Indice:

1. Setup
2. Premessa librerie
3. Modifiche allo sketch originale
4. Note su probabili errori che si presenteranno

QUESTO FILE FA RIFERIMENTO ALLA MODIFICA DELL’ESEMPIO DA LIBRERIA DELLA SD\_CARD E NON AL CODICE IMPLEMENTATO NELLA TI

1 Setup

TI Sd module

Gpio 122 (mosi) mosi

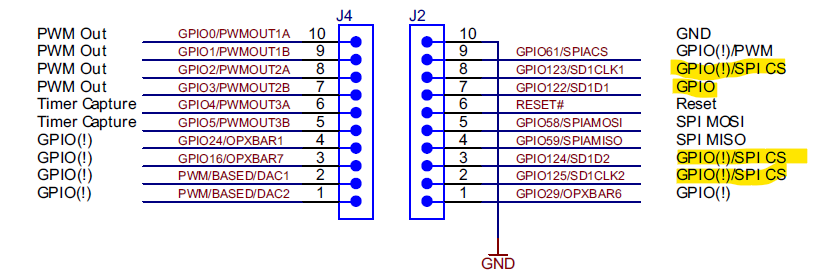
Gpio 123 (miso) miso

Gpio 124 (clock) sck

Gpio 125 (cs) cs

3v 3v

Gnd gnd



2 premessa sulle librerie

Lo sketch originale di C2000 ware usa la libreria fatfs ver R0.04b del 2007 (molto datata). La potete trovare nel disco ad un percorso simile:

C:\ti\c2000\C2000Ware\_3\_04\_00\_00\utilities\third\_party\f2837xd\fatfs\src

3 modifiche allo sketch originale (d’ora in avanti SO) della libreria c2000Ware

* Nel SO sono implementate diverse funzioni equivalenti dei comandi shell linux (es ls, cd, mkdir, …). Non siamo interessati a tutte le funzioni, ma solamente a quella di write, perciò attualemente è funzionante solo questa funzione
* Nel SO non era possibile eseguire una scrittura “in append” su un file pre-esitente

fresult = f\_open(&g\_sFileObject, g\_cTmpBuf, FA\_WRITE | FA\_CREATE\_ALWAYS);

IMPORTANTE: nella libreria fatfs R0.04b l’append non è implementato. È necessario aggiornare la libreria

Al seguente link è possibile scaricare la versione corrente della libreria fatfs (al momento viene usata la versione R0.14b)

<http://elm-chan.org/fsw/ff/00index_e.html>

* CONFIGURAZIONE PROGETTO
  1. estrarre lo zip scaricato dal sito sopra linkato.
  2. Includere nel progetto la cartella appena estratta
  3. copiare nel progetto sd\_card il file ff.c presente nel percorso

[Cartella libreria fatfs nuova]\source

* 1. Modificare gli include presenti in sd\_card.c nel seguente modo

//#include "third\_party/fatfs/src/ff.h"

//#include "third\_party/fatfs/src/diskio.h"

**#include** "ff.h"

**#include** "diskio.h"

* 1. Modificare la funzione Cmd\_write nel seguente modo:

linea 786 circa

fresult = f\_open(&g\_sFileObject, g\_cTmpBuf, FA\_WRITE | FA\_OPEN\_EXISTING |FA\_OPEN\_APPEND);

linea 817 circa

fresult = **f\_write**(&g\_sFileObject, writeBuff, **strlen**(writeBuff), &usBytesWritten);

* 1. Assicurarsi che nel file mmc\_F28379x.c

Gli include siano come i seguenti

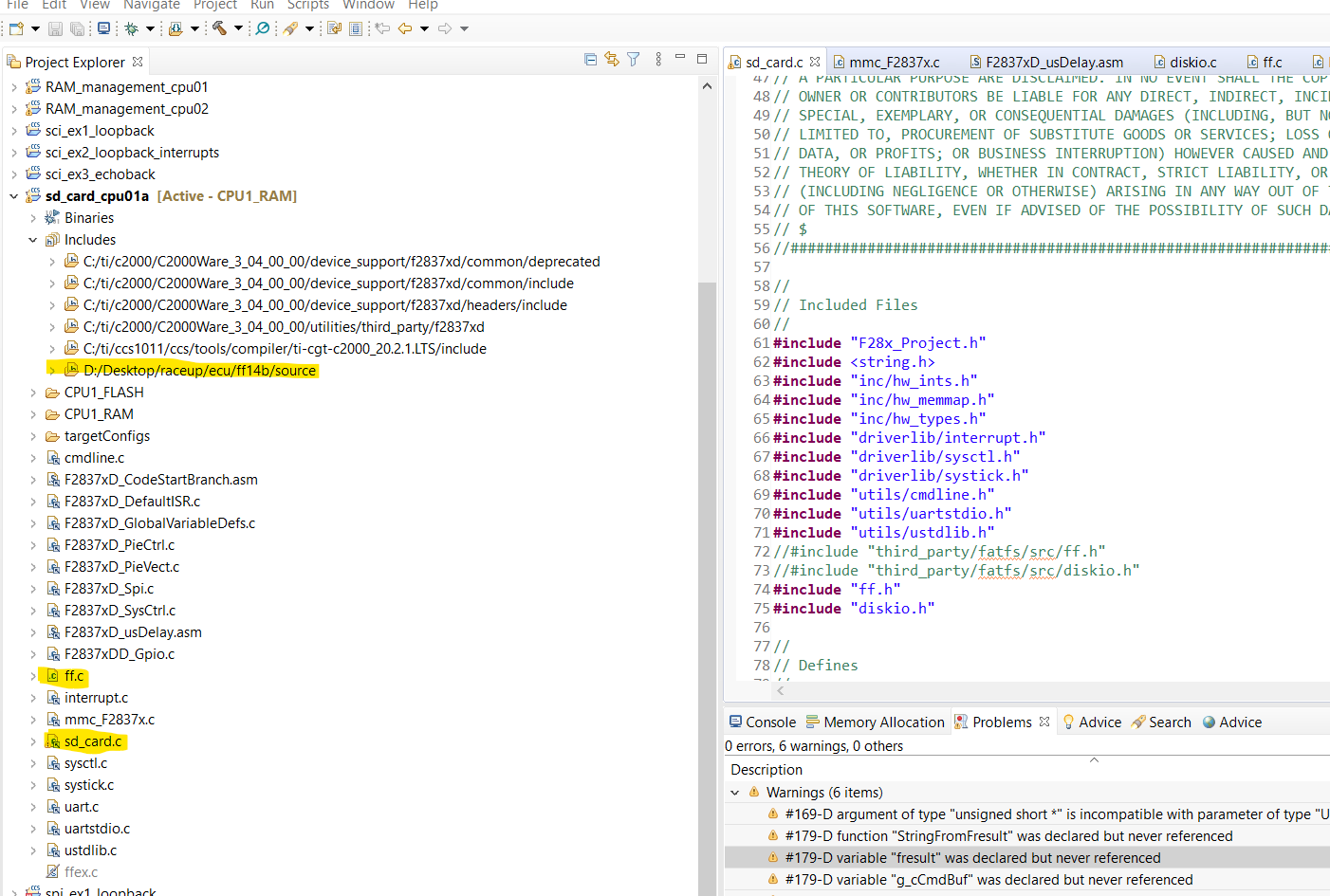
**#include** "F28x\_Project.h"

//#include "F2837xD\_types.h"

**#include** "inc/hw\_types.h"

**#include** "third\_party/fatfs/src/diskio.h"

Il progetto si presenta così al termine:



* CONFIGURAZIONE MAIN

**int**

**main**(**void**)

{

//int nStatus;

FRESULT fresult;

//

// Initialize System Control

//

**InitSysCtrl**();

var = 1000;

**#ifdef** \_FLASH

//

// Copy time critical code and Flash setup code to RAM

// This includes the following functions: InitFlash();

// The RamfuncsLoadStart, RamfuncsLoadSize, and RamfuncsRunStart

// symbols are created by the linker. Refer to the device .cmd file.

//

memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size\_t)&RamfuncsLoadSize);

//

// Call Flash Initialization to setup flash waitstates

// This function must reside in RAM

//

InitFlash();

**#endif**

//

// Initialize interrupt controller and vector table

//

**InitPieCtrl**();

**InitPieVectTable**();

var = 7;

//

// Set the system tick to fire 100 times per second.

//

**SysTickInit**();

**SysTickPeriodSet**(**SysCtlClockGet**(SYSTEM\_CLOCK\_SPEED) / 100);

**SysTickIntRegister**(SysTickHandler);

**SysTickIntEnable**();

**SysTickEnable**();

//

// Enable Interrupts

//

**IntMasterEnable**();

**InitGpio**();

**GPIO\_SetupPinMux**(31, GPIO\_MUX\_CPU1, 0);

**GPIO\_SetupPinOptions**(31, GPIO\_OUTPUT, GPIO\_PUSHPULL);

//

// Configure UART0 for debug output.

//

//ConfigureUART();

//

// Print hello message to user.

//

//UARTprintf("\n\nSD Card Example Program\n");

//UARTprintf("Type \'help\' for help.\n");

//

// Mount the file system, using logical disk 0.

//

var = 4;

**f\_mount**(&g\_sFatFs, "0:", 1);

//f\_mount(0, &g\_sFatFs);

var = 0;

//

diskResult = **disk\_initialize**(0);

//

// diskResult = disk\_status(0);

**char** cmd[40] = "write test.txt primo append\n";

**CmdLineProcess**(cmd);

**while**(var < 3){

**GPIO\_WritePin**(31, 0);

**char** kmd[40] = "write test.txt sto appendendo\n";

**CmdLineProcess**(kmd);

DELAY\_US(1000000);

var++;

**GPIO\_WritePin**(31, 1);

DELAY\_US(1000000);

}

//

// Enter an (almost) infinite loop for reading and processing commands from

// the user.

//

// while(1)

// {

// //

// // Print a prompt to the console. Show the CWD.

// //

// UARTprintf("\n%s> ", g\_cCwdBuf);

//

// //

// // Get a line of text from the user.

// //

// UARTgets(g\_cCmdBuf, sizeof(g\_cCmdBuf));

//

// //

// // Pass the line from the user to the command processor.

// // It will be parsed and valid commands executed.

// //

// nStatus = CmdLineProcess(g\_cCmdBuf);

//

// //

// // Handle the case of bad command.

// //

// if(nStatus == CMDLINE\_BAD\_CMD)

// {

// UARTprintf("Bad command!\n");

// }

// //

// // Handle the case of too many arguments.

// //

// else if(nStatus == CMDLINE\_TOO\_MANY\_ARGS)

// {

// UARTprintf("Too many arguments for command processor!\n");

// }

//

// //

// // Otherwise the command was executed. Print the error

// // code if one was returned.

// //

// else if(nStatus != 0)

// {

// UARTprintf("Command returned error code %s\n",

// StringFromFresult((FRESULT)nStatus));

// }

// }

APPUNTI SULLA MODIFICA DEL MAIN

1. Var è una variabile di debug dichiarata ad inizio progetto potete tranquillamente commentare
2. Tutta la parte di codice che si riferisce ad UART va commentata in quanto non funzionante. Essa riguarda solo uno strumento per il debug (utile, ma non necessario).