1.1.1. BLOB의 command 분석

여기서 부터는 main()함수에서 BLOB가 command를 읽었을 때, 어떤 일이 일어나는 지를 분석할 것이다. 앞에서 이미 보았듯이 분석할 command가 되는 것은 boot, clock, download, flash, help, reblob, reboot, reload, reset, speed, status등이다. 각각의 command를 차례로 볼 것이다.

먼저 간략하게 BLOB의 command가 어떤 역할을 하는지를 source로 들어가기 전에 보도록 하자. 여기서 command를 입력하기 위해서는 10초정도의 대기 시간내에 해야 할 것이다. 그렇지 않다면, 바로 Linux 커널을 booting시키는 곳으로 진행할 것이다.

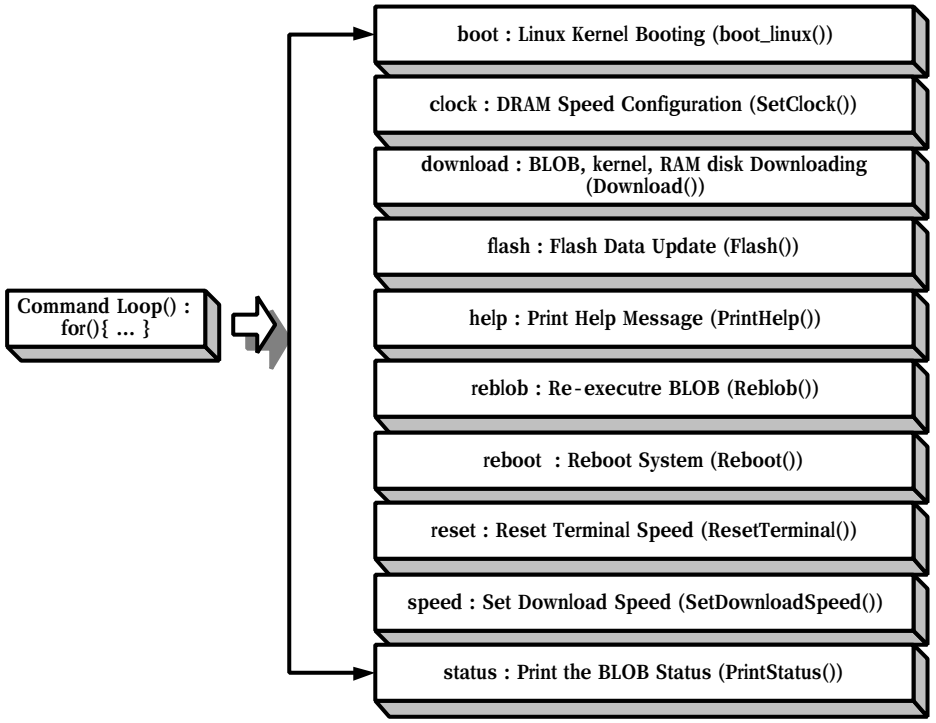

그림 1. BLOB의 Command Loop

[그림 1]은 BLOB에서 사용하고 있는 command와 그것이 해주는 일 및 처리하는 함수를 보여준다. 그림에서 알 수 있듯이, 사용되는 command는 boot, clock, download, flash, help, reblob, reboot, reset, speed, status가 있으며, 각각은 BLOB의 상태를 바꾸는 일을 한다. 즉, BLOB와 커널, RAM disk등을 downloading을 한다던가, 혹은 reboot 및 RAM 설정을 바꾸는등의 일이 될 것이다. 이러한 command loop를 통해서 사용자는 serial을 이용해 BLOB의 동작을 조정해 줄 수 있다. 자, 이제 그럼 위에서 부터 순서대로 실제적인 코드를 보기로 하자.

1.1.1.1. Boot Command

Boot command는 앞에서 이미 보았다. 즉, boot 명령을 받았을 때 boot_linux() 함수를 호출해서, 현재까지 가지고 있는 commandline에 4를 더해서 넘겨주는 일을 한다. 즉, 이것은 boot 명령어 다음에 커널의 command line에 넘겨줄 데이터를 명시할 수 있다는 의미이다. 관련된 것은 앞에서 이미 설명을 다 했으므로 다음으로 넘어가도록 하자.

1.1.1.2. Clock Command

Clock command 역시 clock 이라는 명령어와 같이 option과 같은 것을 명시해서, 이를 SetClock() 함수로 전달할 수 있도록 하고 있다. SetClock() 함수는 DRAM의 파라미터를 설정하거나, speed를 조정하기 위해서 사용하는 함수로서 clock.c에 아래와 같이 정의되어 있다.

```
/* Struct with the SA-1100 PLL + DRAM parameter registers */
enum {
        SA_PPCR,
        SA_MDCNFG,
        SA_MDCAS0,
        SA_MDCAS1,
        SA_MDCAS2
};
...
void SetClock(char *commandline)
{
int i;
u32 regs[5];
u32 startTime, currentTime;

  for(i = 0; i < 5; i++) {
    commandline = GetHexValue(commandline, &regs[i]);
#warning "FIXME: GetHexValue() returns NULL after the fifth call..."
    if((commandline == NULL) && (i != 4)) {
      SerialOutputString(__FUNCTION__ ": can't get hex values \n");
       return;
     }
   }
  /* we go slower, so first set PLL register */
```

```
  PPCR = regs[SA_PPCR];
  MDCNFG = regs[SA_MDCNFG];
  MDCAS0 = regs[SA_MDCAS0];
  MDCAS1 = regs[SA_MDCAS1];
  MDCAS2 = regs[SA_MDCAS2];
  /* sleep for a second */
  startTime = TimerGetTime();
  for(;;) {
    currentTime = TimerGetTime();
    if((currentTime - startTime) > (u32)TICKS_PER_SECOND)
      return;
   }
} /* SetClock */
```

코드 1. SetClock() 함수의 정의

SetClock() 함수는 먼저 받은 commandline을 가지고 GetHexValue() 함수를 호출한다. Command line이 변환된 값은 regs[] 배열에 각각의 register에 따라 채워지게 된다. GetHexValue() 함수의 return값이 NULL이고, 다섯번의 loop를 다 돌지 못했다면, 넘겨받은 commandline을 제대로 다 해석하지 못했다는 것을 의미하므로, SerialOutputString()을 호출해서 error가 생겼다는 것을 나타낸 후, 바로 return한다.
그렇지 않다면, 이 값을 반영하기 위해서 PPCR, MDCNFG, MDCAS0, MDCAS1, MDCAS2 레지스터에 해석한 값을 넣어준다. 또한 이것이 효력을 발생시킬때 까지 가디리기 위해서, 1초 정도의 delay를 준다(for).

```
int MyIsXDigit(char isdigit) {
  if ((isdigit >= '0' && isdigit <= '9' ) ||
      (isdigit >= 'a' && isdigit <= 'f'))
     return 1;
  else
     return 0;
} /* MyIsXDigit */

int MyXDigitValue(char isdigit) {
  if (isdigit >= '0' && isdigit <= '9' )
    return isdigit - '0';
  if (isdigit >= 'a' && isdigit <= 'f')
```

```
    return 10 + isdigit - 'a';
  return -1;
} /* MyXDigitValue */
```

코드 2. MyIsXDigit()과 MyXDigitValue() 함수의 정의

MyIsXDigit()과 MyXDigitValue()함수는 GetHexValue() 함수에서 사용하는 함수로 먼저 잠시보도록 하겠다. 이 함수가 하는 역할은 단순히 hexadecimal인지를 확인하는 것과 hexadecimal로 변형하는 일을 한다.
먼저 MyIsXDigit() 함수는 하나의 문자(isdigit)를 넘겨받아서, 이 값이 '0'과 '9'사이에 있거나, 'a'에서 'f'사이에 있다면 1을 돌려주고, 그렇지 않다면 0을 돌려준다. MyXDigitValue() 함수는 하나의 문자(isdigit)를 받아들여서, 문자가 '0'에서 '9'사이에 있다면, 원래의 문자가 가지는 ASCII값에서 '0'을 뺀 값을 돌려주고, 'a'에서 'f'사이에 있다면, 원래의 문자가 가지는 ASCII값에서 'a'를 뺀다음 이것에 10을 더해서 돌려준다. 즉, 문자가 가지는 hexadecimal로 표현된 의미를 10진수로 바꾼다.

```
/* Converts 8 characters 0-9a-f into a 4 byte hexadecimal value */
char *GetHexValue(char *commandline,u32 *value)
{
  int i;

  if(commandline[0] == '\0') {
    SerialOutputString(__FUNCTION__ ": zero length string \n");
    return(NULL);
   }
  *value=0x00;
  if (MyStrNCmp(commandline, " 0x", 3) == 0) {
     commandline +=3;
  } else if(commandline[0] == ' ') {
     commandline +=1;
  } else {
    SerialOutputString(__FUNCTION__ ":value not hexdecimal \n");
    return NULL;
   }
  SerialOutputString("char: ");
  SerialOutputString(commandline);
  SerialOutputByte(' ');
```

```
 for (i=0; i<8;i++) {
   if (*commandline == '\0') {
    SerialOutputString(__FUNCTION__ ":hex value not 32 bits \n");
    return NULL;
   } else {
    if (MyIsXDigit(*commandline) == 0) {
      SerialOutputString("hex value contains invalid characters \n");
      return NULL;
     } else {
      *value |= (u32) MyXDigitValue(*commandline) << ((7-i)*4);
      }
    }
   commandline++;
  }
 SerialOutputString("hex 0x");
 SerialOutputHex(*value);
 SerialOutputByte('\r');
 return commandline;
} /* GetHexValue */
```

코드 3. GetHexValue() 함수의 정의

먼저 넘겨받은 commandline이 올바른 문자열을 가지는지 확인한다. 그렇지 않다면, 에러 메시지를 보인 후 NULL을 돌려준다. 바꾼 값을 가지는 변수 *value를 초기화 시키고, commandline 변수가 ‘0x’[1]를 가지는지 검사한다. 만약 같다면, commandline 변수를 3증가시켜서, ‘0x’이후의 문자를 가르키도록 만든다. 그렇지 않다면, commandline[0]이 spece(=“ “)와 같은지 비교해서, 같다면 commandlie 변수를 하나 증가 시켜준다. 앞의 두 가지 해당 사항이 없다면, 에러를 출력하고 NULL을 돌려준다.

앞에서 넘겨받은 commandline을 출력하고, for loop를 돌면서 8자리의 문자로 표현된 값을 decimal 값으로 바꾼다. 먼저 commandline[]이 ‘\0’으로 이미 끝이 났는지를 검사한다. 그렇다면, 에러 메시지를 출력하고 NULL을 돌려준다. 그렇지 않다면, commandline[]이 hexadecimal을 가지는지를 검사한다. 그렇지 않다면, 역시 에러 메시지를 보여준 후 NULL을 돌려준다. 위의 두가지 조건이 다 아니라면, 이젠 hexadecimal을 decimal 값으로 변형하기 위해서 MyXDigitValue() 함수를 사용해서 값을 변환한다. 변환된 값은 각 bit position에 맞도록 shift되어 value가 가르키는 값에 OR된다. 한 문자의 처리를 마쳤다면,

[1] 제일 앞쪽에 space를 포함하고 있으므로, 비교하고자 하는 문자열의 길이는 3이다.

다음 commandline 문자의 처리를 위해서 commandline은 증가시킨다. 다 끝났다면, SerialOutputXXX()와 같은 함수를 호출해서 연산의 결과를 화면에 나타내고, 다음번 처리를 위해서 commandline을 돌려주고 복귀한다.

위에서 보았듯이 commandline 상에서 준 문자열이 메모리를 설정하는 레지스터의 내용에 정해진 순서대로 들어가기 때문에, 사용자는 clock이라는 명령을 사용하게 될 때는 순서에 유념해야 할 것이다.

1.1.1.3. Download Command

Download command는 MyStrNCmp()를 호출할 때, 'download '라고 명시해서 뒤에 하나의 space까지를 포함하고 있다. 따라서, download 이후에 주어지는 파라미터 값에 의해서 호출의 결과가 달라질 것이다. Download() 함수의 정의는 main.c에 아래와 같이 되어 있다.

```
void Download(char *commandline)
{
	u32 startAddress = 0;
	int bufLen;
	int *numRead = 0;

	if(MyStrNCmp(commandline, "blob", 4) == 0) {
		/* download blob */
		startAddress = BLOB_RAM_BASE;
		bufLen = blob_status.blockSize - BLOB_BLOCK_OFFSET;
		numRead = &blob_status.blobSize;
		blob_status.blobType = fromDownload;
	} else if(MyStrNCmp(commandline, "kernel", 6) == 0) {
		/* download kernel */
		startAddress = KERNEL_RAM_BASE;
		bufLen = blob_status.blockSize - KERNEL_BLOCK_OFFSET;
		numRead = &blob_status.kernelSize;
		blob_status.kernelType = fromDownload;
	} else if(MyStrNCmp(commandline, "ramdisk", 7) == 0) {
		/* download ramdisk */
		startAddress = RAMDISK_RAM_BASE;
		bufLen = blob_status.blockSize - RAMDISK_BLOCK_OFFSET;
		numRead = &blob_status.ramdiskSize;
```

```
                blob_status.ramdiskType = fromDownload;
        } else {
                SerialOutputString("*** Don't know how to download \"");
                SerialOutputString(commandline);
                SerialOutputString("\"\n");
                return;
        }
```

코드 4. Download() 함수의 정의

Dowload() 함수는 BLOB과 kernel 및 RAM disk를 download받는 역할을 수행한다. 먼저, commandline 변수에 blob이 있는 경우에는 startAddress에 BLOB_RAM_BASE(=0xc1000000)로 두고, bufLen을 blob_status.blockSize - BLOB_BLOCK_OFFSET(=0x00000000)으로 둔다. Serial을 통해서 읽어야 할 byte의 수를 numRead에 두기 위해서 blob_status.blobSize를 가르키도록 만들고, blob_status.blobType에는 serial을 통해서 downloading한다는 것을 나타내기 위해서 fromDownload로 만든다. 나머지 downloading에 관련된 것은 아래에서 처리할 것이다. 여기서는 일단 필요한 파라미터 값만 설정한다.

Kernel을 downloading하기 위해서는 startAddress에 KERNEL_RAM_BASE(=0xC0008000)를 주고, bufLen에는 blob_status.blockSize - KERNEL_BLOCK_OFFSET(=0x00008000)을 둔다. numRead에는 blob_status.kernelSize의 주소를 주고, blob_status.kernelType은 fromDownload로 둔다.

RAM disk를 download하기 위해서는 startAddress에 RAMDISK_RAM_BASE(=0xC0800000)를 주고, bufLen에는 blob_status.blockSize - RAMDISK_BLOCK_OFFSET(=0x00800000)을 둔다. numRead에는 blob_status.ramdiskSize의 주소를 주고, blob_status.ramdiskType에는 fromDownload로 둔다. 나머지 경우에 대해서는 잘못된 download 명령으로 간주한다.

```
        SerialOutputString("Switching to ");
        PrintSerialSpeed(blob_status.downloadSpeed);
        SerialOutputString(" baud \n");
        SerialOutputString("You have 60 seconds to switch your terminal emulator to the same speed and \n");
        SerialOutputString("start downloading. After that " PACKAGE " will switch back to 9600 baud. \n");
        SerialInit(blob_status.downloadSpeed);
        *numRead = UUDecode((char *)startAddress, bufLen);
```

```
        SerialOutputString("\n(Please switch your terminal emulator back to 9600 baud)\n");
        if(*numRead < 0) {
                /* something went wrong */
                SerialOutputString("*** Uudecode receive failed \n");

                /* reload the correct memory */
                Reload(commandline);
                SerialInit(baud9k6);
                return;
        }
        SerialOutputString("Received ");
        SerialOutputDec(*numRead);
        SerialOutputString(" (0x");
        SerialOutputHex(*numRead);
        SerialOutputString(") bytes.\n");
        SerialInit(baud9k6);
}
```

코드 5. Download() 함수의 정의(계속)

이젠 실제적으로 downloading을 할 차례이다. 먼저 현재의 serial speed를 보여주도록 한다. Download speed에는 앞에서 이미 설정한 115200 bps가 있으므로, 이것을 이용해서 serial을 다시 설정하도록 한다(SerialInit()). 설정이 끝났다면, UUDecode() 함수를 사용해서 원하는 만큼을 downloading 받도록 한다. UUDecode() 함수는 startAddress에서 시작해서 bufLen만큼의 크기를 downloading해 줄 것이다. 이것을 마치고나면, 다시 원래의 serial speed(9600 bps)로 바꾸기 위해서 다시 한번 SerialInit() 함수가 호출될 것이다. 만약 downloading중에 문제가 발생했다면, 원래의 이미지를 다시 load하기 위해서 Reload() 함수가 호출될 것다. 성공적으로 downloading이 되었다면, downloading된 실제의 크기를 화면상에 보여줄 것이다.

```
int UUDecode(char *bufBase, int bufLen)
{
/*      Receives and decodes an incoming uuencoded stream. Returns the number of
        bytes put in the buffer on success, or -1 otherwise. */

        int n, linesReceived = 0;
```

```
char ch, *p;
int bytesWritten = 0, retries = 0;
char buf[INT_BUF_SIZE];

/* Search for header line. We don't care about the mode or filename */
retries = 0;
do {
        SerialInputString(buf, sizeof(buf), 6);
        TEST_MAX_RETRIES;
} while (MyStrNCmp(buf, "begin ", 6) != 0);
```

코드 6. UUDecode() 함수의 정의

UUDecode() 함수는 uucodec.c에 정의된 함수이다. 원래 UUDecode와 UUEncode를 합쳐서 Unix-to-Unix copy와 같은 곳에서 사용하기 위한 것으로 여기서는 binary file을 전송하는데 사용하고 있다. 먼저 input을 계속 읽어서, begin이라는 단어가 나오는지를 확인한다. 사용하는 buffer의 크기는 INT_BUF_SIZE (=1024 bytes)이다.

```
/* for each input line */
for (;;) {
        if (SerialInputString(p = buf, sizeof(buf), 2) == 0) {
                SerialOutputString("\n*** Short file. Aborting \n");
                return -1;
        }
        /* Status print to show where we are at right now */
        if((linesReceived++ & 0x007F) == 0) {
                SerialOutputByte('.');
        }
        /*
         * `n' is used to avoid writing out all the characters
         * at the end of the file.
         */
        if ((n = DEC(*p)) <= 0)
                break;
```

코드 7. UUDecode() 함수의 정의(계속)

For loop를 무한번 돌면서 계속 serial을 통해서 downloading을 받도록 한다. 이미 앞에서

header를 읽었으므로, 여기서는 데이터가 될 것이다. 만약 download할 것이 없다면, 잘못된 file을 명시한 것이므로 에러 메시지를 보여주고 -1을 돌려준다. 읽어들인 line수가 특정 값(=0x007F)와 같다면, 진행 상황을 보여주기 위해서 "."을 serial을 통해서 보여준다. DEC() 매크로는 하나의 문자를 decoding하는 것으로, 아래와 같이 구현된다. 관련된 매크로의 정의를 다 묶어서 보여주록 하겠다.

```
#define INT_BUF_SIZE 1024
#define  MAX_RETRIES 10

#define TEST_MAX_RETRIES do {                                          \
          if(retries++ > MAX_RETRIES) {                                \
                SerialOutputString("\n*** Timeout exceeded. Aborting. \n"); \
                  return -1;                                           \
          }                                                            \
} while(0)

#define  DEC(c)  (((c) - ' ') & 077)                  /* single character decode */
#define IS_DEC(c) ( (((c) - ' ') >= 0) &&  (((c) - ' ') <= 077 + 1) )
/* #define IS_DEC(c) (1) */

#define OUT_OF_RANGE do {       \
           SerialOutputByte('\n');   \
       SerialOutputString(buf);      \
       SerialOutputString("\n*** Received character out of range. Aborting. \n"); \
        return -1; \
} while(0)

#define PUT_CHAR(x) do {                \
          if(bytesWritten < bufLen)                  \
                bufBase[bytesWritten++] = x; \
} while(0)
```

코드 8. uucodec.c의 매크로 및 상수 정의

즉, DEC() 매크로는 문자(c)에서 ' '사이의 값을 구한 후, 이 값을 다시 077[2]과 AND 시킨

[2] 이것은 C언어에서 8진수를 나타내는 format이다. 따라서, binay로는 01110111(b)가 될 것이며, hexadecimal로는 0x77이 될 것이다. 즉, 한 문자 길이가 된다(8bit).

값을 만든다. IS_DEC() 매크로는 제대로 한 문자(c)가 decoding이 되었는지를 확인하기 위해서, 문자(c)와 ' '사이의 값이 0에서 077 + 1까지의 값을 가지는지를 확인하게 된다. OUT_OF_RANGE 매크로는 단순히 받은 문자가 범위를 벗어났다는 에러 메시지를 보여주기 위한 것이며, PUT_CHAR() 매크로는 buffer에 하나의 문자를 끝에 채워넣는 역할을 한다. 따라서, 앞에서와 같이 DEC() 매크로를 사용한 것은 현재 decoding하고 있는 문자가 제대로 된 값인지를 확인하는 부분이다.

```
        for (++p; n > 0; p += 4, n -= 3)
                if (n >= 3) {
                        if (!(IS_DEC(*p) && IS_DEC(*(p + 1)) &&
                            IS_DEC(*(p + 2)) && IS_DEC(*(p + 3))))
                                OUT_OF_RANGE;
                        ch = DEC(p[0]) << 2 | DEC(p[1]) >> 4;
                        PUT_CHAR(ch);
                        ch = DEC(p[1]) << 4 | DEC(p[2]) >> 2;
                        PUT_CHAR(ch);
                        ch = DEC(p[2]) << 6 | DEC(p[3]);
                        PUT_CHAR(ch);

                }
                else {
                        if (n >= 1) {
                                if (!(IS_DEC(*p) && IS_DEC(*(p + 1))))
                                        OUT_OF_RANGE;
                                ch = DEC(p[0]) << 2 | DEC(p[1]) >> 4;
                                PUT_CHAR(ch);
                        }
                        if (n >= 2) {
                                if (!(IS_DEC(*(p + 1)) &&
                                        IS_DEC(*(p + 2))))
                                        OUT_OF_RANGE;
                                ch = DEC(p[1]) << 4 | DEC(p[2]) >> 2;
                                PUT_CHAR(ch);
                        }
                        if (n >= 3) {
                                if (!(IS_DEC(*(p + 2)) &&
```

```
                                        IS_DEC(*(p + 3))))
                                        OUT_OF_RANGE;
                                    ch = DEC(p[2]) << 6 | DEC(p[3]);
                                    PUT_CHAR(ch);
                                }
                            }
```

코드 9. UUDecode() 함수의 정의(계속)

이 부분은 UUDecode() 함수에 있는 두개의 for loop중에서 내부에 있는 for loop에 대한 것을 보여준다. 여기서 사용하는 n이라는 변수는 앞에서 for loop를 진행하기 전에 한 문자를 읽어서, 얼마나 많이 decoding을 해야할 부분이 있는지를 알려주는 값이다. 실제 decoding하는 절차는 한번에 4개의 byte씩을 진행하게 된다. n이 3이상이라면, 3개의 문자가 모두 올바른 decode값을 가지는지 확인하고, 이 문자들을 decoding(DEC())해서 각각을 차례로 buffer에 넣게 된다. 나머지는 n이 각각 1보다 크거나 같은 경우와 2보다 크거나 같은 경우, 3보다 크거나 같은 경우로 나누어서 각각의 문자를 decoding해서 buffer에 저장하게 된다.[3]

```
        }
        SerialOutputByte('\n');
        if (SerialInputString(p = buf, sizeof(buf), 2) == 0 || (MyStrNCmp(buf, "end", 3)))
{
                SerialOutputString("*** No \"end\" line. Aborting.\n");
                return(-1);
        }
        return(bytesWritten);
} /* UUDecode */
```

코드 10. UUDecode() 함수의 정의(계속)

이젠 loop를 돌면서 encoding된 data부분에 대한 처리는 마쳤다. 마지막으로, “end”가 끝에 위치하는지를 판단한 후, 그렇지 못하다면 제대로 파일을 전달 받지 못한 것이 되므로, 에러 메시지를 출력한 후 -1을 돌려준다. 성공적으로 수행을 마쳤을 경우, 최종적으로 돌려주는 값은 버퍼에 쓴 문자의 수가 될 것이다.

---

[3] UUDecoding의 자세한 algorithm을 아직은 다 이해하고 있지 못하기에 여기까지만 보는 것으로 일단은 만족하길 바란다.

```
#if 0
/* ENC is the basic 1 character encoding function to make a char printing */
#define  ENC(c) ((c) ? ((c) & 077) + ' ': '`')
void UUEncode(char *bufBase, int bufLen) {
        register int ch, n;
        register char *p;
        char buf[80];

        SerialOutputString("begin 644 testme.jdb \ n");
        while (bufLen > 0) {
                n = (bufLen > 45) ? 45 : bufLen;
                MyMemCpyChar(buf, bufBase, n);
                bufBase += n;
                bufLen  -= n;
                ch = ENC(n);
                SerialOutputByte(ch);
                for (p = buf; n > 0; n  -= 3, p += 3) {
                        ch = *p >> 2;
                        ch = ENC(ch);
                        SerialOutputByte(ch);
                        ch = ((*p << 4) & 060) | ((p[1] >> 4) & 017);
                        ch = ENC(ch);
                        SerialOutputByte(ch);
                        ch = ((p[1] << 2) & 074) | ((p[2] >> 6) & 03);
                        ch = ENC(ch);
                        SerialOutputByte(ch);
                        ch = p[2] & 077;
                        ch = ENC(ch);
                        SerialOutputByte(ch);
                }
                SerialOutputByte(' \ n');
        }
        ch = ENC(' \ 0');
        SerialOutputByte(ch);
        SerialOutputByte(' \ n');
        SerialOutputString("end \ n");
```

```
} /* UUEncode */
#endif
```

코드 11. UUEncode() 함수의 정의

UUEncode()는 파일을 upload하기 위해서 사용하는 UUDecode()의 반대 역할을 수행하는 함수이다. 현재는 사용될 이유가 없기때문에, 일단 #if 0 ~ #endif절로 선언되어 compile시에 빠지게 되지만, 이해의 목적으로 덧붙여 보았다.
ENC()는 문자 단위로 encoding을 수행하는 매크로이다. 앞에서 decoding시에 했던 반대 연산을 수행하게 된다. 먼저 문자가 있다면, 이 문자를 077과 AND 시킨 값에 다시 ' '을 더해주고, 그렇지 않다면 '`'를 두도록 한다. 먼저 "begin .. "으로 시작하는 문자열을 먼저 serial을 통해서 내보내고, buffer를 45문자의 크기 단위로 encoding하도록 만든다. 먼저 주어진 문자 만큼 씩(45나 혹은 남은 문자열의 크기)을 encoding을 할 buffer(buf)에 저장하고, 길이를 나타내는 n을 ENC() 매크로를 사용해서 encoding한 다음, serial을 통해서 전달한다. 나머지 복사된 buffer에 있는 문자는 3문자씩 encoding되어, serial을 통해서 전달된다. 이렇게 전달된 문자열은 '\n'에 의해서 분리되며, 다음번 loop가 남은 문자열에 대해서 수행된다. 데이터에 대한 encoding이 끝나면, '\0'를 encoding가게되며, 다시 '\n'을 보낸 후, 마지막으로 'end \n'으로 encoding을 마치도록 한다.

더 진행하기에 앞서서 잠시 UUDecoding과 UUEncoding에 대해서 이론적인 이야기를 조금 더 하도록 하겠다. 원래 UUdecode와 UUencode는 네트워크 상에서 시스템들 간에 파일을 전송하기 위해서 encoding가거나 decoding하는 목적으로 사용되는 일종의 utility 프로그램이다. 예전의 Unix 시스템에서 시스템간에 파일을 전송하기 위해서 사용한다고 해서 UU(Unix-to-Unix)라는 이름을 붙였다는 것은 이미 보았다. 그러나, 현재는 대부분의 운영체제에서 사용될 수 있도록 porting이 되어 있으며, 또한 쉽게 구할 수도 있다. 대부분의 전자우편에 대해서 첨부물을 전송하기 위한 방편으로도 사용되고 있으며, 기본적으로 text(여기서는 영어문자라는 의미로 사용한다.)가 아닌 binary로 데이터를 보내고자 할 경우에 이것을 사용할 수 있다. 즉, binary로 된 데이터 파일을 7 bit으로 표기되는 ASCII로 만들어서 전달해 준다.

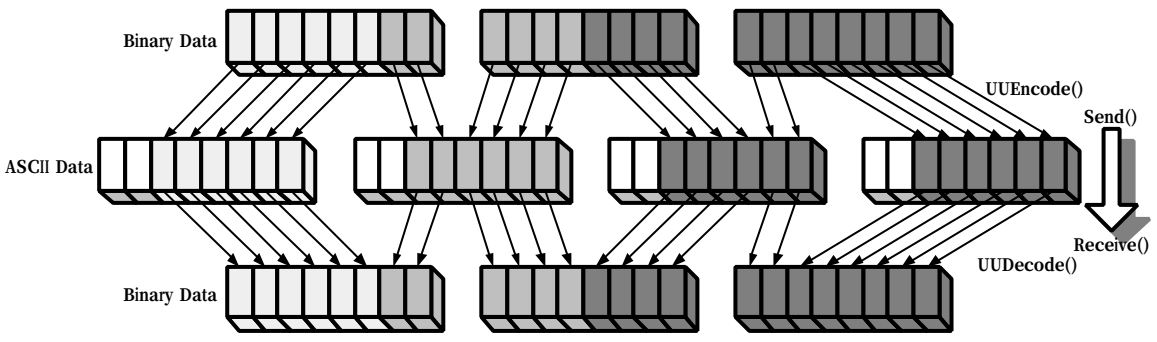

그림 2. UUEncode와 UUDecode

[그림 2]에서 보듯이 binary 형태로 된 data를 먼저 UUencode를 사용해서 ASCII 데이터로 만들어주고, 이를 다른 시스템으로 전달하게 되며, 받는 측에서는 받은 데이터를 다시 UUdecode를 사용해서 decoding해서 binary 형태로 복원해 주게 되는 것이다.