1. Serial 프로그램 예제

1.1. 개요

타겟 보드 자체에 메인 콘솔용으로 사용되는 통신 포트 이외에 두개의 다른 포트를 지원하고 있다.

일반적으로 메인 콘솔용으로 사용되는 포트는 통신용으로 부적합하다. 왜냐하면 커널에서 발생되는 메세지가 메인콘솔로 출력되기 때문에 통신에 방해를 받을 수 있기 때문이다.

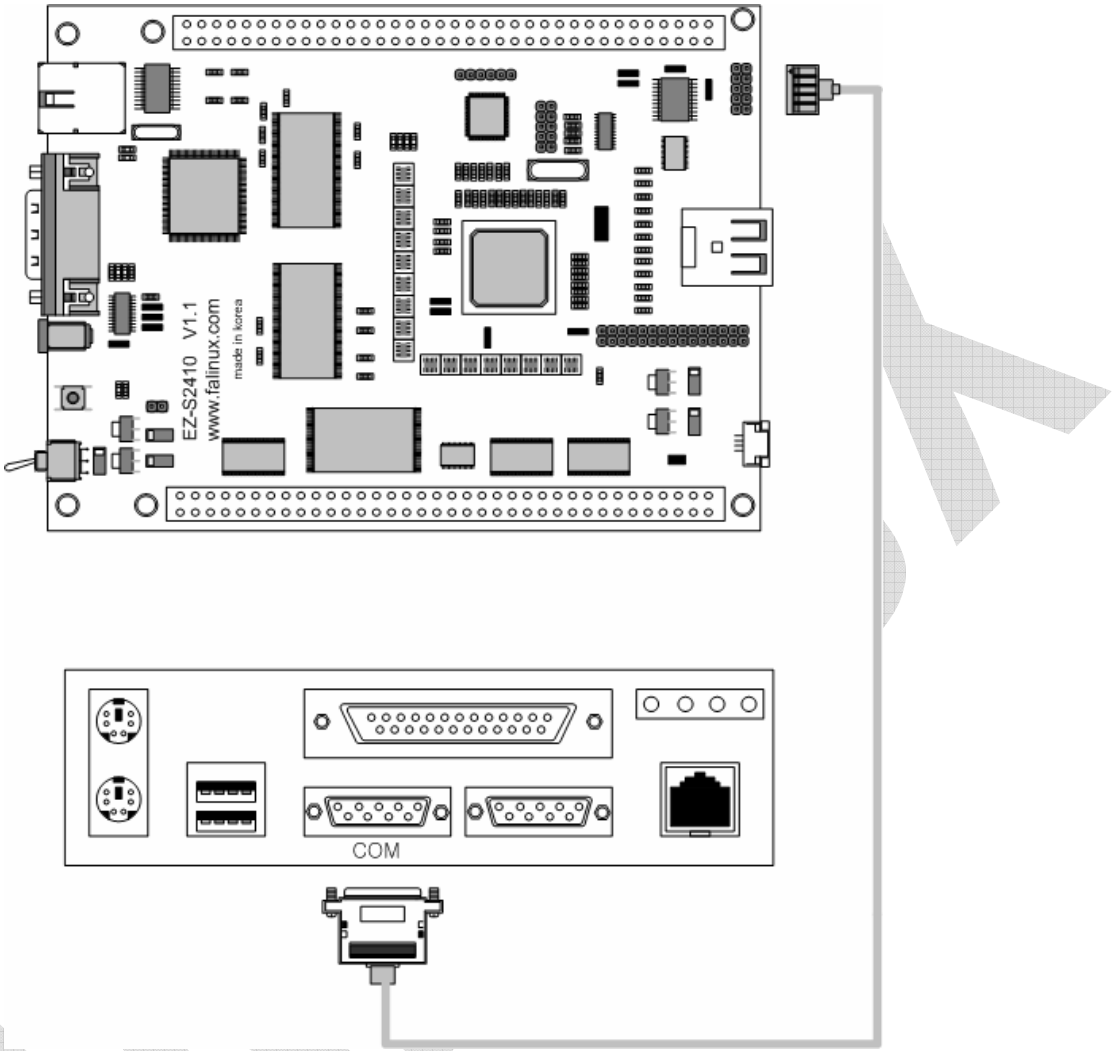
이런 이유로 타겟보드에는 통신 시험이나 응용 통신에 이용할 수 있도록 MCU 의 통신 포트에 MAX2421 레벨 변환기를 추가 하였다.

이 장은 이 두 포트 중 한 포트를 이용하여 리눅스에서 통신 프로그램을 어떻게 작성해야 하는가에 대한 간단한 예제를 들고 있다.

좀더 자세한 시리얼 프로그램 기법에 대해서는 리눅스 프로그램 관련 서적이나 KLDP 에 한글화된 문서를 참조하기 바란다.

1.2. 준비 조건

다음 그림과 같이 시리얼 케이블을 PC와 연결한다.



주의) FA-232C 케이블은 EZ-X5보드에 포함되어 있는 것이 아닙니다.
J8 점퍼는 초기에 제공된 상태로 두시면 됩니다.

1.3. 포트 세팅

리눅스에서는 터미널을 연결하기 위하여 장치 파일을 이용한다. 장치 파일은 /dev/ttySAC* 이다.

하지만 S3C2410의 시리얼 장치 파일은 이름이 다르게 명명된다.
/dev/ttySAC0, /dev/ttySAC1, /dev/ttySAC2 로 명명 되어 있다.

이중에서 /dev/ttySAC2 는 콘솔용으로 사용되고 있다.

이 장에서 설명하는 것은 /dev/ttySAC0 을 이용할 경우로 한정 지을 것이다. /dev/ttySAC1을 사용해도 문제는 없다.

시리얼 장치 파일은 터미널 장치로 분류된다. 따라서 초기 설정이 에코가 되도록 설정 되어 있다. 데이터 전송을 목적으로 한다면 바꾸어 주어야 한다.

장치 파일의 설정에 관한 것은 <asm/termbits.h>에 정의되어 있는 termios 구조체에 저장되어 있다. 구조체의 내용은 아래와 같다.

```
#define NCCS 19
struct termios  {
        tcflag_t c_iflag;       /* input mode flags */
        tcflag_t c_oflag;       /* output mode flags */
        tcflag_t c_cflag;       /* control mode flags */
        tcflag_t c_lflag;       /* local mode flags */
        cc_t c_line;            /* line discipline */
        cc_t c_cc[NCCS];        /* control characters */
    };
```

- c_iflag
  모든 입력 처리를 정의한다. 입력 처리란 read() 함수에 의해 시리얼 포트로 들어온 데이터를 read에 의해 읽기전에 데이터들을 c_iflag에 정의한 대로 처리하는 것을 의미한다.

- c_oflag
  출력처리 하는 방법을 정의한다.

- c_cflag
  baudrate, data bits, stop bits 등의 포트 세팅을 정의한다.

- c_lflag
  echo를 할 것인지 등을 결정한다.

- c_cc 배열
  EOF, STOP 등의 제어 동작들을 어떤 문자로 정의 할 것인가를 결정한다. 제어 문자의 디폴트 문자는 <asm/termios.h>에 정의되어 있다.

1.4. 시리얼 장치의 입력 방법

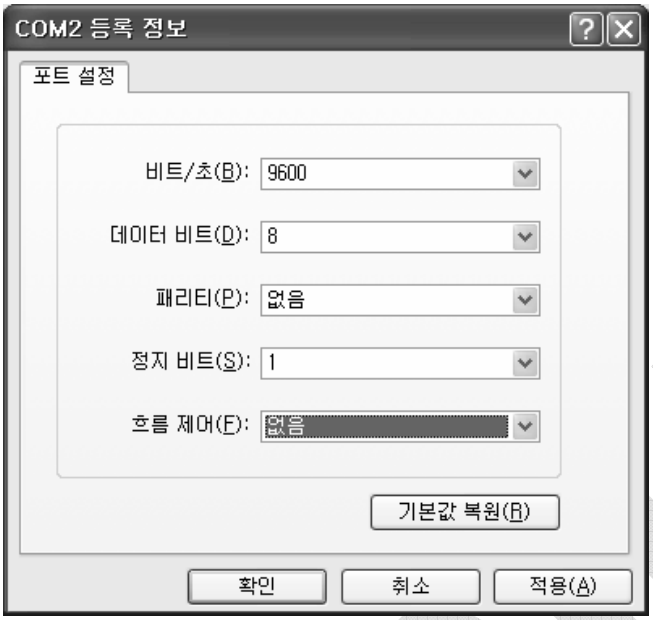
- Canonical 입력 처리(Canonical Input Processing)
  Canonical 입력 처리는 터미널의 기본 처리 방법이다. 이 방법은 한 줄 단위로 처리하는 다른 프로그램과 통신하는데에 사용할 수 있다. 한 줄은 디폴트로 NL(New Line, ASCII는 LF)문자, EOF(End of File)문자, 혹은 EOL(End Of Line)에 의해 종료되는 문자열을 의미한다. CR(Carriage Return, DOS/Windows의 디폴트 EOL 문자임) 문자는 디폴트 세팅에서 한 줄 의 종료 문자로 인식되지 않는다. 또한 Canonical 입력 처리 모드에서는 ERASE, DELETE WORD, REPRINT CHARACTERS 문자들을 처리할 수 있고, CR 문자를 NL 문자로 변환 처리를 할 수 있다.

- Non-Canonical 입력 처리(Non-Canonical Input Processing)
  Non-Canonical 입력 처리 모드에서는 한 번 읽을 때마다 정해진 크기의 문자만을 읽어낼 수 있다. 또한 타이머를 두어서 일정 시간까지 read()가 리턴하지 않는 경우 강제 리턴을 할 수 있다. 이 모드는 항상 정해진 크기의 문자들만을 읽어내거나 대량의 문자들을 전송하고자 할 때 사용한다.

- 비동기 입력
  위의 두가지 모드는 동기 방식이나 비동기 방식으로 사용될 수 있다. 동기 방식은 read의 조건이 만족될 때까지 block되는 방식으로서 디폴트로 설정되어 있다. 동기 방식은 read의 조건이 만족될 때까지 block되는 방식으로서 디폴트로 설정
  되어 있다. 비동기 방식에서는 read() 함수가 바로 리턴되며, 호출한 프로그램에게 signal을 보낸다. 이 signal handler(시그널 처리 함수)로 보내 진다.

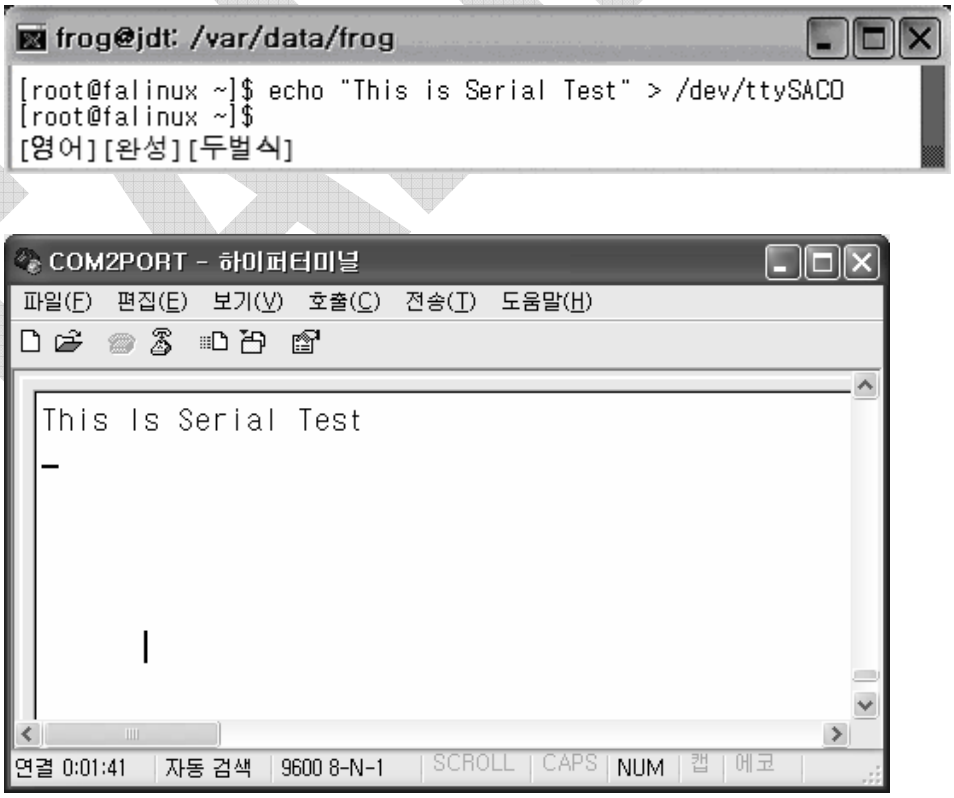
1.5. 프로그램 동작 없이 출력 동작 시험

하이퍼 터미널에서 통신 에뮬레이터의 PC 쪽 환경을 간단하게 다음과 같이 맞춘다.



리눅스에서는 터미널 장치의 기본 설정은 9600Baud 이다.

보드에서 다음 같이 입력 했을 때 하이퍼 터미널에 동일한 메세지가 나온다면 연결이 정상적으로 된 것이다.



1.6. 프로그램 동작 설명 및 소스

이 예제는 간단하게 데이타를 전송하고 수신된 데이타를 표출하는 예제다.

우선 Makefile의 내용을 살펴 보자

[Makefile]

```
#
# APP Makefile
#

CROSS_COMPILE = arm-linux-

CC          = $(CROSS_COMPILE)gcc
LD          = $(CROSS_COMPILE)ld

CFLAGS    += -O
LDFLAGS +=

TARGET    = sap_serial
OBJS       = sample_serial.o
SRCS       = $(OBJS:.o=.c)


all :      $(TARGET)
    cp -f $(TARGET) /nfs/sap

$(TARGET) : $(OBJS)
    $(CC) $(CFLAGS) $(LDFLAGS) $(OBJS) -o $@

%.o:%.c
    @echo "Compiling $< ..."
    $(CC) -c $(CFLAGS) -o $@ $<


dep :
    $(CC) -M $(SRCS) > .depend

clean :
    rm -rf $(OBJS) $(TARGET) core
```

다음은 어플리케이션 소스이다.

[sample_serial.c]

```
//-------------------------------------------------------------------------
// 파 일 : sample_serial.c
// 설 명 : 시리얼 포트를 사용하여 데이타를 전송하는 예제이다.
//
// 작 성 :
//-------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/signal.h>
#include <sys/ioctl.h>
#include <sys/poll.h>

#include <termios.h>

//-------------------------------------------------------------------------
// 설명 : 시리얼포트를 연다.
// 주의 : RTS/CTS 를 제어하지 않는다.
//        시리얼포트를 열고 이전의 포트설정상태를 저장하지 않았다.
//-------------------------------------------------------------------------
int  open_serial( char *dev_name, int baud, int vtime, int vmin )
{
    int         fd;
    struct termios    newtio;

    // 시리얼포트를 연다.
    fd = open( dev_name, O_RDWR | O_NOCTTY );
    if ( fd < 0 )
    {
        // 화일 열기 실패
        printf( "Device OPEN FAIL %s\n", dev_name );
        return -1;
    }

    // 시리얼 포트 환경을 설정한다.
    memset(&newtio, 0, sizeof(newtio));
    newtio.c_iflag = IGNPAR;              // non-parity
    newtio.c_oflag = 0;

    newtio.c_cflag = CS8 | CLOCAL | CREAD;  // NO-rts/cts
```

```
    switch( baud )
    {
    case 115200 : newtio.c_cflag |= B115200; break;
    case 57600  : newtio.c_cflag |= B57600;  break;
    case 38400  : newtio.c_cflag |= B38400;  break;
    case 19200  : newtio.c_cflag |= B19200;  break;
    case 9600   : newtio.c_cflag |= B9600;   break;
    case 4800   : newtio.c_cflag |= B4800;   break;
    case 2400   : newtio.c_cflag |= B2400;   break;
    default     : newtio.c_cflag |= B115200; break;
    }

   //set input mode (non-canonical, no echo,.....)
    newtio.c_lflag = 0;
    newtio.c_cc[VTIME] = vtime;   // timeout 0.1초 단위
    newtio.c_cc[VMIN]  = vmin;    // 최소 n 문자 받을 때까진 대기

    tcflush  ( fd, TCIFLUSH );
    tcsetattr( fd, TCSANOW, &newtio );

    return fd;
}
//-------------------------------------------------------------------------
// 설명 : 시리얼포트를 닫는다.
// 주의 :
//-------------------------------------------------------------------------
void        close_serial( int fd )
{
    close( fd );
}
//-------------------------------------------------------------------------
// 설명 : main
//        32바이트의 데이타를 보낸후 데이타가 들어오는지를 1초동안 감시한다.
//        데이타가 일정시간(1초)동안 없으면 다시 데이타를 전송한다.
// 주의 :
//-------------------------------------------------------------------------
int     main( int argc, char **argv )
{
    int     fd;                 // 시리얼포트 파일핸들
    int     baud;               // 전송속도
    char    dev_name[128];      // 시리얼포트 노드파일 이름
    char    cc, buf[128];       // 데이타 버퍼
    int     rdcnt;

    if ( argc != 3 )
    {
        printf( " sample_serial [device] [baud]\n" \
                "       device : /dev/ttySAC0 ...\n"      \
                "       baud   : 2400 ... 115200\n" );
        return -1;
    }
    printf( " Serial test start...   (%s)\n", __DATE__ );
```

```
        // 인자를 얻어온다.
        strcpy( dev_name, argv[1] );                    // 노드파일 이름
        baud       = strtoul( argv[2], NULL, 10 );      // 전송속도

        // 시리얼 포트를 연다
        // 시리얼포트를 1초동안 대기하거나 32바이트 이상의 데이타가 들어오면
        // 깨어나도록 설정한다.
        fd = open_serial( dev_name, baud, 10, 32 );
        if ( fd < 0 ) return -2;

        for ( cc='A'; cc<='z'; cc++ )
        {
                // 32바이트 데이타를 전송한다.
                memset( buf, cc, 32 );
                write( fd, buf, 32 );

                // 데이타를 읽어온다.
                rdcnt = read( fd, buf, sizeof(buf) );
                if ( rdcnt > 0 )
                {
                        buf[rdcnt] = '\0';
                        printf( "<%s rd=%2d> %s\n", dev_name, rdcnt, buf );
                }

                // 테스트를 위한 지연
                sleep(1);
        }

        // 시리얼 포트를 닫는다.
        close_serial( fd );

         printf( " Serial test end\n" );

         return 0;
}
```

1.7. 프로그램 소스 분석

int fd;
struct termios newtio;

시리얼 포트의 헨들과 termios 구조체를 선언한다. termios 구조체는 위에서 설명하였다.

fd = open(dev_name, O_RDWR | O_NOCTTY);

dev_name 은 /dev/ttySAC0 또는 /dev/ttySAC1 이다.
읽기/쓰기 모드로 장치를 연다.(O_RDWR)
데이터 전송 시에 <ctrl>-c 문자가 오면 프로그램이 종료되지 않도록 하기 위해 controlling tty 가 안되도록 한다.(O_NOCTTY)

newtio.c_cflag = B9600 | CS8 | CLOCAL | CREAD ;

B9600
전송속도. cfsetispeed() 및 cfsetospeed() 함수로도 세팅 가능

CS8
8N1 (8bit, no parity, 1 stopbit)

CLOCAL
Local connection. 모뎀 제어를 하지 않는다.

CREAD
문자 수신을 가능하게 한다.

CRTSCTS
하드웨어 흐름제어.(시리얼 케이블이 모든 핀에 연결되어 있는 경우만 사용)

newtio.c_iflag = IGNPAR;

IGNPAR
Parity 에러가 있는 문자 바이트를 무시한다.
ICRL
CR 문자를 NL 문자로 변환 처리한다. (이 설정을 하지 않으면 다른 컴퓨터는 CR 문자를 한 줄의 종료 문자로 인식하지 않을 수 있다.)

newtio.c_lflag = 0;

0으로 설정하면 Non-Canonical 입력 처리하게 된다.
Non-Canonical 입력 처리 모드에서는 입력이 한 줄 단위로 처리되지 않는다. 따라서, 이 모드에서 설정하는 파라미터는 **c_cc[VTIME]**과 **c_cc[VMIN]** 두 가지이다. 이것은 아래 다시 설명하였다.
ICANON
canonical 입력을 가능하게 한다.

newtio.c_cc[VTIME] = 10;

최소 1초 이상 수신이 없으면 타임 아웃이 걸린다. 이때 read 함수는 0을 반환한다.

newtio.c_cc[VMIN] = 32;

VTIME 값이 0 일 경우 read문이 리턴되기 위한 최소의 수신 문자 개수를 지정한다.

MIN > 0, TIME = 0

MIN은 read가 리턴되기 위한 최소한의 문자 개수. TIME이 0이면 타이머는 사용되지 않는다.(무한대로 기다린다.)

MIN = 0, TIME > 0

TIME은 time-out 값으로 사용된다. Time-out 값은 TIME * 0.1 초이다. Time-out이 일어나기 전에 한 문자라도 들어오면 read는 리턴된다.

MIN > 0, TIME > 0

TIME은 time-out이 아닌 inter-character 타이머로 동작한다. 최소 MIN 개의 문자가 들어오거나 두 문자 사이의 시간이 TIME 값을 넘으면 리턴된다.
문자가 처음 들어올 때 타이머는 동작을 시작하고 이후 문자가 들어올 때마다 재 시작된다.

MIN = 0, TIME = 0

read는 즉시 리턴된다. 현재 읽을 수 있는 문자의 개수나 요청한 문자 개수가 반환된다. Antonino씨에 의하면 read하기 전에 fcntl(fd, F_SETFL, FNDELAY); 를 호출하면 똑같은 결과를 얻을 수 있다.

다음은 termios.h에 있는 디폴트 값을 나열한 것이다.

```
newtio.c_cc[VINTR]              = 0;     /* Ctrl-c */
newtio.c_cc[VQUIT]              = 0;     /* Ctrl-\ */
newtio.c_cc[VERASE]             = 0;     /* del */

newtio.c_cc[VKILL]              = 0;     /* @ */
newtio.c_cc[VEOF]               = 4;     /* Ctrl-d */
newtio.c_cc[VTIME]              = 0;     /* inter-character timer unused */
newtio.c_cc[VMIN]               = 1;     /* blocking read until 1 character arrives */
newtio.c_cc[VSWTC]              = 0;     /* '\0' */
newtio.c_cc[VSTART]             = 0;     /* Ctrl-q */
newtio.c_cc[VSTOP]              = 0;     /* Ctrl-s */
newtio.c_cc[VSUSP]              = 0;     /* Ctrl-z */
newtio.c_cc[VEOL]               = 0;     /* '\0' */
newtio.c_cc[VREPRINT]  = 0;     /* Ctrl-r */
newtio.c_cc[VDISCARD] = 0;     /* Ctrl-u */
newtio.c_cc[VWERASE]  = 0;     /* Ctrl-w */
newtio.c_cc[VLNEXT]             = 0;     /* Ctrl-v */
newtio.c_cc[VEOL2]              = 0;     /* '\0' */
```

tcflush(fd, TCIFLUSH);

통신을 수행하기 이전에 이전에 아직 전송되지 않았거나 수신 처리가 되어 있지 않은 데이타를 모두 비워 버린다.

tcsetattr(fd, TCSANOW, &newtio);

모뎀 라인을 초기화하고 포트 setting을 마친다.

write(fd, buf, strlen(buf));

A~Z 까지의 32개의 문자가 통신 포트를 통해 전송한다.
하이퍼 터미널에서는 이 값이 표시 되어야 한다.

rdcnt = read(fd, buf, sizeof(buf));

수신된 문자를 표출한다.

close(fd);

이 부분은 위의 프로그램에서는 수행되지 않으나 일반적으로 프로그램이 종료되기 전에 수행되어 프로그램이 실행되기 이전의 포트상태로 되돌리고 사용하던 포트를 닫는다.

[주의 사항]

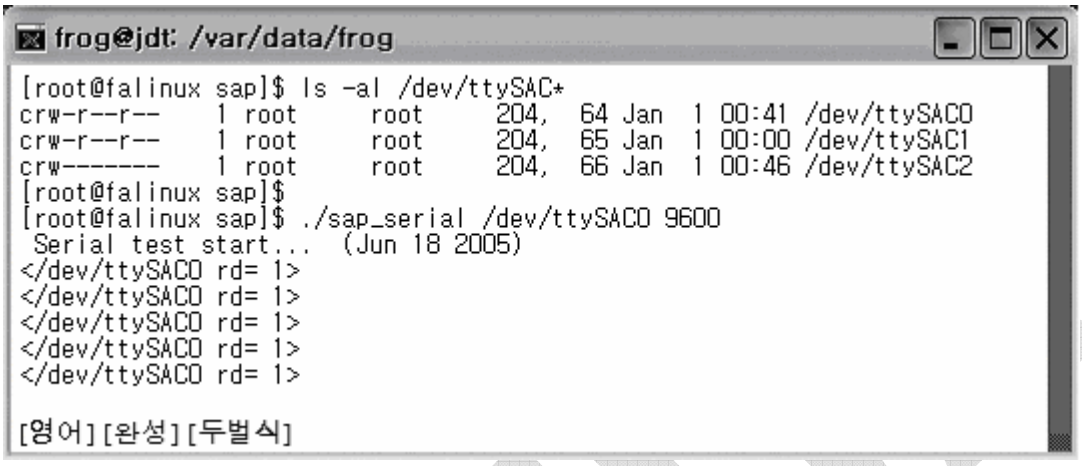
newtio.c_cc[VTIME] = 10;
newtio.c_cc[VMIN] = 32;

의 두 값을 잘 설정하여야 한다. 예제 프로그램에서는 이 두가지 모드를 보여주기 위해서 위와 같이 설정을 사용하였지만 실제 프로그램에서는 자신의 통신 프로토콜에 맞게 수정을 해 주어야 한다.

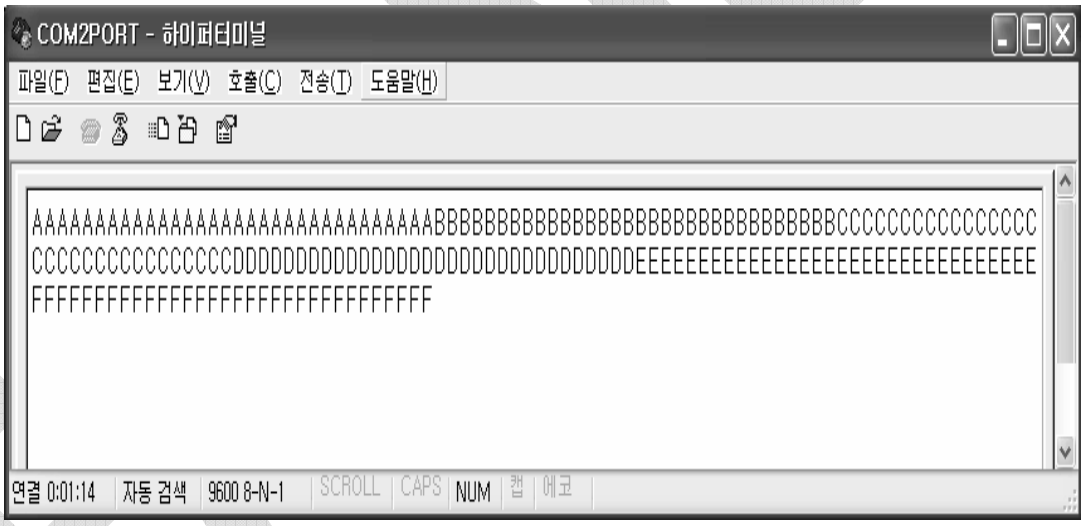
1.8. 프로그램 수행 화면

다음은 수행 화면을 보여준다.

[보드화면]



[PC 화면]



PC 화면에서 Enter키를 누를 때 마다 수신을 한다.
즉, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 32개의 문자를 받고,
Enter키를 누르면 다음 32개의 문자를 받는다.