Using the SDIO Development Card #1

Confidential

CONTRIBUTORS

Written by Greg Wilson and Geoff Richmond
Engineering contributions by Geoff Richmond and Gary Stratton.

Palm, Inc. reserves the right to revise this documentation and to make changes in content from time to time without obligation on the part of Palm, Inc. to provide notification of such revision or changes. PALM, INC. MAKES NO REPRESENTATIONS OR WARRANTIES THAT THE DOCUMENTATION IS FREE OF ERRORS OR THAT THE DOCUMENTATION IS SUITABLE FOR YOUR USE. THE DOCUMENTATION IS PROVIDED ON AN “AS IS” BASIS. PALM, INC. MAKES NO WARRANTIES, TERMS OR CONDITIONS, EXPRESS OR IMPLIED, EITHER IN FACT OR BY OPERATION OF LAW, STATUTORY OR OTHERWISE, INCLUDING WARRANTIES, TERMS, OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND SATISFACTORY QUALITY.

TO THE FULL EXTENT ALLOWED BY LAW, PALM, INC. ALSO EXCLUDES FOR ITSELF AND ITS SUPPLIERS ANY LIABILITY, WHETHER BASED IN CONTRACT OR TORT (INCLUDING NEGLIGENCE), FOR DIRECT, INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL, OR PUNITIVE DAMAGES OF ANY KIND, OR FOR LOSS OF REVENUE OR PROFITS, LOSS OF BUSINESS, LOSS OF INFORMATION OR DATA, OR OTHER FINANCIAL LOSS ARISING OUT OF OR IN CONNECTION WITH THIS DOCUMENTATION, EVEN IF PALM, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Using the SDIO Development Card #1
February 1, 2002
You must be a Palm, Inc. PluggedIn program member
(http://www.palm.com/developers/) and an SD Card Association
member (http://www.sdcard.org/) to have this document.

Palm, Inc.
5470 Great America Pkwy.
Santa Clara, CA 95052
USA
www.palm.com

For the latest version of this document logon to:
https://pluggedin.palm.com/

Using the SDIO Development Card #1

This document is organized into the following sections:

- Introduction
- The Demonstration Application
- Card Design
- Microcontroller Software Driver
- CPLD Firmware Design
- Reprogramming the EDK Card
- Tracing SDIO Commands
- Design Tips and Tricks

Introduction

The SDIO Development Card #1 (hereinafter called the "EDK card") is a sample SDIO design demonstrating an SDIO interface to a microcontroller. It is part of an Expansion Development Kit (EDK) that allows hardware developers to experiment with SDIO hardware and software for initial learning, evaluation and experimentation purposes. The card includes a Microchip PICMicro® FLASH microcontroller, on-board flash memory, and a Xilinx® XPLA3 CPLD for maximum flexibility.

The EDK card comes pre-programmed as an SDIO card with one function. Function 1 implements a thermometer and a general-purpose LED. It also contains a CSA (Code Storage Area) that gets mounted as a read-only file system on Palm OS 4.x.

This document explains the design and use of the EDK card, and should serve as a valuable resource when developing hardware, firmware and software for custom SDIO peripherals.

IMPORTANT: The EDK card should only be used in a static—safe environment. The card's exposed electronics can expose the internal components of the host Palm handheld to potentially damaging static charge.

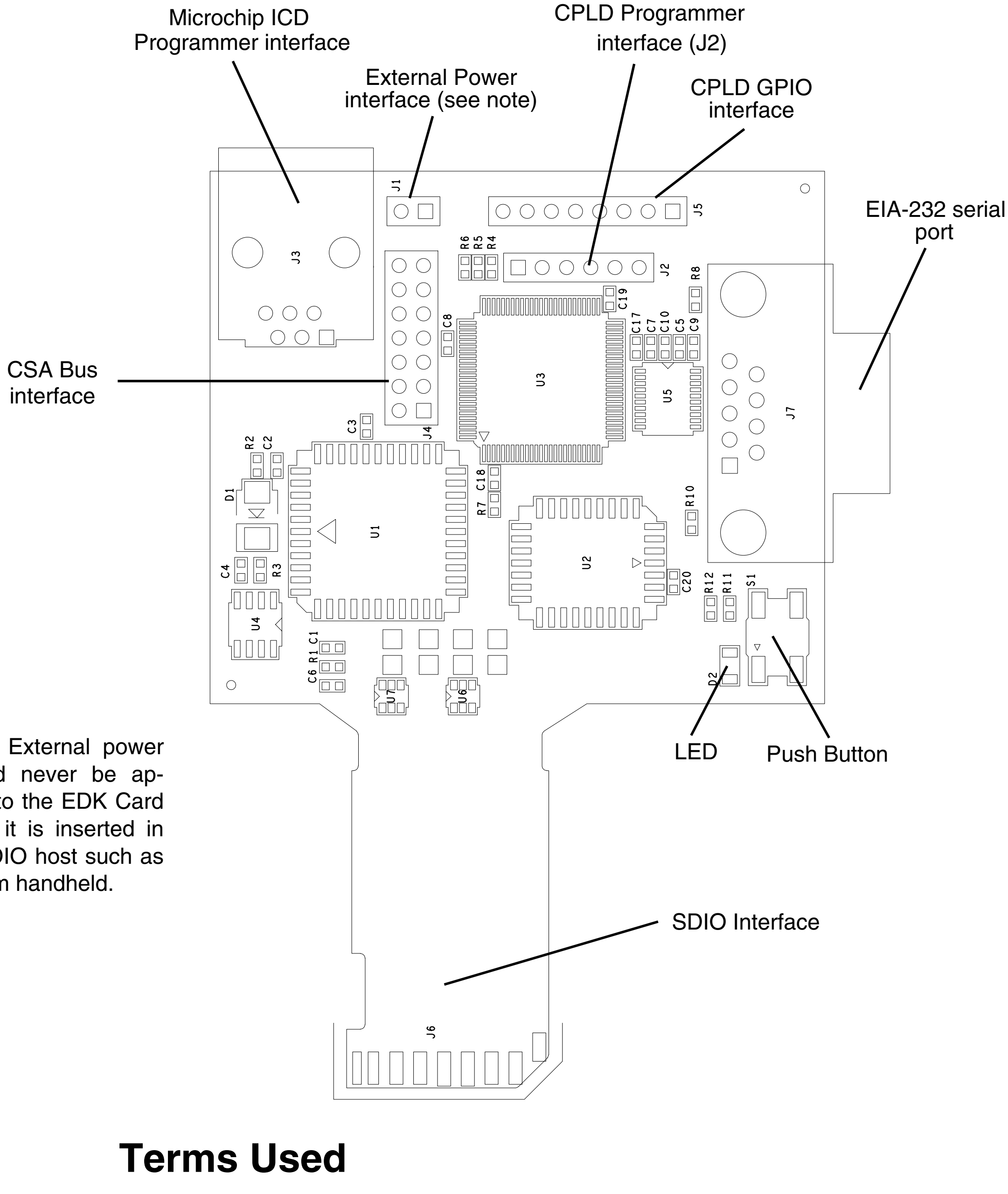
EDK Card Features

The EDK card features the following:

- SDIO interface
- PICMicro FLASH microcontroller for communicating with the SD host and interfacing to the I/O functions and memory
- Xilinx CPLD (Complex Programmable Logic Device) that provides address controller and serial I/O functions
- Flash ROM containing the SDIO CSA data and any drivers, applications, or data required by any card-resident Palm OS software
- LED for output display
- SPST push button switch for simple user input
- Temperature sensor IC
- EIA-232 serial interface
- MPLAB® ICD programmer interface for programming the PICMicro microcontroller
- JTAG programmer interface for programming the CPLD

Figure 1 points out the various interfaces on the EDK card.

Figure 1 EDK card interfaces



Note: External power should never be applied to the EDK Card when it is inserted in an SDIO host such as a Palm handheld.

Terms Used

This document assumes that the reader is familiar with the following terms:

Handheld: The Palm device to which the peripheral interfaces.

EDK: Expansion Development Kit. The hardware available for purchase from the Palm Expansion Parts Store.

HDK: Hardware Development Kit. Electrical interface specifications, environmental specs, 2D drawings, 3D CAD models, and a list of vendor contacts needed to design and build an SDIO peripheral.

Host: A device capable of interfacing with SDIO cards as an SDIO host (master), such as a Palm handheld.

SD: Secure Digital.

SDIO: An extension to the SD specification enabling I/O functionality using the SD interface in a SD-derived form-factor.

SDK: Software Development Kit.

Slot Driver: A driver for the Palm handheld that controls the handheld’s expansion slots. This driver is in turn controlled by the Palm OS Expansion Manager.

Other Useful Documentation

SDIO peripheral developers will find the following documents useful:

- *SD Memory Card Specifications*, available from the SD Card Association.
- *SDIO Specification Version 1.0*, available from the SD Card Association
- *Developing SDIO Peripherals for Palm Handhelds*, available as part of the Palm SDIO SDK. It can be downloaded from the PluggedIn Program.

The Demonstration Application

The EDK card comes pre-programmed with a simple demonstration application that exercises the EDK card’s temperature sensor and LED. The source code and CodeWarrior project used to construct this demonstration application is included as part of the Palm SDIO EDK. It can be found in the `/Software/EDKDemo` folder.

Launching the Demonstration Application

To launch the demonstration program, simply insert the EDK card—with components up and edge contacts toward you[1]—into the SD slot on a Palm handheld that supports SDIO attachments[2]. Upon insertion, the EDK card's LED will light to indicate that power has been applied to the EDK card and the microcontroller is functioning. After a slight delay during which the card is identified and the on-board program is transferred to the handheld, the SDIO demonstration application will launch.

If the demonstration application fails to launch, verify that your handheld is running Palm OS 4.0 or greater. If the handheld is running Palm OS 4.0, also verify that the SDIO slot driver, named `SlotDriver:SDIO-sdsd`, is installed in RAM. (Starting with Palm OS 4.1 this slot driver doesn't need to be installed separately.) Finally, verify that you haven't installed a custom slot driver: the EDK card is only certified to work with Palm's SDIO slot driver.

Card Design

The primary objective of the EDK card is to provide SDIO hardware and software developers with a sample SDIO design using a microcontroller as the interface to the SDIO slot. The interface is flexible enough to be adapted to other designs using different microcontrollers.

Design Requirements

The *SDIO Specification* details the system design requirements of SDIO cards and SDIO hosts. While the specification includes powerful features designed to allow for high-bandwidth communications with a host, multiple functions to operate simultaneously, and for secure data transfer, the requirements for a typical SDIO design compatible with Palm handhelds are usually

[1] This orientation is correct for Palm's m500, m505, and m125 handhelds; other devices may require a different orientation.

[2] Note that the handheld must be running Palm OS 4.0 or greater, and must have the SDIO slot driver installed, either in RAM or in ROM.

far simpler. The EDK card is designed to demonstrate the most common features of a Palm compatible SDIO solution. In addition, it offers the developer a flexible environment for prototyping and experimentation.

The EDK card addresses the following design requirements:

- Powered by the SDIO slot
- 128K flash RAM for code storage of autorun application(s)
- Supports SPI mode
- Generates SDIO interrupts
- Programmer interface connectors for PIC Programmer and JTAG Programmer
- Supports IO_SEND_OP_COND command/response (CMD5/R4)
- Supports IO_RW_DIRECT command/response (CMD52/R5)
- Supports RAW (Read after Write) IO_RW_DIRECT
- Supports IO_RW_EXTENDED command/response (CMD53/R5)
- Embedded I/O Code Storage Area (CSA)
- Power-on reset

Command Set

The EDK card supports the following SDIO commands:

Table 1 Supported SDIO commands

Command	Name	Description
CMD0	GO_IDLE_STATE	Resets card and switches to SPI mode
CMD5	IO_QUERY	Send I/O card data to host
CMD52	IO_RW_DIRECT	Single byte function access command
CMD53	IO_RW_EXTENDED	Multiple byte function access command
CMD59	CRC_ON_OFF	Turns the CRC on or off

The EDK card does not support the following SDIO operations:

- SDIO Suspend/Resume operation
- SDIO Read Wait operation
- SDIO RW Extended Block operation in “infinite block transfer” mode

Principal Hardware Components

The EDK card has been developed to allow for in-circuit programming of the microcontroller and the CPLD components, thereby reducing development time and expense, and allowing the flexibility of upgrades to firmware and software. The microcontroller and CPLD components used in this design were chosen because of the added benefit of free development tools. These include software design, test, simulation, and programming tools.

Microcontroller

The EDK card was implemented using a PICmicro FLASH microcontroller (MCU) from Microchip Technology Inc. This microcontroller was selected for the following reasons:

- Simple, inexpensive programming interface
- Free development tools available
- Low-voltage (3.3 V) operation
- Built-in SPI port, UART, and 10-bit A/D converter
- On-chip flash memory for easy prototyping and updates in the field

Using the PICStart Plus Development Programmer and a programming cable, the PIC16LF877 microcontroller can be programmed using any PC with a serial interface. Programming software for the microcontroller is available at http://www.microchip.com/.

CPLD

The CPLD chosen for the EDK card is a Xilinx CoolRunner XPLA3-series XCR3128XL CPLD. This component offers low power, a relatively large capacity, free development tools, the ability to program in-circuit, and is available in a low-height Chip-Scale (0.8mm pitch BGA-type) package.

Using the Xilinx Parallel Download Cable III and the programming software available from Xilinx at http://www.xilinx.com/[3], the CPLD can be programmed on the EDK card directly from any PC. Any JEDEC file that targets the XCR3128XL can be downloaded to the EDK card in this manner.

Flash ROM

The flash ROM chosen for the SDIO Reference Design is an AMD AM29LV001B 3 volt flash. This component offers low power, 3 volt programming.

LED

A user-programmable LED is connected directly to a Port A pin on the microcontroller.

SPST Switch

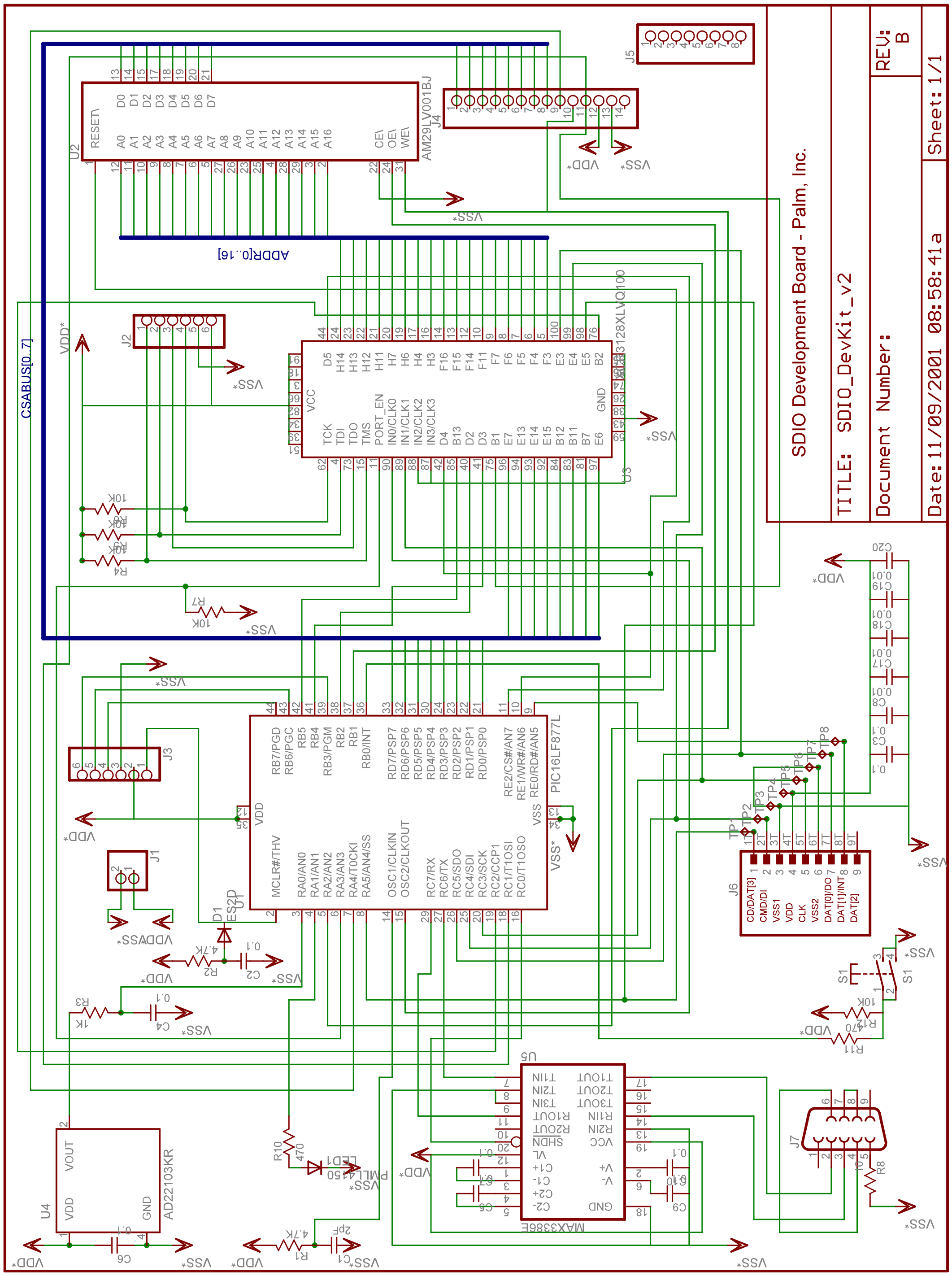
An SPST push button switch is included for simple user input. The switch is connected directly to a Port B pin on the microcontroller.

Schematic

Figure 2 presents the EDK card schematic.

[3] Click on Products, then on Design Tools, then, in the left-hand frame, on Free ISE WebPACK.

Figure 2 SDIO Development Card #1 Rev. B schematic



Parts List

Table 2 lists the parts used in the construction of the EDK card.

Table 2 Parts List

Designator	Value	Package
C1	2 pF Surface Mount Capacitor	C0603
C2-C10	0.1 uF Surface Mount Capacitor	C0603
C17	0.01 uF Surface Mount Capacitor	C0603
C18	0.1 uF Surface Mount Capacitor	C0603
C19-C20	0.01 uF Surface Mount Capacitor	C0603
D1	D10BQ015SMB Schottky Diode	SMB
D2	QTLP652C4 LED Diode Green	
U1	PIC16LF877L Microcontroller	PLCC-44
U2	AM29LV001BJ 1-Megabit Flash Memory	PLCC32R
U3	XCR3128XL-VQ100 CPLD	VQFP100
U4	AD22103KR Low Voltage Temperature Sensor	SO8
U5	MAX3386E RS232 Transceiver	TSSOP20
U6-U7	SRV05-4 Rail Clamp Low Capacitance Diode Array	SOT-23
J1	PINHD-1X2 1x2 0.1" Header	1X02
J2	PINHD-1X6 1x6 0.1" Header	1X06
J3	Low-profile, through-hole, side entry PCB modular jack 6-6	
J4	PINHD-2x7 2x7 0.1″ Header	2x07
S1	SPST SMD Switch	
J7	DB-9 Plug	

Table 2 Parts List *(continued)*

Designator	Value	Package
R1-R2	4.7K ohm Surface Mount Resistor	R0603
R3	1K ohm Surface Mount Resistor	R0603
R4-R7	10K ohm Surface Mount Resistor	R0603
R8	10 ohm Surface Mount Resistor	R0603
R10-R11	470 ohm Surface Mount Resistor	R0603
R12	10K ohm Surface Mount Resistor	R0603

Microcontroller Software Driver

Memory Structure

The EDK card memory structure is composed of four areas defining the mandatory memory-areas and register-sets defined in the SDIO 1.0 specification, as well as the optional CSA memory area. The EDK card design includes 8 KB of on-chip flash memory in the PIC microcontroller, and 128 KB of external flash memory. The on-chip flash memory is used for program storage, while the external flash memory is used primarily for CSA storage.

Table 3 illustrates the memory map of the EDK card.

Table 3 EDK card memory map

Location	Address Range	Name	Description
Microcontroller General Purpose Register	0x1A0 – 0x1BF	CCCR	Card Common Control Register (32 bytes)
Microcontroller General Purpose Register	0x1C0 – 0x1DF	FBR	Function Basic Registers (32 bytes per function)

Table 3 EDK card memory map *(continued)*

Location	Address Range	Name	Description
Microcontroller Program Memory	0x1800 – 0x1FFF (Program Memory)	CIS	Card Information Structure (Tuple Structures)
External Flash Memory	0x00000 – 0x1FFFF	CSA	Function 1 Code Storage Area.

Program Area

The program area extends from 0x0000h to 0x17FF, enabling up to 6 KB of program storage. The memory layout is standard for a PIC microcontroller.

Address Range	Name
0x00 – 0x03	Reset vector
0x04 – 0x1F	Interrupt handler
0x20 – 0x0FF	Page 0 subroutines
0x100 – 0x17FF	Main program
0x1800 – 0x1FFF	CIS storage area

CCCR Area

The Card Common Control Register Area is described in the *SDIO Specification*. This area is used to query the card's capabilities and to control basic card services.

According to the *SDIO Specification*, the CCCR is logically addressed as memory locations 0x00 – 0xFF. In this design, the CCCR physically resides in the microcontroller general register memory space from locations 0x1A0 – 0x1BF. The firmware in the EDK card maps logical memory addresses to the physical memory locations.

Table 4 shows the mapping from SDIO logical addresses to physical memory locations for the CCCR Area.

Table 4 CCCR logical-to-physical address map

SDIO Address	Physical Address	Register Name
0x00	0x1A0	CCCR/SDIO Revision
0x01	0x1A1	SD Specification Revision
0x02	0x1A2	I/O Enable
0x03	0x1A3	I/O Ready
0x04	0x1A4	Int Enable
0x05	0x1A5	Int Pending
0x06	0x1A6	I/O Abort
0x07	0x1A7	Bus Interface Control
0x08	0x1A8	Card Capability
0x09 – 0x0B	0x1A9 – 0x1AB	Common CIS Pointer
0x0C	0x1AC	Bus Suspend
0x0D	0x1AD	Function Select
0x0E	0x1AE	Exec Flags
0x0F	0x1AF	Ready Flags
0x10 – 0x11	0x1B0 – 0x1B1	FN0 Block Size
0x12 – 0xFF	0x1B2 - 0x1BF	Reserved for future use

FBR Area

The Function Basic Register Area is described in the *SDIO Specification*. This area is used to query individual function capabilities and to control basic function services. This area also allows access to any CSA areas residing on the EDK card.

According to the *SDIO Specification*, the FBR is logically addressed as memory locations 0x100 – 0xFFF. In this design, the FBR physically resides in the microcontroller general register memory space from locations 0x1C0 – 0x1DF. The firmware in the EDK card maps logical memory addresses to the physical memory locations.

Table 5 shows the mapping from SDIO logical addresses to physical memory locations for FBR 1.

Table 5 FBR 1 logical-to-physical address map

SDIO Address	Physical Address	Register Name
0x100	0x1C0	Function 1 FBR Info
0x101 – 0x108	0x1C1 – 0x1C8	Reserved for future use
0x109 – 0x10B	0x1C9 – 0x1CB	Pointer to Function 1 CIS
0x10C – 0x10E	0x1CC – 0x1CE	Pointer to Function 1 CSA
0x10F	0x1CF	Data Access Window to Function 1 CSA
0x110 – 0x111	0x1D0 – 0x1D1	I/O Block Size for Function 1
0x112 – 0x1FF	0x1D2	Reserved for future use
0x200 – 0xFFF	unmapped	Unused Function 2-7 FBR and reserved for future use area

CIS Area

The Card Information Structure Area is described in the *SDIO Specification* and in the PC-Card (PCMCIA) specifications (available at `www.pcmcia.org`). This area is used to determine the capabilities of the card and of individual functions.

According to the *SDIO Specification*, the CIS structures are stored in logical addresses 0x0001000 - 0x017FFF. In this design, the CIS structures physically reside in the microcontroller program memory space from locations 0x1800 - 0x1FFF. The firmware in the EDK card maps logical memory addresses to physical memory locations.

Table 6 shows the mapping from SDIO logical addresses to program memory addresses.

Table 6 CIS logical-to-program memory address map

Logical Address	Program Memory Address	Description
0x1000	CIS_TABLE (0x1800)	Card's Common Card Information Structure (CIS)
0x2000	FN1_CIS_TABLE (variable)	Function 1 Card Information Structure (CIS)

The CIS data is stored in program memory as a simple look-up table. To return the value at an offset in the table, the offset is put in the W register and a subroutine call is made to the start of the table. The value at the offset is returned in the W register. For more details on this technique, refer to the RETLW instruction documented in the *PICmicro™ Mid-Range MCU Family Reference Manual*.

Function 1 Registers

The EDK card implements a simple LED and thermometer function. The Function 1 registers are located in the microcontroller general-purpose register space at addresses 0x0020 - 0x003F. The following table describes each of the Function 1 registers.

Table 7 Function 1 register logical-to-physical address map

Function 1 Register address (logical)	Physical address	Description
0x00	0x0020	LED control. Set to 1 to turn the LED on.
0x01	0x0021	Temperature low-order byte.
0x02	0x0022	Temperature high-order byte.
0x03	0x0023	Function 1 command register. Set to 0xA5 to erase the card's EDK Flash memory (this memory serves as the Function 1 CSA).

Table 7 Function 1 register logical-to-physical address map

Function 1 Register address (logical)	Physical address	Description
0x04	0x0024	Firmware version of microcontroller code.
0x05	0x0025	Firmware version of CPLD.

CSA Area

The Code Storage Area is described in the *SDIO Specification*. This area is used to hold drivers, data, and applications that may be required by the host to interact with the SDIO card. The CSA area is accessed through a data window controlled by a series of registers residing in the FBR area. The microcontroller automatically creates the offset from the requested CSA address and the physical memory space, in addition to automatically incrementing the CSA window address after each read.

In order for a CSA to be readable by the Palm OS, it should be in FAT12/FAT16 format and any drivers, data, or applications that should be automatically detected by the Palm handheld should reside in the `/PALM` and `/PALM/Launcher` directories. Once a CSA area is mounted in the handheld memory space, an application may access any data within the CSA, regardless of the data's directory location. For more information on creating a CSA image, see "Programming the EDK Flash Memory" on page 27.

The CSA physically resides in the external flash memory space from 0x00000 – 0x1FFFF.

CPLD Firmware Design

The EDK card's CPLD firmware was developed on Xilinx's WebPOWERED software using the Verilog HDL language, and the compiled binary code was downloaded to the EDK card using a Xilinx Parallel Download Cable. The Verilog language was chosen as this provides a good balance between hardware abstraction (thereby allowing the easy implementation of a new design with different components), development speed, and effort.

Two recommended reference books on implementing Verilog designs for CPLD handhelds are:

- *Real World FPGA Design with Verilog* by Ken Coffman, published by Prentice Hall PTR, 2000
- *The Verilog Hardware Description Language, Fourth Edition* by Donald E. Thomas and Philip R. Moorby, published by Kluwer Academic Publishers, 1998

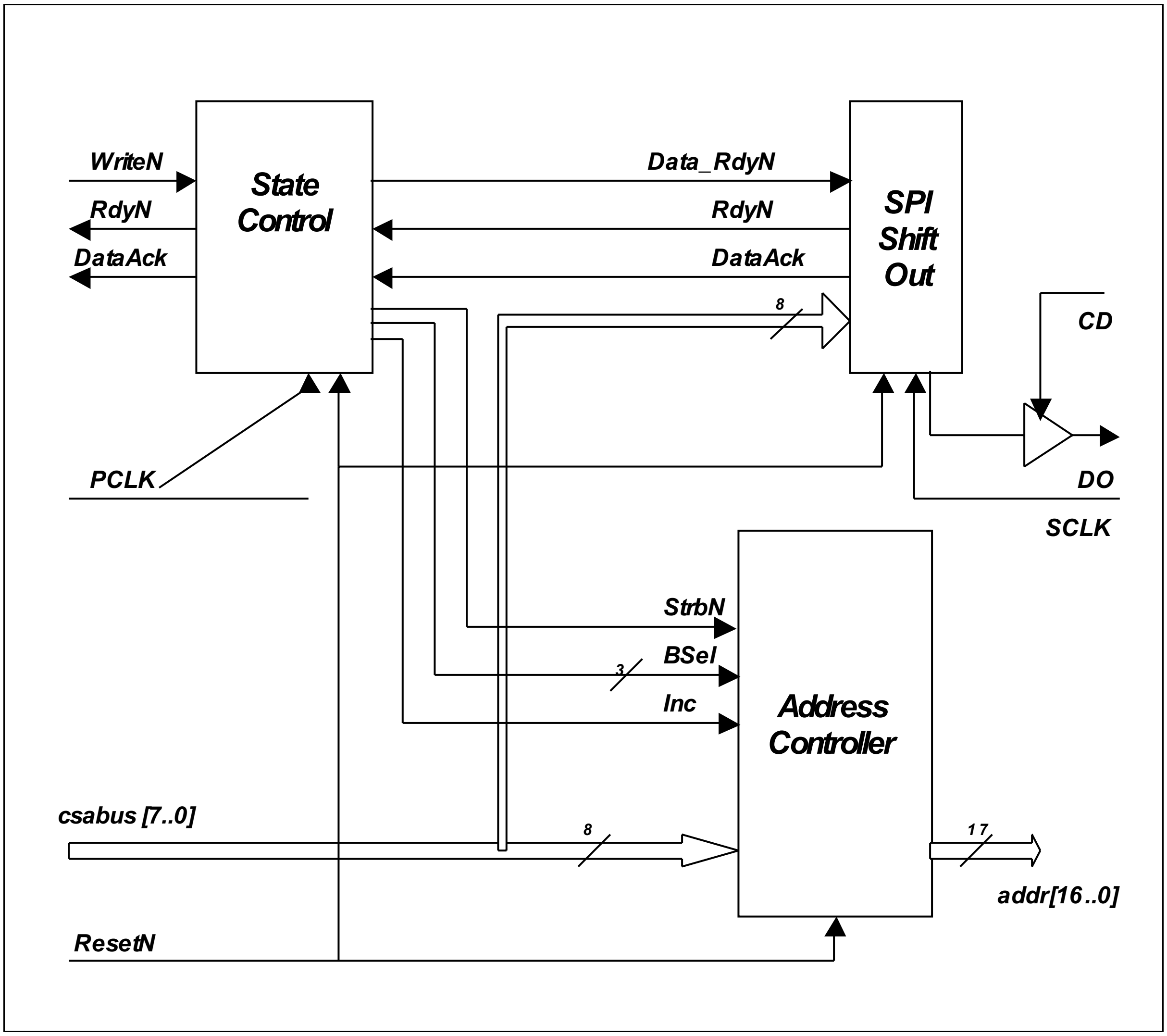
NOTE: The CPLD was used in the EDK card design because it provides an inexpensive way to implement the SDIO specification in a low-volume product. In a real product this specification could be implemented in hardware.

Because the EDK card is primarily intended to allow you to experiment with the interaction between the PIC code and the Palm OS application code, you need not change the programming of the CPLD. Instead, you may want to focus your attention on the PIC's firmware, since it illustrates one way to create a product that connects to the SD interface.

CPLD Firmware Block Diagram

Figure 3 is a block diagram of the CPLD design.

Figure 3 CPLD Block Diagram (SPI Only)



Reprogramming the EDK Card

The EDK card's microcontroller, CPLD, and flash can all be reprogrammed. The following sections walk you through the process of reprogramming each:

- Programming the Microchip PICMicro Microcontroller
- Programming the Xilinx XPLA3 CPLD

- Programming the EDK Flash Memory

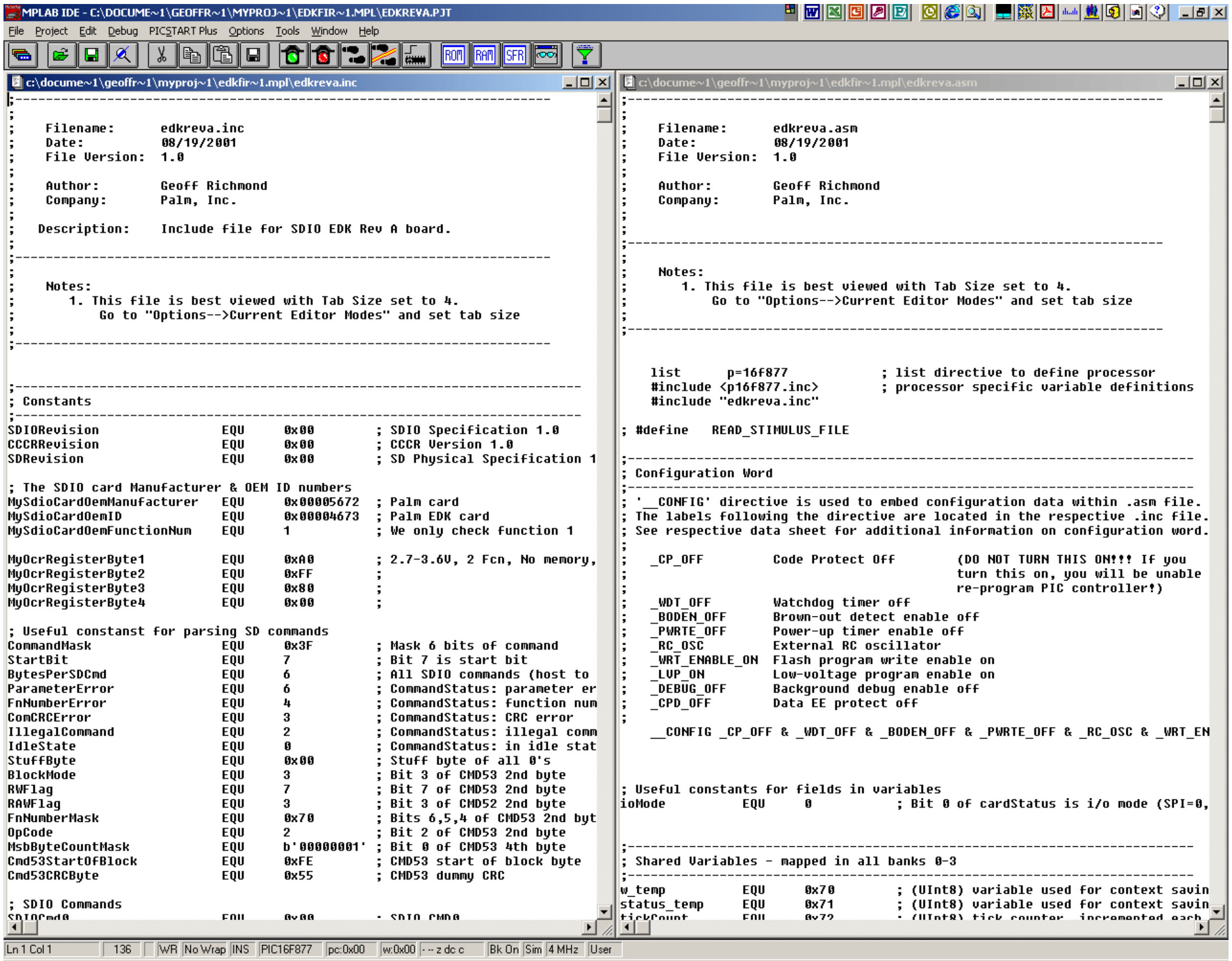
Programming the Microchip PICMicro Microcontroller

To program the PICMicro microcontroller, you need both the Microchip MPLAB IDE software and the MPLAB ICD (In-Circuit Debugger). The MPLAB IDE can be downloaded from `www.microchip.com`. The ICD can be purchased from Digi-Key (call 800-DIGIKEY). Purchase either the DV164001 ICD Module, which doesn't include a power supply, or the DV164003 ICD Deluxe Developer's Kit, which includes both a power supply and a demo board.

Once you have obtained the IDE software and the ICD, the following procedure walks you through the process of programming the microcontroller:

1. Download the Microchip MPLAB IDE toolset and install it on the development PC. The following packages were used in the EDK card development and should be installed:
   - MPLAB IDE Files
   - MPLAB ASM Assembler and Linker
   - MPLAB SIM Simulator Environment
   - MPLAB ICD In-circuit Debugger Software
   - Help Files
2. Once the Microchip MPLAB software is installed, copy the `EDKFirmware.MPLAB` folder (which can be found inside the Palm SDIO EDK's `Firmware` folder) to an appropriate MPLAB Project folder.
3. Start MPLAB.
4. Under the Project menu, open the project file `edkreva.pjt`. The following will be displayed:
   - `edkreva.asm`
   - `edkreva.inc`

   The window should look something like this:


MPLAB IDE - C:\DOCUME~1\GEOFFR~1\MYPROJ~1\EDKFIR~1.MPL\EDKREVA.PJT
File Project Edit Debug PICSTART Plus Options Tools Window Help
ROM RAM SFR
c:\docume~1\geoffr~1\myproj~1\edkfir~1.mpl\edkreva.inc
c:\docume~1\geoffr~1\myproj~1\edkfir~1.mpl\edkreva.asm
Ln 1 Col 1
136
WR
No Wrap
INS
PIC16F877
pc:0x00
w:0x00
- -- z dc c
Bk On
Sim
4 MHz
User

```
;-----------------------------------------------------------------------
;
;    Filename:      edkreva.inc
;    Date:          08/19/2001
;    File Version:  1.0
;
;    Author:        Geoff Richmond
;    Company:       Palm, Inc.
;
;   Description:    Include file for SDIO EDK Rev A board.
;
;-----------------------------------------------------------------------
;
;    Notes:
;       1. This file is best viewed with Tab Size set to 4.
;           Go to "Options-->Current Editor Modes" and set tab size
;
;-----------------------------------------------------------------------



;---------------------------------------------------------------------------
; Constants
;---------------------------------------------------------------------------
SDIORevision                EQU     0x00        ; SDIO Specification 1.0
CCCRRevision                EQU     0x00        ; CCCR Version 1.0
SDRevision                  EQU     0x00        ; SD Physical Specification 1

; The SDIO card Manufacturer & OEM ID numbers
MySdioCardOemManufacturer   EQU     0x00005672  ; Palm card
MySdioCardOemID             EQU     0x00004673  ; Palm EDK card
MySdioCardOemFunctionNum    EQU     1           ; We only check function 1

MyOcrRegisterByte1          EQU     0xA0        ; 2.7-3.6V, 2 Fcn, No memory,
MyOcrRegisterByte2          EQU     0xFF        ;
MyOcrRegisterByte3          EQU     0x80        ;
MyOcrRegisterByte4          EQU     0x00        ;

; Useful constanst for parsing SD commands
CommandMask                 EQU     0x3F        ; Mask 6 bits of command
StartBit                    EQU     7           ; Bit 7 is start bit
BytesPerSDCmd               EQU     6           ; All SDIO commands (host to
ParameterError              EQU     6           ; CommandStatus: parameter er
FnNumberError               EQU     4           ; CommandStatus: function num
ComCRCError                 EQU     3           ; CommandStatus: CRC error
IllegalCommand              EQU     2           ; CommandStatus: illegal comm
IdleState                   EQU     0           ; CommandStatus: in idle stat
StuffByte                   EQU     0x00        ; Stuff byte of all 0's
BlockMode                   EQU     3           ; Bit 3 of CMD53 2nd byte
RWFlag                      EQU     7           ; Bit 7 of CMD53 2nd byte
RAWFlag                     EQU     3           ; Bit 3 of CMD52 2nd byte
FnNumberMask                EQU     0x70        ; Bits 6,5,4 of CMD53 2nd byt
OpCode                      EQU     2           ; Bit 2 of CMD53 2nd byte
MsbByteCountMask            EQU     b'00000001' ; Bit 0 of CMD53 4th byte
Cmd53StartOfBlock           EQU     0xFE        ; CMD53 start of block byte
Cmd53CRCByte                EQU     0x55        ; CMD53 dummy CRC

; SDIO Commands
SDIOCmd0                    EQU     0x00        ; SDIO CMD0
```

```
;-----------------------------------------------------------------------
;
;    Filename:      edkreva.asm
;    Date:          08/19/2001
;    File Version:  1.0
;
;    Author:        Geoff Richmond
;    Company:       Palm, Inc.
;
;
;-----------------------------------------------------------------------
;
;    Notes:
;       1. This file is best viewed with Tab Size set to 4.
;           Go to "Options-->Current Editor Modes" and set tab size
;
;-----------------------------------------------------------------------


   list      p=16f877           ; list directive to define processor
   #include <p16f877.inc>       ; processor specific variable definitions
   #include "edkreva.inc"

; #define   READ_STIMULUS_FILE

;---------------------------------------------------------------------------
; Configuration Word
;---------------------------------------------------------------------------
; '__CONFIG' directive is used to embed configuration data within .asm file.
; The labels following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.
;
;   _CP_OFF         Code Protect Off        (DO NOT TURN THIS ON!!! If you
;                                           turn this on, you will be unable
;                                           re-program PIC controller!)
;   _WDT_OFF        Watchdog timer off
;   _BODEN_OFF      Brown-out detect enable off
;   _PWRTE_OFF      Power-up timer enable off
;   _RC_OSC         External RC oscillator
;   _WRT_ENABLE_ON  Flash program write enable on
;   _LVP_ON         Low-voltage program enable on
;   _DEBUG_OFF      Background debug enable off
;   _CPD_OFF        Data EE protect off
;
    __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_OFF & _RC_OSC & _WRT_EN



; Useful constants for fields in variables
ioMode          EQU     0           ; Bit 0 of cardStatus is i/o mode (SPI=0,


;---------------------------------------------------------------------------
; Shared Variables - mapped in all banks 0-3
;---------------------------------------------------------------------------
w_temp          EQU     0x70        ; (UInt8) variable used for context savin
status_temp     EQU     0x71        ; (UInt8) variable used for context savin
tickCount       EQU     0x72        ; (UInt8) tick counter, incremented each
```

5. Press the F10 key to compile the project.

To download the program to the PIC16LF877 microcontroller, do the following:

1. Remove the EDK card from the Palm handheld.
2. Connect the Microchip ICD Programmer to programmer interface J3.
3. Apply +3.3 V to the VDD pin, connecting VSS to the negative supply terminal.

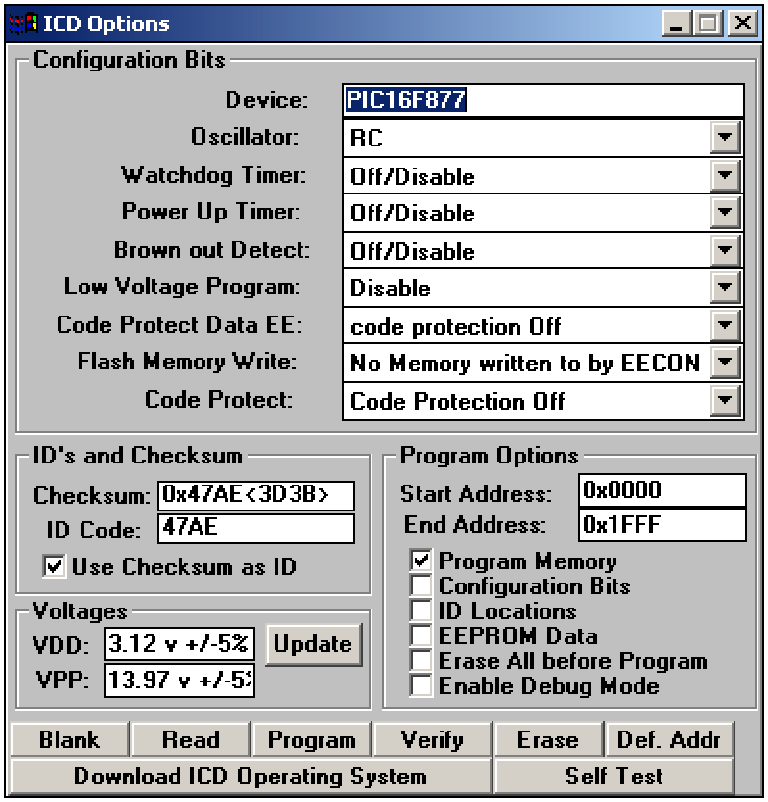
   Note that when the EDK card is inserted into a Palm handheld's SD slot, the handheld will apply +3.3 V. This is the recommended operating and programming voltage that should be applied to VDD (+ terminal) and VSS (- terminal).

WARNING! The EDK card has a maximum voltage rating of 3.6V. **Do not exceed this voltage**.

IMPORTANT: During programming of the PICMicro microcontroller, the EDK card should be removed from the Palm handheld and powered by an external 3.3V power supply (100mA minimum).

The LED on the ICD programmer should turn on or flash.

4. In the MPLAB IDE application, select Development Mode… from the Options menu.
5. On the Tools tab select MPLAB ICD Debugger, then click Apply. The MPLAB ICD will initialize.
6. In the MPLAB ICD window make sure that COM1 is selected. COM2 does not work. This means that you should set HotSync to use COM2 with your serial cradle.
7. In the MPLAB ICD window click Options….
8. Configure the Options window as shown here:



ICD Options

Configuration Bits

Device: PIC16F877
Oscillator: RC
Watchdog Timer: Off/Disable
Power Up Timer: Off/Disable
Brown out Detect: Off/Disable
Low Voltage Program: Disable
Code Protect Data EE: code protection Off
Flash Memory Write: No Memory written to by EECON
Code Protect: Code Protection Off

ID's and Checksum

Checksum: 0x47AE<3D3B>
ID Code: 47AE
☑ Use Checksum as ID

Voltages

VDD: 3.12 v +/-5%
VPP: 13.97 v +/-5

Program Options

Start Address: 0x0000
End Address: 0x1FFF
☑ Program Memory
☐ Configuration Bits
☐ ID Locations
☐ EEPROM Data
☐ Erase All before Program
☐ Enable Debug Mode

WARNING! Never click Blank or Erase. Neither should you select “Erase All before Program” or “Code Protection On”. These can cause the PIC16LF877 to be reset to “code protect” mode, which disables the in-circuit programming capabilities of the microcontroller. If this happens, you will no longer be able to program the PIC16LF877. Only use the “Program” function. If you want to experiment with the other functions, do so on a packaged part that can be discarded if an error occurs.

9. Click Program to program the part.
10. After successful programming, in the MPLAB IDE window select Development Mode… from the Options menu.
11. In the Tools tab select MPLAB SIM Simulator mode and click Apply.

12. Turn off power and disconnect all power and programmer leads.

Programming is complete.

Programming the Xilinx XPLA3 CPLD

To program the Xilinx CoolRunner XPLA3 CPLD, you need a JTAG programming cable—such as the Parallel Cable III, which can be ordered from Xilinx or through a Xilinx distributor—along with the appropriate software. The HDL software design tools required to program this component are available from Xilinx as part of their free WebPOWERED PC software suite, and can be downloaded from the Xilinx website (`www.xilinx.com`). The following software packages are required:

- Design Entry
- XPLA Fitter
- XPLA Programmer
- MXE Simulator
- HDL Bencher

These are optional:

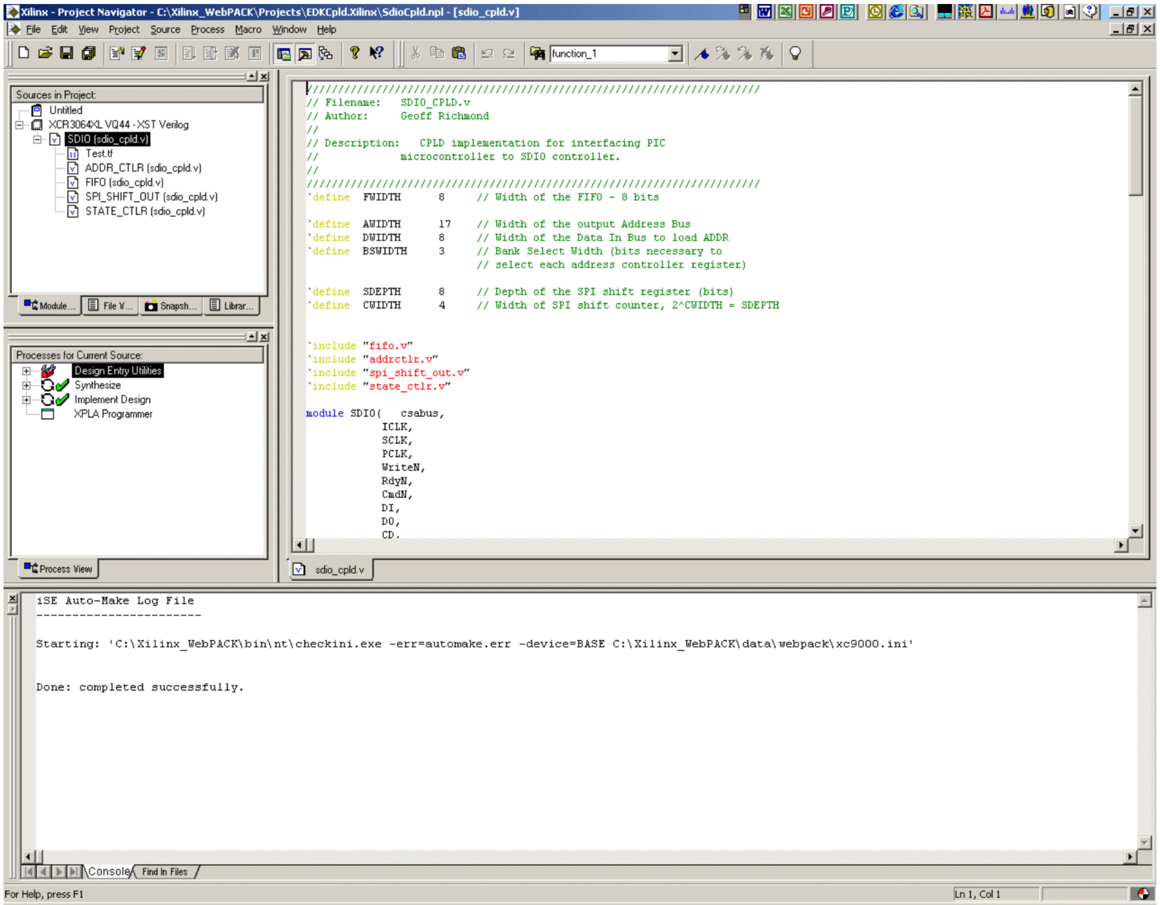
- WebPACK ECS-CPLD Lib
- WebPACK State CAD

Once you have obtained the programming cable and the WebPOWERED software, the following procedure walks you through the process of programming the CPLD:

1. Download the ISE WebPACK tools (listed above) for the CoolRunner XPLA3 series of CPLDs from `www.xilinx.com`, and install on the development PC.
2. Copy the `EdkCpld.Xilinx` folder (which can be found inside the Palm SDIO EDK's `Firmware` folder) into a WebPACK Projects folder.
3. Once the WebPOWERED software is installed on a host machine, do the following:
    a. Start the Xilinx Project Navigator application

b. From the File menu, select Open Project, then navigate to and open the `EdkCpld.Xilinx` folder. Open the `SdioCpld.npl` project file.

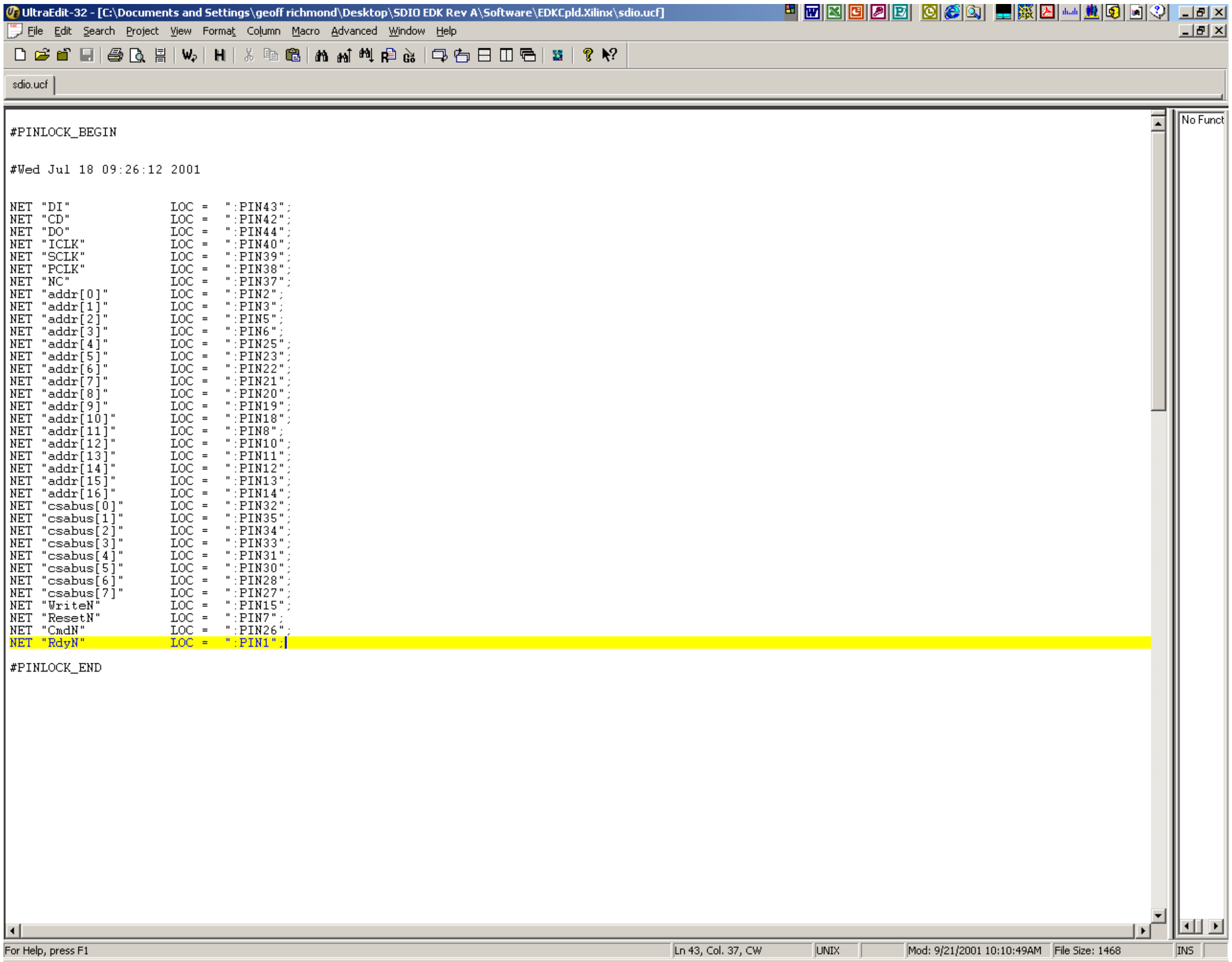
The Project Navigator should look similar to the following:



```
//////////////////////////////////////////////////////////////////////
// Filename:   SDIO_CPLD.v
// Author:     Geoff Richmond
//
// Description:   CPLD implementation for interfacing PIC
//             microcontroller to SDIO controller.
//
//////////////////////////////////////////////////////////////////////
`define  FWIDTH      8     // Width of the FIFO - 8 bits

`define  AWIDTH      17    // Width of the output Address Bus
`define  DWIDTH      8     // Width of the Data In Bus to load ADDR
`define  BSWIDTH     3     // Bank Select Width (bits necessary to
                           // select each address controller register)

`define  SDEPTH      8     // Depth of the SPI shift register (bits)
`define  CWIDTH      4     // Width of SPI shift counter, 2^CWIDTH = SDEPTH


`include "fifo.v"
`include "addrctlr.v"
`include "spi_shift_out.v"
`include "state_ctlr.v"

module SDIO(   csabus,
            ICLK,
            SCLK,
            PCLK,
            WriteN,
            RdyN,
            CmdN,
            DI,
            DO,
            CD,
```

```
iSE Auto-Make Log File
----------------------

Starting: 'C:\Xilinx_WebPACK\bin\nt\checkini.exe -err=automake.err -device=BASE C:\Xilinx_WebPACK\data\webpack\xc9000.ini'


Done: completed successfully.
```

The User Constraints File (`sdio.ucf`) associates the I/O names used in the Verilog software with specific pins on the CPLD component. To inspect this file, use a separate text editor.

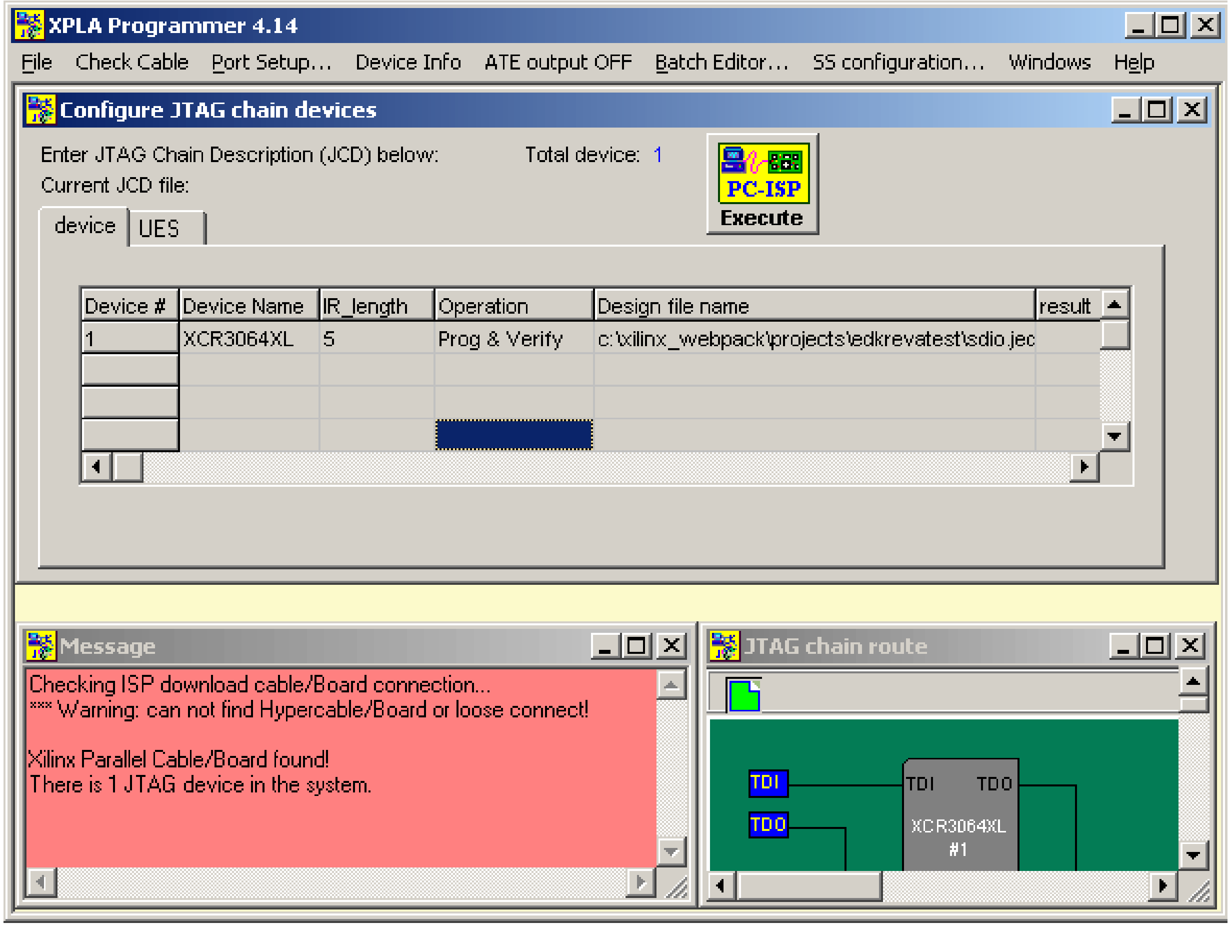

UltraEdit-32 - [C:\Documents and Settings\geoff richmond\Desktop\SDIO EDK Rev A\Software\EDKCpld.Xilinx\sdio.ucf]
File Edit Search Project View Format Column Macro Advanced Window Help
sdio.ucf
No Funct
For Help, press F1
Ln 43, Col. 37, CW
UNIX
Mod: 9/21/2001 10:10:49AM
File Size: 1468
INS

```
#PINLOCK_BEGIN

#Wed Jul 18 09:26:12 2001

NET "DI"             LOC =   ":PIN43";
NET "CD"             LOC =   ":PIN42";
NET "DO"             LOC =   ":PIN44";
NET "ICLK"           LOC =   ":PIN40";
NET "SCLK"           LOC =   ":PIN39";
NET "PCLK"           LOC =   ":PIN38";
NET "NC"             LOC =   ":PIN37";
NET "addr[0]"        LOC =   ":PIN2";
NET "addr[1]"        LOC =   ":PIN3";
NET "addr[2]"        LOC =   ":PIN5";
NET "addr[3]"        LOC =   ":PIN6";
NET "addr[4]"        LOC =   ":PIN25";
NET "addr[5]"        LOC =   ":PIN23";
NET "addr[6]"        LOC =   ":PIN22";
NET "addr[7]"        LOC =   ":PIN21";
NET "addr[8]"        LOC =   ":PIN20";
NET "addr[9]"        LOC =   ":PIN19";
NET "addr[10]"       LOC =   ":PIN18";
NET "addr[11]"       LOC =   ":PIN8";
NET "addr[12]"       LOC =   ":PIN10";
NET "addr[13]"       LOC =   ":PIN11";
NET "addr[14]"       LOC =   ":PIN12";
NET "addr[15]"       LOC =   ":PIN13";
NET "addr[16]"       LOC =   ":PIN14";
NET "csabus[0]"      LOC =   ":PIN32";
NET "csabus[1]"      LOC =   ":PIN35";
NET "csabus[2]"      LOC =   ":PIN34";
NET "csabus[3]"      LOC =   ":PIN33";
NET "csabus[4]"      LOC =   ":PIN31";
NET "csabus[5]"      LOC =   ":PIN30";
NET "csabus[6]"      LOC =   ":PIN28";
NET "csabus[7]"      LOC =   ":PIN27";
NET "WriteN"         LOC =   ":PIN15";
NET "ResetN"         LOC =   ":PIN7";
NET "CmdN"           LOC =   ":PIN26";
NET "RdyN"           LOC =   ":PIN1";

#PINLOCK_END
```

To compile the Verilog code and implement the design, use the WebPACK Project navigator as follows:

1. In the upper left-hand box highlight “SDIO (sdio_cpld.v)”.
2. In the lower left is the “Process View” box. Right-click on “Implement Design”.
3. Select “Rerun All”.

The compilation and implementation should succeed with only warnings about unused pins.



Enter JTAG Chain Description (JCD) below: Total device: 1
Current JCD file:

Device #	Device Name	IR_length	Operation	Design file name	result
1	XCR3064XL	5	Prog & Verify	c:\xilinx_webpack\projects\edkrevatest\sdio.jec	

```
Checking ISP download cable/Board connection...
*** Warning: can not find Hypercable/Board or loose connect!

Xilinx Parallel Cable/Board found!
There is 1 JTAG device in the system.
```

To upload the CPLD program to the XCR3128XL CPLD, perform the following:

1. Remove EDK card from Palm handheld.
2. Connect the Xilinx Parallel Cable III JTAG Programmer to programmer interface J2.
3. Apply +3.3 V to the VDD pin, connecting VSS to the negative supply terminal.

   Note that when the EDK card is inserted into a Palm handheld’s SD slot, the handheld will apply +3.3 V. This is the recommended operating and programming voltage that should be applied to VDD (+ terminal) and VSS (- terminal).

WARNING! The EDK card has a maximum voltage rating of 3.6V. **Do not exceed this voltage**.

4. In the WebPACK Project Navigator select the Process View tab and then double-click on the "XPLA Programmer" entry. This will start the XPLA Programmer application.
5. Under the column labeled "Device Name", double-click on the first cell and then select "XCR3128XL".
6. Under the column labeled "Operation", double-click on the first cell then select "Prog & Verify".
7. Under the column labeled "Design file name", double-click on the first cell and then select the `sdio.jed` file. This is the JEDEC file that will be uploaded to the CPLD.
8. Press the Execute button to program the part.
9. After successful programming, turn off power and disconnect all power and programmer leads.

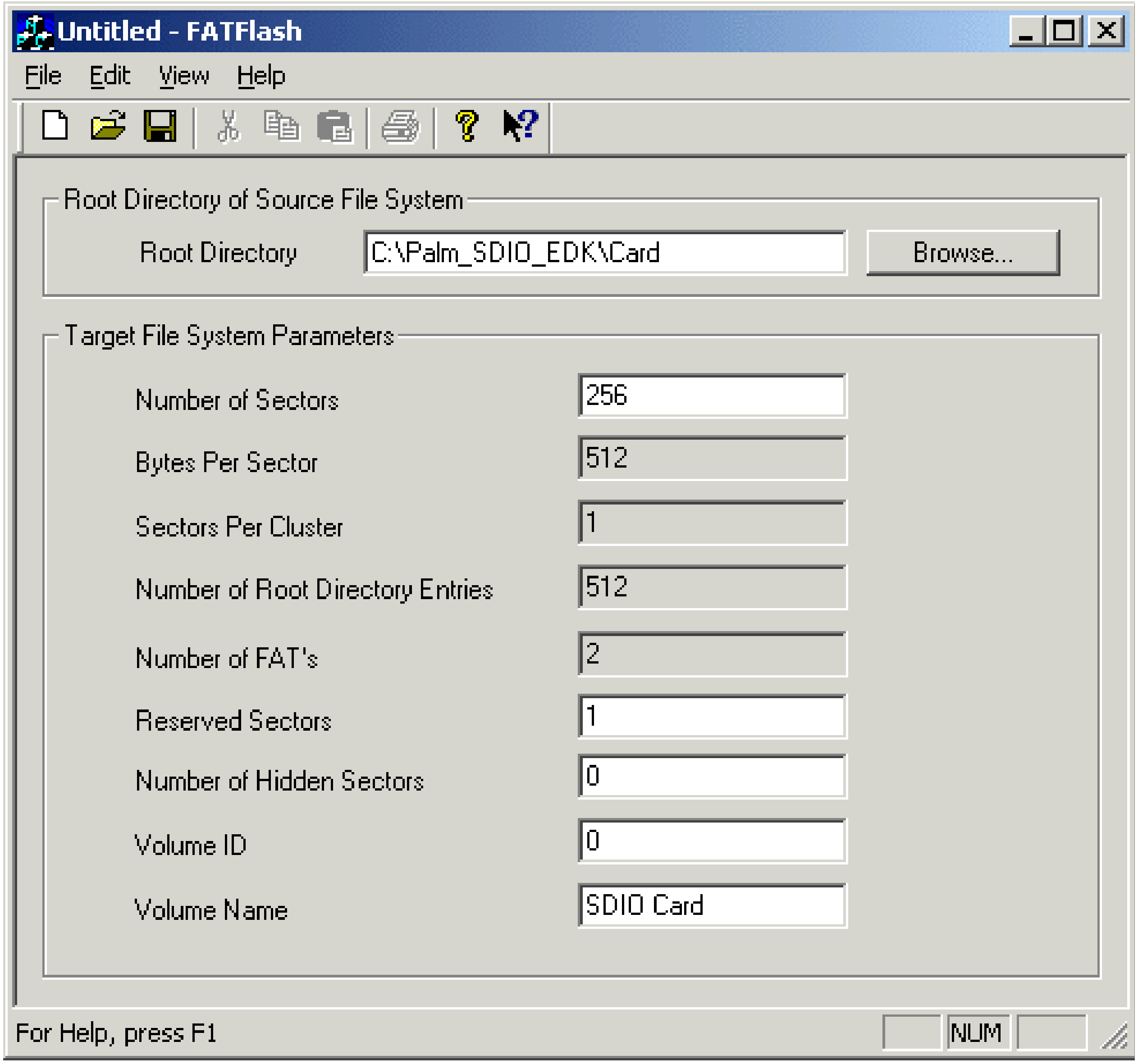
This completes the programming of the CPLD.

Programming the EDK Flash Memory

The EDK card includes 128 KB of programmable flash memory. This memory is used as the Function 1 CSA and contains a FAT12 file system image containing the EDK Demo program. This memory can be reprogrammed using the Demo program's Program Flash Image function.

Before the Program Flash Image function can be used, a valid memory image file must be created and installed on the handheld. To create this memory image, use the FATFlash program included in the EDK download. Figure 4 shows the FATFlash user interface.

Figure 4 FATFlash user interface



Root Directory of Source File System

Root Directory: C:\Palm_SDIO_EDK\Card

Target File System Parameters

Number of Sectors: 256
Bytes Per Sector: 512
Sectors Per Cluster: 1
Number of Root Directory Entries: 512
Number of FAT's: 2
Reserved Sectors: 1
Number of Hidden Sectors: 0
Volume ID: 0
Volume Name: SDIO Card

FATFlash creates a binary image representing a FAT12 file system. You can configure the size of the image (number of bytes), allowing FATFlash to create file system images for various ROM sizes. In the case of the EDK card, the size of the file system created should be the default of 128 KB (256 sectors, 512 bytes per sector). If you wish to adjust the size of the file system image created, adjust the number of sectors.

To create a file system image and program it into the EDK card, do the following:

1. Create a folder on your desktop computer that contains all files and data organized as it should appear in the file system image.
2. Start FATFlash and specify the folder created in step 1., above, as the Root Directory.
3. Adjust the Target File System Parameters if necessary. Note that FATFlash initially sets each parameter to a default value compatible with the flash memory on the EDK card, so you won't need to adjust the parameter values if you are programming the EDK card. See "Target File System Parameters" on page 30 for details on the purpose of each parameter.
4. Select Export File System Image from the File menu. In the dialog that appears, set the file type to "Palm OS File Stream (*.pdb)", supply a filename, and click Save.
5. Assuming that the file is created without error, a confirmation dialog appears. This dialog indicates the full path of the newly-created PDB file.
6. Double-click on the newly-created PDB file to start the Palm Desktop Install Tool and schedule the PDB file for installation on the handheld.
7. Place the Palm handheld into its cradle and perform a HotSync operation.
8. Remove the handheld from its cradle and insert the EDK card into the handheld. This will start the EDK Demo program.
9. Tap Program Flash Image in the EDK Demo program's Options menu.

This will erase the EDK card's flash memory and program the new file system image.

IMPORTANT: Do not remove the EDK card while it is being reprogrammed. As a reminder, a “Please Wait” dialog will remain on the screen until programming is complete.

Complete source code for the EDKDemo program is included in the download package from Palm.

Export Filetypes

FATFlash can create a file system image either as a Palm database file (`.pdb`) or as a binary image file (`.bin`). The Palm database file contains a representation of the file system image that can be loaded on a Palm handheld and accessed using the Palm OS File Streaming APIs. The binary image file contains a binary image of the resulting file system suitable for programming into a ROM. The binary image may be more suitable for custom SDIO card developers.

Target File System Parameters

FATFlash displays the following target file system parameters, which specify how the file system image will be generated:

Number of Sectors Adjust this parameter to vary the total size of the file system image. The default is 256 sectors (each sector is 512 bytes), producing a 128 KB image.

Bytes Per Sector The SD specification specifies 512 bytes per sector. This parameter cannot be changed.

Sectors Per Cluster This parameter is currently set to 1 sector per cluster, and cannot be changed.

Number of Root Directory Entries:
Set to 512, per the SD specification. This parameter cannot be changed.

Number of FAT's: Set to 2. This parameter cannot be changed.

Reserved Sectors: By default, one sector is reserved for the Partition Boot Record. This value can be increased as necessary.

Number of Hidden Sectors
No sectors are hidden by default. You can increase this value if necessary.

Volume ID
An identifier, such as a serial number, to be set in the Partition Boot Record.

Volume Name
Volume name of the resultant file system. This name is displayed in the Palm OS Launcher when the contents of the card are displayed. This field as a maximum length of 11 characters.

FATFlash File Menu

The following commands are listed in the FATFlash application's File menu:

Command	Function
New	Reset the FATFlash configuration to its default state.
Set Root Directory	Specify the root directory from which the desired card image should be created.
Open Configuration File	Load a previously saved FATFlash configuration.
Save Configuration File	Save the current configuration.
Save Configuration File As	Save a copy of the current configuration.
Export File System Image	Create a file system image and save it to a specified file.
Exit	Exit FATFlash

Creating File System Images

To create a file system image that can be programmed into the EDK card's flash memory, ensure that you specify "Palm OS File Stream" as the filetype of the exported file system image. This will cause

FATFlash to create a Palm OS file stream database containing the file system image that can be installed to the Palm device using the Palm Desktop Install Tool. The EDK Demo program's Program Flash Image function can then be used to transfer the file stream database into the flash memory on the EDK card.

The following directories should be present on any file system to be read by Palm OS:

Directory	Purpose
`\PALM`	Root directory for all Palm OS information. To preserve space in the root directory, this entry should be all upper-case. Any lower-case entries require a long filename, which consumes more than one root directory entry.
`\PALM\LAUNCHER`	Directory containing all Palm OS applications (PRCs) that should appear in the Launcher.

If you have a Palm OS application that is to be auto-launched upon card insertion, name the application's PRC file `START.PRC` and place it in the `\PALM` directory. (Note that the name of the PRC file isn't what is displayed in the Palm OS Launcher.) If you want the auto-launched application to also appear when the user displays the card's contents in the Launcher, place a second copy of the PRC in the `\PALM\LAUNCHER` directory.

For more details on how Palm OS handles files and file systems on expansion cards, or for information on file streams, see the Expansion chapter in the *Palm OS Programmer's Companion*.

Tracing SDIO Commands

Using the EDK card and a debug SDIO slot driver on a Palm handheld, you can view a trace of SDIO commands sent and received during card operation. This functionality can be very useful in developing and debugging your custom SDIO cards.

Listing 10 shows a typical identification and initialization sequence when the EDK card is inserted into a Palm handheld.

Listing 10 EDK card initialization and identification

```
Spi Clock actualFrequency=259072 Hz
Power ON card procedure
 turn On card
Spi Clock actualFrequency=259072 Hz

// Set SPI mode
CMD0, argument=0x00000000, response[1]= 0x00

// Read OCR
CMD5, argument=0x00000000, response[5]= 0x00 0x90 0xFF 0x80 0x00
SD I/O Only Card with 1 function
CMD5, argument=0x00180000, response[5]= 0x00 0x90 0xFF 0x80 0x00
Spi Clock actualFrequency=259072 Hz
Power Card procedure Done

Card Identification Begin.

// Read CCCR
CMD53, argument=0x04000010, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Incrementing Address 0x00000000  Hex
data[0x0010]=
 00 00 00 00 00 00 00 00 00 00 10 00 00 00 00 00

// Walk Function 0 tuples
CMD52, argument=0x00200000, response[2]= 0x00 0x21
CMD52, argument=0x00200200, response[2]= 0x00 0x02
CMD52, argument=0x00200800, response[2]= 0x00 0x22
CMD52, argument=0x00200A00, response[2]= 0x00 0x04
CMD52, argument=0x00201400, response[2]= 0x00 0x20
CMD52, argument=0x00201600, response[2]= 0x00 0x04
CMD52, argument=0x00202000, response[2]= 0x00 0x15
CMD52, argument=0x00202200, response[2]= 0x00 0x24
CMD52, argument=0x00206C00, response[2]= 0x00 0xFF

// Read TPLFID_FUNCTION tuple for Function 0
CMD53, argument=0x04200C04, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Incrementing Address 0x00001006  Hex
data[0x0004]=
 00 80 00 78
SDIO, function 0, tuple CISTPL_FUNCE found.
SDIO, function 0, TPLFE_FN0_BLK_SIZE = 128
SDIO, function 0, TPLFE_MAX_TRAN_SPEED = 800000
```

```
Spi Clock actualFrequency=518144 Hz

// Read CISTPL_FUNCID tuple for Function 0
CMD52, argument=0x00200400, response[2]= 0x00 0x0C
CMD52, argument=0x00200600, response[2]= 0x00 0x00

// Read FBR 1
CMD53, argument=0x0402000C, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Incrementing Address 0x00000100  Hex
data[0x000C]=
 40 00 00 00 00 00 00 00 00 00 20 00

// Walk Function 1 tuples
CMD52, argument=0x00400000, response[2]= 0x00 0x21
CMD52, argument=0x00400200, response[2]= 0x00 0x02
CMD52, argument=0x00400800, response[2]= 0x00 0x22
CMD52, argument=0x00400A00, response[2]= 0x00 0x1C
CMD52, argument=0x00404400, response[2]= 0x00 0xFF

// Read CISTPL_FUNCID tuple for Function 1
CMD52, argument=0x00400400, response[2]= 0x00 0x0C
CMD52, argument=0x00400600, response[2]= 0x00 0x00

// Read TPLFID_FUNCTION tuple for Function 1
CMD53, argument=0x04400C1C, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Incrementing Address 0x00002006  Hex
data[0x001C]=
 01 01 00 00 00 00 00 00 00 02 00 03 80 00 00 01 FF 00 08 0A 0F 01 01 01 00 00
00 00
SDIO function 1 TPLFE_MAX_BLOCK_SIZE = 0
SDIO function 1 CCCR SMB = 0 (no block mode transfers)
SDIO function 1 FBR - CSA supported=1, TPLFE_CSA_SIZE = 131072

// Enable Function 1 CSA
CMD52, argument=0x00020000, response[2]= 0x00 0x40
CMD52, argument=0x880200C0, response[2]= 0x00 0xC0
SDIO function 1File system enabled.

// Read Manufacturer ID and OEM ID
CMD53, argument=0x04201804, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Incrementing Address 0x0000100C  Hex
data[0x0004]=
 96 02 00 01
SDIO Card function 1 Manufacturer ID = 0x00000296 OEM ID = 0x00000100

// Read 512 byte PBR
CMD52, argument=0x80021800, response[2]= 0x00 0x00
CMD52, argument=0x80021A00, response[2]= 0x00 0x00
```

```
CMD52, argument=0x80021C00, response[2]= 0x00 0x00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 EB 3C 90 4D 53 44 4F 53 35 2E 30 00 02 01 01 00 02 00 02 FF 00 F8 01 00 12 00
02 00 00 00 00 00
 00 00 00 00 00 00 29 B6 69 DB 70 53 44 49 4F 20 45 44 4B 20 31 20 46 41 54 31
32 20 20 20 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 55 AA
CMD53, argument=0x04202423, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Incrementing Address 0x00001012  Hex
data[0x0023]=
 07 00 50 61 6C 6D 2C 20 49 6E 63 2E 00 53 44 49 4F 20 44 65 76 65 6C 6F 70 6D
65 6E 74 20 43 61
 72 64 2E

// Read Manufacturer ID and OEM ID
CMD53, argument=0x04201804, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Incrementing Address 0x0000100C  Hex
data[0x0004]=
 96 02 00 01

// Read FAT 1
CMD52, argument=0x80021800, response[2]= 0x00 0x00
CMD52, argument=0x80021A02, response[2]= 0x00 0x02
CMD52, argument=0x80021C00, response[2]= 0x00 0x00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 F8 FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
```

```
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00

// Read root directory (including volume name)
CMD52, argument=0x80021800, response[2]= 0x00 0x00
CMD52, argument=0x80021A06, response[2]= 0x00 0x06
CMD52, argument=0x80021C00, response[2]= 0x00 0x00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 53 44 49 4F 20 43 41 52 44 20 20 08 00 00 00 00 00 00 00 00 00 00 A0 95 24 2B
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
```

```
CMD53, argument=0x00021E80, response[2]= 0x00 0x00
 IO_RW_EXTENDED function 0 byte read Fixed Address 0x0000010F  Hex
data[0x0080]=
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
Power OFF card procedure
 turn OFF card
Power Card procedure Done
```

NOTE: This listing is for illustrative purposes only. It may not reflect exactly the commands issued during EDK card insertion. Additional comments have been added to the listing to clarify the events in the identification and initialization process. Please refer to the *SDIO Specification* for a detailed explanation of the card identification and initialization process.

For additional information on the command tracing feature of the SDIO slot driver refer to *Developing SDIO Applications for Palm Handhelds*, which is available through the Plugged In program.

Installing the Debug SDIO Slot Driver

In order to perform command tracing, a debug version of the SDIO slot driver must be resident on your handheld. A debug version of the SDIO slot driver is available through the Plugged In program.

Install it as follows:

1. If necessary, uninstall the existing RAM-resident `SlotDriver:SDIO-sdsd` slot driver from your Palm handheld by performing a hard reset. This step is only required if the slot driver is resident in RAM.
2. Install the debug version of the slot driver using the standard Palm Desktop Install Tool.

3. Perform a soft reset of the device to activate the newly-installed slot driver.

Enabling Command Tracing

You can now enable command tracing from within the EDK Demo program. To do this, simply:

1. Insert the EDK card into the SD slot of your Palm handheld to start the EDK Demo application.
2. Select SDIO Trace All from the EDK Demo application's Options menu.

NOTE: If SDIO Trace All does not appear in the Options menu, the debug version of the SDIO slot driver is not installed.

A trace of all SDIO commands is now sent to the serial port. You can remove and insert the EDK card, while the EDK Demo application is running, to view the identification and initialization process.

The selected tracing option remains active until you perform a soft reset or exit the EDK Demo program.

Viewing Trace Output

The Palm Debugger is a convenient tool for viewing the trace output. Note that current versions of the Palm Debugger require that you connect to the handheld using the serial port, which means that you must have a serial cradle if you are working with a handheld such as the Palm m500 or m505. The following procedure shows you how to use the Palm Debugger to view trace output:

1. Ensure that the HotSync Manager is not running on the desktop, and that a HotSync operation is not in progress.
2. Start the Palm Debugger and set it to monitor the COM port on the desktop to which the Palm serial cradle is connected.
3. Insert the Palm handheld into the serial cradle.
4. Enable SDIO command tracing as described in "Enabling Command Tracing,"above.

NOTE: If you activate command tracing before placing the handheld is in a serial cradle, all debug messages will be routed to the USB cradle by default. However, since you are not connected to a USB cradle, the software will “lock” forever trying to open a non-existent USB port. To recover from this, either reset the handheld or start a USB debugger on your desktop computer and then place the handheld in the USB cradle.

The Metrowerks debugger console window can also be used to monitor trace output, but note that the formatting of the output can be affected by display of CR/LF information.

For more information on using Palm Debugger to view the trace of commands, see “Debugging Your SDIO Card” in *Developing SDIO Peripherals for Palm Handhelds*

Design Tips and Tricks

Developing for a CPLD requires patience and the willingness to attempt multiple solutions for a single problem to ensure the optimal solution for the part. Sometimes apparent simplifications will result in larger code, possibly preventing the firmware from fitting into the part. As well, apparent complications will occasionally simplify the internal design, perhaps allowing a design to fit into a constrained part when previously it did not fit.

Remember that every mA of current drawn by a SDIO peripheral usually comes from the battery of the handheld. While a large (up to 200mA) current drain may fall within the SDIO 1.0 specification’s limits, this will present a poor user experience on any battery-powered handheld, as the operating life of the handheld will be significantly reduced. Whenever possible try to maintain a current draw of less than 100 mA for any SDIO peripheral that is to be used with Palm handhelds.

If you are designing a SDIO card for Palm handhelds, refer to the Palm SDIO Mechanical recommendations to ensure that your peripheral will work across multiple product lines.

In order to comply with the SD and SDIO ESD specifications, additional ESD components such as lightning gaps and TranZorbs (silicon avalanche diodes) may be required in all pins connecting to the SD connector.