CoreCFI Handbook

v2.0

Actel®

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200094-0

Release: March 2007

Trademarks

Table of Contents

Introduction

Core Overview

CoreCFI (Common Flash Interface) provides an industry standard external interface to the embedded Flash memory blocks within the Fusion family of Actel devices. Using CoreCFI, the user is able to communicate (i.e., read, write, and erase) with the embedded Flash memory. This IP block is targeted to provide a functional subset of CoreCFI with a design emphasis given to minimizing design size. Note that this handbook focuses on the operation of CoreCFI and does not provide detail on the structure or the behavior of the Fusion Flash memory. Refer to the *Fusion Family of Mixed-Signal Flash FPGAs* datasheet for details on the Fusion Flash memory. Note that CoreCFI has been designed to be used with an external device, though it can be adapted for use with user-created custom logic within the Fusion FPGA fabric.

CoreCFI has two top-level parameters (Verilog) or generics (VHDL) used to configure the core. For a detailed description of the parameters/generics, refer to Table 3-1 on page 11. CoreCFI block diagram is shown in Figure 1. A typical application using CoreCFI is shown in Figure 2 on page 6. Note that the D pin output enable signal is inverted.

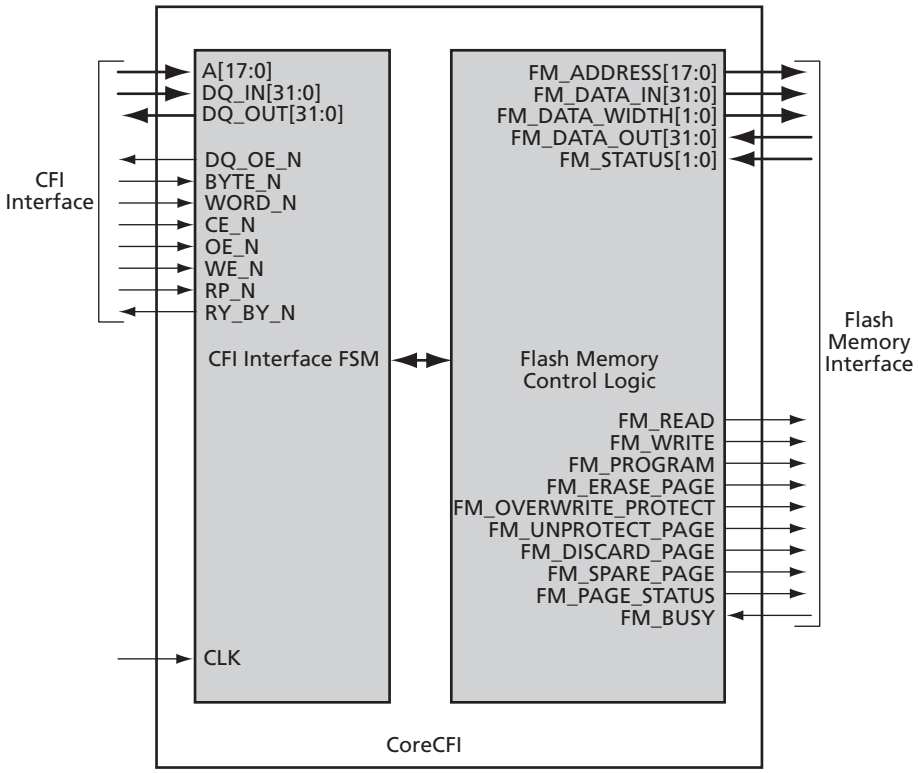

Figure 1 · CoreCFI Block Diagram

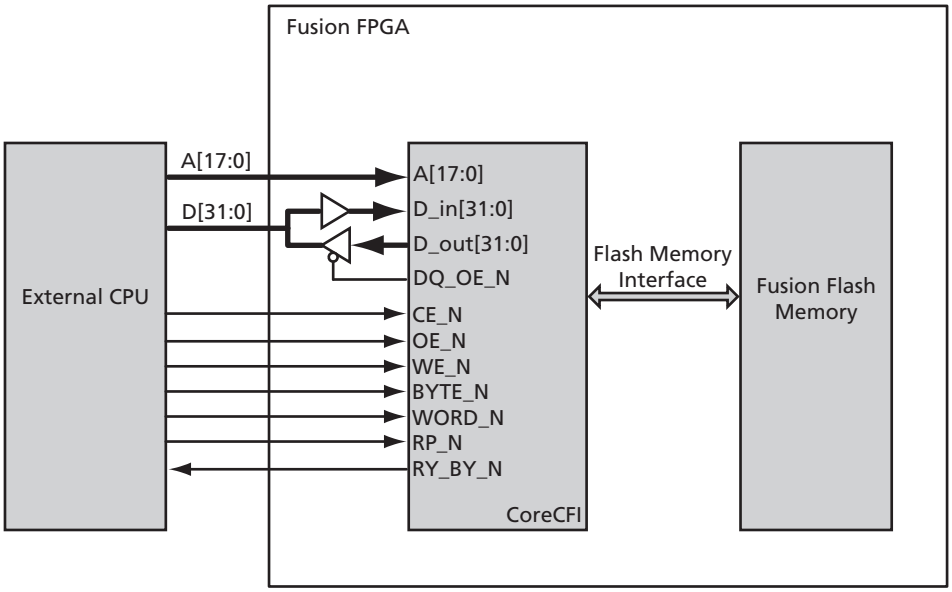

Figure 2 · CoreCFI Typical Application

Device Utilization and Performance

CoreCFI has been implemented in the Actel Fusion™ device family. A summary of the device utilization for CoreCFI is listed in Table 1 and Table 2.

Table 1 · CoreCFI Device Utilization and Performance (minimum configuration)

Family	Cells or Tiles			Device	Utilization	Performance
	Sequential	Combinatorial	Total			
Fusion	175	294	469	AFS090	20%	100 MHz

Note: Data in this table was achieved using typical synthesis and layout settings. Top-level parameters/generics that differ from the default values were set as follows: SIZE = 8.

Table 2 · CoreCFI Device Utilization and Performance (maximum configuration)

Family	Cells or Tiles			Device	Utilization	Performance
	Sequential	Combinatorial	Total			
Fusion	177	306	483	AFS090	21%	100 MHz

Note: Data in this table was achieved using typical synthesis and layout settings. Top-level parameters/generics that differ from the default values were set as follows: SIZE = 18.

1 Functional Description

The CoreCFI design is primarily a state machine that controls the interfaces to the Fusion Flash memory and the external CFI interface. CoreCFI implements a subset of the Common Flash Memory Interface Specification Release 2.0. It supports the following:

- Read and Read Query, Automatic Write and Erase, Lock, and Status operations
- 128-byte write page buffer and write/erase size
- 16-byte page read buffer
- 8-bit, 16-bit, and 32-bit operation

2 Tool Flows

Licenses

CoreCFI is licensed in three ways; depending on your license tool flow, functionality may be limited.

Evaluation

The precompiled simulation libraries provided allow the core to be instantiated in CoreConsole and simulated within Actel Libero® Integrated Design Environment (IDE) as described in "Simulation Flows" on page 10. The design cannot be synthesized, as the source code is not provided.

Obfuscated

Complete RTL code is provided for the core, enabling the core to be instantiated with CoreConsole. Simulation, Synthesis, and Layout can be performed with Libero IDE. The RTL code for the core is obfuscated, and some of the testbench source files are not provided. Instead, they are precompiled into the compiled simulation library.

RTL

Complete RTL source code is provided for the core and testbenches.

CoreConsole

CoreCFI is preinstalled in the CoreConsole IP Deployment Platform (IDP). To use the core,[1] click and drag it from the IP core list into the main window. The core can then be configured using the configuration GUI within CoreConsole, as shown in Figure 2-1 and Figure 2-2 on page 10. The CoreConsole project can be exported to Libero IDE at this point, providing access just to CoreCFI, or other IP blocks can be interconnected, allowing the complete system to be exported from CoreConsole to Libero IDE.

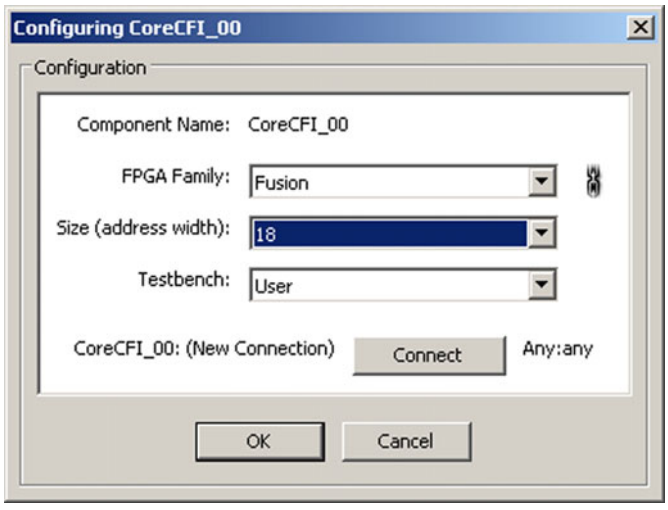

Figure 2-1 · CoreCFI Configuration within CoreConsole

1. *A CoreCFI license is required to generate the design for export to Libero IDE for Simulation and Synthesis .*

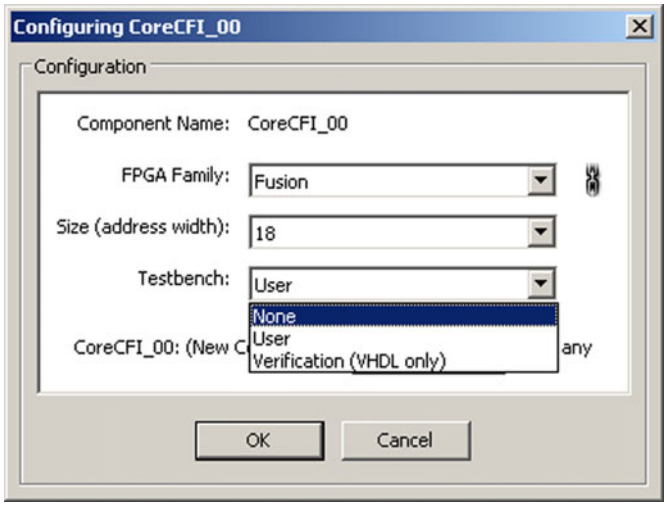

Figure 2-2 · CoreCFI Configuration within CoreConsole – Testbench Selection

Importing into Libero IDE

After generating and exporting the core from CoreConsole, the core can be imported into Libero IDE. Create a new project in Libero IDE and import the CoreConsole project from the *LiberoExport* directory. Libero IDE will then install the core and the selected testbenches, along with constraints and documentation, into its project.

Note: If two or more DirectCores are required, they can both be included in the same CoreConsole project and imported into Libero IDE at the same time.

Simulation Flows

To run simulations, the required testbench flow must be selected within CoreConsole, then **Save & Generate** must be run from the Generate pane. The required testbench is selected through the Core Testbench Configuration GUI. Two simulation testbenches are supported with CoreCFI:

- Simple CoreCFI application testbench (VHDL and Verilog)
- Full CoreCFI verification testbench (VHDL only)

When CoreConsole generates the Libero IDE project, it will install the appropriate testbench files. To run either the simple application or the full verification environment, simply set the design root to the CoreCFI instantiation in the Libero IDE design hierarchy and click the **Simulation** icon in the Libero Design Flow window. This will invoke Model*Sim*® and automatically run the simulation.

Synthesis in Libero IDE

Having set the design root appropriately, click the **Synthesis** icon in Libero IDE. The Synthesis window appears, displaying the Synplicity® project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To run Synthesis, click the **Run** icon.

Place-and-Route in Libero IDE

Having set the design root appropriately and run Synthesis, click the **Layout** icon in Libero IDE to invoke Designer. CoreCFI requires no special place-and-route settings.

3 Interface Description

Parameters

CoreCFI has parameters (Verilog) and generics (VHDL), described in Table 3-1, for configuring the RTL code. All parameters and generics are integers.

Table 3-1 · CoreCFI Parameter/Generic Descriptions

Parameter	Values	Description
FAMILY	0 to 99	Must be set to match the supported FPGA family: 17 – Fusion
SIZE	Integer 6 to 18	Indicates the number of address bits used by CoreCFI (i.e., the size of the Fusion Flash memory accessed by CoreCFI—e.g., 10 = 1 kB, 12 = 4 kB, 16 = 64 kB, 18 = 256 kB).

Signals

The port signals for the CoreCFI macro are defined in Table 3-2 on page 12 and illustrated in Figure 3-1. CoreCFI has 187 I/O signals. The user will need to create the device D pin by instantiating tristate I/O pads using the DQ_OE_N signal and the CoreCFI DQ_IN and DQ_OUT signals. This core is typically used with external device package pins (A, D, CE_N, OE_N, WE_N, RP_N, BYTE_N, WORD_N, and RY_BY_N as I/O pads—a total of 57 external I/Os), and it does not directly instantiate the Flash memory, though it does interface with the Flash memory, as shown in Figure 2 on page 6 (Flash memory interface signals begin with iFM_i). The user instantiates the Flash memory with the SmartGen software tool provided within Libero IDE.

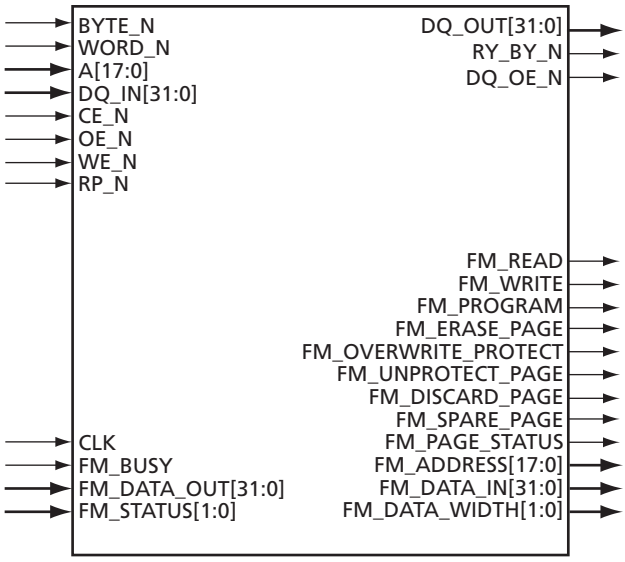

Figure 3-1 · CoreCFI I/O Signal Diagram

Table 3-2 · CoreCFI I/O Signal Descriptions

Name	Type	Description
WORD_N BYTE_N	Inputs	Active low. Controls the data width used by CoreCFI: WORD_N, BYTE_N X0 = 8-bit mode DQ_IN/OUT[7:0] active 01 = 16-bit mode DQ_IN/OUT[15:0] active 11 = 32-bit mode DQ_IN/OUT[31:0] active
A[17:0]	Input	Address inputs during read and write operations. The A[0] input is ignored in 16-bit mode—A[1] becomes the lowest-order address in 16-bit mode. The A[1:0] bits are ignored in 32-bit mode—A[2] becomes the lowest-order address in 32-bit mode (see the WORD_N, BYTE_N description).
DQ_IN[31:0]	Input	Data input pins during any write operation. DQ_IN[31:8] are ignored in 8-bit mode; DQ_IN[31:16] are ignored in 16-bit mode. The user should connect this to the receiver side of the CFI "D" bidirectional pads.
CE_N	Input	CoreCFI is selected when CE_N is asserted. This signal is active low and must be asserted for reads or writes to be executed.
OE_N	Input	DQ_OUT[31:0] will be enabled onto the external CFI databus when CE_N and OE_N are both asserted (assuming the user has used the DQ_OE_N signal as the active low output enable for the DQ_OUT pads—see the DQ_OE_N description). This signal is active low.
WE_N	Input	Writes to CoreCFI will be enabled when CE_N and WE_N are both asserted. Writes are ignored if CE_N or WE_N is not asserted. Writes take one clock cycle to execute. The actual command is executed on the clock edge following the WE_N sample. *At the end of a write, both CE_N and WE_N need to be synchronously deasserted.* This signal is active low.
RP_N	Input	Active low asynchronous reset. This signal resets the state of CoreCFI when asserted. It is recommended that this signal not be asserted while RY_BY_N is asserted; otherwise, the Fusion Flash memory device may be damaged. Reset places CoreCFI in Read Array mode, sets the status to 80h (ready), and tristates the data pins.
DQ_OUT[31:0]	Output	Data output pins during any read operation. DQ_OUT[31:8] are unused in 8-bit mode; DQ_OUT[31:16] are unused in 16-bit mode. The user should use this as the driver to the CFI "D" bidirectional pads.
DQ_OE_N	Output	Active low enable for the DQ_OUT pins. The user should use this signal as the active low output enable for the CFI "D" bidirectional pads. This signal is asserted when CE_N and OE_N are both asserted.

Note: All signals are active high (logic 1) unless otherwise noted.

Table 3-2 · CoreCFI I/O Signal Descriptions (continued)

Name	Type	Description
RY_BY_N	Output	Active low busy signal. When asserted, indicates that the Fusion Flash memory is performing a write operation. This signal is not asserted during a read operation. It is the user's responsibility to hold CE_N LOW for some period of time while the read data becomes valid. The latency will be lower for consecutive same-page accesses and larger for new read operations or cross-page-boundary accesses. Typical timings are 2 clock cycles and 5 clock cycles, respectively, for the above read operations. Alternatively, the user can monitor the Fusion memory FM_BUSY signal, which is asserted during a read operation. Refer to the *Fusion Family of Mixed-Signal Flash FPGAs* datasheet for specific timing information on reading data from the Flash memory.
CLK	Input	Flash memory interface clock. All operations and status are synchronous to the rising edge of this clock signal. Note that CoreCFI synchronizes the asynchronous inputs using this clock.
FM_BUSY	Input	When asserted, indicates that the Fusion Flash memory is performing an operation.
FM_DATA_OUT[31:0]	Input	Data returned from the Fusion Flash memory during a read
FM_STATUS[1:0]	Input	Status of the last operation completed
FM_READ	Output	When asserted, this signal initiates a Flash memory read operation.
FM_WRITE	Output	When asserted, this signal initiates writing the value present on the FM_DATA_IN[31:0] outputs to the assembly buffer of the Flash memory within the Fusion device.
FM_PROGRAM	Output	When asserted, this signal causes the contents of the assembly buffer to be written into the addressed cell array page in the Flash memory within the Fusion device.
FM_ERASE_PAGE	Output	When asserted, the addressed page is erased (all zeroes).
FM_OVERWRITE_PROTECT	Output	When asserted, all program operations will set the overwrite protect bit of the page being programmed.
FM_UNPROTECT_PAGE	Output	When asserted, the page addressed is copied into the Page Buffer, and the Page Buffer is made writable.
FM_DISCARD_PAGE	Output	When asserted, the contents of the Page Buffer are discarded so a new page write can be started.
FM_SPARE_PAGE	Output	When asserted, the sector addressed is used to access the spare page within that sector.
FM_PAGE_STATUS	Output	When this signal is asserted during a read, it indicates that the status for the currently addressed page is being accessed.
FM_ADDRESS[17:0]	Output	These output signals are used as the byte offset in the cell array, assembly buffer, or special function register interface within the Fusion Flash memory.

Note: All signals are active high (logic 1) unless otherwise noted.

Table 3-2 · CoreCFI I/O Signal Descriptions (continued)

Name	Type	Description
FM_DATA_IN[31:0]	Output	Data to be written to the Fusion Flash memory
FM_DATA_WIDTH[1:0]	Output	These output signals are used to select the data width mode of the Fusion Flash memory: 00 = 1 byte in FM_DATA_IN/OUT[7:0] active 01 = 2 bytes in FM_DATA_IN/OUT[15:0] active 10 = 4 bytes in FM_DATA_IN/OUT[31:0] active 11 = 4 bytes in FM_DATA_IN/OUT[31:0] active

Note: All signals are active high (logic 1) unless otherwise noted.

4 Supported CFI Commands

CoreCFI supports the Read Query, Read, Automatic Erase, Automatic Write, Lock, and Status CFI operations. The command descriptions are summarized in Table 4-1. The bus cycles are defined in Figure 4-1 on page 16, Figure 4-2 on page 21, and Figure 4-3 on page 22.

Table 4-1 · Command Descriptions

Command	No. of Bus Cycles	First Bus Cycle			Second Bus Cycle			Notes
		Operation	Address	Data	Operation	Address	Data	
Read Query	2	Write	X	98h	Read	QA	QD	1
Read ID Codes	2	Write	X	90h	Read	IA	ID	
Read Array	1 or 2	Write	X	FFh	Read	AA	AD	2
Read Status	2	Write	X	70h	Read	X	SD	
Clear Status	1	Write	X	50h				
Erase Page	2	Write	PA	20h	Write	PA	D0h	3
Single Write	2	Write	X	40h	Write	AA	AD	4
Multi-Write	2	Write	PA	E8h	Write	PA	N	5, 6
Page Lock	2	Write	X	60h	Write	PA	01h	7
Page Unlock	2	Write	X	60h	Write	PA	D0h	

Legend:

X = Any address within the device	IA = Identifier Address	AD = Array Data
QA = Query Address	ID = Identifier Data	SD = Status Data
QD = Query Data	A = Array Address	PA = Any address within the page

Notes:

1. *The Read Query does not require the address to be 55h.*
2. *The Write portion of the Read Array command is only needed if not already in Read Array mode.*
3. *The Erase Page operation will fail if the page is locked.*
4. *The page portion of the address is ignored for the second bus cycle.*
5. *The page specified by AA is the page the data will be written to. The page portion of the address is ignored once the Multi-Write command has been sent (note that this means that writes will wrap around onto the same current page if the page address goes outside the PA specified with the first bus cycle).*
6. *N is the number of elements (bytes / words / double words), minus one, to be written to the write buffer. Expected count ranges are N = 00h to N = 7Fh (e.g., 1 to 128 bytes) in 8-bit mode, N = 00h to N = 003Fh in 16-bit mode, and N = 00h to N = 1Fh in 32-bit mode. Bus cycles 3 and higher are for writing data into the write buffer. The confirm command (D0h) is expected after exactly N + 1 write cycles; any other command at that point in the sequence will prevent the transfer of the buffer to the array (the write will be aborted).*
7. *Locking a page prevents erasing or writing new data to the page.*
8. *All new commands are ignored while the device is busy.*

Read

CoreCFI Read operations, other than Read Array, are always preceded by a Write command to set up the read sequence. A preceding write is only required for the Read Array operation when the device is not already in Read Array mode. During read operations, CE_N and OE_N must be asserted, and WE_N and RP_N must be deasserted. The Fusion Flash memory device contains a 16-byte read page buffer that enables fast data transfers.

Write

All CoreCFI operations, other than Read Array, are always preceded by a Write command to set up the read sequence. A preceding write is only required for the Read Array operation when the device is not already in Read Array mode. During write operations, CE_N and WE_N must be asserted, and OE_N and RP_N must be deasserted.

Read Query Command

The Read Query command causes CoreCFI to load the query database from a spare page in the Fusion Flash memory. The algorithm for the Read Query command is shown in Figure 4-1. The query data for CoreCFI largely follows the Intel format and is summarized in Table 4-2 on page 17 through Table 4-5 on page 19. Query data is always supplied on the least significant 8 bits—D[7:0]. The address of the query data starts at 10h, 20h, or 40h in 32-bit, 16-bit, or 8-bit mode, respectively. The spare page address will be specified by the SmartGen tool through the use of the QUERY_PAGE generic/parameter. The SmartGen tool will also generate the query data to be stored in the specified spare page.

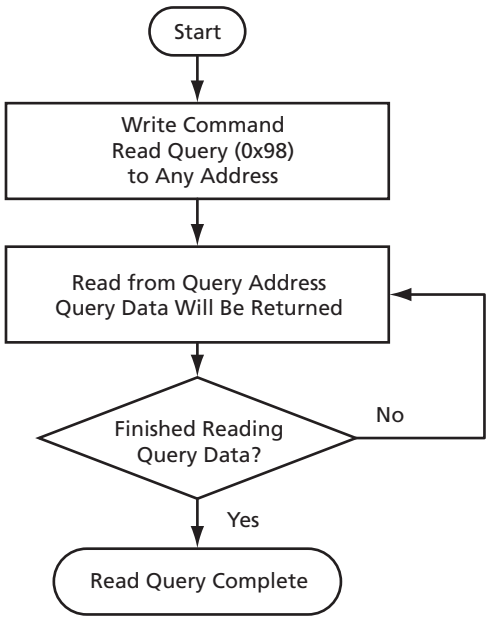

Figure 4-1 · CoreCFI Read Query Flow Diagram

Table 4-2 · CFI Query Identification

Offset [17:2]	Offset [17:1]	Offset [17:0]	Length (bytes)	Description	Hex Data	Notes
0x00	0x00	0x00	1	Manufacturer Code	0x5A	
0x01	0x02	0x04	1	Device Size Code	SIZE	Set to hex representation of SIZE generic
0x010 0x011 0x012	0x020 0x022 0x024	0x040 0x044 0x048	3	Query-unique ASCII string “QRY”	0x51 0x52 0x59	“Q” “R” “Y”
0x013 0x014	0x026 0x028	0x04C 0x050	2	Primary Algorithm Command Set and Control Interface ID Code 16-bit ID code defining a specific algorithm (refer to JEP137)	0x00 0x00	No command set specified
0x015 0x016	0x02A 0x02C	0x054 0x058	2	Address for Primary Algorithm extended Query, Table 4-5 on page 19 Address 0000h means that no extended table exists.	0x31 0x00	Table at offset 0x0031
0x017 0x018	0x02E 0x030	0x05C 0x060	2	Alternative Algorithm Command Set and Control Interface ID Code second specific algorithm supported by the device (refer to JEP137) ID Code = 0000h means that no alternate algorithm is employed.	00x00 0x00	No alternate command set exists in device.
0x019 0x01A	0x032 0x034	0x064 0x068	2	Address for Alternative Algorithm extended Query, Table 4-6 on page 21 Address 0000h means that no alternate extended table exists.	0x00 0x00	No alternate extended query exists in device.

Table 4-3 · CFI Query System Interface Information

Offset [17:2]	Offset [17:1]	Offset [17:0]	Length (bytes)	Description	Hex Data	Notes
0x01B	0x036	0x06C	1	V_{CC} Logic Supply Minimum Program/Erase or Write voltage Bits 7–4: BCD value in volts Bits 3–0: BCD value in hundreds of millivolts	0x30	3.0 V
0x01C	0x038	0x070	1	V_{CC} Logic Supply Maximum Program/Erase or Write voltage Bits 7–4: BCD value in volts Bits 3–0: BCD value in hundreds of millivolts	0x36	3.6 V
0x01D	0x03A	0x074	1	V_{PP} (programming) Supply Minimum Program/Erase voltage Bits 7–4: Hex value in volts Bits 3–0: BCD value in hundreds of millivolts	0x00	0.0 V – No V_{PP} pin
0x01E	0x03C	0x078	1	V_{PP} (programming) Supply Minimum Program/Erase voltage Bits 7–4: HEX value in volts Bits 3–0: BCD value in hundreds of millivolts	0x00	0.0 V – No V_{PP} pin
0x01F	0x03E	0x07C	1	Typical timeout per single byte/word/dword program, 2^N µs (if supported; 00h = not supported)	0x14	16 ms
0x020	0x040	0x080	1	Typical timeout for maximum-size multi-byte program, 2^N µs (if supported; 00h = not supported)	0x14	16 ms
0x021	0x042	0x084	1	Typical timeout per individual block erase, 2^N ms (if supported; 00h = not supported)	0x04	16 ms
0x022	0x044	0x088	1	Typical timeout for full chip erase, 2^N (if supported; 00h = not supported)	0x00	Not supported
0x023	0x046	0x08C	1	Maximum timeout for byte/word/dword program, 2^N times typical (offset 1Fh) (00h = not supported)	0x01	32 ms
0x024	0x048	0x090	1	Maximum timeout for multi-byte program, 2^N times typical (offset 20h) (00h = not supported)	0x01	32 ms
0x025	0x04A	0x094	1	Maximum timeout per individual block erase, 2^N times typical (offset 21h) (00h = not supported)	0x01	32 ms
0x026	0x04C	0x098	1	Maximum timeout for chip erase, 2^N times typical (offset 22h) (00h = not supported)	0x00	Not supported

Table 4-4 · CFI Query Device Geometry Definitions

Offset [17:2]	Offset [17:1]	Offset [17:0]	Length (bytes)	Description	Hex Data	Notes
0x027	0x04E	0x09C	1	Device Size = 2^N in number of bytes	0x12	262,144 bytes
0x028 0x029	0x050 0x052	0x0A0 0x0A4	2	Flash Device Interface Code description (refer to JEP137)	0x02 0x00	16/8 async, but also supports 32
0x02A 0x02B	0x054 0x056	0x0A8 0x0AC	2	Maximum number of bytes in multi-byte program = 2^N	0x07 0x00	128 bytes
0x02C	0x058	0x0B0	1	Number of Erase Block Regions within device. Bits 7:0 = x = number of erase block regions x = 0 means no erase blocking, i.e., the device erases at once in "bulk." *x* specifies the number of regions within the device containing one erase block region.	x01	Symmetrically blocked regions
0x02D 0x02E 0x02F 0x030	0x05A 0x05C 0x05E 0x060	0x0B4 0x0B8 0x0BC 0x0C0	4	Erase Block Region Information Bits 31:16 = z, where the erase block(s) within this region are z × 25 bytes in size. The value z = 0 is used for 128-byte block size. Bits 15:0 = y, where y + 1 = number of erase blocks of identical size within the erase block region	0x00 0x00 0xFF 0x07	128-byte regions 2,048 erase block regions

Table 4-5 · CFI Primary Vendor-Specific Extended Query

Offset [17:2]	Offset [17:1]	Offset [17:0]	Length (bytes)	Description	Hex Data	Notes
0x031 0x032 0x033	0x062 0x064 0x066	0x0C4 0x0C8 0x0CC	3	Primary Algorithm extended Query table; unique ASCII string "PRI"	0x50 0x52 0x49	"P" "R" "I"
0x034	0x068	0x0D0	1	Major version number, ASCII	0x31	"1"
0x035	0x06A	0x0D4	1	Minor version number, ASCII	0x31	"1"
0x036 0x037 0x038 0x039	0x06C 0x06E 0x070 0x072	0x0D8 0x0Dc 0x0E0 0x0E4	4	Optional feature and command support (1 = yes, 0 = no) Bits 9–3 are reserved; undefined bits are 0. If bit 31 is 1, another 31-bit field of optional features is at the end of the bit 30 field. Bit 0: Chip erase supported = no = 0 Bit 1: Suspend erase supported = no = 0 Bit 2: Suspend program supported = no = 0 Bit 3: Legacy lock/unlock supported = no = 0 Bit 4: Queued erase supported = no = 0 Bit 5: Instant individual block locking supported = no = 0 Bit 6: Protection bits supported = no = 0 Bit 7: Page mode read supported = yes = 1	0x80 0x00 0x00 0x00	Page mode read supported

Table 4-5 · CFI Primary Vendor-Specific Extended Query (continued)

Offset [17:2]	Offset [17:1]	Offset [17:0]	Length (bytes)	Description	Hex Data	Notes
0x03A	0x074	0x0E8	1	Supported functions after suspend: Read Array, Status, Query. Other supported operations: Bits 1–7: Reserved; undefined bits are 0. Bit 0: Program supported after erase suspend = yes = 1	0x00	No suspend
0x03B 0x03C	0x076 0x078	0x0EC 0x0F0	2	Block status register mask, 16 bits; active bits are 1; undefined bits are 0.	0x00 0x00	No block status
0x03D	0x07A	0x0F4	1	V_{CC} Logic Supply Highest Performance Program/Erase voltage Bits 0–3: BCD value in hundreds of millivolts Bits 4–7: BCD value in volts	0x33	3.3 V
0x03E	0x07C	0x0F8	1	V_{PP} Optimum Program/Erase Supply voltage Bits 0–3: BCD value in hundreds of millivolts Bits 4–7: Hex value in volts	0x00	0.0 V – No V_{PP} pin
0x03F	0x07E	0x0FC	1	Number of protection register fields in JEDEC ID space	0x01	No protection registers
0x040 0x041 0x042 0x043	0x080 0x082 0x084 0x086	0x100 0x104 0x108 0x10C	4	Protection Description Bits 0–7: Lock/bytes JEDEC-plane physical low address Bits 8–15: Lock/bytes JEDEC-plane physical high address Bits 16–23: N such that 2^N = factory pre-programmed bytes Bits 24–31: N such that 2^N = user-programmable bytes	0x00 0x00 0x00 0x00	No protection registers are supported.
0x044	0x088	0x110	1	Page Mode Read Capability. Number of read page bytes = 2^N in number of bytes	0x04	16 bytes
0x045	0x08A	0x114	1	Number of synchronous mode read configuration fields that follow	0x00	No sync burst
0x046	0x08C	0x118	1	Reserved for future use	0x00	–

Read ID Codes Command

The algorithm for the Read ID command is shown in Figure 4-2. The identifier codes returned are either values stored in the query data spare page or the lock status of a page in the Flash array.

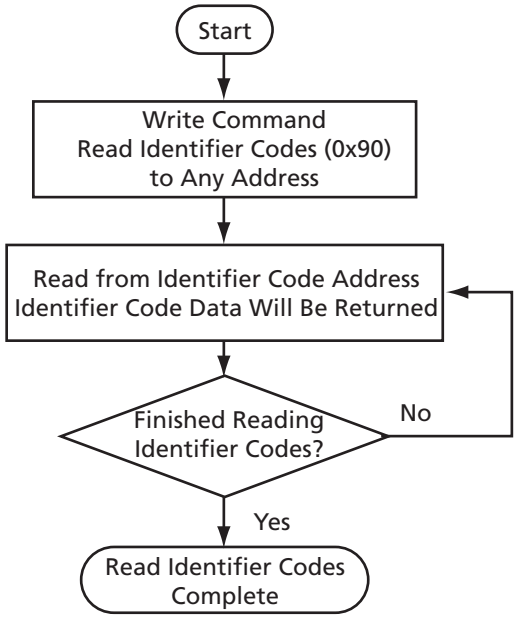

Figure 4-2 · CoreCFI Read ID Codes Flow Diagram

Table 4-6 · CFI Identifier Codes

Offset [17:2]	Offset [17:1]	Offset [17:0]	Length (bytes)	Description	Notes
0x00	0x00	0x00	1	Manufacturer Code	1
0x01	0x02	0x04	1	Device Size Code	2
BA+0x03	BA+0x04	BA+0x08	1	Page Lock Status	3

Notes:

1. *Manufacturer code default value is 0x5A.*
2. *Device size code is the hex representation of the SIZE generic. For example, the device size code is 0x08 for a 256-byte device and is 0x12 for a 256 kB device.*
3. *BA = the base address of the page for which to return status. For example, 0x00080 is the base address for page 1, so address 0x00088 would return the page lock status for page 1, in byte mode. The page lock status is returned on DQ[0] with the other data bits undefined. DQ[0] = 0: the page is unlocked. DQ[0] = 1: the page is locked.*

Read Array Command

The algorithm for the Read command is shown in Figure 4-3. CoreCFI comes out of reset in Read Array mode, and the Read Array command is not required to read the array after reset.

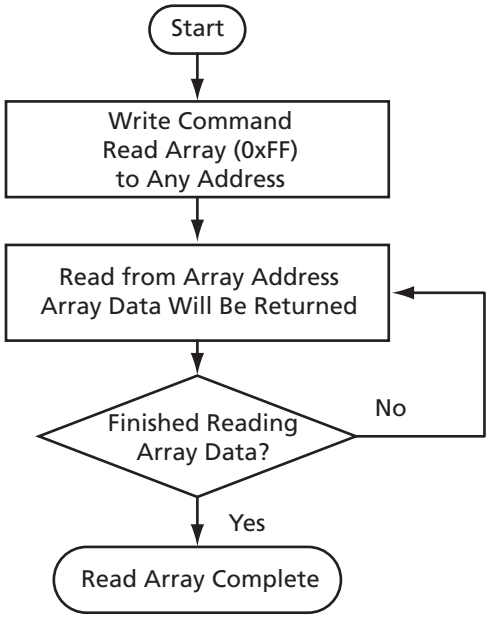

Figure 4-3 · CoreCFI Read Array Flow Diagram

Read Status Command

The algorithm for the Read Status command is shown in Figure 4-4. The status register can be read to determine the success of Write, Page Erase, or Lock Page commands. After writing the Read Status command, all subsequent read operations put out data from the status register until another valid command is written. The status is updated automatically when OE_N is toggled HIGH. When error conditions cause status register bits S5, S4, or S3 to be set, they can only be reset by the Clear Status command.

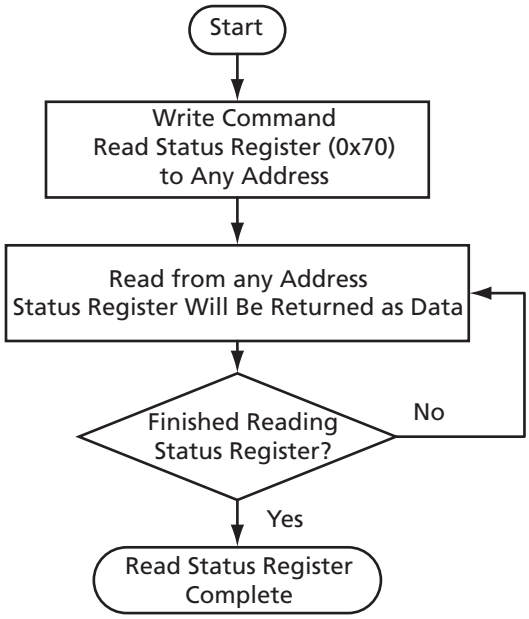

Figure 4-4 · CoreCFI Read Status Flow Diagram

Table 4-7 · Status Register

Status Bit	Description
S7	Busy – Indicates CoreCFI is currently performing a command and is not complete: 0 = Busy, 1 = Ready
S6	TBD
S5	Erase and Clear Lock Bit status 1 = Error in Page Erase or Clear Page Lock Bit 0 = Successful Page Erase or Clear Page Lock Bit Once set, this bit can only be cleared by a Clear Status command.
S4	Write and Set Page Lock Bit status 1 = Error in Write or Set Page Lock Bit 0 = Successful Write or Set Page Lock Bit Once set, this bit can only be cleared by a Clear Status command.
S3–S2	TBD
S1	Device protection status 1 = Page lock bit detected, operation aborted 0 = Page is unlocked Once set, this bit can only be cleared by a Clear Status command.
S0	TBD

Clear Status Command

The algorithm for the Clear Status command is shown in Figure 4-5. When error conditions cause status register bits S5, S4, or S3 to be set, they can only be reset by the Clear Status command.

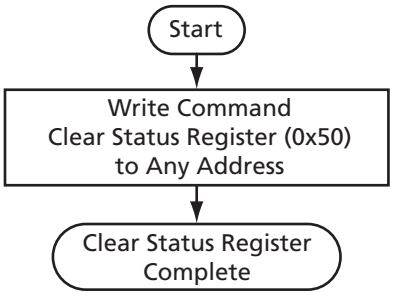

Figure 4-5 · CoreCFI Clear Status Flow Diagram

Erase Page Command

The algorithm for the Erase Page command is shown in Figure 4-6. The Erase Page requires two bus cycles to start: the command itself and a confirm. Once the erase starts, it cannot be interrupted (any subsequent commands are ignored while the erase is in progress). CoreCFI handles the required sequences, and the user can determine when the erase is complete by monitoring status bit S7 until busy is no longer indicated (note that the status is updated automatically, and the Read Status command sequence is not required). Once the Erase Page command has completed, status bits S1 and S5 should be checked to determine if any page erase error occurred. S1 will be set if the page to be erased is locked. S5 will be set if the Erase Page failed. If any of the error status bits are set, they can only be cleared by a Clear Status command.

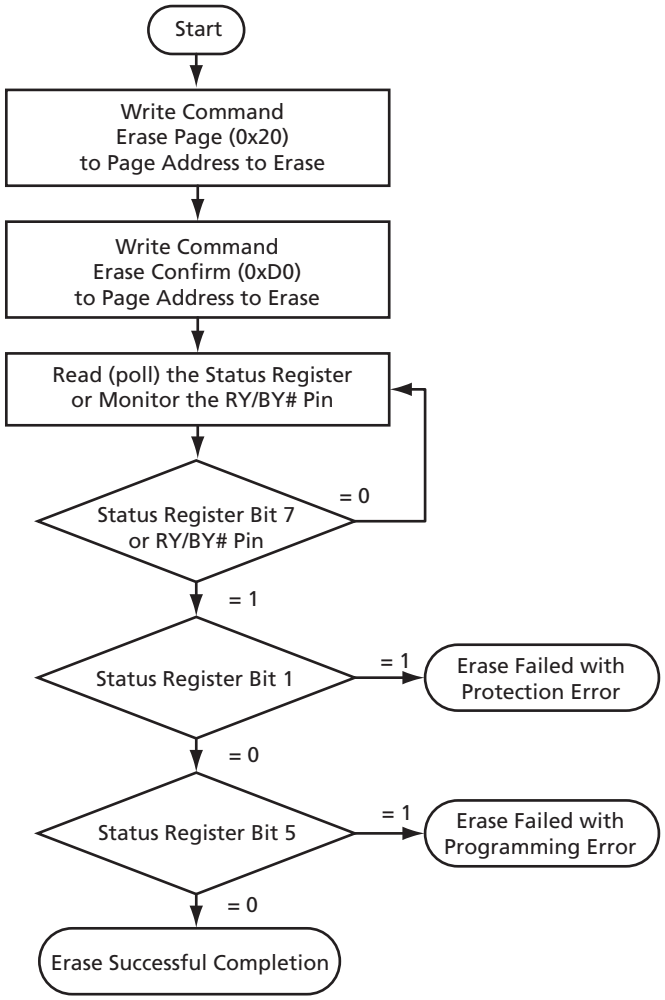

Figure 4-6 · CoreCFI Erase Page Flow Diagram

Single Write Command

The algorithm for the Single Write command is shown in Figure 4-7. The Single Write is used to write a single byte, word, or double word in 8-bit, 16-bit, or 32-bit mode. It should be noted that the Single Write still results in the entire page (that the Single Write data is contained within) being written into memory. Therefore, the user should avoid using single writes where multi-writes are more appropriate (i.e., when more than one location within a page is to be written). A single write is initiated by executing the Single Write command followed by a write to the desired location (note that all other commands are ignored once the write is in progress). Once the Write command has completed (i.e., the status no longer indicates busy—note that the status is updated automatically, and the Read Status command sequence is not required), status bits S1 and S4 should be checked to determine if any write error occurred. S1 will be set if a write is attempted on a page that is locked. S4 will be set if the write failed. If any of the error status bits are set, they can only be cleared by a Clear Status command.

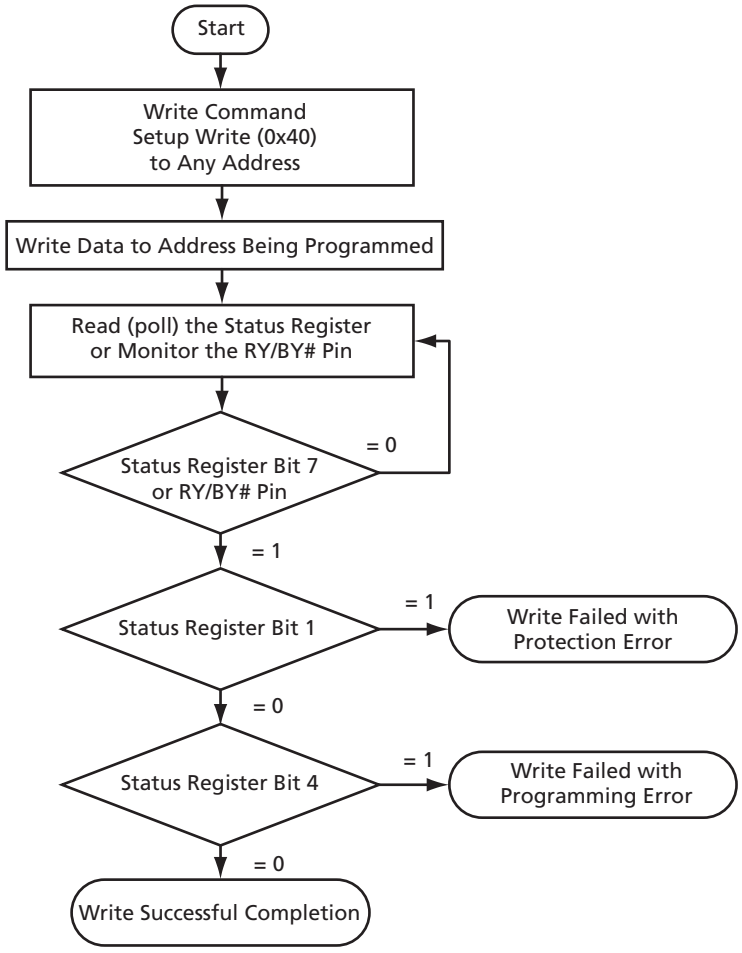

Figure 4-7 · CoreCFI Single Write Flow Diagram

Multi-Write Command

The algorithm for the Multi-Write command is shown in Figure 4-8 on page 28. The Multi-Write is used to write multiple bytes, words, or double words in 8-bit, 16-bit, or 32-bit mode. A multi-write is initiated by executing the Multi-Write command and waiting for the write buffer to become available (i.e., the status no longer indicates busy—note that the status is updated automatically, and the Read Status command sequence is not required). Status bit S1 should then be checked to make sure that it is not set due to the page being locked.

Once the write buffer if available, the second write with a data value of N is executed. N is the number of elements (bytes / words / double words), minus one, to be written to the write buffer—the expected ranges are N = 00h to N = 7Fh (e.g., 1 to 128 bytes) in 8-bit mode, N = 00h to N = 003Fh in 16-bit mode, and N = 00h to N = 1Fh in 32-bit mode.

Once N is written, the multiple writes to the desired locations within the page can be made. Note that once the Multi-Write command has been issued, the page address for the subsequent data writes is ignored (this means that writes will wrap around onto the current page if the page address goes outside the page address specified with the first bus cycle). Once the last data value has been written, the confirm command (D0h) is expected after exactly N + 1 write cycles; any other command at that point in the sequence will prevent the transfer of the buffer to the array (the write will be aborted). Note that all other command sequences are ignored once the confirm is received and the write to the array is started. Once the Multi-Write command has completed (i.e., the status no longer indicates busy—note that the status is updated automatically, and the Read Status command sequence is not required), status bit S4 should be checked to determine if any write error occurred. If any of the error status bits are set, they can only be cleared by a Clear Status command.

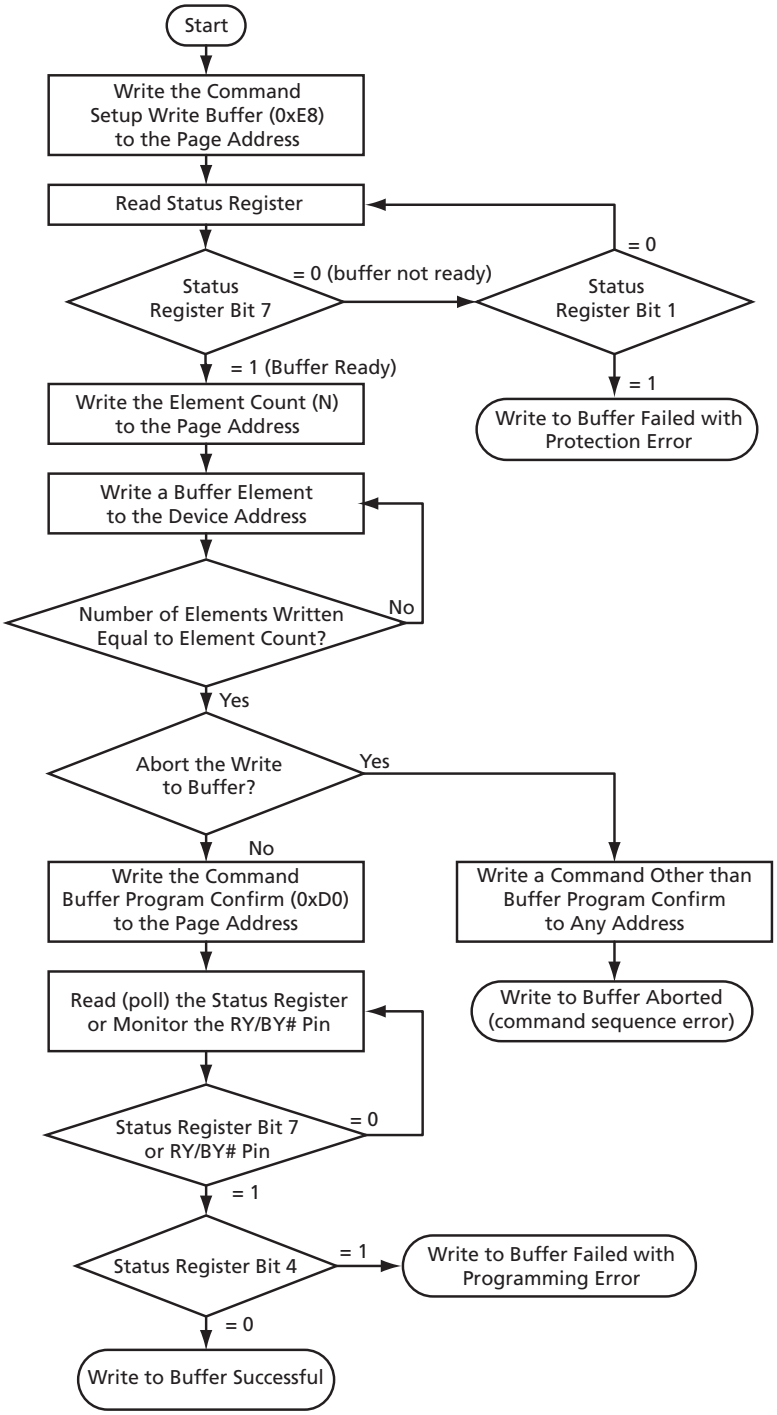

Figure 4-8 · CoreCFI Multi-Write Flow Diagram

Page Lock Command

The algorithm for the Page Lock command is shown in Figure 4-9. Locking a page prevents erasing or writing new data to the page. The Page Lock command is a two-bus-cycle operation—the first is the command itself, and the second specifies the page to be locked. Once the Page Lock command has completed (i.e., the status no longer indicates busy), status bit S4 should be checked to determine if an error occurred. S4 and S5 will be set if a command sequence error occurred; only S4 will be set if the Page Lock failed. If any of the error status bits are set, they can only be cleared by a Clear Status command.

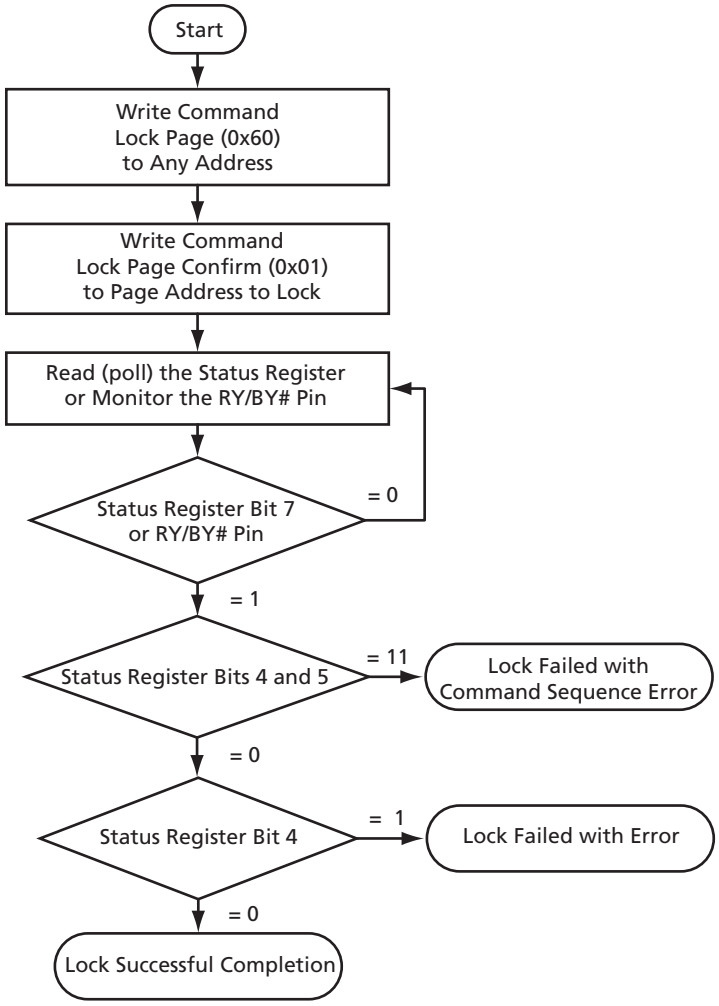

Figure 4-9 · CoreCFI Page Lock Flow Diagram

Page Unlock Command

The algorithm for the Page Unlock command is shown in Figure 4-10. Unlocking a page enables erasing or writing new data to the page. The Page Unlock command is a two-bus-cycle operation—the first is the command itself, and the second specifies the page to be unlocked. Once the Page Unlock command has completed (i.e., the status no longer indicates busy), status bit S5 should be checked to determine if an error occurred. S4 and S5 will be set if a command sequence error occurred; only S5 will be set if the Page Unlock failed. If any of the error status bits are set, they can only be cleared by a Clear Status command.

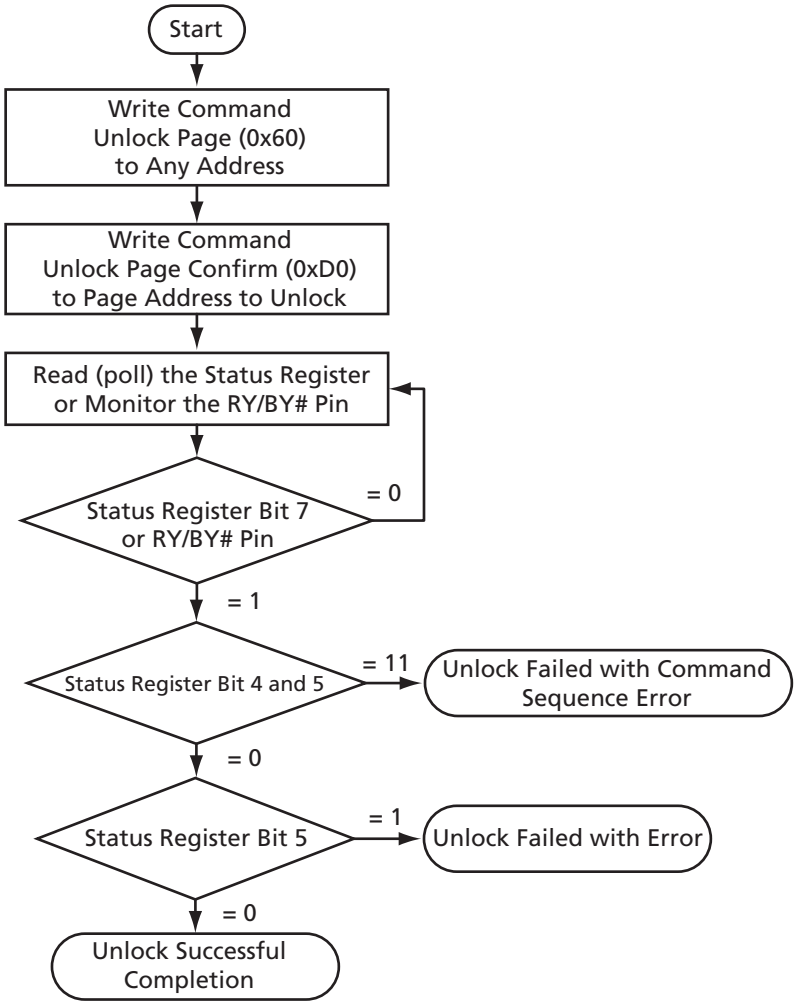

Figure 4-10 · CoreCFI Page Unlock Flow Diagram

Timing Diagrams

An example write waveform is shown in Figure 4-11. Note that the device will become busy for an extended time for any writes that require a write to the device array.

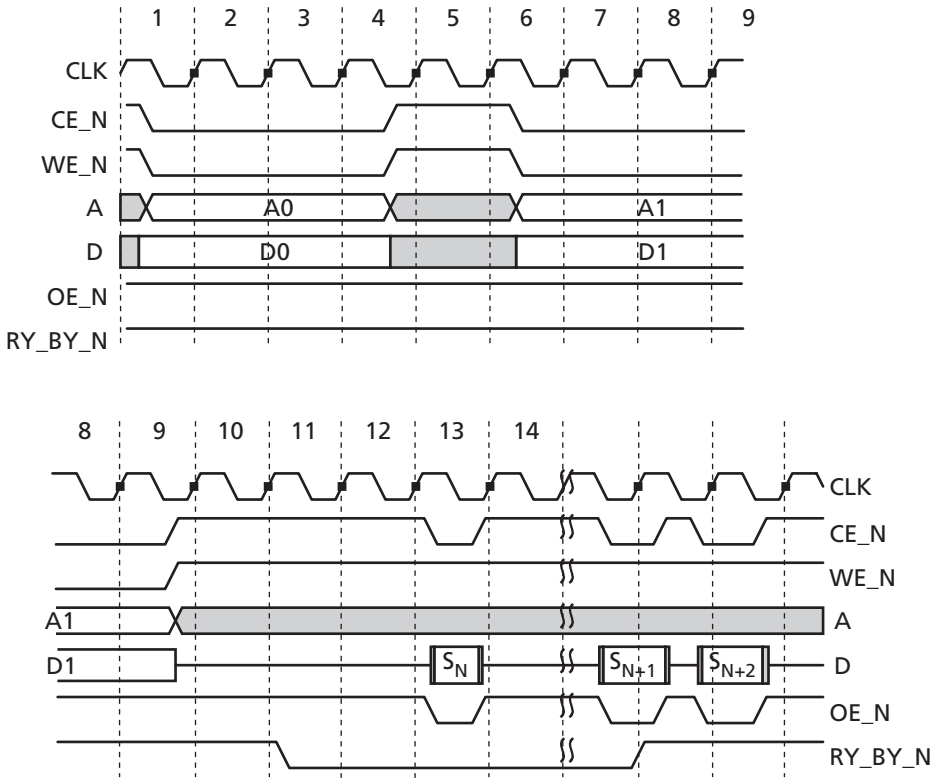

Figure 4-11 · CoreCFI Write Waveform

An example write followed by a read is shown in Figure 4-12.

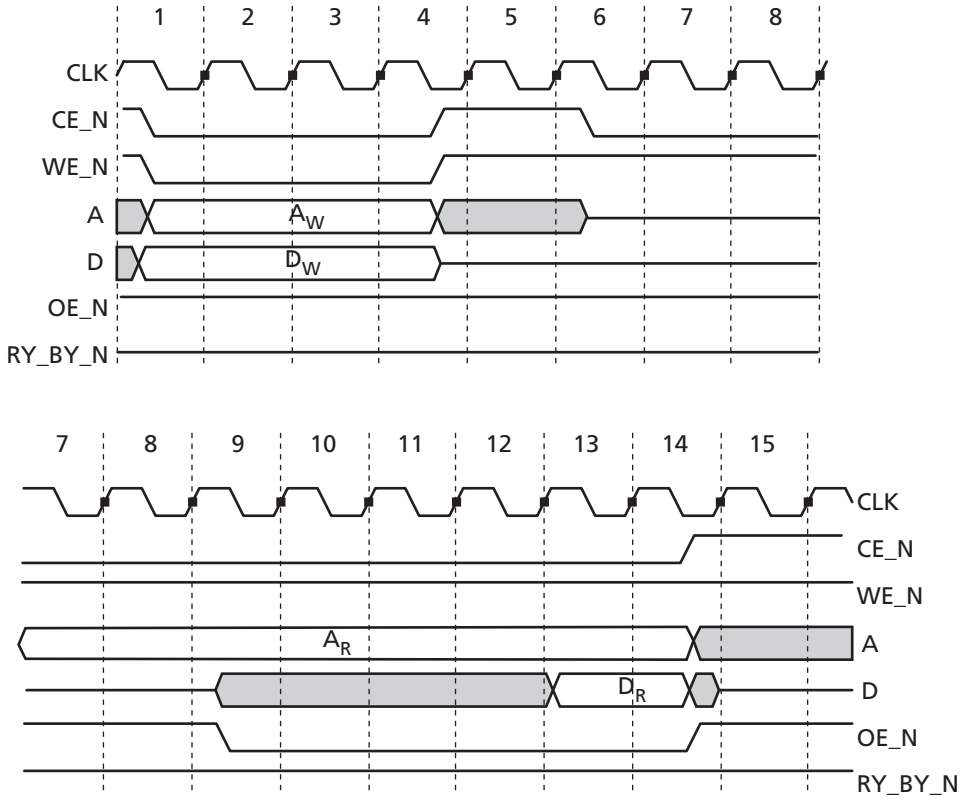

Figure 4-12 · CoreCFI Write–Read Waveform

An example read waveform, which crosses a read page boundary, is shown in Figure 4-13. Note that A_{bX} to A_{bY} represents a transition from one read page to another, and thus, the subsequent output data $D_{bY}0$ (associated with $A_{bY}0$) takes longer to become valid due to the time required to load the 16-byte read page buffer with the new page data.

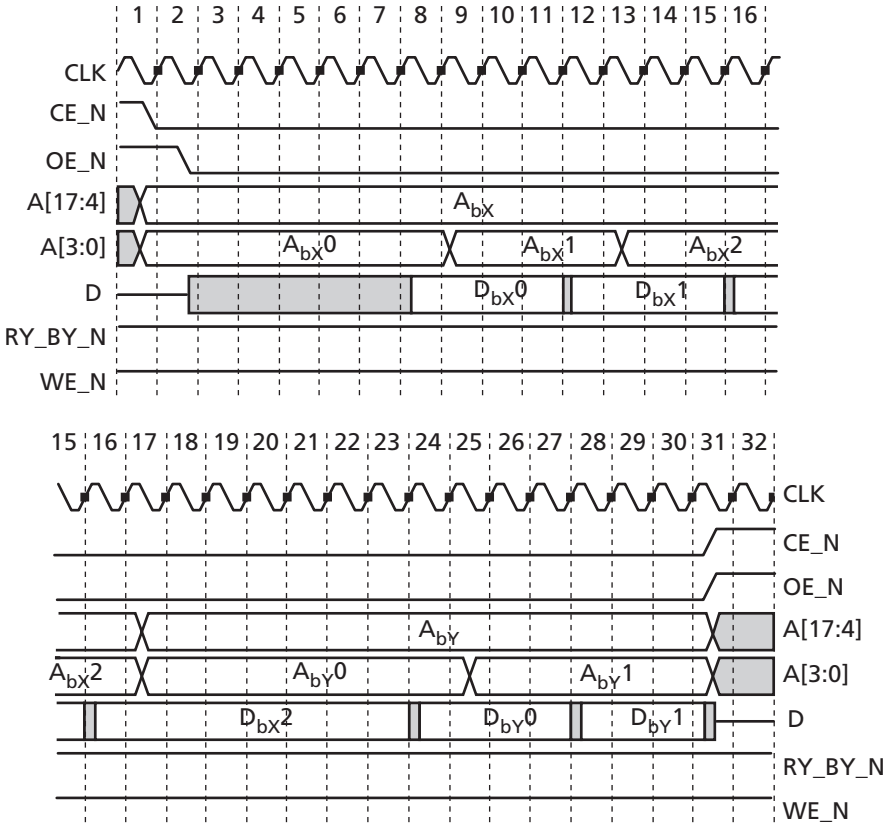

Figure 4-13 · CoreCFI Read Waveform

5 Implementation Hints

This chapter provides various hints to ease the process of implementation and integration of CoreCFI into your own design.

Usage with Internal Flash Memory

Proper operation of the CoreCFI design requires the use of the Fusion Flash memory. The Fusion Flash memory is an integral part of the CoreCFI design—CoreCFI will not function properly without it. CoreCFI provides a transparent interface to the Flash memory that should not be modified; CoreCFI should be connected to the Flash memory as shown in the user testbench file *corecfi_chip.v* (or *corecfi_chip.vhd*) located in the *coreconsole/CORECFI/rtl/<vhdl/vlog>/test/user* directory. If the interface is altered, it is likely that CoreCFI will cease to function properly.

It is anticipated that CoreCFI will be used as an interface for components external to the Fusion device. Components internal to the Fusion device would see the best performance if they use a direct interface to the internal Flash memory.

The Fusion Flash memory used with CoreCFI can be programmed through the CoreCFI interface, or it can be pre-programmed independently from the FPGA fabric by use of the FlashPro software and hardware (refer to the *FlashPro User's Guide* for details on how to program the Flash memory within Fusion devices). At a minimum, it is necessary to pre-program the CFI ID codes associated with CoreCFI into the Fusion Flash memory.

The Fusion Flash memory Program operation writes 128 bytes of data, regardless of the actual write size desired, for a given page (128 bytes) being written. If only one of the 128 bytes has been changed by the user, the other 127 bytes will be written again with the unchanged value. Therefore, it is best to keep this in mind when writing to the Flash memory. The endurance (lifetime) of the Flash memory will be maximized if the user minimizes single-location writes (e.g., write all of the desired locations for a given page using a multi-write instead of multiple single writes). Refer to the *Fusion Family of Mixed-Signal Flash FPGAs* datasheet for information on the Flash memory endurance specifications.

Generating and Programming the CFI Query Database

As previously described, use of CoreCFI requires that the Fusion Flash memory be initialized with the CFI ID codes. Development the programming file is done through the Actel SmartGen tool. SmartGen can be used for CoreCFI by selecting the CoreCFI client, adding it to the system, and generating the memory file for use by the Actel Designer tool. The default CFI codes provided with CoreCFI are shown in Table 5-1 and are provided in the *coreconsole/CORECFI/rtl/<vhdl/vlog>/test/user* directory as *ìcorecfi_query.memî*. Note that the data format is binary and that the ìSIZEî parameter (offset 01 in Table 5-1) should be modified to specify the desired size of the nonvolatile memory (NVM) used by CoreCFI.

Table 5-1 · Default CFI ID Codes

Offset	Default Binary Value	Hex Equivalent
00	01011010	5A
01	00010010	12
02	00000000	00
03	00000000	00
04	00000000	00
05	00000000	00
06	00000000	00
07	00000000	00
08	00000000	00

Table 5-1 · Default CFI ID Codes (continued)

Offset	Default Binary Value	Hex Equivalent
09	00000000	00
0A	00000000	00
0B	00000000	00
0C	00000000	00
0D	00000000	00
0E	00000000	00
0F	00000000	00
10	01010001	51
11	01010010	52
12	01011001	59
13	00000000	00
14	00000000	00
15	00110001	31
16	00000000	00
17	00000000	00
18	00000000	00
19	00000000	00
1a	00000000	00
1b	00110000	30
1c	00110110	36
1d	00000000	00
1e	00000000	00
1f	00011000	14
20	00011000	14
21	00000100	04
22	00000000	00
23	00000001	01
24	00000001	01
25	00000001	01
26	00000000	00
27	00010010	12
28	00000010	02
29	00000000	00
2a	00000111	07
2b	00000000	00
2c	00000001	01

Table 5-1 · Default CFI ID Codes (continued)

Offset	Default Binary Value	Hex Equivalent
2d	00000000	00
2e	00000000	00
2f	11111111	FF
30	00000111	07
31	01010000	50
32	01010010	52
33	01001001	49
34	00110001	31
35	00110001	31
36	10000000	80
37	00000000	00
38	00000000	00
39	00000000	00
3a	00000000	00
3b	00000000	00
3c	00000000	00
3d	00110011	33
3e	00000000	00
3f	00000001	01
40	00000000	00
41	00000000	00
42	00000000	00
43	00000000	00
44	00000100	04
45	00000000	00
46	00000000	00

When SmartGen is used to generate a programming file, it also generates a memory image file and an HDL file for use in simulation. These files can be used to simulate the CoreCFI-based design with the desired memory initialization. Refer to the *SmartGen User's Guide* for details on the use of the SmartGen tool.

6 Testbench Operation and Modification

Verification Testbench

Included with all releases of CoreCFI is a verification testbench that verifies operation of the CoreCFI macro. A simplified block diagram of the verification testbench is shown in Figure 6-1.

The verification test suite includes a verification testbench, a test driver, a set of test cases, and test configurations. The testbench instantiates and interconnects the DUT (design under test), which is the CoreCFI macro; the Fusion Flash memory behavioral model (BMOD); and the test driver. The test driver is used to provide an interface shell between the testbench and individual test cases (note that the testbench is a common testbench for all test cases—i.e., the testbench does not change from test to test). For each test case, there is a specific test configuration that associates the test case with the test driver. Note that there are a number of test cases that must be run to fully verify the CoreCFI IP; this can be done by running the *runsim.do* file.

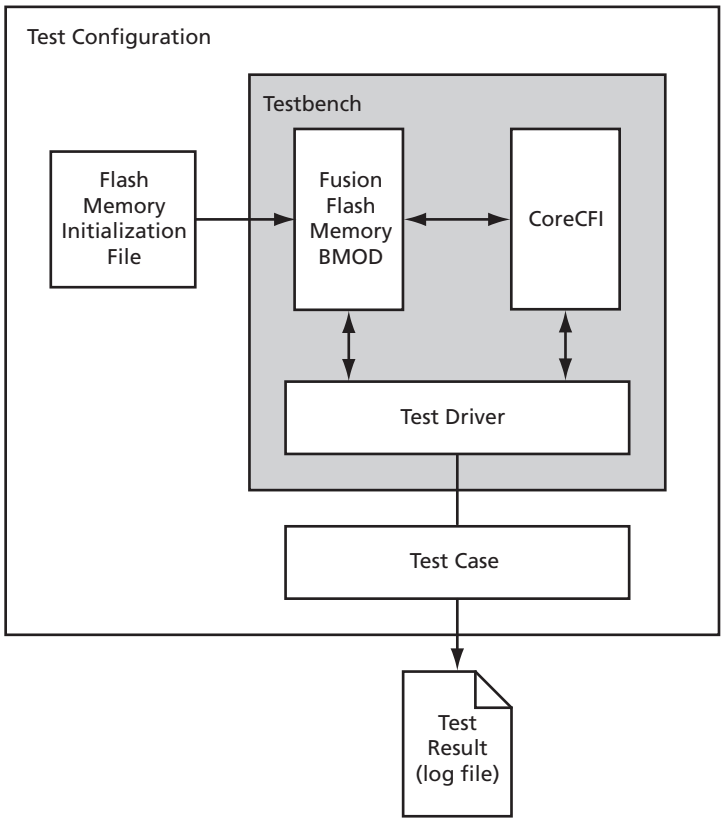

Figure 6-1 · CoreCFI Verification Testbench

The source code for the verification testbench is only available with the CoreCFI RTL release. A compiled Model*Sim* simulation is available with the Obfuscated and Evaluation releases.

Verification Tests

CoreCFI is verified through a number of tests that exercise CoreCFI through the external interface. The CoreCFI verification testbench uses the Fusion Flash memory behavioral model to simulate the behavior of the Flash memory in Fusion devices. The memory behavioral model is provided (as a library cell) as part of Libero IDE for all Actel Fusion products.

The verification testbench includes test procedures to check the following CFI operations:

- Read, Read Query, and Read ID Codes
- Single Write and Multi-Write
- Read and Clear Status
- Page Lock and Unlock
- Erase Page

Simple Application Testbench

An example user testbench is included with the Evaluation, Obfuscated, and RTL releases of CoreCFI. The user testbench is provided in precompiled Model*Sim* format and in RTL source code for all releases (Evaluation, Obfuscated, and RTL) for you to examine and modify to suit your needs. The source code for the user testbench is provided to ease the process of integrating the CoreCFI macro into your design and verifying according to your own custom needs. A block diagram of the user testbench is shown in Figure 6-2.

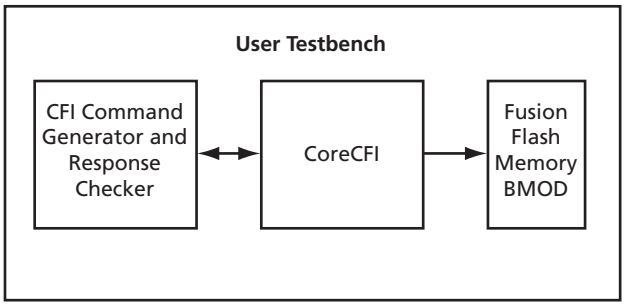

Figure 6-2 · CoreCFI User Testbench

The user testbench includes a simple example design that serves as a reference for users who want to implement their own designs.

The testbench for the example user design implements a subset of the functionality tested in the verification testbench, described in the previous chapter. Conceptually, as shown in Figure 6-2, CoreCFI is instantiated together with a behavioral microcontroller that controls the operation of CoreCFI via reads and writes to access internal registers.

Once you have familiarized yourself with the HDL source code for the user testbench, you may wish to customize, recompile, and run the simulation, as described in the "Interface Description" on page 11.

A

VHDL Testbench Support Routines

The verification testbench for the CoreCFI macro makes use of several support routines. The support routines are described in this appendix for the VHDL verification testbench.

The VHDL support routines (procedures and functions) are provided within a package. The support routines are referenced from within the verification testbench, via library and use clauses. To include these routines in a custom testbench, add the following two lines:

```
library CORECFI_LIB;
use CORECFI_LIB.CoreCFI_pkg.all;
```

A brief description of the support routines is given below.

cfi_erase	Perform the CFI Erase Page operation to erase a page in the Flash memory and wait for busy to deassert.
cfi_single_write	Perform the CFI Single Write command to write a single location in the Flash memory and wait for busy to deassert.
cfi_multi_write	Perform the CFI Multi-Write command to initiate a CFI multiple write command. The parameter write_count must be set to the number of data elements to be written, minus one. The buffer_write procedure must be used after this to fill the internal buffer and initiate the program.
cfi_change_lock	Perform the CFI Lock Page or Unlock Page command to protect or unprotect a page in the Flash memory. The parameter lock should be set to TRUE to protect the page or to FALSE to unprotect the page.
cfi_clear_status	Perform the CFI Clear Status command to clear any latched bits in the status register.
cfi_read_status	Perform the CFI Read Status command to enter the Read Status mode. Use the procedure flash_read for the actual data reads.
cfi_read_id	Perform the CFI read ID codes command to enter the read ID codes mode. Use the procedure flash_read for the actual data reads.
cfi_read_array	Perform the CFI Read Array command to enter the Read Array mode. Use the procedure flash_read for the actual data reads.
cfi_read_querry	Perform the CFI Read Query command to enter the Read Query mode. Use the procedure flash_read for the actual data reads.
proc_write	Write a data value to the CFI interface. This is used by the higher-level CFI procedures to write data to the CFI interface.
init_flash	Initialize the CFI interface and wait for busy to deassert.
flash_read	Read a data value from the CFI interface and verify the returned data. The parameter data_in contains the expected value for the verification. Used after the cfi_read_* procedures to read the data from the CFI.
flash_read_status	Read the status data from the CFI interface. The parameter expect_data contains the expected value for the verification, and the parameter expect_mask selects the bits to be verified (a "1" in the mask for a bit position selects the bit for verification). Note that the device must be in Read Status mode before using this procedure.
buffer_write	Fill the CFI buffer with data for a multiple write. Note that the cfi_multi_write command must be used prior to this to initiate the multiple write.
cfi_poll_busy	Used within other procedures to poll busy while waiting for a command to complete.

B

Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (www.actel.com/custsup/search.html) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at www.actel.com.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/contact/offices/index.html.

Index

U

V

W

For more information about Actel's products, visit our website at http://www.actel.com

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 USA*
Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd*. • River Court, Meadows Business Park • Station Approach, Blackwater • Camberley Surrey GU17 9AB • United Kingdom*
Phone +44 (0) 1276 609 300 • Fax +44 (0) 1276 607 540

Actel Japan *• EXOS Ebisu Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • www.jp.actel.com

Actel Hong Kong *• Suite 2114, Two Pacific Place • 88 Queensway, Admiralty Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200094-0 /3.07