FLASH Memory

The STM32F302R8 has a built-in 64 Kb FLASH memory that is used to store your program. However, it is also possible to use it for storing data that will be saved even if the device is powered off.

The FLASH memory is divided into 32 “pages” each containing 2048 bits of information. When you want to write to a given address in the FLASH memory, you first have to clear the entire page containing that address. This means you have to be very careful; if you accidentally erase some of your program all hell will break loose!

The FLASH memory starts at address 0x08000000 and ends at 0x08010000. If we only use the last page which starts at address 0x0800F800 to store our data, we will have 62 kB available for the program itself. That *should* be sufficient, but you still have to be careful.

Reading from FLASH

Reading from the FLASH is fairly straight forward: we simply read from a specific address in memory and see what is stored there. Here is an example:

```
uint32_t address = 0x0800F800;
uint16_t tempVal;
for ( int i = 0 ; i < 10 ; i++ ){

    tempVal = *(uint16_t *)(address + i * 2);  // Read Command

    printf("%d ", tempVal);
}
```

This piece of code reads the first 10 elements of data stored in the last page of the FLASH memory. Note that everything should be stored as `uint16_t`, so if you have an 8-bit number you have to convert it and if you have a 32 bit number you have to split it up (technically you can write 32 bit numbers aswell, but it’s a lot easier to stick with just 16-bit).

The code works by incrementing the `address` variable for each item, typecasting it to a pointer to a `uint16_t`, and then looking at the value stored at the address. Note that the address has to be incremented by 2 because a 16-bit variable takes up two bytes.

Writing to FLASH

Writing to FLASH is a bit more involved.

First we unlock it by writing:

```
FLASH_Unlock();
FLASH_ClearFlag( FLASH_FLAG_EOP | FLASH_FLAG_PGERR |
    FLASH_FLAG_WRPERR );
```

Now we are in the danger zone, where you can easily mess up your entire program!

The next step is erase the page we wish to write to:

```
FLASH_ErasePage( address );
```

Then we write our data:

```
    for ( int i = 0 ; i < 10 ; i++ ){
        FLASH_ProgramHalfWord(address + i * 2, data[i]);
    }
```

In this example we write the 10 elements contained in the `data`-array.

Finally, we re-lock the FLASH and are back in safe situation:

```
    FLASH_Lock();
```

And that is basically how you use the FLASH memory.

Here is an example that read the first 10 elements in the last page, writes them to PuTTY, increments them all by one, and rewrites them to the memory. This means that every time you run the program, it will print out larger and larger values – even if you unplug it and plug it back in!

```
#include "stm32f30x_conf.h"

int main(void)
{
    uint32_t address = 0x0800F800; // Starting address of the last page
    uint16_t data[10] = {0};
    uint16_t tempVal;

    init_usb_uart(9600);

    /***********************/
    /*** Read from FLASH ***/
    /***********************/

    for ( int i = 0 ; i < 10 ; i++ ){

        tempVal = *(uint16_t *)(address + i * 2);  // Read Command

        printf("%d ", tempVal);
        data[i] = tempVal + 1;
    }
    printf("\n");


    /***********************/
    /*** Write to FLASH ***/
    /***********************/
    FLASH_Unlock(); // Unlock FLASH for modification
    FLASH_ClearFlag( FLASH_FLAG_EOP | FLASH_FLAG_PGERR | FLASH_FLAG_WRPERR );
    FLASH_ErasePage( address ); // Erase entire page before writing

    for ( int i = 0 ; i < 10 ; i++ ){
        FLASH_ProgramHalfWord(address + i * 2, data[i]);
    }

    FLASH_Lock();

    while(1){ }
}
```