USING THE NUCLEO-G070RB

The NUCLEO-G070RB is a newer version of the Nucelo development board family. It is similar to the NUCLEO-F030R8; however, the board has been updated to add two additional LEDs, and the serial I/O (UART) is more complex.

The examples in the book will work with the NUCLEO-G070RB, but they require some minor adjustments to do so. We have provided working NUCLEO-G070RB examples on the website; however, you should be aware of the differences between the examples and the book.

When creating programs for the NUCLEO-G070RB, follow the procedure in the book and select the NUCLEO-G070RB board instead of the NUCLEO-F030R8 board when creating your project. If you are copying the code in the book, you'll have to tweak it as indicated below.

NOTE *The initial* main.c *that's generated uses the following includes for the NUCLEO-F030R8:*

```
#include "stm32f0xx.h"
#include "stm32f0xx_nucleo.h"
```

For the NUCLEO-G070RB, the generated includes are:

```
#include "stm32g0xx.h"
#include "stm32g0xx_nucleo.h"
```

03.blink

The NUCLEO-F030R8 has two LEDs. The user LED (the one you can blink) is numbered LED2 (labeled LD2 on the board). The NUCLEO-G070RB has four LEDs. The user LED is LED4. *03.blink* works if you change all references from `LED2` to `LED4`. For example:

```
GPIO_InitStruct.Pin = LED2_PIN;
```

must be changed to:

```
GPIO_InitStruct.Pin = LED4_PIN;
```

05.button

Again, `LED2` must be changed to `LED4`.

07.house

Works unmodified.

08.serial

`LED2` must be changed to `LED4`.

The HAL firmware has been changed so that it requires a change to a configuration file to bring in the serial I/O library. Edit the file *HAL_Driver/Inc/stm32g0xx_hal_conf.h* and change the line:

```
/* #define HAL_UART_MODULE_ENABLED */
```

to:

```
#define HAL_UART_MODULE_ENABLED
```

08.serial.int

`LED2` must be changed to `LED4`.

Change the *stm32g0xx_hal_conf.h* file to enable the UART code. (See *08.serial.*)

The UART has been updated and contains additional features. Because of this, the names of the interrupt status bits have changed. You will have to change the following constants in the code:

Change `F030R8` to `G070RB`.

Change `USART_ISR_TXE` to `USART_ISR_TXE_TXFNF`.

Change `USART_CR1_TXEIE` to `USART_CR1_TXEIE_TXFNFIE`.

08.serial.buffer.bad

All changes for *08.serial.int* need to be made.

08.serial.buffer.good

All changes for *08.serial.int* need to be made.

08.serial.buffer.ins

All changes for *08.serial.int* need to be made.

10.linker

No changes are needed.

10.config

LED2 must be changed to LED4.

The NUCLEO-G070RB has renamed the FLASH memory section to ROM. Also, the size of the RAM and flash (ROM) are different, so the *LinkerScript.ld* needs some adjustment. The MEMORY section should look like this:

```
MEMORY
{
    RAM (xrw)          : ORIGIN = 0x20000000,        LENGTH = 32K
    ROM (rx)           : ORIGIN = 0x8000000,         LENGTH = 124K
    CONFIG (rwx)       : ORIGIN = 0x8000000+128K-4K, LENGTH = 4K
}
THE_CONFIG = 0x8000000+128K-4K;
```

The API for dealing with flash is different as well. The minimum unit of memory we can change is a double word (64-bit) value, so we've updated the type of resetCount and made it a uint64_t:

```
uint64_t resetCount __attribute__((section(".config.keep"))) = 0;
// # times we've been reset
```

Updating the counter is now different as well because of the different API:

```
/**
 * Update the resetCounter.
 *
 * In C this would be ++resetCounter. Because we are dealing
 * with flash, this is a much more difficult operation.
 */
static HAL_StatusTypeDef updateCounter(void) {
    HAL_FLASH_Unlock(); // Allow flash to be modified.
    uint64_t newResetCount = resetCount + 1;    // Next value for the reset count

    uint32_t pageError = 0;     // Error indication from the erase operation
    // Tell the FLASH system to erase resetCounter (and the rest of the page).
    static FLASH_EraseInitTypeDef eraseInfo = {     // ❶
        .TypeErase = FLASH_TYPEERASE_PAGES,     // Going to erase 1 page
        .Page = GetPage((uint32_t)&resetCount), // The start of the page
        .NbPages = 1                            // One page to erase
    };

    // Erase the page and get the result.
    HAL_StatusTypeDef result = HAL_FLASHEx_Erase(&eraseInfo, &pageError);
    if (result != HAL_OK) {
        HAL_FLASH_Lock();
        return (result);
    }
    // Program the new reset counter into the flash. ❷
    result = HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD,
            (uint32_t)&resetCount, newResetCount);
    if (result != HAL_OK) {
        HAL_FLASH_Lock();
        return (result);
    }
    return (result);
}
```

First, the structure used to erase the page has changed ❶. We have to give it a page number, not an address. The function GetPage turns an address into a page number:

```
static inline uint32_t GetPage(const uint32_t Addr)
{
    return (Addr - FLASH_BASE) / FLASH_PAGE_SIZE;;
}
```

Second, the API to actually write the data to memory has changed ❷. We now use the HAL_FLASH_Program API. The FLASH_TYPEPROGRAM_DOUBLEWORD tells the API to write a double word (4-byte) value to memory. The other choice is FLASH_TYPEPROGRAM_FAST, which writes 32 double words (128 bytes) to memory.

Finally, the return codes have changed. Our error handling logic in *main.c* needed to be updated:

```
    HAL_StatusTypeDef status = updateCounter();

    switch (status) {
        case HAL_OK:
            // Nothing, this is correct.
            break;
        case HAL_ERROR:
            myPuts("HAL_ERROR");
            break;
        case HAL_BUSY:
            myPuts("HAL_BUSY");
            break;
        case HAL_TIMEOUT:
            myPuts("HAL_TIMEOUT");
            break;
       default:
            myPuts("**unknown error code**");
            break;
    }
```

You can find the full program in the NUCLEO-G070RB examples, which are available for download from *https://nostarch.com/bare-metal-c*.