이젠 마지막으로 남은 RAM disk에 대한 것이다. src에는 RAMDISK_RAM_BASE(=0xC0800000)을 주고, dst에는 INITRD_START(Initial RAM Disk Start = KERNEL_START + KERNEL_LEN)를 준다. numWords에는 INITRD_LEN(=0x280000 or 5 x 512K, 대략 2.5 Mbytes정도)를 4로 나눈 값을 주고, blob_status.ramdiskSize 에는 0을 blob_status.ramdiskType에는 fromFlash를 두어서, flash부터 loading함을 설정한다. 역시 SerialOutputString() 함수를 사용해서 ramdisk를 flash로부터 loading한다는 것을 알려준다. 그 외의 경우에는 잘못된 Reload() 함수의 호출이므로 debugging 메시지를 출력한 후 복귀하도록 한다. 따라서, 최종적으로 RAM에 loading된 flash image는 다음과 같게 나타내지게 된다.

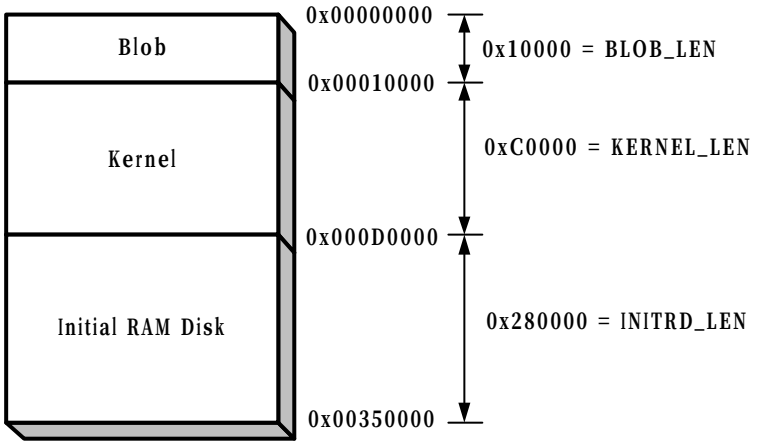

그림 1. Reload() 함수 수행이후의 메모리 상태

여기까지 해서, flash에 있는 메모리를 시스템의 DRAM 영역으로 loading하는 것을 마쳤다. 다시 main() 함수로 돌아가서 계속 진행하도록 하자.

```
/* wait 10 seconds before starting autoboot */
SerialOutputString("Autoboot in progress, press any key to stop ");
for(i = 0; i < 10; i++) {
        SerialOutputByte('.');
        retval = SerialInputBlock(commandline, 1, 1);
        if(retval > 0)
                break;
}
/* no key was pressed, so proceed booting the kernel */
if(retval == 0) {
        commandline[0] = '\0';
```

```
                boot_linux(commandline);
        }
        SerialOutputString("\nAutoboot aborted \n");
        SerialOutputString("Type \"help\" to get a list of commands \n");
```

코드 1. main() 함수의 정의(계속)

SerialOutputString()을 호출해서 autoboot이 진행할 것임을 나타내게된다. 이때 for loop를 돌면서, SerialInputBlock()을 호출해서 입력을 받도록한다. 만약 특정 입력이 있다면, retval은 0이 아닌 값을 가지게 될 것이며, SerialOutputString()을 호출해서 autoboot이 abort되었음을 표시하고, BLOB는 command를 받을 준비를 하게된다. 만약 아무런 input을 받지 않았다면, retval은 0이 될 것이며, commandline에는 '\0'을 첫번째 argument로 주고, boot_linux() 함수를 호출할 것이다.

먼저 SerialInputBlock() 함수를 보도록 하자. SerialInputBlock()함수는 결과적으로는 SerialInputByte() 함수를 호출하므로 SerialInputByte() 함수와 같이 보도록 하겠다. 정의는 serial.c에 있다.

```
int SerialInputByte(char *c)
{
#if defined USE_SERIAL1
        if(Ser1UTSR1 & UTSR1_RNE) {
                int err = Ser1UTSR1 & (UTSR1_PRE | UTSR1_FRE | UTSR1_ROR);
                *c = (char)Ser1UTDR;
#elif defined USE_SERIAL3
        if(Ser3UTSR1 & UTSR1_RNE) {
                int err = Ser3UTSR1 & (UTSR1_PRE | UTSR1_FRE | UTSR1_ROR);
                *c = (char)Ser3UTDR;
#else
#error "Configuration error: No serial port at all"
#endif
                /* If you're lucky, you should be able to use this as
                 * debug information ;-) -- Erik
                 */
                if(err & UTSR1_PRE)
                        SerialOutputByte('@');
                else if(err & UTSR1_FRE)
                        SerialOutputByte('#');
```

```
                else if(err & UTSR1_ROR)
                        SerialOutputByte('$');
                /* We currently only care about framing and parity errors */
                if((err & (UTSR1_PRE | UTSR1_FRE)) != 0) {
                        return SerialInputByte(c);
                } else {
                        led_toggle();
                        return(1);
                }
        } else {
                /* no bit ready */
                return(0);
        }
} /* SerialInputByte */
```

코드 2. SerialInputByte() 함수의 정의

SerialInputByte() 함수는 serial port로부터 input을 받는 함수이다. 성공했을 때는 1을 돌려줄 것이며, 실패시에는 0을 돌려준다. 성공적인 경우에는 읽은 character는 넘겨받은 c 변수에 저장된다. 먼저 사용할 serial port가 무엇인가에 따라서 Ser1UTSR1이나, Ser3UTSR1을 보도록 할 것이며, UTSR1_RNE는 Receive FIFO Not Empty를 나타낸다. 따라서, receiver FIFO가 비어있지 않다는 말은 입력을 받을 수 있다는 말이되므로, input을 받기 위해서 Ser1UTDR이나 Ser3UTDR을 접근하게 된다. 읽은 값은 c에 저장된다. 에러가 있었다면, Ser1UTSR1이나 Ser3UTSR1을 확인해서 이 에러값을 err에 저장한다.
각각의 에러가 의미하는 바는 다음과 같다. PRE의 경우는 Parity Error를 나타내며, FRE는 Framing Error를 나타내고, ROR은 Receive Overrun을 나타낸다. Parity error는 받은 데이터의 parity가 문제가 있음을 나타내며, frame error는 stop bit이 1이 아니라, 0이 되었음을 나타내며, receive overrun은 receiver쪽의 FIFO가 이미 full이 되었음을 의미한다. 각 에러 상황에 대해서 SerialOutputByte() 함수를 호출해서 적절한 문자를 display해준다. 만약 error가 PRE나 FRE라면, 다시 한번 SerialInputByte() 함수를 호출해서 읽게되며, 그렇지 않다면, led_toggle() 함수를 호출해서 led_off()와 led_on()을 이전의 LED 상태(led_state)에 맞게 호출해서 깜빡여 준 후, 1을 돌려줄 것이다. Receive FIFO가 비어있다면, 당연히 0을 돌려준다.

```
int  SerialInputBlock(char *buf, int bufsize, const int timeout)
{
        u32 startTime, currentTime;
```

```
        char c;
        int i;
        int numRead;
        int maxRead = bufsize;

        startTime = TimerGetTime();
        for(numRead = 0, i = 0; numRead < maxRead;) {
                /* try to get a byte from the serial port */
                while(!SerialInputByte(&c)) {
                        currentTime = TimerGetTime();
                        /* check timeout value */
                        if((currentTime - startTime) >
                          (timeout * TICKS_PER_SECOND)) {
                                /* timeout! */
                                return(numRead);
                        }
                }
                buf[i++] = c;
                numRead ++;
        }
        return(numRead);
}
```

코드 3. SerialInputBlock() 함수의 정의

SerialInputBlock()함수는 특정한 문자(character)가 들어왔는지를 보지 않고, 단순히 읽은 문자들의 block을 돌려주는 일을 한다. 먼저 startTime에 현재의 시간(TimerGetTimer())을 보관하고, for loop를 돌면서 문자들을 읽어 들인다. 읽은 문자의 개수가 return값으로 사용된다(numRead). Timeout값은 넘겨준 값에 의존하며, 초단위로 나타낸다. [1] 만약 SerialInputByte()를 while loop를 돌면서 계속 호출해서 특정한 수의 input 문자를 읽어들이면, for loop를 끝낸다. 그렇지 않고, 입력이 없이 timeout이 된다면, 현재의 읽은 문자의 개수 만큼을 돌려주게 된다. 이곳에서 timeout을 검사하기 위해서 사용하는 TimerGetTime() 함수는 time.c에 아래와 같이 정의된다.

```
/* returns the time in 1/TICKS_PER_SECOND seconds */
```

[1] 앞에서 이미 1로 설정해서 호출해 주었다(main()함수).

```
u32  TimerGetTime(void)
{
        /* turn LED always on after one second so the user knows that
       * the board is on
        */
        if((OSCR % TICKS_PER_SECOND) < (TICKS_PER_SECOND >>7))
                led_on();
        return((u32) OSCR);
}
```

코드 4. TimerGetTime() 함수의 정의

즉, OSCR 레지스터를 읽어서 바로 돌려준다. 중간에 시스템이 살아있다는 것을 보여주기 위해서 led_on()을 호출해서 LED를 켜주는 것이 전부이다. TICKS_PER_SECOND는 3686400를 가지는 값으로 OS timer의 초당 clock frequency를 나타내는 값이다.

```
void boot_linux(char *commandline)
{
        register u32 i;
        void (*theKernel)(int zero, int arch) = (void (*)(int, int))KERNEL_RAM_BASE;

        setup_start_tag();
        setup_memory_tags();
        setup_commandline_tag(commandline);
        setup_initrd_tag();
        setup_ramdisk_tag();
        setup_end_tag();

        /* we assume that the kernel is in place */
        SerialOutputString("\nStarting kernel ...\n\n");

        /* turn off I-cache */
        asm ("mrc p15, 0, %0, c1, c0, 0": "=r" (i));
        i &= ~0x1000;
        asm ("mcr p15, 0, %0, c1, c0, 0": : "r" (i));

        /* flush I-cache */
```

```
        asm ("mcr p15, 0, %0, c7, c5, 0": : "r" (i));
        theKernel(0, ARCH_NUMBER);
        SerialOutputString("Hey, the kernel returned! This should not happen. \ n");
}
```

코드 5. boot_linux() 함수의 정의

boot_linux() 함수는 linux.c에 정의된 함수이다. 여기서 하는 일은 커널에 전달될 parameter를 만드는 일과 I-cache를 OFF시키고, flush시키는 일이며, 최종적으로 압축된 Linux을 실행시키는 일이다. 따라서, 압축된 커널의 시작부분에는 uncompression하는 code가 들어가 있을 것이다. 리눅스 커널의 실행시에 중요한 넘겨주어야 할 변수로는 첫번째 argument에는 0을, 두번째 argument에는 architecture 번호를 주어야 한다는 것이다. Kernel이 return을 하지않을 것이므로, 마지막 line에 있는 SerialOutputString()은 실행되지 않을 것이다.

넘겨주는 architecture 번호에 대한 정의는 linux.h에 아래와 같이 되어 있다. 같은 것을 커널 source의 ~/arch/arm/tools/mach_types과 같은 곳에서 찾을 수 있다.

```
#if defined ASSABET
# define ARCH_NUMBER (25)
#elif defined BRUTUS
# define ARCH_NUMBER (16)
#elif defined CLART
# define ARCH_NUMBER (68)
#elif defined LART
# define ARCH_NUMBER (27)
#elif defined NESA
# define ARCH_NUMBER (75)
#elif defined PLEB
# define ARCH_NUMBER (20)
#elif defined SHANNON
# define ARCH_NUMBER (97)
#else
#warning "FIXME: Calling the kernel with a generic SA1100 architecture code. YMMV!"
#define ARCH_NUMBER (18)
#endif
```

코드 6. 각 CPU architecture에 따른 ARCH_NUMBER의 정의

또한, 넘겨준 architecture 번호 값은 커널 source의 ~/arch/arm/boot/compressed/head.S 에서 다음과 같이 사용하게 된다.

```
...
                .align
start:
                .type   start,#function
                .rept   8
                mov     r0, r0
                .endr

                b       1f
                .word   0x016f2818              @ Magic numbers to help the loader
                .word   start                   @ absolute load/run zImage address
                .word   _edata                  @ zImage end address
1:              mov     r7, r1                  @ save architecture ID
                mov     r8, #0                  @ save r0
...
```

코드 7. 커널의 head.S의 시작부분

head.S에서 보듯이 넘겨받은 argument값은 r0에는 0이, r1에는 ARCH_NUMBER가 들어가게 되며, 이 값은 다시 r7에 들어가서, architecture ID로서의 구실을 하게된다. 더 자세한 것은 나중에 SA1100의 booting에 대해서 설명하면서 보게 될 것이므로, 여기서는 이 정도에서 마치기로 하고, setup_XXX() 함수들에 대해서 보기로 하겠다. setup_XXX() 함수는 커널에 넘겨줄 parameter를 생성해주는 함수이다. 이것은 마치 x86에서 LILO를 사용하게 되는 경우와 동일한 것으로 커널에 적절한 argument를 넘겨주어서, 커널에서 사용하는 변수를 설정하는 일을 한다.

먼저 setup_XXX() 함수를 보기 전에 setup_XXX() 함수에서 사용하게 되는 tag 구조체에 대한 정의부터 보기로 하자. 이 구조체는 커널 source의 ~/include/asm-arm/setup.h에 아래와 같은 정의를 가진다.

```
struct tag {
        struct tag_header hdr;                          /* tag의 크기와 magic
number */
        union {
```

```
                struct tag_core             core;
                struct tag_mem32        mem;            /* 메모리의 크기와 시작 주
소 */
                struct tag_videotext    videotext;      /* 비디오 RAM의 위치와
resolution */
                struct tag_ramdisk      ramdisk;        /* RAM disk의 옵션, 크기
및 시작 주소 */
                struct tag_initrd       initrd;         /* Initial RAM disk의 크기와
시작 주소 */
                struct tag_serialnr             serialnr;       /* Serial의 개수 */
                struct tag_revision     revision;       /* Revision 번호 */
                struct tag_videolfb     videolfb;       /* Video Fram Buffer의 설
정 사항 */
                struct tag_cmdline      cmdline;
                /*
                 * Acorn specific
                 */
                struct tag_acorn acorn;                 /* ACORN specific 정보 */
                /*
                 * DC21285 specific                     /* DC21285 specific 정보
*/
                 */
                struct tag_memclk       memclk;         /* 메모리 clock 정보 */
        } u;
};
```

코드 8. tag 구조체의 정의

즉, 위와 같이 정의된 tag 구조체를 적절한 메모리 위치에 두고, 커널에서는 나중에 이곳에 정의된 정보를 다시 읽어서 사용하도록 하는 것이다. 이것이 정의된 위치는 main.h에 있는 BOOT_PARAMS(=0xc0000100)에 들어가게 된다.

```
static void setup_start_tag(void)
{
        params = (struct tag *)BOOT_PARAMS;

        params ->hdr.tag = ATAG_CORE;
```

```
        params->hdr.size = tag_size(tag_core);
        params->u.core.flags = 0;
        params->u.core.pagesize = 0;
        params->u.core.rootdev = 0;
        params = tag_next(params);
}
```

코드 9. setup_start_tag() 함수의 정의

이 함수는 BOOT_PARAMS에 가지고 있는 tag구조체를 초기화 하는 역할을 한다. ATAG_CORE는 커널의 ~/include/asm-arm/setup.h에 정의된 것으로 0x54410001라는 값을 가진다.[2] 크기는 tag_core의 크기를 주도록 하고(tag_size()), union 구조체로 선언된 부분의 core에 대한 초기화를 위해서 flags에 0, pagesize에 0, rootdev에도 0을 준다. 다음 번의 tag의 위치를 주기 위해서 tag_next()를 사용하고 있다.[3]

```
#define tag_next(t)     ((struct tag *)((u32 *)(t) + (t)->hdr.size))
#define tag_size(type)  ((sizeof(struct tag_header) + sizeof(struct type)) >> 2)
```

코드 10. tag관련 매크로의 정의

tag_next() 매크로는 다음 tag구조체의 위치를 나타내는데 사용하며, tag_size() 매크로는 tag header의 길이와 해당하는 type의 크기를 합한 값을 4로 나누어 4 byte 단위로 나타낸 크기를 돌려준다.

```
static void setup_memory_tags(void)
{
        int i;

        for(i = 0; i < NUM_MEM_AREAS; i++) {
                if(memory_map[i].used) {
                        params->hdr.tag = ATAG_MEM;
                        params->hdr.size = tag_size(tag_mem32);
                        params->u.mem.start = memory_map[i].start;
                        params->u.mem.size = memory_map[i].len;
```

[2] 현재로서는 단순히 magic number를 가지는 것으로 보인다.

[3] tag_size()나 tag_next()은 새로운 버전의 커널에 들어 있는 것을 찾을 수 있을 것이다. 커널 버전 2.4.9에서는 ~/include/asm-arm/setup.h에서 찾을 수 있다. 다른 버전의 커널도 확인해 보기 바란다. 이것을 정의하고 있지 않다면, BLOB의 build가 제대로 되지 않을 것이다.

```
                        params = tag_next(params);
                }
        }
}
```

코드 11. setup_memory_tags() 함수의 정의

setup_memory_tags() 함수는 현재 시스템 메모리 설정에 대한 정보를 가르쳐줄 목적으로 사용한다. Tag의 header에 ATAG_MEM을 넣고 tag의 크기를 알려준 다음, 각각의 메모리 map(memory_map[])의 entry에 대해서 시작과 크기를 넣어준다.

```
static void setup_commandline_tag(char *commandline)
{
        char *p;

        /* eat leading white space */
        for(p = commandline; *p == ' '; p++)
                ;
        /* skip non-existent command lines so the kernel will still
       * use its default command line.
         */
        if(*p == '\0')
                return;
        params->hdr.tag = ATAG_CMDLINE;
        params->hdr.size = (sizeof(struct tag_header) + strlen(p) + 1 + 4) >> 2;
        strcpy(params->u.cmdline.cmdline, p);
        params = tag_next(params);
}
```

코드 12. setup_commandline_tag() 함수의 정의

이젠 넘겨받은 command line에 해당하는 tag을 생성할 차례이다. 먼저, 주어진 command line에서 blank를 제거하기 위해서 문자단위로 검사한다. 만약 이러한 검사를 진행하는 도중에 '\0'를 만난다면, 문자열의 끝을 나타내므로 그냥 return한다. 그렇지 않다면, header의 tag필드에는 command line이라는 것을 알려주는 ATAG_CMDLINE을 넣어주고, command line의 크기를 계산해 주고, 넘겨받은 실제 command line에서 blank를 제외한 부분을 copy해서 넣어준다.

```
static void setup_initrd_tag(void)
{
        /* an ATAG_INITRD node tells the kernel where the compressed
       * ramdisk can be found. ATAG_RDIMG is a better name, actually.
         */
        params->hdr.tag = ATAG_INITRD;
        params->hdr.size = tag_size(tag_initrd);

        params->u.initrd.start = RAMDISK_RAM_BASE;
        params->u.initrd.size = INITRD_LEN;

        params = tag_next(params);
}
```

코드 13. setup_initrd_tag() 함수의 정의

setup_initrd_tag()는 initial RAM disk에 대한 tag을 설정하는 것이다. ATAG_INITRD를 tag head에 넣어주고, RAMDISK_RAM_BASE(=0xC0800000)를 initial RAM disk의 시작 주소로 주고, 크기를 INITRD_LEN(=0x280000)으로 설정했다.

```
static void setup_ramdisk_tag(void)
{
        /* an ATAG_RAMDISK node tells the kernel how large the
       * decompressed ramdisk will become.
         */
        params->hdr.tag = ATAG_RAMDISK;
        params->hdr.size = tag_size(tag_ramdisk);

        params->u.ramdisk.start = 0;
        params->u.ramdisk.size = RAMDISK_SIZE;
        params->u.ramdisk.flags = 1;        /* automatically load ramdisk */

        params = tag_next(params);
}
```

코드 14. setup_ramdisk_tag() 함수의 정의

setup_ramdisk_tag() 함수는 RAM disk에 대한 tag을 설정하는 함수이다. 시작 주소로 0을,

크기로는 RAMDISK_SIZE(=8 x 1024 x 1024 or 8Mbytes)를 주었으며, 자동으로 RAM disk를 load하라는 뜻으로 flags에 1을 설정해 주었다.

```
static void setup_end_tag(void)
{
        params ->hdr.tag = ATAG_NONE;
        params ->hdr.size = 0;
}
```

코드 15. setup_end_tag() 함수의 정의

이젠 tag의 마지막임을 나타내기 위해서 setup_end_tag() 함수를 사용한다. ATAG_NONE을 tag header에 넣었고, tag의 크기가 0임을 알려주었다. 이것으로 커널에 넘겨질 tag에 대한 모든 설정은 끝났다.

실제로 커널에서 넘겨받은 tag에 대한 처리가 일어나는 곳은 ~/arch/arm/kernel/setup.c의 parse_tag_XXX()와 같은 함수이다. 이 함수에서는 넘겨준 tag들을 tag이 의미하는 뜻에 따라, 앞에서 이미 보았듯이, core, mamory, commandline, initrd, RAM disk등의 순으로 처리해 나가는 것을 볼 수 있을 것이다.

```
        /* the command loop. endless, of course */
        for(;;) {
                DisplayPrompt(NULL);
                /* wait 10 minutes for a command */
                numRead = GetCommand(commandline, 128, 600);
                if(numRead > 0) {
                        if(MyStrNCmp(commandline, "boot", 4) == 0) {
                                boot_linux(commandline + 4);
                        } else if(MyStrNCmp(commandline, "clock", 5) == 0) {
                                SetClock(commandline + 5);
                        } else if(MyStrNCmp(commandline, "download ", 9) == 0) {
                                Download(commandline + 9);
                        } else if(MyStrNCmp(commandline, "flash ", 6) == 0) {
                                Flash(commandline + 6);
                        } else if(MyStrNCmp(commandline, "help", 4) == 0) {
                                PrintHelp();
                        } else if(MyStrNCmp(commandline, "reblob", 6) == 0) {
```

```
                        Reblob();
                    } else if(MyStrNCmp(commandline, "reboot", 6) == 0) {
                        Reboot();
                    } else if(MyStrNCmp(commandline, "reload ", 7) == 0) {
                        Reload(commandline + 7);
                    } else if(MyStrNCmp(commandline, "reset", 5) == 0) {
                        ResetTerminal();
                    } else if(MyStrNCmp(commandline, "speed ", 6) == 0) {
                        SetDownloadSpeed(commandline + 6);
                    }
                    else if(MyStrNCmp(commandline, "status", 6) == 0) {
                        PrintStatus();
                    } else {
                        SerialOutputString("*** Unknown command: ");
                        SerialOutputString(commandline);
                        SerialOutputByte('\n');
                    }
                }
            }
            return 0;
} /* main */
```

코드 16. main.c 파일(계속)

다시 main.c로 돌아와서, 계속 진행하도록 하겠다. 만약 autoboot이 취소 되었다면, main() 함수는 for loop를 돌면서 사용자의 입력에 따른 반응을 보일 것이다. 먼저 DisplayPrompt() 함수를 호출해서 사용자가 입력을 하기를 기다리게 된다. 여기서 부터 명령어와 관련된 처리는 command.c 파일을 보기 바란다. GetCommand() 함수를 호출해서 내려진 명령을 commandline변수로 읽어서, 이 명령이 어떤 것인 가를 비교해서 (MyStrNCmp()), 해당하는 명령을 수행하는 loop를 돌게된다. 명령의 종류로는 boot, clock, download, flash, help, reblob, reboot, reload, reset, speed, status등이 있으며, 그외의 명령에 대해서는 unknown으로 생각한다. 먼저 DisplayPrompt() 함수와 GetCommand() 함수를 본 후, 각각의 command에 대해서 보도록 하자. main() 함수는 이것으로 마무리한다.

```
/* display a prompt, or the standard prompt if prompt == NULL */
void DisplayPrompt(char *prompt)
{
```

```
        if(prompt == NULL) {
                SerialOutputString(PACKAGE "> ");
        } else {
                SerialOutputString(prompt);
        }
}
```

코드 17. DisplayPrompt() 함수의 정의

DisplayPrompt() 함수는 단순히 “>”문자를 표시하는 역할을 한다. 만약 넘겨받은 prompt변수의 값이 NULL인 경우에는 “>”를 serial을 통해서 쓰게되고, 그렇지 않고 특정한 prompt를 요구하면, 그것을 그대로 보여준다.

```
/* more or less like SerialInputString(), but with echo and backspace  */
int GetCommand(char *command, int len, int timeout)
{
        u32 startTime, currentTime;
        char c;
        int i;
        int numRead;
        int maxRead = len - 1;

        TimerClearOverflow();
        startTime = TimerGetTime();
        for(numRead = 0, i = 0; numRead < maxRead;) {
                /* try to get a byte from the serial port */
                while(!SerialInputByte(&c)) {
                        currentTime = TimerGetTime();
                        /* check timeout value */
                        if((currentTime - startTime) >
                          (timeout * TICKS_PER_SECOND)) {
                                /* timeout */
                                command[i++] = '\0';
                                return(numRead);
                        }
                }
                if((c == '\r') || (c == '\n')) {
```

```
                command[i++] = '\0';
                /* print newline */
                SerialOutputByte('\n');
                return(numRead);
        } else if(c == '\b') { /* FIXME: is this backspace? */
                if(i > 0) {
                        i--;
                        numRead--;
                        /* cursor one position back. */
                        SerialOutputString("\b \b");
                }
        } else {
                command[i++] = c;
                numRead++;
                /* print character */
                SerialOutputByte(c);
        }
    }
    return(numRead);
}
```

코드 18. GetCommand() 함수의 정의

GetCommand()함수는 특정 시간동안 명령 input을 받는 함수이다. 먼저 TimerClearOverflow() 함수를 호출해서, timer가 overflow가 되었다면 이를 지워준다. 정의는 time.c에 아래와 같이 되어있다.

```
int TimerDetectOverflow(void)
{
    return(OSSR & OSSR_M0);
}
void TimerClearOverflow(void)
{
    if(TimerDetectOverflow())
        numOverflows++;
    OSSR = OSSR_M0;
}
```

코드 19. TimerClearOverflow() 함수의 정의

TimerClearOverflow()함수는 TimerDetectOverflow() 함수를 다시 호출해서 overflow가 일어났는지를 확인하게 되고, 만약 그렇다면, numOverflows 변수를 하나 증가 시키게 되며, OSSR(Operating System Status Register)은 OSMR(Operating System Match Register) 4개의 레지스터 중에서 어떤 레지스터와 OSCR(Operating System Counter Register)이 일치하였는지를 나타낸다. 따라서, 이 값이 OSMR0와 일치가 된다면(OSSR_M0 = 0x00000001), overflow가 일어났다고 생각하도록 한다. 즉, 현재 우리가 사용하는 timer와 관련된 match register로 앞에서 OSMR0를 사용하기에 이것만을 기준으로 삼고 있다(TimerInit()). For loop를 돌기전에 미리 현재의 시간을 읽어서(TimerGetTime()) startTime 변수에 저장하고, 이 값을 기준으로 얼마나 시간이 흘렀냐에 따라서 timeout을 시켜준다.

For loop는 최대 읽기 허용 byte를 넘어서거나, timeout이 일어난다면 현재까지 읽은 byte를 돌려주고 복귀할 것이며, 만약 ‘\r’이나 ‘\n’과 같은 문자를 받더라도, 같은 일을 할 것이다. 문자열을 읽는 함수는 SerialInputByte()를 사용한다. 한 문자를 읽을 때마다, timeout이 계산되며, 허용된 timeout을 넘어서게 되면, 현재까지 읽은 command line에 ‘\0’을 덧붙인 문자열을 넘겨줄 것이며, 현재까지 읽은 byte수를 복귀 값으로 전달한다. 만약 읽은 문자가 ‘\b’(=back space)인 경우에는 읽은 문자의 수를 감소시켜주게 되며, 표시하는 문자에는 ‘\b’를 두번 써서, 지우는 효과를 보이도록 할 것이다. 그 외의 문자에 대해서는 읽은 값을 그대로 echo시켜준다.