1.1.1.1. Flash Command

Flash command의 처리는 Flash() 함수가 맡고 있다. 정의는 main.c에 있으며, 메모리에 있는 데이터 block을 다시 flash 메모리에 write하는 일을 한다. write되는 내용에는 BLOB, 커널, RAM 디스크 등이다. 아래와 같다.

```
void Flash(char *commandline)
{
        u32 startAddress = 0;
        tBlockType block;
        int numBytes = 0;
        int maxSize = 0;

        if(MyStrNCmp(commandline, "blob", 4) == 0) {
                startAddress = BLOB_RAM_BASE;
                block = blBlob;
                numBytes = blob_status.blobSize;
                maxSize = BLOB_LEN;
                if(blob_status.blobType == fromFlash) {
                        SerialOutputString("*** No blob downloaded \ n");
                        return;
                }
                SerialOutputString("Saving blob to flash ");
        } else if(MyStrNCmp(commandline, "kernel", 6) == 0) {
                startAddress = KERNEL_RAM_BASE;
                block = blKernel;
                numBytes = blob_status.kernelSize;
                maxSize = KERNEL_LEN;
                if(blob_status .kernelType == fromFlash) {
                        SerialOutputString("*** No kernel downloaded \ n");
                        return;
                }
                SerialOutputString("Saving kernel to flash ");
        } else if(MyStrNCmp(commandline, "ramdisk", 7) == 0) {
                startAddress = RAMDISK_RAM_BASE;
                block = blRamdisk;
```

```
            numBytes = blob_status.ramdiskSize;
            maxSize = INITRD_LEN;
            if(blob_status.ramdiskType == fromFlash) {
                SerialOutputString("*** No ramdisk downloaded \ n");
                return;
            }
            SerialOutputString("Saving ramdisk to flash ");
    } else {
            SerialOutputString("*** Don't know how to flash  \ "");
            SerialOutputString(commandline);
            SerialOutputString("\ " \ n");
            return;
    }
    if(numBytes > maxSize) {
            SerialOutputString("***   Downloaded   image   too   large   for   flash
area \ n");
            SerialOutputString("*** (0x");
            SerialOutputHex(numBytes);
            SerialOutputString(" downloaded, maximum size is 0x");
            SerialOutputHex(maxSize);
            SerialOutputString(" bytes) \ n");
            return;
    }
    EraseBlocks(block);
    SerialOutputByte(' ');
    WriteBlocksFromMem(block, (u32 *)startAddress, numBytes);
    SerialOutputString(" done \ n");
}
```

코드 1. Flash() 함수의 정의

Flash() 함수가 받는 commandline은 앞에서 이미 "flash "가 제거된 문자열이다. 따라서, 받아들은 문자열이 "blob", "kernel", "ramdisk"인지에 따라서, flash에 write할 내용이 달라지게 된다. Write를 하기 위해서는 먼저 flash 메모리에서 해당 데이터 block을 지우고 (EraseBlocks()), 새로운 block을 쓰게 된다(WriteBlocksFromMem()). EraseBlocks() 함수가 필요로 하는 block 변수는 각각에 대해서 blBlob, blKernel, blRamdisk로 정해지며, flash.h에 enum 변수로 정해져 있다. EraseBlocks() 함수 자체는 flash.c에 있다. 또한,

WriteBlocksFromMem() 함수가 필요로 하는 argument로는 어떤 종류의 데이터 block에 대한 것인가를 나타내는 block 변수와 그 block의 시작 주소(startAddress) 및, 써야하는 크기(numbytes)가 된다. 그외의 다른 경우에 대해서는 잘못된 flash 명령이라는 것을 보여주게 되며, 지우거나 쓰기전에 항상 최대로 flash에 write를 해야 하는 크기가 올바른 값을 가지는지 확인하게 된다. WriteBlocksFromMem() 함수도 역시 flash.c에 정의되어 있다.
각각의 경우에 대해서 startAddress가 가지는 값과, 써야하는 크기, 그리고 최대 값은 다음과 같이 볼 수 있다. 즉, BLOB의 경우에는 BLOB_RAM_BASE(=0xC1000000), blob_status.blobSize, BLOB_LEN(=0x10000)이 되며, 커널의 경우에는 KERNEL_RAM_BASE(=0xC0008000), blob_status.kernelSize, KENREL_LEN(=0xc0000)가 되며, RAM 디스크의 경우에는 RAMDISK_RAM_BASE(=0xC0800000), blob_status.ramdiskSize, INITRD_LEN(=0x280000)이 된다. 즉, 이것은 이미 download command에서 downloading을 수행해서 각각의 image를 가지고 있는 주소가 된다.

```
void EraseBlocks(tBlockType which)
{
        char    *thisBlock;
        int     numBlocks, i;

        switch(which) {
        case blBlob:
                thisBlock = (char *)BLOB_START;
                numBlocks = NUM_BLOB_BLOCKS;
                break;
        case blKernel:
                thisBlock = (char *)KERNEL_START;
                numBlocks = NUM_KERNEL_BLOCKS;
                break;
        case blRamdisk:
                thisBlock = (char *)INITRD_START;
                numBlocks = NUM_INITRD_BLOCKS;
                break;
        default:
                /* this should not happen */
                return;
        }
        for(i = 0; i < numBlocks; i++, thisBlock += MAIN_BLOCK_SIZE) {
                SerialOutputByte('.');
                led_toggle();
                if((EraseOne(thisBlock) & STATUS_ERASE_ERR) != 0) {
                    SerialOutputString("\ n*** Erase error at address 0x");
                    SerialOutputHex((u32)thisBlock);
                    SerialOutputByte(' \ n');
                    return;
                }
        }
} /* EraseBlocks */
```

코드 2. EraseBlocks() 함수의 정의

EraseBlocks() 함수는 넘어오는 bBlockType의 which 값에 따라서 switch문으로 분기한다. 즉, blBlob인 경우에는 BLOB 자체의 이미지를 지우라는 것이며, blKernel인 경우에는 kernel을, blRamdisk인 경우에는 RAM disk가 가진 block을 지우라는 뜻이다. 각각에 대해서 시작 주소와 지워야 할 block의 수를 thisBlock과 numBlocks 변수로 나타낸다. 이 변수들에 들어가는 내용은 코드에서 보듯이 XXX_START와 NUM_XXX_BLOCKS로 나타내진다. 실질적인 flash 메모리에 대한 지우기는 EraseOne() 함수가 처리한다. 즉, 지워야할 block의 수 만큼 시작주소부터 한번에 한 block씩 지워 나가게 된다. 여기서 한번에 지우는 단위가 되는 MAIN_BLOCK_SIZE의 값은 32768 * 4(= 32K * 4 = 128K) Bytes가 된다.

```
static u32 EraseOne(const char *whichOne)
{
/* Routine to erase one block of flash */
    volatile u32 *writeMe = (u32 *)whichOne;
    u32         result;
#if defined SHANNON || defined NESA
/* SHANNON이나 NESA가 정의된 경우에 실행되는 코드가 이곳에 있다. */
   …
    return 0;
#else
    *writeMe = data_to_flash(ERASE_SETUP);
    *writeMe = data_to_flash(ERASE_CONFIRM);
    do {
        *writeMe = data_to_flash(STATUS_READ);
        result = data_from_flash(*writeMe);
    } while((~result & STATUS_BUSY) != 0);
    *writeMe = data_to_flash(READ_ARRAY);
    return result;
#endif
} /* EraseOne */
```

코드 3. EraseOne() 함수의 정의

EraseOne() 함수가 넘겨받는 값은 지우기를 할 block의 시작 주소이다. Flash를 지우기 위한 절차는 다음과 같다. 먼저 setup 명령을 내려서(ERASE_SETUP) 지우기를 시작할 준비를 하도록 만들고, confirm으로 ERASE_CONFIRM을 flash에 지우라는 명령으로 내린다. 이젠 flash가 지우기를 끝내기를 기다리면 될 것이다. 이 것은 do{} while() loop를 돌면서 다음과 같은 일을 해주면 될 것이다. 즉, flash가 준비 되었는지를 상태를 읽는 명령으로 STATUS_READ를 하고, flash로 부터 상태를 읽어서, result에 보관한다. 이 값을 NOT한 것에 STATUS_BUSY를 AND 시켜서, 이 값이 0이 아닌동안 대기하면 된다. 즉, flash가 명령을 처리하는데 걸리는 시간적인 delay를 주고자 하는 것이다. 여기서 사용하고 있는 data_to_flash()와 data_from_flash()는 모두 flashasm.S에 정의되어 있다.[1]

```
void WriteBlocksFromMem(tBlockType type, const u32 *source, int length)
{
        volatile u32    *flashBase;
        u32             result;
        int             maxLength, i;


      …
        if((u32)source & 0x03) {
                SerialOutputString("*** Source is not on a word boundary: 0x");
                SerialOutputHex((u32)source);
                SerialOutputByte(' \ n');
                return;
        }
        if(length & 0x03)
                length += 0x04;
        length &= ~((u32) 0x03);
```

[1] LART board에서 사용하는 data_to_flash()와 data_from_flash()만이 현재는 정의된 상태이다. 나머지에 대해서는 확인된 것이 없으며, 그냥 subroutine을 return할 뿐이다. 이 함수들에 대한 분석은 다음으로 미룬다. 아직 코드에 대해서 명확히 파악하지 못했기 때문이다.

```
    switch(type) {
    case blBlob:
            flashBase = (u32 *)BLOB_START;
            maxLength = BLOB_LEN;
            break;
    case blKernel:
            flashBase = (u32 *)KERNEL_START;
            maxLength = KERNEL_LEN;
            break;
    case blRamdisk:
            flashBase = (u32 *)INITRD_START;
            maxLength = INITRD_LEN;
            break;
    default:
            /* this should not happen */
            return;
    }
    if(length > maxLength)
            length = maxLength;
```

코드 4. WriteBlocksFromMem() 함수의 정의

WriteBlocksFromMem() 함수가 넘겨받는 값은 쓰려고 하는 block의 type(blBlob, blKernel, blRamdisk)와 쓰려고하는 데이터의 주소(source), 그리고 길이(length)이다. 만약 length가 4byte단위로 정렬이 되지 않은 경우에는 정렬해(alignment) 준다. 각각의 쓰려고 하는 block의 type에 따라, flash의 기본 주소(flashBase)는 BLOB_START, KERNEL_START, INITRD_START의 값을 가질 것이며, 이 block이 가지는 최대 길이는 각각 BLOB_LEN, KERNEL_LEN, INITRD_LEN이 될 것이다. 만약 최대 길이보다 큰 크기를 가지는 데이터를 쓰고자 할 때는 쓰려고 하는 데이터의 길이를 허용하는 최대 길이(maxLength)로 맞춘다.

```
  …
    for(i = 0; i < length; i+= 4, flashBase++, source++) {
            if((i % MAIN_BLOCK_SIZE) == 0) {
                    SerialOutputByte('.');
                    led_toggle();
            }
```

```
                *flashBase = data_to_flash(PGM_SETUP);
                *flashBase = *source;
            …
                do {
                        *flashBase = data_to_flash(STATUS_READ);
                        result = data_from_flash(*flashBase);
                } while((~result & STATUS_BUSY) != 0);

                *flashBase = data_to_flash(READ_ARRAY);
                if((result & STATUS_PGM_ERR) != 0 || *flashBase != *source) {
                        SerialOutputString("\ n*** Write error at address 0x");
                        SerialOutputHex((u32)flashBase);
                        SerialOutputByte(' \ n');
                        return;
                }
        }
        …
} /* WriteBlocksFromMem */
```

코드 5.. **WriteBlocksFromMem()** 함수의 정의(계속)

이젠 한번에 4 bytes단위로 flash에 쓰는 과정이다. MAIN_BLOCK_SIZE만큼 쓰게 되면, '.'가 나타나며 LED가 점멸할 것이다(led_toggle()). 먼저 PGM_SETUP을 명령을 flash에 내린다. 이때, 내려준 명령에 대해서 주소를 data_to_flash()가 return해 주면, 이곳에 다시 source가 가르키는 내용을 넣어주도록 한다. 이젠 do{} while() loop를 돌면서 flash의 상태를 읽어 BUSY상태가 아닌 경우가 될 때까지 기다려주고, 다시 READ_ARRAY명령을 flash에 내린다. 만약 쓰기에 오류가 있거나, 쓴 내용(*flashBase)과 source가 가르키고 있는 내용이 다르다면 error 메시지를 출력한다. 이 for loop는 해당하는 길이 만큼을 다 쓸 때까지 반복적으로 수행한다.

Flash에 대한 명령 및 status는 다음과 같이 요약해서 볼 수 있다. 앞에서 flash를 지울 때도 사용했던 명령과 status를 같이 보도록 하자.

Command	Value	Description
READ_ARRAY	0x00FF00FF	Flash가 준비된 상태(즉, 다른 명령을 읽기 위해서 enable)가 되도록 만든다..
ERASE_SETUP	0x00200020	하나의 block에 대한 erase를 준비하라. 뒤에 ERASE_CONFIRM이 따라오게 된다.

ERASE_CONFIRM	0x00D000D0	Erase가 되도록 요청한다.
PGM_SETUP	0x00400040	다음의 flash에 대한 프로그램(program) 동작을 위해서 반복하라.
STATUS_READ	0x00700070	성공적으로 동작을 완료한 경우에 상태를 읽도록 한다. 즉, block에 대한 erase나, program, 혹은 lock bit에 대한 설정을 한 후에 상태 값을 읽어오기 위해서 사용한다.
STATUS_CLEAR	0x00500050	상태값을 clear시키기 위해서 사용한다.
STATUS_BUSY	0x00800080	현재 flash가 BUSY상태임을 나타낸다.
STATUS_ERASE_ERR	0x00200020	앞서 실행했던 flash에 대한 erase에 에러가 있었다.
STATUS_PGM_ERR	0x00100010	앞서 실행했던 flash에 대한 program에 에러가 있었다.

표 1. Flash에 대한 명령 및 status 값에 대한 정의

위의 도표에서 보여준 것 이외에도 보드의 종류에 따라서 다른 flash명령들이 존재하지만 그것은 여기서 생략하기로 하겠다. 직접 보고자 하는 사람은 flash.c 파일을 참조하기 바란다.

1.1.1.2. Help Command & Status Command

Help command는 BLOB를 사용하는 방법을 묻는 명령어이다. 단순히 PrintHelp() 함수만 호출해서 간략하게 사용하는 방법에 대한 것을 보여주는 역할을 할 뿐이다. 사용하는 방법만을 잠시 보기 위해서 여기에 인용해 보기로 한다.

```
void PrintHelp(void)
{
        SerialOutputString("Help for " PACKAGE " " VERSION ", the LART
bootloader \ n");
        SerialOutputString("The following commands are supported: \ n");
        SerialOutputString("* boot [kernel options]          Boot Linux with optional
kernel options \ n");
        SerialOutputString("* clock PPCR MDCNFG MDCAS0 MDCAS1 MDCAS2 \ n");
        SerialOutputString("                                 Set the SA1100 core clock
and DRAM timings \ n");
        SerialOutputString("                                 (WARNING: dangerous
command!) \ n");
```

```
        SerialOutputString("* download {blob|kernel|ramdisk}     Download blob/kernel/ramdisk image to RAM \n");
        SerialOutputString("* flash {blob|kernel|ramdisk}        Copy blob/kernel/ramdisk from RAM to flash\n");
        SerialOutputString("* help                               Get this help \n");
        SerialOutputString("* reblob                             Restart blob from RAM \n");
        SerialOutputString("* reboot                             Reboot system \n");
        SerialOutputString("* reload {blob|kernel|ramdisk}       Reload blob/kernel/ramdisk from flash to RAM \n");
        SerialOutputString("* reset                              Reset terminal \n");
        SerialOutputString("* speed                              Set download speed \n");
        SerialOutputString("* status                             Display current status \n");
}
```

코드 6. PrintHelp() 함수의 정의

즉, LART가 제공하는 BLOB에 대한 package정보와 버전 정보를 보여주며, 각각의 명령어를 어떻게 사용하는지와 이 명령어가 어떠한 역할을 하는 지를 보여주고 있다. 따로이 설명할 부분은 없다.

Status command 역시 현재의 BLOB의 상태를 보여주는 역할만 하고 있다. 즉, PrintStatus() 함수를 호출해서 화면에 display하는 일이다. 정의는 PrintHelp() 함수와 마찬가지로 main.c에 아래와 같이 되어 있다.

```
void PrintStatus(void)
{
      /* BLOB의 package정보와 버전 정보를 보여준다. */
        SerialOutputString("Bootloader    : " PACKAGE " \n");
        SerialOutputString("Version       : " VERSION " \n");
        SerialOutputString("Running from  : ");
      /* BLOB의 현재 실행 모드를 보여준다. */
        if(RunningFromInternal())
                SerialOutputString("internal");
        else
```

```
                SerialOutputString("external");
     /* Flash에 대한 상태 정보를 보여준다. */
       SerialOutputString(" flash\ nBlocksize      : 0x");
       SerialOutputHex(blob_status.blockSize);          /* Flash의 block size */
       SerialOutputString("\ nDownload speed: ");        /* Download speed */
       PrintSerialSpeed(blob_status.downloadSpeed);
       SerialOutputString(" baud \ n");
     /* 메모리에 있는 Blob의 상태를 보여준다. */
       SerialOutputString("Blob          : ");
       if(blob_status.blobType == fromFlash) {
                SerialOutputString("from flash\ n");
       } else {
                SerialOutputString("downloaded, ");
                SerialOutputDec(blob_status.blobSize);
                SerialOutputString(" bytes \ n");
       }
     /* 메모리에 있는 Kernel의 상태를 보여준다. */
       SerialOutputString("Kernel        : ");
       if(blob_status.kernelType == fromFlash) {
                SerialOutputString("from flash\ n");
       } else {
                SerialOutputString("downloaded, ");
                SerialOutputDec(blob_status.kernelSize);
                SerialOutputString(" bytes \ n");
       }
      /* 메모리에 있는 RAM disk의 상태를 보여준다. */
       SerialOutputString("Ramdisk       : ");
       if(blob_status.ramdiskType == fromFlash) {
                SerialOutputString("from flash\ n");
       } else {
                SerialOutputString("downloaded, ");
                SerialOutputDec(blob_status.ramdiskSize);
                SerialOutputString(" bytes \ n");
       }
}
```

코드 7. PrintStatus() 함수의 정의

PrintStatus() 함수에서 볼 수 있는 것은 blob_status 구조체가 중심이 되어, 현재의 모든 BLOB의 상태를 구성하고 있다는 점이며, 각각의 모듈(BLOB, kernel, RAM disk)별로 이러한 정보가 유지 된다는 점이다. 또한 이러한 정보 중에서 현재 메모리에 가지고 있는 image가 download된 것인지, 아니면 flash부터 읽어온 것인가도 보여주고 있다는 것이다.

1.1.1.3. Reblob Command

Reblob 명령은 RAM에 download하거나 flash로부터 가져온 BLOB을 새로이 시작하기 위한 것이다. 처리는 Reblob() 함수가 하고 있으며, 정의는 main.c에 아래와 같이 되어 있다.

```
void Reblob(void)
{
        void (*blob)(void) = (void (*)(void))BLOB_RAM_BASE;

        SerialOutputString("Restarting blob from RAM... \ n \ n");
        msleep(500);
        blob();
}
```

코드 8. Reblob() 함수의 정의

Reblob() 함수는 잠시간의 delay를 준다음, 메모리에 있는 BLOB의 이미지를 새로 수행하는 함수이다. 단지 download되거나 flash로 부터 read한 BLOB으로 jump만 한다. 여기서 사용되는 msleep() 함수는 time.c에 아래와 같이 정의되어 있다.

```
void msleep(unsigned int msec)
{
        u32 ticks, start, end;
        int will_overflow = 0;
        int has_overflow = 0;
        int reached = 0;

        if(msec == 0)
                return;
        ticks = (TICKS_PER_SECOND * msec) / 1000;
        start = TimerGetTime();
        /* this could overflow, but it nicely wraps around which is
       * exactly what we want
```

```
         */
        end = start + ticks;
        /* detect the overflow */
        if(end < start) {
                TimerClearOverflow();
                will_overflow = 1;
        }
        do {
                if(will_overflow && !has_overflow) {
                        if(TimerDetectOverflow())
                                has_overflow = 1;
                        continue;
                }
                if(TimerGetTime() >= end)
                        reached = 1;
        } while(!reached);
}
```

코드 9. msleep() 함수의 정의

msleep() 함수가 넘겨받는 값은 얼마동안 sleep을 할 것인가를 결정하는 값으로 millisecond단위 값이다. Millisecond단위로 tick의 개수를 계산해서 ticks에 둔다. 그리고, 시작 시간을 구해서(TimerGetTime()) start에 두도록 한다. 이젠 앞에서 계산된 시간과 현재 시간을 더해서 end로 만들고, 만약 end가 start보다 작다면 overflow가 있었다는 말이 되므로 overflow를 지우기 위해서 TimerClearOverflow() 함수를 호출한다. 또한 overflow 가 발생할 것이라는 것을 나타내기 위해서 will_overflow변수를 1로 설정한다. 나머지는 do{} while() loop를 돌면서 현재 시간이 앞에서 계산된 시간(end)보다 크거나 같을 때까지 수행하는 것이다. Loop내에서 overflow가 발생할지도 모르는 경우에는 TimerDetectOverflow()를 호출해서 실제로 overflow가 있었는지를 검출하게 되며, 이때 has_overflow를 1로 설정한다. 따라서, 이 loop에서 주어진 시간 만큼 대기하면서 수행을 delay시켜주는 것이다.

1.1.1.4. Reboot Command

SA1100의 Reset Controller Register들로는 RSRR(Reset Controller Software Reset Register)과 RCSR(Reset Controller Status Register)이 있다. RSRR은 최하위에 하나의 software reset bit을 가지고 있으며, 이 bit이 설정이 되면, SA1100을 reset시켜주는 역할을 한다. RCSR은 CPU에 의해서 reset의 마지막 원인을 결정하기 위해서 사용되는 값을 가

지는 레지스터로 크게 다음과 같은 reset 상황을 SA1100은 지원한다.

- Hardware reset : hardware적으로 reset이 걸렸다.
- Software reset : software적으로 reset이 걸렸다.
- Watchdog reset : watchdog reset이 걸렸다.
- Sleep mode reset : sleep mode reset이 걸렸다.

각각의 reset 상황에 대해서 하위에서 부터 hardware, software, watchdog, sleep mode의 순으로 한 bit씩 할당되어 있으며, reset후에는 hardware reset bit만이 1로 남으며, 나머지는 전부 0으로 clear된다. 이 register는 각각의 bit에 1을 write하는 것으로 bit을 clear할 수 있다. 각 bit에 있는 0이 의미하는 바는 해당하는 일이 일어나지 않았다는 것이며, 1은 해당하는 일이 발생했다는 말이다. 기타 다른 reserved된 bit은 의미가 없으며, 읽었을 때 0을 돌려줄 것이다.

```
void Reboot(void)
{
        SerialOutputString("Rebooting... \ n \ n");
        msleep(500);
        RCSR = 0;
        RSRR = 1;
}
```

코드 10. Reboot() 함수의 정의

Reboot command는 새로 booting을 시키는 명령이다. 잠시 시간을 delay시킨 후에 RCSR(=0x90030004 : Reset Controller Status Register)에는 0을 주고, RSRR(=0x90030000 : Reset Software Reset Register)에는 1을 써서, software reset을 걸어준다. 각각의 resigster에 대한 정의는 ~/include/asm-arm/arch-sa1100/SA-1100.h 있다.

1.1.1.5. Reload Command

Reload command는 앞에서 이미본 Reload() 함수를 수행한다. 넘겨받은 argument값에 따라서, BLOB, kernel, RAM disk를 새로 load하는 역할을 수행한다.

1.1.1.6. Reset Command

Reset command는 terminal의 speed를 원래의 상태로 복귀하는 명령이다. ResetTerminal() 함수를 호출해서 처리한다. 정의는 main.c에 있으며,

```
void ResetTerminal(void)
{
        int i;

        SerialInit(baud9k6);
        SerialOutputString("              c");
        for(i = 0; i < 100; i++)
                SerialOutputByte(' \ n');
        SerialOutputString("   c");
}
```

코드 11. ResetTerminal() 함수의 정의

ResetTerminal() 함수는 SerialInit()를 호출해서 9600bps로 serial의 baud rate를 맞춰주는 역할만 한다. 나머지는 일종의 debugging에 해당하는 문자열을 보여주는 것 뿐이다.

1.1.1.7. Speed Command

Speed command는 download 속도를 설정해 주는 함수이다. SetDownloadSpeed() 함수를 호출해서 처리해 준다. SetDownloadSpeed() 함수는 main.c에 아래와 같이 정의되어 있다.

```
void SetDownloadSpeed(char *commandline)
{
        if(MyStrNCmp(commandline, "1200", 4) == 0) {
                blob_status.downloadSpeed = baud1k2;
        } else if(MyStrNCmp(commandline, "1k2", 3) == 0) {
                blob_status.downloadSpeed = baud1k2;
        } else if(MyStrNCmp(commandline, "9600", 4) == 0) {
                blob_status.downloadSpeed = baud9k6;
        } else if(MyStrNCmp(commandline, "9k6", 3) == 0) {
                blob_status.downloadSpeed = baud9k6;
        } else if(MyStrNCmp(commandline, "19200", 5) == 0) {
                blob_status.downloadSpeed = baud19k2;
        } else if(MyStrNCmp(commandline, "19k2", 4) == 0) {
                blob_status.downloadSpeed = baud19k2;
        } else if(MyStrNCmp(commandline, "38400", 5) == 0) {
                blob_status.downloadSpeed = baud38k4;
```

```
	} else if(MyStrNCmp(commandline, "38k4", 4) == 0) {
		blob_status.downloadSpeed = baud38k4;
	} else if(MyStrNCmp(commandline, "57600", 5) == 0) {
		blob_status.downloadSpeed = baud57k6;
	} else if(MyStrNCmp(commandline, "57k6", 4) == 0) {
		blob_status.downloadSpeed = baud57k6;
	} else if(MyStrNCmp(commandline, "115200", 6) == 0) {
		blob_status.downloadSpeed = baud115k2;
	} else if(MyStrNCmp(commandline, "115k2", 5) == 0) {
		blob_status.downloadSpeed = baud115k2;
	} else {
		SerialOutputString("*** Invalid download speed value \"");
		SerialOutputString(commandline);
		SerialOutputString("\"\n*** Valid values are:\n");
		SerialOutputString("*** 1200, 9600, 19200, 38400, 57600, 115200,\n");
		SerialOutputString("*** 1k2, 9k6, 19k2, 38k4, 57k6, and 115k2\n");
	}
	SerialOutputString("Download speed set to ");
	PrintSerialSpeed(blob_status.downloadSpeed);
	SerialOutputString(" baud\n");
}
```

코드 12. SetDownloadSpeed() 함수의 정의

넘겨받은 문자열을 비교해서(MyStrNCmp()) 이 값에 따라 blob_status.downloadSpeed를 결정해 준다. 들어갈 수 있는 download speed로는 1200, 9600, 19200, 38400, 57600, 115200 bps가 있으며, 해당하는 값이 없다면 에러 메시지를 보여줄 것이다. 또한 조정된 download speed가 얼마인지도 보여준다. 여기서 설정한 download speed는 나중에 download를 하게 될 때 사용될 것이므로, 실제적인 hardware의 설정은 이곳에서 찾을 수 없다. 단지 blob_status 구조체의 downloadSpeed 필드를 갱신하는 일만 해준다.
여기서 사용되는 PrintSerialSpeed() 함수도 main.c에 아래와 같은 정의를 가지고 있으며, 단순히 현재 설정한 serial의 속도를 보여주는 일만 한다.

```
void PrintSerialSpeed(eBauds speed)
{
	switch(speed) {
        case baud1k2:
                SerialOutputDec(1200);
                break;
        case baud9k6:
                SerialOutputDec(9600);
                break;
        case baud19k2:
                SerialOutputDec(19200);
                break;
        case baud38k4:
                SerialOutputDec(38400);
                break;
        case baud57k6:
                SerialOutputDec(57600);
                break;
        case baud115k2:
                SerialOutputDec(115200);
                break;
        default:
                SerialOutputString("(unknown)");
                break;
        }
}
```

코드 13. **PrintSerialSpeed()** 함수의 정의

넘겨받은 serial의 속도에 따라서 switch문을 통해 해당하는 speed별로 SerialOutputDec() 함수를 호출해서 화면에 나타내 준다.

이상에서 우린 SA1100의 boot loader로 사용된 BLOB에 대한 전체 과정을 살펴보았다. 즉, blob을 통해서 커널을 loading할 준비가 다 된 것이다. 커널은 flash에 저장되어 있거나, 혹은 serial을 통해서 downloading되어야 하며, 사용하게 될 RAM disk도 같은 방법을 통해서 사용할 수 있을 것이다. 초기에 BLOB를 flash에 쓰는 것은 Jflash와 같은 것을 사용하면 될 것이다. 이와 관련된 것은 여기서는 더 이상 논의 하지 않을 것이며, 이젠 커널로 들어가서 어떤 과정들이 일어나는 지를 보도록 하자.