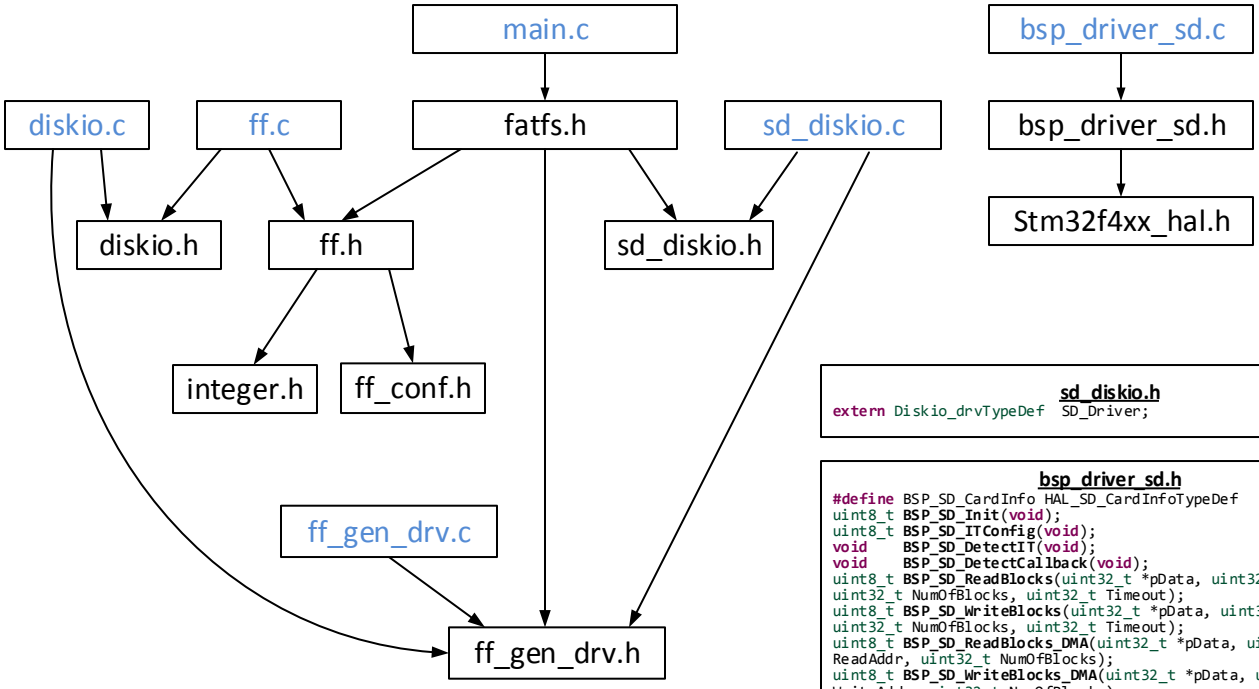

sd_diskio.h

```
extern Diskio_drvTypeDef  SD_Driver;
```

bsp_driver_sd.h

```
#define BSP_SD_CardInfo HAL_SD_CardInfoTypeDef
uint8_t BSP_SD_Init(void);
uint8_t BSP_SD_ITConfig(void);
void    BSP_SD_DetectIT(void);
void    BSP_SD_DetectCallback(void);
uint8_t BSP_SD_ReadBlocks(uint32_t *pData, uint32_t ReadAddr,
uint32_t NumOfBlocks, uint32_t Timeout);
uint8_t BSP_SD_WriteBlocks(uint32_t *pData, uint32_t WriteAddr,
uint32_t NumOfBlocks, uint32_t Timeout);
uint8_t BSP_SD_ReadBlocks_DMA(uint32_t *pData, uint32_t
ReadAddr, uint32_t NumOfBlocks);
uint8_t BSP_SD_WriteBlocks_DMA(uint32_t *pData, uint32_t
WriteAddr, uint32_t NumOfBlocks);
uint8_t BSP_SD_Erase(uint32_t StartAddr, uint32_t EndAddr);
void BSP_SD_IRQHandler(void);
void BSP_SD_DMA_Tx_IRQHandler(void);
void BSP_SD_DMA_Rx_IRQHandler(void);
uint8_t BSP_SD_GetCardState(void);
void    BSP_SD_GetCardInfo(HAL_SD_CardInfoTypeDef *CardInfo);
uint8_t BSP_SD_IsDetected(void);
```

Init

```
main.c: MX_FATFS_Init() →
  fatfs.c : retSD = FATFS_LinkDriver(&SD_Driver, SDPath); →
    ff_gen_drv.c : return FATFS_LinkDriverEx(drv, path, 0);
const Diskio_drvTypeDef  SD_Driver =
{
  SD_initialize, SD_status, SD_read, SD_write, SD_ioctl,
};
Path = 0;
Disk_drvTypeDef disk -hez hozzáadja
```

fatfs.h

```
extern uint8_t retSD;  /* Return value for SD */
extern char SDPath[4]; /* SD logical drive path */
extern FATFS SDFatFS;  /* File system object for SD logical drive */
extern FIL SDFile;     /* File object for SD */
void MX_FATFS_Init(void);
```

diskio.h

```
typedef BYTE DSTATUS;
typedef enum DRESULT;
DSTATUS disk_initialize (BYTE pdrv);
DSTATUS disk_status (BYTE pdrv);
DRESULT disk_read (BYTE pdrv, BYTE* buff, DWORD sector, UINT count);
DRESULT disk_write (BYTE pdrv, const BYTE* buff, DWORD sector, UINT count);
DRESULT disk_ioctl (BYTE pdrv, BYTE cmd, void* buff);
DWORD get_fattime (void);
```

ff_gen_drv.h

```
typedef struct
{
  DSTATUS (*disk_initialize) (BYTE); // Initialize Disk Drive
  DSTATUS (*disk_status)     (BYTE); // Get Disk Status
  DRESULT (*disk_read)       (BYTE, BYTE*, DWORD, UINT);// Read Sector(s)
#if _USE_WRITE == 1
  DRESULT (*disk_write)      (BYTE, const BYTE*, DWORD, UINT); // Write
Sector(s)
#endif /* _USE_WRITE == 1 */
#if _USE_IOCTL == 1
  DRESULT (*disk_ioctl)      (BYTE, BYTE, void*); // I/O control operation
#endif /* _USE_IOCTL == 1 */
}Diskio_drvTypeDef;

// Global Disk IO Drivers structure definition
typedef struct
{
  uint8_t                 is_initialized[_VOLUMES];
  Diskio_drvTypeDef       *drv[_VOLUMES];
  uint8_t                 lun[_VOLUMES];
  __IO uint8_t            nbr;
}Disk_drvTypeDef;

uint8_t FATFS_LinkDriverEx(Diskio_drvTypeDef *drv, char *path, uint8_t lun);
uint8_t FATFS_LinkDriver(Diskio_drvTypeDef *drv, char *path);
uint8_t FATFS_UnLinkDriver(char *path);
uint8_t FATFS_LinkDriverEx(Diskio_drvTypeDef *drv, char *path, BYTE lun);
uint8_t FATFS_UnLinkDriverEx(char *path, BYTE lun);
uint8_t FATFS_GetAttachedDriversNbr(void);
```

ff.h

```
typedef struct FATFS;
typedef struct FIL;
typedef struct DIR;
typedef struct FILINFO;
typedef enum  FRESULT;
FRESULT f_open (FIL* fp, const TCHAR* path, BYTE mode);                /* Open or create a file */
FRESULT f_close (FIL* fp);                                             /* Close an open file object */
FRESULT f_read (FIL* fp, void* buff, UINT btr, UINT* br);              /* Read data from a file */
FRESULT f_write (FIL* fp, const void* buff, UINT btw, UINT* bw);       /* Write data to a file */
FRESULT f_forward (FIL* fp, UINT(*func)(const BYTE*,UINT), UINT btf, UINT* bf);  /* Forward data to the stream */
FRESULT f_lseek (FIL* fp, DWORD ofs);                                  /* Move file pointer of a file object */
FRESULT f_truncate (FIL* fp);                                          /* Truncate file */
FRESULT f_sync (FIL* fp);                                              /* Flush cached data of a writing file */
FRESULT f_opendir (DIR* dp, const TCHAR* path);                        /* Open a directory */
FRESULT f_closedir (DIR* dp);                                          /* Close an open directory */
FRESULT f_readdir (DIR* dp, FILINFO* fno);                             /* Read a directory item */
FRESULT f_findfirst (DIR* dp, FILINFO* fno, const TCHAR* path, const TCHAR* pattern);  /* Find first file */
FRESULT f_findnext (DIR* dp, FILINFO* fno);                            /* Find next file */
FRESULT f_mkdir (const TCHAR* path);                                   /* Create a sub directory */
FRESULT f_unlink (const TCHAR* path);                                  /* Delete an existing file or directory */
FRESULT f_rename (const TCHAR* path_old, const TCHAR* path_new);       /* Rename/Move a file or directory */
FRESULT f_stat (const TCHAR* path, FILINFO* fno);                      /* Get file status */
FRESULT f_chmod (const TCHAR* path, BYTE attr, BYTE mask);             /* Change attribute of the file/dir */
FRESULT f_utime (const TCHAR* path, const FILINFO* fno);               /* Change times-tamp of the file/dir */
FRESULT f_chdir (const TCHAR* path);                                   /* Change current directory */
FRESULT f_chdrive (const TCHAR* path);                                 /* Change current drive */
FRESULT f_getcwd (TCHAR* buff, UINT len);                              /* Get current directory */
FRESULT f_getfree (const TCHAR* path, DWORD* nclst, FATFS** fatfs);    /* Get number of free clusters on the drive */
FRESULT f_getlabel (const TCHAR* path, TCHAR* label, DWORD* vsn);      /* Get volume label */
FRESULT f_setlabel (const TCHAR* label);                               /* Set volume label */
FRESULT f_mount (FATFS* fs, const TCHAR* path, BYTE opt);              /* Mount/Unmount a logical drive */
FRESULT f_mkfs (const TCHAR* path, BYTE sfd, UINT au);                 /* Create a file system on the volume */
FRESULT f_fdisk (BYTE pdrv, const DWORD szt[], void* work);            /* Divide a physical drive into some partitions */
int f_putc (TCHAR c, FIL* fp);                                         /* Put a character to the file */
int f_puts (const TCHAR* str, FIL* cp);                                /* Put a string to the file */
int f_printf (FIL* fp, const TCHAR* str, ...);                         /* Put a formatted string to the file */
TCHAR* f_gets (TCHAR* buff, int len, FIL* fp);                         /* Get a string from the file */

#define f_eof(fp) ((int)((fp)->fptr == (fp)->fsize))
#define f_error(fp) ((fp)->err)
#define f_tell(fp) ((fp)->fptr)
#define f_size(fp) ((fp)->fsize)
#define f_rewind(fp) f_lseek((fp), 0)
#define f_rewinddir(dp) f_readdir((dp), 0)
```