Porting "Chan's FatFS" to the LPC4350

Richard Man , February, 2012

In the competitive embedded microcontroller market, one way a silicon vendor can differentiate its offerings from the rest is by integrating higher-function peripherals onchip. As such, it is no longer unusual to find chips with built-in LCD, USB, CAN, Ethernet, and in some instances, even hardware support for the SD/MMC memory devices.

Such is the case with the NXP LPC4350. With two ARM Cortex cores and many other features, it also sports SD/MMC hardware. Without dedicated SD/MMC support, the firmware typically must use bit banging code which is timing sensitive and adds to the complexity of the code. This example demonstrates how simple it is to port a well known FatFS to work with the LPC4350.

Chan's FatFS

Chan's FatFS has been around since 2006. As it is highly configurable and can be tuned to use very little code and runtime memory, and it has been ported to different microcontrollers ranging from 8-bit AVR and PIC24 to 32-bit ARM devices. It implements the FAT file system, which is an industry standard for use with removable media. You can use it to access files and directories and allow sharing of the data with other devices or simply to use widely available storage media.

FatFS can also be used to manage file resources in non-removable media such as flash memory, but that is outside the scope of this project.

The software is written in highly portable ANSI C. The majority of the code is independent of the target, and a port to a new device only requires implementing a small set of low-level disk I/O functions. The most basic ones are (actual function prototypes are not used due to a special type naming convention):

```
DSTATUS disk_initialize(int drive);
DRESULT disk_read(int drive, unsigned char *buf, unsigned sector, int count);
DRESULT disk_write(int drive, unsigned char *buf, unsigned sector, int count).
```

Due to the high code quality and the generous license of FatFS, it's a rather easy way to get SD/MMC support in an embedded project.

Download the sample code here

Since the LPC4350 has plenty of flash and SRAM, I have opted to enable all the conditional features with the FatFS code, including long filename support. With the demonstration program, the code size is just over 25K bytes.

SDIO Driver

To garner user support, NXP engineers have written a set of example programs and driver files for the new LPC43xx devices. Among them, the SDIO (Secured Digital I/O for SD/MMC functions) driver is so new that I had to get the source code directly from their git server on the web. Undoubtedly, the code will be integrated into the official distribution channel soon.

Along with a number of support functions, the following declarations in the SDIO file are the most important for this project:

```
int sdio_read_blocks(void *buffer, int start_block, int end_block);
int sdio_write_blocks(void *buffer, int start_block, int end_block);
```

As you can see, they match the signatures for the FatFS disk I/O functions almost exactly. With some slight massaging the code worked like a charm, making this probably one of the simplest ports of the FatFS code.

Demonstration Program

I wrote a test program to demonstrate the working of the FatFS code. It uses the serial port to accept commands from the user. On a Windows PC, connect the serial to USB cable (or a direct serial cable if your machine still has a serial port) and use a simple terminal program to set the baud rate to 9600 bps.

When the program starts, it displays a startup message. Typing 'h' gives you the available commands. The program only tests the directory, file reading and writing, and file delete functions. Since the FatFS is known to be robust, there is no point in exhaustively testing all of its functions, which is beyond the scope of this project in any case.

Of note is also the code in the uart_debug.c file. The bulk of the code is from another NXP example project that uses interrupt-driven I/O with the UART. I have added a small amount of code to support command echoing and checking when the low level UART ring buffer is full. It's a good practice to use interrupt-driven I/O at the lowest level since it does not tie up the CPU as much as polling; however one must be careful in the high-level code to check for boundary conditions.

The file also contains its own version of fputc, and fgets functions. By rewriting these functions, the demonstration program can use normal C I/O routines such as printf for debugging as well as user interaction.

Summary

In summary, the LPC4350 looks to pack quite a bit of peformance in a hallmark low-power ARM design. With the SD/MMC support, it's trivial to add an SD card to your system.

Date of release: February 2012
Published in the USA