OpenSTM32 Community Site | printf through uart

System Workbench for STM32

printf through uart

Posted by ftoffolon on 2015-09-04 12:24 http://www.openstm32.org/forumthread1055

Hello everybody,

first of all thanks for your useful forum that have helped me several times.

I'm working on STM32F205 processor with HAL layer from STM32Cube and, for on field debug, I need to implement a serial console to log out some events and eventually send some commands on processor through it.

I've searched on this forum and on internet but I can't find informations that enable me to do that.

On Cube examples there is one with that purouse but, becuase of it was done for all development platforms supported, I think there are missed linker informations / libraries about link my custom fpuct (with my uart write procedure) to be used by printf.
My code with printf lines compile succesfully but no strings are sent to uart (and I don't know where they go).
On some sites I've found infos about implements stubs modules required by GCC (e.g. CodeSourcery toolchain has libcnano that already support this stubs for printf redirection).

Does anyone was involved on this issues ? or does anyone have some infos to set linker properly ?

Thanks in advance
Fabio

Link **Reads**: 27651

Messages: Style: Sort: Search:

Posted by MSchultz on 2015-09-08 17:49

The way I got printf (and all other console-oriented stdio functions) to work was by creating custom implementations of the low-level I/O functions like _read() and _write().

The GCC C library makes calls to the following functions to perform low-level I/O :

```
int _read(int file, char *data, int len)
int _write(int file, char *data, int len)
int _close(int file)
int _lseek(int file, int ptr, int dir)
int _fstat(int file, struct stat *st)
int _isatty(int file)
```

These functions are implemented witihin the GCC C library as stub routines with “weak” linkage. If a declaration of any of the above functions appears in your own code, your substitute routine will override the declaration in the library and be used instead of the default (non-functional) routine.

I have copied my implementations of these functions below. You will need to substitute the calls to UART_TxBlocking() and UART_RxBlocking() with calls to whatever functions you have created that do U(S)ART I/O.

```
#ifdef __GCC__
#include  <errno.h>
#include  <stdio.h>
#include  <sys/stat.h>
#include  <sys/unistd.h>
#include "stdio_helper_gcc.h"
#include "UART.h"

#undef errno
extern int errno;

/******************************************************************************
 *
 ******************************************************************************/

void stdio_setup(int no_init)
{
    if (! no_init)
    {
        UART_Init(0);
    }
    // Turn off buffers, so I/O occurs immediately
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stderr, NULL, _IONBF, 0);
}

/******************************************************************************
 *
 ******************************************************************************/

int _read(int file, char *data, int len)
{
    int bytes_read;

    if (file != STDIN_FILENO)
    {
        errno = EBADF;
        return -1;
    }

    for (bytes_read = 0; bytes_read < len; bytes_read++)
    {
        *data = (char) UART_RxBlocking();
        data++;
    }

    return bytes_read;
}

/******************************************************************************
 *
 ******************************************************************************/

int _write(int file, char *data, int len)
{
    int bytes_written;

    if ((file != STDOUT_FILENO) && (file != STDERR_FILENO))
    {
        errno = EBADF;
        return -1;
    }

    for (bytes_written = 0; bytes_written < len; bytes_written++)
    {
        UART_TxBlocking(*data);
        data++;
    }

    return bytes_written;
}

/********************************************************************************
 *
 ********************************************************************************/

int _close(int file)
{
    return -1;
}

/********************************************************************************
 *
 ********************************************************************************/

int _lseek(int file, int ptr, int dir)
{
    return 0;
}

/********************************************************************************
 *
 ********************************************************************************/

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

/********************************************************************************
 *
 ********************************************************************************/

int _isatty(int file)
{
    if ((file == STDOUT_FILENO) ||
        (file == STDIN_FILENO) ||
        (file == STDERR_FILENO))
    {
        return 1;
    }

    errno = EBADF;
    return 0;
}
#endif
```

Link

Posted by dautrevaux on 2015-09-08 18:44

In fact there is a simpler way to do that; all these functions are already implemented in the **syscalls.c** file provided in projects created by Ac6 System

Workbench. The only think you need to provide are the blocking USART read/write routins that should be defined as:

```
int __io_putchar(int ch) {
    // Code to write character 'ch' on the UART
}

int __io_getchar(void) {
    // Code to read a character from the UART
}
```

The only thing that was not (yet) done by the provided code s to allow to open several “files” on various devices and write or read from them. For this to work you should just redefine _open, _read and _write, probably by modifying the syscalls.c file, and this will work seamlessly.

Furthermore, if you use some RTOS (like FreeRTOS) you can write your low-level I/O routines to sleep while waiting characters and send output using a buffer and interrupts; probably this would need modifying the _read and _write routines in a quite natural way.

Bernard

Link

Posted by ftoffolon on 2015-09-11 10:16

Thanks M Schultz and Bernard,

once again this forum is effective and operative.

Your suggestions works on my platform, both copy and paste code from Schultz or drag syscalls.c file from AC6 files template folder, and of course implementing low level read/write routines. For using syscalls.c, I have to insert also lines:
setvbuf(stdin, NULL, _IONBF, 0);
setvbuf(stdout, NULL, _IONBF, 0);
setvbuf(stderr, NULL, _IONBF, 0);
(as suggested by Schultz) on my init procedure to let library works properly and see my printf on terminal.

Now my printfing goes straight ahead

Thanks again

Link

Posted by dautrevaux on 2015-09-12 17:10

The three setvbuf calls you mention are meant to set all standard streams as unbuffered; however:

- the setvbuf call on stdin is really useless, as stdin is a readonly stream.
- the default setup for stderr is already unbuffered, so this call is usualy also useless
- so only the stdout call will change anything, as, by default, stdout is line buffered
    - by default printf only send characters on the serial line when it reach an end-of-line character (“\n”)
    - including “setvbuf(stdout, NULL, _IONBF, 0);” in your init routine will effectively get characters out as soon as possible

Bernard

Link

Posted by MarcoAmerena on 2015-11-03 16:39

I'm trying to printf over the programming usb on a Nucleo F401. I've tried following the syscalls.c method but still nothing is being displayed in the terminal. Can anyone please help by giving an example of what I have to do in main, I've enabled the uart in cube hal and put printf in the while loop in main.

Thanks
Marcus

Link

Posted by michaelsander on 2016-05-16 23:31

I got the _read(), _write() method by MSchultz to work, mostly.

STM32F437, using HAL from STM32 Cube.

This prints fine:

```
printf(" Hello World \n");
```

The problem is when I print with a leading newline '\n' character in the print string. This

```
printf("\n Hello World ");
```

arrives at _write as a length = 1 string containing only the newline character. The rest of the string appears to be lost, it never gets printed.

I've tried all three options of setbuf(), no buffering and line buffering work the same, as I described, losing the rest of the string after '\n'. Full buffering never prints anything.

Any ideas how I can fix this?

thanks, Michael

Link

Posted by dautrevaux on 2016-05-17 08:59

Hi Michael,

If changing the buffering does not work (I don't understand why), you can alsu use fflush(stdout) to flush the buffer when you want output to be visible before a "\n".

Bernard (Ac6)

Link

Posted by michaelsander on 2016-05-17 16:53

Thanks. I'm not sure flush() is the solution. Maybe I can give a better example:

I put a breakpoint in _write() and looked at the "len" variable.

this code with the '\n' at the end has len = 6

```
printf("abcd \n");
```

this code with the '\n' at the begining has len = 1

```
printf("\n abcd");
```

The first case has the newline at the end of the string and the second case has the newline at the start of the string.

I think in both the no buffering case and the line buffering case the printf code will flush when the '\n' character is seen. The problem is, in the second case, the "abcd" characters after the '\n' are not printed.

thanks, Michael

Link

Posted by rreignier on 2016-08-27 16:22

Hello,

Thank you for the usefull infos.
By copying the syscalls.c file into my project and adding the flollowing function at the end of my main.c, I managed to get some output on a serial monitor by calling print().

```
int __io_putchar(int ch) {
  HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
  return ch;
}
```

But in a project converted to C++, doing the exact same modifications does not work. The function _ _ io_putchar() is never called.

Does anyone know what could cause that?

Thanks.

Link

Posted by toru-ohtsuka on 2017-12-13 09:35

Hello,
I think it is need to add “extern C”. I tried and worked well.
.

```
#ifdef __cplusplus
 extern "C" {
#endif

int __io_putchar(int ch) {
  HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
  return ch;
}

#ifdef __cplusplus
}
#endif
```

Link

Posted by nickypeh on 2018-04-26 12:41

Hi ,

I am able to send the uart if i disable the newlib-nano by deleting -specs=nano.specs in the settings.

However, when i enable it back. it wasn’t able to print to the terminal.
If nano is disable, then i cant use the code printf(“Number : %d”,1); but instead i can print out the normal printf(“Hello”);

Please help here.

Link

Show posts: