STM32 Bootloader

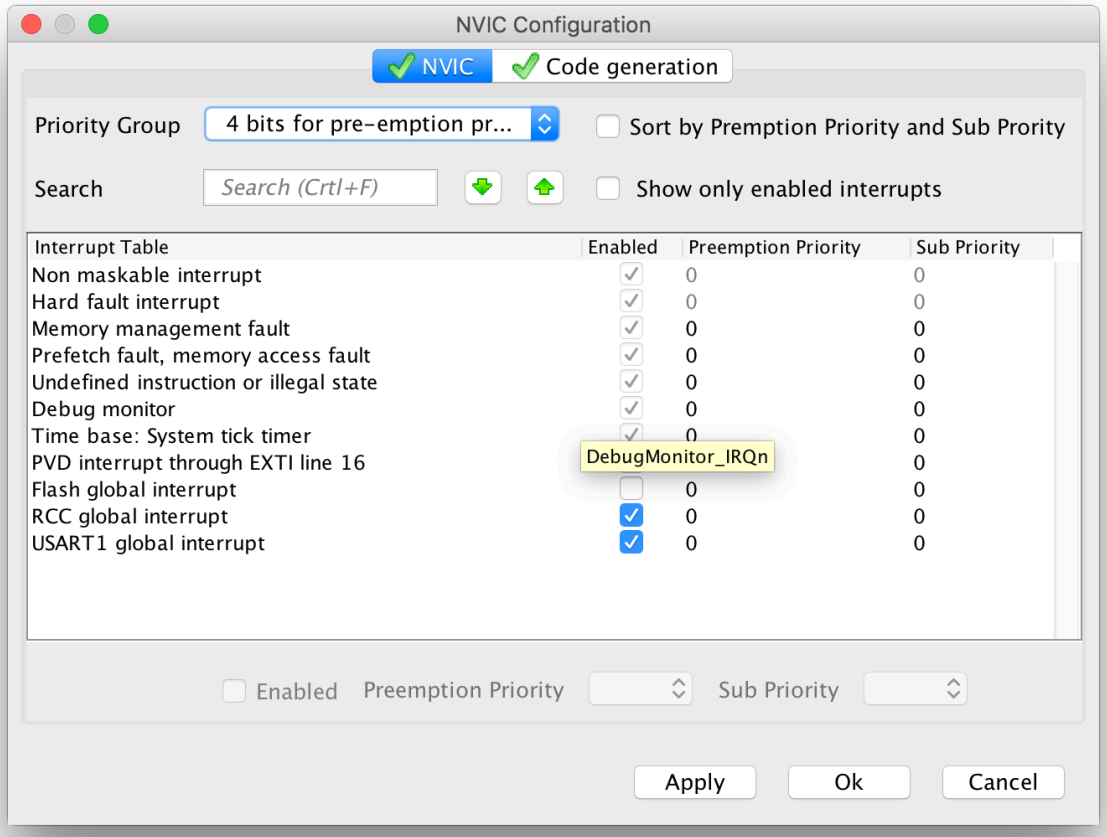
实现一个简易的bootloader，能通过串口执行两条最简单的指令：

- **peek addr：** 以一个字为单位读取内存中 addr 位置的数据，并以十六进制的形式输出
- **poke addr data：** 以一个字为单位修改内存中 addr 位置的数据为 data

串口中断

串口发送在上一个工程已经实现，而串口接收需要设置串口中断。同样利用 CubeMX 可以方便地完成串口中断的设置。

Configuration 选项卡中设置具体功能。选择 Connnectivity 中的 USART1，设置波特率，之后要设置允许串口 USART1 中断。



生成工程后，`MX_USART1_UART_Init()` 函数执行串口初始化并设置中断优先级。

```c
/* USART1 init function */
void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  HAL_UART_Init(&huart1);

  /* Peripheral interrupt init */
  HAL_NVIC_SetPriority(USART1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(USART1_IRQn);
}
```

串口中断时会执行 `USART1_IRQHandler()` 函数处理中断，而且要通过 `HAL_UART_Receive_IT()` 设置串口数据的存储位置，该函数会设置存储位置和接受长度，当接收的数据达到指定长度时，会进入中断回调函数 `HAL_UART_RxCpltCallback()` 处理中断。

环形缓冲区

串口中断输入的数据往往不能及时处理，所以要设置数据的缓冲区，采用环形队列可以起到重复利用内存的作用。设置串口中断的接受长度为 1，即每次接受一字节就进入回调函数，在回调函数中将接收到一字节数据存入环形缓冲区，并将下一次串口的接收位置置为队列的尾地址。

注：由于 PC 端的串口工具不能显示键盘发送的数据，所以在板卡端要将每次收到的字节返回到 PC 端显示。

```
// 串口中断回调函数
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle) {

    // 接收到的字符发送回PC端显示
    char c = *uart_rcv.rear;
    putchar(c);
    if (c == '\r') putchar(c);

    // 缓冲区尾指针移动
    ptrInc(&uart_rcv.rear, uart_rcv.buffer, BUFFSIZE);

    // 队列满，清除最旧数据
    if (uart_rcv.rear == uart_rcv.front)
      ptrInc(&uart_rcv.front, uart_rcv.buffer, BUFFSIZE);

    // 重新设置读取的位置以及读取的长度
    HAL_UART_Receive_IT(UartHandle, uart_rcv.rear, 1);
}
```

对于环形缓冲区的操作，可以封装出各种函数。

```
int uartGetChar(int port);
void uartPutChar(int port, char c);
int uartPeek(int port);
void uartPushback(int port);
int uartHasNewLine(int port);
int uartReadLine(int port, char* buf, int maxlen);
void uartWriteLine(int port, char* buf);
void uartPrintln(int port, char* buf);
void uartPrintf(int port, const char *fmt, ...);
```

`uart_readline(uint8_t *buf, uint16_t maxlen)` 函数在缓冲区内读取最大为 maxlen 长度的数据，并存储在 buf 指针所指向的位置，当遇到换行符结束。

```
int8_t uart_readline(uint8_t *buf, uint16_t maxlen) {
    int count = 0;
    while(count++ < maxlen - 1) {
        if(uart_rcv.front != uart_rcv.rear){
            *buf = *uart_rcv.front;
            ptrInc(&uart_rcv.front, uart_rcv.buffer, BUFFSIZE);
            if (*buf == ENTER) break;
            buf++;
        }
    }
    *buf = '\0';
    return count;
}
```

Bootloader 实现

bootloader 的作用即是与 PC 进行沟通并交换数据，将 PC 端的可执行代码通过串口或其它方式传输到嵌入式板卡的存储器的某个地址，然后将控制权转移到装入的程序。

简单起见，我们只实现了单个 32 位数据的 bootloader，peek 操作读取内存中 addr 位置的数据，poke 操作修改内存中 addr 位置的数据为 data。

在程序中设置一段内存，让 peek 和 poke 的操作限制在这段地址空间。

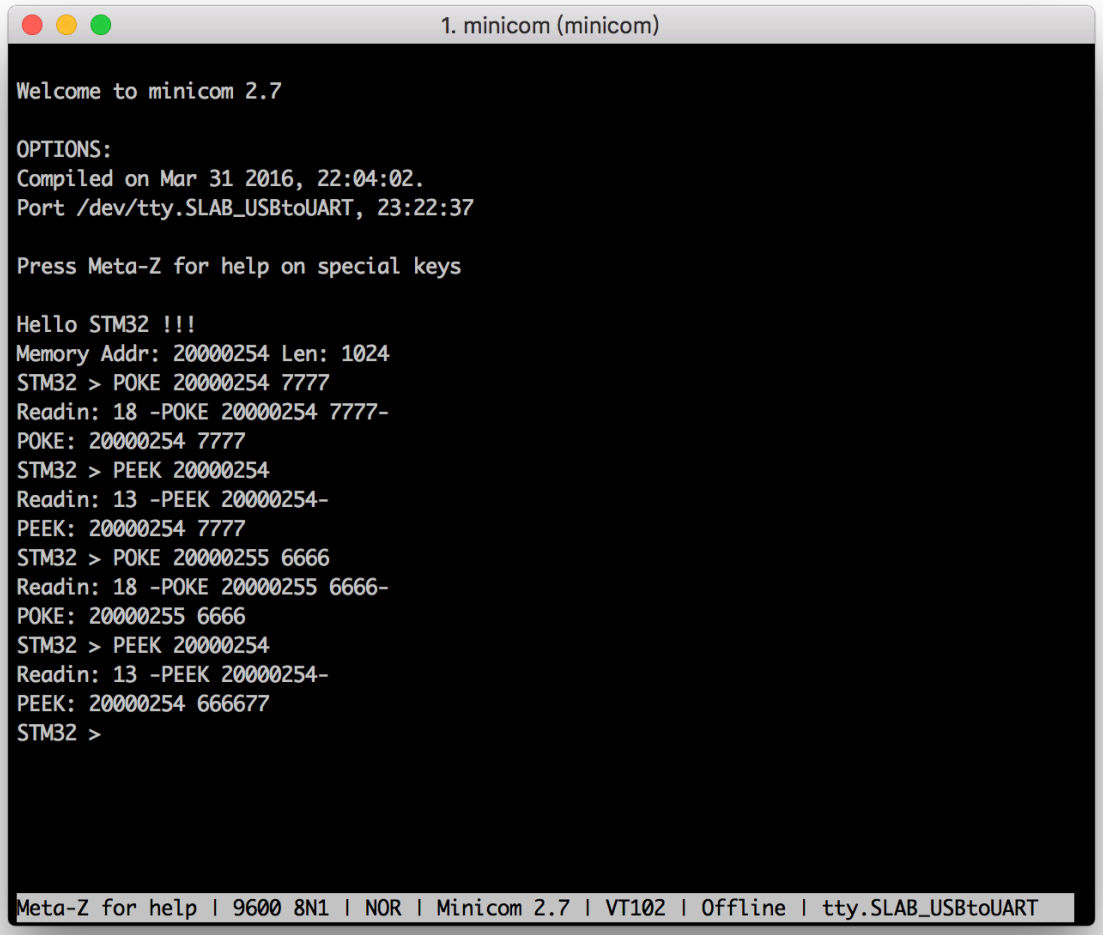
```
unsigned int Memory[MEMSIZE];
```

主循环中每次读取输入的一行数据，并判断操作类型，执行相应操作。

```c
/* Infinite loop */
while (1) {
  /* USER CODE BEGIN 3 */
    int count = 0;
    char str[100];
    char s[100];
    char cmd[100];
    int addr = 0, data = 0;

    printf("STM32 > ");
    count = uart_readline((uint8_t*)str, 100);
    sscanf(str, "%s", cmd);
    if (strcmp(cmd, "PEEK") == 0){
        sscanf(str + 5, "%x %s", &addr, s);
        sprintf(s, "PEEK: %x %x\r\n", addr, *((int*)addr));
        printf(s);
    }else if(strcmp(cmd, "POKE") == 0){
        sscanf(str + 5, "%x %x %s", &addr, &data, s);
        *((int*)addr) = data;
        sprintf(s, "POKE: %x %x\r\n", addr, *((int*)addr));
        printf(s);
    }else{
        printf("ERROR");
    }
}
```

采用类似命令行的方式，每行之前显示提示。


1. minicom (minicom)
Welcome to minicom 2.7
OPTIONS:
Compiled on Mar 31 2016, 22:04:02.
Port /dev/tty.SLAB_USBtoUART, 23:22:37
Press Meta-Z for help on special keys
Hello STM32 !!!
Memory Addr: 20000254 Len: 1024
STM32 > POKE 20000254 7777
Readin: 18 -POKE 20000254 7777-
POKE: 20000254 7777
STM32 > PEEK 20000254
Readin: 13 -PEEK 20000254-
PEEK: 20000254 7777
STM32 > POKE 20000255 6666
Readin: 18 -POKE 20000255 6666-
POKE: 20000255 6666
STM32 > PEEK 20000254
Readin: 13 -PEEK 20000254-
PEEK: 20000254 666677
STM32 >
Meta-Z for help | 9600 8N1 | NOR | Minicom 2.7 | VT102 | Offline | tty.SLAB_USBtoUART