【 기술 노트 24 】

ADS v1.2 및 WinARM C컴파일러에서 printf() 함수의 사용 방법

8051용의 Keil C컴파일러와 80196용의 IC96 C컴파일러에서 printf() 함수를 사용하는 방법은 [기술 노트 17]에서 이미 설명한 바 있으며, WinAVR 20060421 버전 이후부터 적용되는 편리한 printf() 함수 사용 방법은 [기술 노트 23]에서 설명하였다. 여기서는 ARM용의 C컴파일러인 ADS v1.2와 WinARM에서의 printf() 함수 사용 방법을 설명하기로 한다. ADS v1.2에서의 printf() 함수 사용법은 매우 간단하고 널리 알려져 있으나, WinARM에서의 printf() 함수 사용법은 좀더 복잡하기도 하거니와 그것을 잘 설명하고 있는 문헌자료가 거의 없어서 그동안 이 C컴파일러를 사용하는 사람들은 어려움을 겪어왔다.

오늘날 대부분의 마이크로콘트롤러용 C컴파일러에서는 ANSI C 표준의 printf() 함수를 지원하고 있으며, 이것은 내부적으로 기본 라이브러리 파일에서 1문자를 출력하는 저수준의 스트림 출력(low-level stream output) 함수를 사용하여 특정한 장치로 서식지정 문자열을 출력한다. C컴파일러에 따라 사용 방법이 다소 다르기는 하지만 사용자가 이 printf() 함수를 사용하여 출력할 장치를 변경하고자 하면 대체로 사용자가 이 1문자 출력함수를 수정함으로써 가능하게 되어 있다. 즉, 사용자 프로그램에서 지정된 방법으로 이 1문자 출력함수를 정의하여 사용하면 자동적으로 C컴파일러의 라이브러리에 들어있던 이 함수는 무시되고 printf() 함수는 사용자가 정의한 1문자 출력함수를 내부적으로 호출하여 사용하게 된다. 디폴트 출력장치로는 흔히 직렬통신 포트를 사용하고 있다.

이와 같이 C언어 프로그램에서 printf() 함수를 사용하여 %로 서식 지정하는 데이터를 출력한다면 각종 출력상치의 액세스가 매우 편리해진다. 예를 들어 이제까지 우리가 텍스트형 LCD 모듈의 출력 예제에서 했던 것처럼 수치 데이터를 출력하기 위하여 각종 사용자 정의 함수 LCD_2hex(), LCD_2d(), LCD_2d1() 등을 만들어 사용할 필요가 없기 때문이다. 이러한 데이터 출력의 편리함은 RS-232C 직렬통신 포드는 물론이고 덱스드형 LCD 모듈이나 그래픽형 LCD 모듈에도 모두 적용된다. 그러나, 이와 같이 printf() 함수를 사용하여 % 서식 지정으로 출력하는 방법은 사용하기에 편리한 상점이 있는 반면에 이것은 라이브러리 파일에서 상당히 긴 루틴을 링크하여 가져오기 때문에 오브젝트 파일이 매우 길어지고, 스택 메모리를 많이 사용하며, 그에 따라 이 루틴의 실행속도도 낮아진다는 단점을 갖는다.

▣ ADS v1.2의 경우

ADS v1.2 C컴파일러에서도 printf() 함수의 모든 표준적인 기능을 지원하고 있는데, 여기서는 printf() 함수가 내부적으로 저수준 스트림 출력함수 fputc()를 사용하고 있다. 따라서, 사용자 프로그램에서 원하는 주변장치에 데이터를 출력하기 위하여 printf() 함수를 사용하려면

먼저 헤더파일 stdio.h를 인클루드해 주고, 해당 출력장치를 미리 초기화하며, 1문자를 출력할 수 있는 fputc() 함수를 새로 작성하여 주는 것으로 printf() 함수를 사용하기 위한 준비는 끝난다.

ADS v1.2에서 printf() 함수를 사용하는 절차와 방법을 요약하면 다음과 같다.

① 사용자 프로그램의 서두에서 헤더파일 stdio.h를 인클루드한다. printf() 함수에 관련된 모든 사항은 이 헤더파일에 정의되어 있다.

② 1문자를 출력하는 저수준 스트림 함수에서 사용할 주변장치를 초기화한다. 일반적으로 그렇듯이 ADS v1.2에서도 1문자를 출력하는 저수준 스트림 함수에는 해당 출력장치를 초기화하는 기능을 포함하지 않으므로 이를 별도로 초기화해야 한다.

③ 1문자를 출력하는 저수준 스트림 함수 fputc()를 지정된 형식으로 만든다. 이 함수에서는 모든 ASCII 문자를 출력할 수 있어야 한다.

④ 필요할 때 printf() 함수를 사용한다. 여기서는 일반적인 ANSI C 표준의 모든 % 서식 지정이 사용될 수 있다.

▣ WinARM의 경우

WinARM에서도 printf() 함수의 모든 표준적인 기능을 지원하고 있는데, 여기서는 printf() 함수가 내부적으로 저수준 스트림 출력함수 _write_r()을 사용하고 있다. 따라서, 사용자 프로그램에서 원하는 주변장치에 데이터를 출력하기 위하여 printf() 함수를 사용하려면 먼저 헤더파일 stdio.h를 인클루드해 주고, 해당 출력장치를 미리 초기화하며, 1문자를 출력할 수 있는 _write_r() 함수를 새로 작성해야 한다. 단, WinARM에서는 사용자가 비록 사용하지 않더라도 scanf() 함수가 내부적으로 사용하는 저수준 스트림 입력함수 _read_r()까지 함께 만들어주어야 하며, 이밖에도 몇가지의 부수적인 함수를 만들어 주어야 한다.

WinARM에서 printf() 함수를 사용하는 절차와 방법을 요약하면 다음과 같다.

① 사용자 프로그램의 서두에서 헤더파일 stdio.h를 인클루드한다. printf() 함수에 관련된 모든 사항은 이 헤더파일에 정의되어 있다.

② 1문자를 출력하는 저수준 스트림 함수에서 사용할 주변장치를 초기화한다. 일반적으로 그렇듯이 WinARM에서도 1문자를 출력하는 저수준 스트림 함수에는 해당 출력장치를 초기화하는 기능을 포함하지 않으므로 이를 별도로 초기화해야 한다.

③ 1문자를 출력하는 저수준 스트림 함수 _write_r()을 지정된 형식으로 만든다. 이 함수에서는 모든 ASCII 문자를 출력할 수 있어야 한다. 이때, 1문자를 입력하는 함수 _read_r()도

함께 만들어두어야 하며, 만약 사용자 프로그램에서 _scanf() 함수를 사용하지 않거나 또는 1문자의 입력이 필요없다면 이 _read_r() 함수는 내용이 빈 것으로 만들면 된다.

④ 필요할 때 printf() 함수를 사용한다. 여기서는 일반적인 ANSI C 표준의 모든 % 서식 지정이 사용될 수 있다. 단, WinARM에서 printf() 함수를 사용할 때는 스트링의 마지막에 반드시 \n 문자를 사용해야 한다는 점에 유의하라.

▣ % 서식지정 형식의 요약

표준적으로 printf() 함수에서 사용할 수 있는 % 서식지정 형식을 요약하면 다음과 같다. 이는 ANSI C에서 표준으로 사용하는 % 서식지정 형식을 모두 포함하고 있다.

%[-][+][[0]w][.n][l]변환문자

% : 서식 제어 문자열의 시작을 의미한다.
- : 서식은 항상 필드의 오른쪽으로 정렬이 되지만 -를 붙이면 왼쪽으로 정렬된다.
+ : 필드의 서두에 항상 부호를 붙인다.
0 : 필드의 왼쪽 남는 자리를 공백으로 두지 않고 0을 채워 표시한다.
w : 전체 필드의 폭(field width)을 자리수로 나타낸다.
.n : 실수의 경우 소수점 이하의 자리수(precision)를 나타낸다.
l : long형 정수를 나타내며 변환문자 d, u, o, x의 앞에 붙을 수 있다.
변환문자 d : 부호 있는 10진수(signed int)로 표시한다.
변환문지 i : 부호 있는 10진수(signed int)로 표시한다. d와 같다.
변환문자 u : 부호 없는 10진수(unsigned int)로 표시한다.
변환문자 o : 부호 없는 8진수(unsigned int)로 표시한다.
변환문자 x : 부호 없는 16진수(unsigned int)로 표시한다.(a~f를 소문자로 출력)
변환문자 X : 부호 없는 16진수(unsigned int)로 표시한다.(A~F를 대문자로 출력)
변환문자 f : 부동소수점 수(double)를 소수점을 포함하는 실수 형식으로 표시한다.
변환문자 e : 부동소수점 수(double)를 지수 형식으로 표시한다.(e를 소문자로 출력)
변환문자 E : 부동소수점 수(double)를 지수 형식으로 표시한다.(E를 대문자로 출력)
변환문자 g : %f나 %e 중에서 적당한 방식으로 표시한다.
변환문자 G : %f나 %E 중에서 적당한 방식으로 표시한다.
변환문자 c : 1문자(unsigned char)를 표시한다.
변환문자 s : 문자열(char *)을 표시한다. 문자열은 NULL 문자로 종료된다.

예
```
printf("%5d\n",123);          ---> □□123
printf("%05d\n",123);         ---> 00123
printf("%-5d\n",123);         ---> 123□□
printf("%7.2f\n",123.456);    ---> □123.46
printf("%07.2f\n",123.456);   ---> 0123.46
printf("%+7.2f\n",123.456);   ---> +123.46
printf("%10.2e\n",123.456);   ---> □1.23e+002
printf("%10.2E\n",123.456);   ---> □1.23E+002
```

☞ 필드를 지정한 값이 실제 숫자의 폭보다 작으면 서식지정은 무시되고 숫자 전체가 표시된다. 부동소수점수에서 소수점 이하의 맨 아래 자리는 그 아래에서 반올림된다.

(1) ADS v1.2에서의 printf() 사용 프로그램

▣ DBGU 직렬통신 포트로 출력하는 경우

AT91SAM7S256에는 디버그 및 프로그램 다운로드용의 DBGU 포트를 UART 직렬통신 포트로 사용할 수도 있다. printf() 함수를 사용하여 DBGU 포트로 % 서식지정 출력을 내보내는 프로그램을 ADS v1.2에서 작성하여 보면 다음과 같다.

<프로그램 Xtest24_1.c> printf() 함수에 의한 DBGU로의 RS-232C 출력 프로그램

```
/* ========================================================================== */
/*           Xtest24 1.c : User-defined printf() Function for DBGU            */
/* ========================================================================== */
/*                        Designed and programmed by Duck-Yong Yoon in 2007.  */

#include <stdio.h>
#include "AT91SAM7S256.h"
#include "lib AT91SAM7S256.h"
#include "OK7S256ads.h"

void DBGU_TX_char(unsigned int data)            /* transmit a character by DBGU */
{
  while(!(*AT91C_DBGU_CSR & 0x0002));           // wait until TXRDY=1

  *AT91C_DBGU_THR = data;                       // transmit a character
}

int fputc(int ch, FILE *f)                      // for the user-defined printf()
{
  if(ch == '\n')
    { DBGU_TX_char('\r');                       // (CR character = 0x0D)
      DBGU_TX_char('\n');                       // (LF character = 0x0A)
    }
  else
    DBGU_TX_char(ch);

  return ch;
}

int main(void)
{ unsigned int i;
  double x;

  MCU_initialize();                             // initialize AT91SAM7S256 & kit
  Delay_ms(50);                                 // wait for system stabilization
  LCD_initialize();                             // initialize text LCD
  Beep();

  LCD_string(0x80,"    printf()    ");          // display title
  LCD_string(0xC0,"    for DBGU    ");

  AT91F_DBGU_CfgPMC();                          // enable clock of DBGU
  AT91F_DBGU_CfgPIO();                          // DTXD, DRXD by Peripheral A
```

```
    *AT91C_DBGU_MR = 0x00000800;                    // normal mode, no parity
    *AT91C_DBGU_BRGR = 26;                          // MCK/16/26 = 115200 bps
    *AT91C_DBGU_CR = 0x00000150;                    // TX, RX enable

    while(1)
      { for(i=1, x=0.001; i <= 200; i++, x += 0.001) // loop by 200 times
          { printf("printf() function test for DBGU : %3d  %5.3f\n", i, x);
            Delay_ms(500);
          }
        Beep();
      }
}
```

☞ 함수 fputc()의 형식은 반드시 위와 같이 처리해야 한다. 이 함수안에서 DBGU 포트로 1문자를 출력하고 있는 것에 주목하라. 그리고, \n 문자는 상대방에서 수신하여 화면에 출력할 것에 대비하여 이렇게 2개의 ASCII 문자인 캐리지 리턴(0x0D)과 라인 피드(0x0A)로 전송하는 것이 좋다.

이 프로그램을 컴파일하여 *OK-7S256* 키트에 다운로드하고 실행한 후에 출력 결과를 확인하려면 *OK-7S256* 키트의 DBGU 직렬통신 포트인 콘넥터 CN5와 퍼스널컴퓨터의 직렬포트를 RS-232C 통신 케이블로 연결해야 한다. 그리고, 퍼스널컴퓨터의 하이퍼터미널에서 115200 bps, 8 데이터, 패리티 없음, 1 스톱 비트로 설정하여 사용해야 한다.

▣ USART0 직렬통신 포트로 출력하는 경우

이번에는 printf() 함수를 사용하여 USART0 포트로 % 서식지정 출력을 내보내는 프로그램을 ADS v1.2에서 작성하여 보면 다음과 같다.

<프로그램 Xtest24_2.c> printf() 함수에 의한 USART0으로의 RS-232C 출력 프로그램

```
/* ===================================================================== */
/*          Xtest24_2.c : User-defined printf() Function for USART0      */
/* ===================================================================== */
/*                        Designed and programmed by Duck-Yong Yoon in 2007.  */

#include <stdio.h>
#include "AT91SAM7S256.h"
#include "lib_AT91SAM7S256.h"
#include "OK7S256ads.h"

void USART0_TX_char(unsigned int data)          /* transmit a character by USART0 */
{
  while(!(*AT91C_US0_CSR & 0x0002));            // wait until TXRDY=1

  *AT91C_US0_THR = data;                        // transmit a character
}

int fputc(int ch, FILE *f)                      // for the user-defined printf()
```

```
{
  if(ch == '\n')
    { USART0_TX_char('\r');                    // (CR character = 0x0D)
      USART0_TX_char('\n');                    // (LF character = 0x0A)
    }
  else
    USART0_TX_char(ch);

  return ch;
}

int main(void)
{ unsigned int i;
  double x;

  MCU_initialize();                            // initialize AT91SAM7S256 & kit
  Delay_ms(50);                                // wait for system stabilization
  LCD_initialize();                            // initialize text LCD
  Beep();

  LCD_string(0x80,"    printf()    ");         // display title
  LCD_string(0xC0,"   for USART0   ");

  AT91F_US0_CfgPMC();                          // enable clock of USART0
  AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA,AT91C_PA6_TXD0|AT91C_PA5_RXD0,0);
                                               // TXD0, RXD0 by Peripheral A

  *AT91C_US0_MR = 0x100008C0;                  // normal mode, no parity
                                               // 8 data, 1 stop, LSB first, 16x
  *AT91C_US0_BRGR = 26;                        // MCK/16/26 = 115200 bps
  *AT91C_US0_CR = 0x00000150;                  // TX, RX enable

  while(1)
    { for(i=1, x=0.001; i <= 200; i++, x += 0.001) // loop by 200 times
        { printf("printf() function test for USART0 : %3d  %5.3f\n", i, x);
          Delay_ms(500);
        }
      Beep();
    }
}
```

이 프로그램을 컴파일하여 *OK-7S256* 키트에 다운로드하고 실행한 후에 출력 결과를 확인하려면 *OK-7S256* 키트의 USART0 직렬통신 포트인 콘넥터 CN6과 퍼스널컴퓨터의 직렬포트를 RS-232C 통신 케이블로 연결해야 한다.

▣ 텍스트형 LCD 모듈에 출력하는 경우

이번에는 printf() 함수를 사용하여 텍스트형 LCD 모듈에 % 서식지정 출력을 내보내는 프로그램을 ADS v1.2에서 작성하여 보면 다음과 같다.

<프로그램 Xtest24_3.c> printf() 함수에 의한 텍스트형 LCD 모듈로의 출력 프로그램

```
/* ========================================================================== */
/*         Xtest24 3.c : User-defined printf() Function for Text LCD          */
/* ========================================================================== */
/*                          Designed and programmed by Duck-Yong Yoon in 2007.  */

#include <stdio.h>
#include "AT91SAM7S256.h"
#include "lib_AT91SAM7S256.h"
#include "OK7S256ads.h"

int fputc(int ch, FILE *f)                      // for the user-defined printf()
{
  if((ch >= 0x20) && (ch <= 0x7E))              // check from 0x20 to 0x7E
    LCD_data(ch);

  return ch;
}

int main(void)
{ unsigned int i;
  double x;

  MCU_initialize();                             // initialize AT91SAM7S256 & kit
  Delay_ms(50);                                 // wait for system stabilization
  LCD_initialize();                             // initialize text LCD
  Beep();

  LCD_string(0x80,"Integer  =   000");          // display title
  LCD_string(0xC0,"Floating = 0.000");

  while(1)
    { for(i=1, x=0.001; i <= 200; i++, x += 0.001) // loop by 200 times
        { LCD_command(0x8D);
          printf("%3d\n", i);
          LCD_command(0xCB);
          printf("%5.3f\n", x);
          Delay_ms(500);
        }
      Beep();
    }
}
```

☞ 텍스트형 LCD로 1문자를 출력할 때는 ASCII 코드값이 0x20~0x7E의 범위일 때만 정상적으로 출력하도록 하였다. 이렇게 하면 텍스트형 LCD 모듈에서 아무 의미가 없는 여러 가지의 기능제어 문자를 무시할 수 있다.

▣ 그래픽형 LCD 모듈에 출력하는 경우

이번에는 printf() 함수를 사용하여 그래픽형 LCD 모듈에 % 서식지정 출력을 내보내는 프로그램을 ADS v1.2에서 작성하여 보면 다음과 같다.

<프로그램 Xtest24_4.c> printf() 함수에 의한 그레픽형 LCD 모듈로의 출력 프로그램

```
/* ========================================================================== */
/*        Xtest24 4.c : User-defined printf() Function for Graphic LCD        */
/* ========================================================================== */
/*                          Designed and programmed by Duck-Yong Yoon in 2007.  */

#include <stdio.h>
#include "AT91SAM7S256.h"
#include "lib AT91SAM7S256.h"
#include "OK7S256ads.h"
#include "OK7S256Korean.h"

int fputc(int ch, FILE *f)                      // for the user-defined printf()
{
  if((ch >= 0x20) && (ch <= 0x7E))              // check from 0x20 to 0x7E
    GLCD_English(0,ch);

  return ch;
}

int main(void)
{ unsigned int i;
  double x;

  MCU_initialize();                             // initialize AT91SAM7S256 & kit
  Delay_ms(50);                                 // wait for system stabilization
  LCD_initialize();                             // initialize text LCD
  GLCD_clear();                                 // initialize GLCD screen
  Beep();

  LCD_string(0x80,"    printf()    ");          // display title on text LCD
  LCD_string(0xC0,"    for GLCD    ");

  GLCD_string(0,0,1," printf() 함수  ");        // display title on graphic LCD
  GLCD_string(1,0,0,"                ");
  GLCD_string(2,0,0,"  정수 =   000  ");
  GLCD_string(3,0,0,"  실수 = 0.000  ");

  while(1)
    { for(i=1, x=0.001; i <= 200; i++, x += 0.001) // loop by 200 times
        { GLCD_xy(2,11);
          printf("%3d\n", i);
          GLCD_xy(3,9);
          printf("%5.3f\n", x);
          Delay_ms(500);
        }
      Beep();
    }
}
```

☞ 그래픽형 LCD로 1문자를 출력할 때는 ASCII 코드값이 0x20~0x7E의 범위일 때만 정상적으로 출력하도록 하였다. 이렇게 하면 텍스트형 LCD 모듈에서 아무 의미가 없는 여러 가지의 기능제어 문자를 무시할 수 있다. printf()에서는 내부적으로 fputc() 함수를 통하여 무조건 1바이트씩 출력하므로 2바이트로 구성되는 한글 문자를 출력하는 기능이 수행되지 않으며, 이런 기능이 필요한 경우에는 GLCD_string() 함수로 별도 처리하면 된다.

(2) WinARM에서의 printf() 사용 프로그램

▣ DBGU 직렬통신 포트로 출력하는 경우

AT91SAM7S256에는 디버그 및 프로그램 다운로드용의 DBGU 포트를 UART 직렬통신 포트로 사용할 수도 있다. printf() 함수를 사용하여 DBGU 포트로 % 서식지정 출력을 내보내는 프로그램을 WinARM에서 작성하여 보면 다음과 같다.

<프로그램 Xtest24_5.c> printf() 함수에 의한 DBGU로의 RS-232C 출력 프로그램

```
/* ============================================================================= */
/*           Xtest24 5.c : User-defined printf() Function for DBGU             */
/* ============================================================================= */
/*                         Designed and programmed by Duck-Yong Yoon in 2007.  */

#include <stdio.h>
#include "AT91SAM7S256.h"
#include "lib AT91SAM7S256.h"
#include "OK7S256gcc.h"

unsigned int DBGU_RX_char(void)         /* receive a character by DBGU */
{
  while(!(*AT91C_DBGU_CSR & 0x0001));           // wait until RXRDY=1

  return *AT91C_DBGU_RHR;                       // receive a character
}

void DBGU_TX_char(unsigned int data)            /* transmit a character by DBGU */
{
  while(!(*AT91C_DBGU_CSR & 0x0002));           // wait until TXRDY=1

  *AT91C_DBGU_THR = data;                       // transmit a character
}

/* ===== System Call Reentrant Function ================================== */

#include <reent.h>
#include <sys/stat.h>

 ssize_t  read_r(struct _reent *r, int file, void *ptr, size_t len)
{ char c;
  int  i;
  unsigned char *p;

  p = (unsigned char*)ptr;

  for(i=0; i < len; i++)
    { c = DBGU_RX_char();
      *p++ = c;
      DBGU_TX_char(c);

      if(c == 0x0D && i <= (len - 2))
        { *p = 0x0A;
```

```
          DBGU_TX_char(0x0A);
          return i + 2;
        }
    }

  return i;
}

 ssize_t _write_r(struct _reent *r, int file, const void *ptr, size_t len)
{ int i;
  const unsigned char *p;

  p = (const unsigned char*)ptr;

  for(i=0; i < len; i++)
    { if(*p == '\n' )
        DBGU_TX_char('\r');
      DBGU_TX_char(*p++);
    }

  return len;
}

int _close_r(struct _reent *r, int file)
{
  return 0;
}

 off_t _lseek_r(struct _reent *r, int file, _off_t ptr, int dir)
{
  return (_off_t)0;
}

int _fstat_r(struct _reent *r, int file, struct stat *st)
{
  st->st_mode = S_IFCHR;
  return 0;
}

int isatty(int file)
{
  return 1;
}

extern char end[];

static char *heap_ptr;

void * _sbrk_r(struct _reent *_s_r, ptrdiff_t nbytes)
{ char *base;

  if(!heap_ptr)
    heap_ptr = end;
  base = heap_ptr;
  heap_ptr += nbytes;

  return base;
}
```

```
/* ========================================================================== */

int main(void)
{ unsigned int i;
  double x;

  MCU_initialize();                          // initialize AT91SAM7S256 & kit
  Delay_ms(50);                              // wait for system stabilization
  LCD_initialize();                          // initialize text LCD
  Beep();

  LCD_string(0x80,"    printf()    ");       // display title
  LCD_string(0xC0,"    for DBGU    ");

  AT91F_DBGU_CfgPMC();                       // enable clock of DBGU
  AT91F_DBGU_CfgPIO();                       // DTXD, DRXD by Peripheral A

  *AT91C_DBGU_MR = 0x00000800;               // normal mode, no parity
  *AT91C_DBGU_BRGR = 26;                     // MCK/16/26 = 115200 bps
  *AT91C_DBGU_CR = 0x00000150;               // TX, RX enable

  while(1)
    { for(i=1, x=0.001; i <= 200; i++, x += 0.001) // loop by 200 times
        { printf("printf() function test for DBGU : %3d  %5.3f\n", i, x);
          Delay_ms(500);
        }
      Beep();
    }
}
```

☞ ADS v1.2에서 printf() 함수를 사용하려면 사용자 프로그램에서 fputc() 함수를 하나 만들어주는 것으로 가능하였지만, WinARM에서 printf() 함수를 사용하려면 이와 같이 _read_r() 및 _write_r() 함수 이외에도 몇가지 함수를 더 만들어 주어야 한다. 이 때문에 위 소스에서 "System Call Reentrant Function"이라고 묶어진 부분 전체를 syscalls.c와 같은 별도의 파일로 작성하여 링크시키기도 한다. 그러나, 이렇게 이를 별도의 파일로 작성하면 printf() 함수에서 사용할 주변장치를 바꿀 때마다 사용자 프로그램의 소스는 물론이고 이 별도의 파일까지 찾아가서 그 내용을 수정해주어야 하는 불편이 있기 때문에 이 책에서는 이처럼 사용자 프로그램 내부에 이 루틴들을 포함시키는 방법을 사용하였다. 이 루틴들의 내용은 굳이 이해하려고 노력하지 않아도 된다. 다만, 이 루틴에서 각 주변장치에 1문자를 출력하거나 입력하는 함수를 호출하는 부분만을 상황에 따라서 수정할 수 있으면 그것으로 충분하다.

함수 _write_r()의 형식은 반드시 위와 같이 처리해야 한다. 이 함수안에서 DBGU 포트로 1문자를 출력하고 있는 것에 주목하라. 그리고, \n 문자는 상대방에서 수신하여 화면에 출력할 것에 대비하여 이렇게 2개의 ASCII 문자인 캐리지 리턴(0x0D)과 라인 피드(0x0A)로 전송하는 것이 좋다.

WinARM에서 printf() 함수를 사용하는 경우에는 위에서 그랬듯이 문자열의 끝을 반드시 \n 문자로 처리해야 한다. 만약, 문자열의 끝에 \n 문자를 넣지 않으면 이 문자열이 버퍼에만 저장되고 실제로 해당 주변장치로 출력되지 않으므로 프로그램이 올바르게 실행되지 않을 수 있다.

이 프로그램을 컴파일하여 *OK-7S256* 키트에 다운로드하고 실행한 후에 출력 결과를 확인하려면 *OK-7S256* 키트의 DBGU 직렬통신 포트인 콘넥터 CN5와 퍼스널컴퓨터의 직렬포

트를 RS-232C 동신 케이블로 연결해야 한다. 그리고, 퍼스널컴퓨터의 하이퍼터미널에서 115200 bps, 8 데이터, 패리티 없음, 1 스톱 비트로 설정하여 사용해야 한다.

▣ USART0 직렬통신 포트로 출력하는 경우

이번에는 printf() 함수를 사용하여 USART0 포트로 % 서식지정 출력을 내보내는 프로그램을 WinARM에서 작성하여 보면 다음과 같다.

<프로그램 Xtest24_6.c> printf() 함수에 의한 USART0으로의 RS-232C 출력 프로그램

```
/* ========================================================================== */
/*          Xtest24_6.c : User-defined printf() Function for USART0           */
/* ========================================================================== */
/*                          Designed and programmed by Duck-Yong Yoon in 2007.  */

#include <stdio.h>
#include "AT91SAM7S256.h"
#include "lib_AT91SAM7S256.h"
#include "OK7S256gcc.h"

unsigned int USART0_RX_char(void)              /* receive a character by USART0 */
{
  while(!(*AT91C_US0_CSR & 0x0001));           // wait until RXRDY=1

  return *AT91C_US0_RHR;                       // receive a character
}

void USART0_TX_char(unsigned int data)         /* transmit a character by USART0 */
{
  while(!(*AT91C_US0_CSR & 0x0002));           // wait until TXRDY=1

  *AT91C_US0_THR = data;                       // transmit a character
}

/* ===== System Call Reentrant Function ================================== */

#include <reent.h>
#include <sys/stat.h>

 ssize_t  read_r(struct _reent *r, int file, void *ptr, size_t len)
{ char c;
  int  i;
  unsigned char *p;

  p = (unsigned char*)ptr;

  for(i=0; i < len; i++)
    { c = USART0_RX_char();
      *p++ = c;
      USART0_TX_char(c);

      if(c == 0x0D && i <= (len - 2))
        { *p = 0x0A;
```

```
          USART0_TX_char(0x0A);
          return i + 2;
        }
    }

  return i;
}

 ssize_t _write_r(struct _reent *r, int file, const void *ptr, size_t len)
{ int i;
  const unsigned char *p;

  p = (const unsigned char*)ptr;

  for(i=0; i < len; i++)
    { if(*p == '\n' )
        USART0_TX_char('\r');
      USART0_TX_char(*p++);
    }

  return len;
}

int _close_r(struct _reent *r, int file)
{
  return 0;
}

 off_t _lseek_r(struct _reent *r, int file, _off_t ptr, int dir)
{
  return (_off_t)0;
}

int _fstat_r(struct _reent *r, int file, struct stat *st)
{
  st->st_mode = S_IFCHR;
  return 0;
}

int isatty(int file)
{
  return 1;
}

extern char end[];

static char *heap_ptr;

void * _sbrk_r(struct _reent *_s_r, ptrdiff_t nbytes)
{ char *base;

  if(!heap_ptr)
    heap_ptr = end;
  base = heap_ptr;
  heap_ptr += nbytes;

  return base;
}
```

```
/* ==================================================================== */

int main(void)
{ unsigned int i;
  double x;

  MCU_initialize();                              // initialize AT91SAM7S256 & kit
  Delay_ms(50);                                  // wait for system stabilization
  LCD_initialize();                              // initialize text LCD
  Beep();

  LCD_string(0x80,"    printf()    ");           // display title
  LCD_string(0xC0,"   for USART0   ");

  AT91F_US0_CfgPMC();                            // enable clock of USART0
  AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA,AT91C_PA6_TXD0|AT91C_PA5_RXD0,0);
                                                 // TXD0, RXD0 by Peripheral A

  *AT91C_US0_MR = 0x100008C0;                    // normal mode, no parity
                                                 // 8 data, 1 stop, LSB first, 16x
  *AT91C_US0_BRGR = 26;                          // MCK/16/26 = 115200 bps
  *AT91C_US0_CR = 0x00000150;                    // TX, RX enable

  while(1)
    { for(i=1, x=0.001; i <= 200; i++, x += 0.001) // loop by 200 times
        { printf("printf() function test for USART0 : %3d  %5.3f\n", i, x);
          Delay_ms(500);
        }
      Beep();
    }
}
```

이 프로그램을 컴파일하여 *OK-7S256* 키트에 다운로드하고 실행한 후에 출력 결과를 확인하려면 *OK-7S256* 키트의 USART0 직렬통신 포트인 콘넥터 CN6과 퍼스널컴퓨터의 직렬 포트를 RS-232C 통신 케이블로 연결해야 한다.

▣ 텍스트형 LCD 모듈에 출력하는 경우

이번에는 printf() 함수를 사용하여 텍스트형 LCD 모듈에 % 서식지정 출력을 내보내는 프로그램을 WinARM에서 작성하여 보면 다음과 같다.

<프로그램 Xtest24_7.c> printf() 함수에 의한 텍스트형 LCD 모듈로의 출력 프로그램

```
/* ==================================================================== */
/*         Xtest24_7.c : User-defined printf() Function for Text LCD    */
/* ==================================================================== */
/*                        Designed and programmed by Duck-Yong Yoon in 2007.  */
```

```
#include <stdio.h>
#include "AT91SAM7S256.h"
#include "lib_AT91SAM7S256.h"
#include "OK7S256gcc.h"

/* ===== System Call Reentrant Function ================================== */

#include <reent.h>
#include <sys/stat.h>

_ssize_t _read_r(struct _reent *r, int file, void *ptr, size_t len)
{
  return 0;
}

_ssize_t _write_r(struct _reent *r, int file, const void *ptr, size_t len)
{ int i;
  const unsigned char *p;

  p = (const unsigned char*)ptr;

  for(i=0; i < len; i++)
    if((*p >= 0x20) && (*p <= 0x7E))
      LCD_data(*p++);

  return len;
}

int _close_r(struct _reent *r, int file)
{
  return 0;
}

_off_t _lseek_r(struct _reent *r, int file, _off_t ptr, int dir)
{
  return (_off_t)0;
}

int _fstat_r(struct _reent *r, int file, struct stat *st)
{
  st->st_mode = S_IFCHR;
  return 0;
}

int isatty(int file)
{
  return 1;
}

extern char end[];

static char *heap_ptr;

void * _sbrk_r(struct _reent *_s_r, ptrdiff_t nbytes)
{ char *base;

  if(!heap_ptr)
    heap_ptr = end;
```

```
  base = heap_ptr;
  heap_ptr += nbytes;

  return base;
}

/* ------------------------------------------------------------------------ */

int main(void)
{ unsigned int i;
  double x;

  MCU_initialize();                          // initialize AT91SAM7S256 & kit
  Delay_ms(50);                              // wait for system stabilization
  LCD_initialize();                          // initialize text LCD
  Beep();

  LCD_string(0x80,"Integer  =   000");       // display title
  LCD_string(0xC0,"Floating = 0.000");

  while(1)
    { for(i=1, x=0.001; i <= 200; i++, x += 0.001) // loop by 200 times
        { LCD_command(0x8D);
          printf("%3d\n", i);
          LCD_command(0xCB);
          printf("%5.3f\n", x);
          Delay_ms(500);
        }
      Beep();
    }
}
```

☞ 텍스트형 LCD 모듈은 출력장치이므로 여기서는 절대로 scanf() 함수를 사용할 수가 없으며, 따라서 이것이 호출하여 사용할 _read_r() 함수는 형식만 유지하고 내용이 없는 빈 함수로 처리하였다.

텍스트형 LCD로 1문자를 출력할 때는 ASCII 코드값이 0x20~0x7E의 범위일 때만 정상적으로 출력하도록 하였다. 이렇게 하면 텍스트형 LCD 모듈에서 아무 의미가 없는 여러 가지의 기능제어 문자를 무시할 수 있다.

▣ 그래픽형 LCD 모듈에 출력하는 경우

이번에는 printf() 함수를 사용하여 그래픽형 LCD 모듈에 % 서식지정 출력을 내보내는 프로그램을 WinARM에서 작성하여 보면 다음과 같다.

<프로그램 Xtest24_8.c> printf() 함수에 의한 그래픽형 LCD 모듈로의 출력 프로그램

```
/* ------------------------------------------------------------------------ */
/*        Xtest24_8.c : User-defined printf() Function for Graphic LCD      */
/* ------------------------------------------------------------------------ */
/*                        Designed and programmed by Duck-Yong Yoon in 2007.  */

#include <stdio.h>
```

```
#include "AT91SAM7S256.h"
#include "lib_AT91SAM7S256.h"
#include "OK7S256gcc.h"
#include "OK7S256Korean.h"

/* ===== System Call Reentrant Function ================================== */

#include <reent.h>
#include <sys/stat.h>

_ssize_t _read_r(struct _reent *r, int file, void *ptr, size_t len)
{
  return 0;
}

_ssize_t _write_r(struct _reent *r, int file, const void *ptr, size_t len)
{ int i;
  const unsigned char *p;

  p = (const unsigned char*)ptr;

  for(i=0; i < len; i++)
   if((*p >= 0x20) && (*p <= 0x7E))
     GLCD_English(0,*p++);

  return len;
}

int _close_r(struct _reent *r, int file)
{
  return 0;
}

_off_t _lseek_r(struct _reent *r, int file, _off_t ptr, int dir)
{
  return (_off_t)0;
}

int _fstat_r(struct _reent *r, int file, struct stat *st)
{
  st->st_mode = S_IFCHR;
  return 0;
}

int isatty(int file)
{
  return 1;
}

extern char end[];

static char *heap_ptr;

void * _sbrk_r(struct _reent *_s_r, ptrdiff_t nbytes)
{ char *base;

  if(!heap_ptr)
    heap_ptr = end;
```

```
  base = heap_ptr;
  heap_ptr += nbytes;

  return base;
}

/* ============================================================================ */

int main(void)
{ unsigned int i;
  double x;

  MCU_initialize();                          // initialize AT91SAM7S256 & kit
  Delay_ms(50);                              // wait for system stabilization
  LCD_initialize();                          // initialize text LCD
  GLCD_clear();                              // initialize GLCD screen
  Beep();

  LCD_string(0x80,"    printf()    ");       // display title on text LCD
  LCD_string(0xC0,"    for GLCD    ");

  GLCD_string(0,0,1," printf() 함수  ");     // display title on graphic LCD
  GLCD_string(1,0,0,"                ");
  GLCD_string(2,0,0,"  정수 =   000  ");
  GLCD_string(3,0,0,"  실수 = 0.000  ");

  while(1)
    { for(i=1, x=0.001; i <= 200; i++, x += 0.001) // loop by 200 times
        { GLCD_xy(2,2);
          printf("정수 = %3d\n", i);
          GLCD_xy(3,2);
          printf("실수 = %5.3f\n", x);
          Delay_ms(500);
        }
      Beep();
    }
}
```

☞ 그래픽형 LCD로 1문자를 출력할 때는 ASCII 코드값이 0x20~0x7E의 범위일 때만 정상적으로 출력하도록 하였다. 이렇게 하면 텍스트형 LCD 모듈에서 아무 의미가 없는 여러 가지의 기능제어 문자를 무시할 수 있다.

ADS v1.2의 printf() 함수에서는 한글이 처리되지 않는다. 그러나, WinARM의 printf()에서는 내부적으로 _write_r() 함수를 사용하여 각 문자들을 연속으로 출력하므로 2바이트로 된 한글 문자들이 문제없이 처리된다. 물론, 한글 문자열은 굳이 printf() 함수에서 처리하지 말고 GLCD_string() 함수로 별도 처리해도 된다.

【 참고문헌 】

1. 윤덕용, ARM 시작하기 시리즈① - ARM7TDMI AT91SAM7S256으로 시작하기, 도서출판 Ohm사, 2007