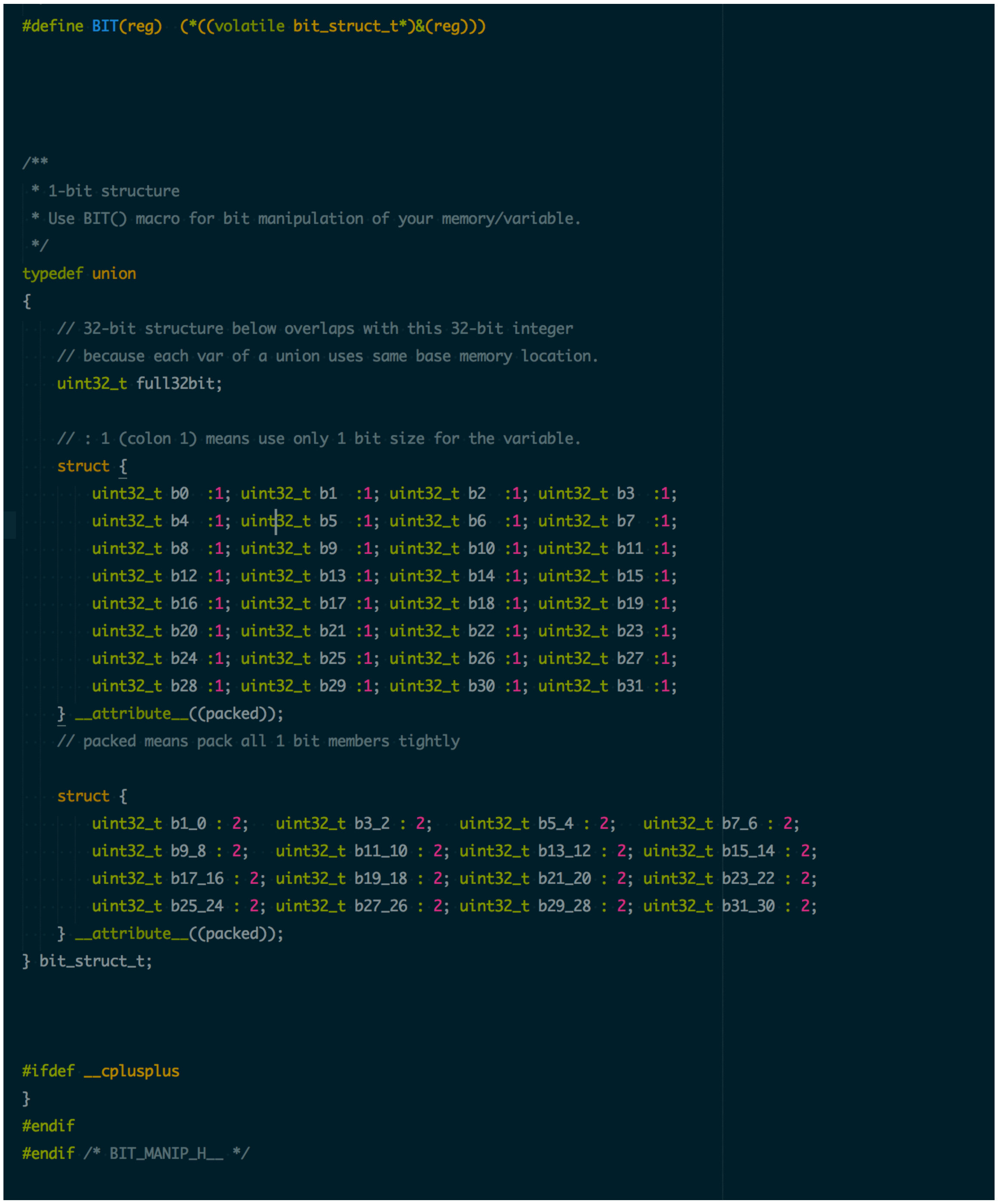
Anahit Sarao
008435583
CMPE 146

Analysis of DiskIO with FATFS Library

1. The lower level architecture for file input and output uses the diskio.c file. The file is dependent on SPI driver, SD card driver, SPI driver semaphore files. The file consists of five main functions. The functions are used to control reading and writing to the disk. The disk can be initialized in two different modes, flash or sd. Flash allows the flash memory to be accessed and SD allows the usage of the SD card port. Within the functions there is a feature known as the SPI lock which allows a mutex to lock all resources for that pertaining I/O. The only way to release the resources is to return a mutex release. This eliminates the possible problem of bus contention on the SPI bus, in other word you cannot I/O to flash and SD at the same time.
2. The numbers for the drives are designated to each SPI bus. A 0 will allow access to the flash memory where 1 will allow access to the SD card port. These parameters can be passed into the functions or used within the terminal task.
3. To be able to read and write using the FATFS library you must use to f_open function which will reference the proper file object needed. The files containing the functionality are ff.c and ff.h.

Extra Credit

BIT Macro is a fast way to allow bit manipulation. This allows access to different pin functionality in a more efficient way. It can be seen as a different way to achieve bit masking. For the first example of BIT(var).b4=1 allows the fourth bit to be changed based on the base address of var. The two structures allow the manipulation of one bit or two bits. This is useful dependent on the bit size of the register. For the second example BIT(var).b15_14 = 2 is used to set two bits simultaneously.



```c
#define BIT(reg)  (*((volatile bit_struct_t*)&(reg)))



/**
 * 1-bit structure
 * Use BIT() macro for bit manipulation of your memory/variable.
 */
typedef union
{
    // 32-bit structure below overlaps with this 32-bit integer
    // because each var of a union uses same base memory location.
    uint32_t full32bit;

    // : 1 (colon 1) means use only 1 bit size for the variable.
    struct {
        uint32_t b0  :1; uint32_t b1  :1; uint32_t b2  :1; uint32_t b3  :1;
        uint32_t b4  :1; uint32_t b5  :1; uint32_t b6  :1; uint32_t b7  :1;
        uint32_t b8  :1; uint32_t b9  :1; uint32_t b10 :1; uint32_t b11 :1;
        uint32_t b12 :1; uint32_t b13 :1; uint32_t b14 :1; uint32_t b15 :1;
        uint32_t b16 :1; uint32_t b17 :1; uint32_t b18 :1; uint32_t b19 :1;
        uint32_t b20 :1; uint32_t b21 :1; uint32_t b22 :1; uint32_t b23 :1;
        uint32_t b24 :1; uint32_t b25 :1; uint32_t b26 :1; uint32_t b27 :1;
        uint32_t b28 :1; uint32_t b29 :1; uint32_t b30 :1; uint32_t b31 :1;
    } __attribute__((packed));
    // packed means pack all 1 bit members tightly

    struct {
        uint32_t b1_0 : 2;   uint32_t b3_2 : 2;   uint32_t b5_4 : 2;   uint32_t b7_6 : 2;
        uint32_t b9_8 : 2;   uint32_t b11_10 : 2; uint32_t b13_12 : 2; uint32_t b15_14 : 2;
        uint32_t b17_16 : 2; uint32_t b19_18 : 2; uint32_t b21_20 : 2; uint32_t b23_22 : 2;
        uint32_t b25_24 : 2; uint32_t b27_26 : 2; uint32_t b29_28 : 2; uint32_t b31_30 : 2;
    } __attribute__((packed));
} bit_struct_t;



#ifdef __cplusplus
}
#endif
#endif /* BIT_MANIP_H__ */
```