Remotely GDB USART1 Project with STM32F429I Discovery Board

1 Introduction

In this article we are going to GDB remotely debug an USART1 project on STM32F429 Discovery board. System clock configuration and USART fractional buad rate calculation will also be introduced.

2 Configurations

Download and build the USART1 project, put it under the same folder with STM32F429I-Discovery Firmware. Build and flash the binary to the Discovery board.

```
% git clone https://github.com/KunYi/stm32F429-usart1.git
% wget
http://www.st.com/st-web-ui/static/active/en/st_prod_software_internet/resource/technical/software/firmware/stsw-stm32138.zip
% unzip stsw-stm32138.zip
% cd stm32F429-usart1
% make
% make flash
```

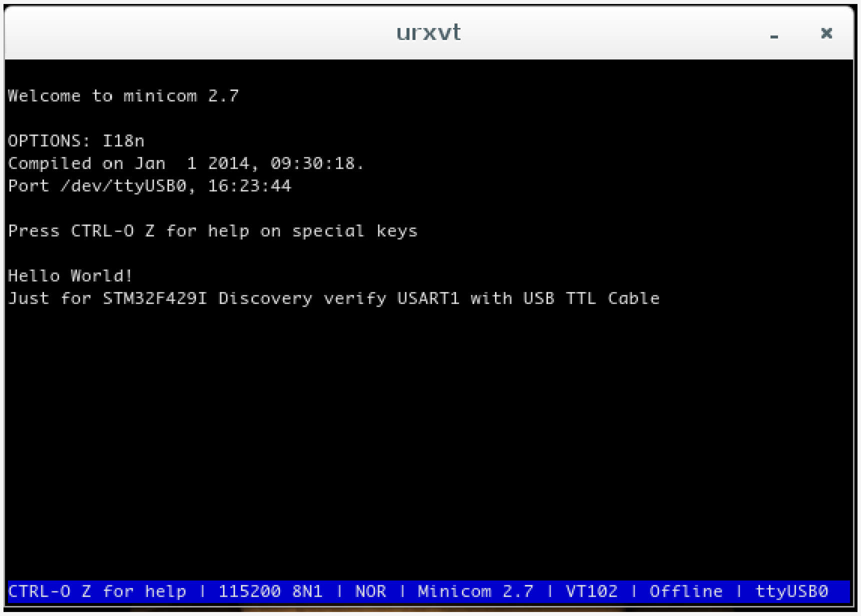
Please reference my previous post for Toolchain or flash tools installation.
If your host is a 64-bit Debian (or Ubuntu) and encounter the followed error:

```
% arm-none-eabi-gdb
arm-none-eabi-gdb: error while loading shared libraries: libncurses.so.5: wrong ELF class: ELFCLASS64
```

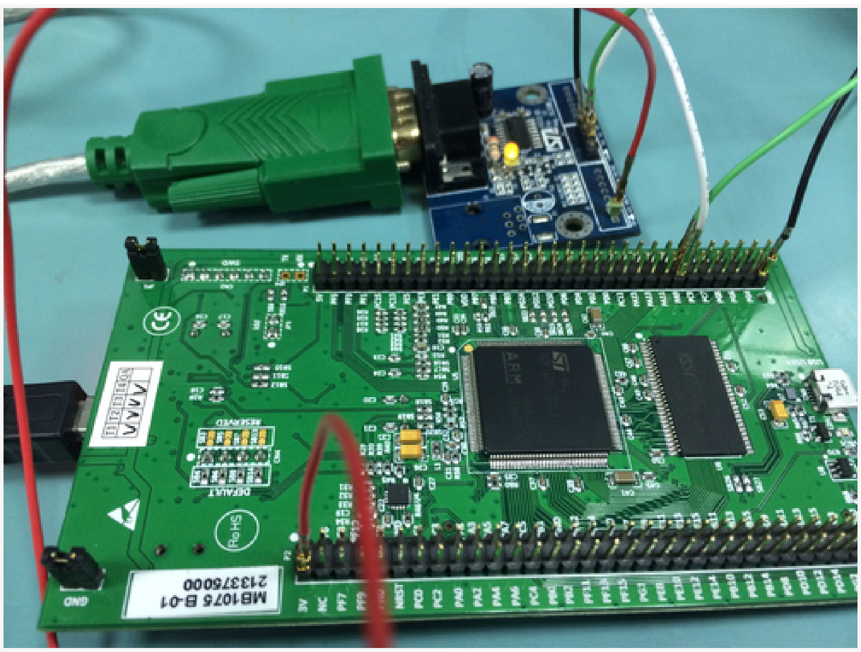
It means that the 32-bit ABI variant of ncurses library is needed:

```
% sudo apt-get install lib32ncurses5
```

When powering up, it configures GPIO PA9 and PA10 as USART1 (AF7), writes "Hello World" strings to USART1, then waits to echo user's inputs. You will see the prints by attaching the TTY with a serial terminal such as minicom, screen, or kermit.



Note. If you are using a USB-to-RS232 cable, a RS232 shifter is needed to convert the RS232 signals to TTL/CMOS level signals from the microcontroller.



3 Remote GDB

Execute the st-util acting as an gdb server to listen at port number 4242.

```
% st-util
2014-03-21T12:52:12 INFO src/stlink-common.c: Loading device parameters....
2014-03-21T12:52:12 INFO src/stlink-common.c: Device connected is: F42x and F43x device, id 0x10036419
2014-03-21T12:52:12 INFO src/stlink-common.c: SRAM size: 0x30000 bytes (192 KiB), Flash: 0x200000
```

```
bytes (2048 KiB) in pages of 16384 bytes
Chip ID is 00000419, Core ID is  2ba01477.
Target voltage is 2870 mV.
Listening at *:4242...
```

In another terminal, start the gdb, and configure the remote target with port number euqal to 4242.

```
% arm-none-eabi-gdb usart1.elf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.6.0.20131129-cvs
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-linux-gnu --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/winfred/quadcopter/stm32F429-usart1/usart1.elf...done.
(gdb) target remote :4242
Remote debugging using :4242
Reset_Handler () at startup_stm32f429_439xx.S:75
75        movs  r1, #0
```

Upon GDB connected, you can happily remote debug with GDB, common commands: step, next, continue, break, info…

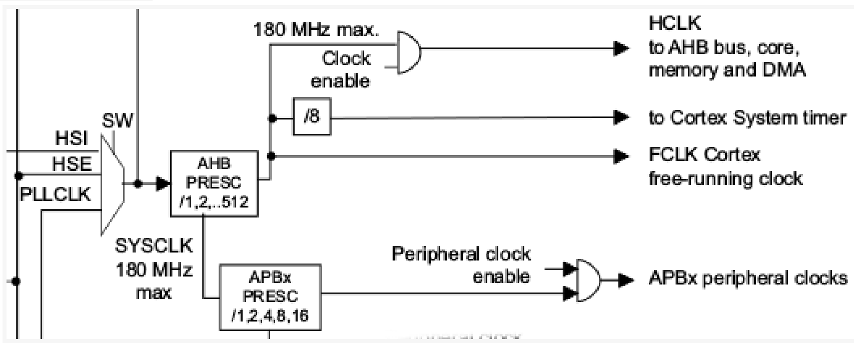
```
(gdb) n
76        b  LoopCopyDataInit
(gdb) b main
Breakpoint 1 at 0x8000284: file main.c, line 104.
(gdb) info reg
r0             0x0      0
r1             0x0      0
r2             0x0      0
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x20030000       0x20030000
lr             0xffffffff       4294967295
pc             0x800049a        0x800049a <Reset_Handler+2>
cpsr           0x41000000       1090519040
```

4 System and Pheripheral Clock Configuration

As you could see while starting GDB, the entry point of the project is Reset_Handler. It is specified in the linker script:

```
ENTRY(Reset_Handler)
```

It copies the data segment from flash to SRAM, initializes the bss segment, calls the clock system intitialization function, SystemInit(), and finally calls the application entry point, main().
STM32F429 clock tree is like:



In this project, it is configured to run at 180 MHz, with HSE (8MHz) used to clock the PLL, and the PLL is used as system clock source.
SetSysClock() is called by SystemInit() to configure the system clock source, PLL Multiplier and Divider factors:

```
static void SetSysClock(void)
{
  RCC->CR |= ((uint32_t)RCC_CR_HSEON);
  ...
  if (HSEStatus == (uint32_t)0x01)
  {
    /* Configure the main PLL */
    RCC->PLLCFGR = PLL_M | (PLL_N << 6) | (((PLL_P >> 1) -1) << 16) |
                   (RCC_PLLCFGR_PLLSRC_HSE) | (PLL_Q << 24);

  }
  ...
  /* Enable the main PLL */
  RCC->CR |= RCC_CR_PLLON;
  ...
}
```

Register RCC_PLLCFGR is used to configure the PLL clock outputs according to the formulas:

$f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLLN / PLLM)$
$= 8\ MHz \times (360 / 8)$
$= 360\ MHz$

$f_{(PLL\ general\ clock\ output)} = f_{(VCO\ clock)} / PLLP$
$= 360\ MHz / 2$

```
              = 180 MHz
```

Thus the system clock is 180 MHz.

SetSysClock() also initializes AHB/APBx prescalers:

```
static void SetSysClock(void)
{
 ...
 /* HCLK = SYSCLK / 1*/
 RCC->CFGR |= RCC_CFGR_HPRE_DIV1;

 /* PCLK2 = HCLK / 2*/
 RCC->CFGR |= RCC_CFGR_PPRE2_DIV2;

 /* PCLK1 = HCLK / 4*/
 RCC->CFGR |= RCC_CFGR_PPRE1_DIV4;
 ...
}
```

AHB prescaler is set to 1, so HCLK to AHB bus, core, memory and DMA will also be 180 MHz. And APB2 prescacler is set to 2, so APB2 peripheral clock will be 90 MHz.

5 Baud Rate Calculation

The oversampling method can be selected by programming the OVER8 bit in the USART_CR1 register and can be either 16 or 8 times the baud rate clock. In this example, oversampling by 16 (OVER8 = 0) is selected to increase the tolerance of the receiver to clock deviations.

Buad rate for standard USART:
Tx/Rx baud = f_{CK} / (8 x (2 - OVER8) x USARTDIV)

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register. When OVER8=0, the fractional part is coded on 4 bits and programmed by the DIV_fraction[3:0] bits in theUSART_BRR register.

In this example, baud rate is 115200,
USARTDIV = f_{CK} / (8 x (2 - OVER8) x baud)
= 90 MHz / (8 x 2 x 115200)
= 48.828125

So the Mantissa would be 48 (0x30), and the Fraction would 0.828125 x 16 = 13.25 ~= 13 (0xD).
USART_BRR register needs to set to 0x30D.

6 Verifying with GDB

Let's verify the configurations with GDB.

```
% arm-none-eabi-gdb usart1.elf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.6.0.20131129-cvs
```

```
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-linux-gnu --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/winfred/quadcopter/stm32F429-usart1/usart1.elf...done.
(gdb) tar rem :4242
Remote debugging using :4242
Reset_Handler () at startup_stm32f429_439xx.S:75
75        movs  r1, #0
(gdb) b stm32f4xx_usart.c:313

Breakpoint 1 at 0x8000f0e: file
../STM32F429I-Discovery_FW_V1.0.1/Libraries/STM32F4xx_StdPeriph_Driver/src/stm32f4xx_usart.c, line
313.
(gdb) c
Continuing.
Note: automatically using hardware breakpoints for read-only addresses.

Breakpoint 1, USART_Init (USARTx=USARTx@entry=0x40011000,
    USART_InitStruct=USART_InitStruct@entry=0x2002ffd8)
    at
../STM32F429I-Discovery_FW_V1.0.1/Libraries/STM32F4xx_StdPeriph_Driver/src/stm32f4xx_usart.c:314
314       if ((USARTx == USART1) || (USARTx == USART6))
(gdb) p RCC_ClocksStatus
$1 = {SYSCLK_Frequency = 180000000, HCLK_Frequency = 180000000,
  PCLK1_Frequency = 45000000, PCLK2_Frequency = 90000000}
(gdb) b stm32f4xx_usart.c:348
Breakpoint 2 at 0x8000f60: file
../STM32F429I-Discovery_FW_V1.0.1/Libraries/STM32F4xx_StdPeriph_Driver/src/stm32f4xx_usart.c, line
348.
(gdb) c
Continuing.

Breakpoint 2, USART_Init (USARTx=USARTx@entry=0x40011000,
    USART_InitStruct=USART_InitStruct@entry=0x2002ffd8)
    at
../STM32F429I-Discovery_FW_V1.0.1/Libraries/STM32F4xx_StdPeriph_Driver/src/stm32f4xx_usart.c:350
350       USARTx->BRR = (uint16_t)tmpreg;
(gdb) p/x tmpreg
$3 = 0x30d
```

7 References

- OMAPpedia : Minicom
- GitHub : stm32F429-usart1
- Wikipedia
  - UART
  - RS-232
- Connecting Serial Console to the STM32F429 Discovery
- Debugging with GDB
- GNU GDB Debugger Command Cheat Sheet
- 32F429IDISCOVERY
  - Reference Manual - STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx and STM32F43xxx advanced ARM-based 32-bit MCUs
- STM32 Evaluation Tools Forum
  - STM32F429 USART1 TX sending garbage
  - {STM32F4-Discovery} USART6 Problem