The TFS in Action

Posted on May 29, 2012

In a recent posting, an abstraction for the M25P80 Serial Memory Device was examined. In a later posting, the application programming interface for the Trivial File System (TFS) was explained. This article will delve into some examples of the TFS in action, showing how, even a file system as simple as TFS can be used successfully in an embedded system. A future posting will detail the internals of the TFS.

The first step in using the TFS is to configure the file system for use. this is accomplished with a few simple definitions:

```
//***********************************************************
// Platform Specific Control - the M25P80 SPI Flash Device.
//***********************************************************
#define -- USE_M25P80_SPI_FLASH            1
#define M25P80_THREAD_SAFE              0

#if -- USE_M25P80_SPI_FLASH
#define -- USE_TRIVIAL_FILE_SYSTEM         1
#define TFS_THREAD_SAFE                 1
#define TFS_HANDLES                     3
#endif
```

The first define enable support for the M25P80. The second bypasses thread safe mutex semaphores. This is possible since the TFS is the only user of the M25P90. Next, the TFS is enabled followed by thread safe semaphores at the file system level. Finally, three file handles are allocated, so up to three files can be opened/acted upon at one time. This is similar to the file handles setting in old MS-DOS™.

The first bit of code that is needed is to open the file system form use. An example is shown below:

```
switch (emTFSOpenFS())
{
    case emOK:
    case emWRN_WRITE_PROTECT:
        break;

    case emERR_UNFORMATTED:
    case emERR_UNKNOWN_VERSION:
        // Warn the user about this slow operation.
        // ... some code deleted ...
        emTFSFormatMedia(16384);
        break;

    case emERR_NO_RESPONSE:
    default:
        // This should never happen.
        // Tell the user of the disaster
        // ... some code deleted ...
        while (1) {}  // Spin loop
        break;
}
```

This code is a bit paranoid; the M25P80 is soldered onto the development board so emERR_NO_RESPONSE *should* be impossible, still the code tests and reacts to this calamity anyway. Overall the code is clear, efficient and simple. Next is an example of touch screen parameters being retrieved. Note that if the file cannot be read, this simple code assumes that the file was not found and calls the screen calibration routine to supply the needed parameters.

```
h = emTFSOpenFile((PU8)"TSP");

if (h >= 0)
{
    if (emTFSReadFile(h,
                      0,
                      sizeof(emTOUCHPARMS),
                      (PU8)&touchParms) == emOK)
        emSetTouchLimits(&touchParms);

    emTFSCloseFile(h);
}
else
{
    setupTouchScreen();
}
```

Finally, an example of touch screen parameters being saved out to non-volatile storage.

```
// Write out the touch screen calibration data.
if ((h = emTFSCreateFile((PU8)"TSP", sizeof(emTOUCHPARMS))) >= 0)
{
    emTFSWriteFile(h, 0, sizeof(emTOUCHPARMS), (PU8)&tp);
    emTFSCloseFile(h);
}
```

Note that when the file is created, we need to tell the TFS about the size of the data that will be stored. In embedded applications this is often simply the “sizezof” the data structure that holds the required information.

In conclusion, the TFS simplifies many of the tasks encountered in embedded systems without the high overhead of more capable, full-featured, file systems. As always, your comments and suggestions are welcomed and invited.

Peter Camilleri (aka Squidly Jones)

This entry was posted in **Embedded Development** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=843]** .