Redirect the Standard IO to an UART terminal

UART is available in most of microcontrollers, so it can be used to print out debug information. Standard C has built-in functions to communicate on the standard io, such as printf(), gets(). To use those functions on an UART port, redirection technique is used, which re-writes some low-level functions.

#arm #stm32 #uart #redirect

Last update: April 28, 2021

Table of Content

UART Redirecting setup

1. Enable and setup an UART port
2. Disable standard system calls
3. Implement new system calls which communicate on the UART port
4. Register redirected system calls on application

UART Redirecting systam calls

Some modified system calls may not properly work with standard IO functions as custom code may not cover all cases, such as for file operations.

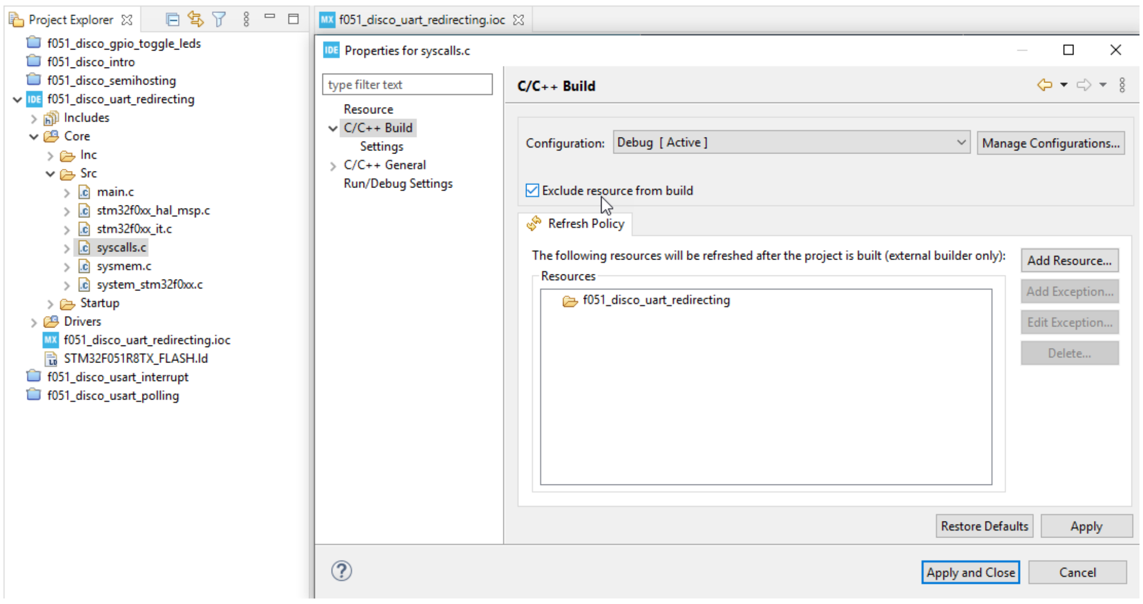
1. Enable UART port

Start a new project and follow the guide in USART - Enable UART port to enable an UART port.

2. System calls

When using Semihosting, the user *system calls* must be disabled, because the *rdimon* library already implements those functions to communicate with host machine.

Using UART Redirecting also needs to disable user system calls too. Go to the file *systamcall.c* and select **Properties** > **C/C++ Build** >> **Exclude this resources from build**.



Configuration: Debug [Active]

☑ Exclude resource from build

The following resources will be refreshed after the project is built (external builder only):

Exclude systemcall.c from build

2.1. Override system calls

Start replacing the system calls by adding in `Core/Inc/uart_redirecting.h`:

uart_redirecting.h

```c
#ifndef INC_UART_REDIRECTING_H_
#define INC_UART_REDIRECTING_H_

#include <sys/stat.h>
#include "main.h" // which includes HAL headers

/* function to set global an UART handler used to redirect */
void Set_Redirect_UART_Port(UART_HandleTypeDef *huart);

/* function declaration, see syscalls.c to get function prototype */
int _read(int file, char *ptr, int len);
int _write(int file, char *ptr, int len);
int _close(int file);
int _fstat(int file, struct stat *st);
int _isatty(int file);
int _lseek(int file, int ptr, int dir);

#endif /* INC_UART_REDIRECTING_H_ */
```

then implement those functions in `Core/Src/uart_redirecting.c`.

Here are some notes:

- Save the UART handler to use in internal functions
- `_isatty()` should return 1 to indicate the terminal
- `_fstat()` should return `S_IFCHR` to indicate character device, which returns char by char
- `_read()` should return char by char

uart_redirecting.c

```c
#include <stdio.h>
#include <errno.h>
#include "uart_redirecting.h"

/* a gloable UART handler used to redirect */
UART_HandleTypeDef *g_huart = NULL;

void Set_Redirect_UART_Port(UART_HandleTypeDef *huart) {
  g_huart = huart;
  /*
   * Disable I/O buffering for STDOUT stream, so that
   * chars are sent out as soon as they are printed.
   */
  setvbuf(stdout, NULL, _IONBF, 0);
}
```

```
int _read(int file, char *ptr, int len) {
  HAL_StatusTypeDef hstatus;
  if (g_huart == NULL) {
    return EIO;
  }
  /* read one byte only, according to _fstat returning character device type */
  hstatus = HAL_UART_Receive(g_huart, (uint8_t*) ptr, 1, HAL_MAX_DELAY);
  if (hstatus == HAL_OK)
    return 1;
  else
    return EIO;
}

int _write(int file, char *ptr, int len) {
  HAL_StatusTypeDef hstatus;
  if (g_huart == NULL) {
    return EIO;
  }
  /* write full string */
  hstatus = HAL_UART_Transmit(g_huart, (uint8_t*) ptr, len, HAL_MAX_DELAY);
  if (hstatus == HAL_OK)
    return len;
  else
    return EIO;
}

int _close(int file) {
  /* no file, just return */
  return -1;
}

int _fstat(int file, struct stat *st) {
  /* return as a character device type, read one by one character */
  st->st_mode = S_IFCHR;
  return 0;
}

int _isatty(int file) {
  /* use as a terminal */
  return 1;
}

int _lseek(int file, int ptr, int dir) {
  /* not allow seek, just read char by char */
  return 0;
}
```

2.2. Enable redirection

In *main.c*, call to the register function `Set_Redirect_UART_Port()` at the beginning of the application main. Then include `<stdio.h>` and use `printf()`, `scanf()` or `gets()`.

main.c

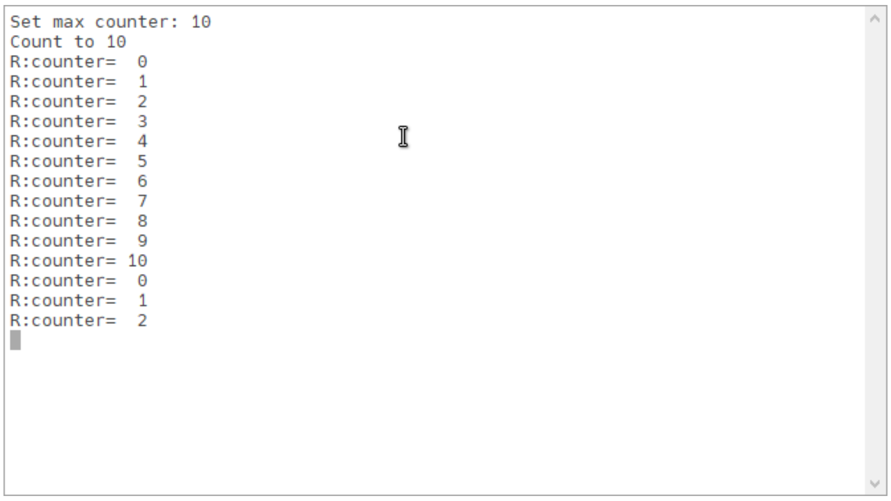
```
#include <stdio.h>
#include "uart_redirecting.h"

int main(void)
{
    char counter = 0;
    int max = 255;
    Set_Redirect_UART_Port(&huart1);

    ... other init functions ...

    printf("Set max counter: ");
    scanf("%d", &max);
    printf("\n\rCount to %d\n\r", max);
    while (1)
    {
        printf("R:counter=%3d\r\n", counter++);
        if (counter > (char) max) {
            counter = 0;
        }
        HAL_Delay(1000);
    }
}
```

Build and run on the target board, and connect the UART port to a COM port on the host machine.



```
Set max counter: 10
Count to 10
R:counter=  0
R:counter=  1
R:counter=  2
R:counter=  3
R:counter=  4
R:counter=  5
R:counter=  6
R:counter=  7
R:counter=  8
R:counter=  9
R:counter= 10
R:counter=  0
R:counter=  1
R:counter=  2
```

Interact with board using UART redirecting