www.emlinux.co.kr

U-BOOT 이해

bsplinux

U-BOOT 설치

- 파일 찾기
  - http://sourceforge.net/project/u-boot
- 설치
  - 압축풀기
    - bzip2 –d u-boot-0.4.0.tar.bz2
  - tar파일 풀기
    - tar –xvf u-boot-0.4.0.tar
- 컴파일
  - make clobber => make clean과 동일
  - make smdk2410_config => arch. & board 에 대한 구성 설정
    - (TOPDIR)/inculde/arm/configs/smdk2410.h -> (TOPDIR)/inculde/config.h
  - make
- 출력 파일 확인(top directory)
  - uboot => ELF format
  - u-boot.bin => binary format
  - u-boot.srec =>S.recode format (motorola serial downloading image file)
  - Make u-boot.dis => dis-assembler 파일 생성

U-BOOT 디렉토리 구조

U-boot-0.4.0 →		
	board	.board에 의존적인 파일
	common	.architecture에 독립적인 파일
	cpu	.architecture에 의존적인 파일
	disk	.Code for disk drive partition handling
	doc	.uboot 관련 문서
	driver	.외부장치의 driver파일
	dtt	.외부 장치들 중 sensor(디지털 온도계)에 관련된 driver파일
	examples	.uboot을 위한 test실행 파일
	fs	.uboot에서 지원하는 file system관련 파일
	include	.header file
	Lib_arm	.arm architecture관련 라이브러리 파일
	net	.network 관련 파일
	post	.Power On Self Test
	rtc	.real time clock driver 파일
	tool	.Tools to build S-Record or U-Boot images, etc.

U-BOOT hardware point

- **Boot loder에서 handling하는 hardware list**
  - core에 의존적인 것들
    - Processor mode
    - Interrupt
    - Cache,mmu
  - SOC에 의존적인 것들
    - Interrupt,watchdog
    - Clock
    - Memory interface(dram controller)
    - TIMER
    - UART
    - RTC
  - 외부장치
    - Flash
    - NAND Flash
    - Ethernet controller
    - RTC
    - LCD Controller
    - Keyboard controller

Board 변경(smdk2410->atb-2410x)(1)

- **Board directory추가**
  - (TOPDIR)/board 디렉토리 안에 새로운 sub –borad 디렉토리를 만든다.
    - ✓ 가장 비슷한 보드 중에서 복사를 해서 만드는 것이 좋습니다.
    - ✓ Board 디렉토리 중 smdk2410 이 비슷하므로 복사 후 re-name한다.
    - ✓ Ex)(TOPDIR)/board/atb2410
- **Board header file 생성**
  - 이것도 마찬가지로 (TOPDIR)/include/configs/smdk2410.h 을 같은 디렉토리로 re-name(ex:atb2410.h)해서 복사 후 하드웨어적으로 다른 부분을 수정한다.
    - ✓ #define CONFIG_SMDK2410 1 => #define CONFIG_ATB2410 1
    - ✓ (TOPDIR)/cpu/arm920t/interrupts file중 get_tbclk()에서 CONFIG_ATB2410 추가한다.
- **(TOPDIR)Makefile 수정**
  - smdk2410_config: unconfig

    @./mkconfig $(@:_config=) arm arm920t smdk2410

    위 항목 및 에 아래 와 같이 동일하게 작성한다.

    atb2410_config: unconfig

    @./mkconfig $(@:_config=) arm arm920t atb2410

Board 변경(smdk2410->atb-2410x)(2)

❖ **(TOPDIR)/include/configs/Atb2410.h 수정(sdram 32Mbyte,flash sst(39VF160) 2Mbyte)**

```
#define CONFIG_INIT_CRITICAL            /* undef for developing(dram loading시) */

#define CONFIG_ARM920T          1       /* This is an ARM920T Core          */
#define CONFIG_S3C2410          1       /* in a SAMSUNG S3C2410 SoC     */
/*#define CONFIG_SMDK2410       1*/     /* on a SAMSUNG SMDK2410 Board  */
#define CONFIG_ATB2410          1

/* input clock of PLL */
#define CONFIG_SYS_CLK_FREQ  12000000   /* the SMDK2410 has 12MHz input clock */
                                        @ atb2410도 12Mhz clock사용

#define USE_920T_MMU            1
#undef CONFIG_USE_IRQ                   /* we don't need IRQ/FIQ stuff */
                                        @ IRQ사용 여부
/*
 * Size of malloc() pool
 */                                     @ 환경설정 저장용으로 128kbyte설정
#define CFG_MALLOC_LEN          (CFG_ENV_SIZE + 128*1024)
```

Board 변경(smdk2410->atb-2410x)(3)

❖ (TOPDIR)/include/configs/ Atb2410.h 수정

```
 /* Hardware drivers */
#define CONFIG_DRIVER_CS8900  1                    @ 이더넷 드라이브로 cs8900을 사용
/*#define CS8900_BASE          0x19000300*/        @ cs8900 base address설정
#define CS8900_BASE 0x18000300                     @ test필요
#define CS8900_BUS16                  1            @ 16bit data-bus사용

/*
 * select serial console configuration
 */
#define CONFIG_SERIAL1        1                    @ console용으로 s3c2410의 UART0을 사용

/***********************************************************
 * RTC
 ***********************************************************/
#define     CONFIG_RTC_S3C24X01                    @ s3c2410에 내장된 RTC을 사용

/* allow to overwrite serial and ethaddr */
#define CONFIG_ENV_OVERWRITE                       @ not used
#define CONFIG_BAUDRATE                 115200     @ console용 UART의 baud rate을 설정
```

Board 변경(smdk2410->atb-2410x)(4)

❖ **(TOPDIR)/include/configs/ Atb2410.h 수정**

```
/* Command definition*/
#define CONFIG_COMMANDS \
                        (CONFIG_CMD_DFL     | \      @ default configs
                        CFG_CMD_CACHE       | \      @ icache,dcache command지원
                        CFG_CMD_REGINFO  | \        @ register information command지
                        CFG_CMD_DATE        | \      @ RTC 지원,date,time command 지원
                        CFG_CMD_ELF)

/* this must be included AFTER the definition of CONFIG_COMMANDS (if any) */
#include <cmd_confdefs.h>

#define CONFIG_BOOTDELAY    3                  @ boot delay 3초
/*#define CONFIG_BOOTARGS    "root=ramfs devfs=mount console=ttySA0,9600" */
/*#define CONFIG_ETHADDR                08:00:3e:26:0a:5b */
#define CONFIG_NETMASK                 255.255.255.0
#define CONFIG_IPADDR                  10.0.0.110
#define CONFIG_SERVERIP                10.0.0.1
/*#define CONFIG_BOOTFILE              "elinos-lart" */
/*#define CONFIG_BOOTCOMMAND           "tftp; bootm" */
```

Board 변경(smdk2410->atb-2410x)(5)

❖ (TOPDIR)/include/configs/ Atb2410.h 수정

```
#define     CFG_LONGHELP                                    @ command help지원
/*#define   CFG_PROMPT              "SMDK2410 # " */       @ prompt text
#define     CFG_PROMPT              "ATB2410 # "
#define     CFG_CBSIZE              256                     @ console용 I/O buffer 수 설정
#define     CFG_PBSIZE (CFG_CBSIZE+sizeof(CFG_PROMPT)+16)   @ print buffer size
#define     CFG_MAXARGS             16                      @ command 최대 인자수
#define CFG_BARGSIZE                CFG_CBSIZE              @ boot Argument buffer size
#define CFG_MEMTEST_START           0x30000000              @ sdram memory start test addr.
/*#define CFG_MEMTEST_END           0x33F00000*/            @ sdram memory end addr.(63M)
#define CFG_MEMTEST_END             0x31F00000              @ atb2410 ->31Mbyte로 설정
                                                                    @ 1Mbyte는 boot loader

#undef  CFG_CLKS_IN_HZ                                      @ not used
/*#define   CFG_LOAD_ADDR           0x33000000*/            /* default load address */
#define CFG_LOAD_ADDR               0x31000000
/* the PWM TImer 4 uses a counter of 15625 for 10 ms, so we need */
/* it to wrap 100 times (total 1562500) to get 1 sec. */
#define     CFG_HZ                  1562500                 @ 1sec설정
/* valid baudrates */
#define CFG_BAUDRATE_TABLE    { 9600, 19200, 38400, 57600, 115200 } @ 지원 가능한 baud rate설정
```

Board 변경(smdk2410->atb-2410x)(6)

❖ **(TOPDIR)/include/configs/ Atb2410.h 수정**

```
/ * Stack sizes
 *
 * The stack sizes are set up in start.S using the settings below
 */
#define CONFIG_STACKSIZE        (128*1024)          /* regular stack */
#ifdef CONFIG_USE_IRQ
#define CONFIG_STACKSIZE_IRQ  (4*1024)              /* IRQ stack */
#define CONFIG_STACKSIZE_FIQ  (4*1024)              /* FIQ stack */
#endif
/*-----------------------------------------------------------------------
 * Physical Memory Map
 */
#define CONFIG_NR_DRAM_BANKS          1             /* we have 1 bank of DRAM */
#define PHYS_SDRAM_1                  0x30000000 /* SDRAM Bank #1 */
/*#define PHYS_SDRAM_1_SIZE           0x04000000 *//* 64 MB */
#define PHYS_SDRAM_1_SIZE             0x02000000 /* 32 MB */

#define PHYS_FLASH_1                  0x00000000 /* Flash Bank #1 */
#define CFG_FLASH_BASE                PHYS_FLASH_1
```

Board 변경(smdk2410->atb-2410x)(7)

❖ (TOPDIR)/include/configs/ Atb2410.h 수정

```
/ * FLASH and environment organization */
/*#define CONFIG_AMD_LV400     1          /* uncomment this if you have a LV400 flash */
#if 0
#define CONFIG_AMD_LV800     1          /* uncomment this if you have a LV800 flash */
#endif*/
#define CFG_MAX_FLASH_BANKS  1          /* max number of memory banks */
/*#ifdef CONFIG_AMD_LV800*/
/*#define PHYS_FLASH_SIZE              0x00100000*/ /* 1MB */
#define PHYS_FLASH_SIZE              0x00200000 /* 2MB */
/*#define CFG_MAX_FLASH_SECT  (19)     *//* max number of sectors on one chip */
#define CFG_MAX_FLASH_SECT   (32)     /*39VF160 64K x 32 = 2Mbyte*/
/*#define CFG_ENV_ADDR                 (CFG_FLASH_BASE + 0x0F0000) *//* addr of environment */
#define CFG_ENV_ADDR                 (CFG_FLASH_BASE + 0x1F0000)    /*64Kbyte*/
/*#endif
#ifdef CONFIG_AMD_LV400
#define PHYS_FLASH_SIZE              0x00080000 /* 512KB */
#define CFG_MAX_FLASH_SECT   (11)     /* max number of sectors on one chip */
#define CFG_ENV_ADDR                 (CFG_FLASH_BASE + 0x070000) /* addr of environment */
#endif*/
```

Board 변경(smdk2410->atb-2410x)(8)

❖ **(TOPDIR)/include/configs/ Atb2410.h 수정**

```
/* timeout values are in ticks */
#define CFG_FLASH_ERASE_TOUT (5*CFG_HZ)          /*5sec, Timeout for Flash Erase */
#define CFG_FLASH_WRITE_TOUT (5*CFG_HZ)          /*5sec,Timeout for Flash Write */

#define     CFG_ENV_IS_IN_FLASH          1       @ flash사용(예외:eeprom,nvram등)
#define     CFG_ENV_SIZE                 0x10000 /* Total Size of Environment Sector */
                                                 @ 64Kbyte

#endif      /* __CONFIG_H */
```

Board 변경(smdk2410->atb-2410x)(9)

❖ (TOPDIR)/board/atb2410/ config.mk 수정

```
#
# SMDK2410 has 1 bank of 64 MB DRAM
# atb2410 has 1 bank of 32 MB DRAM
# 3000'0000 to 3400'0000
# 3000'0000 to 3200'0000
# Linux-Kernel is expected to be at 3000'8000, entry 3000'8000
# optionally with a ramdisk at 3080'0000
#
# we load ourself to 33F8'0000
# we load ourself to 31F8'0000
# download area is 3300'0000
# download area is 3100'0000


/*TEXT_BASE = 0x33F80000*/
TEXT_BASE = 0x31F80000          /* u-boot sdram start address*/
```

U-boot.lds(1) – (TOPDIR)/board/smdk2410/u-boot.lds

- Start.S 파일을 분석하기전에 (TOPDIR)/board/smdk2410/u-boot.lds 파일을 먼저 보면, 이것은 LD(loader&Linker)의 input으로 주어져서,object 파일을 생성하는데 규칙을 제공한다.
- 아래와 같은 부분이 있을 것이다.

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
/*OUTPUT_FORMAT("elf32-arm", "elf32-arm", "elf32-arm")*/
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
      . = 0x00000000;
      . = ALIGN(4);
       .text      :
       {
        cpu/arm920t/start.o (.text)
        *(.text)
       }
      . = ALIGN(4);
      .rodata : { *(.rodata) }
      . = ALIGN(4);
      .data : { *(.data) }
      . = ALIGN(4);
      .got : { *(.got) }
       armboot_end_data = .;
      . = ALIGN(4);
      .bss : { *(.bss) }
       armboot_end = .;
}
```

U-boot.lds(2) – (TOPDIR)/cpu/arm920t/start.S

- OUTPUT_FORMAT은 ELF32의 little endian으로 코드를 생성하겠다는 말이다.
- OUTPUT_ARCH은 binary을 실행할 수 있는 CPU architecture로 ARM을 쓰겠다는 뜻이다.
- ENTRY point은 Program의 사작을 가리키며,시작되는 함수의 이름은 "_start"이다.
- SECTIONS의 정의를 보면,text,rodata,data,got,bss라는 section들이 정의 되어 있다.
  - **.text : 실행할 프로그램 코드 영역**
  - **.rodata: read-only data 영역(const 등으로 지정된 데이터)**
  - **.data: initialized data 영역**
  - **.got: global offset table 영역**
  - **.bss: uninitialized data 영역**
- 특수한 링커 변수 *dot* `.' 는 항상 현재 출력 address point을 담고 있다.
  - **address point 는 출력 섹션의 크기만큼 증가한다.**
- `*'는 어떤 파일명에도 대응한다. `*(.text)'는 모든 입력파일의 모든 입력 섹션 `.text'을 의미한다.
- 프로그램 코드는 0x00000000에서 시작해서 4byte단위로 정렬된 text section에 놓여질 것이다.
- U-boot의 시작은 Entry point에 선언된 _start부터 시작된다.
- _start는 (TOPDIR)/cpu/arm920t/start.S에 정의되어 있다.
- TEXT_BASE에의해 Linker수행 시 symbol들은 상대적 주소를 갖는다.
- Power가 on 한 후 0x00번지(즉,flash)에서 시작하여 memory초기화를 거쳐 flash의 내용을 dram에 relocate하면 비로써 dram 에서 동작 하게 된다.
- Symbol들은 모두 TEXT_BASE의 상대적 주소 값을 가지고 있으므로, dram에 relocate전에는 offset branch 명령만 사용 해야 한다.(B,BL,ADR)

(TOPDIR)/CPU/arm920t directory

Stats.S

Cpu.c

Speed.c

Interrupts.c

Serial.c

Start.S 분석(1) – (TOPDIR)/cpu/arm920t/start.S

- **ARM process의 power on(reset)후 actions**
  - r14_svc(lr) <= 예측할 수 없는 value
  - SPSR_svc <= CPSR(기존 mode)
  - CPSR[4:0] <= 0b10011(Supervisor mode)
  - CPSR[5] <= 0 (T bit = ARM state)
  - CPSR[6] <= 1 (F bit)
  - CPSR[7] <= 1 (I bit)
  - PC = 0x0 (Vector Table)
- **Supervisor mode,arm state,pc 값이 0x0이므로 Linker script file의 entry point에서 선언된 '_start'부터 프로그램 코드가 실행된다.**

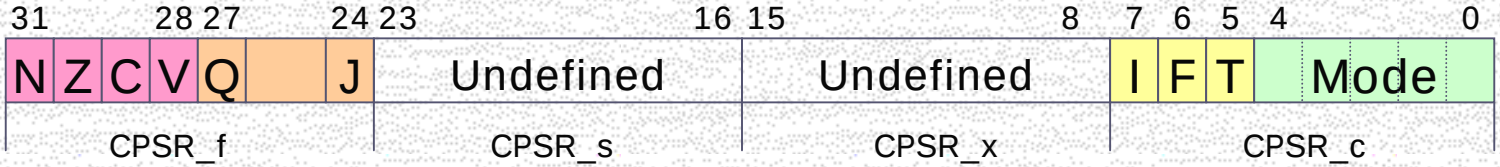
```
.globl _start                                   @ flash start
_start:     b       reset                       @ offset jump이므로 dram으로 이동하지 않는다.
            ldr         pc, _undefined_instruction
            ldr         pc, _software_interrupt
            ldr         pc, _prefetch_abort
            ldr         pc, _data_abort
            ldr         pc, _not_used
            ldr         pc, _irq
            ldr         pc, _fiq
```

Start.S 분석(2) – (TOPDIR)/cpu/arm920t/start.S

- **.global _start**
  - Linker에게 '_start' symbol 처리할 수 있도록 export한다.
- **_start의 첫번째 instruction은 'b reset'이므로 reset 함수로 branch하여 reset으로 제어가 옮겨진다.**
- **나머지는 ARM에서 발생할 수 있는 exception으로 상황에 대한 처리 routine들로 exception이 발생하면 해당하는 핸들러로 branch할 것이다.**
  - **_undefined_instruction**
  - **_software_interrupt**
  - **_prefetch_abort**
  - **_data_abort**
  - **_not_used**
  - **_irq**
  - **_fiq**

Start.S 분석(3) – (TOPDIR)/cpu/arm920t/start.S

- **CPSR 레지스터**



- **MODE**

M[4:0]	Mode	Accessible registers
0b10000	User	PC, R14 to R0, CPSR
0b10001	FIQ	PC, R14_fiq to R8_fiq, R7 to R0, CPSR, SPSR_fiq
0b10010	IRQ	PC, R14_irq, R13_irq, R12 to R0, CPSR, SPSR_irq
0b10011	Supervisor	PC, R14_svc, R13_svc, R12 to R0, CPSR, SPSR_svc
0b10111	Abort	PC, R14_abt, R13_abt, R12 to R0, CPSR, SPSR_abt
0b11011	Undefined	PC, R14_und, R13_und, R12 to R0, CPSR, SPSR_und
0b11111	System	PC, R14 to R0, CPSR (ARM architecture v4 and above)

Start.S 분석(3) – (TOPDIR)/cpu/arm920t/start.S

❖ **Reset () => mode setting 및 interrupt disable**

```
reset: /*set the cpu to SVC32 mode*/
        mrs     r0,cpsr                 @ cpsr값을 r0레지스터로 옮긴다.
        bic     r0,r0,#0x1f             @ Mode bit을 clear한다.
        orr     r0,r0,#0xd3             @ interrupt 을 disable,supervisor mode
        msr     cpsr,r0                 @ r0값으로 cpsr값을 바꾼다.
```

```
#if defined(CONFIG_S3C2400)
#define pWTCON          0x15300000      @ turn off the watchdog
#define INTMSK          0x14400008      @ Interupt-Controller base addresses
#define CLKDIVN         0x14800014      @ clock divisor register
#elif defined(CONFIG_S3C2410)
#define pWTCON          0x53000000      @ Watchdog Timer Mode
#define INTMSK          0x4A000008      @ Interrupt Mask Control
#define INTSUBMSK 0x4A00001C            @ Interrupt sub mask
#define CLKDIVN         0x4C000014      @ clock divisor Control
#endif
```

Start.S 분석(4) – (TOPDIR)/cpu/arm920t/start.S

❖ **Reset ()=> Watchdog disable & all interrupt source masking**

```
        ldr     r0, =pWTCON         @ 0x53000000
        mov     r1, #0x0
        str     r1, [r0]            @ watchdog timer을 disable한다.

        /*
         * mask all IRQs by setting all bits in the INTMR - default
         */
        mov     r1, #0xffffffff
        ldr     r0, =INTMSK         @ 0x4A000008
        str     r1, [r0]            @ interrupt 을 모두 masking 한다.(disable)
#if defined(CONFIG_S3C2410)
        ldr     r1, =0x3ff
        ldr     r0, =INTSUBMSK      @ 0x4A00001C
        str     r1, [r0]            @ cpu가 s3c2410일 경우 sub interrupt을 masking 한다.
#endif
```

Start.S 분석(5) – (TOPDIR)/cpu/arm920t/start.S

- **Reset () => Clock setting**
  - CLKDIVN (CLOCK DIVIDER CONTROL REGISTER) 0x4C000014
    - [2] reserved
    - [1] HDIVN : 0 [HCLK = FCLK], 1 [HCLK = FCLK/2]
    - [0] PDIVN : 0 [PCLK = HCLK], 1 [PCLK = HCLK/2]
  - FCLK,HCLK,and PCLK
    - FCLK : ARM920T에서 사용한다.
    - HCLK : AHB bus을 위하여 사용한다.(memory ,interrupt ,LCD ,DMA, USB Host controller)
    - PCLK : APB bus을 위하여 사용한다.(WDT,IIS,I2C,PWM timer,MMC,ADC,UART,GPIO,RTC,SPI)

HDIVN	PDIVN	FCLK	HCLK	PCLK	Divide Ratio
0	0	FCLK	FCLK	FCLK	1:1:1(default)
0	1	FCLK	FCLK	FCLK/2	1:1:2
1	0	FCLK	FCLK/2	FCLK/2	1:2:2
1	1	FCLK	FCLK/2	FCLK/4	1:2:4 (recommended)

  - POWER MODE = NORMAL MODE일 경우 FCLK = MPLL clock(Mpll)과 동일하다.
    - PLL Control Register(MPLLCON and UPLLCON)
    - Mpll = (m * Fin)/(p * 2^{s})
    - m = (MDIV + 8), p = (PDIV + 2), s = SDIV, Fin = 입력 주파수

Start.S 분석(5) – (TOPDIR)/cpu/arm920t/start.S

❖ Reset () => Clock setting

- MPLLCON 0x4c000004
  - ✓ [19:12] => MDIV(Main divider) : initial value (0x5c)
  - ✓ [9:4] => PDIV(pre-divider control) : initial value (0x08)
  - ✓ [1:0] => SDIV(Post divider control) : initial value (0x00)
- Mpll = (m * Fin)/(p * 2^{s})
  - ✓ m = (0x5c + 8), p = (0x08 + 2), s = 0x00, Fin = 12Mhz
  - ✓ Mpll = (m(100) * Fin(12Mhz)) / (p(10) * 2^{0}) => 1200Mhz / 10 =>120Mhz
  - ✓ FCLK = 120Mhz

```
/* FCLK:HCLK:PCLK = 1:2:4 */
/* default FCLK is 120 MHz ! */
ldr     r0, =CLKDIVN        @ 0x4C000014
mov     r1, #3              @ HDIVN,PDIVN 이 모두 1로 setting
str     r1, [r0]            @ FCLK:HCLK:PCLK = 1:2:4 (120M:60M:30MHz)
```

```
#ifdef CONFIG_INIT_CRITICAL     @ (TOPDIR)/inculde/configs/smdk2410.h
        bl      cpu_init_crit   @ sub rountine cpu_init_crit 을 호출한다.
#endif
```

Start.S 분석(6) – (TOPDIR)/cpu/arm920t/start.S

- **Reset () => Memory setting(1)**
  - **<MCR|MRC>{cond} p#,<expression1>,Rd,cRn,cRm{,<expression2>}**
    - MRC : coprocessor register을 CPU register로 이동 (L=1)
    - MCR : CPU register 을 coprocessor register로 이동 (L=0)
    - {cond} : Two character condition mnemonic
    - p# : coprocessor 번호
    - <opcode_1> : coprocessor-specific opcode. <opcode_1> 은 항상 0 이다.
    - Rd : CPU register number
    - cRn and cRm : coprocessor register numbers
    - <opcode_2> : coprocessor-specific opcode. 이 필드는 생략 할 수 있다.
  - **CP15 REGISTER MAP은 s3c2410 manual을 참조.**
    - REGISTER1(c1) : CONTROL REGISTER => page 2-10
    - REFISTER7(c7) : CACHE OPERATIONS => page 2-15
    - REFISTER8(c8) : TLB OPRERATIONS => page 2-18

Start.S 분석(7) – (TOPDIR)/cpu/arm920t/start.S

❖ Cpu_init_crit () => Memory initialize (1)

```
cpu_init_crit:-
        mov     r0, #0                  @ flush: 플러시,기억 장치 부분의 내용을 비우는 것.
        mcr     p15, 0, r0, c7, c7, 0   @ flush v4 I/D caches
        mcr     p15, 0, r0, c8, c7, 0   @ flush v4 TLB
                                        @ s3c2410 datasheet p2-4참조
        mrc     p15, 0, r0, c1, c0, 0   @ disable MMU stuff and caches
        bic     r0, r0, #0x00002300     @  clear bits 13, 9:8 (--V- --RS)
        bic     r0, r0, #0x00000087     @ clear bits 7, 2:0 (B--- -CAM)
        orr     r0, r0, #0x00000002     @ set bit 2 (A)
        orr     r0, r0, #0x00001000     @ set bit 12 (I) I-Cache enable
        mcr     p15, 0, r0, c1, c0, 0

        mov     ip, lr                  @ 또다시 BL을 사용하므로 lr을 ip레지스터로 이동
        bl      memsetup                @ SDRAM  initialize sub rountine 을 호출한다.
        mov     lr, ip                  @ sub rountine을 요청했던 곳으로 pc 값을 이동-
        mov     pc, lr                  @ 하기 위해서 ip값을 lr 레지스터로 이동한다.
                                        @ 최종적으로 lr값을 pc 레지스터로 이동,즉 return 한다.
```

memsetup.S 분석(1) – (TOPDIR)/board/smdk2410/memsetup.S

❖ **memsetup() => Memory Bank initialize(Bank0 – Bank7)**

```
#define BWSCON                  0x48000000
/* BWSCON */
#define DW8                     (0x0)
#define DW16                    (0x1)
#define DW32                    (0x2)
#define WAIT                    (0x1<<2)
#define UBLB                    (0x1<<3)

#define B1_BWSCON               (DW32) /* bank 1 32bit data bus width */
#define B2_BWSCON               (DW16)
#define B3_BWSCON               (DW16 + WAIT + UBLB)
#define B4_BWSCON               (DW16)
#define B5_BWSCON               (DW16)
#define B6_BWSCON               (DW32)
#define B7_BWSCON               (DW32)

/* BANK0CON */
#define B0_Tacs                 0x0         /*  0clk */
#define B0_Tcos                 0x0         /*  0clk */
#define B0_Tacc                 0x7         /* 14clk */
#define B0_Tcoh                 0x0         /*  0clk */
#define B0_Tah                  0x0         /*  0clk */
#define B0_Tacp                 0x0
#define B0_PMC                  0x0         /* normal */
```

memsetup.S 분석(2) – (TOPDIR)/board/smdk2410/memsetup.S

❖ **memsetup() => Memory Bank initialize(Bank0 – Bank7)**

```
/* BANK1CON */
#define B1_Tacs          0x0      /*  0clk */
#define B1_Tcos          0x0      /*  0clk */
#define B1_Tacc          0x7      /* 14clk */
#define B1_Tcoh          0x0      /*  0clk */
#define B1_Tah           0x0      /*  0clk */
#define B1_Tacp          0x0
#define B1_PMC           0x0

/* BANK2CON */
            ...
/* BANK3CON */
#define B3_Tacs          0x0      /*  0clk */
#define B3_Tcos          0x3      /*  4clk */
#define B3_Tacc          0x7      /* 14clk */
#define B3_Tcoh          0x1      /*  1clk */
#define B3_Tah           0x0      /*  0clk */
#define B3_Tacp          0x3      /*  6clk */
#define B3_PMC           0x0      /* normal */
/* BANK4CON */
            ...
/* BANK5CON */
            ...
```

memsetup.S 분석(1) – (TOPDIR)/board/smdk2410/memsetup.S

❖ memsetup() => Memory Bank initialize(Bank0 – Bank7)

```
@ K4S561632C-TC75 133Mhz
@ clock cycle 7.5ns(1/133M), Trcd(RAS to CAS delay) 20ns, Trp(row precharge time) 20ns
@ Trc (Row Cycle Time) 65ns, Refresh period 64ms, row addr. Ra0 – Ra12 , column addr. Ca0 – Ca8
#define B6_MT          0x3     @ SDRAM select
#define B6_Trcd        0x1     @ 20ns / 7.5ns => 2.66 cycle=>3cycle
#define B6_SCAN        0x1     @ column addr. Ca0 – Ca8  => 9bit

#define B7_MT          0x3     @ SDRAM select
#define B7_Trcd        0x1     @ 20ns / 7.5ns => 2.66 cycle=>3cycle
#define B7_SCAN        0x1     @ column addr. Ca0 – Ca8  => 9bit
/* REFRESH parameter */
#define REFEN          0x1     @ Refresh enable
#define TREFMD         0x0     @ CBR(CAS before RAS)/Auto refresh
#define Trp            0x0     @ 20ns / 7.5ns => 2.66 cycle=>3cycle ?? 2 clk
#define Trc            0x3     @ 65ns / 7.5ns => 8.66 cycle => 9cycle ?? 7clk
#define Tchr           0x2     @ 3clk , reserved
@?? 아마도 오차값 이 아닐지
/* period=15.6us(1/64ms), HCLK=60Mhz, (2048+1-15.6*60) */
#define REFCNT         1113
```

memsetup.S 분석(5) – (TOPDIR)/board/smdk2410/memsetup.S

❖ memsetup() => Memory Bank initialize(Bank0 – Bank7)

```
_TEXT_BASE:
        .word   TEXT_BASE
.globl memsetup                 @ Linker에게 'memsetup' symbol 처리할 수 있도록 export한다.
memsetup:                       @ memory control configuration
                                @ SMRDATA은 memory가 아직 초기화가 되지 않았으므로 flash에서 읽기위하여
                                @ r0에 상대주소를 저장한다.즉 flash의 _start번지부터의 offset값이 된다.
                                @ ex) SMRDATA(33f80420) - TEXT_BASE(33f80000) = 0x00000420
        ldr   r0, =SMRDATA      @ literal pools data의 (=)address을 r0레지스터로 이동
        Ldr   r1, _TEXT_BASE    @ _TEXT_BASE address의 데이터 값을 r1레지스터로 이동
        Sub   r0, r0, r1        @ r0 상대 주소 값을 저장한다.
        Ldr   r1, =BWSCON       @ Bus Width Status Control Register의 base address을 r1에 넣는다.
        add   r2, r0, #13*4     @ 13개의 word Register을 r0 register와 더한 후 r2 레지스터로 이동
0:                              @ 즉,13번 loop을 돌리기 위해 compare할 레지스터 setting(end address)
        ldr   r3, [r0], #4      @ r0가 가리키는 address의 데이터를 r3레지스터로 이동 후 r0값을 4증가시키다.
        str   r3, [r1], #4      @ r1이 가리키는 address에 r3의 값을 저장 후 r1값을 4증가시킨다.
        cmp    r2, r0           @ r2,r0을 비교하여 그 결과 값을 CPSR에 setting
        bne    0b               @ 같지 않으면 계속 loop수행,13개의 memory관련 레지스터를 초기화 하다.

        mov       pc, lr        @ memsetup 함수를 호출한 cpu_init_crit 함수로 복귀
```

memsetup.S 분석(4) – (TOPDIR)/board/smdk2410/memsetup.S

❖ memsetup() => Memory Bank initialize(Bank0 – Bank7)

@ **0x00000420**

```
SMRDATA:
    .word (0+(B1_BWSCON<<4)+(B2_BWSCON<<8)+(B3_BWSCON<<12)+(B4_BWSCON<<16)+(B5_BWSCON<<20) \
                        +(B6_BWSCON<<24)+(B7_BWSCON<<28))
    .word ((B0_Tacs<<13)+(B0_Tcos<<11)+(B0_Tacc<<8)+(B0_Tcoh<<6)+(B0_Tah<<4)+(B0_Tacp<<2)+(B0_PMC))
    .word ((B1_Tacs<<13)+(B1_Tcos<<11)+(B1_Tacc<<8)+(B1_Tcoh<<6)+(B1_Tah<<4)+(B1_Tacp<<2)+(B1_PMC))
    .word ((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(B2_PMC))
    .word ((B3_Tacs<<13)+(B3_Tcos<<11)+(B3_Tacc<<8)+(B3_Tcoh<<6)+(B3_Tah<<4)+(B3_Tacp<<2)+(B3_PMC))
    .word ((B4_Tacs<<13)+(B4_Tcos<<11)+(B4_Tacc<<8)+(B4_Tcoh<<6)+(B4_Tah<<4)+(B4_Tacp<<2)+(B4_PMC))
    .word ((B5_Tacs<<13)+(B5_Tcos<<11)+(B5_Tacc<<8)+(B5_Tcoh<<6)+(B5_Tah<<4)+(B5_Tacp<<2)+(B5_PMC))
    .word ((B6_MT<<15)+(B6_Trcd<<2)+(B6_SCAN))
    .word ((B7_MT<<15)+(B7_Trcd<<2)+(B7_SCAN))
    .word ((REFEN<<23)+(TREFMD<<22)+(Trp<<20)+(Trc<<18)+(Tchr<<16)+REFCNT)
    .word 0x32          @ 128MB/128MB, SCLK은 access동안만 active, SDRAM power down mode, burst disable
    .word 0x30          @ burst type(Sequential), CAS latency 3
    .word 0x30          @ burst type(Sequential), CAS latency 3
```

Start.S 분석(8) – (TOPDIR)/cpu/arm920t/start.S

❖ **relocate() => armboot 을 ram으로 재배치**

```
relocate:                                   @ relocate armboot to RAM
                                            @ _start을 ldr로 읽으면 0x33f80000이 된다.memory
                                            @ memory에는 아직 데이터가 존재하지 않으며 flash에서 loading
        adr     r0, _start                  @  => SUB r0, PC, #offset to _Start ,_start의상대 address
        ldr     r2, _armboot_start          @ _start의 address (0x33f80000),system.map 참조
        ldr     r3, _armboot_end            @ armboot_end(0x33f995c8), linker script 참조

        sub     r2, r3, r2                  @ r2 <- armboot memory size
        ldr     r1, _TEXT_BASE              @ r1 <- destination address
        add     r2, r0, r2                  @ r2 <- source end address
        /*
         * r0 = source address
         * r1 = target address
         * r2 = source end address
         */
copy_loop:
        ldmia   r0!, {r3-r10}               @ r0가 가리키는 address의 데이터를 r3-r10까지 저장,write back
        stmia   r1!, {r3-r10}               @ r1이 가리키는 address에 r3-r10의 데이터 값을 저장,write back
        cmp     r0, r2                      @ destination address가 source end address 와 같은지 check
        ble     copy_loop                   @ armboot memory size만큼 loop을 수행한다.
```

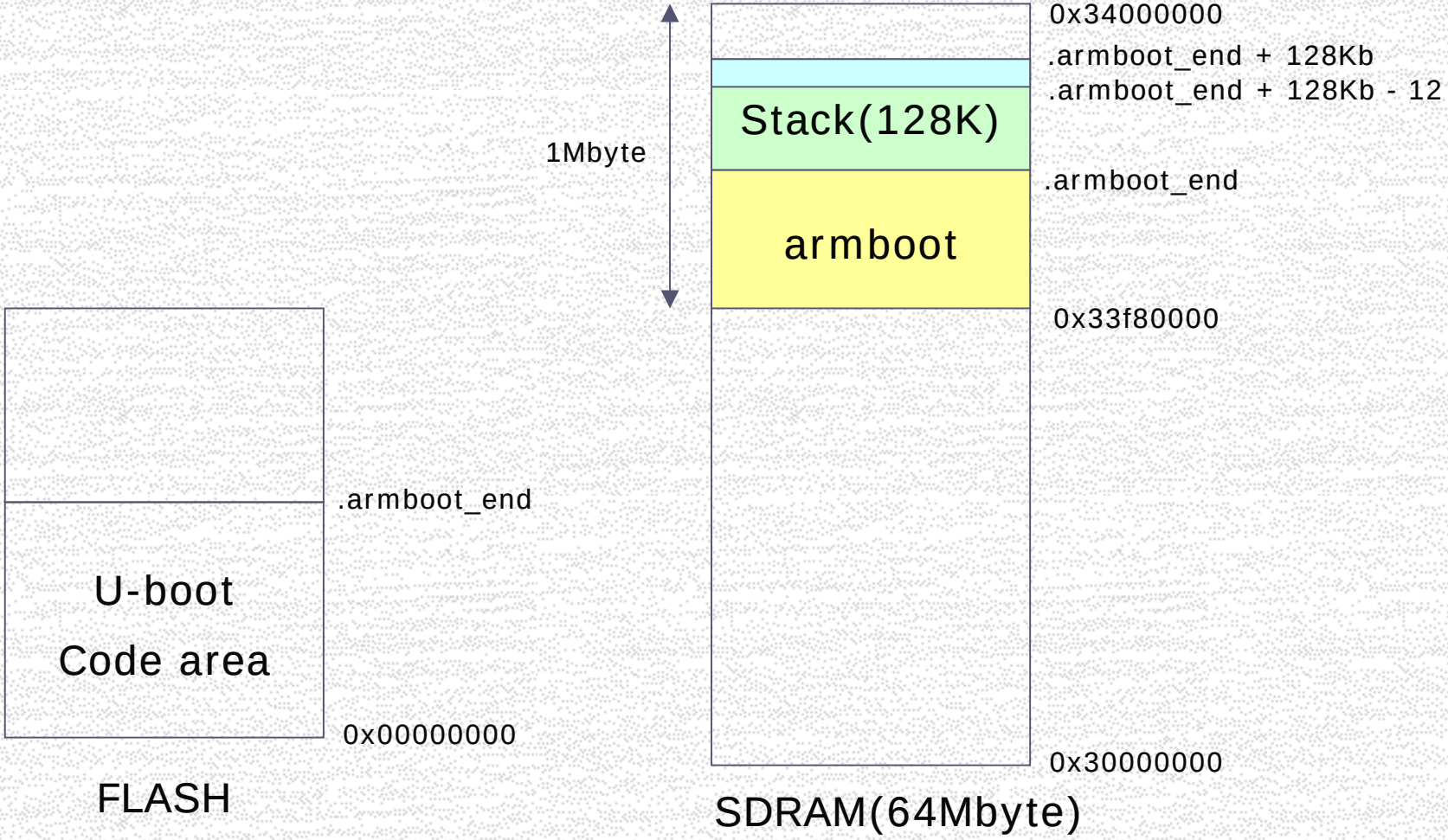
Start.S 분석(9) – (TOPDIR)/cpu/arm920t/start.S

- **relocate() => stack address지정 및 “c” routine으로 jump**
- **Stack point의 최상위에서 3word을 비워두는 이유는 abort exception이 발생했을 경우 exception이 발생하기전의 PC값과 CPSR값을 저장하여 debug정보에 사용하기위해서 이다.**
- **Stack point을 설정해 주었으므로 C routine을 실행 하기 위해 board.c 파일에 포함되어 있는 start_armboot 함수로 branch한다.**

```
/*CONFIG_STACKSIZE        (128*1024), (TOPDIR)/include/configs/smdk2410.h*/
/* set up the stack */
        ldr     r0, _armboot_end        @armboot_end, linker script 참조
        add     r0, r0, #CONFIG_STACKSIZE
        sub     sp, r0, #12             @ leave 3 words for abort-stack
        ldr     pc, _start_armboot      @ dram의 start_armboot로 brach
                                        @ 지금부터는 dram 에서 동작된다.
_start_armboot: .word start_armboot     @(TOPDIR)/lib_arm/board.c
```

Start.S 분석(9) – (TOPDIR)/cpu/arm920t/start.S

❖ **relocate() => 재배치후 메모리 map(smdk2410 보드 기준)**



Start.S 분석(10) – (TOPDIR)/cpu/arm920t/start.S

- **Start_armboot**에 들어가기 전에 **exception**처리 **routine**부터 살펴보자.
- **Arm**에서는 여러 가지의 **exception**을 제공하나 여기서는 두 가지만 설명하기로 한다.
  - Undefined instruction exception
  - IRQ exception
  - 나머지도 위의 두 가지 처리방식에서 크게 벗어나지않는다.

Start.S 분석(11) – (TOPDIR)/cpu/arm920t/start.S

- **Undefined instruction exception**
  - coprocessor 가 응답하지 않는 경우에 발생한다.
  - 정의 되어 있지 않는 **instruction**을 실행 했을 경우 발생한다.
  - **CPU actions**
    - r14_und = undefined instruction이 발생한 주소 + 4
    - SPSR_und = CPSR
    - CPSR[5:0] = 0b011011
    - CPSR[6] = unchanged
    - CPSR[7] = 1
    - PC = 0x4

```
/* exception handlers */
@ 2에 5승 즉, 32byte로 정렬해라 는 뜻이다 .
@ 두 번째 표현식이 없으면 보태지는 바이트는 0으로 채워진다 .
        .align  5
undefined_instruction:
        get_bad_stack                   @ macro
        bad_save_user_regs              @ macro
        bl      do_undefined_instruction    @ C routine
```

Start.S 분석(12) – (TOPDIR)/cpu/arm920t/start.S

❖ Macro get_bad_stack

```
.macro get_bad_stack                    @ und mode
ldr     r13, _armboot_end               @ r13_und 레지스터에  armboot_end address을 load한다.
add     r13, r13, #CONFIG_STACKSIZE     @ r13_und <= r13_und + 128Kbyte
sub     r13, r13, #8                    @ r13_und <= r13_und  - 8

str     lr, [r13]                       @ r13_und 레지스터가 가리키는 address에 lr_und값을 저장한다.
mrs     lr, spsr                        @ r13_und 레지스터가 가리키는 address에 4을 더한
str     lr, [r13, #4]                   @ address에 spsr_und값을 저장한다.
@전 mode가 supervisor mode인 것 처럼 만든다.
mov     r13, #MODE_SVC                  @ r13_und 레지스터에 SVC-Mode value을 넣는다.
msr     spsr, r13                       @ spsr_und 에 r13_und값을 저장한다.
mov     lr, pc                          @ lr_und에는 현재 pc 값에 8을 더한 주소를 저장한다.
movs    pc, lr                          @ SVS모드로 전환 하면서 다음 instruction으로 pc값을 옮긴다.
.endm
@ 위에 내용을 정리하면 undefined exception이 발생하면, undefined instruction이 발생한 CPSR값과
@ 복귀할 PC값을 저장하기위해 stack에서 비워두었던 최상위 주소의 2word에 lr_und,spsr_und 레지스터 값을
@ 저장한 후 supervisor mode로 전환한다.
@ *_und는 undefined excetion에 한하여 설명한 것이다, 이 macro는 모든 exception에서 공통으로 사용된다.
```

Start.S 분석(13) – (TOPDIR)/cpu/arm920t/start.S

❖ Macro bad_save_user_regs

```
.macro  bad_save_user_regs              @ SVC mode
sub     sp, sp, #S_FRAME_SIZE           @ stack point (sp_SVC) 을 72(18words)만큼 아래로 이동한다
stmia   sp, {r0 - r12}                  @ 현재의 r0-r12의 레지스터 내용을 stack에 저장한다.
ldr     r2, _armboot_end                @ r2 레지스터에  armboot_end address을 load한다.
add     r2, r2, #CONFIG_STACKSIZE       @ r2 <= r2 + 128Kbyte
sub     r2, r2, #8                      @ r2 <= r2  - 8
ldmia   r2, {r2 - r3}                   @ pc(lr_und) -> r2 , cpsr(spsr_und) ->r3
add     r0, sp, #S_FRAME_SIZE           @ stack point(sp_SVC) + S_FRAME_SIZE -> r0

add     r5, sp, #S_SP                   @ r0-r12이후 저장될 레지스터의 stack 위치를 저장
mov     r1, lr                          @ lr_SVC -> r1
stmia   r5, {r0 - r3}                   @ sp_SVC, lr_SVC, pc, cpsr을 stack에 저장
mov     r0, sp                          @ 저장된 레지스터의 base address을 r0로 이동
.endm
@ 위에 내용을 정리하면 undefined intruction 이 발생했을 때의 r0-r12,sp,lr,pc,cpsr을 stack에 저장 후
@ base address인 r0을 do_undefined_instruction() 함수의 인자로 사용한다.
@ do_undefined_instruction() -> (TOPDIR)/cpu/arm920t/interrupts.c
```

Start.S 분석(14) – (TOPDIR)/cpu/arm920t/start.S

- **Do_undefined_instruction**
- **Pt_regs 구조체의 배열은 Stack에 저장되어진 레지스터가 된다.**
- **Exception이 발생 하기 이전의 모든 레지스터를 확인할 수 있다.**

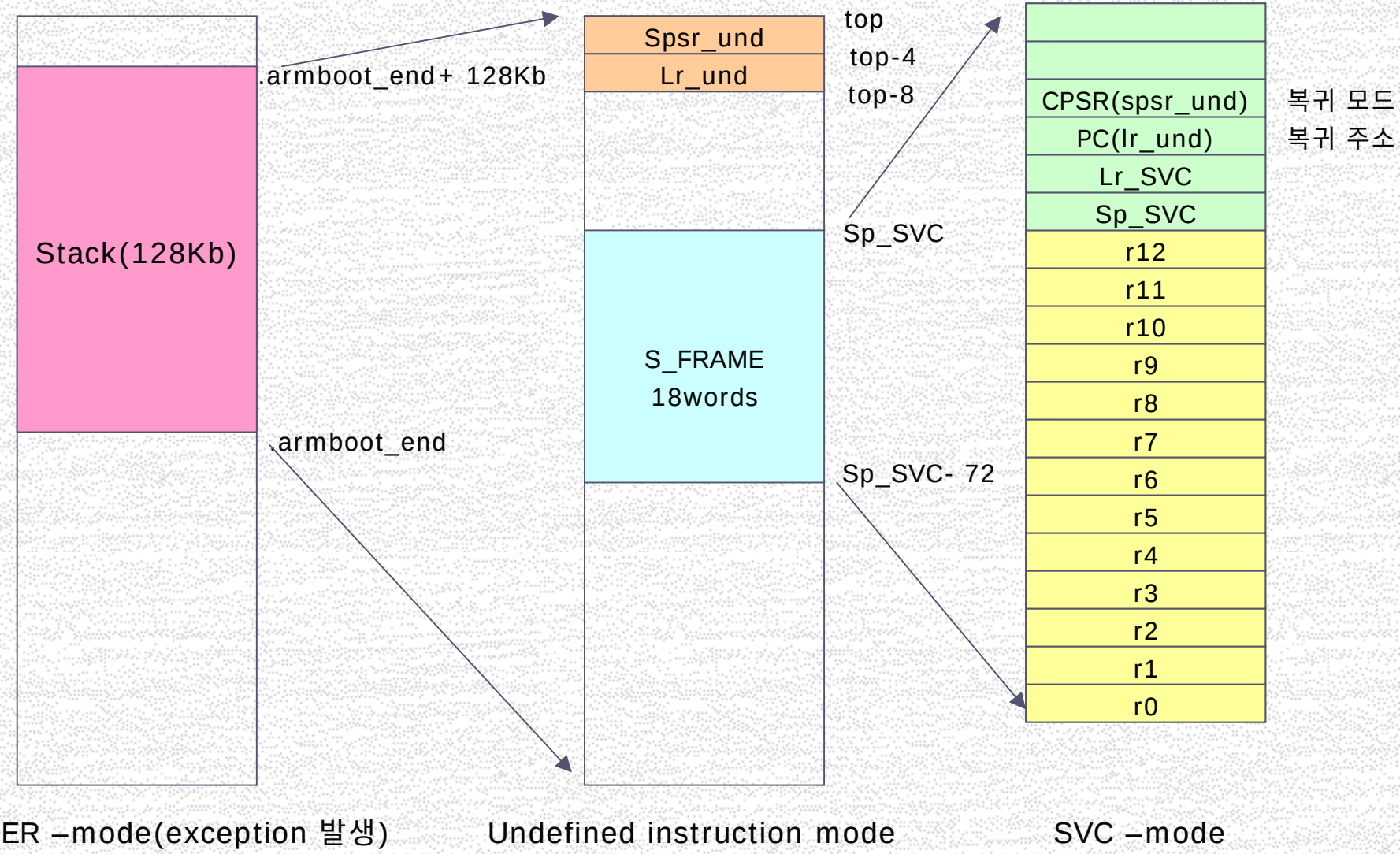
(TOPDIR)/include/asm-arm/proc-armv/ptrace.h

```
struct pt_regs {
        long uregs[18];
};
```

(TOPDIR)/cpu/arm920t/interrupts.c

```
void do_undefined_instruction (struct pt_regs *pt_regs)
{
        printf ("undefined instruction\n");
        show_regs (pt_regs);
        bad_mode ();
}
```

Start.S 분석(15) – (TOPDIR)/cpu/arm920t/interrupts.c



Start.S 분석(11) – (TOPDIR)/cpu/arm920t/start.S

- **IRQ exception**
  - IRQ pin에 active 입력이 들어올 경우 발생한다.
  - **CPU actions**
    - ✓ r14_irq = address of next inst. to be executed + 4
    - ✓ SPSR_irq = CPSR
    - ✓ CPSR[5:0] = 0b010010
    - ✓ CPSR[6] = unchanged
    - ✓ CPSR[7] = 1
    - ✓ PC = 0x18

```
        .align      5
irq:
        get_irq_stack
        irq_save_user_regs
        bl          do_irq
        irq_restore_user_regs
```

Start.S 분석(12) – (TOPDIR)/cpu/arm920t/start.S

❖ Macro get_irq_stack

```
#ifdef CONFIG_USE_IRQ
/* IRQ stack memory (calculated at run-time) */
.globl IRQ_STACK_START
IRQ_STACK_START:
        .word   0x0badc0de                  @ 임임의 값으로 추정된다.
@ 실제 값은 (TOPDIR)/cpu/arm920t/cpu.c cpu_init()에서 정의된다.
@ (TOPDIR)/lib_arm/board.c start_armboot()에서 interrupt을 enable 해야만 활성화 된다.
```

```
int cpu_init (void)                         @ (TOPDIR)/include/configs/smdk2410.h
{           /* setup up stack if necessary  @ CONFIG_STACKSIZE_IRQ ,_FIQ(4*1024)
#ifdef CONFIG_USE_IRQ
        IRQ_STACK_START = _armboot_end +
                          CONFIG_STACKSIZE + CONFIG_STACKSIZE_IRQ - 4;
        FIQ_STACK_START = IRQ_STACK_START + CONFIG_STACKSIZE_FIQ;
        _armboot_real_end = FIQ_STACK_START + 4;
#else
        _armboot_real_end = _armboot_end + CONFIG_STACKSIZE;
#endif  /* CONFIG_USE_IRQ */
        return (0);
}
```

Start.S 분석(13) – (TOPDIR)/cpu/arm920t/start.S

❖ Macro irq_save_user_regs

```
.macro get_irq_stack                    @ setup IRQ stack
        ldr     sp, IRQ_STACK_START
        .endm
```

```
.macro  irq_save_user_regs
sub     sp, sp, #S_FRAME_SIZE      @ stack point (sp_irq) 을 72(18words)만큼 아래로 이동한다.
stmia   sp, {r0 - r12}             @ 현재의 r0-r12의 레지스터 내용을 stack에 저장한다.
add     r8, sp, #S_PC              @ R8 <- sp_irq + S_PC(60)
stmdb   r8, {sp, lr}^              @ sp_usr, lr_usr을 r8이 가리키는 stack point-12에 저장
str     lr, [r8, #0]               @ r8이 가리키는 stack point 에 lr_irq을 저장
mrs     r6, spsr
str     r6, [r8, #4]               @ (r8 + 4)가리키는 stack point 에 spsr_irq을 저장
str     r0, [r8, #8]               @ (r8 + 8)가리키는 stack point 에 R0을 저장(OLD_R0)
mov     r0, sp                     @ 저장된 레지스터의 base address을 r0로 이동
.endm
```

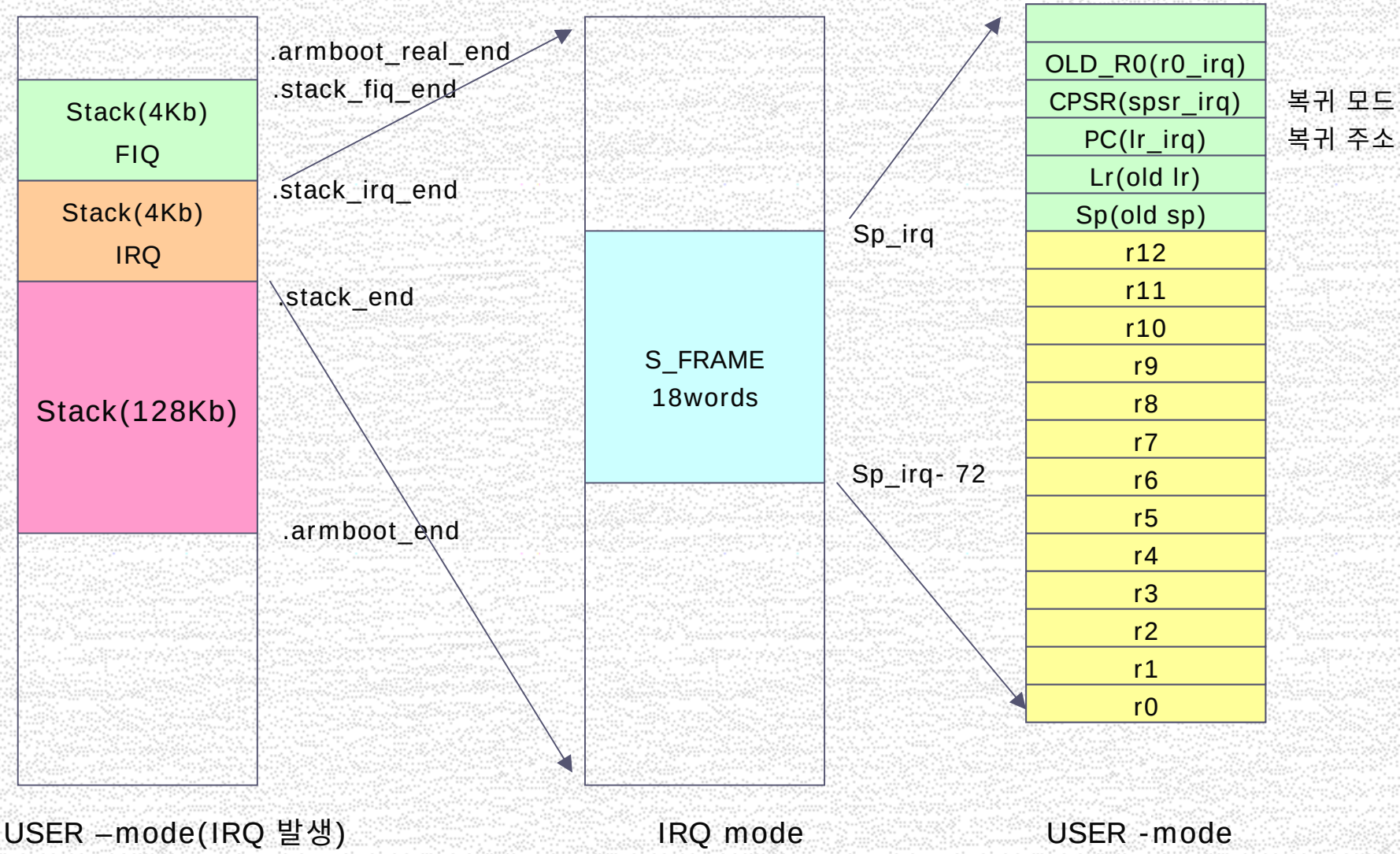
Start.S 분석(14) – (TOPDIR)/cpu/arm920t/start.S

❖ Macro irq_restore_user_regs

```
void do_irq (struct pt_regs *pt_regs)
{
        printf ("interrupt request\n");
        show_regs (pt_regs);
        bad_mode ();
}
```

```
.macro    irq_restore_user_regs
ldmia     sp, {r0 - lr}^          @ user mode의 레지스터를 r0-lr까지 stack의 내용으로 복원
mov       r0, r0                  @ dummy instruction
ldr       lr, [sp, #S_PC]         @ stack에 저장했던 lr_irq값을 lr_irq에 복원,lr_irq =>복귀할 주소
add       sp, sp, #S_FRAME_SIZE   @ sp_irq에서 사용한 stack을 반환한다.
subs      pc, lr, #4              @ spsr_irq을 cpsr로 이동하고 return한다.
.endm
```

Start.S 분석(15) – (TOPDIR)/cpu/arm920t/interrupts.c



Start.S 분석(16) – (TOPDIR)/cpu/arm920t/start.S

❖ **Void reset_cpu(ulong addr)**

```
        .align      5
.globl reset_cpu
reset_cpu:

#ifdef CONFIG_S3C2400                   @ 2410만 설명함
        중략....

#else /* !CONFIG_S3C2400 */
        mov     ip, #0
        mcr     p15, 0, ip, c7, c7, 0       @ I&D cache flush한다.
        mcr     p15, 0, ip, c8, c7, 0       @ I&D TLB (v4)을 flush한다.
        mrc     p15, 0, ip, c1, c0, 0       @ get ctrl register
        bic     ip, ip, #0x000f             @ Little endian, data cache, fault check, mmu disable
        bic     ip, ip, #0x2100             @ exception ->low address,inst. Cache disable
        mcr     p15, 0, ip, c1, c0, 0       @ ctrl register
        mov     pc, r0                      @ r0가 0면 software reset이 된다.
#endif /* CONFIG_S3C2400 */
```

인라인 어셈블리 기초(1)

- **인라인 어셈블리 다음과 같은 구조로 되어있다. 물론 사용하지 않는 것은 빼고 사용할 수 있다.**
  - 어셈블리 코드
  - OUTPUT 변수
  - INPUT변수
  - 값이 바뀌는 레지스터
- **사용되는 문법**
  - __asm__ __volatile__(asms: output : input : clobber);
  - __asm__ : 다음에 나오는 것이 인라인 어셈블리임을 나타낸다. ANSI에서는 __asm__으로만 정의되어 있으므로 'asm'과 같은 keyword는 사용하지 않는 것이 바람직하다.
  - __volatile__ : 이 키워드를 사용하면 컴파일러는 프로그램머가 입력한 그대로 남겨두게 된다. 즉 최적화나 위치를 옮기는 등의 일을 하지 않는다.
  - asms : 따옴표로 둘러싸인 어셈블리 코드. 코드 내에서는 %x과 같은 형태로 input,output 파라미터를 사용할 수 있으며 컴파일하면 파라미터가 치환된 대로 어셈블리 코드로 나타난다.
  - Output : 변수들은 따옴표로 둘러싸여 있으며, 각각은 쉼표로 구분되어진다. 결과 값은 출력하는 변수를 적는다.
  - Input : output과 같은 방식으로 사용하고 인라인 어셈블리 코드에 넘겨주는 파라미터를 적는다.
  - Clobber : output,input에 명시되어 있지 않지만 asms를 실행해서 값이 변하는 것을 적어준다. 각 변수는 쉼표로 구분되고 각각을 따옴표로 감싸준다.

인라인 어셈블리 기초(2)

- **Asms는 반드시 있어야 하지만 output, input, clobber는 각각 없을 수도 있다. 만약 clobber가 없는 경우 라면 clobber와 바로 앞의 콜론(:) 을 같이 쓰지 않아도 된다. 마찬 가지로 input,clobber가 없다면 output까지만 쓰면 된다. 그러나 output,clobber는 있고 input이 없는 경우엔 다음과 같이 input만을 제외한 나머지는 반드시 써주어야 한다.**
  - __asm__ __volatile__ (asms : output: : clobber);
- **어셈블리**
  - 인라인 어셈블리 중 asms 에 해당하는 실제코드를 적는 부분은 AT&T 어셈블리 문법을 따르고 여기에 적인 그대로가 컴파일 후 gasm에 넘겨지기 때문에 gasm의 문법을 따라야 한다.
  - 명령의 구분은 세미콜론(;)이나 개행문자(\n)으로 한다.
  - 인라인 에셈블리에선 %0,%1등을 사용해 input,output 오퍼랜드를 나타낸다. Output에서 부터 시작해 input에 나열된 변수들의 순서대로 %0,%1,...으로 번호가 매겨진다.
  - 모든 코드는 따옴표 안에 있어야하기 때문에 많은 수의 명령을 한줄로 쓰면 보기도 않좋기 때문에 명령어수가 많아지면 각 명령을 따옴표로 감싸고 뒤에 \t\n 을 넣고 다음 줄에 다시 명령을 따옴표로 적으면 된다.
- **OUTPUT / INPUT**
  - Output,input은 constraints와 변수이름이 쉼표로 구분된 리스트로 이루어져 있다.
  - Constraints은 의미를 나타내는 문자와 몇 가지의 modifier를 조합해 만들어진다.

인라인 어셈블리 기초(3)

- **Constraints(gnu- gcc manual 참조,또는 info gcc)**
  - “m” : 아키텍쳐가 지원하는 모든 종류의 메모리를 사용하는 오퍼랜드
  - “o” : 옵셋화 가능한 어드레스를 사용하는 메모리 오퍼랜드
  - “V” : 옵셋화 불가능한 어드레스를 사용하는 메모리 오퍼랜드
  - “<“ : 자동감소(미리 감소하거나 나중에 감소한다.) 어드레스용 메모리 오퍼랜드
  - “>” : 자동 증가(미리 증가하거나 나중에 증가한다.) 어드레스용 메모리 오퍼랜드
  - “r” : 일반 레지스터 사용 오퍼랜드
  - “d”,”a”,”f”,... : 아키텍쳐에 의해 변하는 오퍼랜드
  - “I” : immediate 정수 값을 나타내는 오퍼랜드. 심볼로된 상수도 여기에 해당된다.
  - “n” : immediate 정수 값으로 알려진 정수 값을 나타낸다. 많은 시스템이 어셈블할 때 한 워드 이하의 오퍼랜드용 상수를 지원하지 않으므로 “i“ 보다는 “n”을 사용하는 것이 바람직하다.
  - “I”,”J”,”K”,...”P” : 아키텍쳐에 의해 변하는 오퍼랜드
  - “E” : immediate 플로팅 오퍼랜드로 호스트와 같은 타겟 플로팅 포인트 포맷인 경우에만 사용 가능하다.
  - “F” : immediate 플로팅 오퍼랜드
  - “G”,”H” : 아키텍쳐에 의해 변하는 오퍼랜드
  - “s” : 값이 명확히 정해지지 않은 immediate 정수를 나타내는 오퍼랜드
  - “g” : 특수 레지스터를 제외한 일반 레지스터,메모리 혹은 immediate 정수 중 아무것이나 나타내는 오퍼랜드

인라인 어셈블리 기초(4)

- **Constraints(gnu- gcc manual 참조,또는 info gcc)**
  - “0”,”1”,”2”,...”9” : 같이 사용된 오퍼랜드의 번호를 나타냄.
  - “p” : 올바른 메모리 어드레스를 나타내는 오퍼랜드. “load address”와”push address”명령을 위한 것.
  - “Q”,”R”,”S”,...”U” : 아키텍쳐에 의해 변하는 오퍼랜드
- **ARM Family Constaints**
  - “f” : 플로팅 포인트 레지스터
  - “F” : 0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0, 10.0중의 하나를 나타내는 플로팅 포인트 상수
  - “G” : 음수 값인 경우의 “F”
  - “I” : 데이터 프로세싱 명령에서 유효한 immediate 정수 값 오퍼랜드. 0-255사이의 2의 배수 값을 나타낸다.
  - “J” : -4095에서 4095 사이의 정수.
  - “K” : “I”을 만족하는 값을 1의 보수 취한 것.
  - “L” : “I”을 만족하는 값을 음수로 취한 값(2의 보수)
  - “M” : 0에서 32사이의 정수 값
  - “Q” : 한 레지스터에 담겨있는 정확한 어드레스를 나타내는 메모리
  - “R” : constant pool내의 아이템
  - “S” : 현재 파일의 텍스트 세그먼트 내의 심볼

인라인 어셈블리 기초(5)

- **Modifier(gnu- gcc manual 참조,또는 info gcc)**
  - "=" : 오퍼랜드가 쓰기 전용임을 나타낸다. 이전 값은 없어지고 새로운 값으로 교체됨.
  - "+" : 읽기, 쓰기 모두 가능. "="는 보통 output용 "+"는 input/output 모두에 사용 가능하다. 나머지는 다른 모든 오퍼랜드는 input전용으로 간주된다.
  - "&" : "earlyclobber" 오퍼랜드를 나타내고 input오퍼랜드를 사용하는 명령이 끝나기 전에 변경된다는 것을 의미한다. 그래서 input 오퍼랜드나 메모리 어드레스의 일부을 나타내는 레지스터엔 못 쓴다.

    Gcc는 input변수가 다 사용되고 output에 사용된다고 가정하기 때문에 input에 사용된 변수가 output 과 같게 되고 또 output이 input보다 먼저 사용되는 경우가 발생할 수 있다. 이런 경우를 막기위해 output에 사용된 변수가 input이 모두 사용되기 전에 변경될 수도 있다고 알려줘야만 input과 output이 같아져 생기는 에러를 막을 수 있다.
  - "%" : %뒤에 따라오는 오퍼랜드로 대체 가능함을 나타낸다.
  - "#" : #이후의 쉼표가 나올 때까지의 모든 문자를 Constraints로 취급하지 않는다.

Cpu.c 분석(1) – (TOPDIR)/cpu/arm920t/cpu.c

❖ **Co-processor 15, register #1(control register)**

bit	name	function	Value	RST	Int
31	iA bit	Asynchronous clock select	iA:0 nF:0 = FastBus, iA:1 nF:0 = Reserved	0	0
30	nF bit	notFastBus select	iA:0 nF:1 = Sync. , iA:1 nF:1 = Async.	0	0
29:15	-	Reserved	Read = unpredictable write= Should be zero	0...	0...
14	RR bit	Round robin replacement	0= Random , 1= Round robin replacement	0	0
13	V bit	Base location of exception	0= Low addr. , 1= High addr.(hardware)	0,1	0
12	I bit	Instruction cache enable	0= instruction cache disable ,1 = enable	0	1
11:10	-	Reserved	Read = 00 , Write = 00	00	00
9	R bit	ROM protection	S:0 R:0 = no access, S:1 R:1= Reserved	0	0
8	S bit	System protection	S:1 R:0= SVC(read only), User(no access) S:0 R:1= SVC(read only),User(read only)	0	0
7	B bit	Big-endian/little-endian	0 = Little-endian , 1 = big-endian	0	0
6:3	-	Reserved	Read = 1111 , write = 1111	0	0
2	C bit	Data cache enable	0 = data cache disable , 1 = enable	0	0
1	A bit	Alignment fault enable	0 = fault checking disable , 1 = enable	0	1
0	M bit	MMU enable	0 = MMU disable , 1 = enable	0	0

Cpu.c 분석(2) – (TOPDIR)/cpu/arm920t/cpu.c

❖ **cpu.c – read_p15_c1()**

➢ Co-processor 15번의 register #1을 읽어서 value값에 저장하고 return한다.

```
static unsigned long read_p15_c1 (void)
{
        unsigned long value;
        @ inline asm을 volatile type이라고 컴파일러 알려준다.
                __asm__ __volatile__(
        @ asms :\n 개행 문자
                "mrc    p15, 0, %0, c1, c0, 0   @ read control reg\n"
        @ output : 오퍼랜드로 레지스터를 사용하고 출력변수는 value이다.
                : "=r" (value)
        @ input : none
                :
        @ clobber : input,output에는 정의되어 있지 않지만 asm을 실행해서 값이 변하는 것을 적어준다 .(stack)
                : "memory");
#ifdef MMU_DEBUG
        printf ("p15/c1 is = %08lx\n", value);
#endif
        return value;
}
```

Cpu.c 분석(3) – (TOPDIR)/cpu/arm920t/cpu.c

❖ **cpu.c – write_p15_c1()**

➢ Co-processor 15번의 register #1에 value값에 저장한다.

```
static void write_p15_c1 (unsigned long value)
{
#ifdef MMU_DEBUG
        printf ("write %08lx to p15/c1\n", value);
#endif
        @ inline asm을 volatile type이라고 컴파일러 알려준다.
        __asm__ __volatile__(
        @ asms :\n 개행 문자
                "mcr    p15, 0, %0, c1, c0, 0   @ write it back\n"
        @ output : none
                :
        @ input : 오퍼랜드를 레지스터로 사용하고 입력변수 value이다.
                : "r" (value)
        @ clobber : input,output에는 정의되어 있지 않지만 asm을 실행해서 값이 변하는 것을 적어준다.
                : "memory");
        @ co-processor 15번의 레지스터 #1을 read한다.???
        read_p15_c1 ();
}
```

Cpu.c 분석(4) – (TOPDIR)/cpu/arm920t/cpu.c

❖ **cpu.c – cpu_init()**

➢ Interrupt을 사용한다면 IRQ,FIQ의 stack을 구하고, armboot의 실제 end address을 구한다.

```
@ IRQ exception에서 설명한 내용 참조.
@ CONFIG_STACKSIZE          (128*1024)          /* regular stack */
@ CONFIG_STACKSIZE_IRQ      (4*1024)            /* IRQ stack */
@ CONFIG_STACKSIZE_FIQ      (4*1024)            /* FIQ stack */
int cpu_init (void)
{
        @ interrupt을 사용한다면 IRQ,FIQ용 각각의 stack 위치를 구하고,
        @ armboot 실제 end address을 구한다. (현재는 interrupt을 사용하지 않는다.)
#ifdef CONFIG_USE_IRQ
        IRQ_STACK_START = _armboot_end +
                          CONFIG_STACKSIZE + CONFIG_STACKSIZE_IRQ - 4;
        FIQ_STACK_START = IRQ_STACK_START + CONFIG_STACKSIZE_FIQ;
        _armboot_real_end = FIQ_STACK_START + 4;
#else
        _armboot_real_end = _armboot_end + CONFIG_STACKSIZE;
#endif  /* CONFIG_USE_IRQ */
        return (0);
}
```

Cpu.c 분석(5) – (TOPDIR)/cpu/arm920t/cpu.c

❖ **cpu.c – cleanup_before_linux()**

➢ Linux을 실행하기 전에 I/D-cache을 disable하고 I/D-cache을 flush한다.

```
#define C1_DC                   (1<<2)          /* dcache off/on */
#define C1_IC                   (1<<12)         /* icache off/on */
int cleanup_before_linux (void)
{
        unsigned long i;

        disable_interrupts ();

        @ turn off I/D-cache
        asm ("mrc p15, 0, %0, c1, c0, 0":"=r" (i));
        i &= ~(C1_DC | C1_IC);
        asm ("mcr p15, 0, %0, c1, c0, 0": :"r" (i));

        @ flush I/D-cache
        i = 0;
        asm ("mcr p15, 0, %0, c7, c7, 0": :"r" (i));
        return (0);
}
```

Cpu.c 분석(6) – (TOPDIR)/cpu/arm920t/cpu.c

❖ **cpu.c – do_reset()**

➢ Interrupt,mmu,cache를 disable시키고 mmu와 cache는 flush한 다음 0번지로 pc값을 옮긴다.

```
int do_reset (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[])
{
        @ start.S에 있는 function을 호출하기 위하여 extern선언 한다.
        extern void reset_cpu (ulong addr);

        @ interrupt (irq/fiq)을 disable한다.
        disable_interrupts ();
        @ co-processor 15을 초기화 하고 0x00000000번지로 pc값을 옮긴다.
        @ mmu,cache을 flush하고 disable 시킨다.
        reset_cpu (0);
        /*NOTREACHED*/
        return (0);
}
```

Cpu.c 분석(7) – (TOPDIR)/cpu/arm920t/cpu.c

❖ **cpu.c – icache_enable,disable()**

➢ I-cache을 on/off하거나 현재 상태를 검사하는 함수 이다.

```
void icache_enable (void){
        ulong reg;

        reg = read_p15_c1 ();
        cp_delay ();                    @ co-processor용 delay
        write_p15_c1 (reg | C1_IC);     @ icache enable
}
void icache_disable (void){
        ulong reg;

        reg = read_p15_c1 ();
        cp_delay ();                    @ co-processor용 delay
        write_p15_c1 (reg & ~C1_IC);    @ icache enable
}
int icache_status (void){
        return (read_p15_c1 () & C1_IC) != 0;    @ 현재 icache가 on되어 있으면 1을 return
}
```

Cpu.c 분석(8) – (TOPDIR)/cpu/arm920t/cpu.c

❖ **cpu.c – dcache_enable,disable()**

➢ d-cache을 on/off하거나 현재 상태를 검사하는 함수 이다.

```
void dcache_enable (void){
        ulong reg;

        reg = read_p15_c1 ();
        cp_delay ();                    @ co-processor용 delay
        write_p15_c1 (reg | C1_DC);     @ dcache enable
}
void dcache_disable (void){
        ulong reg;

        reg = read_p15_c1 ();
        cp_delay ();                    @ co-processor용 delay
        reg &= ~C1_DC;
        write_p15_c1 (reg);             @ dcache enable
}
int dcache_status (void){
        return (read_p15_c1 () & C1_DC) != 0;    @ 현재 dcache가 on되어 있으면 1을 return
}
```

Speed.c 분석(1) – (TOPDIR)/cpu/arm920t/speed.c

- **PLL(1)-CPU PLL routine을 분석하기 전에 S3C2410 datasheet참조**
  - S3C2410은 MPLL(main PLL),UPLL(USB PLL) 2개 가지고 있다.

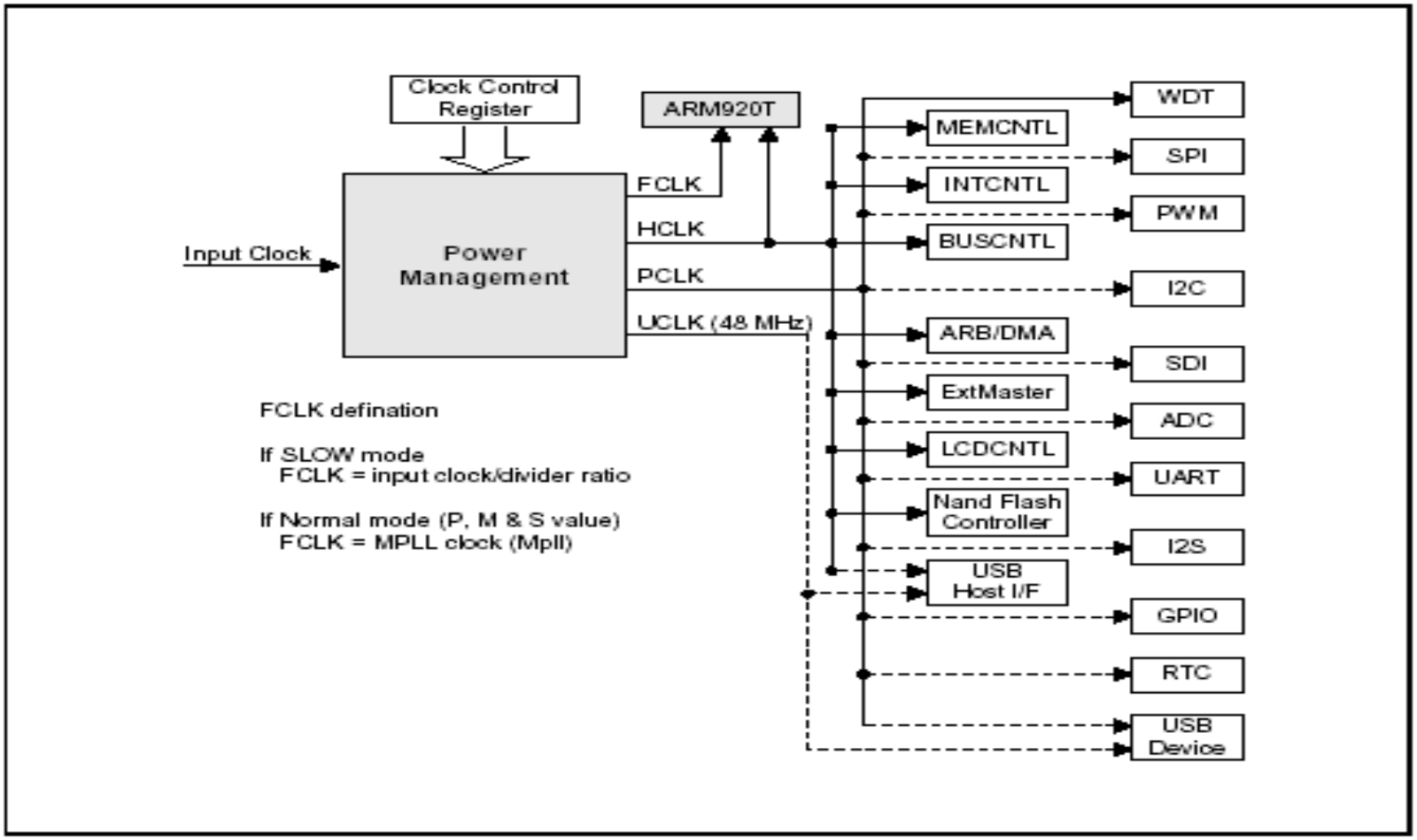


Figure 7-7. The Clock Distribution Block Diagram

Speed.c 분석(2) – (TOPDIR)/cpu/arm920t/speed.c

❖ **PLL(2)**

➢ S3C2410의 MPLL(main PLL)은 CLOCK Controller에 의해 FCLK,HCLK,PCLK으로 구성된다.

FCLK, HCLK, and PCLK

FCLK is used by ARM920T.
HCLK is used for AHB bus, which is used by the ARM920T, the memory controller, the interrupt controller, the LCD controller, the DMA and the USB host block.
PCLK is used for APB bus, which is used by the peripherals such as WDT, IIS, I2C, PWM timer, MMC interface, ADC, UART, GPIO, RTC and SPI.

The S3C2410X supports selection of Dividing Ratio between FCLK, HLCK and PCLK. This ratio is determined by HDIVN and PDIVN of CLKDIVN control register.

HDIVN	PDIVN	FCLK	HCLK	PCLK	Divide Ratio
0	0	FCLK	FCLK	FCLK	1 : 1 : 1 (Default)
0	1	FCLK	FCLK	FCLK / 2	1 : 1 : 2
1	0	FCLK	FCLK / 2	FCLK / 2	1 : 2 : 2
1	1	FCLK	FCLK / 2	FCLK / 4	1 : 2 : 4 (recommended)

CLOCK DIVIDER CONTROL (CLKDIVN) REGISTER

Register	Address	R/W	Description	Reset Value
CLKDIVN	0x4C000014	R/W	Clock divider control register	0x00000000

CLKDIVN	Bit	Description	Initial State
Reserved	[2]	Special bus clock ratio for the chip verification.	0
HDIVN	[1]	0: HCLK has the clock same as the FCLK. 1: HCLK has the clock same as the FCLK/2.	0
PDIVN	[0]	0: PCLK has the clock same as the HCLK. 1: PCLK has the clock same as the HCLK/2.	0

Speed.c 분석(3) – (TOPDIR)/cpu/arm920t/speed.c

- **PLL(3)**
  - Mpll = (m * Fin)/(p * 2^{s})
  - m = (MDIV + 8), p = (PDIV + 2), s = SDIV, Fin = 입력 주파수
  - m = (0x5c + 8), p = (0x08 + 2), s = 0x00, Fin = 12Mhz
    - ✓ Mpll = (m(100) * Fin(12Mhz)) / (p(10) * 2) => 1200Mhz / 10 =>120Mhz(FCLK)

Register	Address	R/W	Description	Reset Value
MPLLCON	0x4C000004	R/W	MPLL configuration register	0x0005C080
UPLLCON	0x4C000008	R/W	UPLL configuration register	0x00028080

PLLCON	Bit	Description	Initial State
MDIV	[19:12]	Main divider control	0x5C / 0x28
PDIV	[9:4]	Pre-divider control	0x08 / 0x08
SDIV	[1:0]	Post divider control	0x0 / 0x0

NOTE: When you set MPLL&UPLL values simultaneously, set MPLL value first and then UPLL value.

Register Name	Address (B. Endian)	Address (L. Endian)	Acc. Unit	Read/ Write	Function
Clock & Power Management					
LOCKTIME	0x4C000000	←	W	R/W	PLL Lock Time Counter
MPLLCON	0x4C000004				MPLL Control
UPLLCON	0x4C000008				UPLL Control
CLKCON	0x4C00000C				Clock Generator Control
CLKSLOW	0x4C000010				Slow Clock Control
CLKDIVN	0x4C000014				Clock divider Control

Speed.c 분석(4) – (TOPDIR)/cpu/arm920t/speed.c

- **S3c2410.h – (TOPDIR)/include/s3c2410.h**
  - S3c2410의 hardware관련 register들을 정리한 header 파일이다.

```
        ....
#define S3C24X0_CLOCK_POWER_BASE    0x4C000000
        ....
static inline S3C24X0_CLOCK_POWER * const S3C24X0_GetBase_CLOCK_POWER(void)
{
        return (S3C24X0_CLOCK_POWER * const)S3C24X0_CLOCK_POWER_BASE;
}
```

- **S3c24x0.h – (TOPDIR)/include/s3c24x0.h**

```
        ....
typedef struct {
        S3C24X0_REG32       LOCKTIME;
        S3C24X0_REG32       MPLLCON;
        S3C24X0_REG32       UPLLCON;
        S3C24X0_REG32       CLKCON;
        S3C24X0_REG32       CLKSLOW;
        S3C24X0_REG32       CLKDIVN;
} /*__attribute__((__packed__))*/ S3C24X0_CLOCK_POWER;
```

Speed.c 분석(5) – (TOPDIR)/cpu/arm920t/speed.c

❖ **speed.c – get_PLLCLK()**

➢ 현재 setting되어있는 MPLL,UPLL값을 얻어오는 함수이다.(MPLL = 0,UPLL = 1)

```
static ulong get_PLLCLK(int pllreg)
{
   S3C24X0_CLOCK_POWER * const clk_power = S3C24X0_GetBase_CLOCK_POWER();
   ulong r, m, p, s;

   if (pllreg == MPLL)                    @ #define MPLL 0
         r = clk_power->MPLLCON;          @ MPLLCON register을 read한다.
   else if (pllreg == UPLL)               @ #define UPLL 1
         r = clk_power->UPLLCON;          @ UPLLCON register을 read한다.
   else
         hang();                          @ error처리,(TOPDIR)/lib_arm/board.c
@Mpll = (m * Fin)/(p * 2 s )
@m = (MDIV + 8), p = (PDIV + 2), s = SDIV, Fin = 입력 주파수
   m = ((r & 0xFF000) >> 12) + 8;         @ MDIV[19:12]
   p = ((r & 0x003F0) >> 4) + 2;          @ PDIV[9:4]
   s = r & 0x3;                           @ SDIV[1:0]
   return((CONFIG_SYS_CLK_FREQ * m) / (p << s));
} @ CONFIG_SYS_CLK_FREQ = 12000000,(TOPDIR)/include/configs/smdk2410.h
```

Speed.c 분석(6) – (TOPDIR)/cpu/arm920t/speed.c

- **speed.c – get_FCLK,HCLK,PCLK,UCLK()**
  - 각CLOCK값을 읽어온다.

```
ulong get_FCLK(void){
   return(get_PLLCLK(MPLL));              @ FCLK(MPLL)값을 return한다.
}

ulong get_HCLK(void){
   S3C24X0_CLOCK_POWER * const clk_power = S3C24X0_GetBase_CLOCK_POWER();
@ CLKDIVN register값을 읽어서 HDIVN bit의 값이 1이면 FCLK/2값을 0이면 FLCK값을 return한다.
   return((clk_power->CLKDIVN & 0x2) ? get_FCLK()/2 : get_FCLK());
}

ulong get_PCLK(void){
   S3C24X0_CLOCK_POWER * const clk_power = S3C24X0_GetBase_CLOCK_POWER();
@ CLKDIVN register값을 읽어서 PDIVN bit의 값이 1이면 HCLK/2값을 0이면 HLCK값을 return한다.
 return((clk_power->CLKDIVN & 0x1) ? get_HCLK()/2 : get_HCLK());
}

ulong get_UCLK(void){
   return(get_PLLCLK(UPLL));              @ UPLL값을 얻어온다.
}
```

nterrupts.c 분석(1) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **Interrupts.c – enable,disable_interrupts()**

➢ Interrupt(IRQ만)을 enable 또는 disable하는 함수이다.

```
void enable_interrupts (void){
        unsigned long temp;
        __asm__ __volatile__("mrs %0, cpsr\n"          @ cpsr값을 r0로 옮긴다.
                             "bic %0, %0, #0x80\n"     @ I bit을 clear한다.
                             "msr cpsr_c, %0"          @ r0값을 cpsr로 옮긴다.
                             : "=r" (temp)
                             : "memory");
}
int disable_interrupts (void){
        unsigned long old,temp;
        __asm__ __volatile__("mrs %0, cpsr\n"          @ cpsr값을 r0로 옮긴다.
                             "orr %1, %0, #0xc0\n"     @ I bit,F bit을 set한다.
                             "msr cpsr_c, %1"          @ r1값을 cpsr_c로 옮긴다.
                             : "=r" (old), "=r" (temp)
                             :
                             : "memory");
        return (old & 0x80) == 0;                      @ disable하기전의 I bit상태를 return
}@ disable하기전의 IRQ interrupt가 enable되어있었다면 1로 return한다.
```

nterrupts.c 분석(2) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **Interrupts.c – do_<exception handler>()**

➢ 각 exception에 대한 C 로 구현된 처리 함수이다.

```
void do_undefined_instruction (struct pt_regs *pt_regs){
        printf ("undefined instruction\n");        @ console에 현재 발생한 exception내용을 출력.
        show_regs (pt_regs);                       @ exception이 발생한 mode의 reg. 내용을 출력.
        bad_mode ();                               @ "panic"을 출력하고 warm reset을 한다.
}
void do_prefetch_abort (struct pt_regs *pt_regs)
        ...
void do_data_abort (struct pt_regs *pt_regs)
        ...
void do_not_used (struct pt_regs *pt_regs)
        ...
void do_fiq (struct pt_regs *pt_regs)
        ...
void do_irq (struct pt_regs *pt_regs){
        printf ("interrupt request\n");            @ console에 현재 발생한 exception내용을 출력.
        show_regs (pt_regs);                       @ exception이 발생한 mode의 reg. 내용을 출력.
        bad_mode ();                               @ "panic"을 출력하고 warm reset을 한다.
}
```

nterrupts.c 분석(3) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **Interrupts.c –bad_mode(),show_reg()**

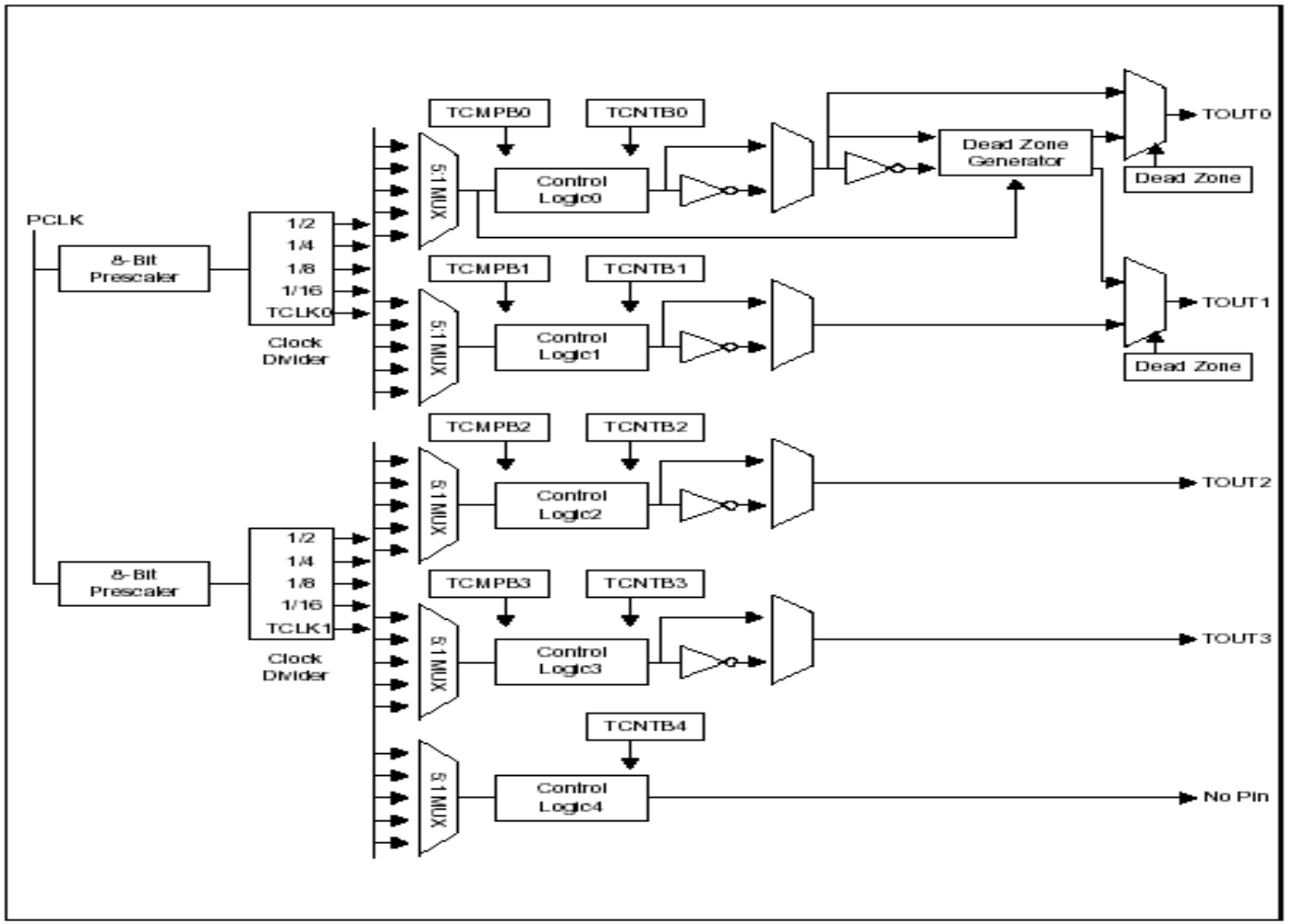
➢ Exception이 발생한 mode의 register bank의 모든 register을 console에 출력한다.

```
@ struct pt_req은  (TOPDIR)/include/asm-arm/proc-armv/ptrace.h에 정의되어 있다.
void bad_mode (void){
        panic ("Resetting CPU ...\n");
        reset_cpu (0);                          @ warm reset
}

@ exception이 발생한 mode의 register bank의 모든 register을 console에 출력한다.
void show_regs (struct pt_regs *regs){
        @ r0 – r10,fp,ip,sp,lr,pc값을 console에 출력한다.
        @ condition code(N,Z,C,V)을 console에 출력한다.
        @ IRQ on/off상태를 console에 출력한다.
        @ FIQ on/off상태를 console에 출력한다.
        @ processor mode을 console에 출력한다.
        @ thumb state 상태였는지 console에 출력한다.
}
```

nterrupts.c 분석(4) – (TOPDIR)/cpu/arm920t/interrupts.c

- **Timer(1)-timer routine을 분석하기 전에 S3C2410 datasheet참조**
  - S3C2410은 16bit timer을 5개 가지고 있다.



nterrupts.c 분석(5) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **Timer(2)**

➢ Timer count(TCNTn)은 기본적으로 매 timer clock마다 감소한다.

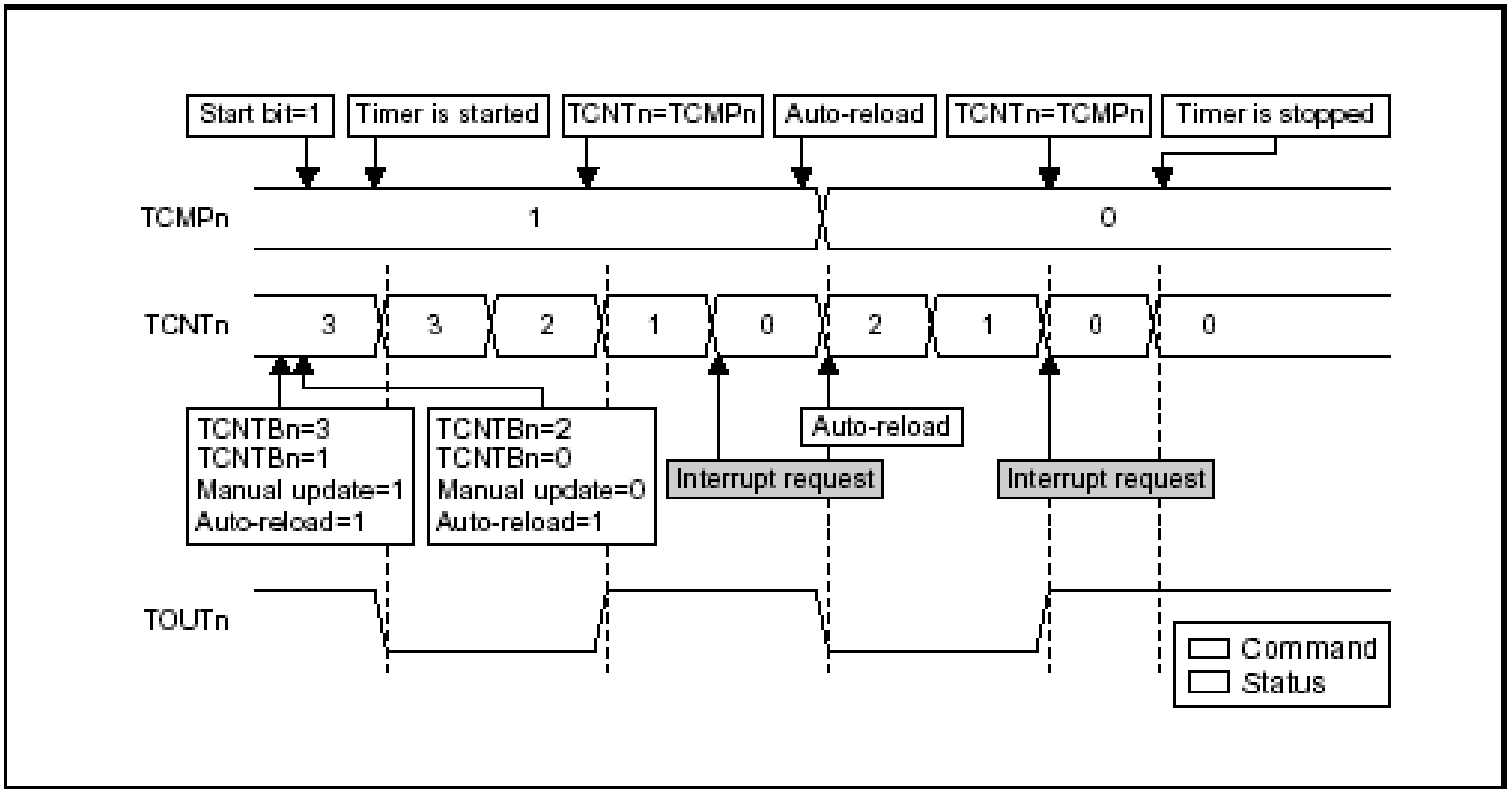


Figure 10-2. Timer Operations

nterrupts.c 분석(6) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ Timer(3)

- 4bit DIVIDER & 8bit Prescaler 1,2을 가지고 timer의 input clock을 만든다.

PRESCALER & DIVIDER

An 8-bit prescaler and a 4-bit divider make the following output frequencies:

4-bit divider settings	Minimum resolution (prescaler = 0)	Maximum resolution (prescaler = 255)	Maximum interval (TCNTBn = 65535)
1/2 (PCLK = 50 MHz)	0.0400 us (25.0000 MHz)	10.2400 us (97.6562 KHz)	0.6710 sec
1/4 (PCLK = 50 MHz)	0.0800 us (12.5000 MHz)	20.4800 us (48.8281 KHz)	1.3421 sec
1/8 (PCLK = 50 MHz)	0.1600 us (6.2500 MHz)	40.9601 us (24.4140 KHz)	2.6843 sec
1/16 (PCLK = 50 MHz)	0.3200 us (3.1250 MHz)	81.9188 us (12.2070 KHz)	5.3686 sec

- 8bit Prescaler 1,2값을 설정한다.

TIMER CONFIGURATION REGISTER0 (TCFG0)

Timer input clock Frequency = PCLK / {prescaler value+1} / {divider value}
{prescaler value} = 0~255
{divider value} = 2, 4, 8, 16

Register	Address	R/W	Description	Reset Value
TCFG0	0x51000000	R/W	Configures the two 8-bit prescalers	0x00000000

TCFG0	Bit	Description	Initial State
Reserved	[31:24]		0x00
Dead zone length	[23:16]	These 8 bits determine the dead zone length. The 1 unit time of the dead zone length is equal to that of timer 0.	0x00
Prescaler 1	[15:8]	These 8 bits determine prescaler value for Timer 2, 3 and 4.	0x00
Prescaler 0	[7:0]	These 8 bits determine prescaler value for Timer 0 and 1.	0x00

nterrupts.c 분석(7) – (TOPDIR)/cpu/arm920t/interrupts.c

- **Timer(4)**
  - 각 Timer의 Divider값을 설정,DMA request channel을 선택.

TIMER CONFIGURATION REGISTER1 (TCFG1)

Register	Address	R/W	Description	Reset Value
TCFG1	0x51000004	R/W	5-MUX & DMA mode selecton register	0x00000000

TCFG1	Bit	Description	Initial State
Reserved	[31:24]		00000000
DMA mode	[23:20]	Select DMA request channel 0000 = No select (all interrupt) 0001 = Timer0 0010 = Timer1 0011 = Timer2 0100 = Timer3 0101 = Timer4 0110 = Reserved	0000
MUX 4	[19:16]	Select MUX input for PWM Timer4. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = External TCLK1	0000
MUX 3	[15:12]	Select MUX input for PWM Timer3. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = External TCLK1	0000
MUX 2	[11:8]	Select MUX input for PWM Timer2. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = External TCLK1	0000
MUX 1	[7:4]	Select MUX input for PWM Timer1. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = External TCLK0	0000
MUX 0	[3:0]	Select MUX input for PWM Timer0. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = External TCLK0	0000

nterrupts.c 분석(8) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ Timer(5)

- 각 Timer의 start/stop,reload,update값을 설정

TIMER CONTROL (TCON) REGISTER

Register	Address	R/W	Description	Reset Value
TCON	0x51000008	R/W	Timer control register	0x00000000

TCON	Bit	Description	Initial state
Timer 4 auto reload on/off	[22]	Determine auto reload on/off for Timer 4. 0 = One-shot 1 = Interval mode (auto reload)	0
Timer 4 manual update (note)	[21]	Determine the manual update for Timer 4. 0 = No operation 1 = Update TCNTB4	0
Timer 4 start/stop	[20]	Determine start/stop for Timer 4. 0 = Stop 1 = Start for Timer 4	0
Timer 3 auto reload on/off	[19]	Determine auto reload on/off for Timer 3. 0 = One-shot 1 = Interval mode (auto reload)	0
Timer 3 output inverter on/off	[18]	Determine output inverter on/off for Timer 3. 0 = Inverter off 1 = Inverter on for TOUT3	0
Timer 3 manual update (note)	[17]	Determine manual update for Timer 3. 0 = No operation 1 = Update TCNTB3 & TCMPB3	0
Timer 3 start/stop	[16]	Determine start/stop for Timer 3. 0 = Stop 1 = Start for Timer 3	0
Timer 2 auto reload on/off	[15]	Determine auto reload on/off for Timer 2. 0 = One-shot 1 = Interval mode (auto reload)	0
Timer 2 output inverter on/off	[14]	Determine output inverter on/off for Timer 2. 0 = Inverter off 1 = Inverter on for TOUT2	0
Timer 2 manual update (note)	[13]	Determine the manual update for Timer 2. 0 = No operation 1 = Update TCNTB2 & TCMPB2	0
Timer 2 start/stop	[12]	Determine start/stop for Timer 2. 0 = Stop 1 = Start for Timer 2	0
Timer 1 auto reload on/off	[11]	Determine the auto reload on/off for Timer1. 0 = One-shot 1 = Interval mode (auto reload)	0
Timer 1 output inverter on/off	[10]	Determine the output inverter on/off for Timer1. 0 = Inverter off 1 = Inverter on for TOUT1	0
Timer 1 manual update (note)	[9]	Determine the manual update for Timer 1. 0 = No operation 1 = Update TCNTB1 & TCMPB1	0
Timer 1 start/stop	[8]	Determine start/stop for Timer 1. 0 = Stop 1 = Start for Timer 1	0

TIMER CONTROL (TCON) REGISTER (Continued)

TCON	Bit	Description	Initial state
Reserved	[7:5]	Reserved	
Dead zone enable	[4]	Determine the dead zone operation. 0 = Disable 1 = Enable	0
Timer 0 auto reload on/off	[3]	Determine auto reload on/off for Timer 0. 0 = One-shot 1 = Interval mode(auto reload)	0
Timer 0 output inverter on/off	[2]	Determine the output inverter on/off for Timer 0. 0 = Inverter off 1 = Inverter on for TOUT0	0
Timer 0 manual update (note)	[1]	Determine the manual update for Timer 0. 0 = No operation 1 = Update TCNTB0 & TCMPB0	0
Timer 0 start/stop	[0]	Determine start/stop for Timer 0. 0 = Stop 1 = Start for Timer 0	0

nterrupts.c 분석(9) – (TOPDIR)/cpu/arm920t/interrupts.c

- **Timer(6)**
  - Timer routine을 분석하기 전에 S3C2410 Datasheet참조

TIMER 4 COUNT BUFFER REGISTER (TCNTB4)

Register	Address	R/W	Description	Reset Value
TCNTB4	0x5100003C	R/W	Timer 4 count buffer register	0x00000000

TCNTB4	Bit	Description	Initial State
Timer 4 count buffer register	[15:0]	Set count buffer value for Timer 4	0x00000000

TIMER 4 COUNT OBSERVATION REGISTER (TCNTO4)

Register	Address	R/W	Description	Reset Value
TCNTO4	0x51000040	R	Timer 4 count observation register	0x00000000

TCNTO4	Bit	Description	Initial State
Timer 4 observation register	[15:0]	Set count observation value for Timer 4	0x00000000

nterrupts.c 분석(10) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **Timer(7)**

➢ Timer 관련 register들이다.

Register Name	Address (B. Endian)	Address (L. Endian)	Acc. Unit	Read/ Write	Function
PWM Timer					
TCFG0	0x51000000	←	W	R/W	Timer Configuration
TCFG1	0x51000004				Timer Configuration
TCON	0x51000008				Timer Control
TCNTB0	0x5100000C				Timer Count Buffer 0
TCMPB0	0x51000010				Timer Compare Buffer 0
TCNTO0	0x51000014			R	Timer Count Observation 0
TCNTB1	0x51000018			R/W	Timer Count Buffer 1
TCMPB1	0x5100001C				Timer Compare Buffer 1
TCNTO1	0x51000020			R	Timer Count Observation 1
TCNTB2	0x51000024			R/W	Timer Count Buffer 2
TCMPB2	0x51000028				Timer Compare Buffer 2
TCNTO2	0x5100002C			R	Timer Count Observation 2
TCNTB3	0x51000030			R/W	Timer Count Buffer 3
TCMPB3	0x51000034				Timer Compare Buffer 3
TCNTO3	0x51000038			R	Timer Count Observation 3
TCNTB4	0x5100003C			R/W	Timer Count Buffer 4
TCNTO4	0x51000040			R	Timer Count Observation 4

nterrupts.c 분석(11) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **S3c2410.h – (TOPDIR)/include/s3c2410.h**

- S3c2410의 hardware관련 register들을 정리한 header 파일이다.

```
        ....
#define S3C24X0_TIMER_BASE          0x51000000
        ....
static inline S3C24X0_TIMERS * const S3C24X0_GetBase_TIMERS(void)
{
        return (S3C24X0_TIMERS * const)S3C24X0_TIMER_BASE;
}
        .....
```

❖ **S3c24x0.h – (TOPDIR)/include/s3c24x0.h**

```
        ....
#typedef struct {
        S3C24X0_REG32     TCFG0;
        S3C24X0_REG32     TCFG1;
        S3C24X0_REG32     TCON;
        S3C24X0_TIMER     ch[4];
        S3C24X0_REG32     TCNTB4;
        S3C24X0_REG32     TCNTO4;
} /*__attribute__((__packed__))*/ S3C24X0_TIMERS;
```

nterrupts.c 분석(12) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **Interrupts.c – interrupt_init()**

➢ Timer4 초기화 함수이다.(timer4은 외부출력이 없다.)

```
int interrupt_init (void){
        S3C24X0_TIMERS * const timers = S3C24X0_GetBase_TIMERS();
        @ timer register base address -> timers
        timers->TCFG0 = 0x0f00;                    @ timer2,3,4의 prescaler값을 15(0xf)으로 설정한다
        if (timer_load_val == 0)                   @ init. Value = 0
        {
@ 10ms clock period 생성 : PCLK /{prescaler + 1}/divider => PCLK / {prescaler+1} * divider
@ get PCLK= 50Mhz, 4 bit divider default vale = ½, prescaler = 15(0x0f) + 1
@ PCLK 50Mhz => 50000000hz/16*2 = 1562500hz => 0.64us(1cycle) x 1562500 = 1sec
                timer_load_val = get_PCLK()/(2 * 16 * 100);
        }
        lastdec = timers->TCNTB4 = timer_load_val;          @ TCNTB = 15625(10ms)
@ Timer4 => autoload,manual update,timer stop
        timers->TCON = (timers->TCON & ~0x0700000) | 0x600000;
@ Timer4 => autoload, timer start,여기서부터 10ms 마다 timer4 가 작동된다.
        timers->TCON = (timers->TCON & ~0x0700000) | 0x500000;
        timestamp = 0;
        return (0);
}
```

nterrupts.c 분석(13) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **Interrupts.c – timer관련 함수(1)**

➢ Timer값을 setting하고 reset시키는 함수이다.

```
static inline ulong READ_TIMER(void)
{           @ 현재 timer4의 count값을 읽는다.
            S3C24X0_TIMERS * const timers = S3C24X0_GetBase_TIMERS();
            return (timers->TCNTO4 & 0xffff);          @ 16bit timer(real counter)
}
void set_timer (ulong t)
{           @ time value 의 start값을 setting한다. 입력된 time값을 timestamp변수에 저장 한다.
            timestamp = t;
}
void reset_timer_masked (void){
            @ timer 값을 reset시키기 위해 마지막으로 count된 값을 저장하고 timestamp변수 값을 clear한다.
            lastdec = READ_TIMER();
            timestamp = 0;
}

void reset_timer (void)
{           @ timer 값을 reset시키기 위해 마지막으로 count된 값을 저장하고 timestamp변수 값을 clear한다.
            reset_timer_masked ();
}
```

nterrupts.c 분석(14) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **Interrupts.c – timer관련 함수(2)**

➢ 현재의 Timer value을 얻어오는 함수이다.

```
ulong get_timer_masked (void)
{
        ulong now = READ_TIMER();                    @ 현재 timer count값을 읽는다. (감소 count)
@ 과거 timer count값이 현재 timer count값보다 크거나 같은지를 검사한다.
        if (lastdec >= now) {
@ normal mode, 과거 timer count – 현재 timer count(ex: 15000 – 14990 => timestamp + 10)
                timestamp += lastdec - now;
        } else {
@ overflow발생, 과거 timer count + timer setting - 현재 timer count(ex: 10 + 15625 – 15600 => 35)
                timestamp += lastdec + timer_load_val - now;
        }
        lastdec = now;                               @ 현재 timer count값 과거 count value에 저장.

        return timestamp;                            @ 현재의 time value을 return
}
ulong get_timer (ulong base)    {
        return get_timer_masked () - base;
}
```

nterrupts.c 분석(15) – (TOPDIR)/cpu/arm920t/interrupts.c

❖ **Interrupts.c – timer관련 함수(3)**

➢ 1sec에 필요한 timer value을 얻어오는 함수이다.

```
unsigned long long get_ticks(void)
{
        return get_timer(0);
}
@ #define          CFG_HZ    1562500 ;(TOPDIR)/include/configs/smdk2410.h???
ulong get_tbclk (void)
{
        ulong tbclk;

#if defined(CONFIG_SMDK2400) || defined(CONFIG_TRAB)
        tbclk = timer_load_val * 100;   @ 추천
#elif defined(CONFIG_SMDK2410) || defined(CONFIG_ATB2410) || defined(CONFIG_VCMA9)
        tbclk = CFG_HZ;                 @ 10ms(15625) * 100 = 1sec(CFG_HZ)
#else
#error "tbclk not configured"
#endif

        return tbclk;
}
```

nterrupts.c 분석(16) – (TOPDIR)/cpu/arm920t/interrupts.c

- **Interrupts.c – udelay**
  - Us동안 delay하는 함수이다.

```
@timer_load_val(10ms) = 15625
@1ms = 1562.5 , 100us = 156.25, 10us = 15.625, 1us = 1.5625
@usec = 10us가정하면
@usec/1000 = 0.01
@tmo(0.01) * (15625 *100) = 15625
@tmo = 15625 /1000 = 15.625
@현재 count value + 15.625
@15 timer clock동안 delay된다.=>0.64us x 15 = 10us
void udelay (unsigned long usec){
        ulong tmo;

        tmo = usec / 1000;
        tmo *= (timer_load_val * 100);
        tmo /= 1000;

        tmo += get_timer (0);
        while (get_timer_masked () < tmo);
}
```

Serial.c 분석(1) – (TOPDIR)/cpu/arm920t/serial.c

❖ **serial(1)-serial routine을 분석하기 전에 S3C2410 datasheet참조**

- S3C2410은 16byte-FIFO을 가지고있는 3개의 UART을 가지고있다.(IRDA Driver지원)

Register Name	Address (B. Endian)	Address (L. Endian)	Acc. Unit	Read/ Write	Function
UART					
ULCON0	0x50000000	←	W	R/W	UART 0 Line Control
UCON0	0x50000004				UART 0 Control
UFCON0	0x50000008				UART 0 FIFO Control
UMCON0	0x5000000C				UART 0 Modem Control
UTRSTAT0	0x50000010			R	UART 0 Tx/Rx Status
UERSTAT0	0x50000014				UART 0 Rx Error Status
UFSTAT0	0x50000018				UART 0 FIFO Status
UMSTAT0	0x5000001C				UART 0 Modem Status
UTXH0	0x50000023	0x50000020	B	W	UART 0 Transmission Hold
URXH0	0x50000027	0x50000024		R	UART 0 Receive Buffer
UBRDIV0	0x50000028	←	W	R/W	UART 0 Baud Rate Divisor
ULCON1	0x50004000	←	W	R/W	UART 1 Line Control
UCON1	0x50004004				UART 1 Control
UFCON1	0x50004008				UART 1 FIFO Control
UMCON1	0x5000400C				UART 1 Modem Control
UTRSTAT1	0x50004010			R	UART 1 Tx/Rx Status
UERSTAT1	0x50004014				UART 1 Rx Error Status
UFSTAT1	0x50004018				UART 1 FIFO Status
UMSTAT1	0x5000401C				UART 1 Modem Status
UTXH1	0x50004023	0x50004020	B	W	UART 1 Transmission Hold
URXH1	0x50004027	0x50004024		R	UART 1 Receive Buffer
UBRDIV1	0x50004028	←	W	R/W	UART 1 Baud Rate Divisor
ULCON2	0x50008000	←	W	R/W	UART 2 Line Control
UCON2	0x50008004				UART 2 Control
UFCON2	0x50008008				UART 2 FIFO Control
UTRSTAT2	0x50008010			R	UART 2 Tx/Rx Status
UERSTAT2	0x50008014				UART 2 Rx Error Status
UFSTAT2	0x50008018				UART 2 FIFO Status
UTXH2	0x50008023	0x50008020	B	W	UART 2 Transmission Hold
URXH2	0x50008027	0x50008024		R	UART 2 Receive Buffer
UBRDIV2	0x50008028	←	W	R/W	UART 2 Baud Rate Divisor

Serial.c 분석(2) – (TOPDIR)/cpu/arm920t/serial.c

- **serial(2)-ULCON**
  - 각 UART의 serial통신 규약 설정(전송bit,stop bit,parity, Normal/Irda) register

Register	Address	R/W	Description	Reset Value
ULCON0	0x50000000	R/W	UART channel 0 line control register	0x00
ULCON1	0x50004000	R/W	UART channel 1 line control register	0x00
ULCON2	0x50008000	R/W	UART channel 2 line control register	0x00

ULCONn	Bit	Description	Initial State
Reserved	[7]		0
Infra-Red Mode	[6]	Determine whether or not to use the Infra-Red mode. 0 = Normal mode operation 1 = Infra-Red Tx/Rx mode	0
Parity Mode	[5:3]	Specify the type of parity generation and checking during UART transmit and receive operation. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/checked as 1 111 = Parity forced/checked as 0	000
Number of Stop Bit	[2]	Specify how many stop bits are to be used for end-of-frame signal. 0 = One stop bit per frame 1 = Two stop bit per frame	0
Word Length	[1:0]	Indicate the number of data bits to be transmitted or received per frame. 00 = 5-bits 01 = 6-bits 10 = 7-bits 11 = 8-bits	00

Serial.c 분석(3) – (TOPDIR)/cpu/arm920t/serial.c

❖ serial(3)-UCON

➢ 각 UART의 baud rate의 입력 clock,tx/rx interrupt type,error,mode설정 register

Register	Address	R/W	Description	Reset Value
UCON0	0x50000004	R/W	UART channel 0 control register	0x00
UCON1	0x50004004	R/W	UART channel 1 control register	0x00
UCON2	0x50008004	R/W	UART channel 2 control register	0x00

UCONn	Bit	Description	Initial State
Clock Selection	[10]	Select PCLK or UCLK for the UART baud rate. 0=PCLK : UBRDIVn = (int)(PCLK / (bps x 16)) -1 1=UCLK(@GPH8) : UBRDIVn = (int)(UCLK / (bps x 16)) -1	0
Tx Interrupt Type	[9]	Interrupt request type. 0 = Pulse (Interrupt is requested as soon as the Tx buffer becomes empty in Non-FIFO mode or reaches Tx FIFO Trigger Level in FIFO mode.) 1 = Level (Interrupt is requested while Tx buffer is empty in Non-FIFO mode or reaches Tx FIFO Trigger Level in FIFO mode.)	0
Rx Interrupt Type	[8]	Interrupt request type. 0 = Pulse (Interrupt is requested the instant Rx buffer receives the data in Non-FIFO mode or reaches Rx FIFO Trigger Level in FIFO mode.) 1 = Level (Interrupt is requested while Rx buffer is receiving data in Non-FIFO mode or reaches Rx FIFO Trigger Level in FIFO mode.)	0
Rx Time Out Enable	[7]	Enable/Disable Rx time out interrupt when UART FIFO is enabled. The interrupt is a receive interrupt. 0 = Disable 1 = Enable	0
Rx Error Status Interrupt Enable	[6]	Enable the UART to generate an interrupt upon an exception, such as a break, frame error, parity error, or overrun error during a receive operation. 0 = Do not generate receive error status interrupt. 1 = Generate receive error status interrupt.	0
Loopback Mode	[5]	Setting loopback bit to 1 causes the UART to enter the loopback mode. This mode is provided for test purposes only. 0 = Normal operation 1 = Loopback mode	0
Send Break Signal	[4]	Setting this bit causes the UART to send a break during 1 frame time. This bit is automatically cleared after sending the break signal. 0 = Normal transmit 1 = Send break signal	0
Transmit Mode	[3:2]	Determine which function is currently able to write Tx data to the UART transmit buffer register. 00 = Disable 01 = Interrupt request or polling mode 10 = DMA0 request (Only for UART0), DMA3 request (Only for UART2) 11 = DMA1 request (Only for UART1)	00
Receive Mode	[1:0]	Determine which function is currently able to read data from UART receive buffer register. 00 = Disable 01 = Interrupt request or polling mode 10 = DMA0 request (Only for UART0), DMA3 request (Only for UART2) 11 = DMA1 request (Only for UART1)	00

Serial.c 분석(4) – (TOPDIR)/cpu/arm920t/serial.c

- **serial(4)-UFCON**
  - 각 UART의 FIFO관련 제어 register

Register	Address	R/W	Description	Reset Value
UFCON0	0x50000008	R/W	UART channel 0 FIFO control register	0x0
UFCON1	0x50004008	R/W	UART channel 1 FIFO control register	0x0
UFCON2	0x50008008	R/W	UART channel 2 FIFO control register	0x0

UFCONn	Bit	Description	Initial State
Tx FIFO Trigger Level	[7:6]	Determine the trigger level of transmit FIFO. 00 = Empty 01 = 4-byte 10 = 8-byte 11 = 12-byte	00
Rx FIFO Trigger Level	[5:4]	Determine the trigger level of receive FIFO. 00 = 4-byte 01 = 8-byte 10 = 12-byte 11 = 16-byte	00
Reserved	[3]		0
Tx FIFO Reset	[2]	Auto-cleared after resetting FIFO 0 = Normal 1= Tx FIFO reset	0
Rx FIFO Reset	[1]	Auto-cleared after resetting FIFO 0 = Normal 1= Rx FIFO reset	0
FIFO Enable	[0]	0 = Disable 1 = Enable	0

Serial.c 분석(5) – (TOPDIR)/cpu/arm920t/serial.c

- **serial(5)-UMCON**
  - UART0,1 의 AUTO FLOW CONTOL(RTS) REGISTER

Register	Address	R/W	Description	Reset Value
UMCON0	0x5000000C	R/W	UART channel 0 Modem control register	0x0
UMCON1	0x5000400C	R/W	UART channel 1 Modem control register	0x0
Reserved	0x5000800C	-	Reserved	Undef

UMCONn	Bit	Description	Initial State
Reserved	[7:5]	These bits must be 0's	00
Auto Flow Control (AFC)	[4]	0 = Disable 1 = Enable	0
Reserved	[3:1]	These bits must be 0's	00
Request to Send	[0]	If AFC bit is enabled, this value will be ignored. In this case the S3C2410X will control nRTS automatically. If AFC bit is disabled, nRTS must be controlled by software. 0 = 'H' level (Inactivate nRTS) 1 = 'L' level (Activate nRTS)	0

Serial.c 분석(6) – (TOPDIR)/cpu/arm920t/serial.c

- **serial(6)-UTRSTAT**
  - 각 UART의 TX/RX STATUS REGISTER

Register	Address	R/W	Description	Reset Value
UTRSTAT0	0x50000010	R	UART channel 0 Tx/Rx status register	0x6
UTRSTAT1	0x50004010	R	UART channel 1 Tx/Rx status register	0x6
UTRSTAT2	0x50008010	R	UART channel 2 Tx/Rx status register	0x6

UTRSTATn	Bit	Description	Initial State
Transmitter empty	[2]	Set to 1 automatically when the transmit buffer register has no valid data to transmit and the transmit shift register is empty. 0 = Not empty 1 = Transmitter (transmit buffer & shifter register) empty	1
Transmit buffer empty	[1]	Set to 1 automatically when transmit buffer register is empty. 0 =The buffer register is not empty 1 = Empty (In Non-FIFO mode, Interrupt or DMA is requested. In FIFO mode, Interrupt or DMA is requested, when Tx FIFO Trigger Level is set to 00 (Empty)) If the UART uses the FIFO, users should check Tx FIFO Count bits and Tx FIFO Full bit in the UFSTAT register instead of this bit.	1
Receive buffer data ready	[0]	Set to 1 automatically whenever receive buffer register contains valid data, received over the RXDn port. 0 = Empty 1 = The buffer register has a received data (In Non-FIFO mode, Interrupt or DMA is requested) If the UART uses the FIFO, users should check Rx FIFO Count bits and Rx FIFO Full bit in the UFSTAT register instead of this bit.	0

Serial.c 분석(7) – (TOPDIR)/cpu/arm920t/serial.c

- **serial(7)-UMSTAT**
  - UART0,1 의 AUTO FLOW CONTOL(CTS) REGISTER

Register	Address	R/W	Description	Reset Value
UMSTAT0	0x5000001C	R	UART channel 0 Modem status register	0x0
UMSTAT1	0x5000401C	R	UART channel 1 Modem status register	0x0
Reserved	0x5000801C	–	Reserved	Undef

UMSTAT0	Bit	Description	Initial State
Reserved	[3]		0
Delta CTS	[2]	Indicate that the nCTS input to the S3C2410X has changed state since the last time it was read by CPU. (Refer to Figure 11-8.) 0 = Has not changed 1 = Has changed	0
Reserved	[1]		0
Clear to Send	[0]	0 = CTS signal is not activated (nCTS pin is high.) 1 = CTS signal is activated (nCTS pin is low.)	0

Serial.c 분석(8) – (TOPDIR)/cpu/arm920t/serial.c

- **serial(8)-UFCON**
  - 각 UART의 TX/RX BUFFER REGISTER

Register	Address	R/W	Description	Reset Value
UTXH0	0x50000020(L) 0x50000023(B)	W (by byte)	UART channel 0 transmit buffer register	–
UTXH1	0x50004020(L) 0x50004023(B)	W (by byte)	UART channel 1 transmit buffer register	–
UTXH2	0x50008020(L) 0x50008023(B)	W (by byte)	UART channel 2 transmit buffer register	–

UTXHn	Bit	Description	Initial State
TXDATAn	[7:0]	Transmit data for UARTn	–

Register	Address	R/W	Description	Reset Value
URXH0	0x50000024(L) 0x50000027(B)	R (by byte)	UART channel 0 receive buffer register	–
URXH1	0x50004024(L) 0x50004027(B)	R (by byte)	UART channel 1 receive buffer register	–
URXH2	0x50008024(L) 0x50008027(B)	R (by byte)	UART channel 2 receive buffer register	–

URXHn	Bit	Description	Initial State
RXDATAn	[7:0]	Receive data for UARTn	–

Serial.c 분석(9) – (TOPDIR)/cpu/arm920t/serial.c

- **serial(9)-UBRDIV**
  - 각 UART의 BAUD RATE DIVISIOR REGISTER
  - UBRDIVn = (INT)(PCLK / (BPS X16)) –1
  - UBRDIVn = (INT)(UCLK / (BPS X16)) –1
  - EX)UCLK(40MHZ) => (40000000 / (115200 X 16)) – 1 = 21.7 – 1 = 21 – 1 = 20

Register	Address	R/W	Description	Reset Value
UBRDIV0	0x50000028	R/W	Baud rate divisior register 0	–
UBRDIV1	0x50004028	R/W	Baud rate divisior register 1	–
UBRDIV2	0x50008028	R/W	Baud rate divisior register 2	–

UBRDIVn	Bit	Description	Initial State
UBRDIV	[15:0]	Baud rate division value UBRDIVn >0	–

Serial.c 분석(10) – (TOPDIR)/cpu/arm920t/serial.c

- **S3c2410.h – (TOPDIR)/include/s3c2410.h**
  - S3c2410의 hardware관련 register들을 정리한 header 파일이다.

```
        ....
##define S3C24X0_UART_BASE                    0x50000000
        ....
static inline S3C24X0_UART * const S3C24X0_GetBase_UART(S3C24X0_UARTS_NR nr)
{
        return (S3C24X0_UART * const)(S3C24X0_UART_BASE + (nr * 0x4000));
}
```

Serial.c 분석(11) – (TOPDIR)/cpu/arm920t/serial.c

- **S3c24x0.h – (TOPDIR)/include/s3c24x0.h**
  - S3c2410,S3C2400공용으로 사용하는 hardware관련 구조체들을 정리한 header 파일이다.

```
        ....
typedef struct {
        S3C24X0_REG32       ULCON;
        S3C24X0_REG32       UCON;
        S3C24X0_REG32       UFCON;
        S3C24X0_REG32       UMCON;
        S3C24X0_REG32       UTRSTAT;
        S3C24X0_REG32       UERSTAT;
        S3C24X0_REG32       UFSTAT;
        S3C24X0_REG32       UMSTAT;
#ifdef __BIG_ENDIAN
        S3C24X0_REG8        res1[3];
        S3C24X0_REG8        UTXH;
        S3C24X0_REG8        res2[3];
        S3C24X0_REG8        URXH;
#else /* Little Endian */
        S3C24X0_REG8        UTXH;
        S3C24X0_REG8        res1[3];
        S3C24X0_REG8        URXH;
        S3C24X0_REG8        res2[3];
#endif
        S3C24X0_REG32       UBRDIV;
} /*__attribute__((__packed__))*/ S3C24X0_UART;
```

Serial.c 분석(12) – (TOPDIR)/cpu/arm920t/serial.c

❖ **Global_data.h – (TOPDIR)/include/asm-arm/global_data.h**

```
        ....
typedef struct      global_data {
        bd_t                    *bd;
        unsigned long           flags;
        unsigned long           baudrate;
        unsigned long           have_console;       /* serial_init() was called */
        unsigned long           reloc_off;  /* Relocation Offset */
        unsigned long           env_addr; /* Address  of Environment struct */
        unsigned long           env_valid; /* Checksum of Environment valid? */
        unsigned long           fb_base;   /* base address of frame buffer */
#ifdef CONFIG_VFD
        unsigned char           vfd_type;  /* display type */
#endif
        ...
} gd_t;
        ...
#define DECLARE_GLOBAL_DATA_PTR     register gd_t *gd asm ("r8")
```

Serial.c 분석(13) – (TOPDIR)/cpu/arm920t/serial.c

- **Serial.c - Serial_setbrg(),serial_init()**
  - Console용 UART을 초기화하는 함수이다.

```
void serial_setbrg (void){
        DECLARE_GLOBAL_DATA_PTR;                                  @ global struct의 point
        S3C24X0_UART * const uart = S3C24X0_GetBase_UART(UART_NR); @ console용 UART을 선택
        int i;
        unsigned int reg = 0;
@CONFIG_BAUDRATE=115200 <= (TOPDIR)/include/configs/smdk2410.h
@init_baudrate() <= (TOPDIR)/lib_arm/board.c
@UBRDIVn = (int)((PCLK)/(bps x 16)) –1 = (50000000/(115200 x 16)) –1 = 26
        reg = get_PCLK() / (16 * gd->baudrate) - 1;        /* FIFO enable, Tx/Rx FIFO clear */
        uart->UFCON = 0x07;                  @FIFO enable,Tx/Rx FIFO clear
        uart->UMCON = 0x0;                   @no auto flow contol
        uart->ULCON = 0x3;                   @Normal,no parity,1stop,8bit
@PCLK,tx=level,rx=edge,disable timeout int.,enable rx error int.,normal,interrupt or polling
        uart->UCON = 0x245;
        uart->UBRDIV = reg;                  @baud rate 설정,115200bps
        for (i = 0; i < 100; i++);           @delay
}
int serial_init (void){
        serial_setbrg ();
        return (0);
}
```

Serial.c 분석(14) – (TOPDIR)/cpu/arm920t/serial.c

- **Serial.c - Serial_gets(), Serial_putc(), Serial_tstc(), Serial_puts()**
  - Console에서 한문자를 읽거나 string문자를 출력하는 함수이다.

```
int serial_getc (void){
        S3C24X0_UART * const uart = S3C24X0_GetBase_UART(UART_NR);
        @ receive buffer register에 입력된 data가 올때까지 대기하다가 data있으면 rx buffer값을 return
        while (!(uart->UTRSTAT & 0x1));
        return uart->URXH & 0xff;
}
void serial_putc (const char c){
        S3C24X0_UART * const uart = S3C24X0_GetBase_UART(UART_NR);
        ...
        while (!(uart->UTRSTAT & 0x2));          @ tx buffer register가 비어있는지 검사한다.
        ...
        uart->UTXH = c;                          @ tx data을 출력한다.
        if (c == '\n')                           @ tx data가 new line이면 return명령을 출력한다.
                serial_putc ('\r');
}
int serial_tstc (void){@ receive buffer register에 입력된 data가 있는 지만  검사한다.
        S3C24X0_UART * const uart = S3C24X0_GetBase_UART(UART_NR);
        return uart->UTRSTAT & 0x1;
}
voidserial_puts (const char *s){
        @ tx data가 null이 올때까지 한문자씩 계속 출력한다.
        while (*s) {serial_putc (*s++);}
}
```

(TOPDIR)/board/smdk2410 directory

Memsetup.S

Smdk2410.c

Flash.c

smdk2410.c 분석(1) – (TOPDIR)/board/smdk2410/smdk2410.c c

❖ **Global_data.h – (TOPDIR)/include/asm-arm/global_data.h**

```
        ....
typedef struct      global_data {
        bd_t                    *bd;
        unsigned long           flags;
        unsigned long           baudrate;
        unsigned long           have_console;       /* serial_init() was called */
        unsigned long           reloc_off;  /* Relocation Offset */
        unsigned long           env_addr; /* Address  of Environment struct */
        unsigned long           env_valid; /* Checksum of Environment valid? */
        unsigned long           fb_base;    /* base address of frame buffer */
#ifdef CONFIG_VFD
        unsigned char           vfd_type;  /* display type */
#endif
        ...
} gd_t;
        ...
#define DECLARE_GLOBAL_DATA_PTR     register gd_t *gd asm ("r8")
```

smdk2410.c 분석(2) – (TOPDIR)/board/smdk2410/smdk2410.c c

❖ **U_boot.h – (TOPDIR)/include/asm-arm/u_boot.h**

```
#ifndef _U_BOOT_H_
#define _U_BOOT_H_          1

typedef struct bd_info {
   int                  bi_baudrate;              /* serial console baudrate */
   unsigned long        bi_ip_addr;               /* IP Address */
   unsigned char        bi_enetaddr[6];           /* Ethernet adress */
   struct environment_s              *bi_env;
   ulong          bi_arch_number;                 /* unique id for this board */
   ulong          bi_boot_params;                 /* where this board expects params */
   struct                                         /* RAM configuration */
   {
          ulong start;
          ulong size;
   } bi_dram[CONFIG_NR_DRAM_BANKS];
} bd_t;
#define bi_env_data bi_env->data
#define bi_env_crc  bi_env->crc
#endif      /* _U_BOOT_H_ */
```

smdk2410.c 분석(3) – (TOPDIR)/board/smdk2410/smdk2410.c

❖ **Smdk2410.c – board_init()**

```
#define FCLK_SPEED 1
#if FCLK_SPEED==0              /* Fout = 203MHz, Fin = 12MHz for Audio */
        ...
#elif FCLK_SPEED==1                        /* Fout = 202.8MHz */
#define M_MDIV      0xA1
#define M_PDIV      0x3
#define M_SDIV      0x1
#endif

#define USB_CLOCK 1
#if USB_CLOCK==0
        ...
#elif USB_CLOCK==1
#define U_M_MDIV  0x48
#define U_M_PDIV  0x3
#define U_M_SDIV  0x2
#endif
```

smdk2410.c 분석(4) – (TOPDIR)/board/smdk2410/smdk2410.c

❖ Smdk2410.c – board_init() 계속

```
int board_init (void){
        DECLARE_GLOBAL_DATA_PTR;
        S3C24X0_CLOCK_POWER * const clk_power = S3C24X0_GetBase_CLOCK_POWER();
        S3C24X0_GPIO * const gpio = S3C24X0_GetBase_GPIO();

        /* to reduce PLL lock time, adjust the LOCKTIME register */
        clk_power->LOCKTIME = 0xFFFFFF;

@ configure MPLL = (m * Fin)/(p * 2^S), m = (MDIV + 8), p = (PDIV + 2), s = SDIV, Fin = 입력 주파수
@ FCLK = ((0xa1 + 8) * 12Mhz)/((0x03+2) * 2^1) = 2028/10 = 202.8Mhz
        clk_power->MPLLCON = ((M_MDIV << 12) + (M_PDIV << 4) + M_SDIV);
        /* some delay between MPLL and UPLL */
        delay (4000);

@ UCLK = ((0x48 + 8) * 12Mhz)/((0x03+2) * 2^2) = 960/20 = 48Mhz
        clk_power->UPLLCON = ((U_M_MDIV << 12) + (U_M_PDIV << 4) + U_M_SDIV);

        /* some delay between MPLL and UPLL */
        delay (8000);
```

smdk2410.c 분석(5) – (TOPDIR)/board/smdk2410/smdk2410.c

❖ **Smdk2410.c – board_init() 계속**

```
        gpio->GPACON = 0x007FFFFF;   @ GPA22-0 : 모두 추가기능으로 설정(bus확장용)
        gpio->GPBCON = 0x00044555;  @ GPB10,8,6 : input,   GPB9,7,5,4,3,2,1,0: output
        gpio->GPBUP = 0x000007FF;    @ GPB10-0 : pull-up disable
        gpio->GPCCON = 0xAAAAAAAA; @ GPC15-0 : 모두 추가기능으로 설정(lcd제어용)
        gpio->GPCUP = 0x0000FFFF;    @ GPC15-0 : pull-up disable
        gpio->GPDCON = 0xAAAAAAAA; @ GPD15-0 : 모두 추가기능으로 설정(lcd제어용)
        gpio->GPDUP = 0x0000FFFF;    @ GPD15-0 : pull-up disable
        gpio->GPECON = 0xAAAAAAAA; @ GPE15-0 : 모두 추가기능으로 설정(특수 bus제어용)
        gpio->GPEUP = 0x0000FFFF;    @ GPE15-0 : pull-up disable
        gpio->GPFCON = 0x000055AA;  @ GPF3-0 : EINT3-0 , GPF7-4 = output
        gpio->GPFUP = 0x000000FF;    @ GPF7-0 : pull-up disable
@GPG15-12 :  nYPON,YMON,nXPON,xMON ,  GPG11 : EINT19,  GPG10,9,8 : output,
@GPG7: SPICLK1, GPG6:SPIMOSI1, GPG4:SPIMISO1, GPG3,1,0:EINT11,9,8,  GPG2:nSS0
        gpio->GPGCON = 0xFF95FFBA;
        gpio->GPGUP = 0x0000FFFF;   @ GPG15-0 = pull-up disable
@ GPH10-8:output,  GPH7:nCTS1, GPH6:nRTS1, GPH5:RXD1, GPH4:TXD1, GPH3:RXD0,GPH2:TXD0
@ GPH1:nRTS0, GPH0:nCTS0
        gpio->GPHCON = 0x002AFAAA;
        gpio->GPHUP = 0x000007FF;   @ GPH10-0 = pull-up disable
```

smdk2410.c 분석(6) – (TOPDIR)/board/smdk2410/smdk2410.c

❖ **Smdk2410.c – board_init() 계속**

```
    /* arch number of SMDK2410-Board */
    gd->bd->bi_arch_number = 193;          @ kernel에서 사용할 architecture number

    /* adress of boot parameters */
    gd->bd->bi_boot_params = 0x30000100;  @ kernel의 boot parameter의 start address

    icache_enable();                       @ icache enable

    dcache_enable();                       @ dcache enable

    return 0;
}
```

smdk2410.c 분석(7) – (TOPDIR)/board/smdk2410/smdk2410.c

❖ **Smdk2410.c – dram_init()**

```
@(TOPDIR)/include/configs/smdk2410.h
@#define CONFIG_NR_DRAM_BANKS      1          /* we have 1 bank of DRAM */
@#define PHYS_SDRAM_1              0x30000000 /* SDRAM Bank #1 */
@#define PHYS_SDRAM_1_SIZE         0x04000000 /* 64 MB */


int dram_init (void)
{
        DECLARE_GLOBAL_DATA_PTR;

        gd->bd->bi_dram[0].start = PHYS_SDRAM_1;      @sdram start address등록
        gd->bd->bi_dram[0].size = PHYS_SDRAM_1_SIZE;  @sdram size등록

        return 0;
}
```

FLASH Memory 제어(1)

- Flash 메모리의 Architecture는 크게 비트 선과 접지선 사이에 셀이 병렬로 배치된 NOR형 구조와 직렬로 배치된 NAND형 구조로 나눌 수 있고, 다시 NOR형은 그 변형 구조인 AND형, DINOR형, VGA(Virtual Ground Array)형으로 나눌 수 있다. NOR형은 read와 program 동작을 위한 address decoding을 RAM의 것과 유사하게 구성하여 주변회로가 간단해지고 read access time이 작아지는 장점이 있으나 각 셀마다 비트선의 접촉전극이 필요하므로 NAND형에 비하여 셀 면적이 커지는 단점이 있다. NAND 형은 읽기 동작에 앞서 먼저 해당 block을 선택해야만 하고, 각 셀이 직렬로 연결되어 동작 저항이 크기 때문에 읽기 속도가 상대적으로 느리다는 단점이 있다. 한편 개량 NOR형은 기존의 NOR형과 NAND형의 장점을 취한 것으로 복수 개의 셀 트랜지스터를 공통 소스 선과 공통 비트 선 사이에 병렬로 구성하여 비트선의 접촉전극을 생략한 구조이다.

- Flash 메모리의 개발과 생산은 세계 주요 반도체 업체간의 공동개발 또는 위탁생산이라는 전략적 제휴를 통해 이루어지고 있다. NOR 구조의 제품에서는 미국의 인텔과 일본의 샤프, 그리고 미국 AMD와 일본 후지쯔의 제휴가 대표적이며, NAND형 제품에서는 일본의 도시바와 한국의 삼성전자가 제휴하였다.

- 고속 random access read 동작이 중요한 단일 칩형 제품에 있어서는 NOR형 구조를 채택한 인텔-샤프 진영과 AMD-후지쯔 진영이 전체 시장의 80% 이상을 장악하고 있다. NAND형은 그 구조 특성상 읽기 동작 시에 block 선택시간이 필요하고, random access 동작측면에서 크게 불리하기 때문에 단일 칩형 품목 대신 card 형 제품에 주력하고 있는 실정이다.

- 부트로더에서는 NOR형 FLASH을 사용하므로 NOR FLASH을 주로 설명하겠다.

FLASH Memory 제어(2)

- NOR형 FLASH는 일반 RAM과같이 읽을 수 있으며 NAND형에 비해 만들기가 어렵기 때문에 비싸다.
- 8메가 바이트이상의 FLASH를 사용한다면 대개는 인텔 제품을 사용하다. 대만이나 기타의 제조사는 4메가 바이트 정도만을 생산하고 있기 때문이다. (그러나 지금은 8 메가를 생산하는 제조사도 있다.)
- 플래쉬를 읽는것은 메모리와 같지만 쓸 경우에는 다음과 같은 흐름으로 쓰게된다.
  - 영역 지우기(sector or block erase)
  - 쓰기(write)
  - 데이터 고정(fusing or programming)
- 플래쉬는 erase을 하면 모든 데이터는 0xffff(16bit)로 변경된다.
  - 플래쉬의 데이터는 1은 0으로 바뀌지만 0은 결코 1이 될 수 없다.
  - Ex1) 0xff에 0xa0을 쓰면 0xa0가 된다.
  - Ex2) 0x00에 0xa0을 쓰면 0x00이 된다.
  - 그러한 이유로 플래쉬에 새로운 데이터를 쓰려면 항상 해당하는 sector을 erase하고 write해야 한다.
  - 그러나 0x0f라고 쓰인 곳에 0x03을 쓰면 0x03이 된다. 이런 특성을 이용하여 코딩을 하면 sector erase을 하지않고도 데이터를 변경 할 수 있어 좀더 빠른 수행을 할 수 있다.
- 그밖에 인텔 플래쉬 중에는 스트라타 플래쉬라는 것이 있는데 보통의 플래쉬와 달리 쓰기시에 한 개의 워드만 쓸수 있는 것이 아니라 한번에 16개의 워드를 쓸 수 있어 퓨징 시간을 그 만큼 줄일 수 있다.
- 대부분의 플래쉬는 부트블럭 플래쉬로 부트블록이 일반 블록 보다 세분화되어 부트영역의 메모리 손실을 줄일 수 있으며, type에는 bottom,top형태 두 가지를 제공한다.
- CHIP erase시간은 대략 1초 – 5초이며 word prgramming time은 200us – 600us가 소요된다.

FLASH Memory 제어(3)

- **Write Operation status**
  - DQ7
    - ✓ Erase Algorithm : Erase Algorithm 동안에는 DQ7은 0으로 있다가 Erase Algorithm 완료되면 1로 변한다.
    - ✓ Program Algorithm : 내부적인 write operation이 완료 되면 이어서 나오는 read cycle에서 write 되어진 데이터와 동일한 DQ7(bit7)이 제공된다.
      (나머지 status bit들이다. DQ6,DQ5,DQ2등..)
  - DQ6
    - ✓ Erase or Program operation 중에 read cycle을 수행하면 이 비트는 계속적으로 toggle할 것이다. 만약 Erase or Program operation이 내부적으로 완료되면 이 비트는 toggling을 멈춘다.
  - DQ5
    - ✓ Erase or Program operation에서 규정된 internal pulse count limit을 초과 할 경우 0에서 1 로 변한다.
  - DQ2
    - ✓ Erase중이거나 Erase Suspend operation mode중 read cycle을 수행하면 이 비트는 계속적으로 toggle할 것이다. 만약 Erase or Erase Suspend operation이 내부적으로 완료되면 이 비트는 toggling을 멈춘다.(Program operation에서는 toggling을 하지 않는다.)

FLASH Memory 제어(4)

- Write Operation status은 Erase 또는 Progra operation에서 다음 operation을 원활하게 수행시키기 위해 제공되는 status bit이다.
- Flash 마다 지원되는 bit가 존재할 수도 존재하지 않을 수도 있다. 현재 사용되는 flash중에 SST사 39VF160은 DQ7,DQ6만이 지원된다.
- Programmer마다 status bit중 DQ7만 사용하거나,아니면 DQ6만 사용하거나,또는 DQ7,DQ6,DQ5,DQ2을 전부 사용하여 programming 할 수있다.
- Toggle bit을 사용할 경우는 최소한 두번 이상 read 하여 toggle bit의 변화가 없는지 확인해야 한다.

- FLASH분석에 들어가기전에 관련된 파일명을 알아보자.
  - (TOPDIR)/include/flash.h
  - (TOPDIR)/board/smdk2410/flash.c
  - (TOPDIR)/common/flash.c

FLASH.h 분석(1) – (TOPDIR)/include/flash.h

❖ **Flash header file (각 bank의 flash정보를 가지고 있는 구조체)**

```
typedef struct {
        ulong       size;                       /* total bank size in bytes          */
        ushort      sector_count;               /* number of erase units             */
        ulong       flash_id;                   /* combined device & manufacturer code  */
        ulong       start[CFG_MAX_FLASH_SECT];  /* physical sector start addresses */
        uchar       protect[CFG_MAX_FLASH_SECT]; /* sector protection status     */
#ifdef CFG_FLASH_CFI
        uchar       portwidth;                  /* he width of the port                 */
        uchar       chipwidth;                  /* the width of the chip                */
        ushort      buffer_size;                /* # of bytes in write buffer           */
        ulong       erase_blk_tout;             /* maximum block erase timeout          */
        ulong       write_tout;                 /* maximum write timeout                */
        ulong       buffer_write_tout;          /* maximum buffer write timeout         */
#endif
} flash_info_t;
```

FLASH.h 분석(2) – (TOPDIR)/include/flash.h

❖ **Flash header file (flash에 관한 error 및 protect flags)**

```
#define ERR_OK                  0
#define ERR_TIMOUT              1
#define ERR_NOT_ERASED          2
#define ERR_PROTECTED           4
#define ERR_INVAL               8
#define ERR_ALIGN               16
#define ERR_UNKNOWN_FLASH_VENDOR 32
#define ERR_UNKNOWN_FLASH_TYPE  64
#define ERR_PROG_ERROR          128

/*-----------------------------------------------------------------------
 * Protection Flags for flash_protect():
 */
#define FLAG_PROTECT_SET    0x01
#define FLAG_PROTECT_CLEAR  0x02
```

FLASH.h 분석(3) – (TOPDIR)/include/flash.h

❖ **Flash header file 계속(제조사 ID & Device ID)**

```
#define AMD_MANUFACT        0x00010001      /* AMD      manuf. ID in D23..D16, D7..D0 */
#define FUJ_MANUFACT        0x00040004      /* FUJITSU manuf. ID in D23..D16, D7..D0 */
#define ATM_MANUFACT        0x001F001F      /* ATMEL */
#define STM_MANUFACT        0x00200020      /* STM (Thomson) manuf. ID in D23.. -"- */
#define SST_MANUFACT        0x00BF00BF      /* SST      manuf. ID in D23..D16, D7..D0 */
#define MT_MANUFACT         0x00890089      /* MT       manuf. ID in D23..D16, D7..D0 */
#define INTEL_MANUFACT      0x00890089      /* INTEL   manuf. ID in D23..D16, D7..D0 */
#define INTEL_ALT_MANU      0x00B000B0      /* alternate INTEL namufacturer ID       */
#define MX_MANUFACT         0x00C200C2      /* MXIC     manuf. ID in D23..D16, D7..D0 */
#define TOSH_MANUFACT       0x00980098      /* TOSHIBA manuf. ID in D23..D16, D7..D0 */
        중략.....
#define AMD_ID_LV800T       0x22DA22DA      /* 29LV800T ID ( 8 M, top boot sector)   */
#define AMD_ID_LV800B       0x225B225B      /* 29LV800B ID ( 8 M, bottom boot sect) */
        중략.....
#define FUJI_ID_29F800BA    0x22582258      /* MBM29F800BA ID  (8M) */
#define FUJI_ID_29F800TA    0x22D622D6      /* MBM29F800TA ID  (8M) */
        중략.....
#define SST_ID_xF800A       0x27812781      /* 39xF800A ID ( 8M = 512K x 16 )        */
#define SST_ID_xF160A       0x27822782      /* 39xF800A ID (16M =          1M x 16 ) */
```

FLASH.c 분석(1) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c - define**

```
@ 각 bank당 flash정보를 위한 구조체 선언
flash_info_t            flash_info[CFG_MAX_FLASH_BANKS];

@ Functions prototype 선언
/*-----------------------------------------------------------------------
 * Functions
 */
static ulong flash_get_size (vu_long *addr, flash_info_t *info);
static int write_word (flash_info_t *info, ulong dest, ulong data);
static void flash_get_offsets (ulong base, flash_info_t *info);


@ 16bit flash을 위한 command address 및 data bus width 선언
#define ADDR0          0x5555
#define ADDR1          0x2aaa
#define FLASH_WORD_SIZE unsigned short
```

FLASH.c 분석(2) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_init()**

```
unsigned long flash_init (void)
{
        unsigned long size;
        int i;
        uint pbcr;
        @ 각 bank의 flash id을 초기화한다.
        for (i=0; i<CFG_MAX_FLASH_BANKS; ++i) {
                flash_info[i].flash_id = FLASH_UNKNOWN;
        }

@ 제조사 및 device id을 flash로 부터 read하여 flash의 size,sector수을 얻어온다.
@ FLASH_BASE0_PRELIM : bank0 flash address =>/(TOPDIR)/include/configs/smdk2410.h
        size =  flash_get_size((vu_long *)FLASH_BASE0_PRELIM, &flash_info[0]);

@ flash로 부터 id을 read하여 정의되어 있지 않는 flash이면 "알수 없는 flash"라는 메시지를 console로 출력
        if (flash_info[0].flash_id == FLASH_UNKNOWN) {
                printf ("## Unknown FLASH on Bank 0 - Size = 0x%08lx = %ld MB\n",
                        size_b0, size_b0<<20);
        }
```

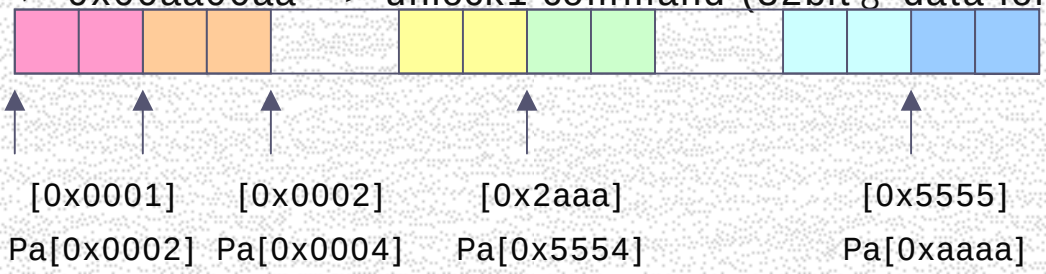
FLASH.c 분석(3) – (TOPDIR)/board/smdk2410/flash.c

❖ Flash.c – flash_init() 계속

```
        if (CFG_MAX_FLASH_BANKS == 1)        @ 단지 하나의 bank만 flash을 지원한다.
        {
@ maker와 device id을 가지고 각  sector의 start address을 flash_info에 저장한다.
            flash_get_offsets (FLASH_BASE0_PRELIM, &flash_info[0]);
@ monitor_flash_len = _armboot_end_data - _armboot_start;=>(TOPDIR)/lib-arm/board.c
@ monitor(arm_boot) 영역을 protection을 건다.(read-only)
            flash_protect(FLAG_PROTECT_SET,
                                    FLASH_BASE0_PRELIM,
                                    FLASH_BASE0_PRELIM+monitor_flash_len-1,
                                    &flash_info[0]);
@ 환경 변수 영역을 protection을 건다.(read-only)
            flash_protect(FLAG_PROTECT_SET,
                                    CFG_ENV_ADDR,
                                    CFG_ENV_ADDR + CFG_ENV_SIZE - 1,
                                    &flash_info[0]);
            flash_info[0].size = size;
        }
        else{       panic( "## ERROR – only bank0 support!\n"); }
        return(size);
}
```

FLASH.c 분석(4) – (TOPDIR)/board/smdk2410/flash.c

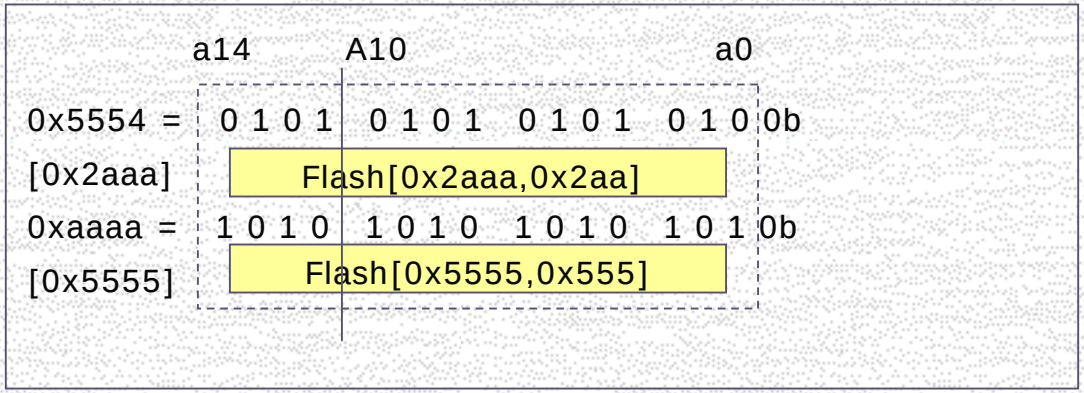
- **Flash.c – flash_get_size()**
  - Flash의 제조사 ID와 device ID을 자동으로 검출하여 선택된 flash의 sector수와 size을 얻는다.
  - volatile FLASH_WORD_SIZE *addr2 = (FLASH_WORD_SIZE *)addr;
    - addr 은 flash의 base address이다.
    - addr2은 base address를 volatile type으로 선언한다.
    - Volatile :
      - 컴파일러가 컴파일하고 있는 코드의 상황과는 관계 없이 바뀔 수 있는 메모리 변수가 있다면,해당 변수에 대해 특별한 최적화를 하지 못하도록 컴파일러를 제약하는 키워드이다.
      - 또는 주소의 메모리를 캐쉬를 거치지 않고 직접 접근할 수 있습니다.
  - addr2[ADDR0] = (FLASH_WORD_SIZE)0x00AA00AA;
    - Flash의 base address이다.
    - ADDR0 => word align 된 offset address
    - 0x00aa00aa => unlock1 command (32bit용 data format)



FLASH.c 분석(5) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_get_size() 계속**

- Amd계열의 flash는 command cycle일 때 a18 – a11은 don't care이다.
- 따라서, 0x2aaa는 0x2aa가 되고,0x5555는 0x555가 된다.
- 그러므로, SST flash와 amd flash의 command address을 공용으로 사용해도 문제는 없다.
- Address을 배열로 하지않고 직접 access할 경우는 아래와 같이한다.
  - (*(volatile u16 *)(CFG_FLASH_BASE + (0x00000555 << 1))) => amd
  - (*(volatile u16 *)(CFG_FLASH_BASE + (0x000002AA << 1))) => amd
  - (*(volatile u16 *)(CFG_FLASH_BASE + (0x00005555 << 1))) => sst
  - (*(volatile u16 *)(CFG_FLASH_BASE + (0x00002AAA << 1))) => sst



FLASH.c 분석(6) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_get_size() 계속**

```
static ulong flash_get_size (vu_long *addr, flash_info_t *info){
        short i;
        FLASH_WORD_SIZE value;
        ulong base = (ulong)addr;
    volatile FLASH_WORD_SIZE *addr2 = (FLASH_WORD_SIZE *)addr; @ flash base address
        addr2[ADDR0] = (FLASH_WORD_SIZE)0x00AA00AA;@ unlock1 command data
        addr2[ADDR1] = (FLASH_WORD_SIZE)0x00550055; @ unlock2 command data
        addr2[ADDR0] = (FLASH_WORD_SIZE)0x00900090; @ auto select command data
        value = addr2[0];                          @ flash로 부터 제조사 ID read 한다.
        switch (value) {
        case (FLASH_WORD_SIZE)AMD_MANUFACT:        @제조사 검색=>AMD, FUJITSU,SST
                info->flash_id = FLASH_MAN_AMD;    @flash info에 제조사 등록
                break;
                중략...
        default:                                   @제조사 추가 시 default 위부분에 등록
                info->flash_id = FLASH_UNKNOWN;
                info->sector_count = 0;
                info->size = 0;
                return (0);                        @등록되어 있지 않거나 알수 없는 제조사 id
        }
```

FLASH.c 분석(7) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_get_size() 계속**

```
value = addr2[1];                          @ flash로 부터 device ID read 한다.
switch (value) {
        중략...
case (FLASH_WORD_SIZE)AMD_ID_LV400T:@ Device 검색한다.
        info->flash_id += FLASH_AM400T;@flash info에 device 등록
        info->sector_count = 11;       @flash info에 sector 의 최대 갯수을 등록
        info->size = 0x00080000;       @512Kb,flash info에 size을 등록
        break;
        중략...
case (FLASH_WORD_SIZE)SST_ID_xF800A:
        info->flash_id += FLASH_SST800A;
        info->sector_count = 16;
        info->size = 0x00100000;       @1Mb
        break;
        중략...
default:                                   @device 추가 시 가 maker 밑부분에 추가 등록
        info->flash_id = FLASH_UNKNOWN;
        return (0);                        @등록되어 있지 않거나 알수 없는 device id
}
```

FLASH.c 분석(8) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_get_size() 계속**

```
@sector의 start address을 flash info에 등록한다. 뒤에서 설명
                        중략 ...
@flash info에 등록되어 있는 각sector  protection bit을 검사하여 flash info에 등록 한다.
            for (i = 0; i < info->sector_count; i++) {
@amd일 경우 각sector address+2가 가리키는 데이터의 bit0가 protection 검사 비트이다.
@ bit0 =1이면 protection이 되어 있다. SST는 별도의 sector protection을 제공하지 않는다.
                        addr2 = (volatile FLASH_WORD_SIZE *)(info->start[i]);
            if ((info->flash_id & FLASH_VENDMASK) == FLASH_MAN_SST)
             info->protect[i] = 0;
            else
             info->protect[i] = addr2[2] & 1;
             }
             /* Prevent writes to uninitialized FLASH.
             if (info->flash_id != FLASH_UNKNOWN) {
@ command cycle이 끝나면 반드시 reset command 주어서 read cycle로 전환 해야 한다.
                        addr2 = (FLASH_WORD_SIZE *)info->start[0];
                        *addr2 = (FLASH_WORD_SIZE)0x00F000F0;          /* reset bank */
             }
             return (info->size);
}
```

FLASH.c 분석(9) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_get_offset()**

➢ Flash의 type에 맞추어 각 sector의 start address을 얻는다.

```
static void flash_get_offsets (ulong base, flash_info_t *info)
{           int i;
 @sector의 start address table을 flash info에 등록한다.
            if (((info->flash_id & FLASH_VENDMASK) == FLASH_MAN_SST) ||
              (info->flash_id  == FLASH_AM040)){
              for (i = 0; i < info->sector_count; i++)
@SST는 별도의 boot sector는 지원하지 않으며 그대신 amd보다 작은 단위의 sector을 가지고 있다.
@sector size(4Kb -> 512sectors), block(64Kb -> 31blocks)
                        info->start[i] = base + (i * 0x00010000);          @block size 64Kb
      } else {
              if (info->flash_id & FLASH_BTYPE) {
@bottom boot sector type일 경우 base address을 기준으로 sector의 size을 더하여 start address을 얻는다.
            info->start[0] = base + 0x00000000;                  @16Kb(0x4000)
            info->start[1] = base + 0x00004000;                  @8Kb(0x2000)
            info->start[2] = base + 0x00006000;                  @8Kb(0x2000)
            info->start[3] = base + 0x00008000;                  @32Kb(0x8000) => 64K(0x10000)
            for (i = 4; i < info->sector_count; i++) {           @sector수 만큼 loop수행
                        info->start[i] = base + (i * 0x00010000) - 0x00030000;}
```

FLASH.c 분석(10) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_get_offset() 계속**

```
@ top boot sector type일 경우는 flash의 base address에 size을 더하여 max. address부터 sector size을
@ 빼면서 boot sector을 구한 후 동일 size구간에서 loop를 돌려 sector의 start address 얻는다.
@ base => flash base address
@ info->size => flash size

} else {
                        /* set sector offsets for top boot block type                    */
                        i = info->sector_count - 1;
                        info->start[i--] = base + info->size - 0x00004000;
                        info->start[i--] = base + info->size - 0x00006000;
                        info->start[i--] = base + info->size - 0x00008000;
                        for (; i >= 0; i--) {
                                info->start[i] = base + i * 0x00010000;
                        }
                }
        }
}
```

FLASH.c 분석(11) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_print_info()**

➢ Flash정보를 console에 display한다.

```
void flash_print_info  (flash_info_t *info)
{
        중략 ...
        if (info->flash_id == FLASH_UNKNOWN) {
                printf ("missing or unknown FLASH type\n");
                return;      @ 알 수 없는 flash id일 경우 error를 console에 display하고 빠져 나온다.
        }
        switch (info->flash_id & FLASH_VENDMASK) {
        case FLASH_MAN_AMD:        printf ("AMD ");                  break;
        중략 ...      @ flash의 vendor을 console에 display한다. 추가 flash등록 시 이곳에 코드 삽입
        default:                printf ("Unknown Vendor ");    break;
        }
        switch (info->flash_id & FLASH_TYPEMASK) {
        case FLASH_AM040:  printf ("AM29F040 (512 Kbit, uniform sector size)\n");
                                        break;
        중략 ...      @ flash의 part name을 console에 display한다. 추가 flash등록 시 이곳에 코드 삽입
        default:                printf ("Unknown Chip Type\n");
                                        break;
```

FLASH.c 분석(12) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_print_info() 계속**

```
@ flash의 size (KB단위)와 sector의 총 개수를 console에 display한다.
        printf ("  Size: %ld KB in %d Sectors\n",info->size >> 10, info->sector_count);
        printf ("  Sector Start Addresses:");
        for (i=0; i<info->sector_count; ++i) {
                if (i != (info->sector_count-1))
                size = info->start[i+1] - info->start[i];      @ 맨 마지막 sector size구하기
        else
                size = info->start[0] + info->size - info->start[i]; @ sector size구하기
        erased = 1;
        flash = (volatile unsigned long *)info->start[i]; @ ??? Unsigned short => test 필요
        size = size >> 2;              @ long word access 을 하기위해서 4로 나눈다.???? 2로 나누어야 한다.
        for (k=0; k<size; k++){   @ sector size만큼 flash data값을 read하여 data가 지워졌는지 확인한다.
          if (*flash++ != 0xffffffff){ erased = 0; break; }
        }                              @ data가 지워져 있으면 erased변수를 1로 setting 한다.
        if ((i % 5) == 0)  printf ("\n   "); @ sector을 한 라인에 5개씩 display시킨다.
        printf (" %08lX%s%s",info->start[i],erased ? " E" : "  ",info->protect[i] ? "RO " : "   ");
        }                              @ sector의 start address,Erase 상태,protect 상태를 display한다.
        printf ("\n");
        return;
} @ 초기 코드가 아무래도 32bit flasg 에서 시작된 것 같다. 데이터들이 모두 32bit width로 처리 되어 있다.
```

FLASH.c 분석(13) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_erase()**

➢ Flash의 sector을 erase한다. Error = 1, Ok =0 으로 return

```
int         flash_erase (flash_info_t *info, int s_first, int s_last)
{           중략 ...
            if ((s_first < 0) || (s_first > s_last)) {
                        if (info->flash_id == FLASH_UNKNOWN) { printf ("- missing\n"); }
                        else { printf ("- no sectors to erase\n"); }
                        return 1;
            } @ start sector 가 0이하 이거나 start sector가 end sector보다 클 때 error처리 한다.
            if (info->flash_id == FLASH_UNKNOWN) {
                        printf ("Can't erase unknown flash type - aborted\n");
                        return 1; @ 알 수 없는 flash id일 경우 error처리 한다.
            }
            prot = 0;
            for (sect=s_first; sect<=s_last; ++sect) {
                        if (info->protect[sect]) { prot++; }
            }
            if (prot) { printf ("- Warning: %d protected sectors will not be erased!\n",prot); }
            else { printf ("\n");} @ erase하려는 sector가 protect 걸려 있으면 warning 처리한다.
            l_sect = -1;
```

FLASH.c 분석(14) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_erase() 계속**

```
@ erase시 time out에 문제가 발생할 수 있으므로 인터럽트를 disable한다.
flag = disable_interrupts();
@ protect가 걸려있지 않는 sector을 erase 한다.
for (sect = s_first; sect<=s_last; sect++) {
if (info->protect[sect] == 0) {   @ protect가 걸려있지 않는 sector을 check한다.
   addr2 = (FLASH_WORD_SIZE *)(info->start[sect]);
 @ erase할 sector의 start address을 console에 display한다.
   printf("Erasing sector %p\n", addr2);
 @ SST flash일 경우 sector erase command가 아니라 block erase command를 사용한다.
   if ((info->flash_id & FLASH_VENDMASK) == FLASH_MAN_SST) {
            addr[ADDR0] = (FLASH_WORD_SIZE)0x00AA00AA;   @ unlock1 command
            addr[ADDR1] = (FLASH_WORD_SIZE)0x00550055;   @ unlock2 command
            addr[ADDR0] = (FLASH_WORD_SIZE)0x00800080;   @ erase setup comman
            addr[ADDR0] = (FLASH_WORD_SIZE)0x00AA00AA;   @ unlock1 command
            addr[ADDR1] = (FLASH_WORD_SIZE)0x00550055;   @ unlock2 command
            addr2[0] = (FLASH_WORD_SIZE)0x00500050;      @ block erase command
            for (i=0; i<50; i++)   @ SST을 위한 erase을 위해 50ms동안 기다린다.
                                udelay(1000);
              }
             @ addr 은 flash의 base address이다. addr2은 sector의 start address이다.
```

FLASH.c 분석(15) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_erase() 계속**

```
        else {
                addr[ADDR0] = (FLASH_WORD_SIZE)0x00AA00AA; @ unlock1 command
                addr[ADDR1] = (FLASH_WORD_SIZE)0x00550055;  @ unlock2 command
                addr[ADDR0] = (FLASH_WORD_SIZE)0x00800080;  @ erase setup command
                addr[ADDR0] = (FLASH_WORD_SIZE)0x00AA00AA; @ unlock1 command
                addr[ADDR1] = (FLASH_WORD_SIZE)0x00550055; @ unlock2 command
                addr2[0] = (FLASH_WORD_SIZE)0x00300030;       @ sector erase command
                 }
          l_sect = sect;
          wait_for_DQ7(info, sect);                                  @ erase polling check
                }
        }
        if (flag) @ interrupt을 disable하기 전에 enable이 되어 있었는지 확인한다.
                enable_interrupts();                          @ interrupt을 enable한다.
        udelay (1000);                                        @ 1ms 동안 기다린다.
        addr = (FLASH_WORD_SIZE *)info->start[0];             @ flash의 base address
        addr[0] = (FLASH_WORD_SIZE)0x00F000F0;@ reset command을 실행하여 read cycle로 전환
        printf (" done\n");
        return 0;
}
```

FLASH.c 분석(16) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – wait_for_DQ7()**

➢ Time out동안 flash에서 data을 read해서 DQ7bit 가 1로 set 될때 가지 loop를 수행한다.

```
int wait_for_DQ7(flash_info_t *info, int sect)
{
ulong start, now, last;
volatile FLASH_WORD_SIZE *addr = (FLASH_WORD_SIZE *)(info->start[sect]);
start = get_timer (0);                          @ timer에서 time tick 값을 얻어온다.
last  = start;
while ((addr[0] & (FLASH_WORD_SIZE)0x00800080) != (FLASH_WORD_SIZE)0x00800080) {
        if ((now = get_timer(start)) > CFG_FLASH_ERASE_TOUT) {
        printf ("Timeout\n");                   @ smdk2410.h파일에 정의된 time out이 되면
        return -1;                              @ error(–1)로 return한다.
                }
                if ((now - last) > 1000) {      @ 1초마다 '.' console에 display한다.
                putc ('.');
        last = now;
        }
  }
        return 0;                               @ Time out전에 DQ7 bit가 1이 되면 OK(0)으로 return한다.
}
```

FLASH.c 분석(17) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – write_word()**

➢ Flash에 long word단위로 write을 한다.

```
/* return 값
 * 0 - OK
 * 1 - write timeout
 * 2 - Flash not erased */

static int write_word (flash_info_t * info, ulong dest, ulong data)
{
        volatile FLASH_WORD_SIZE *addr2 = (FLASH_WORD_SIZE *) (info->start[0]);
        volatile FLASH_WORD_SIZE *dest2 = (FLASH_WORD_SIZE *) dest;
        volatile FLASH_WORD_SIZE *data2 = (FLASH_WORD_SIZE *) & data;
        ulong start;
        int i;
@ flash에 쓰려는 address의 데이터가 erase 되어 있는지 검사를 한다.
@ 만약 erase되어 있다면 0xffff로 되어 있어 data와 masking해도 그 결과 값이 data의 내용과 같아야 한다.
        if ((*((volatile FLASH_WORD_SIZE *) dest) &
           (FLASH_WORD_SIZE) data) != (FLASH_WORD_SIZE) data) {
                return (2);          @ flash의 write하려는 address의 data가 erase 안되어 있다.
        }
```

FLASH.c 분석(18) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – write_word() 계속**

```
        @long word type의 data을 word단위로 flash에 write한다.
        for (i = 0; i < 4 / sizeof (FLASH_WORD_SIZE); i++) {
        int flag;
        @ program write시 time out에 문제가 발생할 수 있으므로 인터럽트를 disable한다.
        flag = disable_interrupts ();
        addr2[ADDR0] = (FLASH_WORD_SIZE) 0x00AA00AA;          @ unlock1 command
        addr2[ADDR1] = (FLASH_WORD_SIZE) 0x00550055;          @ unlock2 command
        addr2[ADDR0] = (FLASH_WORD_SIZE) 0x00A000A0;          @ program command
        dest2[i] = data2[i];  @ memory의 data을 word단위로 destination address에 저장 한다.
        if (flag)                @ interrupt을 disable하기 전에 enable이 되어 있었는지 확인한다.
                enable_interrupts ();
        @ flash의 destination address을 read해서 write한 데이터와 d7 bit의 값이 같은지 확인한다.
        start = get_timer (0);
        while ((dest2[i] & (FLASH_WORD_SIZE) 0x00800080) !=
             (data2[i] & (FLASH_WORD_SIZE) 0x00800080)) {
                if (get_timer (start) > CFG_FLASH_WRITE_TOUT) { return (1); } @ Time out
        }
}
        return (0); @ 정상으로 write되었다.
}
```

FLASH.c 분석(19) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – write_buff()**

➢ Flash에 특정주소에 cnt수 만큼 memory의 데이터를 저장하는 함수이다.

```
int write_buff (flash_info_t *info, uchar *src, ulong addr, ulong cnt)
{       ulong cp, wp, data;
        int i, l, rc;                   @ test 필요
        wp = (addr & ~3);               @ addr & 0xfffffffc ,flash의 address를 2word단위로 정렬한다
        if ((l = addr - wp) != 0) {     @ 2word단위로 정렬 되지 않고 시작되는 byte들을 처리한다.
        data = 0;   @ flash의 start address가 0xXXXXXXX2라고 하면 1,0은 flash로 부터 데이터 읽는다.
        for (i=0, cp=wp; i<l; ++i, ++cp) {
                data = (data << 8) | (*(uchar *)cp);
        } @ address 2부터 3까지 memory의 데이터를 저장하고, 저장한 만큼 cnt값을 감소시킨다.
        for (; i<4 && cnt>0; ++i) {
                data = (data << 8) | *src++;  --cnt;  ++cp;
        } @ flash의 start address가 1이고 cnt값이 2이면,마지막 byte는 flash로 부터 데이터를 읽는다.
        for (; cnt==0 && i<4; ++i, ++cp) {
                data = (data << 8) | (*(uchar *)cp);
        } @ flash에 2word단위로 write한다. 0가 아니면 error값을 갖고 return한다.
        if ((rc = write_word(info, wp, data)) != 0) { return (rc); }
        wp += 4; @ flash address에 word정렬을 위하여 4을 다한다.
        }
```

FLASH.c 분석(20) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – write_buff() 계속**

```
@ 2word단위로 정렬 되는 부분을 처리한다.

    while (cnt >= 4) {
                    data = 0;
@ memory의 byte 데이터를 2word단위로 조합한다.
                    for (i=0; i<4; ++i) {
                            data = (data << 8) | *src++;
                    }
@ flash에 2word단위로 write한다. 0가 아니면 error값을 갖고 return한다.
                    if ((rc = write_word(info, wp, data)) != 0) {
                            return (rc);
                    }
@ flash address에 word정렬을 위하여 4을 다한다.
                    wp  += 4;
                    cnt -= 4;
            }
@ 나머지 byte들이 없이 count가 0가 되면 정상(0) return한다.
            if (cnt == 0) {
                    return (0);
            }
```

FLASH.c 분석(21) – (TOPDIR)/board/smdk2410/flash.c

❖ Flash.c – write_buff() 계속

```
@ 2word단위로 정렬 되지 않은 나머지 부분을 처리한다.
        data = 0;
@ 남아있는 byte을 2word 처리하기 위해 8bit씩 shift하여 조합한다.
        for (i=0, cp=wp; i<4 && cnt>0; ++i, ++cp) {
                data = (data << 8) | *src++;
                --cnt;
        }
@ 2word정렬에서 모자라는 부분을 flash에서 read하여 처리한다.
        for (; i<4; ++i, ++cp) {
                data = (data << 8) | (*(uchar *)cp);
        }
@ flash에 2word단위로 write한다. 0가 아니면 error값을 갖고 return한다.
        return (write_word(info, wp, data));
}
```

FLASH.c 분석(1) – (TOPDIR)/common/flash.c

❖ **Flash.c – flash_protect()**

➢ 입력된 주소 범위 안에 있는 sector의 protect 속성을 update하는 함수이다.

```
@ flag => 1(protect), 2(unprotect)   from => start address   to => end address
void
flash_protect (int flag, ulong from, ulong to, flash_info_t *info)
{
        ulong b_end = info->start[0] + info->size - 1;          @ flash end address
        short s_end = info->sector_count - 1;                   @ sector end number
        int i;

@ 잘못된 input data가 입력되면 return한다.
        if (info->sector_count == 0 || info->size == 0 || to < from) {
                return;
        }

@ flash에 대한 정보를 가지고 있지 않거나 flash영역과 protect영역이 overlay되어 있지 않으면 return한다.
        if (info->flash_id == FLASH_UNKNOWN ||
           to < info->start[0] || from > b_end) {
                return;
        }
```

FLASH.c 분석(2) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_protect() 계속**

```
        for (i=0; i<info->sector_count; ++i) {
                ulong end;              @ 현재 sector의 마지막 address
                @ 마지막 sector address을 b_end로 사용한다.
                end = (i == s_end) ? b_end : info->start[i + 1] - 1;

                @ 입력된 범위 안에 현재의 sector가 들어가면 해당하는 sector을 protection update한다.
                @ from(start addr.) <= current sector range >= to(end addr.)
                @ 현재는 hardware protection은 사용하지 않으며 software protection을 사용한다.
                @ flash_real_protection() - (TOPDIR)/board/smdk2410/flash.c
        if (from <= end && to >= info->start[i]) {
                        if (flag & FLAG_PROTECT_CLEAR) {        @ protect clear한다.
#if defined(CFG_FLASH_PROTECTION)
                                flash_real_protect(info, i, 0);
#else
                                info->protect[i] = 0;
#endif  /* CFG_FLASH_PROTECTION */
                        }
                        else if (flag & FLAG_PROTECT_SET) {     @ protect을 set한다.
#if defined(CFG_FLASH_PROTECTION)
                                flash_real_protect(info, i, 1);
#else
                                info->protect[i] = 1;
#endif  /* CFG_FLASH_PROTECTION */
                        }
                }
        } @ hardware protection을 지원하려면 함수를 만들어주어야 하며,지금은 지원하지 않는다.
}
```

FLASH.c 분석(3) – (TOPDIR)/common/flash.c

❖ **Flash.c – flash_write()**

➢ Flash에 특정주소에 cnt수 만큼 memory의 데이터를 저장하는 함수이다.

```
/*-----------------------------------------------------------------------
 * Copy memory to flash.
 * Make sure all target addresses are within Flash bounds,
 * and no protected sectors are hit.
 * Returns:
 * ERR_OK          0 - OK
 * ERR_TIMOUT      1 - write timeout
 * ERR_NOT_ERASED  2 - Flash not erased
 * ERR_PROTECTED   4 - target range includes protected sectors
 * ERR_INVAL       8 - target address not in Flash memory
 * ERR_ALIGN       16 - target address not aligned on boundary
 *                 (only some targets require alignment)
 */ @ (TOPDIR)/include/flash.h에 error값이 정의되어 있다.
int
flash_write (uchar *src, ulong addr, ulong cnt)
{
#ifdef CONFIG_SPD823TS
        return (ERR_TIMOUT);            /* any other error codes are possible as well */
#else
        int i;
        ulong       end       = addr + cnt - 1;         @ end address
        flash_info_t *info_first = addr2info (addr);@ start address에 해당하는 flash_info 가져온다.
        flash_info_t *info_last  = addr2info (end );@end address에 해당하는 flash_info 가져온다.
        flash_info_t *info;
        @ 현재는 bank을 하나만 사용하므로 flash_info 구조체도 하나만 존재한다.
```

FLASH.c 분석(4) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_write() 계속**

```
@ flash에 write할 count갯수가 0이면 return한다.
if (cnt == 0) {
        return (ERR_OK);
}
@ target address에 해당하는 flash 정보를 가지고 있는 구조체가 존재하지 않으면 error로 return한다.
if (!info_first || !info_last) {
        return (ERR_INVAL);
}

for (info = info_first; info <= info_last; ++info) {
        ulong b_end = info->start[0] + info->size; @ flash end address
        short s_end = info->sector_count - 1;      @ sector end number
        for (i=0; i<info->sector_count; ++i) {     @ sector 수 만큼 loop을 수행한다.
                ulong e_addr = (i == s_end) ? b_end : info->start[i + 1];
                @ 입력된 주소 범위에 해당하는 sector에 protect가 set되어 있는지 검사한다.
                if ((end >= info->start[i]) && (addr < e_addr) &&
                   (info->protect[i] != 0) ) {
                @ protect가 set되어 있으면 error로 return한다.
                        return (ERR_PROTECTED);
                }
        }
}
```

FLASH.c 분석(5) – (TOPDIR)/board/smdk2410/flash.c

❖ **Flash.c – flash_write() 계속**

```
        @ 최종적으로 flash에 write한다.
        for (info = info_first; info <= info_last && cnt>0; ++info) {
                ulong len;
                @ flash size에서 현재 address 빼면 현재 남아 있는 flash size을 구할 수 있다.
                len = info->start[0] + info->size - addr;
                @ 남아 있는 flash size가 크면 입력된 cnt만큼만 flash에 write한다.
                @ 적으면 남아있는 flash size만큼만 write을 한다. ??? Error처리
                if (len > cnt)
                        len = cnt;
                @ 입력된 start address부터 len만큼 memory의 데이터 내용을 flash에 write한다.
                if ((i = write_buff(info, src, addr, len)) != 0) {
                @ write시 error발생하면 error로 return한다.
                        return (i);
                }
                cnt  -= len;
                addr += len;
                src  += len;
        }
        return (ERR_OK);
#endif /* CONFIG_SPD823TS */
}
```

FLASH.c 분석(6) – (TOPDIR)/common/flash.c

- **Flash.c – flash_perror()**
  - Flash error을 console에 display한다.

```
void flash_perror (int err)
{
        switch (err) {
        case ERR_OK:
                break;
        case ERR_TIMOUT:
                puts ("Timeout writing to Flash\n");
                break;
        case ERR_NOT_ERASED:
                puts ("Flash not Erased\n");
                break;
        case ERR_PROTECTED:
                puts ("Can't write to protected Flash sectors\n");
                break;

                중략……

        case ERR_PROG_ERROR:
                puts ("General Flash Programming Error\n");
                break;
        default:
                printf ("%s[%d] FIXME: rc=%d\n", __FILE__, __LINE__, err);
                break;
        }
}
```

(TOPDIR)/lib_arm directory

Board.c

...

..

board.c 분석(1) – (TOPDIR)/lib_arm/board.c

❖ **board.c – int(init_fnc_t)(void)**

➢ Board에 관련된 모든 자원을 초기화한다.

```
typedef int (init_fnc_t) (void);

init_fnc_t *init_sequence[] = {
@ (TOPDIR)/cpu/arm920t/cpu.c
        cpu_init,                       @ uboot의 실제 end address을 구한다.(stack 포함)
@ (TOPDIR)/board/smdk2410/smdk2410.c
        board_init,                     @ clock 및 GPIO 초기화 ,kernel 입력값,I/D cache enable
@ (TOPDIR)/cpu/arm920t/interrupts.c
        interrupt_init,                 @ console에 사용할 timer을 초기화한다.
@ (TOPDIR)/common/env_flash.c
        env_init,                       @ environment 초기화한다.
@ (TOPDIR)/lib_arm/board.c
        init_baudrate,                  @ console에 사용할 UART의 baud rate을 설정한다.
@ (TOPDIR)/cpu/arm920t/serial.c
        serial_init,                    @ console에 사용할 UART를 초기화한다.
@ (TOPDIR)/common/console.c
        console_init_f,                 @ console을 output용으로만 초기화한다.
@ (TOPDIR)/lib_arm/board.c
        display_banner,                 @ uboot version및 code,data,stack의 위치를 console에 출력
@ (TOPDIR)/board/smdk2410/smdk2410.c
        dram_init,                      @ sdram start address,size을 global struct 에등록
@ (TOPDIR)/lib_arm/board.c
        display_dram_config,            @ sdram start address,size을 console에 출력
        NULL,
};
```

board.c 분석(2) – (TOPDIR)/lib_arm/board.c

❖ **board.c – start_armboot()**

➢ 어셈블리 코드(start.S)수행후 최초로 실행되는 C 코드이다. 초기화 및 MAIN함수로이동

```
void start_armboot (void)
{
        DECLARE_GLOBAL_DATA_PTR;

        ulong size;
        gd_t gd_data;
        bd_t bd_data;
        init_fnc_t **init_fnc_ptr;
        char *s;
#if defined(CONFIG_VFD)
        ...
#endif
        /* Pointer is writable since we allocated a register for it */
        gd = &gd_data;
        memset (gd, 0, sizeof (gd_t));
        gd->bd = &bd_data;
        memset (gd->bd, 0, sizeof (bd_t));          @ global_data struct의 data을 0으로 초기화한다.
        @ uboot의 flash size을 구한다.
        monitor_flash_len = _armboot_end_data - _armboot_start;
        @ board에관련된 모든 자원을 초기화한다.
        for (init_fnc_ptr = init_sequence; *init_fnc_ptr; ++init_fnc_ptr) {
                if ((*init_fnc_ptr)() != 0) {
                        hang ();
                }
        }
```

board.c 분석(3) – (TOPDIR)/lib_arm/board.c

❖ board.c – start_armboot()계속

```
/* configure available FLASH banks */
        size = flash_init ();                       @ flash을 초기화하고 flash size을 얻는다.
        display_flash_config (size);                @ flash의 size을 console에 출력한다.

#ifdef CONFIG_VFD
        ...
#else
        @ heap영역 설정=>_armboot_real_end+CFG_MALLOC_LEN(128Kbyte)
        mem_malloc_init (_armboot_real_end);
#endif /* CONFIG_VFD */

#if (CONFIG_COMMANDS & CFG_CMD_NAND)
        ...
#endif

#ifdef CONFIG_HAS_DATAFLASH
        ...
#endif
        /* initialize environment */
        @ flash에서 sdram으로 copy한다.
        env_relocate ();

#ifdef CONFIG_VFD
        ...
#endif
```

board.c 분석(4) – (TOPDIR)/lib_arm/board.c

❖ board.c – start_armboot() 계속

```
/* IP Address */
        bd_data.bi_ip_addr = getenv_IPaddr ("ipaddr");

        /* MAC Address */
        {
                int i;
                ulong reg;
                char *s, *e;
                uchar tmp[64];

                i = getenv_r ("ethaddr", tmp, sizeof (tmp));
                s = (i > 0) ? tmp : NULL;

                for (reg = 0; reg < 6; ++reg) {
                        bd_data.bi_enetaddr[reg] = s ? simple_strtoul (s, &e, 16) : 0;
                        if (s)
                                s = (*e) ? e + 1 : e;
                }
        }
        @ 설정되어 있는 device가있으면 초기화한다.(i2c,lcd,keyboard)
        devices_init ();      /* get the devices list going. */
```

board.c 분석(5) – (TOPDIR)/lib_arm/board.c

❖ board.c – start_armboot() 계속

```
        /* Syscalls are not implemented for ARM. But allocating
         * this allows the console_init routines to work without #ifdefs
         */
        syscall_tbl = (void **) malloc (NR_SYSCALLS * sizeof (void *));

        console_init_r ();      /* fully init console as a device */

#if defined(CONFIG_MISC_INIT_R)
        ...
#endif

        /* enable exceptions */
        @ interrupt을 enable을 한다. Smdk2410보드에서는 interrupt을 사용하지않는다.(smdk2410.h)
        enable_interrupts ();

#ifdef CONFIG_DRIVER_CS8900
        cs8900_get_enetaddr (gd->bd->bi_enetaddr);
#endif

#ifdef CONFIG_DRIVER_LAN91C96
        ...
#endif /* CONFIG_DRIVER_LAN91C96 */
```

board.c 분석(6) – (TOPDIR)/lib_arm/board.c

❖ board.c – start_armboot() 계속

```
/* Initialize from environment */
        if ((s = getenv ("loadaddr")) != NULL) {
                load_addr = simple_strtoul (s, NULL, 16);
        }

#if (CONFIG_COMMANDS & CFG_CMD_NET)
        ...
#endif  /* CFG_CMD_NET */


#ifdef BOARD_POST_INIT
        ...
#endif

        /* main_loop() can return to retry autoboot, if so just run it again. */
        for (;;) {
                main_loop ();          @(TOPDIR)/common/main.c

        }

        /* NOTREACHED - no way out of command loop except booting */
}
```