

Debug with ARM Semihosting

ARM semihosting is a distinctive feature of the ARM platform, that allows to use input and output functions on a host computer that get forwarded to the microcontrollers over a hardware debugger. It is helpful when there is no input/output interface dedicated for logging on the target MCU.

[#arm](#) [#stm32](#) [#debug](#) [#semihosting](#)

Last update: 2021-06-16 00:22:01

Table of Content

- 1. Debugging
- 2. Semihosting
 - 2.1. Hardware setup
 - 2.2. Software setup
 - 2.2.1. Linker options
 - 2.2.2. Exclude user system calls
 - 2.2.3. Initialize semihosting
 - 2.3. Debugger option
- 3. Debug with Semihosting

✓ Semihosting setup

1. Connect a debugger via SWD interface
2. Include semihosting lib in GCC linker `-l rdimon --specs=rdimon.specs`
3. Exclude the default `syscall.c` implementation
4. Initialize in the startup of the application with `initialise_monitor_handles();`
5. Run OpenOCD with command `monitor arm semihosting enable`

⚠ Semihosting notes

Semihosting implementation in OpenOCD is designed so that every string must be terminated with the newline character `\n` before the string appears on the OpenOCD console.

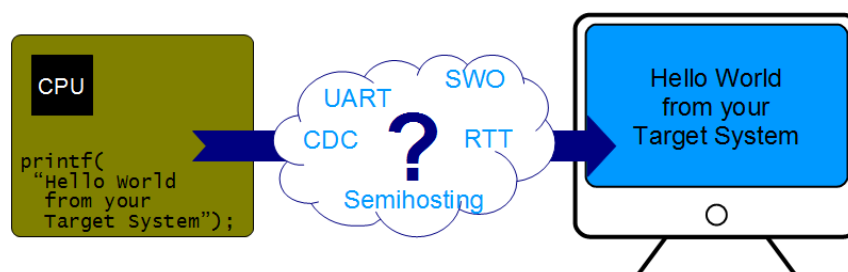
Semihosting only works during a debug session, and it's slow and affects the system performance.

CPU is halt when semihosting is executing in host machine, therefore semihosting is not suitable for realtime application.

1. Debugging

There are some debug techniques used to inspect the firmware running on ARM-based MCUs:

- **Semihosting**: build-in to every ARM chips, need adding additional library and running in debug mode
- **Redirection**: forward to a **UART port** but need using GPIO and extra hardware (USB to Serial converter), or forward to a **Virtual COM port** but need an USB peripheral
- **Serial Wire View (SWV)**: fast output over dedicated Single Wire Output (SWO) pin, but it's only available on Cortex-M3+, and this is uni-direction communication.
- **Real Time Transfer (RTT)**: extremely fast but only work with SEGGER Debugger, can have a realtime bi-direction communication.



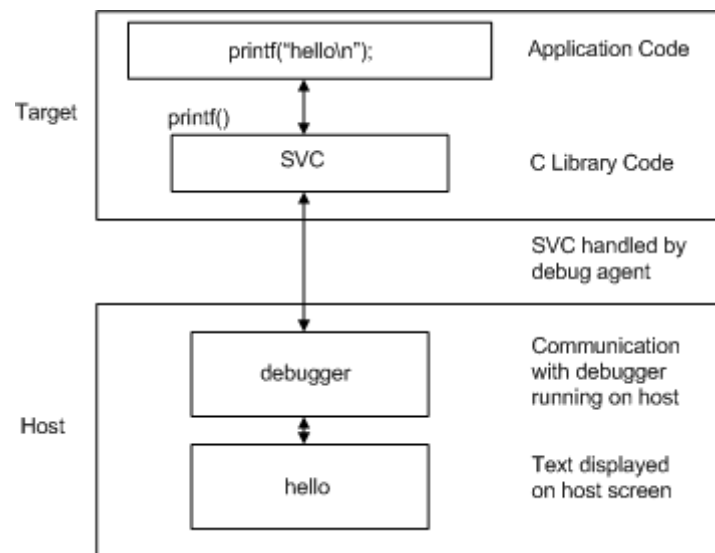
Ways to print debug

2. Semihosting

ARM semihosting is a distinctive feature of the ARM platform, that allows to use input and output functions on a host computer that get forwarded to the microcontrollers over a hardware debugger, such as `printf()` and `scanf()`, or even `fopen()`.

Semihosting is implemented by a set of defined software instructions, for example, `SVC`, that generate exceptions from program control. The application invokes the appropriate semihosting call and the debugger then handles the exception by communicating with the debugging application on the host computer.

ARM processors prior to ARMv7 use the `SVC` instructions, formerly known as `SWI` instructions, to make semihosting calls. However, for an ARMv6-M or ARMv7-M, in a Cortex-M1 or Cortex-M3 processor, semihosting is implemented using the `BKPT` instruction.

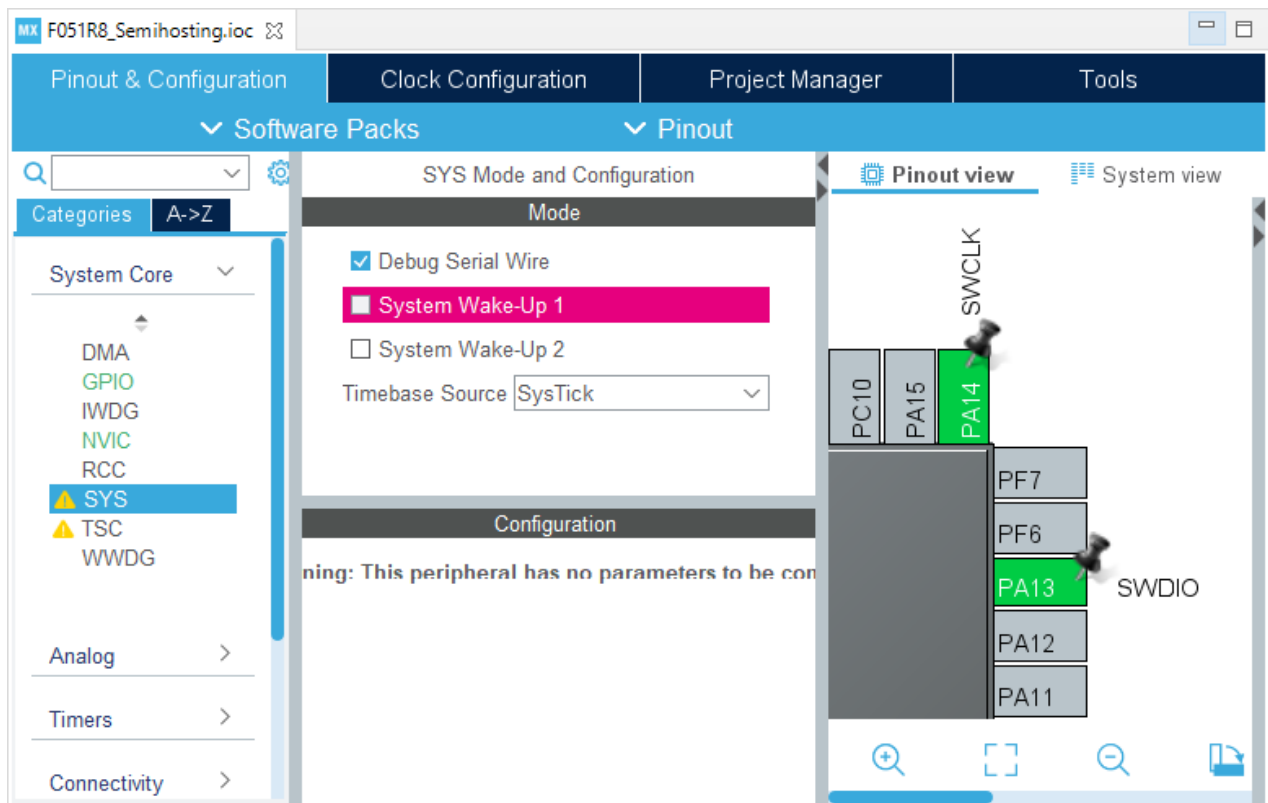


Semihosting overview

2.1. Hardware setup

Semihosting needs to be run under a debug session to communicate with semihosting-enabled debugger. In STM32, debugging channel maybe ST-LINK debugger (onboard, or external) which connects to the MCU via **SWCLK** and **SWDIO** in the SWD interface.

In the CubeMX, under the Pinout and Configuration tab, select the **SYS** module and check on the **Debug Serial Wire** option to assign the Debug function on pins **PA14** and **PA13**.



Enable Serial Wire Debug pins

2.2. Software setup

To use semihosting, it has to be set in linker options, and initialized in the main program.

2.2.1. Linker options

GNU ARM libs use `newlib` to provide standard implementation of C libraries. However, to reduce the code size and make it independent to hardware, there is a lightweight version `newlib-nano` used in MCUs.

However, `newlib-nano` does not provide an implementation of low-level system calls which are used by C standard libs, such as `print()` or `scan()`. To make the application compilable, a new library named `nosys` should be added. This library just provide an simple implementation of low-level system calls which mostly return a by-pass value.

i The lib `newlib-nano` is enabled via linker options `--specs=nano.specs`, and `nosys` is enabled via linker option `--specs=nosys.specs`. These two libs are included by default in GCC linker options in generated project.

There is an `rdimon` library that implements interrupt for some special system calls, which pauses the processor and interact with debugger host to exchange data, such as `SYS_WRITE`

(0x05) or `SYS_READ (0x06)`. This library provides low-level system calls to handling the the `newlib-nano` specs.

 The lib `rdimon` is enabled via linker option `--specs=rdimon.specs -l rdimon`

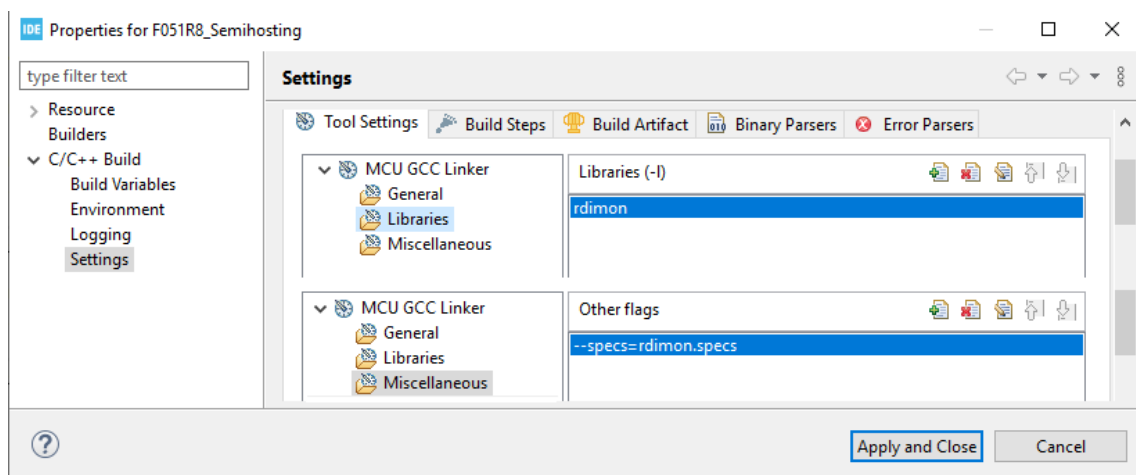
Example of using GUI config:

Open **Project Properties** » **C/C++ Build** » **Settings** » **Tool Settings tab** » **MCU GCC Linker**:

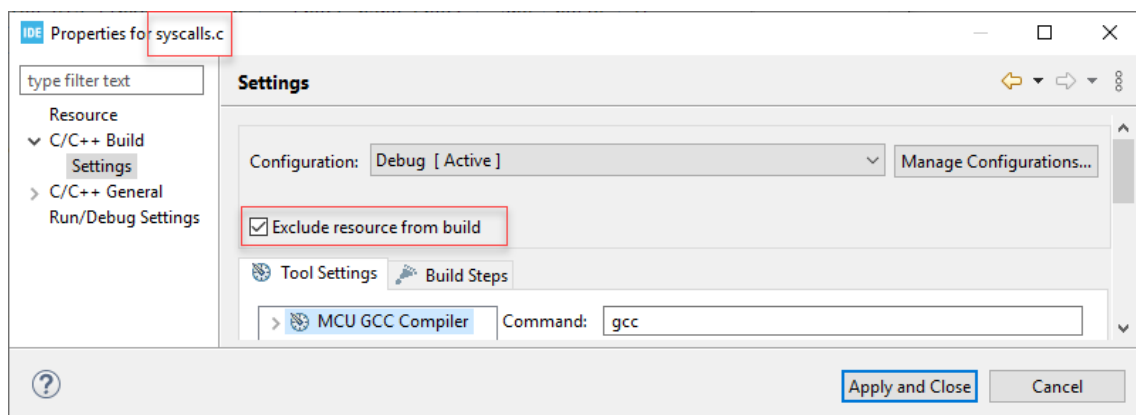
- Libraries: add `rdimon`
- Miscellaneous flags: add `--specs=rdimon.specs`

2.2.2. Exclude user system calls

In addition, STM32CubeIDE automatically generates `syscalls.c` with a simple implementation for `nosys.specs`. Must exclude `syscalls.c` from build to avoid compilation error of multiple definitions.



Add semihosting in GCC Linker



Exclude syscalls.c

2.2.3. Initialize semihosting

The `rdimon` library has to be initialized before it can run properly. It exposes a function to do that, then use it:

```
extern void initialise_monitor_handles(void);
```

in the `main()` function:

```
int main(void) {
    initialise_monitor_handles();
}
```

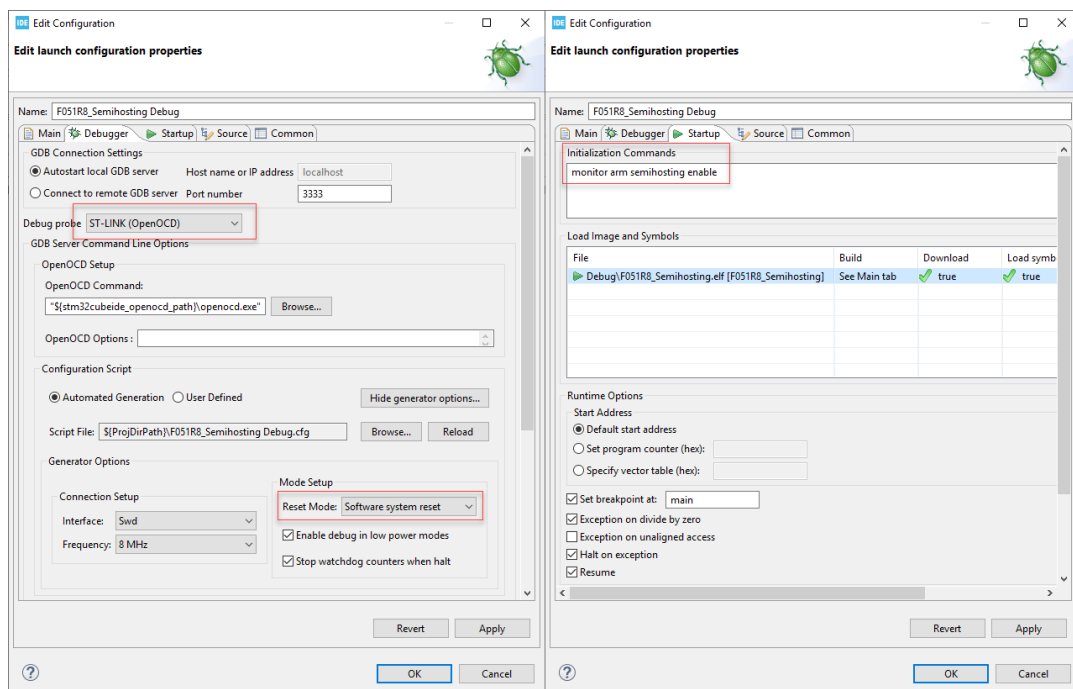
After that, the application can use `printf()`, `scanf()`, or `gets()`.

2.3. Debugger option

The final thing is to enable semihosting on debugger that will handle the interruption fired from MCUs. Debugger has to communicate with MCUs and the host computer. In some cases, it has to use *Software Reset* method to help debugger reset the target MCU.

Use **ST-LINK over OpenOCD** with the start up command:

```
monitor arm semihosting enable
```



Enable semihosting in debugger

3. Debug with Semihosting

Run the project in debug mode and then interact with MCUs. Here are some lines of code to print a message, get a string, and write to a file on the host machine:

main.c

```
#include "main.h"
#include <stdio.h>
#include <string.h>

uint8_t counter = 0;
char buffer[255];

extern void initialise_monitor_handles(void);

int main(void)
{
    initialise_monitor_handles();

    printf("Please enter your name: \n");
    gets(buffer);
    printf("\nAh, I know you, %s!\n", buffer);

    // test.out will be created in the host machine
    FILE *fd= fopen("D:\\test.out", "w+");
    if(fd) {
        fwrite(buffer, sizeof(char), strlen(buffer), fd);
        fclose(fd);
    }

    while (1)
    {
        printf("counter = %d\n", counter++);
        HAL_Delay(1000);
    }
}
```

When using `fopen()`, should specify the location, such as `~/test.out` /* on Linux */ or `D:\\test.out` /* on Windows */, if not, the target file will be created in the folder containing the `openocd.exe` program.

The screenshot shows the OpenOCD console window for an STM32 Cortex-M C/C++ application. The window has tabs for Console, Problems, Executables, Debugger Console, and Memory. The console output is as follows:

```

F051R8_Semihosting Debug [STM32 Cortex-M C/C++ Application]
Open On-Chip Debugger 0.11.0-rc2+dev-00037-g4c4dbd9 (2021-02-09-13:39)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : STLINK V2J37S0 (API v2) VID:PID 0483:3748
Info : Target voltage: 2.946817
Info : clock speed 4000 kHz
Info : stlink_dap_op_connect(connect)
Info : SWD DPIDR 0x0bb11477
Info : STM32F051R8Tx.cpu: hardware has 4 breakpoints, 2 watchpoints
Info : starting gdb server for STM32F051R8Tx.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : accepting 'gdb' connection on tcp/3333
Info : device id = 0x20006440
Info : flash size = 64kbytes
undefined debug reason 8 - target needs reset
Info : accepting 'gdb' connection on tcp/3333
undefined debug reason 8 - target needs reset
semihosting is enabled

target halted due to debug-request, current mode: Thread
xPSR: 0xc1000000 pc: 0x08000530 msp: 0x20002000, semihosting
target halted due to debug-request, current mode: Thread
xPSR: 0xc1000000 pc: 0x08000530 msp: 0x20002000, semihosting
Please enter your name:
Semihosting
Warn : keep_alive() was not invoked in the 1000 ms timelimit. GDB alive packet not sent!
Warn : negative acknowledgment, but no packet pending

Ah, I know you, Semihosting!
counter = 0
counter = 1
counter = 2

```

Interact with semihosting