STM32 - Redirect the Standard IO to an UART

Standard C has built-in functions to communicate on the standard IO, such as printf(), gets(). As UART is available in almost MCUs, it is a good idea to use an UART port as the standard IO interface. Redirection technique is implemented by re-writing some low-level functions which redirect the data to the selected UART port.

#arm #stm32 #uart #redirect

Last update: 2021-08-04 17:31:07

Table of Content

1. UART Redirection
    1.1. Enable UART port
    1.2. System calls
    1.3. Override system calls
    1.4. Enable redirection

UART Redirection setup

1. Disable the standard system calls implementation in `syscalls.c`
2. Implement new system calls which communicate on the an UART port
3. Enable and setup an UART port
4. Register redirected system calls on the target UART port
5. Use `prinf` and `scanf` for standard output and input

UART Redirection system 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. UART Redirection

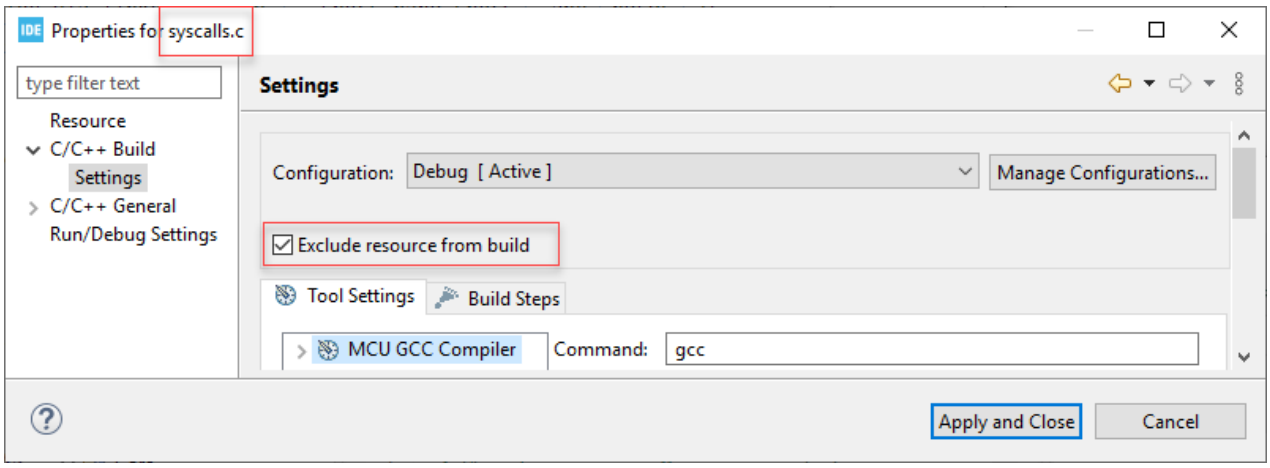
1.1. Enable UART port

Start a new project and follow the guide in UART to enable an UART port in the Normal (polling) mode.

1.2. System calls

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

Using UART Redirection also needs to disable the user's system calls too. Go to the file `syscalls.c` and select **Properties » C/C++ Build » Exclude this resources from build**.



Exclude systemcall.c from build

1.3. Override system calls

Start replacing the system calls by implement necessary functions in new module `uart_redirection`.

The header file is located in `Core/Inc/uart_redirection.h` which declares main functions to be replaced:

uart_redirection.h

```
#ifndef INC_UART_REDIRECTION_H_
#define INC_UART_REDIRECTION_H_

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

/* function to set global an UART handler used to redirect */
void Set_UART_Redirection_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_REDIRECTION_H_ */
```

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

Here are some notes:

- Save the UART handler to a global variable to use in the internal functions
- `_isatty()` should return 1 to indicate the terminal
- `_fstat()` should return `S_IFCHR` to indicate character device, which returns char by char
- `_write()` can print out as many characters as possible, and standard IO always knows the length of data
- `_read()` should return one byte as the UART is set as a character device. However, there is a reason to read one byte at a time.

  Read to buffer from standard IO

  When debugging the `_read` function, standard IO request to read 1024 bytes, and the function should return the actual read byte. The easiest way to react to unknown input length is to read one by one character only. However, this method has overhead in processing due to recalling the read function multiple times.

uart_redirection.c

```c
#include <stdio.h>
#include "uart_redirection.h"

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

void Set_UART_Redirection_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 0;
  }
  /* 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 0;
}

int _write(int file, char *ptr, int len) {
  HAL_StatusTypeDef hstatus;
  if (g_huart == NULL) {
    return 0;
  }
  /* write full string */
  hstatus = HAL_UART_Transmit(g_huart, (uint8_t*) ptr, len, HAL_MAX_DELAY);
  if (hstatus == HAL_OK)
    return len;
  else
    return 0;
}

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;
}
```

A simple implementation

A simpler implementation is to override only `_read` and `_write` function, as seen in the debugging with Serial Wire Viewer tutorial, as they are defined as weak functions and standard IO eventually call to them to at driver layer.

main.c

```
int _read(int file, char *ptr, int len) {
    HAL_StatusTypeDef hstatus;
    hstatus = HAL_UART_Receive(&huart1, (uint8_t*) ptr, 1, HAL_MAX_DELAY);
    if (hstatus == HAL_OK)
        return 1;
    else
        return 0;
}

int _write(int file, char *ptr, int len) {
    HAL_StatusTypeDef hstatus;
    hstatus = HAL_UART_Transmit(&huart1, (uint8_t*) ptr, len, HAL_MAX_DELAY);
    if (hstatus == HAL_OK)
        return len;
    else
        return 0;
}
```

\

1.4. Enable redirection

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

main.c

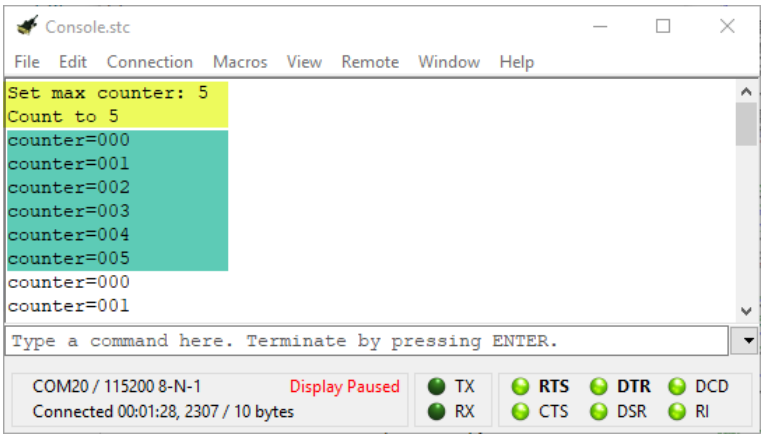
```
#include <stdio.h>
#include "uart_redirection.h"

char counter = 0;
int max = 255;

int main(void) {
    int imax = 255;
    /*Init UART1 */
    ...
    Set_Redirect_UART_Port(&huart1);
```

```
    printf("Set max counter: ");
    scanf("%d", &imax);
    max = (unsigned char)imax;
    printf("Count to %d\n\r", max);
    while (1)
    {
        printf("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.



Interaction by UART Redirection

⚠ Current implementation use UART Polling mode with reading / writing one byte at a time, therefore, it has some overhead and affect to the system performance.