AN83902

PSoC® 3 and PSoC 5LP – Creating a CFP Management Interface

Author: Rodolfo Lossio
Associated Project: Yes
Associated Part Family: PSoC 3, PSoC 5LP
Software Version: PSoC Creator™ 4.2 or later

AN83902 shows how to create a CFP (C Form-factor Pluggable) Management Interface using PSoC® 3 or PSoC 5LP. Included are two example projects that demonstrate the Management Data Input/Output (MDIO) Interface Component, which controls the interface bus used in CFP optical modules.

Contents

1 Introduction....1
2 Introduction to the MDIO Interface....2
2.1 Why Use PSoC?....4
3 Getting Started With the MDIO Interface Component..6
3.1 User Interface....8
3.2 Example Project 1 – Basic Mode....9
3.3 Example Project 2 – Advanced Mode....12
4 Summary....16
Document History....17
Worldwide Sales and Design Support....18

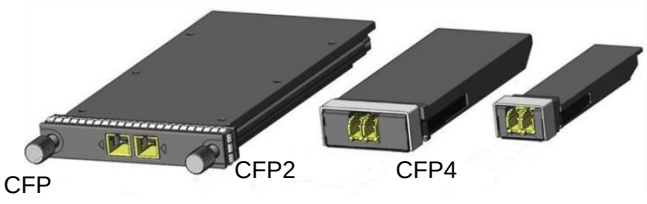
1 Introduction

Global Internet traffic is constantly increasing and is expected to reach 2 zettabytes per year by 2019. The explosion of streaming video and cloud computing is the main reason for the increase in traffic.

To keep pace with this traffic, the Internet requires an upgrade. CFP (C Form-factor Pluggable) optical modules deliver 100-Gbps data rates to meet the demand. The CFP Multi-Source Agreement (MSA) defines specifications for CFP modules, including the Hardware and Management Interface Specifications.

CFP modules are part of routers, which have limited front panel space. Figure 1 shows the evolution of CFP module form factors defined by the CFP MSA. Increases in front panel density result from decreases in module size and power. CFP and CFP2 are in demand and are currently shipping, and the market is moving toward CFP4 optical modules to quadruple port density compared with CFP.

Figure 1. CFP Optical Module Form Factor



Module	Length (mm)	Width (mm)	Height (mm)	Area (mm^2)
CFP	145.0	82.0	13.6	11890
CFP2	107.5	41.5	12.4	4461
CFP4	89.0	21.0	9.5	1869

The CFP Management Interface is the main communication interface between a host and a CFP module. The host uses this interface to control and monitor the startup, shutdown, and normal operation of the CFP modules that it manages.

The primary protocol of the CFP Management Interface is specified using the MDIO Interface bus, which conforms to the general specification of IEEE 802.3, Clause 45. This application note focuses on creating an MDIO interface with PSoC 3 and PSoC 5LP.

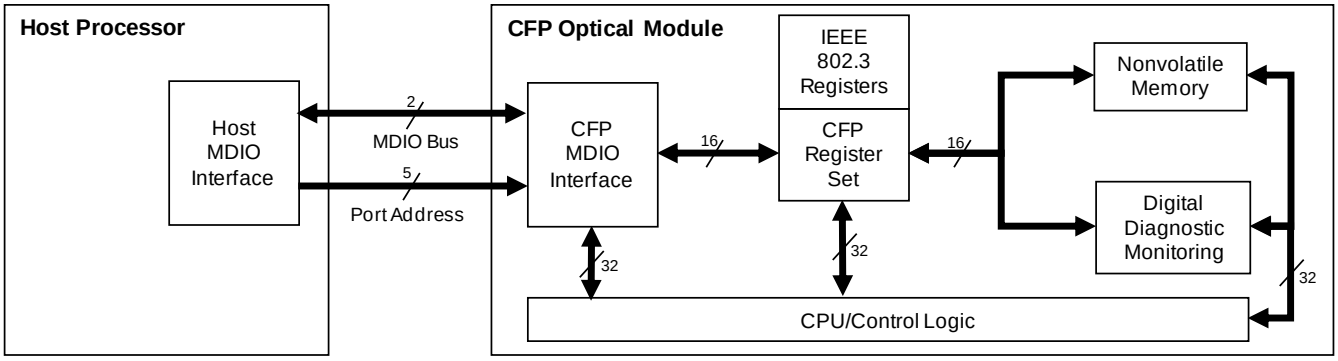
Although this application note provides basic information on the CFP Management Interface, it is not intended to replace the CFP MSA specifications. Please refer to the latest CFP Management Interface Specification and the CFP, CPF2 or CFP4 Hardware Specification for details.

This application note assumes that you are familiar with developing applications using PSoC Creator™ for PSoC 3 or PSoC 5LP. If you are new to these products, you can find introductory information in AN54181 — Getting Started with PSoC 3 and AN77759 – Getting Started with PSoC 5LP. If you are new to PSoC Creator, see the PSoC Creator home page.

Figure 2 shows an example of the CFP Management Interface architecture. The host MDIO interface can communicate with multiple CFP modules over the MDIO bus. The port address of each CFP module on the MDIO bus can be configured through the port address wires, allowing 32 modules on the same bus. The CFP module has five MDIO port address pins; CFP2 and CFP4 have only three MDIO port address pins, allowing up to eight distinct port addresses.

The architecture also recommends having a dedicated MDIO logic block to handle the high rate of MDIO data and a CFP register set. The CFP register set stores CFP optical module ID information, configuration, diagnostic data, and vendor-specific data. It is configured as 16-bit words, each occupying one MDIO address. The register set is divided into two register groups: nonvolatile register (NVR) and volatile register (VR). Over the internal bus system, the VRs are connected to a device that executes the host control commands and reports various digital diagnostics monitoring (DDM) data. The nonvolatile memory (NVM) block can be used to back up the NVRs.

Figure 2. CFP Management Interface Architecture (Source: http://cfp-msa.org)



2 Introduction to the MDIO Interface

The MDIO interface is a two-wire serial bus used to link the host MDIO interface to the CFP module. The MDIO interface specification is controlled by IEEE 802.3. MDIO is similar to I^2C.

IEEE 802.3 has two clauses—Clause 22 and Clause 45—that describe two types of MDIO interface. Clause 22 is the basic MDIO interface specification, while Clause 45 adds support for low-voltage devices and a 16-bit address range.

The MDIO bus has two wires:

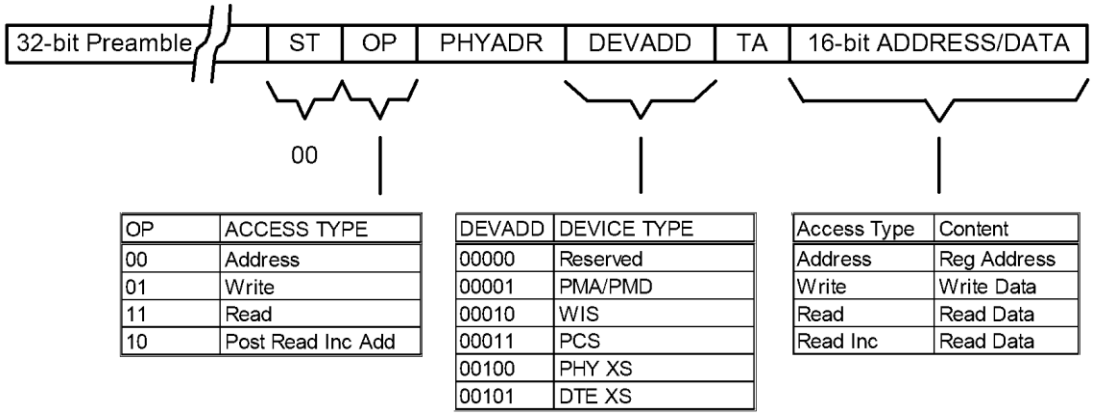
- MDC clock, driven by the host
- MDIO data, bidirectional; read direction is toward the host

The MDIO host (also known as the Station Management Entity) initiates all communication in MDIO and is responsible for driving the clock on the MDC wire. The MDC is specified to have a frequency of up to 4 MHz. The standard specifies a timing distinction such that when the host sources the MDIO signal, it provides a minimum 10-ns setup-and-hold time for the MDC signal. If the slave is supplying the MDIO signal, the specification allows the clock-to-data delay to be a minimum of 0 ns and a maximum of 300 ns.

The MDIO and MDC pins are implemented using 5-V or 3.3-V TTL signals. Clause 45 also adds “0 to 1.5 V” signal logic (1.2 V LVCMOS-compatible).

An MDIO frame consists of a 32-bit preamble and a 32-bit command body, as Figure 3 shows.

Figure 3. MDIO Frame Structure



OP	ACCESS TYPE
00	Address
01	Write
11	Read
10	Post Read Inc Add

DEVADD	DEVICE TYPE
00000	Reserved
00001	PMA/PMD
00010	WIS
00011	PCS
00100	PHY XS
00101	DTE XS

Access Type	Content
Address	Reg Address
Write	Write Data
Read	Read Data
Read Inc	Read Data

The preamble bits are driven by the host, and they are all logic level HIGH. The bits in the frame command body are as follows:

- Start of frame (ST): Always equals 00 and is driven by the host (2 bits).
- Operation code (OP): Frame type, driven by the host (2 bits). Frame type can be Address, Write, Read, or Read and post address increment.
- Physical Address (PHYADR): Physical port address (5 bits or 3 bits), same as port address already described.
- Device Type (DEVADD): Type of device being addressed, driven by the host (5 bits). This allows the host to access other devices in addition to PHYs. It allows for multiple MDIO slave devices within the same CFP module.
- Turn around (TA): If the frame is a read frame, these bits provide an opportunity for the MDIO interface to start driving the MDIO data wire (2 bits).
- Address or Data (16-bit ADDR/DATA): Depends on the frame type:
  - Address frame: 16-bit address driven by the host
  - Write frame: 16-bit data driven by the host
  - Read frame: 16-bit data driven by the MDIO interface

To read or write a register in a specific slave on a given port, the host opens the register by issuing an address frame with the PHYADR.DEVADD.ADDR, as Figure 4 shows.

Figure 4. MDIO Address Frame Example

1111…11	00	00	PPPPP	BBBBB	10	AAAA…AA
Preamble	ST	OP	PHYADR	DEVADR	TA	ADDR/DATA

The selected slave latches the address, and then the host issues a read, write, or read-address-increment frame. For the write operation, the host provides the valid data in the frame. For the read operation, the host tri-states the MDIO line, and the slave sources the data. For the read-address-increment, the slave supplies the data as

Figure 7, yet increments its address register and is ready for the next read or read-address-increment command. If the address register contains 0FFFFh, it will not roll over to 0. The formats of the various frames are shown in Figure 5, Figure 6, and Figure 7.

Figure 5. MDIO Write Frame Example, Data Slot Driven by Master

1111…11	00	01	PPPPP	BBBBB	01	DDDD…DD
Preamble	ST	OP	PHYADR	DEVADR	TA	ADDR/DATA

Figure 6. MDIO Read Frame Example, Data Slot Driven by Slave

1111…11	00	11	PPPPP	BBBBB	Z0	DDDD…DD
Preamble	ST	OP	PHYADR	DEVADR	TA	ADDR/DATA

Figure 7. MDIO Read Address Increment Frame Example, Data Slot Driven by Slave

1111…11	00	10	PPPPP	BBBBB	Z0	DDDD…DD
Preamble	ST	OP	PHYADR	DEVADR	TA	ADDR/DATA

Legend for Figure 4, Figure 5, Figure 6, and Figure 7:

PPPPP:	5-bit physical port address
BBBBB:	5-bit device type (DEVADD)
Z:	Tri-state bit
AAAA…AA:	16-bit register address
DDDD…DD:	16-bit data

2.1 Why Use PSoC?

PSoC 3 and PSoC 5LP devices are ideal for implementing a CFP Management Interface because they offer the following features:

- Programmable digital resources to implement custom interfaces, such as the MDIO Interface, that cannot be found in any off-the-shelf microcontroller
- DMA controller for fast data transfer between the CFP register set and other memory blocks, freeing the CPU to manage the high-level control logic
- ADCs with multiple channels to measure analog signals, like temperature and voltage, which is a common requirement of the DDM block
- DACs to control the optical lenses of a CFP module; you can also use high-resolution pseudo-DACs implemented with PWMs.
- Support for digital interfaces, like I^2C and SPI, to communicate with other chips in the CFP module
- EEPROM for implementing the NVM

PSoC Creator provides all these features as standalone Components that are easy to use and configure, as described in the following subsections.

2.1.1 MDIO Interface Component

You can configure this Component to generate an interrupt for any frame received from the MDIO bus. You can then implement your data-handling algorithm in firmware. Or you can configure the Component to automatically handle all the CFP registers in hardware, freeing the CPU to implement more advanced CFP module features. The PSoC Creator MDIO Interface Component currently implements only Clause 45.

This application note includes two PSoC Creator example projects to show you how to use the MDIO Interface Component. These projects use the CY8CKIT-030 or CY8CKIT-050B PSoC Development Kits (DVKs); the projects can be adapted to work with the CY8CKIT-001 DVK by reassigning the pins. The examples show how to support the MDIO interface and get data on and off the MDIO bus. Note that the higher level functionality of a CFP module is beyond the scope of the example projects.

2.1.2 DMA Component

Some CFP registers may affect the status of other parts of the hardware, such as output pin states, or they may control the access to other CFP registers. In some cases, the response time to a change in the CFP registers must be within a few microseconds, requiring a powerful processor. In these situations, DMA can free the CPU and implement a fast method to change almost any register mapped in the PSoC memory, including registers that affect the hardware blocks inside PSoC.

The following documents provide some useful material on how to use DMAs with PSoC:

- Application note: AN52705 – Getting Started with DMA
- White paper: Reducing CPU Loading through Data Buffering of ADCs Using DMA
- Datasheet: Direct Memory Access (DMA) PSoC Creator Component

2.1.3 ADC Component

Some monitoring signals handled by the DDM block require an ADC to measure temperature. The following documents show examples of temperature measurement with different temperature sensors:

- Application note: AN66477 – Temperature Measurement with a Thermistor
- Application note: AN70698 – Temperature Measurement with an RTD
- Application note: AN60590 – Temperature Measurement with a Diode

You can also find datasheets for the ADC Components:

- Datasheet: ADC Successive Approximation Register (ADC_SAR)
- Datasheet: Delta Sigma Analog to Digital Converter (ADC_DelSig)

2.1.4 DAC Component

One PSoC can have as many as four dedicated 8-bit DACs. In CFP modules, DACs may be used to control the optical lenses and to create voltage references for internal operation. One example is to generate the 1.2-V reference for the MDIO bus. Learn more about how to use the DACs with PSoC:

- Datasheet: 8-Bit Voltage Digital to Analog Converter (VDAC8)
- Application note: AN69133 – Easy Waveform Generation with the WaveDAC8 Component
- Application note: AN64275 – Getting More Resolution from 8-Bit DACs

2.1.5 PWM Component

The number of DACs in PSoC may not be enough to satisfy your CFP module requirements. However, you can use high-resolution PWM blocks to implement a pseudo-DAC. Depending on which resources you have left, you can add as many as 10 pseudo-DACs to your design. For more information, see the following datasheets:

- Datasheet: Pulse Width Modulator (PWM) PSoC Creator Component
- Datasheet: Trim and Margin PSoC Creator Component

2.1.6 Digital Interface Components

PSoC Creator supports common digital interfaces with Components for SPI, I^2C, and UART. To learn more, refer to the Component datasheets:

- I^2C Master/Multi-Master/Slave PSoC Creator Component
- Serial Peripheral Interface (SPI) Slave PSoC Creator Component
- Serial Peripheral Interface (SPI) Master PSoC Creator Component
- Universal Asynchronous Receiver Transmitter (UART) PSoC Creator Component

2.1.7 EEPROM Component

PSoC supports up to 2 KB of EEPROM memory. This is sufficient to support all the NVR registers specified in the CFP MSA Specification. If you need more nonvolatile registers, you can also use the internal flash memory to store custom registers. For more information, go to the following links:

- Datasheet: EEPROM PSoC Creator Component
- Knowledge Base Article: Flash Memory Organization and Array ID Parameter

3 Getting Started With the MDIO Interface Component

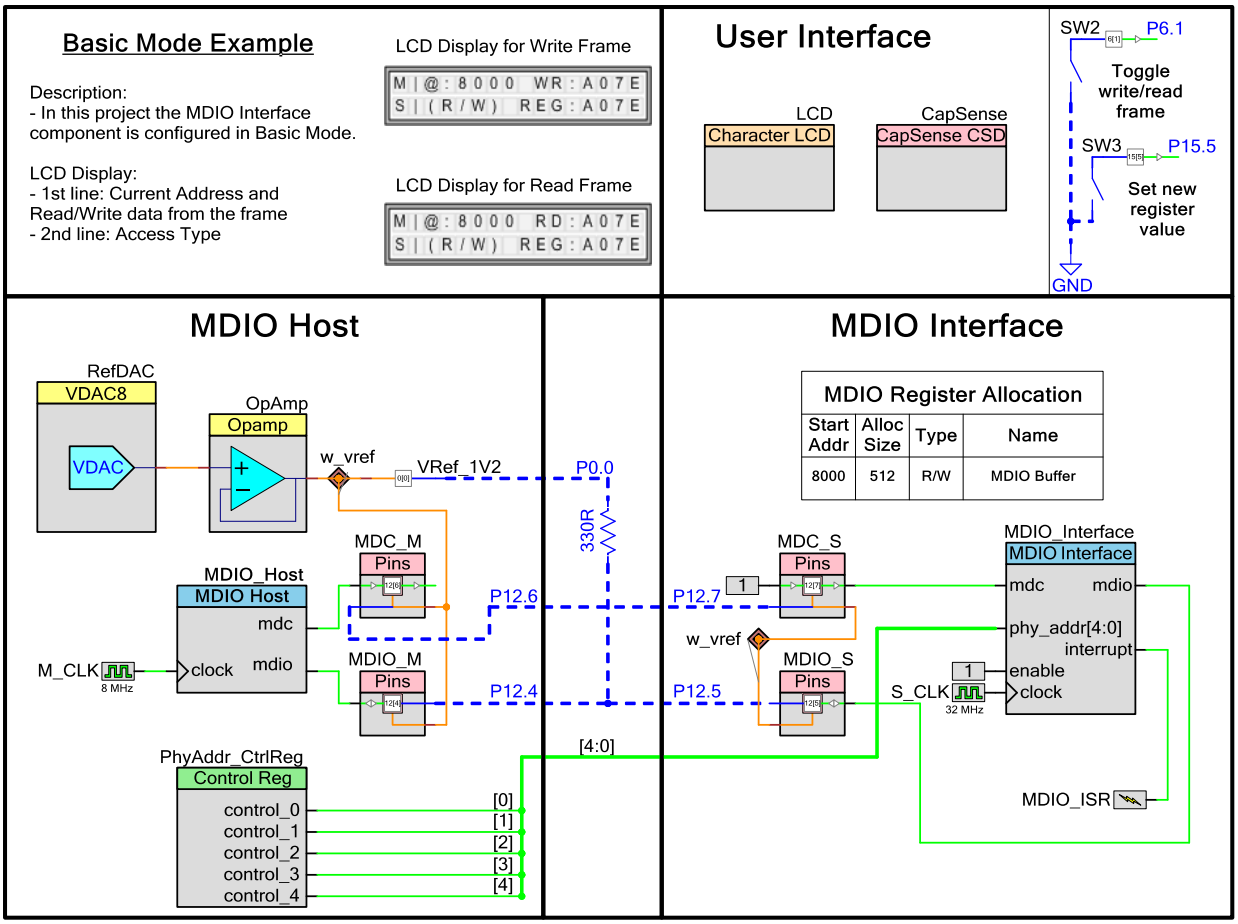
Included with this application note is a *.zip* file containing two PSoC Creator projects that use the MDIO Interface Component. Save the *.zip* file to a convenient location on your hard drive and extract the contents to a local folder.

To get started with the example projects, double-click the *AN83902.cywrk* PSoC Creator workspace file.

1. In the Workspace Explorer tab to the left of the screen, expand Example Project 1 by clicking on the small "+" icon to the left of it.
2. Double-click *TopDesign.cysch* to open the top-level design schematic for the hardware blocks inside PSoC.

Figure 8 shows the top-level design schematic for Example Project 1. The RefDAC and Opamp Components generate and buffer a 1.2-V reference for the MDIO bus. The MDIO_Host Component generates MDIO frames with a 4-MHz bus clock. The PhyAddr_CtrlReg Component simulates the port address wires.

Figure 8. Top-Level Design Schematic for Example Project 1



The MDIO Interface Component is in the Cypress Component Catalog in the Communications folder of the **Cypress** tab (see Figure 9). The **Default** tab contains the companion MDIO Host Component provided with this application note as part of the bundled example projects.

Figure 9. PSoC Creator Component Catalog

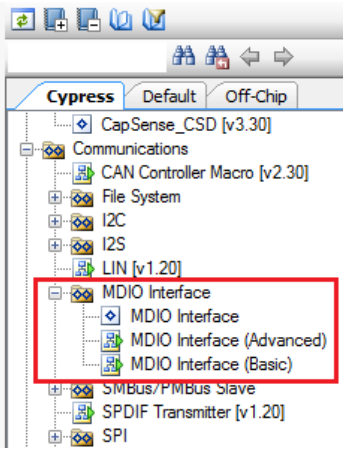


Figure 9 also shows that the standard Cypress Component Catalog has two types of predefined MDIO Interface configuration in the form of macros:

- MDIO Interface (Advanced): Configures the Component in advanced mode, which automatically handles access to the CFP registers.
- MDIO Interface (Basic): Configures the Component in basic mode, which generates an interrupt to firmware at the end of an MDIO frame. The user is responsible for handling access to the CFP registers in firmware.

See also the MDIO Interface Component configuration options, as Figure 12 on page 9 shows.

All the example projects provided with this application note are designed to run on the CY8CKIT-030 or CY8CKIT-050B PSoC DVKs. Each project works in loopback mode, meaning it contains an MDIO Host Component that exercises and verifies the functionality of the MDIO Interface Component. An external MDIO host may be used instead of the MDIO Host Component.

To test this project with your own MDIO host, set the MDIO_HOST_ENABLE constant in *main.c* to 0, recompile, and program the DVK again. This makes the project an MDIO interface-only project. Connect port pins P12[7] and P12[5] directly to your MDIO bus instead of to P12[6] and P12[4], respectively.

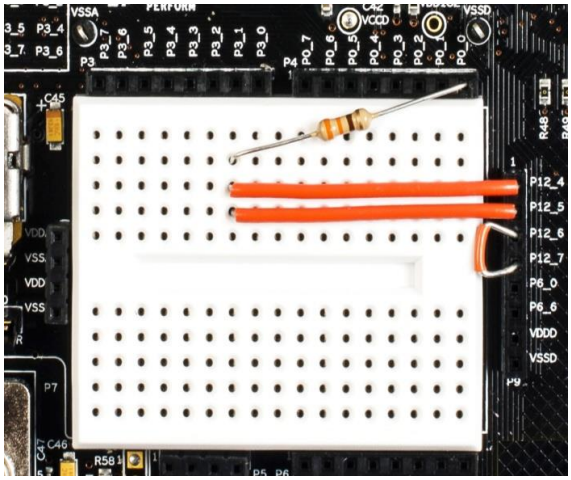
Note that the MDIO_S (P12[5]) and MDC_S (P12[7]) pins, which are connected to the MDIO Interface Component, are configured as transparent sync mode. This is to guarantee the setup and hold timing specification of the MDIO bus.

You can also configure the project to disable the MDIO Interface Component; set MDIO_INTERFACE_ENABLE to 0. Connect port pins P12[6] and P12[4] to your MDIO bus instead of to P12[7] and P12[5], respectively.

If desired, the host-only and interface-only versions can be programmed into two different DVKs. This allows you to observe how messages are passed back and forth between two independent PSoC devices.

To implement the loopback mode properly, follow the blue connections illustrated in Figure 8 on page 6. All the items displayed in blue are off-chip Components and connections, which represent external Components and wiring. Figure 10 shows the external connections on the DVK.

Figure 10. CY8CKIT-050B External Connections



Note: A typical value of the pull-up resistor on the MDIO bus data wire is 330 Ω. This resistance is much lower than that used with other open-drain buses, such as I^2C.

A VDAC and an opamp are included in the design to provide a 1.2-V reference for all pins connected to the bus wires. This same signal is used as the power source for the 330-Ω pull-up resistor on the MDIO bus data wire.

If the external connection is improperly done, an error message displays on the LCD.

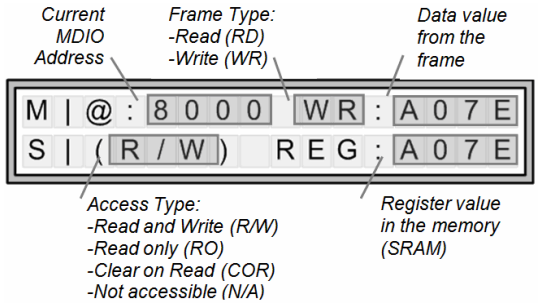
3.1 User Interface

In each project, the LCD displays information about the data exchanged between the MDIO Host and the MDIO Interface. Figure 11 shows the meaning of each field in the display; all numbers are shown in hexadecimal.

The first line in the LCD shows the current address accessed by the MDIO Host. The host always sends an address frame before a read or write frame. If the host sends a write frame, the data value written in the frame is also displayed on the first line of the LCD. If the host sends a read frame, the data value read from the frame is displayed instead.

The second line displays information about the register stored in the SRAM memory. The register that belongs to a specific MDIO address has four access types; see the list in Figure 11.

Figure 11. LCD Display Information



The two mechanical buttons, SW2 and SW3, on the DVK are for user input. The buttons control the information displayed on the LCD as follows:

- SW2: Changes the type of frame being driven on the MDIO bus. It toggles between read and write frames.
- SW3: Sets a new register value in the memory at the current MDIO address.

The capacitive buttons and sliders on the kit PCB are also for user input. They control the current address display in the LCD. The following directions are given with the board oriented such that the LCD is toward the bottom.

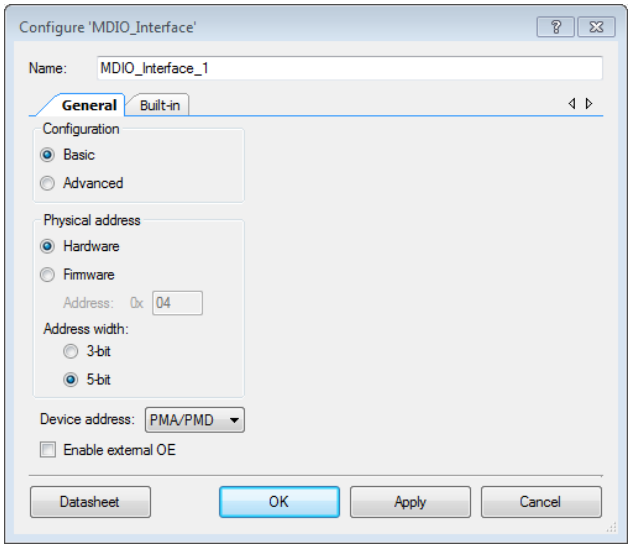
- Slide your finger up to quickly increment the MDIO address.

- Slide your finger down to quickly decrement the MDIO address.
- Press the button on the top to increment the MDIO address by one unit.
- Press the button on the bottom to decrement the MDIO address by one unit.

3.2 Example Project 1 – Basic Mode

Double-click the MDIO Interface Component to open its configuration dialog, as Figure 12 shows.

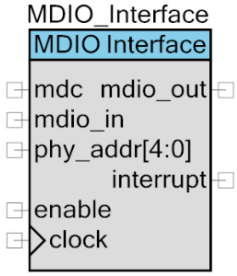
Figure 12. MDIO Interface Component Configuration



In the first example project, the MDIO Interface Component is configured for basic mode. In both projects, the physical address (PHYADR) is configured in hardware by an external control register to a value of 0x1F. The device address (DEVADD) is set to PMA/PMD (0x01).

The option “Enable external OE” causes the Component to change the MDIO bidirectional terminal to two terminals: mdio_in and mdio_out (see Figure 13).

Figure 13. MDIO Interface Component with MDIO Terminals Exposed



This may be useful when the MDIO bus is multiplexed with multiple MDIO Interface Components inside one PSoC device.

In Example Project 1, keep the original configuration and click OK. Make sure the DVK is connected to the computer’s USB port via jumper J1. Then program the project into the DVK by choosing **Debug** > **Program** in the PSoC Creator menu. After programming, the text displayed on the DVK LCD is similar to that shown in Figure 11.

Next, press the capacitive buttons or the slider to change the current address: SW2 to toggle between read and write frames or SW3 to change the register value in the memory. In write mode, the register is written by the master over the MDIO interface to a new random value about every two seconds. In read mode, the value is changed to a new random value every time SW3 is pressed and is then read by the master over the MDIO interface.

In this example, the data value from the frame should always match the register value in the SRAM for all R/W registers.

3.2.1 Project Firmware

Now review the project firmware. In the PSoC Creator Workspace Explorer window, double-click *main.c* under the "Source Files" directory for Example Project 1.

All the functions in the MDIO Interface Component API include the prefix "MDIO_Interface." To see the entire list of APIs supported by this Component, refer to its datasheet by right-clicking on the Component in the schematic (see Figure 8 on page 6) and pressing the Datasheet button.

In Example Project 1, a buffer of 512 bytes words is allocated for the MDIO registers (0x8000–0x81FF). The MDIO Host can write and read all the registers of this buffer.

For the MDIO Interface Component configured in basic mode, if the physical address and device address match the preconfigured values, then it generates a short pulse at the interrupt terminal at the end of the frame.

The MDIO Interface Component's interrupt terminal is connected to the Interrupt Component MDIO_ISR (see Figure 8 on page 6). Code 1 shows an example of the ISR code for this interrupt. This code is located in the *main.c* file of the project.

Code 1. MDIO Interrupt Handler

```
CY_ISR(MDIO_ISR_Handler)
{
    /* Get data received from the host */
    MDIO_Interface_ProcessFrame(&opCode, &regData);

    /* Process the data based on theOpCode */
    switch(11eartb)
    {
    case MDIO_Interface_POS_READ:
        /* Increment the address */
        MdioAddress += 1;
        /* Prepare data in case the nextframe is a read frame */
        PrepareDataForNextFrame();
        break;

    case MDIO_Interface_READ:
        /* Prepare data in case the nextframe is a read frame */
        PrepareDataForNextFrame();
        break;

    case MDIO_Interface_WRITE:
        /* Obtain the index to access */
        MdioIndex = MdioAddress -  BASE_MDIO_ADDRESS;
        /* Chck if the address belong totheMDIO buffer*/
        if (MdioIndex< MDIO_BUF_SIZE)
        {
            MdioBuffer[MdioIndex] = regData;
        }
        break;

    case MDIO_Interface_ADDRESS:
        /* Update the address */
        MdioAddress = regData;
        /* Prepare data in case the nextframe is a read frame */
        PrepareDataForNextFrame();
        break;
    }

    11eartbeat += 1;
}
```

In the ISR, the frame bits received from the MDIO bus are parsed to obtain the operation code (OP) and the 16-bit frame data. The OP causes the following:

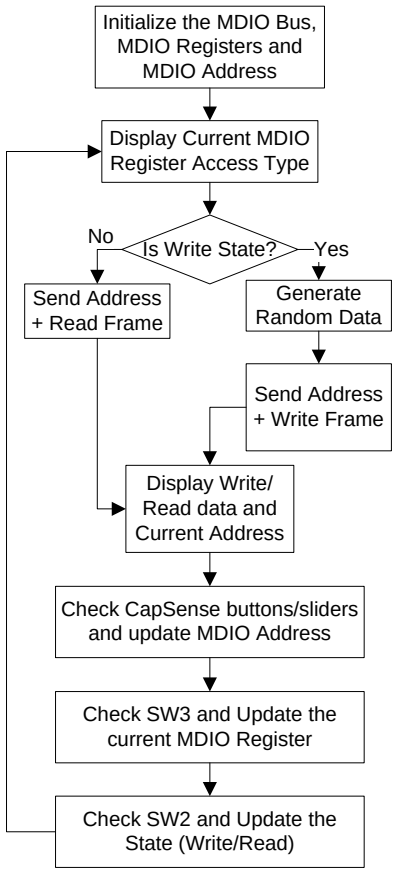
- Address OP (2'b00): The MDIO address is updated with the register data from the frame. The MDIO register value at this address is stored in an internal FIFO from the Component. The bits stored in the FIFO are transmitted to the host in the next read frame.
- Read OP (2'b11): The register value at the current address is stored in the internal FIFO from the Component. Note that the internal FIFO already contained the bits to be transmitted after the address frame was received.
- Pos Read OP (2'b10): The current address is incremented, and the register value at the new address is stored in the internal FIFO from the Component.
- Write OP (2'b01): The register data received is written to the current MDIO address.

The PrepareDataForNextFrame() function uses one of the APIs of the MDIO Interface Component, called "MDIO_Interface_PutData(regData)." This API stores the given data in the internal FIFO from the Component, which will be transmitted to the host in the next read frame.

At the end of the MDIO interrupt handler, a variable named “12eartbeat” is incremented to indicate that the frames are being processed. This variable is used by the main code to display an error message in case of failure.

Figure 14 shows the flow chart of the top-level execution in the main loop.

Figure 14. Top-Level Firmware Flow Chart



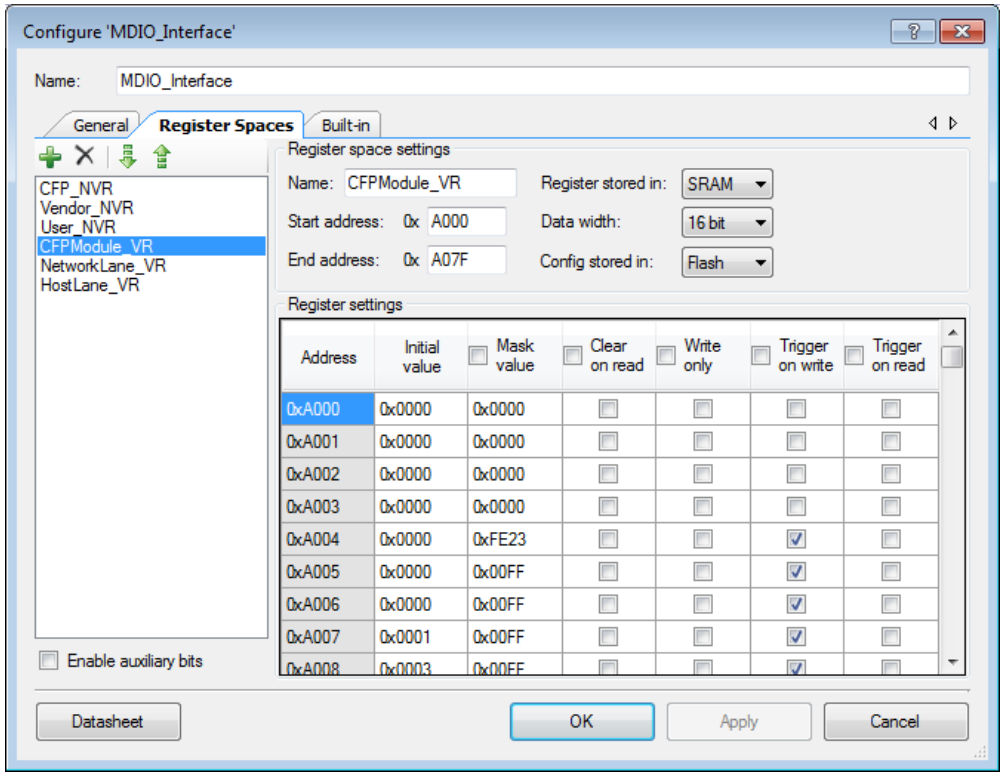
Congratulations! You have completed your first MDIO interface design with PSoC. You can easily modify this example design to support all other features that a CFP MDIO Interface requires, such as clear-on-read registers, writable mask bits, and write-only registers.

3.3 Example Project 2 – Advanced Mode

In Example Project 1, the MDIO Interface Component is configured in basic mode, which requires an ISR to handle the data transfer between the MDIO bus and the MDIO registers. In Example Project 2, the MDIO Interface Component is configured in advanced mode. This means that all data transfers are done in hardware without using the CPU.

1. Close the *main.c* and *TopDesign.cysch* files for Example Project 1. In the Workspace Explorer window, right-click on Project “Example2” and select “Set As Active Project.”
2. Open *TopDesign.cysch* for Example Project 2. Double-click the MDIO_Interface Component to open the Component Configuration Tool. In the **General** tab, the only difference between Example Project 1 and Example Project 2 is the configuration mode. Select the **Register Spaces** tab to configure the register spaces (group of sequential CFP registers) supported, as Figure 15 shows.

Figure 15. MDIO Interface Component Configuration Tool Register Spaces Tab



The register spaces are configured based on the CFP MSA Management Interface Specification document, available at http://www.cfp-msa.org. You can change the starting address and ending address of each register space listed on the left side. All the registers that belong to one register space can be stored in SRAM or flash. If a register space belongs to the NVM block, you can configure it to be stored in SRAM and use the EEPROM Component from PSoC Creator to back up the registers. Or you can back it up directly in PSoC flash memory. This topic is beyond the scope of this application note and is left for application code.

Besides configuring the overall parameters for the register spaces, you can configure parameters for individual CFP registers. To do this, fill the table presented in the configuration tool for each address from the selected register space. Each register supports six configuration settings:

- Initial value of the register
- Writable mask bits
- Clear on read (COR) register
- Write only register
- Trigger on write frames
- Trigger on read frames

In Example Project 2, the CFPModule_VR register space has a start address equal to 0xA000 (see Figure 15), which means the first element of the “Register settings” table is assigned to the address 0xA000, the second one to 0xA001, and so on.

The CFP MSA Specification document contains tables describing the configuration settings of each register that belongs to the CFP register spaces. All the register settings of the MDIO Interface Component tool conform to the description in the CFP MSA Specification.

After the Component is configured, the firmware can easily access the registers in the allocated register spaces by using the MDIO Interface Component API functions.

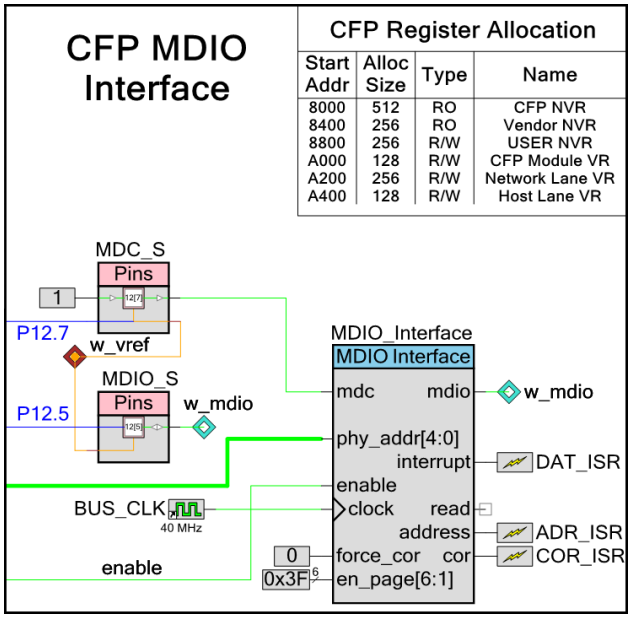
The MDIO Interface Component guarantees against data corruption when the MDIO Host (over the bus) and the firmware (API calls) access the same register at the same time.

In the Example Project 2 schematic, three interrupts are triggered under specific conditions, as Figure 16 shows:

- DAT_ISR: triggered when the MDIO Host writes to a register
- ADR_ISR: triggered when the MDIO Host sends an address frame
- COR_ISR: triggered when the MDIO Host reads a COR register

Note that the Component configured in advanced mode has additional terminals that are not presented in basic mode. See the MDIO Interface Component datasheet for details.

Figure 16. Part of the Top-Level Design from Example Project 2



These terminals do not necessarily need to be connected to Interrupt Components. They can be used to trigger other hardware blocks or a DMA channel. For example, a DMA channel can be used to transfer the current MDIO address to a queue every time the address frame is received.

In Example Project 2, keep the original configuration and click OK. Make sure the DVK is connected to the computer's USB port via jumper J1 and then program Example Project 2 into the DVK by choosing **Debug** > **Program** in the PSoC Creator menu. After programming, you can read or write any register in any MDIO address. To change the MDIO address, use the capacitive buttons and the slider.

As mentioned previously, all the registers configured in this project conform to the CFP MSA Specification document. Table 1 on page 14 summarizes the registers specified in the document.

If the current address is smaller than 0x8000, the read and write frames are ignored and the LCD displays "0xFFFF." Any other register address not listed in the table is not supported and therefore returns "0x0000."

Table 1. Registers Summary

Register Addresses	Access Type	Writable Mask Bits
0x8000 ~ 0x81FF	Read Only	0x00
0x8400 ~ 0x84FF	Read Only	0x00
0x8800 ~ 0x88FF	Read/Write	0xFF
0xA000 ~ 0xA003	Read Only	0x0000
0xA004	Read/Write	0xFE23

Register Addresses	Access Type	Writable Mask Bits
0xA005 ~ 0xA00B	Read/Write	0x00FF
0xA00C ~ 0xA00F	Read Only	0x0000
0xA010	Read/Write	0xFE00
0xA011	Read/Write	0x7FEF
0xA012 ~ 0xA013	Read/Write	0xFFFF
0xA014	Read/Write	0x74E0
0xA015	Read/Write	0xFFFF
0xA016 ~ 0xA021	Read Only	0x0000
0xA022 ~ 0xA027	Clear on Read	0x0000
0xA028	Read/Write	0x01FE
0xA029	Read/Write	0xA7F8
0xA02A	Read/Write	0x0062
0xA02B ~ 0xA02C	Read/Write	0x0FFF
0xA02D ~ 0xA07F	Read Only	0x0000
0xA200 ~ 0xA21F	Read Only	0x0000
0xA220 ~ 0xA23F	Clear on Read	0x0000
0xA240 ~ 0xA24F	Read/Write	0xFFFF
0xA250 ~ 0xA25F	Read/Write	0xE0DC
0xA260 ~ 0xA27F	Read Only	0x0000
0xA280 ~ 0xA28F	Read/Write	0xFFFF
0xA290 ~ 0xA2FF	Read Only	0x0000
0xA400 ~ 0xA40F	Read Only	0x0000
0xA410 ~ 0xA41F	Clear on Read	0x0000
0xA420 ~ 0xA42F	Read/Write	0x0003
0xA430 ~ 0xA43F	Read Only	0x0000
0xA440 ~ 0xA44F	Read/Write	0x0007
0xA450 ~ 0xA47F	Read Only	0x0000

When writing in one of the read/write registers listed in the table, consider the writable bit mask. Because some bits are read-only, the data frame value in the first line of the LCD may not be the same as the register value in the second line of the LCD.

When reading any register listed in the table except the COR registers, the first and second lines of the LCD show the same value. For COR registers, it is possible to observe the register being cleared every time you set a random value by pressing SW3.

The flow chart of the top-level firmware is similar to Example Project 1 (see Figure 14 on page 12).

Congratulations! You have completed an MDIO interface example project using PSoC 3 or PSoC 5LP that implements all the features of the CFP MDIO Interface. You can easily change this example project to implement the other requirements of the CFP Management Interface.

This project also supports the capability to be used in loopback mode, host only, or interface only. Configuration is done using the MDIO_HOST_ENABLE and MDIO_INTERFACE_ENABLE defines, as explained in Example Project 1.

4 Summary

This application note provided an introductory description of the CFP Management Interface and how to create it using PSoC 3 and PSoC 5LP.

The PSoC Creator MDIO Interface Component lets you quickly implement a CFP Management Interface for CFP modules. PSoC custom logic allows you to implement custom protocol interfaces, such as MDIO, without using the CPU.

The unique ability of PSoC to combine custom digital logic, analog signal processing, and an MCU in a single device enables you to integrate combinations of MCUs with FPGAs or CPLDs. This powerful integration not only reduces BOM cost, but it also results in PCB board layouts that are less congested, more reliable, and more compact.

About the Author

Name:	Rodolfo Lossio
Title:	Systems Engineer Sr. Staff
Background:	Rodolfo Lossio has a BS in Control and Automation Engineering from Federal University of Santa Catarina, Brazil, and an MSEE from National Chiao Tung University, Taiwan.

Document History

Document Title: AN83902 – PSoC® 3 and PSoC 5LP – Creating a CFP Management Interface

Document Number: 001-83902

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4016917	RLOS	05/31/2013	New application note.
*A	4542869	RLOS	10/22/1024	Updated project to PSoC Creator 3.0 SP2. Removed reference to the external DFF. The new version of the MDIO Interface no longer requires it.
*B	5281164	RLOS	06/01/2016	Updated project to PSoC Creator 3.3 SP1 Updated introduction text Updated template
*C	5705416	AESATMP9	04/21/2017	Updated logo and copyright.
*D	6197199	RLOS	06/05/2018	Updated project to PSoC Creator 4.2

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

©Cypress Semiconductor Corporation, 2013-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.