XAPP138 (v2.7) July 11, 2002

Virtex FPGA Series Configuration and Readback

Summary

This application note is offered as complementary text to the configuration section of the Virtex™ data sheet. It is strongly recommended that the Virtex data sheets be reviewed prior to reading this note. Virtex FPGAs offer a broader range of configuration and readback capabilities than previous generations of Xilinx FPGAs. This note first provides a comparison of how Virtex configuration is different from previous Xilinx FPGAs, followed by a complete description of the configuration process and flow. Each of the configuration modes are outlined and discussed in detail, concluding with a complete description of data stream formats, and readback functions and operations.

Introduction

Configuration is the process of loading a design bitstream into the FPGA internal configuration memory. Readback is the process of reading that data.

Virtex configuration logic is significantly different from that of the XC4000 series, but maintains a great deal of compatibility to all Xilinx FPGA families. This information was prepared with the XC4000 series user in mind, but the new user of Xilinx FPGAs need not review XC4000 series configuration-related material.

Virtex Series vs. XC4000 Series Configuration

This section discusses the major configuration differences between the Virtex series and previous Xilinx FPGA families.

Configuration Modes and Daisy-Chains

Virtex FPGAs may be configured in eight different modes, shown in Table 1. There are four primary modes (Master Serial, Slave Serial, SelectMAP, and Boundary Scan), each with the option of having I/Os asserted or floating during configuration.

If pull-ups are selected for configuration, they are only active during configuration. After configuration, unused I/Os are de-asserted.

Serial Modes

The Master and Slave Serial modes perform essentially the same as those of previous FPGA families. For a detailed description, see "Master/Slave Serial Modes" on page 10.

Table 1: **Virtex Configuration Modes**

Configuration Mode	M2	M1	M0	Pull-ups
Master Serial	0	0	0	No
Slave Serial	1	1	1	No
SelectMAP	1	1	0	No
Boundary Scan	1	0	1	No
Master Serial (w/pull-ups)	1	0	0	Yes

Table 1: **Virtex Configuration Modes *(Continued)***

Configuration Mode	M2	M1	M0	Pull-ups
Slave Serial (w/pull-ups)	0	1	1	Yes
SelectMAP (w/pull-ups)	0	1	0	Yes
Boundary Scan (w/pull-ups)	0	0	1	Yes

Parallel Modes

The SelectMAP mode is the 8-bit parallel mode for Virtex devices that is similar to Express mode in XC4000XLA and Spartan™-XL. As with these other Xilinx device families, D0 is considered the MSB. For a detailed description, see "SelectMAP Mode" on page 11. Previous users of peripheral modes should find the transition to SelectMAP fairly straight-forward.

Virtex devices do not have a Master Parallel mode. Users who prefer to store configuration data on parallel EPROMs should read the Xilinx application note **XAPP137 "Configuring Virtex FPGAs from Parallel EPROMs"**.

Daisy-Chaining

Virtex FPGAs can be serially daisy-chained for configuration just as all previous Xilinx FPGAs, see "Master/Slave Serial Modes" on page 10. All devices in the chain must be in one of the serial modes. The SelectMAP mode does not support any serial daisy-chaining. Multiple Virtex devices can, however, be configured through the SelectMAP interface in a parallel fashion, see "SelectMAP Mode" on page 11. An example of this is also demonstrated in application note **XAPP137 "Configuring Virtex FPGAs from Parallel EPROMs"**.

Boundary Scan Interface

The Boundary Scan interface is always active from the moment of power-up; before, during, and after configuration. When resetting the configuration memory, $\overline{\text{PROGRAM}}$ going Low also resets the JTAG TAP controller. Boundary Scan modes select the optional pull-ups and prevent configuration in any other modes.

Configuring Virtex devices through the Boundary Scan interface is not described in this note. For more information on the Virtex Boundary Scan interface, refer to application note **XAPP139 "Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan"**.

Initialization and Timing

The initialization sequence for Virtex devices is somewhat simpler than for previous FPGAs. Upon power-up, the $\overline{\text{INIT}}$ signal is held Low while the FPGA initializes the internal circuitry and clears the internal configuration memory. Configuration may not commence until this cycle is complete, indicated by the positive transition of $\overline{\text{INIT}}$. Previous FPGA families required an additional waiting period after $\overline{\text{INIT}}$ went High before configuration could begin, Virtex devices do not. As soon as $\overline{\text{INIT}}$ transitions High after power-up, configuration may start. The Virtex configuration logic does, however, require several CCLK transitions to initialize itself. For this purpose, the Virtex bitstream is padded with several dummy data words at the beginning of the configuration stream. See "Bitstream Format" on page 15.

Mixed Voltage Environments

Virtex devices have separate voltage sources for the internal core circuitry (V_{CORE} = 2.5V) and the I/O circuitry (SelectI/O™). The SelectI/O resource is separated into eight banks of I/O groups. Each bank may be configured with one of several I/O standards. Refer to the Virtex data sheets for I/O banking rules and available I/O standards. Before and during configuration, all I/O banks are set for the LVTTL standard, which requires an output voltage (V_{CCO}) of 3.3 V for normal operation.

All configuration output pins are located within banks 2 and 3. Therefore, only V_{CCO_2} and V_{CCO_3} pins need a 3.3V supply for output configuration pins to operate normally. This is a requirement for Master Serial configuration and readback through the SelectMAP ports.

If the FPGA is being configured in Master Serial mode, and banks 2 and 3 are being configured for an I/O standard that requires a V_{CCO} other than 3.3V, then V_{CCO_2} and V_{CCO_3} need to be switched from the 3.3V used during configuration to the voltage required after configuration.

If readback is performed through the SelectMAP mode after configuration, then V_{CCO_2} and V_{CCO_3} require a 3.3V supply after configuration as well.

For Serial Slave and SelectMAP configuration modes, V_{CCO} can be any voltage (as long as it is ≥ 1.8 V ≤ 3.3 V) provided one meets the V_{IH}/V_{IL} levels of the resulting input buffer (see data sheet). Any pin that is a shared I/O, such as $\overline{INIT}$, DOUT/BUSY, and DONE should have an added pull-up resistor. The dedicated CONFIG pins should be pulled up to at least V_{CCINT}. Additionally, V_{CCO_2} must be pulled to a value above 1.0V during power-up of the FPGA.

JTAG inputs are independent of V_{CCO} and work between 2.5V and 3.3V TTL levels. TDO is sourced from V_{CCO_2} and should be 1.8V, 2.5V, or 3.3V depending on what the TDI of the next device accepts.

BitGen Switches and Options

This section describes new optional settings for bitstream generation that pertain only to Virtex devices. The new BitGen options are listed in Table 2 and described below.

Table 2: **Virtex-Specific BitGen Options**

Switch	Default Setting	Optional Setting
Readback	N/A	N/A
ConfigRate MHz (nominal)	4	4, 5, 7, 8, 9, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60
StartupClk	CCLK	UserClk, JtagClk
DONE_cycle	4	1, 2, 3, 5, 6
GTS_cycle	5	1, 2, 3, 4, 6, DONE
GSR_cycle	6	1, 2, 3, 4, 5, DONE
GWE_cycle	6	1, 2, 3, 4, 5, DONE
LCK_cycle	NoWait	0, 1, 2, 3, 4, 5, 6
Persist	No	Yes, No
DriveDONE	No	Yes
DonePipe	No	Yes
Security	None	Level1, Level2
UserID	FFFF FFFF	<hex string> (32-bit)
Gclkdel0	11111	<binary string>11111
Gclkdel1	11111	<binary string>
Gclkdel2	11111	<binary string>
Gclkdel3	11111	<binary string>

Readback

The Readback option causes BitGen to write out a readback command file `<design>.rbb`. For more information, see "Readback" on page 24.

ConfigRate

The ConfigRate is the internally generated frequency of CCLK in Master Serial mode. The initial frequency is 2.5 MHz. The CCLK changes to the selected frequency after the first 60 bytes of the bitstream have been loaded. For details, see "Bitstream Format" on page 15. It should also be noted that the CCLK periods have a variance of –30% to +45% from the specified value.

StartupClk

The StartupClk option selects a clock source to synchronize the Start-up Sequence. The default is CCLK which is standard for most configuration schemes. However, some applications require that the Start-up Sequence be synchronized to another clock source (UserClk) which must be specified in the user design. If configuring in Boundary Scan, select the JTAGClk option. For more information on Boundary Scan, refer to application note **XAPP139 "Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan"**.

DONE_cycle

The DONE_cycle specifies which state of the Start-up Sequence releases the DONE pin. For more information on the Start-up Sequence, see "Start-up Sequence" on page 5.

GSR_cycle

The GSR_cycle specifies which state of the Start-up Sequence releases the internal GlobalSetReset signal. The GSR signal holds all internal flip-flops in their configured initial state. The DONE setting asserts the GSR asynchronously as DONE transitions High unless the DonePipe option is used. If the DonePipe option is used, it releases GSR on the first rising edge of the StartupClk after DONE transitions High.

GWE_cycle

The GWE_cycle specifies which state in the Start-up Sequence releases the internal GlobalWriteEnable signal. This signal is not accessible to the user. It keeps all flip-flops, and RAM from changing state. However, DLL is not affected by GWE. The DONE setting asserts GWE asynchronously as DONE transitions High unless the DonePipe option is used. If the DonePipe option is used, it releases GWE on the first rising edge of the StartupClk after DONE transitions High.

GTS_cycle

The GTS_cycle specifies which cycle of the Start-up Sequence releases the internal Global 3-state signal. The GTS signal holds all outputs disabled. The DONE setting asserts the GTS asynchronously as DONE transitions High unless the DonePipe option is used. If the DonePipe option is used, it releases GSR on the first rising edge of the StartupClk after DONE transitions High.

LCK_cycle

The LCK_cycle specifies in which state the Start-up Sequence should stay until a DLL has established a Lock. The default setting of *NoWait* is used whenever a DLL is not used in a design. When a DLL is used and the default setting is selected, the Start-up Sequence should not be delayed for a DLL lock. If a wait state is specified by this option, the Start-up Sequence proceeds to the specified state, but then waits in that state until DLL lock occurs.

Since there are four DLLs per device, the LCK_cycle option must be used with a DLL attribute in the design. For more information on DLL attributes, see application note **XAPP132 "Using the Virtex Delay-Locked Loop"**.

Persist

If the Persist option is unspecified, or specified with a default setting of *No*, then all configuration pins other than CCLK, $\overline{\text{PROGRAM}}$, and DONE become user I/O after configuration. The Persist switch causes the configuration pins to retain their configuration function even after configuration. The *X8* setting applies to the SelectMAP interface and *X8* setting must be

selected if readback is to be performed through the SelectMAP interface. The Persist switch does not affect Boundary Scan ports.

DriveDONE

By default, the DONE pin is an open-drain driver. However, if the DriveDONE option is set to *Yes*, then DONE becomes an active driver, and no external pull-up is needed.

DonePipe

Independent of the DONE_cycle setting, after releasing DONE, the Start-up Sequence waits for DONE to be externally asserted High before continuing. The rise time of DONE depends on its external capacitive loading and is less than one CCLK period.

The DonePipe option adds a pipeline register stage between the DONE pin and the start-up circuitry. Useful for high configuration speeds when the rise time for DONE cannot be faster than one CCLK period.

Security

Security level settings restrict access to configuration and readback operations. If the Persistence option is not set, then configuration ports are not available after configuration. However, the Boundary Scan ports are always active and have access to configuration and readback.

Setting security *Level 1* disables all readback functions from either the SelectMAP or Boundary Scan ports.

Setting security *Level 2* disables all configuration and readback functions from all configuration and Boundary Scan ports.

The only way to remove a security level in a configured device is to de-configure it by asserting $\overline{\text{PROGRAM}}$ or recycling power.

UserID

The UserID is a 32-bit data word accessible through the Boundary Scan USERCODE command. The data word can be any arbitrary 32-bit value. To include a UserID in a bitstream, set the UserID option to the HEX representation of the desired data word <XXXXXXXX>h.

Gclkdel

The Gclkdel option adds delay to one of the four global clock buffers. This option is only used in PCI applications.

CCLK and LengthCount

Previously, Xilinx FPGA families used a LengthCount number embedded in the bitstream. This LengthCount number indicated to the FPGA how many CCLK cycles should be observed before activating the Start-up Sequence to activate the FPGA. This method also requires the FPGA not to receive any CCLK transitions prior to the loading of the bitstream. Otherwise, the LengthCount would be wrong and the Start-up Sequence would not activate at the appropriate time. Thus, free-running oscillators can not be used to generate the CCLK for configuration.

Virtex FPGAs do not use any such LengthCount number in configuration bitstreams. The Start-up Sequence for Virtex devices is controlled by a set of configuration commands that are embedded near the end of the configuration bitstream. See "Bitstream Format" on page 15. Therefore, Virtex FPGAs may have a free running oscillator driving the CCLK pin.

Start-up Sequence

Start-up is the transition from the configuration state to the operational state. The Start-up Sequence activates an FPGA upon the successful completion of configuration. The Start-up Sequencer is an 8-phase sequential state machine that transitions from phase 0 to phase 7. See Figure 1.

The Start-up Sequencer performs the following tasks:

1. Releases the DONE pin.
2. Negates GTS, activating all the I/Os.
3. Asserts GWE, allowing all RAMs and flip-flops to change state (flip-flops cannot change state while GSR is asserted).
4. Negates GSR, allowing all flip-flops to change state.
5. Asserts EOS. The End-Of-Start-up flag is always set in phase 7. This is an internal flag that is not user accessible.

The order of the Start-up Sequence is controlled by BitGen options. The default Start-up Sequence is the bold line shown in Figure 1. The Start-up Sequence may also be stalled at any phase until either DONE has been externally forced High, or a specified DLL has established LOCK. For details, See "BitGen Switches and Options" on page 3.

At the cycle selected for the DONE to be released, the sequencer always waits in that state until the DONE is externally released. This is similar to the SyncToDONE behavior in the XC4000 FPGAs. However, this does not hold off the GTS, GSR, or GWE if they are selected to be released prior to DONE. Therefore, DONE is selected first in the sequence for default settings.

For a true SyncToDONE behavior, set the GTS, GSR, and GWE cycles to a value of DONE in the BitGen options. This causes these signals to transition as DONE externally transitions High.

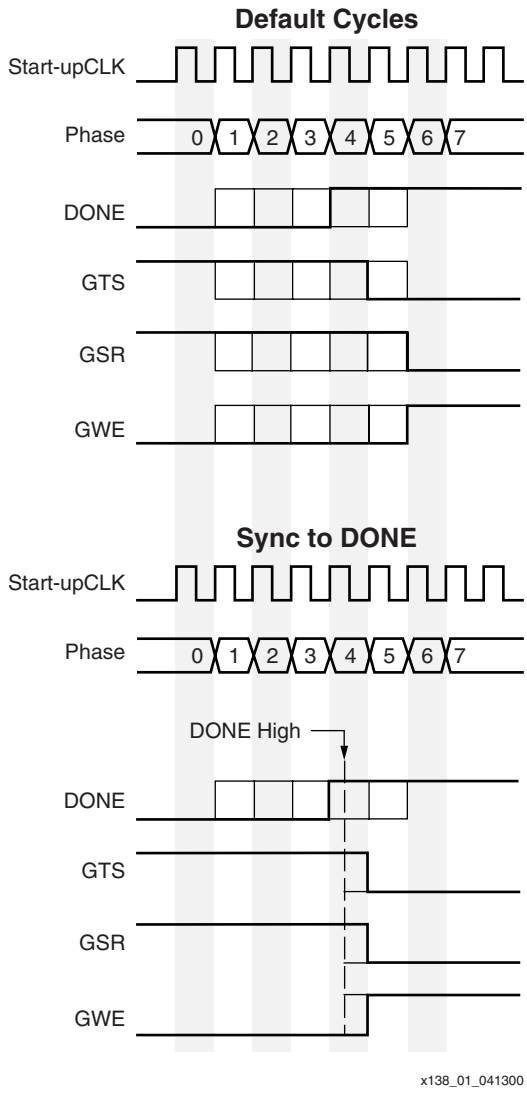

Figure 1: **Default Start-up Sequence**

Configuration Process and Flow

The external configuration process is simply a matter of loading the configuration bitstream into the FPGA using the selected configuration mode. The configuration process follows the flow illustrated in Figure 2.

Power-Up

The V_{CCint} power pins must be supplied with a 2.5V source. The rise time for the core voltage should be a maximum of 50 ms to rise from 1.0V to 2.4V. The IOB output voltage input for Bank 2 (V_{CCO_2}) is also used as a logic input to the Power-On-Reset (POR) circuitry. This value must be greater than 1.0V for power-up to continue. If this bank is not being used, a pull-up should be added to V_{CCO_2}.

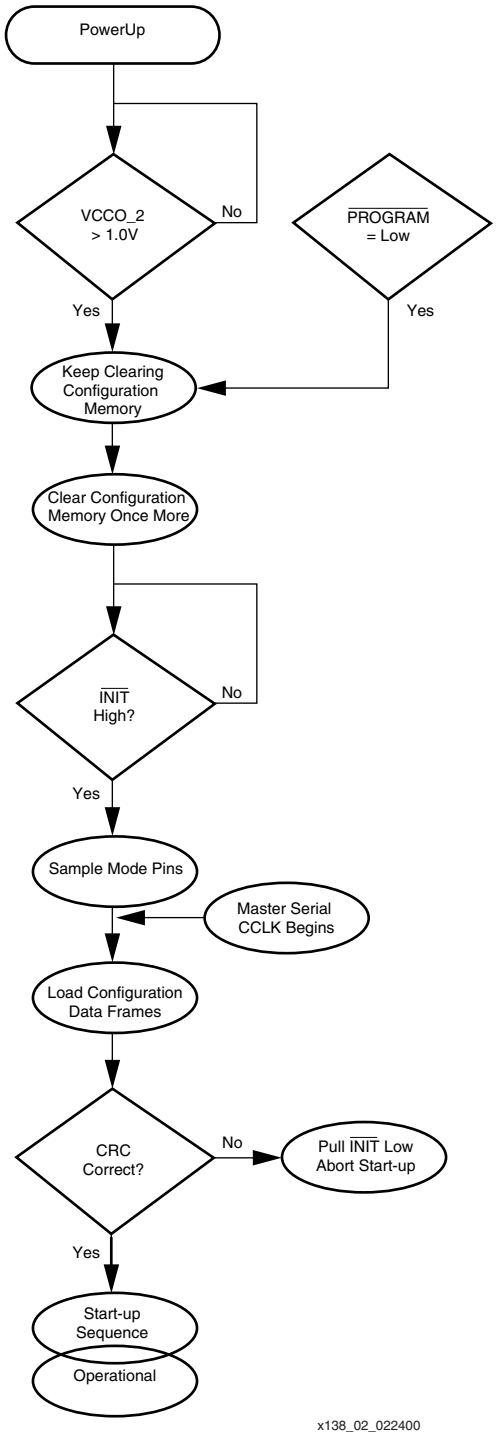

Figure 2: **Configuration Flow Diagram**

Clearing Configuration Memory

After power-up, the configuration memory is automatically cleared. The $\overline{\text{INIT}}$ pin transitions High when the clearing of configuration memory is complete. A logic Low on the $\overline{\text{PROGRAM}}$ input resets the configuration logic and holds the FPGA in the clear configuration memory state. As long as the $\overline{\text{PROGRAM}}$ pin is held Low, the FPGA continues to clear its configuration memory while holding $\overline{\text{INIT}}$ Low to indicate the configuration memory is being cleared. When $\overline{\text{PROGRAM}}$ is released, the FPGA continues to hold $\overline{\text{INIT}}$ Low until it has completed clearing all

the configuration memory. The minimum Low pulse time for $\overline{\text{PROGRAM}}$ is 300 ns. There is no maximum value.

Delaying Configuration

The $\overline{\text{INIT}}$ pin may also be held Low externally to delay configuration of the FPGA. The FPGA samples its mode pins on the rising edge of $\overline{\text{INIT}}$. After $\overline{\text{INIT}}$ has gone High, configuration may begin. No additional time-out or waiting periods are required, but configuration does not need to commence immediately after the transition of $\overline{\text{INIT}}$. The configuration logic does not begin processing data until the synchronization word from the bitstream is loaded.

Loading Configuration Data

The details of loading the configuration data are discussed in the following sections of the configuration modes, see "Master/Slave Serial Modes" on page 10 and **SelectMAP Mode**, page 11.

CRC Error Checking

Twice during the loading of configuration data, an embedded CRC value is checked against an internally calculated CRC value. The first check is just before the last configuration frame is loaded, and the second is at the very end of configuration. If the CRC values do not match, $\overline{\text{INIT}}$ is asserted Low to indicate that a CRC error has occurred. Start-up is aborted, and the FPGA does not become active.

To reconfigure the device, the $\overline{\text{PROGRAM}}$ pin should be asserted to reset the configuration logic. Recycling power also resets the FPGA for configuration. For more information on CRC calculation, see "Cyclic Redundancy Checking Algorithm" on page 23.

Start-up and Operational States

Upon successful completion of the final CRC check, the FPGA enters the Start-up Sequence. This sequence releases DONE (it goes High), activates the I/Os, de-asserts GSR, and asserts GWE. At this point, the FPGA becomes active and functional with the loaded design. For more information on start-up, see "Start-up Sequence" on page 5.

Configuration Pins

Certain pins in the FPGA are designated for configuration and are listed in Table 3. Some pins are dedicated to the configuration function and others are dual-function pins that can be user I/O after configuration.

Table 3: **List of Configuration Pins**

Name	Direction	Driver Type	Description
Dedicated Pins			
CCLK	Input/Output	Active	Configuration clock. Output in Master mode.
PROGRAM	Input		Asynchronous reset to configuration logic.
DONE	Input/Output	Active/ Open-Drain	Configuration status and start-up control.
M2, M1, M0	Input		Configuration mode selection.
TMS	Input		Boundary Scan mode select.
TCK	Input		Boundary Scan clock.
TDI	Input		Boundary Scan data input.
TDO	Output	Active	Boundary Scan data output.

Table 3: **List of Configuration Pins *(Continued)***

Name	Direction	Driver Type	Description
Dual Function Pins			
DIN (D0)	Input		Serial configuration data input.
D1:D7	Input/Output	Active Bidirectional	SelectMAP configuration data input, readback data output.
CS	Input		Chip Select (SelectMAP mode only).
WRITE	Input		Active Low write select, read select (SelectMAP only).
BUSY/DOUT	Output	3-state	Busy/Ready status for SelectMAP mode. Serial configuration data output for serial daisy-chains (active).
INIT	Input/Output	Open-Drain	Delay configuration, indicate configuration error.

Master/Slave Serial Modes

In serial configuration mode, the FPGA is configured by loading one bit per CCLK cycle. In Master Serial mode, the FPGA drives the CCLK pin. In Slave Serial mode, the FPGAs CCLK pin is driven by an external source. In both serial configuration modes, the MSB of each data byte is always written to the DIN pin first.

The Master Serial mode is designed so the FPGA can be configured from a Serial PROM (Figure 3). The speed of the CCLK is selectable by BitGen options, see "BitGen Switches and Options" on page 3. Be sure to select a CCLK speed supported by the SPROM.

The Slave Serial configuration mode allows for FPGAs to be configured from other logic devices, such as microprocessors, or in a daisy-chain fashion. Figure 3 shows a Master Serial FPGA configuring from an SPROM with a Slave Serial FPGA in a daisy-chain with the Master.

Daisy-Chain Configuration

Virtex FPGAs may only be daisy-chained with XC4000X, Spartan-XL, Spartan-II, or other Virtex FPGAs for configuration. There are no restrictions on the order of the chain. However, if a Virtex FPGA is placed as the Master and a non-Virtex FPGA is placed as a slave, select a configuration CCLK speed supported by all devices in the chain.

The separate bitstreams for the FPGAs in a daisy-chain are required to be combined into a single PROM file by using either the PROM File Formatter or the PROMgen utility. Separate PROM files may not be simply concatenated together to form a daisy-chain bitstream.

The first device in the chain is the first to be configured. No data is passed onto the DOUT pin until all the data frames, start-up command, and CRC check have been loaded. CRC checks only include the data for the current device, not for any others in the chain. After finishing the first stream, data for the next device is loaded. The data for the downstream device appears on DOUT typically about 40 CCLK cycles after being loaded into DIN. This is due to internal packet processing. Each daisy-chained bitstream carries its own synchronization word. Nothing of the first bitstream is passed to the next device in the chain other than the daisy-chained configuration data.

The DONE_cycle must be set before GTS and GSR, or the GTS_cycle and GSR_cycle must be set to the value DONE for the Start-up Sequence of each Virtex device not to begin until all of the DONE pins have been released. When daisy-chaining multiple Virtex devices, either set the last device in the chain to DriveDONE, or add external pull-up resistors to counteract the combined capacitive loading on DONE. If non-Virtex devices are included in the daisy-chain, it

is important to set their bitstreams to SyncToDONE with BitGen options. For more information on Virtex BitGen options, see "BitGen Switches and Options" on page 3.

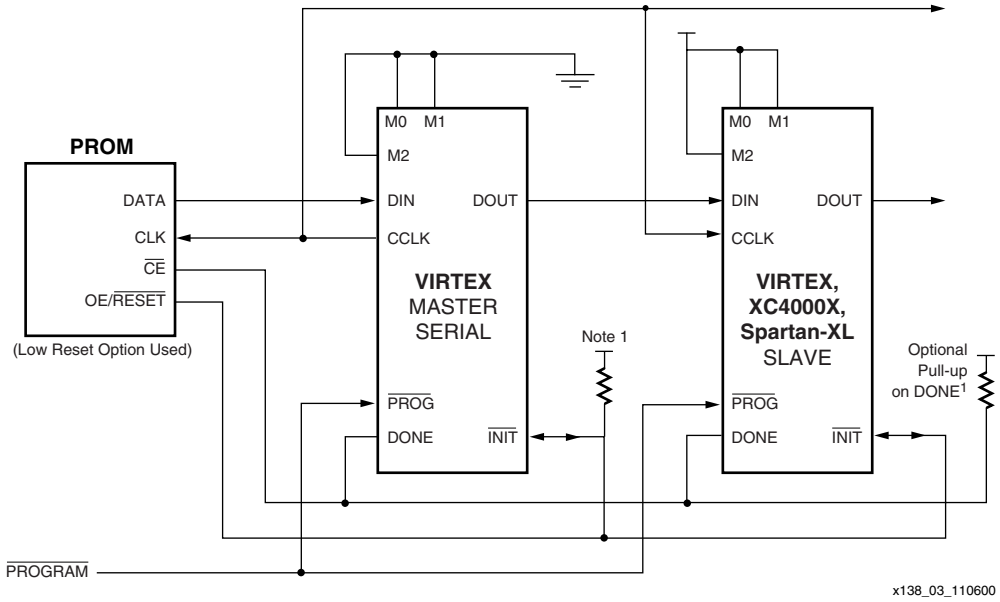

Figure 3: **Master/Slave Serial Mode Circuit Diagram**

Notes:

1. If no Virtex device is selected to DriveDONE, an external pull-up of 330Ω should be added to the common DONE line. With Spartan-XL devices a 4.7KΩ pull-up resistor should be added to the common DONE line. This pull-up is not needed if DriveDONE is selected. If used, DriveDONE should only be selected for the last device in the configuration chain.

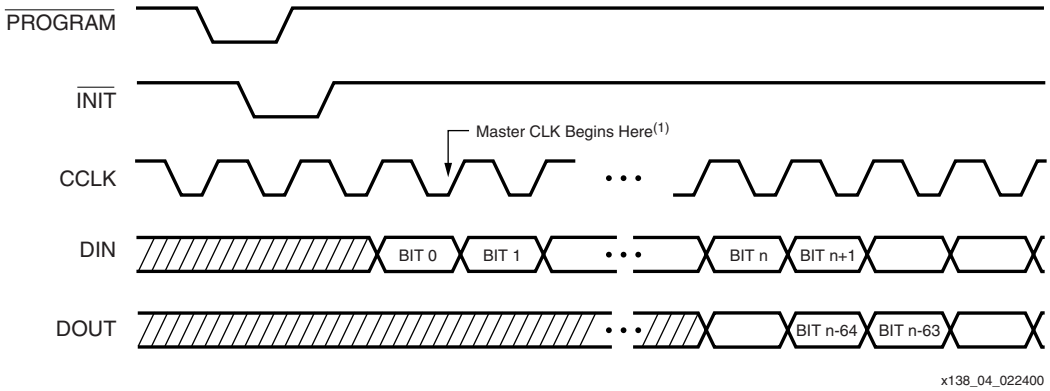

Figure 4: **Serial Configuration Clocking Sequence**

Notes:

1. For Slave configurations a free running CCLK may be used as indicated in Figure 4. For Master configurations, the CCLK does not transition until after initialization as indicated by the arrow.

SelectMAP Mode

The SelectMAP mode provides an 8-bit bidirectional data bus interface to the Virtex configuration logic that may be used for both configuration and readback. Virtex devices may not be serially daisy-chained when the SelectMAP interface is used. However, they may be connected in a parallel-chain as shown in Figure 5. The DATA pins (D0:D7), CCLK, WRITE, BUSY, PROGRAM, DONE, and INIT may be connected in common between all of the devices. CS inputs should be kept separate so each device may be accessed individually. If all devices

are to be configured with the same bitstream, readback is not being used, and CCLK is less than 50 MHz, the $\overline{CS}$ pins may be connected to a common line so the devices are configured simultaneously.

Although Figure 5 does not show a control module for the SelectMAP interface, the SelectMAP interface is typically driven by a processor, micro controller, or some other logic device such as an FPGA or a CPLD.

DATA Pins (D[0:7])

The D0 through D7 pins function as a bidirectional data bus in the SelectMAP mode. Configuration data is written to the bus, and readback data is read from the bus. The bus direction is controlled by the $\overline{WRITE}$ signal. See "Bitstream Format" on page 15. The D0 pin is considered the MSB bit of each byte.

$\overline{WRITE}$

When asserted Low, the $\overline{WRITE}$ signal indicates that data is being written to the data bus. When asserted High, the $\overline{WRITE}$ signal indicates that data is being read from the data bus.

$\overline{CS}$

The Chip Select input ($\overline{CS}$) enables the SelectMAP data bus. To write or read data onto or from the bus, the $\overline{CS}$ signal must be asserted Low. When $\overline{CS}$ is High, Virtex devices do not drive onto or read from the bus.

BUSY

When $\overline{CS}$ is asserted, the BUSY output indicates when the FPGA can accept another byte. If BUSY is Low, the FPGA reads the data bus on the next rising CCLK edge where both $\overline{CS}$ and $\overline{WRITE}$ are asserted Low. If BUSY is High, the current byte is ignored and must be reloaded on the next rising CCLK edge when BUSY is Low. When $\overline{CS}$ is not asserted, BUSY is 3-stated.

BUSY is only necessary for CCLK frequencies above 50 MHz. For frequencies at or below 50 MHz, BUSY is ignored, see "Express-Style Loading" on page 13. For parallel chains, as shown in Figure 5, where the same bitstream is to be loaded into multiple devices simultaneously, BUSY should not be used. Thus, the maximum CCLK frequency for such an application must be less than 50 MHz.

CCLK

The CCLK pin is a clock input to the SelectMAP interface that synchronizes all loading and reading of the data bus for configuration and readback. Additionally, the CCLK drives internal configuration circuitry. The CCLK may be driven either by a free running oscillator or an externally-generated signal.

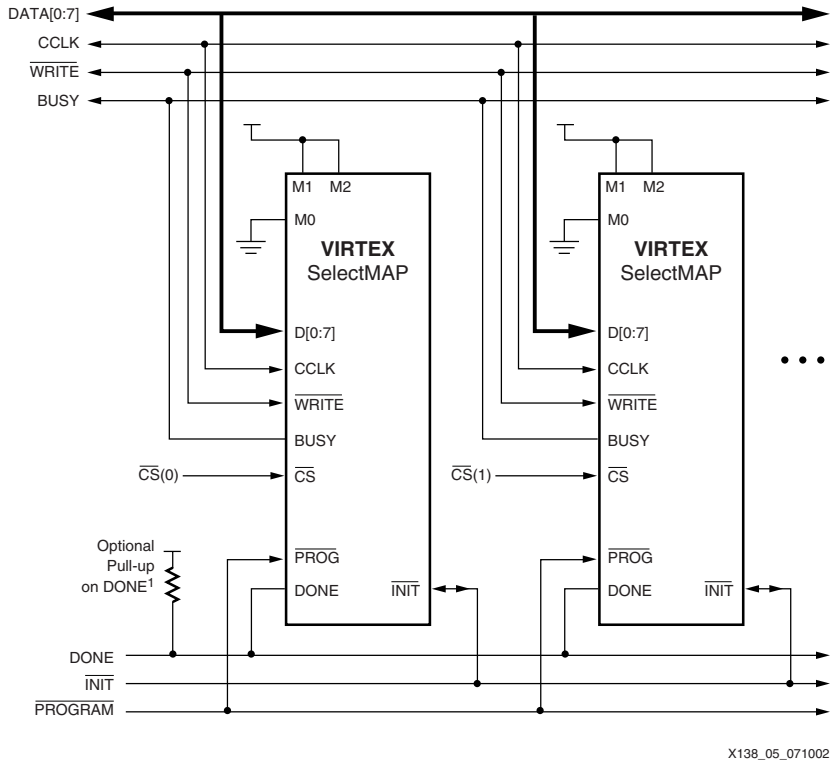

Figure 5: **SelectMAP Mode Circuit Diagram**

Notes:

1. If none of the Virtex devices have been selected to DriveDONE, add an external pull-up of 330Ω should be added to the common DONE line. This pull-up is not needed if DriveDONE is selected. If used, DriveDONE should only be selected for the last device in the configuration chain.

Free-Running CCLK

A free-running oscillator may be used to drive Virtex CCLK pins. For applications that can provide a continuous stream of configuration data, refer to the timing diagram discussed in **Express-Style Loading**, page 13. For applications that cannot provide a continuous data stream, missing the clock edges, refer to the timing diagram discussed in **Non-contiguous Data Strobe**, page 14. An alternative to a free-running CCLK is discussed in **Controlled CCLK**, page 15.

Express-Style Loading

In express-style loading, a data byte is loaded on every rising CCLK edge as shown in Figure 6. If the CCLK frequency is less than 50 MHz, this can be done without handshaking. For frequencies above 50 MHz, the BUSY signal must be monitored. If BUSY is High, the current byte must be reloaded when BUSY is Low.

The first byte may be loaded on the first rising CCLK edge that $\overline{\text{INIT}}$ is High, and when both $\overline{\text{CS}}$ and $\overline{\text{WRITE}}$ are asserted Low. $\overline{\text{CS}}$ and $\overline{\text{WRITE}}$ may be asserted anytime before or after $\overline{\text{INIT}}$ has gone High. However, the SelectMAP interface is not active until after $\overline{\text{INIT}}$ has gone High. The order of $\overline{\text{CS}}$ and $\overline{\text{WRITE}}$ does not matter, but $\overline{\text{WRITE}}$ must be asserted throughout configuration. If $\overline{\text{WRITE}}$ is de-asserted before all data has been loaded, the FPGA aborts the operation. To complete configuration, the FPGA must be reset by $\overline{\text{PROGRAM}}$ and reconfigured with the entire stream. For applications that need to de-assert $\overline{\text{WRITE}}$ between bytes, see "Controlled CCLK" on page 15.

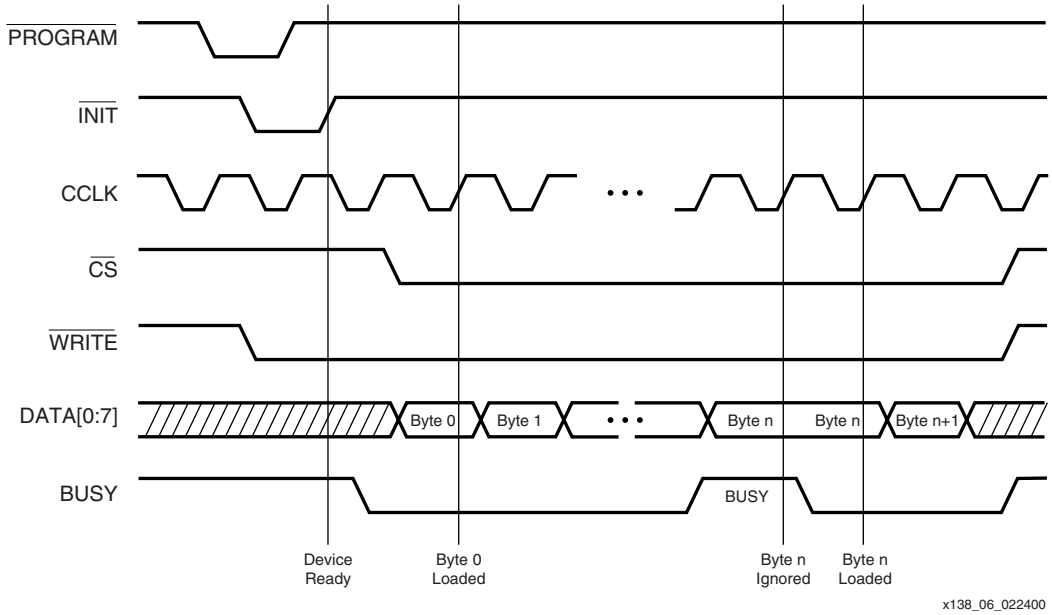

Figure 6: **"Express Style" Continuous Data Loading in SelectMAP**

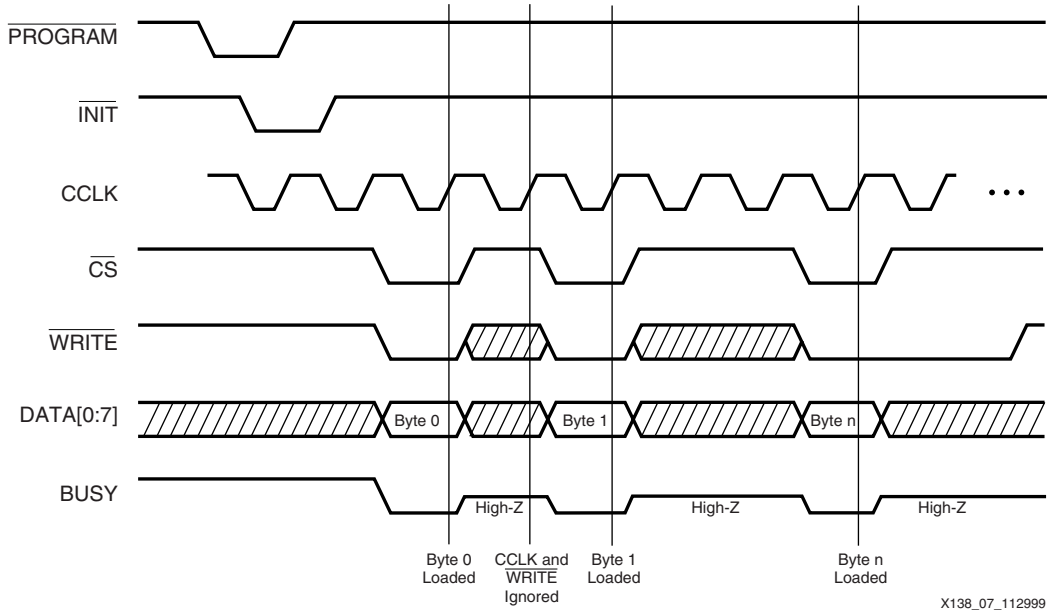

Figure 7: **Separating Data Loads by Multiple CCLK Cycles Using $\overline{CS}$**

Non-contiguous Data Strobe

In applications where multiple clock cycles may be required to access the configuration data before each byte can be loaded into the SelectMAP interface, data may not be ready for each consecutive CCLK edge. In such a case, the $\overline{CS}$ signal may be de-asserted until the next data byte is valid on the DATA[0:7] pins. This is demonstrated in Figure 7. While $\overline{CS}$ is High, the SelectMAP interface does not expect any data and ignores all CCLK transitions. However, $\overline{WRITE}$ must continue to be asserted while $\overline{CS}$ is asserted. If $\overline{WRITE}$ is High during a positive CCLK transition while $\overline{CS}$ is asserted, the FPGA aborts the operation. For applications that need to de-assert the $\overline{WRITE}$ signal without de-asserting $\overline{CS}$, see "Controlled CCLK" on page 15.

Controlled CCLK

Some applications require that $\overline{\text{WRITE}}$ be de-asserted between the loading of configuration data bytes asynchronously from the $\overline{\text{CS}}$. Typically, this would be due to the $\overline{\text{WRITE}}$ signal being a common connection to other devices on the board, such as memory storage elements. In such a case, driving CCLK as a controlled signal instead of a free-running oscillator makes this type of operation possible. In Figure 8, the CCLK, $\overline{\text{CS}}$, and $\overline{\text{WRITE}}$ are asserted Low while a data byte becomes active. Once the CCLK has gone High, the data is loaded. $\overline{\text{WRITE}}$ may be de-asserted and re-asserted as many times as necessary, just as long as it is Low before the next rising CCLK edge.

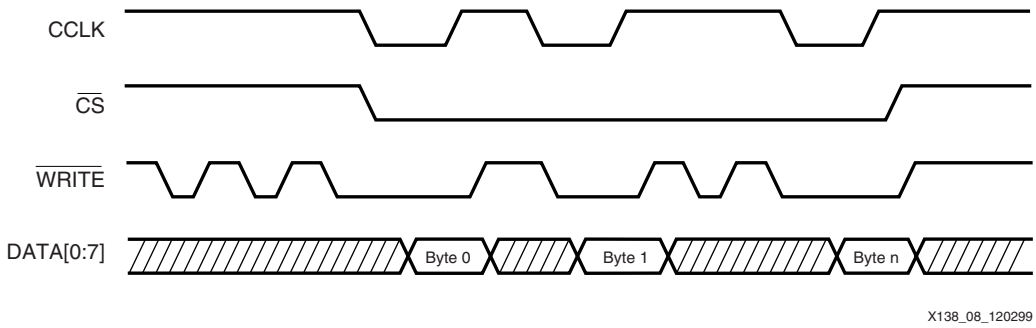

Figure 8: **Controlling CCLK for $\overline{\text{WRITE}}$ De-assertion**

Bitstream Format

The Virtex bitstream has a very different format from that of all other Xilinx FPGAs. The typical FPGA user does not need a bit-level understanding of the configuration stream. However, for the purpose of debugging, designing embedded readback operations, or otherwise complex styles of configuring multiple FPGAs, a review of the bitstream format is recommended. Therefore, this section describes the Virtex bitstream, the internal configuration logic, and the internal processing of configuration data.

Data Frames

The internal configuration memory is partitioned into segments called "Frames." The portions of the bitstream that actually get written to the configuration memory are "Data Frames." The number and size of frames varies with device size as shown in Table 4. The total number of configuration bits for a particular device is calculated by multiplying the number of frames by the number of bits per frame, and then adding the total number of bits needed to perform the *Configuration Register Writes* shown in Table 7.

Table 4: **Virtex Configuration Data Frames**

Device	Frames	Bits per Frame	Configuration Bits
XCV50	1453	384	559,200
XCV50E	1637	384	630,048
XCV100	1741	448	781,216
XCV100E	1925	448	863,840
XCV150	2029	512	1,040,096
XCV200	2317	576	1,335,840
XCV200E	2501	576	1,442,016
XCV300	2605	672	1,751,808
XCV300E	2789	672	1,875,648

Table 4: **Virtex Configuration Data Frames *(Continued)***

Device	Frames	Bits per Frame	Configuration Bits
XCV400	3181	800	2,546,048
XCV400E	3365	800	2,693,440
XCV405E	4285	800	3,430,400
XCV600	3757	960	3,601,968
XCV600E	4125	960	3,961,632
XCV800	4333	1088	4,715,552
XCV812E	5989	1088	6,519,648
XCV1000	4909	1248	6,127,744
XCV1000E	5277	1248	6,587,520
XCV1600E	6037	1376	8,308,992
XCV2000E	6613	1536	10,159,648
XCV2600E	7477	1728	12,927,336
XCV3200E	8341	1952	16,283,712

Configuration Registers

Table 5: **Internal Configuration Registers**

Symbol	Register Name	Address
CMD	Command	0100b
FLR	Frame Length	1011b
COR	Configuration Option	1001b
MASK	Control Mask	0110b
CTL	Control	0101b
FAR	Frame Address	0001b
FDRI	Frame Data Input	0010b
CRC	Cyclic Redundancy Check	0000b
FDRO	Frame Data Output	0011b
LOUT	Daisy-chain Data Output (DOUT)	1000b

The Virtex configuration logic was designed so that an external source may have complete control over all configuration functions by accessing and loading addressed internal configuration registers over a common configuration bus. The internal configuration registers that are used for configuration and readback are listed in Table 5. All configuration data, except the synchronization word and dummy words, is written to internal configuration registers.

Command Register (CMD)

The CMD is used to execute the commands shown in Table 6. These commands are executed by loading the binary code into the CMD register.

Table 6: **CMD Register Commands**

Symbol	Command	Binary Code
RCRC	Reset CRC Register	0111b
SWITCH	Change CCLK Frequency	1001b
WCFG	Write Configuration Data	0001b
RCFG	Read Configuration Data	0100b
LFRM	Last Frame Write	0011b
START	Begin Start-Up Sequence	0101b

Frame Length Register (FLR)

The FLR is used to indicate the frame size to the internal configuration logic. This allows the internal configuration logic to be identical for all Virtex devices. The value loaded into this register is the number of actual configuration bits that get loaded into the configuration memory frames. The actual frame sizes in the bitstream, shown in Table 4, are slightly longer than the FLR value to account for internal pipelining and rounding up to the nearest 32-bit word.

Configuration Option Register (COR)

The COR is loaded with the user selected options from bitstream generation. See **BitGen Switches and Options**, page 3.

Control Register (CTL)

The CTL controls internal functions such as *Security* and *Port Persistence*.

Mask Register (MASK)

The MASK is a safety mechanism that controls which bits of the CTL register can be reloaded. Prior to loading new data into the CTL register, each bit must be independently enabled by its corresponding bit in the MASK register. Any CTL bit not selected by the MASK register is ignored when reloading the CTL register.

Frame Address Register (FAR)

The FAR sets the starting frame address for the next configuration data input write cycle.

Frame Data Register Input (FDRI)

The FDRI is a pipeline input stage for configuration data frames to be stored in the configuration memory. Starting with the frame address specified in the FAR, the FDRI writes its contents to the configuration memory frames. The FDRI automatically increments the frame address after writing each frame for the number of frames specified in the FDRI write command. This is detailed in the next section.

CRC Register (CRC)

The CRC is loaded with a CRC value that is embedded in the bitstream and compared against an internally calculated CRC value. Resetting the CRC register and circuitry is controlled by the CMD register.

Frame Data Register Output (FDRO)

The FDRO is an outgoing pipeline stage for reading frame data from the configuration memory during readback. This works the same as the FDRI but with the data flowing in the other direction.

Legacy Data Output Register (LOUT)

The pipeline data to be sent out the DOUT pin for serially daisy-chained configuration data output.

Configuration Data Processing Flow

The complete (standard) reconfiguration of a Virtex device internally follows the flow chart shown in Figure 9. All the associated commands to perform configuration are listed in Table 7.

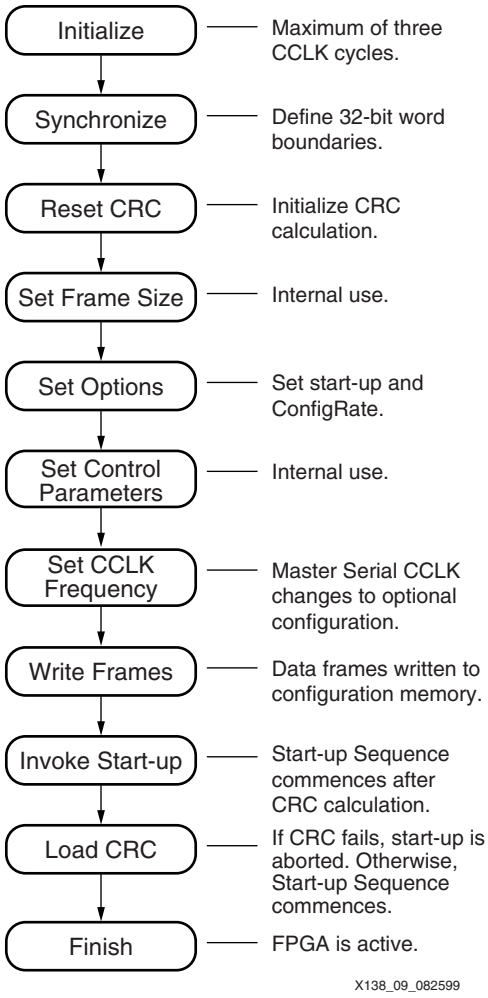

Figure 9: **Internal Configuration Processing Flow**

Table 7: **Configuration Register Writes for a Virtex Device**

Type	32-bit Words
Command Set 1	
Dummy words	(See Note)
Synchronization word	1
Write CMD (RCRC)	2
Write FLR	2
Write COR	2
Write MASK	2

Table 7: **Configuration Register Writes for a Virtex Device *(Continued)***

Type	32-bit Words
Write CMD (SWITCH)	2
Command Set 2	
Write FAR	2
Write CMD (WCFG)	2
Write FDRI	2
Write FAR	2
Write FDRI	1
Write FAR	2
Write FDRI	1
Write CRC	2
Write CMD (LFRM)	2
Write FDRI	1
Command Set 3	
Write CMD (START)	2
Write CTL	2
Write CRC	2
Dummy words	4
TOTAL	39

Notes:

1. One dummy word. Future versions of BitGen may insert a different number of dummy words at the beginning of the stream.

The first command set prepares the internal configuration logic for the loading of the data frames. The internal configuration logic is first initialized with several CCLK cycles represented by dummy word(s), and then synchronized to recognize the 32-bit word boundaries by the synchronization word. The CRC register and circuitry must then be reset by writing the RCRC command to the CMD register. The frame length size for the device being configured is then loaded into the FLR register. The configuration options are loaded into the COR followed by the control mask. The CCLK frequency selected is specified in the COR; however, to switch to that frequency the SWITCH command must be loaded into the CMD register. Now the data frames can be loaded.

The second command set loads the configuration data frames. First, a WCFG (Write Configuration) command is loaded into the CMD register activating the circuitry that writes the data loaded into the FDRI into the configuration memory cells. To load a set of data frames, the starting address for the first frame is first loaded to the FAR, followed by a write command, and then by the data frames to the FDRI. The FDRI write command also specifies the amount of data that is to follow in terms of the number of 32-bit words that comprise the data frames being written. Typically, three large sets of frames are loaded by this process. When all but the last frame has been loaded, an initial CRC checksum is loaded into the CRC register. The Last Frame command (LFRM) is loaded into the CMD register followed by a final FDRI write command and the last data frame into the FDRI register.

The third command set initializes the Start-up Sequence and finishes CRC checking. After all the data frames have been loaded, the START command is loaded into the CMD register,

followed by any internal control data to the CTL and by the final CRC value into the CRC register. The four dummy words at the end are flushed through the system to provide the finishing CCLK cycles to activate the FPGA.

The Standard Bitstream

Virtex devices have the ability to be only partially re-configured or read back; however, this topic is beyond the scope of this note. For more information on partial configuration, refer to application note **XAPP151 "Virtex Configuration Architecture User Guide"**. The standard bitstream, currently generated by BitGen, follows the format shown in Table 8, Table 9, and Table 10. **This format assumes D0 is considered the MSB**. It is divided into three tables to follow the three command sets described in the previous subsection.

Table 8 shows the first set of commands in the bitstream that prepare the configuration logic for rewriting the memory frames. All commands are described as 32-bit words, since configuration data is internally processed from a common 32-bit bus.

From Table 8, the first dummy word pads the front of the bitstream to provide the clock cycles necessary for initialization of the configuration logic. No actual processing takes place until the synchronization word is loaded. Since the Virtex configuration logic processes data as 32-bit words, but may be configured from a serial or 8-bit source, the synchronization word is used to define the 32-bit word boundaries. That is, the first bit after the synchronization word is the first bit of the next 32-bit word, and so on.

Table 8: **Bitstream Header and Configuration Options**

Data Type	Data Field
Dummy word	FFFF FFFFh
Synchronization word	AA99 5566h
Packet Header: Write to CMD register	3000 8001h
Packet Data: RCRC	0000 0007h
Packet Header: Write to FLR register	3001 6001h
Packet Data: Frame Length	0000 00--h
Packet Header: Write to COR	3001 2001h
Packet Data: Configuration options	---- ----h
Packet Header: Write to MASK	3000 C001h
Packet Data: CTL mask	0000 0000h
Packet Header: Write to CMD register	3000 8001h
Packet Data: SWITCH	0000 0009h
Packet Header: Write to FAR register	3000 2001h
Packet Data: Frame address	0000 0000h
Packet Header: Write to CMD register	3000 8001h
Packet Data: WCFG	0000 0001h

After synchronization, all data (register writes and frame data) are encapsulated in *packets.* There are two kinds of packets: Type 1 and Type 2. Type 1 packets are used for register writes. A combination of Type 1 and Type 2 packets are used for frame data writes. A packet contains two different sections: Header and Data. A Type 1 Packet Header, shown in Figure 10, is always a single 32-bit word that describes the packet type, whether it is a read/write function or a specific configuration register address (see Table 5) as the destination, and how many

32-bit words are in the following Packet Data portion. A Type 1 Packet Data portion may contain anywhere from 0 to 2,047 32-bit data words.

The first packet in Table 8 is a Type 1 packet header that specifies writing one data word to the CMD register. The following packet data is a data word specifying a reset of the CRC register (compare the data field of Table 8 to the binary codes of Table 6).

The second packet in Table 8 loads the frame size into the FLR. The value is the frame size from Table 4, divided by 32, minus 1, and converted to Hex (e.g., the FLR for a V300 is 14h).

The third packet loads the configuration options into the COR register. The binary description of this register is not documented in this application note. Following this is a similar write of the SWITCH command to the CMD register which selects the CCLK frequency specified in the COR. Next, the starting frame address is loaded into the FAR. Finally, the WCFG command is loaded into the CMD register so the loading of frame data may commence.

Table 9 shows the packets that load all the data frames starting with a Type 1 packet to load the starting frame address, which is always 0h.

Packet Header	Type	Operation (Write/Read)	Register Address (Destination)	Byte Address	Word Count (32-bit Words)
Bits[31:0]	31:29	28:27	26:13	12:11	10:0
Type 1	001	10/01	XXXXXXXXXXXXX	XX	XXXXXXXXXXX

Figure 10: **Type 1 Packet Header**

Packet Header	Type	Operation (Write/Read)	Word Count (32-bit Words)
Bits[31:0]	31:29	28:27	26:0
Type 2	010	10/01	XXXXXXXXXXXXXXXXXXXXXXXXXXX

Figure 11: **Type 2 Packet Header**

Table 9: **Bitstream Data Frames and CRC**

Data Type	Data Field
Packet Header: Write to FDRI	3000 4000h
Packet Header Type 2: Data words	5--- ----h
Packet Data: Configuration data frames in 32-bit words. Total number of words specified in Type 2 Packet Header	---- ----h ....
Packet Header: Write to FAR register	3000 2001h
Packet Data: Next frame address	---- ----h
Packet Header: Write to FDRI	3000 4---h
Packet Data: Configuration data frames in 32-bit words. Total number of words specified in Packet Header	---- ----h ....
Packet Header: Write to FAR register	3000 2001h
Packet Data: Next frame address	---- ----h
Packet Header: Write to FDRI	3000 4---h
Packet Data: Configuration data frames in 32-bit words. Total number of words specified in packet header	---- ----h ....
Packet Header: Write to CRC	3000 0001h
Packet Data: CRC value	---- ----h
Packet Header: Write to CMD register	3000 8001h

Table 9: **Bitstream Data Frames and CRC *(Continued)***

Data Type	Data Field
Packet Data: LFRM	0000 0003h
Packet Header: Write to FDRI	3000 4---h
Packet Data Last: Configuration Data Frame in 32-bit words. Total number of words specified in packet header	---- ----h ....

The loading of data frames requires a combination of Type 1 and Type 2 packets. Type 2 packets must always be preceded by a Type 1 packet that contains no packet data. A Type 2 packet also contains both a header and a data portion, but the Type 2 packet data can be up to 1,048,575 data words in size.

The Type 2 packet header, shown in Figure 11, differs slightly from a Type 1 packet header in that there is no Register Address or Byte Address fields.

To write a set of data frames to the configuration memory, after the starting frame address has been loaded into the FAR (Table 8), a Type 1 packet header issues a write command to the FDRI, followed by a Type 2 packet header specifying the number of data words to be loaded, and then followed by the actual frame data as Type 2 packet data. Writing data frames may require a Type 1/Type 2 packet combination, or a Type 1 only. This depends on the amount of data being written.

This series of FAR and FDRI writes is executed three times to load all but the last data frame. Before the last data frame is loaded, a CRC check is made. To load the last frame, a LFRM command is written to the CMD register followed by a Type 1/Type 2 packet combination to the FDRI, just as before, except that there is no FAR specified. The FAR is not needed when writing the last data frame.

Table 10 shows the packets needed to issue the start-up operations and load the final CRC check. The FPGA does not go active until after the final CRC is loaded. The number of clock cycles required to complete the start-up depends on the BitGen options. Completion of the configuration process requires eight to 16 clock cycles after the final CRC is loaded. Typically, DONE is released within the first seven CCLK cycles after the final CRC value is loaded but, the rest of the dummy data at the end of the stream should continue to be loaded. The FPGA needs the additional clock cycles to finish internal processing, but this is not a concern when a free-running oscillator is used for CCLK. In serial mode this requires only 16 bits (two bytes), but in SelectMAP mode, this requires 16 bytes of dummy words at the end of the bitstream. Since the intended configuration mode to be used is unknown by Bitgen, four 32-bit dummy words (16 bytes) are always placed at the end of the bitstream.

Table 10: **Bitstream Final CRC and Start-up**

Data Type	Data Field
Packet Header: Write to CMD register	3000 8001h
Packet Data: START	0000 0005h
Packet Header: Write to CTL	3000 A001h
Packet Data: Control commands	0000 0000h
Packet Header: Write to CRC	3000 0001h
Packet Data: CRC value	---- ----h
Dummy word	0000 0000h
Dummy word	0000 0000h
Dummy word	0000 0000h
Dummy word	0000 0000h

Cyclic Redundancy Checking Algorithm

Virtex configuration utilizes a standard 16-bit CRC checksum algorithm to verify bitstream integrity during configuration. The 16-bit CRC polynomial is shown below.

$$CRC-16 = X^{16} + X^{15} + X^{2} + 1$$

The algorithm is implemented by shifting the data stream into a 16-bit shift register, shown in Figure 12. Register Bit(0) receives an XOR of the incoming data and the output of Bit(15). Bit(2) receives an XOR of the input to Bit(0) and the output of Bit(1). Bit(15) receives an XOR of the input to Bit(0) and the output of Bit(14).

A CRC Reset resets all the CRC registers to zero. As data is shifted into the CRC circuitry, a CRC calculation accumulates in the registers. When the CRC value is loaded into the CRC calculation register, the ending CRC checksum is loaded into the CRC Register. The value loaded into the CRC Register should be zero; otherwise, the configuration failed CRC check.

Not all of the configuration stream is loaded into the CRC circuitry. Only data that is written to one of the registers shown in Table 11 is included. For each 32-bit word that is written to one of the registers (Table 11), the address code for the register and the 32-bit data word is shifted LSB first into the CRC calculation circuitry, see Figure 12. When multiple 32-bit words are written to the same register, the same address is loaded after each word. All other data in the configuration stream is ignored and does not affect the CRC checksum.

Table 11: **CRC Registers**

Symbol	Register Name	Address
CMD	Command	0100b
FLR	Frame Length	1011b
COR	Configuration Option	1001b
MASK	Control Mask	0110b
CTL	Control	0101b
FAR	Frame Address	0001b
FDRI	Frame Data Input	0010b
CRC	Cyclic Redundancy Check	0000b

This description is a model that may be used to generate an identical CRC value. The actual circuitry in the device is a slightly more complex Parallel CRC circuit that produces the same result.

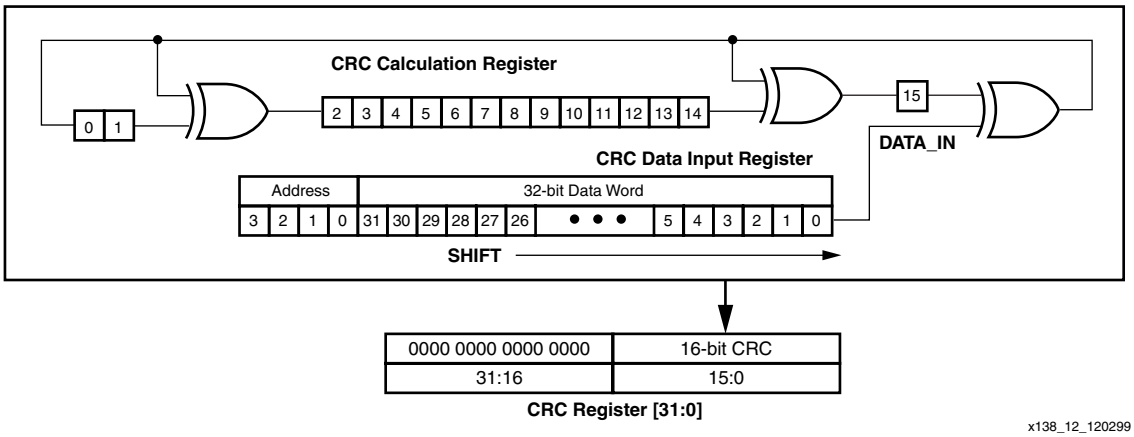

Figure 12: **Serial 16-bit CRC Circuitry**

Readback

Readback is the process of reading all the data in the internal configuration memory. This can be used to verify that the current configuration data is correct and to read the current state of all internal CLB and IOB registers as well as the current LUT RAM and block RAM values.

Readback is only available through the SelectMAP and Boundary Scan interfaces. This application note only demonstrates the use of the SelectMAP interface for performing readback. For information on using the Boundary Scan interface for readback refer to application note **XAPP139 "Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan".**

Readback Verification and Capture

Readback verification is used to verify the validity of the stored configuration data. This is most commonly used in space-based applications where exposure to radiation may alter the data stored in the configuration memory cells.

Readback capture is used to list the states of all the internal flip-flops. This can be used for hardware debugging and functional verification. When Capture is initiated, the internal register states are loaded into unused spaces in the configuration memory which may be extracted after a readback of the configuration memory.

While both *Verify* and *Capture* can be performed in one readback, each require slightly different preparation and post processing.

Preparing for Readback in Design Entry

If only a readback verification is to be performed, there are no additional steps at the time of design entry. However, if readback capture is to be used, the Virtex library primitive CAPTURE_VIRTEX must be instantiated in the user design as shown in Figure 13.

The CAPTURE_VIRTEX component is used in the FPGA design to control when the logic states of all the registers are captured into configuration memory. The CLK pin may be driven by any clock source that would synchronize Capture to the changing logic states of the registers. The CAP pin is an enable control. When CAP is asserted, the register states are captured in memory on the CLK rising edge.

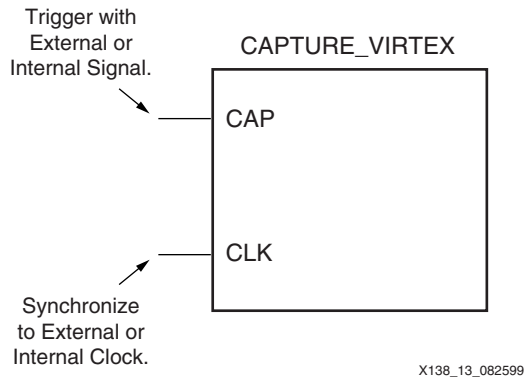

Figure 13: **Readback Capture Library Primitive**

Enabling Readback in the Software

Since readback is performed through the SelectMAP interface after configuration, the configuration ports must continue to be active by setting the persistence switch in BitGen. Additionally, a readback bit file, which contains the commands to execute a readback and a bitmap for data verification, may be optionally generated by setting the readback option in BitGen. An example of the BitGen command line is shown below:

```
bitgen -w -l -m -g readback -g persist:X8 ...
```

The **-w** overwrites existing output. The **-l** generates a *Logic Allocation* file, which is discussed in **Capturing Register States**, page 33. The **-m** generates a *Mask* file, which is discussed in

Verifying Configuration Data, page 29. The **-g readback** generates the *readback bit* file, which is discussed in **Readback Operations**, page 25, and the **-g persist:X8** keeps the SelectMAP interface active after configuration. A listing of all the associated BitGen files used for readback is shown in Table 12.

For more information about BitGen options see "BitGen Switches and Options" on page 3.

Table 12: **BitGen files used in Readback**

File Name	File Type	File Extension	File Description
Readback B	Binary	.rbb	Binary command set and verification bitmap.
Readback A	ASCII	.rba	ASCII command set and verification bitmap.
Mask	Binary	.msk	Binary command set and verification data mask.
Logic Allocation	ASCII	.ll	ASCII bit number and location of captured signal names.

Readback Operations

Readback is performed by reading a data packet from the FDRO register. The flow for this process is shown in Figure 14.

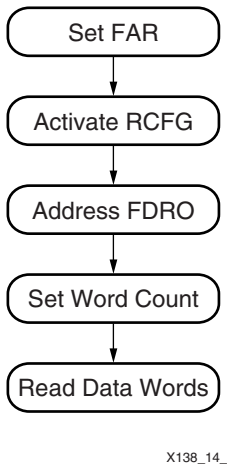

Figure 14: **CLB Readback Operation Flow**

The entire configuration memory map cannot be read in one readback sequence. Three sequences are required: one for the CLB frames and two for the block RAM frames. However, all of the configuration data frames that need to be read for verification, as well as all of the register states stored by Capture, are contained within the CLB frames. The other frame sections contain the configuration data for the columns of block RAMs. The block RAM configuration data need not be used for verification purposes, but may be used to extract the current internal states of the block RAMs just as Capture is used to extract the current internal states of registers. Therefore, a full readback and capture would require three separate readback sequences, but a simple verification requires only one (the CLB frames). This section describes the process for readback of the CLB Frames. For readback of the block RAM frames, first review this section and then refer to **Readback of Block RAM Frames**, page 35.

Table 13 shows the command set to initiate a readback of the CLB Frames. This command set is provided in the `<design>.rbb` file shown in Figure 19, `<design>.rba` and the `<design>.msk` file shown in Figure 17, page 30.

To perform the first readback sequence after configuration, it is not necessary to re-synchronize the SelectMAP interface. However, if re-synchronization is required, an ABORT process should be executed followed by loading the synchronization word. See Table 13. If a re-

synchronization is not necessary, the synchronization word may be omitted from the readback command set. If the synchronization word is reloaded, it is merely interpreted as a “No Operation” command and is ignored. The total readback command set, not including the synchronization word, is 24 bytes.

Table 13: **Readback Command Set for CLB Frames**

Data Type	Data Field
Synchronization word	AA99 5566h
Packet Header: Write to FAR register	3000 2001h
Packet Data: Starting frame address	0000 0000h
Packet Header: Write to CMD register	3000 8001h
Packet Data: RCFG	0000 0004h
Packet Header: Read from FDRO	2800 6000h
Packet Header Type 2: Data words	48-- ----h

Since all data loaded through the SelectMAP interface is processed as 32-bit words, re-synchronization is needed when either an unknown number (or a number that is not a multiple of four bytes) of data write cycles have taken place since the last command was loaded.

Once the configuration logic is synchronized, set the starting frame address in FAR as shown in Table 13. For a complete readback of the CLB frames, this is always 32 x 0h. However, this value is different for the block RAM frames. See "Readback of Block RAM Frames" on page 35.

Next, load the RCFG command into the CMD register to set the FPGA for readback. Address the FDRO register with a Type *1 read packet data header* that specifies "0" following data words. Follow that with a *Type 2 read data packet header* which specifies the number of 32-bit words to be readback. The number of data words to be readback depends on which Virtex device is being readback, shown in Table 14. Type 1 or Type 2 headers may be used depending on the amount of data that is to be readback. See "Bitstream Format" on page 15.

Table 14: **CLB Frame Word Counts per Device**

Device	TYPE 2 Packet Header for CLB Frames	CLB Frames Word Count
XCV50	4800 3E04h	15876
XCV50E	4800 408Ch	16524
XCV100	4800 581Ah	22554
XCV100E	4800 5B0Eh	23310
XCV150	4800 76B0h	30384
XCV200	4800 99C6h	39366
XCV200E	4800 9D92h	40338
XCV300	4800 CB07h	51975
XCV300E	4800 CF75h	53109
XCV400	4801 29F3h	76275
XCV400E	4801 2F39h	77625
XCV405E	4801 4997h	84375
XCV600	4801 A90Ah	108810

Table 14: **CLB Frame Word Counts per Device *(Continued)***

Device	TYPE 2 Packet Header for CLB Frames	CLB Frames Word Count
XCV600E	4801 B5B2h	112050
XCV800	4802 2E36h	142902
XCV812E	4802 6EC2h	159426
XCV1000	4802 D80Dh	186381
XCV1000E	4802 E881h	190593
XCV1600E	4803 9EAFh	237231
XCV2000E	4804 7670h	292464
XCV2600E	4805 BB7Eh	375678
XCV3200E	4807 4799h	477081

Now the readback data is ready to be clocked out. The readback sequence is shown in waveform format in Figure 15. First, assert the $\overline{CS}$ and $\overline{WRITE}$ signals and load the readback command set data described above, or from either the `<design>.rbb, .rba` or `.msk` file. See "Verifying Configuration Data" on page 29. for a detailed description of these files. Then, de-assert $\overline{WRITE}$ and $\overline{CS}$, and de-activate any external drivers on the D0 through D7 pins. To begin reading back the data, assert $\overline{CS}$ leaving $\overline{WRITE}$ High. The readback data bytes are driven out on each positive edge CCLK transition. Continue to clock for the entire readback (Word Count x 4) bytes, and then de-assert $\overline{CS}$. The process may be repeated for additional readbacks.

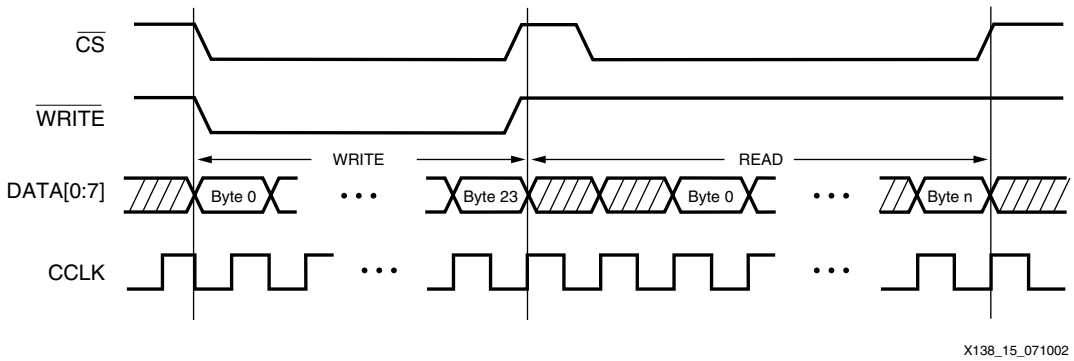

Figure 15: **Readback Clocking Sequence**

Readback Data Format

The readback data stream contains the information contained within the configuration memory map (data frames) plus additional pad data produced by the pipelining process of reading the data. The readback stream does not contain any of the commands, options, or packet information found in the configuration stream, nor does it contain any CRC values, since this information is stored in internal configuration registers, not the configuration memory. Additionally, no CRC calculation is performed during readback.

The readback stream consists of data frames each proceeded by one 32-bit word of pad data, shown in Figure 16. The first frame readback is pad data as well, and should be discarded along with the 32-bit word of pad data that proceeds it and every other frame. The size of each frame per device is shown in Table 15. The readback frame sizes are one 32-bit word smaller than the frame sizes listed for the configuration bitstream. This is because the configuration frame sizes account for the extra 32-bit pad word needed to push the configuration data through the FDRI pipeline.

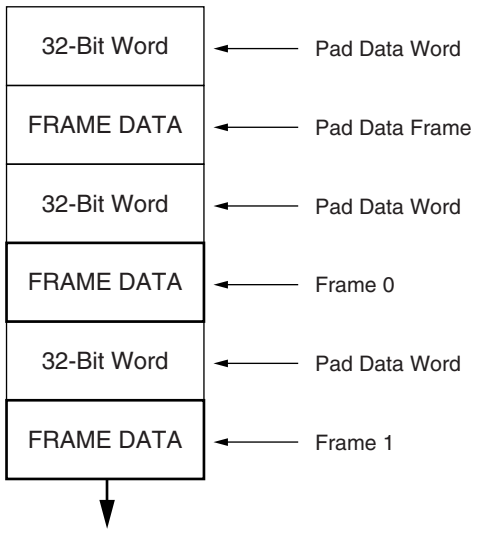

Figure 16: **Readback Data Stream**

Table 15: **Readback System Bytes for CLB Frames**

Device	CLB Frames	Bytes per Frame	Frame Bytes	Pad Bytes	Readback Bytes
XCV50	1322	44	58168	5336	63504
XCV50E	1376	44	60544	5556	66100
XCV100	1610	52	83720	6496	90216
XCV100E	1664	52	86528	6716	93244
XCV150	1898	60	113880	7656	121536
XCV200	2186	68	148648	8816	157464
XCV200E	2240	68	152320	9036	161356
XCV300	2474	80	197920	9980	207900
XCV300E	2528	80	202240	10200	212440
XCV400	3050	96	292800	12300	305100
XCV400E	3104	96	297984	12520	310504
XCV405E	3374	96	323904	13600	337504
XCV600	3626	116	420616	14624	435240
XCV600E	3734	116	433144	15060	448204
XCV800	4202	132	554664	16944	571608
XCV812E	4688	132	618816	18892	637708
XCV1000	4778	152	726256	19268	745524
XCV1000E	4886	152	742672	19704	762376
XCV1600E	5516	168	926688	22240	948928

Table 15: **Readback System Bytes for CLB Frames *(Continued)***

Device	CLB Frames	Bytes per Frame	Frame Bytes	Pad Bytes	Readback Bytes
XCV2000E	6092	188	1145296	24564	1169860
XCV2600E	6956	212	1474672	28044	1502716
XCV3200E	7820	240	1876800	31528	1908328

Verifying Configuration Data

Readback verification is a process of making a bit per bit comparison of the readback data frames to the bitmap in the `<design>.rbb` readback file. However, not all of the readback data should be used for verification. There are three types of data bits that cannot be verified against the bitmap: pad data, RAM bits, and Capture bits. The pad data is that described in the previous section, see Figure 17 and Table 15. RAM bits are configuration memory cells that hold the contents of LUT RAMs and block RAMs. These values are dynamically changing per the user design. The Capture bits are the memory locations reserved for capturing internal register states.

While the pad data words are separate data frames and must be ignored by any system performing readback, the RAM and Capture bits are sprinkled throughout the data frames and must be masked out. The `<design>.msk` mask file is used to mask out the RAM and Capture bits. An example of a mask file is shown in Figure 17.

The declarations portion is throw-away data. The first command set is for the CLB frames and includes the synchronization word which may be omitted if an Abort has not been executed. The second and third command sets are required for block RAM0 and block RAM1.

<design>.msk

```
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX FF FF FF FF AA 99 55 66 30
00 20 01 00 00 00 00 30 00 80 01 00 00 00 04 28
00 60 00 48 XX XX XX 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 -- -- -- -- -- -- -- -- -- -- -- -- -- -- 00
00 00 00 00 30 00 20 01 02 00 00 00 28 00 6X XX
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00
00 -- -- -- -- -- -- -- -- -- -- -- -- -- -- 00
00 00 00 00 30 00 20 01 02 02 00 00 28 00 6X XX
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 -- -- -- -- -- -- -- -- -- -- -- -- -- -- 00
```

File declarations and header; Design Name, target device date, etc. Size is dynamic.

Dummy/Synchronization word.

CLB Frame Readback command set. Always 32 bytes.

Masking data.

BlockRAM 0 Readback command set. Always 12 bytes.

Masking data.

BlockRAM 1 Readback command set. Always 12 bytes.

Masking data.

x138_17_72699

Figure 17: **Readback Mask File for the Virtex Family**

The masking data is used to determine which of the data frame bits are configuration bits and should be verified against the bitmap in the `<design>.rbb` readback file, and which bits are either RAM or Capture bits and thus should be skipped. The MSK file will mask out the 32 bits following each frame, but does not mask out the first 32-bit portion of the readback stream, nor the first frame pad data or following 32-bit pipeline data portion. See Figure 18. The equation for this file follows.

$$RBB[i] = MSK[i] \times DATA[i]$$

Each bit position of the masking data corresponds to the bit position of the readback data. Therefore, the first masking data bit specifies if the first bit of the first valid frame should be verified against the bitmap `<design>.rbb` file. If the mask bit is a "0b," the frame bit should be verified. If the mask bit is a "1b," the frame bit should not be verified. Since the mask file has the 32-bit pad data portions trailing the frames, at the end of each mask is a superfluous 32-bit portion which may be ignored.

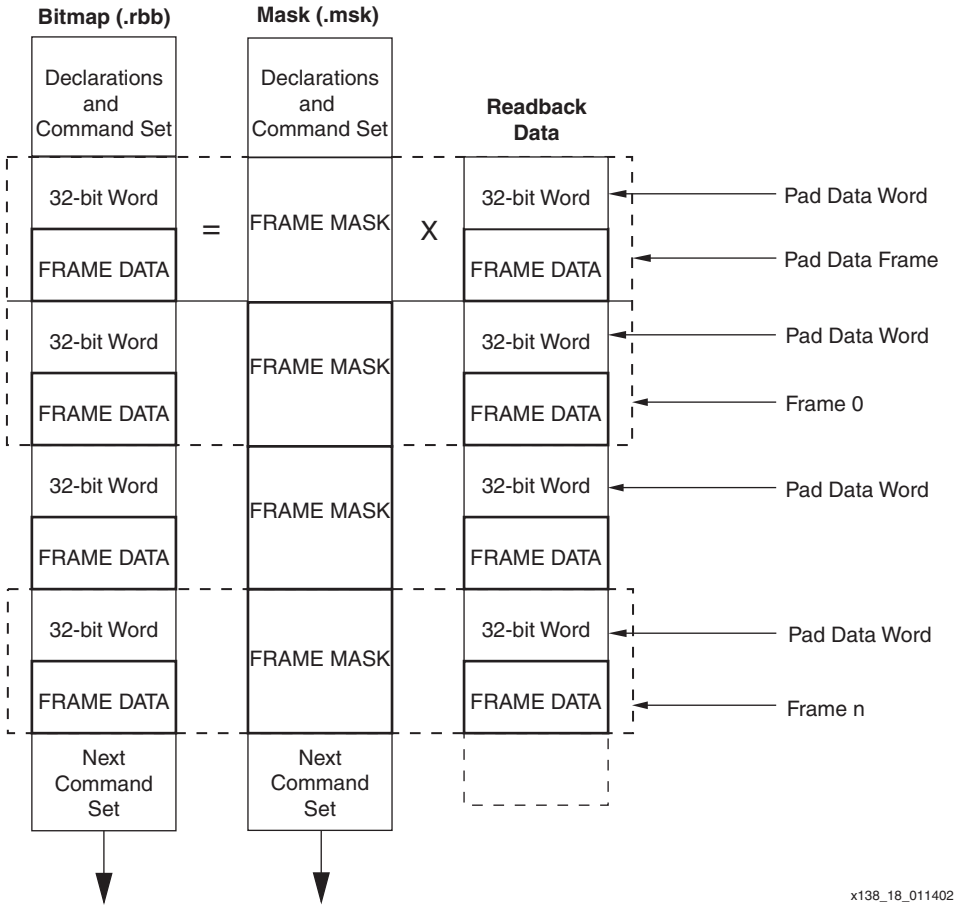

Figure 18: **Readback Data Stream Alignment**

The readback bit file `<design>.rbb` provides the configuration stream bitmap (data frames) for verifying the readback data stream. This must be used instead of the bitstream `<design>.bit` file, because, the frame data is encapsulated inside packets along with command data that is not written into configuration memory. The readback bit file is shown in Figure 19.

<design>.rbb

```
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX FF FF FF FF AA 99 55 66 30
00 20 01 00 00 00 00 30 00 80 01 00 00 00 04 28
00 60 00 48 XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX -- -- -- -- -- -- -- -- -- -- -- -- -- -- XX
XX XX XX XX 30 00 20 01 02 00 00 00 28 00 6X XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX -- -- -- -- -- -- -- -- -- -- -- -- -- -- XX
XX XX XX XX 30 00 20 01 02 02 00 00 28 00 6X XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX -- -- -- -- -- -- -- -- -- -- -- -- -- -- XX
```

File declarations and header; Design Name, target device date, etc. Size is dynamic.

Dummy/Synchronization word.

Readback command set. Always 32 bytes

Bitmap data.

BlockRAM 0 Readback command set. Always 12 bytes.

Bitmap data.

BlockRAM 1 Readback command set. Always 12 bytes.

Bitmap data.

x138_19_72699

Figure 19: **Readback Bit File for the Virtex Family**

Unlike the mask file, the bitmap does take into account that the pad data in between the data frames in the readback data stream proceed, rather than follow, each frame. The pad data portions in the readback data stream should not be verified against the bitmap. However, for every bit of pad data that is discarded from the readback data stream, a corresponding bit must be discarded from the bitmap data. A flow chart to demonstrate how a readback verification algorithm should work is provided in Figure 20.

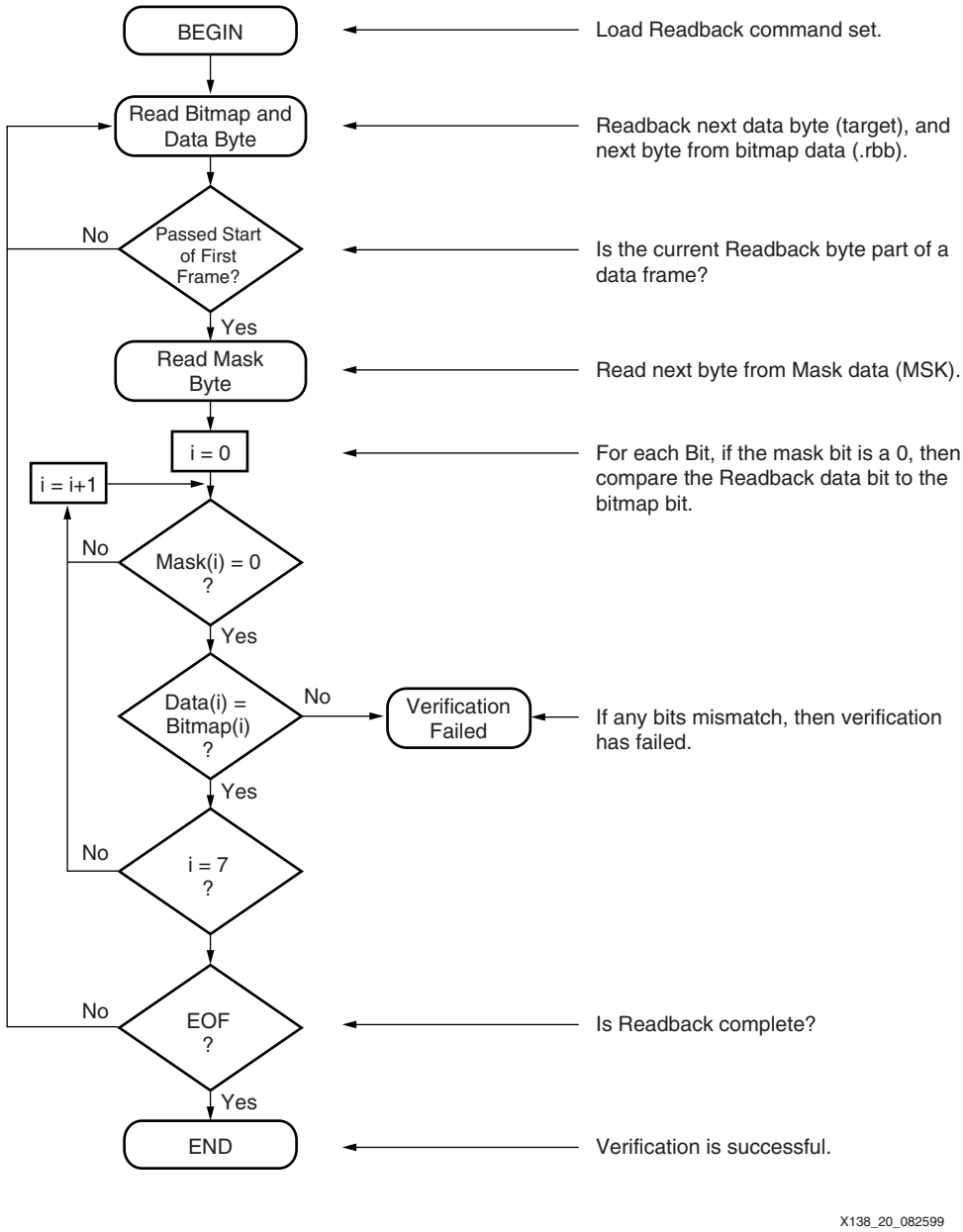

Figure 20: **Readback Verification Flow Diagram**

Capturing Register States

If Capture has been used to include the states of all internal registers in the readback data stream, the logic allocation `<design>.ll` file can be used to locate those signal state bits within the data frames. The logic allocation file includes all CLB and IOB outputs as well as all block RAM values, whether they are used in the design or not.

An example of portions of a Logic Allocation file is shown in the following text. Each Bit line includes a `<Bit offset> <Frame> <Frame offset> <CLB_location.Slice> <Type>` and a user net name if this node is used in the design.

```
;Revision 3
; Created by bitgen 2.1i at Fri. Mar 19 10:47:49 1999
; Bit lines have the following form:
```

```
; <offset> <frame number> <frame offset> <information>
;
Info Capture=Used
Info STARTSEL0=1
Info Persist=1
Bit      3274      11      35 Block=CLB_R16C13.S1 Latch=XQ
Bit      3292      11      53 Block=CLB_R15C13.S1 Latch=XQ
Bit      3310      11      71 Block=CLB_R14C13.S1 Latch=XQ
Bit      3328      11      89 Block=CLB_R13C13.S1 Latch=XQ
Bit      3346      11     107 Block=CLB_R12C13.S1 Latch=XQ
Bit      3364      11     125 Block=CLB_R11C13.S1 Latch=XQ

Bit    409801    1265     266 Block=P9 Latch=PAD
Bit    409808    1265     273 Block=P7 Latch=PAD
Bit    409818    1265     283 Block=P6 Latch=PAD

Bit    360970    1115      35 Block=CLB_R16C1.S1 Latch=XQ Net=N$8

Bit    419598    1296      19 Block=RAMB4_R3C1 Ram=B:BIT47
Bit    419599    1296      20 Block=RAMB4_R3C1 Ram=B:BIT15
Bit    419600    1296      21 Block=RAMB4_R3C1 Ram=B:BIT14
```

The bit offsets describe the position of a junk bit in the configuration stream `<design>.bit`, and is not useful for this purpose. To calculate which bit in the readback stream correlates to a desired node or signal from the LL file, the frame-number, frame-offset, and frame-length must be used in the equation below.

$$\textit{Readback Bit Number} = (\textit{FrameNumber} \times \textit{FrameLength}) + \textit{Bitmap Length} - \textit{FrameOffset} + 32$$

The Frame Number and Frame Offset are given in the .11 file. The Frame Length is shown on Table 4, page 15.

The readback bit number, calculated above, is relevant to the entire readback stream. That is, all readback data, including the pad frame and pad data words, should be counted to find the state bit represented by the readback bit number. Table 16 shows the different Bitmap Lengths for each Virtex device.

Table 16: **Bitmap Length by Virtex Device**

Device	Bitmap Length
XCV50	324
XCV50E	324
XCV100	396
XCV100E	396
XCV150	468
XCV200	540
XCV200E	540
XCV300	612
XCV300E	612
XCV400	756
XCV400E	756
XCV405E	756
XCV600	900
XCV600E	900

Table 16: **Bitmap Length by Virtex Device *(Continued)***

Device	Bitmap Length
XCV800	1044
XCV812E	1044
XCV1000	1188
XCV1000E	1188
XCV1600E	1332
XCV2000E	1476
XCV2600E	1692
XCV3200E	1908

The frame order for the block RAMs assumes that all the bits from a complete readback of all the CLB frames have been counted, and that the count continues on with the readback of the block RAM frames starting from the lowest block number. Therefore, when readback of block RAMs is used, the data frame bit count must be offset by the number of bits in all the CLB frames. This offset may be obtained from the Frame Bytes number in Table 15 and multiplying by eight.

Readback of Block RAM Frames

A readback of the block RAM frames may follow the same procedure as that for the CLB frames. However, when a readback of the block RAM is initiated, control of the block RAM is taken from the user logic so the block RAM elements may be accessed by the configuration circuitry. In order to make this hand off smooth and glitch free, it is recommended that a shutdown be performed prior to readback, and then a start-up again after readback. After a shutdown sequence has been performed, all user logic and I/O is disabled until a start-up sequence is performed. This is the same start-up sequence used in configuration. See "Start-up Sequence" on page 5. The start-up sequencer is used for both start-up and shut-down.

In applications where it is preferable not to shut-down the device, any user logic designed to drive the block RAM should also be designed to halt any write operations just prior to and during the block RAM readback session.

The flow for a block RAM readback is shown in Figure 21 with the shut-down and start-up sequences shown in the grey areas of Figure 21. RAM readback follows the same process as for CLB frames, but with a different FAR value. See "Readback Operations" on page 25. However, the synchronization step may be omitted if a previous readback sequence has already synchronized the SelectMAP interface.

The shut-down sequence is enabled by setting bit 15 of the COR. The preferred method of setting this value is to read the current value of COR, toggle bit 15 to a logic “1,” and then load the new 32-bit value back into the COR.

The full command set is shown in Table 18. After loading the COR, the START command, followed by the RCRC command must be written to the CMD register. For the Shut-down Sequence to commence, the device needs to be clocked eight times. This also flushes the data pipeline. Once the Shut-down Sequence is complete, it is safe to imitate the readback sequence of the block RAM.

To read back both columns of block RAMs, the readback sequence must be repeated for each column. The sequence is the same except for different FAR values as shown in Table 17.

Table 17: **Virtex Family Devices Starting Frame Address**

Frame Type	FAR
BlockRAM_0	0200 0000h
BlockRAM_1	0202 0000h

In Virtex-E and Virtex-EM devices the block RAM address alternates around the center to the right and left side starting with address "1" instead of the "0" in the Virtex family devices. The addressing is interweaved by starting with the block RAM to the right of the center at "1". The block to the left of the center is for address "2", and the block to the right of block "1" is "3". For a more detailed discussion, please refer to XAPP151.

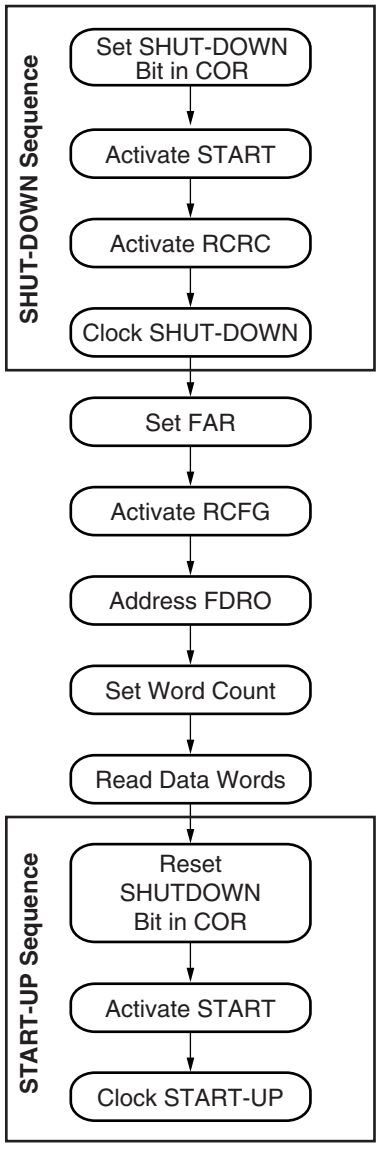

Figure 21: **Block RAM Readback Operation Flow**

Table 18: **Readback Command Set for Block RAM Frames**

Data Type	Data Field
Shut-down Sequence	
Packet Header: Read from COR	2801 2001h
Packet Data: Configuration options	---- ----h
Packet Header: Write to COR	3001 2001h
Packet Data: Set bit (15) to 1	---- ----h
Packet Header: Write to CMD register	3000 8001h
Packet Data: START	0000 0005h
Packet Header: Write to CMD register	3000 8001h
Packet Data: RCRC	0000 0007h
Dummy word	FFFF FFFFh
Dummy word	FFFF FFFFh
Block RAM Readback Sequence	
Packet Header: Write to FAR register	3000 2001h
Packet Data: Starting frame address	0200 0000h
Packet Header: Write to CMD register	3000 8001h
Packet Data: RCFG	0000 0004h
Packet Header: Read from FDRO	2800 6000h
Packet Header Type 2: Data words	48-- ----h
Start-up Sequence	
Packet Header: Write to COR	3001 2001h
Packet Data: Set bit (15) to 1.	---- ----h
Packet Header: Write to CMD register	3000 8001h
Packet Data: START	0000 0005h
Packet Header: Write to CMD register	3000 8001h
Packet Data: RCRC	0000 0007h
Dummy word	FFFF FFFFh
Dummy word	FFFF FFFFh

The two sets of block RAM frames in every device size consist of 65 frames in each column; however, the frame sizes vary per device. The word count for block RAM readback and the associated code for each device are shown in Table 19.

Table 19: **Word Counts per Frame Section**

Device	Type 2 Packet Header for Block RAM Frames	Block RAM Frames Word Count (each)
XCV50	4800 030Ch	780
XCV50E	4800 030Ch	780
XCV100	4800 038Eh	910

Table 19: **Word Counts per Frame Section *(Continued)***

Device	Type 2 Packet Header for Block RAM Frames	Block RAM Frames Word Count (each)
XCV100E	4800 038Eh	910
XCV150	4800 0410h	1040
XCV200	4800 0492h	1170
XCV200E	4800 0492h	1170
XCV300	4800 0555h	1365
XCV300E	4800 0555h	1365
XCV400	4800 0659h	1625
XCV400E	4800 0659h	1625
XCV405E	4800 0659h	1625
XCV600	4800 079Eh	1950
XCV600E	4800 079Eh	1950
XCV800	4800 08A2h	2210
XCV812E	4800 08A2h	2210
XCV1000	4800 09E7h	2535
XCV1000E	4800 09E7h	2535
XCV1600E	4800 0AEBh	2795
XCV2000E	4800 0C30h	3120
XCV2600E	4800 0DB6h	3510
XCV3200E	4800 0F7Dh	3965

After the completion of the readback session, a Start-up Sequence must be performed to reactivate the user logic and I/O. To enable the start-up, the shut-down bit (15) of the COR must be reset to a logic 0. Then, just as with the Shut-down Sequence, the START and RCRC commands must be loaded into the CMD register and the Sequence must be clocked eight times, at which time the device may resume normal operation.

Virtex-E Device Addendum

The configuration modes and operation of the 1.8V Virtex-E and Virtex-EM (Extended Memory) devices are similar with the 2.5V Virtex devices. The designer differences between the Virtex family and Virtex-E or Virtex-EM devices are discussed in this addendum.

Power Supplies

Virtex-E V_{CCINT}, the supply voltage for the internal logic and memory, is 1.8V instead of 2.5V for Virtex devices.

I/O Standards Supported

Virtex-E devices can be used with 20 high-performance interface standards, including LVDS and LVPECL differential signalling standards. A new LVCMOS I/O standard based on 1.8V V_{CCO} is also supported. I/O pins are 3.0V tolerant with appropriated external resistors. PCI 5.0V is not supported.

I/O Banking

In Virtex-E devices, the banking rules are different because the input buffers (with LVTTL, LVCMOS, and PCI standards) are powered by V_{CCO} instead of V_{CCINT}. For these standards, only input and output buffers that have the same V_{CCO} can be combined together in the same bank.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/21/99	1.0	Initial release
07/29/99	1.1	Overall update to application note
09/23/99	1.2	Virtex-E update
02/24/00	2.0	Revised to new template, new drawings
08/03/00	2.1	Virtex-EM update to tables 4, 7, 8, 10, 14,15,18, and 19. Revised Figures 1 & 3.
09/21/00	2.2	Update Figure 3, Table 8, and Table 9.
10/04/00	2.3	Revised Tables 2, 3, 4, 7, 8, 9 and **LCK_cycle** section. Removed Table 20.
07/25/01	2.4	Removed bidirectional arrow from $\overline{CS}$ in Figure 5.
11/05/01	2.5	Changed XCV1000 bit count in Table 4.
01/14/02	2.6	Changed Figure 18.
07/11/02	2.7	Change Requests (CRs) fixed.