



## Application Note

# CompactFlash<sup>®</sup> Interface for eZ80AcclaimPlus!<sup>™</sup> MCUs

AN015404-0608



## Abstract

This Application Note describes an interface between Zilog's eZ80AcclaimPlus!<sup>™</sup> microcontroller unit (MCU) and a CompactFlash<sup>®</sup> Storage Card. The CompactFlash card is accessed in all modes of operation, the memory mode, the contiguous I/O mode, the primary I/O mode, the secondary I/O mode, and the TrueIDE mode.

- **Note:** *The source code file associated with this Application Note, AN0154-SC01.zip, is available for download at [www.zilog.com](http://www.zilog.com).*

## eZ80AcclaimPlus!<sup>™</sup> Flash MCU Overview

The eZ80AcclaimPlus! on-chip Flash MCUs are an exceptional value when designing high performance, 8-bit MCU-based systems. With speeds up to 50 MHz and an on-chip Ethernet MAC (eZ80F91 only), you have the performance necessary to execute complex applications quickly and efficiently. Combining Flash and SRAM, eZ80AcclaimPlus! devices provide the memory required to implement communication protocol stacks and achieve flexibility when performing in-system updates of application firmware.

The eZ80AcclaimPlus! Flash MCU can operate in full 24-bit linear mode addressing 16 MB without a Memory Management Unit. Additionally, support for the Z80<sup>®</sup>-compatible mode allows you to execute legacy code within multiple 64 KB memory blocks with minimum modifications. With an external bus supporting eZ80<sup>®</sup>, Z80, Intel, and Motorola bus modes and a rich set of serial communications peripherals, you have several options when interfacing to external devices.

Some of the many applications for eZ80AcclaimPlus! devices include vending machines, point-of-sale terminals, security systems, automation, communications, industrial control and facility monitoring, and remote control.

## Discussion

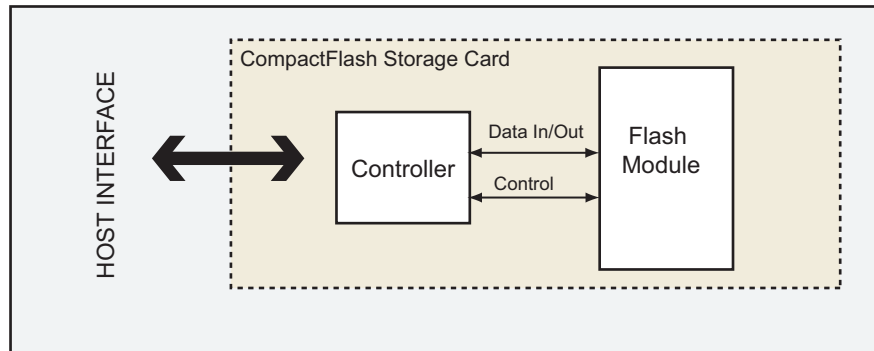
The CompactFlash interface is a vendor-independent specification that can be implemented to develop CompactFlash products that are compatible with a wide range of user applications and across a gamut of manufacturers.

The early CompactFlash cards were meant exclusively for Flash data storage; however, currently under the CF+ specification, they support various I/O devices, such as Ethernet cards and fax/modem cards, to name a few.

## CompactFlash<sup>®</sup> Storage Card

Figure 1 on page 2 displays a CompactFlash Storage Card that contains a dedicated controller and Flash memory module(s). The controller interfaces with a host system, allowing data to be written to and read from the Flash memory module(s).

The host interface consists of address lines from A0:A10, data lines from D0:D15, and control signals.



**Figure 1. Structure of a CompactFlash® Storage Card**

The address space requirement for interfacing a CompactFlash Storage Card is just 2 KB, irrespective of the storage capacity of the card. The data bus can be interfaced as an 8-bit or a 16-bit bus, depending on the host support. The Flash modules are not directly accessible to the host, even at the byte level. The controller transfers the data to and from the Flash modules in blocks of 512 bytes. The host essentially communicates with the controller, which contains control/status registers and a data buffer mapped within the 2 KB address space of the host.

### Description of the CompactFlash® Storage Card

The CompactFlash Storage Card used in this application is a 64 MB card from [Kingston Technology](#). The memory internally is divided into three different sections: the attribute memory, the command/status tuple, and the storage memory.

**Attribute Memory**—The Attribute memory is a set of registers that store card identification and configuration information. The card can be configured in the memory and I/O modes using these registers. The starting address of these registers is fixed at an offset of 0x200.

**Command/Status Tuple**—The Command/Status tuple are sequentially addressed control registers with different starting addresses depending upon the

mode of operation. When a host writes to the tuple, the tuple operates as a set of registers that holds the command parameters and the command. It is essential to write the command parameters before the command because the controller executes the command according to the parameters present in other registers.

Similarly, when a host reads to the command/status tuple, the tuple operates as a set of registers that holds the status information about the last executed command. The start address of the command/status tuple is 0x000. The first register at address 0x000 is for the data register, which does not contribute to the actual command/status information.

**Storage Memory**—Storage memory is a large area of Flash memory. Access to this space is handled by the controller in a manner that is transparent to the host. The address space requirement for interfacing a CompactFlash Storage Card is just 2 KB, irrespective of the storage capacity of the card.

► **Note:** *The addresses of attribute memory, control/status, and data registers lie within the 2 KB space.*

## Modes of Operation

This section discusses the several modes in which the CompactFlash operates.

**Memory Mode**—The CompactFlash card enters this mode at Power On or RESET, if the  $\overline{OE}$  pin is held at a logic HIGH (1).

**TrueIDE Mode**—This mode becomes the default mode of operation for the CompactFlash if the  $\overline{OE}$  pin is grounded at the time of Power On. The TrueIDE mode is probably the most suitable mode for embedded operations because it uses only three out of 11 address lines for addressing the CompactFlash Storage Card.

**I/O Mode(s)**—There are three distinct I/O modes: the primary I/O mode, the secondary I/O mode, and the contiguous I/O mode. At Power On, the CompactFlash card enters the TrueIDE mode or the Memory mode depending on the status of the  $\overline{OE}$  pin. To enter into any of the I/O modes, it is necessary to modify the card configuration register (in the

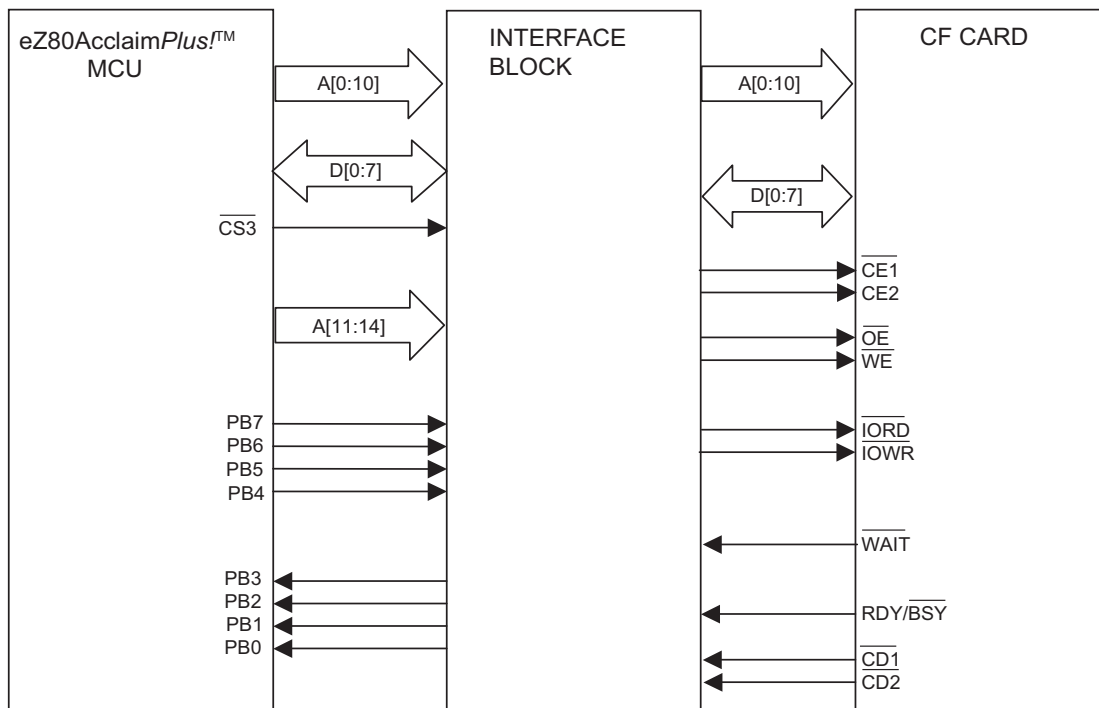
attribute memory space) accordingly, and reset the card.

## Developing the CompactFlash® Interface

The hardware architecture and the software implementation to develop the CompactFlash Storage Card interface for the eZ80AcclaimPlus! MCU is described in this section.

## Hardware Architecture

The CompactFlash Storage Card consists of pins that operate according to the mode in which the card is operated, resulting in different interfaces for different modes. The architecture displayed in [Figure 2](#) considers a superset of these requirements to display all the three modes with the same hardware setup. The block diagram in [Figure 2](#) is the basis for the schematics in [Appendix A—Schematics](#) on page 12.



**Figure 2. System Connections for the CompactFlash® Interface with an eZ80AcclaimPlus! MCU**

The hardware is designed to interface with the CompactFlash in the modes of operation mentioned previously. It is therefore necessary to understand the hardware requirements for entering and operating the different modes. These requirements are described under two domains: requirements common to all the modes of operation, and requirements that vary for different modes. The signals required for all the modes of operation of the CompactFlash are described first, followed by signals specific to each mode.

### Signals Required for All Modes of Operation of CompactFlash®

At the outset, it is necessary to determine that a card is present in the slot for operation. Two signals are located at the two extremes of the CompactFlash connector to facilitate card detection ( $\overline{CD1}$ ,  $\overline{CD2}$ ) and correct insertion. These signals are pulled low inside the card. An OR-gate (74LVC32AD) combines these *card detect* signals to derive one signal that is connected to the Port B pin0 (PB0) of the eZ80F91 MCU. PB0 is polled by the software to determine if the CompactFlash card is present.

The eZ80F91 data bus is byte wide, therefore the D[0:7] data bus is connected to the corresponding data lines/pins of the CompactFlash. The CompactFlash HIGH data bus, D[8:15], is not connected.

The eZ80F91 address bus, A[0:23], is described as different groups of address lines.

The 11 least-significant bits (lsb) of the eZ80F91 address bus, A[0:10], are connected directly to A[0:10] of the CompactFlash connector. This address area defines a space for 2 KB (2048) addresses. [Table 1](#) lists the usage of address lines, A[11:14].

**Table 1. Connection Details between eZ80F91 Address Bits and CompactFlash® Pins**

F91 Address Bit	CompactFlash® Pin (number)
A11	$\overline{REG}$
A12	$\overline{CE1}/\overline{CE1}/\overline{CS0}$
A13	$\overline{CE2}/\overline{CE2}/\overline{CS1}$
A14	CSEL
A15	Not used

The eight most-significant bits (msb), A[16:23], decide the *chip select*, providing you the flexibility to map the 2 KB address space anywhere within the 16 MB area supported by the eZ80F91 device. For this CompactFlash Interface application, the A[16:23] byte is assigned the value of 0xA8.

As per the CompactFlash specification, 10 K $\Omega$  resistors are used to pull up each of the following signals to  $V_{DD}$ :

- READY
- $\overline{INPACK}$
- WAIT
- WP
- $\overline{VS1}$
- VS2
- BVD1
- BVD2

### Mode Specific Signals

Some of the control signals in the CompactFlash card are used for multiple purposes while some of the signals differ for different modes. All of these signals require additional hardware and pins from the CompactFlash host.

Switching between the memory mode and any of the I/O modes is achieved through the software, see [Modes of Operation](#) on page 3. However, such a

mode change also requires a change in hardware. In the memory mode, the  $\overline{RD}$  and  $\overline{WR}$  strobes from the host are connected to the  $\overline{OE}$  and  $\overline{WE}$  signals of the CompactFlash card. In all other modes, the host signals are connected to  $\overline{IORD}$  and  $\overline{IOWR}$  signals of the CompactFlash card.

The switching functionality is achieved through the use of a de-multiplexer (de-mux). The chip select (as decided by the A[16:23]) and PB7 pin control the action of the de-mux. When the chip select is asserted, the PB7 pin is HIGH for the memory mode, and LOW for other modes of CompactFlash operation.

For all modes other than the memory mode, the de-mux routes  $\overline{RD}$  and  $\overline{WR}$  signals from the eZ80F91 to  $\overline{IORD}$  and  $\overline{IOWR}$ , respectively. For the memory mode, the de-mux routes the  $\overline{RD}$  and  $\overline{WR}$  signals directly to the CompactFlash  $\overline{OE}$  and  $\overline{WE}$  signals respectively.

**Table 2. Demultiplexer Logic Truth Table**

CS x	PB 7	$\overline{WR}$	$\overline{RD}$	Action	Remarks
0	0	1	0	$\overline{IORD} = \overline{RD}$	
0	0	0	1	$\overline{IOWR} = \overline{WR}$	
0	1	1	0	$\overline{MEM\_MODE} = \overline{RD}$	If PB6 = 1, then $\overline{OE} = \overline{MEM\_MODE}$
0	1	0	1	$\overline{WE} = \overline{WR}$	

**Resetting the CompactFlash® Storage Card at Run-time**—The CompactFlash interface can switch between the memory and the I/O modes at run-time. To achieve this functionality, it is required to reset the CompactFlash Storage Card. In the application described in this document the eZ80F91 port pin, PB4, is connected directly to the CompactFlash RESET pin. By toggling PB4, the CompactFlash card can be RESET at any point of time.

**Controlling the OE pin at RESET**—The CompactFlash Storage Card enters the TrueIDE mode if the  $\overline{OE}$  pin is grounded at RESET. The CompactFlash Storage Card enters the memory mode if the  $\overline{OE}$  pin is at logic 1 at RESET. Due to this functionality, the status of  $\overline{OE}$  pin must be controlled at the time of RESET. To control the status of the  $\overline{OE}$  pin, a jumper (J17) is provided at the  $\overline{OE}$  pin. It is advisable that the jumper (J17) be inserted at Power On to allow the CF Storage Card to enter the default mode, which is the TrueIDE mode (see [Switching between Modes](#) on page 7).

Table 3 lists the port B GPIO lines used to interface the CompactFlash control signals.

**Table 3. Port B GPIO Interface to CompactFlash® Control Signals**

Bit	Direction	Functionality
PB7	Output	To route ( $\overline{RD}$ , $\overline{WR}$ ) from eZ80F91 to either ( $\overline{OE}$ , $\overline{WE}$ ) or ( $\overline{IORD}$ , $\overline{IOWR}$ ) depending on the memory or IO/IDE mode respectively
PB6	Output	To select the True IDE mode of operation for the CompactFlash
PB5	Output	To indicate successful transfers/debug
PB4	Output	To RESET the CompactFlash card through software – useful in IDE Master/Slave modes
PB3	Input	BVD1/ $\overline{STSCHG}$ / $\overline{PDIAG}$
PB2	Input	BVD2/ $\overline{SPKR}$ / $\overline{DASP}$
PB1	Input	RDY $\overline{BSY}$ / $\overline{IREQ}$ / $\overline{INTRQ}$
PB0	Input	Combined Card Detect signal

### The Interface Block

The CompactFlash card–eZ80AcclaimPlus! interface logic comprises of a single OR-gate, a de-mux (74LVC139AD), and a host of tri-state buffers.(see [Figure 2](#) on page 3).

Table 4 lists the derived addresses for the CompactFlash registers based on the discussions in the sections on [Signals Required for All Modes of Operation of CompactFlash®](#) on page 4, and [Mode](#)

[Specific Signals](#) on page 4. The buffers and a few jumpers are provided for future developments or for porting the interface to other Zilog® processors.

**Table 4. Derived Addresses for the CompactFlash® Registers**

Mode	A23-16	A15-12	A11-8	A7-4	A3-0	Complete Range
Memory	A8h	xx10	10xx	xxxx	0h – 7h	A82800h – A82807h
Contiguous I/O	A8h	xx10	0x00	0000	0h – 7h	A82000h – A82007h
Primary I/O	A8h	xx10	0x01	1111	0h – 7h	A821F0h – A821F7h
Secondary I/O	A8h	xx10	0x01	0111	0h – 7h	A82170h – A82177h

x : Indicates 'don't care' condition. All 'x' are replaced by zeros.

h : Indicates the preceding value to be a hexadecimal.

## Software Implementation

The CompactFlash Interface software is implemented as a set of API functions to perform the following operations:

- Initialize and set the CompactFlash mode
- Write to a CompactFlash sector
- Identify the CompactFlash drive
- Set features for the CompactFlash card in terms of mode specifications
- Read from a CompactFlash sector
- Erase a CompactFlash sector
- Fill even and odd bytes of a CompactFlash sector

For details on the API functions, see [Appendix C—API Description](#) on page 22. In addition to the API functions, several supporting functions are implemented that may be useful while using the CompactFlash card. The supporting functions are as follows:

- Software block data transfer (DMA operation)
- Wait routines (for various CompactFlash specified delays)
- Card detection/Error checking
- Function to receive the sector, head, and cylinder information

The CompactFlash interface works with the command tuple, see [Command/Status Tuple](#) on page 2. The registers in the tuple are addressed in a sequential manner. Therefore, it is sufficient to determine the address of the first register in a specified mode. These addresses are hardware- and mode-dependent and are hard-coded in the software for every mode.

The current CompactFlash Interface application supports all the modes of operation described in the [Modes of Operation](#) on page 3. When you select a particular mode of operation, the addresses, being mode-dependent, are loaded into the appropriate pointer and you are prompted to perform an operation on the CompactFlash. When the command to perform an operation is received, the corresponding



registers in the tuple are loaded with the required command parameters. This is followed by a block transfer of data to/from the CompactFlash.

Two buffers, of 512 bytes each, are implemented in the CompactFlash Interface software to transmit and receive data to/from the CompactFlash.

### Switching between Modes

The procedure for accessing the modes divides the CF modes into two groups. The memory mode and the various I/O modes can be entered through software at run-time. At Power On, the CompactFlash Storage Card enters the TrueIDE mode. Instructions for switching from the TrueIDE mode to other modes are displayed in the HyperTerminal application. It is not possible to revert back to the TrueIDE mode from other modes, by executing the commands displayed in the HyperTerminal application

► **Note:** *The TrueIDE mode needs a power recycle (On-Off) with the  $\overline{OE}$  pin grounded. Due to this reason, it is not possible to enter the TrueIDE mode through software commands displayed in the HyperTerminal application.*

The default mode of operation for the CompactFlash card is the TrueIDE mode, hence all the commands for this mode can be run directly. The CompactFlash attributes are set to an 8-bit mode. They are marked as the 0<sup>th</sup> drive through the `set_features()` API, such that the CompactFlash card can be switched to any mode of operation.

To switch to the memory mode or any of the I/O modes, the appropriate bits are written in the configuration register as follows:

Bit	I/O Mode
1	Contiguous I/O
2	Primary I/O
3	Secondary I/O

While switching to the I/O mode, the  $\overline{RD}$  and  $\overline{WR}$  signals from the eZ80F91 must be connected to the  $\overline{IORD}$  and  $\overline{IOWR}$  signals of the CompactFlash card.

### Processing the Commands

Commands are issued to the CompactFlash card when it is in the required mode. You need to provide the starting sector, cylinder, and head for the `read_sectors()` and `write_sectors()` APIs. These APIs feed the values provided by you into appropriate command tuple registers and write the command word (0x20 for READ and 0x30 for WRITE) in the command register.

For the READ command, the CompactFlash interface fills the buffer with one sector of data (512 bytes) and prompts you for action through the status registers. The CompactFlash features a FIFO like structure that requires the host (eZ80F91) to read the data from the same location repeatedly.

For a WRITE command, the entire data is first written to the CompactFlash buffer before the CompactFlash interface starts writing the data in the designated sector. When the data is completely written into the sector, the CompactFlash interface informs the host through its status registers.

The algorithm for the `read_sector()` API, which is the sequence of events for reading from a sector, is outlined below:

1. Select the CompactFlash mode.
2. The pointers for the command/status tuple and the read buffers are loaded, by default. This process is transparent to you.
3. Get the sector, cylinder, and head from where reading must begin on the CompactFlash card.
4. Load values to the appropriate command registers in the command tuple.
5. Load the read command in the command register.
6. Wait for the CompactFlash card to be ready.

7. Read 512 bytes from the CompactFlash data register.
8. Indicate success/error.

The algorithm for the `write_sector()` API, which is the sequence of events for writing to a sector, is outlined below:

1. Select the CompactFlash mode.
2. Load the pointers for command/status tuple, and write buffers.
3. Get the data in the write buffer.
4. Get the sector, cylinder, and head from where writing must begin on the CompactFlash card.
5. Load values to the appropriate command registers in the command tuple.
6. Load the write command in the command register.
7. Wait for the CompactFlash card to be ready.
8. Write 512 bytes from the CompactFlash data register.
9. Indicate success/error.

## Testing

This section describes the setup, requirements, and the procedure followed to demonstrate the CompactFlash Interface application using the eZ80F91 MCU.

### Setup

The CompactFlash Storage Card is inserted into the CompactFlash connector on the CompactFlash Interface Board. The eZ80F91 Module is mounted on the JP1 and JP2 connectors on the Interface Board. Jumpers J2, J17, and J21 are ON.

The serial port associated with the HyperTerminal application is connected to the J9 connector on the Interface Board using a serial cable. Either a power supply of 5 V, or four batteries of AA size are connected to the J7 and J14 connectors on the Interface

Board, respectively. Connecting both the sources of power simultaneously does not cause any damage.

► **Note:** *Shorting the J17 jumper drives the OE# pin of the CompactFlash to ground. This is essential to enable the TrueIDE mode.*

## Equipment Used

The following equipment are used:

- eZ80F91 Module.
- CompactFlash Interface Board (developed according to the schematics provided in [Appendix A—Schematics](#) on page 12).
- ZDS II—IDE for eZ80Acclaim! MCUs, v4.8.0.
- CompactFlash Storage Card from Kingston technology with 64 MB memory.
- PC with HyperTerminal application configured to 9600 bps, 8 bit, no-parity, 1 Stop bit, and No flow control.

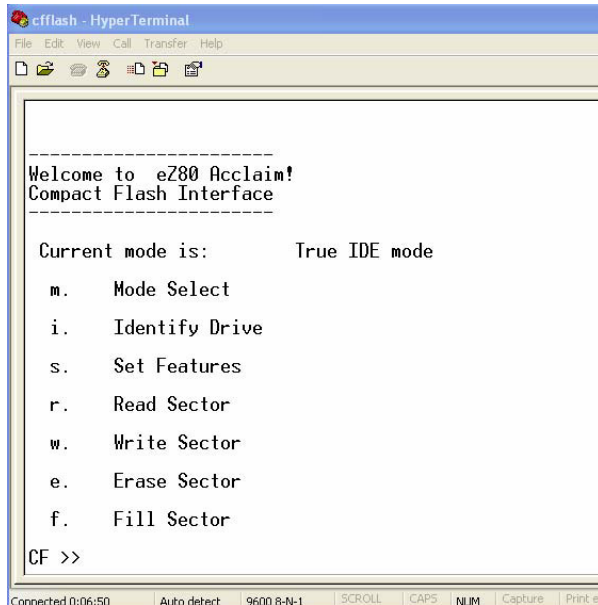
## Test Procedure

Follow the steps below to demonstrate the CompactFlash Interface application:

1. The connections between the eZ80F91 Module, and the CompactFlash Interface Board are made as described in the [Setup](#) section.
2. Launch ZDS II for eZ80Acclaim!; build the program using the application code (`AN0154-SC01.zip` file).
3. Connect ZPAK II to the J8 connector on the CompactFlash Interface Board. Download the code to the on-chip Flash using ZDS II (For more information, refer to *Zilog Developer Studio II - eZ80Acclaim! User Manual (UM0144)*).
4. Launch the HyperTerminal application.



- Execute the code using ZDS II or RESET the Board. The CF >> prompt is displayed in the HyperTerminal application along with the command menu.



```

cflash - HyperTerminal
File Edit View Call Transfer Help
-----
Welcome to eZ80 Acclaim!
Compact Flash Interface
-----
Current mode is:      True IDE mode
m.  Mode Select
i.  Identify Drive
s.  Set Features
r.  Read Sector
w.  Write Sector
e.  Erase Sector
f.  Fill Sector
CF >>

```

**Figure 3. HyperTerminal Application with Command Menu**

- The CompactFlash Storage Card defaults to the TrueIDE mode.
  - To continue in the TrueIDE mode, execute the commands displayed in the HyperTerminal application.
  - To select a different mode, enter *m* in the HyperTerminal console. Follow the instructions displayed in the HyperTerminal application to change the mode of operation.
- Enter *i* to identify the CompactFlash Storage Card. This operation tests the `identify_drive()` API, which reads the CompactFlash parameters. Compare these with the datasheet provided for the card you are using.
- At the CF >> prompt, enter *w* to write to a CompactFlash sector. This operation tests the `write_sector()` API. The write sub-menu appears.

- Enter values for Track, Head, and Sector within the prompted range.
- At the CF >> prompt, enter *r* to read to a CompactFlash sector. This operation tests the `read_sector()` API. The read sub-menu appears.
- Enter the same values used in step 9. Also select the display mode: either hexadecimal or ASCII. The data that was previously written to the CompactFlash sector is read by the eZ80F91 processor and displayed in the HyperTerminal application.

## Test Results

The CompactFlash card was identified with the `identify_drive()` API. For this API the display in the HyperTerminal application is as follows:

```

CF >>  i
-----
Vendor Information: TOSHIBA
THNCF064MMA
-----
Default number of cylinders = 978
Default number of heads     = 4
Default sectors per Track   = 32
Default sectors per card    = 125184
Current number of cylinders = 978
Current number of heads     = 4
Current sectors per Track   = 32
Current capacity in sectors = 1
-----

```

This information received from the CompactFlash card is the same as the CompactFlash specification.

A sector of the CompactFlash is written to, using the `write_sector()` API. For this API the display in the HyperTerminal application is as follows:



```

CF >> w
Please enter the starting sector
(0x01-0x20): 01
Please enter the Head Number (0x0-
0x3): 0
Please enter the Track number
(0x000-0x3d2): 000
----- Write buffer -----
Zilog India is a software design
center with the charter to fulfill
the software needs of Zilog Inc and
those of it's customers by enabling
creative, comprehensive solutions
using Zilog semiconductor family
offerings for new and traditional
markets. Zilog India provides state
of art embedded software solutions
that compliment Zilog semiconductor
solutions with the specific goals of
improving productivity, time-to-
market, affordability, quality and
other such major concerns of our
customers worldwide.- Zilog
----- Write buffer -----

```

---

The same sector was read back using the `read_sector()` API. For this API the display in the HyperTerminal application is as follows:

```

CF >> r
Please enter the starting sector
(0x01-0x20): 01
Please enter the Head Number (0x0-
0x3): 0
Please enter the Track number
(0x000-0x3d2): 000
Please enter the Display mode (0x0:
ASCII -0x1: Hex): 0
----- Read Buffer ASCII -----
Zilog India is a software design
center with the charter to fulfill
the software needs of Zilog Inc and
those of it's customers by enabling
creative, comprehensive solutions
using Zilog semiconductor family
offerings for new and traditional
markets. Zilog India provides state
of art embedded software solutions
that compliment Zilog semiconductor
solutions with the specific goals of
improving productivity, time-to-
market, affordability, quality and
other such major concerns of our
customers worldwide.- Zilog
----- Read Buffer ASCII -----

```

---

Data transfer to and from the CompactFlash card is performed and found to be error free.

## Summary

This Application Note describes an interface between an eZ80AcclaimPlus! MCU and the CompactFlash Storage Card. It provides you with the basic building block to develop a slew of applications using the eZ80AcclaimPlus! webserver, and the CompactFlash Storage Card.

You can benefit from all the advantages that the CompactFlash technology offers, namely a power efficient and flexible way of storing digital data such as web-pages, audio/video files, or simple log data.

## References

The documents associated with eZ80Acclaim!, eZ80AcclaimPlus!, and CompactFlash® products available on [www.zilog.com](http://www.zilog.com) are listed below:

- eZ80® CPU User Manual (UM0077)
- eZ80F91 Product Specification (PS0192)
- Zilog Developer Studio II—eZ80Acclaim! User Manual (UM0144)
- CF+ and CompactFlash® Specification Revision 1.4, available at [www.compactflash.org](http://www.compactflash.org)
- Kingston CompactFlash, MKF-260, available at [www.kingston.com/products/cf\\_white\\_paper.pdf](http://www.kingston.com/products/cf_white_paper.pdf)
- CompactFlash® Storage Card, THNCF064MM, available at [www.toshiba.com/taec](http://www.toshiba.com/taec)

## Appendix A—Schematics

Figure 4 is a schematic diagram displaying the MDS connectors.

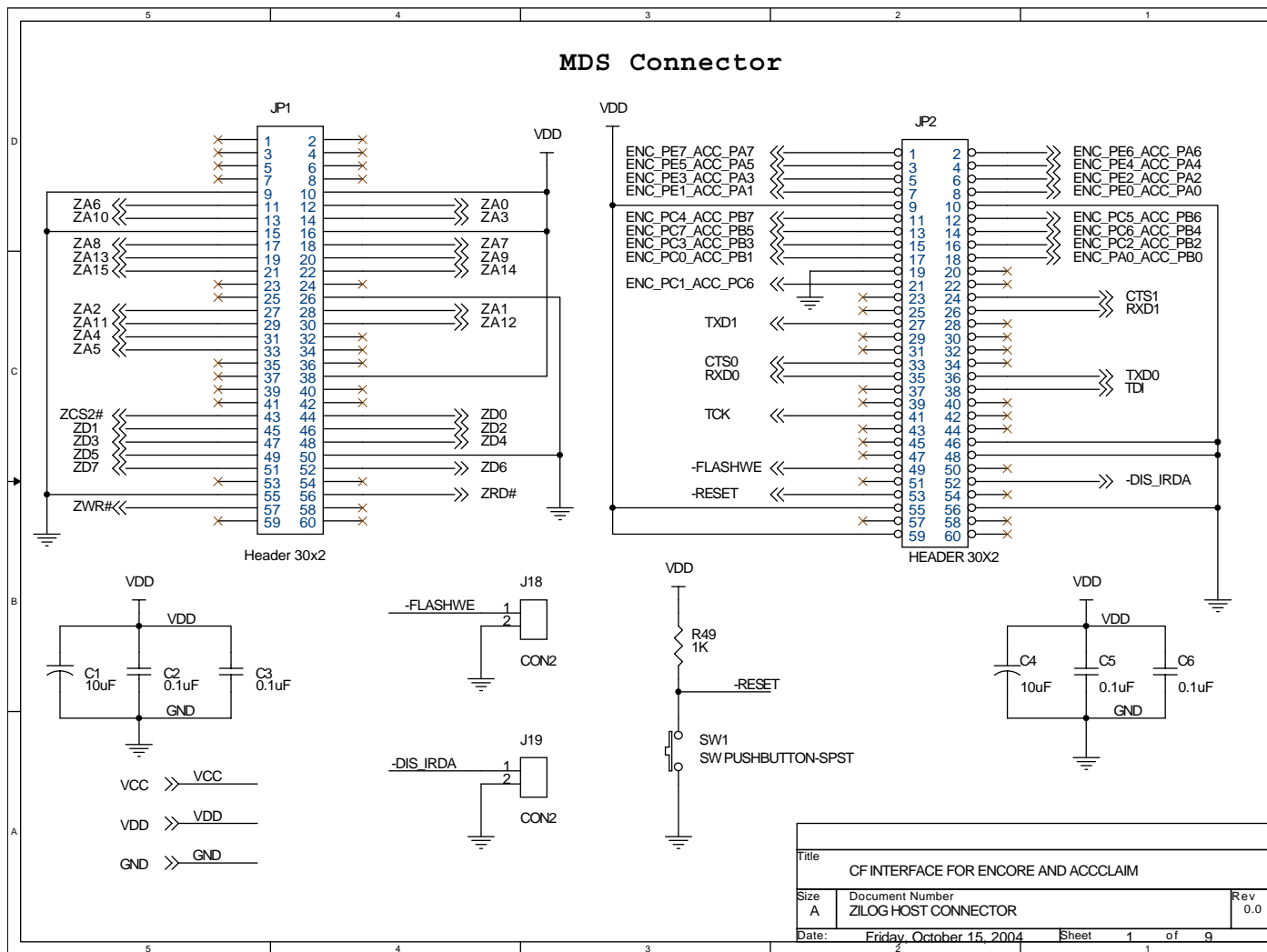


Figure 4. Schematics Displaying MDS Connectors

Figure 5 is the schematic diagram displaying the interface between the eZ80F91 MCU and the CompactFlash® Interface Board.

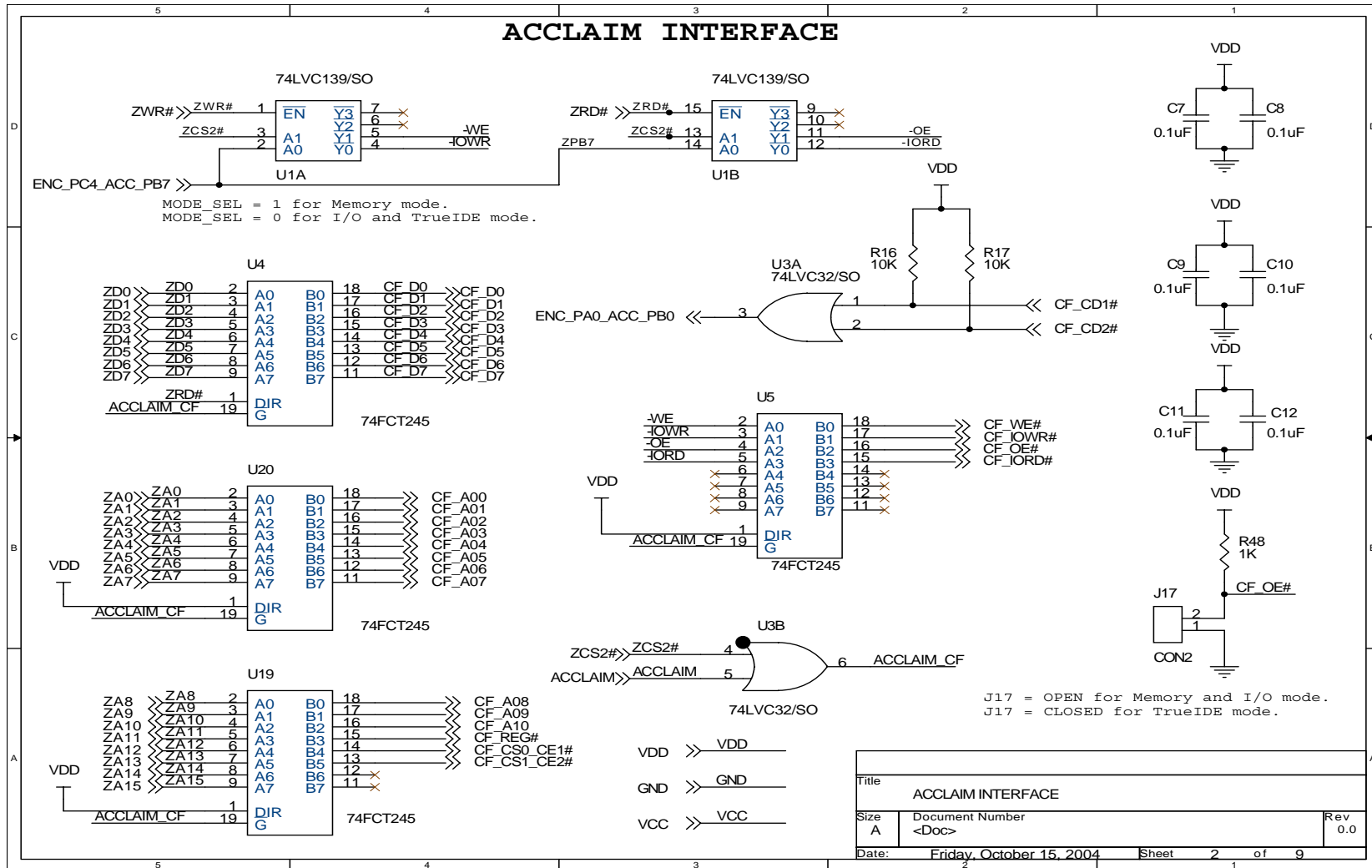


Figure 5. Schematics Displaying the Interface between the eZ80F91 MCU and the CompactFlash® Interface Board

Figure 6 is the schematic diagram displaying the interface between the Z8 Encore! and the CompactFlash® Interface Board.

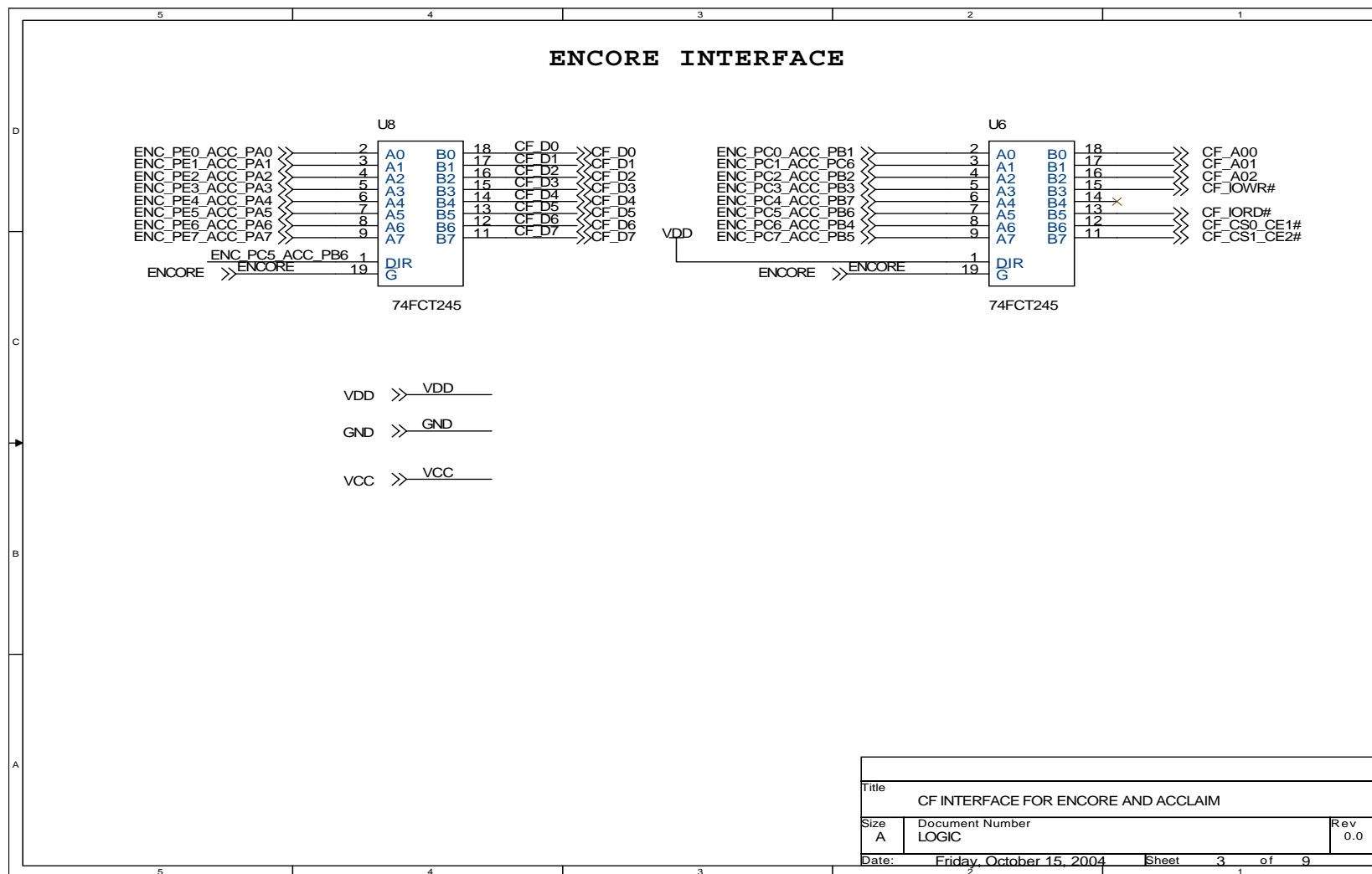


Figure 6. Schematics Displaying the Interface between the Z8 Encore! and the CompactFlash® Interface Board





Figure 7 is the schematic diagram displaying the interface between a PC and the CompactFlash® Interface Board.

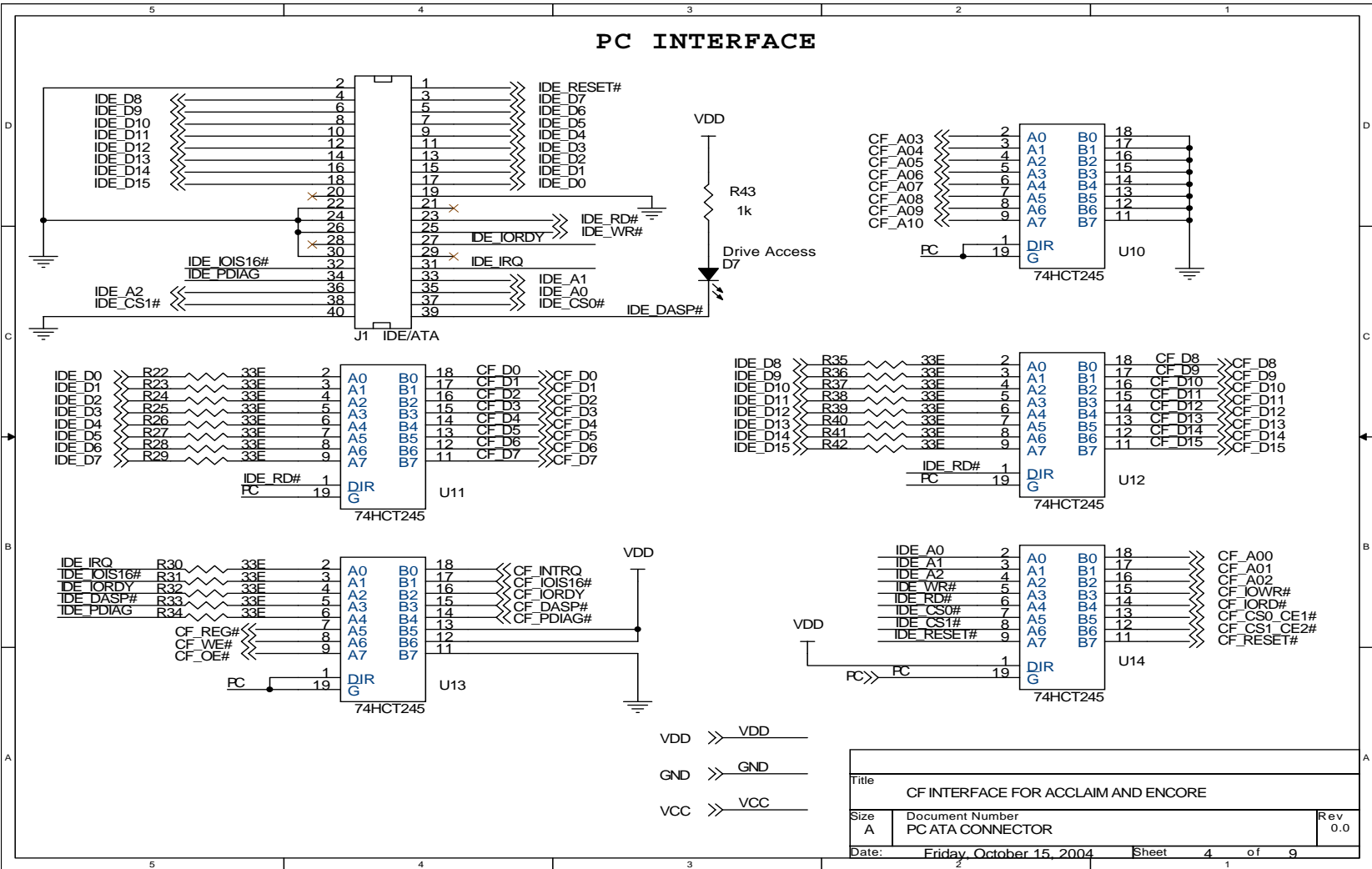


Figure 7. Schematic Displaying the Interface between a PC and the CompactFlash® Interface Board

Figure 8 is the schematic diagram displaying the interface common to the eZ80F91 MCU, the Z8 Encore! MCU, and a PC.

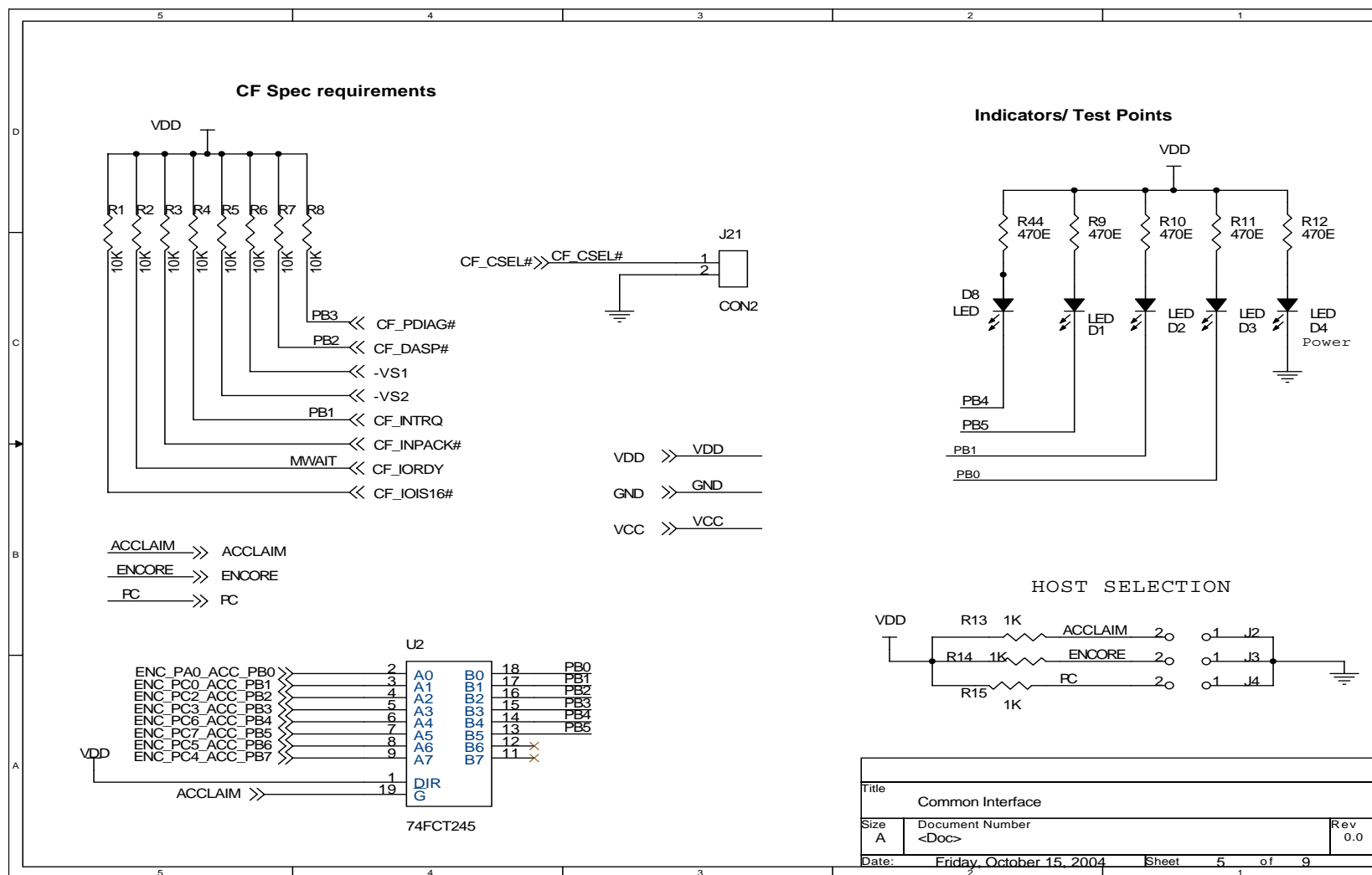


Figure 8. Schematics Displaying an Interface Common to Z8 Encore! and a PC

Figure 9 is the schematic diagram displaying the CompactFlash® Connector.

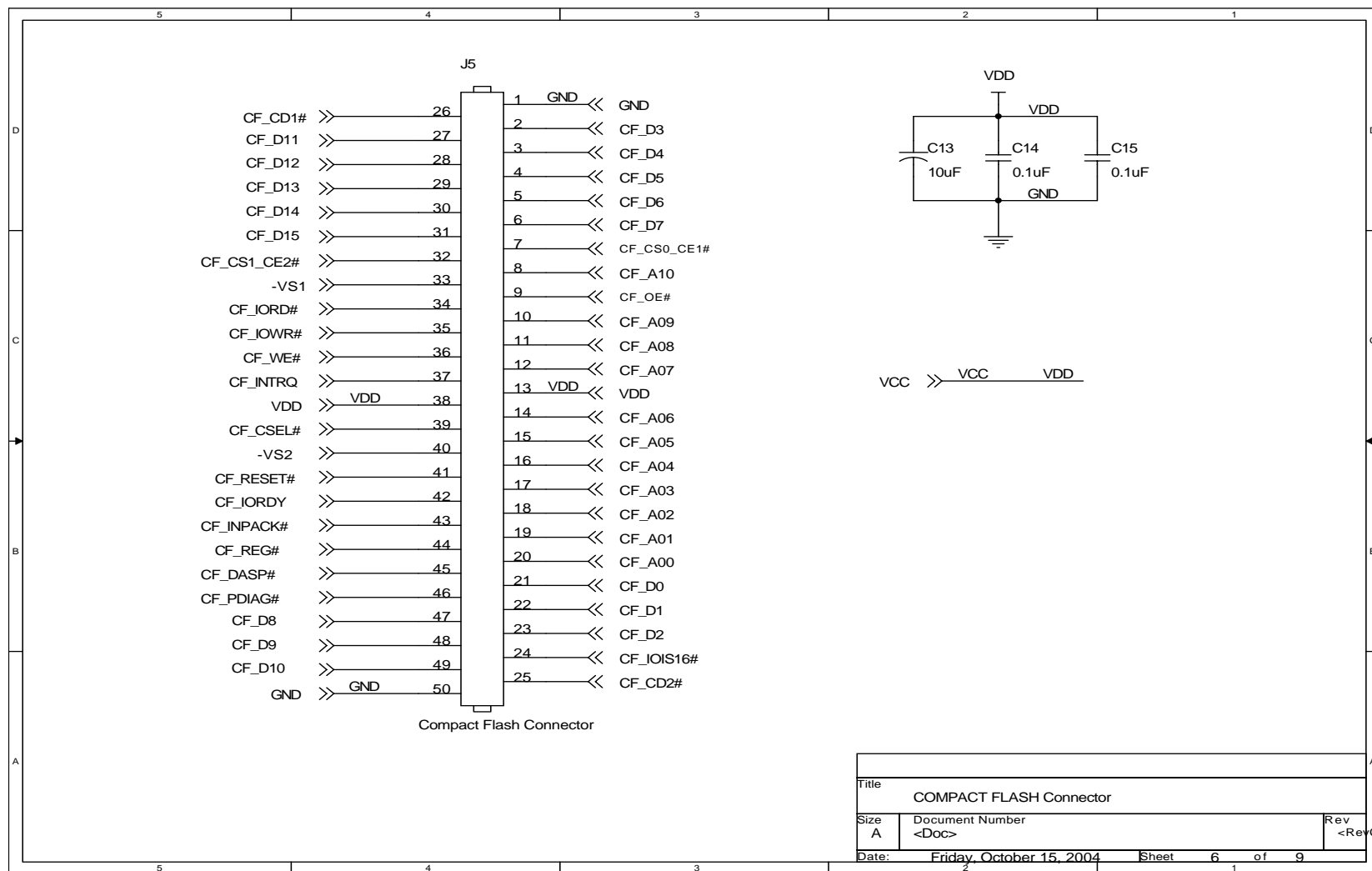


Figure 9. Schematics Displaying the CompactFlash® Connector

Figure 10 is the schematic diagram displaying the power supply for the CompactFlash® Interface Board.

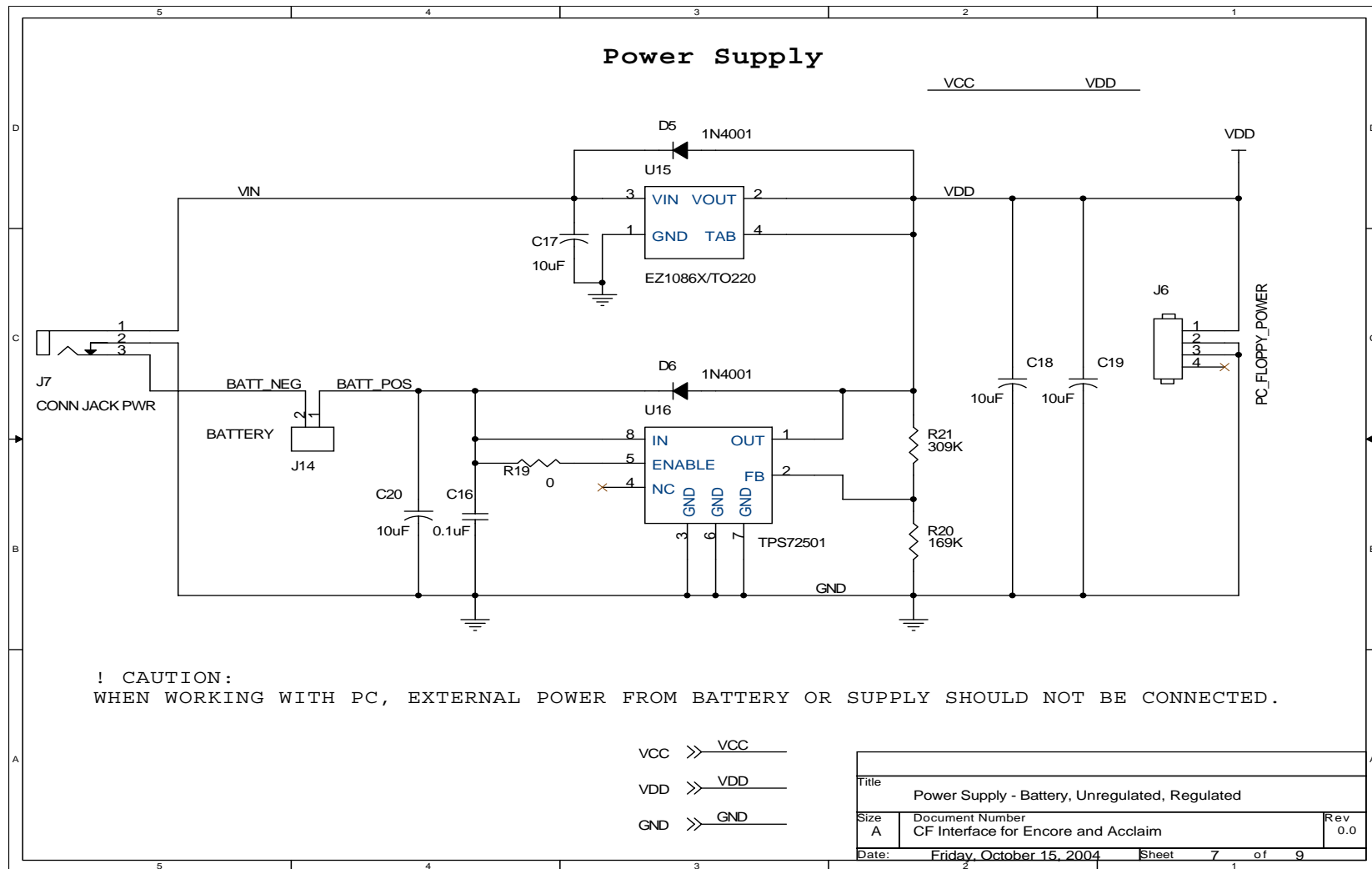


Figure 10. Schematics Displaying the Power Supply for the CompactFlash® Interface Board

Figure 11 is the schematic diagram displaying UART and Download Cable interface.

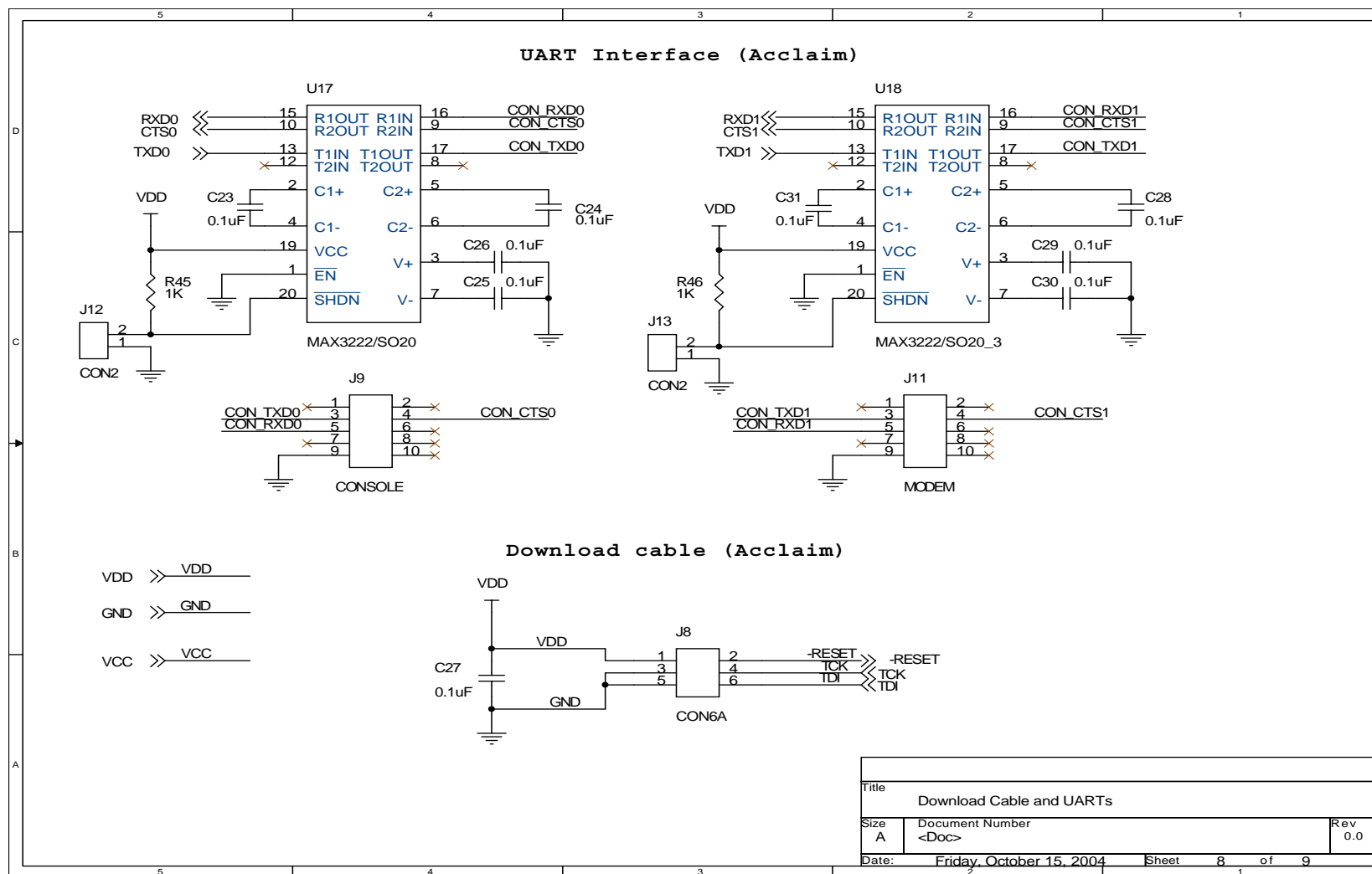


Figure 11. Schematics Displaying UART and Download Cable Interface

Figure 12 is a schematic displaying the Debug connectors.

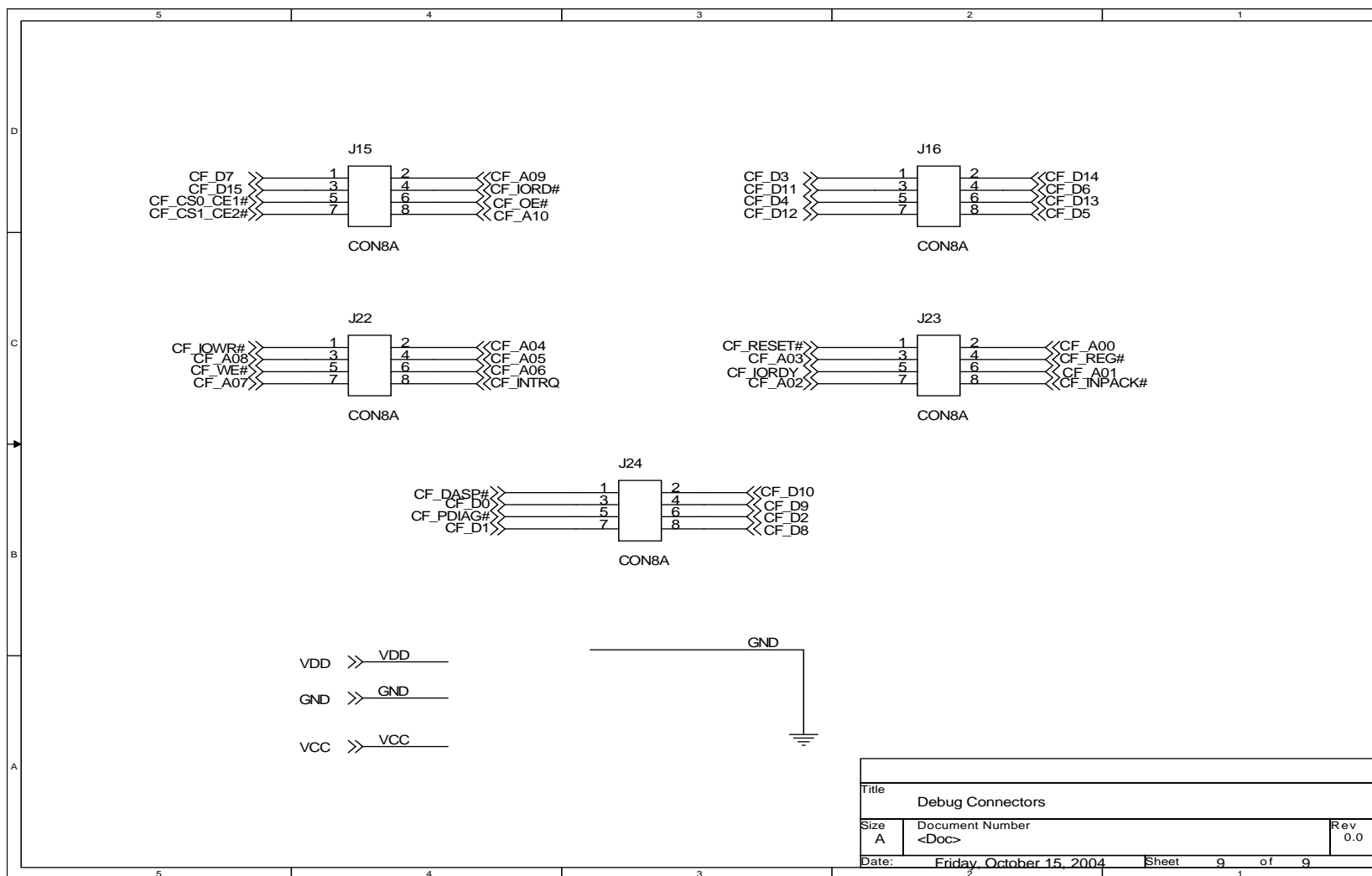
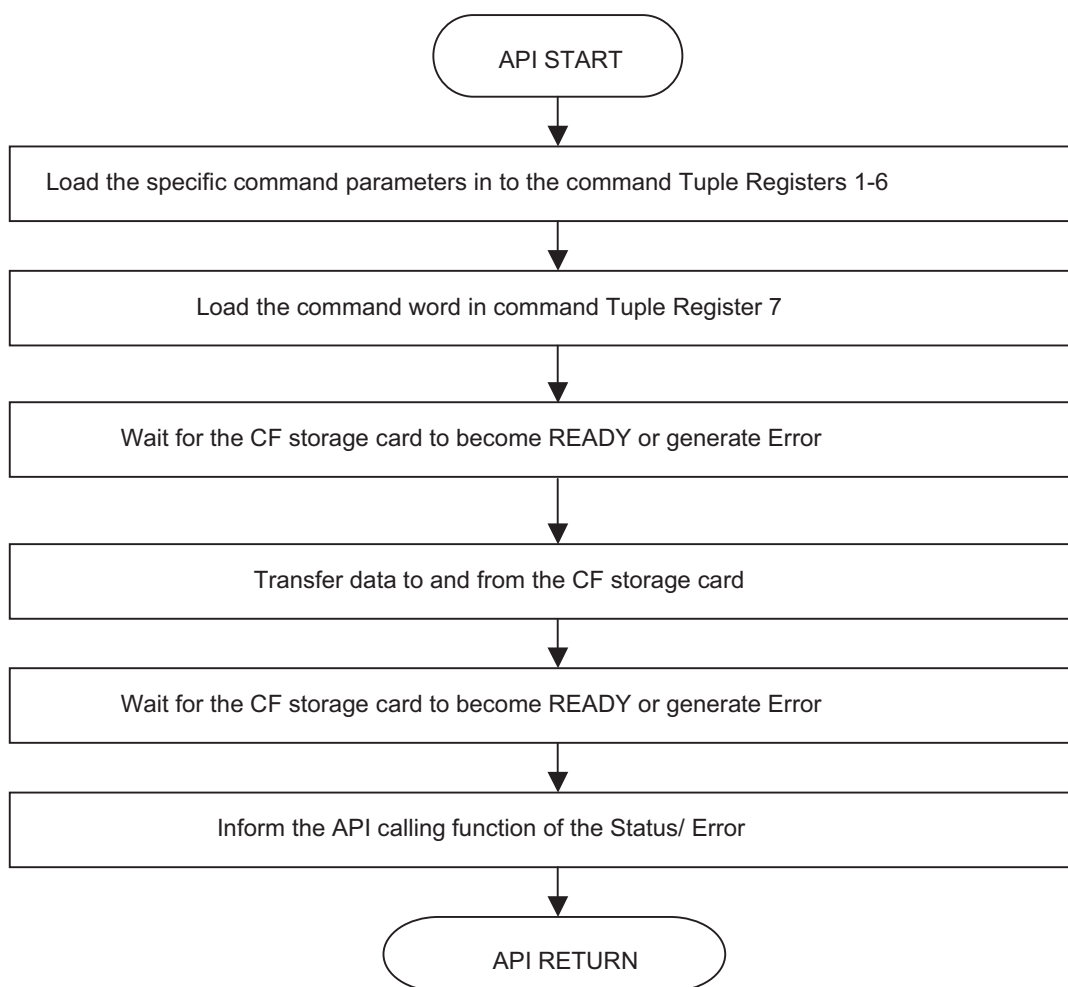


Figure 12. Schematics Displaying Debug Connectors



## Appendix B—Flowcharts

This Appendix displays the flowchart of CompactFlash Interface APIs.



**Figure 13. Flowchart for CompactFlash® APIs**

## Appendix C—API Description

This appendix provides a description of the APIs used to interface the eZ80F91 MCU with the CompactFlash® interface described in this Application Note.

Table 5 presents a list of CompactFlash Interface APIs for quick reference.

**Table 5. CompactFlash® Interface APIs**

API Name	Description
<code>char load_CF_mode()</code>	Selects the CompactFlash mode
<code>char identify_drive()</code>	Identifies the CompactFlash card
<code>char read_sectors()</code>	Reads specified sector on the CompactFlash card
<code>char write_sectors()</code>	Writes to a specified sector on the CompactFlash card
<code>char erase_sectors()</code>	Erases specified sector on the CompactFlash card
<code>char set_features()</code>	Displays/sets features for the CompactFlash card

In general, all the above APIs receive relevant information from corresponding calling functions from the host, and load the CompactFlash command tuple with the received parameters and command words. The APIs handle the data transfer (when required) between the CompactFlash card and the eZ80AcclaimPlus! web-server. Finally the APIs check the error register and report the success and failure of the operation to the calling function.

Descriptions for the CompactFlash Interface APIs begin on the next page.

## char load\_CF\_mode()

### Prototype

```
char load_CF_mode(char mode)
```

### Description

The `load_CF_mode()` API initializes the CompactFlash card in a specified mode of operation. It also modifies the pointers to the command tuple.

The `mode` parameter takes `m`, `c`, `p`, or `s` as values to reset the CompactFlash card to memory mode, contiguous I/O mode, primary I/O mode, or secondary I/O mode, respectively.

### Argument(s)

<code>char mode</code>	The mode to which the CompactFlash is to be reset
------------------------	---

### Return Value(s)

<code>0</code>	On Success
----------------	------------

<code>error_codes</code>	On Failure
--------------------------	------------

### Example

```
result = load_CF_mode(mode);
```

## char identify\_drive()

### Prototype

```
char identify_drive(void)
```

### Description

The `identify_drive()` API reads the CompactFlash memory structure (number of heads, sectors, and tracks) and the manufacturer's information from the CompactFlash Storage Card and copies the information into the read buffer for the host to use. The host calling function parses the read buffer and displays the information on the HyperTerminal.

### Argument(s)

void

### Return Value(s)

0	On Success
error_codes	On Failure

### Example

```
result = identify_drive();
```

## char read\_sectors()

### Prototype

```
char read_sectors(int sector,int head,int track)
```

### Description

The `read_sectors()` API receives the track, head, and sector number of the sector to be read from the host. The API then transfers the data residing in the specified location from the CompactFlash card to the host's read buffer. On completing the operation, the API returns a success or a failure.

### Argument(s)

int	sector	The sector number from where data is to be read
int	head	The head number from where data is to be read
int	track	The track number from where data is to be read

### Return Value(s)

0	On Success
error_code	On Failure

### Example

```
result = read_sectors (sector, head, track) ;
```

## char write\_sectors()

### Prototype

```
char write_sectors(int sector,int head,int track)
```

### Description

The `write_sectors()` API receives the track, head, and sector number of the sector where the host requires to write the data. The API then transfers the data from the host's write buffer to the CompactFlash card. On completing the operation, the API returns a success or a failure.

### Argument(s)

int sector	The sector number where data is to be written on the CompactFlash card
int head	The head number where data is to be written on the CompactFlash card
int track	The track number where data is to be written on the CompactFlash card

### Return Value(s)

0	On Success
error_code	On Failure

### Example

```
result = write_sectors (sector, head, track) ;
```



## char erase\_sectors()

### Prototype

```
char erase_sectors(int sector,int head,int track)
```

### Description

The `erase_sectors()` API is used to erase previously written data in a specified sector. Functionally, this API is similar to the `write_sectors()` API, except that instead of the transferring data from the host buffer to the specified sector, the `erase_sector()` API populates the specified sector with `0xFF`.

### Argument(s)

int sector	The sector number from where data is to be erased
int head	The head number from where data is to be erased
int track	The track number from where data is to be erased

### Return Value(s)

0	On Success
error_code	On Failure

### Example

```
result = erase_sectors (sector, head, track) ;
```

## char set\_features()

### Prototype

```
char set_features(void)
```

### Description

The `set_features()` API is used by the host to establish or select certain features of the CompactFlash card.

### Argument(s)

void

### Return Value(s)

0            On Success

error\_code   On Failure

### Example

```
result = set_features();
```



**Warning:** DO NOT USE IN LIFE SUPPORT

### **LIFE SUPPORT POLICY**

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

### **As used herein**

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

### **Document Disclaimer**

©2008 by Zilog, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

eZ80, Z80, eZ80Acclaim!, and eZ80AcclaimPlus! are trademarks or registered trademarks of Zilog, Inc. All other product or service names are the property of their respective owners.