



How do I use the printf function on STM32?

Asked 4 years ago Active 2 months ago Viewed 53k times



I am trying to figure out how to use the printf function to print to the serial port.

19

My current setup is [STM32CubeMX](#) generated code and SystemWorkbench32 with the [STM32F407 discovery board](#).



I see in stdio.h that the printf prototype is defined as:



9

```
int _EXFUN(printf, (const char *__restrict, ...)
             __ATTRIBUTE__((__format__ (__printf__, 1, 2))));
```

What does it mean? Where is the exact location of this function definition? What would be general point of finding out how to use this kind of function to output?

microcontroller

c

stm32

stm32f4

stm32cubemx

edited Dec 31 '15 at 11:54



Bence Kaulics

5,955

10

27

53

asked Dec 15 '15 at 0:08



user505160

722

4

11

23

5 I think you need to write your own "int _write(int file, char *ptr, int len)" to send your standard output to your serial port such as [here](#). I believe this is normally done in a file called Syscalls.c which handles "System Calls Remapping". Try Googling "Syscalls.c". – [Tut](#) Dec 15 '15 at 12:29

1 This is not a electronics problem. Go see the manual for your compiler library. – [Olin Lathrop](#) Dec 18 '15 at 12:10

Are you wanting to do printf debugging via semihosting (over the debugger) or just printf in general? – [Daniel](#) Dec 22 '15 at 2:10

As @Tut said, the _write() function is what I did. Details in my answer below. – [cp.engr](#) Jan 12 '17 at 22:37

this was a great tutorial: youtu.be/Oc58lkj-JNI – [Joseph Pighetti](#) Oct 3 '19 at 18:43

6 Answers



I got the first method from this page working on my STM32F072.

19

<http://www.openstm32.org/forumthread1055>

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and [our Terms of Service](#).



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 within 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 used STM32CubeMX to setup USART1 (`huart1`) as a serial port. Since I only wanted `printf()` , I only needed to populate the `_write()` function, which I did as follows. This is conventionally contained in `syscalls.c` .

```
#include <errno.h>
#include <sys/unistd.h> // STDOUT_FILENO, STDERR_FILENO

int _write(int file, char *data, int len)
{
    if ((file != STDOUT_FILENO) && (file != STDERR_FILENO))
    {
        errno = EBADE;
        return -1;
    }

    // arbitrary timeout 1000
    HAL_StatusTypeDef status =
        HAL_UART_Transmit(&huart1, (uint8_t*)data, len, 1000);

    // return # of bytes written - as best we can tell
    return (status == HAL_OK ? len : 0);
}
```

edited Jan 12 '17 at 22:40

answered Jan 12 '17 at 17:18



cp.engr

382 2 12

What if I use a Makefile and not an IDE? Something is missing from this to work in my Makefile project... Could anyone help? – [Tedi](#) Jan 14 '18 at 16:43

@Tedi, are you using GCC? This is a compiler-specific answer. What have you tried, and what were the results? – [cp.engr](#) Jan 15 '18 at 2:27

Yes I use GCC. I found the problem. The `_write()` function was called. The problem was in my code for sending the string. I misused Segger's RTT library but now works fine. Thanks. All working now. – [Tedi](#) Jan 15 '18 at 17:58

4

`_EXFUN` is a macro, probably containing some interesting directives that tell the compiler that it should check the format-string for being printf-compatible and ensuring that the arguments to printf match the format string.

To learn how to use printf, I suggest the [man page](#) and a bit more googling. Write some simple C programs that use printf and run them on your computer before you try transitioning to using it on the micro.

The interesting question will be "where does the printed text go?". On a unix-like system, it goes to "standard out", but a microcontroller has no such thing. The CMSIS debug libraries can send printf text to the arm semihosting debug port, i.e. into your gdb or openocd session but I've no idea what SystemWorkbench32 will do.

If you're not working in a debugger, it might make more sense to use sprintf to format the strings you want to print and then send those strings over a serial port or to whatever display you might have attached.

Beware: printf and its related code are very large. That probably doesn't matter much on a 32F407, but it is a real problem on devices with little flash.

answered Dec 15 '15 at 1:27



William Brodie-Tyrrell

2,112 6 12

IIRC `_EXFUN` marks a function to be exported, i.e. to be used in user code, and defines the calling convention. On most (embedded) platforms the default calling convention can be used. But on x86 with dynamic libraries, you might need to define the function as `__cdecl` to avoid errors. At least on newlib/cygwin, `_EXFUN` is only used to set `__cdecl`. — [erebos](#) Dec 15 '15 at 12:42

4

@AdamHaun's answer is all you need, with `sprintf()` it is easy to create a string and then send it. But if you really want a `printf()` function of your own, then [Variable Argument Functions \(va_list\)](#) is the way.

With `va_list` a custom print function looks like the following:

```
#include <stdio.h>
#include <stdarg.h>
#include <string.h>

void vprint(const char *fmt, va_list argp)
{
    char string[200];
    if(0 < vsprintf(string,fmt,argp)) // build string
```

```

    }
}

void my_printf(const char *fmt, ...) // custom printf() function
{
    va_list argp;
    va_start(argp, fmt);
    vprint(fmt, argp);
    va_end(argp);
}

```

Usage example:

```

uint16_t year = 2015;
uint8_t month = 12;
uint8_t day = 18;
char* date = "date";

// "Today's date: 2015-12-18"
my_printf("Today's %s: %d-%d-%d\r\n", date, year, month, day);

```

Note that while this solution gives you convenient function to use, but it is slower than sending raw data or using even `sprintf()`. With high datarates I think it won't be sufficient.

Another option, and probably better option is to use ST-Link, SWD debugger along with ST-Link Utility. And use **Printf via SWO viewer**, here is the manual of [ST-Link Utility](#), relevant part starts on page 31.

The Printf via SWO Viewer displays the printf data sent from the target through SWO. It allows to display some useful information on the running firmware.

edited Aug 18 '17 at 12:52



[TisteAndii](#)

1,283 7 19

answered Dec 18 '15 at 12:55



[Bence Kaulics](#)

5,955 10 27 53

you don't use va_arg, how do you step through the variable arguments? – [iouzzr](#) Feb 28 '19 at 15:36



printf() is (usually) part of the C standard library. If your version of the library comes with source code, you might find an implementation there.

1



It would probably be easier to use `sprintf()` to generate a string, then use another function to send the string through the serial port. That way all the difficult formatting is done for you and you don't have to hack up the standard library.

answered Dec 15 '15 at 1:26



[Adam Haun](#)


17.6k 4 34 78

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and [our Terms of Service](#).



Jan 13 '17 at 3:24

It might be preconfigured to send text out through the debugger connection, as Bence and William suggested in their answers. (That's not what the questioner wanted, of course.) – [Adam Haun](#) Jan 13 '17 at 14:24

That would typically only happen as a result of code you choose to link in to support your board, but regardless you change that by defining your own output function. The problem with your proposal is that it's not based on understanding how the library is *meant* to be used - rather than do that you're proposing a multi-step process for each output, which among other things will make a mess of any existing portable code that does things the normal way. – [Chris Stratton](#) Jan 13 '17 at 15:43 

STM32 is a freestanding environment. The compiler does not need to provide stdio.h - it is entirely optional. But if it does provide stdio.h, it must provide *all* of the library. The freestanding systems compilers that do implement printf tend to do it as UART communication. – [Lundin](#) Aug 18 '17 at 13:02

For those struggling, add the following to syscalls.c:

1

```
extern UART_HandleTypeDef huart1; // access huart1 instance
//extern int __io_putchar(int ch) __attribute__((weak)); // comment this out

__attribute__((weak)) int __io_putchar(int ch)
{
    HAL_StatusTypeDef status = HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xFFFF);
    return (status == HAL_OK ? ch : 0);
}
```

Connect to your COM port via putty and you should be good.

answered Sep 26 '19 at 22:34

[Michael Brown](#)

163 6

If you want a different approach and don't want to overwrite functions or declare your own, you can simply use `snprintf` to achieve the same goal. But it's not as elegant.

1

```
char buf[100];
snprintf(buf, 100, "%X %X", val1, val2);
HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
```

answered Sep 27 '19 at 8:51

[LeoDJ](#)

11 2