1.1.1. main.c의 분석

main.c 파일이 BLOB의 핵심 routine이 들어있는 부분이다. 이곳에서 BLOB가 하는 일은 다음과 같이 요약해 볼 수 있다. 먼저, serial을 초기화 하고, timer를 동작시키며, 시스템의 메모로 map을 구한 후, BLOB의 상태를 초기화 한다. 이것을 마치고나면, 커널과 RAM disk를 Flash로 부터 읽어서 RAM으로 가져다 두게되고, 사용자의 입력을 기다리게 된다. main.c에 있는 entry point가 되는 routine은 main() 함수이다.

```
int main(void)
{
        u32 blockSize = 0x00800000;
        int numRead = 0;
        char commandline[128];
        int i;
        int retval = 0;

        /* Turn the LED on again, so we can see that we safely made it
       * into C code.
        */
        led_on();

        /* We really want to be able to communicate, so initialise the
         * serial port at 9k6 (which works good for terminals)
         */
        SerialInit(baud9k6);
        TimerInit();

        /* Print the required GPL string */
        SerialOutputString("\nConsider yourself LARTed!\n\n");
        SerialOutputString(PACKAGE " version " VERSION "\n"
                           "Copyright (C) 1999 2000 2001 "
                           "Jan-Derk Bakker and Erik Mouw\n"
                           "Copyright (C) 2000 "
                           "Johan Pouwelse\n");
        SerialOutputString(PACKAGE " comes with ABSOLUTELY NO WARRANTY; "
                           "read the GNU GPL for details.\n");
```

```
        SerialOutputString("This is free software, and you are welcome "
                           "to redistribute it \n");
        SerialOutputString("under certain conditions; "
                           "read the GNU GPL for details. \n");
```

코드 1. main.c 파일

먼저, main() 함수는 LED를 켜기 위해서 led_on()함수를 호출한다. 또한 serial과 timer를 각각 초기화 하기 위해서 SerialInit()와 TimerInit()함수를 호출한다. SerialInit() 함수를 호출할 때 넘겨주는 값은 baud9k6(=23)으로 serial.h에 아래와 같은 정의를 가진다.

```
typedef enum { /* Some useful SA-1100 baud rates */
        baud1k2 = 191,
        baud9k6 = 23,
        baud19k2 = 11,
        baud38k4 = 5,
        baud57k6 = 3,
        baud115k2 = 1
} eBauds;
```

코드 2. SA1100의 Serial을 위한 baud rate의 정의

위와 같이 정의된 값은 아래의 공식을 이용해서 baud rate를 정하는데 사용되는 값이다. 정의된 값은 BRD에 해당한다.

$$BaudRate = \frac{3.6864x10^6}{16\times(BRD+1)}$$

이 공식에서 사용될 BRD 값은 나중에 사용되는 serial port의 UART control register에 들어가서, serial의 baud rate를 결정한다. 여기서 잠시 serial의 초기화를 보고 넘어가도록 하자. 코드는 serial.c에 아래와 같다.

```
void SerialInit(eBauds baudrate)
{
#if defined USE_SERIAL1
        while(Ser1UTSR1 & UTSR1_TBY) {
        }
        Ser1UTCR3 = 0x00;
        Ser1UTSR0 = 0xff;
```

```
        Ser1UTCR0 = ( UTCR0_1StpBit | UTCR0_8BitData );
        Ser1UTCR1 = 0;
        Ser1UTCR2 = (u32)baudrate;
        Ser1UTCR3 = ( UTCR3_RXE | UTCR3_TXE );
#elif defined USE_SERIAL3
        while(Ser3UTSR1 & UTSR1_TBY) {
        }
        Ser3UTCR3 = 0x00;
        Ser3UTSR0 = 0xff;
        Ser3UTCR0 = ( UTCR0_1StpBit | UTCR0_8BitData );
        Ser3UTCR1 = 0;
        Ser3UTCR2 = (u32)baudrate;
        Ser3UTCR3 = ( UTCR3_RXE | UTCR3_TXE );
#else
#error "Configuration error: No serial port used at all!"
#endif
}
```

코드 3. SerialInit() 함수의 정의

즉, serial port 1을 사용할지 아니면, serial port 3을 사용할지에 따라서, 각각이 사용하는 register set이 달라지지만 근본적으로 동일한 register set을 serial port 1과 3이 같이 가지고 있으므로, 들어가는 내용은 동일하다. 먼저, 각각의 seial port의 상태가 busy이라면, 상태가 바뀔 때까지 기다린다. 그리고나서, UTCR3(UART Control Register 3)에 0x00을 써서 Rx와 Tx를 OFF시킨후, UTSR0(UART Status Register 0)에 0xFF를 써서 지우도록(clear) 한다. 다시 UTCR0에 1 stop bit과 8 bit data로 설정을 변경한 후, UTCR1에는 0을 UTCR2에는 전달받은 baud rate 정보를 기입해서 어떤 baud rate으로 동작할 지를 결정해 준다. 이것을 다 마치면, 이젠 UTCR3에 Rx Enable가 Tx Enable bit을 설정(set)해서 Rx와 Tx가 일어나도록 만들어준다. 여기서 UTRC1은 앞의 공식에서 사용한 BRD의 상위 4 bit을, UTCR2는 하위 8 bit을 가지도록 만들어 주는 것이다[1].

TimerInit() 함수의 정의는 time.c에 있으며, 아래와 같이 되어 있다. 이곳에서 하는 일은 시스템에 있는 timer interrupt의 발생빈도를 설정하는 것이다.

---

[1] 여기서 정의된 register들에 대한 것은 ~/include/asm-arm/arch-sa1100/SA-1100.h에 그 정의가 나온다. 따라서, BLOB를 compile하기 위해서는 적절한 SA1100의 patch가 된 커널의 source를 가지고 있어야 할 것이다. 이하 부분에서 나오는 각종 레지스터의 정의도 마찬가지다.

```
void TimerInit(void)
{
        /* clear counter */
        OSCR = 0;
        /* we don't want to be interrupted */
        OIER = 0;
        /* wait until OSCR > 0 */
        while(OSCR == 0)
                ;
        /* clear match register 0 */
        OSMR0 = 0;
        /* clear match bit for OSMR0 */
        OSSR = OSSR_M0;
        numOverflows = 0;
}
```

코드 4. TimerInit() 함수의 정의

SA1100에는 크게 OSCR(Operating System Counter Register)라는 up -counter register와 4개의 OSMR(Operating System Match Register)가 존재한다. OSCR은 reset의 영향을 받지 않으며, OSMR은 사용자가 read와 write를 할 수 있다. 만약 OSCR이 OSMR의 특정 레지스터와 일치하고, interrrupt enable bit이 설정된 경우에는 OSSR(Operating System Status Register)에 해당 bit이 설정되게 되며, 이러한 bit은 interrupt controller bit에 route되어 인터럽트를 발생시키도록 할 수 있다. 또한 OSMR3는 watchdog match register의 역할을 수행할 수 있도록 되어 있어서, 만약 OSCR이 OSMR3와 같아진다면, SA1100을 reset시킬 수 있다. Reset이후에 known state로 갈 수 있는 관련된 레지스터로는 또한 WMER(Watchdog Match Enable Register)가 있으며, 사용자는 FIQ(Fast Interrupt)와 IRQ(Interrupt) interrupt를 CPU에 enable 시키기 전에, 레지스터들을 초기화 시켜야 하며, status register는 clear시켜주어야 한다.
위에서는 OSCR을 먼저 0으로 지우고(clear), 다시 OIER(Operating System Timer Enable Register)를 다시 0으로 지웠다. 이것으로 counter와 interrupt를 다 clear 시켜주었다. 이젠 OSCR에 countering이 일어날 때까지 기다렸다가, OSMR0를 다시 0으로 만들어 주었으며, OSSR의 OSMR0와 일치하는 bit을 clear 시켜주었다. 즉, OSSR의 특정 bit에 1을 write하는 것으로 clear 시킨다. 또한 numOverflows는 Overflow가 일어난 회수를 가지는 변수로 0으로 초기화 시킨다. 이것을 timer에 대한 초기화를 마친다.

나머지 BLOB의 main() 함수에 대한 것은 serial을 통해서 string을 쓰는 것으로,

SerialOutputString() 함수[2]를 호출해서 처리하고 있다. 앞으로도 이 함수를 많이 사용하게 될 것이므로 여기서 보도록 하자. 정의된 곳은 serial.c 파일이며 아래와 같다.

```
/*
 * Output a single byte to the serial port.
 */
void SerialOutputByte(const char c)
{
#if defined USE_SERIAL1
        /* wait for room in the tx FIFO */
        while((Ser1UTSR0 & UTSR0_TFS) == 0) ;
        Ser1UTDR = c;
#elif defined USE_SERIAL3
        /* wait for room in the tx FIFO */
        while((Ser3UTSR0 & UTSR0_TFS) == 0) ;
        Ser3UTDR = c;
#else
#error "Configuration error: No serial port used at all!"
#endif
        /* If \n, also do \r */
        if(c == '\n')
                SerialOutputByte('\r');
}
/*
 * Write a null terminated string to the serial port.
 */
void SerialOutputString(const char *s) {
        while(*s != 0)
                SerialOutputByte(*s++);

} /* SerialOutputString */
```

코드 5. SerialOutputString() 함수의 정의

SerialOutputString()은 단순히 character string을 입력으로 받아서, 이것을 한 byte단위로

[2] 앞에서 나온 PACKAGE와 VERSION과 같은 문자열을 나타내는 macro는 BLOB를 compile하는 과정에서 생성된다.

SerialOutputByte() 함수를 호출해서 출력하는 일을 한다. 실제적인 출력은 SerialOutputByte()에서 일어난다. SerialOutputByte() 함수는 사용하게되는 serial port에 따라 Ser1UTDR과 Ser3UTDR 레지스터에 하나의 character를 적어넣는 일을한다. 먼저 쓰기전에 UTSR 레지스터를 읽어서, Tx FIFO가 full인지 아닌지를 검증하게 된다. 만약 Tx FIFO아 half-full이하이게 되면, UTDR레지스터에 데이터를 쓴다. 만약 쓰려고한 데이터가 "\n" character일 경우에는 "\r"을 추가해서 line feed가 되도록 만든다. 보시다시피 이 함수는 재귀적인(recursive)한 호출로 사용되고 있음을 알 수 있을 것이다. 이젠 계속 main() 함수를 보도록 하자.

```
        /* get the amount of memory */
        get_memory_map();

        /* initialise status */
        blob_status.kernelSize = 0;                    /* 커널의 크기를 0으로 초기화 시
킨다. */
        blob_status.kernelType = fromFlash;            /* fromFlash로 kernelType을
초기화 시킨다. */
        blob_status.ramdiskSize = 0;                   /* RAM disk의 크기를 0으로 초기
화 시킨다. */
        blob_status.ramdiskType = fromFlash;           /* fromFlash로 ramdiskType을
초기화 시킨다. */
        blob_status.blockSize  =  blockSize;                        /*  blockSize필드를
blockSize(=0x00800000)으로 둔다.*/
        blob_status.downloadSpeed = baud115k2;         /* Baud rate을 11500BPS로 설
정한다. */

        /* Load kernel and ramdisk from flash to RAM */
        Reload("blob");
        Reload("kernel");
        Reload("ramdisk");

#ifdef BLOB_DEBUG
        /* print some information */
        SerialOutputString("Running from ");
        if(RunningFromInternal())
                SerialOutputString("internal");
```

```
        else
                SerialOutputString("external");
        SerialOutputString(" flash, blockSize = 0x");
        SerialOutputHex(blockSize);
        SerialOutputByte(' \ n');
#endif
```

코드 6. main.c 파일(계속)

get_memory_map() 함수는 시스템에 있는 메모리의 양을 구하는 함수이다. 즉, 이후에 커널 및 RAM disk를 load하기 위해서, 현재 시스템에 얼마나 많은 메모리가 있는지를 알아야 할 것이다. 또한 blob_status 변수에 대한 초기화도 이곳에서 일어나게 된다. 이 변수는 main.h에 있는 blob_status_t 구조체로 정의되며, 아래와 같다.

```
typedef struct {
        int kernelSize;                 /* 커널의 크기 */
        block_source_t kernelType;          /* 커널을 어디서 가져오는 지를 정의한다.
*/
        int ramdiskSize;                /* RAM disk의 크기 */
        block_source_t ramdiskType;     /* RAM disk를 어디서 가져오는 지를 정의한
다. */
        int blobSize;                   /* BLOB 자체의 크기 */
        block_source_t blobType;        /* BLOB 자체를 어디서 가져오는 지를 정의
한다. */
        u32 blockSize;                  /* 하나의 Block의 크기를 정의한다. */
        eBauds downloadSpeed;           /* Downloading Speed를 정의한다. */
} blob_status_t;
```

코드 7. blob_status_t 구조체의 정의

이 구조체에서 사용하고 있는 block_source_t 구조체와 eBauds는 다시 아래와 같이 main.h와 앞에서 본 serial.h에 정의되어 있다.

```
typedef enum {
        fromFlash = 0,
        fromDownload = 1
} block_source_t;
```

코드 8. block_source_t 구조체의 정의

block_source_t 구조체에는 두개의 필드가 있으며, 어디서 image를 가져오는 지를 정하고 있다. 각각을 enumeration으로 설정해서 fromFlash와 fromDownload에 0과 1을 주고 있다.
앞에서 blob_status_t 구조체를 초기화 시키는 것은 이미 보았으며, 이것을 마치고나면, Reload() 함수를 호출해서 blob와 kenrel 및 ramdisk를 reload한다. BLOB_DEBUG이 선언된 경우에는 debugging을 위한 메시지를 화면에 나타내기 위해서 SerialOutputXXX()라는 함수를 사용하게 되는데, 그 중간에 BLOB가 internal이냐 external이냐를 가리기 위해서 RunningFromInternal()이란 inline 함수를 사용하게 된다. 이 inline 함수는 flash.h에 아래와 같은 정의를 가진다.

```
static inline int RunningFromInternal(void) {

        if(((*(u32 *)0xA0000010) & 0x04) == 0)
                return 1;
        else
                return 0;
}
```

코드 9. RunningFromInternal() inline 함수의 정의

이 함수는 0xA0000010에 있는 값을 0x04와 AND시켜서 이 값이 0이면 1을 돌려주고, 그렇지 않으면 0을 돌려주는 함수이다. 실제 0xA0000010은 Static Memory Control Register(MSC0)을 나타내는 레지스터로 이곳의 RBw0에 있는 bit과 0x04가 AND된다. 즉, 이 bit은 ROM의 bus width를 나타내는 값으로 1인 경우에는 16 bit을, 0인 경우에는 32 bit bus width를 가진다는 뜻이다. Reset 이후에는 이 값은 ROM_SEL pin의 inverse값을 가지도록 설정되며, 즉, 현재 ROM에서 실행되고 있는지 아니면, RAM에서 수행되고 있는지를 알려준다.
더 진행하기에 전에, 앞에서본 get_memory_map() 함수와 Reload() 함수를 보도록 하자. 각각의 함수는 memory.c와 main.c에 정의되어 있다.

```
void get_memory_map(void)
{
        u32 addr;
        int i;

        /* init */
        for(i = 0; i < NUM_MEM_AREAS; i++)
```

```
                memory_map[i].used = 0;
        /* first write a 0 to all memory locations */
        for(addr = MEMORY_START; addr < MEMORY_END; addr += TEST_BLOCK_SIZE)
                * (u32 *)addr = 0;
        /* scan memory in blocks */
        i = 0;
        for(addr = MEMORY_START; addr < MEMORY_END; addr += TEST_BLOCK_SIZE) {
                if(testram(addr) == 0) {
                        /* yes, memory */
                        if(* (u32 *)addr != 0) { /* alias? */
#ifdef BLOB_DEBUG
        /* Debugging을 위한 목적으로 serial을 통해서 output을 내보낸다. */
#endif
                                if(memory_map[i].used)
                                        i++;
                                continue;
                        }
                        /* not an alias, write the current address */
                        * (u32 *)addr = addr;
#ifdef BLOB_DEBUG
        /* Debugging을 위한 목적으로 serial을 통해서 output을 내보낸다. */
#endif
                        /* does this start a new block? */
                        if(memory_map[i].used == 0) {
                                memory_map[i].start = addr;
                                memory_map[i].len = TEST_BLOCK_SIZE;
                                memory_map[i].used = 1;
                        } else {
                                memory_map[i].len += TEST_BLOCK_SIZE;
                        }
                } else {
                        /* no memory here */
                        if(memory_map[i].used == 1)
                                i++;
```

```
                }
        }
```

코드 10. get_memory_map() 함수의 정의

먼저 사용하게 될 메모리의 메모리 맵(map)을 나타내는 memory_map[]변수의 used 필드를 전부 0으로 clear시킨다. NUM_MEM_AREAS는 전체 메모리 맵의 영역이 얼마나 있는가를 가르키는 상수 값으로 32를 가지며, memory.h에 정의되어 있다. 이 32라는 숫자는 alias로 연결되어 있는 부분을 찾는데 충분한 값으로 설정되어야 한다. memory_map[] 변수는 아래와 같은 memory_area_t 구조체로 정의된다. 이젠 이렇게 적어놓은 메모리에 대해서 다시 loop를 돌면서 실제로 메모리가 있는지를 확인하는 작업이 되며, 앞에서 본 것과 유사한 testram() 함수가 사용된다. 단지 앞에서 본 함수와 차이가 나는 점은 이 함수는 C에서 호출이 되는 함수이가 때문에, 사용하는 레지스터들에 대해서 save하는 것이 따르게 된다. return값은 앞에서와 마친가지로 0인 경우에는 존재한다는 것을 나타내며, 그렇지 않은 경우에는 1이 돌려받게 된다. 정의되 곳은 testmem2.S 파일이다. 함수의 앞과 뒤 부분을 유심히 보도록 하자.[3]

```
typedef struct {
        u32 start;              /* 메모리 영역의 시작 주소 */
        u32 len;                /* 메모리 영역의 길이 */
        int used;               /* 사용되었는 지를 나타내는 flag */
} memory_area_t;
```

코드 11. memory_area_t 구조체의 정의

메모리를 1MBytes 단위(=TEST_BLOCK_SIZE)씩 증가 시키면서 각 메모리 block의 시작 위치에 loop를 돌면서 0을 쓰도록 한다. 이때 메모리의 시작(MEMORY_START)와 메모리의 마지막 주소(MEMORY_END)는 각각 0xC0000000과 0xE0000000이다. 이 영역에 있는 메모리는 SA1100의 메뉴얼에서 찾을 수 있듯이 DRAM BANK 0, 1, 3, 4가 된다.
testram() 함수의 호출 결과가 0이 아니라면, RAM이 존재하지 않는다는 말이되므로, memory_map[]에 해당 block의 used 필드가 1인 경우에만 블록의 갯수를 나타내는 변수인 i를 증가시켜주고, 그렇지 않다면 다음 loop로 진행한다. 만약, 메모리가 있지만 앞에서 설정한 block의 시작 주소에 쓴 내용이 0과 같이 않다면, 이 메모리가 alias되어서 사용되고 있다는 것을 의미할 수도 있다. 이때는 memory_map[]의 해당 block에 대한 used 필드를 봐서 1로 설정된 경우에만 블록의 수를 증가 시켜주기 위해서 i를 증가 시킨다. 그렇지 않다면, 현재의 주소 값(addr)을 그대로 addr변수가 가르키는 곳에 적어주어서, alias를 판

[3] 여기서는 추가적인 설명을 덧붙이지 않겠다. 앞에서 본 testram()함수를 참고하기 바란다.

단할 있는 근거를 만들어준다.
만약, 현재의 block이 사용중이지 않다면(memory_map[i].used == 0), memory_map[]의 element를 설정한다. 즉, 시작 주소에는 addr을 기입하고, 크기는 1 Mbytes(=TEST_BLOCK_SIZE), used 필드에는 1로 준다. 사용중이라면 (memory_map[i].used == 1), 단지 크기만을 증가시켜준다(TEST_BLOCK_SIZE를 더해줌). 즉, 이 block은 새로운 block이 아니며, 원래 있던 block의 일부로 생각해 준다는 것이다.

```
        SerialOutputString("Memory map: \ n");
        for(i = 0; i < NUM_MEM_AREAS; i++) {
                if(memory_map[i].used) {
                        SerialOutputString("  0x");
                        SerialOutputHex(memory_map[i].len);
                        SerialOutputString(" @ 0x");
                        SerialOutputHex(memory_map[i].start);
                        SerialOutputString(" (");
                        SerialOutputDec(memory_map[i].len / (1024 * 1024));
                        SerialOutputString(" MB) \ n");
                }
        }
}
```

코드 12. get_memory_map() 함수의 정의(계속)

이젠 debugging과 각종 정보를 사용자에게 보여줄 차례이다. 각각의 memory_map[] element에 대해서 loop를 돌면서 메모리의 ㅣ작과 길이, 그리고, 크기를 써주게 된다. 이때 이용되는 함수가 SerialOutputHex()와 SerialOutputDec()이다. 각각 아래와 같이 serial.c에 정의되어 있다.

```
void SerialOutputHex(const u32 h)
{
        char c;
        int i;

        for(i = NIBBLES_PER_WORD - 1; i >= 0; i--) {       /* NIBBLES_PER_WORD =
8 */
                c = (char)((h >> (i * 4)) & 0x0f);
                if(c > 9)
```

```
                        c += ('A' - 10);
                else
                        c += '0';
                SerialOutputByte(c);
        }
}
void SerialOutputDec(const u32 d)
{
        int leading_zero = 1;
        u32 divisor, result, remainder;

        remainder = d;
        for(divisor = 1000000000;
            divisor > 0;
            divisor /= 10) {
                result = remainder / divisor;
                remainder %= divisor;
                if(result != 0 || divisor == 1)
                        leading_zero = 0;
                if(leading_zero == 0)
                        SerialOutputByte((char)(result) + '0');
        }
}
```

코드 13. SerialOutputHex()와 SerialOutputDec() 함수의 정의

각각의 함수는 넘겨받은 32 bit크기의 변수를 hexadecimal 값과 decimal 값으로 serial을 통해서 내보내는 일을 하며, 결과적으로 앞에서 본 SerialOutputByte() 함수를 사용해서 한 byte단위로 처리한다. 이것은 단순히 data를 알아보기 쉽게 conversion하는 함수이다.

```
void Reload(char *commandline)
{
        u32 *src = 0;
        u32 *dst = 0;
        int numWords;

        if(MyStrNCmp(commandline, "blob", 4) == 0) {
```

```
                src = (u32 *)BLOB_RAM_BASE;
                dst = (u32 *)BLOB_START;
                numWords = BLOB_LEN / 4;
                blob_status.blobSize = 0;
                blob_status.blobType = fromFlash;
                SerialOutputString("Loading blob from flash ");
        } else if(MyStrNCmp(commandline, "kernel", 6) == 0) {
                src = (u32 *)KERNEL_RAM_BASE;
                dst = (u32 *)KERNEL_START;
                numWords = KERNEL_LEN / 4;
                blob_status.kernelSize = 0;
                blob_status.kernelType = fromFlash;
                SerialOutputString("Loading kernel from flash ");
        } else if(MyStrNCmp(commandline, "ramdisk", 7) == 0) {
                src = (u32 *)RAMDISK_RAM_BASE;
                dst = (u32 *)INITRD_START;
                numWords = INITRD_LEN / 4;
                blob_status.ramdiskSize = 0;
                blob_status.ramdiskType = fromFlash;
                SerialOutputString("Loading ramdisk from flash ");
        } else {
                SerialOutputString("*** Don't know how to reload \ "");
                SerialOutputString(commandline);
                SerialOutputString("\ " \ n");
                return;
        }
        MyMemCpy(src, dst, numWords);
        SerialOutputString(" done \ n");
}
```

코드 14. Reload() 함수의 정의

Reload() 함수는 commandline이라는 변수로 string을 넘겨받는다. 이 값을 MyStrCmp()에 해당하는 것이 있는지를 알 수 있도록, blob나 kernel, ramdisk와 같은 문자열과, 비교하는 문자열의 길이를 나타내는 것을 같이 준다. MyStrCmp() 함수는 넘겨받은 문자열을 비교해서 같은지 다른지 만을 보게된다. 아래와 같은 정의를 가진다. MyStrCmp()의 정의는 util.c에 있다. 이 파일에서 MyMemCpy() 함수도 같이 보기로 하자.

```
int MyStrNCmp(const char *s1, const char *s2, int maxlen)
{
	int i;

	for(i = 0; i < maxlen; i++) {
		if(s1[i] != s2[i])
			return ((int) s1[i]) - ((int) s2[i]);
		if(s1[i] == 0)
			return 0;
	}
	return 0;
} /* MyStrNCmp */
```

코드 15. MyStrCmp() 함수의 정의

MyStrNCmp() 함수는 단지 넘겨받은 두개의 문자열이 같은지 만을 보는 함수이다. 넘겨받은 문자열의 길이 만큼을 비교한다. 같다면 0을 돌려줄 것이며, 다르다면, 첫번째 문자열에서 두번째 문자열의 다른 부분에 대한 string index를 뺀 값을 돌려줄 것이다.

```
void MyMemCpy(u32 *dest, const u32 *src, int numWords)
{
#ifdef BLOB_DEBUG
	SerialOutputString("\n### Now copying 0x");
	SerialOutputHex(numWords);
	SerialOutputString(" words from 0x");
	SerialOutputHex((int)src);
	SerialOutputString(" to 0x");
	SerialOutputHex((int)dest);
	SerialOutputByte('\n');
#endif
	while(numWords--) {
		if((numWords & 0xffff) == 0x0)
			SerialOutputByte('.');
		*dest++ = *src++;
	}
#ifdef BLOB_DEBUG
```

```
        SerialOutputString(" done \ n");
#endif
} /* MyMemCpy */
```

코드 16. MyMemCpy() 함수의 정의

MyMemCpy() 함수는 목적지(dest)와 원래의 주소(src) 및 복사할 크기를 넘겨받는 함수로 단순히 한번에 32 bit 만큼씩을 loop를 돌면서 복사(copy)하는 일을 해준다. 이때, 복사하려는 크기가 0xFFFF와 AND시켜서 0인 경우에는 "."를 serial로 출력해서 얼마나 복사가 진행되고 있는지를 나타낸다.

앞에서 하던 이야기로 돌아가서, 이젠 각 if 절에 있는 MyStrNComp()를 보도록 하자. 가장 먼저 비교하는 것은 "blob"이다. 비교해서 같다면, src와 dst에 각각 BLOB_RAM_BASE(=0xC1000000)와 BLOB_START(=0x00000000)을 가져온다.[4] 또한 numWords에는 BLOB_LEN(=0x10000 or 64K)을 4로 나눈 값을 넣고, blob_status.blobSize에 0을 blobType에는 fromFlash(=0 or download from flash)를 두도록 한다. 그리고나서, 이제 이것을 loading한다는 것을 보여주기 위해서 serial에 "loading blob from flash"를 쓴다. 나머지는 MyMemCpy()에서 dst에 src가 가르키는 메모리를 copy해 줄 것이다.
commandline 변수에 "kernel"을 받았을 경우에는 src에 KERNEL_RAM_BASE(=0xC0008000)[5]을 두고, dst에는 KERNEL_START(=0x10000 or 64K)를 준다. numWords에는 커널의 길이를 나타내는 KERNEL_LEN(=0xC0000 or 3 x 256K)를 4로 나눈 값을 주어 옮길 값이 4 byte단위로 얼마나 되는 지를 표시하며, blob_status.kenelSize에는 0을 blob_status.kernelType에는 fromFlash를 두어서 커널을 flash로부터 loading함을 표시한다. 마지막으로 커널을 loading한다는 것을 나타내기 위해서 SerialOutputString()을 써서 serial로 output을 내보낸다. 역시 실제적인 copy는 MyMemCpy()가 하게 된다.

---

[4] 이것은 Assabet의 경우를 가정해서 이렇게 해주었다. 자세한 것은 flash.h를 참조하기 바란다.
[5] 이 값은 나중에 커널을 compile할 때 적재할 주소를 알려주는 부분에서 0xC0008000으로 설정해 주기 때문이다.