어떠한 peripheral이 동작을 하게 된다.

Peripheral 이 주소 값을 가지는 이유?
미리 셋팅된 결과가 아니다. FPGA로 설계함.

6.3.3 RCC clock configuration register (RCC_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2		MCO2 PRE[2:0]			MCO1 PRE[2:0]			I2SSCR	MCO1		RTCPRE[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPRE2[2:0]			PPRE1[2:0]			Reserved		HPRE[3:0]				SWS1	SWS0	SW1	SW0
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	r	r	rw	rw

Bit 24 **PLLON**: Main PLL (PLL) enable

Set and cleared by software to enable PLL.

Cleared by hardware when entering Stop or Standby mode. This bit cannot be reset if PLL clock is used as the system clock.

0: PLL OFF

1: PLL ON

Bit 19 **CSSON**: Clock security system enable

Set and cleared by software to enable the clock security system. When CSSON is set, the clock detector is enabled by hardware when the HSE oscillator is ready, and disabled by hardware if an oscillator failure is detected.

0: Clock security system OFF (Clock detector OFF)

1: Clock security system ON (Clock detector ON if HSE oscillator is stable, OFF if not)

Bit 16 **HSEON**: HSE clock enable

Set and cleared by software.

Cleared by hardware to stop the HSE oscillator when entering Stop or Standby mode. This bit cannot be reset if the HSE oscillator is used directly or indirectly as the system clock.

0: HSE oscillator OFF

1: HSE oscillator ON

RCC→ CFGR = 0x0000 0000;

RCC clock configuration register

: clock 을 어떤식으로 구성할지 결정하는 레지스터

Reset value : 0 → 리셋 하는 이유는?

: 하드웨어 구동 시 다른 회로가 플로팅 되는 것을 방지한다.

24,19,16 bit →

1111 1110 1111 0110 FFFF

RCC → CR &= (uint_t)FEF6 FFFF

*발진기들의 특성

오실레이터들의 → RC회로 특성, 높은 전압이 들어가면 팍 튀었다가 일정한 주파수로 유지된다. → 일정한 주파수로 되기 까지 delay 시간이 있다. HSERDY 비트(발진,전압이 확들어면 보드가 탐)는 이 준비기간동안 꺼둔다.

Bit 18 **HSEBYP**: HSE clock bypass

Set and cleared by software to bypass the oscillator with an external clock. The external clock must be enabled with the HSEON bit, to be used by the device.

The HSEBYP bit can be written only if the HSE oscillator is disabled.

0: HSE oscillator not bypassed

1: HSE oscillator bypassed with an external clock

Bit 17 **HSERDY**: HSE clock ready flag

Set by hardware to indicate that the HSE oscillator is stable. After the HSEON bit is cleared, HSERDY goes low after 6 HSE oscillator clock cycles.

0: HSE oscillator not ready

1: HSE oscillator ready

BOOT CODE

```
/* Reset HSEON, CSSON and PLLON bits */
RCC->CR &= (uint32_t)0xFEF6FFFF;

/* Reset PLLCFGR register */
RCC->PLLCFGR = 0x24003010;
```

6.3.2 RCC PLL configuration register (RCC_PLLCFGR)

Address offset: 0x04

Reset value: 0x2400 3010

Access: no wait state, word, half-word and byte access.

This register is used to configure the PLL clock outputs according to the formulas:

- $f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLLN / PLLM)$
- $f_{(PLL\ general\ clock\ output)} = f_{(VCO\ clock)} / PLLP$
- $f_{(USB\ OTG\ FS,\ SDIO,\ RNG\ clock\ output)} = f_{(VCO\ clock)} / PLLQ$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLLQ3	PLLQ2	PLLQ1	PLLQ0	Reserved	PLLSRC	Reserved				PLLP1	PLLP0
				rw	rw	rw	rw		rw					rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	PLLN									PLLM5	PLLM4	PLLM3	PLLM2	PLLM1	PLLM0
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BOOT CODE

```
/* Reset HSEBYP bit */
RCC->CR &= (uint32_t)0xFFFBFFFF;

/* 모든 Interrupt를 비활성화한다. */
RCC->CIR = 0x00000000;
```

RCC → PLLCFGR = 0x2400 3010;

pll(phase locked loop) - 클록생성하는 회로 (fpga)

→ pll 리셋 시키고 있다. 이유는?

pll은 높은 주파수를 만들때 사용한다. 이 때 높은 진동수를 가지면 오실레이터들이 일정 주파수를 잡기 어렵다. 그래서 RCC_PLLCFGR 비트도 꺼둠으로써 ploating되는 것을 방지한다. → 아무것도 하지말라고 초기화 시키는 것.

AHB, APB(BUS) – data 가 흘러 들어가는 속도 제어

RCC → CR &= (uint32_t)0xFFFBFFFF;

18번 비트가 0 이다.

Bit 18 **HSEBYP**: HSE clock bypass

Set and cleared by software to bypass the oscillator with an external clock. The external clock must be enabled with the HSEON bit, to be used by the device.
The HSEBYP bit can be written only if the HSE oscillator is disabled.
0: HSE oscillator not bypassed
1: HSE oscillator bypassed with an external clock

오실레이터 external clock 을 사용할것인지. NO

6.3.4 RCC clock interrupt register (RCC_CIR)

Address offset: 0x0C

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								CSSC	PLLSAI RDYC	PLLI2S RDYC	PLL RDYC	HSE RDYC	HSI RDYC	LSE RDYC	LSI RDYC
								w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserv ed	PLLSAI RDYIE	PLLI2S RDYIE	PLL RDYIE	HSE RDYIE	HSI RDYIE	LSE RDYIE	LSI RDYIE	CSSF	PLLSAI RDYF	PLLI2S RDYF	PLL RDYF	HSE RDYF	HSI RDYF	LSE RDYF	LSI RDYF
	rw	rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r

RCC → CIR = 0x0000 0000;

RCC interrupt clear register이다. 안에 비트를 살펴 보면됨.

```
#ifdef DATA_IN_ExtSRAM
  SystemInit_ExtMemCtl();
#endif /* DATA_IN_ExtSRAM */

  /* System Clock Source, PLL 곱셈기, 나눗셈기, AHB/APBx Prescalers와 Flash 설정을 구성한다. */
  SetSysClock();
```

Hsion

34분 30초

RCC_CFGR

39분 30초

Code: RCC → CR &= (uint32_0)0xFFFDFFFF;

54분
clock tree 오실레이터 발진기 osc크리스탈기반

미분적분 하는 나눗셈로직, 곱셈로직 등을 하는 pll 블록을 만드는것

RCC_CONFIG 레지스터 - 스위치를 조정함.

Peripheral , time square로 흘러 들어가는 pll주파수를 APB로 낮추는 역할을 한다. → 타기때문에..

1시 7분
Code: setsysclock 에대해

→ RCC_CR을 본다

1시간 15분
방사선

1시간 25분

왜 해야돼는겨!!

어셈블리어로 시스템콜 호출하기

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>

int main(void)
{
   int i;
   unsigned int test_arr[7] = {0};

   register unsigned int *r0 asm("r0") = 0;
   register unsigned int r1 asm("r1") = 0;
   register unsigned int r2 asm("r2") = 0;
   register unsigned int r3 asm("r3") = 0;
   register unsigned int r4 asm("r4") = 0;
   register unsigned int r5 asm("r5") = 0;
   register unsigned int r6 asm("r6") = 0;
   register int r7 asm("r7") = 0;

   r0 = test_arr;

   asm volatile("mov r1, #0x3\n"
            "mov r2,r1,lsl #2\n"
            "mov r4,#0x2\n"
            "add r3,r1,r2,lsl r4\n"
            "stmia r0!,{r1,r2,r3}\n"
            "str r4,[r0]\n"
            "mov r5,#128\n"
            "mov r6,r5,lsr #3\n"
            "stmia r0,{r4,r5,r6}\n"
            "sub r0,r0,#12\n"
            "ldmia r0,{r4,r5,r6}\n"
            "swp r6,r4,[r0]");
```

```
  for(i=0;i<7;i++)
    printf("test_arr[%d] = %d\n", i, test_arr[i]);

  printf("r4 = %u, r5 = %u, r6 = %u\n", r4,r5,r6);

  r7=2; //systemcall 전용 r7

  asm volatile("swi #0" : "=r" (r0) : "r" (r7) : "memory"); //swi: software inturrupt: "출력" , "입력", "특수 지시어"
                                        // memory : instruction 스케줄러하지마라. 연산이 끝나지않으면 끝날 수 없
  if(r0>0)
    printf("r0 = %p, Parent\n",r0);
  else if(r0 == 0)
    printf("r0=%p,Child\n",r0);
  else
  {
    perror("fork() ");
    exit(-1);
  }
  return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_2$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test_arr[0] = 3
test_arr[1] = 12
test_arr[2] = 51
test_arr[3] = 2
test_arr[4] = 128
test_arr[5] = 16
test_arr[6] = 0
r4 = 3, r5 = 12, r6 = 3
r0 = 0x191d, Parent
r0=(nil),Child
```

PinMode (led, OUTPUT)

GPIO 구조의 Direcction 레지스터를 출력으로 설정
pull up 을 쓸건지 Pull Down 을 쓸건지 지정함.
동작주파수를 지정함. 보편적으로 25나 50임

→ mcu setting

GPIO 구조 데이터 입출력 레지스터
값을 0을 넣거나 1을 넣으면 ON/OFF가 됨.

펌웨어 레벨