Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ЭВМ

ОТЧЕТ

по лабораторной работе No 6

“Память, ПДП, SD-карта”

Вариант 13

Выполнил:
ст. гр. 050503 Кириллов В.И.

Проверил:
ассистент Шеменков В.В.

Минск 2023

Задание

1. В соответствии с вариантом написать программу, которая заносит получаемые с заданной частотой от периферийного устройства данные в память, используя прямой доступ к памяти. Данные отображаются в виде графика на экране в зависимости от заданного режима отображения и сохраняются в файл на SD-карту. При нажатии указанной кнопки сохраненные данные считываются из файла и выводятся на экран в виде графика, при ее повторном нажатии происходит возврат в режим измерения. Максимально использовать высокоуровневые библиотеки. Усложненный вариант задания предусматривает снятие сигналов SPI интерфейса осциллографом, написание отчета с пояснением полученных результатов.

13	Потенцио-метр	Любая PAD	400	70	Динамически с каждым прочитанным значени-ем. Сохранение в файл — по кнопке S2

Высокоуровневые библиотеки

Примеры программ для микроконтроллера MSP430F5529 написаны с использованием API для управления компонентами микроконтроллера и экспериментальной платы. Заголовочные файлы API находятся по *TI / msp430 / src /*.*. Структура библиотеки следующая:

- */CTS* – библиотека для поддержки функций сенсорной клавиатуры;
- */structure.h* – описание используемых библиотекой структур данных;
- */CTS_HAL.h* – функции ядра библиотеки, поддержка методов измерения RO, fRO,

RC, установка прерываний таймеров;

- */CTS_Layer.h* – слой API, содержит функции отслеживания базового уровня

сенсора, определения нажатия каждого сенсора и т.д.;

- */F5xx_F6xx_Core_Lib* – библиотека ядра;
- */HAL_UCS.h* – функции работы с унифицированной системой тактирования — выбор источников сигнала MCLK, SMCLK, ACLK, установка делителя, настройки генераторов XT1, XT2, режим блока FLL;
- */HAL_PMM.h* – функции работы с менеджером питания;
- */HAL_FLASH.h* – библиотека для работы с FLASH-памятью;
- */FatFs* – стек файловой системы FAT для поддержки SD-карты;
- */MSP-EXP430F5529_HAL* — библиотека для поддержки основных устройств

экспериментальной платы;

- */HAL_Wheel.h* – работа с потенциометром;
- */HAL_SDCard.h* – работа с SD-картой памяти;

− */HAL_Dogs102x6.h* – работа с ЖКИ экраном, включая простейшие графические

функции;

− */HAL_Cma3000.h* – работа с акселерометром;

− */HAL_Buttons.h* – работа с кнопками;

− */HAL_Board.h* – работа со светодиодами;

− */HAL_AppUart.h* – работа с USCI в режиме UART;

− */USB* – стек USB для экспериментальной платы;

− */UserExperienceDemo* — пример приложения с использованием высокоуровневых

библиотек. Именно это приложение использовалось в лабораторной работе No1 для знакомства с комплектом.

SD-карта памяти

MMC/SD или SD-карта памяти представляет собой функционально и конструктивно законченный модуль, который содержит в своем составе собственно память и управляющий микроконтроллер. Обмен данными возможен по двум протоколам: MMC и SPI. Протокол MMC обеспечивает большую скорость и возможность параллельного включения нескольких карт и является основным. Тем не менее для многих платформ удобнее использовать протокол SPI, который поддерживается микроконтроллером на аппаратном уровне.

Схема подключения SD-карты памяти приведена на рисунке 6.1. Как видно из рисунка, карта подключается по интерфейсу SPI, причем на одном и том же канале, что и ЖКИ. Соответственно, при программировании необходимо учитывать этот момент: на шине SPI не должны одновременно присутствовать два устройства, которым разрешена работа.

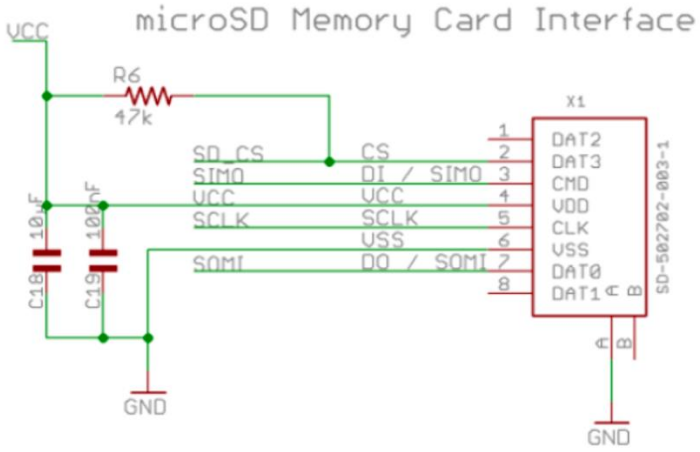


Схема подключения разъема SD-карты памяти

Заголовочный файл *HAL_SDCard.h* определяет следующие функции:

void SDCard_init(void) — подключение линий микроконтроллера и инициализация интерфейса SPI в режиме 3-проводной, Master, MSB, 8-бит, активный уровень CLK – низкий, источник тактирования SMCLK, частота тактирования 397 КГц (при инициализации должна быть менее 400 КГц).

void SDCard_fastMode(void) — устанавливает тактовую частоту 12,5 МГц для быстрого обмена.

*void SDCard_readFrame(uint8_t *pBuffer, uint16_t size)* — чтение данных из памяти, 1 параметр — указатель на буфер приема, 2 параметр — количество байт.

*void SDCard_sendFrame(uint8_t *pBuffer, uint16_t size)* — запись данных в память, 1 параметр — указатель на буфер с данными, 2 параметр — количество байт.

void SDCard_setCSHigh(void) — установка сигнала выбора устройства в 1. *void SDCard_setCSLow(void)* — сброс сигнала выбора устройства в 0.

Выводы SD-разъема	**Обозначение линии на схеме**	**Назначение**	**Вывод MSP430F5529**	**Требуемый режим**
DAT3	SD_CS	Разрешение устройства	P3.7/TB0OUTH	P3.7
CMD	SIMO	SIMO данные (запись в память)	P4.1/ PM_UCB1SIMO/ PM_UCB1SDA	PM_UCB1SIMO
CLK	SCLK	Синхросигнал	P4.3/ PM_UCB1CLK/ PM_UCA1STE	PM_UCB1CLK
DAT0	SOMI	SOMI данные (чтение из памяти)	P4.2/ PM_UCB1SOMI/ PM_UCB1SCL	PM_UCB1SOMI

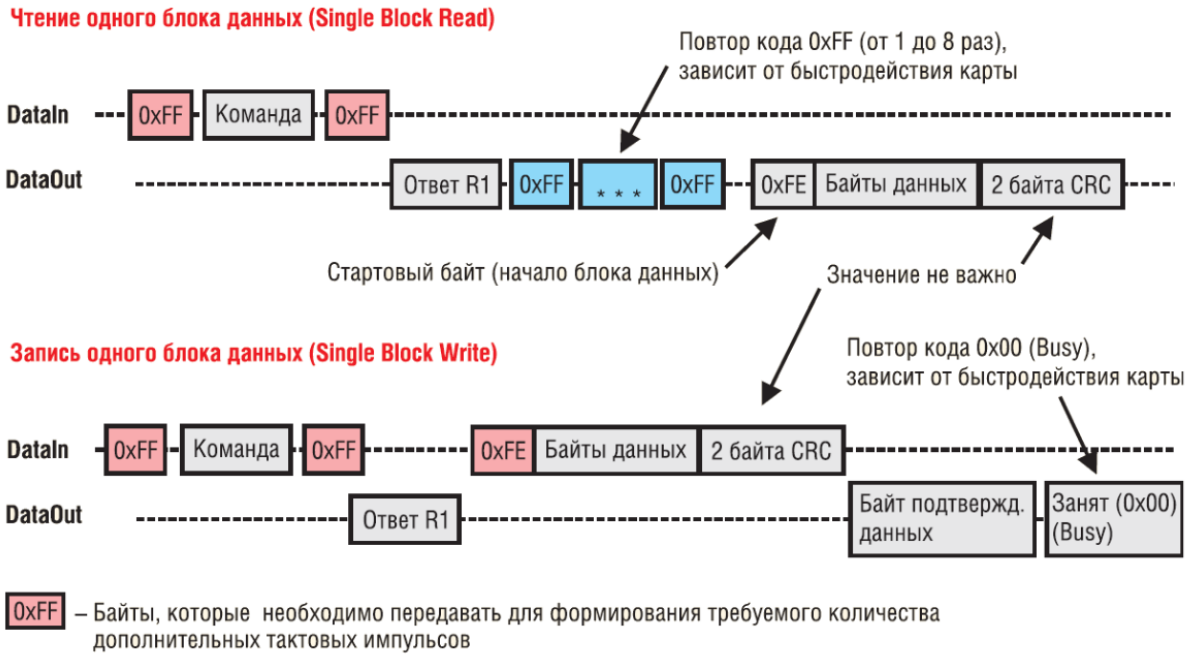
Однако эти функции всего лишь подключают SPI интерфейс, и этого недостаточно для работы с MMC/SD картой. Для корректной связи с ней необходимо поддерживать протокол обмена, установленный для MMC/SD. Кратко рассмотрим его особенности. Более подробно работа с SD-картой памяти рассмотрена в [28, 29].

MMC/SD карта принимает от микроконтроллера ряд команд, на которые она выдаёт либо ответы определённого типа, либо блоки данных. .

Блоки данных могут быть различной длины и состоят из стартового байта (0xFE), собственно данных (их длина 1...N байт, где N определяется размером физического сектора, в большинстве случаев – 512 байт) и двух байт контрольной суммы. Контрольная сумма является опциональной в SPI

интерфейсе и, как правило, не используется для упрощения процедуры обмена. Значения двух байт контрольной суммы можно игнорировать, но сами эти байты должны обязательно передаваться/приниматься для соблюдения протокола обмена.

Перед передачей команды или после этого микроконтроллер должен выдавать не менее 8 тактовых импульсов по линии CLK, т.е. просто передавать «лишний» байт 0xFF. При чтении блока данных после передачи соответствующей команды микроконтроллер принимает байты 0xFF до тех пор, пока не встретится байт 0xFE (стартовый байт блока данных). Любой иной байт (отличный от 0xFF), полученный в этот момент, будет означать ошибку.



Все команды, воспринимаемые MMC/SD картой, имеют длину 6 байт. Индекс команды (порядковый номер) находится в битах 0..5 первого байта команды, биты 7 и 6 всегда содержат 0 и 1 соответственно. Следующие 4 байта содержат аргумент команды, например, 32-битный адрес первого байта данных. Последний байт команды сдержит в битах 1..7 контрольную сумму, бит 0 всегда равен 1.

MMC/SD карта после приёма команды выдаёт ответ, содержащий один, два или пять байт. Первым передаётся старший байт. Ответ формата R1 содержит один байт. Структура ответа R1:

- – бит 7 – всегда 0;
- – бит 6 — ошибка параметра команды;

- − бит 5 — ошибка адреса;
- − бит 4 — ошибка стирания;
- − бит 3 — ошибка контрольной суммы CRC;
- − бит 2 — неверная команда;
- − бит 1 — прервана команда стирания;
- − бит 0 — режим простоя, выполняется инициализация.

Ответ R2 состоит из двух байт, причём первый байт ответа идентичен структуре ответа R1. Структура второго байта в ответе R2:

  - − бит 7 — выход за пределы / ошибка перезаписи;
  - − бит 6 — ошибка параметра при стирании;
  - − бит 5 — попытка записи в защищенную от записи область;
  - − бит 4 — ошибка коррекции;
  - − бит 3 — внутренняя ошибка;
  - − бит 2 — общая / неизвестная ошибка;

- − бит 1 — попытка стирания защищенного от записи сектора / ошибка блокирования/разблокирования;
- − бит 0 — карта заблокирована.
  Ответ R3 состоит из 5 байт. Первый байт идентичен ответу R1, остальные 4 байта

представляют собой содержимое регистра OCR.
При записи данных в MMC/SD карту после получения блока данных карта отвечает

байтом подтверждения данных. Бит 4 подтверждения всегда равен 0, бит 0 — 1. В битах 1..3 указывается статус операции, успешной записи соответствует значение 010.

Команды записи и чтения сопровождаются пересылкой блоков данных. Каждый блок данных начинается со стартового байта. Следующий за ним байт – это фактические данные. Завершаются фактические данные двумя байтами контрольной суммы (16 бит CRC). Так как в режиме SPI контрольную сумму можно не вычислять, значения этих двух байтов не имеют значения, но сами байты контрольной суммы обязательны. Если операция чтения данных завершилась неудачно и карта не может предоставить запрашиваемые данные, то она будет посылать байт ошибки данных.

После подачи напряжения питания MMC/SD карта находится в режиме MMC, а не в режиме SPI. Для перевода карты в режим SPI и инициализации карты необходимо выполнить определённую последовательность действий:

- − не выбирая устройство (сигнал CS = 1) послать 80 импульсов по линии CLK (передать 10 байт 0xFF);
- − выбрать MMC/SDC карту (CS = 0);

- − послать команду CMD0 (сброс): 0x40, 0, 0, 0, 0, 0x95 (в этой команде контрольная

  сумма должна иметь реальное значение (0x95), т.к. данная команда посылается в тот

  момент, когда MMC/SD карта находится в режиме MMC, а не SPI);

- − дождаться правильного ответа 0x01;
- − в цикле посылать команду CMD1 (инициализация) и ждать, когда будет получен

ответ 0x00 (этот ответ означает, что карта инициализирована в режиме SPI и готова принимать команды). Для SD-карт в случае отклонения команды CMD1 рекомендуется использовать команду ACMD41.
В таблице приведён ряд команд для MMC/SD карты в режиме SPI:

Команда	**Код**	**Аргу-мент**	**Ответ**	**Данные**	**Аббревиатура**	**Описание**
CMD0	40h	Нет(0)	R1	Нет	GO_IDLE_STATE	Программный сброс
CMD1	41h	Нет(0)	R1	Нет	SEND_OP_COND	Запуск процесса инициализации
ACMD41	69h	*	R1	Нет	APP_SEND_OP_COND	Только для карт SDC. Запуск процесса инициализации
CMD8	48h	**	R7	Нет	SEND_IF_COND	Только для карт SDC v2. Проверка диапазона напряжения питания
CMD9	49h	Нет(0)	R1	Да	SEND_CSD	Чтение регистра CSD
CMD10	4Ah	Нет(0)	R1	Да	SEND_CID	Чтение регистра CID
CMD12	4Ch	Нет(0)	R1b	Нет	STOP_TRANSMISSION	Остановка чтения данных

CMD16	50h	Длина блока [31:0]	R1	Нет	SET_BLOCKLEN	Установка размера блока чтения записи
CMD17	51h	Адрес [31:0]	R1	Да	READ_SINGLE_BLOCK	Чтение блока
CMD18	52h	Адрес [31:0]	R1	Да	READ_MULTIPLE_BLOCK	Чтение нескольких блоков
CMD23	57h	Число блоков [15:0]	R1	Нет	SET_BLOCK_COUNT	Только для MMC. Указание количества блоков для передачи со следующей командой многоблочного чтения/записи
ACMD23	57h	Число блоков [22:0]	R1	Нет	SET_WR_BLOCK_ERASE_ COUNT	Только для SD. Указание количества блоков для предварительного стирания для последующей команды многоблочной записи
CMD24	58h	Адрес [31:0]	R1	Да	WRITE_BLOCK	Запись блока
CMD25	59h	Адрес [31:0]	R1	Да	WRITE_MULTIPLE_ BLOCK	Запись нескольких блоков
CMD55	77h	Нет(0)	R1	Нет	APP_CMD	Начало команды ACMD
CMD58	7Ah	Нет(0)	R3	Нет	READ_OCR	Чтение OCR
ACMD означает последовательность двух команд CMD55 + CMD; * - Бит 30 - HCS, остальные в 0 ** - Биты 31..12 = 0, биты 11..8 — напряжение питания, биты 7..0 — 0xAA						

После простоя более 5 мс карта памяти переходит в энергосберегающий режим, и способна принимать только команды CMD0, CMD1 и CMD58. Поэтому процесс инициализации (CMD1) необходимо практически каждый раз повторять при чтении/записи блока данных или делать проверку состояния карты.

Основные регистры контроллера карты, которые доступны по SPI протоколу:

- - CID (Card identification data): содержит данные, по которым можно идентифицировать карту памяти (серийный номер, ID производителя, дату

  изготовления и т.д.);

- - CSD (Card-specific data): содержит всевозможную информацию о карте памяти (от

  размера сектора карты памяти до потребления в режиме чтения/записи);

- - OCR (Operation Conditions Register): содержит напряжения питания карты памяти,

  тип питания карты памяти, статус процесса инициализации карты.

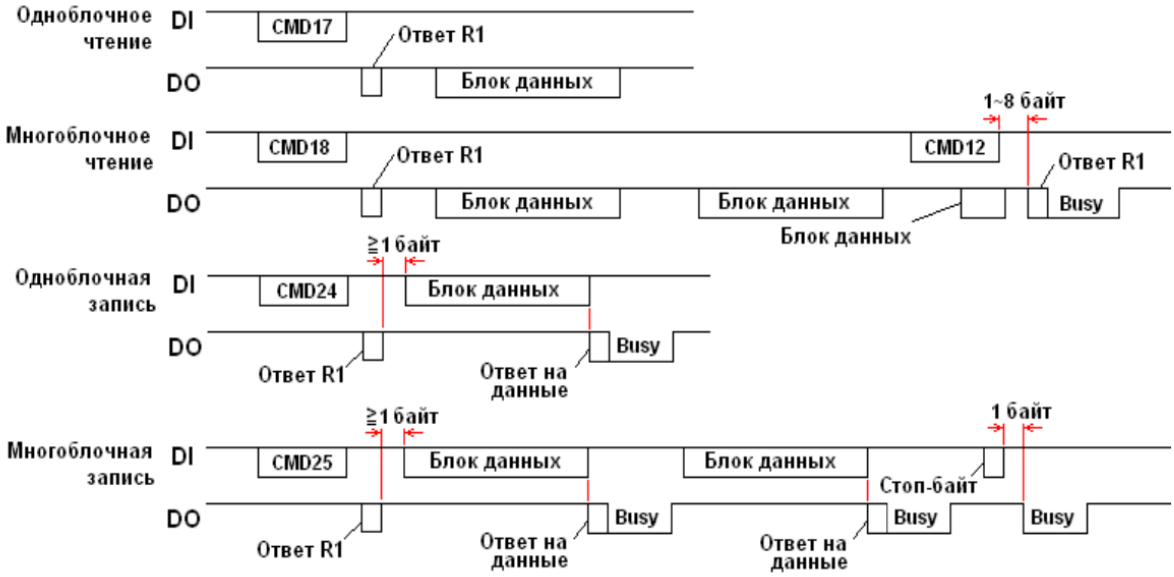


Схема одноблочного и многоблочного чтения/записи

Схема передачи в режимах одноблочной и многоблочной чтения/записи приведена на рисунке 6.3. Одноблочное чтение инициируется командой CMD17. Ее аргумент задает адрес начала чтения. Чтение осуществляется побайтно. В ответ на команду CMD17 карта выдает ведущему контроллеру пакет данных. После обнаружения правильного маркера данных ведущий контроллер принимает следующий за ним блок данных и два байта CRC, которые необходимо принять, даже если CRC не используется. По умолчанию размер блока 512 байтов, но его можно изменить командой CMD16. Если во время операции чтения произошла какая-нибудь ошибка, вместо пакета данных будет возвращен маркер ошибки.

С помощью команды многоблочного чтения CMD18 можно прочитать последовательность нескольких блоков, начиная с заданного адреса. Если перед этой командой с помощью команды CMD23 (только для MMC) не было задано число передаваемых блоков, будет инициировано неограниченное многоблочное чтение, то есть операция чтения будет продолжаться, пока ведущий контроллер не прервёт ее командой CMD12. Байт, получаемый сразу же после передачи CMD12, является наполняющим, его не нужно учитывать. После этого байта следует ответ на команду.

После того, как карта приняла команду записи CMD24, ведущий контроллер после байтового промежутка (один или более байтов) передает пакет данных. Формат пакета такой же, как и у команды блочного чтения.

После передачи пакета карта сразу же выдает ответ на данные, за которым следует флаг занятости. Большинство карт не могут менять размер записываемого блока, он является фиксированным и составляет 512 байтов. По правилам режима SPI сигнал CS должен находится в активном уровне в течение всей транзакции, однако есть исключение из этого правила. Когда карта занята, ведущий контроллер может снять сигнал CS, чтобы освободить шину SPI для какого-нибудь другого SPI-устройства. Если же снова выбрать карту в то время, когда она занята выполнением внутреннего процесса, карта снова установит сигнал DO в низкий уровень.

Поэтому, чтобы сократить время ожидания, лучше выполнять проверку на занятость непосредственно перед выдачей команды и пакета данных, а не ожидать освобождения карты после посылки команды. Кроме того, внутренний процесс инициируется спустя байт после ответа данных, т.е. необходимо выдать 8 тактовых импульсов, чтобы инициировать внутреннюю операцию записи. Состояние сигнала CS во время этих восьми тактовых импульсов не учитывается, поэтому можно совместить эту инициацию с процессом освобождения шины.

С помощью команды многоблочной записи CMD25 можно записать последовательность из нескольких блоков, начиная с заданного адреса. Если перед этой командой число передаваемых блоков не было задано с помощью команды CMD23 (только для MMC) или ACMD23 (для SD), транзакция будет инициирована как неограниченная многоблочная запись, то есть операция записи будет продолжаться, пока ведущий контроллер не прервет ее передачей маркера остановки передачи (стоп-байт, FDh). Флаг занятости появится байт спустя после стоп-байта. Что же касается SD, то транзакция многоблочной записи должна прерываться стоп-байтом независимо от того, является ли она предопределенной или неограниченной.

Функции пользовательского API, необходимые для работы с MMC/SD картой находятся в заголовочном файле */FatFs/mmc.h.* На пользовательском уровне карта памяти представляется диском. Функции этого слоя:

DSTATUS disk_status (BYTE drv) — получение состояния диска. Передается номер диска (0).

DSTATUS disk_initialize (BYTE drv) — инициализация диска. Параметры и результат аналогичны предыдущей функции.

*DRESULT disk_read (BYTE drv, BYTE *buff, DWORD sector, BYTE count)* – чтение данных с диска. Параметры: номер диска, указатель на буфер для размещения данных, начальный номер сектора для чтения (LBA), количество секторов.

*DRESULT disk_write (BYTE drv, const BYTE *buff, DWORD sector, BYTE count)* — запись данных на диск. Параметры: номер диска, указатель на буфер с данными для записи, начальный номер сектора для записи (LBA), количество секторов.

*DRESULT disk_ioctl (BYTE drv, BYTE ctrl, void *buff)* — команда управления. Параметры: номер диска, код команды, указатель на буфер для приема/передачи данных команды управления.

uint8_t detectCard(void) — обнаружение карты и попытка подключения если карта не обнаружена. Возвращает 1, если карта готова к работе, 0 — карта не обнаружена.

Этого уже достаточно, чтобы обмениваться неформатированными данными. Физически память MMC/SD карты разбита на сектора по 512 байт, карта имеет, как правило, файловую систему FAT16. Поэтому для полноценной поддержки обмена файлами, которые потом будут видимы при использовании SD-карты на других устройствах, необходимо еще и поддержать файловую систему FAT. Структуру MBR, FAT и формат каталога в этой файловой системе рассматривать не будем. Заголовочный файл */FatFs/ff.h* содержит необходимые функции:

FRESULT f_mount (BYTE, FATFS)* — подключение/отключение логического диска. *FRESULT f_open (FIL*, const TCHAR*, BYTE)* — открытие или создание файла.

FRESULT f_read (FIL, void*, UINT, UINT*)* — чтение данных из файла. *FRESULT f_lseek (FIL*, DWORD)* — перемещение файлового указателя. *FRESULT f_close (FIL*)* — закрытие открытого файла.
FRESULT f_opendir (DIRS, const TCHAR*)* — открытие существующего каталога. *FRESULT f_readdir (DIRS*, FILINFO*)* — чтение эелементов каталога.

FRESULT f_stat (const TCHAR, FILINFO*)* — получение состояния файла. *FRESULT f_write (FIL*, const void*, UINT, UINT*)* — запись данных в файл. *FRESULT f_getfree (const TCHAR*, DWORD*, FATFS**)* — получение количества

свободных кластеров на диске.

FRESULT f_forward (FIL, UINT(*)(const BYTE*,UINT), UINT, UINT*)* — помещение данных в поток.

FRESULT f_mkfs (BYTE, BYTE, UINT) — создание файловой системы на диске. *FRESULT f_chdrive (BYTE)* — смена текущего диска.
FRESULT f_chdir (const TCHAR)* — смена текущего каталога.
FRESULT f_getcwd (TCHAR, UINT)* — получение текущего каталога.

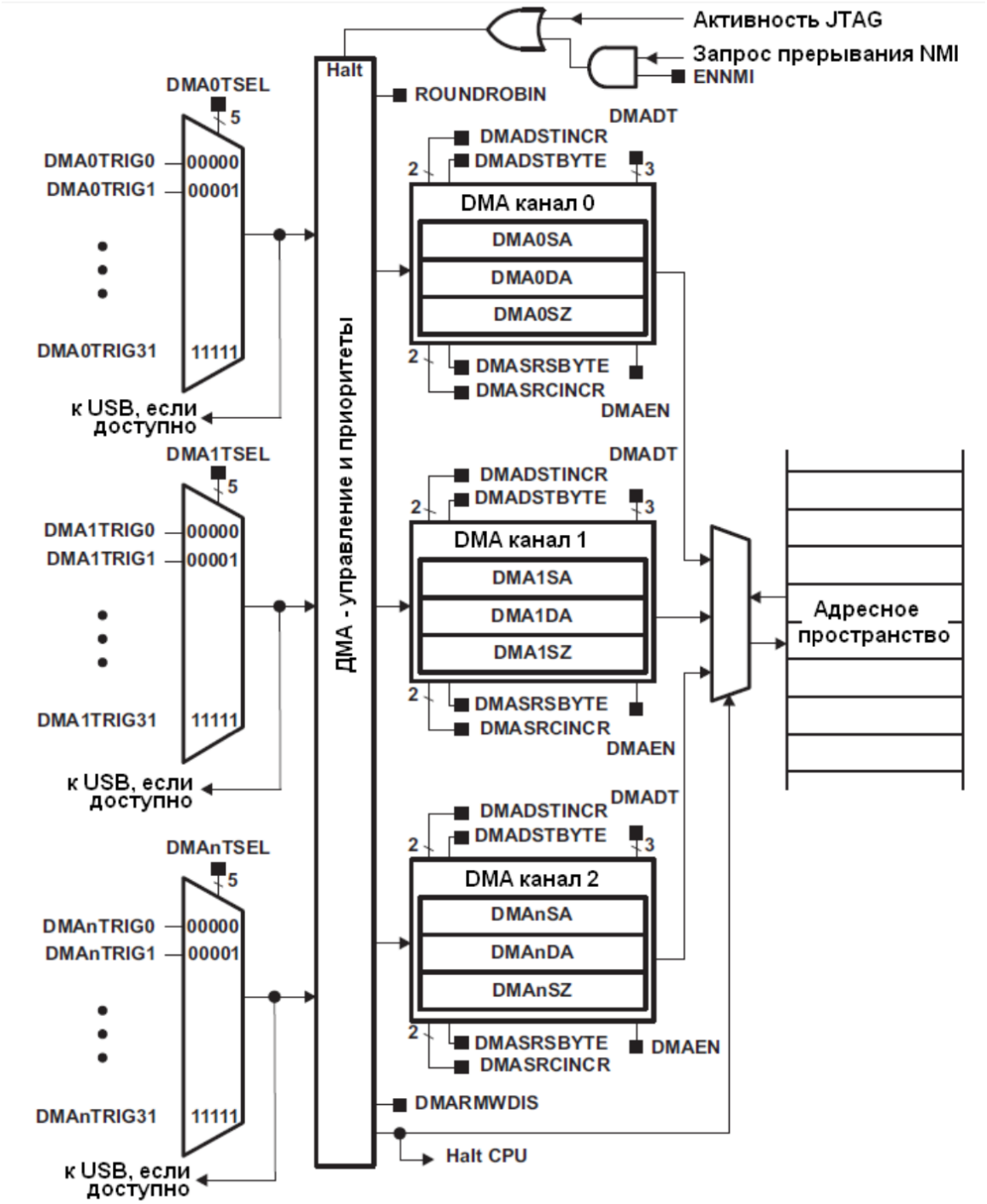
int f_putc (TCHAR, FIL)* — запись символа в файл.
int f_puts (const TCHAR, FIL*)* — запись строки в файл.
int f_printf (FIL, const TCHAR*, ...)* — запись форматированной строки в файл. *TCHAR* f_gets (TCHAR*, int, FIL*)* — стение строки из файла.
Кроме того, приведены следующие макроопределения:
#define *f_eof(fp)* (((fp)->fptr == (fp)->fsize) ? 1 : 0)
#define *f_error(fp)* (((fp)->flag & FA__ERROR) ? 1 : 0)

#define *f_tell(fp)* ((fp)->fptr)
#define *f_size(fp)* ((fp)->fsize)

Прямой доступ к памяти

Прежде, чем рассматривать прямой доступ к памяти, изучим, как организована работа с ОЗУ. ОЗУ (RAM) микроконтроллера MSP430F5529 разделена на 4 сектора по 2 Кб. Сектор 0 содержит адреса 002400h – 002BFFh, сектор 1 — 002C00h – 0033FFh, сектор 2 — 003400h - 003BFFh, сектор 3 — 003C00h – 0043FFh. Кроме того, имеется 2 Кб сектор USB RAM (сектор 7), который может использоваться как обычное ОЗУ, если не используется USB. Каждый из секторов может быть отключен битом RCRSyOFF регистра

RCCTL0. Чтение из отключенного сектора всегда дает 0. Стек располагается в ОЗУ, поэтому нельзя отключать сектор, содержащий стек, если используются прерывания или LPM режим. В LPM режиме процессор микроконтроллера отключен, поэтому память находится в режиме удержания для уменьшения тока утечки. Адрес регистра RCCTL0 – 6900h. Доступ к этому регистру защищен ключом. Перед изменением RCCTL0_L (младшего байта, содержащего флаги отключения секторов, номер бита соответствует номеру сектора), в регистр RCCTL0_H необходимо записать код 5Ah. При чтении старший байт содержит 69h.

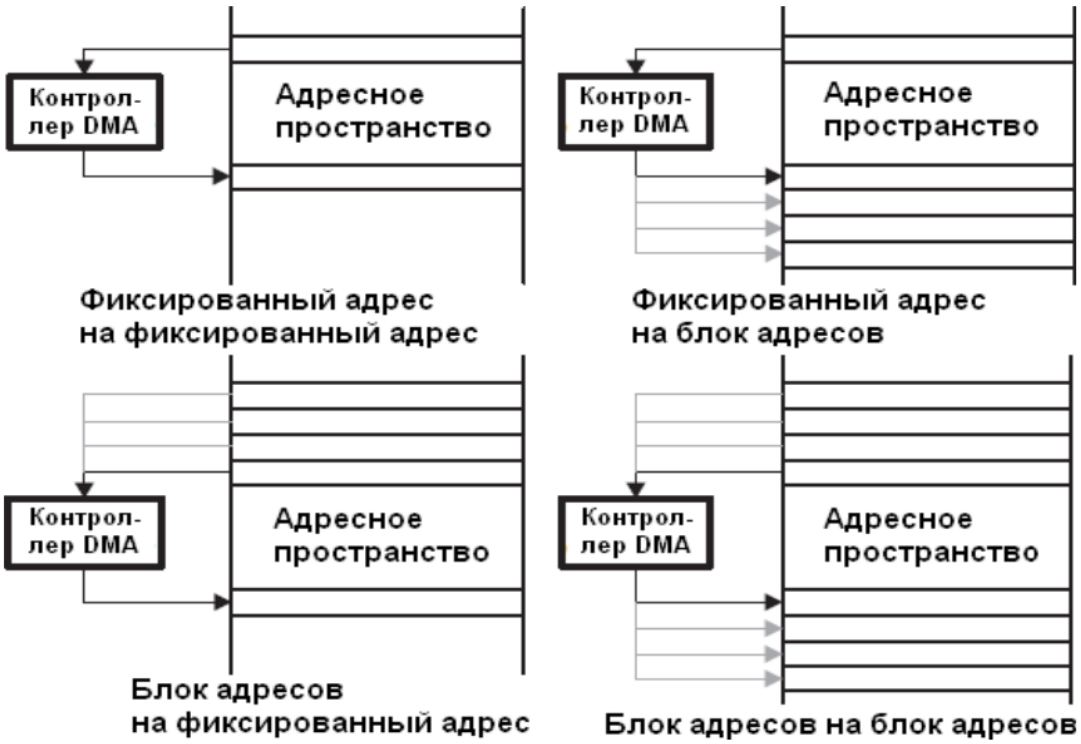


Структура контроллера DMA

Контроллер прямого доступа к памяти (DMA) выполняет пересылку данных между адресами без участия центрального процессора. В микроконтроллере MSP430F5529 контроллер DMA содержит 3 канала. Использование DMA может увеличить

производительность периферии, а также снизить ток потребления, поскольку центральный процессор может оставаться в LPM режиме. Характеристики DMA-контроллера:

- − три независимых канала;
- − программируемые приоритеты каналов;
- − требуется всего 2 MCLK такта на пересылку;
- − возможность пересылки байт, слов или смешанные;
- − размер блока данных до 65 К байт или слов;
- − программируемый выбор триггеров передачи;
- − пересылки по перепаду сигнала триггера или по уровню;
- − 4 режима адресации;
- − 3 режима пересылки: одиночные, блочные и многоблочные. Схема контроллера представлена на рис. 6.4.



Режимы адресации данных в DMA

Доступны следующие режимы адресации: фиксированный адрес на фиксированный адрес, фиксированный адрес на блок адресов, блок адресов на фиксированный адрес, блок адресов на блок адресов (рис. 6.5). Биты DMASRCINCR и DMADSTINCR выбирают, будут ли адреса источника и приемника, соответственно, инкрементироваться, декрементироваться или оставаться без изменений. Пересылки возможны байт в байт, байт в слово (старший байт результата обнуляется), слово в байт (пересылается младший байт источника) и слово в слово.

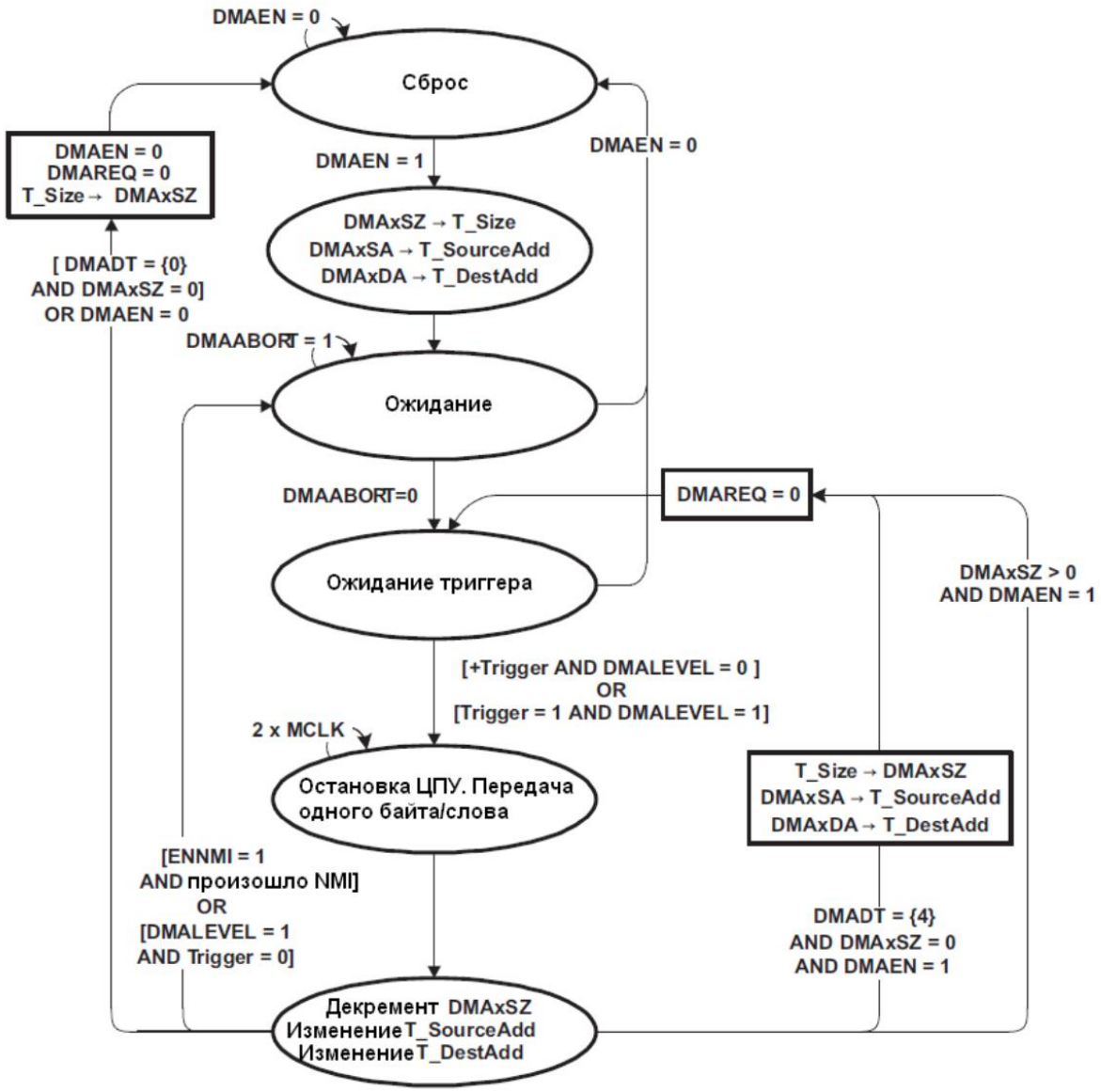


Диаграмма состояний в режиме одиночных пересылок

Биты DMADT задают 6 режимов пересылки, программируемые отдельно для каждого из каналов:

000 — одиночная пересылка;
001 — блочная пересылка;
010, 011 — импульсная блочная пересылка;
100 — повторяющаяся одиночная пересылка;
101 — повторяющаяся блочная пересылка;
110, 111 — повторяющаяся импульсная блочная пересылка.
В режиме одиночных пересылок текущие значения адреса источника, адреса

назначения и количества пересылок копируются во временные регистры, которые изменяются после каждой пересылки. Каждая пересылка требует срабатывания триггера,

DMAEN автоматически сбрасывается, когда сделано DMAxSZ пересылок. Когда DMAxSZ становится равен 0, он повторно копируется, и устанавливается соответствующий флаг DMAIFG. В случае повторяющихся одиночных пересылок, DMAEN остается активным, и при каждом новом срабатывании триггера происходит пересылка. Для примера на рис. 6.6 приведена диаграмма состояний режима одиночных пересылок. В остальных режимах диаграмма незначительно отличается.

В случае блочной пересылки по срабатыванию одного триггера пересылается блок данных, в процессе пересылки игнорируются любые другие сработавшие триггеры. DMAxSZ определяет размер блока. На протяжении всего обмена процессор остановлен. DMAEN автоматически сбрасывается после передачи блока. В случае повторяющихся блочных пересылок DMAEN остается активным, и по окончании пересылки новое срабатывание триггера вызывает новую пересылку блока. В остальном режим подобен одиночным пересылкам. Пересылка занимает 2 x MCLK x DMAxSZ тактов.

В импульсно-блочных пересылках после каждых 4 пересылок байт либо слов, на 2 такта MCLK включается процессор. В результате он занимает 20% времени. В остальном режим подобен блочному. Существенное отличие в повторяющемся импульсно-блочном режиме: так как DMAEN остается активным, то после пересылки блока новый блок начинает пересылку следующего, и для этого не требуется срабатывание нового триггера. Такая пересылка может быть остановлена сбросом DMAEN бита либо прерыванием NMI при установленном ENNMI.

Для режима DMALEVEL = 1 триггер определен по уровню сигнала. Для правильной работы этого режима источником сигнала триггера должен быть выбран внешний источник DMAE0. Пересылка активна все время, пока сигнал триггера остается высоким и DMAEN = 1. В этом режиме сигнал триггера должен оставаться высоким на все время пересылки. Если он станет низким для блочной или импульсно-блочной пересылки, контроллер DMA будет остановлен в текущем состоянии и продолжит работу при возврате сигнала в высокий уровень. В это время простоя могут быть изменены регистры DMA. Режим рекомендуется использовать для режима пересылок, когда DMAEN автоматически сбрасывается (DMADT = 0 .. 3).

Если триггеры разных каналов срабатывают одновременно, на выполнение ставится пересылка того канала, у которого максимальный приоритет. По умолчанию приоритет каналов DMA0 – DMA1 – DMA2. Если установлен бит ROUNDROBIN, то после завершения пересылки каналу назначается минимальный приоритет (т.е. при постоянном одновременном срабатывании триггеров обслуживание каналов будет циклическим). Если срабатывает триггер канала с большим приоритетом, текущая пересылка не прерывается.

Для каждого канала выбор сигнала источника для триггера выполняется битами DMAxTSEL, при этом обязательно DMAEN должен быть равен 0. Перечень источников одинаков для каждого из каналов и приведен в таблице:

Источники сигналов триггера для DMA-каналов

№	Источник	Описание и определение флагов в msp430f5529.h
0	DMAREQ	Запрос DMA (программный запуск). Триггер срабатывает при установке бита. Сигнал DMAREQ автоматически сбрасывается после начала пересылки. Определение: DMA0TSEL__DMA_REQ

№	Источник	Описание и определение флагов в msp430f5529.h
1	TA0CCR0 CCIFG	Запуск по каналам таймеров. Триггер срабатывает при установке бита. Соответствующий сигнал CCIFG автоматически сбрасывается после начала пересылки. Если установлен соответствующий бит CCIE, выбранный флаг CCIFG не запускает пересылку DMA. Макроопределения: DMA0TSEL__TA0CCR0 .. DMA0TSEL__TB0CCR2
2	TA0CCR2 CCIFG	
3	TA1CCR0 CCIFG	
4	TA1CCR2 CCIFG	
5	TA2CCR0 CCIFG	
6	TA2CCR2 CCIFG	
7	TB0CCR0 CCIFG	
8	TB0CCR2 CCIFG	
16	UCA0RXIFG	Запуск по каналам USCI. Триггер срабатывает при приеме (RX)/готовности к передаче (TX) данных по соответствующему каналу USCI. Сигнал RXIFG/TXIFG автоматически сбрасывается после начала пересылки. Если установлен соответствующий бит RXIE/TXIE, выбранный флаг RXIFG/TXIFG не запускает пересылку DMA. Макроопределения: DMA0TSEL__USCIA0RX .. DMA0TSEL__USCIB1TX
17	UCA0TXIFG	
18	UCB0RXIFG	
19	UCB0TXIFG	
20	UCA1RXIFG	
21	UCA1TXIFG	
22	UCB1RXIFG	
23	UCB1TXIFG	
24	ADC12IFGx	Запуск по АЦП. Триггер срабатывает при установке бита (завершении одно-канального преобразования АЦП или завершении последнего преобразования в последовательности). Программная установка бита не запускает триггер. Все ADC12IFG флаги автоматически сбрасываются, когда к соответствующему ADC12MEMx обратился контроллер DMA. Макроопределение: DMA0TSEL__ADC12IFG
27	USB FNRXD	USB триггер. Макроопределение: DMA0TSEL__USB_FNRXD
28	USB ready	USB триггер. Макроопределение: DMA0TSEL__USB_READY
29	MPY ready	Запуск по умножителю. Триггер срабатывает, когда умножитель готов для нового операнда. Макроопределение: DMA0TSEL__MPY
30	DMAxIFG	DMA2IFG – для канала 0, DMA0IFG – для канала 1, DMA1IFG – для канала 2. Триггер срабатывает при установке бита. Сигнал DMAxIFG не сбрасывается автоматически. Макроопределения: DMA0TSEL__DMA2IFG, DMA1TSEL__DMA0IFG, DMA2TSEL__DMA1IFG
31	DMAE0	Пересылка по внешнему сигналу триггера. Макроопределение: DMA0TSEL__DMAE0

Контроллер DMA требует 1-2 тактов MCLK для синхронизации перед каждым обменом, потом 2 такта MCLK на пересылку байта либо слова и 1 такт ожидания после пересылки. Таким образом, пересылка займет 4-5 тактов. В случае, если источник MCLK выключен, контроллер DMA временно включает MCLK, генерируемую DCOCLK, для

выполнения пересылки. В этом случае дополнительно потребуется еще 5 мкс для запуска DCOCLK.

Пересылки DMA не прерываются системными прерываниями, прерывания ожидают завершения пересылки. Только прерывание NMI может прервать пересылку, если установлен бит ENNMI. Выполнение обработчиков прерываний приостанавливается для DMA пересылки. Чтобы этого не происходило, на время выполнения обработчика прерываний следует отключать DMA контроллер.

Каждый канал DMA имеет собственный флаг DMAIFG. Флаг устанавливается, когда соответствующий DMAxSZ становится равным нулю. Если при этом установлены флаги DMAIE и GIE, возникает запрос на прерывание.

Состав регистров контроллера DMA и назначение отдельных полей приведены в таблицах 6.4 и 6.5. Другие флаги для выбора источников триггеров указаны в табл. 6.3.

Регистры контроллера DMA

Регистр	Адрес	Назначение
DMACTL0	0500h	Общий регистр управления DMA
DMACTL1	0502h	Общий регистр управления DMA
DMACTL4	0508h	Общий регистр управления DMA
DMAIV	050Eh	Вектор прерываний DMA
DMA0CTL	0510h	Управление каналом 0 DMA
DMA0SA	0512h	Адрес источника канала 0 DMA
DMA0DA	0516h	Адрес приемника канала 0 DMA
DMA0SZ	051Ah	Размер пересылки канала 0 DMA
DMA1CTL - DMA1SZ	0520h - 052Ah	Аналогичные регистры канала 1 DMA
DMA2CTL-DMA2SZ	0530h - 053Ah	Аналогичные регистры канала 2 DMA

Поля регистров контроллера DMA

Регистр	Биты	Поле	Назначение	Определение флагов в msp430f5529.h
DMACTL0	8-12	DMA1TSEL	Выбор источника триггера канала 1 DMA	DMA1TSEL_0.. DMA1TSEL_31
	0-4	DMA0TSEL	Выбор источника триггера канала 1 DMA	DMA0TSEL_0.. DMA0TSEL_31
ACT	4-0	DMA2TSEL	Выбор источника триггера канала 1 DMA	DMA2TSEL_0.. DMA2TSEL_31
DMACTL4	2	DMARMWDIS	Запрет прерывания цикла операции процессора на шине (чтение/изменение/ запись).Пересылка DMA ожидает завершения операции процессора. Если бит не установлен, DMA пересылка	DMARMWDIS

Регистр	Биты	Поле	Назначение	Определение флагов в msp430f5529.h
			может прерывать операцию процессора.	
	1	ROUNDROBIN	Установка циклического приоритета каналов	ROUNDROBIN
	0	ENNMI	Разрешение прерывания DMAпересылки посредством NMI	ENNMI
DMAxCTL	12-14	DMADT	Режим пересылки	DMADT_0 .. DMADT_7
	10-11	DMADSTINCR	Инкремент адреса назначения после пересылки (при пересылке слов адрес изменяется на 2): 00,01 — без изменений, 10 — декремент, 11 - инкремент	DMADSTINCR_0.. DMADSTINCR_3
	8-9	DMASRCINCR	Инкремент адреса источника после пересылки (при пересылке слов адрес изменяется на 2): 00,01 — без изменений, 10 — декремент, 11 - инкремент	DMASRCINCR_0.. DMASRCINCR_3
	7	DMADSTBYTE	Размер данных приемника: 0 - слово, 1 - байт	DMASRCBYTE, DMADSTBYTE, DMASWDW, DMASBDW, DMASWDB, DMASBDB
	6	DMASRCBYTE	Размер данных источника: 0 - слово, 1 - байт	
	5	DMALEVEL	Режим срабатывания триггера: 0 — по переднему фронту, 1 — по высокому уровню	DMALEVEL
	4	DMAEN	Разрешение DMA (=1)	DMAEN
	3	DMAIFG	Флаг прерывания DMA	DMAIFG
	2	DMAIE	Разрешение прерывания DMA	DMAIE
	1	DMAABORT	Флаг, устанавливается, если пересылка DMA была прервана NMI	DMAABORT
	0	DMAREQ	Запрос DMA. Программно-управляемый за-пуск пересылки. Сбрасывается автоматически	DMAREQ
DMAxSA	0-19	DMAxSA	Адрес источника. Обращение к регистру требует расширенных операций. Использо-вание операций для слов очищает регистр	DMA0SA.. DMA2SA
DMAxDA	0-19	DMAxDA	Адрес назначения. Обращение к регистру требует расширенных операций. Использование операций для слов очищает регистр	DMA0DA.. DMA2DA
D	0-15	DMAxSZ	Количество передаваемых данных (байт	DMA0SZ..

Регистр	Биты	Поле	Назначение	Определение флагов в msp430f5529.h
			или слов)	DMA2SZ
DMAIV	0-15	DMAIV	Вектор прерываний	DMAIV_NONE, DMAIV_DMA0IFG.. DMAIV_DMA2IFG

Листинг кода

Structure.h

```
/* --COPYRIGHT--,BSD
 * Copyright (c) 2012, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
```

```
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * --/COPYRIGHT--*/
/*
 *  structure.h
 *  This example of the RO_COMPB_TA1_TA0 implementation uses 5 elements and one
 *  sensor.
 *
 */
//*************************************************************************************************
// The following elements need to be configured by the user.
//*************************************************************************************************
#ifndef CTS_STRUCTURE_H_
#define CTS_STRUCTURE_H_

#include "msp430.h"
#include <stdint.h>

/* Public Globals */
// Identify all elements defined in structure.c
extern const struct Element PAD1;
extern const struct Element PAD2;
extern const struct Element PAD3;
extern const struct Element PAD4;
extern const struct Element PAD5;

// Identify all sensors defined in structure.c
extern const struct Sensor keypad;
```

```
//****** RAM ALLOCATION ************************************************************
// TOTAL_NUMBER_OF_ELEMENTS represents the total number of elements used, even if
// they are going to be segmented into seperate groups.  This defines the
// RAM allocation for the baseline tracking.  If only the TI_CAPT_Raw function
// is used, then this definition should be removed to conserve RAM space.
#define TOTAL_NUMBER_OF_ELEMENTS 5
// If the RAM_FOR_FLASH definition is removed, then the appropriate HEAP size
// must be allocated. 2 bytes * MAXIMUM_NUMBER_OF_ELEMENTS_PER_SENSOR + 2 bytes
// of overhead.
#define RAM_FOR_FLASH
//****** Structure Array Definition ********************************************
// This defines the array size in the sensor strucure.  In the event that
// RAM_FOR_FLASH is defined, then this also defines the amount of RAM space
// allocated (global variable) for computations.
#define MAXIMUM_NUMBER_OF_ELEMENTS_PER_SENSOR  5
//****** Choosing a Measurement Method ***************************************
// These variables are references to the definitions found in structure.c and
// must be generated per the application.
// possible values for the method field

// OSCILLATOR DEFINITIONS
//#define RO_COMPAp_TA0_WDTp            64
//#define RO_PINOSC_TA0_WDTp            65
//#define RO_PINOSC_TA0                 66
//#define RO_COMPAp_TA1_WDTp            67
//#define RO_COMPB_TA0_WDTA             68
//#define RO_COMPB_TA1_WDTA             69
//#define RO_COMPB_TB0_WDTA             70
#define RO_COMPB_TA1_TA0                71
//#define RO_PINOSC_TA0_TA1             72
//#define RO_CSIO_TA2_WDTA              73
//#define RO_CSIO_TA2_TA3               74
//#define RO_PINOSC_TA1_WDTp            75
//#define RO_PINOSC_TA1_TB0             76
```

```
        // RC DEFINITIONS
        //#define RC_PAIR_TA0               01

        // FAST RO DEFINITIONS
        //#define fRO_CSIO_TA2_TA3         23
        //#define fRO_PINOSC_TA0_TA1       24
        //#define fRO_PINOSC_TA0_SW        25
        //#define fRO_COMPB_TA0_SW         26
        //#define fRO_COMPB_TA1_SW         27
        //#define fRO_COMPAp_TA0_SW        28
        //#define fRO_COMPAp_SW_TA0        29
        //#define fRO_COMPAp_TA1_SW        30
        //#define fRO_COMPB_TA1_TA0        31
        //#define fRO_PINOSC_TA1_TA0           32
        //#define fRO_PINOSC_TA1_TB0           33

        //******                WHEEL                and                SLIDER
**********************************************************
        // Are wheel or slider representations used?
        //#define SLIDER
        //#define ILLEGAL_SLIDER_WHEEL_POSITION            0xFFFF
        //#define WHEEL

        //*****************************************************************
******************
// End of user configuration section.
//*****************************************************************
**************
//*****************************************************************
**************
//*****************************************************************
**************

//possible timer source clock dividers, different from clock module dividers
#define TIMER_TxCLK          0x0000  // TxSSEL
#define TIMER_ACLK    0x0100
#define TIMER_SMCLK          0x0200
#define TIMER_INCLK          0x0300
```

```
#define TIMER_SOURCE_DIV_0 0x0000  // ID_0, IDX_0
#define TIMER_SOURCE_DIV_1 0x0040
#define TIMER_SOURCE_DIV_2 0x0080
#define TIMER_SOURCE_DIV_3 0x00C0

#define GATE_WDT_ACLK      0x0004
#define GATE_WDT_SMCLK     0x0000
#define GATE_WDTp_ACLK     0x0004
#define GATE_WDTp_SMCLK    0x0000

#define WDTp_GATE_32768    0x0000  // watchdog source/32768
#define WDTp_GATE_8192     0x0001  // watchdog source/8192
#define WDTp_GATE_512      0x0002  // watchdog source/512
#define WDTp_GATE_64       0x0003  // watchdog source/64

#define GATE_WDTA_SMCLK    0x0000
#define GATE_WDTA_ACLK     0x0020
#define GATE_WDTA_VLO      0x0040
#define GATE_WDTA_XCLK     0x0060

#define WDTA_GATE_2G       0x0000  // watchdog source/2G
#define WDTA_GATE_128M     0x0001  // watchdog source/128M
#define WDTA_GATE_8192K    0x0002  // watchdog source/8192K
#define WDTA_GATE_512K     0x0003  // watchdog source/512K
#define WDTA_GATE_32768    0x0004  // watchdog source/32768
#define WDTA_GATE_8192     0x0005  // watchdog source/8192
#define WDTA_GATE_512      0x0006  // watchdog source/512
#define WDTA_GATE_64       0x0007  // watchdog source/64

// The below variables are used to excluded portions of code not needed by
// the method chosen by the user. Uncomment the type used prior to compilation.
// Multiple types can be chosen as needed.
// What Method(s) are used in this application?

#ifdef RO_COMPAp_TA0_WDTp
   #define RO_TYPE
   #define RO_COMPAp_TYPE
```

```
#define WDT_GATE
#define HAL_DEFINITION
//what devices have Pxsel2 ??
// msp430f2112, 2122, 2132
// msp430G2112, G2212, G2312, G2412, G2152, G2252, G2352, G2452
// SEL2REGISTER
#ifdef __MSP430F2112
  #define SEL2REGISTER
#endif
#ifdef __MSP430F2122
  #define SEL2REGISTER
#endif
#ifdef __MSP430F2132
  #define SEL2REGISTER
#endif
#ifdef __MSP430G2112
  #define SEL2REGISTER
#endif
#ifdef __MSP430G2212
  #define SEL2REGISTER
#endif
#ifdef __MSP430G2312
  #define SEL2REGISTER
#endif
#ifdef __MSP430G2412
  #define SEL2REGISTER
#endif
#ifdef __MSP430G2152
  #define SEL2REGISTER
#endif
#ifdef __MSP430G2252
  #define SEL2REGISTER
#endif
#ifdef __MSP430G2352
  #define SEL2REGISTER
#endif
#ifdef __MSP430G2452
```

```
      #define SEL2REGISTER
    #endif
#endif

#ifdef RO_PINOSC_TA0_WDTp
    #define RO_TYPE
    #define RO_PINOSC_TYPE
    #define WDT_GATE
    #define HAL_DEFINITION
#endif

#ifdef RO_PINOSC_TA0
    #define RO_TYPE
    #define RO_PINOSC_TYPE
    #define ACCUMULATE_TYPE
    #define HAL_DEFINITION
#endif

#ifdef RO_COMPAp_TA1_WDTp
    #define RO_TYPE
    #define RO_COMPAp_TYPE
    #define WDT_GATE
    #define HAL_DEFINITION
#endif

#ifdef RO_COMPB_TA0_WDTA
    #define RO_TYPE
    #define RO_COMPB_TYPE
    #define WDT_GATE
    #define HAL_DEFINITION
#endif

#ifdef RO_COMPB_TA1_WDTA
    #define RO_TYPE
    #define RO_COMPB_TYPE
    #define WDT_GATE
    #define HAL_DEFINITION
```

```
#endif

#ifdef RO_COMPB_TB0_WDTA
   #define RO_TYPE
   #define RO_COMPB_TYPE
   #define WDT_GATE
   #define HAL_DEFINITION
#endif

#ifdef RC_PAIR_TA0
   #define RC_TYPE
   #define RC_PAIR_TYPE
   #define ACCUMULATE_TYPE
   #define HAL_DEFINITION
#endif

#ifdef fRO_PINOSC_TA0_SW
        #define RO_TYPE
   #define RO_PINOSC_TYPE
   #define TIMER_SCALE
        #define HAL_DEFINITION
#endif

#ifdef fRO_COMPB_TA0_SW
   #define RO_TYPE
   #define RO_COMPB_TYPE
   #define TIMER_SCALE
        #define HAL_DEFINITION
#endif

#ifdef fRO_COMPB_TA1_SW
   #define RO_TYPE
   #define RO_COMPB_TYPE
   #define TIMER_SCALE
        #define HAL_DEFINITION
#endif
```

```
#ifdef fRO_COMPAp_TA0_SW
   #define RO_TYPE
   #define RO_COMPAp_TYPE
   #define TIMER_SCALE
        #define HAL_DEFINITION
#endif

#ifdef fRO_COMPAp_TA1_SW
   #define RO_TYPE
   #define RO_COMPAp_TYPE
   #define TIMER_SCALE
   #define HAL_DEFINITION
#endif

#ifdef fRO_COMPAp_SW_TA0
   #define RO_TYPE
   #define RO_COMPAp_TYPE
   #define HAL_DEFINITION
#endif

#ifdef RO_COMPB_TA1_TA0
   #define RO_TYPE
   #define RO_COMPB_TYPE
   #define TIMER_SCALE
   #define TIMER0A0_GATE
   #define HAL_DEFINITION
#endif

#ifdef fRO_COMPB_TA1_TA0
   #define RO_TYPE
   #define RO_COMPB_TYPE
   #define TIMER_SCALE
   #define TIMER1A0_GATE
   #define HAL_DEFINITION
#endif

#ifdef RO_PINOSC_TA0_TA1
```

```
    #define RO_TYPE
    #define RO_PINOSC_TYPE
    #define TIMER_SCALE
    #define TIMER1A0_GATE
    #define HAL_DEFINITION
#endif

#ifdef fRO_PINOSC_TA0_TA1
    #define RO_TYPE
    #define RO_PINOSC_TYPE
    #define TIMER_SCALE
    #define TIMER0A0_GATE
    #define HAL_DEFINITION
#endif

#ifdef RO_CSIO_TA2_WDTA
    #define RO_TYPE
    #define RO_CSIO_TYPE
    #define WDT_GATE
    #define HAL_DEFINITION
#endif

#ifdef RO_CSIO_TA2_TA3
    #define RO_TYPE
    #define RO_CSIO_TYPE
    #define TIMER_SCALE
    #define TIMER3A0_GATE
    #define HAL_DEFINITION
#endif

#ifdef fRO_CSIO_TA2_TA3
    #define RO_TYPE
    #define RO_CSIO_TYPE
    #define TIMER_SCALE
    #define TIMER2A0_GATE
    #define HAL_DEFINITION
#endif
```

```
#ifdef RO_PINOSC_TA1_WDTp
   #define RO_TYPE
   #define RO_PINOSC_TYPE
   #define WDT_GATE
   #define HAL_DEFINITION
#endif

#ifdef RO_PINOSC_TA1_TB0
   #define RO_TYPE
   #define RO_PINOSC_TYPE
   #define TIMER_SCALE
   #define TIMERB0_GATE
   #define HAL_DEFINITION
#endif

#ifdef fRO_PINOSC_TA1_TA0
   #define RO_TYPE
   #define RO_PINOSC_TYPE
   #define TIMER_SCALE
   #define TIMER1A0_GATE
   #define HAL_DEFINITION
#endif

#ifdef fRO_PINOSC_TA1_TB0
        #define RO_TYPE
        #define RO_PINOSC_TYPE
        #define TIMER_SCALE
        #define TIMER1A0_GATE
        #define HAL_DEFINITION
#endif



#ifdef SLIDER
        #define SLIDER_WHEEL
#endif
```

```
#ifdef WHEEL
        #define SLIDER_WHEEL
#endif

#define RO_MASK         0xC0        // 1100 0000
#define RC_FRO_MASK     0x3F        // 0011 1111

/*
 *  The element structure identifies port or comparator input definitions for
 *  each element.
 */
struct Element{

#ifdef RO_PINOSC_TYPE
// These register address definitions are needed for each sensor only
// when using the PinOsc method
  uint8_t *inputPxselRegister;    // PinOsc: port selection address
  uint8_t *inputPxsel2Register;   // PinOsc: port selection 2 address
#endif

#ifdef RC_PAIR_TYPE
// these fields are specific to the RC type.
  uint8_t *inputPxoutRegister;    // RC: port output address: PxOUT
  volatile uint8_t *inputPxinRegister;     // RC: port input address: PxIN
  uint8_t *inputPxdirRegister;    // RC+PinOsc: port direction address
  uint8_t *referencePxoutRegister;// RC: port output address: PxOUT
  uint8_t *referencePxdirRegister;// RC: port direction address: PxDIR
  uint8_t referenceBits;          // RC: port bit definition
#endif

  uint16_t inputBits;               // Comp_RO+FastRO+RC+PinOsc: bit
                                    // definition
                                    //
                                    // for comparator input bit
                                    // location in CACTL2 or CBCTL0
```

```
  uint16_t threshold;                // specific threshold for each button
  uint16_t maxResponse;              // Special Case: Slider max counts
};

/*
 *  The sensor structure identifies HAL and timing definitions for
 *  each sensor.
 */

struct Sensor{
  // the method acts as the switch to determine which HAL is called
  uint8_t halDefinition;          // COMPARATOR_TYPE (RO), RC, etc
                                  // RO_COMPA, RO_COMPB, RO_PINOSC
                                  // RC_GPIO, RC_COMPA, RC_COMPB
                                  // FAST_SCAN_RO
#ifdef RO_CSIO_TYPE
/*
 *  This register address definition is needed to indicate which CSIOxCTL
 *  register is associated with the Timer identified in the HAL.
 */
  uint16_t *inputCapsioctlRegister;
#endif

  uint8_t numElements;            // number of elements within group
  uint8_t baseOffset;             // the offset within the global
                                  // base_cnt array

  struct                                  Element                                  const
*arrayPtr[MAXIMUM_NUMBER_OF_ELEMENTS_PER_SENSOR];
                                  // an array of pointers

//*****************************************************************************
**************
// Reference structure definitions for comparator types, for the RC method the
// reference is defined within the element.

#ifdef RO_COMPAp_TYPE
```

```
  uint8_t * refPxoutRegister;      // RO+FastRO: port output address
  uint8_t * refPxdirRegister;      // RO+FastRO: port direction address
  uint8_t refBits;                 // RO+FastRO: port bit definition

  uint8_t * txclkDirRegister;      // PxDIR
  uint8_t * txclkSelRegister;      // PxSEL
  uint8_t txclkBits;               // Bit field for register

  uint8_t *caoutDirRegister;       // PxDIR
  uint8_t *caoutSelRegister;       // PxSEL
  uint8_t caoutBits;               // Bit field for register

  // This is only applicable to the RO_COMPAp_TYPE
#ifdef SEL2REGISTER
  uint8_t *caoutSel2Register;
  uint8_t *txclkSel2Register;
#endif

  uint8_t refCactl2Bits;           // RO: CACTL2 input definition,
                                   // CA0 (P2CA0),CA1(P2CA4),
                                   // CA2(P2CA0+P2CA4)
  uint8_t capdBits;
#endif

#ifdef RO_COMPB_TYPE
  uint8_t *cboutTAxDirRegister;  // CBOUT_TA0CLK
  uint8_t *cboutTAxSelRegister;  // CBOUT_TA0CLK
  uint8_t cboutTAxBits;          // Bit field for register
  uint16_t cbpdBits;
#endif

//*****************************************************************************
// Timer definitions
//  The basic premise is to count a number of clock cycles within a time
//  period, where either the clock source or the timer period is a function
//  of the element capacitance.
```

```
//
// RC Method:
//        Period: accumulationCycles * charge and discharge time of RC
//        circuit where C is capacitive touch element
//
//        clock source: measGateSource/sourceScale
// RO Method:
//        Period: accumulationCycles*measGateSource/sourceScale
//              (with WDT sourceScale = 1, accumulationCycles is WDT control
//               register settings)
//
//        clock source: relaxation oscillator where freq is a function of C
//
// fRO Method:
//        Period: accumulationCycles * 1/freq, freq is a function of C
//
//        clock source: measGateSource/sourceScale

  uint16_t measGateSource;        // RC+FastRO: measurement timer source,
                                  // {ACLK, TACLK, SMCLK}
                                  // Comp_RO+PinOsc: gate timer source,
                                  // {ACLK, TACLK, SMCLK}
#ifdef TIMER_SCALE
  uint16_t sourceScale;           // RO+FastRO: gate timer,
                                  // TA/TB, scale: 1,1/2,1/4,1/8
                                  // RC+FastRO: measurement timer, TA/TB/TD
                                  // scale: 16, 8, 4, 2, 1, ?, ?, 1/8
                                  // Not used for WDTp/WDTA
#endif

  uint16_t accumulationCycles;

//*****************************************************************************
// Other definitions

#ifdef SLIDER_WHEEL
```

```
  uint8_t points;                 // Special Case: Number of points
                                  // along slider or wheel
  uint8_t sensorThreshold;
#endif

};

/*
 *  The GCC language extension within CCS is needed, otherwise a warning will
 *  be generated during compilation when no problems exist or an error will be
 *  generated (instead of a warning) when a problem does exist.
 */
#ifndef TOTAL_NUMBER_OF_ELEMENTS
 #warning "WARNING: TOTAL_NUMBER_OF_ELEMENTS is not defined in structure.h. Only TI_CAPT_RAW function is enabled."
#endif

#ifndef RAM_FOR_FLASH
 #warning "WARNING: The HEAP must be set appropriately.  Please refer to SLAA490 for details."
#endif

#ifndef HAL_DEFINITION
 #warning "WARNING: At least one HAL definition must be made in structure.h."
#endif

#endif /* CTS_STRUCTURE_H_ */
```

Structure.c

```
/* --COPYRIGHT--,BSD
 * Copyright (c) 2012, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
```

```
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * --/COPYRIGHT--*/
/*
 *  structure.c
 *  RO_COMPB_TA1_TA0 example with the MSP430F5529 Experimenters Board
```

```
 *
 *  Schematic
 *   PAD5-+----------------------------R--+--<P1.6/TA1CLK/CBOUT
 *        |                             |
 *        | PAD4-+----------------------R--+
 *        |      |                      |
 *        |      | PAD3-+---------------R--+
 *        |      |      |               |
 *        |      |      | PAD2-+--------R--+
 *        |      |      |      |         |
 *        |      |      |      | PAD1-+-R--+
 *        |      |      |      |      |
 *        |      |      |      |      +-------->CB0
 *        |      |      |      +--------------->CB1
 *        |      |      +---------------------->CB2
 *        |      +----------------------------->CB3
 *        +-------------------------------->CB4
 *
 *  In this example the five pads on the experimenter's board are grouped
 *  together in the sensor keypad.
 *
 *  The element and sensor definitions found in the configuration file
 *  structure.c use designated initializer lists. This allows members to be
 *  initialized in any order and also enhances the readability of the element
 *  being initialized.
 *  This feature requires the GCC language extension found in Code Composer
 *  Studio (CCS).  C99 is the default dialect found in IAR and therefore the
 *  default settings can be used.
 */
#include "structure.h"
/*
 * The element defines the input of the comparator mux, the maximum response,
 * and the threshold. The threshold and maxResponse values are based upon the gate
time, defined
 * in the keypad sensor definition.  In this example the gate time is 1.5ms.
 */
const struct Element PAD1 =
```

```
{   //CB0
    .inputBits = CBIMSEL_0,
    .maxResponse = 250,
    .threshold = 125
};
const struct Element PAD2 =
{   //CB1
    .inputBits = CBIMSEL_1,
    .maxResponse = 390,
    .threshold = 195
};
const struct Element PAD3 =
{   //CB2
    .inputBits = CBIMSEL_2,
    .maxResponse = 340,
    .threshold = 170
};
const struct Element PAD4 =
{   //CB3
    .inputBits = CBIMSEL_3,
    .maxResponse = 500,
    .threshold = 230
};
const struct Element PAD5 =
{   //CB4
    .inputBits = CBIMSEL_4,
    .maxResponse = 400,
    .threshold = 200
};
/*
 *  The sensor defines the grouping of elements, the method to measure change in
 *  capacitance, and the measurement time of each element.
 */
const struct Sensor keypad =
{
    .halDefinition = RO_COMPB_TA1_TA0,
    .numElements = 5,
```

```
    .baseOffset = 0,
    .cbpdBits = 0x001F, //CB0,CB1,CB2,CB3,CB4
    .arrayPtr[0] = &PAD1,
    .arrayPtr[1] = &PAD2,
    .arrayPtr[2] = &PAD3,
    .arrayPtr[3] = &PAD4,
    .arrayPtr[4] = &PAD5,
    .cboutTAxDirRegister = (uint8_t *)&P1DIR,  // PxDIR
    .cboutTAxSelRegister = (uint8_t *)&P1SEL,  // PxSEL
    .cboutTAxBits = BIT6, // P1.6
    // Timer Information
    .measGateSource = TIMER_ACLK,      //  ACLK
    .sourceScale = TIMER_SOURCE_DIV_0, // ACLK/1
    /*  50 ACLK/1 cycles or 50*1/32Khz = 1.5ms  */
    .accumulationCycles = 50
};
```

mmc.h

```
uint8_t detectCard(void);
```

mmc.c

```
/*-----------------------------------------------------------------------/
/  Bitbanging MMCv3/SDv1/SDv2 (in SPI mode) control module
/-----------------------------------------------------------------------/
/
/  Copyright (C) 2010, ChaN, all right reserved.
/
/ * This software is a free software and there is NO WARRANTY.
/ * No restriction on use. You can use, modify and redistribute it for
/  personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY.
/ * Redistributions of source code must retain the above copyright notice.
/
------------------------------------------------------------------------*/

#include <intrinsics.h>         /* Include MSP430-specific intrincs */
```

```
#include "diskio.h"            /* Common include file for FatFs and disk I/O layer */
#include "HAL_SDCard.h"        /* MSP-EXP430F5529 specific SD Card driver */

/*-----------------------------------------------------------------------*/
/* Platform dependent macros and functions needed to be modified         */
/*-----------------------------------------------------------------------*/

// CPU Frequency.
#define MCLK_FREQ   25000000
#define       INIT_PORT()     SDCard_init()       /* Initialize MMC control port */
#define FAST_MODE()       SDCard_fastMode()    /* Maximize SD Card transfer speed */
#define DLY_US(n)       __delay_cycles(n * (MCLK_FREQ/1000000))  // Delay n microseconds          // KLQ

#define       CS_H()          SDCard_setCSHigh()  /* Set MMC CS "high" */
#define CS_L()          SDCard_setCSLow()   /* Set MMC CS "low" */

BYTE INS = 1;    // KLQ
#define       WP              (0)             /* Card is write protected (yes:true, no:false, default:false) */

/*-----------------------------------------------------------------------*/
/* Platform dependent RTC Function for FatFs module                      */
/*-----------------------------------------------------------------------*/
DWORD get_fattime(void)
{
        DWORD tmr;

   // TODO: Customize to use the MSP430 RTC

        /* Pack date and time into a DWORD variable */
        tmr =   (((DWORD)2001 - 80) << 25)    // rtcYear
                        | ((DWORD)9 << 21)              // rtcMon
                        | ((DWORD)11 << 16)             // rtcMday
                        | (WORD)(4 << 11)               // rtcHour
                        | (WORD)(30 << 5)               // rtcMin
```

```
                        | (WORD)(0 >> 1);             // rtcSec

        return tmr;
}

/*-----------------------------------------------------------------------
  Module Private Functions
-----------------------------------------------------------------------*/

/* MMC/SD command (SPI mode) */
#define CMD0     (0)                /* GO_IDLE_STATE */
#define CMD1     (1)                /* SEND_OP_COND */
#define      ACMD41    (0x80+41)   /* SEND_OP_COND (SDC) */
#define CMD8     (8)                /* SEND_IF_COND */
#define CMD9     (9)                /* SEND_CSD */
#define CMD10    (10)          /* SEND_CID */
#define CMD12    (12)          /* STOP_TRANSMISSION */
#define ACMD13 (0x80+13)   /* SD_STATUS (SDC) */
#define CMD16    (16)          /* SET_BLOCKLEN */
#define CMD17    (17)          /* READ_SINGLE_BLOCK */
#define CMD18    (18)          /* READ_MULTIPLE_BLOCK */
#define CMD23    (23)          /* SET_BLOCK_COUNT */
#define      ACMD23    (0x80+23)   /*  SET_WR_BLK_ERASE_COUNT  (SDC)
*/
#define CMD24    (24)          /* WRITE_BLOCK */
#define CMD25    (25)          /* WRITE_MULTIPLE_BLOCK */
#define CMD41    (41)          /* SEND_OP_COND (ACMD) */
#define CMD55    (55)          /* APP_CMD */
#define CMD58    (58)          /* READ_OCR */

/* Card type flags (CardType) */
#define CT_MMC       0x01        /* MMC ver 3 */
#define CT_SD1       0x02        /* SD ver 1 */
#define CT_SD2       0x04        /* SD ver 2 */
#define CT_SDC       (CT_SD1|CT_SD2)       /* SD */
#define CT_BLOCK     0x08        /* Block addressing */
```

```
static
DSTATUS Stat = STA_NOINIT;      /* Disk status */

static
BYTE CardType;                  /*  b0:MMC,  b1:SDv1,  b2:SDv2,  b3:Block
addressing */



/*-----------------------------------------------------------------------*/
/* Transmit bytes to the MMC                                             */
/*-----------------------------------------------------------------------*/

static
void xmit_mmc (
        const BYTE* buff,           /* Data to be sent */
        UINT bc                     /* Number of bytes to send */
)
{
   SDCard_sendFrame((uint8_t *)buff, bc);
}

/*-----------------------------------------------------------------------*/
/* Receive bytes from the MMC                                            */
/*-----------------------------------------------------------------------*/

static
void rcvr_mmc (
        BYTE *buff,         /* Pointer to read buffer */
        UINT bc             /* Number of bytes to receive */
)
{

   SDCard_readFrame(buff, bc);
}
```

```
/*-----------------------------------------------------------------------*/
/* Wait for card ready                                                   */
/*-----------------------------------------------------------------------*/

static
int wait_ready (void)    /* 1:OK, 0:Timeout */
{
   BYTE d;
   UINT tmr;


   for (tmr = 5000; tmr; tmr--) {    /* Wait for ready in timeout of 500ms */
      rcvr_mmc(&d, 1);
      if (d == 0xFF) return 1;
      DLY_US(100);
   }

   return 0;
}



/*-----------------------------------------------------------------------*/
/* Deselect the card and release SPI bus                                 */
/*-----------------------------------------------------------------------*/

static
void deselect (void)
{
   BYTE d;

   CS_H();
   rcvr_mmc(&d, 1);
}
```

```
/*-----------------------------------------------------------------------*/
/* Select the card and wait for ready                                    */
/*-----------------------------------------------------------------------*/

static
int select (void)    /* 1:OK, 0:Timeout */
{
   CS_L();
   if (!wait_ready()) {
      deselect();
      return 0;
   }
   return 1;
}



/*-----------------------------------------------------------------------*/
/* Receive a data packet from MMC                                        */
/*-----------------------------------------------------------------------*/

static
int rcvr_datablock (    /* 1:OK, 0:Failed */
   BYTE *buff,           /* Data buffer to store received data */
   UINT btr          /* Byte count */
)
{
   BYTE d[2];
   UINT tmr;


   for (tmr = 1000; tmr; tmr--) {    /* Wait for data packet in timeout of 100ms */
      rcvr_mmc(d, 1);
      if (d[0] != 0xFF) break;
      DLY_US(100);
   }
```

```
   if (d[0] != 0xFE) return 0;        /* If not valid data token, retutn with error */

   rcvr_mmc(buff, btr);            /* Receive the data block into buffer */
   rcvr_mmc(d, 2);                  /* Discard CRC */

   return 1;                        /* Return with success */
}



/*-----------------------------------------------------------------------*/
/* Send a data packet to MMC                                             */
/*-----------------------------------------------------------------------*/

static
int xmit_datablock (    /* 1:OK, 0:Failed */
   const BYTE *buff,    /* 512 byte data block to be transmitted */
   BYTE token           /* Data/Stop token */
)
{
   BYTE d[2];


   if (!wait_ready()) return 0;

   d[0] = token;
   xmit_mmc(d, 1);               /* Xmit a token */
   if (token != 0xFD) {        /* Is it data token? */
      xmit_mmc(buff, 512);    /* Xmit the 512 byte data block to MMC */
      rcvr_mmc(d, 2);           /* Dummy CRC (FF,FF) */
      rcvr_mmc(d, 1);           /* Receive data response */
      if ((d[0] & 0x1F) != 0x05)    /* If not accepted, return with error */
         return 0;
   }

   return 1;
}
```

```
/*-----------------------------------------------------------------------*/
/* Send a command packet to MMC                                          */
/*-----------------------------------------------------------------------*/

static
BYTE send_cmd (        /* Returns command response (bit7==1:Send failed)*/
   BYTE cmd,        /* Command byte */
   DWORD arg        /* Argument */
)
{
   BYTE n, d, buf[6];


   if (cmd & 0x80) {    /* ACMD<n> is the command sequense of CMD55-CMD<n>
*/
      cmd &= 0x7F;
      n = send_cmd(CMD55, 0);
      if (n > 1) return n;
   }

   /* Select the card and wait for ready */
   deselect();
   if (!select()) return 0xFF;

   /* Send a command packet */
   buf[0] = 0x40 | cmd;           /* Start + Command index */
   buf[1] = (BYTE)(arg >> 24);        /* Argument[31..24] */
   buf[2] = (BYTE)(arg >> 16);        /* Argument[23..16] */
   buf[3] = (BYTE)(arg >> 8);        /* Argument[15..8] */
   buf[4] = (BYTE)arg;                /* Argument[7..0] */
   n = 0x01;                        /* Dummy CRC + Stop */
   if (cmd == CMD0) n = 0x95;        /* (valid CRC for CMD0(0)) */
   if (cmd == CMD8) n = 0x87;        /* (valid CRC for CMD8(0x1AA)) */
   buf[5] = n;
```

```
   xmit_mmc(buf, 6);

   /* Receive command response */
   if (cmd == CMD12) rcvr_mmc(&d, 1);    /* Skip a stuff byte when stop reading
*/
   n = 10;                              /* Wait for a valid response in timeout of 10 attempts
*/
   do
      rcvr_mmc(&d, 1);
   while ((d & 0x80) && --n);

   return d;           /* Return with the response value */
}



/*-----------------------------------------------------------------------

  Public Functions

-----------------------------------------------------------------------*/


/*-----------------------------------------------------------------------*/
/* Get Disk Status                                                       */
/*-----------------------------------------------------------------------*/

DSTATUS disk_status (
   BYTE drv           /* Drive number (0) */
)
{
   DSTATUS s = Stat;


   if (drv || !INS) {
      s = STA_NODISK | STA_NOINIT;
   } else {
```

```
        s &= ~STA_NODISK;
        if (WP)
            s |= STA_PROTECT;
        else
            s &= ~STA_PROTECT;
    }
    Stat = s;

    return s;
}



/*-----------------------------------------------------------------------*/
/* Initialize Disk Drive                                                 */
/*-----------------------------------------------------------------------*/

DSTATUS disk_initialize (
        BYTE drv            /* Physical drive nmuber (0) */
)
{
/* TI: Inserted pragma to supress IAR compiler warning incicating 'cmd'
 * is not used. If removed however the compile fails */
#ifdef __IAR_SYSTEMS_ICC__
#pragma diag_suppress=Pe550
#endif
#ifdef __TI_COMPILER_VERSION__
#pragma diag_suppress 552
#endif
        BYTE n, ty, cmd, buf[4];
#ifdef __IAR_SYSTEMS_ICC__
#pragma diag_default=Pe550
#endif
//#ifdef __TI_COMPILER_VERSION__
//#pragma diag_default 552
//#endif
    UINT tmr;
```

```
    DSTATUS s;


    INIT_PORT();                /* Initialize control port */

    DLY_US(100);

    s = disk_status(drv);       /* Check if card is in the socket */
    if (s & STA_NODISK) return s;

    CS_H();
    for (n = 10; n; n--) rcvr_mmc(buf, 1);    /* 80 dummy clocks */

    ty = 0;
    if (send_cmd(CMD0, 0) == 1) {           /* Enter Idle state */
      if (send_cmd(CMD8, 0x1AA) == 1) {    /* SDv2? */
        rcvr_mmc(buf, 4);                          /* Get trailing return value of R7 resp */
        if (buf[2] == 0x01 && buf[3] == 0xAA) {       /* The card can work at vdd range of 2.7-3.6V */
          for (tmr = 1000; tmr; tmr--) {         /* Wait for leaving idle state (ACMD41 with HCS bit) */
            if (send_cmd(ACMD41, 1UL << 30) == 0) break;
            DLY_US(1000);
          }
          if (tmr && send_cmd(CMD58, 0) == 0) {    /* Check CCS bit in the OCR */
            rcvr_mmc(buf, 4);
            ty = (buf[0] & 0x40) ? CT_SD2 | CT_BLOCK : CT_SD2;    /* SDv2 */
          }
        }
      } else {                            /* SDv1 or MMCv3 */
        if (send_cmd(ACMD41, 0) <= 1)     {
          ty = CT_SD1; cmd = ACMD41;    /* SDv1 */
        } else {
          ty = CT_MMC; cmd = CMD1;    /* MMCv3 */
        }
        for (tmr = 1000; tmr; tmr--) {           /* Wait for leaving idle state */
```

```
            if (send_cmd(ACMD41, 0) == 0) break;
            DLY_US(1000);
        }
        if (!tmr || send_cmd(CMD16, 512) != 0)    /* Set R/W block length to 512 */
            ty = 0;
    }
  }
  CardType = ty;
  deselect();

  if (ty) {      /* Initialization succeded */
    FAST_MODE();
    s &= ~STA_NOINIT;
  }
  else {       /* Initialization failed */
    s |= STA_NOINIT;
  }
  Stat = s;


  return s;
}



/*-----------------------------------------------------------------------*/
/* Read Sector(s)                                                        */
/*-----------------------------------------------------------------------*/

DRESULT disk_read (
  BYTE drv,           /* Physical drive nmuber (0) */
  BYTE *buff,           /* Pointer to the data buffer to store read data */
  DWORD sector,       /* Start sector number (LBA) */
  BYTE count          /* Sector count (1..128) */
)
{
  DSTATUS s;
```

```
    s = disk_status(drv);
    if (s & STA_NOINIT) return RES_NOTRDY;
    if (!count) return RES_PARERR;
    if (!(CardType & CT_BLOCK)) sector *= 512;    /* Convert LBA to byte address
if needed */

    if (count == 1) {    /* Single block read */
        if ((send_cmd(CMD17, sector) == 0)    /* READ_SINGLE_BLOCK */
            && rcvr_datablock(buff, 512))
            count = 0;
    }
    else {              /* Multiple block read */
        if (send_cmd(CMD18, sector) == 0) {    /* READ_MULTIPLE_BLOCK */
            do {
                if (!rcvr_datablock(buff, 512)) break;
                buff += 512;
            } while (--count);
            send_cmd(CMD12, 0);              /* STOP_TRANSMISSION */
        }
    }
    deselect();

    return count ? RES_ERROR : RES_OK;
}



/*-----------------------------------------------------------------------*/
/* Write Sector(s)                                                       */
/*-----------------------------------------------------------------------*/

DRESULT disk_write (
    BYTE drv,           /* Physical drive nmuber (0) */
    const BYTE *buff,    /* Pointer to the data to be written */
    DWORD sector,        /* Start sector number (LBA) */
```

```
    BYTE count          /* Sector count (1..128) */
)
{
    DSTATUS s;


    s = disk_status(drv);
    if (s & STA_NOINIT) return RES_NOTRDY;
    if (s & STA_PROTECT) return RES_WRPRT;
    if (!count) return RES_PARERR;
    if (!(CardType & CT_BLOCK)) sector *= 512;    /* Convert LBA to byte address
if needed */

    if (count == 1) {    /* Single block write */
        if ((send_cmd(CMD24, sector) == 0)    /* WRITE_BLOCK */
            && xmit_datablock(buff, 0xFE))
            count = 0;
    }
    else {              /* Multiple block write */
        if (CardType & CT_SDC) send_cmd(ACMD23, count);
        if (send_cmd(CMD25, sector) == 0) {    /* WRITE_MULTIPLE_BLOCK */
            do {
                if (!xmit_datablock(buff, 0xFC)) break;
                buff += 512;
            } while (--count);
            if (!xmit_datablock(0, 0xFD))    /* STOP_TRAN token */
                count = 1;
        }
    }
    deselect();

    return count ? RES_ERROR : RES_OK;
}



/*-----------------------------------------------------------------------*/
```

```
/* Miscellaneous Functions                                    */
/*-------------------------------------------------------------------------*/

DRESULT disk_ioctl (
   BYTE drv,       /* Physical drive nmuber (0) */
   BYTE ctrl,      /* Control code */
   void *buff      /* Buffer to send/receive control data */
)
{
   DRESULT res;
   BYTE n, csd[16];
   WORD cs;


   if (disk_status(drv) & STA_NOINIT)              /* Check if card is in the socket
*/
      return RES_NOTRDY;

   res = RES_ERROR;
   switch (ctrl) {
      case CTRL_SYNC :       /* Make sure that no pending write process */
         if (select()) {
            deselect();
            res = RES_OK;
         }
         break;

      case GET_SECTOR_COUNT :      /* Get number of sectors on the disk
(DWORD) */
         if ((send_cmd(CMD9, 0) == 0) && rcvr_datablock(csd, 16)) {
            if ((csd[0] >> 6) == 1) {    /* SDC ver 2.00 */
               cs= csd[9] + ((WORD)csd[8] << 8) + 1;
               *(DWORD*)buff = (DWORD)cs << 10;
            } else {                  /* SDC ver 1.XX or MMC */
               n = (csd[5] & 15) + ((csd[10] & 128) >> 7) + ((csd[9] & 3) << 1) + 2;
               cs = (csd[8] >> 6) + ((WORD)csd[7] << 2) + ((WORD)(csd[6] & 3) <<
10) + 1;
```

```
                *(DWORD*)buff = (DWORD)cs << (n - 9);
            }
            res = RES_OK;
        }
        break;

    case GET_BLOCK_SIZE :   /* Get erase block size in unit of sector (DWORD)
*/
        *(DWORD*)buff = 128;
        res = RES_OK;
        break;

    default:
        res = RES_PARERR;
  }

  deselect();

  return res;
}



// KLQ

uint8_t validateCSD(void)
{
 BYTE csd0[16], csd1[16], i;
 WORD sum=0;

 // Pull the CSD -- twice.  If the response codes are invalid, then we know the card
isn't there or initialized.
 if ((send_cmd(CMD9, 0) == 0) && rcvr_datablock(csd0, 16))
  if ((send_cmd(CMD9, 0) == 0) && rcvr_datablock(csd1, 16))
  {
    // The response codes were good -- but maybe the SPI input was just floating
low.  Let's evaluate the CSD data.
```

```
    // First, look for all zero or all ones.  If the SPI input is floating, these are the most likely outcomes.
    for(i=0;i<=15;i++)
      sum += csd0[i];
    if(!((sum == 0) || (sum == 4096)))
    {
      // The response was a mix of 0's and 1's.  Floating inputs could still do that -- but it's unlikely they'd
      // produce the same pattern twice.  Compare to ensure the two are identical.
      i = 0;
      while(i<=15)
      {
        if(csd0[i] != csd1[i])
          break;
        i++;
      }
      if(i>15)
        return 1;
    }
  }
 return 0;
}


// Attempt to detect the card by commanding it to return its CSD register and evaluating it.  Returns the
// result, and also updates FatFs's internal INS variable.
// The proper way to detect a card is by sensing its presence on the DAT3 signal.  The EXP board doesn't
// contain the necessary h/w, so this s/w method works instead.
uint8_t detectCard(void)
{
 // Check for a valid CSD response
 if(validateCSD())
 {
   disk_status(0); // Update the INS variable
   return 1;       // Card is present
```

```
  }

  // We didn't get a valid response.  So we now know the status is one of two things:
  // a) The card isn't there at all;
  // b) or, it was just inserted recently, and needs to be initialized

  INS = 0x01;          // Trick disk_initialize into thinking it's inserted...
  disk_initialize(0);  // Attempt to initialize it

  INS = validateCSD(); // Try again
  disk_status(0);      // Update the INS variable

  return INS;          // 1 = card is present; 0 = not present
}
```

Integer.h

```
/*-----------------------------------------*/
/* Integer type definitions for FatFs module */
/*-----------------------------------------*/

#ifndef _INTEGER
#define _INTEGER

#ifdef _WIN32    /* FatFs development platform */

#include <windows.h>
#include <tchar.h>

#else           /* Embedded platform */

// Surpress warning for multiple defs of the same type.
// This is done so the FatFs can be a stanalone modular entity.
#ifdef __IAR_SYSTEMS_ICC__
#pragma diag_suppress=Pe301
#endif
#ifdef __TI_COMPILER_VERSION__
```

```
#pragma diag_suppress 303
#endif

/* These types must be 16-bit, 32-bit or larger integer */
typedef int            INT;
typedef unsigned int   UINT;

/* These types must be 8-bit integer */
typedef char           CHAR;
typedef unsigned char  UCHAR;
typedef unsigned char  BYTE;

/* These types must be 16-bit integer */
typedef short          SHORT;
typedef unsigned short USHORT;
typedef unsigned int   WORD;
typedef unsigned short WCHAR;

/* These types must be 32-bit integer */
typedef long           LONG;
typedef unsigned long  ULONG;
typedef unsigned long  DWORD;

#endif

#endif
```

HAL_Wheel.h

```
/*****************************************************************************
 *
 *  HAL_Wheel.h
 *
 *  Copyright (C) 2010 Texas Instruments Incorporated - http://www.ti.com/
 *
 *  Redistribution and use in source and binary forms, with or without
```

```
 *  modification, are permitted provided that the following conditions
 *  are met:
 *
 *    Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *    Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the
 *    distribution.
 *
 *    Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 *  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
 *
********************************************************************************/

#ifndef HAL_WHEEL_H
#define HAL_WHEEL_H

#include <stdint.h>

extern void Wheel_init(void);
extern uint8_t Wheel_getPosition(void);
extern uint16_t Wheel_getValue(void);
extern void Wheel_disable(void);
extern void Wheel_enable(void);

#endif /* HAL_WHEEL_H */
```

HAL_Wheel.c

```
/*******************************************************************************
*************
 *
 *  HAL_Wheel.c - Driver for the scroll wheel
 *
 *  Copyright (C) 2010 Texas Instruments Incorporated - http://www.ti.com/
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions
 *  are met:
 *
 *    Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *    Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the
```

```
 *    distribution.
 *
 *    Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 *        SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 *   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *

**************************************************************************
************/

/**************************************************************************
*********//**
 * @file       HAL_Wheel.c
 * @addtogroup HAL_Wheel
```

```
 * @{

**********************************************************************
**********/
#include "msp430.h"
#include "HAL_Wheel.h"

#define WHEEL_PORT_DIR P8DIR
#define WHEEL_PORT_OUT P8OUT
#define WHEEL_ENABLE  BIT0
#define ADC_PORT_SEL  P6SEL
#define ADC_INPUT_A5  BIT5

uint16_t positionData;
uint16_t positionDataOld;

/**********************************************************************
*********//**
 * @brief   Set up the wheel
 * @param   None
 * @return  None

**********************************************************************
**********/

void Wheel_init(void)
{
   WHEEL_PORT_DIR |= WHEEL_ENABLE;
   WHEEL_PORT_OUT |= WHEEL_ENABLE;               // Enable wheel

   ADC12CTL0 = ADC12SHT02 + ADC12ON;             // Sampling time, ADC12
on
   ADC12CTL1 = ADC12SHP;                        // Use sampling timer
   ADC12MCTL0 = ADC12INCH_5;                     // Use A5 (wheel) as input
   ADC12CTL0 |= ADC12ENC;                       // Enable conversions
   ADC_PORT_SEL |= ADC_INPUT_A5;                 // P6.5 ADC option select
(A5)
```

```
}

/**********************************************************************
*********//**
 * @brief   Determine the wheel's position
 * @param   None
 * @return  Wheel position (0~7)

**********************************************************************
**********/

uint8_t Wheel_getPosition(void)
{
   uint8_t position = 0;

   Wheel_getValue();
   //determine which position the wheel is in
   if (positionData > 0x0806)
      position = 7 - (positionData - 0x0806) / 260;  //scale the data for 8 different
positions
   else
      position = positionData / 260;

   return position;
}

/**********************************************************************
*********//**
 * @brief   Determine the raw voltage value across the potentiometer
 * @param   None
 * @return  Value

**********************************************************************
**********/

uint16_t Wheel_getValue(void)
{
```

```
   //measure ADC value
   ADC12IE = 0x01;                          // Enable interrupt
   ADC12CTL0 |= ADC12SC;                        // Start sampling/conversion
   __bis_SR_register(LPM0_bits + GIE);           // LPM0, ADC12_ISR will force
exit
   ADC12IE = 0x00;                          // Disable interrupt

   //add hysteresis on wheel to remove fluctuations
   if (positionData > positionDataOld)
      if ((positionData - positionDataOld) > 10)
         positionDataOld = positionData;          //use new data if change is beyond
                                    // fluctuation threshold
      else
         positionData = positionDataOld;          //use old data if change is not beyond
                                    // fluctuation threshold
   else
   if ((positionDataOld - positionData) > 10)
      positionDataOld = positionData;              //use new data if change is beyond
                                    // fluctuation threshold
   else
      positionData = positionDataOld;              //use old data if change is not beyond
                                    // fluctuation threshold

   return positionData;
}

/*****************************************************************************
*********//**
 * @brief   Disable wheel
 * @param   None
 * @return  none

*****************************************************************************
************/

void Wheel_disable(void)
{
```

```
   WHEEL_PORT_OUT &= ~WHEEL_ENABLE;                //disable wheel
   ADC12CTL0 &= ~ADC12ENC;                          // Disable conversions
   ADC12CTL0 &= ~ADC12ON;                          // ADC12 off
}

/*************************************************************************
*********//**
 * @brief   Enable wheel
 * @param   None
 * @return  none

*************************************************************************
************/

void Wheel_enable(void)
{
   WHEEL_PORT_OUT |= WHEEL_ENABLE;                 //disable wheel
   ADC12CTL0 |= ADC12ON;                          // ADC12 on
   ADC12CTL0 |= ADC12ENC;                          // Enable conversions
}

/*************************************************************************
*********//**
 * @brief Handles ADC interrupts.
 *
 *          Stores result of single ADC conversion for reading position of the scroll
wheel.
 * @param  none
 * @return none

*************************************************************************
************/

#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void)
{
   switch (__even_in_range(ADC12IV, ADC12IV_ADC12IFG15))
```

```
{
  // Vector  ADC12IV_NONE:  No interrupt
  case  ADC12IV_NONE:
    break;

  // Vector  ADC12IV_ADC12OVIFG:  ADC overflow
  case  ADC12IV_ADC12OVIFG:
    break;

  // Vector  ADC12IV_ADC12TOVIFG:  ADC timing overflow
  case  ADC12IV_ADC12TOVIFG:
    break;

  // Vector  ADC12IV_ADC12IFG0: ADC12IFG0:
  case  ADC12IV_ADC12IFG0:
    positionData = ADC12MEM0;              // ADC12MEM = A0 > 0.5AVcc?
    __bic_SR_register_on_exit(LPM0_bits);     // Exit active CPU
    break;

  // Vector  ADC12IV_ADC12IFG1:  ADC12IFG1
  case  ADC12IV_ADC12IFG1:
    break;

  // Vector ADC12IV_ADC12IFG2:  ADC12IFG2
  case ADC12IV_ADC12IFG2:
    break;

  // Vector ADC12IV_ADC12IFG3:  ADC12IFG3
  case ADC12IV_ADC12IFG3:
    break;

  // Vector ADC12IV_ADC12IFG4:  ADC12IFG4
  case ADC12IV_ADC12IFG4:
    break;

  // Vector ADC12IV_ADC12IFG5:  ADC12IFG5
  case ADC12IV_ADC12IFG5:
```

```
    break;

// Vector ADC12IV_ADC12IFG6:  ADC12IFG6
case ADC12IV_ADC12IFG6:
    break;

// Vector ADC12IV_ADC12IFG7:  ADC12IFG7
case ADC12IV_ADC12IFG7:
    break;

// Vector ADC12IV_ADC12IFG8:  ADC12IFG8
case ADC12IV_ADC12IFG8:
    break;

// Vector ADC12IV_ADC12IFG9:  ADC12IFG9
case ADC12IV_ADC12IFG9:
    break;

// Vector ADC12IV_ADC12IFG10:  ADC12IFG10
case ADC12IV_ADC12IFG10:
    break;

// Vector ADC12IV_ADC12IFG11:  ADC12IFG11
case ADC12IV_ADC12IFG11:
    break;

// Vector ADC12IV_ADC12IFG12:  ADC12IFG12
case ADC12IV_ADC12IFG12:
    break;

// Vector ADC12IV_ADC12IFG13:  ADC12IFG13
case ADC12IV_ADC12IFG13:
    break;

// Vector ADC12IV_ADC12IFG14:  ADC12IFG14
case ADC12IV_ADC12IFG14:
    break;
```

```
      // Vector ADC12IV_ADC12IFG15:  ADC12IFG15
      case ADC12IV_ADC12IFG15:
         break;

      default:
         break;
   }
}

/*************************************************************************
*********//**
 * @}

*************************************************************************
************/
```

Ffconf.h

```
/*---------------------------------------------------------------------------/
/  FatFs - FAT file system module configuration file  R0.08b (C)ChaN, 2011
/---------------------------------------------------------------------------/
/
/ CAUTION! Do not forget to make clean the project after any changes to
/ the configuration options.
/
/----------------------------------------------------------------------------*/
#ifndef _FFCONF
#define _FFCONF 8237 /* Revision ID */


/*---------------------------------------------------------------------------/
/ Function and Buffer Configurations
/----------------------------------------------------------------------------*/

#define       _FS_TINY          1       /* 0:Normal or 1:Tiny */
/* When _FS_TINY is set to 1, FatFs uses the sector buffer in the file system
```

```
/  object instead of the sector buffer in the individual file object for file
/  data transfer. This reduces memory consumption 512 bytes each file object. */


#define _FS_READONLY    0    /* 0:Read/Write or 1:Read only */
/* Setting _FS_READONLY to 1 defines read only configuration. This removes
/  writing functions, f_write, f_sync, f_unlink, f_mkdir, f_chmod, f_rename,
/  f_truncate and useless f_getfree. */


#define _FS_MINIMIZE0    /* 0 to 3 */
/* The _FS_MINIMIZE option defines minimization level to remove some
functions.
/
/   0: Full function.
/   1: f_stat, f_getfree, f_unlink, f_mkdir, f_chmod, f_truncate and f_rename
/      are removed.
/   2: f_opendir and f_readdir are removed in addition to 1.
/   3: f_lseek is removed in addition to 2. */


#define    _USE_STRFUNC 0    /* 0:Disable or 1/2:Enable */
/* To enable string functions, set _USE_STRFUNC to 1 or 2. */


#define    _USE_MKFS        1    /* 0:Disable or 1:Enable */
/* To enable f_mkfs function, set _USE_MKFS to 1 and set _FS_READONLY to 0
*/


#define    _USE_FORWARD    0    /* 0:Disable or 1:Enable */
/* To enable f_forward function, set _USE_FORWARD to 1 and set _FS_TINY to
1. */


#define    _USE_FASTSEEK    0    /* 0:Disable or 1:Enable */
/* To enable fast seek feature, set _USE_FASTSEEK to 1. */
```

```
/*---------------------------------------------------------------------------/
/ Locale and Namespace Configurations
/---------------------------------------------------------------------------*/

#define _CODE_PAGE  932
/* The _CODE_PAGE specifies the OEM code page to be used on the target system.
/  Incorrect setting of the code page can cause a file open failure.
/
/   932  - Japanese Shift-JIS (DBCS, OEM, Windows)
/   936  - Simplified Chinese GBK (DBCS, OEM, Windows)
/   949  - Korean (DBCS, OEM, Windows)
/   950  - Traditional Chinese Big5 (DBCS, OEM, Windows)
/   1250 - Central Europe (Windows)
/   1251 - Cyrillic (Windows)
/   1252 - Latin 1 (Windows)
/   1253 - Greek (Windows)
/   1254 - Turkish (Windows)
/   1255 - Hebrew (Windows)
/   1256 - Arabic (Windows)
/   1257 - Baltic (Windows)
/   1258 - Vietnam (OEM, Windows)
/   437  - U.S. (OEM)
/   720  - Arabic (OEM)
/   737  - Greek (OEM)
/   775  - Baltic (OEM)
/   850  - Multilingual Latin 1 (OEM)
/   858  - Multilingual Latin 1 + Euro (OEM)
/   852  - Latin 2 (OEM)
/   855  - Cyrillic (OEM)
/   866  - Russian (OEM)
/   857  - Turkish (OEM)
/   862  - Hebrew (OEM)
/   874  - Thai (OEM, Windows)
/        1    - ASCII only (Valid for non LFN cfg.)
```

```
*/


#define	_USE_LFN 0		/* 0 to 3 */
#define	_MAX_LFN	255		/* Maximum LFN length to handle (12 to 255) */
/* The _USE_LFN option switches the LFN support.
/
/   0: Disable LFN feature. _MAX_LFN and _LFN_UNICODE have no effect.
/   1: Enable LFN with static working buffer on the BSS. Always NOT reentrant.
/   2: Enable LFN with dynamic working buffer on the STACK.
/   3: Enable LFN with dynamic working buffer on the HEAP.
/
/  The LFN working buffer occupies (_MAX_LFN + 1) * 2 bytes. To enable LFN,
/  Unicode handling functions ff_convert() and ff_wtoupper() must be added
/  to the project. When enable to use heap, memory control functions
/  ff_memalloc() and ff_memfree() must be added to the project. */


#define	_LFN_UNICODE 0	/* 0:ANSI/OEM or 1:Unicode */
/* To switch the character code set on FatFs API to Unicode,
/  enable LFN feature and set _LFN_UNICODE to 1. */


#define _FS_RPATH		0	/* 0 to 2 */
/* The _FS_RPATH option configures relative path feature.
/
/   0: Disable relative path feature and remove related functions.
/   1: Enable relative path. f_chdrive() and f_chdir() are available.
/   2: f_getcwd() is available in addition to 1.
/
/  Note that output of the f_readdir fnction is affected by this option. */


/*---------------------------------------------------------------------------/
/ Physical Drive Configurations
```

```
/---------------------------------------------------------------------------*/

#define _VOLUMES      1
/* Number of volumes (logical drives) to be used. */


#define       _MAX_SS          512          /* 512, 1024, 2048 or 4096 */
/* Maximum sector size to be handled.
/  Always set 512 for memory card and hard disk but a larger value may be
/  required for on-board flash memory, floppy disk and optical disk.
/  When _MAX_SS is larger than 512, it configures FatFs to variable sector size
/   and GET_SECTOR_SIZE command must be implememted to the disk_ioctl function. */


#define       _MULTI_PARTITION 0      /* 0:Single partition or 1:Multiple partition */
/* When set to 0, each volume is bound to the same physical drive number and
/ it can mount only first primaly partition. When it is set to 1, each volume
/ is tied to the partitions listed in VolToPart[]. */


#define       _USE_ERASE      0       /* 0:Disable or 1:Enable */
/* To enable sector erase feature, set _USE_ERASE to 1. CTRL_ERASE_SECTOR command
/  should be added to the disk_ioctl functio. */



/*---------------------------------------------------------------------------/
/ System Configurations
/---------------------------------------------------------------------------*/

#define _WORD_ACCESS      0       /* 0 or 1 */
/* Set 0 first and it is always compatible with all platforms. The _WORD_ACCESS
/  option defines which access method is used to the word data on the FAT volume.
/
```

```
/   0: Byte-by-byte access.
/   1: Word access. Do not choose this unless following condition is met.
/
/  When the byte order on the memory is big-endian or address miss-aligned word
/  access results incorrect behavior, the _WORD_ACCESS must be set to 0.
/  If it is not the case, the value can also be set to 1 to improve the
/  performance and code size. */


/* A header file that defines sync object types on the O/S, such as
/  windows.h, ucos_ii.h and semphr.h, must be included prior to ff.h. */

#define _FS_REENTRANT     0              /* 0:Disable or 1:Enable */
#define _FS_TIMEOUT       1000  /* Timeout period in unit of time ticks */
#define       _SYNC_t                 HANDLE   /* O/S dependent type of sync
object. e.g. HANDLE, OS_EVENT*, ID and etc.. */

/* The _FS_REENTRANT option switches the reentrancy (thread safe) of the FatFs
module.
/
/   0: Disable reentrancy. _SYNC_t and _FS_TIMEOUT have no effect.
/   1: Enable reentrancy. Also user provided synchronization handlers,
/      ff_req_grant, ff_rel_grant, ff_del_syncobj and ff_cre_syncobj
/      function must be added to the project. */


#define       _FS_SHARE       0       /* 0:Disable or >=1:Enable */
/* To enable file shareing feature, set _FS_SHARE to 1 or greater. The value
   defines how many files can be opened simultaneously. */


#endif /* _FFCONFIG */
```

ff.h

```
/*-----------------------------------------------------------------------/
/  FatFs - FAT file system module include file  R0.08b    (C)ChaN, 2011
```

```
/----------------------------------------------------------------------------/
/ FatFs module is a generic FAT file system module for small embedded systems.
/ This is a free software that opened for education, research and commercial
/ developments under license policy of following trems.
/
/  Copyright (C) 2011, ChaN, all right reserved.
/
/ * The FatFs module is a free software and there is NO WARRANTY.
/ * No restriction on use. You can use, modify and redistribute it for
/   personal, non-profit or commercial product UNDER YOUR RESPONSIBILITY.
/ * Redistributions of source code must retain the above copyright notice.
/
/----------------------------------------------------------------------------*/

#ifndef _FATFS
#define _FATFS    8237  /* Revision ID */

#ifdef __cplusplus
extern "C" {
#endif

#include "integer.h"        /* Basic integer types */
#include "ffconf.h"                 /* FatFs configuration options */

#if _FATFS != _FFCONF
#error Wrong configuration file (ffconf.h).
#endif



/* Definitions of volume management */

#if _MULTI_PARTITION                /* Multiple partition configuration */
#define LD2PD(vol) (VolToPart[vol].pd)    /* Get physical drive# */
#define LD2PT(vol) (VolToPart[vol].pt)     /* Get partition# */
typedef struct {
        BYTE pd;    /* Physical drive# */
```

```
	BYTE pt;	/* Partition # (0-3) */
} PARTITION;
extern const PARTITION VolToPart[];	/* Volume - Physical location resolution table */

#else					/* Single partition configuration */
#define LD2PD(vol) (vol)	/* Logical drive# is bound to the same physical drive# */
#define LD2PT(vol) 0		/* Always mounts the 1st partition */

#endif



/* Type of path name strings on FatFs API */

#if _LFN_UNICODE			/* Unicode string */
#if !_USE_LFN
#error _LFN_UNICODE must be 0 in non-LFN cfg.
#endif
#ifndef _INC_TCHAR
typedef WCHAR TCHAR;
#define _T(x) L ## x
#define _TEXT(x) L ## x
#endif

#else					/* ANSI/OEM string */
#ifndef _INC_TCHAR
typedef char TCHAR;
#define _T(x) x
#define _TEXT(x) x
#endif

#endif
```

```
/* File system object structure (FATFS) */

typedef struct {
	BYTE	fs_type;		/* FAT sub-type (0:Not mounted) */
	BYTE	drv;			/* Physical drive number */
	BYTE	csize;			/* Sectors per cluster (1,2,4...128) */
	BYTE	n_fats;			/* Number of FAT copies (1,2) */
	BYTE	wflag;			/* win[] dirty flag (1:must be written back) */
	BYTE	fsi_flag;		/* fsinfo dirty flag (1:must be written back) */
	WORD	id;			/* File system mount ID */
	WORD	n_rootdir;		/* Number of root directory entries (FAT12/16) */
#if _MAX_SS != 512
	WORD	ssize;			/* Bytes per sector (512,1024,2048,4096) */
#endif
#if _FS_REENTRANT
	_SYNC_t	sobj;			/* Identifier of sync object */
#endif
#if !_FS_READONLY
	DWORD	last_clust;		/* Last allocated cluster */
	DWORD	free_clust;		/* Number of free clusters */
	DWORD	fsi_sector;		/* fsinfo sector (FAT32) */
#endif
#if _FS_RPATH
	DWORD	cdir;			/* Current directory start cluster (0:root) */
#endif
	DWORD	n_fatent;		/* Number of FAT entries (= number of clusters + 2) */
	DWORD	fsize;			/* Sectors per FAT */
	DWORD	fatbase;		/* FAT start sector */
	DWORD	dirbase;		/* Root directory start sector (FAT32:Cluster#) */
	DWORD	database;		/* Data start sector */
	DWORD	winsect;		/* Current sector appearing in the win[] */
	BYTE	win[_MAX_SS];	/* Disk access window for Directory, FAT (and Data on tiny cfg) */
```

```
} FATFS;



/* File object structure (FIL) */

typedef struct {
	FATFS*	fs;			/* Pointer to the owner file system object */
	WORD	id;			/* Owner file system mount ID */
	BYTE	flag;		/* File status flags */
	BYTE	pad1;
	DWORD	fptr;		/* File read/write pointer (0 on file open) */
	DWORD	fsize;		/* File size */
	DWORD	sclust;			/* File start cluster (0 when fsize==0) */
	DWORD	clust;		/* Current cluster */
	DWORD	dsect;		/* Current data sector */
#if !_FS_READONLY
	DWORD	dir_sect;	/* Sector containing the directory entry */
	BYTE*	dir_ptr;	/* Ponter to the directory entry in the window */
#endif
#if _USE_FASTSEEK
	DWORD*	cltbl;		/* Pointer to the cluster link map table (null on file open) */
#endif
#if _FS_SHARE
	UINT lockid;		/* File lock ID (index of file semaphore table) */
#endif
#if !_FS_TINY
	BYTE	buf[_MAX_SS];	/* File data read/write buffer */
#endif
} FIL;
```

```
/* Directory object structure (DIRS) */

typedef struct {
	FATFS*	fs;			/* Pointer to the owner file system object */
	WORD	id;			/* Owner file system mount ID */
	WORD	index;		/* Current read/write index number */
	DWORD	sclust;			/* Table start cluster (0:Root dir) */
	DWORD	clust;		/* Current cluster */
	DWORD	sect;		/* Current sector */
	BYTE*	dir;		/* Pointer to the current SFN entry in the win[] */
	BYTE*	fn;			/* Pointer to the SFN (in/out) {file[8],ext[3],status[1]} */
#if _USE_LFN
	WCHAR*	lfn;		/* Pointer to the LFN working buffer */
	WORD	lfn_idx;	/* Last matched LFN index number (0xFFFF:No LFN) */
#endif
} DIRS;



/* File status structure (FILINFO) */

typedef struct {
	DWORD	fsize;		/* File size */
	WORD	fdate;		/* Last modified date */
	WORD	ftime;		/* Last modified time */
	BYTE	fattrib;	/* Attribute */
	TCHAR	fname[13];	/* Short file name (8.3 format) */
#if _USE_LFN
	TCHAR*	lfname;			/* Pointer to the LFN buffer */
	UINT	lfsize;		/* Size of LFN buffer in TCHAR */
#endif
} FILINFO;
```

```
/* File function return code (FRESULT) */

typedef enum {
	FR_OK = 0,				/* (0) Succeeded */
	FR_DISK_ERR,			/* (1) A hard error occured in the low level disk I/O layer */
	FR_INT_ERR,				/* (2) Assertion failed */
	FR_NOT_READY,			/* (3) The physical drive cannot work */
	FR_NO_FILE,				/* (4) Could not find the file */
	FR_NO_PATH,				/* (5) Could not find the path */
	FR_INVALID_NAME,		/* (6) The path name format is invalid */
	FR_DENIED,				/* (7) Acces denied due to prohibited access or directory full */
	FR_EXIST,				/* (8) Acces denied due to prohibited access */
	FR_INVALID_OBJECT,		/* (9) The file/directory object is invalid */
	FR_WRITE_PROTECTED,		/* (10) The physical drive is write protected */
	FR_INVALID_DRIVE,		/* (11) The logical drive number is invalid */
	FR_NOT_ENABLED,			/* (12) The volume has no work area */
	FR_NO_FILESYSTEM,		/* (13) There is no valid FAT volume on the physical drive */
	FR_MKFS_ABORTED,		/* (14) The f_mkfs() aborted due to any parameter error */
	FR_TIMEOUT,				/* (15) Could not get a grant to access the volume within defined period */
	FR_LOCKED,				/* (16) The operation is rejected according to the file shareing policy */
	FR_NOT_ENOUGH_CORE,		/* (17) LFN working buffer could not be allocated */
	FR_TOO_MANY_OPEN_FILES	/* (18) Number of open files > _FS_SHARE */
```

```
} FRESULT;



/*--------------------------------------------------------------*/
/* FatFs module application interface                           */

FRESULT f_mount (BYTE, FATFS*);                                  /*
Mount/Unmount a logical drive */
FRESULT f_open (FIL*, const TCHAR*, BYTE);                       /* Open or
create a file */
FRESULT f_read (FIL*, void*, UINT, UINT*);                       /* Read data from a
file */
FRESULT f_lseek (FIL*, DWORD);                                   /* Move file
pointer of a file object */
FRESULT f_close (FIL*);                                          /*
Close an open file object */
FRESULT f_opendir (DIRS*, const TCHAR*);                         /* Open an
existing directory */
FRESULT f_readdir (DIRS*, FILINFO*);                             /* Read a
directory item */
FRESULT f_stat (const TCHAR*, FILINFO*);                         /* Get file status */
FRESULT f_write (FIL*, const void*, UINT, UINT*);   /* Write data to a file */
FRESULT f_getfree (const TCHAR*, DWORD*, FATFS**); /* Get number of
free clusters on the drive */
FRESULT f_truncate (FIL*);                                       /* Truncate
file */
FRESULT f_sync (FIL*);                                           /*
Flush cached data of a writing file */
FRESULT f_unlink (const TCHAR*);                                 /* Delete an
existing file or directory */
FRESULT   f_mkdir (const TCHAR*);                                /*
Create a new directory */
FRESULT f_chmod (const TCHAR*, BYTE, BYTE);                      /* Change
attriburte of the file/dir */
FRESULT f_utime (const TCHAR*, const FILINFO*);                  /* Change
timestamp of the file/dir */
```

```
FRESULT f_rename (const TCHAR*, const TCHAR*);          /* Rename/Move a
file or directory */
FRESULT f_forward (FIL*, UINT(*)(const BYTE*,UINT), UINT, UINT*);
        /* Forward data to the stream */
FRESULT f_mkfs (BYTE, BYTE, UINT);                      /* Create a
file system on the drive */
FRESULT f_chdrive (BYTE);                               /* Change
current drive */
FRESULT f_chdir (const TCHAR*);                         /* Change
current directory */
FRESULT f_getcwd (TCHAR*, UINT);                        /* Get current
directory */
int f_putc (TCHAR, FIL*);                               /* Put a
character to the file */
int f_puts (const TCHAR*, FIL*);                        /* Put a string to the
file */
int f_printf (FIL*, const TCHAR*, ...);                 /* Put a formatted
string to the file */
TCHAR* f_gets (TCHAR*, int, FIL*);                      /* Get a string
from the file */

#ifndef EOF
#define EOF (-1)
#endif

#define f_eof(fp) (((fp)->fptr == (fp)->fsize) ? 1 : 0)
#define f_error(fp) (((fp)->flag & FA__ERROR) ? 1 : 0)
#define f_tell(fp) ((fp)->fptr)
#define f_size(fp) ((fp)->fsize)




/*--------------------------------------------------------------*/
/* Additional user defined functions                            */

/* RTC function */
```

```
#if !_FS_READONLY
DWORD get_fattime (void);
#endif

/* Unicode support functions */
#if _USE_LFN                                /* Unicode - OEM code conversion */
WCHAR ff_convert (WCHAR, UINT);             /* OEM-Unicode bidirectional conversion */
WCHAR ff_wtoupper (WCHAR);                  /* Unicode upper-case conversion */
#if _USE_LFN == 3                           /* Memory functions */
void* ff_memalloc (UINT);                   /* Allocate memory block */
void ff_memfree (void*);                    /* Free memory block */
#endif
#endif

/* Sync functions */
#if _FS_REENTRANT
int ff_cre_syncobj (BYTE, _SYNC_t*);/* Create a sync object */
int ff_req_grant (_SYNC_t);                 /* Lock sync object */
void ff_rel_grant (_SYNC_t);          /* Unlock sync object */
int ff_del_syncobj (_SYNC_t);         /* Delete a sync object */
#endif
```

```
/*--------------------------------------------------------------*/
/* Flags and offset address                                     */


/* File access control and file status flags (FIL.flag) */

#define       FA_READ                 0x01
#define       FA_OPEN_EXISTING  0x00
#define FA__ERROR                     0x80
```

```
#if !_FS_READONLY
#define	FA_WRITE		0x02
#define	FA_CREATE_NEW		0x04
#define	FA_CREATE_ALWAYS	0x08
#define	FA_OPEN_ALWAYS		0x10
#define FA__WRITTEN		0x20
#define FA__DIRTY		0x40
#endif


/* FAT sub type (FATFS.fs_type) */

#define FS_FAT12	1
#define FS_FAT16	2
#define FS_FAT32	3


/* File attribute bits for directory entry */

#define	AM_RDO	0x01	/* Read only */
#define	AM_HID	0x02	/* Hidden */
#define	AM_SYS	0x04	/* System */
#define	AM_VOL	0x08	/* Volume label */
#define AM_LFN	0x0F	/* LFN entry */
#define AM_DIR	0x10	/* Directory */
#define AM_ARC	0x20	/* Archive */
#define AM_MASK	0x3F	/* Mask of defined bits */


/* Fast seek function */
#define CREATE_LINKMAP	0xFFFFFFFF



/*--------------------------------*/
/* Multi-byte word access macros  */
```

```
#if _WORD_ACCESS == 1     /* Enable word access to the FAT structure */
#define       LD_WORD(ptr)            (WORD)(*(WORD*)(BYTE*)(ptr))
#define       LD_DWORD(ptr)           (DWORD)(*(DWORD*)(BYTE*)(ptr))
#define       ST_WORD(ptr,val)        *(WORD*)(BYTE*)(ptr)=(WORD)(val)
#define       ST_DWORD(ptr,val)       *(DWORD*)(BYTE*)(ptr)=(DWORD)(val)
#else                                 /* Use byte-by-byte access to the FAT structure */
#define       LD_WORD(ptr)
        (WORD)(((WORD)*((BYTE*)(ptr)+1)<<8)|(WORD)*(BYTE*)(ptr))
#define       LD_DWORD(ptr)
        (DWORD)(((DWORD)*((BYTE*)(ptr)+3)<<24)|((DWORD)*((BYTE*)(ptr
)+2)<<16)|((WORD)*((BYTE*)(ptr)+1)<<8)|*(BYTE*)(ptr))
#define       ST_WORD(ptr,val)        *(BYTE*)(ptr)=(BYTE)(val);
*((BYTE*)(ptr)+1)=(BYTE)((WORD)(val)>>8)
#define       ST_DWORD(ptr,val)       *(BYTE*)(ptr)=(BYTE)(val);
*((BYTE*)(ptr)+1)=(BYTE)((WORD)(val)>>8);
*((BYTE*)(ptr)+2)=(BYTE)((DWORD)(val)>>16);
*((BYTE*)(ptr)+3)=(BYTE)((DWORD)(val)>>24)
#endif

#ifdef __cplusplus
}
#endif

#endif /* _FATFS */
```

ff.c

```
/*-----------------------------------------------------------------------/
/  FatFs - FAT file system module  R0.08b              (C)ChaN, 2011
/-----------------------------------------------------------------------/
/ FatFs module is a generic FAT file system module for small embedded systems.
/ This is a free software that opened for education, research and commercial
/ developments under license policy of following terms.
/
/  Copyright (C) 2011, ChaN, all right reserved.
/
```

```
/ * The FatFs module is a free software and there is NO WARRANTY.
/ * No restriction on use. You can use, modify and redistribute it for
/  personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY.
/ * Redistributions of source code must retain the above copyright notice.
/
/----------------------------------------------------------------------------/
/ Feb 26,'06 R0.00  Prototype.
/
/ Apr 29,'06 R0.01  First stable version.
/
/ Jun 01,'06 R0.02  Added FAT12 support.
/                   Removed unbuffered mode.
/                   Fixed a problem on small (<32M) partition.
/ Jun 10,'06 R0.02a Added a configuration option (_FS_MINIMUM).
/
/ Sep 22,'06 R0.03  Added f_rename().
/                   Changed option _FS_MINIMUM to _FS_MINIMIZE.
/ Dec 11,'06 R0.03a Improved cluster scan algorithm to write files fast.
/                   Fixed f_mkdir() creates incorrect directory on FAT32.
/
/ Feb 04,'07 R0.04  Supported multiple drive system.
/                   Changed some interfaces for multiple drive system.
/                   Changed f_mountdrv() to f_mount().
/                   Added f_mkfs().
/ Apr 01,'07 R0.04a Supported multiple partitions on a physical drive.
/                   Added a capability of extending file size to f_lseek().
/                   Added minimization level 3.
/                   Fixed an endian sensitive code in f_mkfs().
/ May 05,'07 R0.04b Added a configuration option _USE_NTFLAG.
/                   Added FSInfo support.
/                   Fixed DBCS name can result FR_INVALID_NAME.
/                   Fixed short seek (<= csize) collapses the file object.
/
/ Aug 25,'07 R0.05  Changed arguments of f_read(), f_write() and f_mkfs().
/                   Fixed f_mkfs() on FAT32 creates incorrect FSInfo.
/                   Fixed f_mkdir() on FAT32 creates incorrect directory.
/ Feb 03,'08 R0.05a Added f_truncate() and f_utime().
```

/ Fixed off by one error at FAT sub-type determination.
/ Fixed btr in f_read() can be mistruncated.
/ Fixed cached sector is not flushed when create and close without write.
/
/ Apr 01,'08 R0.06 Added fputc(), fputs(), fprintf() and fgets().
/ Improved performance of f_lseek() on moving to the same or following cluster.
/
/ Apr 01,'09 R0.07 Merged Tiny-FatFs as a configuration option. (_FS_TINY)
/ Added long file name feature.
/ Added multiple code page feature.
/ Added re-entrancy for multitask operation.
/ Added auto cluster size selection to f_mkfs().
/ Added rewind option to f_readdir().
/ Changed result code of critical errors.
/ Renamed string functions to avoid name collision.
/ Apr 14,'09 R0.07a Separated out OS dependent code on reentrant cfg.
/ Added multiple sector size feature.
/ Jun 21,'09 R0.07c Fixed f_unlink() can return FR_OK on error.
/ Fixed wrong cache control in f_lseek().
/ Added relative path feature.
/ Added f_chdir() and f_chdrive().
/ Added proper case conversion to extended char.
/ Nov 03,'09 R0.07e Separated out configuration options from ff.h to ffconf.h.
/ Fixed f_unlink() fails to remove a sub-dir on _FS_RPATH.
/ Fixed name matching error on the 13 char boundary.
/ Added a configuration option, _LFN_UNICODE.
/ Changed f_readdir() to return the SFN with always upper case on non-LFN cfg.
/
/ May 15,'10 R0.08 Added a memory configuration option. (_USE_LFN = 3)
/ Added file lock feature. (_FS_SHARE)
/ Added fast seek feature. (_USE_FASTSEEK)
/ Changed some types on the API, XCHAR->TCHAR.
/ Changed fname member in the FILINFO structure on Unicode cfg.
/ String functions support UTF-8 encoding files on Unicode cfg.
/ Aug 16,'10 R0.08a Added f_getcwd(). (_FS_RPATH = 2)

```
/                Added sector erase feature. (_USE_ERASE)
/                Moved file lock semaphore table from fs object to the bss.
/               Fixed a wrong directory entry is created on non-LFN cfg when the given
name contains ';'.
/                Fixed f_mkfs() creates wrong FAT32 volume.
/ Jan 15,'11 R0.08b Fast seek feature is also applied to f_read() and f_write().
/                f_lseek() reports required table size on creating CLMP.
/                Extended format syntax of f_printf function.
/                Ignores duplicated directory separators in given path names.
/---------------------------------------------------------------------------*/

#include "ff.h"                 /* FatFs configurations and declarations */
#include "diskio.h"             /* Declarations of low level disk I/O functions */


/*--------------------------------------------------------------------------

   Module Private Definitions

---------------------------------------------------------------------------*/

#if _FATFS != 8237
#error Wrong include file (ff.h).
#endif


/* Definitions on sector size */
#if _MAX_SS != 512 && _MAX_SS != 1024 && _MAX_SS != 2048 &&
_MAX_SS != 4096
#error Wrong sector size.
#endif
#if _MAX_SS != 512
#define     SS(fs)((fs)->ssize) /* Multiple sector size */
#else
#define     SS(fs)512U              /* Fixed sector size */
#endif
```

```
/* Reentrancy related */
#if _FS_REENTRANT
#if _USE_LFN == 1
#error Static LFN work area must not be used in re-entrant configuration.
#endif
#define	ENTER_FF(fs)		{ if (!lock_fs(fs)) return FR_TIMEOUT; }
#define	LEAVE_FF(fs, res)	{ unlock_fs(fs, res); return res; }
#else
#define	ENTER_FF(fs)
#define LEAVE_FF(fs, res)	return res
#endif

#define	ABORT(fs, res)		{ fp->flag |= FA__ERROR; LEAVE_FF(fs, res); }


/* File shareing feature */
#if _FS_SHARE
#if _FS_READONLY
#error _FS_SHARE must be 0 on read-only cfg.
#endif
typedef struct {
	FATFS *fs;			/* File ID 1, volume (NULL:blank entry) */
	DWORD clu;				/* File ID 2, directory */
	WORD idx;			/* File ID 3, directory index */
	WORD ctr;			/* File open counter, 0:none, 0x01..0xFF:read
open count, 0x100:write mode */
} FILESEM;
#endif


/* Misc definitions */
#define LD_CLUST(dir)(((DWORD)LD_WORD(dir+DIR_FstClusHI)<<16)	|
LD_WORD(dir+DIR_FstClusLO))
#define	ST_CLUST(dir,cl)	{ST_WORD(dir+DIR_FstClusLO,	cl);
ST_WORD(dir+DIR_FstClusHI, (DWORD)cl>>16);}
```

```
/* DBCS code ranges and SBCS extend char conversion table */

#if _CODE_PAGE == 932       /* Japanese Shift-JIS */
#define _DF1S     0x81  /* DBC 1st byte range 1 start */
#define _DF1E     0x9F  /* DBC 1st byte range 1 end */
#define _DF2S     0xE0  /* DBC 1st byte range 2 start */
#define _DF2E     0xFC /* DBC 1st byte range 2 end */
#define _DS1S     0x40  /* DBC 2nd byte range 1 start */
#define _DS1E     0x7E  /* DBC 2nd byte range 1 end */
#define _DS2S     0x80  /* DBC 2nd byte range 2 start */
#define _DS2E     0xFC /* DBC 2nd byte range 2 end */

#elif _CODE_PAGE == 936    /* Simplified Chinese GBK */
#define _DF1S     0x81
#define _DF1E     0xFE
#define _DS1S     0x40
#define _DS1E     0x7E
#define _DS2S     0x80
#define _DS2E     0xFE

#elif _CODE_PAGE == 949    /* Korean */
#define _DF1S     0x81
#define _DF1E     0xFE
#define _DS1S     0x41
#define _DS1E     0x5A
#define _DS2S     0x61
#define _DS2E     0x7A
#define _DS3S     0x81
#define _DS3E     0xFE

#elif _CODE_PAGE == 950    /* Traditional Chinese Big5 */
#define _DF1S     0x81
#define _DF1E     0xFE
#define _DS1S     0x40
#define _DS1E     0x7E
```

```
#define _DS2S      0xA1
#define _DS2E      0xFE

#elif _CODE_PAGE == 437     /* U.S. (OEM) */
#define _DF1S      0
#define                                                    _EXCVT
{0x80,0x9A,0x90,0x41,0x8E,0x41,0x8F,0x80,0x45,0x45,0x45,0x49,0x49,0x49,0x
8E,0x8F,0x90,0x92,0x92,0x4F,0x99,0x4F,0x55,0x55,0x59,0x99,0x9A,0x9B,0x9C
,0x9D,0x9E,0x9F, \

        0x41,0x49,0x4F,0x55,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0x
AC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,
0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xE
C,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xF
A,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 720     /* Arabic (OEM) */
#define _DF1S      0
#define                                                    _EXCVT
{0x80,0x81,0x45,0x41,0x84,0x41,0x86,0x43,0x45,0x45,0x45,0x49,0x49,0x8D,0x
8E,0x8F,0x90,0x92,0x92,0x93,0x94,0x95,0x49,0x49,0x98,0x99,0x9A,0x9B,0x9C
,0x9D,0x9E,0x9F, \

        0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0
xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB
9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \
```

```
        0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 737     /* Greek (OEM) */
#define _DF1S      0
#define                                                                 _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x92,0x92,0x93,0x94,0x95,0x96,0x97,0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87, \

        0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0xAA,0x92,0x93,0x94,0x95,0x96,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0x97,0xEA,0xEB,0xEC,0xE4,0xED,0xEE,0xE7,0xE8,0xF1,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 775     /* Baltic (OEM) */
#define _DF1S      0
#define                                                                 _EXCVT
{0x80,0x9A,0x91,0xA0,0x8E,0x95,0x8F,0x80,0xAD,0xED,0x8A,0x8A,0xA1,0x8D,0x8E,0x8F,0x90,0x92,0x92,0xE2,0x99,0x95,0x96,0x97,0x97,0x99,0x9A,0x9D,0x9C,0x9D,0x9E,0x9F, \

        0xA0,0xA1,0xE0,0xA3,0xA3,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
```

```
CC,0xCD,0xCE,0xCF,0xB5,0xB6,0xB7,0xB8,0xBD,0xBE,0xC6,0xC7,0xA5,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

    0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE3,0xE8,0xE8,0xEA,0xEA,0xE
E,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xF
A,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 850    /* Multilingual Latin 1 (OEM) */
#define _DF1S      0
#define                                                    _EXCVT
{0x80,0x9A,0x90,0xB6,0x8E,0xB7,0x8F,0x80,0xD2,0xD3,0xD4,0xD8,0xD7,0xD
E,0x8E,0x8F,0x90,0x92,0x92,0xE2,0x99,0xE3,0xEA,0xEB,0x59,0x99,0x9A,0x9
D,0x9C,0x9D,0x9E,0x9F, \

    0xB5,0xD6,0xE0,0xE9,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0x
AC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,
0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

    0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

    0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE7,0xE7,0xE9,0xEA,0xEB,0xE
D,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xF
A,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 852    /* Latin 2 (OEM) */
#define _DF1S      0
#define                                                    _EXCVT
{0x80,0x9A,0x90,0xB6,0x8E,0xDE,0x8F,0x80,0x9D,0xD3,0x8A,0x8A,0xD7,0x8
D,0x8E,0x8F,0x90,0x91,0x91,0xE2,0x99,0x95,0x95,0x97,0x97,0x99,0x9A,0x9B,
0x9B,0x9D,0x9E,0x9F, \

    0xB5,0xD6,0xE0,0xE9,0xA4,0xA4,0xA6,0xA6,0xA8,0xA8,0xAA,0x8D,0x
AC,0xB8,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,
0xBA,0xBB,0xBC,0xBD,0xBD,0xBF, \
```

```
        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC6,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD1,0xD1,0xD2,0xD3,0xD2,0xD5,0xD6,0xD7,0xB7,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xE0,0xE1,0xE2,0xE3,0xE3,0xD5,0xE6,0xE6,0xE8,0xE9,0xE8,0xEB,0xE
D,0xED,0xDD,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xF
A,0xEB,0xFC,0xFC,0xFE,0xFF}

#elif _CODE_PAGE == 855     /* Cyrillic (OEM) */
#define _DF1S      0
#define                                                          _EXCVT
{0x81,0x81,0x83,0x83,0x85,0x85,0x87,0x87,0x89,0x89,0x8B,0x8B,0x8D,0x8D,0
x8F,0x8F,0x91,0x91,0x93,0x93,0x95,0x95,0x97,0x97,0x99,0x99,0x9B,0x9B,0x9
D,0x9D,0x9F,0x9F, \

        0xA1,0xA1,0xA3,0xA3,0xA5,0xA5,0xA7,0xA7,0xA9,0xA9,0xAB,0xAB,0
xAD,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB6,0xB6,0xB8,0xB8,0x
B9,0xBA,0xBB,0xBC,0xBE,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD1,0xD1,0xD3,0xD3,0xD5,0xD5,0xD7,0xD7,0xDD,0x
D9,0xDA,0xDB,0xDC,0xDD,0xE0,0xDF, \

        0xE0,0xE2,0xE2,0xE4,0xE4,0xE6,0xE6,0xE8,0xE8,0xEA,0xEA,0xEC,0xE
C,0xEE,0xEE,0xEF,0xF0,0xF2,0xF2,0xF4,0xF4,0xF6,0xF6,0xF8,0xF8,0xFA,0xF
A,0xFC,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 857     /* Turkish (OEM) */
#define _DF1S      0
#define                                                          _EXCVT
{0x80,0x9A,0x90,0xB6,0x8E,0xB7,0x8F,0x80,0xD2,0xD3,0xD4,0xD8,0xD7,0x9
8,0x8E,0x8F,0x90,0x92,0x92,0xE2,0x99,0xE3,0xEA,0xEB,0x98,0x99,0x9A,0x9
D,0x9C,0x9D,0x9E,0x9E, \

        0xB5,0xD6,0xE0,0xE9,0xA5,0xA5,0xA6,0xA6,0xA8,0xA9,0xAA,0xAB,0x
```

```
AC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,
0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xD
E,0x59,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA
,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 858     /* Multilingual Latin 1 + Euro (OEM) */
#define _DF1S      0
#define                                                          _EXCVT
{0x80,0x9A,0x90,0xB6,0x8E,0xB7,0x8F,0x80,0xD2,0xD3,0xD4,0xD8,0xD7,0xD
E,0x8E,0x8F,0x90,0x92,0x92,0xE2,0x99,0xE3,0xEA,0xEB,0x59,0x99,0x9A,0x9
D,0x9C,0x9D,0x9E,0x9F, \

        0xB5,0xD6,0xE0,0xE9,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0x
AC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,
0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD1,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE7,0xE7,0xE9,0xEA,0xEB,0xE
D,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xF
A,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 862     /* Hebrew (OEM) */
#define _DF1S      0
#define                                                          _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9
C,0x9D,0x9E,0x9F, \
```

```
        0x41,0x49,0x4F,0x55,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0x
AC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,
0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xE
C,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xF
A,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 866     /* Russian (OEM) */
#define _DF1S      0
#define                                                    _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9
C,0x9D,0x9E,0x9F, \

        0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x
8D,0x8E,0x8F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,
0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0x90,0x91,0x92,0x93,0x9d,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x
9D,0x9E,0x9F,0xF0,0xF0,0xF2,0xF2,0xF4,0xF4,0xF6,0xF6,0xF8,0xF9,0xFA,0xF
B,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 874     /* Thai (OEM, Windows) */
#define _DF1S      0
#define                                                    _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
```

```
x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9
C,0x9D,0x9E,0x9F, \

    0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0
xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB
9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

    0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

    0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xE
C,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xF
A,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 1250 /* Central Europe (Windows) */
#define _DF1S      0
#define                                                    _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x8A,0x9B,0x8
C,0x8D,0x8E,0x8F, \

    0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0
xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xA3,0xB4,0xB5,0xB6,0xB7,0xB8,0x
A5,0xAA,0xBB,0xBC,0xBD,0xBC,0xAF, \

    0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

    0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xFF}

#elif _CODE_PAGE == 1251 /* Cyrillic (Windows) */
#define _DF1S      0
```

```
#define                                                         _EXCVT
{0x80,0x81,0x82,0x82,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
x8E,0x8F,0x80,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x8A,0x9B,0x8
C,0x8D,0x8E,0x8F, \

        0xA0,0xA2,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0
xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB2,0xA5,0xB5,0xB6,0xB7,0xA8,0x
B9,0xAA,0xBB,0xA3,0xBD,0xBD,0xAF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF}

#elif _CODE_PAGE == 1252 /* Latin 1 (Windows) */
#define _DF1S      0
#define                                                         _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0xAd,0x9B,0x8
C,0x9D,0xAE,0x9F, \

        0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0x
AC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9
,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0x9F}

#elif _CODE_PAGE == 1253 /* Greek (Windows) */
```

```
#define _DF1S	0
#define _EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \

	0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

	0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xA2,0xB8,0xB9,0xBA, \

	0xE0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xF2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xFB,0xBC,0xFD,0xBF,0xFF}

#elif _CODE_PAGE == 1254 /* Turkish (Windows) */
#define _DF1S	0
#define _EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x8A,0x9B,0x8C,0x9D,0x9E,0x9F, \

	0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

	0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

	0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0x9F}
```

```
#elif _CODE_PAGE == 1255 /* Hebrew (Windows) */
#define _DF1S      0
#define                                                                 _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9
C,0x9D,0x9E,0x9F, \

        0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0x
AC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9
,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xE
C,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xF
A,0xFB,0xFC,0xFD,0xFE,0xFF}

#elif _CODE_PAGE == 1256 /* Arabic (Windows) */
#define _DF1S      0
#define                                                                 _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x8
C,0x9D,0x9E,0x9F, \

        0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0
xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB
9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0x41,0xE1,0x41,0xE3,0xE4,0xE5,0xE6,0x43,0x45,0x45,0x45,0x45,0xEC,0
xED,0x49,0x49,0xF0,0xF1,0xF2,0xF3,0x4F,0xF5,0xF6,0xF7,0xF8,0x55,0xFA,0x
55,0x55,0xFD,0xFE,0xFF}
```

```
#elif _CODE_PAGE == 1257 /* Baltic (Windows) */
#define _DF1S      0
#define                                                    _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9
C,0x9D,0x9E,0x9F, \

        0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0
xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xA8,0x
B9,0xAA,0xBB,0xBC,0xBD,0xBE,0xAF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xFF}

#elif _CODE_PAGE == 1258 /* Vietnam (OEM, Windows) */
#define _DF1S      0
#define                                                    _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0
x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0xA
C,0x9D,0x9E,0x9F, \

        0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0x
AC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9
,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
CC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD
9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \

        0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0x
```

```
EC,0xCD,0xCE,0xCF,0xD0,0xD1,0xF2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9
,0xDA,0xDB,0xDC,0xDD,0xFE,0x9F}

#elif _CODE_PAGE == 1        /* ASCII (for only non-LFN cfg) */
#if _USE_LFN
#error Cannot use LFN feature without valid code page.
#endif
#define _DF1S      0

#else
#error Unknown code page

#endif


/* Character code support macros */
#define IsUpper(c)(((c)>='A')&&((c)<='Z'))
#define IsLower(c)         (((c)>='a')&&((c)<='z'))
#define IsDigit(c)  (((c)>='0')&&((c)<='9'))

#if _DF1S            /* Code page is DBCS */

#ifdef _DF2S         /* Two 1st byte areas */
#define IsDBCS1(c)       (((BYTE)(c) >= _DF1S && (BYTE)(c) <= _DF1E) ||
((BYTE)(c) >= _DF2S && (BYTE)(c) <= _DF2E))
#else                /* One 1st byte area */
#define IsDBCS1(c)       ((BYTE)(c) >= _DF1S && (BYTE)(c) <= _DF1E)
#endif

#ifdef _DS3S         /* Three 2nd byte areas */
#define IsDBCS2(c)       (((BYTE)(c) >= _DS1S && (BYTE)(c) <= _DS1E) ||
((BYTE)(c) >= _DS2S && (BYTE)(c) <= _DS2E) || ((BYTE)(c) >= _DS3S &&
(BYTE)(c) <= _DS3E))
#else                /* Two 2nd byte areas */
#define IsDBCS2(c)       (((BYTE)(c) >= _DS1S && (BYTE)(c) <= _DS1E) ||
((BYTE)(c) >= _DS2S && (BYTE)(c) <= _DS2E))
#endif
```

```
#else                   /* Code page is SBCS */

#define IsDBCS1(c)      0
#define IsDBCS2(c)      0

#endif /* _DF1S */


/* Name status flags */
#define NS              11              /* Index of name status byte in fn[] */
#define NS_LOSS         0x01  /* Out of 8.3 format */
#define NS_LFN          0x02  /* Force to create LFN entry */
#define NS_LAST         0x04  /* Last segment */
#define NS_BODY                 0x08  /* Lower case flag (body) */
#define NS_EXT          0x10  /* Lower case flag (ext) */
#define NS_DOT          0x20  /* Dot entry */


/* FAT sub-type boundaries */
/* Note that the FAT spec by Microsoft says 4085 but Windows works with 4087!
*/
#define MIN_FAT16       4086  /* Minimum number of clusters for FAT16 */
#define         MIN_FAT32       65526/* Minimum number of clusters for FAT32 */


/* FatFs refers the members in the FAT structures as byte array instead of
/ structure member because the structure is not binary compatible between
/ different platforms */

#define BS_jmpBoot                      0       /* Jump instruction (3) */
#define BS_OEMName                      3       /* OEM name (8) */
#define BPB_BytsPerSec                  11      /* Sector size [byte] (2) */
#define BPB_SecPerClus                  13      /* Cluster size [sector] (1) */
#define BPB_RsvdSecCnt                  14      /* Size of reserved area [sector] (2) */
#define BPB_NumFATs                     16      /* Number of FAT copies (1) */
```

```
#define BPB_RootEntCnt          17    /* Number of root dir entries for
FAT12/16 (2) */
#define BPB_TotSec16    19      /* Volume size [sector] (2) */
#define BPB_Media               21    /* Media descriptor (1) */
#define BPB_FATSz16             22    /* FAT size [sector] (2) */
#define BPB_SecPerTrk   24      /* Track size [sector] (2) */
#define BPB_NumHeads            26    /* Number of heads (2) */
#define BPB_HiddSec             28    /* Number of special hidden sectors (4)
*/
#define BPB_TotSec32    32      /* Volume size [sector] (4) */
#define BS_DrvNum               36    /* Physical drive number (2) */
#define BS_BootSig              38    /* Extended boot signature (1) */
#define BS_VolID        39      /* Volume serial number (4) */
#define BS_VolLab               43    /* Volume label (8) */
#define BS_FilSysType   54      /* File system type (1) */
#define BPB_FATSz32             36    /* FAT size [sector] (4) */
#define BPB_ExtFlags    40      /* Extended flags (2) */
#define BPB_FSVer               42    /* File system version (2) */
#define BPB_RootClus    44      /* Root dir first cluster (4) */
#define BPB_FSInfo              48    /* Offset of FSInfo sector (2) */
#define BPB_BkBootSec           50    /* Offset of backup boot sectot (2) */
#define BS_DrvNum32             64    /* Physical drive number (2) */
#define BS_BootSig32    66      /* Extended boot signature (1) */
#define BS_VolID32              67    /* Volume serial number (4) */
#define BS_VolLab32             71    /* Volume label (8) */
#define BS_FilSysType32         82    /* File system type (1) */
#define     FSI_LeadSig             0     /* FSI: Leading signature (4) */
#define     FSI_StrucSig            484   /* FSI: Structure signature (4) */
#define     FSI_Free_Count          488   /* FSI: Number of free clusters (4) */
#define     FSI_Nxt_Free            492   /* FSI: Last allocated cluster (4) */
#define MBR_Table               446   /* MBR: Partition table offset (2) */
#define     SZ_PTE                  16    /* MBR: Size of a partition table
entry */
#define BS_55AA                 510   /* Boot sector signature (2) */

#define     DIR_Name            0     /* Short file name (11) */
#define     DIR_Attr            11    /* Attribute (1) */
```

```
#define	DIR_NTres		12	/* NT flag (1) */
#define	DIR_CrtTime			14	/* Created time (2) */
#define	DIR_CrtDate			16	/* Created date (2) */
#define	DIR_FstClusHI		20	/* Higher 16-bit of first cluster (2) */
#define	DIR_WrtTime			22	/* Modified time (2) */
#define	DIR_WrtDate			24	/* Modified date (2) */
#define	DIR_FstClusLO		26	/* Lower 16-bit of first cluster (2) */
#define	DIR_FileSize		28	/* File size (4) */
#define	LDIR_Ord		0	/* LFN entry order and LLE flag (1) */
#define	LDIR_Attr		11	/* LFN attribute (1) */
#define	LDIR_Type		12	/* LFN type (1) */
#define	LDIR_Chksum			13	/* Sum of corresponding SFN entry */
#define	LDIR_FstClusLO		26	/* Filled by zero (0) */
#define	SZ_DIR				32		/* Size of a directory entry */
#define	LLE				0x40 /* Last long entry flag in LDIR_Ord */
#define	DDE				0xE5 /* Deleted directory enrty mark in DIR_Name[0] */
#define	NDDE				0x05 /* Replacement of a character collides with DDE */


/*---------------------------------------------------------*/
/* Work area                                               */

#if _VOLUMES
static
FATFS *FatFs[_VOLUMES];	/* Pointer to the file system objects (logical drives) */
#else
#error Number of drives must not be 0.
#endif

static
WORD Fsid;				/* File system mount ID */
```

```
#if _FS_RPATH
static
BYTE CurrVol;                   /* Current drive */
#endif

#if _FS_SHARE
static
FILESEM   Files[_FS_SHARE];     /* File lock semaphores */
#endif

#if _USE_LFN == 0               /* No LFN */
#define     DEF_NAMEBUF             BYTE sfn[12]
#define INIT_BUF(dobj)          (dobj).fn = sfn
#define     FREE_BUF()

#elif _USE_LFN == 1             /* LFN with static LFN working buffer */
static WCHAR LfnBuf[_MAX_LFN+1];
#define     DEF_NAMEBUF             BYTE sfn[12]
#define INIT_BUF(dobj)          { (dobj).fn = sfn; (dobj).lfn = LfnBuf; }
#define     FREE_BUF()

#elif _USE_LFN == 2         /* LFN with dynamic LFN working buffer on the stack */
#define     DEF_NAMEBUF             BYTE sfn[12]; WCHAR lbuf[_MAX_LFN+1]
#define INIT_BUF(dobj)          { (dobj).fn = sfn; (dobj).lfn = lbuf; }
#define     FREE_BUF()

#elif _USE_LFN == 3         /* LFN with dynamic LFN working buffer on the heap */
#define     DEF_NAMEBUF             BYTE sfn[12]; WCHAR *lfn
#define INIT_BUF(dobj)          { lfn = ff_memalloc((_MAX_LFN + 1) * 2); \
                                  if (!lfn) LEAVE_FF((dobj).fs, FR_NOT_ENOUGH_CORE); \
                                  (dobj).lfn = lfn;   (dobj).fn = sfn; }
```

```
#define        FREE_BUF()                      ff_memfree(lfn)

#else
#error Wrong LFN configuration.
#endif




/*------------------------------------------------------------------------

   Module Private Functions

------------------------------------------------------------------------*/


/*-----------------------------------------------------------------------*/
/* String functions                                                      */
/*-----------------------------------------------------------------------*/

/* Copy memory to memory */
static
void mem_cpy (void* dst, const void* src, UINT cnt) {
        BYTE *d = (BYTE*)dst;
        const BYTE *s = (const BYTE*)src;

#if _WORD_ACCESS == 1
        while (cnt >= sizeof(int)) {
                *(int*)d = *(int*)s;
                d += sizeof(int); s += sizeof(int);
                cnt -= sizeof(int);
        }
#endif
        while (cnt--)
                *d++ = *s++;
}
```

```
/* Fill memory */
static
void mem_set (void* dst, int val, UINT cnt) {
	BYTE *d = (BYTE*)dst;

	while (cnt--)
		*d++ = (BYTE)val;
}

/* Compare memory to memory */
static
int mem_cmp (const void* dst, const void* src, UINT cnt) {
	const BYTE *d = (const BYTE *)dst, *s = (const BYTE *)src;
	int r = 0;

	while (cnt-- && (r = *d++ - *s++) == 0) ;
	return r;
}

/* Check if chr is contained in the string */
static
int chk_chr (const char* str, int chr) {
	while (*str && *str != chr) str++;
	return *str;
}



/*-----------------------------------------------------------------------*/
/* Request/Release grant to access the volume                            */
/*-----------------------------------------------------------------------*/
#if _FS_REENTRANT

static
int lock_fs (
	FATFS *fs		/* File system object */
)
```

```
{
	return ff_req_grant(fs->sobj);
}


static
void unlock_fs (
	FATFS *fs,		/* File system object */
	FRESULT res		/* Result code to be returned */
)
{
	if (res != FR_NOT_ENABLED &&
		res != FR_INVALID_DRIVE &&
		res != FR_INVALID_OBJECT &&
		res != FR_TIMEOUT) {
		ff_rel_grant(fs->sobj);
	}
}
#endif



/*----------------------------------------------------------------------*/
/* File shareing control functions                                      */
/*----------------------------------------------------------------------*/
#if _FS_SHARE

static
FRESULT chk_lock (	/* Check if the file can be accessed */
	DIRS* dj,		/* Directory object pointing the file to be checked */
	int acc			/* Desired access (0:Read, 1:Write, 2:Delete/Rename) */
)
{
	UINT i, be;

	/* Search file semaphore table */
```

```
	for (i = be = 0; i < _FS_SHARE; i++) {
		if (Files[i].fs) {	/* Existing entry */
			if (Files[i].fs == dj->fs &&		/* Check if the file matched with an open file */
				Files[i].clu == dj->sclust &&
				Files[i].idx == dj->index) break;
		} else {				/* Blank entry */
			be++;
		}
	}
	if (i == _FS_SHARE)	/* The file is not opened */
		return (be || acc == 2) ? FR_OK : FR_TOO_MANY_OPEN_FILES;
	/* Is there a blank entry for new file? */

	/* The file has been opened. Reject any open against writing file and all write mode open */
	return (acc || Files[i].ctr == 0x100) ? FR_LOCKED : FR_OK;
}



static
int enq_lock (	/* Check if an entry is available for a new file */
	FATFS* fs	/* File system object */
)
{
	UINT i;

	for (i = 0; i < _FS_SHARE && Files[i].fs; i++) ;
	return (i == _FS_SHARE) ? 0 : 1;
}



static
UINT inc_lock (	/* Increment file open counter and returns its index (0:int error) */
	DIRS* dj,	/* Directory object pointing the file to register or increment */
	int acc		/* Desired access mode (0:Read, !0:Write) */
```

```
)
{
	UINT i;


	for (i = 0; i < _FS_SHARE; i++) {	/* Find the file */
		if (Files[i].fs == dj->fs &&
			Files[i].clu == dj->sclust &&
			Files[i].idx == dj->index) break;
	}

	if (i == _FS_SHARE) {				/* Not opened. Register it as new.
*/
		for (i = 0; i < _FS_SHARE && Files[i].fs; i++) ;
		if (i == _FS_SHARE) return 0; /* No space to register (int err) */
		Files[i].fs = dj->fs;
		Files[i].clu = dj->sclust;
		Files[i].idx = dj->index;
		Files[i].ctr = 0;
	}

	if (acc && Files[i].ctr) return 0;		/* Access violation (int err) */

	Files[i].ctr = acc ? 0x100 : Files[i].ctr + 1;	/* Set semaphore value */

	return i + 1;
}


static
FRESULT dec_lock (	/* Decrement file open counter */
	UINT i			/* Semaphore index */
)
{
	WORD n;
	FRESULT res;
```

```
	if (--i < _FS_SHARE) {
		n = Files[i].ctr;
		if (n == 0x100) n = 0;
		if (n) n--;
		Files[i].ctr = n;
		if (!n) Files[i].fs = 0;
		res = FR_OK;
	} else {
		res = FR_INT_ERR;
	}
	return res;
}


static
void clear_lock (	/* Clear lock entries of the volume */
	FATFS *fs
)
{
	UINT i;

	for (i = 0; i < _FS_SHARE; i++) {
		if (Files[i].fs == fs) Files[i].fs = 0;
	}
}
#endif



/*-----------------------------------------------------------------------*/
/* Change window offset                                                  */
/*-----------------------------------------------------------------------*/

static
FRESULT move_window (
	FATFS *fs,		/* File system object */
```

```
	DWORD sector	/* Sector number to make appearance in the fs->win[] */
)					/* Move to zero only writes back dirty window */
{
	DWORD wsect;


	wsect = fs->winsect;
	if (wsect != sector) {	/* Changed current window */
#if !_FS_READONLY
		if (fs->wflag) {	/* Write back dirty window if needed */
			if (disk_write(fs->drv, fs->win, wsect, 1) != RES_OK)
				return FR_DISK_ERR;
			fs->wflag = 0;
			if (wsect < (fs->fatbase + fs->fsize)) {		/* In FAT area */
				BYTE nf;
				for (nf = fs->n_fats; nf > 1; nf--) {	/* Reflect the change to all FAT copies */
					wsect += fs->fsize;
					disk_write(fs->drv, fs->win, wsect, 1);
				}
			}
		}
#endif
		if (sector) {
			if (disk_read(fs->drv, fs->win, sector, 1) != RES_OK)
				return FR_DISK_ERR;
			fs->winsect = sector;
		}
	}

	return FR_OK;
}




/*-----------------------------------------------------------------------*/
```

```
/* Clean-up cached data                                        */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY
static
FRESULT sync (  /* FR_OK: successful, FR_DISK_ERR: failed */
        FATFS *fs  /* File system object */
)
{
        FRESULT res;


        res = move_window(fs, 0);
        if (res == FR_OK) {
                /* Update FSInfo sector if needed */
                if (fs->fs_type == FS_FAT32 && fs->fsi_flag) {
                        fs->winsect = 0;
                        /* Create FSInfo structure */
                        mem_set(fs->win, 0, 512);
                        ST_WORD(fs->win+BS_55AA, 0xAA55);
                        ST_DWORD(fs->win+FSI_LeadSig, 0x41615252);
                        ST_DWORD(fs->win+FSI_StrucSig, 0x61417272);
                        ST_DWORD(fs->win+FSI_Free_Count, fs->free_clust);
                        ST_DWORD(fs->win+FSI_Nxt_Free, fs->last_clust);
                        /* Write it into the FSInfo sector */
                        disk_write(fs->drv, fs->win, fs->fsi_sector, 1);
                        fs->fsi_flag = 0;
                }
                /* Make sure that no pending write process in the physical drive */
                if (disk_ioctl(fs->drv, CTRL_SYNC, (void*)0) != RES_OK)
                        res = FR_DISK_ERR;
        }

        return res;
}
#endif
```

```
/*-----------------------------------------------------------------------*/
/* Get sector# from cluster#                                             */
/*-----------------------------------------------------------------------*/


DWORD clust2sect (      /* !=0: Sector number, 0: Failed - invalid cluster# */
        FATFS *fs,          /* File system object */
        DWORD clst              /* Cluster# to be converted */
)
{
        clst -= 2;
        if (clst >= (fs->n_fatent - 2)) return 0;         /* Invalid cluster# */
        return clst * fs->csize + fs->database;
}




/*-----------------------------------------------------------------------*/
/* FAT access - Read value of a FAT entry                                */
/*-----------------------------------------------------------------------*/


DWORD get_fat ( /* 0xFFFFFFFF:Disk error, 1:Internal error, Else:Cluster status
*/
        FATFS *fs, /* File system object */
        DWORD clst      /* Cluster# to get the link information */
)
{
        UINT wc, bc;
        BYTE *p;


        if (clst < 2 || clst >= fs->n_fatent)       /* Chack range */
                return 1;
```

```
	switch (fs->fs_type) {
	case FS_FAT12 :
		bc = (UINT)clst; bc += bc / 2;
		if (move_window(fs, fs->fatbase + (bc / SS(fs)))) break;
		wc = fs->win[bc % SS(fs)]; bc++;
		if (move_window(fs, fs->fatbase + (bc / SS(fs)))) break;
		wc |= fs->win[bc % SS(fs)] << 8;
		return (clst & 1) ? (wc >> 4) : (wc & 0xFFF);

	case FS_FAT16 :
		if (move_window(fs, fs->fatbase + (clst / (SS(fs) / 2)))) break;
		p = &fs->win[clst * 2 % SS(fs)];
		return LD_WORD(p);

	case FS_FAT32 :
		if (move_window(fs, fs->fatbase + (clst / (SS(fs) / 4)))) break;
		p = &fs->win[clst * 4 % SS(fs)];
		return LD_DWORD(p) & 0x0FFFFFFF;
	}

	return 0xFFFFFFFF;	/* An error occurred at the disk I/O layer */
}




/*-----------------------------------------------------------------------*/
/* FAT access - Change value of a FAT entry                              */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY

FRESULT put_fat (
	FATFS *fs,	/* File system object */
	DWORD clst,	/* Cluster# to be changed in range of 2 to fs->n_fatent - 1 */
	DWORD val	/* New value to mark the cluster */
```

```
)
{
	UINT bc;
	BYTE *p;
	FRESULT res;


	if (clst < 2 || clst >= fs->n_fatent) {	/* Check range */
		res = FR_INT_ERR;

	} else {
		switch (fs->fs_type) {
		case FS_FAT12 :
			bc = clst; bc += bc / 2;
			res = move_window(fs, fs->fatbase + (bc / SS(fs)));
			if (res != FR_OK) break;
			p = &fs->win[bc % SS(fs)];
			*p = (clst & 1) ? ((*p & 0x0F) | ((BYTE)val << 4)) : (BYTE)val;
			bc++;
			fs->wflag = 1;
			res = move_window(fs, fs->fatbase + (bc / SS(fs)));
			if (res != FR_OK) break;
			p = &fs->win[bc % SS(fs)];
			*p = (clst & 1) ? (BYTE)(val >> 4) : ((*p & 0xF0) | ((BYTE)(val
>> 8) & 0x0F));
			break;

		case FS_FAT16 :
			res = move_window(fs, fs->fatbase + (clst / (SS(fs) / 2)));
			if (res != FR_OK) break;
			p = &fs->win[clst * 2 % SS(fs)];
			ST_WORD(p, (WORD)val);
			break;

		case FS_FAT32 :
			res = move_window(fs, fs->fatbase + (clst / (SS(fs) / 4)));
			if (res != FR_OK) break;
```

```
                        p = &fs->win[clst * 4 % SS(fs)];
                        val |= LD_DWORD(p) & 0xF0000000;
                        ST_DWORD(p, val);
                        break;

                default :
                        res = FR_INT_ERR;
                }
                fs->wflag = 1;
        }

        return res;
}
#endif /* !_FS_READONLY */




/*-----------------------------------------------------------------------*/
/* FAT handling - Remove a cluster chain                                 */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY
static
FRESULT remove_chain (
        FATFS *fs,                  /* File system object */
        DWORD clst                          /* Cluster# to remove a chain from */
)
{
        FRESULT res;
        DWORD nxt;
#if _USE_ERASE
        DWORD scl = clst, ecl = clst, resion[2];
#endif

        if (clst < 2 || clst >= fs->n_fatent) {     /* Check range */
                res = FR_INT_ERR;
```

```
	} else {
		res = FR_OK;
		while (clst < fs->n_fatent) {			/* Not a last link? */
			nxt = get_fat(fs, clst);		/* Get cluster status */
			if (nxt == 0) break;				/* Empty cluster? */
			if (nxt == 1) { res = FR_INT_ERR; break; }	/* Internal error? */
			if (nxt == 0xFFFFFFFF) { res = FR_DISK_ERR; break; }	/* Disk error? */
			res = put_fat(fs, clst, 0);		/* Mark the cluster "empty" */
			if (res != FR_OK) break;
			if (fs->free_clust != 0xFFFFFFFF) {	/* Update FSInfo */
				fs->free_clust++;
				fs->fsi_flag = 1;
			}
#if _USE_ERASE
			if (ecl + 1 == nxt) {	/* Next cluster is contiguous */
				ecl = nxt;
			} else {				/* End of contiguous clusters */
				resion[0] = clust2sect(fs, scl);
	/* Start sector */
				resion[1] = clust2sect(fs, ecl) + fs->csize - 1;	/* End sector */
				disk_ioctl(fs->drv, CTRL_ERASE_SECTOR, resion);
	/* Erase the block */
				scl = ecl = nxt;
			}
#endif
			clst = nxt;	/* Next cluster */
		}
	}

	return res;
}
#endif
```

```
/*-----------------------------------------------------------------------*/
/* FAT handling - Stretch or Create a cluster chain                      */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY
static
DWORD create_chain (  /* 0:No free cluster, 1:Internal error, 0xFFFFFFFF:Disk error, >=2:New cluster# */
        FATFS *fs,              /* File system object */
        DWORD clst              /* Cluster# to stretch. 0 means create a new chain. */
)
{
        DWORD cs, ncl, scl;
        FRESULT res;


        if (clst == 0) {        /* Create a new chain */
                scl = fs->last_clust;                   /* Get suggested start point */
                if (!scl || scl >= fs->n_fatent) scl = 1;
        }
        else {                                  /* Stretch the current chain */
                cs = get_fat(fs, clst);                 /* Check the cluster status */
                if (cs < 2) return 1;                   /* It is an invalid cluster */
                if (cs < fs->n_fatent) return cs; /* It is already followed by next cluster */
                scl = clst;
        }

        ncl = scl;                              /* Start cluster */
        for (;;) {
                ncl++;                                                  /* Next cluster */
                if (ncl >= fs->n_fatent) {          /* Wrap around */
                        ncl = 2;
                        if (ncl > scl) return 0;      /* No free cluster */
```

```
                }
                cs = get_fat(fs, ncl);                  /* Get the cluster status */
                if (cs == 0) break;                     /* Found a free cluster */
                if (cs == 0xFFFFFFFF || cs == 1)/* An error occurred */
                        return cs;
                if (ncl == scl) return 0;               /* No free cluster */
        }

        res = put_fat(fs, ncl, 0x0FFFFFFF);   /* Mark the new cluster "last link" */
        if (res == FR_OK && clst != 0) {
                res = put_fat(fs, clst, ncl);         /* Link it to the previous one if needed
*/
        }
        if (res == FR_OK) {
                fs->last_clust = ncl;                   /* Update FSINFO */
                if (fs->free_clust != 0xFFFFFFFF) {
                        fs->free_clust--;
                        fs->fsi_flag = 1;
                }
        } else {
                ncl = (res == FR_DISK_ERR) ? 0xFFFFFFFF : 1;
        }

        return ncl;             /* Return new cluster number or error code */
}
#endif /* !_FS_READONLY */



/*-----------------------------------------------------------------------*/
/* FAT handling - Convert offset into cluster with link map table        */
/*-----------------------------------------------------------------------*/

#if _USE_FASTSEEK
static
DWORD clmt_clust (      /* <2:Error, >=2:Cluster number */
        FIL* fp,                /* Pointer to the file object */
```

```
        DWORD ofs               /* File offset to be converted to cluster# */
)
{
        DWORD cl, ncl, *tbl;


        tbl = fp->cltbl + 1;/* Top of CLMT */
        cl = ofs / SS(fp->fs) / fp->fs->csize;   /* Cluster order from top of the file */
        for (;;) {
                ncl = *tbl++;                   /* Number of cluters in the fragment */
                if (!ncl) return 0;             /* End of table? (error) */
                if (cl < ncl) break; /* In this fragment? */
                cl -= ncl; tbl++;               /* Next fragment */
        }
        return cl + *tbl;     /* Return the cluster number */
}
#endif        /* _USE_FASTSEEK */



/*-----------------------------------------------------------------------*/
/* Directory handling - Set directory index                              */
/*-----------------------------------------------------------------------*/

static
FRESULT dir_sdi (
        DIRS *dj,             /* Pointer to directory object */
        WORD idx              /* Directory index number */
)
{
        DWORD clst;
        WORD ic;


        dj->index = idx;
        clst = dj->sclust;
        if (clst == 1 || clst >= dj->fs->n_fatent)      /* Check start cluster range */
```

```
            return FR_INT_ERR;
        if (!clst && dj->fs->fs_type == FS_FAT32)/* Replace cluster# 0 with root cluster# if in FAT32 */
            clst = dj->fs->dirbase;

        if (clst == 0) {        /* Static table (root-dir in FAT12/16) */
            dj->clust = clst;
            if (idx >= dj->fs->n_rootdir)           /* Index is out of range */
                return FR_INT_ERR;
            dj->sect = dj->fs->dirbase + idx / (SS(dj->fs) / SZ_DIR);       /* Sector# */
        }
        else {                          /* Dynamic table (sub-dirs or root-dir in FAT32) */
            ic = SS(dj->fs) / SZ_DIR * dj->fs->csize;   /* Entries per cluster */
            while (idx >= ic) {/* Follow cluster chain */
                clst = get_fat(dj->fs, clst);                   /* Get next cluster */
                if (clst == 0xFFFFFFFF) return FR_DISK_ERR;  /* Disk error */
                if (clst < 2 || clst >= dj->fs->n_fatent) /* Reached to end of table or int error */
                    return FR_INT_ERR;
                idx -= ic;
            }
            dj->clust = clst;
            dj->sect = clust2sect(dj->fs, clst) + idx / (SS(dj->fs) / SZ_DIR);
        /* Sector# */
        }

        dj->dir = dj->fs->win + (idx % (SS(dj->fs) / SZ_DIR)) * SZ_DIR;    /* Ptr to the entry in the sector */

        return FR_OK;       /* Seek succeeded */
}
```

```
/*-----------------------------------------------------------------------*/
/* Directory handling - Move directory index next                        */
/*-----------------------------------------------------------------------*/

static
FRESULT dir_next (      /* FR_OK:Succeeded, FR_NO_FILE:End of table, FR_DENIED:EOT and could not stretch */
        DIRS *dj,           /* Pointer to directory object */
        int stretch         /* 0: Do not stretch table, 1: Stretch table if needed */
)
{
        DWORD clst;
        WORD i;


        i = dj->index + 1;
        if (!i || !dj->sect)    /* Report EOT when index has reached 65535 */
                return FR_NO_FILE;

        if (!(i % (SS(dj->fs) / SZ_DIR))) {     /* Sector changed? */
                dj->sect++;                             /* Next sector */

                if (dj->clust == 0) {       /* Static table */
                        if (i >= dj->fs->n_rootdir)       /* Report EOT when end of table */
                                return FR_NO_FILE;
                }
                else {                                  /* Dynamic table */
                        if (((i / (SS(dj->fs) / SZ_DIR)) & (dj->fs->csize - 1)) == 0) {
        /* Cluster changed? */
                                clst = get_fat(dj->fs, dj->clust);                /* Get next cluster */
                                if (clst <= 1) return FR_INT_ERR;
                                if (clst == 0xFFFFFFFF) return FR_DISK_ERR;
                                if (clst >= dj->fs->n_fatent) {
        /* When it reached end of dynamic table */
```

```
#if !_FS_READONLY
                                BYTE c;
                                if (!stretch) return FR_NO_FILE;
        /* When do not stretch, report EOT */
                                clst = create_chain(dj->fs, dj->clust);         /*
Stretch cluster chain */
                                if (clst == 0) return FR_DENIED;
        /* No free cluster */
                                if (clst == 1) return FR_INT_ERR;
                                if (clst == 0xFFFFFFFF) return FR_DISK_ERR;
                                /* Clean-up stretched table */
                                if        (move_window(dj->fs,        0))        return
FR_DISK_ERR;   /* Flush active window */
                                mem_set(dj->fs->win, 0, SS(dj->fs));
        /* Clear window buffer */
                                dj->fs->winsect = clust2sect(dj->fs, clst);    /*
Cluster start sector */
                                for (c = 0; c < dj->fs->csize; c++) {           /*   Fill
the new cluster with 0 */
                                        dj->fs->wflag = 1;
                                        if      (move_window(dj->fs,      0))      return
FR_DISK_ERR;
                                        dj->fs->winsect++;
                                }
                                dj->fs->winsect -= c;
        /* Rewind window address */
#else
                                return FR_NO_FILE;                      /*       Report
EOT */
#endif
                        }
                        dj->clust = clst;                       /* Initialize data for
new cluster */
                        dj->sect = clust2sect(dj->fs, clst);
                }
        }
    }
```

```
	dj->index = i;
	dj->dir = dj->fs->win + (i % (SS(dj->fs) / SZ_DIR)) * SZ_DIR;

	return FR_OK;
}



/*-----------------------------------------------------------------------*/
/* LFN handling - Test/Pick/Fit an LFN segment from/to directory entry   */
/*-----------------------------------------------------------------------*/
#if _USE_LFN
static
const BYTE LfnOfs[] = {1,3,5,7,9,14,16,18,20,22,24,28,30};  /* Offset of LFN chars in the directory entry */


static
int cmp_lfn (                   /* 1:Matched, 0:Not matched */
	WCHAR *lfnbuf,          /* Pointer to the LFN to be compared */
	BYTE *dir               /* Pointer to the directory entry containing a part of LFN */
)
{
	UINT i, s;
	WCHAR wc, uc;


	i = ((dir[LDIR_Ord] & ~LLE) - 1) * 13;	/* Get offset in the LFN buffer */
	s = 0; wc = 1;
	do {
		uc = LD_WORD(dir+LfnOfs[s]);	/* Pick an LFN character from the entry */
		if (wc) {	/* Last char has not been processed */
			wc = ff_wtoupper(uc);		/* Convert it to upper case */
```

```
                    if (i >= _MAX_LFN || wc != ff_wtoupper(lfnbuf[i++]))  /*
Compare it */
                        return 0;                       /* Not matched */
                } else {
                    if (uc != 0xFFFF) return 0;       /* Check filler */
                }
        } while (++s < 13);                             /* Repeat until all chars in the
entry are checked */

        if ((dir[LDIR_Ord] & LLE) && wc && lfnbuf[i])/* Last segment matched
but different length */
                return 0;

        return 1;                                       /* The part of LFN matched */
}



static
int pick_lfn (                  /* 1:Succeeded, 0:Buffer overflow */
        WCHAR *lfnbuf,          /* Pointer to the Unicode-LFN buffer */
        BYTE *dir               /* Pointer to the directory entry */
)
{
        UINT i, s;
        WCHAR wc, uc;


        i = ((dir[LDIR_Ord] & 0x3F) - 1) * 13;      /* Offset in the LFN buffer */

        s = 0; wc = 1;
        do {
                uc = LD_WORD(dir+LfnOfs[s]);            /* Pick an LFN character
from the entry */
                if (wc) {       /* Last char has not been processed */
                    if (i >= _MAX_LFN) return 0; /* Buffer overflow? */
                    lfnbuf[i++] = wc = uc;                  /* Store it */
```

```
            } else {
                if (uc != 0xFFFF) return 0;          /* Check filler */
            }
        } while (++s < 13);                          /* Read all character
in the entry */

        if (dir[LDIR_Ord] & LLE) {                   /* Put terminator if it is the
last LFN part */
            if (i >= _MAX_LFN) return 0;         /* Buffer overflow? */
            lfnbuf[i] = 0;
        }

        return 1;
}


#if !_FS_READONLY
static
void fit_lfn (
        const WCHAR *lfnbuf,  /* Pointer to the LFN buffer */
        BYTE *dir,                  /* Pointer to the directory entry */
        BYTE ord,                   /* LFN order (1-20) */
        BYTE sum                    /* SFN sum */
)
{
        UINT i, s;
        WCHAR wc;


        dir[LDIR_Chksum] = sum;                  /* Set check sum */
        dir[LDIR_Attr] = AM_LFN;            /* Set attribute. LFN entry */
        dir[LDIR_Type] = 0;
        ST_WORD(dir+LDIR_FstClusLO, 0);

        i = (ord - 1) * 13;                      /* Get offset in the LFN buffer */
        s = wc = 0;
        do {
```

```
            if (wc != 0xFFFF) wc = lfnbuf[i++];  /* Get an effective char */
            ST_WORD(dir+LfnOfs[s], wc);         /* Put it */
            if (!wc) wc = 0xFFFF;               /* Padding chars following last char */
        } while (++s < 13);
        if (wc == 0xFFFF || !lfnbuf[i]) ord |= LLE;  /* Bottom LFN part is the start of
LFN sequence */
        dir[LDIR_Ord] = ord;                    /* Set the LFN order */
}

#endif
#endif



/*-----------------------------------------------------------------------*/
/* Create numbered name                                                  */
/*-----------------------------------------------------------------------*/
#if _USE_LFN
void gen_numname (
        BYTE *dst,              /* Pointer to generated SFN */
        const BYTE *src,  /* Pointer to source SFN to be modified */
        const WCHAR *lfn,       /* Pointer to LFN */
        WORD seq                /* Sequence number */
)
{
        BYTE ns[8], c;
        UINT i, j;


        mem_cpy(dst, src, 11);

        if (seq > 5) {          /* On many collisions, generate a hash number instead of
sequential number */
            do seq = (seq >> 1) + (seq << 15) + (WORD)*lfn++; while (*lfn);
        }

        /* itoa (hexdecimal) */
```

```
        i = 7;
        do {
                c = (seq % 16) + '0';
                if (c > '9') c += 7;
                ns[i--] = c;
                seq /= 16;
        } while (seq);
        ns[i] = '~';

        /* Append the number */
        for (j = 0; j < i && dst[j] != ' '; j++) {
                if (IsDBCS1(dst[j])) {
                        if (j == i - 1) break;
                        j++;
                }
        }
        do {
                dst[j++] = (i < 8) ? ns[i++] : ' ';
        } while (j < 8);
}
#endif



/*-----------------------------------------------------------------------*/
/* Calculate sum of an SFN                                               */
/*-----------------------------------------------------------------------*/
#if _USE_LFN
static
BYTE sum_sfn (
        const BYTE *dir           /* Ptr to directory entry */
)
{
        BYTE sum = 0;
        UINT n = 11;
```

```
        do sum = (sum >> 1) + (sum << 7) + *dir++; while (--n);
        return sum;
}
#endif




/*-----------------------------------------------------------------------*/
/* Directory handling - Find an object in the directory                  */
/*-----------------------------------------------------------------------*/

static
FRESULT dir_find (
        DIRS *dj                    /* Pointer to the directory object linked to the file name */
)
{
        FRESULT res;
        BYTE c, *dir;
#if _USE_LFN
        BYTE a, ord, sum;
#endif

        res = dir_sdi(dj, 0);                       /* Rewind directory object */
        if (res != FR_OK) return res;

#if _USE_LFN
        ord = sum = 0xFF;
#endif
        do {
                res = move_window(dj->fs, dj->sect);
                if (res != FR_OK) break;
                dir = dj->dir;                                  /* Ptr to the directory entry of current index */
                c = dir[DIR_Name];
```

```
		if (c == 0) { res = FR_NO_FILE; break; }	/* Reached to end of table */
#if _USE_LFN	/* LFN configuration */
		a = dir[DIR_Attr] & AM_MASK;
		if (c == DDE || ((a & AM_VOL) && a != AM_LFN)) { /* An entry without valid data */
			ord = 0xFF;
		} else {
			if (a == AM_LFN) {			/* An LFN entry is found */
				if (dj->lfn) {
					if (c & LLE) {		/* Is it start of LFN sequence? */
						sum = dir[LDIR_Chksum];
						c &= ~LLE; ord = c;	/* LFN start order */
						dj->lfn_idx = dj->index;
					}
					/* Check validity of the LFN entry and compare it with given name */
					ord = (c == ord && sum == dir[LDIR_Chksum] && cmp_lfn(dj->lfn, dir)) ? ord - 1 : 0xFF;
				}
			} else {					/* An SFN entry is found */
				if (!ord && sum == sum_sfn(dir)) break;	/* LFN matched? */
				ord = 0xFF; dj->lfn_idx = 0xFFFF;	/* Reset LFN sequence */
				if (!(dj->fn[NS] & NS_LOSS) && !mem_cmp(dir, dj->fn, 11)) break;	/* SFN matched? */
			}
		}
#else		/* Non LFN configuration */
		if (!(dir[DIR_Attr] & AM_VOL) && !mem_cmp(dir, dj->fn, 11)) /* Is it a valid entry? */
			break;
#endif
```

```
		res = dir_next(dj, 0);		/* Next entry */
	} while (res == FR_OK);

	return res;
}




/*-----------------------------------------------------------------------*/
/* Read an object from the directory                                     */
/*-----------------------------------------------------------------------*/
#if _FS_MINIMIZE <= 1
static
FRESULT dir_read (
	DIRS *dj			/* Pointer to the directory object that pointing the entry to be read */
)
{
	FRESULT res;
	BYTE c, *dir;
#if _USE_LFN
	BYTE a, ord = 0xFF, sum = 0xFF;
#endif

	res = FR_NO_FILE;
	while (dj->sect) {
		res = move_window(dj->fs, dj->sect);
		if (res != FR_OK) break;
		dir = dj->dir;					/* Ptr to the directory entry of current index */
		c = dir[DIR_Name];
		if (c == 0) { res = FR_NO_FILE; break; }	/* Reached to end of table */
#if _USE_LFN	/* LFN configuration */
		a = dir[DIR_Attr] & AM_MASK;
```

```
            if (c == DDE || (!_FS_RPATH && c == '.') || ((a & AM_VOL) && a
!= AM_LFN)) {    /* An entry without valid data */
                ord = 0xFF;
            } else {
                if (a == AM_LFN) {                  /* An LFN entry is found
*/
                    if (c & LLE) {                  /* Is it start of LFN
sequence? */
                        sum = dir[LDIR_Chksum];
                        c &= ~LLE; ord = c;
                        dj->lfn_idx = dj->index;
                    }
                    /* Check LFN validity and capture it */
                    ord = (c == ord && sum == dir[LDIR_Chksum] &&
pick_lfn(dj->lfn, dir)) ? ord - 1 : 0xFF;
                } else {                            /* An SFN entry is found
*/
                    if (ord || sum != sum_sfn(dir))  /* Is there a valid LFN? */
                        dj->lfn_idx = 0xFFFF;           /* It has no LFN. */
                    break;
                }
            }
#else       /* Non LFN configuration */
            if (c != DDE && (_FS_RPATH || c != '.') && !(dir[DIR_Attr] &
AM_VOL)) /* Is it a valid entry? */
                break;
#endif
            res = dir_next(dj, 0);                  /* Next entry */
            if (res != FR_OK) break;
        }

        if (res != FR_OK) dj->sect = 0;

        return res;
}
#endif
```

```
/*-----------------------------------------------------------------------*/
/* Register an object to the directory                                   */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY
static
FRESULT dir_register ( /* FR_OK:Successful, FR_DENIED:No free entry or too many SFN collision, FR_DISK_ERR:Disk error */
        DIRS *dj                        /* Target directory with object name to be created */
)
{
        FRESULT res;
        BYTE c, *dir;
#if _USE_LFN     /* LFN configuration */
        WORD n, ne, is;
        BYTE sn[12], *fn, sum;
        WCHAR *lfn;


        fn = dj->fn; lfn = dj->lfn;
        mem_cpy(sn, fn, 12);

        if (_FS_RPATH && (sn[NS] & NS_DOT))        /* Cannot create dot entry */
                return FR_INVALID_NAME;

        if (sn[NS] & NS_LOSS) {                    /* When LFN is out of 8.3 format, generate a numbered name */
                fn[NS] = 0; dj->lfn = 0;           /* Find only SFN */
                for (n = 1; n < 100; n++) {
                        gen_numname(fn, sn, lfn, n);   /* Generate a numbered name */
                        res = dir_find(dj);            /* Check if the name collides with existing SFN */
                        if (res != FR_OK) break;
                }
```

```
            if (n == 100) return FR_DENIED;             /* Abort if too many collisions */
            if (res != FR_NO_FILE) return res;  /* Abort if the result is other than 'not collided' */
            fn[NS] = sn[NS]; dj->lfn = lfn;
        }

        if (sn[NS] & NS_LFN) {                  /* When LFN is to be created, reserve an SFN + LFN entries. */
            for (ne = 0; lfn[ne]; ne++) ;
            ne = (ne + 25) / 13;
        } else {                                        /* Otherwise reserve only an SFN entry. */
            ne = 1;
        }

        /* Reserve contiguous entries */
        res = dir_sdi(dj, 0);
        if (res != FR_OK) return res;
        n = is = 0;
        do {
            res = move_window(dj->fs, dj->sect);
            if (res != FR_OK) break;
            c = *dj->dir;                               /* Check the entry status */
            if (c == DDE || c == 0) { /* Is it a blank entry? */
                if (n == 0) is = dj->index;         /* First index of the contiguous entry */
                if (++n == ne) break;       /* A contiguous entry that required count is found */
            } else {
                n = 0;                                  /* Not a blank entry. Restart to search */
            }
            res = dir_next(dj, 1);                  /* Next entry with table stretch */
        } while (res == FR_OK);

        if (res == FR_OK && ne > 1) {           /* Initialize LFN entry if needed */
```

```
            res = dir_sdi(dj, is);
            if (res == FR_OK) {
                sum = sum_sfn(dj->fn);  /* Sum of the SFN tied to the LFN */
                ne--;
                do {                            /* Store LFN entries in bottom first */
                    res = move_window(dj->fs, dj->sect);
                    if (res != FR_OK) break;
                    fit_lfn(dj->lfn, dj->dir, (BYTE)ne, sum);
                    dj->fs->wflag = 1;
                    res = dir_next(dj, 0);      /* Next entry */
                } while (res == FR_OK && --ne);
            }
        }

#else  /* Non LFN configuration */
        res = dir_sdi(dj, 0);
        if (res == FR_OK) {
            do {   /* Find a blank entry for the SFN */
                res = move_window(dj->fs, dj->sect);
                if (res != FR_OK) break;
                c = *dj->dir;
                if (c == DDE || c == 0) break;   /* Is it a blank entry? */
                res = dir_next(dj, 1);                  /* Next entry with table stretch */
            } while (res == FR_OK);
        }
#endif

        if (res == FR_OK) {             /* Initialize the SFN entry */
            res = move_window(dj->fs, dj->sect);
            if (res == FR_OK) {
                dir = dj->dir;
                mem_set(dir, 0, SZ_DIR);        /* Clean the entry */
                mem_cpy(dir, dj->fn, 11);       /* Put SFN */
#if _USE_LFN
```

```
                        dir[DIR_NTres] = *(dj->fn+NS) & (NS_BODY | NS_EXT);
        /* Put NT flag */
#endif
                        dj->fs->wflag = 1;
                }
        }

        return res;
}
#endif /* !_FS_READONLY */




/*-----------------------------------------------------------------------*/
/* Remove an object from the directory                                   */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY && !_FS_MINIMIZE
static
FRESULT dir_remove ( /* FR_OK: Successful, FR_DISK_ERR: A disk error */
        DIRS *dj                        /* Directory object pointing the entry to be removed */
)
{
        FRESULT res;
#if _USE_LFN     /* LFN configuration */
        WORD i;

        i = dj->index;        /* SFN index */
        res = dir_sdi(dj, (WORD)((dj->lfn_idx == 0xFFFF) ? i : dj->lfn_idx));
        /* Goto the SFN or top of the LFN entries */
        if (res == FR_OK) {
                do {
                        res = move_window(dj->fs, dj->sect);
                        if (res != FR_OK) break;
                        *dj->dir = DDE;                 /* Mark the entry "deleted" */
                        dj->fs->wflag = 1;
```

```
                        if (dj->index >= i) break;/* When reached SFN, all entries of the
object has been deleted. */
                        res = dir_next(dj, 0);              /* Next entry */
                } while (res == FR_OK);
                if (res == FR_NO_FILE) res = FR_INT_ERR;
        }

#else                   /* Non LFN configuration */
        res = dir_sdi(dj, dj->index);
        if (res == FR_OK) {
                res = move_window(dj->fs, dj->sect);
                if (res == FR_OK) {
                        *dj->dir = DDE;                 /* Mark the entry "deleted" */
                        dj->fs->wflag = 1;
                }
        }
#endif

        return res;
}
#endif /* !_FS_READONLY */




/*-----------------------------------------------------------------------*/
/* Pick a segment and create the object name in directory form           */
/*-----------------------------------------------------------------------*/

static
FRESULT create_name (
        DIRS *dj,                       /* Pointer to the directory object */
        const TCHAR **path      /* Pointer to pointer to the segment in the path string
*/
)
{
#ifdef _EXCVT
```

```
	static const BYTE excvt[] = _EXCVT;		/* Upper conversion table for extended chars */
#endif

#if _USE_LFN	/* LFN configuration */
	BYTE b, cf;
	WCHAR w, *lfn;
	UINT i, ni, si, di;
	const TCHAR *p;

	/* Create LFN in Unicode */
	for (p = *path; *p == '/' || *p == '\\'; p++) ;	/* Strip duplicated separator */
	lfn = dj->lfn;
	si = di = 0;
	for (;;) {
		w = p[si++];					/* Get a character */
		if (w < ' ' || w == '/' || w == '\\') break;	/* Break on end of segment */
		if (di >= _MAX_LFN)				/* Reject too long name */
			return FR_INVALID_NAME;
#if !_LFN_UNICODE
		w &= 0xFF;
		if (IsDBCS1(w)) {				/* Check if it is a DBC 1st byte (always false on SBCS cfg) */
			b = (BYTE)p[si++];			/* Get 2nd byte */
			if (!IsDBCS2(b))
				return FR_INVALID_NAME;	/* Reject invalid sequence */
			w = (w << 8) + b;			/* Create a DBC */
		}
		w = ff_convert(w, 1);			/* Convert ANSI/OEM to Unicode */
		if (!w) return FR_INVALID_NAME;	/* Reject invalid code */
#endif
		if (w < 0x80 && chk_chr("\"*:<>\?|\x7F", w)) /* Reject illegal chars for LFN */
			return FR_INVALID_NAME;
```

```
            lfn[di++] = w;                      /* Store the Unicode char
*/
        }
        *path = &p[si];                         /* Return pointer to the
next segment */
        cf = (w < ' ') ? NS_LAST : 0;           /* Set last segment flag if end of path */
#if _FS_RPATH
        if ((di == 1 && lfn[di-1] == '.') || /* Is this a dot entry? */
            (di == 2 && lfn[di-1] == '.' && lfn[di-2] == '.')) {
            lfn[di] = 0;
            for (i = 0; i < 11; i++)
                dj->fn[i] = (i < di) ? '.' : ' ';
            dj->fn[i] = cf | NS_DOT;        /* This is a dot entry */
            return FR_OK;
        }
#endif
        while (di) {                            /* Strip trailing spaces and dots
*/
            w = lfn[di-1];
            if (w != ' ' && w != '.') break;
            di--;
        }
        if (!di) return FR_INVALID_NAME;/* Reject nul string */

        lfn[di] = 0;                            /* LFN is created */

        /* Create SFN in directory form */
        mem_set(dj->fn, ' ', 11);
        for (si = 0; lfn[si] == ' ' || lfn[si] == '.'; si++) ;      /* Strip leading spaces and
dots */
        if (si) cf |= NS_LOSS | NS_LFN;
        while (di && lfn[di - 1] != '.') di--;  /* Find extension (di<=si: no
extension) */

        b = i = 0; ni = 8;
        for (;;) {
            w = lfn[si++];                      /* Get an LFN char */
```

```
		if (!w) break;					/* Break on end of the LFN */
		if (w == ' ' || (w == '.' && si != di)) {	/* Remove spaces and dots */
			cf |= NS_LOSS | NS_LFN; continue;
		}

		if (i >= ni || si == di) {		/* Extension or end of SFN */
			if (ni == 11) {				/* Long extension */
				cf |= NS_LOSS | NS_LFN; break;
			}
			if (si != di) cf |= NS_LOSS | NS_LFN;	/* Out of 8.3 format */
			if (si > di) break;			/* No extension */
			si = di; i = 8; ni = 11;	/* Enter extension section */
			b <<= 2; continue;
		}

		if (w >= 0x80) {				/* Non ASCII char */
#ifdef _EXCVT
			w = ff_convert(w, 0);		/* Unicode -> OEM code */
			if (w) w = excvt[w - 0x80];	/* Convert extended char to upper (SBCS) */
#else
			w = ff_convert(ff_wtoupper(w), 0);	/* Upper converted Unicode -> OEM code */
#endif
			cf |= NS_LFN;				/* Force create LFN entry */
		}

		if (_DF1S && w >= 0x100) {		/* Double byte char (always false on SBCS cfg) */
			if (i >= ni - 1) {
				cf |= NS_LOSS | NS_LFN; i = ni; continue;
			}
			dj->fn[i++] = (BYTE)(w >> 8);
		} else {					/* Single byte char */
```

```
                    if (!w || chk_chr("+,;=[]", w)) { /* Replace illegal chars for SFN */
                        w = '_'; cf |= NS_LOSS | NS_LFN;/* Lossy conversion */
                    } else {
                        if (IsUpper(w)) {           /* ASCII large capital */
                            b |= 2;
                        } else {
                            if (IsLower(w)) {   /* ASCII small capital */
                                b |= 1; w -= 0x20;
                            }
                        }
                    }
                }
                dj->fn[i++] = (BYTE)w;
        }

        if (dj->fn[0] == DDE) dj->fn[0] = NDDE;   /* If the first char collides with deleted mark, replace it with 0x05 */

        if (ni == 8) b <<= 2;
        if ((b & 0x0C) == 0x0C || (b & 0x03) == 0x03)     /* Create LFN entry when there are composite capitals */
                cf |= NS_LFN;
        if (!(cf & NS_LFN)) {                                   /* When LFN is in 8.3 format without extended char, NT flags are created */
                if ((b & 0x03) == 0x01) cf |= NS_EXT;       /* NT flag (Extension has only small capital) */
                if ((b & 0x0C) == 0x04) cf |= NS_BODY;   /* NT flag (Filename has only small capital) */
        }

        dj->fn[NS] = cf;    /* SFN is created */

        return FR_OK;


#else  /* Non-LFN configuration */
```

```
	BYTE b, c, d, *sfn;
	UINT ni, si, i;
	const char *p;

	/* Create file name in directory form */
	for (p = *path; *p == '/' || *p == '\\'; p++) ;	/* Strip duplicated separator */
	sfn = dj->fn;
	mem_set(sfn, ' ', 11);
	si = i = b = 0; ni = 8;
#if _FS_RPATH
	if (p[si] == '.') { /* Is this a dot entry? */
		for (;;) {
			c = (BYTE)p[si++];
			if (c != '.' || si >= 3) break;
			sfn[i++] = c;
		}
		if (c != '/' && c != '\\' && c > ' ') return FR_INVALID_NAME;
		*path = &p[si];
	/* Return pointer to the next segment */
		sfn[NS] = (c <= ' ') ? NS_LAST | NS_DOT : NS_DOT;  /*  Set  last
segment flag if end of path */
		return FR_OK;
	}
#endif
	for (;;) {
		c = (BYTE)p[si++];
		if (c <= ' ' || c == '/' || c == '\\') break;   /* Break on end of segment */
		if (c == '.' || i >= ni) {
			if (ni != 8 || c != '.') return FR_INVALID_NAME;
			i = 8; ni = 11;
			b <<= 2; continue;
		}
		if (c >= 0x80) {				/* Extended char? */
			b |= 3;					/* Eliminate NT flag
*/
#ifdef _EXCVT
			c = excvt[c-0x80];		/* Upper conversion (SBCS) */
```

```
#else
#if !_DF1S  /* ASCII only cfg */
                return FR_INVALID_NAME;
#endif
#endif
            }
            if (IsDBCS1(c)) {                   /* Check if it is a DBC 1st byte (always false on SBCS cfg) */
                d = (BYTE)p[si++];                  /* Get 2nd byte */
                if (!IsDBCS2(d) || i >= ni - 1)   /* Reject invalid DBC */
                    return FR_INVALID_NAME;
                sfn[i++] = c;
                sfn[i++] = d;
            } else {                                /* Single byte code */
                if (chk_chr("\"*+,:;<=>\?[]|\x7F", c)) /* Reject illegal chrs for SFN */
                    return FR_INVALID_NAME;
                if (IsUpper(c)) {               /* ASCII large capital? */
                    b |= 2;
                } else {
                    if (IsLower(c)) {           /* ASCII small capital? */
                        b |= 1; c -= 0x20;
                    }
                }
                sfn[i++] = c;
            }
        }
        *path = &p[si];                             /* Return pointer to the next segment */
        c = (c <= ' ') ? NS_LAST : 0;           /* Set last segment flag if end of path */

        if (!i) return FR_INVALID_NAME;         /* Reject nul string */
        if (sfn[0] == DDE) sfn[0] = NDDE;   /* When first char collides with DDE, replace it with 0x05 */

        if (ni == 8) b <<= 2;
```

```
	if ((b & 0x03) == 0x01) c |= NS_EXT;	/* NT flag (Name extension has only small capital) */
	if ((b & 0x0C) == 0x04) c |= NS_BODY;	/* NT flag (Name body has only small capital) */

	sfn[NS] = c;		/* Store NT flag, File name is created */

	return FR_OK;
#endif
}




/*-----------------------------------------------------------------------*/
/* Get file information from directory entry                             */
/*-----------------------------------------------------------------------*/
#if _FS_MINIMIZE <= 1
static
void get_fileinfo (		/* No return code */
	DIRS *dj,			/* Pointer to the directory object */
	FILINFO *fno		/* Pointer to the file information to be filled */
)
{
	UINT i;
	BYTE nt, *dir;
	TCHAR *p, c;


	p = fno->fname;
	if (dj->sect) {
		dir = dj->dir;
		nt = dir[DIR_NTres];		/* NT flag */
		for (i = 0; i < 8; i++) {	/* Copy name body */
			c = dir[i];
			if (c == ' ') break;
			if (c == NDDE) c = (TCHAR)DDE;
```

```
                    if (_USE_LFN && (nt & NS_BODY) && IsUpper(c)) c +=
0x20;
#if _LFN_UNICODE
                    if (IsDBCS1(c) && i < 7 && IsDBCS2(dir[i+1]))
                        c = (c << 8) | dir[++i];
                    c = ff_convert(c, 1);
                    if (!c) c = '?';
#endif
                    *p++ = c;
                }
                if (dir[8] != ' ') {            /* Copy name extension */
                    *p++ = '.';
                    for (i = 8; i < 11; i++) {
                        c = dir[i];
                        if (c == ' ') break;
                        if (_USE_LFN && (nt & NS_EXT) && IsUpper(c)) c +=
0x20;
#if _LFN_UNICODE
                        if (IsDBCS1(c) && i < 10 && IsDBCS2(dir[i+1]))
                            c = (c << 8) | dir[++i];
                        c = ff_convert(c, 1);
                        if (!c) c = '?';
#endif
                        *p++ = c;
                    }
                }
                fno->fattrib = dir[DIR_Attr];                   /* Attribute */
                fno->fsize = LD_DWORD(dir+DIR_FileSize);    /* Size */
                fno->fdate = LD_WORD(dir+DIR_WrtDate);          /* Date */
                fno->ftime = LD_WORD(dir+DIR_WrtTime);          /* Time */
            }
            *p = 0;             /* Terminate SFN str by a \0 */

#if _USE_LFN
            if (fno->lfname && fno->lfsize) {
                TCHAR *tp = fno->lfname;
                WCHAR w, *lfn;
```

```
            i = 0;
            if (dj->sect && dj->lfn_idx != 0xFFFF) {/* Get LFN if available */
                lfn = dj->lfn;
                while ((w = *lfn++) != 0) {                 /* Get an LFN char */
#if !_LFN_UNICODE
                    w = ff_convert(w, 0);                   /* Unicode -> OEM conversion */
                    if (!w) { i = 0; break; }               /* Could not convert, no LFN */
                    if (_DF1S && w >= 0x100)                /* Put 1st byte if it is a DBC (always false on SBCS cfg) */
                        tp[i++] = (TCHAR)(w >> 8);
#endif
                    if (i >= fno->lfsize - 1) { i = 0; break; }     /* Buffer overflow, no LFN */
                    tp[i++] = (TCHAR)w;
                }
            }
            tp[i] = 0;      /* Terminate the LFN str by a \0 */
        }
#endif
}
#endif /* _FS_MINIMIZE <= 1 */




/*-----------------------------------------------------------------------*/
/* Follow a file path                                                    */
/*-----------------------------------------------------------------------*/

static
FRESULT follow_path (/* FR_OK(0): successful, !=0: error code */
        DIRS *dj,                   /* Directory object to return last directory and found object */
```

```
        const TCHAR *path       /* Full-path string to find a file or directory */
)
{
        FRESULT res;
        BYTE *dir, ns;


#if _FS_RPATH
        if (*path == '/' || *path == '\\') { /* There is a heading separator */
                path++;       dj->sclust = 0;               /* Strip it and start from the root
dir */
        } else {                                            /* No heading separator */
                dj->sclust = dj->fs->cdir;         /* Start from the current dir */
        }
#else
        if (*path == '/' || *path == '\\')    /* Strip heading separator if exist */
                path++;
        dj->sclust = 0;                                     /* Start from the root dir */
#endif

        if ((UINT)*path < ' ') {                  /* Nul path means the start directory
itself */
                res = dir_sdi(dj, 0);
                dj->dir = 0;

        } else {                                            /* Follow path */
                for (;;) {
                        res = create_name(dj, &path);  /* Get a segment */
                        if (res != FR_OK) break;
                        res = dir_find(dj);                 /* Find it */
                        ns = *(dj->fn+NS);
                        if (res != FR_OK) {                     /* Failed to find the
object */
                                if (res != FR_NO_FILE) break;/* Abort if any hard error
occured */
                                /* Object not found */
```

```
                        if (_FS_RPATH && (ns & NS_DOT)) {   /* If dot entry is not exit */
                            dj->sclust = 0; dj->dir = 0;    /* It is the root dir */
                            res = FR_OK;
                            if (!(ns & NS_LAST)) continue;
                        } else {                            /* Could not find the object */
                            if (!(ns & NS_LAST)) res = FR_NO_PATH;
                        }
                        break;
                    }
                    if (ns & NS_LAST) break;            /* Last segment match. Function completed. */
                    dir = dj->dir;                      /* There is next segment. Follow the sub directory */
                    if (!(dir[DIR_Attr] & AM_DIR)) {    /* Cannot follow because it is a file */
                        res = FR_NO_PATH; break;
                    }
                    dj->sclust = LD_CLUST(dir);
                }
            }

            return res;
}




/*-----------------------------------------------------------------------*/
/* Load boot record and check if it is an FAT boot record                */
/*-----------------------------------------------------------------------*/

static
BYTE check_fs (  /* 0:The FAT BR, 1:Valid BR but not an FAT, 2:Not a BR, 3:Disk error */
    FATFS *fs, /* File system object */
```

```
	DWORD sect	/* Sector# (lba) to check if it is an FAT boot record or not */
)
{
	if (disk_read(fs->drv, fs->win, sect, 1) != RES_OK)	/* Load boot record */
		return 3;
	if (LD_WORD(&fs->win[BS_55AA]) != 0xAA55)		/* Check record signature (always placed at offset 510 even if the sector size is >512) */
		return 2;

	if ((LD_DWORD(&fs->win[BS_FilSysType]) & 0xFFFFFF) == 0x544146)	/* Check "FAT" string */
		return 0;
	if ((LD_DWORD(&fs->win[BS_FilSysType32]) & 0xFFFFFF) == 0x544146)
		return 0;

	return 1;
}



/*-----------------------------------------------------------------------*/
/* Check if the file system object is valid or not                       */
/*-----------------------------------------------------------------------*/

static
FRESULT chk_mounted (	/* FR_OK(0): successful, !=0: any error occurred */
	const TCHAR **path,	/* Pointer to pointer to the path name (drive number) */
	FATFS **rfs,		/* Pointer to pointer to the found file system object */
	BYTE chk_wp			/* !=0: Check media write protection for write access */
)
```

```
{
	BYTE fmt, b, *tbl;
	UINT vol;
	DSTATUS stat;
	DWORD bsect, fasize, tsect, sysect, nclst, szbfat;
	WORD nrsv;
	const TCHAR *p = *path;
	FATFS *fs;

	/* Get logical drive number from the path name */
	vol = p[0] - '0';			/* Is there a drive number? */
	if (vol <= 9 && p[1] == ':') {		/* Found a drive number, get and strip it */
		p += 2; *path = p;		/* Return pointer to the path name */
	} else {				/* No drive number is given */
#if _FS_RPATH
		vol = CurrVol;			/* Use current drive */
#else
		vol = 0;			/* Use drive 0 */
#endif
	}

	/* Check if the logical drive is valid or not */
	if (vol >= _VOLUMES)			/* Is the drive number valid? */
		return FR_INVALID_DRIVE;
	*rfs = fs = FatFs[vol];			/* Return pointer to the corresponding file system object */
	if (!fs) return FR_NOT_ENABLED;		/* Is the file system object available? */

	ENTER_FF(fs);				/* Lock file system */

	if (fs->fs_type) {			/* If the logical drive has been mounted */
		stat = disk_status(fs->drv);
```

```
            if (!(stat & STA_NOINIT)) {         /* and the physical drive is kept initialized (has not been changed), */
#if !_FS_READONLY
                if (chk_wp && (stat & STA_PROTECT))  /* Check write protection if needed */
                    return FR_WRITE_PROTECTED;
#endif
                return FR_OK;                   /* The file system object is valid */
            }
        }

        /* The logical drive must be mounted. */
        /* Following code attempts to mount a volume. (analyze BPB and initialize the fs object) */

        fs->fs_type = 0;                        /* Clear the file system object */
        fs->drv = (BYTE)LD2PD(vol);             /* Bind the logical drive and a physical drive */
        stat = disk_initialize(fs->drv);  /* Initialize low level disk I/O layer */
        if (stat & STA_NOINIT)                  /* Check if the initialization succeeded */
            return FR_NOT_READY;                /* Failed to initialize due to no media or hard error */
#if _MAX_SS != 512                              /* Get disk sector size (variable sector size cfg only) */
        if (disk_ioctl(fs->drv, GET_SECTOR_SIZE, &fs->ssize) != RES_OK)
            return FR_DISK_ERR;
#endif
#if !_FS_READONLY
        if (chk_wp && (stat & STA_PROTECT))  /* Check disk write protection if needed */
            return FR_WRITE_PROTECTED;
#endif
        /* Search FAT partition on the drive. Supports only generic partitionings, FDISK and SFD. */
        fmt = check_fs(fs, bsect = 0);          /* Check sector 0 if it is a VBR */
```

```
	if (fmt == 1) {				/* Not an FAT-VBR, the disk may be partitioned */
		/* Check the partition listed in top of the partition table */
		tbl = &fs->win[MBR_Table + LD2PT(vol) * SZ_PTE];/* Partition table */
		if (tbl[4]) {				/* Is the partition existing? */
			bsect = LD_DWORD(&tbl[8]);
	/* Partition offset in LBA */
			fmt = check_fs(fs, bsect);		/* Check the partition */
		}
	}
	if (fmt == 3) return FR_DISK_ERR;
	if (fmt) return FR_NO_FILESYSTEM;		/* No FAT volume is found */

	/* Following code initializes the file system object */

	if (LD_WORD(fs->win+BPB_BytsPerSec) != SS(fs))	/* (BPB_BytsPerSec must be equal to the physical sector size) */
		return FR_NO_FILESYSTEM;

	fasize = LD_WORD(fs->win+BPB_FATSz16);		/* Number of sectors per FAT */
	if (!fasize) fasize = LD_DWORD(fs->win+BPB_FATSz32);
	fs->fsize = fasize;

	fs->n_fats = b = fs->win[BPB_NumFATs];		/* Number of FAT copies */
	if (b != 1 && b != 2) return FR_NO_FILESYSTEM;	/* (Must be 1 or 2) */
	fasize *= b;					/* Number of sectors for FAT area */

	fs->csize = b = fs->win[BPB_SecPerClus];	/* Number of sectors per cluster */
```

```
	if (!b || (b & (b - 1))) return FR_NO_FILESYSTEM;	/* (Must be power of 2) */

	fs->n_rootdir = LD_WORD(fs->win+BPB_RootEntCnt);	/* Number of root directory entries */
	if (fs->n_rootdir % (SS(fs) / SZ_DIR)) return FR_NO_FILESYSTEM;	/* (BPB_RootEntCnt must be sector aligned) */

	tsect = LD_WORD(fs->win+BPB_TotSec16);			/* Number of sectors on the volume */
	if (!tsect) tsect = LD_DWORD(fs->win+BPB_TotSec32);

	nrsv = LD_WORD(fs->win+BPB_RsvdSecCnt);			/* Number of reserved sectors */
	if (!nrsv) return FR_NO_FILESYSTEM;			/* (BPB_RsvdSecCnt must not be 0) */

	/* Determine the FAT sub type */
	sysect = nrsv + fasize + fs->n_rootdir / (SS(fs) / SZ_DIR);	/* RSV+FAT+DIRS */
	if (tsect < sysect) return FR_NO_FILESYSTEM;	/* (Invalid volume size) */
	nclst = (tsect - sysect) / fs->csize;			/* Number of clusters */
	if (!nclst) return FR_NO_FILESYSTEM;			/* (Invalid volume size) */
	fmt = FS_FAT12;
	if (nclst >= MIN_FAT16) fmt = FS_FAT16;
	if (nclst >= MIN_FAT32) fmt = FS_FAT32;

	/* Boundaries and Limits */
	fs->n_fatent = nclst + 2;				/* Number of FAT entries */
	fs->database = bsect + sysect;				/* Data start sector */
	fs->fatbase = bsect + nrsv;				/* FAT start sector */
```

```
	if (fmt == FS_FAT32) {
		if (fs->n_rootdir) return FR_NO_FILESYSTEM;		/* (BPB_RootEntCnt must be 0) */
		fs->dirbase = LD_DWORD(fs->win+BPB_RootClus);	/* Root directory start cluster */
		szbfat = fs->n_fatent * 4;				/* (Required FAT size) */
	} else {
		if (!fs->n_rootdir) return FR_NO_FILESYSTEM;/* (BPB_RootEntCnt must not be 0) */
		fs->dirbase = fs->fatbase + fasize;			/* Root directory start sector */
		szbfat = (fmt == FS_FAT16) ?				/* (Required FAT size) */
			fs->n_fatent * 2 : fs->n_fatent * 3 / 2 + (fs->n_fatent & 1);
	}
	if (fs->fsize < (szbfat + (SS(fs) - 1)) / SS(fs))	/* (BPB_FATSz must not be less than required) */
		return FR_NO_FILESYSTEM;

#if !_FS_READONLY
	/* Initialize cluster allocation information */
	fs->free_clust = 0xFFFFFFFF;
	fs->last_clust = 0;

	/* Get fsinfo if available */
	if (fmt == FS_FAT32) {
		fs->fsi_flag = 0;
		fs->fsi_sector = bsect + LD_WORD(fs->win+BPB_FSInfo);
		if (disk_read(fs->drv, fs->win, fs->fsi_sector, 1) == RES_OK &&
			LD_WORD(fs->win+BS_55AA) == 0xAA55 &&
			LD_DWORD(fs->win+FSI_LeadSig) == 0x41615252 &&
			LD_DWORD(fs->win+FSI_StrucSig) == 0x61417272) {
				fs->last_clust = LD_DWORD(fs->win+FSI_Nxt_Free);
				fs->free_clust = LD_DWORD(fs->win+FSI_Free_Count);
		}
```

```
        }
#endif
        fs->fs_type = fmt;          /* FAT sub-type */
        fs->id = ++Fsid;            /* File system mount ID */
        fs->winsect = 0;            /* Invalidate sector cache */
        fs->wflag = 0;
#if _FS_RPATH
        fs->cdir = 0;               /* Current directory (root dir) */
#endif
#if _FS_SHARE                       /* Clear file lock semaphores */
        clear_lock(fs);
#endif

        return FR_OK;
}




/*-----------------------------------------------------------------------*/
/* Check if the file/dir object is valid or not                          */
/*-----------------------------------------------------------------------*/

static
FRESULT validate (      /* FR_OK(0): The object is valid, !=0: Invalid */
        FATFS *fs,          /* Pointer to the file system object */
        WORD id             /* Member id of the target object to be checked */
)
{
        if (!fs || !fs->fs_type || fs->id != id)
                return FR_INVALID_OBJECT;

        ENTER_FF(fs);           /* Lock file system */

        if (disk_status(fs->drv) & STA_NOINIT)
                return FR_NOT_READY;
```

```
	return FR_OK;
}



/*----------------------------------------------------------------------

  Public Functions

----------------------------------------------------------------------*/



/*----------------------------------------------------------------------*/
/* Mount/Unmount a Logical Drive                                        */
/*----------------------------------------------------------------------*/

FRESULT f_mount (
	BYTE vol,		/* Logical drive number to be mounted/unmounted */
	FATFS *fs		/* Pointer to new file system object (NULL for unmount)*/
)
{
	FATFS *rfs;


	if (vol >= _VOLUMES)			/* Check if the drive number is valid */
		return FR_INVALID_DRIVE;
	rfs = FatFs[vol];				/* Get current fs object */

	if (rfs) {
#if _FS_SHARE
		clear_lock(rfs);
#endif
#if _FS_REENTRANT					/* Discard sync object of the current volume */
		if (!ff_del_syncobj(rfs->sobj)) return FR_INT_ERR;
```

```
#endif
		rfs->fs_type = 0;		/* Clear old fs object */
	}

	if (fs) {
		fs->fs_type = 0;		/* Clear new fs object */
#if _FS_REENTRANT				/* Create sync object for the new volume */
		if (!ff_cre_syncobj(vol, &fs->sobj)) return FR_INT_ERR;
#endif
	}
	FatFs[vol] = fs;			/* Register new fs object */

	return FR_OK;
}



/*-----------------------------------------------------------------------*/
/* Open or Create a File                                                 */
/*-----------------------------------------------------------------------*/

FRESULT f_open (
	FIL *fp,			/* Pointer to the blank file object */
	const TCHAR *path,	/* Pointer to the file name */
	BYTE mode			/* Access mode and file open mode flags */
)
{
	FRESULT res;
	DIRS dj;
	BYTE *dir;
	DEF_NAMEBUF;


	fp->fs = 0;			/* Clear file object */
```

```
#if !_FS_READONLY
	mode &= FA_READ | FA_WRITE | FA_CREATE_ALWAYS | FA_OPEN_ALWAYS | FA_CREATE_NEW;
	res = chk_mounted(&path, &dj.fs, (BYTE)(mode & ~FA_READ));
#else
	mode &= FA_READ;
	res = chk_mounted(&path, &dj.fs, 0);
#endif
	INIT_BUF(dj);
	if (res == FR_OK)
		res = follow_path(&dj, path);	/* Follow the file path */
	dir = dj.dir;

#if !_FS_READONLY	/* R/W configuration */
	if (res == FR_OK) {
		if (!dir)	/* Current dir itself */
			res = FR_INVALID_NAME;
#if _FS_SHARE
		else
			res = chk_lock(&dj, (mode & ~FA_READ) ? 1 : 0);
#endif
	}
	/* Create or Open a file */
	if (mode & (FA_CREATE_ALWAYS | FA_OPEN_ALWAYS | FA_CREATE_NEW)) {
		DWORD dw, cl;

		if (res != FR_OK) {					/* No file, create new */
			if (res == FR_NO_FILE)			/* There is no file to open, create a new entry */
#if _FS_SHARE
				res = enq_lock(dj.fs) ? dir_register(&dj) : FR_TOO_MANY_OPEN_FILES;
#else
				res = dir_register(&dj);
#endif
```

```
                mode |= FA_CREATE_ALWAYS;        /* File is created */
                dir = dj.dir;                   /* New entry */
            }
            else {                              /* Any object is already existing */
                if (dir[DIR_Attr] & (AM_RDO | AM_DIR)) {    /* Cannot overwrite it (R/O or DIRS) */
                    res = FR_DENIED;
                } else {
                    if (mode & FA_CREATE_NEW)   /* Cannot create as new file */
                        res = FR_EXIST;
                }
            }
            if (res == FR_OK && (mode & FA_CREATE_ALWAYS)) {  /* Truncate it if overwrite mode */
                dw = get_fattime();                 /* Created time */
                ST_DWORD(dir+DIR_CrtTime, dw);
                dir[DIR_Attr] = 0;                  /* Reset attribute */
                ST_DWORD(dir+DIR_FileSize, 0);      /* size = 0 */
                cl = LD_CLUST(dir);                 /* Get start cluster */
                ST_CLUST(dir, 0);                   /* cluster = 0 */
                dj.fs->wflag = 1;
                if (cl) {                           /* Remove the cluster chain if exist */
                    dw = dj.fs->winsect;
                    res = remove_chain(dj.fs, cl);
                    if (res == FR_OK) {
                        dj.fs->last_clust = cl - 1; /* Reuse the cluster hole */
                        res = move_window(dj.fs, dw);
                    }
                }
            }
        }
```

```
        else { /* Open an existing file */
                if (res == FR_OK) {                                     /*      Follow
succeeded */
                        if (dir[DIR_Attr] & AM_DIR) {                   /* It is a directory */
                                res = FR_NO_FILE;
                        } else {
                                if ((mode & FA_WRITE) && (dir[DIR_Attr] &
AM_RDO)) /* R/O violation */
                                        res = FR_DENIED;
                        }
                }
        }
        if (res == FR_OK) {
                if (mode & FA_CREATE_ALWAYS)                            /*      Set     file
change flag if created or overwritten */
                        mode |= FA__WRITTEN;
                fp->dir_sect = dj.fs->winsect;                  /* Pointer to the directory
entry */
                fp->dir_ptr = dir;
#if _FS_SHARE
                fp->lockid = inc_lock(&dj, (mode & ~FA_READ) ? 1 : 0);
                if (!fp->lockid) res = FR_INT_ERR;
#endif
        }

#else                           /* R/O configuration */
        if (res == FR_OK) {                                     /* Follow succeeded */
                if (!dir) {                                     /* Current dir itself */
                        res = FR_INVALID_NAME;
                } else {
                        if (dir[DIR_Attr] & AM_DIR)  /* It is a directory */
                                res = FR_NO_FILE;
                }
        }
#endif
        FREE_BUF();
```

```
	if (res == FR_OK) {
		fp->flag = mode;				/* File access mode */
		fp->sclust = LD_CLUST(dir);			/* File start cluster */
		fp->fsize = LD_DWORD(dir+DIR_FileSize);	/* File size */
		fp->fptr = 0;					/* File pointer */
		fp->dsect = 0;
#if _USE_FASTSEEK
		fp->cltbl = 0;					/* Normal seek mode */
#endif
		fp->fs = dj.fs; fp->id = dj.fs->id;	/* Validate file object */
	}

	LEAVE_FF(dj.fs, res);
}




/*-----------------------------------------------------------------------*/
/* Read File                                                             */
/*-----------------------------------------------------------------------*/

FRESULT f_read (
	FIL *fp,		/* Pointer to the file object */
	void *buff,		/* Pointer to data buffer */
	UINT btr,		/* Number of bytes to read */
	UINT *br		/* Pointer to number of bytes read */
)
{
	FRESULT res;
	DWORD clst, sect, remain;
	UINT rcnt, cc;
	BYTE csect, *rbuff = buff;


	*br = 0;	/* Initialize byte counter */
```

```
	res = validate(fp->fs, fp->id);					/* Check validity */
	if (res != FR_OK) LEAVE_FF(fp->fs, res);
	if (fp->flag & FA__ERROR)					/* Aborted file? */
		LEAVE_FF(fp->fs, FR_INT_ERR);
	if (!(fp->flag & FA_READ))					/* Check access mode */
		LEAVE_FF(fp->fs, FR_DENIED);
	remain = fp->fsize - fp->fptr;
	if (btr > remain) btr = (UINT)remain;		/* Truncate btr by remaining bytes */

	for ( ;  btr;							/* Repeat until all data read */
		rbuff += rcnt, fp->fptr += rcnt, *br += rcnt, btr -= rcnt) {
		if ((fp->fptr % SS(fp->fs)) == 0) {		/* On the sector boundary? */
			csect = (BYTE)(fp->fptr / SS(fp->fs) & (fp->fs->csize - 1));	/* Sector offset in the cluster */
			if (!csect) {						/* On the cluster boundary? */
				if (fp->fptr == 0) {		/* On the top of the file? */
					clst = fp->sclust;			/* Follow from the origin */
				} else {						/* Middle or end of the file */
#if _USE_FASTSEEK
					if (fp->cltbl)
						clst = clmt_clust(fp, fp->fptr);	/* Get cluster# from the CLMT */
					else
#endif
						clst = get_fat(fp->fs, fp->clust);		/* Follow cluster chain on the FAT */
				}
				if (clst < 2) ABORT(fp->fs, FR_INT_ERR);
```

```
                        if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR_DISK_ERR);
                        fp->clust = clst;                   /* Update current cluster */
                    }
                    sect = clust2sect(fp->fs, fp->clust);   /* Get current sector */
                    if (!sect) ABORT(fp->fs, FR_INT_ERR);
                    sect += csect;
                    cc = btr / SS(fp->fs);                  /* When remaining bytes >= sector size, */
                    if (cc) {                               /* Read maximum contiguous sectors directly */
                        if (csect + cc > fp->fs->csize)     /* Clip at cluster boundary */
                            cc = fp->fs->csize - csect;
                        if (disk_read(fp->fs->drv, rbuff, sect, (BYTE)cc) != RES_OK)
                            ABORT(fp->fs, FR_DISK_ERR);
#if !_FS_READONLY && _FS_MINIMIZE <= 2                      /* Replace one of the read sectors with cached data if it contains a dirty sector */
#if _FS_TINY
                        if (fp->fs->wflag && fp->fs->winsect - sect < cc)
                            mem_cpy(rbuff + ((fp->fs->winsect - sect) * SS(fp->fs)), fp->fs->win, SS(fp->fs));
#else
                        if ((fp->flag & FA__DIRTY) && fp->dsect - sect < cc)
                            mem_cpy(rbuff + ((fp->dsect - sect) * SS(fp->fs)), fp->buf, SS(fp->fs));
#endif
#endif
                        rcnt = SS(fp->fs) * cc;             /* Number of bytes transferred */
                        continue;
                    }
#if !_FS_TINY
                    if (fp->dsect != sect) {                /* Load data sector if not in cache */
```

```
#if !_FS_READONLY
                        if (fp->flag & FA__DIRTY) {          /* Write-back dirty sector cache */
                            if (disk_write(fp->fs->drv, fp->buf, fp->dsect, 1) != RES_OK)
                                ABORT(fp->fs, FR_DISK_ERR);
                            fp->flag &= ~FA__DIRTY;
                        }
#endif
                        if (disk_read(fp->fs->drv, fp->buf, sect, 1) != RES_OK)      /* Fill sector cache */
                            ABORT(fp->fs, FR_DISK_ERR);
                    }
#endif
                    fp->dsect = sect;
                }
                rcnt = SS(fp->fs) - (fp->fptr % SS(fp->fs));  /* Get partial sector data from sector buffer */
                if (rcnt > btr) rcnt = btr;
#if _FS_TINY
                if (move_window(fp->fs, fp->dsect))          /* Move sector window */
                    ABORT(fp->fs, FR_DISK_ERR);
                mem_cpy(rbuff, &fp->fs->win[fp->fptr % SS(fp->fs)], rcnt);   /* Pick partial sector */
#else
                mem_cpy(rbuff, &fp->buf[fp->fptr % SS(fp->fs)], rcnt);/* Pick partial sector */
#endif
            }

            LEAVE_FF(fp->fs, FR_OK);
}




#if !_FS_READONLY
```

```
/*-----------------------------------------------------------------------*/
/* Write File                                                            */
/*-----------------------------------------------------------------------*/

FRESULT f_write (
        FIL *fp,                    /* Pointer to the file object */
        const void *buff,   /* Pointer to the data to be written */
        UINT btw,                   /* Number of bytes to write */
        UINT *bw                    /* Pointer to number of bytes written */
)
{
        FRESULT res;
        DWORD clst, sect;
        UINT wcnt, cc;
        const BYTE *wbuff = buff;
        BYTE csect;


        *bw = 0;      /* Initialize byte counter */

        res = validate(fp->fs, fp->id);                 /* Check validity */
        if (res != FR_OK) LEAVE_FF(fp->fs, res);
        if (fp->flag & FA__ERROR)                       /* Aborted file? */
                LEAVE_FF(fp->fs, FR_INT_ERR);
        if (!(fp->flag & FA_WRITE))                     /* Check access mode */
                LEAVE_FF(fp->fs, FR_DENIED);
        if ((DWORD)(fp->fsize + btw) < fp->fsize) btw = 0;      /* File size cannot
reach 4GB */

        for ( ;  btw;                                   /* Repeat until all data
written */
                wbuff += wcnt, fp->fptr += wcnt, *bw += wcnt, btw -= wcnt) {
                if ((fp->fptr % SS(fp->fs)) == 0) {      /* On the sector boundary? */
                        csect = (BYTE)(fp->fptr / SS(fp->fs) & (fp->fs->csize - 1));
        /* Sector offset in the cluster */
                        if (!csect) {                   /* On the cluster
boundary? */
```

```
                        if (fp->fptr == 0) {        /* On the top of the file? */
                            clst = fp->sclust;          /* Follow from the origin
*/
                            if (clst == 0)              /* When no cluster is
allocated, */
                                fp->sclust = clst = create_chain(fp->fs, 0);
        /* Create a new cluster chain */
                        } else {                                /* Middle or end of
the file */
#if _USE_FASTSEEK
                            if (fp->cltbl)
                                clst = clmt_clust(fp, fp->fptr);  /*          Get
cluster# from the CLMT */
                            else
#endif
                                clst = create_chain(fp->fs, fp->clust); /*
Follow or stretch cluster chain on the FAT */
                        }
                        if (clst == 0) break;               /* Could not allocate a
new cluster (disk full) */
                        if (clst == 1) ABORT(fp->fs, FR_INT_ERR);
                        if      (clst       ==      0xFFFFFFFF)     ABORT(fp->fs,
FR_DISK_ERR);
                        fp->clust = clst;                   /* Update current cluster
*/
                    }
#if _FS_TINY
                    if (fp->fs->winsect == fp->dsect && move_window(fp->fs, 0))
        /* Write-back sector cache */
                        ABORT(fp->fs, FR_DISK_ERR);
#else
                    if (fp->flag & FA__DIRTY) {         /* Write-back sector cache
*/
                        if  (disk_write(fp->fs->drv,  fp->buf,  fp->dsect,  1)  !=
RES_OK)
                            ABORT(fp->fs, FR_DISK_ERR);
                        fp->flag &= ~FA__DIRTY;
```

```
                }
#endif
                sect = clust2sect(fp->fs, fp->clust);    /* Get current sector */
                if (!sect) ABORT(fp->fs, FR_INT_ERR);
                sect += csect;
                cc = btw / SS(fp->fs);                   /* When remaining bytes
>= sector size, */
                if (cc) {                                /* Write maximum
contiguous sectors directly */
                    if (csect + cc > fp->fs->csize)   /* Clip at cluster boundary
*/
                        cc = fp->fs->csize - csect;
                    if  (disk_write(fp->fs->drv,  wbuff,  sect,  (BYTE)cc)  !=
RES_OK)
                        ABORT(fp->fs, FR_DISK_ERR);
#if _FS_TINY
                    if (fp->fs->winsect - sect < cc) {       /*  Refill  sector
cache if it gets invalidated by the direct write */
                        mem_cpy(fp->fs->win, wbuff + ((fp->fs->winsect -
sect) * SS(fp->fs)), SS(fp->fs));
                        fp->fs->wflag = 0;
                    }
#else
                    if (fp->dsect - sect < cc) { /* Refill sector cache if it gets
invalidated by the direct write */
                        mem_cpy(fp->buf,  wbuff  +  ((fp->dsect  -  sect)  *
SS(fp->fs)), SS(fp->fs));
                        fp->flag &= ~FA__DIRTY;
                    }
#endif
                    wcnt = SS(fp->fs) * cc;          /*  Number  of  bytes
transferred */
                    continue;
                }
#if _FS_TINY
                if (fp->fptr >= fp->fsize) {       /* Avoid silly cache filling at
growing edge */
```

```
                        if (move_window(fp->fs, 0)) ABORT(fp->fs, FR_DISK_ERR);
                        fp->fs->winsect = sect;
                    }
#else
                    if (fp->dsect != sect) {            /* Fill sector cache with file data */
                        if (fp->fptr < fp->fsize &&
                            disk_read(fp->fs->drv, fp->buf, sect, 1) != RES_OK)
                                ABORT(fp->fs, FR_DISK_ERR);
                    }
#endif
                    fp->dsect = sect;
                }
                wcnt = SS(fp->fs) - (fp->fptr % SS(fp->fs));/* Put partial sector into file I/O buffer */
                if (wcnt > btw) wcnt = btw;
#if _FS_TINY
                if (move_window(fp->fs, fp->dsect)) /* Move sector window */
                    ABORT(fp->fs, FR_DISK_ERR);
                mem_cpy(&fp->fs->win[fp->fptr % SS(fp->fs)], wbuff, wcnt);/* Fit partial sector */
                fp->fs->wflag = 1;
#else
                mem_cpy(&fp->buf[fp->fptr % SS(fp->fs)], wbuff, wcnt);  /* Fit partial sector */
                fp->flag |= FA__DIRTY;
#endif
        }

        if (fp->fptr > fp->fsize) fp->fsize = fp->fptr;     /* Update file size if needed */
        fp->flag |= FA__WRITTEN;                            /* Set file change flag */

        LEAVE_FF(fp->fs, FR_OK);
```

```
}



/*-----------------------------------------------------------------------*/
/* Synchronize the File Object                                           */
/*-----------------------------------------------------------------------*/

FRESULT f_sync (
        FIL *fp             /* Pointer to the file object */
)
{
        FRESULT res;
        DWORD tim;
        BYTE *dir;


        res = validate(fp->fs, fp->id);             /* Check validity of the object */
        if (res == FR_OK) {
                if (fp->flag & FA__WRITTEN) {       /* Has the file been written? */
#if !_FS_TINY       /* Write-back dirty buffer */
                        if (fp->flag & FA__DIRTY) {
                                if   (disk_write(fp->fs->drv,   fp->buf,   fp->dsect,   1)   !=
RES_OK)
                                        LEAVE_FF(fp->fs, FR_DISK_ERR);
                                fp->flag &= ~FA__DIRTY;
                        }
#endif
                        /* Update the directory entry */
                        res = move_window(fp->fs, fp->dir_sect);
                        if (res == FR_OK) {
                                dir = fp->dir_ptr;
                                dir[DIR_Attr] |= AM_ARC;
        /* Set archive bit */
                                ST_DWORD(dir+DIR_FileSize, fp->fsize);          /*
Update file size */
```

```
                        ST_CLUST(dir, fp->sclust);
        /* Update start cluster */
                        tim = get_fattime();
        /* Update updated time */
                        ST_DWORD(dir+DIR_WrtTime, tim);
                        fp->flag &= ~FA__WRITTEN;
                        fp->fs->wflag = 1;
                        res = sync(fp->fs);
                    }
                }
        }

        LEAVE_FF(fp->fs, res);
}

#endif /* !_FS_READONLY */




/*-----------------------------------------------------------------------*/
/* Close File                                                            */
/*-----------------------------------------------------------------------*/

FRESULT f_close (
        FIL *fp             /* Pointer to the file object to be closed */
)
{
        FRESULT res;

#if _FS_READONLY
        FATFS *fs = fp->fs;
        res = validate(fs, fp->id);
        if (res == FR_OK) fp->fs = 0;   /* Discard file object */
        LEAVE_FF(fs, res);

#else
```

```
	res = f_sync(fp);		/* Flush cached data */
#if _FS_SHARE
	if (res == FR_OK) {		/* Decrement open counter */
#if _FS_REENTRANT
		res = validate(fp->fs, fp->id);
		if (res == FR_OK) {
			res = dec_lock(fp->lockid);
			unlock_fs(fp->fs, FR_OK);
		}
#else
		res = dec_lock(fp->lockid);
#endif
	}
#endif
	if (res == FR_OK) fp->fs = 0;  /* Discard file object */
	return res;
#endif
}




/*-----------------------------------------------------------------------*/
/* Current Drive/Directory Handlings                                     */
/*-----------------------------------------------------------------------*/

#if _FS_RPATH >= 1

FRESULT f_chdrive (
	BYTE drv		/* Drive number */
)
{
	if (drv >= _VOLUMES) return FR_INVALID_DRIVE;

	CurrVol = drv;

	return FR_OK;
```

```
}



FRESULT f_chdir (
	const TCHAR *path	/* Pointer to the directory path */
)
{
	FRESULT res;
	DIRS dj;
	DEF_NAMEBUF;


	res = chk_mounted(&path, &dj.fs, 0);
	if (res == FR_OK) {
		INIT_BUF(dj);
		res = follow_path(&dj, path);		/* Follow the path */
		FREE_BUF();
		if (res == FR_OK) {					/* Follow completed */
			if (!dj.dir) {
				dj.fs->cdir = dj.sclust;	/* Start directory itself */
			} else {
				if (dj.dir[DIR_Attr] & AM_DIR)	/* Reached to the directory */
					dj.fs->cdir = LD_CLUST(dj.dir);
				else
					res = FR_NO_PATH;		/* Reached but a file */
			}
		}
		if (res == FR_NO_FILE) res = FR_NO_PATH;
	}

	LEAVE_FF(dj.fs, res);
}
```

```
#if _FS_RPATH >= 2
FRESULT f_getcwd (
	TCHAR *path,	/* Pointer to the directory path */
	UINT sz_path	/* Size of path */
)
{
	FRESULT res;
	DIRS dj;
	UINT i, n;
	DWORD ccl;
	TCHAR *tp;
	FILINFO fno;
	DEF_NAMEBUF;


	*path = 0;
	res = chk_mounted((const TCHAR**)&path, &dj.fs, 0); /* Get current volume */
	if (res == FR_OK) {
		INIT_BUF(dj);
		i = sz_path;		/* Bottom of buffer (dir stack base) */
		dj.sclust = dj.fs->cdir;			/* Start to follow upper dir from current dir */
		while ((ccl = dj.sclust) != 0) {	/* Repeat while current dir is a sub-dir */
			res = dir_sdi(&dj, 1);			/* Get parent dir */
			if (res != FR_OK) break;
			res = dir_read(&dj);
			if (res != FR_OK) break;
			dj.sclust = LD_CLUST(dj.dir); /* Goto parent dir */
			res = dir_sdi(&dj, 0);
			if (res != FR_OK) break;
			do {					/* Find the entry links to the child dir */
				res = dir_read(&dj);
				if (res != FR_OK) break;
```

```
                        if (ccl == LD_CLUST(dj.dir)) break; /* Found the entry
*/
                        res = dir_next(&dj, 0);
                } while (res == FR_OK);
                if (res == FR_NO_FILE) res = FR_INT_ERR;/* It cannot be 'not
found'. */
                if (res != FR_OK) break;
#if _USE_LFN
                fno.lfname = path;
                fno.lfsize = i;
#endif
                get_fileinfo(&dj, &fno);        /* Get the dir name and push it to
the buffer */
                tp = fno.fname;
                if (_USE_LFN && *path) tp = path;
                for (n = 0; tp[n]; n++) ;
                if (i < n + 3) {
                        res = FR_NOT_ENOUGH_CORE; break;
                }
                while (n) path[--i] = tp[--n];
                path[--i] = '/';
        }
        tp = path;
        if (res == FR_OK) {
                *tp++ = '0' + CurrVol;                  /* Put drive number */
                *tp++ = ':';
                if (i == sz_path) {                     /* Root-dir */
                        *tp++ = '/';
                } else {                                        /* Sub-dir */
                        do              /* Add stacked path str */
                                *tp++ = path[i++];
                        while (i < sz_path);
                }
        }
        *tp = 0;
        FREE_BUF();
}
```

```
	LEAVE_FF(dj.fs, res);
}
#endif /* _FS_RPATH >= 2 */
#endif /* _FS_RPATH >= 1 */



#if _FS_MINIMIZE <= 2
/*-----------------------------------------------------------------------*/
/* Seek File R/W Pointer                                                 */
/*-----------------------------------------------------------------------*/

FRESULT f_lseek (
	FIL *fp,		/* Pointer to the file object */
	DWORD ofs		/* File pointer from top of file */
)
{
	FRESULT res;


	res = validate(fp->fs, fp->id);		/* Check validity of the object */
	if (res != FR_OK) LEAVE_FF(fp->fs, res);
	if (fp->flag & FA__ERROR)			/* Check abort flag */
		LEAVE_FF(fp->fs, FR_INT_ERR);

#if _USE_FASTSEEK
	if (fp->cltbl) {	/* Fast seek */
		DWORD cl, pcl, ncl, tcl, dsc, tlen, ulen, *tbl;

		if (ofs == CREATE_LINKMAP) {	/* Create CLMT */
			tbl = fp->cltbl;
			tlen = *tbl++; ulen = 2;	/* Given table size and required table size */
			cl = fp->sclust;			/* Top of the chain */
			if (cl) {
				do {
```

```
                        /* Get a fragment */
                        tcl = cl; ncl = 0; ulen += 2;       /* Top, length and
used items */
                        do {
                                pcl = cl; ncl++;
                                cl = get_fat(fp->fs, cl);
                                if (cl <= 1) ABORT(fp->fs, FR_INT_ERR);
                                if (cl == 0xFFFFFFFF) ABORT(fp->fs,
FR_DISK_ERR);
                        } while (cl == pcl + 1);
                        if (ulen <= tlen) {          /* Store the length and top
of the fragment */
                                *tbl++ = ncl; *tbl++ = tcl;
                        }
                } while (cl < fp->fs->n_fatent);/* Repeat until end of
chain */
                }
                *fp->cltbl = ulen;  /* Number of items used */
                if (ulen <= tlen)
                        *tbl = 0;               /* Terminate table */
                else
                        res = FR_NOT_ENOUGH_CORE;   /* Given table size
is smaller than required */

        } else {                                        /* Fast seek */
                if (ofs > fp->fsize)         /* Clip offset at the file size */
                        ofs = fp->fsize;
                fp->fptr = ofs;                         /* Set file pointer */
                if (ofs) {
                        fp->clust = clmt_clust(fp, ofs - 1);
                        dsc = clust2sect(fp->fs, fp->clust);
                        if (!dsc) ABORT(fp->fs, FR_INT_ERR);
                        dsc += (ofs - 1) / SS(fp->fs) & (fp->fs->csize - 1);
                        if (fp->fptr % SS(fp->fs) && dsc != fp->dsect) {   /*
Refill sector cache if needed */
#if !_FS_TINY
#if !_FS_READONLY
```

```
                                if (fp->flag & FA__DIRTY) {          /* Write-back
dirty sector cache */
                                        if    (disk_write(fp->fs->drv,    fp->buf,    fp-
>dsect, 1) != RES_OK)
                                                ABORT(fp->fs, FR_DISK_ERR);
                                        fp->flag &= ~FA__DIRTY;
                                }
#endif
                                if   (disk_read(fp->fs->drv,   fp->buf,   dsc,   1)   !=
RES_OK)   /* Load current sector */
                                        ABORT(fp->fs, FR_DISK_ERR);
#endif
                                fp->dsect = dsc;
                        }
                }
        }
    } else
#endif

    /* Normal Seek */
    {
        DWORD clst, bcs, nsect, ifptr;

        if (ofs > fp->fsize                             /* In read-only mode, clip
offset with the file size */
#if !_FS_READONLY
                && !(fp->flag & FA_WRITE)
#endif
                ) ofs = fp->fsize;

        ifptr = fp->fptr;
        fp->fptr = nsect = 0;
        if (ofs) {
            bcs = (DWORD)fp->fs->csize * SS(fp->fs);/*     Cluster     size
(byte) */
            if (ifptr > 0 &&
```

```
                        (ofs - 1) / bcs >= (ifptr - 1) / bcs) {     /* When seek to
same or following cluster, */
                        fp->fptr = (ifptr - 1) & ~(bcs - 1);      /* start from the
current cluster */
                        ofs -= fp->fptr;
                        clst = fp->clust;
                    } else {
        /* When seek to back cluster, */
                        clst = fp->sclust;                                  /* start
from the first cluster */
#if !_FS_READONLY
                        if (clst == 0) {                                    /* If no
cluster chain, create a new chain */
                            clst = create_chain(fp->fs, 0);
                            if (clst == 1) ABORT(fp->fs, FR_INT_ERR);
                            if (clst == 0xFFFFFFFF) ABORT(fp->fs,
FR_DISK_ERR);
                            fp->sclust = clst;
                        }
#endif
                        fp->clust = clst;
                    }
                    if (clst != 0) {
                        while (ofs > bcs) {                                 /*
Cluster following loop */
#if !_FS_READONLY
                            if (fp->flag & FA_WRITE) {                      /*
Check if in write mode or not */
                                clst = create_chain(fp->fs, clst);      /*
Force stretch if in write mode */
                                if (clst == 0) {                            /*
When disk gets full, clip file size */
                                    ofs = bcs; break;
                                }
                            } else
#endif
```

```
                                clst = get_fat(fp->fs, clst);       /* Follow
cluster chain if not in write mode */
                                if (clst == 0xFFFFFFFF) ABORT(fp->fs,
FR_DISK_ERR);
                                if (clst <= 1 || clst >= fp->fs->n_fatent) ABORT(fp-
>fs, FR_INT_ERR);
                                fp->clust = clst;
                                fp->fptr += bcs;
                                ofs -= bcs;
                        }
                        fp->fptr += ofs;
                        if (ofs % SS(fp->fs)) {
                                nsect = clust2sect(fp->fs, clst); /* Current sector */
                                if (!nsect) ABORT(fp->fs, FR_INT_ERR);
                                nsect += ofs / SS(fp->fs);
                        }
                }
        }
        if (fp->fptr % SS(fp->fs) && nsect != fp->dsect) {      /* Fill sector
cache if needed */
#if !_FS_TINY
#if !_FS_READONLY
                if (fp->flag & FA__DIRTY) {             /* Write-back dirty
sector cache */
                        if (disk_write(fp->fs->drv, fp->buf, fp->dsect, 1) !=
RES_OK)
                                ABORT(fp->fs, FR_DISK_ERR);
                        fp->flag &= ~FA__DIRTY;
                }
#endif
                if (disk_read(fp->fs->drv, fp->buf, nsect, 1) != RES_OK)
        /* Fill sector cache */
                        ABORT(fp->fs, FR_DISK_ERR);
#endif
                fp->dsect = nsect;
        }
#if !_FS_READONLY
```

```
			if (fp->fptr > fp->fsize) {		/* Set file change flag if the file size is extended */
				fp->fsize = fp->fptr;
				fp->flag |= FA__WRITTEN;
			}
#endif
		}

		LEAVE_FF(fp->fs, res);
}



#if _FS_MINIMIZE <= 1
/*-----------------------------------------------------------------------*/
/* Create a Directroy Object                                             */
/*-----------------------------------------------------------------------*/

FRESULT f_opendir (
		DIRS *dj,			/* Pointer to directory object to create */
		const TCHAR *path	/* Pointer to the directory path */
)
{
		FRESULT res;
		DEF_NAMEBUF;


		res = chk_mounted(&path, &dj->fs, 0);
		if (res == FR_OK) {
			INIT_BUF(*dj);
			res = follow_path(dj, path);		/* Follow the path to the directory */
			FREE_BUF();
			if (res == FR_OK) {					/* Follow completed */
				if (dj->dir) {					/* It is not the root dir */
```

```
                        if (dj->dir[DIR_Attr] & AM_DIR) {  /* The object is a
directory */
                            dj->sclust = LD_CLUST(dj->dir);
                        } else {                                    /* The object
is not a directory */
                            res = FR_NO_PATH;
                        }
                    }
                    if (res == FR_OK) {
                        dj->id = dj->fs->id;
                        res = dir_sdi(dj, 0);                   /* Rewind dir */
                    }
                }
                if (res == FR_NO_FILE) res = FR_NO_PATH;
            }

            LEAVE_FF(dj->fs, res);
}




/*-----------------------------------------------------------------------*/
/* Read Directory Entry in Sequense                                      */
/*-----------------------------------------------------------------------*/

FRESULT f_readdir (
    DIRS *dj,           /* Pointer to the open directory object */
    FILINFO *fno        /* Pointer to file information to return */
)
{
    FRESULT res;
    DEF_NAMEBUF;


    res = validate(dj->fs, dj->id);             /* Check validity of the object */
    if (res == FR_OK) {
```

```
            if (!fno) {
                res = dir_sdi(dj, 0);                   /* Rewind the directory
object */
            } else {
                INIT_BUF(*dj);
                res = dir_read(dj);                     /* Read an directory item
*/
                if (res == FR_NO_FILE) {                /* Reached end of dir */
                    dj->sect = 0;
                    res = FR_OK;
                }
                if (res == FR_OK) {                             /* A valid entry is
found */
                    get_fileinfo(dj, fno);              /* Get the object
information */
                    res = dir_next(dj, 0);              /* Increment index for
next */
                    if (res == FR_NO_FILE) {
                        dj->sect = 0;
                        res = FR_OK;
                    }
                }
                FREE_BUF();
            }
        }

        LEAVE_FF(dj->fs, res);
}



#if _FS_MINIMIZE == 0
/*-----------------------------------------------------------------------*/
/* Get File Status                                                       */
/*-----------------------------------------------------------------------*/

FRESULT f_stat (
```

```
    const TCHAR *path,      /* Pointer to the file path */
    FILINFO *fno            /* Pointer to file information to return */
)
{
    FRESULT res;
    DIRS dj;
    DEF_NAMEBUF;


    res = chk_mounted(&path, &dj.fs, 0);
    if (res == FR_OK) {
        INIT_BUF(dj);
        res = follow_path(&dj, path);   /* Follow the file path */
        if (res == FR_OK) {                         /* Follow completed */
            if (dj.dir)             /* Found an object */
                get_fileinfo(&dj, fno);
            else                    /* It is root dir */
                res = FR_INVALID_NAME;
        }
        FREE_BUF();
    }

    LEAVE_FF(dj.fs, res);
}



#if !_FS_READONLY
/*-----------------------------------------------------------------------*/
/* Get Number of Free Clusters                                           */
/*-----------------------------------------------------------------------*/

FRESULT f_getfree (
    const TCHAR *path,      /* Pointer to the logical drive number (root dir) */
    DWORD *nclst,           /* Pointer to the variable to return number of free clusters */
```

```
	FATFS **fatfs		/* Pointer to pointer to corresponding file system object to return */
)
{
	FRESULT res;
	DWORD n, clst, sect, stat;
	UINT i;
	BYTE fat, *p;


	/* Get drive number */
	res = chk_mounted(&path, fatfs, 0);
	if (res == FR_OK) {
		/* If free_clust is valid, return it without full cluster scan */
		if ((*fatfs)->free_clust <= (*fatfs)->n_fatent - 2) {
			*nclst = (*fatfs)->free_clust;
		} else {
			/* Get number of free clusters */
			fat = (*fatfs)->fs_type;
			n = 0;
			if (fat == FS_FAT12) {
				clst = 2;
				do {
					stat = get_fat(*fatfs, clst);
					if (stat == 0xFFFFFFFF) { res = FR_DISK_ERR; break; }
					if (stat == 1) { res = FR_INT_ERR; break; }
					if (stat == 0) n++;
				} while (++clst < (*fatfs)->n_fatent);
			} else {
				clst = (*fatfs)->n_fatent;
				sect = (*fatfs)->fatbase;
				i = 0; p = 0;
				do {
					if (!i) {
						res = move_window(*fatfs, sect++);
						if (res != FR_OK) break;
```

```
                                        p = (*fatfs)->win;
                                        i = SS(*fatfs);
                                    }
                                    if (fat == FS_FAT16) {
                                        if (LD_WORD(p) == 0) n++;
                                        p += 2; i -= 2;
                                    } else {
                                        if ((LD_DWORD(p) & 0x0FFFFFFF) == 0)
n++;
                                        p += 4; i -= 4;
                                    }
                                } while (--clst);
                            }
                            (*fatfs)->free_clust = n;
                            if (fat == FS_FAT32) (*fatfs)->fsi_flag = 1;
                            *nclst = n;
                        }
                }
                LEAVE_FF(*fatfs, res);
}




/*-----------------------------------------------------------------------*/
/* Truncate File                                                         */
/*-----------------------------------------------------------------------*/

FRESULT f_truncate (
        FIL *fp             /* Pointer to the file object */
)
{
        FRESULT res;
        DWORD ncl;


        res = validate(fp->fs, fp->id);         /* Check validity of the object */
```

```
	if (res == FR_OK) {
		if (fp->flag & FA__ERROR) {		/* Check abort flag */
			res = FR_INT_ERR;
		} else {
			if (!(fp->flag & FA_WRITE))		/* Check access mode */
				res = FR_DENIED;
		}
	}
	if (res == FR_OK) {
		if (fp->fsize > fp->fptr) {
			fp->fsize = fp->fptr;	/* Set file size to current R/W point */
			fp->flag |= FA__WRITTEN;
			if (fp->fptr == 0) {/* When set file size to zero, remove entire cluster chain */
				res = remove_chain(fp->fs, fp->sclust);
				fp->sclust = 0;
			} else {				/* When truncate a part of the file, remove remaining clusters */
				ncl = get_fat(fp->fs, fp->clust);
				res = FR_OK;
				if (ncl == 0xFFFFFFFF) res = FR_DISK_ERR;
				if (ncl == 1) res = FR_INT_ERR;
				if (res == FR_OK && ncl < fp->fs->n_fatent) {
					res = put_fat(fp->fs, fp->clust, 0x0FFFFFFF);
					if (res == FR_OK) res = remove_chain(fp->fs, ncl);
				}
			}
		}
		if (res != FR_OK) fp->flag |= FA__ERROR;
	}

	LEAVE_FF(fp->fs, res);
}
```

```
/*-----------------------------------------------------------------------*/
/* Delete a File or Directory                                            */
/*-----------------------------------------------------------------------*/

FRESULT f_unlink (
        const TCHAR *path               /* Pointer to the file or directory path */
)
{
        FRESULT res;
        DIRS dj, sdj;
        BYTE *dir;
        DWORD dclst;
        DEF_NAMEBUF;


        res = chk_mounted(&path, &dj.fs, 1);
        if (res == FR_OK) {
                INIT_BUF(dj);
                res = follow_path(&dj, path);           /* Follow the file path */
                if (_FS_RPATH && res == FR_OK && (dj.fn[NS] & NS_DOT))
                        res = FR_INVALID_NAME;                  /* Cannot remove
dot entry */
#if _FS_SHARE
                if (res == FR_OK) res = chk_lock(&dj, 2);  /* Cannot remove open
file */
#endif
                if (res == FR_OK) {                             /* The object is
accessible */
                        dir = dj.dir;
                        if (!dir) {
                                res = FR_INVALID_NAME;          /* Cannot remove
the start directory */
                        } else {
                                if (dir[DIR_Attr] & AM_RDO)
                                        res = FR_DENIED;        /* Cannot remove
R/O object */
                        }
```

```
                dclst = LD_CLUST(dir);
                if (res == FR_OK && (dir[DIR_Attr] & AM_DIR)) {    /* Is it a
sub-dir? */
                        if (dclst < 2) {
                                res = FR_INT_ERR;
                        } else {
                                mem_cpy(&sdj, &dj, sizeof(DIRS)); /* Check if
the sub-dir is empty or not */
                                sdj.sclust = dclst;
                                res = dir_sdi(&sdj, 2);             /* Exclude dot
entries */
                                if (res == FR_OK) {
                                        res = dir_read(&sdj);
                                        if (res == FR_OK                 /* Not empty
dir */
#if _FS_RPATH
                                                || dclst == sdj.fs->cdir     /* Current dir */
#endif
                                                ) res = FR_DENIED;
                                        if (res == FR_NO_FILE) res = FR_OK;
        /* Empty */
                                }
                        }
                }
                if (res == FR_OK) {
                        res = dir_remove(&dj);          /* Remove the directory
entry */
                        if (res == FR_OK) {
                                if (dclst)                          /* Remove the
cluster chain if exist */
                                        res = remove_chain(dj.fs, dclst);
                                if (res == FR_OK) res = sync(dj.fs);
                        }
                }
        }
        FREE_BUF();
    }
```

```
	LEAVE_FF(dj.fs, res);
}



/*-----------------------------------------------------------------------*/
/* Create a Directory                                                    */
/*-----------------------------------------------------------------------*/

FRESULT f_mkdir (
	const TCHAR *path		/* Pointer to the directory path */
)
{
	FRESULT res;
	DIRS dj;
	BYTE *dir, n;
	DWORD dsc, dcl, pcl, tim = get_fattime();
	DEF_NAMEBUF;


	res = chk_mounted(&path, &dj.fs, 1);
	if (res == FR_OK) {
		INIT_BUF(dj);
		res = follow_path(&dj, path);			/* Follow the file path */
		if (res == FR_OK) res = FR_EXIST;		/* Any object with same name is already existing */
		if (_FS_RPATH && res == FR_NO_FILE && (dj.fn[NS] & NS_DOT))
			res = FR_INVALID_NAME;
		if (res == FR_NO_FILE) {				/* Can create a new directory */
			dcl = create_chain(dj.fs, 0);		/* Allocate a cluster for the new directory table */
			res = FR_OK;
			if (dcl == 0) res = FR_DENIED;		/* No space to allocate a new cluster */
```

```
                if (dcl == 1) res = FR_INT_ERR;
                if (dcl == 0xFFFFFFFF) res = FR_DISK_ERR;
                if (res == FR_OK)                       /* Flush FAT */
                        res = move_window(dj.fs, 0);
                if (res == FR_OK) {                     /* Initialize
the new directory table */
                        dsc = clust2sect(dj.fs, dcl);
                        dir = dj.fs->win;
                        mem_set(dir, 0, SS(dj.fs));
                        mem_set(dir+DIR_Name, ' ', 8+3);    /* Create "." entry */
                        dir[DIR_Name] = '.';
                        dir[DIR_Attr] = AM_DIR;
                        ST_DWORD(dir+DIR_WrtTime, tim);
                        ST_CLUST(dir, dcl);
                        mem_cpy(dir+SZ_DIR, dir, SZ_DIR);       /* Create ".."
entry */
                        dir[33] = '.'; pcl = dj.sclust;
                        if (dj.fs->fs_type == FS_FAT32 && pcl == dj.fs->dirbase)
                                pcl = 0;
                        ST_CLUST(dir+SZ_DIR, pcl);
                        for (n = dj.fs->csize; n; n--) {    /* Write dot entries and
clear following sectors */
                                dj.fs->winsect = dsc++;
                                dj.fs->wflag = 1;
                                res = move_window(dj.fs, 0);
                                if (res != FR_OK) break;
                                mem_set(dir, 0, SS(dj.fs));
                        }
                }
                if (res == FR_OK) res = dir_register(&dj);  /* Register the
object to the directoy */
                if (res != FR_OK) {
                        remove_chain(dj.fs, dcl);           /* Could not
register, remove cluster chain */
                } else {
                        dir = dj.dir;
```

```
                        dir[DIR_Attr] = AM_DIR;                 /*
Attribute */
                        ST_DWORD(dir+DIR_WrtTime, tim);         /*
Created time */
                        ST_CLUST(dir, dcl);                     /*
Table start cluster */
                        dj.fs->wflag = 1;
                        res = sync(dj.fs);
                    }
                }
                FREE_BUF();
            }

            LEAVE_FF(dj.fs, res);
}




/*-----------------------------------------------------------------------*/
/* Change Attribute                                                      */
/*-----------------------------------------------------------------------*/

FRESULT f_chmod (
    const TCHAR *path,      /* Pointer to the file path */
    BYTE value,                 /* Attribute bits */
    BYTE mask               /* Attribute mask to change */
)
{
    FRESULT res;
    DIRS dj;
    BYTE *dir;
    DEF_NAMEBUF;


    res = chk_mounted(&path, &dj.fs, 1);
    if (res == FR_OK) {
```

```
		INIT_BUF(dj);
		res = follow_path(&dj, path);		/* Follow the file path */
		FREE_BUF();
		if (_FS_RPATH && res == FR_OK && (dj.fn[NS] & NS_DOT))
			res = FR_INVALID_NAME;
		if (res == FR_OK) {
			dir = dj.dir;
			if (!dir) {				/* Is it a root directory? */
				res = FR_INVALID_NAME;
			} else {				/* File or sub directory */
				mask &= AM_RDO|AM_HID|AM_SYS|AM_ARC;	/* Valid attribute mask */
				dir[DIR_Attr] = (value & mask) | (dir[DIR_Attr] & (BYTE)~mask);	/* Apply attribute change */
				dj.fs->wflag = 1;
				res = sync(dj.fs);
			}
		}
	}

	LEAVE_FF(dj.fs, res);
}




/*----------------------------------------------------------------------*/
/* Change Timestamp                                                     */
/*----------------------------------------------------------------------*/

FRESULT f_utime (
	const TCHAR *path,	/* Pointer to the file/directory name */
	const FILINFO *fno	/* Pointer to the time stamp to be set */
)
{
```

```
    FRESULT res;
    DIRS dj;
    BYTE *dir;
    DEF_NAMEBUF;


    res = chk_mounted(&path, &dj.fs, 1);
    if (res == FR_OK) {
        INIT_BUF(dj);
        res = follow_path(&dj, path);    /* Follow the file path */
        FREE_BUF();
        if (_FS_RPATH && res == FR_OK && (dj.fn[NS] & NS_DOT))
            res = FR_INVALID_NAME;
        if (res == FR_OK) {
            dir = dj.dir;
            if (!dir) {                              /* Root directory */
                res = FR_INVALID_NAME;
            } else {                                 /* File or sub-directory */
                ST_WORD(dir+DIR_WrtTime, fno->ftime);
                ST_WORD(dir+DIR_WrtDate, fno->fdate);
                dj.fs->wflag = 1;
                res = sync(dj.fs);
            }
        }
    }

    LEAVE_FF(dj.fs, res);
}




/*-----------------------------------------------------------------------*/
/* Rename File/Directory                                                 */
/*-----------------------------------------------------------------------*/

FRESULT f_rename (
```

```
        const TCHAR *path_old,      /* Pointer to the old name */
        const TCHAR *path_new       /* Pointer to the new name */
)
{
        FRESULT res;
        DIRS djo, djn;
        BYTE buf[21], *dir;
        DWORD dw;
        DEF_NAMEBUF;


        res = chk_mounted(&path_old, &djo.fs, 1);
        if (res == FR_OK) {
                djn.fs = djo.fs;
                INIT_BUF(djo);
                res = follow_path(&djo, path_old);          /* Check old object */
                if (_FS_RPATH && res == FR_OK && (djo.fn[NS] & NS_DOT))
                        res = FR_INVALID_NAME;
#if _FS_SHARE
                if (res == FR_OK) res = chk_lock(&djo, 2);
#endif
                if (res == FR_OK) {                                 /* Old object
is found */
                        if (!djo.dir) {                             /* Is root dir?
*/
                                res = FR_NO_FILE;
                        } else {
                                mem_cpy(buf, djo.dir+DIR_Attr, 21);         /* Save
the object information except for name */
                                mem_cpy(&djn, &djo, sizeof(DIRS));          /*
Check new object */
                                res = follow_path(&djn, path_new);
                                if (res == FR_OK) res = FR_EXIST;       /* The new
object name is already existing */
                                if (res == FR_NO_FILE) {                    /* Is it a
valid path and no name collision? */
/* Start critical section that any interruption or error can cause cross-link */
```

```
                                res = dir_register(&djn);           /* Register
the new entry */
                                if (res == FR_OK) {
                                        dir = djn.dir;
        /* Copy object information except for name */
                                        mem_cpy(dir+13, buf+2, 19);
                                        dir[DIR_Attr] = buf[0] | AM_ARC;
                                        djo.fs->wflag = 1;
                                        if (djo.sclust != djn.sclust && (dir[DIR_Attr]
& AM_DIR)) {            /* Update .. entry in the directory if needed */
                                                dw          =          clust2sect(djn.fs,
LD_CLUST(dir));
                                                if (!dw) {
                                                        res = FR_INT_ERR;
                                                } else {
                                                        res = move_window(djn.fs, dw);
                                                        dir    =    djn.fs->win+SZ_DIR;
        /* .. entry */
                                                        if (res == FR_OK && dir[1] ==
'.') {
                                                                dw = (djn.fs->fs_type ==
FS_FAT32 && djn.sclust == djn.fs->dirbase) ? 0 : djn.sclust;
                                                                ST_CLUST(dir, dw);
                                                                djn.fs->wflag = 1;
                                                        }
                                                }
                                        }
                                        if (res == FR_OK) {
                                                res = dir_remove(&djo);         /*
Remove old entry */
                                                if (res == FR_OK)
                                                        res = sync(djo.fs);
                                        }
                                }
/* End critical section */
                        }
                }
```

```
            }
            FREE_BUF();
        }
        LEAVE_FF(djo.fs, res);
}

#endif /* !_FS_READONLY */
#endif /* _FS_MINIMIZE == 0 */
#endif /* _FS_MINIMIZE <= 1 */
#endif /* _FS_MINIMIZE <= 2 */



/*-----------------------------------------------------------------------*/
/* Forward data to the stream directly (available on only tiny cfg)      */
/*-----------------------------------------------------------------------*/
#if _USE_FORWARD && _FS_TINY

FRESULT f_forward (
        FIL *fp,                                /* Pointer to the file object */
        UINT (*func)(const BYTE*,UINT),  /* Pointer to the streaming function */
        UINT btr,                               /* Number of bytes to forward */
        UINT *bf                                /* Pointer to number of bytes
forwarded */
)
{
        FRESULT res;
        DWORD remain, clst, sect;
        UINT rcnt;
        BYTE csect;


        *bf = 0;        /* Initialize byte counter */

        res = validate(fp->fs, fp->id);                 /* Check validity of
the object */
        if (res != FR_OK) LEAVE_FF(fp->fs, res);
```

```
	if (fp->flag & FA__ERROR)			/* Check error flag */
		LEAVE_FF(fp->fs, FR_INT_ERR);
	if (!(fp->flag & FA_READ))			/* Check access mode */
		LEAVE_FF(fp->fs, FR_DENIED);

	remain = fp->fsize - fp->fptr;
	if (btr > remain) btr = (UINT)remain;		/* Truncate btr by remaining bytes */

	for ( ;  btr && (*func)(0, 0);			/* Repeat until all data transferred or stream becomes busy */
		fp->fptr += rcnt, *bf += rcnt, btr -= rcnt) {
		csect = (BYTE)(fp->fptr / SS(fp->fs) & (fp->fs->csize - 1));	/* Sector offset in the cluster */
		if ((fp->fptr % SS(fp->fs)) == 0) {		/* On the sector boundary? */
			if (!csect) {				/* On the cluster boundary? */
				clst = (fp->fptr == 0) ?	/* On the top of the file? */
					fp->sclust : get_fat(fp->fs, fp->clust);
				if (clst <= 1) ABORT(fp->fs, FR_INT_ERR);
				if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR_DISK_ERR);
				fp->clust = clst;			/* Update current cluster */
			}
		}
		sect = clust2sect(fp->fs, fp->clust);		/* Get current data sector */
		if (!sect) ABORT(fp->fs, FR_INT_ERR);
		sect += csect;
		if (move_window(fp->fs, sect))			/* Move sector window */
			ABORT(fp->fs, FR_DISK_ERR);
```

```
            fp->dsect = sect;
            rcnt = SS(fp->fs) - (WORD)(fp->fptr % SS(fp->fs));      /*      Forward
data from sector window */
            if (rcnt > btr) rcnt = btr;
            rcnt = (*func)(&fp->fs->win[(WORD)fp->fptr % SS(fp->fs)], rcnt);
            if (!rcnt) ABORT(fp->fs, FR_INT_ERR);
        }

        LEAVE_FF(fp->fs, FR_OK);
}
#endif /* _USE_FORWARD */



#if _USE_MKFS && !_FS_READONLY
/*-----------------------------------------------------------------------*/
/* Create File System on the Drive                                       */
/*-----------------------------------------------------------------------*/
#define N_ROOTDIR    512        /* Number of root dir entries for FAT12/16 */
#define N_FATS       1          /* Number of FAT copies (1 or 2) */


FRESULT f_mkfs (
        BYTE drv,       /* Logical drive number */
        BYTE sfd,       /* Partitioning rule 0:FDISK, 1:SFD */
        UINT au                 /* Allocation unit size [bytes] */
)
{
        static const WORD vst[] = { 1024,   512,  256,  128,   64,    32,   16,    8,    4,
2,   0};
        static const WORD cst[] = {32768, 16384, 8192, 4096, 2048, 16384, 8192,
4096, 2048, 1024, 512};
        BYTE fmt, md, *tbl;
        DWORD n_clst, vs, n, wsect;
        UINT i;
        DWORD b_vol, b_fat, b_dir, b_data; /* Offset (LBA) */
        DWORD n_vol, n_rsv, n_fat, n_dir;   /* Size */
```

```
	FATFS *fs;
	DSTATUS stat;


	/* Check mounted drive and clear work area */
	if (drv >= _VOLUMES) return FR_INVALID_DRIVE;
	fs = FatFs[drv];
	if (!fs) return FR_NOT_ENABLED;
	fs->fs_type = 0;
	drv = LD2PD(drv);

	/* Get disk statics */
	stat = disk_initialize(drv);
	if (stat & STA_NOINIT) return FR_NOT_READY;
	if (stat & STA_PROTECT) return FR_WRITE_PROTECTED;
#if _MAX_SS != 512					/* Get disk sector size */
	if (disk_ioctl(drv, GET_SECTOR_SIZE, &SS(fs)) != RES_OK)
		return FR_DISK_ERR;
#endif
	if (disk_ioctl(drv, GET_SECTOR_COUNT, &n_vol) != RES_OK || n_vol < 128)
		return FR_DISK_ERR;
	b_vol = (sfd) ? 0 : 63;	/* Volume start sector */
	n_vol -= b_vol;
	if (au & (au - 1)) au = 0;	/* Check validity of the AU size */
	if (!au) {					/* AU auto selection */
		vs = n_vol / (2000 / (SS(fs) / 512));
		for (i = 0; vs < vst[i]; i++) ;
		au = cst[i];
	}
	au /= SS(fs);		/* Number of sectors per cluster */
	if (au == 0) au = 1;
	if (au > 128) au = 128;

	/* Pre-compute number of clusters and FAT syb-type */
	n_clst = n_vol / au;
	fmt = FS_FAT12;
```

```
        if (n_clst >= MIN_FAT16) fmt = FS_FAT16;
        if (n_clst >= MIN_FAT32) fmt = FS_FAT32;

        /* Determine offset and size of FAT structure */
        if (fmt == FS_FAT32) {
                n_fat = ((n_clst * 4) + 8 + SS(fs) - 1) / SS(fs);
                n_rsv = 32;
                n_dir = 0;
        } else {
                n_fat = (fmt == FS_FAT12) ? (n_clst * 3 + 1) / 2 + 3 : (n_clst * 2) + 4;
                n_fat = (n_fat + SS(fs) - 1) / SS(fs);
                n_rsv = 1;
                n_dir = (DWORD)N_ROOTDIR * SZ_DIR / SS(fs);
        }
        b_fat = b_vol + n_rsv;                  /* FAT area start sector */
        b_dir = b_fat + n_fat * N_FATS;         /* Directory area start sector */
        b_data = b_dir + n_dir;                 /* Data area start sector */
        if (n_vol < b_data + au) return FR_MKFS_ABORTED;  /* Too small volume
*/

        /* Align data start sector to erase block boundary (for flash memory media)
*/
        if (disk_ioctl(drv, GET_BLOCK_SIZE, &n) != RES_OK || !n || n > 32768) n
= 1;
        n = (b_data + n - 1) & ~(n - 1); /* Next nearest erase block from current data
start */
        n = (n - b_data) / N_FATS;
        if (fmt == FS_FAT32) {          /* FAT32: Move FAT offset */
                n_rsv += n;
                b_fat += n;
        } else {                                /* FAT12/16: Expand FAT size */
                n_fat += n;
        }

        /* Determine number of clusters and final check of validity of the FAT sub-
type */
        n_clst = (n_vol - n_rsv - n_fat * N_FATS - n_dir) / au;
```

```
	if (   (fmt == FS_FAT16 && n_clst < MIN_FAT16)
		|| (fmt == FS_FAT32 && n_clst < MIN_FAT32))
		return FR_MKFS_ABORTED;

	/* Create partition table if required */
	if (sfd) {	/* No patition table (SFD) */
		md = 0xF0;
	} else {	/* With patition table (FDISK) */
		DWORD n_disk = b_vol + n_vol;

		mem_set(fs->win, 0, SS(fs));
		tbl = fs->win+MBR_Table;
		ST_DWORD(tbl, 0x00010180);			/* Partition start in CHS */
		if (n_disk < 63UL * 255 * 1024) {	/* Partition end in CHS */
			n_disk = n_disk / 63 / 255;
			tbl[7] = (BYTE)n_disk;
			tbl[6] = (BYTE)((n_disk >> 2) | 63);
		} else {
			ST_WORD(&tbl[6], 0xFFFF); /* CHS saturated */
		}
		tbl[5] = 254;
		if (fmt != FS_FAT32)				/* System ID */
			tbl[4] = (n_vol < 0x10000) ? 0x04 : 0x06;
		else
			tbl[4] = 0x0c;
		ST_DWORD(tbl+8, 63);				/* Partition start in LBA */
		ST_DWORD(tbl+12, n_vol);			/* Partition size in LBA */
		ST_WORD(fs->win+BS_55AA, 0xAA55); /* MBR signature */
		if (disk_write(drv, fs->win, 0, 1) != RES_OK)	/* Put the MBR into first physical sector */
			return FR_DISK_ERR;
		md = 0xF8;
	}

	/* Create volume boot record */
	tbl = fs->win;							/* Clear sector */
```

```
        mem_set(tbl, 0, SS(fs));
        mem_cpy(tbl, "\xEB\xFE\x90" "MSDOS5.0", 11);/* Boot jump code, OEM name */
        i = SS(fs);                                 /* Sector size */
        ST_WORD(tbl+BPB_BytsPerSec, i);
        tbl[BPB_SecPerClus] = (BYTE)au;             /* Sectors per cluster */
        ST_WORD(tbl+BPB_RsvdSecCnt, n_rsv);         /* Reserved sectors */
        tbl[BPB_NumFATs] = N_FATS;                  /* Number of FATs */
        i = (fmt == FS_FAT32) ? 0 : N_ROOTDIR;/* Number of rootdir entries */
        ST_WORD(tbl+BPB_RootEntCnt, i);
        if (n_vol < 0x10000) {                      /* Number of total sectors */
                ST_WORD(tbl+BPB_TotSec16, n_vol);
        } else {
                ST_DWORD(tbl+BPB_TotSec32, n_vol);
        }
        tbl[BPB_Media] = md;                        /* Media descriptor */
        ST_WORD(tbl+BPB_SecPerTrk, 63);             /* Number of sectors per track */
        ST_WORD(tbl+BPB_NumHeads, 255);             /* Number of heads */
        ST_DWORD(tbl+BPB_HiddSec, b_vol);           /* Hidden sectors */
        n = get_fattime();                          /* Use current time as VSN */
        if (fmt == FS_FAT32) {
                ST_DWORD(tbl+BS_VolID32, n);        /* VSN */
                ST_DWORD(tbl+BPB_FATSz32, n_fat);   /* Number of sectors per FAT */
                ST_DWORD(tbl+BPB_RootClus, 2);      /* Root directory start cluster (2) */
                ST_WORD(tbl+BPB_FSInfo, 1);         /* FSInfo record offset (VBR+1) */
                ST_WORD(tbl+BPB_BkBootSec, 6);      /* Backup boot record offset (VBR+6) */
                tbl[BS_DrvNum32] = 0x80;            /* Drive number */
```

```
            tbl[BS_BootSig32] = 0x29;                   /* Extended boot signature
*/
            mem_cpy(tbl+BS_VolLab32, "NO NAME    " "FAT32   ", 19);
        /* Volume label, FAT signature */
        } else {
            ST_DWORD(tbl+BS_VolID, n);                  /* VSN */
            ST_WORD(tbl+BPB_FATSz16, n_fat);    /* Number of sectors per
FAT */
            tbl[BS_DrvNum] = 0x80;                      /* Drive number */
            tbl[BS_BootSig] = 0x29;                     /* Extended boot signature
*/
            mem_cpy(tbl+BS_VolLab, "NO NAME    " "FAT     ", 19);    /*
Volume label, FAT signature */
        }
        ST_WORD(tbl+BS_55AA, 0xAA55);                   /* Signature (Offset
is fixed here regardless of sector size) */
        if (disk_write(drv, tbl, b_vol, 1) != RES_OK)       /* Write VBR */
            return FR_DISK_ERR;
        if (fmt == FS_FAT32)                                /*        Write
backup VBR if needed (VBR+6) */
            disk_write(drv, tbl, b_vol + 6, 1);

        /* Initialize FAT area */
        wsect = b_fat;
        for (i = 0; i < N_FATS; i++) {          /* Initialize each FAT copy */
            mem_set(tbl, 0, SS(fs));                /* 1st sector of the FAT  */
            n = md;                                         /*        Media
descriptor byte */
            if (fmt != FS_FAT32) {
                n |= (fmt == FS_FAT12) ? 0x00FFFF00 : 0xFFFFFF00;
                ST_DWORD(tbl+0, n);                     /* Reserve cluster
#0-1 (FAT12/16) */
            } else {
                n |= 0xFFFFFF00;
                ST_DWORD(tbl+0, n);                     /* Reserve cluster
#0-1 (FAT32) */
                ST_DWORD(tbl+4, 0xFFFFFFFF);
```

```
                    ST_DWORD(tbl+8, 0x0FFFFFFF);  /* Reserve cluster #2 for root dir */
                }
                if (disk_write(drv, tbl, wsect++, 1) != RES_OK)
                    return FR_DISK_ERR;
                mem_set(tbl, 0, SS(fs));            /* Fill following FAT entries with zero */
                for (n = 1; n < n_fat; n++) {       /* This loop may take a time on FAT32 volume due to many single sector writes */
                    if (disk_write(drv, tbl, wsect++, 1) != RES_OK)
                        return FR_DISK_ERR;
                }
            }

            /* Initialize root directory */
            i = (fmt == FS_FAT32) ? au : n_dir;
            do {
                if (disk_write(drv, tbl, wsect++, 1) != RES_OK)
                    return FR_DISK_ERR;
            } while (--i);

#if _USE_ERASE /* Erase data area if needed */
            {
                DWORD eb[2];

                eb[0] = wsect; eb[1] = wsect + (n_clst - ((fmt == FS_FAT32) ? 1 : 0)) * au - 1;
                disk_ioctl(drv, CTRL_ERASE_SECTOR, eb);
            }
#endif

            /* Create FSInfo if needed */
            if (fmt == FS_FAT32) {
                ST_DWORD(tbl+FSI_LeadSig, 0x41615252);
                ST_DWORD(tbl+FSI_StrucSig, 0x61417272);
                ST_DWORD(tbl+FSI_Free_Count, n_clst - 1);   /* Number of free clusters */
```

```
            ST_DWORD(tbl+FSI_Nxt_Free, 2);                  /*          Last
allocated cluster# */
            ST_WORD(tbl+BS_55AA, 0xAA55);
            disk_write(drv, tbl, b_vol + 1, 1);     /* Write original (VBR+1) */
            disk_write(drv, tbl, b_vol + 7, 1);     /* Write backup (VBR+7) */
        }

        return (disk_ioctl(drv, CTRL_SYNC, (void*)0) == RES_OK) ? FR_OK :
FR_DISK_ERR;
}

#endif /* _USE_MKFS && !_FS_READONLY */




#if _USE_STRFUNC
/*-----------------------------------------------------------------------*/
/* Get a string from the file                                            */
/*-----------------------------------------------------------------------*/
TCHAR* f_gets (
        TCHAR* buff,    /* Pointer to the string buffer to read */
        int len,        /* Size of string buffer (characters) */
        FIL* fil        /* Pointer to the file object */
)
{
        int n = 0;
        TCHAR c, *p = buff;
        BYTE s[2];
        UINT rc;


        while (n < len - 1) {                   /* Read bytes until buffer gets filled */
            f_read(fil, s, 1, &rc);
            if (rc != 1) break;                 /* Break on EOF or error */
            c = s[0];
```

```
#if _LFN_UNICODE                        /* Read a character in UTF-8 encoding */
            if (c >= 0x80) {
                if (c < 0xC0) continue;    /* Skip stray trailer */
                if (c < 0xE0) {                 /* Two-byte sequense */
                    f_read(fil, s, 1, &rc);
                    if (rc != 1) break;
                    c = ((c & 0x1F) << 6) | (s[0] & 0x3F);
                    if (c < 0x80) c = '?';
                } else {
                    if (c < 0xF0) {             /* Three-byte sequense */
                        f_read(fil, s, 2, &rc);
                        if (rc != 2) break;
                        c = (c << 12) | ((s[0] & 0x3F) << 6) | (s[1] & 0x3F);
                        if (c < 0x800) c = '?';
                    } else {                    /* Reject four-byte sequense */
                        c = '?';
                    }
                }
            }
#endif
#if _USE_STRFUNC >= 2
            if (c == '\r') continue;      /* Strip '\r' */
#endif
            *p++ = c;
            n++;
            if (c == '\n') break;         /* Break on EOL */
        }
        *p = 0;
        return n ? buff : 0;              /* When no data read (eof or error), return with error. */
}



#if !_FS_READONLY
#include <stdarg.h>
```

```
/*-----------------------------------------------------------------------*/
/* Put a character to the file                                           */
/*-----------------------------------------------------------------------*/
int f_putc (
        TCHAR c,   /* A character to be output */
        FIL* fil       /* Pointer to the file object */
)
{
        UINT bw, btw;
        BYTE s[3];


#if _USE_STRFUNC >= 2
        if (c == '\n') f_putc ('\r', fil);      /* LF -> CRLF conversion */
#endif

#if _LFN_UNICODE     /* Write the character in UTF-8 encoding */
        if (c < 0x80) {                 /* 7-bit */
                s[0] = (BYTE)c;
                btw = 1;
        } else {
                if (c < 0x800) {     /* 11-bit */
                        s[0] = (BYTE)(0xC0 | (c >> 6));
                        s[1] = (BYTE)(0x80 | (c & 0x3F));
                        btw = 2;
                } else {                        /* 16-bit */
                        s[0] = (BYTE)(0xE0 | (c >> 12));
                        s[1] = (BYTE)(0x80 | ((c >> 6) & 0x3F));
                        s[2] = (BYTE)(0x80 | (c & 0x3F));
                        btw = 3;
                }
        }
#else                           /* Write the character without conversion */
        s[0] = (BYTE)c;
        btw = 1;
#endif
        f_write(fil, s, btw, &bw);          /* Write the char to the file */
```

```
	return (bw == btw) ? 1 : EOF;  /* Return the result */
}



/*-----------------------------------------------------------------------*/
/* Put a string to the file                                              */
/*-----------------------------------------------------------------------*/
int f_puts (
	const TCHAR* str,	/* Pointer to the string to be output */
	FIL* fil		/* Pointer to the file object */
)
{
	int n;


	for (n = 0; *str; str++, n++) {
		if (f_putc(*str, fil) == EOF) return EOF;
	}
	return n;
}



/*-----------------------------------------------------------------------*/
/* Put a formatted string to the file                                    */
/*-----------------------------------------------------------------------*/
int f_printf (
	FIL* fil,		/* Pointer to the file object */
	const TCHAR* str,	/* Pointer to the format string */
	...			/* Optional arguments... */
)
{
	va_list arp;
	BYTE f, r;
```

```
UINT i, j, w;
ULONG v;
TCHAR c, d, s[16], *p;
int res, cc;


va_start(arp, str);

for (cc = res = 0; cc != EOF; res += cc) {
	c = *str++;
	if (c == 0) break;			/* End of string */
	if (c != '%') {				/* Non escape character */
		cc = f_putc(c, fil);
		if (cc != EOF) cc = 1;
		continue;
	}
	w = f = 0;
	c = *str++;
	if (c == '0') {				/* Flag: '0' padding */
		f = 1; c = *str++;
	} else {
		if (c == '-') {			/* Flag: left justified */
			f = 2; c = *str++;
		}
	}
	while (IsDigit(c)) {			/* Precision */
		w = w * 10 + c - '0';
		c = *str++;
	}
	if (c == 'l' || c == 'L') {	/* Prefix: Size is long int */
		f |= 4; c = *str++;
	}
	if (!c) break;
	d = c;
	if (IsLower(d)) d -= 0x20;
	switch (d) {				/* Type is... */
	case 'S' :				/* String */
```

```
                p = va_arg(arp, TCHAR*);
                for (j = 0; p[j]; j++) ;
                res = 0;
                while (!(f & 2) && j++ < w) res += (cc = f_putc(' ', fil));
                res += (cc = f_puts(p, fil));
                while (j++ < w) res += (cc = f_putc(' ', fil));
                if (cc != EOF) cc = res;
                continue;
        case 'C' :                              /* Character */
                cc = f_putc((TCHAR)va_arg(arp, int), fil); continue;
        case 'B' :                              /* Binary */
                r = 2; break;
        case 'O' :                              /* Octal */
                r = 8; break;
        case 'D' :                              /* Signed decimal */
        case 'U' :                              /* Unsigned decimal */
                r = 10; break;
        case 'X' :                              /* Hexdecimal */
                r = 16; break;
        default:                                /* Unknown type (passthrough) */
                cc = f_putc(c, fil); continue;
        }

        /* Get an argument and put it in numeral */
        v = (f & 4) ? va_arg(arp, long) : ((d == 'D') ? (long)va_arg(arp, int) : va_arg(arp, unsigned int));
        if (d == 'D' && (v & 0x80000000)) {
                v = 0 - v;
                f |= 8;
        }
        i = 0;
        do {
                d = (TCHAR)(v % r); v /= r;
                if (d > 9) d += (c == 'x') ? 0x27 : 0x07;
                s[i++] = d + '0';
        } while (v && i < sizeof(s) / sizeof(s[0]));
```

```
            if (f & 8) s[i++] = '-';
            j = i; d = (f & 1) ? '0' : ' ';
            res = 0;
            while (!(f & 2) && j++ < w) res += (cc = f_putc(d, fil));
            do res += (cc = f_putc(s[--i], fil)); while(i);
            while (j++ < w) res += (cc = f_putc(' ', fil));
            if (cc != EOF) cc = res;
        }

        va_end(arp);
        return (cc == EOF) ? cc : res;
}

#endif /* !_FS_READONLY */
#endif /* _USE_STRFUNC */
```

Diskio.h

```
/*-----------------------------------------------------------------------
/  Low level disk interface modlue include file   (C)ChaN, 2009
/-----------------------------------------------------------------------*/

#ifndef _DISKIO
#define _DISKIO

#include "integer.h"


/* Status of Disk Functions */
typedef BYTE      DSTATUS;

/* Results of Disk Functions */
typedef enum {
        RES_OK = 0,             /* 0: Successful */
        RES_ERROR,              /* 1: R/W Error */
        RES_WRPRT,              /* 2: Write Protected */
        RES_NOTRDY,             /* 3: Not Ready */
```

```
	RES_PARERR		/* 4: Invalid Parameter */
} DRESULT;


/*---------------------------------------*/
/* Prototypes for disk control functions */

int assign_drives (int, int);
DSTATUS disk_initialize (BYTE);
DSTATUS disk_status (BYTE);
DRESULT disk_read (BYTE, BYTE*, DWORD, BYTE);
DRESULT disk_write (BYTE, const BYTE*, DWORD, BYTE);
DRESULT disk_ioctl (BYTE, BYTE, void*);



/* Disk Status Bits (DSTATUS) */

#define STA_NOINIT		0x01	/* Drive not initialized */
#define STA_NODISK		0x02	/* No medium in the drive */
#define STA_PROTECT		0x04	/* Write protected */


/* Command code for disk_ioctrl fucntion */

/* Generic command (mandatory for FatFs) */
#define CTRL_SYNC			0	/* Flush disk cache (for write functions) */
#define GET_SECTOR_COUNT	1	/* Get media size (for only f_mkfs()) */
#define GET_SECTOR_SIZE		2	/* Get sector size (for multiple sector size (_MAX_SS >= 1024)) */
#define GET_BLOCK_SIZE		3	/* Get erase block size (for only f_mkfs()) */

#endif
```

Blink.c

```
#include <msp430.h>
#include "HAL_Dogs102x6.h"
#include "HAL_Cma3000.h"
#include "ff.h"
#include "structure.h"
#include "CTS_Layer.h"
#include <stdlib.h>
#include <math.h>

#define GPIO_DIR_INPUT(...) GPIO_DIR_INPUT_SUB(__VA_ARGS__)
#define GPIO_DIR_INPUT_SUB(port, pin) (P##port##DIR &= ~(1 << (pin)))
#define GPIO_PULLUP(...) GPIO_PULLUP_SUB(__VA_ARGS__)
#define GPIO_PULLUP_SUB(port, pin) P##port##REN |= (1 << (pin)); P##port##OUT |= (1 << (pin))
#define GPIO_READ_PIN(...) GPIO_READ_PIN_SUB(__VA_ARGS__)
#define GPIO_READ_PIN_SUB(port, pin) ((P##port##IN & (1 << (pin))) ? 1 : 0)
#define GPIO_WRITE_PIN(...) GPIO_WRITE_PIN_SUB(__VA_ARGS__)
#define GPIO_WRITE_PIN_SUB(port, pin, value) (P##port##OUT = (P##port##OUT & ~(1 << (pin))) | (value << (pin)))

const struct Sensor Sensor1 =
{
   .halDefinition = RO_COMPB_TA1_TA0,
   .numElements = 1,
   .baseOffset = 0,
   .cbpdBits = 0x0001,
   .arrayPtr[0] = &PAD1,
   .cboutTAxDirRegister = (uint8_t *)&P1DIR,
   .cboutTAxSelRegister = (uint8_t *)&P1SEL,
   .cboutTAxBits = BIT6,
   .measGateSource = TIMER_ACLK,
   .sourceScale = TIMER_SOURCE_DIV_0,
   .accumulationCycles = 50
};
```

```
#define S2_PORT 2
#define S2_PIN 2

#define DRAW_TEXT_ROW 7
#define LINE_Y 45

#define FILE_NAME "buffer.bin"

#define BUFFER_SIZE 90
#define BUFFER_COUNT (BUFFER_SIZE / 2)

volatile uint16_t buffer[BUFFER_COUNT];
volatile uint8_t index = 0;

uint8_t first_press_PAD = 0;
uint8_t no_press_PAD = 0;
uint8_t first_press_S2 = 0;
uint8_t no_press_S2 = 0;
uint8_t file_draw = 0;
UINT bw = 0;

uint16_t accel_y = 0;

void Cma3000_readPotentiometer();

uint16_t get_draw_value(uint8_t index)
{
   uint16_t draw_value = (uint16_t)((float)DOGS102x6_X_SIZE *
(float)buffer[index] / 360);
   return draw_value;
}

void draw()
{
   uint16_t draw_value = get_draw_value(index);
   uint8_t x_pos = index;
   if (x_pos == 45) {
```

```
    Dogs102x6_clearRow(x_pos / 8);
  }
  if (x_pos % 8 == 0) {
    Dogs102x6_clearRow(x_pos / 8);
  }
  Dogs102x6_pixelDraw(draw_value, x_pos, DOGS102x6_DRAW_NORMAL);
  Dogs102x6_horizontalLineDraw(0, DOGS102x6_X_SIZE - 1, LINE_Y,
DOGS102x6_DRAW_NORMAL);
  uint16_t val = buffer[index];
  int f = 0;
  char str[40] = "";
  if (val == 180) {
    val = 0;
    f = 2;
  } else if (val <= 180) {
    val = 180 - val;
    f = 1;
  } else {
    val = val - 180;
  }
  int8_t i = 1;
  do {
    str[i++] = (char)(val % 10 + '0');
    val = val / 10;
  } while (val > 0 && index < 40);
  uint8_t j = 0;
  for (i; i >= j; i--) {
    char temp = str[i];
    str[i] = str[j];
    str[j] = temp;
    j++;
  }
  switch(f) {
  case 2:
    str[0] = ' ';
  break;
  case 1:
```

```
        str[0] = '-';
    break;
    default:
        str[0] = '+';
    }
    Dogs102x6_clearRow(7);
    Dogs102x6_stringDraw(DRAW_TEXT_ROW, 0, str,
DOGS102x6_DRAW_NORMAL);
}
void draw_from_file(FIL file)
{
    no_press_PAD = 1;
    first_press_PAD = 0;
    file_draw ^= 1;
    if (file_draw) {
        f_open(&file, FILE_NAME, FA_READ);
        f_read(&file, buffer, BUFFER_SIZE, &bw);
        Dogs102x6_clearScreen();
        Dogs102x6_horizontalLineDraw(0, DOGS102x6_X_SIZE - 1, LINE_Y,
DOGS102x6_DRAW_NORMAL);
        f_close(&file);
        uint16_t i = 0;
        for (i = 0; i < BUFFER_COUNT; i++) {
            uint16_t draw_value = get_draw_value(i);
            Dogs102x6_pixelDraw(draw_value, i, DOGS102x6_DRAW_NORMAL);
        }
    } else {
        Dogs102x6_clearScreen();
        Dogs102x6_horizontalLineDraw(0, DOGS102x6_X_SIZE - 1, LINE_Y,
DOGS102x6_DRAW_NORMAL);
    }
}

void write_to_file(FIL file)
{
    if (first_press_S2 == 0)
        first_press_S2 = 1;
```

```
    else if(first_press_S2 == 1) {
        no_press_S2 = 1;
        first_press_S2 = 0;
        f_open(&file, FILE_NAME, FA_WRITE | FA_CREATE_ALWAYS);
        f_write(&file, buffer, BUFFER_SIZE, &bw);
        f_close(&file);
    }
}
uint16_t main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    GPIO_DIR_INPUT(S2_PORT, S2_PIN);
    GPIO_PULLUP(S2_PORT, S2_PIN);
    Dogs102x6_init();
    Dogs102x6_backlightInit();
    Dogs102x6_setBacklight(255);
    Dogs102x6_clearScreen();
    Dogs102x6_horizontalLineDraw(0, DOGS102x6_X_SIZE - 1, LINE_Y,
DOGS102x6_DRAW_NORMAL);
    TI_CAPT_Init_Baseline(&Sensor1);
    TI_CAPT_Update_Baseline(&Sensor1,5);
    Cma3000_init();
    FATFS fs;
    FIL file;
    FRESULT res = f_mount(0, &fs);
    if (res == FR_NO_FILESYSTEM) {
        f_mkfs(0, 0, 512);
    }
    DMACTL0 = DMA0TSEL_5;
    DMA0CTL = DMADT_0+DMAEN+DMAIE;
    DMA0SZ = 1;
    __data16_write_addr((unsigned short) &DMA0SA,(unsigned long) &accel_y);
    __data16_write_addr((unsigned short) &DMA0DA,(unsigned long)
&buffer[index]);
    TA2CCR0 = 200;
    TA2CTL = TIMER_ACLK + TIMER_SOURCE_DIV_0;
    TA2CTL |= (TACLR + MC__UP);
```

```
   while (1) {
      __bis_SR_register(LPM0_bits+GIE);
      if (file_draw == 0) {
         draw();
      }
      Cma3000_readPotentiometer();
      struct Element * keypressed = 0;
      keypressed = (struct Element *)TI_CAPT_Buttons(&Sensor1);
      if (keypressed == 0) {
         no_press_PAD = 0;
      }
      if(keypressed && no_press_PAD == 0) {
         if (first_press_PAD == 0) {
            first_press_PAD = 1;
         }
         else if(first_press_PAD == 1) {
            draw_from_file(file);
         }
      }
      uint8_t value_S2 = !GPIO_READ_PIN(S2_PORT, S2_PIN);
      if (value_S2 == 0) {
         no_press_S2 = 0;
      }
      if(value_S2 && no_press_S2 == 0) {
         write_to_file(file);
      }
      index++;
      if (index == BUFFER_COUNT) {
         index = 0;
      }
      __data16_write_addr((unsigned short) &DMA0DA,(unsigned long)
&buffer[index]);
      DMA0CTL |= DMAEN;
      TA2CTL |= (TACLR + MC__UP);
   }
}
#pragma vector=DMA_VECTOR
```

```
__interrupt void DMA_ISR(void)
{
   switch(__even_in_range(DMAIV,16)) {
   case 2:
      TA2CTL &= ~MC__UP;
      _bic_SR_register_on_exit(LPM0_bits);
   break;
   default:
      break;
   }
}

void Cma3000_readPotentiometer()
{
   Cma3000_yAccel = Cma3000_readRegister(DOUTY);
   __delay_cycles(50 * 25);
   Cma3000_zAccel = Cma3000_readRegister(DOUTZ);
   __delay_cycles(50 * 25);
   double value = atan2((double)Cma3000_yAccel, (double)Cma3000_zAccel);
   accel_y = (uint16_t)((int16_t)(value * 180.0 / M_PI) + 180);
}
```