Virtex Configuration Guide

XILINX®

Contents

Appendix C: Glossary

About This Manual

This manual describes the function and operation of Xilinx hardware and software involving FPGA configuration techniques. The chapters cover the following topics:

- Configuration and Readback
- Configuration and Readback Through JTAG
- Configuration From Parallel EPROMs With a CPLD
- Advanced Configuration Architecture Techniques
- Frequently Asked Questions
- Cable Hardware
- MultiLINX Cable Sets

In addition, three appendices provide reference information:

- Appendix A - Virtex Configuration Beginner's Guide
- Appendex B - BitGen and PROMGen Program Information
- Appendix C - Glossary

Before using this manual, familiarize yourself with operations that are common to all Xilinx software tools:

- Booting the system
- Selecting a tool for use
- Specifying operations
- Managing design data

These topics are covered in the *Quick Start Guide* that is shipped with the software. Other publications with related information include the *Hardware Debugger Guide* and the *JTAG Programmer Guide*.

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists resources you can access from this page. You can also directly access some of these resources using the provided URLs.

Resource	Description/URL
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://support.xilinx.com/support/searchtd.htm
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm
Data Book	Pages from *The Programmable Logic Data Book*, which describe device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://support.xilinx.com/xcell/xcell.htm
Tech Tips	Latest news, design tips, and patch information on the Xilinx design environment http://support.xilinx.com/support/techsup/journals/index.htm
Self-Support and Problem Solvers	Self-Supportability tools and Problem Solvers as follows: How to Find Answers: How to use support.xilinx.com to solve your problem Search our Knowledge Base: Fill out this keyword search form to find the answers to your questions Answer Browser: Browse throught the Answers Knowledge base by part type Configurtion Problem Solver: This problem solver will automatically fix your configuration issues Install Problem Solver: This problem solver will automatically troubleshoot software istallation issues Programmer Solutions: Device support list, software, and HW-130 information Virtex Power Estimator: Estimate Virtex power consumptions with our web form or download the Excel spreadsheet Technical Tips: This is *the* resource for hot issues and tips to get up and running quickly http://support.xilinx.com/support/troubleshoot

Typographical Conventions

The following typographical conventions are used in this manual:

- Red text indicates a cross-reference to information within this document. Click red text to open the specified cross-reference.
- Blue-underlined text indicates a link to a Web page. Click blue-underlined text to browse the specified Web site.
- `Courier font` indicates messages, prompts, and program outputs displayed by the system.
- `speed grade: -100`
- **`Courier bold`** indicates literal commands that you enter in a syntactical statement. However, braces "{ }" in Courier bold are not literal and square brackets "[ ]" in Courier bold are literal only in the case of bus specifications, such as bus [7:0].
- **`rpt_del_net=`** Courier bold also indicates menu commands:

  `File` → **`Open`**
- *Italic font* denotes the following items.
  - Variables that are substituted with user-defined values
  - **`edif2ngd`** *design_name*
  - References to other documents.
  - See the *Virtex-E Data Sheet* for more information.
  - Emphasis in text
  - If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
- Square brackets "[ ]" indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

  `edif2ngd` [*option_name*] *design_name*
- Braces "{ }" enclose a list of items from which you must choose one or more.

  `lowpwr =`{**`on`** | **`off`**}
- A vertical bar " | " separates items in a list of choices.

  `lowpwr =`{**`on`** | **`off`**}
- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT'
IOB #2: Name = CLKIN'
.
.
.
```

- A horizontal ellipsis ". . ." indicates that an item can be repeated one or more times.

  `allow block` *block_name loc1 loc2* `...` *locn;*

Chapter 1

Introduction

What Is Configuration?

Configuration is the process of programming a Field Programmable Gate Array (FPGA). Typically, configuration involves a data source, an appropriate FPGA, and control logic. For first-time Virtex users, a Virtex configuration beginner's guide is given in "Appendix A".

Devices in the Virtex™ series of FPGAs have four possible configuration modes.

- Master Serial Mode
- Slave Serial Mode
- SelectMAP™ Mode
- JTAG or Boundary Scan

Configuration Modes

The Master Serial Mode is the simplest configuration method where the FPGA loads configuration data from a Serial PROM, basically using the FPGA to provide the clock and virtually load itself. In the Master Serial Mode, the FPGA drives the configuration clock and provides all of the control logic. Data is loaded at one bit per CCLK.

The Slave Serial Mode involves the use of a transmission method and an external clock. It accommodates daisy chain configurations. In the Slave Serial Mode, an external clock, a microprocessor, another FPGA, or a download cable are required. Data is loaded to the target FPGA at one bit per CCLK.

The SelectMAP Mode provides for parallel reading and writing through byte-wide ports. As in the Slave Serial Mode, an external clock, a microprocessor, another FPGA, or a download cable are required. However, data is loaded to the target FPGA at one byte per CCLK.

The JTAG or Boundary Scan Mode is the industry standard serial programming mode. It utilizes external logic to drive boundary-scan specific pins (TCK, TMS, TDI, TDO), and a microprocessor, or a compatible download cable (JTAG or MultiLINX). Data is loaded to the target FPGA at one bit per TCK.

Xilinx Virtex series devices can be configured in all of these modes, each offering specific advantages explained later in this guide.

Configuration Process

There are four major phases to the configuration process.

- Configuration Memory Clear
- Initialization
- Loading Configuration Data
- Device Start-up

Configuration Memory Clear

In the memory clear phase, the non-configuration I/O pins are 3-stated with optional pull-up resistors depending on the mode pins. The $\overline{\text{INIT}}$ and DONE pins are driven Low by the FPGA, and the memory is cleared. The $\overline{\text{PROGRAM}}$ pin is checked after each memory pass. When $\overline{\text{PROGRAM}}$ is High, memory is cleared one final time and initialization can begin.

Initialization

For the initialization phase, the $\overline{\text{INIT}}$ pin is released, the mode pins are sampled, the appropriate pins become active, and the configuration process begins. It is possible to delay configuration by externally holding $\overline{\text{INIT}}$ Low.

Loading Configuration Data

Once configuration begins, the target FPGA starts to receive data frames. CRC is checked before and after the last data frame. If the CRC checks prove valid, the device start-up phase can begin.

Device Start-Up

Device start-up is a transition phase from the configuration mode to normal programmed device operation. Although the order of the start-up events are user programmable via software, the default sequence of events are as follows:

- DONE Pin Released
- All I/O Pins Active
- Global Write Enable Released
- Global Reset Released

Upon completion of the Start-up Sequence, the target FPGA is operational.

Virtex series and Spartan-II family have specific start-up considerations. There are four internal signals to activate and six start-up clock cycles.

- GWE (Global Write Enable)
- GSR (Global Set/Reset)
- GTS (Global 3-state)
- DONE (External Done Pin)
- Start-Up Clocks 1-6

A sequencer waits in the DONE phase until DONE goes High. It is possible to create "Sync-to-Done" behavior by setting GTS, GSR, and GWE to the same cycle as DONE.

Configuration PROMs

A popular method of programming target FPGAs directly is to program via PROMs. Xilinx has two Configuration PROM families, the XC1700 and the XC1800 supporting various serial and parallel programming modes, including JTAG. The PROMs are cascadable for storing multiple or longer bitstreams. Some of the PROMs are in-system reprogrammable. For details about these PROM families, see the **Configuration Solutions** page on the Xilinx website.

Chapter 2

Virtex FPGA Series Configuration and Readback

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 can 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 26.

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 SpartanXL. As with these other Xilinx device families, D0 is considered the MSB. For a detailed description, see "SelectMAP Mode" on page 28. Previous users of peripheral modes should find the transition to SelectMAP fairly straightforward.

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 26. 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 28. 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 cannot 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 can 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 31.

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 can 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 pins are located within banks 2 and 3. Therefore, only V_{CCO_2} and V_{CCO_3} pins need a 3.3 V 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 3.3 V, then V_{CCO_2} and V_{CCO_3} need to be switched from the 3.3 V 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 3.3 V supply after configuration as well.

For Serial Slave and SelectMAP configuration modes, V_{CCO} can be any voltage (as long as it is $\geq$ 1.8 V $\leq$ 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 $\overline{\text{INIT}}$, DOUT/BUSY, and DONE should have an added pull-up resistor or utilize the internal pull-up resistors. The dedicated CONFIG and JTAG 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 2.5 V and 3.3 V TTL levels. TDO is sourced from V_{CCO_2} and should be 1.8 V, 2.5 V, or 3.3 V 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

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

Switch	Default Setting	Optional Setting
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 40.

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 31. 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 22.

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 31. Therefore, Virtex FPGAs can 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 can 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 19.

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.

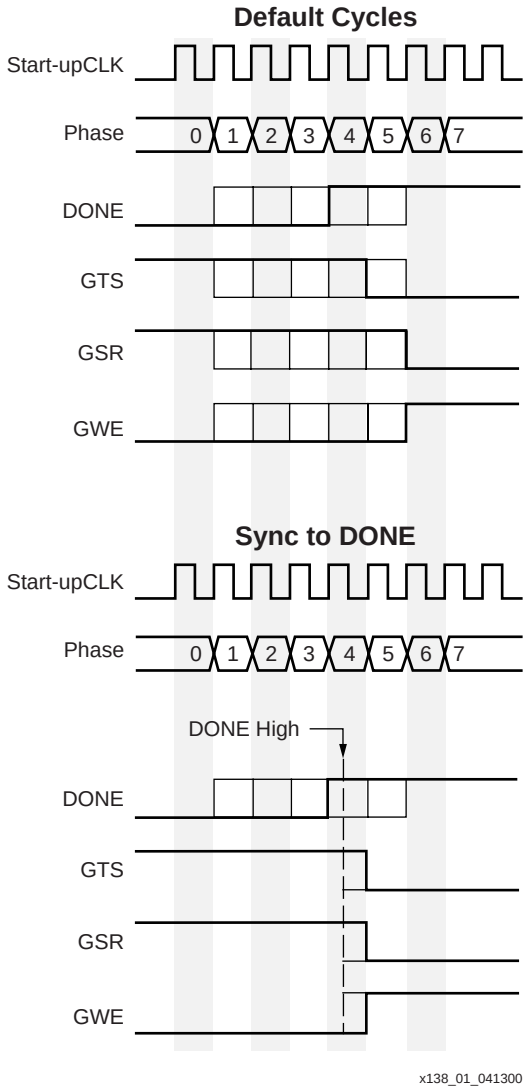

Figure 1: **Default Start-up Sequence**

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 1.0 V to 2.4 V. 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 1.0 V for power-up to continue. If this bank is not being used, a pull-up should be added to V_{CCO_2}.

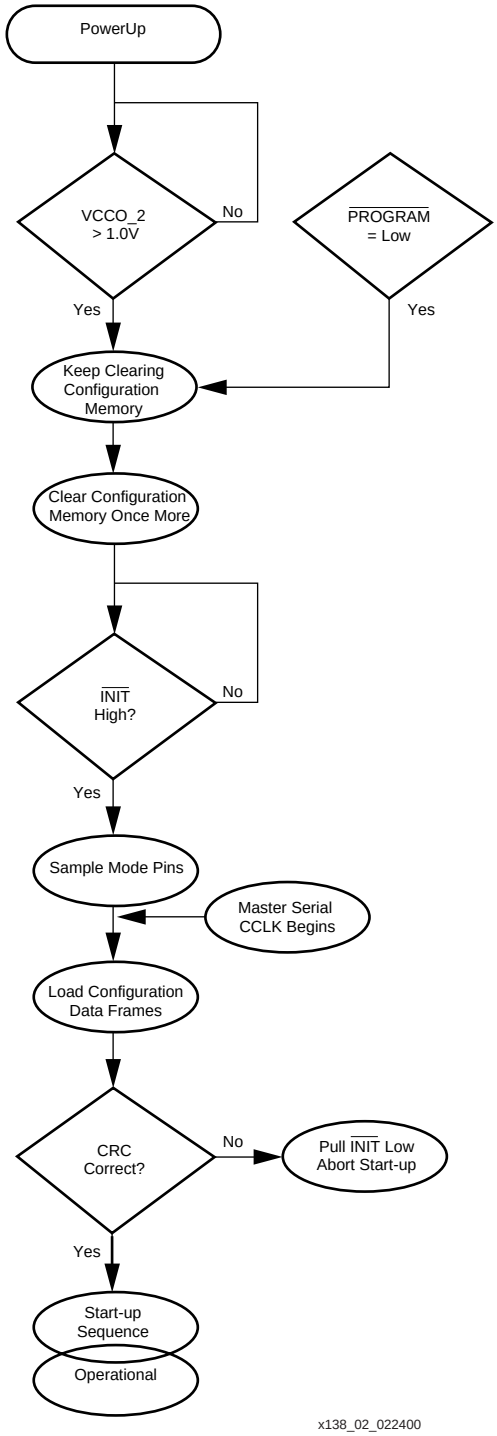

Figure 2: **Configuration Flow Diagram**

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 can 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 can 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 26 and "SelectMAP Mode" on page 28.

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 39.

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 22.

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.

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

Name	Direction	Driver Type	Description
TCK	Input		Boundary Scan clock.
TDI	Input		Boundary Scan data input.
TDO	Output	Active	Boundary Scan data output.
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 19. 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 can only be daisy-chained with XC4000X, SpartanXL, 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 can 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 19.

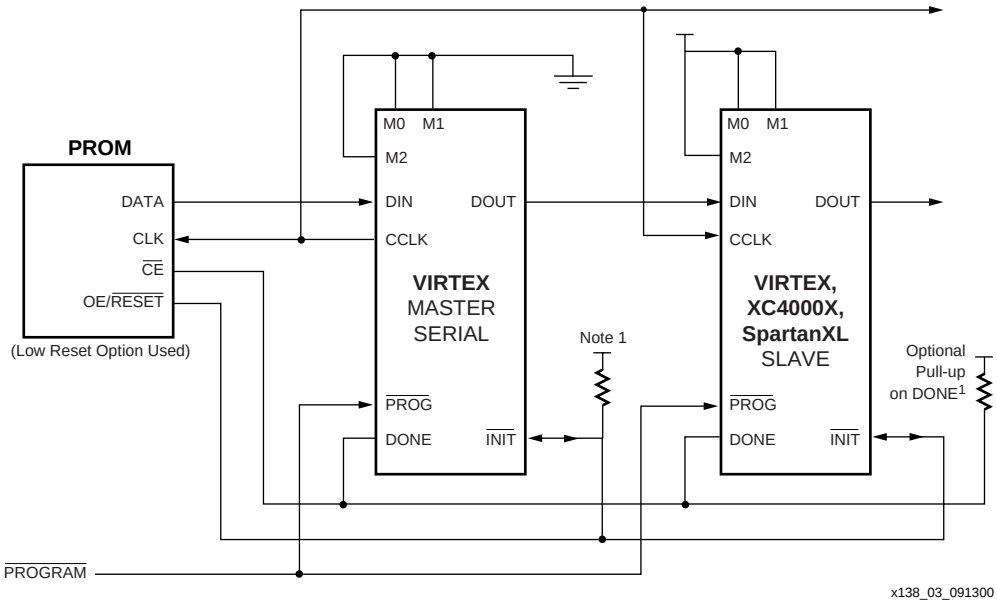

Figure 3: **Master/Slave Serial Mode Circuit Diagram**

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 SpartanXL devices a 4.7 kΩ 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.

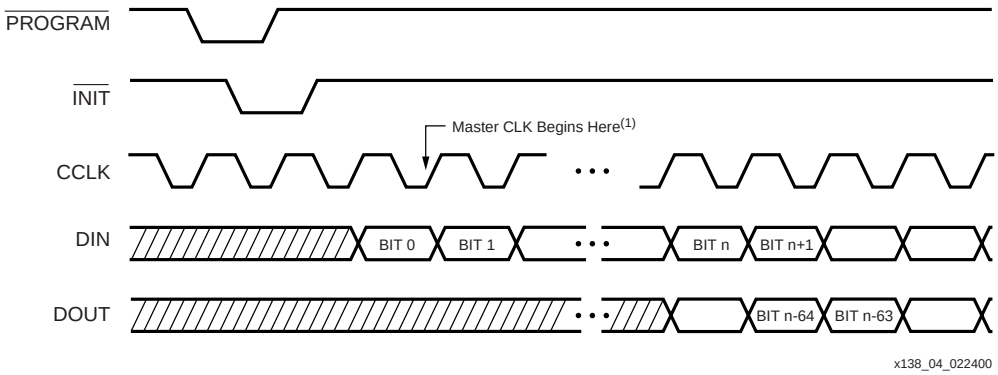

Figure 4: **Serial Configuration Clocking Sequence**

Notes:

1. For Slave configurations a free running CCLK can 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 can be used for both configuration and readback. Virtex devices cannot be serially daisy-chained when the SelectMAP interface is used. However, they can be connected in a parallel-chain as shown in Figure 5. The DATA pins (D0:D7), CCLK, $\overline{\text{WRITE}}$, BUSY, $\overline{\text{PROGRAM}}$, DONE, and $\overline{\text{INIT}}$ can be connected in common between all of the devices. $\overline{\text{CS}}$ inputs should be kept separate so each device can 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 $\overline{\text{CS}}$ pins can 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 $\overline{\text{WRITE}}$ signal. See "Bitstream Format" on page 31. The D0 pin is considered the MSB bit of each byte.

$\overline{\text{WRITE}}$

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

$\overline{\text{CS}}$

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

BUSY

When $\overline{\text{CS}}$ 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 $\overline{\text{CS}}$ and $\overline{\text{WRITE}}$ 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 $\overline{\text{CS}}$ is not asserted, BUSY is tri-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 29. 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 can be driven either by a free running oscillator or an externally-generated signal.

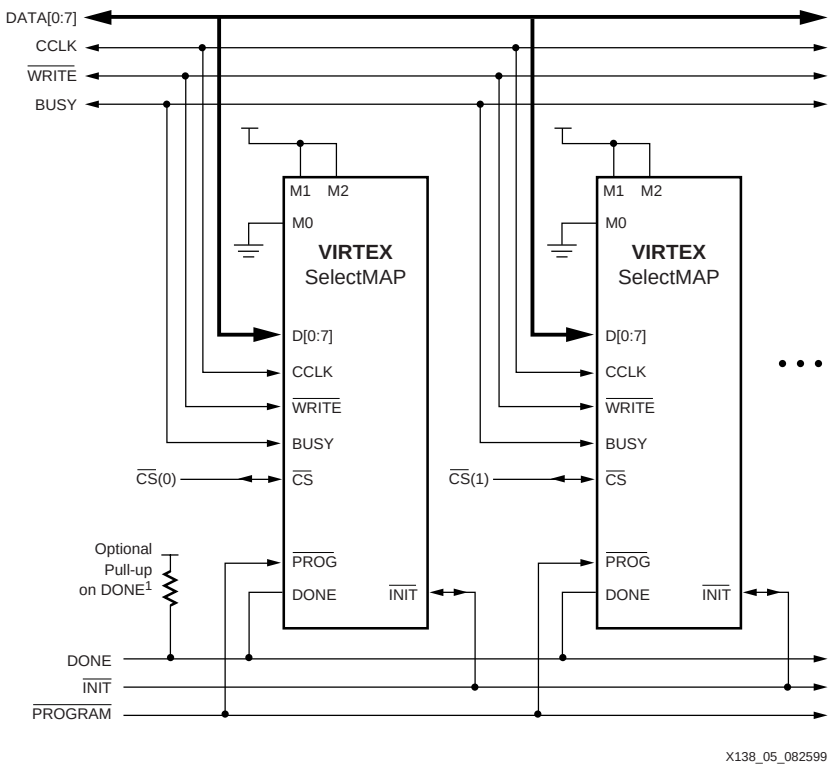

Figure 5: **SelectMAP Mode Circuit Diagram**

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 can 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" on page 29. For applications that cannot provide a continuous data stream, missing the clock edges, refer to the timing diagram discussed in "Non-Contiguous Data Strobe" on page 30. An alternative to a free-running CCLK is discussed in "Controlled CCLK" on page 31.

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 can 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 $\overline{\text{WRITE}}$ can 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 31.

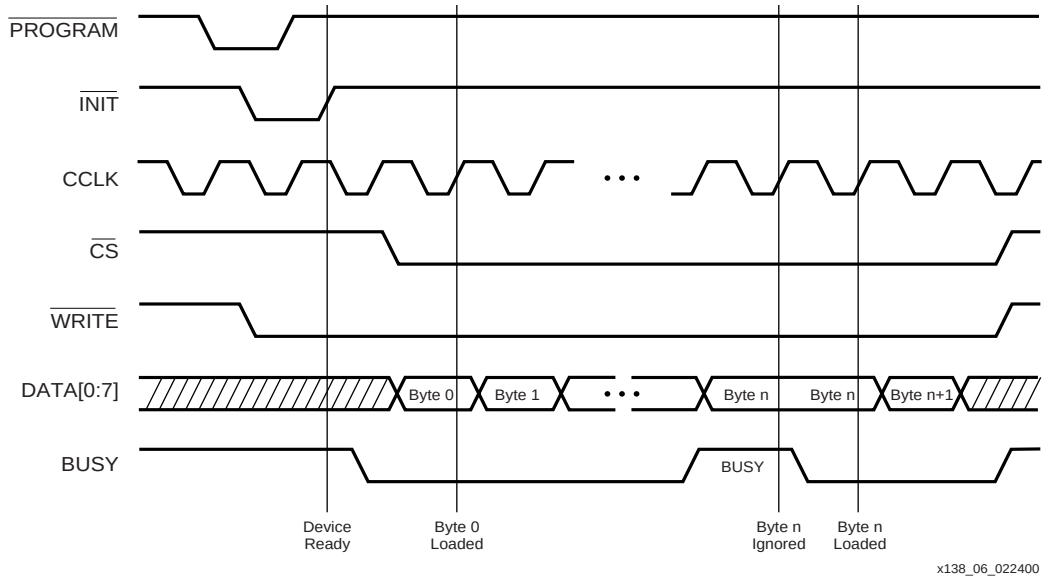

Figure 6: **"Express Style" Continuous Data Loading in SelectMAP**

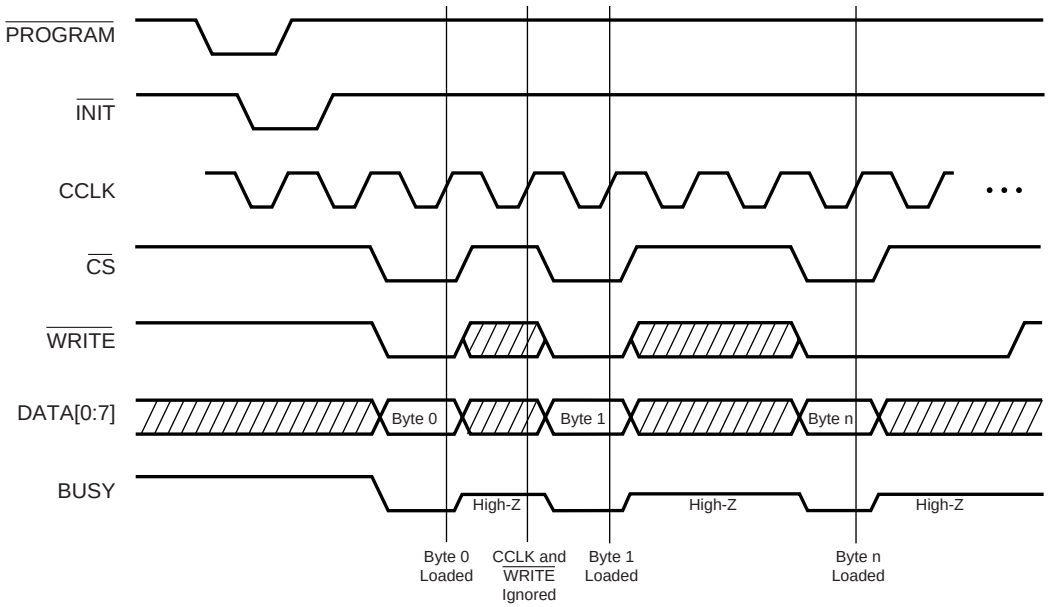

Figure 7: **Separating Data Loads by Multiple CCLK Cycles Using $\overline{CS}$**

Non-Contiguous Data Strobe

In applications where multiple clock cycles can be required to access the configuration data before each byte can be loaded into the SelectMAP interface, data cannot be ready for each consecutive CCLK edge. In such a case, the $\overline{CS}$ signal can 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 31.

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. $\overline{\text{WRITE}}$ can be de-asserted and re-asserted as many times as necessary, just as long as it is Low before the next rising CCLK edge.

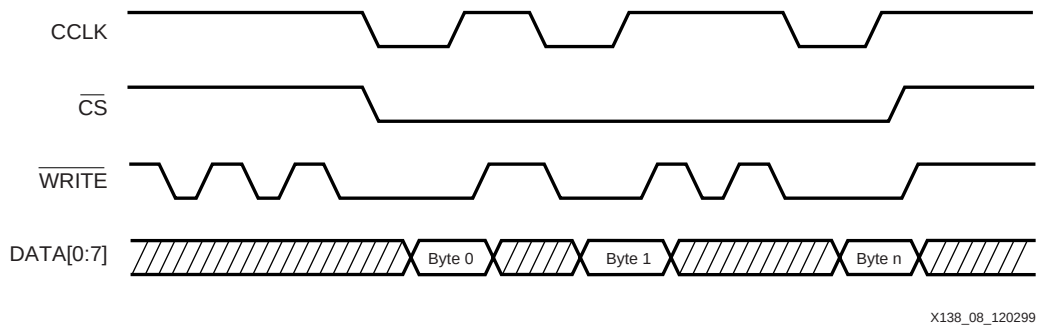

Figure 8: **Controlling CCLK for $\overline{\text{WRITE}}$ De-Assertion**

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

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

Device	Frames	Bits per Frame	Configuration Bits
XCV300	2605	672	1,751,808
XCV300E	2789	672	1,875,648
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	6,127,680
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 can 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" on page 19.

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.

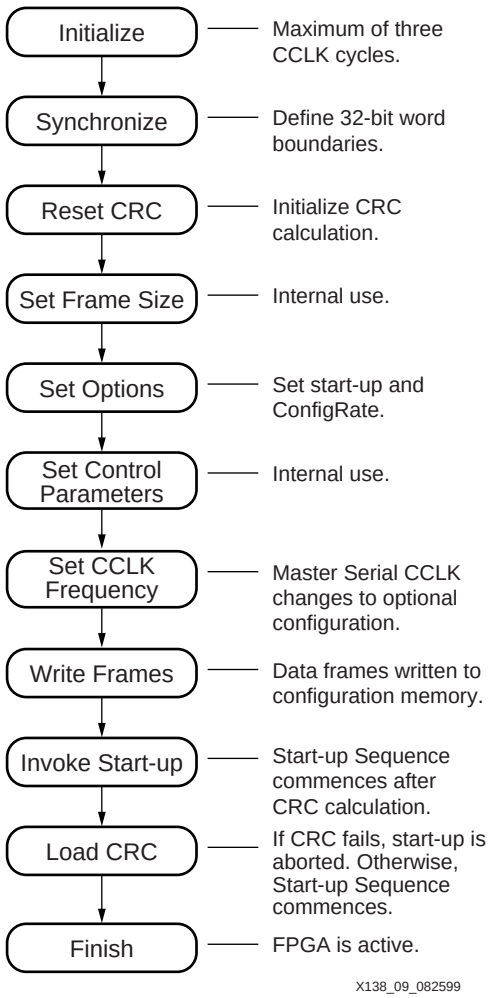

Figure 9: **Internal Configuration Processing Flow**

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
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 can 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 can 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

Table 8: **Bitstream Header and Configuration Options *(Continued)***

Data Type	Data Field
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 can 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 can 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	XXXXXXXXXX

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	XXXXXXXXXXXXXXXXXXXXXXXXXX

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
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 can 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 can 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.

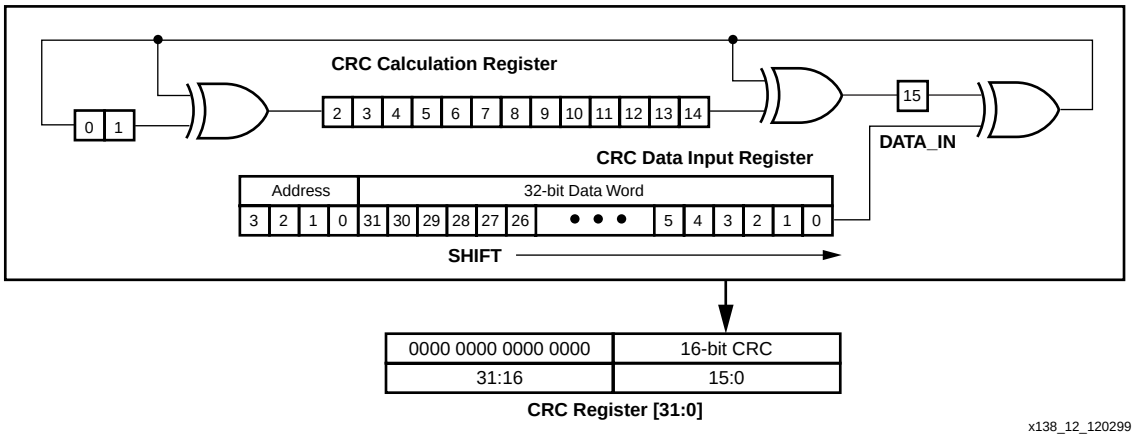

Figure 12: **Serial 16-bit CRC Circuitry**

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 can 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 can 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 can 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.

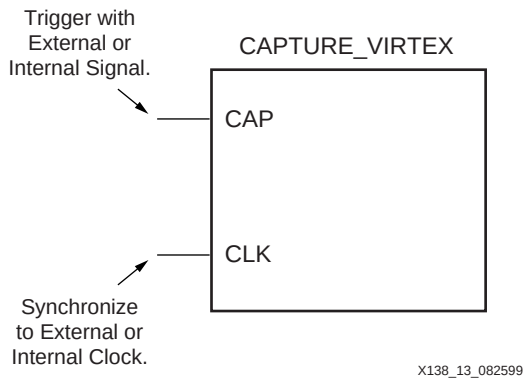

Figure 13: **Readback Capture Library Primitive**

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, can 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" on page 50. The **-m** generates a *Mask* file, which is discussed in "Verifying Configuration Data" on page 46. The **-g readback** generates the *readback bit* file, which is discussed in "Readback Operations" on page 42, 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 19.

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.

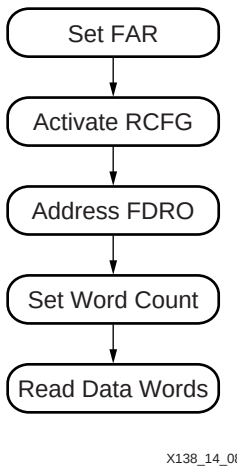

Figure 14: **CLB Readback Operation Flow**

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 can 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" on page 52.

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 on page 47.

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 can 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 52.

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 can be used depending on the amount of data that is to be readback. See "Bitstream Format" on page 31.

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

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

Device	TYPE 2 Packet Header for CLB Frames	CLB Frames Word Count
XCV600	4801 A90Ah	108810
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 46. 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 can be repeated for additional readbacks.

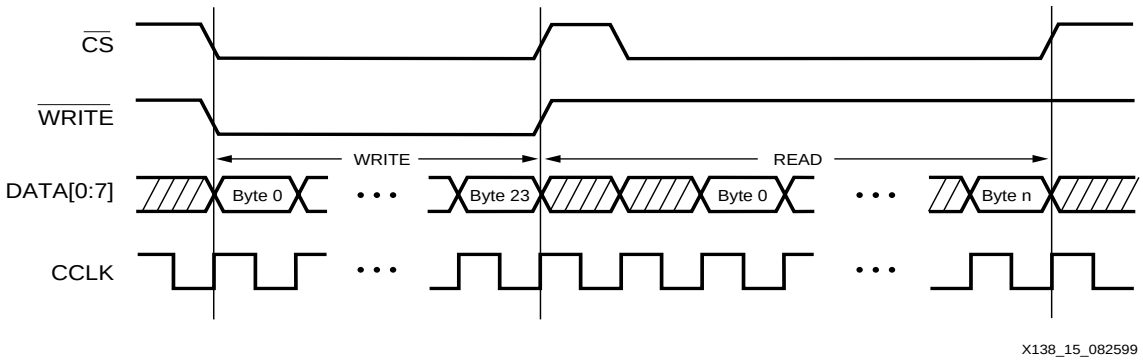

Figure 15: **Readback Clocking Sequence**

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.

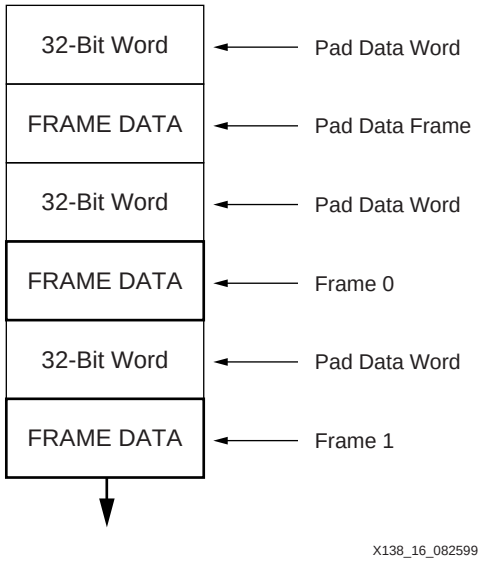

Figure 16: **Readback Data Stream**

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

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

Device	CLB Frames	Bytes per Frame	Frame Bytes	Pad Bytes	Readback Bytes
XCV1600E	5516	168	926688	22240	948928
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 can 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 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 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 can be ignored.

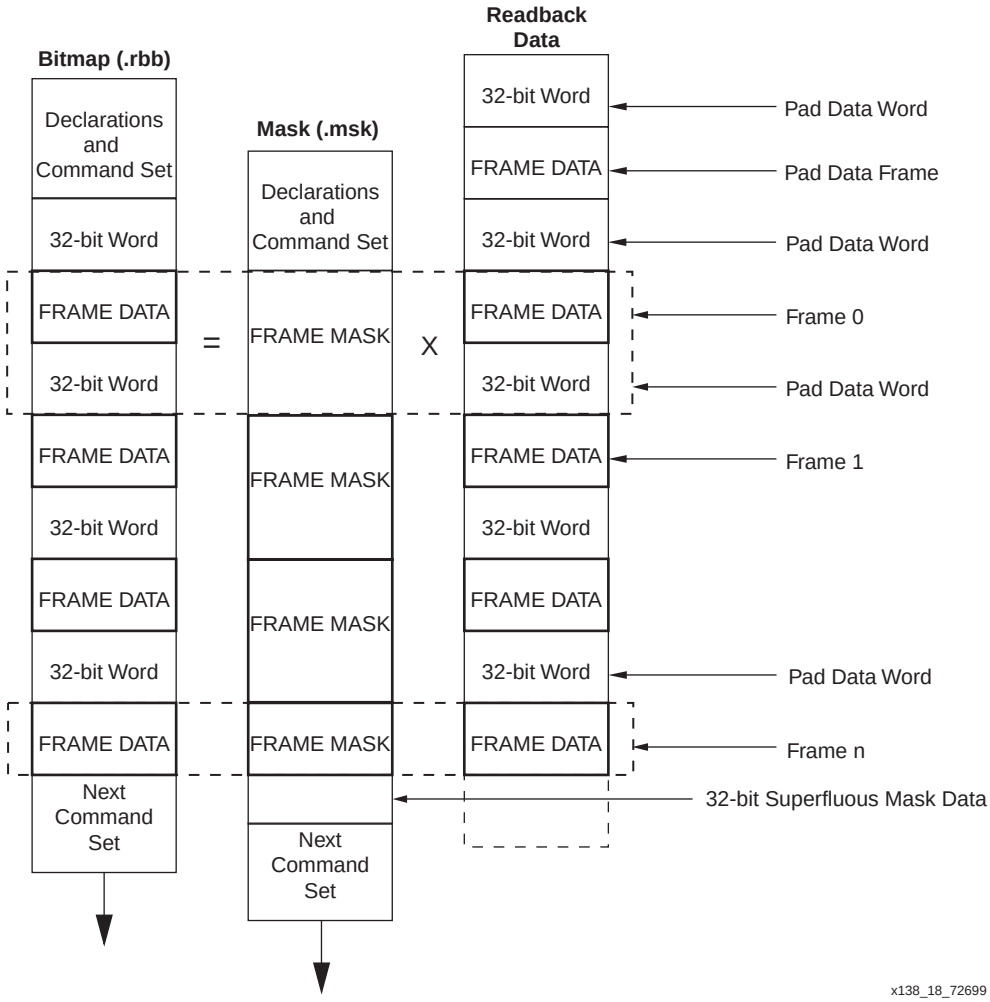

Figure 18: **Readback Data Stream Alignment**

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.

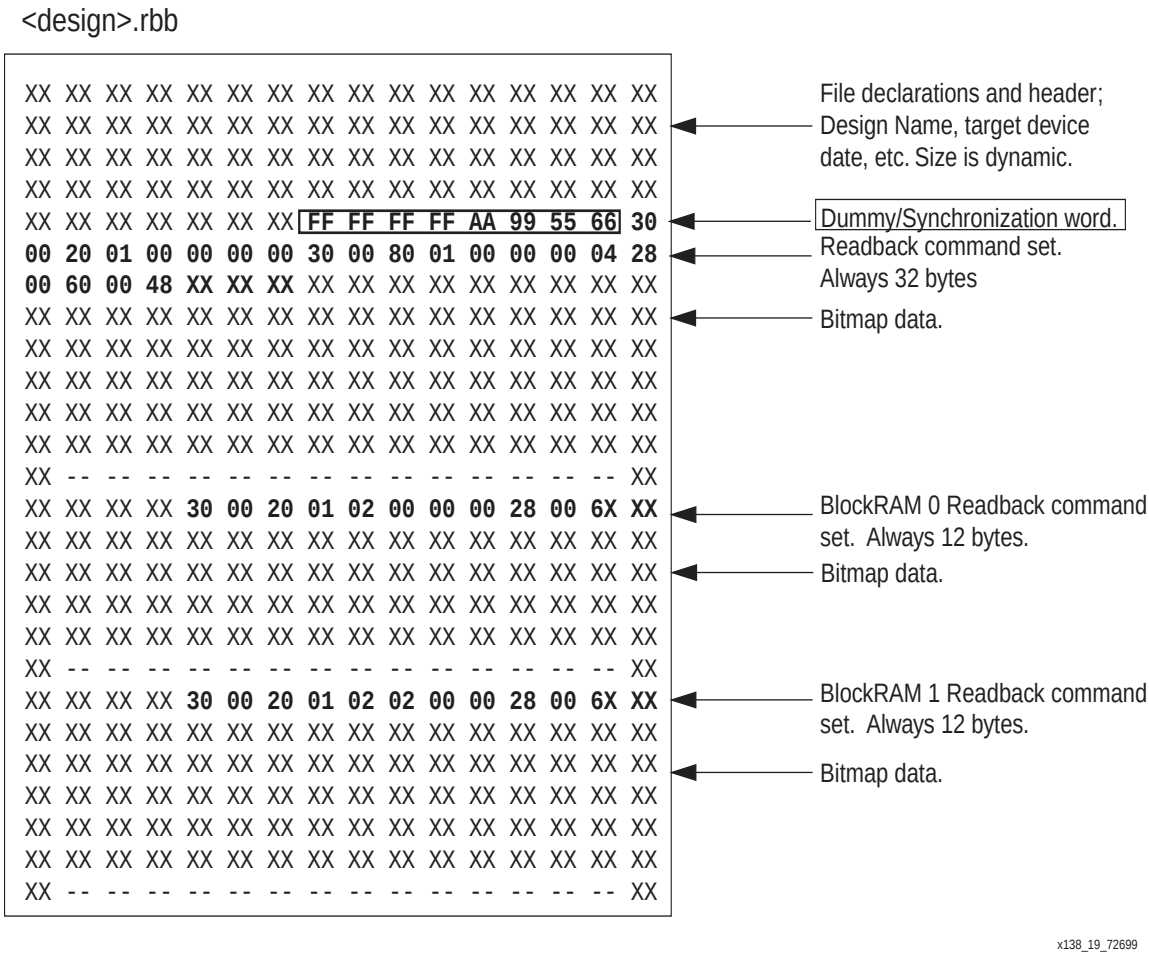

Figure 19: **Readback Bit File for the Virtex Family**

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.

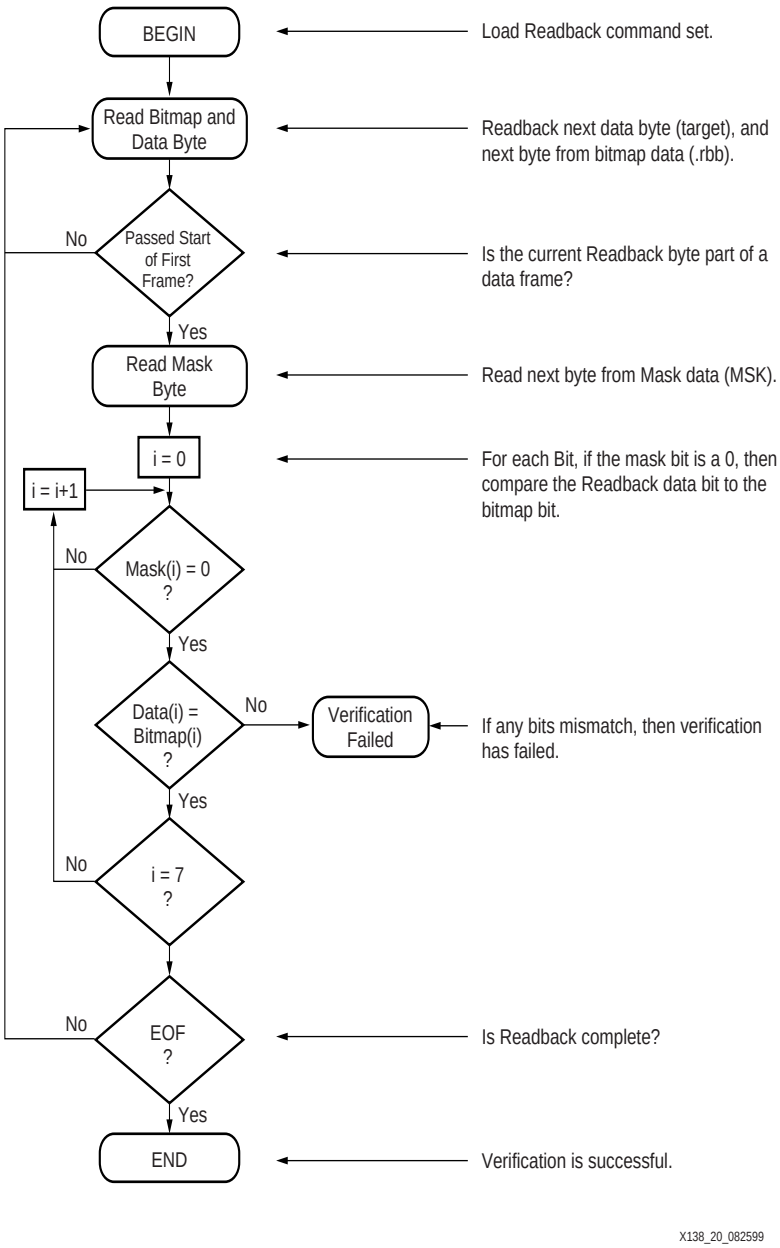

Figure 20: **Readback Verification Flow Diagram**

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.

Readback Bit Number = (*FrameNumber* x *FrameLength*) + *Bitmap Length* – *FrameOffset* + 32

The Frame Number and Frame Offset are given in the .11 file. The Frame Length is shown on Table 4 on page 31.

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 can 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 can 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 can 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 22. 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 42. However, the synchronization step can 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.

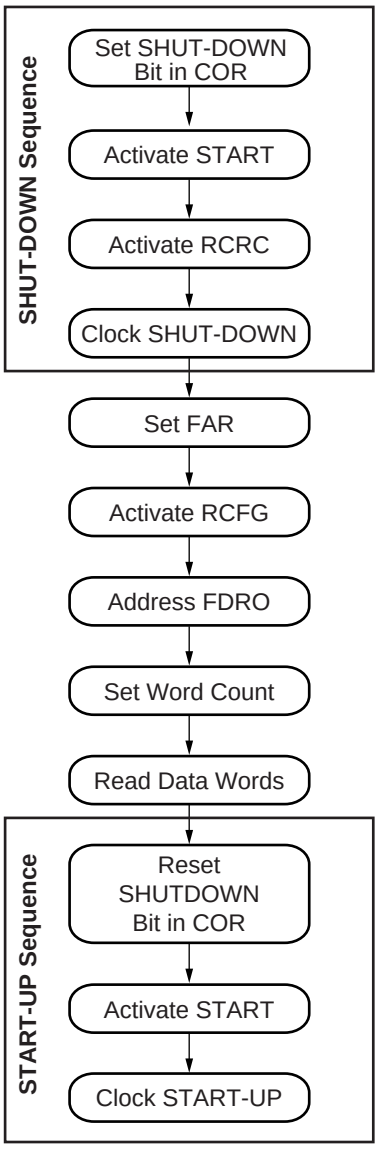

Figure 21: **Block RAM Readback Operation Flow**

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
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 can 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.

XILINX®

Chapter 3

Configuration and Readback Through JTAG

Introduction

The IEEE 1149.1 Test Access Port and Boundary-Scan architecture, commonly referred to as JTAG, is a popular testing method. JTAG is an acronym for the Joint Test Action Group, the technical subcommittee initially responsible for developing the standard. This standard provides a means to assure the integrity of individual components and the interconnections between them at the board level. With increasingly dense multi-layer PC boards, and more sophisticated surface mounting techniques, boundary-scan testing is becoming widely used as an important debugging standard.

Devices containing boundary-scan logic can send data out on I/O pins in order to test connections between devices at the board level. The circuitry can also be used to send signals internally to test the device specific behavior. These tests are commonly used to detect opens and shorts at both the board and device level.

In addition to testing, boundary-scan offers the flexibility for a device to have its own set of user-defined instructions. The added common vendor specific instructions, such as configure and verify, have increased the popularity of boundary-scan testing and functionality.

Boundary-Scan for Virtex Devices

The Virtex family is fully compliant with the IEEE Standard 1149.1 Test Access Port and Boundary-Scan Architecture. The architecture includes all mandatory elements defined in the IEEE 1149.1 Standard. These elements include the Test Access Port (TAP), the TAP controller, the instruction register, the instruction decoder, the boundary-scan register, and the bypass register. The Virtex family also supports some optional instructions; the 32-bit identification register, and a configuration register in full compliance with the standard. Outlined in the following sections are the details of the JTAG architecture for Virtex devices.

Test Access Port

The Virtex TAP contains four mandatory dedicated pins as specified by the protocol (Table 20).

Table 20: **Virtex TAP Controller Pins**

Pin	Description
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
TCK	Test Clock

There are three input pins and one output pin to control the 1149.1 boundary-scan TAP controller. In addition to the required pins, there are optional control pins such as $\overline{\text{TRST}}$ (Test Reset) and enable pins which can be found on devices from other manufacturers. It is important to be aware of these optional signals when interfacing Xilinx devices with parts from different vendors, because they can need to be driven.

The TAP controller is a 16-state state machine shown in Figure 22. The four mandatory TAP pins are outlined below.

- TMS - This pin determines the sequence of states through the TAP controller on the rising edge of TCK. TMS has an internal resistive pull-up to provide a logic High if the pin is not driven.
- TCK - This pin is the JTAG test clock. It sequences the TAP controller and the JTAG registers in the Virtex devices.
- TDI - This pin is the serial input to all JTAG instruction and data registers. The state of the TAP controller and the current instruction held in the instruction register determine which register is fed by the TDI pin for a specific operation. TDI has an internal resistive pull-up to provide a logic High to the system if the pin is not driven. TDI is applied into the JTAG registers on the rising edge of TCK.
- TDO - This pin is the serial output for all JTAG instruction and data registers. The state of the TAP controller and the current instruction held in the instruction register determine which register (instruction or data) feeds TDO for a specific operation. TDO changes state on the falling edge of TCK and is only active during the shifting of instructions or data through the device. This pin is 3-stated at all other times.
- Notes: As specified by the IEEE Standard, the TMS and TDI pins all have internal pull-ups. These internal pull-ups of 50-150 kΩ are active regardless of the mode selected.
- When using boundary-scan operations in Virtex devices, V_{CCO} for Bank 2 must be at 3.3 V for the TDO pin to operate at the required LVTTL level.

TAP Controller

Figure 22 diagrams a 16-state finite state machine. The four TAP pins control how data is scanned into the various registers. The state of the TMS pin at the rising edge of TCK determines the sequence of state transitions. There are two main sequences, one for shifting data into the data register and the other for shifting an instruction into the instruction register.

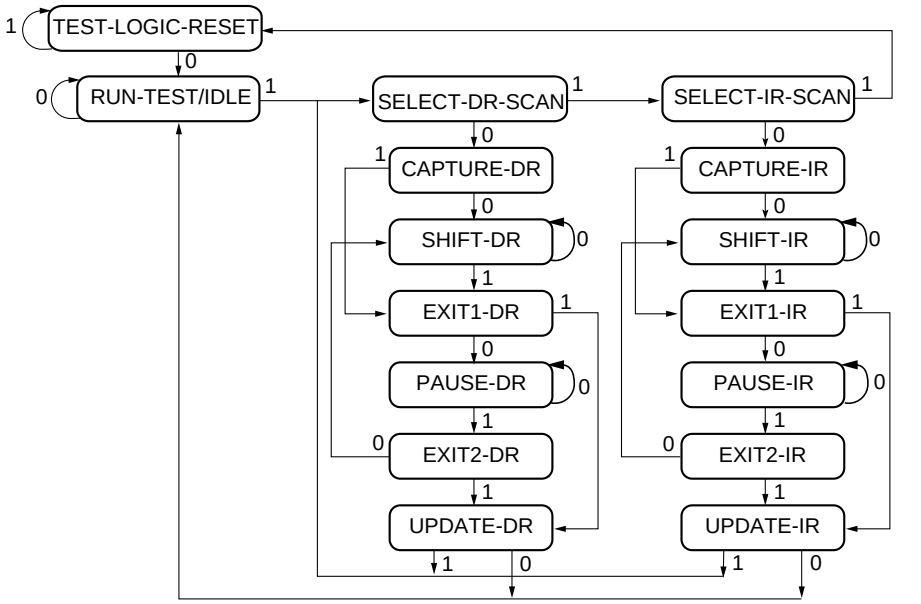

Figure 22: **State Diagram for the TAP Controller**

Boundary-Scan Instruction Set

To determine the operation to be invoked, a 5-bit instruction is loaded into the instruction register. Table 21 lists the available instructions for Virtex devices.

Table 21: **Virtex Boundary Scan Instructions**

Boundary Scan Command	Binary Code (4:0)	Description
EXTEST	00000	Enables boundary-scan EXTEST operation
SAMPLE	00001	Enables boundary-scan SAMPLE operation
USER1	00010	Access user-defined register 1
USER2	00011	Access user-defined register 2
CFG_OUT	00100	Access the configuration bus for readback
CFG_IN	00101	Access the configuration bus for configuration
INTEST	00111	Enables boundary-scan INTEST operation
USERCODE	01000	Enables shifting out user code
IDCODE	01001	Enables shifting out of ID code
HIGHZ	01010	3-states output pins while enabling the bypass register
JSTART	01100	Clocks the start-up sequence when StartClk is TCK
BYPASS	11111	Enables BYPASS
RESERVED	All other codes	Xilinx reserved instructions

The mandatory IEEE 1149.1 commands are supported in Virtex devices as are several Xilinx vendor-specific commands. Virtex devices have a powerful command set. The EXTEST, INTEST, SAMPLE / PRELOAD, BYPASS, IDCODE, USERCODE, and HIGHZ

instructions are all included. The TAP also supports two internal user-defined registers (USER1 and USER2) and configuration / readback of the device. The Virtex boundary-scan operations are independent of mode selection. The boundary-scan mode in Virtex devices overrides other mode selections. For this reason, boundary-scan instructions using the boundary-scan register (SAMPLE / PRELOAD, INTEST, EXTEST) must not be performed during configuration. All instructions except USER1 and USER2 are available before a Virtex device is configured. After configuration, all instructions are available.

JSTART is an instruction specific to the Virtex architecture and configuration flow. As described in Table 21, the JSTART instruction clocks the startup sequence when the appropriate bitgen option is selected. The instruction does not work correctly without the correct bitgen option selected.

```
bitgen -g startupclk:jtagclk designName.ncd
```

For details on the standard boundary-scan instructions EXTEST, INTEST, and BYPASS, please refer to the IEEE Standard. The user-defined registers (USER1 / USER2) are described in a later section of this chapter.

Boundary-Scan Architecture

Virtex device registers include all registers required by the IEEE 1149.1 Standard. In addition to the standard registers, the family contains optional registers for simplified testing and verification (Table 22).

Table 22: **Virtex JTAG Registers**

Register Name	Register Length	Description
Instruction register	5 bits	Holds current instruction OPCODE and captures internal device status.
Boundary scan register	3 bits per I / O	Controls and observes input, output, and output enable.
Bypass register	1 bit	Device bypass.
Identification register	32 bits	Captures device ID.
JTAG configuration register	32 bits	Allows access to the configuration bus when using the CFG_IN or CFG_OUT instructions.
USERCODE register	32 bits	Captures user-programmable code

Boundary-Scan Register

The test primary data register is the boundary-scan register. Boundary-scan operation is independent of individual IOB configurations. Each IOB, bonded or un-bonded, starts as bidirectional with 3-state control. Later, it can be configured to be an input, output, or 3-state only. Therefore, three data register bits are provided per IOB (Figure 23).

When conducting a data register (DR) operation, the DR captures data in a parallel fashion during the CAPTURE-DR state. The data is then shifted out and replaced by new data during the SHIFT-DR state. For each bit of the DR, an update latch is used to hold the input data stable during the next SHIFT-DR state. The data is then latched during the UPDATE-DR state when TCK is low.

The update latch is opened each time the TAP Controller enters the UPDATE-DR state. Care is necessary when exercising an INTEST or EXTEST to ensure that the proper data has been latched before exercising the command. This is typically accomplished by using the SAMPLE / PRELOAD instruction.

Consider internal pull-up and pull-down resistors when developing test vectors for testing opens and shorts. The boundary-scan mode determines if the IOB has a pull-up resistor. (Table 26). Figure 23 is a representation of Virtex Boundary-Scan Architecture.

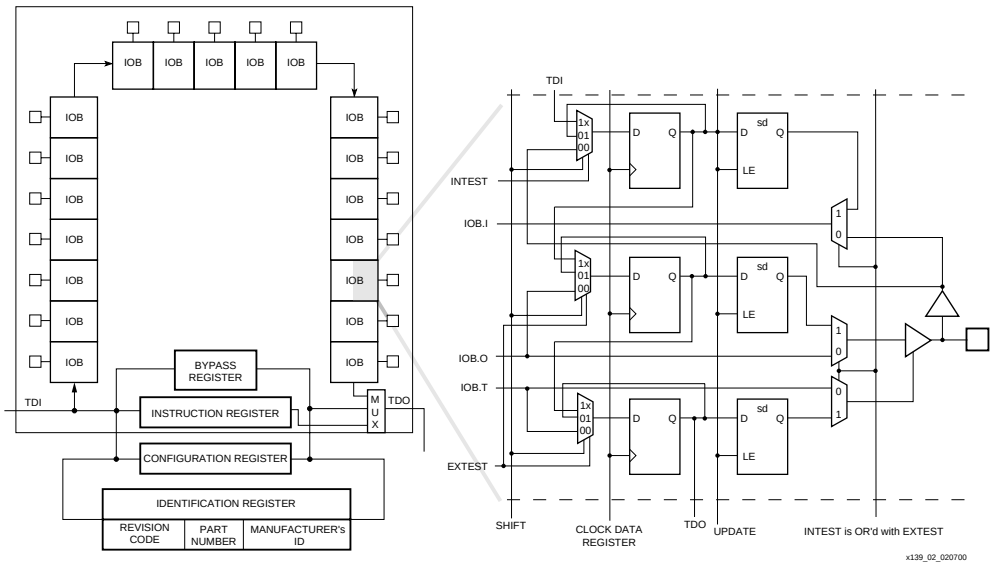

Figure 23: **Virtex Series Boundary Scan Logic**

Bit Sequence

The order in each non-TAP IOB is described in this section. The input is first, then the output, and finally the 3-state IOB control. The 3-state IOB control is closest to the TDO. The input-only pins contribute only the input bit to the boundary-scan I/O data register. The bit sequence of the device is obtainable from the "Boundary-Scan Description Language Files" (BSDL files) for the Virtex family. These files can be obtained from the Xilinx software download area. The bit sequence is independent of the design. It always has the same bit order and the same number of bits.

Bypass Register

The other standard data register is the single flip-flop BYPASS register. It passes data serially from the TDI pin to the TDO pin during a bypass instruction. This register is initialized to zero when the TAP controller is in the UPDATE-DR state.

Instruction Register

The instruction register is a 5-bit register that loads the OPCODE necessary for the Virtex boundary-scan instruction set. This register loads the current OPCODE and captures internal device status.

Configuration Register (Boundary-Scan)

The configuration register is a 32-bit register. This register allows access to the configuration bus and readback operations.

Identification Register

Virtex devices have an identification register, commonly referred to as the IDCODE register. This register is based upon IEEE Standard 1149.1 and allows easy identification of

the part being tested or programmed via boundary scan. The general format of the IDCODE in the Virtex family is identified in Table 23.

Table 23: **Virtex Identification Register**

Revision Code	Part Number		Manufacturers ID	LSB
	Family Code	Part Size Code		
31 ... 28	27 ... 21	20 ... 12	11 ... 1	0
XXXX	0000011	YYYYYYYYY	000 1001 001	1

Specific Virtex family IDCODEs are listed in Table 24.

Table 24: **IDCODEs Assigned to Virtex FPGAs**

FPGA	IDCODE
XCV50	v0610093h
XCV50E	v0A10093h
XCV100	v0614093h
XCV100E	v0A14093h
XCV150	v0618093h
XCV200	v061C093h
XCV200E	v0A1C093h
XCV300	v0620093h
XCV300E	v0A20093h
XCV400	v0628093h
XCV400E	v0A28093h
XCV405E	v0C28093h
XCV600	v0630093h
XCV600E	v0A30093h
XCV800	v0638093h
XCV812E	v0C38093h
XCV1000	v0640093h
XCV1000E	v0A40093h
XCV1600E	v0A48093h
XCV2000E	v0A50093h
XCV2600E	v0A5C093h
XCV3200E	v0A68093h

Note: The “v” in the IDCODE is the revision code field.

USERCODE Register

USERCODE is supported in the Virtex family as well. This register allows a user to specify a design-specific identification code. The USERCODE can be programmed into the device and read back for verification at a later time. The USERCODE is embedded into the bitstream during bitstream generation (bitgen -g UserID option) and is only valid after configuration.

USER1, USER2 Registers

The USER1 and USER2 registers are only valid after configuration. These two registers must be defined by the user within the design. These registers can be accessed after they are defined by the TAP pins.

The BSCAN_VIRTEX library macro is required when creating these registers. This symbol is only required for driving internal scan chains (USER1 and USER2). The BSCAN_VIRTEX macro provides two user pins (SEL1 and SEL2) for determining usage of USER1 or USER2 instructions respectively. For these instructions, two corresponding pins (TDO1 and TDO2) allow user scan data to be shifted out of TDO. In addition, there are individual clock pins (DRCK1 and DRCK2) for each user register. There is a common input pin (TDI) and shared output pins that represent the state of the TAP controller (RESET, SHIFT, and UPDATE). Unlike earlier FPGA families that required the BSCAN macro to dedicate TAP pins for boundary scan, Virtex TAP pins are dedicated and do not require the BSCAN_VIRTEX macro for normal boundary-scan instructions or operations.

Note that these are user-defined registers. The example (Figure 24) is one of many implementations. For HDL, the BSCAN_VIRTEX macro needs to be instantiated in the design.

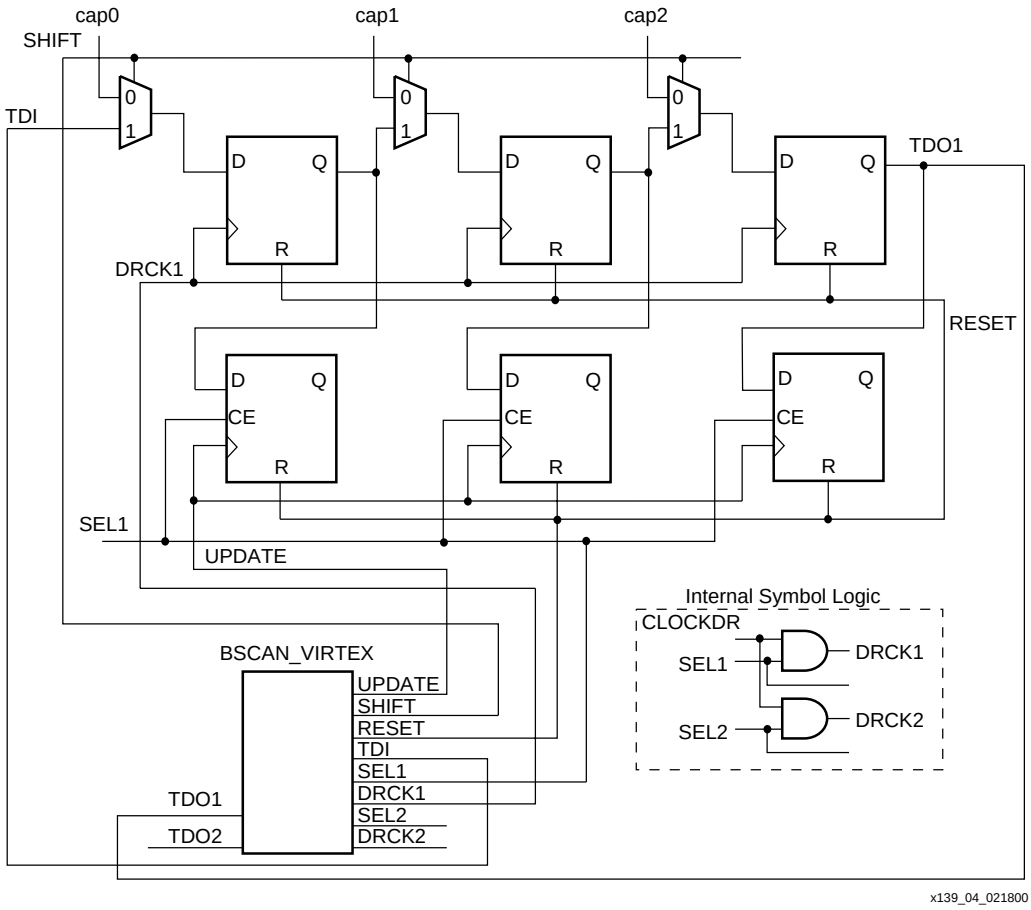

Figure 24: **BSCAN_VIRTEX (Example Usage)**

Using Boundary Scan in Virtex Devices

Characterization data for some of the most commonly requested timing parameters shown in Figure 25 is listed in Table 25.

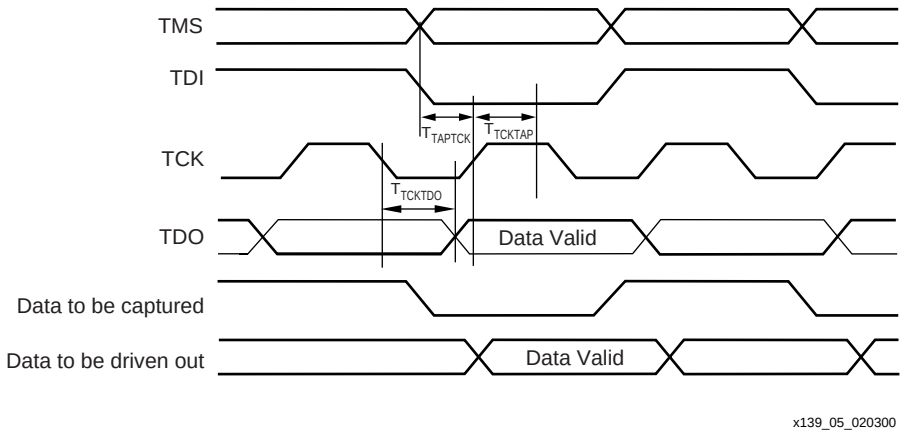

Figure 25: **Virtex Boundary Scan Port Timing Waveforms**

Table 25: **Boundary-Scan Port Timing Specifications**

Symbol	Parameter	-6	-5	-4	Units
T_{TAPTCK}	TMS and TDI setup time before TCK	4.0	4.0	4.0	ns, min
T_{TCKTAP}	TMS and TDI hold times after TCK	2.0	2.0	2.0	ns, min
T_{TCKTDO}	TCK falling edge to TDO output valid	11.0	11.0	11.0	ns, min
F_{TCK}	Maximum TCK clock frequency	33.0	33.0	33.0	MHz, max

For further information on the Startup sequence, bitstream, and internal configuration registers referenced here, refer to “” chapter in this guide.

Configuring Through Boundary-Scan

One of the most common boundary-scan vendor-specific instructions is the configure instruction. An individual Virtex device is configured via JTAG on power-up using TAP. If the Virtex device is configured on power-up, it is advisable to tie the mode pins to one of the boundary-scan configuration mode settings; 101 (M2 = 1, M1 = 0, M0 = 1: contains no pull-ups on I/Os) or 001 (M2 = 0, M1 = 0, M0 = 1: contains pull-ups on I/Os). Table 26 shows the mode pin settings for all the configuration modes, including boundary-scan modes.

Table 26: **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
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

Configuration flow for Virtex device configuration with JTAG is shown in Figure 27. The sections that follow describe how the Virtex device can be configured as a single device via boundary-scan or as part of a multiple-device scan chain.

A configured device can be reconfigured by toggling the TAP and entering a CFG_IN instruction after pulsing the $\overline{\text{PROGRAM}}$ pin or issuing the shut-down sequence. (Refer to “” section in this guide). For additional details on power-up or the start-up sequence in Virtex devices, refer to “” chapter in this guide. For additional detailed information on using Virtex devices in an embedded solution, see Xilinx application note XAPP058.

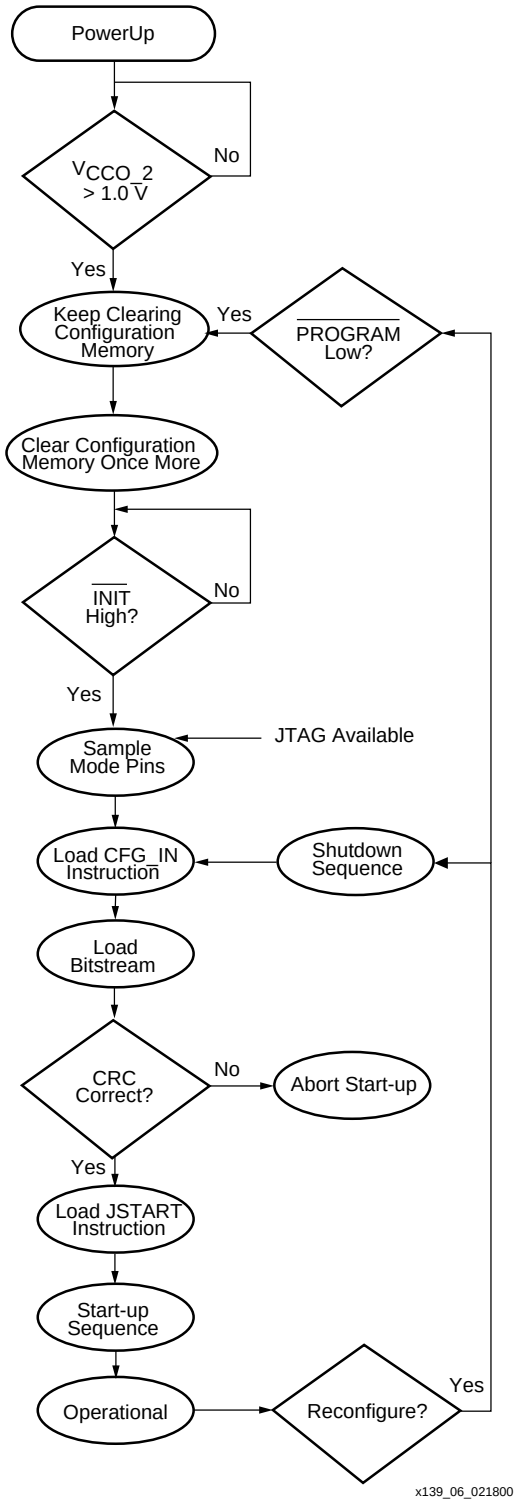

Figure 26: **Device Configuration Flow Diagram**

Single Device Configuration

To configure a Virtex part as a single device via boundary-scan operations, follow the following steps. Ensure the bitstream is generated with the JTAG clock option.

```
bitgen -g startupclk:jtagclk designName.ncd
```

Also, when programming with JTAG Programmer software, verify that the most current version is used.

Table 27 describes the TAP controller commands required to configure a Virtex device. Refer to Figure 22 for TAP controller states. These TAP controller commands are issued automatically if configuring the part with the JTAG Programmer software.

Table 27: **Single Device Configuration Sequence**

TAP Controller Step Description		Set & Hold		# of Clocks
		TDI	TMS	TCK
1	On power-up, place a logic "one" on the TMS and clock the TCK five times. This ensures starting in the TLR (Test-Logic-Reset) state.	X	1	5
2	Move into the RTI state.	X	0	1
3	Move into the SELECT-IR state.	X	1	2
4	Enter the SHIFT-IR state.	X	0	2
5	Start loading the CFG_IN instruction[1].	0101	0	4
6	Load the last bit of CFG_IN instruction when exiting SHIFT-IR. as defined in the IEEE standard.	0	1	1
7	Enter the SELECT-DR state.	X	1	2
8	Enter the SHIFT-DR state.	X	0	2
9	Shift in the Virtex bitstream. Bit_n (MSB) is the first bit in the bitstream[1].	bit_1 ...bit_n	0	(bits in bitstream) – 1
10	Shift in the last bit of the bitstream. Bit_0 (LSB) shifts on the transition to EXIT1-DR.	bit_0	1	1
11	Enter UPDATE-DR state.	X	1	1
12	Enter the SELECT-IR state.	X	1	2
13	Move to the SHIFT-IR state.	X	0	2
14	Start loading the JSTART instruction.(1) The JSTART instruction initializes the startup sequence.	1100	0	4
15	Load the last bit of the JSTART instruction.	0	1	1
16	Move to the SELECT-DR state.	X	1	2
17	Move to SHIFT-DR and clock the STARTUP sequence by applying a minimum of 12 clock cycles to the TCK.	X	0	≥14
18	Move to the UPDATE-DR state.	X	1	2
19	Return to the RTI state. The device is now functional.	X	0	1

Note: In the TDI column, the right-most bit is shifted in first.

Multiple Device Configuration

It is possible to configure multiple Virtex devices in a chain. The devices in the JTAG chain are configured one at a time. The multiple device configuration steps are described generally to be applied to any size chain. Ensure the bitstream is generated with the JTAG clock option.

```
bitgen -g startupclk:jtagclk designName.ncd
```

Refer to the State Diagram in 22 for the following TAP controller steps.

1. On power-up, place a logic “one” on the TMS and clock the TCK five times. This ensures starting in the TLR (Test-Logic-Reset) state.
2. Load the CFG_IN instruction into the target device (and BYPASS in all other devices.)
3. For a chain of Virtex devices, it might be necessary to shift in leading zeros before the bitstream if the value of N is not equal to zero. Use the following equation to determine the number of leading zeros for each bitstream. M is the targeted device position in the chain. As shown in Figure 27, the first device position in the chain is zero. N is the number of zeros required. Mod is the modulus operation.

$$N = 32 - \mathrm{mod}\left(\frac{M}{32}\right)$$

$$N = 32 - M \quad \text{for} \quad (M \leq 32)$$

Example

The third device, position 2 in Figure 27, requires 30 leading zeros.

$$N = 32 - 2 \qquad N = 30 \qquad \text{number of leading 0s}$$

The following example is for position 47:

$$N = 32 - \mathrm{mod}\left(\frac{47}{32}\right) \quad N = 17 \text{ number of leading 0s}$$

4. Go through RTI (RUN-TEST / IDLE)

Repeat steps 2 through 4 for each successive device.

5. Load the JSTART command into all devices.
6. Go to SHIFT-DR and clock TCK 12 times.

All devices are active at this point.

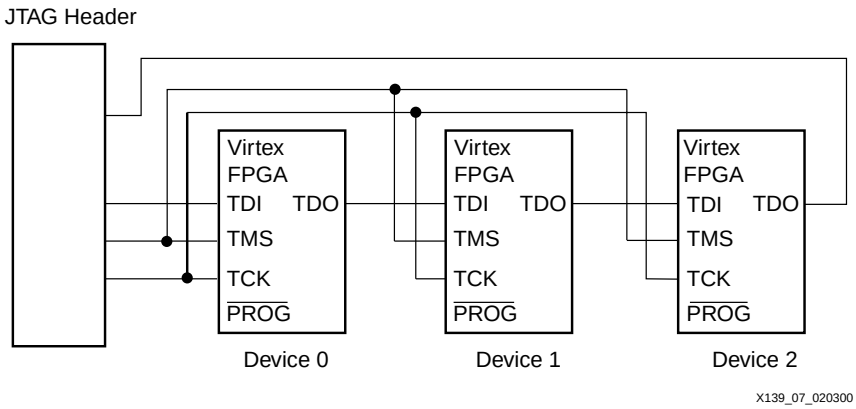

Figure 27: **Boundary Scan Chain of Devices**

Reconfiguring Through Boundary Scan

The ability of Virtex devices to perform partial reconfiguration is the reason that the configuration memory is not cleared when reconfiguring the device. When reconfiguring a chain of devices, refer to step 3 in Table 27. There are two methods to reconfigure Virtex devices without possible internal contention. The first method is to pulse the $\overline{\text{PROGRAM}}$ pin which resets the internal configuration memory. The alternate method is to perform a shutdown sequence, placing the device in a safe state. The following shutdown sequence

includes using internal registers. (For details on internal registers, refer to "" chapter in this guide.)

1. Load the CFG_IN instruction into the JTAG instruction register. Next, go to the SHIFT-DR state.
2. In the SHIFT-DR state, shift in the following sequences in steps 2 through 4. (This writes the COR (Configuration Option Register) with the SHUTDOWN bit = 1. It also indicates that the startup sequencer should perform a shutdown sequence.) The most significant bit (MSB) is the left bit and it is shifted in first.

   ```
   0011 0000 0000 0001 0010 0000 0000 0001
   -> Header: Write to COR
   ```

   ```
   0000 0000 1010 0000 1011 1111 0010 1101
   -> COR data sets SHUTDOWN = 1
   ```

3. Write the START command to the CMD (Command) register by shifting in the following data:

   ```
   0011 0000 0000 0000 1000 0000 0000 0001
   -> Header: Write to CMD
   ```

   ```
   0000 0000 0000 0000 0000 0000 0000 0101
   -> START command
   ```

4. Write the precalculated CRC value to the CRC (Cyclic Redundancy Check) register, or write the RCRC (Reset CRC Register) command to the CMD register as shown:

   ```
   0011 0000 0000 0000 1000 0000 0000 0001
   -> Header: Write to CMD
   ```

   ```
   0000 0000 0000 0000 0000 0000 0000 0111
   -> RCRC command
   ```

   ```
   0000 0000 0000 0000 0000 0000 0000 0000
   -> flush pipe
   ```

5. Proceed to the SHIFT-IR and load the JTAG JSTART command into the instruction register.
6. Go to the SHIFT-DR and clock TCK 12 times to clock the shutdown sequence.
7. Proceed to the SHIFT-IR state and load the CFG_IN instruction again.
8. In the SHIFT-DR state, shift in the sequences in steps 8 and 9. This writes the AGHIGH command to the CMD register to assert the GHIGH_B signal. This prevents contention while writing configuration data.

   ```
   0011 0000 0000 0000 1000 0000 0000 0001
   -> Header: Write to CMD
   ```

   ```
   0000 0000 0000 0000 0000 0000 0000 1000
   -> AGHIGH command asserts GHIGH_B
   ```

9. Write the COR with SHUTDOWN = 0 and go to RTI (RUN-TEST / IDLE) by shifting in the following sequence:

   ```
   0011 0000 0000 0001 0010 0000 0000 0001
   -> Header: Write to COR
   ```

   ```
   0000 0000 1010 0000 0011 1111 0010 1101
   -> COR data sets SHUTDOWN = 0
   ```

   ```
   0000 0000 0000 0000 0000 0000 0000 0000
   -> flush pipe
   ```

Debugging Configuration

To verify successful configuration, there are several options. Some of the most helpful verification steps include using TAP pins and the readback command. Using the Virtex TAP controller and status pins is discussed first.

When using TAP controller pins, TDO is only driven in the SHIFT-DR and SHIFT-IR state. If the output of the TDO can be changed via an external pull-up resistor, the TAP is not in SHIFT-IR or SHIFT-DR. If the TAP can be controlled precisely, use this to test the application.

In JTAG configuration, the status pin (DONE) functions the same as in the other configuration modes. The DONE pin can be monitored to determine if a bitstream has been completely loaded into the device. If DONE is Low, the entire bitstream has not been sent or the start-up sequence is not finished. If DONE is High, the entire bitstream has been received correctly. The $\overline{\text{INIT}}$ pin functions similar to a normal $\overline{\text{INIT}}$ but does not indicate a configuration error in boundary-scan configuration.

If the DONE pin is not asserted High, there are several possible reasons.

1. The bitstream option, `bitgen -g startupclk:jtagclk` described in ""Software Support and Data Files" on page 70" might not have been used.
2. The JSTART instruction was not issued.
3. There was an error in the bitstream.

In addition to external pin monitoring, an internal test can be conducted. The second method includes the following steps to capture the contents of the internal device status register:

1. Move the TAP to the TLR (Test-Logic-Reset) state.
2. Go to the SHIFT-IR state and load in the CFG_IN instruction.
3. Go to the SHIFT-DR state and shift in the following 64-bit pattern with the MSB (left-most bit), shifted in first.

   ```
   0010 1000 0000 0000 1110 0000 0000 0001
   -> Header: Read Status Register
   0000 0000 0000 0000 0000 0000 0000 0000
   -> flush pipe
   ```

4. After shifting in this pattern, load the CFG_OUT instruction in the SHIFT-IR state.
5. Move to SHIFT-DR and clock TCK 32 times while reading TDO. The data seen on TDO is the content of the status register. The last bit out is a one if a CRC error occurred. If successful, it should be as follows.

   ```
   0000 0000 0000 0000 0111 1111 0101 1110
   CRC Bit
   ```

The device status register also gives the status of the DONE and $\overline{\text{INIT}}$ signals. For information on the status register, refer to Figure 28 and Chapter 5 in this guide.

Refer to the **XAPP104:** "A Quick JTAG ISP Checklist" for general techniques on how to reduce noise and signal degradation in JTAG chains.

																DONE	INIT	MODE			GHIGH_B	GSR_B	GWE_B	GTS_CFG	IN_ERROR	LOCK				CRC_ERROR
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 28: **Status Register Fields**

Readback Instructions

Readback is available through both the Boundary-Scan and SelectMAP™ interfaces. It is the process of verifying that the current configuration data in the device is correct by reading the data in the internal configuration memory.

Verify Readback

Readback verification is not unique to JTAG. This section discusses boundary-scan-specific steps to perform a readback operation. For detailed information on verify / readback not specific to JTAG, refer to "" chapter in this guide.

1. Load in the CFG_IN instruction into the JTAG IR and then go to the SHIFT-DR.
2. Shift in a packet to write the starting frame address into the FAR (Frame Address Register). For a full-chip readback, this is frame zero of CLB column zero.
3. Shift in a packet to write the RCFG (Read Configuration Data) command into the CMD register.
4. Shift in a packet header requesting a read of the Frame Data Output Register (FDRO). The word count should reflect the number of frames you want to read.
5. Shift in an extra 32 bits. These are to flush the pipeline through the packet processor. Then go back to RTI.
6. Load the CFG_OUT instruction into the JTAG IR and then go to the Shift-DR.
7. Clock TCK and read TDO. There is one word plus one frame of garbage before the readback data appears.
8. To read block RAM data, repeat steps one through seven by substituting the appropriate block RAM address for the starting CLB address. It is recommended that the system be halted prior to block RAM readback.

Software Support and Data Files

The current version of the Xilinx JTAG Programmer software that supports Virtex devices is 2.1i. The Xilinx tool set includes the JTAG Programmer to program and get Virtex IDCODEs. For test vectors EXTEST or INTEST, or to utilize other JTAG features present in the devices, see www.xilinx.com for third party boundary-scan software tools.

IMPORTANT NOTE:

To perform any configuration operations through JTAG, the bitgen option should be set for the JTAG clock option.

```
bitgen -g startupclk:jtagclk designName.ncd
```

For Readback operations, this option can be used.

```
bitgen -w -l -m -g readback
```

Readback is not supported in the current version of JTAG Programmer Software.

XILINX®

Chapter 4

Configuring Virtex FPGAs From Parallel EPROMs

Introduction

Xilinx FPGAs have historically supported a Master Parallel Configuration Mode that allowed an FPGA to configure itself directly from a parallel (byte-wide) PROM. The Virtex family of Xilinx FPGAs does not have a Master Parallel mode. This chapter describes a simple interface design to configure a Virtex device from a parallel PROM using the SelectMAP™ configuration mode.

Many FPGA users prefer to store configuration data on parallel PROMs because they are available in greater storage capacities than serial PROMs. A parallel PROM stores data as an addressed byte which is accessed by an address bus.

Configuring a Virtex FPGA through the SelectMAP mode is eight times faster than bit-serial mode. The SelectMAP configuration mode utilizes an 8-bit configuration bus and 7 control signals for synchronization and handshaking of data.

Combining larger memory capacities with an 8-bit interface results in faster configuration with fewer storage elements and faster system initialization. This is made possible by adding a small interface design to provide addresses for the parallel PROM and the necessary control signals for SelectMAP configuration.

SelectMAP in Virtex

SelectMAP uses a byte-wide bidirectional access port for reading and writing the configuration data of a Virtex FPGA. It is very similar to the Express mode used by XC4000XLA and SpartanXL FPGAs. SelectMAP uses the following signals:

- D7...D0 - D7 through D0 comprise the 8-bit bidirectional data bus.
- CCLK - CCLK is the configuration clock input signal used by the internal configuration logic.
- $\overline{\text{PROGRAM}}$ - The $\overline{\text{PROGRAM}}$ input signal resets the internal configuration logic and re-initializes the internal configuration memory.
- DONE - The DONE output indicates the completion of configuration and the beginning of the Startup sequence.
- $\overline{\text{INIT}}$ - The bidirectional open-drain $\overline{\text{INIT}}$ pin is used to hold off configuration initialization, and it indicates when CRC errors occur in the configuration data.
- $\overline{\text{WRITE}}$ - The $\overline{\text{WRITE}}$ input is a write strobe that must be asserted and held throughout the loading of data.
- BUSY - When $\overline{\text{CS}}$ is asserted, BUSY output indicates whether the current byte is being loaded or ignored. When $\overline{\text{CS}}$ is not asserted, BUSY is disabled and pulled High.
- CS - The $\overline{\text{CS}}$ input is the Chip Select signal used to enable the FPGA to be sensible of

the SelectMAP interface.

- M2 M1 M0 - The MODE pins M[2:0] must be set to select the desired configuration mode. For SelectMAP configuration, these must be 110.

Configuration Process Steps

The following steps are for performing a Virtex SelectMAP configuration.

1. **Reset the Configuration Logic**. If configuration follows power-up or a recycling of the power source, then the configuration logic will automatically be initialized. However, if the FPGA is to be reconfigured without the recycling of power, then the $\overline{\text{PROGRAM}}$ input must be asserted Low for at least 300 ns to reset the configuration logic.
2. **Time-out Start of Configuration**. After the release of the $\overline{\text{PROGRAM}}$ input, or the powering up of the FPGA, the $\overline{\text{INIT}}$ output stays Low while the internal configuration memory is automatically cleared.

   Unlike previous generations of FPGAs, Virtex does not require an additional time-out period following initialization. Configuration can begin as soon as $\overline{\text{INIT}}$ has gone High. The Virtex FPGA uses the first three clock cycles after $\overline{\text{INIT}}$ has gone High to initialize the configuration circuitry; however, a Virtex Bit-Stream is padded with 8 dummy bytes at the beginning of the data stream to account for this.
3. **Loading Configuration Data**. To begin configuration, the $\overline{\text{WRITE}}$ and $\overline{\text{CS}}$ inputs must be asserted Low and held. Each byte is loaded on the rising edge of CCLK. If BUSY was Low during the CCLK transition then the byte was accepted. If it was High then the byte was ignored and must be reloaded on the next clock cycle. BUSY is a synchronous signal; Therefore, CCLK can not be suspended until BUSY is de-asserted. For configuration clocks speeds below 50 MHz, BUSY is guaranteed to be inactive, and thus can be ignored entirely, and removed from the interface design.
4. **End of Configuration**. The DONE output transitions High to indicate the end of configuration and the beginning of the startup sequence. During the startup sequence the D7...D0, $\overline{\text{WRITE}}$, BUSY, $\overline{\text{INIT}}$, and $\overline{\text{CS}}$ pins all become user I/O. If any of these pins are used by the configured design, then any driving source used only for configuration purposes must be disabled so as not to contend.

   Depending on the startup options selected in BitGen, completing the startup phase requires as many as eight CCLK cycles after DONE has gone High. However, since the example shown below uses a free-running oscillator, this is not a concern. Virtex FPGAs complete the startup phase even if the $\overline{\text{CS}}$ signal is de-asserted

Interface Design

To configure the Virtex FPGA from a parallel PROM, a small interface is needed to generate the PROM addresses and load the data into the FPGA. Any one of several

technologies might be used. A Xilinx CPLD is a simple and cost-effective implementation, as shown in Figure 29.

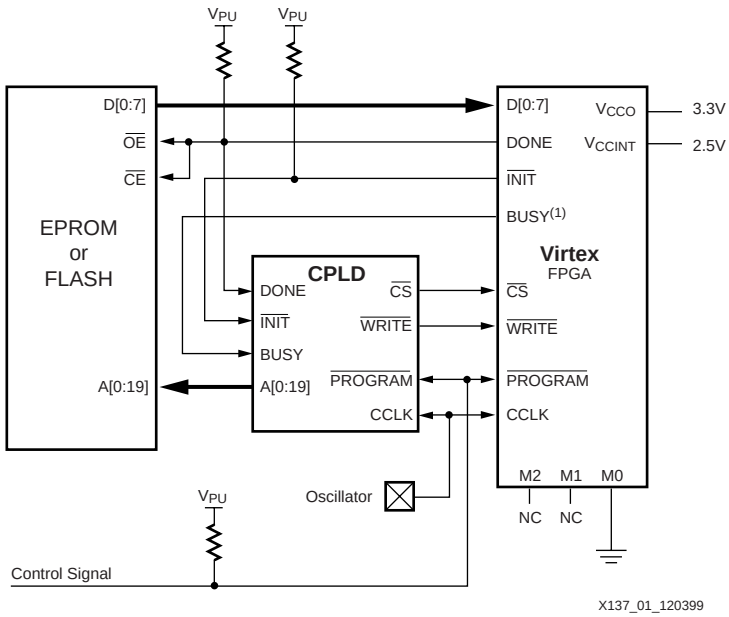

Figure 29: **SelectMAP and EPROM Interface Circuit Diagram**

Voltage Supplies and Pull-Ups

The Virtex FPGA was designed for mixed-voltage environments. The internal voltage (V_{CORE}) requires a 2.5 V supply. During configuration, the I/O are automatically set to the Low Voltage Transistor-to-Transistor (LVTTL) standard, which requires a V_{CCO} of 3.3 V. If this is incompatible with the I/O standard to be used after configuration, V_{CCO} can be driven for the desired standard during configuration with the configuration pins (DONE, INIT, and BUSY) up to VPU. If the V_{CCO} is left floating, then V_{CCO_2} must have a pull-up also.

The VPU refers to the needed I/O voltage for the CPLD and PROM. Most likely, this will either be 3.3 V or 5.0 V. Either value is acceptable to the Virtex FPGA. No translation is needed. The recommended pull-up value is 4.7 kΩ. This value is not critical. Smaller pull-up resistors will of course increase current draw when that node is driven to a logic Low.

The PROMMAP Design

The CPLD design PROMMAP, shown in Figure 31, consists of an address counter, a write control register, some 3-state buffers and random logic. The $\overline{PROG}$ input acts as a global reset. DONE acts as a global 3-state. $\overline{INIT}$ and BUSY hold the address counter from incrementing while the FPGA initializes or processes data.

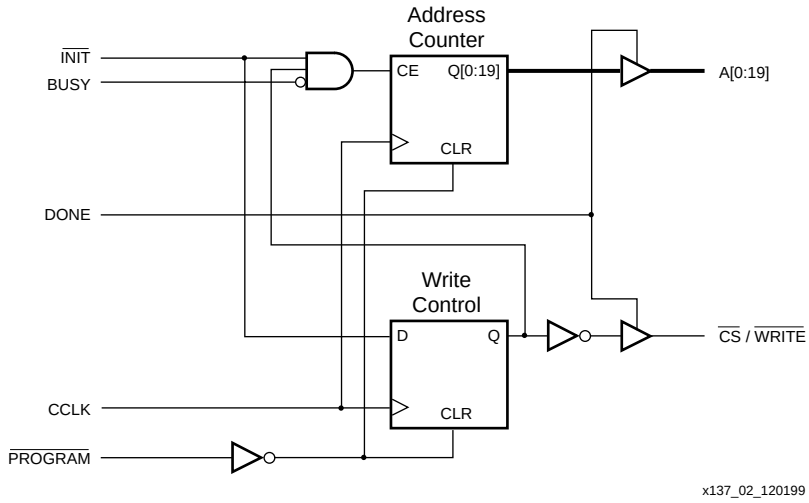

Figure 30: **PROMMAP Interface Design**

Address Counter

The address counter needs to be large enough to accommodate the address bus width of the PROM. A 1-MB (8-Mb) PROM that requires a 20-bit address bus is shown in Figure 31. For connecting multiple PROMs, see "Interfacing Multiple PROMs" on page 75.

WRITE Control Register

The CS and WRITE input signals to the FPGA must be asserted and held Low for configuration. If only one FPGA is being targeted for configuration, these two signals can be derived from the same source. For targeting multiple FPGAs, see "Multiple FPGAs in a Parallel Chain" on page 76. Configuration must be delayed until the INIT signal has gone High, indicating that FPGA initialization is complete.

Oscillator

The Virtex configuration logic was designed to work with a free-running oscillator. The maximum input oscillator frequency for a Virtex FPGA is 66 MHz. In this design, the maximum frequency for the oscillator is constrained by the access time of the PROM (t_{ACC}) plus the setup time for the SelectMAP data inputs (t_{SMDCC})

$$\text{Oscillator Frequency} = \frac{1}{t_{ACC} + t_{SMDCC}}$$

The t_{SMDCC} for a Virtex FPGA is 2.0 ns. A typical t_{ACC} for an EPROM (for example. AT27c080) is 100 ns. Using these values the maximum frequency of the oscillator is 9.6 MHz (76.8 Mb / s). This is far below the CCLK frequency capabilities of the Virtex (66 MHz) and CPLD (66.6 MHz in a XC9536-10). Therefore, a faster PROM would allow for faster configuration speeds.

PROMMAP Design Files

The PROMMAP design was built in an XC9536VQ44-10 device and tested with an XCV300BG432 device and an AT27c080. VHDL and Verilog source code examples, as well as a Foundation Project Schematic for the design, can be downloaded from the Xilinx web site at: **ftp://ftp.xilinx.com/pub/applications/xapp/xapp137.zip**.

Interfacing Multiple PROMs

If the needed memory capacity requires the use of multiple PROMs, then the interface design can be altered to accommodate PROM daisy-chaining as shown in Figure 31.

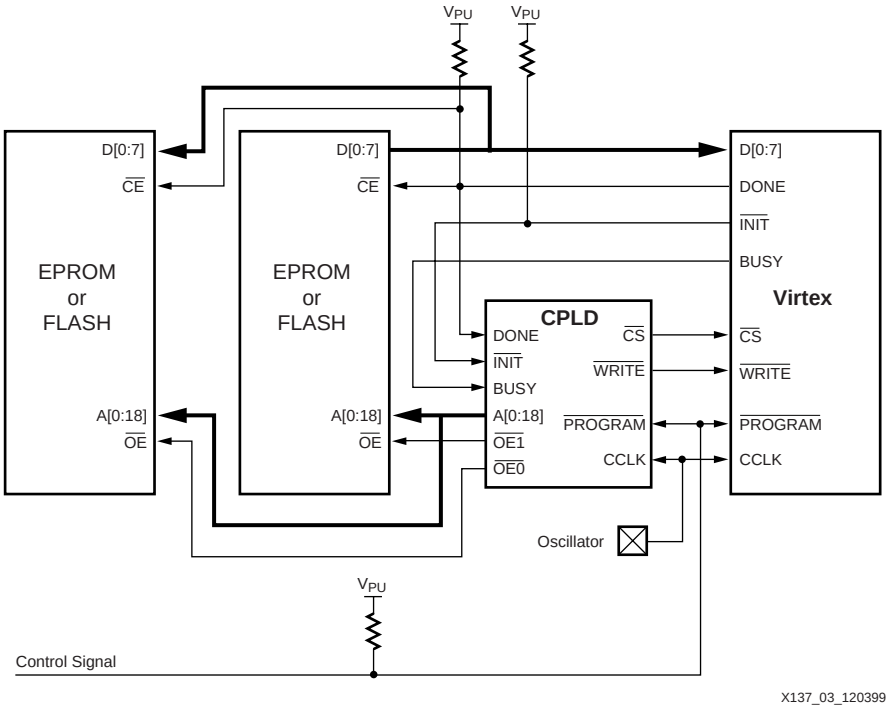

Figure 31: **Connecting Multiple PROMs in Daisy-Chain**

The output enable (OE) input pins of the PROMs can be driven by an address decode of the most significant bits (MSB) of the address counter in the PROMMAP design, as shown in Figure 32.

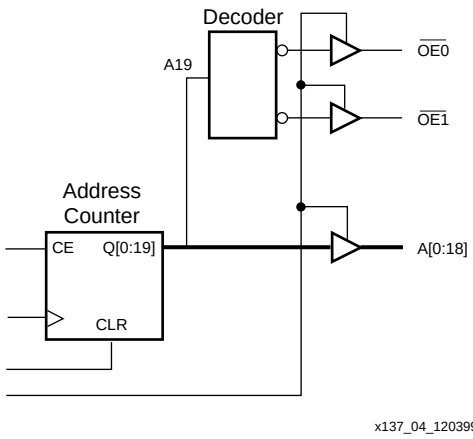

Figure 32: **Selecting Multiple PROMs**

For the example shown in Figure 31, the 8-Mb PROM has been replaced with two 4-Mb PROMs. The common address bus is now only 19-bits wide and the MSB A(19) is used to select between the PROMs (see Table 28).

Table 28: **Address Decode for Two PROM Selection**

A(19)	OE0	OE1
0	0	1
1	1	0

The $\overline{OE}$ outputs are just a binary decode of an additional MSB of the address counter. To add more PROMs, simply increase the size of the counter and binary decoder.

Multiple FPGAs in a Parallel Chain

SelectMAP configuration does not support a standard daisy-chain configuration as bit-serial modes do. However, in an application such as this, configuring multiple FPGAs, the PROMMAP design can be altered to accommodate a parallel chain so that multiple interface chips are not necessary. As shown in Figure 33, the parallel-chaining of multiple FPGAs is similar to cascading multiple PROMs.

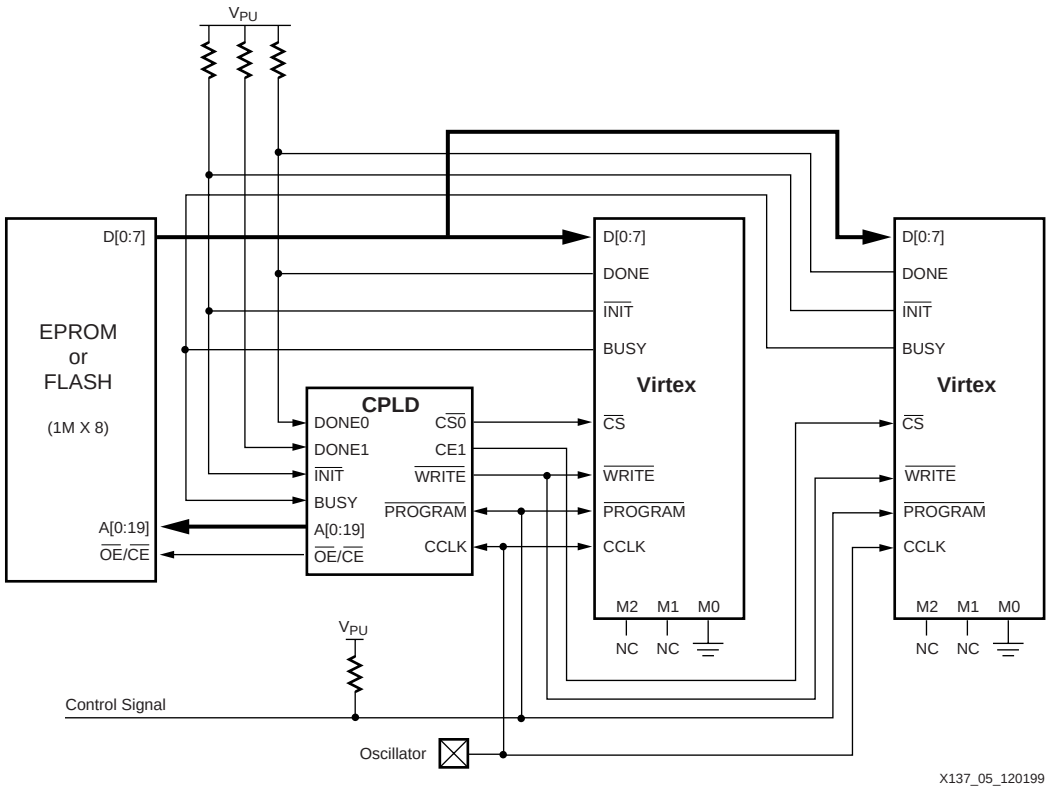

Figure 33: **Connecting Multiple FPGAs for Parallel Configuration Chain**

For each FPGA, the DONE and $\overline{CS}$ signals must be connected to the interface design separately. The BUSY and $\overline{WRITE}$ signals can be a common line between FPGAs. When the first FPGA DONE signal goes High, the interface must disable the $\overline{CS}$ input of that FPGA and enable the $\overline{CS}$ input of the next FPGA in the chain, until all DONE signals are High. An example of this is shown in Figure 34.

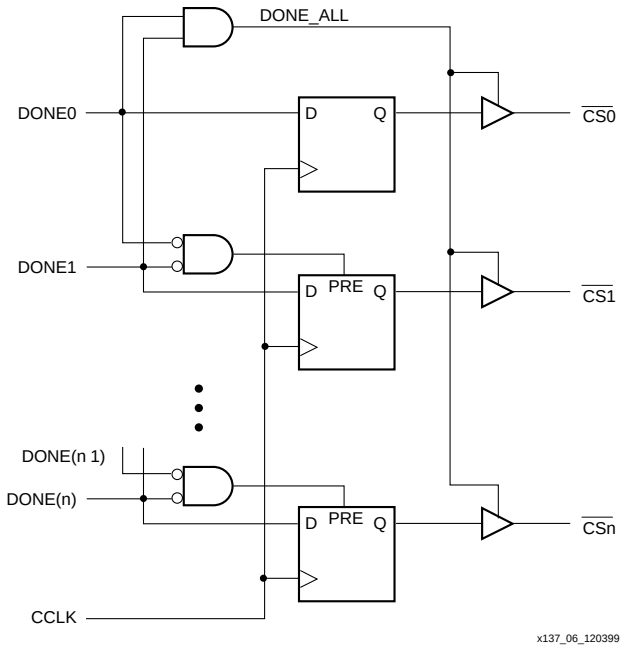

Figure 34: **Sequencing FPGA Chip Selects**

Using this method causes each FPGA to become active after its DONE signal goes High. Since FPGAs do not go active simultaneously, it is important that none of the configuration pins used for SelectMAP configuration be used as I/O in the FPGA designs. In such cases, there could be contention if one or more FPGAs became configured and active while other FPGAs are not configured.

Alternatively, if it is desired to use these pins after configuration, the common $\overline{\text{WRITE}}$ signal input for each FPGA must be connected to GTS and GSR (or other reset signal), so these pins are not driven by active FPGAs while others are still configuring.

Figure 34 shows how the $\overline{\text{CS}}$ outputs are cascaded accordingly with the DONE inputs. In the single FPGA example, the DONE input is the global 3-state control for all outputs. For multiple FPGAs, the global 3-state signal (DONE_ALL) is generated by an AND function of all the FPGA DONE signals.

The dotted extension in Figure 34 demonstrates that more FPGAs can be added to the chain by adding chip select registers. For each segment, the register is held asynchronously PREset when the DONE (n-1) and the associated DONE (n) are both Low. When DONE (n-1) goes High, PREset is released and the register loads the value of the associated DONE (n) signal on each clock cycle. When the associated DONE goes High, the corresponding $\overline{\text{CS}}$ output goes High, disabling the configuration for that FPGA in the chain. This removes the PREset of the next segment and allows configuration of the next FPGA in the chain. The first segment does not make use of a PREset. When all DONE signals are High, all outputs become disabled.

The $\overline{\text{WRITE}}$ signal needs to be held Low throughout the entire configuration process, but can be disabled when DONE goes High. Since $\overline{\text{CS}}$ selects do not assert and the address counter does not increment until $\overline{\text{INIT}}$ is High, the $\overline{\text{WRITE}}$ signal input to the output 3-state buffer can simply be a GND connection (a true Open-Drain style output).

Synchronizing Start-Up

An alternative method to activating FPGAs sequentially would be to synchronize the startup of all FPGAs by combining all the DONE lines into one signal. Virtex automatically synchronizes the startup sequence to wait for DONE to be externally released. It is,

however, important that the BitGen option to "DriveDone" is deselected. Unless otherwise specified, the DONE pin is an open-drain output.

Combining the DONE signals means they cannot be used to indicate when configuration must transition to the next FPGA in the chain. Address decoding must be used instead to disable and enable the individual $\overline{CS}$ outputs.

In Figure 35, decoding logic presets the $\overline{CS}$ control register at Address 0 (sets all $\overline{CS}$ registers High in the beginning) and toggles the register at the starting and ending addresses for the desired configuration data. Set the starting address of the first register to 1h or it will not toggle. The starting address decode for one stage can also serve as the ending address decode of the previous stage.

Synchronizing startup in this manner simplifies the board connections and avoids contention during configuration. However, this also increases the size of the PROMMAP design. The starting and ending address values for each stage is easily determined after generating the final PROM file.

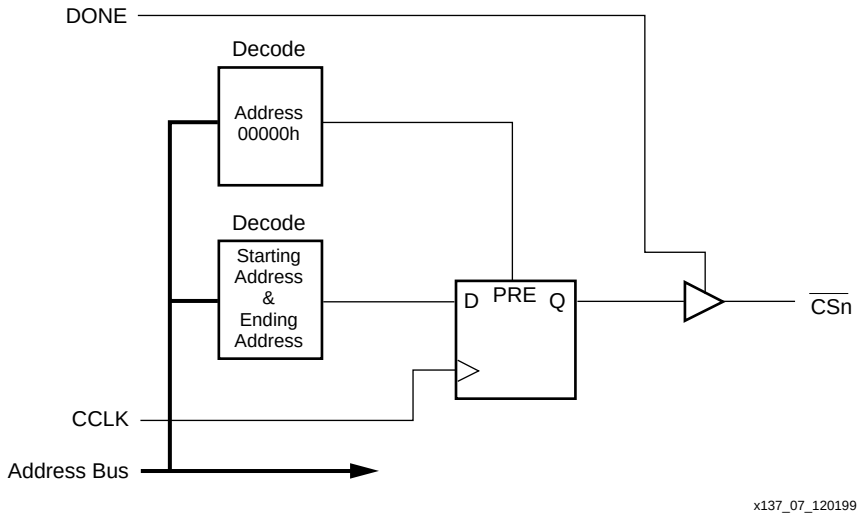

Figure 35: **$\overline{CS}$ Address Decoding**

Cascading PROM Files

PROMGen must be run on each FPGA design separately. The resulting PROM files must be concatenated, and it is important that the PROM files are in the desired order. Do not use PROMGen to create a single PROM file, since PROMGen is used for bit-serial daisy-chaining. If used, this generates an incorrect bitstream for this configuration scheme.

The PROM files for each of the FPGAs in the parallel-chain must be concatenated. This can be accomplished in a variety of ways depending on the programmer used.

If your programmer uses HEX files, a simple hex file concatenation is all that is needed. The PROMGen utility has a switch option to create HEX files.

If your PROM programmer software has edit ability, each PROM file and address can be read individually.

If the programmer requires a single PROM file in addressed format (MCS, TEK, EXO), some manual editing might be required. Note that a chain of Virtex PROM files is typically too large for MCS and TEK formats. EXO is a better choice and easier to use.

To concatenate EXO files, generate an EXO file for the first bit file in the chain with PROMGen loading from address 0h, and then examine the file. The last line and possibly the second-to-last line should be deleted. Before deleting the second-to-last line, note the starting address.

For example, the last two lines might read as follows.

```
S20803576000000003D
S804000000FB
```

This shows that data `00000000` is loaded starting at address `35760h`. This line is only four bytes or a partial line (a full line is 16 bytes) and should be deleted; the next PROM file generated should start at address `35760h`. Then the two files can be concatenated. For more stages to the chain, continue with this same process.

The four data bytes that are being removed from the PROM file are at the end of a 16-byte section of dummy data that represents the clock cycles required to complete the start-up sequence (actually, only 8 clock cycles are needed). Since a free running oscillator is used in this case, the unwanted dummy bits can be discarded.

Chapter 5

Advanced Architecture Configuration Techniques

Introduction

The Virtex™ architecture supports powerful new configuration modes, including partial reconfiguration. Such mechanisms are designed to give advanced applications access to and manipulation of on-chip data through configuration interfaces.

This chapter is an overview of the Virtex architecture, emphasizing data bit location in the configuration bitstream. Knowing bit locations is the basis for accessing and altering on-chip data. FPGA applications can be built that change or examine the functionality of the operating circuit without stopping the circuit being loaded in the device.

CLBs, IOBs, and Configurations

Each Virtex device contains configurable logic blocks (CLBs), input-output blocks (IOBs), block RAMs, clock resources, programmable routing, and configuration circuitry, Figure 36. These logic functions are configurable through the configuration bitstream. Configuration bitstreams contain a mix of commands and data. Configuration bitstreams can be read and written through one of the configuration interfaces on the device. The Virtex, Virtex-E, and Virtex-E Extended Memory (Virtex-EM) families differ primarily in the amount of block RAM available.

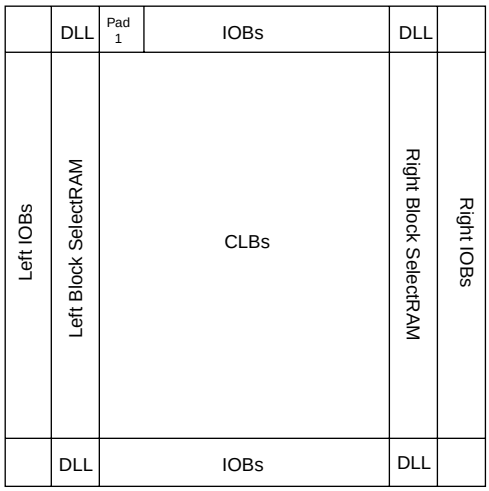

Figure 36: **Virtex Architecture Overview**

Configuring Virtex Devices

Virtex devices can be configured through the SelectMAP™ interface, master / slave serial interfaces, or the Boundary-Scan interface. The collection of configuration bits is called a configuration bitstream. The bitstream is a series of configuration commands and configuration data, as shown in Figure 37. Bitstream architecture is discussed in "Configuration Logic Basics" on page 96.

CMD 1	data	CMD 2	data	CMD 3	data	CMD 1

Figure 37: **Bitstream Example**

Warning! This document discusses mechanisms for manipulating the configuration bits for the Virtex devices. If portions of the bitstream other than those described here are altered, the device might be damaged. Xilinx is not liable for any consequences of erroneous device programming.

Writing some or all of a configuration is done by issuing configuration commands to the desired interface followed by the configuration data.

The SelectMAP interface is an 8-bit interface on the device with data pins labeled D[7:0]. The configuration bitstream can be written eight bits per clock cycle. Virtex devices can be configured to retain the (D[7:0], BUSY / DOUT, $\overline{\text{INIT}}$, $\overline{\text{WRITE}}$, and $\overline{\text{CS}}$) SelectMAP pins, allowing further reconfiguration via those pins. If further reconfiguration is not required, those pins can be configured as user I / O.

When the master / slave serial or Boundary Scan interface is used for configuration or reconfiguration, the configuration bitstream is transmitted one bit per clock cycle.

Timing relationships for the configuration interfaces are discussed in the Virtex series data sheets located at http://www.xilinx.com/products/virtex.htm.

Reading Configuration Bits From a Virtex Device

Configuration data can be read using the SelectMAP interface or the Boundary Scan interface. The master / slave serial interfaces can not be used to read a configuration. Reading all or some of a configuration is done by issuing configuration read commands to the desired interface, then reading the data from the same interface. Configuration commands and data formats are discussed in "Configuration Logic Basics" on page 96.

The SelectMAP interface has an 8-bit data port. To use the SelectMAP interface after configuration, all 12 SelectMAP pins must remain as SelectMAP as opposed to user I / O (using the BitGen option: `-g Persist:Yes`).

The Boundary Scan (JTAG) interface allows bit-serial access to the configuration. It is a permanent interface that is always present.

Configuration Columns

The Virtex configuration memory can be visualized as a rectangular array of bits. The bits are grouped into vertical *frames* that are one-bit wide and extend from the top of the array to the bottom. A frame is the atomic unit of configuration - it is the smallest portion of the configuration memory that can be written to or read from.

Frames are grouped together into larger units called columns. In Virtex, Virtex-E, and Virtex-EM devices, there are several different types of columns.

Table 29: **Configuration Column Type**

Column Type	No. of Frames	No. per Device
Center	8	1
CLB	48	No. of CLB columns
IOB	54	2
Block SelectRAM Interconnect	27	No. of Block SelectRAM columns
Block SelectRAM Content	64	No. of Block SelectRAM columns

Each device contains one center column that includes configuration for the four global clock pins. Two IOB columns represent configuration for all of the IOBs on the left and right edges of the device. The majority of columns are CLB columns which each contain one column of CLBs and the two IOBs above and below those CLBs. The remaining two column types involve the block RAM: one for content and the other for interconnect. For each RAM column, one of each type is present (for values for all Virtex devices, see Table 38 on page 90).

The columns for a sample Virtex device are shown below.

Table 30: **Configuration Column Example (XCV50)**

Left IOB Column (54 frames)	Block SelectRAM Content (64 frames)	Block SelectRAM Interconnect (27 frames)	2 IOBs	…	2 IOBs	2 GCLK	2 IOBs	…	2 IOBs	Block SelectRAM Interconnect(27 frames)	Block SelectRAM Content (64 frames)	Right IOB Column (54 frames)
			CLB Column (48 frames)		CLB Column (48 frames)	Center Column (8 frames)	CLB Column (48 frames)		CLB Column (48 frames)			
			2 IOBs		2 IOBs	2 GCLK	2 IOBs		2 IOBs			

Configuration Addressing

Block Type, Major Address, Minor Address

The total address space is divided into block types. There are two block types: RAM and CLB. The RAM type contains only the block SelectRAM content columns (not interconnect). The CLB type contains all other column types.

Both the RAM and CLB address spaces are subdivided into major and minor addresses. Each configuration column has a unique major address within the RAM or CLB space. Each configuration frame has a unique minor address within its column.

The numbering schemes for the block type and minor address are identical between Virtex, Virtex-E, and Virtex-EM devices. The major address numbering scheme differs between

families and is encapsulated in Table 31. But in both families, even major addresses are on the left half of the device while the odd major addresses are on the right half of the device.

Table 31: **Major Addressing Scheme by Family**

Column Type	Block Type	Virtex	Virtex-E	Virtex-EM
First MJA	CLB	0	0	0
	RAM	0	1	1
MJA Order	CLB	1: Center 2: CLB 3: IOB 4: BRAM Interconnect	1: Center 2: CLB / BRAM Interconnect 3: IOB	1: Center 2: CLB 3: IOB 4: BRAM Interconnect
	RAM	BRAM Content	BRAM Content	BRAM Content

Virtex Major Addresses

The CLB address space begins with "0" for the center column and alternates between the right and left halves of the device for all the CLB columns, then IOB columns, and finally block SelectRAM interconnect columns.

The RAM address space has "0" for the left block SelectRAM content column and "1" for the right column. An XCV50 is shown in Table 35. The shaded columns are in the RAM address space.

Table 32: **Virtex Family Allocation of Frames to Device Resources (XCV50)**

	Left IOB Column (54 frames)	Block SelectRAM Content (64 frames)	Block SelectRAM Interconnect (27 frames)	CLB Column (48 frames)	…	CLB Column (48 frames)	Center Column (8 frames)	CLB Column (48 frames)	…	CLB Column (48 frames)	Block SelectRAM Interconnect (27 frames)	Block SelectRAM Content (64 frames	Right IOB Column (54 frames)
				2 IOBs		2 IOBs	2 GCLK	2 IOBs		2 IOBs			
				2 IOBs		2 IOBs	2 GCLK	2 IOBs		2 IOBs			
	IO_{Left}	RAM0	BIC0	C1		C12	Center	C13		C24	BIC1	RAM1	IO_{Right}
M.A.	26	0	28	24		2	0	1		23	27	1	25

Virtex-EM Major Addresses

The Virtex-EM family has the same numbering as the Virtex family but the block SelectRAM content columns begin with "1".

Virtex-E Major Addresses

The Virtex-E family has block SelectRAM interspersed between CLB columns. Both the CLB and RAM major addressing have been changed to allow for this.

The CLB address space again begins with "0" for the center column and alternates between the right and left sides of the device for CLB columns *and* block SelectRAM interconnect

columns and finally IOB columns. The difference is that in Virtex-E devices, the block SelectRAM interconnect appears in the middle of the CLB addresses, while in the Virtex and Virtex-EM families, they appear at the very end after the IOB columns. Also, note that the block SelectRAM content columns begin with “1” (Virtex devices begin with “0”).

An XCV50E is shown in Table 33. The shaded columns are in the RAM address space.

Table 33: **Virtex-E Family: Allocation of Frames to Device Resources (for XCV50E)**

	Left IOB Column (54 frames)	Block SelectRAM Content (64 frames)	Block SelectRAM Interconnect (27 frames)	CLB Column (48 frames)		Block SelectRAM Content (64 frames)	Block SelectRAM Interconnect (27 frames)	CLB Column (48 frames)		Center Column (8 frames)		Right IOB Column
	2 IOBs			2 IOBs	…			2 IOBs	…	2 GCLK	…	2 IOBs
	2 IOBs			2 IOBs				2 IOBs		2 GCLK		2 IOBs
	IO_{Left}	RAM0	BIC0	C1		RAM1	BIC1	C7		C12	Center	C13
M.A.	30	4	28	26		2	14	12		2	0	1

	CLB18	BIC2	RAM2	C19		C24	BIC3	RAM3	IO_{Right}
	2 IOBs			2 IOBs	…	2 IOBs			
	2 IOBs			2 IOBs		2 IOBs			
	CLB Column (48 frames)	Block SelectRAM Interconnect (27 frames)	Block SelectRAM Content (64 frames)	CLB Column (48 frames)		CLB Column (48 frames)	Block SelectRAM	Block SelectRAM	
M.A.	11	13	1	15		25	27	3	29

Frames

Frames are read and written sequentially with ascending addresses for each operation. Multiple consecutive frames can be read or written with a single configuration command. The smallest amount of data that can be read or written with a single command is a single frame. The entire CLB array plus the IOBs and block SelectRAM interconnect can be read or written with a single command. Each block SelectRAM Content must be read or written separately.

Frame Sizes

Frame size depends on the number of rows in the device. The number of configuration bits in a frame is 18 x .(number of CLB_rows+2) and is padded with zeroes on the right (bottom) to fit in 32-bit words. See "Frame Organization" on page 86. for more details. An additional padding word is needed at the end of each frame for pipelining. Table 35 shows

the frame sizes for all Virtex devices. This table also shows the size, in words, of the bitstream for the CLB address space and the number of words in each RAM block.

Table 34: **Virtex and Virtex-E Family Members**

Device	Row × Col.	Bits/ Frame	Words/ Frame	Virtex			Virtex-E			
				RAM Cols	No. of 32-bit Read-back Words[1]		RAM Cols	RAM Space	No. of 32-bit Read-back Words	
					CLB (all)	RAM (1 col)			CLB (all)	RAM (1 col)
XCV50	16 x 24	384	12	2	15,876	780	-	-	-	-
XCV50E	16 x 24	384	12	-	-	-	4	6	16,524	780
XCV100	20 x 30	448	14	2	22,554	910	-	-	-	-
XCV100E	20 x 30	448	14	-	-	-	4	12	23,310	910
XCV150	24 x 36	512	16	2	30,384	1,040	-	-	-	-
XCV200	28 x 42	576	18	2	39,366	1,170	-	-	-	-
XCV200E	28 x 42	576	18	-	-	-	4	12	40,338	1,170
XCV300	32 x 48	672	21	2	51,975	1,365	-	-	-	-
XCV300E	32 x 48	672	21	-	-	-	4	12	53,109	1,365
XCV400	40 x 60	800	25	2	76,275	1,625	-	-	-	-
XCV400E	40 x 60	800	25	-	-	-	4	12	77,625	1,625
XCV405E	40 x 60	800	25	-	-	-	14	4	84,375	1,625
XCV600	48 x 72	960	30	2	108,810	1,950	-	-	-	-
XCV600E	48 x 72	960	30	-	-	-	6	12	112,050	1,950
XCV800	56 x 84	1088	34	2	142,902	2,210	-	-	-	-
XCV812E	56 x 84	1088	34	-	-	-	20	4	159,426	2,210
XCV1000	64 x 96	1248	39	2	186,381	2,535	-	-	-	-
XCV1000E	64 x 96	1248	39	-	-	-	6	12	190,593	2,535
XCV1600E	72 x 108	1376	43	-	-	-	8	12	237,231	2,795
XCV2000E	80 x 120	1536	48	-	-	-	8	12	292,464	3,120
XCV2600E	92 x 138	1728	54	-	-	-	8	12	375,678	3,510
XCV3200E	104 x 156	1952	61	-	-	-	8	12	477,081	3,965

Notes:
1. Includes pad frame in calculation.

Frame Organization

Each frame sits vertically in the device, with the "front" of the frame at the top. As shown in Figure 38, it is convenient to consider the frame horizontally when it is viewed as part of a bitstream. The top IOBs are shown on the left followed by the CLBs in the column and the bottom IOBs on the right. Bits in frames are allocated as follows.

For CLB columns, the first 18 bits control the two IOBs at the top of the column; then 18 bits are allocated for each CLB row; finally, the next 18 bits control the two IOBs at the bottom

of the CLB column. The frame then contains enough "pad" bits to make it an integral multiple of 32 bits.

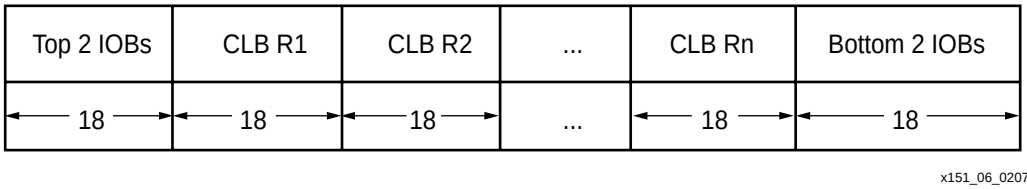

Figure 38: **CLB Column Frame Organization**

For IOB columns, the frame allocates 18 bits per three IOB rows, Figure 40. When reading and writing frames, bits are grouped into 32-bit words, starting on the left (corresponding to the top of the device). If the last word does not completely fill a 32-bit word, it is padded on the right with zeroes.

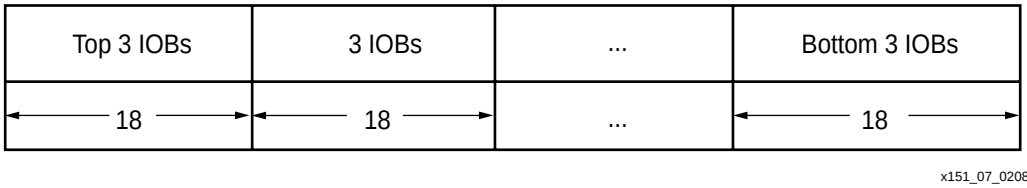

Figure 39: **IOB Column Frame Organization**

For block SelectRAM content columns (Figure 40), the first 18 bits are pad bits; then 72 bits are allocated for each RAM row; finally, there are another 18 pad bits. The frame then adds enough "pad" bits to make it an integral multiple of 32 bits.

Figure 40: **Block SelectRAM Content Column Frame Organization**

Location of LUT SelectRAM Bits

With respect to the beginning of a configuration frame, relative locations of LUT SelectRAM bits within the bitstream are the same for every CLB slice. The bits of a LUT SelectRAM are spread across 16 consecutive frame Minor Addresses. Each frame Minor Address contains all instances of a single bit index for that column. These 16 frames contain all 16 bits of the LUT SelectRAM for a column of CLB slices. You must read or write the 16 frames containing those bits to read or write the entire LUT SelectRAM.

Notes:

1. LUT SelectRAM bits are stored inverted. Flip-Flop data are stored in their true sense. When reading or writing LUT SelectRAM data from the bitstream, it is negated from the logic sense of the data. For example, a 4-input AND gate has the truth table LUT[15:0] = 1000000000000000.
2. This truth table is stored internally in the LUT SelectRAM as LUT[15:0] = 0111111111111111. Of course, user logic reading the data from the LUT SelectRAM would read the correct logic value, LUT[15:0] = 1000000000000000.

Configuration Data Operations

Reading Configuration Data

Configuration information can be read from the Virtex devices if readback is enabled in the current configuration. (See the description of the Control Register Field SBITS, "Control Register (CTL)" on page 102.) CLB and IOB configuration data can be read while the device is operating.

When reading data from a Virtex device, a pad frame is read before any of the data frames. This is in addition to the pad word in each data frame. The pad frame must be included in the word count to be read from the device.

Each frame read from the device consists of the number of words shown in Table 35.

For example, the XCV150 has 16-word frames. When reading frames, the first word of a frame is a pad word. Including the pad frame, the XCV150 has 1,899 (30,384 / 16) frames.

Table 35: **XCV150 Frame Padding for Device Read**

Pad Frame (16 words)	
Pad Word	Data Frame 0 (15 words)
⋮	⋮
Pad Word	Data Frame n (15 words)

Writing Configuration Data

Configuration information can be written to a Virtex device while the device is running if writing is enabled in the current configuration. (See the Control Register field SBITS in Table 60 on page 102.) Rewriting the same configuration data does not generate any transient signals.

Changing configuration data might generate transient signals, especially if LUT values or signal routing is changed. For this case, all the logic cells and routing can be placed in a non-contentious state by asserting the GHIGH_B signal. See the description for the "Command Register (CMD)" on page 99,

When writing configuration data to the Virtex device, whether from the SelectMAP or JTAG interfaces, the data frames are written to the device with each frame followed by a pad word. After all the data frames are written, a pad frame must be written (to flush internal pipelines). The pad words and pad frame must be included in the number of words to be written to the device. (Table 34 on page 86.)

For example, the XCV1000 has 39-word frames. When writing frames, the last word of the frame is a pad word. Including the pad frame, the XCV1000 has 4,779 (186,381 / 39) frames. See Table 36.

Table 36: **XCV150 Frame Padding for Device Read**

Data Frame 0 (38 words)	Pad Word
⋮	
Data Frame n (38 words)	Pad Word
Pad Frame (39 words)	

Altering Configuration Data

Virtex devices support the Read-Modify-Write (RMW) method of changing LUT SelectRAM data. Therefore, it is possible to read or alter the contents of one or more LUT SelectRAMs through the JTAG or SelectMAP interfaces. *When writing data to one or more*

*LUT SelectRAMs, all the bits in the frame **must** have valid configuration information.* This can be assured by altering valid configurations from bitstream files or from frames read from a properly configured Virtex device. When using the RMW method, it can be expeditious to read the data, ignoring the pad frame and the pad word of only the first frame. This aligns the remaining configuration data in the Device Write format. After modifying the configuration, the altered data can be written to the Virtex device followed by a pad word and a pad frame.

Notes:

1. Frames span an entire column of CLB slices (or IOBs). Thus, when changing LUT SelectRAM bit 0 for a single CLB slice (*e.g.*, R3C4.S1), LUT SelectRAM bit 0 for *all* slices in that column (*i.e.*, R*C4.S1) is written with the same command. One must ensure that either all other data in the slice is constant or changed externally through partial configurations.
2. LUT SelectRAMs not configured externally should not lie in the same slice with LUTs or LUT SelectRAMs that are re-configured externally. Such a mixture can cause the unrelated LUT SelectRAM data to be re-configured when the frames are written to the device. Figure 42 on page 103 shows what can happen if this restriction is not observed.
3. In this example, it is the objective to perform a Read-Modify-Write on the LUT SelectRAM in R2C3.S1 in column 3, which is shown in the first column in the figure. Row 2 contains a LUT containing the value AB. Row 3 contains a SelectRAM containing the value C3. Because the Read and Write operations operate on an entire frame, the RAM in R3C3.S1 is also read and written when R2C3.S1 is read and written. Performing the Readback operation reads all the LUT and SelectRAM values for the column. Before the new value for the LUT is written, it is possible for the on-chip circuitry to write a new value, such as 14, into the SelectRAM. When the new value, BD, is written via the configuration interface into LUT R2C3.S1, the value C3 is also written into RAM R3C3.S1. This might not always be desirable (It is design-dependent).

It is not necessary to retain the contents of the pad frame or pad words. However, pad words and pad frames *are* included in the CRC calculation.

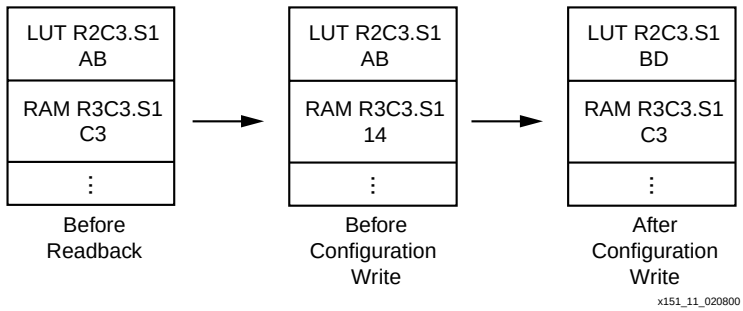

Figure 41: **Potential Write Conflicts**

Definitions

Two sets of variables are defined to determine where a desired bit is in the configuration data. The first is a set of “independent” variables, or design attributes, namely characteristics of the design that are known, that is, the device size and which CLB and bit(s) within the CLB to locate, Table 38. The second set is a set of “dependent” variables or

design variables, namely values that must be calculated to find the bit(s) of interest listed in Table 39. The functions used in equations are found in Table 39.

Table 37: **Design Attributes (Independent Variables)**

Term	Definition
Chip_Cols	No. of CLB columns on the Virtex device
Chip_Rows	No. of CLB rows on the Virtex device
Chip_Rams	No. of block SelectRAM columns on the Virtex device
RAM_Space	Spacing of block SelectRAM columns (in terms of CLB columns)
FL	No. of 32-bit words in the frame.
RW	1 for Read, 0 for Write
CLB_Col	Column number of the desired CLB
CLB_Row	Row number of the desired CLB
Slice	0 or 1
FG	0 for the F-LUT, 1 for the G-LUT
lut_bit	The desired bit from the given LUT. Bits in the LUT are indexed from 0 to 15.
XY	0 for the X Flip-Flop, 1 for the Y Flip-Flop
RAM_Col	Column number of the desired block SelectRAM
RAM_Row	Row number of the desired block SelectRAM
ram_bit	The desired bit from the given block SelectRAM. Bits are indexed from 0 to 4095.

Table 38: **Design Variables (Dependent Variables)**

Term	Definition
MJA	Frame Major Address
MNA	Frame Minor Address
fm_st_wd	The index of the word within a full configuration segment that corresponds to the starting word of the desired frame. A full configuration segment is defined as the following: 1) for CLB/IOB, all CLB, IOB, and RAM interconnect frames beginning at MJA=0, MNA=0 and 2) for block SelectRAM, all RAM content frames for the given RAM column. Words are numbered starting at 0.
fm_wd	The index of the 32-bit word within a frame that contains the desired bit. Words in a frame are numbered starting at 0.
fm_wd_bit_idx	The bit index of the desired bit within frame word fm_wd. Words are indexed in “big-endian” style, with bit 31 on the left and bit 0 on the right.
fm_bit_idx	Bit index within a frame of the desired bit. Numbered starting with 0 as the left-most (first) bit. Bit numbering within a frame continues across all the words in the frame.

Table 39: **Functions Used in Equations**

Functions	Definition
floor(x)	The largest integer not larger than *x*. *E.g.*, floor (3.1) = 3, because the next largest integer, 4, is larger than 3.1, floor (3) =3.
ceiling(x)	The smallest integer greater than or equal to *x*. For example, ceiling (3.2) = 4, because 4 is greater than 3.2, and 3 is not, ceiling (4) = 4.
%	The modulus operation. 5% 2 = 1, 5% 3 = 2, 5% 5 = 0.

CLB LUT SelectRAM Dependent Variables

Table 40 shows equations for the LUT SelectRAM Dependent Variables that were defined in Table 38 on page 90.

Table 40: **Virtex Equations for LUT SelectRAM Dependent Variables**

Term	Definition
MJA	if (CLB_Col ≤ Chip_Cols / 2), then Chip_Cols – CLB_Col x 2 + 2 else 2 x CLB_Col – Chip_Cols – 1
MNA	lut_bit + 32 – Slice x (2 x lut_bit + 17)
fm_bit_idx	3 + 18 x CLB_Row – FG + RW x 32
fm_st_wd	FL x (8 + (MJA –1) x 48 + MNA) + RW x FL
fm_wd	floor(fm_bit_idx / 32)
fm_wd_bit_idx	31 + 32 x fm_wd – fm_bit_idx

Virtex-E CLB LUT SelectRAM Dependent Variables

The additional block SelectRAM columns in Virtex-E devices require an adjustment to the MJA. The following equations in Table 41 calculate the adjustment necessary for each of the affected dependent variables. For each variable, calculate the base value based upon the Virtex equations only and then add the resulting Virtex adjustment.

Notes:
1. Do NOT input the final results back into the original Virtex equations (e.g., do not use MJA_{final} value to calculate fm_st_wd).
2. MJA_{final} = MJA + MJA_adj
3. $fm\_st\_wd_{final}$ = fm_st_wd + fm_st_wd_adj

Table 41: **Virtex-E Families Adjustments for LUT SelectRAM Dependent Variables**

Term		Definition
MJA_adj	Left	RAM_Bound = (Chip_Rams / 2 - 1) x RAM_Space MJA_adj = 2 x ceiling((RAM_Bound - CLB_Col + 1) / RAM_Space)
	Right	RAM_Bound = Chip_Cols - (Chip_Rams / 2 - 1) x RAM_Space + 1 MJA_adj = 2 x ceiling((CLB_Col - RAM_Bound + 1) / RAM_Space)
fm_st_wd_adj		FL x (27 x MJA_adj)

The left and right sides of the device are treated differently as shown in Table 42:

Table 42: **CLB Location**

CLB Location	CLB Column
Left	CLB_Col ≤ Chip_Cols / 2
Right	CLB_Col > Chip_Cols / 2

LUT SelectRAM Examples

See "Examples" on page 108. for several examples of reading and evaluating configuration data. The examples illustrate how to make use of these equations to find the desired data in a bitstream.

CLB Flip-Flop Dependent Variables

Equations for the CLB flip-flops can be found in Table 44. Their locations are calculated similarly to the LUT SelectRAM locations. Equations for CLB FF Dependent Variables are defined in Table 38 on page 90.

Table 43: **Virtex Equations for CLB FF Dependent Variables**

Term	Definition
MJA	if (CLB_Col ≤ Chip_Cols / 2) then Chip_Cols – CLB_Col x 2 + 2 else 2 x CLB_Col – Chip_Cols – 1
MNA	Slice x (12 x XY – 43) – 6 x XY + 45
fm_bit_idx	(18 x CLB_Row) + 1 + (32 x RW)
fm_st_wd	FL x (8 + (MJA –1) x 48 + MNA) + RW x FL
fm_wd	floor(fm_bit_idx / 32)
fm_wd_bit_idx	31 + 32 x fm_wd – fm_bit_idx

Virtex-E CLB Flip-Flop Dependent Variables

Apply the adjustments given in Table 42 and Table 43.

IOB Dependent Variables

Each IOB contains four values that can be captured into special registers. These values are:

- I — the input flip-flop
- O — the output flip-flop
- T — the flip-flop for the 3-state control
- P — the value of the I/O pad

These values are captured by utilizing the CAPTURE_VIRTEX symbol in your design. The Libraries Guide has more details on the use of this symbol. The following registers will be read as part of the readback data.

Access to the IOB flip-flops is different for the top and bottom IOBs versus the left and right IOBs.

The top and bottom IOBs are part of the CLB column frames. There are two IOBs at the top and bottom of each CLB column.

The left and right IOBs are in columns by themselves. There are three IOBs per CLB row.

IOBs are numbered clockwise around the die. Pad 1 is located at the left side of the top edge, above CLB column 1. The equations for where to find IOB flip-flops in the bitstream are based on the pad number which is the same for a given size device, not the package pin name, which varies from package to package. The mapping from package pin names to pad numbers can be found in EPIC or fpga_editor.

Table 44 contains the numeric pad indices for the pads on all four edges of the device in terms of the number of CLB columns and rows on the device.

Table 44: **IOB Pad Indices**

Pad Location	Pad Index i
Top	$1 \le i \le Chip\_Cols \times 2$
Right	$Chip\_Cols \times 2 + 1 \le i \le Chip\_Cols \times 2 + Chip\_Rows \times 3$
Bottom	$Chip\_Cols \times 2 + Chip\_Rows \times 3 + 1 \le i \le Chip\_Cols \times 4 + Chip\_Rows \times 3$
Left	$Chip\_Cols \times 4 + Chip\_Rows \times 3 + 1 \le i \le Chip\_Cols \times 4 + Chip\_Rows \times 6$

Table 45 shows the equations for the dependent variables for the IOB flip-flops. The variable ***i*** in this table refers to the index of pad ***i***

Table 45: **Equations for IOB Dependent Variables**

Term			Definition
MJA	Top		if $(i \le Chip\_Cols)$ then $Chip\_Cols - ceiling(i/2) \times 2 + 2$ else $2 \times ceiling(i/2) - Chip\_Cols - 1$
	Right		$Chip\_Cols + 1$
	Bottom		if $(i > 3 \times (Chip\_Cols + Chip\_Rows))$ then $2 \times ceiling((i - 3 \times Chip\_Cols - 3 \times Chip\_Rows)/2)$ else $Chip\_Cols - 2 \times floor((i - 2 \times Chip\_Cols - 3 \times Chip\_Rows - 1)/2) - 1$
	Left		$Chip\_Cols + 2$

Table 45: **Equations for IOB Dependent Variables *(Continued)***

Term			Definition
MNA	Top	I	$-25 \times (i\%2) + 45$
		O	$-13 \times (i\%2) + 39$
		T	$-5 \times (i\%2) + 35$
		P	$-4 \times (i\%2) + 25$
	Right	I	$t = (i - 2 \times Chip\_Cols)\ \%\ 3;\ MNA = 27.5 \times t^2 - 57.5 \times t + 32$
		O	$t = (i - 2 \times Chip\_Cols)\ \%\ 3;\ MNA = 21.5 \times t^2 - 51.5 \times t + 38$
		T	$t = (i - 2 \times Chip\_Cols)\ \%\ 3;\ MNA = 17.5 \times t^2 - 47.5 \times t + 42$
		P	50
	Bottom	I	$25 \times ((i - 2 \times Chip\_Cols - 3 \times Chip\_Rows)\%2) + 20$
		O	$13 \times ((i - 2 \times Chip\_Cols - 3 \times Chip\_Rows)\%2) + 26$
		T	$5 \times ((i - 2 \times Chip\_Cols - 3 \times Chip\_Rows)\%2) + 30$
		P	$4 \times ((i - 2 \times Chip\_Cols - 3 \times Chip\_Rows)\%2) + 21$
	Left	I	$t = (i - 4 \times Chip\_Cols - 3 \times Chip\_Rows)\%3;\ MNA = 17.5 \times t^2 - 47.5 \times t + 45$
		O	$t = (i - 4 \times Chip\_Cols - 3 \times Chip\_Rows)\%3;\ MNA = 23.5 \times t^2 - 53.5 \times t + 39$
		T	$t = (i - 4 \times Chip\_Cols - 3 \times Chip\_Rows)\%3;\ MNA = 27.5 \times t^2 - 57.5 \times t + 35$
		P	50
fm_bit_idx	Top		$32 \times RW$
	Right	I, O, &T	$18 \times (1 + floor\ ((i - 2 \times Chip\_Cols - 1)/3)) + 32 \times RW$
		P	$t = (i - 2 \times Chip\_Cols)\ \%\ 3;$ $fm\_bit\_idx = 18 \times (1 + floor\ ((i - 2 \times Chip\_Cols - 1)/3)) + 6 \times t^2 - 17 \times t + 15 + 32 \times RW$
	Bottom		$18 \times (Chip\_Rows + 1) + 32 \times RW$
	Left	I, O, &T	$18 \times (Chip\_Rows - floor($ $(i - 4 \times Chip\_Cols - 3 \times Chip\_Rows - 1)/3)) + 32 \times RW$
		P	$t = (i - 4 \times Chip\_Cols - 3 \times Chip\_Rows)\%3;$ $fm\_bit\_idx = 18 \times (Chip\_Rows - floor\ ((i - 4 \times Chip\_Cols - 3 \times Chip\_Rows - 1)/3))$ $- 10.5 \times t^2 + 21.5 \times t + 4 + 32 \times RW$
fm_st_wd			if (MJA > Chip_Cols + 1) then $FL \times (54 \times MJA - 46 + MNA - 6 \times Chip\_Cols) + RW \times FL$ else $FL \times (8 + (MJA - 1) \times 48 + MNA) + RW \times FL$
fm_wd			$floor\ (fm\_bit\_idx/32)$
fm_wd_bit_idx			$31 + 32 \times fm\_wd - fm\_bit\_idx$

Virtex-E IOB Dependent Variables

Similar to CLB dependent variables, only the variables affected by the major address are different in Virtex-E and Virtex-EM devices versus Virtex devices. A simple way to account for the change is to determine the CLB column where the pad resides (only true for

top and bottom). Then, the adjustment calculated for the CLB column can be applied. The conversion to CLB columns is found in Table 46. Adjustments are listed in Table 47.

Table 46: **Pad -> CLB Column**

Term		Definition
CLB_Col	Top	ceiling(i / 2)
	Bottom	ceiling((4 x Chip_Cols + 3 x Chip_Rows + 1 - i) / 2)

Table 47: **Virtex-E Families Adjustments for IOB Dependent Variables**

Term		Definition
MJA_adj	Top	use Table 41 and Table 42
	Right	Chip_Rams
	Bottom	use Table 41 and Table 42
	Left	Chip_Rams
fm_st_wd_adj		FL x (27 x MJA_adj)

Block SelectRAM Dependent Variables

The equations for the block SelectRAM dependent variables are given in Table 49. The relationship of a particular memory cell index in the context of a given configuration is described in **XAPP130 "Using the Virtex Block SelectRAM+ Resource".**

Table 48: **Virtex Equations for Block SelectRAM Dependent Variables**

Term	Definition
MJA	if (RAM_Col < Chip_Rams / 2) then 2 x (Chip_Rams / 2 - 1 - RAM_Col) else 2 x (RAM_Col - Chip_Rams / 2) + 1
MNA	1 x floor(((ram_bit / 64)% 64) / 32) + 2 x floor(((ram_bit / 64)% 32) / 16) + 4 x floor(((ram_bit / 64)% 16) / 8) + 8 x floor(((ram_bit / 64)% 8) / 4) + 16 x floor(((ram_bit / 64)% 4) / 2) + 32 x floor(((ram_bit / 64)% 2) / 1) equivalent to MNA = div64[0:5] where div64[5:0] = floor(ram_bit / 64)
fm_bit_id	obtain value for bitpos from Table 49. fm_bit_idx = 18 + 72 x RAM_Row + bitpos
fm_st_wd	FL x MNA + RW x FL
fm_wd	floor(fm_bit_idx / 32)
fm_wd_bit_idx	31 + 32 x fm_wd – fm_bit_idx

Table 49: **Virtex Block SelectRAM Bit Position Within a Given Block SelectRAM**

ram_bit % 64	bitpos	ram_bit % 64	bitpos	ram_bit % 64	bitpos	ram_bit % 64	bitpos	ram_bit % 64	bitpos	ram_bit % 64	bitpos	ram_bit % 64	bitpos	ram_bit % 64	bitpos
0	42	8	45	16	29	24	26	32	43	40	44	48	28	56	27
1	58	9	61	17	13	25	10	33	59	41	60	49	12	57	11
2	41	10	46	18	30	26	25	34	40	42	47	50	31	58	24
3	57	11	62	19	14	27	9	35	56	43	63	51	15	59	8
4	50	12	53	20	21	28	18	36	51	44	52	52	20	60	19
5	49	13	54	21	22	29	17	37	48	45	55	53	23	61	16
6	66	14	69	22	5	30	2	38	67	46	68	54	4	62	3
7	65	15	70	23	6	31	1	39	64	47	71	55	7	63	0

Virtex-E Block SelectRAM Dependent Variables

The RAM major address numbering scheme changed slightly such that the lowest MJA on the left side is now “2” instead of “0”. As in the case for the CLB equations, the adjustments in Table 50 should be applied after the Virtex equations.

Table 50: **Virtex-E and Virtex-EM Adjustments for Block SelectRAM Dependent Variables**

Term	Definition
MJA_adj	if (RAM_Col < Chip_Rams/2) MJA_adj = 2

Configuration Logic Basics

Configuration Data

Configuration data is organized as 32-bit words. There are two major commands that the configuration data can contain; Read and Write. A configuration command is executed when the configuration command is read or written to the appropriate command register. The OP field contains 01 for a Read operation and 10 for a Write operation. (See Table 51).

Table 51: **OP Field Code**

OP	CODE
Read	01
Write	10

A command is organized as a packet with a header word and optional data words. The header word is the first word written to the appropriate command register for a read or write operation. The header word contains a type field (001), an operand field, a register address field, and a word count field. The format for the command header is shown in Table 52.

Table 52: **Command Header Format**

Type				OP				Register Address												RSV	Word Count										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	x	x	0	0	0	0	0	0	0	0	0	0	x	x	x	x	0	0	x	x	x	x	x	x	x	x	x	x	x

Notes:
1. Locations within fields containing a zero or one must have these values. An X in a bit field indicates that the value is variable and must be set.
2. Heavy vertical lines are used to separate fields. Light vertical lines separate nibbles in the word.

The Register Address field defines the target of this command, as defined in Table 54 on page 98.

The header word count field contains an integer between 0 and 2,047 and indicates the number of words that follow the header. Larger word counts (between 2,048 and 1,048,575 words) are achieved by setting the header word count to 0. The Extension header word has a type field = 010, an OP field that must match the OP field in the preceding Command Header word, and a 20-bit word count, as shown Table 53.

Table 53: **Large Block Count Header Extension Format**

Type				OP				Word Count																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	x	x	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Notes:
1. Locations within fields containing a zero or one must have these values. An X in a bit field indicates that the value is variable and must be set.
2. Heavy vertical lines are used to separate fields. Light vertical lines separate nibbles in the word.

Configuration Flow

Virtex devices are configured by presenting configuration data to the SelectMAP or JTAG interfaces in a specific sequence.

For initial configuration, sequence is as follows:

1. Issue one or more pad words (SelectMAP only).
2. Issue Sync word (SelectMAP only).
3. Reset CRC.
4. Set FLR.
5. Set COR.
6. Set MASK.
7. Set CTL. (Set PERSIST if you want to keep SelectMAP active after this configuration.)
8. Issue a SWITCH (switch CCLK frequency) command to the CMD register. This is necessary only in Master Serial configuration mode.

Reading Configuration

These commands can be issued to read a full configuration or for a partial configuration after the device has been completely configured.

1. Issue a Sync word (SelectMAP only) if the previous configuration command was aborted.
2. Set the FAR to the starting address.
3. Issue a RCFG (read configuration) command to the CMD register.

4. Write the number of words to be read to the FDRO register.
5. Flush the command pipeline with a pad word.
6. Read data frames.

Writing Configuration

These commands can be issued either as part of an initial configuration or for a partial configuration after the device has been configured.

1. Issue a Sync word (SelectMAP only) if the previous configuration command was aborted.
2. If the frames being written can cause contention, then assert the GHIGH_B signal.
3. Set the FAR to the starting address.
4. Issue a WCFG (write configuration) command to the CMD register.
5. Write the number of words to be written to the FDRI register.
6. Write data frames.
7. If the GHIGH_B signal was asserted, de-assert it by writing the LFRM (Last Frame) command to the CMD register and write one pad frame.

Configuration Registers

Configuration logic is accessed and controlled via a collection of 32-bit registers called the configuration registers (see Table 54). Registers are described in the following sections.

Table 54: **Configuration Register Addresses**

Register Name	Mnemonic	R/W	Binary Address
CRC	CRC	R/W	0000
Frame Address	FAR	R/W	0001
Frame Data Input	FDRI	W	0010
Frame Data Output	FDRO	R	0011
Command	CMD	R/W	0100
Control	CTL	R/W	0101
Control Mask	MASK	R/W	0110
Status	STAT	R	0111
Legacy Output	LOUT	W	1000
Configuration Option	COR	R/W	1001
Reserved	—	—	1010
Frame Length	FLR	R/W	1011
Reserved	—	—	1100
Reserved	—	—	1101
Reserved	—	—	1110
Reserved	—	—	1111

Command Register (CMD)

The content of the Command Register (CMD) is interpreted by the configuration state machine. Configuration commands control the operation of the configuration state machine, the Frame Data Register (FDR), and some of the global signals. The command in the Command Register is executed each time the FAR is loaded with a new value. The effect of each command is defined in Table 55.

Table 55: **Configuration Commands**

Cmd	Code	Description
Rsvd	0000	Reserved
WCFG	0001	**Write Configuration Data —** Used prior to writing configuration data to the FDRI. It takes the internal configuration state machine through a sequence of states that control the shifting of the FDR and the writing of the configuration memory. (See "Frame Data Input Register (FDRI)" on page 104).
Rsvd	0010	Reserved
LFRM	0011	**Last Frame —** This command is loaded prior to writing the last (pad) data frame if the GHIGH_B signal was asserted. This command is not necessary if the GHIGH_B signal was not asserted. This allows overlap of the last frame write with the release of the GHIGH_B signal.
RCFG	0100	**Read Configuration Data —** Used prior to reading frame data from the FDRO. Similar to the WCFG command in its effect on the FDR (see "Frame Data Output Register (FDRO)" on page 104).
START	0101	**Begin Start-Up Sequence —** Starts the start-up sequence. This command is also used to start a shutdown sequence prior to partial reconfiguration. The Start-Up Sequence begins with the next successful CRC check (see "Cyclic Redundancy Check (CRC)" on page 102).
RCAP	0110	**Reset Capture —** Used when performing capture in single-shot mode. This command must be used to reset the capture signal if single-shot capture has been selected.
RCRC	0111	**Reset CRC —** Used to reset CRC register (see "Cyclic Redundancy Check (CRC)" on page 102).
AGHIGH	1000	**Assert GHIGH_B Signal —** Used prior to reconfiguration to prevent contention while writing new configuration data. All CLB outputs and signals are forced to a one.
SWITCH	1001	**Switch CCLK Frequency —** Used to change (increase) the frequency of the Master CCLK. The new frequency is specified in Table 58 on page 101.
Rsvd	1010	Reserved
Rsvd	1011	Reserved
Rsvd	1100	Reserved
Rsvd	1101	Reserved
Rsvd	1110	Reserved
Rsvd	1111	Reserved

Configuration Option Register (COR)

The Configuration Option Register (COR) is used to select configuration options that are illustrated in Table 56. Entries in this table are further defined and explained in Table 57.

Table 56: **COR Fields**

	DONE_PIPE	DRIVE_DONE	SINGLE	OSCFSEL						SSCLKSRC		LOCK_WAIT				SHUTDOWN	DONE_CYCLE			LCK_CYCLE			GTS_CYCLE			GWE_CYCLE			GSR_CYCLE		
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Notes:
1. Locations within fields containing a zero or one must have these values. An *X* in a bit field indicates that the value is variable and must be set.
2. Heavy vertical lines are used to separate fields. Light vertical lines separate nibbles in the word.

Table 57: **Configuration Option Register Fields**

Field	Bit Indices	Description	BitGen Default
DONE_PIPE	30	0: No pipeline stage for DONEIN. 1: Add pipeline stage to DONEIN. FPGA waits on DONE that is delayed by one (1) cycle of the StartupClk rather than the pin itself. Use this option when StartupClk is running at high speeds.	0
DRIVE_DONE	29	0: DONE pin is open drain. 1: DONE pin is actively driven high.	0
SINGLE	28	Readback capture is one-shot.	0
OSCFSEL	27:22	Select CCLK frequency in Master Serial configuration mode.	2
SSCLKSRC	21:20	Startup sequence clock source 00: Cclk 01: UserClk 1x: JTAGClk	0
LOCK_WAIT	19:16	4-bit mask indicating which DLL lock signals to wait for during LCK cycle. The 4 bits (from MSB to LSB) correspond to DLLs TL=3, TR=2, BL=1, BR=0 (top-left, top-right, etc.). In Virtex-E devices where there are 8 DLLs, each mask bit applies to the 2 DLLs in that quadrant of the device. The default is not to wait for any DLL lock signals.	0
SHUTDOWN	15	Indicate whether doing a Start-Up or Shutdown sequence. 0: Start-Up 1: Shutdown sequence	0
DONE_CYCLE	14:12	Start-up phase in which DONE pin is released	3
LCK_CYCLE	11:9	Stall in this start-up phase until DLL locks are asserted.	7
GTS_CYCLE	8:6	Start-up phase in which I/Os switch from 3-state to user design	4
GWE_CYCLE	5:3	Start-up phase in which the global write-enable is asserted	5
GSR_CYCLE	2:0	Start-up phase in which the global set/reset is negated	5

Table 58 shows the allowed values for the OSCFSEL field of the COR. Setting OSCFSEL to one of these values will set the Master CCLK frequency to the specified value.

Table 58: **OSCFSEL-Specified Master CCLK Frequencies**

CCLK (MHz)	OSCFSEL	CCLK (MHz)	OSCFSEL	CCLK (MHz)	OSCFSEL
4.3	000010	13	001010	41	100111
5.4	010001	15	001101	45	110011
6.9	000100	20	010111	51	101010
8.1	000101	26	011010	55	110100
9.2	000110	30	011101	60	101101
10.0	000111	34	110010	—	—

Notes:
1. These values are accurate to +45%, – 30%.

Table 59 shows the values of the DONE_CYCLE, LCK_CYCLE, GTS_CYCLE, GWE_CYCLE, and GSR_CYCLE fields in COR. This table shows the step in the start-up sequence when each of these signals becomes active.

Table 59: **COR Start-Up Cycle Fields**

Field Value	DONE_CYCLE (DONE Active)	GTS_CYCLE (GTS_CFG Inactive)	GSR_CYCLE (GSR Inactive)	GWE_CYCLE (GWE Active)	LCK_CYCLE
000	1	1	1	1	0
001	2	2	2	2	1
010	3	3	3	3	2
011	4 (default)	4	4	4	3
100	5	5 (default)	5	5	4
101	6	6	6 (default)	6 (default)	5
110	—	DoneIn†	DoneIn†	DoneIn†	6
111	Keep State	Keep State	Keep State	Keep State	Don't Wait (default)

Notes:
1. † DONE if DonePipe = No, else the delayed version of DONE.

Control Register (CTL)

The Control Register (CTL) fields are defined in Table 60. Control Register bits are given in Table 60.

Table 60: **Control Register Fields**

																						SBITS		PERSIST						GTS_USR_B
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	0	0	0	0	0	x

Notes:

1. Locations within fields containing a zero or one must have these values. An *X* in a bit field indicates that the value is variable and must be set.
2. Heavy vertical lines are used to separate fields. Light vertical lines separate nibbles in the word.

Table 61: **Control Register Bits**

Name	Bit Indices	Description
SBITS	8:7	Security level: 0: Read / Write OK (default) 1: Readback disabled. 2,3: Readback disabled, Writing disabled except CRC register.
PERSIST	6	Configuration interface remains after configuration. 0: No (default) 1: Yes
GTS_USR_B	0	Active-low global 3-state I/Os. Turn off pull-ups if GTS_CFG_B is also asserted.

Cyclic Redundancy Check (CRC)

A data input error checking mechanism is provided through the Cyclic Redundancy Check (CRC) register. When data is written to any configuration register (except LOUT) a 16-bit CRC value is calculated using both the register data and the address. This value is saved in the CRC register. At the end of any series of writes a pre-calculated CRC block-check value can be written to the CRC register. If the resulting value is non-zero, an error is indicated. The CRC_ERROR bit is accessible through the status register. If a CRC error is detected, configuration logic is put in the ERROR mode. The following section is an algorithm for computing CRC.

CRC Algorithm

```
/* Initialization */
bcc = 0;
skip_pad = true;
more_words = true;
/* Check for write operation. */
do {
  w = next_word;
  if (w[31:27] == '00101') {
    /* A Read OP. Don't use in CRC*/
    wc = w[10:0];
```

```
      if (wc == 0) {
        w = next_word;
        wc = w[19:0];
      }
      while (wc--  > 0) {
        w = next_word;
      }
    }
    elsif (w[31:27] == '00110') {
      /* A Write OP. Use in CRC. */
      addr = w[16:13];
      if (addr Œ {0,1,2,4,5,6,9,D,B}) {
        wc = w[10:0];
        if (wc == 0) {
            /* wc is in next word. */
          w = next_word;
          wc = w[19:0];
        }
        while (wc--  > 0) {
          w = next_word;
          sw[35:0] = addr, word;
          for (i=0; i<36; i++) {
            x16 = bcc[15] XOR sw[i];
            x15 = bcc[14] XOR x16;
            x2 = bcc[1] XOR x16;
            bcc[15:0] =
                x15,bcc[13:2],x2,bcc[0],x16;
          }
          /* Note the bit order */
          crc[15:0] = bcc[0:15];
        }
      }
    }
    else {
      /* Pad word - ignore */
    }
  } while (more_words)
```

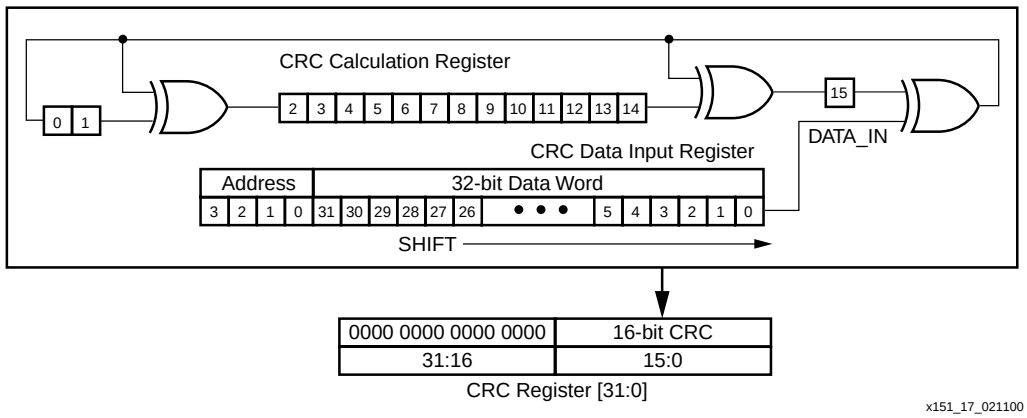

Figure 42: **CRC Calculation Register**

Frame Address Register (FAR)

The Frame Address Register (FAR) holds the address of the current frame. The address is divided into three parts, the block type, the major address, and the minor address. The block type field indicates whether the CLB or block RAM address space is used. The

command in the command register is executed each time the FAR is loaded with a new value.

The major address selects the CLB or RAM column, the minor address selects the frame within the column. The minor address is incremented each time a full data frame is read from or written to the frame data register. If the last frame within the CLB column is selected when the increment occurs, the major address is incremented and the minor address is reset to zero, otherwise the minor address is incremented. However, the block RAM major address is not incremented automatically .

Notes:
1. The block RAM major address is not incremented automatically. To address a different block RAM content column, the FAR must be loaded with the new major address.

See Table 63 for the definitions of valid values for the block type field. The FAR field definitions are given in Table 64

Table 62: **Block Type Codes**

Type	Codes
CLB	00
RAM	01

Table 63: **Frame Address Fields (FAR)**

	Block Type	Major Address (Column Address)	Minor Address (Frame Address)	
31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
0 0 0 0	0 x x x	x x x x	x x x x	x x x x	x x x 0	0 0 0 0	0 0 0 0

Notes:
1. Locations within fields containing a zero or one must have these values. An *X* in a bit field indicates that the value is variable and must be set.
2. Heavy vertical lines are used to separate fields. Light vertical lines separate nibbles in the word.

Frame Data Input Register (FDRI)

The Frame Data Input Register (FDRI) is used to load configuration frame data into a Virtex device.

The Frame Data Register (FDR) is a shift register into which data is loaded prior to transfer to the configuration memory. Configuration data is written to the Virtex device by loading the command register with the WCFG command and then loading the FDR with at least two frames of 32-bit words.

The write operation is pipelined such that the first frame of data is written to the configuration memory while the second frame is being shifted in. The last frame (the pad frame) is always dummy data which is not actually written to the configuration memory. Each frame write must include enough 32-bit data words to load the frame fully. There is one pad word at the end of each frame which is required for the pipelining hardware.

Frame Data Output Register (FDRO)

The Frame Data Output Register (FDRO) is for reading configuration data or captured data from the Virtex device, a process called readback. Readback is performed by loading the command register with the RCFG command and then addressing the FDRO with a read command.

Frame Length Register (FLR)

Near the beginning of the configuration bitstream the Frame Length Register (FLR) is written with the length of a frame, as measured in 32-bit words. This length count is used to provide sequencing information for the configuration read and write operations. Note that the FLR must be written before any FDR operation works. It is not necessary to set the FLR more than once. If the number of bits in a frame is not evenly divisible by 32, the length count of the frame must be rounded up to the next highest integer. The values for the FLR for all the current Virtex series devices are given in Table 64.

Notes:

The FLR contains a value that is one less than the number of words that are read from or written to a given frame. This is because the extra word needed for pipelining is not counted.

Table 64: **Frame Length Register Value**

Device	Row x Col	Frame Length	No. Words per Frame	FLR Value
XCV50/E	16 x 24	384	12	11
XCV100/E	20 x 30	448	14	13
XCV150	24 x 36	512	16	15
XCV200/E	28 x 42	576	18	17
XCV300/E	32 x 48	672	21	20
XCV400/E	40 x 60	800	25	24
XCV405E	40 x 60	800	25	24
XCV600/E	48 x 72	960	30	29
XCV800	56 x 84	1088	34	33
XCV812E	56 x 84	1088	34	33
XCV1000/E	64 x 96	1248	39	38
XCV1600E	72 x 108	1376	43	42
XCV2000E	80 x 120	1536	48	47
XCV2600E	92 x 138	1728	54	53
XCV3200E	104 x 156	1952	61	60

Legacy Output Register (LOUT)

The Legacy Output Register (LOUT) is used for daisy chaining the configuration bitstream to other Xilinx devices. Data written to the LOUT is serialized and appears on the DOUT pin.

Mask Register (MASK)

The Mask Register (MASK) is a mask register for writes to the CTL register. A “1” in bit N of the mask allows that bit position to be written in the CTL register. The default value of the mask is all “0”s.

Status Register (STAT)

The Status Register (STAT) is loaded with current values of several control or status signals. The register can be read via the reconfiguration block or via JTAG. The fields in the

Status register are illustrated in Table 65. The values of the signals given in Table 65 can be read from the status register.

Table 65: **Status Register Fields**

																DONE	INIT	MODE			GHIGH_B	GSR_B	GWE_B	GTS_CFG	IN_ERROR	LOCK				CRC_ERROR
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Notes:

1. An *X* in a bit field indicates that the value is variable.
2. Heavy vertical lines are used to separate fields. Light vertical lines separate nibbles in the word.

Table 66: **Status Register Bits**

Name	Bit Indices	Description
DONE	14	Input from DONE pin
INIT	13	Value of $\overline{\text{INIT}}$
MODE	12:10	Value of M2, M1, M0 mode pins
GHIGH_B	9	0 = GHIGH_B asserted
GSR_B	8	0 = all flip-flops are Reset / Set
GWE_B	7	1 = flip-flops and block RAM are write disabled
GTS_CFG	6	0 = I / Os are 3-stated
IN_ERROR	5	Legacy input error
LOCK	4:1	Output from DLL lock signals. 1 = DLL is locked.
CRC_ERROR	0	Indicates that a CRC error has occurred.

Configuration Interface

There are two configuration interfaces to the Virtex devices — the bit-serial Boundary Scan interface and the 8-bit byte-serial SelectMAP interface. Conceptually, XCV50 configuration data appears as in Table 67.

Table 67: **XCV50 Frame Padding for Reads**

Data Frame 0 (11 words)	Pad Word
⋮	
Data Frame *n* (11 words)	Pad Word
Pad Frame (12 words)	

Frames and words within frames are written in the same order in both configuration interfaces, starting with Frame 0, word 0 (the left-most in the picture), followed by word 1, etc. Bits within each word are written from left to right (*MSB first*) in the bit-serial configuration interfaces.

Within the SelectMAP interface, data is written a byte at a time. A sample word is shown in Figure 43. The top row indicates the device pin names. The bottom row indicates the bit indices within a configuration word. Byte 0 loads first, followed by byte 1, *and so on*. The MSB of each byte (that is, bits 31, 23, 15, and 7) is loaded on pin D0. The LSB of each byte (*that is*, bits 24, 16, 8, and 0) is loaded on pin D7.

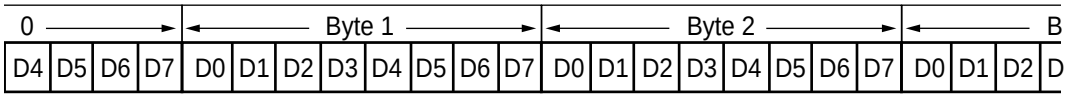

Figure 43: **SelectMAP Byte and Bit Ordering**

Partial Reconfiguration of CLBs

Partial reconfiguration can be performed with and without shutting down the device. If contention occurs it is advised to go through a shutdown sequence prior to loading configuration frames. When shutting down the DONE pin will go active and pull low. If this is not desirable the DONE pin can be prevented from going low by selecting the DONE_CYCLE to Keep State (111). Similarly, to prevent GSR from becoming active and resetting the current state, the GSR_CYCLE can be set to Keep State. When not shutting down, changes occur frame by frame, but is completely done by the end of the Last Frame packet (see Example 1). The DONE pin will not change in this case. A typical shutdown, reconfiguration in the CLB address space, and restart sequence is shown below.

Typical Shutdown, Reconfiguration in the CLB Address Space, and Restart Sequence

FFFF FFFF	Dummy Word
AA99 5566	Sync Word. Does not realign an already synchronized SelectMap port
3001 2001	COR
0080 FF2F	Shutdown bit set, optionally set bit 29 DRIVE_DONE Alternately COR can be read, bit 15 set, and written back
3000 8001	CMD
0000 0005	Start Shutdown
3000 8001	CMD
0000 0007	Reset CRC
0000 0000	
0000 0000	Clock shutdown sequence
0000 0000	
0000 0000	
3000 8001	CMD
0000 0008	Assert GHIGH
3000 8001	CMD
0000 0001	Write Configuration
3000 2001	FAR
0--- ----	Block Type CLB (00); Major/Minor Address
3000 4---	FDRI + Word Count if count is <1024. Otherwise use TypeII Header.
	Write FRAME DATA If CLB Frames are not written to non-consecutive addresses repeat FAR word, followed by new Major/Minor Address followed by FDRI + Word Count and then Frame Data.
3000 0001	Write CRC
---- ----	CRC
3000 8001	CMD

```
0000 0003      LFRM
3000 4---      FDRI + Word Count

                    Write PAD FRAME

3001 2001          COR
0080 3F2D          Default COR Options
                   Alternately COR can be read, bit 15 cleared, and written back
3000 8001          CMD
0000 0005          Begin Start-Up
3000 0001          Write CRC
---- ----          CRC
0000 0000
0000 0000          4 Dummy Words
0000 0000
0000 0000
```

Examples

Several examples of reading and evaluating configuration data are provided to illustrate the following.:

Example 1: Read and Write Semaphores in an XCV100 at CLB R1 C1, Slice 0

Semaphores are a useful communication mechanism documented in XAPP 153, “Status and Control Semaphore Registers using Partial Reconfiguration.” Figure 44 shows an abstraction of a microprocessor writing control information to an FPGA and reading status information. One convention for implementing semaphores in Virtex devices is to use two bits of a 16-bit, dual-port RAM, as illustrated in Figure 45. This occupies one CLB slice with the F-LUT implementing the control semaphore and the status semaphore implemented in the G-LUT. Address 15 of the G-LUT is used for on-chip writes to the semaphore and address 14 of the F-LUT is used for on-chip reads. Conversely, the off-chip microprocessor is reading the G-LUT[15], and writing F-LUT[14].

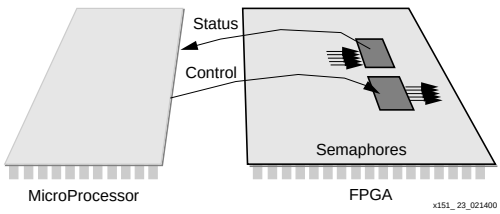

Figure 44: **Semaphore Abstraction**

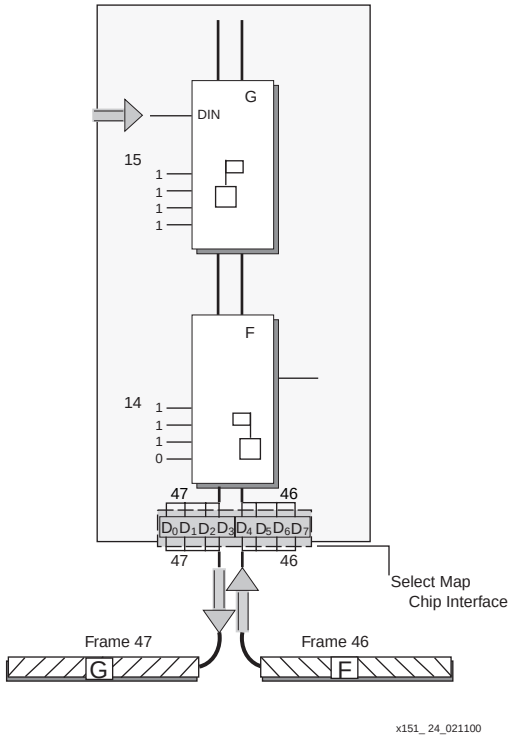

Figure 45: **Semaphore Read/Write Implementation**

Using a Semaphore Abstraction dual-port RAM ensures that the LUT SelectRAMs are placed in the CLB in a predictable manner. *When writing data to one or more LUT SelectRAMs or flip-flops on the device, all bits in the frame* ***must*** *have valid configuration information.* This is assured by altering valid configurations from bitstream files or from frames read from a properly configured Virtex device. The latter approach is used in this example. Attributes for this design are summarized in Table 68.

Table 68: **Design Attributes for Example 1**

Attribute	G-LUT [15]	F-LUT [14]	
	Read	Read	Write
Chip_Rows	20		
Chip_Cols	30		
FL	14		
CLB_Row	1		
CLB_Col	1		
Slice	0		
FG	1	0	
lut_bit	15	14	
RW	1	1	0

From the equations in Table 40 on page 91, the values shown in Table 69 can be calculated.

Table 69: **Semaphore Example Variables, Equations, and Values**

Variable	Equation	Value(s)		
		G-LUT[15]	F-LUT[14]	
		Read	Read	Write
MJA	$1 \le 30/2 \Rightarrow 30 - 1 \times 2 + 2$	30		
MNA	lut_bit + 32– 0 x (...)	47	46	
fm_bit_idx	3 + 18 x 1 – FG + RW x 32	52	53	21
fm_st_wd	14 x (8 + (30 – 1) x 48 + {46,47}) +RW x (14 +1) = 14 x (1,400 + {46,47}) + 14 x RW = 19,600 + 14 x {46,47} + 14 x RW	20,272	20,258	20,244
fm_wd	floor (20/32)	1	1	0
fm_wd_bit_idx	31 + 32 x {1,1,0} – {52,53,21}	11	10	10

From off-chip, for reading the G-LUT[15] bit, read one frame (MNA=47). For writing the F-LUT[14] bit, we show how to read then write one frame (MNA=46), as opposed to modifying data from a bitstream file (both are valid methods). The frames on the XCV100 contain 13 32-bit words and one pad word. Remember that fm_st_wd is calculated assuming the entire configuration has been read. However, only the pad frame and then frames 46 and 47 from Major Address 30 are being read. The desired bit is in Frame 1, word 1, which is word 15.

The commands for reading both frames (and the pad frame) are given in Table 70.

Table 70: **Commands to Read Two Data Frames**

Instruction	Data																													
	Hex	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sync Word	AA99 5566	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	0
Write next (1) word to FAR	3000 2001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
CLB MJA=30, MNA=46	003C 5800	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	**1**	0	0	0	0	0	0	0	0	0	0
Write next word to CMD	3000 8001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Register value for RCFG	0000 0004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Read from FDRO	2800 602A	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	1	0
Flush pipe	0000 0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(read 42 words)																																	

Notes:

1. Binary data is grouped in two ways for ease of interpretation. Thin vertical lines separate nibble boundaries. Heavy vertical lines separate field boundaries.

The 42 words read are shown in Table 71. F-LUT[14] is in the second frame (frame=1, word=1) at bit 10. The value read is a “1”, but because the LUT bits are inverted, the logic value is zero. G-LUT[15] is in word one of the third data frame (frame=2, word=1) at bit 11.The value read is a “1”, which is a logic zero.

To write to the F-LUT semaphore, start with the second data frame (frame=1) that was just read. Words 1 - 13 of that frame will become words 0 - 12 of the frame to be written. Word 13 of this new frame is a pad word and can have any value (typically 0 is chosen). In this new frame, set bit 11 of word 0 (zero) to the desired value of the semaphore. A pad frame must follow the data frame. The commands from Table 71 write these two frames into the device at the proper location.

Table 71: **Three Frames Containing the Semaphore**

Frame	Word	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	1	**1**	0	0	0	0	0	0	0	0	0	0	F[14]
	2	1	1	1	1	1	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	
	3	0	0	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	0	0	
	4	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	
	5	1	1	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	
	6	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	0	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	
	7	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	0	1	1	0	0	0	0	
	8	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	
	9	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	
	10	1	0	1	1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	
	11	1	1	1	1	1	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	
	12	0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	
	13	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Table 71: **Three Frames Containing the Semaphore *(Continued)***

Frame	Word	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	0	**1**	1	0	0	0	0	0	0	0	0	0	0	G[15]
	2	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	
	3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	0	0	
	4	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	
	5	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	1	1	
	6	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	
	7	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	
	8	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
	9	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	
	10	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	
	11	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	
	12	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	
	13	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Table 72: **Commands to Write a Semaphore Value**

Instruction	Data																																	
	Hex	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Sync Word	AA99 5566	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	0	
Write next (1) word to FAR	3000 2001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	
MJA=30, MNA=46	003C 5C00	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	
Write next word to CMD	3000 8001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Register value for WCFG	0000 0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Write 28 words to FDRI	3000 401C	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	
Data Word 0	A01A EC00	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	
Data Word 1	FF0F 3FC0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	
Data Word 2	0FF0 02FC	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	0	0	

Table 72: **Commands to Write a Semaphore Value *(Continued)***

Instruction	Data																													
	Hex	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
⋮	⋮	⋮																															
Data Word 27	0000 0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Notes:

1. Binary data is grouped in two ways for ease of interpretation. Thin vertical lines separate nibble boundaries. Heavy vertical lines separate field boundaries.

Example 2: Reading the Complete Configuration from an XCV50

Steps:

1. If flip-flop values are needed, clock the on-chip signal, CAPTURE, to capture flip-flop values. See the Xilinx Libraries Guide for use of the CAPTURE_VIRTEX cell.
2. Write the starting frame address (CLB MJA=0 MNA=0) into the FAR.
3. Write the RCFG command to the CMD register.
4. Address the FDRO register with a READ operation and word count equal to the number of 32-bit words in the CLB frames plus one pad frame.
5. Read the data following the timing diagrams in the SelectMAP interface section.
6. Write the address for RAM block 0 to the FAR.
7. Address the FDRO register with a read operation and word count equal to the number of 32-bit words in the RAM block plus one pad frame.
8. Read the data.
9. Write the address for RAM block 1 to the FAR.
10. Address the FDRO register with a read operation and word count equal to the number of 32-bit words in the RAM block plus one pad frame.
11. Read the data.

When using SelectMAP mode to read data words from the Virtex device, de-assert $\overline{CS}$, de-assert $\overline{WRITE}$, assert $\overline{CS}$, then clock the data out. When using JTAG, load the JTAG IR (instruction register) with the CFG_OUT instruction. Then go to the SDR (Shift-DR) state and shift the data out. (See Table 73).

Table 73: **Example 2 - Read Complete Configuration (for XCV50)**

Instruction	Data																													
	Hex	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sync Word	AA99 5566	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	0
Write next word to FAR.	3000 2001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
CLB MJA=0, MNA=0	0000 0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Write next word to CMD register.	3000 8001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Register value for RCFG	0000 0004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Table 73: **Example 2 - Read Complete Configuration (for XCV50) *(Continued)***

Read from FDRO register.	2800 6000	0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
15876 words	4800 3E04	0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 0 0
Flush pipe	0000 0000	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(Read 15876 words.)		
Write to FAR register.	3000 2001	0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
RAM MJA=0, MNA=0	0200 0000	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Read 780 words from FDRO reg.	2800 630C	0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0
Flush pipe	0000 0000	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(Read 780 words)		
Write to FAR register.	3000 2001	0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
RAM MJA=1, MNA=0	0202 0000	0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Read 780 words from FDRO reg.	2800 630C	0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0
Flush pipe	0000 0000	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(Read 780 words)		

Example 3: Read the Slice 0 G-LUT from CLB R1 C1 from the Complete Configuration of an XCV50

The commands for reading the bitstream from the Virtex device are given in "Example 2: Reading the Complete Configuration from an XCV50" on page 113. Using an XCV50 device, the independent attributes are show in Table 74.

Table 74: **XCV50 Independent Attributes**

Independent Attributes	Values
Chip_Rows	16
Chip_Cols	24
FL	12
CLB_Row	1
CLB_Col	1
FG	1
Slice	0
RW	1

From the equations given earlier in Table 43 on page 92, calculating the range of values for fm_st_wd indicates that the word 13740 of the configuration is the starting word of the 12-word (FL=12) frame containing bit 0 of the G-LUT in Slice 0 of CLB R1C1 (Table 75).

The configuration bits for the given frame are as follows: G-LUT[0] is in fm_wd=0, at bit 11.

Table 75: **Variables, Equations and Values for Slice 0 G-LUT**

Variables	Equations	Values			
MJA	$1 \le 24/2 \Rightarrow 24 - 1 \times 2 + 2$	24			
MNA	0 x (…) + lut_bit + 32	0: 32 1: 33 2: 34 3: 35	4: 36 5: 37 6: 38 7: 39	8: 40 9: 41 10: 42 11: 43	12: 44 13: 45 14: 46 15: 47
fm_bit_idx	3 + 18 x 1 –1 + RW x 32	52			
fm_st_wd	12 x (8+ (24 –1) x 48 + [32:47]) +1 x 12 = 12 x (1112 + [32:47]) +12 = 13,356 + 12 x 32:47]	13,740:13,920			
fm_wd	floor(52 / 32)	1			
fm_wd_bit_idx	31 + 32 – 52	11			

Table 76: **Configuration Bits for Slice 0, CLB R1C1 G-LUT [0]**

Bitstream Word	Frame 32-bit Word																												Frame Word
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
13740	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13741	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1
13742	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	2
13743	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	3
13744	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	4
13745	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	5
13746	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	6
13747	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	7
13748	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	8
13749	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	9
13750	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	10
13751	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11

All 16 LUT SelectRAM bits are in the following words at the same bit index, 11.

Table 77: **Location of All 16 LUT SelectRAM Bits**

Bitstream Word	Frame 32-bit Word																												LUT Bit
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
13741	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
13753	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1
13765	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	2
13777	0	0	0	0	1	0	0	0	0	0	0	1	1	1	0	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	3

Table 77: **Location of All 16 LUT SelectRAM Bits *(Continued)***

Bitstream Word	Frame 32-bit Word																												LUT Bit
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
13789	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	4
13801	1	0	1	0	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	5
13813	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	6
13825	0	1	1	0	1	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	7
13837	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	8
13849	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	9
13861	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	10
13873	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1	0	1	1	0	0	0	1	0	0	0	0	0	0	11
13885	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	12
13897	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	13
13909	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	14
13921	1	0	1	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	15

The 16 bits are $\overline{\text{LUT}}$[15:0]=0111111111111111. The LUT SelectRAM bits are inverted from their logic values. The “logical” contents are LUT[15:0]=1000000000000000. Thus, this G-LUT implements a 4-input AND function.

Example 4: Read the Slice 1 F-LUT from CLB R19 C16 from an XCV100

Commands for reading the bitstream from the Virtex device are given in Table 80 on page 117. Use the following independent attributes to find the given F-LUT given in Table 78.

Table 78: **Virtex Bitstream Command Attributes**

Independent Attributes	Values
Chip_Rows	20
Chip_Cols	30
FL	14
CLB_Row	19
CLB_Col	16
FG	0
Slice	1
RW	1

From the equations in Table 43 on page 92, calculating fm_st_wd indicates the starting word with respect to a configuration that starts at MJA=0, MNA=0. Because the frames we are interested in start at MJA=1, MNA=0, which is fm_st_wd = 126, so the first 126 words are not needed (0–125). Therefore, to find the given Slice 1 F-LUT, see Table 79.

Table 79: **Variables, Equations, and Values for Slice 1 F-LUT**

Variables	Equations	Values			
MJA	16 > 30 / 2 ⇒ 2 x 16 – 30 –1	1			
MNA	lut_bit + 32 – Slice x (2 x lut_bit +17) = [0:15] + 32 – (2 x [0:15] + 17) = 15 – [0:15]	0: 15 1: 14 2: 13 3: 12	4: 11 5: 10 6: 9 7: 8	8: 7 9: 6 10: 5 11: 4	12: 3 13: 2 14: 1 15: 0
fm_bit_idx	3 + 18 x 19 – 0+ 32	377			
fm_st_wd	14 x (8 + (1– 1) x 48 + [15:0]) +1 x 14 = 14 x (8 + [15:0]) +14 = 126 + 14 x [15:0]	= 336:126			
fm_wd	floor(377 / 32)	11			
fm_wd_bit_idx	31 + 32 x 11 – 377	6			

Sixteen frames need to be read, one for each bit in the LUT SelectRAM. The bits in LUT SelectRAMs in Slice 1 occur in the opposite order that they do for Slice 0 LUT SelectRAMs.

Frames are read sequentially with ascending addresses. If read in LUT bit order, LUT[0], LUT[1], …, LUT[15]. These are stored in descending addresses which require 16 separate read operations each reading one data frame and one pad frame. However, if read in ascending address order, LUT[15], LUT[14], …, LUT[0], all 16 data frames are read with a single read operation. This requires only one pad frame for all 16 frames. Thus, it takes less time to read ascending frames starting at MJA=1, MNA=0 and finishing with frame MJA=1, MNA=15. The frames in the XCV100 contain 14 32-bit words and a single pad word. Commands for reading only the F-LUT data are given in Table 80.

Table 80: **Commands to Read Slice 1 F-LUT**

Instruction	Data																													
	Hex	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sync Word	AA99 5566	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	0
Write next (1) word to FAR.	3000 2001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
CLB MJA = 1, MNA = 0	0002 0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Write next word to CMD	3000 8001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Register value for RCFG	0000 0004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Read from FDRO	2800 60EE	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	1	1	1	0
Flush pipe	0000 0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(Read 238 words)																																	

The 377th bit of each frame is the bit in the F-LUT. This LUT SelectRAM bit is in the frame's word index 11, bit index 6.

The configuration bits for the given frame in Table 81 are as follows: LUT Bit 15 is in MJA=1, MNA=0, fm_wd=11, at bit 6.

Table 81: **Frame Containing F-LUT [15]**

Frame Word	Frame 32-bit Word																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0
4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
5	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1
6	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
7	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
8	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1
10	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
12	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0
13	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0

LUT bit 0 is in MJA=1, MNA=15, fm_wd=11, at bit 6 as shown in Table 82.

Table 82: **Frame Containing F-LUT [0]**

Bitstream Word	Frame 32-bit Word																														Framework
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
336	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
337	0	1	0	0	1	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
338	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	2
339	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	3
340	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	4
341	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	5
342	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	6
343	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	7
344	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	8
345	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	9
346	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	10

Table 82: **Frame Containing F-LUT [0] *(Continued)***

Bitstream Word	Frame 32-bit Word																												Framework
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
347	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	0	0	0	11
348	0	0	0	0	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	12
349	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13

For the sake of brevity, the sixteen 11th words are given in Table 83 in the order they appear in the bitstream.

Table 83: **Sixteen Words Containing the F-LUT Bit**

Bitstream Word	Frame 32-bit Word																												LUT Bit
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
137	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	15
151	1	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	1	0	0	0	0	0	0	14
165	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	1	0	0	0	0	0	0	13
179	1	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	1	0	0	0	0	0	0	12
193	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	11
207	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	10
221	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	9
235	1	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	1	0	0	0	0	0	0	8
249	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	7
263	1	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	0	6
277	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	0	0	0	5
291	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	4
305	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	3
319	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	0	2
333	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	1
347	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	0	0	0	0

The bits are $\overline{\text{LUT}}$[15:0] = `0111111111111111`. The LUT SelectRAM bits are inverted from the logic sense. The logical contents are LUT[15:0] = `1000000000000000`. Thus, this F-LUT implements a 4-input AND gate.

Example 5: Read All Bits in Slice 0 G-LUTs from CLB C2 and XCV50

Given the following attributes in Table 84, the necessary values to find the G-LUT data can be computed.

Table 84: **All Bits: Slice 0 G–LUTs from CLB C2 and XCV50**

Independent Attributes	Values
Chip_Rows	16
Chip_Cols	24
FL	12
CLB_Row	1:16
CLB_Col	2
FG	1
Slice	0
lut_bit	0:15
RW	1

From the equations in Table 43 on page 92 and Table 85, the dependent variables can be calculated.

Table 85: **Dependent Variables, Equations, and Values**

Variables	Equations	Values			
MJA	$2 \le 24/2 \Rightarrow 24 - 2$ x 2 + 2	22			
MNA	lut_bit + 32 – Slice × (2 × lut_bit + 17) = lut_bit + 32 – 0 × (2 × lut_bit + 17) = [0:15] + 32	0:32	4:36	8:40	12:44
		1:33	5:37	9:41	13:45
		2:34	6:38	10:42	14:56
		3:35	7:39	11:43	15:47
fm_bit_idx	3 + 18 x CLB_Row – 1+ 32 = 34 + 18 x [1:16]	1:52	5:124	9:196	13:268
		2:70	6:142	10:214	14:286
		3:88	7:160	11:232	15:304
		4:106	8:178	12:250	16:322
fm_st_wd	12 × (8 + (22 –1) × 48 + MNA) + 1 × 12 = 12 x (1,016 + MNA) + 12 = 12,204 + 12 x MNA	0: 12,588	4: 12,636	8: 12,684	12: 12,732
		1: 12,600	5: 12,648	9: 12,696	13: 12,744
		2: 12,612	6: 12,660	10: 12,708	14: 12,756
		3: 12,624	7: 12,672	11: 12,720	15: 12,768
fm_wd	floor (fm_bit_idx / 32)	1:1	5:3	9:6	13:8
		2:2	6:4	10:6	14:8
		3:2	7:5	11:7	15:9
		4:3	8:5	12:7	16:10

Table 85: **Dependent Variables, Equations, and Values *(Continued)***

Variables	Equations	Values			
fm_wd_bit_idx	31 + 32 x fm_wd – fm_bit_idx	1:11	5:3	9:27	13:19
		2:25	6:17	10:9	14:1
		3:7	7: 31	11:23	15:15
		4:21	8:13	12:5	16:29

Note that fm_bit_idx, fm_wd, and fm_wd_bit_idx have 16 values, one for each row on the XCV50. The commands for reading the LUT SelectRAM data are given in Table 86. Table 87 shows where the data lies in the first frame, which contains G[0] for the entire column. The process is the same for the other 15 frames.

From the calculation for fm_st_wd, Frame 0 would start at word 12,588 reading the whole configuration. The instructions in Table 86 start at that word, so word 0 (ignoring the 12 words in the pad frame) is the same as word 12,588 of the entire CLB configuration. The 12 words in the frame are shown in Table 87.

The LUT SelectRAM bits have been shaded for ease of identification. It can be seen from the calculation of fm_bit_idx that the LUT SelectRAM bits are in the order 1:16. For example, from the above calculations for fm_wd and fm_wd_bit_idx, G-LUT[0] in R1C2 is in fm_wd 1, fm_wd_bit_idx 11. This bit is the shaded bit in word 1 at bit index 11. Similarly, the G-LUT[0] for the other 15 CLB rows are also shaded in Table 88.

Table 86: **Commands to Read R*C2.S0 G-LUT**

Instruction	Data																													
	Hex	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sync Word	AA99 5566	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	0
Write next (1) word to FAR	3000 2001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
CLB MJA=22, MNA=32	002C 4000	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Write next word to CMD	3000 8001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Register value for RCFG	0000 0004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Read from FDRO	2800 60CC	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	1	1	0	0
Flush pipe	0000 0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(Read 204 words.)																																	

Table 87: **Frame for R*C2.S0 G-LUTs, Bit G[0]**

Word	Frame 32-bit Word																												CLB Row
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
2	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	2, 3
3	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	4, 5
4	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	6
5	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	7, 8
6	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	9, 10
7	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	11, 12
8	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	13, 14
9	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	15
10	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	16
11	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Example 6: Read Block SelectRAM Index 387 of RAM R2 C0 from an XCV100E

Commands for reading the bitstream from the Virtex device are given in Table 90 on page 123. Use the following independent attributes in Table 88 to find the given F-LUT.

Table 88: **Virtex Bitstream Command Attributes**

Independent Attributes	Values
Chip_Rams	4
FL	14
RAM_Row	2
RAM_Col	0
ram_bit	387
RW	1

Since we are attempting to read only one bit location, only one frame is necessary to be read. We use the equations of Table 48, Table 49, and Table 50 to determine the dependent variables. See also Table 89.

Table 89: **Variables, Equations, and Values for Slice 1 F-LUT**

Variables	Equations	Values
MJA_virtex	0 <4/2 ⇒ 2 x (4/2 -1 -0)	2
MJA_adj	0 <4/2 ⇒ 2	2
MJA	MJA_virtex + MJA_adj	4

Table 89: **Variables, Equations, and Values for Slice 1 F-LUT *(Continued)***

Variables	Equations	Values
MNA	floor(((387 / 64)%64) / 32)+ 2 x floor(((387 / 64)%32) / 16) + 4 x floor(((387 / 64)%16) / 8) + 8 x floor(((387 / 64)%8) / 4) + 16 x floor(((387 / 64)%4) / 2) + 32 x floor(((387 / 64)%2) / 1)	24
fm_bit_idx	3 + 18 x 19 – 0 + 32	219
fm_st_wd	14 x 24 + 1 x 14	350
fm_wd	floor(219 / 32)	6
fm_wd_bit_idx	31 + 32 x 6 – 219	4

Commands for reading this memory index are given in Table 90.

Table 90: **Commands to Read Block SelectRAM Index 387**

Instruction	Data																													
	Hex	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sync Word	AA99 5566	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	0
Write next (1) word to FAR	3000 2001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
CLB MJA = 4, MNA = 24	0208 3000	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Write next word to CMD	3000 8001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Register value for RCFG	0000 0004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Read from FDRO.	2800 601C	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0
Flush pipe	0000 0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(Read 28 words)																																	

RAM bit 387 is at fm_wd=6, bit 4. See Table 91.

Table 91: **Frame MNA=24**

Frame Word	Frame 32-bit Word																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0
4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
5	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1
6	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Table 91: **Frame MNA=24 *(Continued)***

Frame Word	Frame 32-bit Word																											
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
8	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1
10	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
12	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0
13	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0

Chapter 6

Configuration FAQs

This chapter is a collection of questions and answers to the more prevalent issues that have been identified concerning configuration. Entries are organized by function / feature and in some cases product.

Validating Configuration

When can I start to configure?

After a power-on time and a small delay before $\overline{\text{INIT}}$ is released. For Virtex devices, the power-on time is 2 ms and the delay before $\overline{\text{INIT}}$ is released is 0.1 ms, for a total delay for 2.1 ms.

Why does $\overline{\text{INIT}}$ not go High?

1. The rise time for the core voltage is too slow. Maximum rise time should be 50 ms (1V to 2.4V).
2. The $\overline{\text{PROGRAM}}$ pulse is too short. An FPGA feature is the ability to re-configure the device after power-up by toggling the $\overline{\text{PROGRAM}}$ pin to Low, resetting the configuration state machine. After the $\overline{\text{PROGRAM}}$ pin is released, the FPGA re-starts the configuration process. The minimum pulse width for the $\overline{\text{PROGRAM}}$ pin on a Virtex series device is 200 ns. However, if the $\overline{\text{PROGRAM}}$ pin is connected to the $\overline{\text{PROGRAM}}$ pins of other families (SpartanXL or XC4000) the minimum pulse width is 300 ns. *(See 8022.)*
3. The $\overline{\text{PROGRAM}}$ pin is held Low.

Why does $\overline{\text{INIT}}$ go Low during programming?

The $\overline{\text{INIT}}$ Pin goes Low on a certain frame, usually indicating a Frame Error has occurred during configuration. The configuration pins seem to be steady (no noise issue), the Preamble from DOUT is correct (0010), and the data input matches the bit file.

A Data Frame Error can be an indication of a corrupted bit file. If the file has been ported through e-mail or an ftp site, it is possible that it has become corrupted during the transfer. For ASCII file transfer, try binary mode or compressed mode to avoid corruption. ASCII mode transfer can add extra carriage returns or line feeds to the bit file.

Is there a problem when WRITE is toggled during a Virtex serial configuration (DONE does not go High)?

A problem occurs only if all the following conditions are met.

The $\overline{\text{CS}}$ and $\overline{\text{WRITE}}$ lines are both used as I / O after configuration.

$\overline{CS}$ is Low during configuration,

AND $\overline{WRITE}$ toggles during configuration.

If ALL three conditions are met, configuration fails and DONE doesn't go High. Toggling the $\overline{PROGRAM}$ pin and reloading configuration WITHOUT one of the above conditions results in a successful configuration. *(See 5742.)*

What if DONE goes High but the device does not start?

A Virtex device is configured at a high speed, DONE goes High, but the device does not start – the I/O is still 3-stated.

1. If the DONE pin rises slowly, attach a 330 Ω external resistor to DONE. If DONE still does not rise within one CCLK cycle, set the bitgen option DonePipe=Yes. With this option set to Yes, the device adds a pipelined register stage to the DONE input (CFG_DONE) path. This is only needed when the DONE rise time exceeds the CCLK period.for example, CCLK is running at 33MHz in slave serial mode. If the rise time of DONE is > 25 ns, delay the CFG_DONE signal using DonePipe=Yes.
2. If the DONE pin rises slowly, try setting the bitgen option DriveDone=Yes to actively drive the DONE pin.

   Note: This can only be done when configuring one device or when the device is the last one in a daisy-chain.
3. If the rise time of the DONE pin is within one cycle of CCLK, but the device still does not start, try the following bitgen options:
   - DONE_cycle = 3
   - GSR_cycle = Done
   - GWE_cycle = Done
   - GTS_cycle = Done

During configuration it is possible that the FPGA receives all configuration data correctly, but still fails to finish the configuration sequence. This is most likely because of an incomplete start-up sequence.

1. Completing a STARTUP sequence generally just requires a few more clock cycles. By default, the clock signal used for STARTUP is CCLK. However, other (for example, USER or JTAG) clocks can be specified with the STARTUP component to select the startup clock options for bitstream generation. Verify the clock selected for your design and assert the clock. The bitgen.ut file lists the bitgen options used (if the GUI is used).
2. Another possibility is that the DONE pin is being held Low externally, or is simply not being pulled HIgh. If necessary, separate the DONE pin from the board to verify if an external source is holding it Low.The DONE pin is an Open-Drain driver that must be pulled-up to a logic High. The FPGA does have a programmable internal pull-up resistor for the DONE pad, but using an external 4.7KΩ resistor with 4000/5200 devices or an external 330 Ω resistor with Virtex devices is recommended.

Virtex has a potential race condition that can occur during STARTUP involving sensing the state of the DONE pin. The failure mode is that the global 3-state net is not released, thus keeping all I/Os in a High-Z state. There is a higher probability of this occurring when there is a weak pull-up resistor on DONE or a heavily loaded DONE net. The typical complaint is that all pins are stuck High, or if using CLKDLL, that it is not working.

1. Use a 330 Ω resistor to 3.3 V for the DONE pin.
2. For the last Virtex device in the chain, set the bitgen option "-g DriveDone:Yes". This actively drives the DONE pin High on the last device.

What if DONE does not go High, and $\overline{\text{INIT}}$ does not go Low?

This condition indicates that the Virtex synchronization word is not being captured, or a bit is being added or lost.

The Virtex synchronization word is: AA995566. The first two hex values of AA are represented in a binary byte as: 1010 1010.

1. The MSB of each byte must be on D0 data pin of the Virtex device. In this case, the left-most “one” is placed on D0, the adjacent “zero” is placed on D1, and so forth. Ensure that data is appropriately presented on the Virtex data pins.
2. If a bit is either added or lost during configuration (a clock glitch - or noise) the remainder of the bitstream is misaligned. Each unit after the false edge is one bit off, so it can not be recognized properly. As a consequence, no more data can be loaded, there is no CRC, INIT doesn’t go Low, and DONE doesn’t go High.

Why does the DONE Pin appear to be externally held Low?

The DONE pin is an open-drain driver that must be pulled up to achieve a logic High. Although the FPGA does have a programmable internal pull-up resistor to the DONE pad, use an external 330 Ω resistor for Virtex devices. If necessary, separate the DONE pin from the board to verify if an external source is holding it Low.

Double check the STARTUP sequence options selected for Bitstream generation. SyncToDone is recommended. It is required if multiple FPGAs are daisy chained in Serial or Express Mode configuration. SyncToDone prevents the STARTUP sequence from continuing (I/Os active) until the DONE pin is externally allowed to go High. If the FPGA is daisy chained, DONE is probably being held Low by one of the other FPGAs.

Recommended STARTUP Options for Virtex Series and Virtex Derivatives (Spartan XL):

- DONE_cycle:4
- GTS_cycle:5
- GSR_cycle:6
- GWE_cycle:6

Notes:

1. These options might need to be altered, depending on the system topology.

How do you know if the device has been synchronized (synchronization word has been loaded)?

Virtex requires a 32-bit synchronization word (0xAA995566) before it recognizes configuration data. How do you determine whether a device has been synchronized?

There is no immediate way to determine whether a device has been synchronized. The bitstream produced by bitgen is padded with dummy words at the beginning, so there is ample opportunity to get the synchronization word loaded. However, if you suspect that the device is not properly synchronized, creative use of the LOUT register can indicate proper synchronization.

Note: Writing to the LOUT register is only valid in serial modes.

The LOUT register is a pipeline register that transfers data to the DOUT pin. Inserting a small LOUT write immediately after the synchronization word is loaded can indicate to the user if the sync word is loaded properly. If so, the device recognizes the LOUT write packet, and data is seen on the LOUT pin. There is a latency of 40 clock cycles between data being written to LOUT and data appearing on DOUT. If the sync word isn't loaded properly, no packets are recognized, and the DOUT pin remains High.

Example:

The Bitstream as produced by bitgen (separated by DWORDS for clarity) does included the proper synchronization word.

```
FFFFFFFF
AA995566
30008001 (CMD Write)
```

To verify this, uses a hex editor to insert a packet for an LOUT Write before the first CMD Write:

```
FFFFFFFF
A995566
30010001 (Packet header for an LOUT write with Word Count = 1)
00000000 (32 bits of data)
30008001 (1st CMD Write)
```

This example produces 32 zeros on DOUT and configuration begins, indicating a valid synchronization word. Although this solution uses LOUT Writes to indirectly indicate that the synchronization word is loaded, it doesn't preclude embedding other LOUT Writes in the bitstream for other reasons at any time during configuration. For a more in-depth discussion of the Virtex configuration format, configuration registers, and packets, please refer to Xilinx application note XAPP138.

What I/O standard applies to the Virtex Family configuration pins, and is a specific V_{CCO} or V_{REF} required?

Inputs are all LVTTL and function with 2.5- to 5-V TTL levels. Inputs are sourced from V_{CCINT} and are not affected by V_{CCO}. Outputs (D_{OUT}/ $D_{0:7}$, TDO, BUSY, CCLK) have been constrained to Banks 2 & 3. For these outputs, VCCO should be 3.3 V.

Why is nothing appearing on D_{OUT}?

With Virtex devices, configuration doesn't pass data until it is complete. It does not start until the Virtex synchronization word (AA995566) is given.This is in contrast to the XC4000 devices, which broadcast the preamble and length count.

In Master Serial mode, what happens if the Prom data is corrupted?

If the data is corrupted, the CRC turns out incorrect. Upon detecting a CRC error, the $\overline{\text{INIT}}$ pin is pulled Low but CCLK continues clocking. The part must be re initialized by pulling $\overline{\text{PROGRAM}}$ low (min 200 ns, or 300 ns if other devices are in the chain). Cycling power also works. In most cases, the PROM must be swapped.

Configuring Multiple Devices

What configuration modes can be used with Virtex device in a daisy-chain configuration?

Both Master Serial and Slave Serial modes can be used.

Can you configure Virtex and 4KX devices mixed in a daisy-chain?

1. Virtex and XC4000X devices can be mixed in a serial daisy chain, as long as the following conditions are met:
   a. All DONE pins are wired together with an external pull-up to 3.3V

b. XC4000X devices must have the bitstreams generated with the following options:
   - LC_Alignement = DONE
   - SyncToDone = Yes

c. If programming by a micro-device, at least three clocks must be issued after DONE has gone High or the devices must use a user clock for the startup sequence.

There are no ordering requirements for the daisy chain. Families should be grouped together for design simplicity, which also reduces the bitstream size.

The following devices are permitted in the serial chain:

- Virtex
- XC4000EX
- XC4000XL, XC4000XLA
- XC4000XV
- SpartanXL, Spartan-II

All other devices are NOT permitted in the serial chain, including:

- XC3000*, XC3100*
- XC4000, XC4000E
- SPARTAN, XC5200

2. When Virtex devices are daisy-chained with non-Virtex devices, the internal length count is disabled (for all devices where this is possible, namely 4KEX and newer). This is a bit that PROMGEN switches. SpartanXL devices must set SyncToDone = YES. For Virtex devices, GSR, GWE, and GTS should all toggle on the same start-up cycle as DONE. If there are ten Virtex devices and one SpartanXL in a chain, regardless of the position in the chain, the SpartanXL length count is disabled.

What is the maximum data amount in a daisy chain configuration?

For Virtex, Virtex-E, and Spartan-II devices in a configuration daisy chain, the maximum data amount of data that can be sent to the D_{OUT} pin is 2^{20-1} (1,048,575) 32-bit words, or 33,554,400 bits.The configuration bitstream of downstream devices is limited to this size.

Virtex-II will allow for a 2^{27-1} word maximum at this time.

Limits come directly from the maximum configuration packet size. The bitstream for downstream devices must fit within a single packet.

Configuring Virtex Series Devices

Is the Virtex serial configuration mode similar to the XC4000 serial configuration mode?

The Virtex and XC4000 serial configuration modes are very similar except for the number of status pins. The Virtex device does not have $\overline{LDC}$ or HDC pins.

What is the default configuration mode for Virtex devices?

When Virtex mode pins are left unconnected, the default configuration mode is Slave Serial.

What is the relationship with Virtex devices between CCLK and DOUT during configuration?

With Virtex devices, data appears on the D_{OUT} pin on the rising edge of CCLK. In non-Virtex devices, data appears on the D_{OUT} pin on the falling edge of CCLK.

Does the Virtex internal configuration bus clock turn off after configuration?

Yes. If another configuration command is sent to the JTAG port, the bus clock turns back on until the command is completed.

Notes:

1. In a V1000 device, the bus clock can consume 100 mA during configuration

What configuration speeds are possible with Virtex devices?

1. The available configuration rates in MHz for Virtex devices are 4, 5, 7, 8, 9 10, 13, 15 20, 26, 30, 34, 41, 45, 51, 55, and 60. However, configuration clock speeds are typically limited by the maximum frequency supported by Serial PROMs and the devices in the configuration chain. Check the specifications on these other devices, if used.
2. Maximum configuration rates are:
   - Virtex only serial - 66 MHz
   - JTAG - 33 MHz (can be limited by other devices)
   - SelectMAP - 50 MHz (no BUSY) or 66 MHz (possible BUSY)

How can you enable Virtex pull-ups in the I/Os during configuration?

Virtex devices have eight configuration modes. The standards modes of Master Serial, Slave Serial, SelectMAP, and Boundary Scan are augmented by the same four modes with pull-ups.

1. Virtex I/Os are 3-stated during configuration, but users can enable the I/O pull-ups by changing the mode pin settings. The pull-ups are the internal pull-ups in the I/O blocks. Activation of the pull-up resistors prior to configuration is controlled globally by the mode pins. For instance, if the mode pins are set to 011, this enables the internal pull-ups in the IOs for the Slave Serial mode for the device being configured. For more detailed Virtex configuration information, please see “Virtex Configuration and Readback” in the Applications Chapter.
2. Mode pin settings for the four configuration modes with pull-ups

Table 92: **Configuration Mode Pin Settings**

M2	M1	M0	Mode
1	0	0	Master Serial with Pull-ups
0	0	1	Boundary Scan (JTAG) with Pull-ups
0	1	0	SelectMAP with Pull-ups
0	1	1	Slave Serial with Pull-ups

3. After configuration, all unused I/Os are configured as a 3-state LVTTL output buffers with a weak pull-down resistor.

Since Virtex devices do not pass data on the DOUT pins during configuration, how is this information accessed?

Virtex device configuration data is completely different from earlier Xilinx families; it does not have a header, preamble, or length count information at the beginning of the bitstream. Additionally, Virtex devices do not “overflow” and pass configuration data out of the DOUT register as with previous FPGA families.

Instead, the only way to pass data out through D_{OUT} is by writing to the LOUT register. LOUT is a Legacy Data Out register that pipelines data to the DOUT pin.

If a Virtex device is the only device in a configuration chain, no data is passed via DOUT or written to the $_{LOUT}$ register.

If a Virtex device is in a configuration chain, it is necessary to pass data via DOUT to configure the complete chain of devices. If the PROM File Formatter (or PROMGen) is used to concatenate bit streams into a PROM file, the required writes to the LOUT register is handled automatically. However, if a simple concatenation of the bit files is done, the user must add the writes to the LOUT register, which forces the data to be passed via the DOUT pin for chain configuration. For specific information on register writes, refer to "Virtex FPGA Series Configuration and Readback" on page 17 in this guide.

How long does the Virtex $\overline{\text{PROGRAM}}$ signal have to be held Low?

Virtex devices require a minimum of 200 ns duration of the $\overline{\text{PROGRAM}}$ signal. There is no maximum limit, unlike all other Xilinx FPGA families.

How do you prevent Virtex DONE from going High before CLKDLL has been locked?

The DONE pin indicates that configuration is complete and the device is in start-up. By default, the configuration start-up sequence does not wait until the CLKDLLs have been locked. The following 2-step process illustrates how to ensure that Done does not go High until all of the CLKDLLs are locked.

1. Attach the STARTUP_WAIT attribute to the CLKDLL in the UCF file:

   ```
   INST <DLL instance name> STARTUP_WAIT=TRUE
   ```

2. Use the following bitgen option:

   ```
   -g LCK_cycle:3
   ```

How many loads can the Virtex configuration pin CCLK drive?

In a daisy-chained application, there could be multiple PROMs and FPGAs driven by a single CCLK.

Generally, the Virtex rule of thumb is eight loads maximum on a single CCLK. A separate drive can be necessary for more than eight. Additionally, the faster the configuration rate, the greater the possibility that an additional driver is needed. An alternative approach is to create two or more smaller configuration chains (provide multiple masters), reducing the load on each CCLK.

When is the non-JTAG configuration mode of the Virtex chip determined?

The non-JTAG configuration mode for a Virtex device is determined after V_{CCINT} reaches its minimum voltage and $\overline{\text{INIT}}$ has gone High. Once $\overline{\text{INIT}}$ goes High, the values on mode pins, M0, M1, and M2 determine the mode.

What pins are used to monitor the status of non-JTAG Virtex configuration?

DONE and $\overline{\text{INIT}}$ are used to determine the status of non-JTAG Virtex configuration. If the bitstream has not been loaded, the DONE pin is Low. It goes High once the bitstream has been loaded. During configuration, the $\overline{\text{INIT}}$ pin is stays High. If an error is detected during configuration, the $\overline{\text{INIT}}$ pin goes Low and the device no longer responds to inputs. During configuration, D_{OUT} won't transition and remains disabled and pulled High unless configuration data needs to be forwarded to a daisy-chained device. After configuration, all unused IOs, including D_{OUT} and $\overline{\text{INIT}}$ if unused, are disabled but not pulled up. Unused IOs float Low during normal operation.

Can CRC in Virtex devices be disabled as with XC4000 devices?

No, BitGen does not have a switch to disable CRC for Virtex. Virtex CRC checking is different than with the XC4000 series and other FPGA series. In previous series, CRC is done after each frame. In the Virtex series, checking is just prior to loading the last frame and also at the end of the bitstream. For more information, see "Virtex Configuration and Readback" in the Applications Chapter.

How do I configure a Virtex E in mixed voltage environment?

Configuration Inputs

Configuration inputs function with 2.5- to 3.3-V input levels if V_{CCO} is 2.5 V or 1.8 V. In fact, if the input levels are 2.5 V, a 2.5-V V_{CCO} is preferred. The input buffers used during configuration are the single-ended input buffers sourced by V_{CCO}. The V_{IH} / V_{IL} levels obtained are a function of V_{CCO} (LVTTL, LVCMOS2, or LVCMOS18 as specified in the Virtex-E data sheet). As long as inputs meet V_{IH} / V_{IL} levels, configuration is accomplished. More margin is better.

Master Serial or SelectMAP Readback

For Master Serial mode, assuming the FPGA is interfacing to a 3.3-V SPROM, banks 2 and 3 need to be at 3.3 V during configuration.

For SelectMAP mode, data I/Os are LVTTL (or LVCMOS2, or LVCMOS18, depending on the V_{CCO} level) until the part has been configured and is operational (gone through start-up). At that time, I/Os become whatever they were configured to be in the design. If the programming voltage level is different than user IO levels, voltages must be switched.

Serial Slave and SelectMAP

For Serial Slave and SelectMAP, V_{CCO} can be any voltage Š≥1.8 V provided it meets the V_{IH} / V_{IL} levels of the input buffer that results (see above). Any pin that is a shared I/O, such as $\overline{INIT}$ and DOUT/BUSY should be pulled up to the V_{CCO} voltage on its bank (and DONE of course). The dedicated configuration and JTAG pins in the corners work fine as long as they're pulled up to at least V_{CCINT} (1.8 V) They can be safely pulled up to 3.6 V.

JTAG

JTAG inputs are independent of V_{CCO} and with 2.5- to 3.3-V input levels. TDO is sourced from V_{CCO_2} and should be 1.8, 2.5 or 3.3 V, depending on what the TDI of the next device can accept.

With a slow VCC power ramp, is it better to delay programming by holding PROGRAM Low or holding INIT Low?

It depends. Usually, holding $\overline{INIT}$ Low is cleaner, because holding $\overline{PROGRAM}$ Low can cause a higher current draw. For smaller devices, this is insignificant (~10 0mA), but is somewhat more for larger devices. However, $\overline{PROGRAM}$ is always noticed. Once $\overline{INIT}$ goes High, pulling it Low again (for example, V_{ccint} Low) has no effect. $\overline{INIT}$ can only delay configuration; it does not abort configuration or initiate re-configuration.

What are the connections for the $\overline{PROGRAM}$, DONE, and $\overline{INIT}$ lines in serial mode?

Connect the $\overline{INIT}$ lines to the PROM OE/RESET lines, so they all start when the FPGA stops clearing its configuration memory.

The DONE line need only be connected to $\overline{CE}$ of the first device. When DONE goes High, it de-asserts $\overline{CE}$ on the first PROM. This causes $\overline{CEO}$ to go High, pulling $\overline{CE}$ High on the next PROM, etc.

$\overline{\text{PROGRAM}}$ need only be connected to any $\overline{\text{CF}}$ line. A JTAG instruction can be loaded into any PROM to pulse $\overline{\text{CF}}$ and reconfigure the device. Refer to Appendix for more information on XC1800 PROMs.

Can only one Virtex device in a chain be re-programmed without affecting the other configured devices when all DONE pins are tied together?

Once the part has gone through the start-up sequence, the Done pin has no impact unless a device has the DriveDone=yes BitGen option selected. An FPGA with that option selected drives DONE High with a strong 12 mA pullup resistor, which is difficult to counter. Only use the DriveDone option for the last device in a daisy chain when the CCLK rate is fast. For this situation, a more appropriate scenario would be a SelectMAP configuration.

Configuring in the SelectMAP Mode

How do you configure multiple Virtex devices in the SelectMAP mode and have all Virtex devices become active at the same time?

Connect the CCLK, Data, $\overline{\text{WRITE}}$, BUSY, DONE, and $\overline{\text{INIT}}$ pins in parallel. Each Virtex device is loaded separately by asserting the $\overline{\text{CS}}$ pin of each device.

The Host processor follows these steps.

1. Set $\overline{\text{WRITE}}$ Low using this sequence:
   a. 1.Set $\overline{\text{CS}}$ of one device Low.
   b. 2. Send configuration data.
   c. 3. Deselect the device by setting its $\overline{\text{CS}}$ pin High.
2. Repeat Sequence for all other devices.
3. Disable Data Source.
4. Set $\overline{\text{WRITE}}$ High.
5. Configuration Completed - DONE goes High when all devices have released it.
6. The programming sequence for SelectMAP mode can be found in flow chart form in the Virtex data sheet.

Can Virtex Devices be daisy-chained in the SelectMAP Mode?

Daisy chaining is not possible in SelectMAP mode. The BUSY signal used in the SelectMAP mode is the same pin as DOUT, which normally provides downstream devices with data.

A parallel loading situation is possible. See "How do you configure multiple Virtex devices in the SelectMAP mode and have all Virtex devices become active at the same time?" on page 133.

What are the Initialization timing requirements for Virtex configuration in the SelectMAP mode?

There is some confusion involving timing of $\overline{\text{CS}}$ and the first write of a data word with Virtex devices. Following power-up, Virtex devices require three clock cycles for initialization. If data write is attempted earlier, no data is transferred.At first glance, it could appear that data isn't being written until the second rising CCLK after $\overline{\text{CS}}$ is asserted. The waveforms or the SelectMAP and Serial modes assume a generated bitstream is being loaded. This bitstream has 32 padding bits at the beginning, so if a bit or a byte (or two) are

ignored because of this initialization timing, there are no ill effects. See Chapter 2 of this guide for more details about Virtex configuration issues.

Does Virtex SelectMAP have dead cycles?

There are no SelectMAP dead cycles. Virtex has an internal 32-bit configuration bus that pulls data from an external configuration port. In some instances (50+ MHz CCLK and a slow BUSCLK) BUSY can be asserted.

Can you strobe the Virtex programming clock in JTAG, Slave Serial, or SelectMAP modes?

Yes, because Virtex has an internal configuration bus clock that is separate from the TCK / CCLK used at the IO interface.

During Virtex device configuration, it is possible to stop the CCLK either High or Low.

Can you strobe Virtex $\overline{CS}$ in SelectMAP?

Yes, but do not toggle $\overline{WR}$ at the same time, it might generate an Abort. $\overline{CS}$ is also used as an asynchronous control of the BUSY pin and in conjunction with $\overline{WR}$ for the data output pins.

When using SelectMAP, how is data generated for a microprocessor to configure the device?

With a microprocessor, the files would be either Hex or some readable text format. Use promgen to generate a hex files and pass each 8-bit work to the D_0 - D_7 pins. A Hex file is appropriate for either serial or SelectMAP configuration with the data stored as LSB-to-MSB (D_0 - D_7). In other words, D_0 is the MSB when the Hex file is used for the PROM Filed Formatter. The default for the Hex files is to swap bits (promgen -b -p hex -UO bitfile_name). When the option is de-selected, D_0 becomes the LSB. Within the SelectMAP interface, data is written a byte at a time. A sample word is shown in the diagram below. The top row indicates the chip pin names. The bottom row indicates the bit indices within a configuration word. Byte 0 loads first, followed by byte 1, *et cetera*. The MSB of each byte is loaded on pin D0. The LSB of each byte is loaded on pinD7.

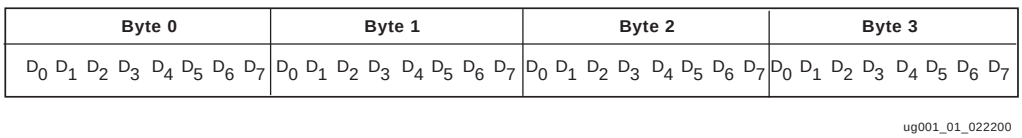

Figure 46: **SelectMAP Byte and Bit Ordering**

Configuring in the JTAG (Boundary Scan) Mode

How do you program multiple Virtex devices in a JTAG daisy chain?

If multiple Virtex devices are connected in a JTAG daisy chain (TDO of one device connects to TDI of another device, all TCK pins are connected together, and all TMS pins are connected together, it is possible to configure this entire daisy chain via the Virtex JTAG pins.

Both hardware and software procedures are involved.

First, prepare the VIrtex bitstreams for JTAG configuration. For each bitstream in the Virtex daisy chain, create the BIT file as follows:

```
bitgen -g startupclk: jtagclk designName.ncd
```

where: *designName***.ncd** is the name of the routed NCD file from Place and Route.

If you do not do this, the Virtex device accepts the bitstream, DONE goes High and $\overline{\text{INIT}}$ goes High, but the device does not respond. Using the -g option tells the Virtex device to use the JTAG clocks on the TCK pin to finish the configuration process.

If you do not want to use the TCK to clock the startup sequence, you must use the STARTUP_VIRTEX symbol to attach a specific internal clock net for the startup sequence. Bitgen must then be instructed to use the userclk for startup with the **-g:userclk** option.

If configuring a virtex device via JTAG after power-up, set the mode pins for the device to 101 (No pull-ups) or 001 (pull-ups). If you do not set these mode pins before power-up, and you then enter a configuration instruction, it is possible that the device might not configure correctly.

How do you program a single Virtex device via JTAG?

A Virtex device can be configured via its dedicated JTAG pins.

Prepare the bitstream. Use A1.5i / F1.5i or later software when creating a Virtex design to be loaded via JTAG.

1. Create a BIT file that uses the following bitgen option.

   ```
   bitgen -g startupclk:jtagclk designName.ncd
   ```

   where: *designName***.ncd** is the name of the routed NCD file.

   If you do not do this, the Virtex device accepts the bitstream, DONE goes High and INIT goes High, but the device does not respond. Using the **-g** option tells the Virtex device to use the JTAG clocks on the TCK pin to finish the configuration process.

   If you do not want to use the TCK to clock the startup sequence, you must use the STARTUP_VIRTEX symbol to attach a specific internal clock net for the startup sequence. Bitgen must then be instructed to use the userclk for startup with the **-g:userclk** option.

   Failure to define a startup clock results in incomplete configuration.

2. Prepare the Virtex device for JTAG configuration. A Virtex device can be initially configured or re-configured via JTAG.

   Initial Configuration - Set the mode pins of the Virtex device to 101 (JTAG without pull-ups) or 001 (JTAG with pull-ups) before power-up to avoid accidental configuration in another mode.

   Re-configuration - toggle the Test Access Port (TAP) and enter the CFG_IN instruction.

3. Load the bitstream.

   a. At power-up, place a “one” on TMS and clock TCK five times, taking the TAP through the Test-Logic-Reset (TLR) state.

   b. Move to the Shift-IR state.

   c. Load the CFGG-IN instruction.

   d. Go to the Shift-IR state and shift in the Virtex bitstream. After the entire bitstream has been shifted in, go to the TLR state.

   e. Go to Shift-IR and load the JSTART instruction. Then go to SHIFT-DR and apply a minimum of ten clocks to TCK.

   f. Return to TLR. The device is now functional.

How do you validate JTAG configuration of a single Virtex device?

Troubleshooting JTAG configuration of an individual Virtex device is straightforward except that bitstream errors are not flagged automatically. Note that the TAP is always active on Virtex devices and can be used for troubleshooting. DONE functions exactly the same in the JTAG configuration mode as it does in the non-JTAG configuration mode and can be monitored to validate successful downloading of the bitstream into the target device.

1. If DONE is Low, there are three possible reasons:
   a. The entire bitstream has not been sent.
   b. The entire bitstream has been sent, but the start-up sequence is not finished.
   c. An error occurred.
2. If the bitstream was created with the **-g** option for jtagclk, *and* the JSTART instruction was executed, *and* DONE is Low, it is possible that a bitstream error occurred. In non-JTAG configurations of Virtex devices, this type of error is flagged by the status of the $\overline{\text{INIT}}$ pin, but in JTAG configurations the $\overline{\text{INIT}}$ pin is not used for status reporting. However, it is possible to check status of the internal $\overline{\text{INIT}}$ signal to see if there is a bitstream error.

   If a Virtex JTAG configuration is complete, and the DONE pin stays Low, follow these steps to determine if there is a bitstream error.
   a. Move TAP to the TLR state.
   b. Load the CFG_IN instruction.
   c. Go to the Shift-DR state and shift in this 64-bit pattern.

   ```
   2800E00100
   0010 1000 0000 0000 1110 0000 0000 0001 0000 0000
   000000
   0000 0000 0000 0000 0000 0000
   ```

   d. Shift in data starting from the underlined bit.
   e. Load the CFG_OUT instruction.
   f. Go to Shift-DR and clock TCK 32 times while reading TDO.
   g. The data is the content of the STATUS register. The last bit out is a “one” if there is a CRC error. With a successful configuration, the 32 bits are as follows.

   ```
   0000 0000 0000 0000 0 1 1 111 1 1 0 1 0 1111 0_
   ```

   The CRC_ERROR bit occurs at the underlined position.
3. If DONE is High, the Virtex device received the bitstream, and it is not corrupted.

How do the JTAG pins voltage levels react when different voltages are applied to V_{CCO} and V_{CCIO} on Virtex devices?

For Virtex devices, the inputs are driven by V_{CCINT}, so V_{CCO} is irrelevant. However, the output TAP pin TDO drives the rail to the value of the V_{CCO} supply on its bank. Set this bank (usually bank 2) to 3.3 V.

For Virtex-E devices, no V_{CCO} requirement exists for the JTAG input pins (TDI, TMS, TCK).They function with 1.8-, 2.5-, or 3.3-V input levels.

The output pin (TDO) is sourced from V_{CCO} in bank 2. For proper 3.3-V LVTTL operation, supply the bank with 3.3 V. However, if the receiving TDI pin can accept a lower threshold voltage (2.5 or 1.8 V), this voltage can be applied for the bank.

What precautions are necessary with JTAG programming using the Parallel Cable?

Do not connect both sets of flying leads on the parallel cable when using the JTAG mode with the JTAG Programmer software (the Slave / Serial mode with the Hardware Debugger software works fine). With the JTAG mode and the JTAG Programmer software, do the following.

1. Set BitGen options to use JTAGCCLK rather than the default CCLK for the start-up sequence.
2. Do not connect the cable $\overline{\text{PROGRAM}}$ lead, because this signal is held Low, overriding the default weak pullup in the FPGA.
3. When re-configuring with JTAG, either pulse $\overline{\text{PROGRAM}}$ Low or cycle power (JTAG Programmer limitation for Virtex, SpartanXL, XC4000XLA - silicon limitation for 5-V Spartan).

What are the V_{CCO} requirements for JTAG I/O pins?

For Virtex and Virtex-E series, the JTAG input pins (TDI, TMS, TCK) respond to 2.5-V or 3.3-V input signals. The output pin (TDO) is sourced from V_{CCO} in bank 2. For proper LVTTL operation, this bank should be supplied with 3.3 V. However, if the receiving TDI input can accept a lower voltage (2.5 V or 1.8 V), the lower voltage level can be applied to this bank.

During a JTAG configuration of a Virtex device, how is the status of the DONE and $\overline{\text{INIT}}$ signals checked?

The status of the DONE and $\overline{\text{INIT}}$ signals is observed by loading their values into the Virtex JTAG instruction register. This is accomplished by performing a CAPTURE-IR and shifting out four bits from the instruction register. The first bit shifted out is a one and the second bit shifted out is a zero. The third and fourth bits shift out indicating the status of the DONE and $\overline{\text{INIT}}$ signals, respectively.

Can Virtex device TAP pins be used as regular I/O?

Virtex TAP pins are fully dedicated boundary scan pins and cannot be used as regular I/Os.

Is any power saving realized by tying mode pins to the JTAG mode?

None at all.

Does toggling the $\overline{\text{PROGRAM}}$ pin reset the TAP?

The TAP is reset for Virtex, Virtex-E, Virtex-EM, Spartan, and SpartanXL devices. The $\overline{\text{PROGRAM}}$ pin is a non-compliance pin.

Can a clock other than the JTAGCLK be used to initiate the start-up sequence?

The start-up sequence is the same for all programming modes and can use a USERCLK, CCLK, or JTAGCLK. Using CCLK makes little sense, but it can be done. Only outputs are 3-state during configuration, so input signals can still get into the chip (if not, CLKDLL could not lock pre-DONE).

Can the $\overline{\text{INIT}}$ or $\overline{\text{PROGRAM}}$ pins be controlled from JTAG?

The $\overline{\text{INIT}}$ pin can be toggled; the $\overline{\text{PROGRAM}}$ pin cannot.

Is a TRST pin planned for future devices?

Not at this time.

Is there a SimPrim model for the JTAG port and BSCAN Macro?

BSCAN is not a Macro, but a primitive. Currently there is no way to simulate the JTAG port because there are no simulation models for the TAP port. There is a black box UNISIM model for BSCAN primitives, but the cell is not included in during back annotation. The Xilinx tool TRCE does not have a timing model for the internal connections.

Can the JTAG Programmer software access the Status Register to indicate that DONE is High instead of sending out a bogus message?

With v 2.1iSP3, JTAG Programmer software polls the status register and indicates the status of DONE. Apparently this is also an option for non Virtex parts if the proper switches are set in Bitgen.

Is Jam/Staple supported?

Emerald is seen as a direct output.

Can Bitgen –g startupclk:jtagclk be used with Spartan devices?

This option is used with Virtex and Virtex derivatives only (*for example,* Spartan II). Future versions of the software can automatically select this when JTAG configuration is selected.

Is BSDL a subset of a specific VHDL version?

BSDL should be compatible with all VHDL versions.

Will XSVF support continue and what is the impact on large device file size?

XSVF support is expected to continue indefinitely. XSVF large device file size should approximately 1/4 the size of the file for the same device in SVF format.

Why are multiple SVF files generated?

The JTAG Programmer generates an SVF file for each device in a chain because of testing requirements. With multiple devices in one SVF file, test vectors are generated incorrectly.

Could the JTAG Programmer provide a test pattern?

The JTAG Programmer is used for device programming, and the addition of test pattern generation is not planned. JTAG test software can be used for this purpose.

Does Xilinx provide configured BSDL files for post configuration devices?

To create a BSDL file that represents a configured Xilinx device, the BSDL file must be modified based on the design loaded in the device.

Most users do not need the post-configuration BSDL representation of their part. If this is the case, the user can use the pre-configured file without any modifications. These files can be found by selecting the BSDL link.

http://support.xilinx.com/support/software.htm

The released BSDL files match JTAG behavior prior to completion of configuration. The BSDL file without modifications has all
I/Os defined as 3-state bidirectional pins.

After configuration, most pins are usually configured as an input or an output, not 3-state stable bidirectional. If a BSDL file matching the behavior of the chip after the design has been loaded is desired, some alterations are necessary. The following steps create such a BSDL file for Virtex.

1. Enable USER instructions as appropriate (see below).
2. Set disable result of all pads as configured.
3. Set safe state of boundary cells as necessary.
4. Rename entity if necessary to avoid name collisions.
5. Modify USERCODE value in USERCODE_REGISTER declaration.

   To create a Virtex post-configuration BSDL file (quoted exactly from actual BSDL files), enable USER instructions as appropriate.

   If you do not use the user instructions, no modifications are needed. If you use either USER1 or USER2, include appropriate entries in the REGISTER_ACCESS description. For more information, consult Supplement B to IEEE Std. 1149.

6. Set disable result of all pads as configured.

   The disable result is the value of the signal when it is disabled. It is specified in the BOUNDARY_REGISTER section of the BSDL file.

   If pullup = NO and pulldown = NO, then the disable result value is Z

   If pullup = YES and pulldown = NO, then the disable result value is PULL1.

   If pullup = NO and pulldown = YES, then the disable result value is PULL0.

   Unused pads are PULL0.

   Following are three BSDL code examples of the BOUNDARY_REGISTER section. For this example, code has been used from Revision 1.2 of xcv100_pq240.bsd. This is the untouched code for pin 57 as a bidirectional pin.

   Attribute BOUNDARY_REGISTER of XCV100_PQ240: entity is:

```
-- cellnum (type, port, function, safe[, ccell, disval, disrslt])
" 0 (BC_1, *, controlr, 1)," &
" 1 (BC_1, PAD60, output3, X, 0, 1, PULL0)," &
" 2 (BC_1, PAD60, input, X)," &
" 3 (BC_1, *, controlr, 1)," &
" 4 (BC_1, PAD59, output3, X, 3, 1, PULL0)," &
" 5 (BC_1, PAD59, input, X)," &
"  6 (BC_1, *, internal, X)," &
" 7 (BC_1, *, internal, X)," &
" 8 (BC_1, *, internal, X)," &
" 9 (BC_1, *, controlr, 1)," &
" 10 (BC_1, PAD57, output3, X, 9, 1, PULL0)," &
" 11 (BC_1, PAD57, input, X)," &
" 12 (BC_1, *, controlr, 1)," &
```

   If pin 57 is configured an input, the code is modified as follows:

```
" 9 (BC_1, *, internal, 1)," &
  " 10 (BC_1, *, internal, X)," &
" 11 (BC_1, PAD57, input, X)," &
```

   If pin 57 is configured an output, the code is modified as follows:

```
" 9 (BC_1, *, internal, 1)," &
" 10 (BC_1, PAD57, output2, X),"
  " 11 (BC_0, PAD57, observe_only, X)," &
```

   Repeat these modifications for every configured pin in the design.

7. Set safe state of boundary cells as necessary.

The safe bit supplies a value to be loaded when board-level test generation software might otherwise choose a random value (it is not being forced). The safe bit can be used for several purposes.

a. The value in a control cell that turns off associated drivers

b. The value that an output should have during INTEST to minimize driver current

c. The preferred value to present to on-chip logic at a component input during EXTEST

Control cells with the proper value (corresponding to the pad tri-state) do not need to change.

Input/Output values are design dependent, so application knowledge is necessary to set them. (This cannot be automatically set based solely on the NCD file.)

8. If necessary, rename entity to avoid name collisions.

   If the entity name used in the BSDL file collides with other BSDL or VHDL files, rename the entity.

9. Modify USERCODE value in USERCODE_REGISTER declaration.

   Fill in USERCODE value supplied during bitgen if using this function.

What is the JTAG ID code format and what are the ID codes for Virtex-E devices?

The Virtex JTAG ID Code register has the following format.

```
3322 2222222 211111111 110000000000
1098 7654321 098765432:109876543210
vvvv:fffffff:aaaaaaaaa:ccccccccccc1
```

where v is the version of the mask set

	Final Prod	ES Only
V50 =	0000	0000
V100 =	0010	0001 and below
V150 =	0001	0000
V200 =	0010	0001 and below
V300 =	1000	0111 and below
V400 =	0001	0000
V600 =	0001	0000
V800 =	0010	0001 and below
V1000 =	0100	0011 and below

Note: Until the Production mask set has been qualified, the devices will be marked with the -ES label.

f is the family code = 0000011 = 0x03

a is the number of array rows in the part

```
V50     = 16 = 000010000 = 0x010
v100    =  20    = 000010100 = 0x014
v150    = 24 =     000011000 =  0x018
v200    =   28   =       000011100 =  0x01C
v300    =   32   =       000100000 = 0x020
v400    =   40   =       000101000 =  0x028
V600    = 48 = 000110000 = 0x030
V800    = 56 = 000111000 = 0x038
V1000   = 64 = 001000000 = 0x040
```

c is the company code = 00001001001 = 0x049*

Notes:

1. * Since the last bit of the JTAG idcode is always one, the last three hex digits appear as 0x093.

The ID Codes assigned to the Virtex-E FPGAs are shown below.

Table 93: **Virtex-E Device ID Codes**

FPGA	IDCODE
XCV50E	v0A10093h
XCV100E	v0A14093h
XCV200E	v0A1C093h
XCV300E	v0A20093h
XCV400E	v0A28093h
XCV600E	v0A30093h
XCV1000E	v0A40093h
XCV1600E	v0A48093h
XCV2000E	v0A50093h
XCV2600E	v0A5C093h
XCV3200E	v0A68093h

What PROM formats are supported?

Xilinx supports three PROM file formats for programming parallel proms and Xilinx devices--Intel MCS, Motorola EXORmacs, and Tektronix HEX.

Intel MCS-86 Hexadecimal Object - File Format Code 88

The Intel 16-bit Hexadecimal Object record format has a 9-character (4-field) prefix that defines the start of the record, byte count, load address, and record type, as well as a 2-character checksum suffix. The four record types are:

1. 00 = Data Record
2. 01 = End Record (signals end of file)
3. 02 = Extended Address Record (provides the offset to determine the absolute destination address)
4. 03 = Start Record (ignored during input and not sent during output by Data I/O translator firmware)

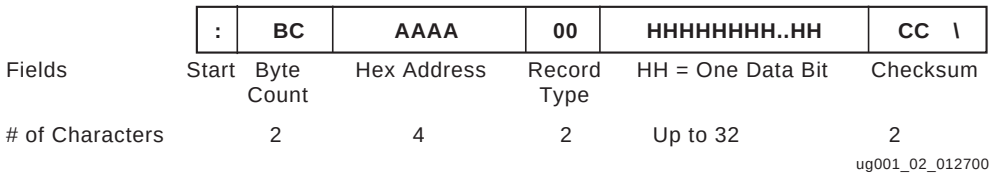

Figure 47: **00 = Input Data Record**

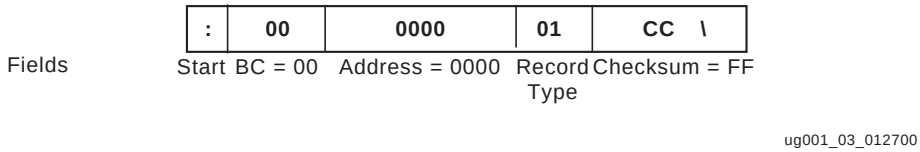

Figure 48: **01 = End Record**

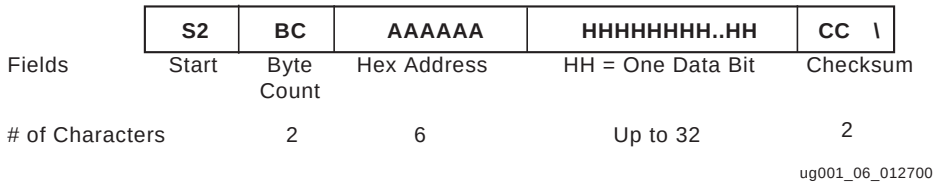

Figure 49: **Data Record (if address field has six characters)**

03 - Start Record (not used by Data I/O firmware)

The Checksum is the twos complement of the binary summation of the preceding bytes in the record (including the byte count, address, and any data bytes) in hexadecimal notation. The extended address record (type 02) defines bits 4 to 19 of the 20-bit segment base address. This address is added to subsequent data record addresses to provide the absolute address. This record can appear randomly in the file, but for this application it is the initial record.

Notes:

1. Always specify the address offset when using this format, even when the offset is zero.

The Data I/O firmware forces the record size to 16 (decimal) if the record size specified is greater than 16.

Motorola EXORmacs File - Format Code 87

A Motorola data file can begin with a sign-on record, but for this application it is ignored. Data records have an 8- or 9-character prefix and a 2-character checksum suffix.

S1	BC	AAAA	HHHHHHHH..HH	CC \
Start	Byte Count	Hex Address	HH = One Data Bit	Checksum
	2	4	Up to 32	2

Figure 50: **Data Record (if address field has four characters)**

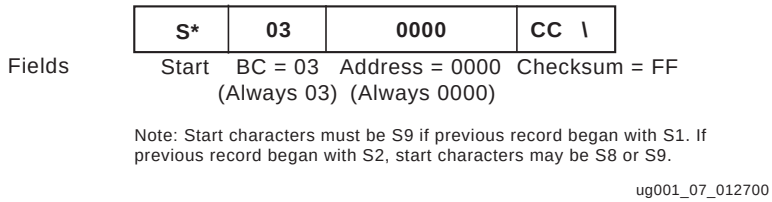

Figure 51: **End-of-File Record**

Byte Count is the number of data bytes in the record plus three for a four hex character address (or plus four for a six hex character address) for checksum and address.

The checksum is the ones complement of the binary summation of the preceding bytes in the record (including byte count, address, and data bytes) in hexadecimal notation.

Tektronix Hexadecimal - File Format Code 86

The Tektronix Hexadecimal format for a data file consists of a 9-character prefix (start character, address, byte count, and prefix checksum) followed by data byte(s) and ending with a 2-character data checksum. The end-of-file record has only control characters used to signal the end of transmission, a byte count, and a verification checksum.

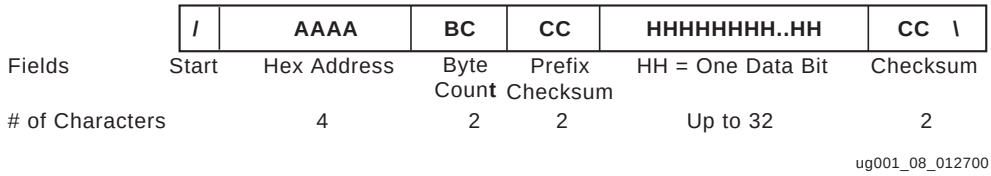

Figure 52: **Data Record**

The prefix checksum is the 8-bit sum of the 4-bit hexadecimal value of the six digits that make up the address and byte count. The data checksum is the 8-bit sum, modulo 256, of the 4-bit hexadecimal values of the digits that make up the data bytes.

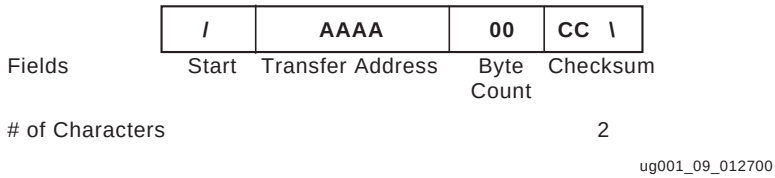

Figure 53: **End-of-FIle Record**

What are multi-purpose pins and how can they be reserved for configuration (prohibited for use as I/O)?

There are several special-purpose pins available after configuration for I/O, and most require direct instantiation (*for example*, TDO, MD1, etc.) Specific primitives must be used to utilize these pins, and they can not be prohibited or pin locked.However, multi-purpose pins can be used by PAR, and these can be prohibited in the UCF file as in the following example.

```
CONFIG PROHIBIT = P36;
```

For the Virtex device series, the following multi-purpose pins can be prohibited (package dependent).

Table 94: **Virtex Series Multi-purpose Pins Available for Configuration**

Function	PQ240	HQ240	BG256	BG352	BG432	BG560
INIT	P123	P123	U18	AD2	AJ2	AH5
BUSY / DOUT	P178	P178	D18	E4	D3	D4

What is the polarity of DriveDone in the COR register?

Zero (0): DONE is Open Drain (default).

One (1): DONE is actively driven High.

What are the timing characteristics for the Virtex JTAG TAP controller?

Virtex timing (advanced) for the JTAG TAP is as follows:

- TMS or TDI set-up with respect to TCK: 4 ns
- TMS or TDI hold with respect to TCK: 2 ns
- TCK-to-TDO (TDO clocked on falling edge of TCK): 11 ns
- Maximum TCK frequency: 33 MHz

Cable Interfaces

There is confusion about the names of Xilinx configuration (download/programming) cables. Which cable is which?

Currently, there are three Xilinx cables useful for configuration (downloading) known by various official and unofficial names and labels.

Table 95: **Xilinx Configuration (Download) Cables**

Part	Cable Name	Also Known as . . .	Labels
DLC4	XChecker Cable	Serial Cable	Slow, Old
DLC5	Parallel Cable III	JTAG Cable, Parallel Cable	
Cable 98	MultiLINX Cable	USB Cable	New

What combinations of cables, devices, and software does Xilinx support?

Xilinx supports various combinations cables, devices, and software, depending on the device series.

Table 96: **MultiLINX Cable With Hardware Debugger v.2.1i or later**

	Configuration Mode			
Family	Slave/Serial	SelectMAP	Readback/Verification	JTAG
Spartan	√		√	√
Spartan XL	√		√	√
XC3000A	√			
XC4000E	√		√	√

Table 96: **MultiLINX Cable With Hardware Debugger v.2.1i or later *(Continued)***

	Configuration Mode			
Family	**Slave/Serial**	**SelectMAP**	**Readback/Verification**	**JTAG**
XC4000EX	√		√	√
XC4000XL	√		√	√
XC4000XLA	√		√	√
XC4000XV	√		√	√
Virtex	√	√	√	√
Virtex-E	√	√	√	√

Table 97: **MultiliNX Cable With JTAG Programmer v2.1i SP3 or later**

	Configuration Mode			
Family	**Slave/Serial**	**SelectMAP**	**Readback/ Verification**	**JTAG**
XC1804				√
XC9500			√	√
XC9500XL			√	√
XC9500XV			√	√
Spartan				√
Spartan XL				√
XC4000E				√
XC4000EX				√
XC4000XL				√
XC4000XLA				√
XC4000XV				√
Virtex				√
Virtex-E				√

Table 98: **Parallel Cable III (JTAG Cable) With Hardware Debugger v2.1i or later**

	Configuration Mode			
Family	**Slave/Serial**	**SelectMAP**	**Readback/ Verification**	**JTAG**
Spartan	√			
Spartan XL	√			
XC3000A	√			
XC4000E	√			
XC4000EX	√			
XC4000XL	√			
XC4000XLA	√			

Table 98: **Parallel Cable III (JTAG Cable) With Hardware Debugger v2.1i or later**

	Configuration Mode			
Family	**Slave/Serial**	**SelectMAP**	**Readback/ Verification**	**JTAG**
XC4000XV	√			
Virtex	√			
Virtex-E	√			

Table 99: **Parallel Cable III (JTAG Cable) With JTAG Programmer v 2.1i SP3 or later**

	Configuration Mode			
Family	**Slave/Serial**	**SelectMAP**	**Readback/ Verification**	**JTAG**
XC1804				√
XC9500				√
XC9500XL				√
XC9500XV				√
Spartan				√
Spartan XL				√
XC4000E				√
XC4000EX				√
XC4000XL				√
XC4000XLA				√
XC4000XV				√
Virtex				√
Virtex-E				√

Table 100: **XChecker Cable With Hardware Debugger v2.1i or later**

	Configuration Mode			
Family	**Slave/Serial**	**SelectMAP**	**Readback/ Verification[1]**	**JTAG**
Spartan	√		√	√
Spartan XL	√		√	√
XC3000A	√			
XC4000E	√		√	√
XC4000EX	√		√	√
XC4000XL	√		√	√
XC4000XLA	√		√	√
XC4000XV	√		√	√

Table 100: **XChecker Cable With Hardware Debugger v2.1i or later *(Continued)***

	Configuration Mode			
Family	**Slave/Serial**	**SelectMAP**	**Readback/ Verification[1]**	**JTAG**
Virtex	√	√	√	√
Virtex-E	√	√	√	√

Note 1: With configuration bitstreams of 250,000 bits or less.

Table 101: **XChecker Cable With JTAG Programmer v2.1i SP3 or later**

	Configuration Mode			
Family	**Slave/Serial**	**SelectMAP**	**Readback/Verification**	**JTAG**
XC1804				√
XC9500			√	√
XC9500XL			√	√
XC9500XV			√	√
Spartan				√
Spartan XL				√
XC4000E				√
XC4000EX				√
XC4000XL				√
XC4000XLA				√
XC4000XV				√
Virtex				√
Virtex-E				√

What cables can I use for programming Virtex devices?

The XChecker and Parallel Cable (DLC5) can be used to configure Virtex in both Slave-Serial and JTAG (boundary scan) modes. For JTAG mode, use JTAG Programmer v.2.1i or later.

Using the XChecker Cable with Virtex Devices

The XChecker cable contains a 5-V FPGA that requires 5 V V_{CC}. The simplest method is to use a 3.3V XChecker power adaptor which guarantees proper operation. However, the following two methods are also viable.

1. Supply the XChecker cable with a 5-V supply with common ground and connect the remaining pins directly.
2. Simply connect the XChecker cable to a 3.3-V supply and connect the remaining pins directly.The 3.3-V supply generally provides just enough power for the XChecker FPGA to operate. As long as the output signals from the board to the XChecker cable reach 3.3 V, this is high enough to register a TTL logic “one” (1). V_{CCO} must be set to 3.3 V for Banks 2 and 3 with V_{CCINT} set to 2.5 V.

Can I use the Parallel Cable with Virtex Devices?

The parallel cable can be directly connected to a Virtex device with V_{CCO} set to 3.3 V for Banks d2 and 3, and V_{CCINT} set to 2.5 V.

Note: When using either download cable, set V_{CCO} for Banks 2 and 3 to a 3.3-V standard to enable the Virtex device to drive signals back to the cable.

What voltage standards are supported by Xilinx configuration cables?

Three Xilinx configuration cables presently available are the MultiLINX Cable, the Parallel Cable III (JTAG), and the XChecker Cable.

- MultiLINX Cables configure 5-, 3.3-, and 2.5-V devices. Default levels are 3.3 V and 5 V.
- The Parallel Cable III (JTAG) configures 5-, 3.3-, and 2.5-V devices by simply connecting the V_{CC} pin of the cable to the desired power supply.
- XChecker Cables configure 5-V or 3.3-V devices. The default level is 5 V. To configure a 3.3-V device, use the 3.3V XChecker power adaptor.

What power supply voltages are acceptable for configuration using the JTAG (boundary scan) cable and programming mode?

The Parallel Cable III (JTAG cable) operates with 5-, 3.3- or 2.5-V power supplies, but the configuration voltage requirements of the device being programmed (*for example,* 3.3 V for the XC4000XV and Virtex series, 2.5 V for the Virtex-E series) must be considered.

When using the MulitLINX cable with Hardware Debugger v2.1 software, why would the system or application crash?

If the Hardware Debugger software loses communication with the MultiLINX cable while a download is in progress, the application or system can crash or freeze. This can be caused by disconnecting the cable or the cable power supply. Always wait for all operations to complete before powering down or removing the MultiLINX cable.

When programming an FPGA using Hardware Debugger via the parallel port, what do I do if the program pin goes Low after configuration, erasing the FPGA?

1. To fix this problem, set the following environment variable:
   ```
   SET XIL_HWD_PCAB_FIX=1
   ```
   Environment variables can be set in the **`autoexec.bat`** file for Windows 95/98. The exact line above can be added to the file using the **`sysedit`** program.

   In Windows NT, right-click on the “My Computer” desktop icon, select Properties, then select the Environment tab. The Variable is 'XIL_HWD_PCAB_FIX', Value is '1'. Reboot the PC.
2. Disconnect the parallel cable after configuration. This is effective when the program pin glitches several minutes after configuration is completed.

Since the $\overline{\text{INIT}}$ pin indicates when the configuration memory cleared, why doesn't the parallel cable have an $\overline{\text{INIT}}$ pin to determine this status?

1. The parallel cable pulses $\overline{\text{PROGRAM}}$ Low and waits a prescribed amount of time before sending DATA to the device. This wait period is longer than the time required to clear configuration memory so there is no reason to check the $\overline{\text{INIT}}$ pin.

2. The $\overline{INIT}$ pin is NOT monitored by the cable for errors. The cable waits for DONE to go High and times out. Check to see if $\overline{INIT}$ is low separately to determine if a frame error caused a problem.

What interface connections to a host system are supported by the MultiLINX Cable?

The MultiLINX cable supports RS232 and USB interface connections to a host system. The Hardware Debugger v 2.1i software supports RS232 connections for Windows NT, 95X, and 98, as well as Solaris. The software supports USB connections for Windows 95C and 98.

What happens when MultiLINX baud rate is changed?

Changing the baud rate for the MultiLINX Cable on COM ports resets the FPGA in the cable. Wait to continue until a momentary delay required to reconfigure the cable occurs.

Can you use a parallel-to-serial adapter with the Multilinx cable?

No.

Is it possible to use an extension with Xilinx configuration cables?

Using a cable extension depends on the port being used (Parallel, Serial, USB). Extending the cable can cause data corruption (due to signal attenuation) and should be avoided. Follow these guidelines.

- With serial ports, an extension up to five feet is normally acceptable. The RS-232C standard allows cable lengths of up to 15 meter. Use only a firmly connected quality cable away from possible noise sources.
- With parallel ports, do not use an extension. The parallel port drivers can not support it. Signals attenuate and data become corrupted. Also, for PC users, do not connect the cable through a software programming key.

Can the MultiLINX Cable be used for configuring multiple devices?

The MultiLINX cable was designed as a configuration prototype tool for up to four Xilinx FPGAs connected with flying wires. To use the MultiLINX cable to drive large loads, use buffers to insure that signals are not distorted. The following guidelines are provided.

Table 102: **Worst-Case Loads for Each Flying Wire**

Maximum Capacitive Load	100 pF
Maximum Pullup / Pulldown Resistance	500 Ω

Table 103: **3. 3-V or 5-V Operation**

V_{OH} (Max)	V_{OH} (Min)	V_{OL} (Max)	V_{OL} (Min)	V_{IH} (Max)	V_{IH} Min)	V_{IL} Max)	V_{IL} (Min)
5 V	2.4 V	0.4 V	0 V	5 V	1.65 V	1.5 V	0 V

Table 104: **2.5-V Operation**

V_{OH} (Max)	V_{OH} (Min)	V_{OL} Max)	V_{OL} (Min)	V_{IH} (Max)	V_{IH} (Min)	V_{IL} (Max)	V_{IL} (Min)
2.5 V	2.0 V	0.4 V	0 V	5 V	1.65 V	1.5 V	0 V

Chapter 7

Getting Started With the MultiLINX System

This chapter serves as an introduction to the MultiLINX™ cable set, including features, a description, applications, software support, and how to integrate cable access to a board. For additional information on the MultiLINX cable set and other Xilinx hardware products, please refer to the **Hardware User Guide**.

Introduction

The MultiLINX cable set shown in the Figure 54 is a new peripheral hardware product released by Xilinx in 1999. It is used primarily for downloading configuration and programming data from a host computer to Xilinx FPGAs and CPLDs in a target system.

The MultiLINX system supports a USB (Universal Serial Bus) interface with communication speeds up to 12 Mb/s, reducing download times by a factor of 120X relative to previous cables. The MultiLINX cable set includes all appropriate flying leads for multiple configuration mode support. In addition, MultiLINX cable sets support readback modes such as verification, capture, and the Virtex SelectMAP interface, providing quick and easy functional verification of programmable logic applications.

MultiLINX cable internal hardware is upgraded via software, facilitating the addition of new cable features and simplifying support. Upgrades are completely seamless and invisible to users.

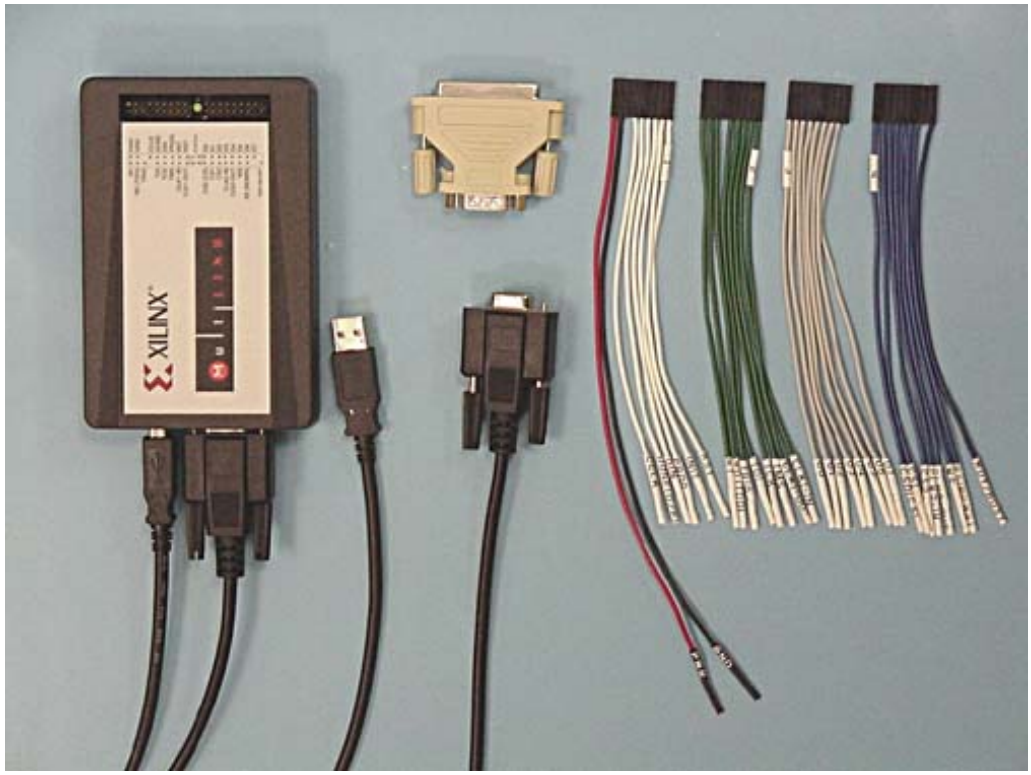

Figure 54: **MultiLINX Cable Pod and Attachments**

Obtaining a MultiLINX System

Alliance Series™ and Foundation Series™ 2.1i software packages are shipped with a mail-in form to purchase a MultiLINX cable set. Cable sets can also be purchased over the internet or via credit card orders by contacting the Xilinx Customer Service Hotline.

MultiLINX System Software Support

Alliance and Foundation version 2.1i is the first version of Xilinx software to support the MultiLINX cable set. Additionally, a new software tool, ChipScope, is also now available for use with the MultiLINX system. Please refer to the Table 105 for a cross reference of software feature support.

Table 105: **Software Support Cross-Reference**

S/W	Family	Configuration	Verification	Capture
Hardware Debugger 2.1i	Virtex	SS/SM	SM	NS
	Spartan	SS	SS	NS
	XC4000	SS	SS	NS
	XC3000	SS	NS	NS

Table 105: **Software Support Cross-Reference *(Continued)***

S/W	Family	Configuration	Verification	Capture
ChipScope 1.0	Virtex	SM	SM	SM
	Spartan	N/A	N/A	N/A
	XC4000	N/A	N/A	N/A
	XC3000	N/A	N/A	N/A

Notes:

SS = Slave Serial, SM = SelectMAP,

NS = Not Supported, NA = Not Applicable

Hardware Debugger 2.1i

The Hardware Debugger program is a standard feature for the Alliance and Foundation series software packages. The Hardware Debugger version 2.1i currently supports the use of the MultiLINX cable set for FPGA Configuration and Verification. XC4000 and Spartan Series FPGAs can be configured and verified serially through the MultiLINX cable set, refer to the Table 106. Virtex series FPGAs can be configured in the serial mode, and can be configured as well as verified using the SelectMAP interface. The SelectMAP interface uses a byte-wide parallel data bus.

The USB connection to the MultiLINX cable set is supported on Windows 98 platforms. The RS232 serial connection is supported on Windows 95/98/NT and Solaris/HP UNIX platforms.

ChipScope

The ChipScope functional verification tool is currently sold separately through the Xilinx web site. This program uses the MultiLINX cable set exclusively and currently only supports Virtex series FPGAs. ChipScope uses the SelectMAP interface of Virtex to support Configuration, Verification, and Internal Logic State Capture. Capture allows for the functional verification and debugging of the FPGAs' configured design.

ChipScope supports the high speed USB interface to the MultiLINX cable set on Windows 98 platforms and the RS232 connection on Windows 95/98/NT platforms. UNIX support is not available.

JTAG Programmer

JTAG Programmer software 2.1i does not support the MultiLINX cable set. JTAG Programmer software is a standard feature of the Alliance Series™ and Foundation series™ software packages. JTAG Programmer in WebPack 2.1WP5.x supports the MultiLINX cable set. This version can be downloaded from http://support.xilinx.com/support/software.htm.

The MultiLINX System

The MultiLINX cable set comes complete with the MultiLINX pod, a USB cable, a 9-pin communication port RS232 serial cable with two 25-pin communication port adapters (male and female conversion), four sets of flying leads, warranty form, and connection guide.

Connecting to the Host Computer

The MultiLINX cable set pod can be connected to the RS232 serial cable or the USB cable. Only one cable should be used at a time.

The RS232 cable can be connected directly to the 9-pin communication port of a PC. The 25-pin adapter allows for connections to the 25-pin serial communication port of a PC or UNIX Workstation.

The USB cable is keyed for a “hub” connection at one end (rectangular) and a “function” connection at the other end (square). The square end connects to the MultiLINX cable set pod. The rectangular end can be connected to the USB 1 or USB 2 port of a PC or to a USB Hub. Most PCs today come with USB ports included on the motherboard. If a PC does not have a USB port, it can be possible to add a USB card to the system. Consult a PC manufacturer for accessories.

Connecting to a Target System

The MultiLINX cable set has four sets of flying-lead connectors that are used to connect to a target system. The lead ends fit standard 25-mil square header pins. Such pins are commercially available in either Wire-wrap or Solder-end style in break-away sections with 1/10 in. spacing. The target end of the leads are loose, so board placement of the header pins is not relevant as long as they are no more than a few inches from each other. Extending the leads is possible, but can introduce noise which can interfere with cable set operation.

The PWR and GND leads can either be connected to the target system or a separate power supply; However, GND should always be common between the target system and the MultiLINX cable set for proper I/O communication. The MultiLINX cable set requires between 2.5 V and 5 V on the PWR lead and 2 W. For more information on the operating specifications of the MultiLINX cable set, consult the MultiLINX data sheet.

The I/O and configuration port connections support LVTTL I/O standard for either 5V, 3.3V, or 2.5V specifications. Current FPGAs and CPLDs use the LVTTL 3.3V standard. The following sub-sections provide the required lead connections for supported configuration interfaces. In all cases, the PWR and GND leads must be connected to a power source for the cable set to function properly.

Slave Serial Connection

The Slave Serial Configuration Mode for any FPGA requires the use of the following leads:

- PROG
- CCLK
- DONE
- INIT
- DIN

These connections are shown in Figure 55. For more information on the Slave Serial Configuration Mode please consult the data sheet for the target FPGA family (e.g., SpartanXL).

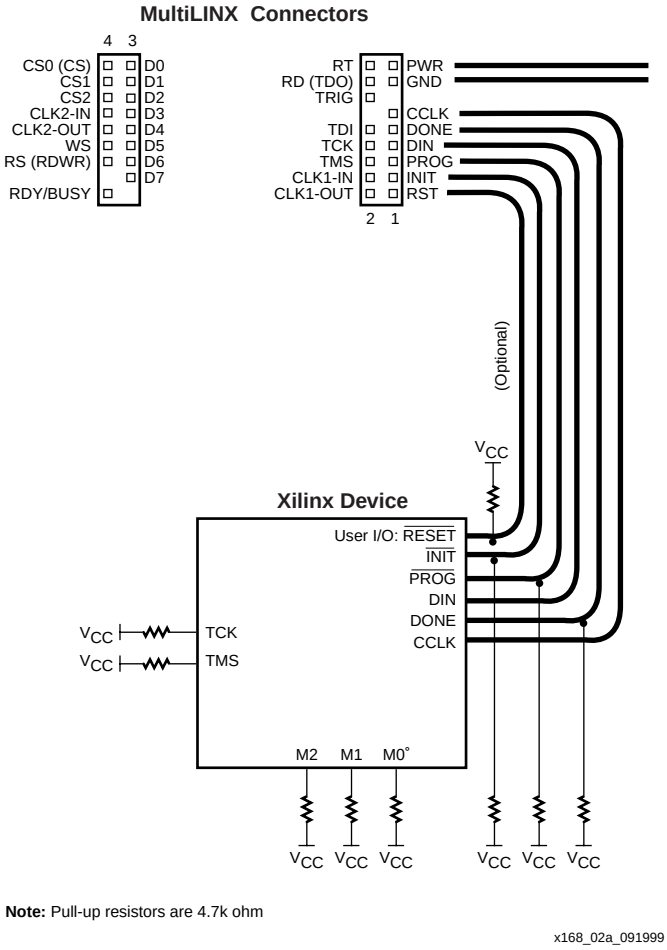

Figure 55: **Slave Serial Configuration**

Additionally, if Configuration Verification is also to be performed on XC4000 or Spartan series FPGAs, the following leads must also be used as shown in Figure 56.

- RT
- RD

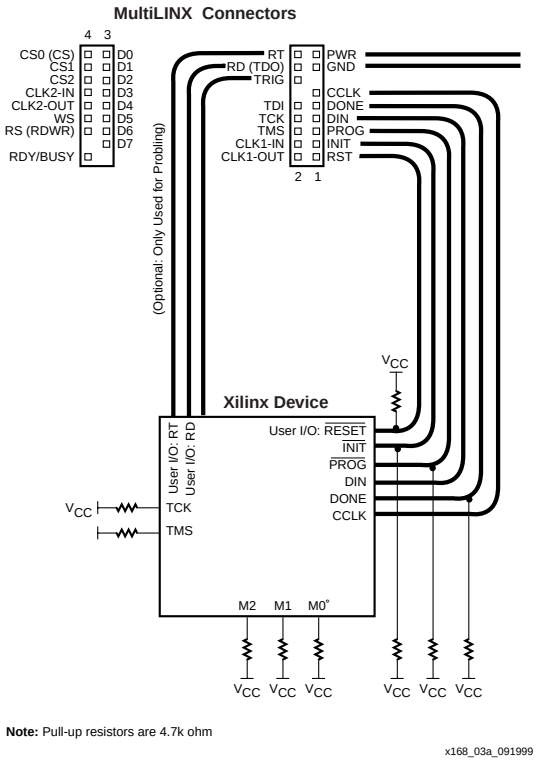

Figure 56: **Slave Serial Configuration and Verify**

Notes:

1. Serial verification is not used with Virtex FPGAs.

SelectMAP Connection

The SelectMAP interface to Virtex FPGAs, shown in Figure 57, requires the use of the following leads.

- PROG
- CCLK
- DONE
- INIT
- D0 - D7
- CS0 (CS)
- RS (RDWR)
- RDY / BUS

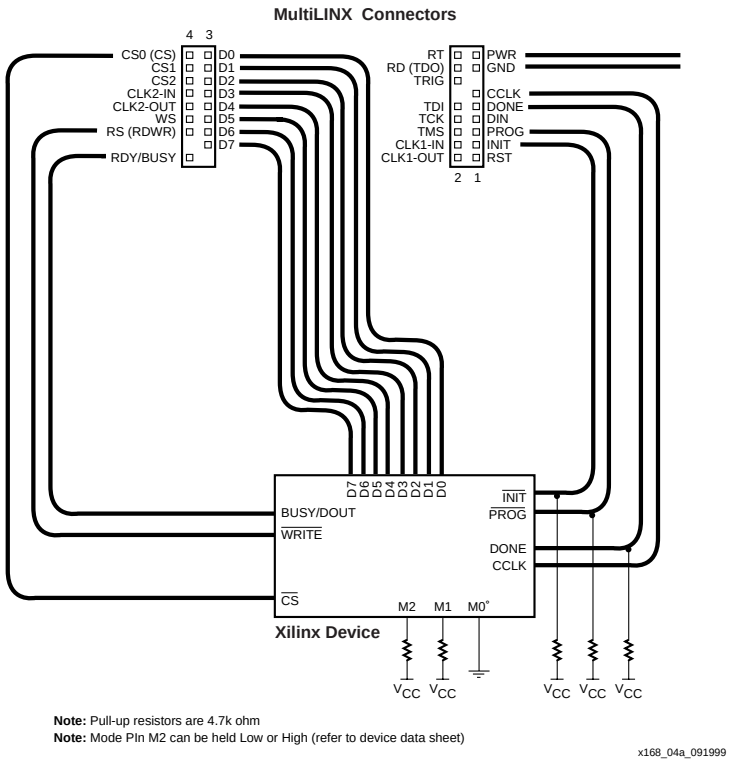

Figure 57: **SelectMAP Configuration and Verify**

Boundary Scan Connection

Boundary Scan connections, shown in Figure 58, need use only the standard four JTAG leads.

- TMS
- TCK
- TDI
- TDO

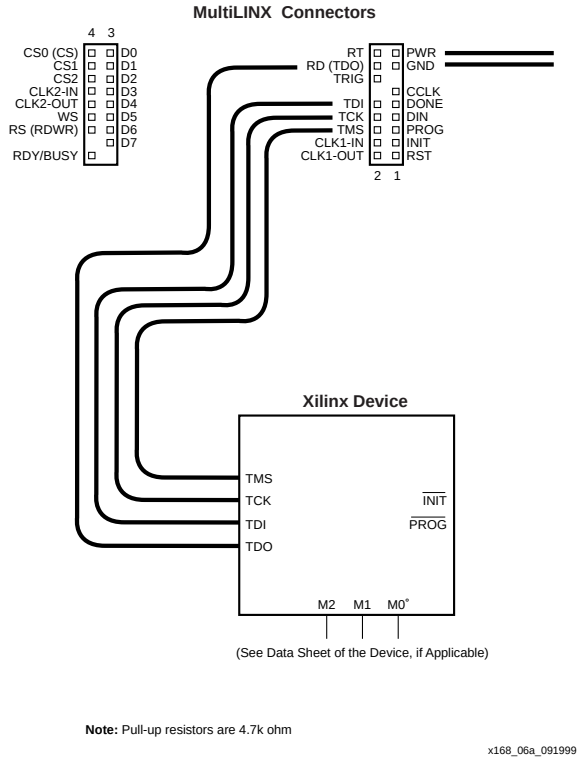

Figure 58: **Boundary Scan Connection**

Boundary Scan Connection for XC4000 and Spartan Devices

Boundary Scan connections, shown in Figure 59, need use only the standard four JTAG leads.

- TMS
- TCK
- TDI
- TDO

However, reconfiguration requires access to the $\overline{\text{PROGRAM}}$ and $\overline{\text{INIT}}$ pins of the XC4000 and Spartan devices.

- PROGRAM
- $\overline{\text{INIT}}$

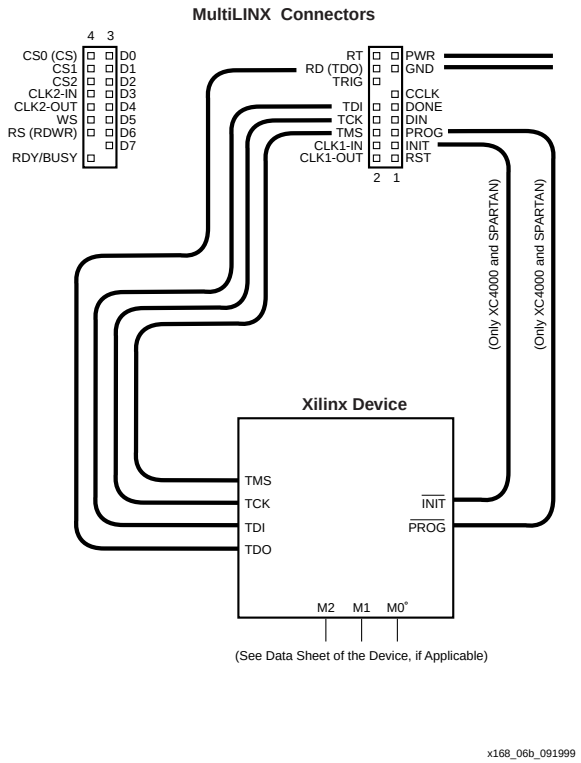

Figure 59: **Boundary Scan Connection for XC4000 and Spartan Devices**

The MultiLINX Cable System

Features

Supports serial download, verification and debug for any size FPGA.

- 4000X, Spartan, and Virtex Series
- Supports JTAG configuration for 9500, 4000, Virtex, Spartan and Spartan series
- 12 Mb/s full-speed USB device
- Direct Connection: No 2.5 V or 3.3 V adapters required
- Supports 3.3 V and 2.5 V LVTTL I/O Standards
- USB certified and compliant
- Accepts any supply voltage from 2.5 V to 5 V as power input (from target application or external power supply)

- Comes complete with USB cable, RS-232 cable, DB25M to DB9M adapter, and DB25F to DB9M adapter
- Configures to target hardware with four versatile flying wires

The new MultiLINX cable set takes advantage of the USB port found on newer PCs and provides download performance rates of up to 12 Mb/s throughput. The MultiLINX cable set features an adjustable voltage interface enabling communication with 5 V, 3.3 V, or 2.5 V systems and I/O devices.

Software support for the MultiLINX cable set is provided by the Hardware Debugger software in version 2.1i of the Alliance Series and Foundation Series software products for all modes except JTAG. JTAG mode support is provided by the JTAG Programmer software in WebPack 2.1WP5.x.

Description

The MultiLINX cable set is "CE" compliant as specified by the EMC Directives EN 55022 and EN 50082-1.

The MulitiLINX cable set consists of an electronics pod with a status LED and built-in RS232 and USB connectors, and four "flying wire" sets for connecting the pod to the target device. In addition, there are a USB cable and an RS232 cable for connecting the MultiLINX system to host computers as well as two DB9-to-DB25 adapters (one male, one female) for connecting to workstations.

The MultiLINX system/host computer connection is established via an RS232-compatible serial port with a DB9 cable or via a USB-compatible serial port with a shielded USB cable.

Table 106: **RS232 Maximum Baud Rate**

Computer Platform	Maximum Baud Rate
PC Compatible	57,600 baud
SPARC	38,400 baud
HP	38,400 baud

Table 107: **USB Platform Support**

Computer Platform/OS	USB Support (12 Mbits/s)
PC Compatible/Windows 98	Yes
SPARC	N/A
HP	N/A

Note: The MultiLINX Cable does not get power from the USB port.

Specifications

Table 108: **MultiLINX System Contents**

Item	Quantity	Comments
Electronics Pod	1	3.33" x 5.06" x 1.03", 5.8 oz. (160 gm)
Flying Wire Sets	4	For connecting to target devices
RS232 Cable	1	For connecting to host computer
USB Cable	1	For connecting to host computer
DB9 to DB25 Adapters	2	One Male, One Female
Total Shipping Weight	1 Box	1.5 lbs (690 gm)

Table 109: **Recommended Conditions**

Parameter	Operating	Non-Operating
DC Voltage	2.5V to 5.0V	N / A
Maximum DC Signal Voltage per Flying Wire	–0.5V to 5.5V	N / A
Typical Current Requirements	300 mA @ 5.0V, 500 mA @ 3.3V, 750 mA @ 2.5V	N / A
Power Req.	2 W Max.	N / A
Temp. Range	0° C to 55° C	–40° C to 85° C
Altitude	–100 to 15,000’	–1000 to 50,000’

Table 110: **Reliability Characteristics**

Parameter	Operating	Non-Operating
Relative Humidity	5% to 90% (non-condensing at 40° C)	5% to 95% (non-condensing at 65° C)
Vibration	0.5 mm (pp), 5 - 55 Hz	1.5 mm (pp), 5 - 55 Hz
Shock	5.0 G, 11 ms, 1/2 sine	30 G, 11 ms, 1/2 sine
Bench Drop	3’ drop onto a solid surface without physical damage	3’ drop onto a solid surface without physical damage

Notes:

1. Each MultiLINX Flying wire can drive a total capacitive load of 100 pF.
2. Each MultiLINX Flying wire can drive a total pull-up resistance of 500 Ω.
3. Each MultiLINX Flying wire can drive a total pull-down resistance of 5000 Ω.

Chapter 8

Cable Hardware

Introduction

This chapter gives specific information about connecting and using the In System Programming (ISP) Download Cables. These cables can be used to configure FPGAs and CPLDs. The following sections are in this chapter.

- "Cable Overview"
- "Cable Baud Rates"
- "MultiLINX System"
- "Parallel Cable III"
- "XChecker Cable"
- "Power-Up Sequence"
- "Download Cable Schematic"

Cable Overview

Three cables are available for use with Xilinx Alliance and Foundation software. The MultiLINX Cable supports USB and RS-232 serial port connections, the Parallel Cable III supports parallel ports, and the XChecker Cable supports RS-232 serial ports.

Selecting A Cable

Determine the most suitable cable to use, depending upon the tasks you want to perform.

MultiLINX Cable

You can use the MultiLINX Cable to download & readback your Xilinx programmable logic device. The MultiLINX cable hardware communicates with the host through the Universal Serial Bus (USB) port or an RS-232 interface. The additional flying wires support the various configuration modes available on Xilinx configuration cables.

Parallel Cable

The Parallel Cable III connects to the parallel printer port of a PC. This cable can be used to download and readback configuration data for all serial modes including JTAG.

XChecker Cable

The XChecker Cable connects to the serial port of both workstations and PCs. This cable can be used for design verification and debugging in addition to data download and readback.

Table 111: **Cable Support**

Name	Function	Platform
MultiLINX Cable (Model: DLC6)	Download, Readback	PC, Workstation
Parallel Cable III (Model: DLC5)	Download, Readback via JTAG	PC
XChecker Cable (Model: DLC4)	Download, Readback, and Debug	PC, Workstation

Supported Devices

Since cables can be used to configure FPGAs and CPLDs, the following Xilinx product lines are supported.

- Spartan, SpartanXL, SpartanII
- Virtex and Virtex-E
- XC9500, XC9500XL, XC9500XV
- XC4000E, XC4000L, XC4000EX, XC4000XL, XC4000XV, XC4000LXLA
- XC3000A, XC3000L
- XC3100A, XC3100L
- XC5200

Notes:

Spartan devices are labeled XCS*n*, XCS*n*XL, and XC2S*n*. Virtex families are labeled XCV*n and XCVnE*. In these labeling conventions, "*n*" represents the device number.

Software Support

Make sure that you use the appropriate configuration software for your device type.

- JTAG Programmer Software is used to configure FPGAs and CPLDs, and supports both the XChecker and the Parallel Cable III. This is a GUI based program.
- Hardware Debugger Software supports the MultiLINX, Parallel, and XChecker download cables and is used for FPGA configuration. This is a GUI based program with a waveform-viewer.

Notes:

All Hardware Debugger Software versions prior to 2.1i do not support the MultiLINX Cable. The Hardware Debugger Software only supports the MultiLINX Cable in the 2.1i release. For a complete list of part / cable / software combinations, refer to Solution 8097.

For specific information on using the download cables with the Hardware Debugger Software, see the *Hardware Debugger Guide*. Consult the *JTAG Programmer Guide* for more information about using this software. The JTAG programmer supports MultiLinx. The command line / batch mode JTAG programmer application is JTAGPROG, included with Xilinx software.

Cable Limitations

The MultiLINX Cable is compatible in supporting Readback for all the FPGAs supported by the XChecker cable. In addition to the supporting legacy devices, the MultiLINX Cable supports the devices that were not supported by the XChecker cable. Supported devices include those devices in the 4000E, 4000XL, and SPARTAN families whose bit file size is more than 256K bits. The MultiLINX Cable will also support readback for the new Virtex family.

Notes:

Debug is not available with the MultiLINX Cable when using the Hardware Debugger Software in the 2.1i Xilinx release version.

XChecker Hardware Drawbacks

- Cannot support readback for devices whose bit file size is more than 256K bits.
- Only supports RS-232.
- Has less user control (only 2 sets of 8 flying wires each).

MultiLINX Hardware Advantages

- Fast download, readback & verify using the USB port.
- More configuration modes are supported.
- Supports both RS-232 ports and USB ports.
- Compatible with the currently supported devices for Readback & Verify.
- Supports new devices that are not supported by XChecker due to RAM size limitation.
- Works at multiple supply voltages (5 V, 3.3 V, and 2.5 V).
- Supports JTAG configuration for all Xilinx devices.
- Supports SelectMAP configuration mode for Virtex.

Previous Cable Versions

This section details considerations for using previous download cables with the Hardware Debugger Software.

You can use Hardware Debugger software with all previous parallel and serial download cables. However, these previous cables can only be used to download a configuration bitstream, they cannot be used for readback.

If you do use Hardware Debugger with a previous parallel or serial download cable version, keep the following points in mind.

- Previous versions of the download cable were made to download XC3000 and XC2000 designs, not XC4000 designs. The basic limitation of the previous cables is that they do not have a PROG pin to initiate a re-program in XC4000 devices. They also do not have an INIT pin to check for Cyclical Redundancy Check (CRC) errors during configuration.

Notes:

To use a parallel download cable prior to the Parallel Cable III to download designs to the XC4000 family devices, you must manually toggle the PROG pin to low. PROG is active when it is low. (The Parallel Cable III has a wire for the PROG pin.)

- Previous download cables do not support readback or verification.
- For the PC, the download cable is a parallel cable, requiring connection to the parallel port. (The XChecker cable is serial.)

There are only two situations when you might prefer using previous download cables instead of the XChecker Cable or MultiLINX Cable.

- You have circuit boards with header connectors keyed to match the previous cable

headers. However, you could use the XChecker Cable with its flying lead connectors. Simply match the labeled flying leads to the equivalent signals on your system.

- You have circuit boards where power consumption is a critical factor. (The XChecker Cable requires about 100 mA at 5 V and the MultiLINX Cable requires about 300mA at 5 V, 500mA at 3.3 V, and 750mA at 2.5 V; the Parallel Cable used with PCs draws less power from the target LCA board.) In such cases, you can use the Hardware Debugger software to download the bitstream.

Cable Baud Rates

Baud Rates for the supported MultiLINX Cable Set and the Parallel and XChecker Cables are shown in Table 112.

Table 112: **Cable Baud Rates**

Cable	PC	WorkStation
USB MultiLINX	12Mb/s (Win98)	Not supported
RS-232 MultiLINX	9600, 19200, 38400, and 57600	9600, 19200, and 38400
Parallel	9600	Not supported
XChecker	9600, 19200, 38400, and 115200	9600, 19200, and 38400

MultiLINX System

The MultiLINX System is a cable set for configuring and verifying Xilinx FPGAs and CPLDs.

Flying Leads

The MultiLINX System is shipped with four flying lead sets, a USB Cable, an RS-232 Cable, and two adapters. For detailed information on the MultiLINX flying lead sets and supported operation modes, refer to the “MultiLINX System” chapter of this guide.

Figure 60 shows the MultiLINX cable set electronics pod and the four flying lead sets for connecting to the target device.

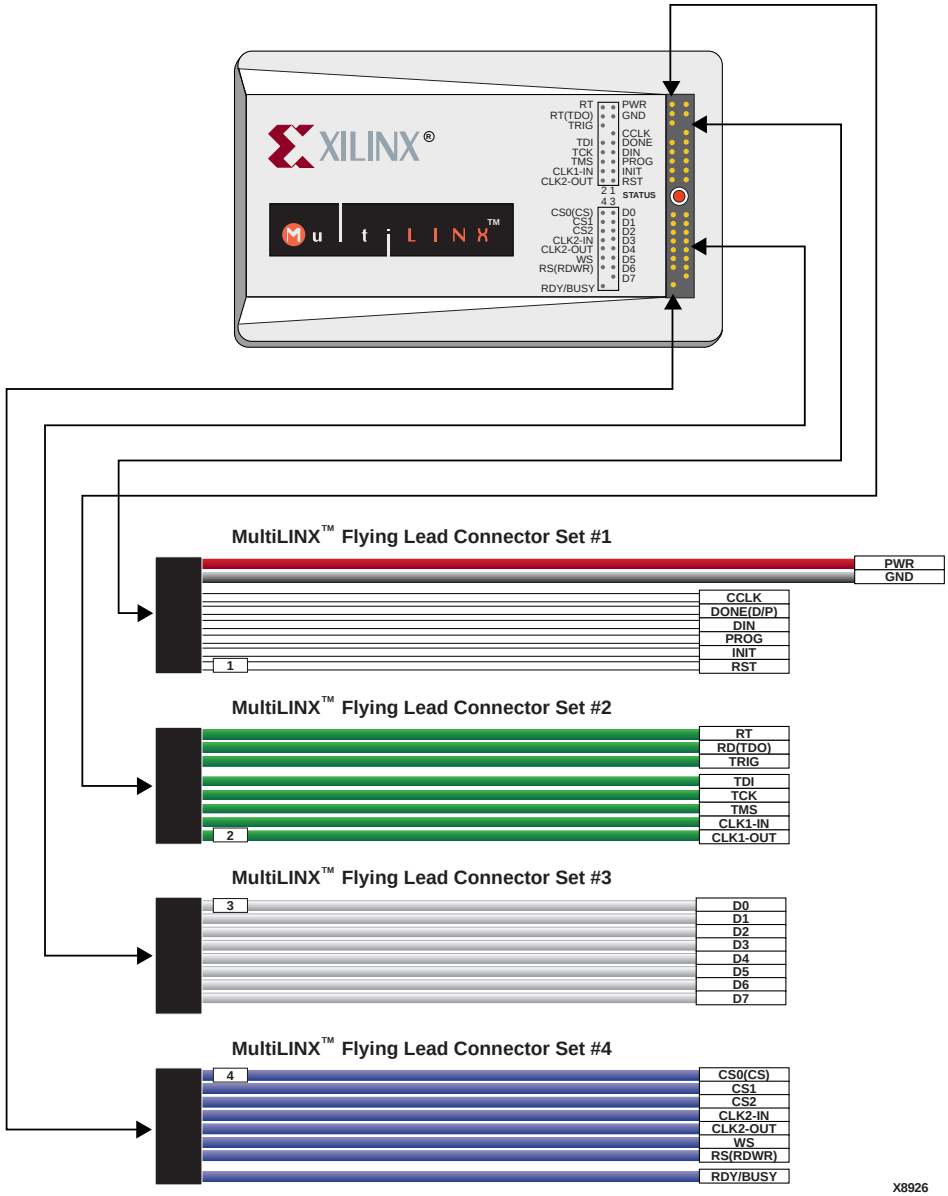

Figure 60: **MultiLINX Electronics Pod and Flying Lead Connectors**

Figure 61 shows the top and bottom view of the MultiLINX System electronics pod.

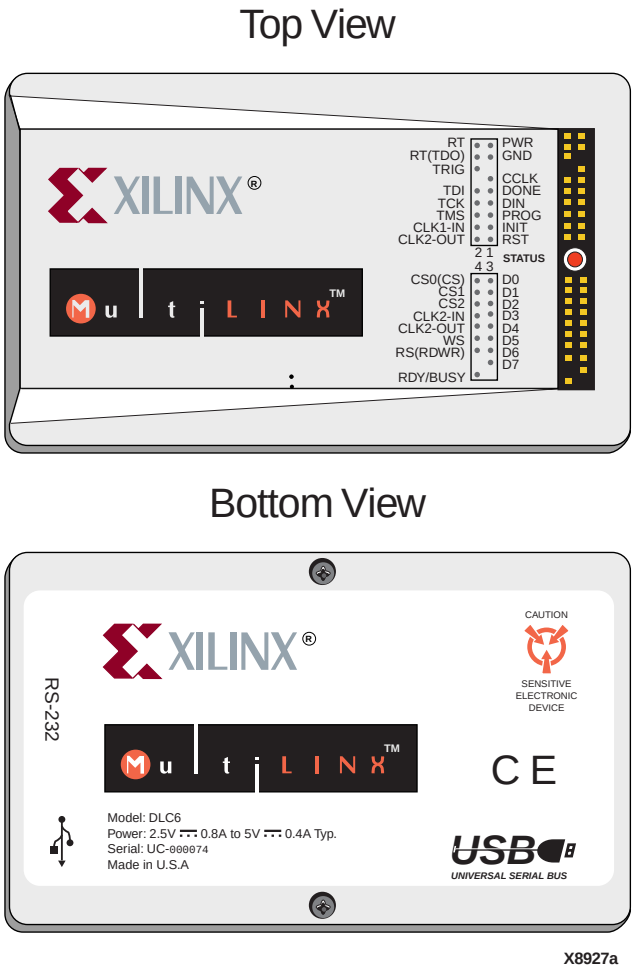

Figure 61: **MultiLINX Electronics Pod**

Power for the MultiLINX System

MultiLINX System power is supplied by the target device circuit board or an optional external power supply, not from the host computer. The red (PWR) and black (GND) wires from Flying Lead Set #1 are connected to the V_{CC} and Ground of the circuit board powering the Xilinx device. Connection of optional external power is shown in the Figure 62.

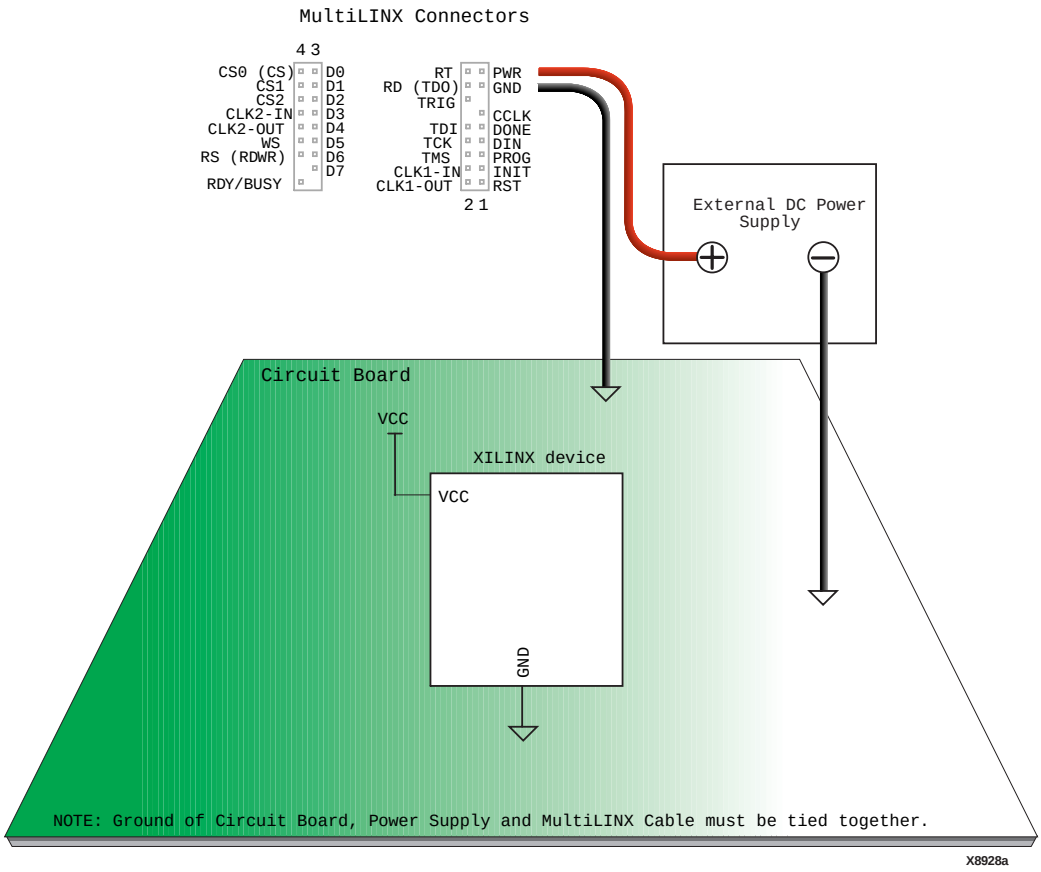

Figure 62: **Optional External Power for the MultiLINX System**

When using an external power supply, assure that the supply, MultiLINX System, and circuit board grounds are all tied together. An advantage of using an external DC power supply is that power to the circuit board is not compromised, and power to the MultiLINX System is not lost even when the circuit board power is off.

Parallel Cable III

The Parallel Cable III is a cable assembly that contains a buffer to protect the PC parallel port and a set of headers to connect to the target system.

The cable can be used with a single CPLD or FPGA device, or with several devices connected in a daisy chain.

Cable transmission speed is determined solely by the speed at which the host PC can transmit data through its parallel port interface.

Using the Parallel Cable III requires a PC equipped with an AT- compatible parallel port interface and a DB25 standard printer connector.

Flying Leads

This cable is shipped with two sets of flying leads, one for FPGAs and one for CPLDs. The CPLD leads are labelled “JTAG”, and the FPGA leads are labelled “FPGA”.

Each flying lead set has a 9-pin (6 signals, 3 keys) header connector on one end. This connector fits onto one of the two cable headers. These header connectors are keyed to assure proper orientation to the cable assembly.

On the other end of each flying lead set are six individual wires with female connectors. The female connectors fit onto standard 0.025 inch square male pins. Figure 63 shows the Parallel Cable III and its FPGA flying lead set.

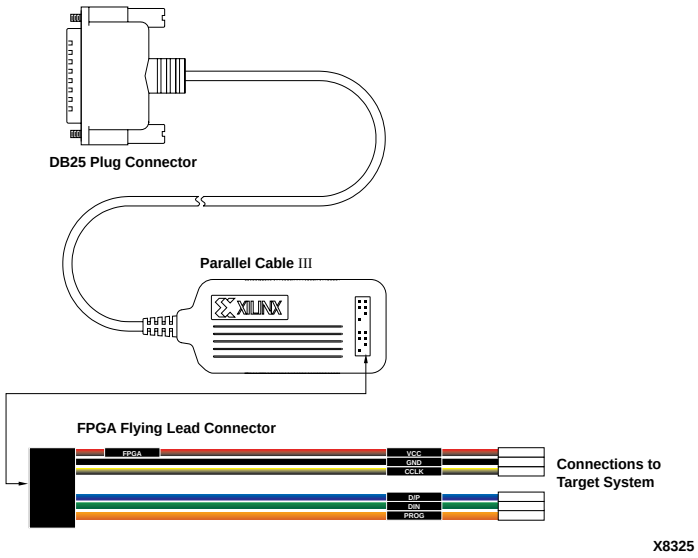

Figure 63: **Parallel Cable III and FPGA Flying Lead Set**

Figure 64 shows top and bottom views of the Parallel Cable III.

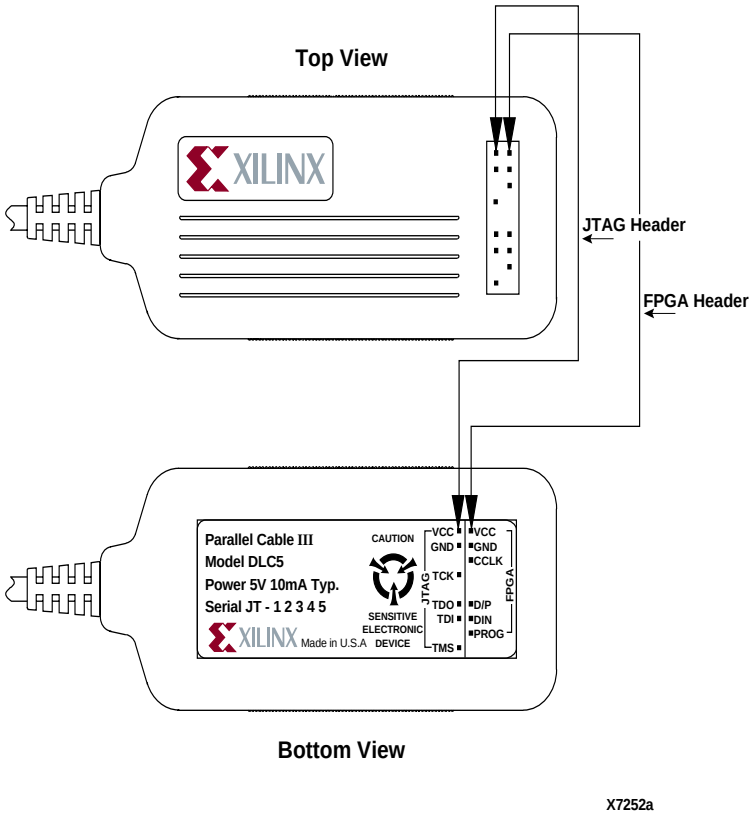

Figure 64: **Parallel Cable III**

Notes:

The Parallel Cable III is grey; the XChecker Cable is beige.

Configuring CPLDs With the Parallel Cable III

When connecting the CPLD flying leads for configuration, make sure to use the “JTAG” header. Figure 65 shows the connections between the Parallel Cable III CPLD flying leads and a target system.

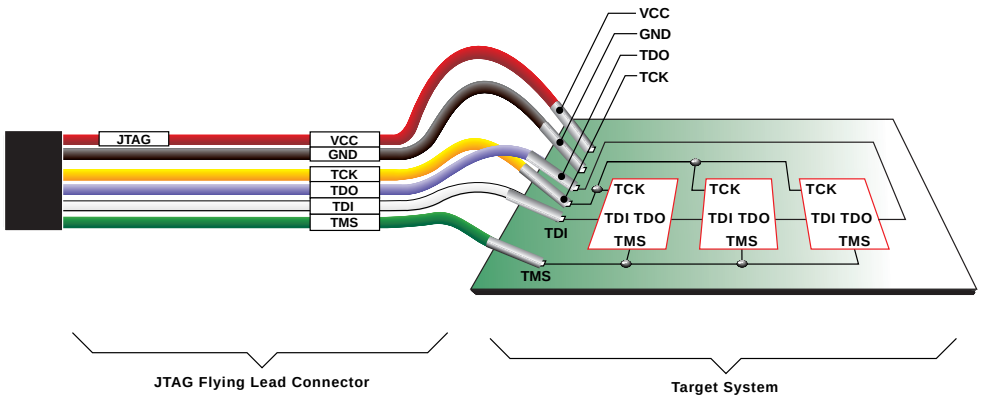

Figure 65: **Parallel Cable III Connections to CPLD Device**

Table 113 describes the pin functions and connections for configuring CPLDs with the Parallel Cable III.

Table 113: **Parallel Cable III CPLD Pin Connections**

Name	Function	Connections
V_{CC}	Power – Supplies V_{CC} (5 V, 10 mA, typically) to the cable.	To target system V_{CC}
GND	Ground – Supplies ground reference to the cable.	To target system ground
TCK	Test Clock – Drives the test logic for all devices on a JTAG chain.	Connect to system TCK pin.
TDO	Test Data Output – Data from the target system	Connect to system TDO pin.
TDI	Test Data Input – Transmits serial test instructions and data.	Connect to system TDI pin.
TMS	Test Mode Select – Decoded by the JTAG state machine to control test operations.	Connect to system TMS pin.

Notes:

TRST is an optional pin in the JTAG (IEEE 1149.1) specification, and is not used by XC9500 CPLDs. If using non-Xilinx parts with a TRST pin, connect it to V_{CC}.

Configuring FPGAs With the Parallel Cable III

This section details the connections needed to configure FPGAs with the Parallel Cable III.

Figure 66 and Figure 67 show which pins to connect for devices in the XC4000 and XC3000 FPGA series.

Notes:

If you are using the Xilinx FPGA Design Demonstration Board, see the "Mode Switch Settings" section of the "FPGA Design Demonstration Board" chapter of the "Hardware User Guide" "for specific configuration information.

Figure 66 shows the flying lead connections to XC4000 FPGAs.

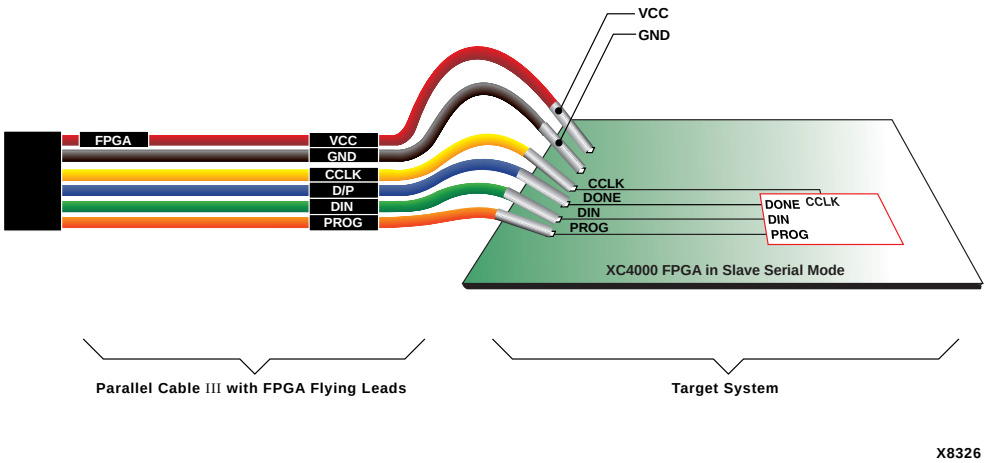

Figure 66: **Parallel Cable III Connections to XC4000 Devices**

To configure XC3000 FPGAs, the PROG wire is not used, as shown in the Figure 67.

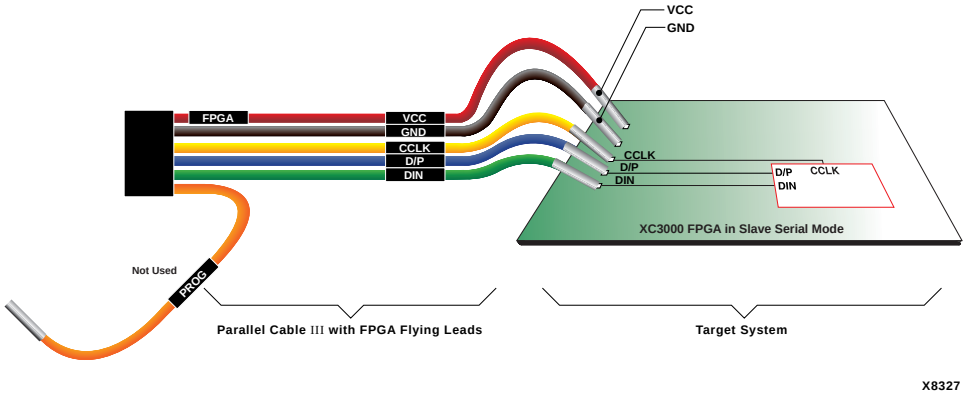

Figure 67: **Parallel Cable III Connections to XC3000 Devices**

Notes:

If you are using a Xilinx FPGA Demonstration Board, see the “Mode Switch Settings” section of the “FPGA Design Demonstration Board” chapter of the “Hardware User Guide” for configuration information.

XChecker Cable

The XChecker hardware consists of a cable assembly with internal logic, a test fixture, and two flying lead sets to connect the cable to the target system. XChecker hardware can be used with a single FPGA or CPLD or several devices connected in a daisy chain.

Connection to the host computer is via a standard DB-9 or DB-25 RS-232 serial port. Different serial port connections require an appropriate adapter.

Flying Leads

The XChecker Cable is shipped with two lying lead sets. The flying lead connectors have a 9-position header connector on one end. The other end has eight individual wires with female connectors that fit onto standard 0.025 inch square male pins.

Appropriate pins on the target system for connecting to the download cable are required. See the “” section and the “” section for connection details.

Figure 68 shows the XChecker Cable hardware and flying lead sets.

XChecker Cable

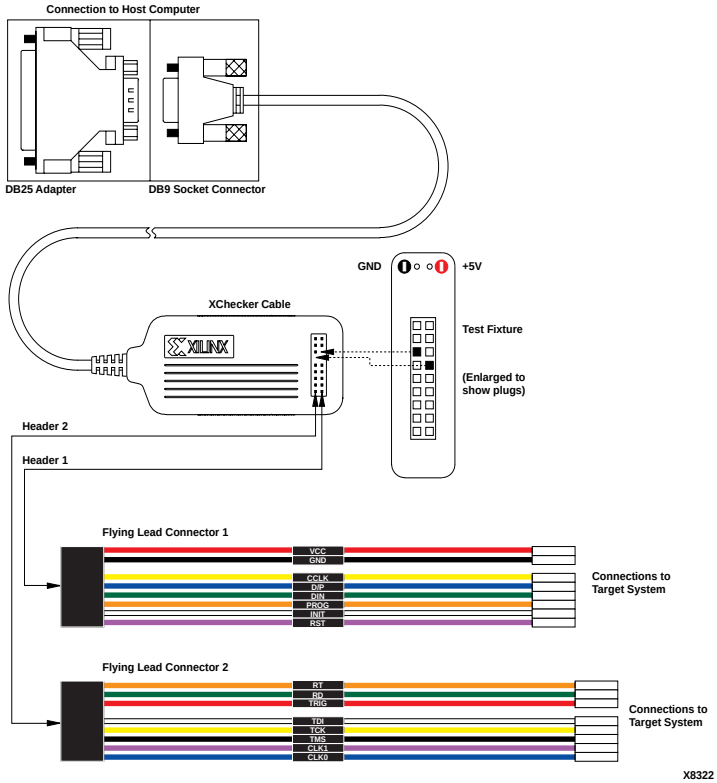

Figure 68: **XChecker Cable and Flying Lead Sets**

Figure 68 shows top and bottom views of the XChecker Cable electronics module.

Notes:

1. The XChecker Cable is beige; the Parallel Cable III is grey.
2. The flying lead sets are keyed to fit appropriate cable headers.

XChecker Baud Rates

Communication between the host system and the XChecker Cable depends on host system capability. The XChecker Cable supports several Baud rates and platforms, as shown in Table 114.

Table 114: **Valid Baud Rates**

Platform	9600	19200	38400	115.2K
IBM® PC	X	X	X	X
NEC PC	X			
SUN®	X	X	X	
HP 700	X	X	X	X

Notes:

X indicates applicable baud rate

Configuring CPLDs With the XChecker Cable

JTAG Programmer software should be used to program in JTAG mode. When configuring a CPLD with the XChecker Cable, only six of the sixteen flying leads are used. For JTAG boundary-scan systems only the V_{CC}, GND, TDI, TCK, TMS and RD (TDO) pins are connected.

TRST is an optional pin in the JTAG (IEEE 1149.1) specification, not used by XC9500 CPLDs (If non-Xilinx parts have a TRST pin, connected it to V_{CC}).

Once installed properly, the connectors provide power to the cable as well as allowing configuration data download and readback.

Table 115 summarizes CPLD pin connections to the target circuit board.

Table 115: **XChecker Cable Pin Connections for CPLDs**

Name	Function	Connections
V_{CC}	Power - Supplies V_{CC} (5 V, 100 mA, typically) to the cable	To target system V_{CC}
GND	Ground - Supplies ground reference to the cable	To target system ground
RD (TDO)	Read Data - Data from the target system is read at this pin.	Connect to system TDO pin.
TDI	Test Data In - Test instructions and data.	Connect to system TDI pin.
TCK	Test Clock - Drives test logic for all devices on boundary-scan chain.	Connect to system TCK pin.
TMS	Test Mode Select - Decoded by TAP controller to control test operations.	Connect to system TMS pin.
CLKI	Not used.	Unconnected.
CLKO	Not used.	Unconnected.
CCLK	Not used.	Unconnected.
D/P	Not used.	Unconnected.
DIN	Not used.	Unconnected.
PROG	Not used.	Unconnected.
INIT	Not used.	Unconnected.
RST	Not used.	Unconnected.
RT	Not used.	Unconnected.
TRIG	Not used.	Unconnected.

Configuring FPGAs With the XChecker Cable

This section details the connections needed to configure FPGAs with the XChecker Cable.

Notes:

If you are using the Xilinx FPGA Design Demonstration Board, see the “Mode Switch Settings” section of the “FPGA Design Demonstration Board” chapter of the “Hardware User Guide” for specific configuration information.

Figure 69 and Figure 70 show which pins to connect for devices in the XC4000 and XC3000 FPGA series.

Use Header 1 (Figure 68) to connect the XChecker Cable to the target system for configuring FPGAs. When configuring XC4000 FPGAs, the RST (Reset) wire is not used, Figure 69.

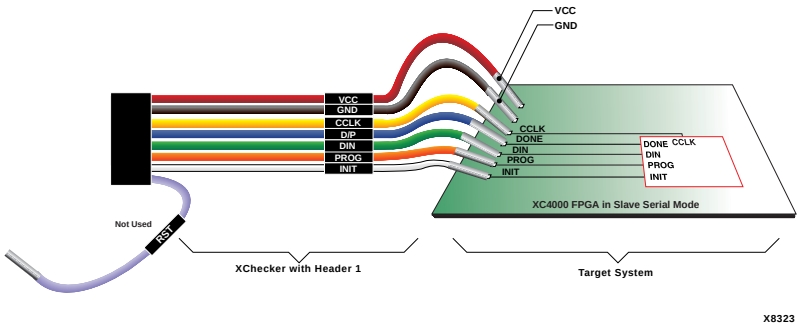

Figure 69: **XChecker Connections to XC4000 Device**

When configuring XC3000 FPGAs, the PROG wire is not used, Figure 70. In both cases, the FPGA must be in the Serial Slave Mode.

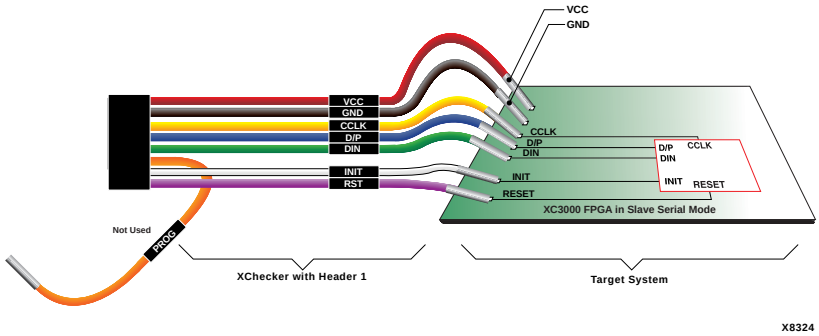

Figure 70: **XChecker Connections to XC3000 Device**

Power-Up Sequence

This section details the suggested methods for pin connections and cable operation.

Pin Connection Considerations

The following adjustments make the process of connecting and downloading easier:

- Provide appropriate pins on your printed circuit board for your device type.
- Place pins on board so that flying leads can reach them. The flying leads that are shipped with the cable are six inches long. While pins can be a couple inches apart, do not have any two pins more than six inches apart.
- Keep header pins on your board a minimum of 0.10 inches apart.

Cable Connection Procedure

The following steps are required for download cable operation.

1. Connect the cable to your host system. Make sure to use the appropriate port and adapter, if necessary.
2. Connect the cable to your target system or demonstration board. Always power up the host system before the target system. The power for the drivers is derived from the target system.

3. Connect the cable's GND wire to the corresponding signal on the target board. Next, connect V_{CC} to the +5 V on the target board.
4. Connect the appropriate pins for device configuration.
5. Power up the target system.
6. Cable protection ensures that the host system port cannot be damaged through normal cable operation. For increased safety, please check that the power to the host computer is on before the target system is powered up.
7. Start the appropriate Xilinx software package and configure your device. The JTAG Programmer Software and Hardware Debugger Software will automatically identify the download cables when correctly connected. If you need to set up the cable manually, see the "Setting Up the Cable" section.

Notes:

1. The download cables will not operate if the target system's power is turned off before or during software operations. Make certain that this power connection is on and stable. Your system's power should be on during ISP operations.
2. When powering down, turn off the target demonstration board first, and then the host machine.

Setting Up The Cable

If you are using the Hardware Debugger Software and a PC as a host system, manually select your cable as follows.

`Output → Cable Setup`

Select your cable type, then click **OK**. If you are using the XChecker cable, you can also select a BAUD rate. See Table 114.

If you are using the JTAG Programmer software, select the cable manually as follows.

`Output → Cable Auto Connect`

Select your cable type, then click **OK**.

Download Cable Schematic

The following figure is an internal schematic of the Parallel Cable III. You must use the recommended lengths for parallel cables. Xilinx cables are typically six feet (approximately

two meters) in length between the connector and active circuitry. Keep the wires between the headers and target system as short as possible.

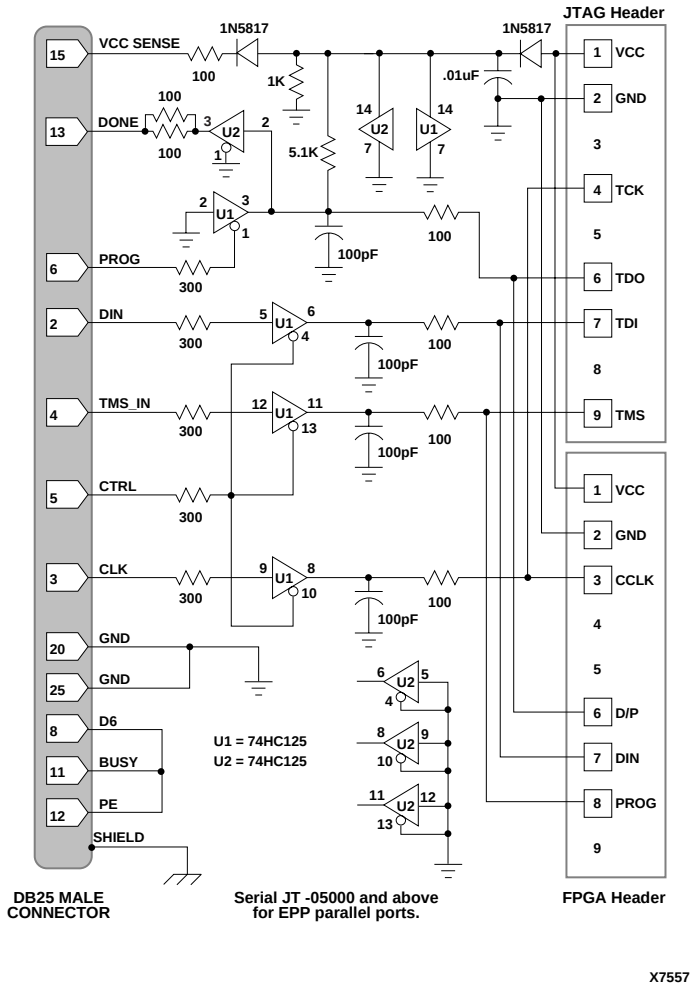

Figure 71: **Parallel Cable III Schematic**

Chapter 9

The MultiLINX System

The MultiLINX™ System is the next generation configuration and readback tool for FPGAs and CPLDs. During the integration of Xilinx programmable logic into your design, the MultiLINX System cable set can be used to troubleshoot your configuration setup and diagnose configuration problems.

The MultiLINX cable set uses either a serial or USB port on a host computer. Maximum throughput is available by using the USB interface.

This chapter contains the following sections:

- "Additional MultiLINX Documentation"
- "MultiLINX System Platform Support"
- "MultiLINX Flying Wires"
- "Device Configuration Modes"

Additional MultiLINX Documentation

For a quick reference when using the MultiLINX cable set, refer to "Getting Started With the MultiLINX System" on page 151 in this guide. Topics covered include:

- Using a USB port, mixed voltage environments, connections for all supported modes
- How to setup a prototype application for use with the MultiLINX cable set
- Cable set descriptions, capabilities, and associated software tools

MultiLINX System Platform Support

The MultiLINX cable set supports the following platforms:

- Win 95
- Win 95C
- Win 98
- Win NT 4.0
- Solaris 2.6
- HP 10.2

Table 116: **MultiLINX System Platform Support**

Supported Platforms	USB	RS-232
Win 95		X
Win 95C	X	X
Win 98	X	X
Win NT 4.0		X

Table 116: **MultiLINX System Platform Support**

Supported Platforms	USB	RS-232
Solaris 2.6		X
HP 10.2		X

Notes:

X indicates applicable ports that can be used with the MultiLINX cable set on specified platforms.

MultiLINX Flying Wires

The MultiLINX cable set is shipped with four sets of flying leads. Figure 72 shows these four sets of MultiLINX flying leads.

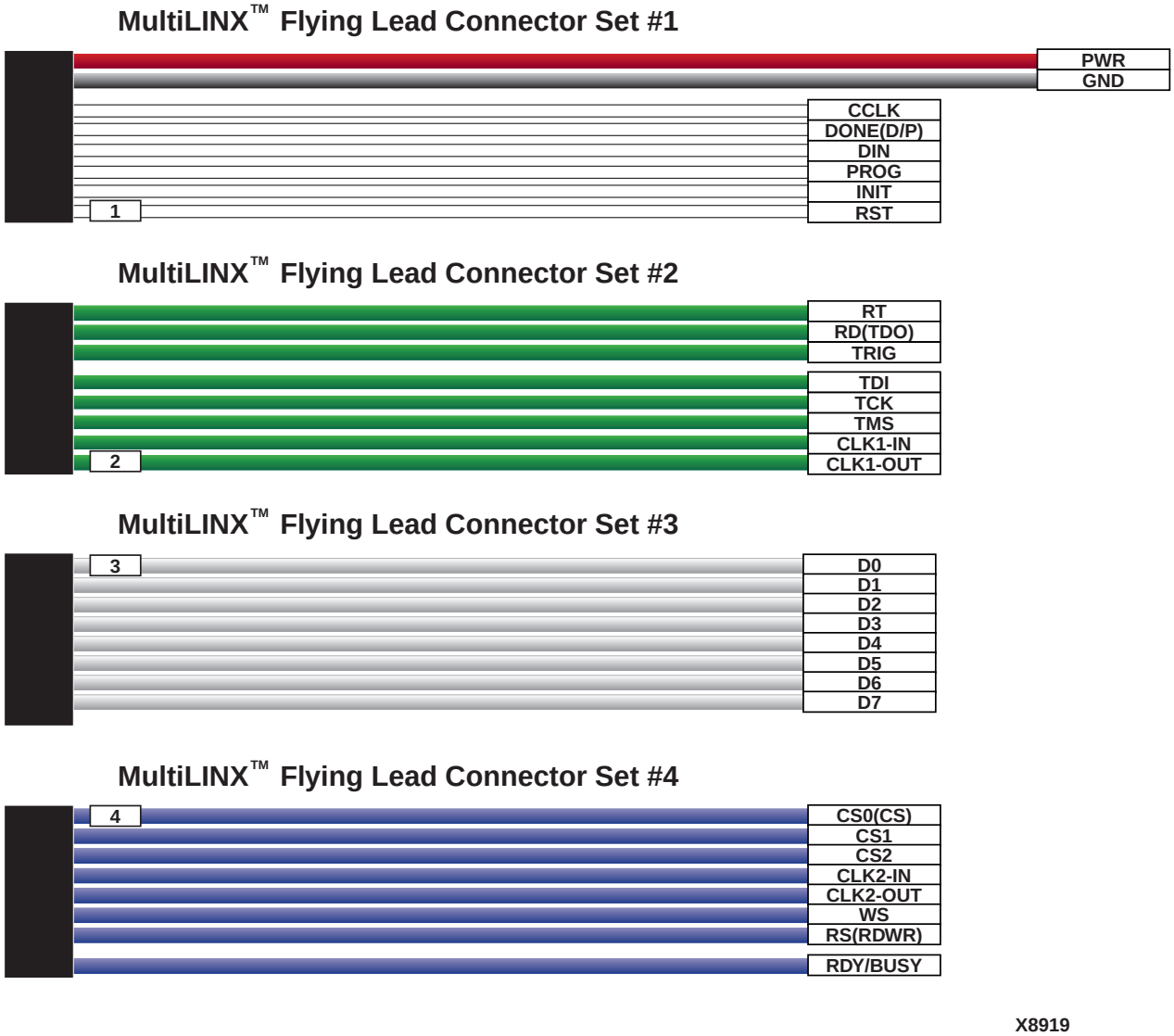

Figure 72: **MultiLINX Flying Lead Sets**

Table 117 describes wire functions of the MultiLINX Flying Lead sets.

Table 117: **MultiLINX Flying Lead Set Pins**

Signal Name	Function
PWR	*Power* — Supplies 2.5 V - 5 V to cable
GND	*Ground* — Supplies ground reference to cable
CCLK	*Configuration Clock* —The configuration clock pin, and the default clock for readback operation.
DONE (D / P)	*Done/Program* — represents the D / ~P pin for XC3000A / L and XC3100A devices, and DONE for XC4000, XC5200 and Spartan devices. This pin indicates that the configuration process is complete for XC4000, XC5200, and Spartan devices. This same pin initiates a reconfiguration, and indicates that the configuration process is complete on XC3000 FPGAs.
DIN	*Data In* — Provides configuration data to target system during configuration and is 3-state at all other times.
PROG	*Program* — A Low indicates the device is clearing configuration memory. An active Low signal initiates the configuration process.
INIT	*Initialize* — Initialization sequencing pin during configuration (Indicates start of configuration). A logic zero on this pin during configuration indicates a data error.
RST	*Reset* — Used to reset internal FPGA logic. Connection to this pin is optional during configuration. During configuration, a Low pulse causes XC3000A devices to restart configuration. After configuration, this pin can drive Low to reset target FPGA internal latches and flip-flops. RST is active High for XC4000 / XC5200 devices.
RT	*Read Trigger* — Pin used to initiate a readback of target FPGA. MultiLINX cable output. Hardware Debugger provides Low-to-High transition on RT to initiate readback.
RD (TDO)	*Read Data* — MultiLINX cable input. Hardware Debugger receives the readback data through the RD pin after readback is initiated. Pin used to initiate a readback of target FPGA TDO is for JTAG.
TRIG	*System Trigger* — MultiLINX cable input. A High on this pin signals MultiLINX electronics to initiate a readback and causes the RT pin to go High.
RD (TDO) TDI TCK TMS	These pins are used for JTAG Programmer device configuration. The JTAG / boundary scan pins function for FPGA and CPLD JTAG operations.
CLKI-IN	*Clock Input* — Transmits system clock to the MultiLINX electronics. Clock must be between 120 kHz and 10 MHz. Connect this pin to target system clock to synchronize the readback trigger with target system clock.
CLK1-OUT	*Clock Output* — Drives target system clock. Clock can come from either the CLKI-IN pin or be internally generated by the MultiLINX cable set when CLKI-IN is unconnected

Table 117: **MultiLINX Flying Lead Set Pins *(Continued)***

Signal Name	Function
D0-D7	*Data Bus* — This pin is used for Virtex SelectMAP Mode. An 8-bit data bus supporting SelectMAP and Express configuration modes.
CS0 (CS)	*Chip Select 0* — Default Chip Select for Virtex SelectMAP Mode.
CS1	*Chip Select 1*— Optional Chip Select for Virtex SelectMAP Mode.
CS2	*Chip Select 2*— Optional Chip Select for Virtex SelectMAP Mode.
CLK2-IN	*Clock Input* — Transmits system clock to the MultiLINX electronics. Clock must be between 120 kHz and 10 MHz. Connect this pin to target system clock to synchronize the readback trigger with target system clock.
CLK2-OUT	*Clock Output* — Drives target system clock. Clock can come from either the CLK2-IN pin or be internally generated by the MultiLINX cable set when CLK2-IN is unconnected.
WS	*Write Select* — Reserved
RS (RDWR)	*Read/Write* — The RDWR pin is used as an active High READ and an active Low WRITE control signal to Virtex FPGAs.
RDY / BUSY	*Busy Pin* — Busy pin on Virtex.

MultiLINX Communication Rates

Communication between the host system and the MultiLINX cable set depends on host system capability. The MultiLINX cable set supports several communication rates. With the USB interface, the MultiLINX cable set can run at 12 Mb / s. With the PC RS-232 interface, the MultiLINX cable set can run from 9600 Baud to 57.6 K Baud.

MultiLINX Power Requirements

Power to the MultiLINX cable set comes from the target FPGA circuit board, not from the host system. The red (PWR) and black (GND) wires of Flying Wire Set #1 are connected to the V_{CC} and Ground lines of the circuit board that is powering the target device. Minimum input voltage to the cable is 2.5 V (0.8 A). The maximum input voltage is 5 V (0.4 A).

External Power for the MultiLINX Cable Set

An optional method of powering the MultiLINX cable set is to use an external DC power supply (not supplied) as shown in Figure 73.

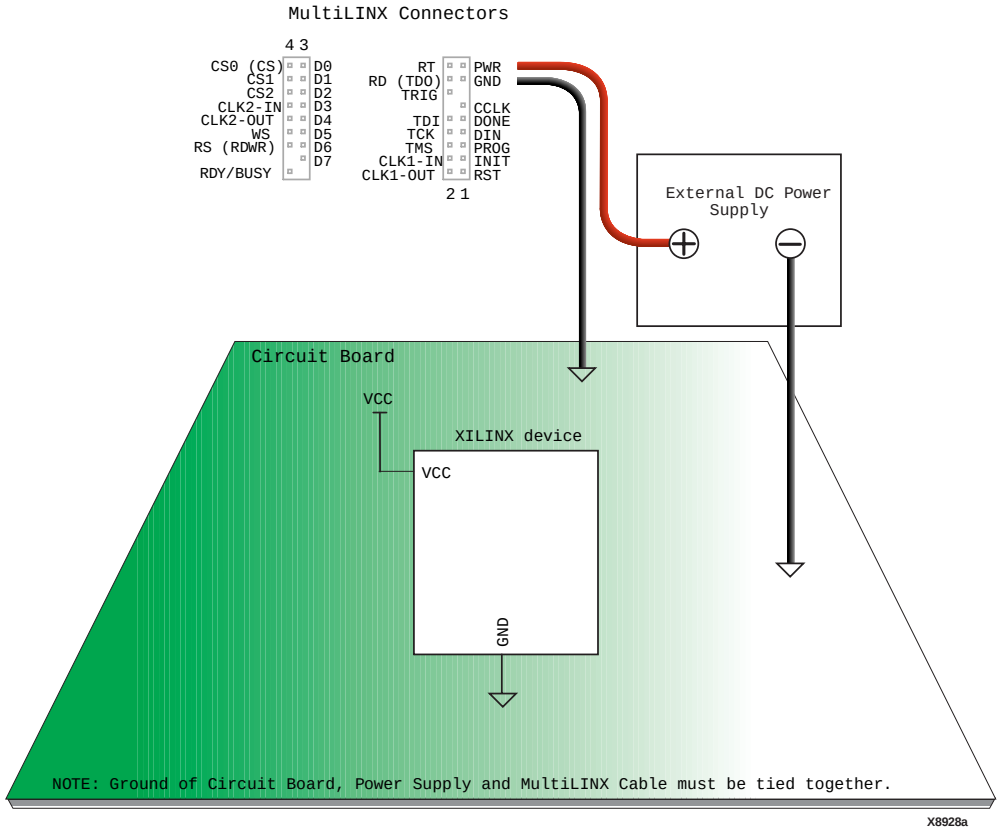

Figure 73: **Optional External Power for the MultiLINX Cable Set**

Notes:

1. The voltage supplied to the MultiLINX cable set does not need to match the voltage powering the target device. The cable generates its own voltages from the power supplied.

When using an external power supply, make sure that the grounds of the power supply, the MultiLINX cable set, and the circuit board are all tied together. An advantage of the external DC power supply is that power to the target circuit board is uncompromised, and power to the MultiLINX cable is not lost when power to the circuit board power is turned off.

Device Configuration Modes

Various MultiLINX device configuration modes are supported for each device family, as shown in Table 118.

Table 118: **MultiLINX Configuration Modes for Xilinx FPGAs**

Configuration Mode	Device Family					
	Virtex	Spartan	XC9500	XC5200	XC4000	XC3000
SelectMAP	X					
Slave Serial	X	X		X		X
JTAG	X	X	X	X	X	
Readback / Verify	X	X	X	X	X	X

Downloading Configuration Data - Slave Serial Mode

This section details the connections needed to download configuration data with the MultiLINX cable set in Slave Serial Mode.

XC3000 Devices

Figure 74 shows the Slave Serial Mode connections to a XC3000 device for Downloading Configuration Data.

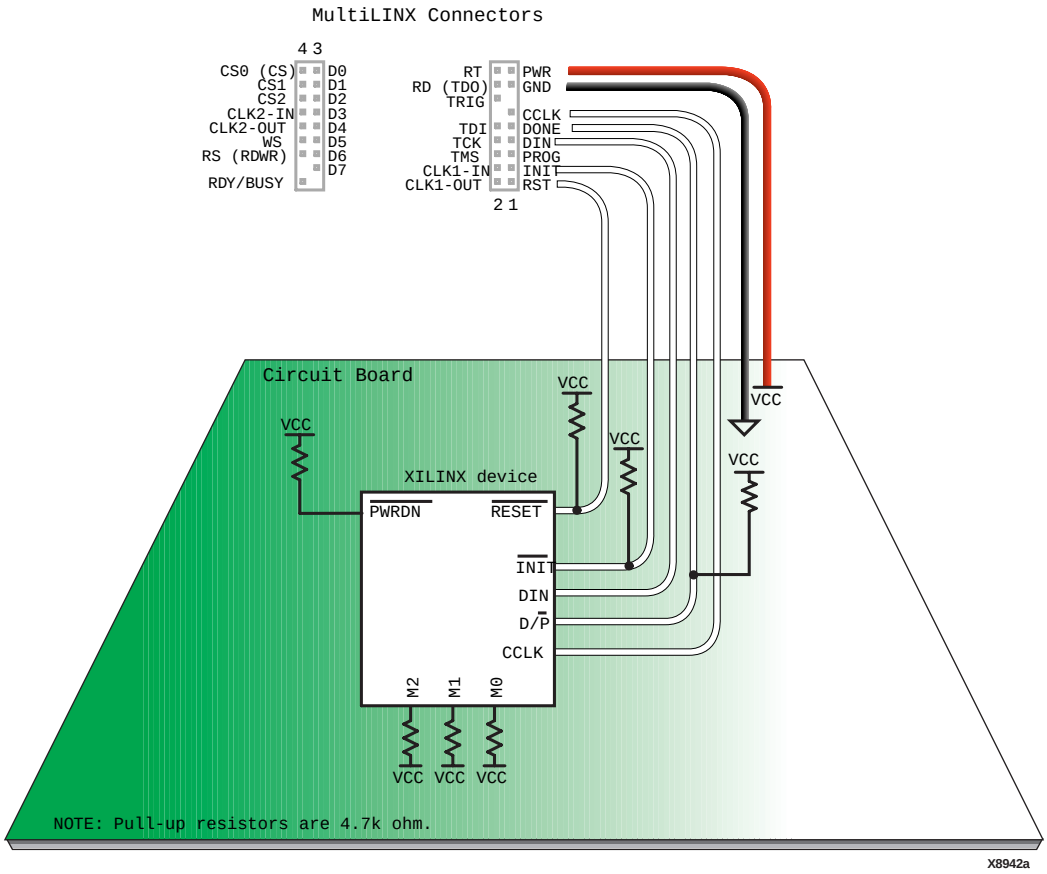

Figure 74: **XC3000 Slave Serial Mode**

Virtex, Spartan, XC5200, and XC4000 Devices

Figure 75 shows the Slave Serial Mode connections for Virtex, Spartan, XC5200, and XC4000 devices.

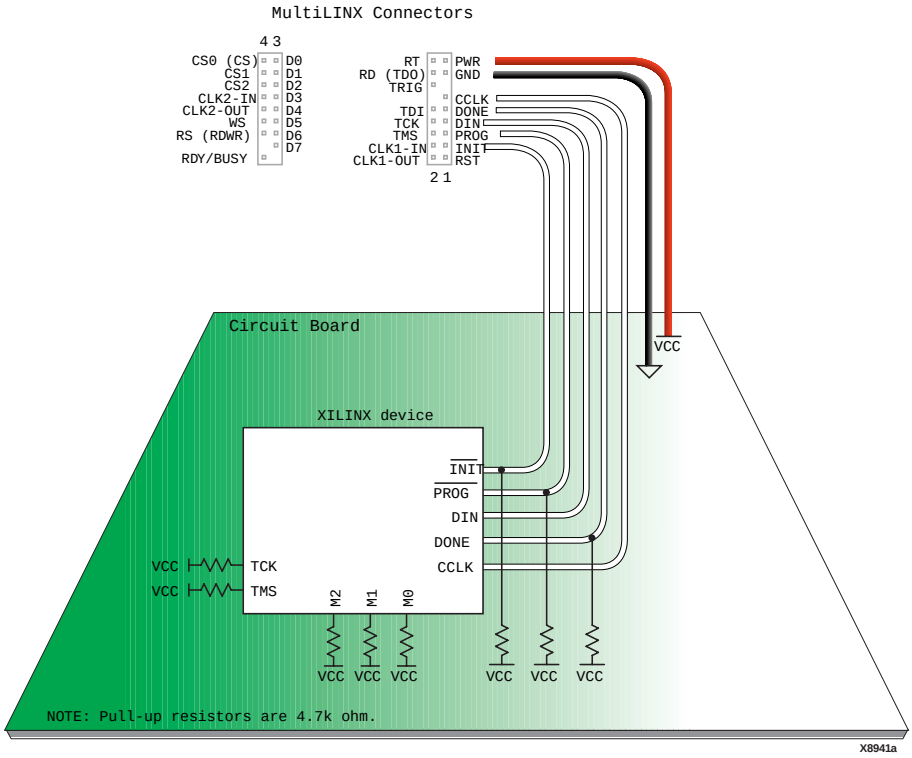

Figure 75: **Virtex, Spartan, XC5200, and XC4000 Slave Serial Mode**

Downloading Configuration Data - SelectMAP Mode

This section details the connections needed for downloading configuration data with the MultiLINX cable set in the SelectMAP Mode.

Virtex Devices

Figure 76 shows the SelectMAP Mode connections for Virtex devices.

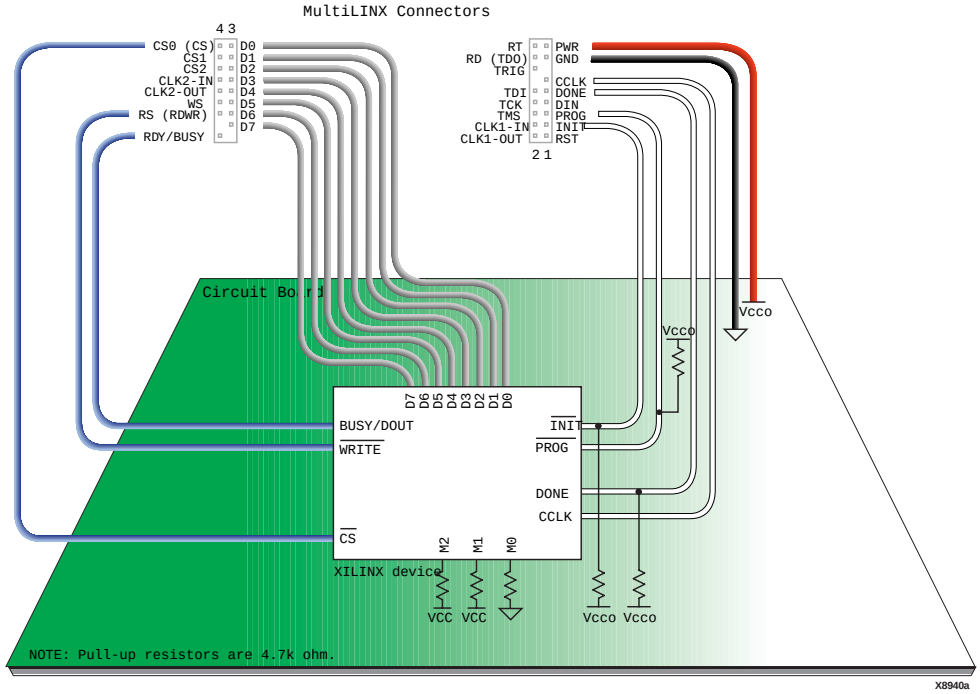

Figure 76: **Virtex SelectMAP Mode**

Downloading Configuration Data - JTAG Mode

This section details the connections needed for downloading configuration data with the MultiLINX cable set in JTAG Mode for the various Xilinx Device Families.

Figures 2-6 through 2 - 8 show the JTAG Mode connections for the different Xilinx device families.

XC9000 Devices

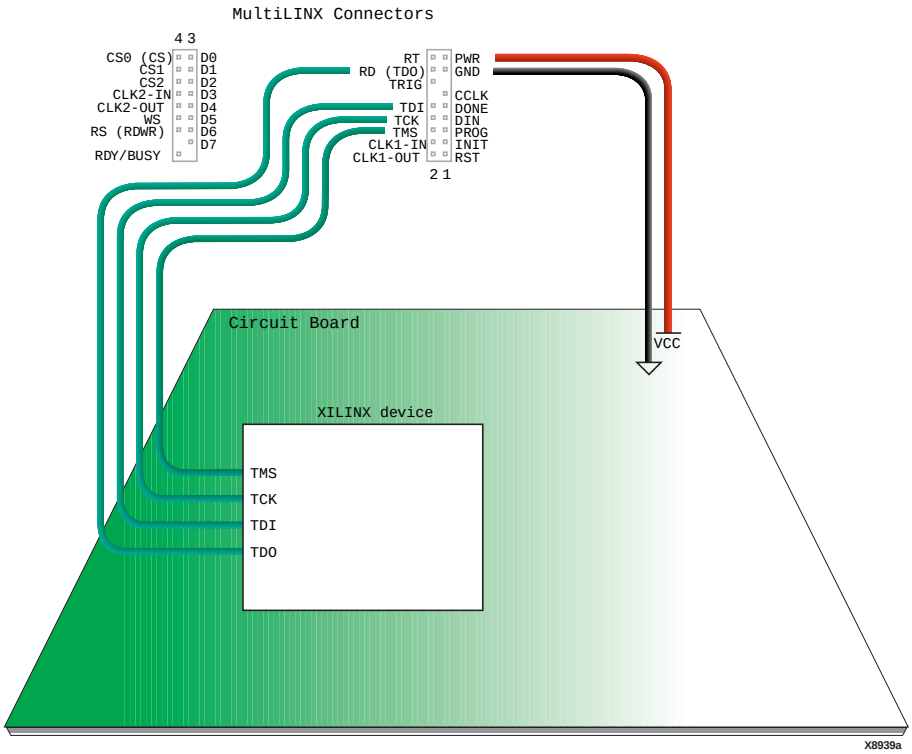

Figure 77: **JTAG Mode for XC9000**

Virtex Devices

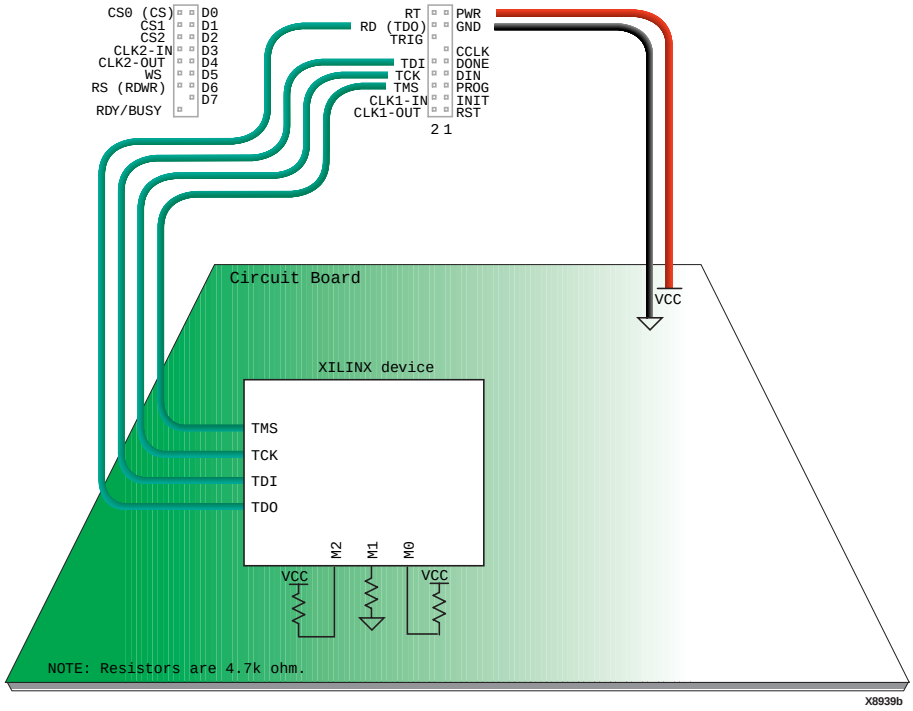

Figure 78: **Virtex JTAG Mode**

Spartan, XC5300, and XC4000 Devices

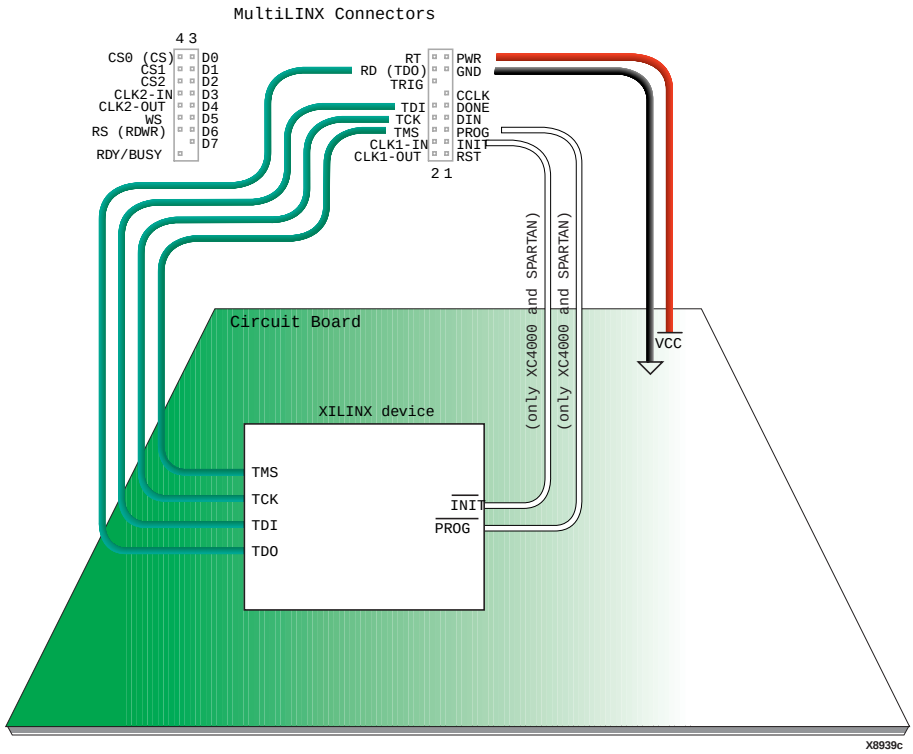

Figure 79: **Spartan, XC5200, and XC4000 JTAG Mode**

Downloading and Verifying - Slave Serial Mode

This section details the connections needed for downloading and verifying configuration data with the MultiLINX cable set in Slave Serial Mode.

XC3000 Devices

Figure 80 shows the Slave Serial Mode connections for XC3000 devices.

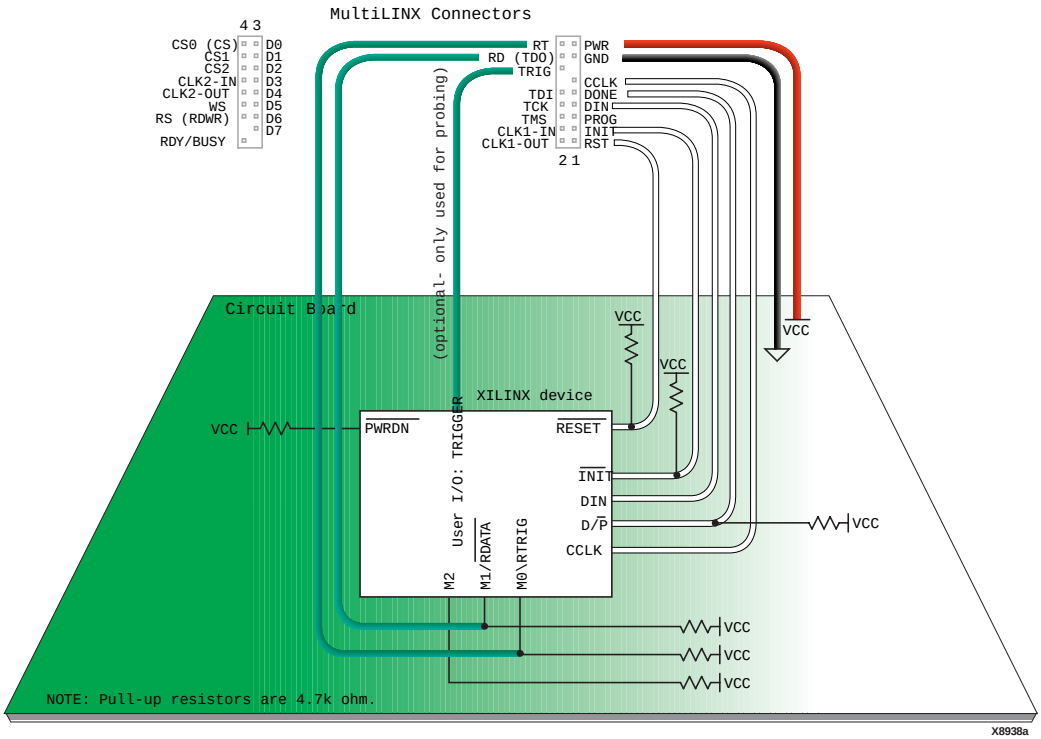

Figure 80: **XC3000 Slave Serial Mode**

Spartan, XC5200, and XC4000 Devices

Figure 81 shows the Slave Serial Mode connections for Spartan, XC5200, and XC4000 devices.

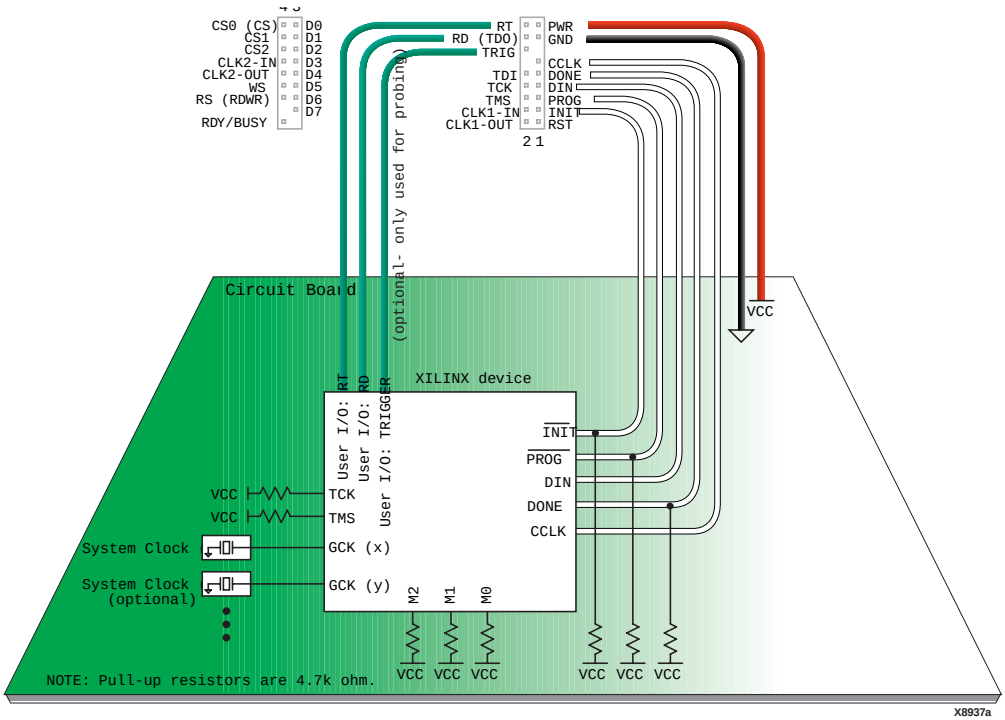

Figure 81: **Spartan, XC5200, and XC4000 Slave Serial Mode**

Downloading and Verifying - SelectMAP Mode

Virtex Devices

Figure 82 shows the SelectMAP Mode connections for downloading and verifying configuration data with Virtex devices.

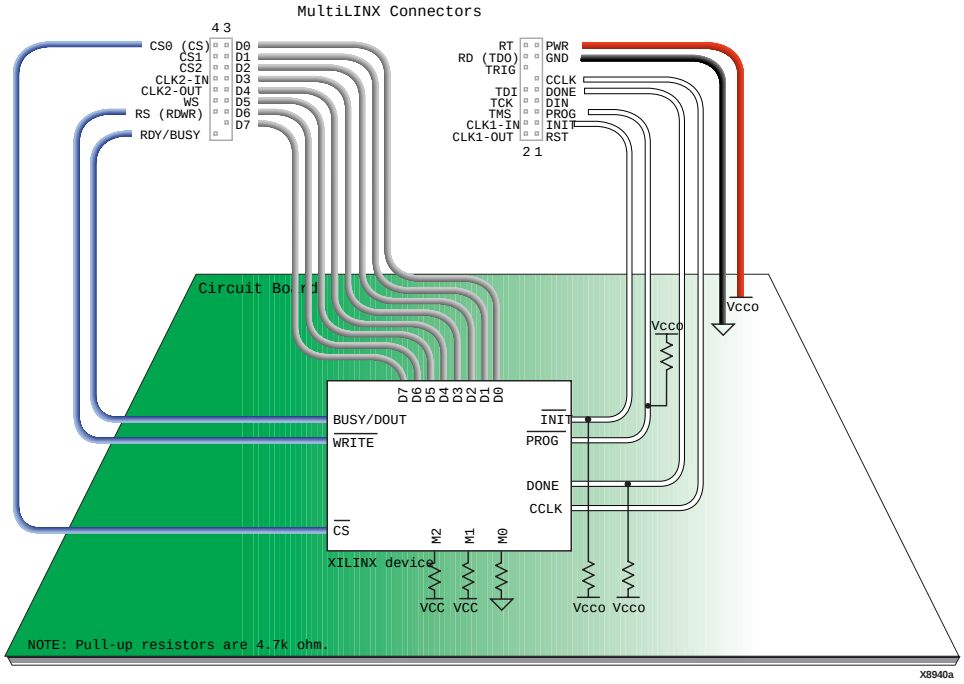

Figure 82: **Virtex SelectMAP Mode**

Downloading and Verifying - JTAG Mode

Figures 9-12 through 9-14 show the JTAG Mode connections for downloading and verifying configuration data for the different Xilinx device families.

XC9000 Devices

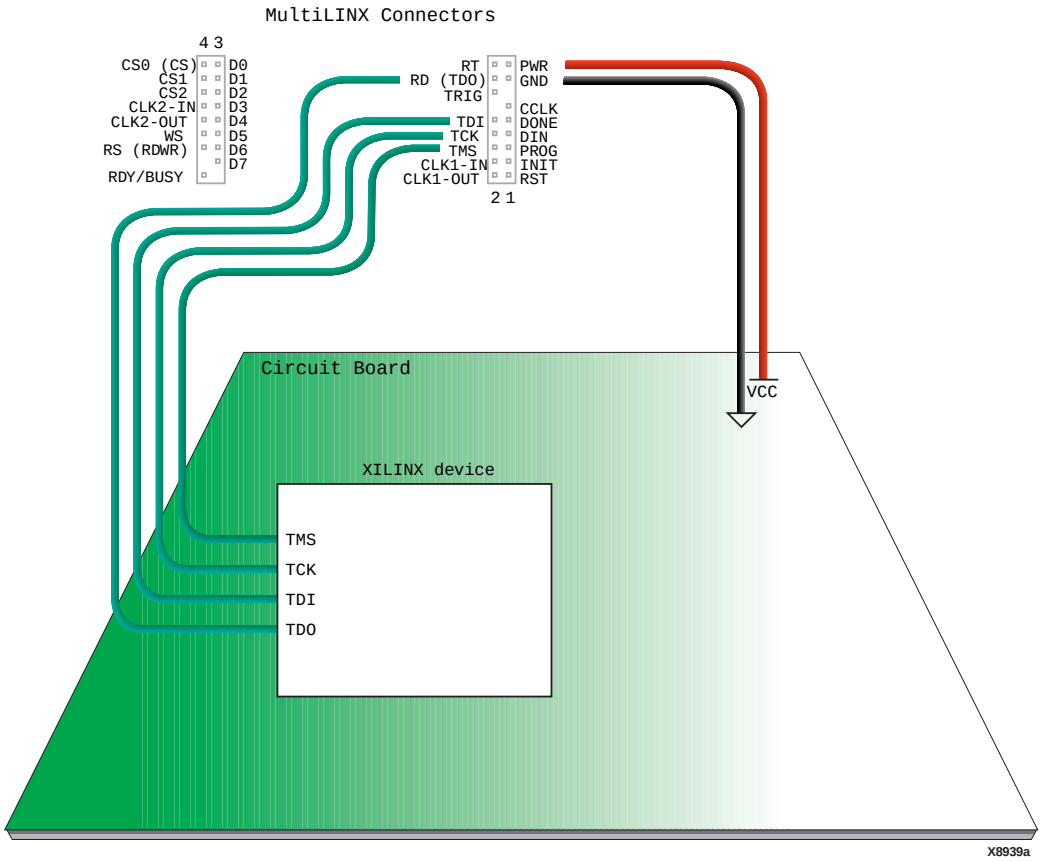

Figure 83: **XC9000 Devices - Downloading and Verifying Configuration Data, JTAG Mode**

Virtex Devices

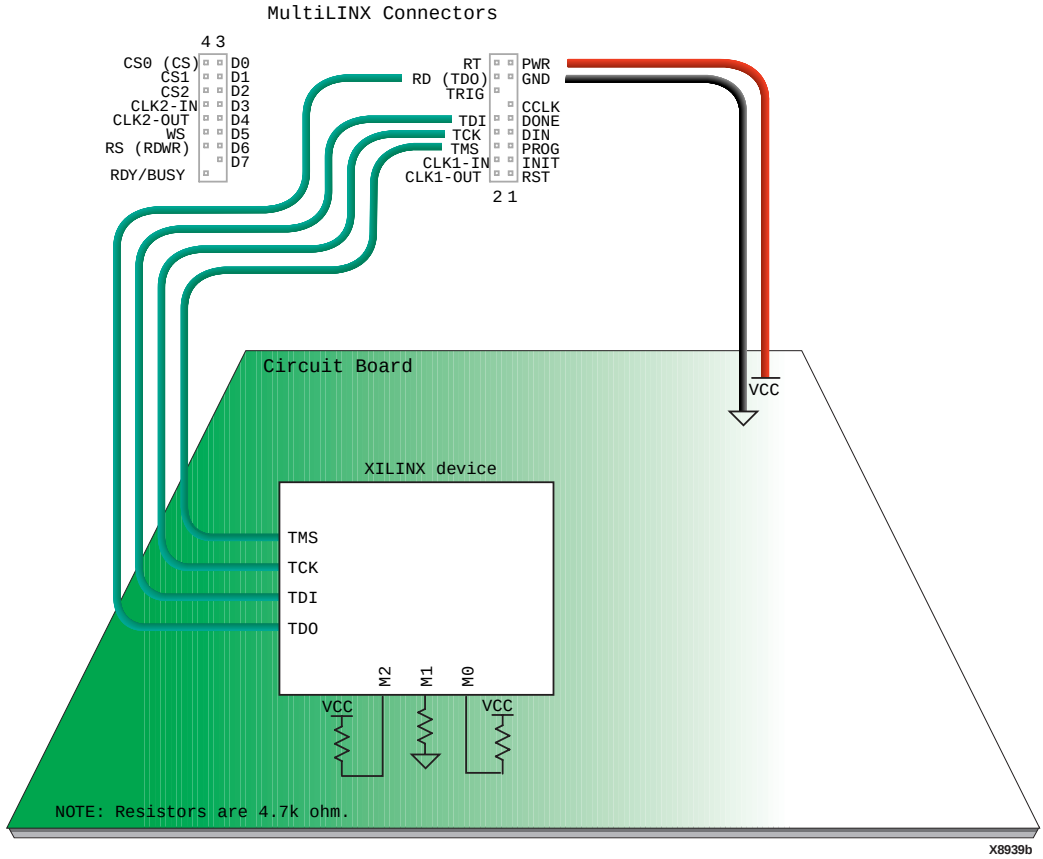

Figure 84: **Virtex Devices - Downloading and Verifying Configuration Data, JTAG Mode**

Spartan, XC5200, and XC4000 Devices

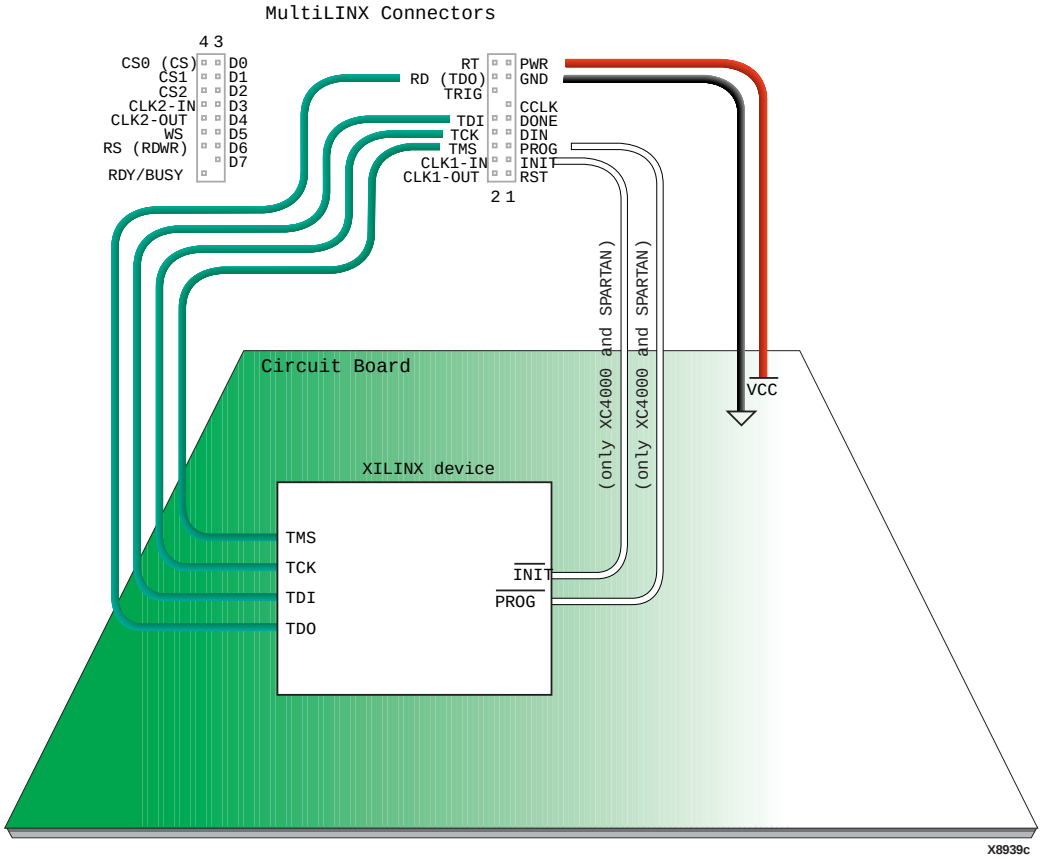

Figure 85: **Spartan, XC5200, and XC4000 Devices - Downloading and Verifying Configuration Data, JTAG Mode**

Verifying Configuration Data Only

This section details the connections needed for verifying configuration data using the MultiLINX cable set.

Spartan, XC5200, and XC4000 Devices

Figure 86 shows the connections for verifying configuration data with Spartan, XC5200, and XC4000 devices.

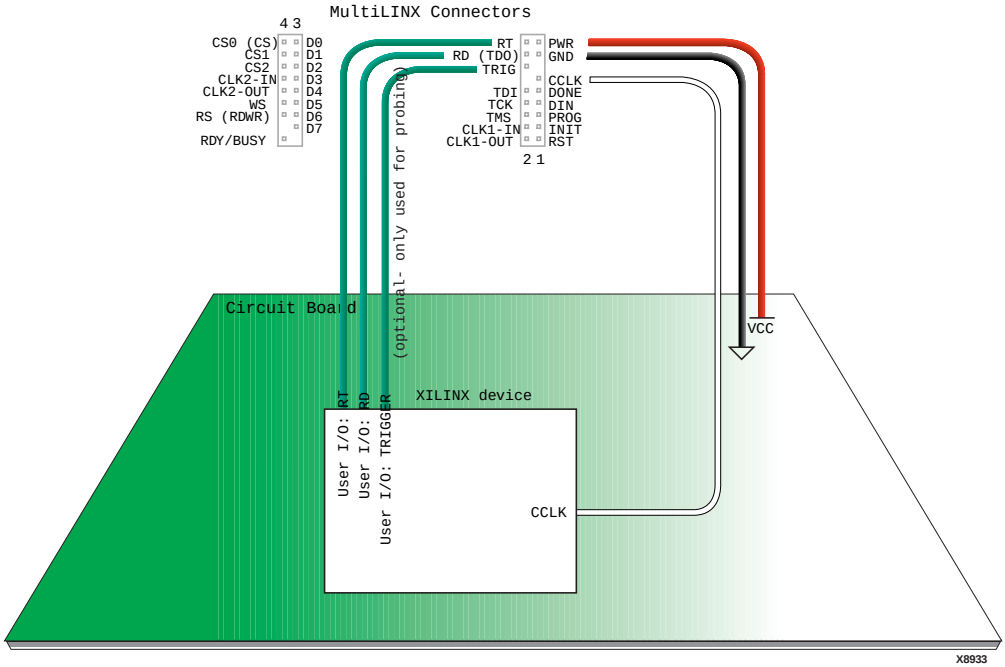

Figure 86: **Spartan, XC5200, and XC4000 Configuration Data Verification Only**

XC3000 Devices

Figure 87 shows the connections for verifying configuration data with XC3000 devices.

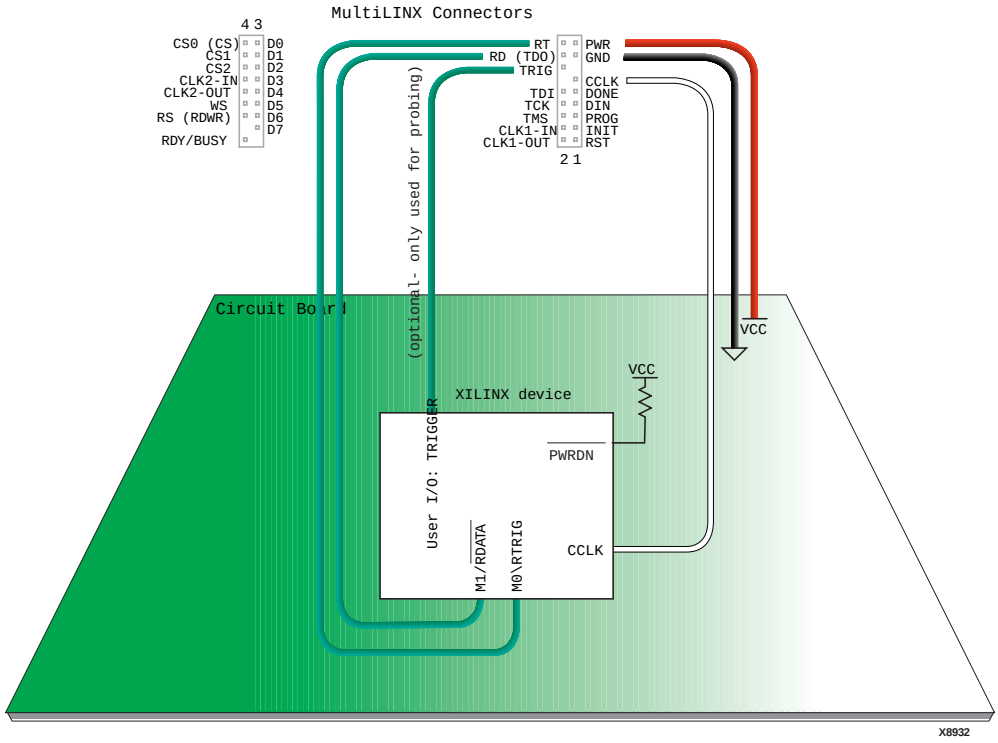

Figure 87: **XC3000 Configuration Data Verification Only**

XILINX®

Appendix A

Virtex Configuration Beginner's Guide

Introduction

The Virtex architecture uses SRAM configuration elements that require configuration data to be loaded each time the device powers up or when the FPGA needs to be reloaded in the system. The process of physically loading the SRAM data into the FPGA is called configuration. A stream of configuration bits generated by the software tools representing the design is loaded in either serial or parallel fashion. This stream of configuration bits is called a bitstream. After all configuration bits have been downloaded, the startup phase is executed. During startup, the device releases global resources, such as global 3-state, global write enable, and global reset. After startup, the device goes into user mode and performs the programmed function.

Power Consideration

The maximum V_{CC} rise time must remain less than the requirement of <50 ms. Refer to the Virtex data sheet at http://www.xilinx.com/partinfo/ds003.pdf. 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 1.0 V for power-up to continue. If this bank is not being used, a pull-up should be added to V_{CCO_2}.

The duration of the V_{CC} ramp is dependent on the amount of current that is available from the power supply. In this way, a Virtex device can be modeled as a capacitor of 500 µF. If a large amount of current is available, the V_{CC} ramp will be very fast. When it has reached its final voltage, it no longer requires the current. Likewise, if the available current is limited, the rise time will be lengthened.

For example, if the designer allows for at least 0.5 amperes of supply current to any Virtex family member and the power supply has a limiting response, the device will always initialize properly and be ready for configuration. If the current is not available, the device might not initialize. The Virtex device tends to initialize with low supply currents. However, the data sheet's 50 ms ramp-up time will be exceeded, if less than 500 mA is available. The 50-ms maximum ramp-up time is under consideration for relaxation at this time, as the Virtex family does not appear to be sensitive to duration of the ramp-up time. As noted, if more current is available and if the ramp-up time is faster than the 2 ms specified, the current into the device can actually be larger than 0.5 amperes.

There are new power supply ICs (for example, Texas Instrument's TI TPS5210) that are able to source up to 20 amperes at 2.5 V, with a transient regulation of better than 2% for no load to full load in less than 0.1 ms. Such devices can be considered in designs where the minimum and maximum currents vary by such a large amount. For more details on the Texas Instrument power supplies, go to http://www.ti.com.

Master Serial Configuration From PROMs

When looking for a sure and simple way to configure a Virtex or Virtex-E device, the Master Serial mode from PROMs is the simplest interface to select, using only five active FPGA pins and four PROM pins. The Master Serial mode is selected by simply pulling all three mode pins (M2, M1, M0) to Ground. In this mode, the user I/Os are floating during configuration. Alternately, if the user wants the I/O pins pulled up, the mode pin should be set to M2=1, M1=0 and M0=0. If the mode pins are left unconnected, then the configuration defaults to Slave Serial mode, because by default the mode pins are weakly pulled up.

When the FPGA is in Master Serial mode, it generates a configuration clock that drives the serial PROM. The user does not need to program a clock because the FPGA generates a clock (CCLK) with selectable frequencies. A short access time after the rising clock edge, data appears on the PROM DATA output pin that is connected to the FPGA DIN pin. The FPGA accepts this data on each rising edge of the CCLK and generates the appropriate number of clock pulses to complete the configuration. Once configured, it disables the PROM.

Starting at the slow default frequency, a wide range of frequencies can be selected for CCLK. Configuration bits then switch CCLK to a higher frequency for the remainder of the configuration. Switching to a lower frequency is prohibited. The CCLK frequency is set using the ConfigRate option in the bitstream generation software. The maximum CCLK frequency that can be selected is 60 MHz.

When selecting a CCLK frequency, ensure that the serial PROM and any daisy-chained FPGAs are fast enough to support the clock rate. For the XC1700E/L family, the maximum frequency is 15 MHz; and for the XC1800 family, the maximum is 33 MHz. On power-up, the CCLK frequency is approximately 2.5 MHz. This frequency is used until the ConfigRate bits have been loaded, then the frequency changes to the selected ConfigRate. Unless a different frequency is specified in the bitstream generator (BitGen) by the designer, the default ConfigRate is 4 MHz.

Configuring a Single Virtex Device with a Serial XC1700 PROM

Currently, Xilinx offers a one-chip PROM solution for all its Virtex FPGAs. That is, the largest Virtex device can be configured from a single PROM. See Table 119. However, the introduction of future Virtex family members might require multiple PROMs.

For generating the bitstream that configures the FPGA, either the Xilinx Alliance or Foundation series development system compiles the FPGA design file into a standard Hex format, which is then transferred to the PROM by most commercial PROM programmers.

Table 119: **Virtex Series Devices**

				Virtex			Virtex-E			
					# of 32-bit Read-back Words (1)				# of 32-bit Read-back Words (1)	
Device	Row × Col.	Bits/ Frame	Words/ Frame	RAM Cols	CLB (all)	RAM (1 col)	RAM Cols	RAM Space	CLB (all)	RAM (1 col)
XCV50	16 × 24	384	12	2	15,876	780	-	-	-	-
XCV50E	16 × 24	384	12	-	-	-	4	6	16,524	780
XCV100	20 × 30	448	14	2	22,554	910	-	-	-	-
XCV100E	20 × 30	448	14	-	-	-	4	12	23,310	910
XCV150	24 × 36	512	16	2	30,384	1,040	-	-	-	-
XCV200	28 × 42	576	18	2	39,366	1,170	-	-	-	-

Table 119: **Virtex Series Devices *(Continued)***

Device	Row × Col.	Bits/ Frame	Words/ Frame	Virtex			Virtex-E			
				RAM Cols	# of 32-bit Read-back Words (1)		RAM Cols	RAM Space	# of 32-bit Read-back Words (1)	
					CLB (all)	RAM (1 col)			CLB (all)	RAM (1 col)
XCV200E	28 × 42	576	18	-	-	-	4	12	40,338	1,170
XCV300	32 × 48	672	21	2	51,975	1,365	-	-	-	-
XCV300E	32 × 48	672	21	-	-	-	4	12	53,109	1,365
XCV400	40 × 60	800	25	2	76,275	1,625	-	-	-	-
XCV400E	40 × 60	800	25	-	-	-	4	12	77,625	1,625
XCV405E	40 × 60	800	25	-	-	-	14	4	84,375	1,625
XCV600	48 × 72	960	30	2	108,810	1,950	-	-	-	-
XCV600E	48 × 72	960	30	-	-	-	6	12	112,050	1,950
XCV800	56 × 84	1088	34	2	142,902	2,210	-	-	-	-
XCV812E	56 × 84	1088	34	-	-	-	20	4	159,426	2,210
XCV1000	64 × 96	1248	39	2	186,381	2,535	-	-	-	-
XCV1000E	64 × 96	1248	39	-	-	-	6	12	190,593	2,535
XCV1600E	72 × 108	1376	43	-	-	-	8	12	237,231	2,795
XCV2000E	80 × 120	1536	48	-	-	-	8	12	292,464	3,120
XCV2600E	92 × 138	1728	54	-	-	-	8	12	375,678	3,510
XCV3200E	104 × 156	1952	61	-	-	-	8	12	477,081	3,965

Notes:
1. Includes pad frame in calculation.

Connecting the FPGA Device with the PROM

- The DATA output of the PROM drives the DIN input of the lead FPGA device.
- The Master FPGA CCLK output drives the CLK input of the PROM.
- The CEO output of a PROM drives the CE input of the next PROM in a daisy chain (if any).
- The $\overline{\text{RESET}}$ / OE input of all PROMs is best driven by the INIT output of the FPGA device. This connection assures that the PROM address counter is reset before the start of any (re)configuration, even when a reconfiguration is initiated by a V_{CC} glitch. Other methods, such as driving $\overline{\text{RESET}}$ / OE from LDC or system reset, assume that the PROM internal power-on-reset is always in step with the FPGA's internal power-on-reset. This is not a safe assumption.
- The CE input of the lead (or only) PROM is driven by the DONE output of the FPGA device, provided that DONE is not permanently grounded. CE can also be permanently tied Low, but this keeps the DATA output active and causes an unnecessary supply current of 10 mA maximum.

Virtex devices have separate voltage sources for the internal core circuitry and the I/O circuitry (SelectI/O™). The SelectI/O resource is separated into eight banks of I/O groups. Each bank can 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 at http://www.xilinx.com/partinfo/databook.htm#virtex. Before and during configuration, all I/O banks are set for the Low Voltage TTL (LVTTL) standard, which requires an output voltage (V_{CCO}) of 3.3 V for operation with the XC1700 family. All configuration pins are located within banks 2 and 3. Therefore, only V_{CCO_2} and V_{CCO_3} pins need a 3.3 V supply for output configuration pins to operate normally. If banks 2 and 3 are being configured for an I/O standard that requires a V_{CCO} other than 3.3 V, then V_{CCO_2} and V_{CCO_3} need to be switched from the 3.3 V used during startup to the voltage required after configuration.

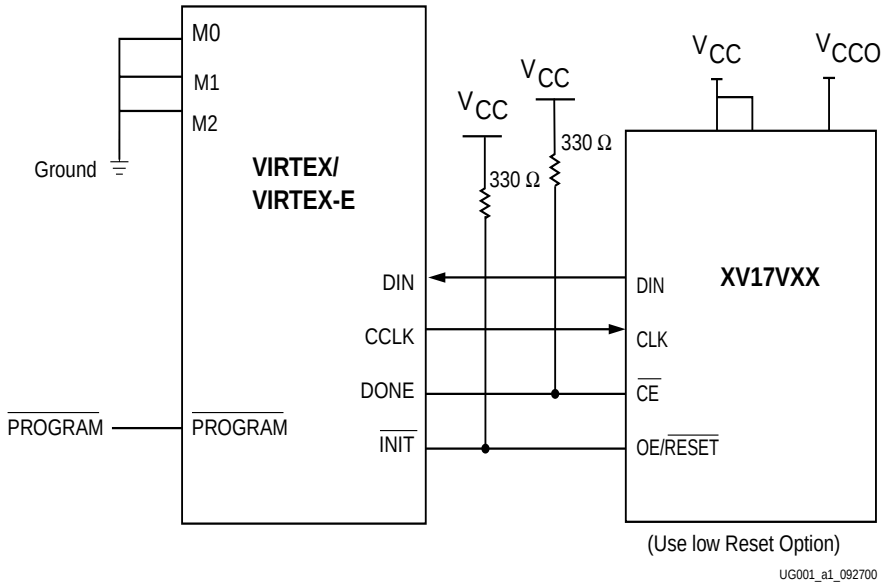

Figure 88: **Virtex/Virtex-E Device and XV17VXX Device Schematic**

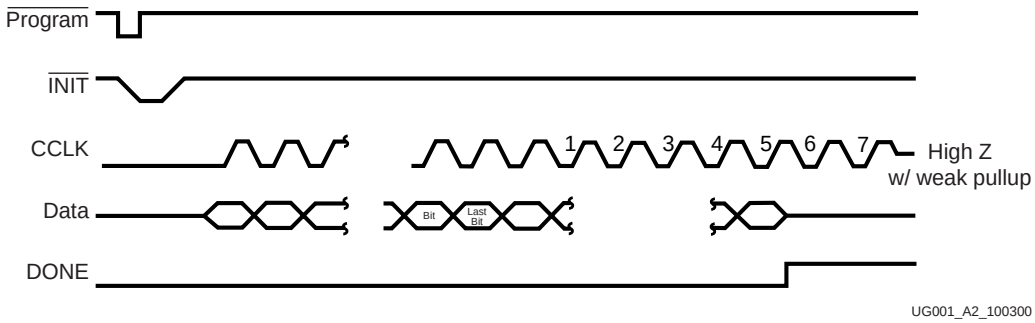

Figure 89: **Master Serial Clocking Sequence**

Configuration Flow

The configuration of Virtex devices is a 3-phase process. First, the configuration memory is cleared. Next, configuration data is loaded into the SRAM memory; and finally, the logic is activated by a startup process.

After power-up or asserting PROGRAM, the configuration memory is automatically cleared. The INIT pin transitions High when the clearing of configuration memory is complete. A logic Low on the 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 that 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 Virtex configuration logic requires several CCLK transitions to initialize itself. For this purpose, the Virtex bitstream is padded with a dummy data word at the beginning of the configuration stream, followed by a synchronization word. The minimum Low pulse time for $\overline{\text{PROGRAM}}$ is 300 ns. There is no maximum value. The FPGA samples its mode pins on the rising edge of $\overline{\text{INIT}}$. After $\overline{\text{INIT}}$ has gone High, configuration can begin. No additional timeout or waiting periods are required, but configuration does not need to commence immediately after the transition of $\overline{\text{INIT}}$. For further delaying configuration, the $\overline{\text{INIT}}$ pin can also be held Low externally. If necessary at all, it is preferred to delay configuration by holding the $\overline{\text{INIT}}$ pin Low, rather than the $\overline{\text{PROGRAM}}$ pin. The configuration logic does not begin processing data until the synchronization word from the bitstream is loaded.

After initialization and after the synchronization word (embedded in the bitstream) has been recognized, the bitstream is loaded one bit at a time. Bitstream sizes for the various devices are listed in Table 120. Physically, the FPGA configuration is organized in Frames. For more information, consult Xilinx applications notes, XAPP138: “Virtex FPGA Series Configuration and Readback” and XAPP151: “Virtex Series Configuration Architecture User Guide.”

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.

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 in a default sequence. The DONE pin going High deselects the PROM. The CCLK stops and goes into 3-state mode with a weak pullup. At this point, the FPGA becomes active and functional with the loaded design.

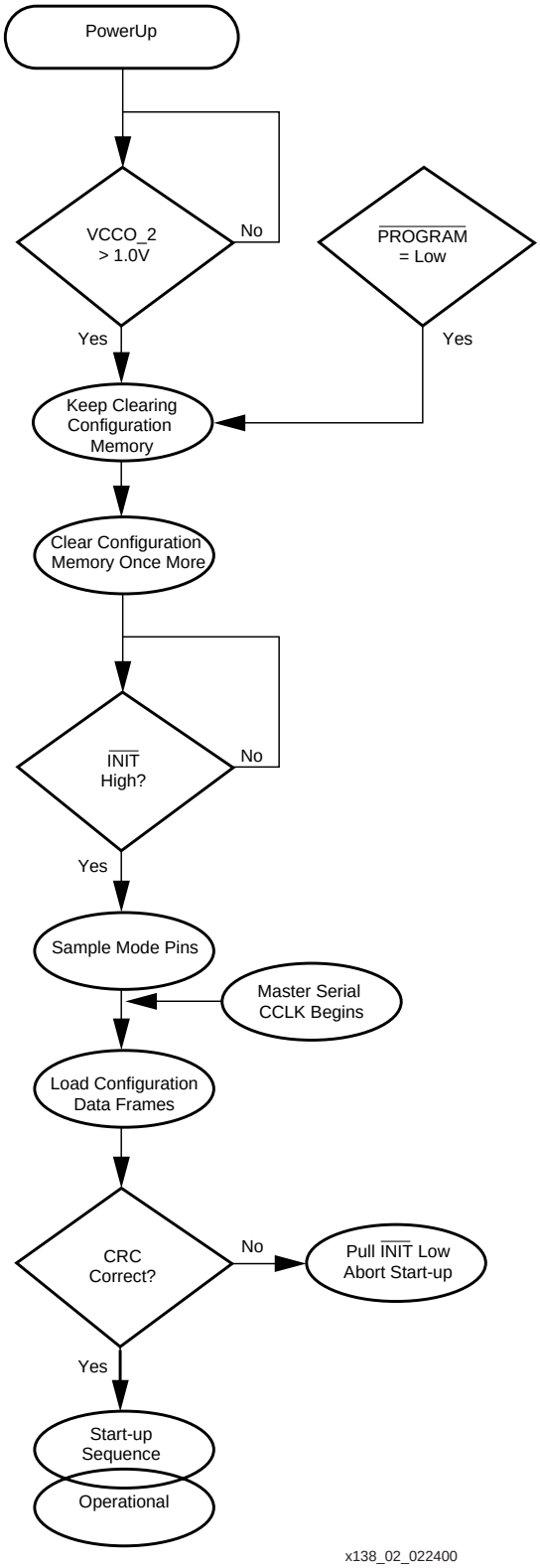

Figure 90: **Configuration Flow Diagram**

Table 120: **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 (user defined)	---- ----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

Software Options

For a simple, single FPGA with single PROM design, the default options in the software are typically sufficient and safe to use. The user can elect to change the timing sequence for the release of the global signals for more complex configuration schemes. For more information, consult the Xilinx application note, XAPP138: "Virtex FPGA Series Configuration and Readback" at http://www.xilinx.com/xapp/xapp138.pdf

Debugging Hints

The following are suggested actions to take or areas to check when debugging:

1. Are VCC_{INT}, V_{CCO} (banks 2 + 3) set to correct values?
2. Check for solder bridges between pins.
3. Make sure the $\overline{INIT}$ pin is not driven Low.
4. Is $\overline{PROGRAM}$ High, $\overline{INIT}$ High, DONE Low before starting?
5. Are the mode pins set to the correct values?
6. Is the FPGA generating CCLK and is it reaching the PROM?
7. Is there ringing (reflections) on the CCLK?
8. Is the PROM programmed for the correct OE / $\overline{RESET}$ polarity?
9. Is there a 330-Ω PULLUP on DONE? (or DriveDone if last in chain).

10. Is the V_{PP} pin of the PROM connected to V_{CC}?
11. Are the configuration bits making it to the DIN pin of the FPGA?
12. Make sure the PROM is powered up and sending data when the FPGA starts CCLK.
13. $\overline{\text{INIT}}$ does not go High:
    a. The rise time for the core voltage is too slow. Maximum rise time should be 50 ms (1 V to 2.4 V). Hold of start of configuration.
    b. The $\overline{\text{PROGRAM}}$ pulse is too short. An FPGA feature is the ability to reconfigure the device after power-up by toggling the $\overline{\text{PROGRAM}}$ pin to Low, resetting the configuration state machine. After the $\overline{\text{PROGRAM}}$ pin is released, the FPGA restarts the configuration process. The minimum pulse width for the $\overline{\text{PROGRAM}}$ pin on a Virtex series device is 200 ns.
    c. The $\overline{\text{PROGRAM}}$ pin is held Low.
14. $\overline{\text{INIT}}$ goes Low during programming:
    a. The $\overline{\text{INIT}}$ pin goes Low on a certain frame, usually indicating a Frame Error has occurred during configuration. A Data Frame Error can be an indication of a corrupted bit file. If the file has been ported through e-mail or an FTP site, it is possible that it has become corrupted during the transfer. For ASCII file transfer, try binary mode or compressed mode to avoid corruption. ASCII mode transfer can add extra carriage returns or line feeds to the bit file.
15. DONE does not go High:

    A problem occurs if all the following conditions are met:

    a. The $\overline{\text{CS}}$ and $\overline{\text{WRITE}}$ lines are both used as I/O after configuration.
    b. $\overline{\text{CS}}$ is Low during configuration, and $\overline{\text{WRITE}}$ toggles during configuration.
    c. If ALL three conditions are met, configuration fails and DONE does not go High. Toggling the $\overline{\text{PROGRAM}}$ pin and reloading configuration *without* one of the above conditions results in a successful configuration.
16. If DONE goes High, but the device does not start:

    A Virtex device is configured at high speed, DONE goes High, but the device does not start, that is, the I/O is still 3-stated.
17. If the DONE pin rises slowly, attach a 330-Ω external resistor to DONE..
    a. If DONE still does not rise within one CCLK cycle, set the BitGen option to DonePipe=Yes. With this option set to Yes, the device adds a pipelined register stage to the DONE input (CFG_DONE) path. This is only needed when the DONE rise time exceeds the CCLK period. For example, CCLK is running at 33 MHz in Slave Serial mode. If the rise time of DONE is >25 ns, delay the CFG_DONE signal using DonePipe=Yes.
    b. If the DONE pin rises slowly, try setting the BitGen option to DriveDone = Yes to actively drive the DONE pin.

Notes:

1. This can only be done when configuring one device or when the device is the last one in a daisy-chain. For more information, consult Chapter 7 of the *Virtex Configuration Guide*.

18. DONE does not go High, and $\overline{\text{INIT}}$ does not go Low?

    This condition indicates that the Virtex synchronization word is not being captured, or a bit is being added or lost. The Virtex synchronization word is: AA995566. The first two hex values of AA are represented in a binary byte as: 1010 1010. If a bit is either added or lost during configuration (caused by a clock glitch or noise), the remainder of the bitstream is misaligned. Each unit after the false edge is one bit off, so it cannot be

recognized properly. As a consequence, no more data can be loaded, there is no CRC, $\overline{\text{INIT}}$ does not go Low, and DONE does not go High.

19. The DONE pin appears to be externally held Low?

    The DONE pin is an open-drain driver that must be pulled up to achieve a logic High. Although the FPGA does have a programmable internal pull-up resistor to the DONE pad, use an external 330-Ω resistor for Virtex devices. If necessary, separate the DONE pin from the board to verify if an external source is holding it Low.

20. Double check the STARTUP sequence options selected for Bitstream generation. Recommended STARTUP Options for Virtex Series:

    DONE_cycle: 4

    GTS_cycle: 5

    GSR_cycle: 6

    GWE_cycle: 6

21. The FPGA heats up significantly and / or draws significantly more current than expected.

    This is usually caused by loading an incorrect bitstream (for example, trying to load an XCV300 bitstream into an XCV400 device).

Appendix B

BitGen and PROMGen Program Information

BitGen

The BitGen program is compatible with the following families.

- XC3000A / L
- XC3100A / L
- XC4000E / L
- XC4000EX / XL / XV / XLA
- XC5200
- Spartan / XL / 2
- Virtex

BitGen produces a bitstream for Xilinx device configuration. After the design has been completely routed, it is necessary to configure the device so that it can execute the desired function. This is done with BitGen, the Xilinx bitstream generation program. BitGen takes a fully routed NCD (Circuit Description) file as its input and produces a configuration bitstream—a binary file with a .bit extension.

The BIT file contains all of the configuration information from the NCD file defining the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file can then be downloaded into the FPGA memory cells, or it can be used to create a PROM file (see 41).

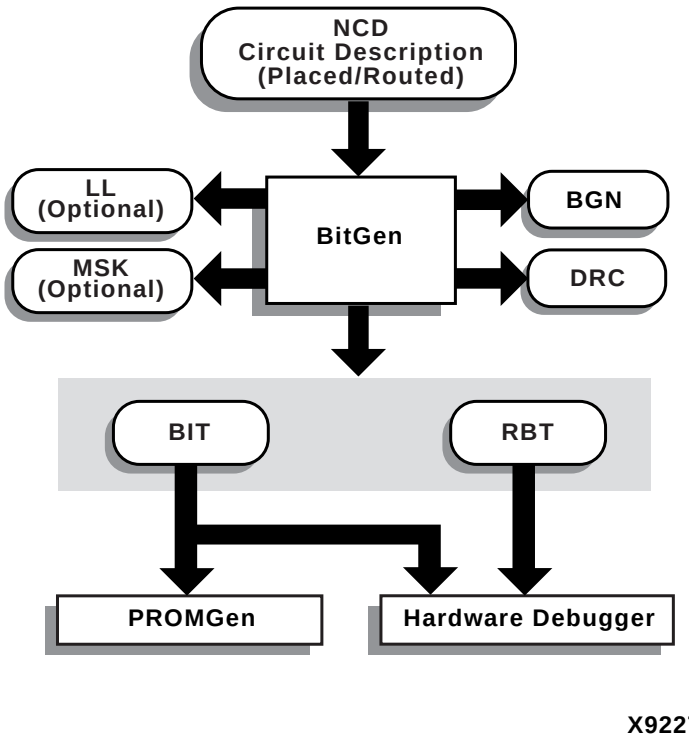

Figure 91: **BitGen Diagram**

BitGen Syntax

The following syntax creates a bitstream from your NCD file.

bitgen [*options*] *infile*[**.ncd**] [*outfile*] [*pcf_file*]

options is one or more of the options listed in the "BitGen Options" on page 211.

Infile is the name of the NCD design for which you want to create the bitstream. You can specify only one design file, and it must be the first file specified on the command line.

You do not have to use an extension. If you do not, **.ncd** is assumed. If you do use an extension, it must be **.ncd**.

Outfile is the name of the output file. If you do not specify an output file name, BitGen creates one in the same directory as the input file. If you specify -l on the command line, the extension is .ll (see -l command line option). If you specify -m (see -m command line option), the extension is .msk. If you specify -b, the extension is .rbt. Otherwise the extension is .bit. If you do not specify an extension, BitGen appends one according to the aforementioned rules. If you do include an extension, it must also conform to the rules.

Pcf_file is the name of a physical constraints (PCF) file. BitGen uses this file to determine which nets in the design are critical for tiedown (see "-t (Tie Unused Interconnect)" on page 236). BitGen automatically reads the .pcf file by default. If the physical constraints file is the second file specified on the command line, it must have a .pcf extension. If it is the third file specified, the extension is optional; .pcf is assumed. If a .pcf file name is specified, it must exist, otherwise the input design name with a .pcf extension is read if that file exists.

A report file containing all BitGen's output is automatically created under the same directory as the output file. The report file has the same root name as the output file with a .bgn extension.

BitGen Files

The input files that BitGen requires and the output files that BitGen generates are described below.

Input Files

Input to BitGen consists of the following files.

- NCD file—a physical description of the design mapped, placed and routed in the target device. The NCD file must be fully routed.
- PCF—an optional user-modifiable ASCII Physical Constraints File. If you specify a PCF file on the BitGen command line,
  BitGen uses this file to determine which nets in the design are critical for tiedown (see "-t (Tie Unused Interconnect)" on page 236).

Output Files

Output from BitGen consists of the following files.

- BIT file—a binary file with a .bit extension. The BIT file contains all of the configuration information from the NCD file defining the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file can then be downloaded into the FPGA memory cells, or it can be used to create a PROM file (see "PROMGen" on page 237).
- RBT file—an optional "rawbits" file with an .rbt extension. The rawbits file is ASCII ones and zeros representing the data in the bitstream file. If you enter a -b option on the BitGen command line, an RBT file is produced in addition to the binary BIT file (see "-b (Create Rawbits File)" on page 212).
- LL file—an optional ASCII logic allocation file with a .ll extension. The logic allocation file indicates the bitstream position of latches, flip-flops, and IOB inputs and outputs. A .ll file is produced if you enter a -l option on the BitGen command line (see "-l (Create a Logic Allocation File)" on page 236).
- MSK file—an optional mask file with an .msk extension. This file is used to compare relevant bit locations for executing a readback of configuration data contained in an operating FPGA. A MSK file is produced if you enter a -m option on the BitGen command line see "-m (Generate a Mask File)" on page 236).
- BGN file—a report file containing information about the BitGen run.
- DRC file—a Design Rule Check (DRC) file for the design. A DRC runs and the DRC file is produced unless you enter a -d option on the BitGen command line see "-d (Do Not Run DRC)" on page 212).

BitGen Options

Following is a description of the command line options and how they affect the behavior of BitGen.

-a (Tie All Interconnect)

Used with the -t option to force tiedown to fail if all nodes are not tied. This option also allows tiedown to implement user signals.

-b (Create Rawbits File)

Create a "rawbits" (*file_name*.rbt) file. The rawbits file consists of ASCII ones and zeros representing the data in the bitstream file.

If you are using a microprocessor to configure a single FPGA, you can include the rawbits file in the source code as a text file to represent the configuration data. The sequence of characters in the rawbits file is the same as the sequence of bits written into the FPGA.

-d (Do Not Run DRC)

Do not run DRC (Design Rule Check). Without the -d option, BitGen runs a DRC and saves the DRC results in two output files: the BitGen report file (*file_name*.bgn) and the DRC file (*file_name*.drc). If you enter the -d option, no DRC information appears in the report file and no DRC file is produced.

Running DRC before a bitstream is produced detects any errors that could cause the FPGA to malfunction. If DRC does not detect any errors, BitGen produces a bitstream file (unless you use the -j option described in the "-j (No BIT File)" on page 236).

You cannot disable the DRC with the -d option if you have specified a -t (Tie Unused Interconnect) option. The DRC always runs if you specify -t.

-f (Execute Commands File)

```
-f command_file
```

The -f option executes the command line arguments in the specified *command_file*.

```
-g (Set Configuration)
```

The -g option specifies the startup timing and other bitstream options for Xilinx FPGAs. The settings for the -g option depend on the design's architecture. These settings are described in the following sections.

- "-g (Set Configuration—XC3X00 Devices)"
- "-g (Set Configuration—XC4000 and Spartan)"
- "-g (Set Configuration—XC5200 Devices)"
- "-g (Set Configuration—Virtex and Spartan2 Devices)"

-g (Set Configuration—XC3X00 Devices)

The -g option has sub-options that represent settings you use to set the configuration for an XC3X00A / L design. These options have the following syntax.

```
bitgen -g option:setting
```

For example, to set the input signal thresholds to CMOS level instead of TTL level, use the following syntax.

```
bitgen -g inputs:CMOS
```

The following sections describe the startup sequences for the -g option applied to an XC3X00 design DONE pin.

Enables or disables internal pull-up on the DONE / $\overline{\text{PROGRAM}}$ (D / $\overline{\text{P}}$) pin. The Pullnone setting indicates there is no connection to the pull-up.

Use this option only if you are planning to connect an external pull-up resistor to this pin. The internal pull-up resistor has a value of 2 to 8K ohm and is automatically connected if you do not use this option.

The D / $\overline{P}$ pins configure an open-drain driver that requires a pull-up resistor to indicate the end of the configuration.

Architectures:	XC3000A / L, XC3100A / L
Settings:	Pullup, Pullnone
Default:	Pullup

DoneTime

Releases the DONE / $\overline{PROGRAM}$ (D / $\overline{P}$) pin one CClk cycle before the IOBs become active (Before setting) or one CClk cycle after the IOBs become active (After setting).

The After setting clearly indicates the end of the configuration process. The Before setting can be used to de-activate external configuration drivers so that they do not contend with active outputs on the same pin. The use of After would create a 1-CClk-period contention. The alternative, using the LDC output, might cause a short contention spike. Before avoids these problems.

Architectures:	XC3000A / L, XC3100A / L
Settings:	Before, After
Default:	Before

Input

This option sets the FPGA design input-signal thresholds to TTL or CMOS level for interface capability. CMOS improves noise immunity and reduces static power consumption.

The special-purpose clock inputs, TCLKIN, BCLKIN, and $\overline{PWRDN}$ always require CMOS-level signals, even if the FPGA design input thresholds are specified as TTL compatible

Architectures:	XC3000A / L, XC3100A / L
Settings:	TTL, CMOS
Default:	TTL

LC_Alignment

Determines how length count is calculated to control when the device changes from configuration to user operation. The two methods of calculating length count, DONE Alignment and Length Count Alignment, are discussed in *The Programmable Logic Data Book*. The *FPGA Configuration Guidelines Application Note* also contains length count information.

Architectures:	XC3000A / L, XC3100A / L
Settings:	Length, DONE
Default	Length

Oscillator

This option specifies crystal oscillator options for XC3X00 series devices. The crystal oscillator is associated with the auxiliary clock buffer in the lower-right corner of the die.

The Disable option disables the FPGA crystal oscillator; Enable enables it. The EnableDiv2 option enables the oscillator and divides the crystal output frequency by two in order to guarantee a symmetrical clock signal

Architectures:	XC3000A / L, XC3100A / L
Settings:	Disable, Enable, EnableDiv2
Default	Disable

ReadBack

This option specifies readback options for XC3X00 families. After the FPGA design has been configured, the FPGA configuration data can be read back and compared with the original configuration data. Readback is initiated by a Low-to-High transition on the M0 / RTRIG pin. Once you give the readback command, external logic must drive the CClk input to read back each data bit. The readback data appears on the $\overline{\text{RDATA}}$ pin.

The Disable option disables readback. The Once option enables a one-time readback and Command enables readback on command.

The Disable and Once options are used for design security. The Once option allows only one readback, typically performed during manufacturing. After this, readback can never be invoked again.
If the FPGA device is powered by a standby battery and the configuration source is removed, the FPGA design configuration data is completely secure from being read or copied.

Architectures:	XC3000A / L, XC3100A / L
Settings:	Command, Disable, Once
Default	Command

ResetTime

Removes INTERNAL RESET one clock cycle before or one clock cycle after the IOB becomes active.

When you specify the After setting, the outputs go active while all internal flip-flops are still being held in Reset. When you specify the Before setting, the internal logic becomes operational before the outputs go active.

Architectures:	XC3000A / L, XC3100A / L
Settings:	Before, After
Default	After

-g (Set Configuration—XC4000 and Spartan)

Notes:

For Spartan2 -g options, see "-g (Set Configuration—Virtex and Spartan2 Devices)" on page 229.

This option specifies the startup timing and other bitstream options for the XC4000E / L, XC4000EX / XL / XV, and Spartan devices. Timing sequences are predefined startup defaults with the following syntax:

```
bitgen -g timing_sequence
```

There are four valid startup sequences: Cclk_Nosync, Cclk_Sync, Uclk_Nosync, and Uclk_Sync. These startup sequences are described in the next section. For more information about startup timing, refer to *The Programmable Logic Data Book.*

The default startup sequence for the -g option is Cclk_Nosync. This startup sequence makes an XC4000 or Spartan device compatible with an XC3X00 device that is set for early Done and late Reset. Enter the following,

```
bitgen -g cclk_nosync
```

The -g option has sub-options that represent settings you use to set the configuration for an XC4000 or Spartan design. These options have the following syntax:

```
bitgen -g option:setting
```

For example, to enable Cyclic Redundancy Checking (CRC), use the following syntax:

```
bitgen -g crc:enable
```

The following sections describe startup sequences for the -g option.

Startup Sequences and the -g Option

This section describes the four predefined startup sequences and their defaults; then it describes the options, their settings, and their defaults.

Notes:

When mixing devices, the one with the latest "finished point" should be the master. The master stops clocking when it reaches the finished point. See *The Programmable Logic Data Book* for more information.

Cclk_Nosync

This is the default startup sequence for the -g option. Selecting this sequence causes the following defaults to take effect.

StartupClk:	CClk
SyncToDone:	No
DoneActive:	C1
OutputsActive:	C2
GSRInactive:	C3

This startup sequence makes an XC4000, Spartan, or XC5200 device consistent with an XC3X00 device set for early Done and late Reset.

Cclk_Sync

Selecting this sequence causes the following defaults to take effect.

StartupClk:	CClk
SyncToDone:	Yes
DoneActive:	C1
OutputsActive:	DI_PLUS_1
GSRInactive:	DI_PLUS_1

This startup sequence is the most consistent with the XC3X00 devices, since it synchronizes the release of GSR and I/Os to the external DoneIn signal. This startup sequence makes an XC4000 or Spartan device consistent with an XC3X00 device set for early Done and late Reset.

Uclk_Nosync

Selecting this sequence causes the following defaults to take effect

StartupClk:	Useclk
SyncToDone:	No
DoneActive:	U2
OutputsActive:	U3
GSRInactive:	U4

This startup sequence makes XC4000 or Spartan devices inconsistent with XC3X00 devices if they are in the same daisy chain, since the release of Done is synchronized to an external User Clock. There is no synchronization of I/Os or GSR to DoneIn.

Uclk_Sync

Selecting this sequence causes the following defaults to take effect.

StartupClk:	Userclk
SyncToDone:	Yes
DoneActive:	U2
OutputsActive:	D1_PLUS_1
GSRInactive:	D1_PLUS_2

This startup sequence makes XC4000 or Spartan devices inconsistent with XC3X00 devices if they are in the same daisy chain, since the release of Done is synchronized to an external User Clock. I/Os and GSR are synchronous to the clocks following DoneIn.

When using Uclk_Sync or Uclk_Nosync, you must provide a user clock to finish the configuration sequence. Without a user clock the FPGA will not configure.

Sub-Options for Startup Sequence (-g Option)

The sub-options available with the four startup sequences are described below. These sub-options use the **-g** *option*:*setting* syntax.

AddressLines

Determines the number of address lines (18 or 22) used for device configuration. The **22** setting activates four extra device pins as configuration address lines.

Architectures:	XC4000EX only (XC4000XL, XC4000XLA, and XC4000XV always have 22 active address lines)
Settings:	18, 22
Default	18

BSCAN_Config

When disabled, BSCAN_Config inhibits the BSCAN-based configuration after the device is successfully configured. This feature allows board testing without the risk of reconfiguring XLA devices by toggling the TCK / TMS / TDI / TDO lines.

Architectures:	XC4000XLA, XC4000XV, SpartanXL
Settings:	Disable, Enable
Default:	Enable

BSCAN_Status

When enabled, BSCAN_Status allows direct sensing of the DONE configuration state after performing a BSCAN-based configuration. Previously, there was no direct method for determining if a BSCAN-based configuration was successful.

Architectures:	XC4000XLA, XC4000XV, SpartanXL
Settings:	Disable, Enable
Default:	Disable

5-V Tolerant

If set to On, this option allows a 3.3V device circuitry to tolerate 5V operation. For any device that operates on a mixed circuit environment with 3.3V and 5V, ensure that On is set. For any circuitry that operates exclusively on 3.3V, such as in a laptop computer, set the option to Off. The Off option reduces power consumption.

Architectures:	XC4000XLA, XC4000XV, SpartanXL
Settings:	On, Off
Default:	On

ConfigRate

Selects the configuration clock rate. There are two choices: slow or fast. Slow is equivalent to 1 MHz, and fast is equivalent to 8 MHz (nominal).

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, and SpartanXL
Settings:	Slow, Fast
Default:	Slow

CRC

Enables or disables Cyclic Redundancy Checking (CRC) on a chip-by-chip basis during configuration.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, and SpartanXL
Settings:	Enable, Disable
Default:	Enable

DoneActive

Selects the event that activates the FPGA Done signal. There are a maximum of four events that you can select from at one time. These events are CClk edges or external (user) clock edges.

The actual options available at any time depend on the selections made for StartupClk and SyncToDone

Architectures:	XC4000E/L, XC4000EX/XL/XV/XLA, Spartan, and SpartanXL
Settings:	**C1** — first-CClk rising edge after the length count is met. **C2** — second-CClk rising edge after the length count is met. **C3** — third-CClk rising edge after the length count is met. **C4** — fourth-CClk rising edge after the length count is met. **U2** — second-valid-user-clock rising edge after C1. **U3** — third-valid-user-clock rising edge after C1. **U4** — fourth-valid-user-clock rising edge after C1.
Default:	C1

Valid settings for DoneActive are as follows:

StartupClk	SyncToDone	DoneActive
CClk	Yes	C1, C2 or C3
CClk	No	C1, C2, C3, or C4
UserClk	Yes	C1 or U2
UserClk	No	C1, U2, U3, or U4

DONEPin

Enables or disables internal pull-up on the DONE pin. The Pullnone setting indicates there is no connection to the pull-up.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, and SpartanXL
Settings:	Pullup, Pullnone
Default:	Pullup

ExpressMode

When enabled, ExpressMode creates a unique type of bitstream for configuration.

Architectures:	XC4000XLA, XC4000XV
Settings:	Disable, Enable
Default:	Disable

GSRInactive

Selects the event that releases the internal set-reset to the latches and flip-flops. You can select one of nine events: a CClk edge, an external (user) clock edge, or the external signal DoneIn. Only some of these events become options at one time depending on the combination of StartupClk and SyncToDone selected.

Architectures: XC4000E/L, XC4000EX/XL/XLA/XV, Spartan, SpartanXL

Settings:

C2 — second-CClk rising edge after the length count is met.

C3 — third-CClk rising edge after the length count is met.

C4 — fourth-CClk rising edge after the length count is met.

U2 — second-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met).

U3 — third-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met).

U4 — fourth-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met).

DI — when the DoneIn signal goes High.

DI_PLUS_1 — first CClk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High.

DI_PLUS_2 — second CClk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High.

Default: C3

Valid settings for GSRInactive are as follows:

StartupClk	SyncToDone	GSRInactive
CClk	Yes	C2, C3, DI, or DI_PLUS_1
CClk	No	C2, C3, or C4
UserClk	Yes	U2, DI, DI_PLUS_1, or DI_PLUS_2
UserClk	No	U2, U3, or U4

Input

Sets the input threshold level for IOBs.

Architectures: XC4000E/L, XC4000EX, Spartan

Settings: TTL, CMOS

Default: TTL

LC_Alignment

The LC_Alignment option determines how length count is calculated to control when the device changes from configuration to user operation. The two methods of calculating length count, DONE Alignment and Length Count Alignment, are discussed in the Configuration section of the The Programmable Logic Data Book. The FPGA Configuration Guidelines Application Note also contains length count information.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Length, DONE
Default	Length

M0 Pin

Adds a pull-up or a pull-down to the M0 (Mode 0) pin. Selecting one option enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, SpartanXL
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullnone

M1 Pin

Adds a pull-up or a pull-down to the M1 (Mode 1) pin. Selecting one option enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullnone

M2 Pin

Adds a pull-up or a pull-down to the M2 (Mode 2) pin. Selecting one option enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullnone

Output

Sets the output level for IOBs,

Architectures:	XC4000E/L, XC4000EX, Spartan
Settings:	TTL, CMOS
Default:	TTL

OutputsActive

Selects the event that releases the I/O from 3-state condition and turns the configuration related pins operational. There are a maximum of four events that you can select from at one time. These events are selected from a group of CClk edges, a group of external (user) clock edges, and the external signal DoneIn. The actual options available at any time depend on the selections made for StartupClk and SyncToDone.

Architectures: XC4000E/L, XC4000EX/XL/XLA/XV, Spartan, SpartanXL

Settings:

C2 — second-CClk rising edge after the length count is met.

C3 — third-CClk rising edge after the length count is met.

C4 — fourth-CClk rising edge after the length count is met.

U2 — second-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met)

U3 — third-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met)

U4 — fourth-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met)

DI — when the DoneIn signal goes High

DI_PLUS_1 — first CClk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High

DI_PLUS_2 — second CClk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High

Default: C2

Valid settings for OutputsActive are as follows:

StartupClk	SyncToDone	OutputsActive
CClk	Yes	C2, C3, DI, or DI_PLUS_1
CClk	No	C2, C3, or C4
UserClk	Yes	U2, DI, DI_PLUS_1, or DI_PLUS_2
UserClk	No	U2, U3, or U4

PowerDown

Enables or disables internal pull-up on the PowerDown pin. The Pullnone setting indicates there is no connection to the pull-up.

Architectures: SpartanXL

Settings:	Pullup, Pullnone
Default:	Pullup

ReadAbort

Enables or disables aborting the readback sequence during the readback sequence.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Enable, Disable
Default:	Disable

ReadCapture

Enables or disables readback of configuration bitstream.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Enable, Disable
Default:	Disable

ReadClk

Sets the readback clock to be CClk or to a user-supplied clock (from a net inside the FPGA that is connected to the 'i' pin of the RDCLK schematic block).

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	CClk (pin—see Note), Rdbk (user-supplied)
Default:	CClk

Notes:

In modes where CClk is an output, the pin is driven by the internal oscillator.

StartupClk

Selects a user-supplied clock or the internal CClk for controlling the post-configuration startup phase of the FPGA initialization

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	CClk (pin—see Note), UserClk (user-supplied)
Default:	CClk

Notes:

In modes where CClk is an output, the pin is driven by the internal oscillator.

SyncToDone

Synchronizes the I/O startup sequence to the external DoneIn signal.

Architectures:	XC4000E/L, XC4000EX/XL/XLA/XV, Spartan, SpartanXL
Settings:	Yes, No
Default:	No

TDO Pin

Adds a pull-up, a pull-down, or neither to the TDO pin (Test Data Out for Boundary Scan). Selecting one option enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	XC4000E/L, XC4000EX/XL/XLA/XV, Spartan, SpartanXL
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullnone

-g (Set Configuration—XC5200 Devices)

The -g option has sub-options that represent settings you use to set the configuration for an XC5200 design. These options have the following syntax.

```
bitgen -g option:setting
```

For example, to enable Cyclic Redundancy Checking (CRC), use the following syntax.

```
bitgen -g crc:enable
```

The following sections describe the startup sequences for the -g option.

BSReconfig

Enable or disable reconfiguration via boundary scan.

Architectures:	XC5200
Settings:	Disable, Enable
Default:	Disable

BSReadback

Enable or disable reading back configuration data via boundary scan.

Architectures:	XC5200
Settings:	Disable, Enable
Default:	Disable

ConfigRate

Selects the configuration clock rate. There are three choices: slow, med, and fast. Slow is equivalent to .75 MHz, med is equivalent to 6 MHz, and fast is equivalent to 12 MHz (nominal).

Architectures:	XC5200
Settings:	Slow, Med, Fast
Default:	Slow

CRC

Enables or disables Cyclic Redundancy Checking (CRC) on a chip-by-chip basis during configuration.

Architectures:	XC5200
Settings:	Enable, Disable
Default:	Enable

Input

This option sets the FPGA design input-signal thresholds to TTL or CMOS level for interface capability. CMOS improves noise immunity and reduces static power consumption.

The special-purpose clock inputs, TCLKIN, BCLKIN, and $\overline{\text{PWRDN}}$ always require CMOS-level signals, even if the FPGA design input thresholds are specified as TTL compatible.

Architectures:	XC5200
Settings:	TTL, CMOS
Default:	TTL

DoneActive

Selects the event that activates the FPGA Done signal. There are a maximum of four events that you can select from at one time. These events are CClk edges or external (user) clock edges.
The actual options available at any time depend on the selections made for StartupClk and SyncToDone.

Architectures:	XC5200

Settings: **C1** — first-CClk rising edge after the length count is met.

C2 — second-CClk rising edge after the length count is met.

C3 — third-CClk rising edge after the length count is met.

C4 — fourth-CClk rising edge after the length count is met.

U2 — second-valid-user-clock rising edge after C1.

U3 — third-valid-user-clock rising edge after C1.

U4 — fourth-valid-user-clock rising edge after C1.

Default: C1

Valid settings for DoneActive are as follows:

StartupClk	SyncToDone	DoneActive
CClk	Yes	C1, C2 or C3
CClk	No	C1, C2, C3, or C4
UserClk	Yes	C1 or U2
UserClk	No	C1, U2, U3, or U4

DonePin

Enables or disables internal pull-up on the DONE pin. The Pullnone setting indicates there is no connection to the pull-up.

Architectures: XC5200

Settings: Pullup, Pullnone

Default: Pullup

GSRInactive

Selects the event that releases the internal set-reset to the latches and flip-flops. You can select one of nine events: a CClk edge, an external (user) clock edge, or the external signal DoneIn.

Only some of these events become options at one time depending on the combination of StartupClk and SyncToDone selected.

Architectures: XC5200

Settings:

C2 — second-CClk rising edge after the length count is met.

C3 — third-CClk rising edge after the length count is met.

C4 — fourth-CClk rising edge after the length count is met.

U2 — second-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met).

U3 — third-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met).

U4 — fourth-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met).

DI — when the DoneIn signal goes High

DI_PLUS_1 — first CClk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High

DI_PLUS_2 — second CClk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High

Default: C3

Valid settings for GSRInactive are as follows:

StartupClk	SyncToDone	GSRInactive
CClk	Yes	C2, C3, DI, or DI_PLUS_1
CClk	No	C2, C3, or C4
UserClk	Yes	U2, DI, DI_PLUS_1, or DI_PLUS_2
UserClk	No	U2, U3, or U4

Input

Sets the input threshold level for IOBs.

Architectures: XC5200

Settings: TTL, CMOS

Default: TTL

LC_Alignment

The LC_Alignment option determines how length count is calculated to control when the device changes from configuration to user operation. The two methods of calculating length count, DONE Alignment and Length Count Alignment, are discussed in the

Configuration section of *The Programmable Logic Data Book*. The *FPGA Configuration Guidelines* Application Note also contains length count information.

Architectures:	XC5200
Settings:	Length, DONE
Default:	Length

OscClk

Determines whether the XC5200 oscillator is driven by the internal 16-MHz clock (CClk setting) or by a user clock (UserClk setting). If you specify UserClk, the clock must be connected to the OSC.CK pin of the device's OSC component.

Architectures:	XC5200
Settings:	UserClk, CClk
Default:	CClk

Outputs Active

Selects the event that releases the I/O from 3-state condition and turns the configuration related pins operational. There are a maximum of four events that you can select from at one time. These events are selected from a group of CClk edges, a group of external (user) clock edges, and the external signal DoneIn.
The actual options available at any time depend on the selections made for StartupClk and SyncToDone.

Architectures: XC5200

Settings:

C2 — second-CClk rising edge after the length count is met.

C3 — third-CClk rising edge after the length count is met.

C4 — fourth-CClk rising edge after the length count is met.

U2 — second-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met).

U3 — third-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met).

U4 — fourth-valid-user-clock rising edge after C1 (first-CClk rising edge after length count is met).

DI — when the DoneIn signal goes High

DI_PLUS_1 — first CClk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High

DI_PLUS_2 — second CClk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High

Default: C2

Valid settings for OutputsActive are as follows:

StartupClk	SyncToDone	OutputsActive
CClk	Yes	C2, C3, DI, or DI_PLUS_1
CClk	No	C2, C3, or C4
UserClk	Yes	U2, DI, DI_PLUS_1, or DI_PLUS_2
UserClk	No	U2, U3, or U4

ProgPin

Enables or disables internal pull-up on the $\overline{\text{PROGRAM}}$ pin. The pull-up affects the pin after configuration. The Pullnone setting indicates there is no connection to the pull-up.

Architectures: XC5200

Settings: Pullup, Pullnone

Default: Pullup

ReadAbort

Enables or disables aborting the readback sequence during the readback sequence.

Architectures: XC5200

Settings: Enable, Disable

Default: Disable

ReadCapture

Enables or disables readback of configuration bitstream.

Architectures:	XC5200
Settings:	Enable, Disable
Default:	Disable

ReadClk

Sets the readback clock to be CClk or to a user-supplied clock (from a net inside the FPGA that is connected to the 'i' pin of the RDCLK schematic block).

Architectures:	XC5200
Settings:	CClk (pin—see Note), Rdbk (user-supplied)
Default:	CClk

Notes:

In modes where CClk is an output, the pin is driven by the internal oscillator.

StartupClk

Selects a user-supplied clock or the internal CClk for controlling the post-configuration startup phase of the FPGA initialization.

Architectures:	XC5200
Settings:	CClk (pin—see Note), UserClk (user-supplied)
Default:	CClk

Notes:

In modes where CClk is an output, the pin is driven by the internal oscillator.

SyncToDone

Synchronizes the I/O startup sequence to the external DoneIn signal.

Architectures:	XC5200
Settings:	Yes, No
Default:	No

-g (Set Configuration—Virtex and Spartan2 Devices)

The -g option has sub-options that represent settings you use to set the configuration for a Virtex or Spartan2 design. These options have the following syntax.

```
bitgen -g option:setting
```

For example, to enable Readback, use the following syntax.

```
bitgen -g Readback
```

The following sections describe startup sequences for the -g option.

ReadBack

This option allows you to perform Readback by the creating the necessary bitstream.

ConfigRate

Virtex and Spartan2 use an internal oscillator to generate the configuration clock, CClk, when configuring in a master mode. Use the configuration rate option to select the rate for this clock.

Architectures:	Virtex, Spartan2
Settings:	To find out settings, enter **bitgen -h virtex**. Values are in MHz. The default is 4.
Default:	The default is the first item listed with **bitgen -h virtex** command.

Gclkdel0, Gclkdel1, Gclkdel2, Gclkdel3

Use these options to add delays to the global clocks. *You should not use this option unless instructed by Xilinx.*

Architectures:	Virtex, Spartan2
Settings:	11111, *binary string*
Default:	11111

StartupClk

The startup sequence following the configuration of a device can be synchronized to either CClk, a User Clock, or the JTAG Clock. The default is CClk.

- CClk

  Enter CClk to synchronize to an internal clock provided in the FPGA device.
- UserClk

  Enter UserClk to synchronize to a user-defined signal connected to the CLK pin of the STARTUP symbol.
- JTAG Clock

  Enter JtagClk to synchronize to the clock provided by JTAG. This clock sequences the TAP controller which provides the control logic for JTAG.

Architectures:	Virtex, Spartan2
Settings:	CClk (pin—see Note), UserClk (user-supplied), JtagCLK
Default:	CClk

Notes:

In modes where CClk is an output, the pin is driven by an internal oscillator.

PowerupClk

Selects which clock to synchronize to at the end of power up.

Architectures:	Virtex, Spartan2

Settings:	IntOsc, UserClk, CClk
Default:	IntOsc

CClkPin

Adds an internal pull-up to the CClk pin. The Pullnone setting disables the pullup.

Architectures:	Virtex, Spartan2
Settings:	Pullnone, Pullup
Default:	Pullup

DonePin

Adds an internal pull-up to the DonePin pin. The Pullnone setting disables the pullup.

Use this option only if you are planning to connect an external pull-up resistor to this pin. The internal pull-up resistor is automatically connected if you do not use this option.

Architectures:	Virtex, Spartan2
Settings:	Pullup, Pullnone
Default:	Pullup

DrivePDStatusPin

Enables this output power-down status pin.

Architectures:	Spartan2 only
Settings:	Yes, No
Default:	Yes

M0 Pin

The M0 pin is used to determine the configuration mode. Adds an internal pull-up, pull-down or neither to the M0 pin. The following settings are available. The default is PullUp. Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M0 pin.

Architectures:	Virtex, Spartan2
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

M1 Pin

The M1 pin is used to determine the configuration mode. Adds an internal pull-up, pull-down or neither to the M1 pin. The following settings are available. The default is PullUp.

Select **Pullnone** to disable both the pull-up resistor and pull-down resistor on the M1 pin.

Architectures:	Virtex, Spartan2
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

M2 Pin

The M2 pin is used to determine the configuration mode. Adds an internal pull-up, pull-down or neither to the M2 pin. The default is PullUp. Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M2 pin.

Architectures:	Virtex, Spartan2
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

PDStatus Pin

Adds an internal pull-up to the PDStatusPin pin. The Pullnone setting disables the pullup.

Use this option only if you are planning to connect an external pull-up resistor to this pin.

Architectures:	Spartan2 only
Settings:	Pullup, Pullnone
Default:	Pullnone

PowerdownPin

Adds an internal pull-up to the input PowerdownPin pin. The Pullnone setting disables the pullup.

Use this option only if you are planning to connect an external pull-up resistor to this pin.

Architectures:	Spartan2 only
Settings:	Pullup, Pullnone
Default:	Pullnone

ProgPin

Adds an internal pull-up to the ProgPin pin. The Pullnone setting disables the pullup. The pull-up affects the pin after configuration.

Architectures:	Virtex, Spartan2
Settings:	Pullup, Pullnone
Default:	Pullnone

TCK Pin

Adds a pull-up, a pull-down or neither to the TCK pin, the JTAG test clock. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex, Spartan2
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

TDI Pin

Adds a pull-up, a pull-down, or neither to the TDI pin, the serial data input to all JTAG instructions and JTAG registers. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex, Spartan2
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

TDO Pin

Adds a pull-up, a pull-down, or neither to the TdoPin pin, the serial data output for all JTAG instruction and data registers. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex, Spartan2
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

TMS Pin

Adds a pull-up, pull-down, or neither to the TMS pin, the mode input signal to the TAP controller. The TAP controller provides the control logic for JTAG. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down

Architectures:	Virtex, Spartan2
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

GSR Cycle

Selects the Startup phase that releases the internal set-reset to the latches, flip-flops, and BRAM output latches. The Done setting releases GSR when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes

Architectures:	Virtex, Spartan2
Settings:	Done, 1, 2, 3, 4, 5, 6, Keep
Default:	6

Keep should only be used when partial reconfiguration is going to be implemented. Keep prevents the configuration state machine from asserting control signals that could cause the loss of data.

GWE Cycle

Selects the Startup phase that asserts the internal write enable to flip-flops, LUT RAMs, and shift registers. It also enables the BRAMs. Before the Startup phase both BRAM writing

and reading are disabled.The Done setting asserts GWE when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes. The Keep setting is used to keep the current value of the GWE signal

Architectures:	Virtex, Spartan2
Settings:	Done, 1, 2, 3, 4, 5, 6, Keep
Default:	6

GTS Cycle

Selects the Startup phase that releases the internal tristate control to the IO buffers. The Done setting releases GTS when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes

Architectures:	Virtex, Spartan2
Settings:	Done, 1, 2, 3, 4, 5, 6, Keep
Default:	5

LCK Cycle

Selects the Startup phase to wait until DLLs lock. If NoWait is selected, the Startup sequence does not wait for DLLs.

Architectures:	Virtex, Spartan2
Settings:	0,1, 2, 3, 4, 5, 6, NoWait
Default:	NoWait

DONE Cycle

Selects the Startup phase that activates the FPGA Done signal. Done is delayed when DonePipe=Yes.

Architectures:	Virtex, Spartan2
Settings:	1, 2, 3, 4, 5, 6
Default:	4

Persist

This option is needed for Readback and Partial Reconfiguration using the configuration pins. It determines the data bus width and which IOBs are always in Configuration mode. These IOBs will be excluded from general use. The X* setting must be selected to enable Readback.

Architectures:	Virtex, Spartan2
Settings:	No, X1, X8
Default:	No

Select X1 for serial modes or X8 for Super8 mode. The following table lists pins that are persistent in serial and Super8 configurations:

Serial Modes	Super8 Mode
CFG_RDY ($\overline{\text{INIT}}$) (I/O)	CFG_RDY ($\overline{\text{INIT}}$) (I/O)
DOUT (O)	BUSY (O)
DIN (I)	DATA 0 (I/O)
	DATA 1(I/O)
	DATA 2(I/O)
	DATA 3(I/O)
	DATA 4(I/O)
	DATA 5(I/O)
	DATA 6(I/O)
	DATA 7(I/O)
	$\overline{\text{CS}}$ (I)
	RD$\overline{\text{WR}}$ (I)

DriveDone

This option actively drives CFG_DONE (Done) high as opposed to using pullup.

Architectures:	Virtex, Spartan2
Settings:	No, Yes
Default:	No

DonePipe

This option is intended for use with FPGAs being set up in a high-speed daisy chain configuration.When set to Yes, the FPGA waits on the CFG_DONE (DONE) pin to go High and then waits for the first clock edge before moving to the Done state.

Architectures:	Virtex, Spartan2
Settings:	No, Yes
Default:	No

Security

Selecting Level1 disables Readback. Selecting Level2 disables Readback and Partial Reconfiguration.

Architectures:	Virtex, Spartan2
Settings:	None, level1, Level2
Default:	None

UserID

You can enter up to an 8-digit hexadecimal code in the User ID register. You can use the register to identify implementation revisions.

-h or -help (Command Usage)

-h *architecture*

Displays a usage message for BitGen. The usage message displays all available options for BitGen operating on the specified *architecture*.

-j (No BIT File)

Do not create a bitstream file (.bit file). This option is generally used when you want to generate a report without producing a bitstream. For example, if you wanted to run DRC without producing a bitstream file, you would use the -j option.

Notes:

The .msk or .rbt files may still be created.

-l (Create a Logic Allocation File)

This option creates an ASCII logic allocation file (*design*.**ll**) for the selected design. The logic allocation file indicates the bitstream position of latches, flip-flops, and IOB inputs and outputs.

In some applications, you may want to observe the contents of the FPGA internal registers at different times. The file created by the -l option helps you identify which bits in the current bitstream represent outputs of flip-flops and latches. Bits are referenced by frame and bit number within the frame.

The Hardware Debugger uses the **design.ll** file to locate signal values inside a readback bitstream.

-m (Generate a Mask File)

Creates a mask file. This file is used to compare relevant bit locations for executing a readback of configuration data contained in an operating FPGA.

-n (Save a Tied design)

This command is used with the -t option (described below) to save the tied NCD file as *_file_name*.ncd (note the underscore in front of the file name). The tied design file is placed in the same directory as the output file. It has the same root name as the output file with an .ncd extension. If you do not specify an output file, the tied design file is placed in the input file's directory and is named *_file_name*.ncd, where *_file_name* is the root name of the input file. Use TRACE to run timing analysis on the tied design. You can also use the FPGA Editor to check the effects of the tiedown. This option is not supported for Virtex or Spartan2.

-t (Tie Unused Interconnect)

This option causes all unused interconnect to be tied to a logic low or to a known level, keeping internal noise and power consumption to a minimum. When you use the -t option, DRC runs first (before tiedown). BitGen terminates if any DRC error occurs. A DRC warning does not cause the bitstream generation program to abort, but it may cause tiedown to fail.

After DRC, the -t option does the following.

- Ties all possible unused interconnect to tie sites or unused CLB outputs and configures those outputs with a logic low (F=0 or G=0)

- Attempts to tie any remaining interconnect to CLB outputs which have not been designated as critical
- Attempts to tie remaining interconnect to the global or to the auxiliary clock buffer outputs if unused (only in conjunction with the -a option)

The only condition under which tie will add interconnect to a "critical" net is if you use the -u option (allowing interconnect to be added to critical nets as a "last resort"). A "critical" net is one with a priority greater than 3.

The -t option does not add an XC4000 or XC5200 tristate buffer input (I) pin or tristate (T) pin to a net.

When you add interconnect to used CLB or buffer outputs, delays may be added on any net to which the outputs are connected. To prevent the added delay, assign the net a priority greater than 3. You can do this through the physical constraints file or through the FPGA Editor. See the PRIORITIZE physical constraint in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*. Note that flagging too many nets as critical could cause the tiedown to fail. When an interconnect is tied to a user-defined net, you get a message giving the number of nodes added to the net. Delay characteristics for the net associated with that source may change. (Only in conjunction with the -a option)

When certain pins cannot be tied, you receive a warning message supplying information about the design's untied interconnect.

To remove the obstacles that have caused tiedown to fail, look carefully at nets close to an untied PIP. An input pin could have multiple input PIPs, and all of them could source the pin. If each of these PIPs is associated with a critical net, they are not used, and the input pin is left untied. To correct the problem, make one of the nets "non-critical." Do this by removing the PRIORITIZE constraint from the net in the PCF file or in the FPGA Editor. Then run TRACE (the timing analysis program) and evaluate any delay that might have been added to the net. (Only in conjunction with the -a option)

If you use the -n option, the tied design is saved in a file *_file_name*.ncd (note the underscore before the file name). You can load the file into the FPGA Editor and examine the results of tiedown. You can look at all of the original nets that have been affected by tiedown and the net delays before and after tiedown.

Like unused internal interconnect, unused external I/O pins on the chip must also have defined signal levels, that is, they must not be in a floating condition. In XC4000E/EX FPGAs, unused IOBs are automatically pulled HIGH with pull-up resistors.

Partial tiedown is the new default. Tiedown will print the number of untied nodes and then continue. See the -a option also. Partial tiedown *never* ties to user signals.

This option is not supported for Virtex or Spartan2.

-u (Use Critical Nets Last)

Because of possible added delay, tiedown does not add interconnect to any net that has been assigned a priority greater than 3. This option allows interconnect to be added to critical nets as a "last resort."

This option is not supported for Virtex or Spartan2.

-w (Overwrite Existing Output File)

Enables you to overwrite an existing BIT, LL, MSK, or RBT output file.

PROMGen

The PROMGen program is compatible with the following families.

- XC3000A/L
- XC3100A/L

- XC4000E / L
- XC4000EX / XL / XV / XLA
- XC5200
- Spartan / XL / 2
- Virtex / E / 2

PROMGen formats a BitGen-generated configuration bitstream (BIT) file into a PROM format file. The PROM file contains configuration data for the FPGA device. PROMGen converts a BIT file into one of three PROM formats: MCS-86 (Intel), EXORMAX (Motorola), or TEKHEX (Tektronix). It can also generate a Hex file format.

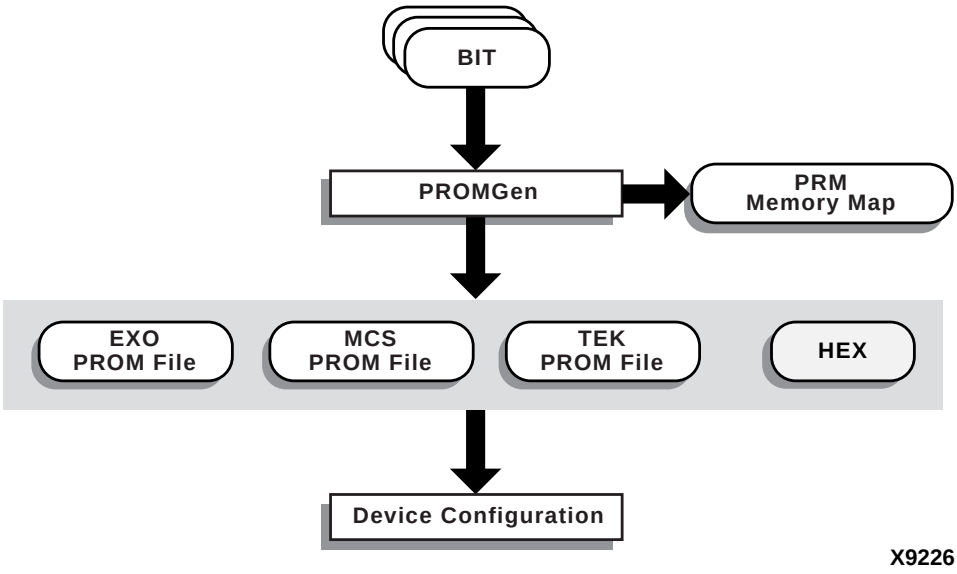

Figure 92: **PROMGen**

There are two functionally equivalent versions of PROMGen. There is a stand-alone version you can access from an operating system prompt. You can also access an interactive version, called the PROM File Formatter, from inside the Design Manager for Alliance or the Project Manager in Foundation. This chapter describes the stand-alone version; the interactive version is described in the *PROM File Formatter Guide*.

You can also use PROMGen to concatenate bitstream files to daisy-chain FPGAs.

Notes:

If the destination PROM is one of the Xilinx Serial PROMs, you are using a Xilinx PROM Programmer, and the FPGAs are not being daisy-chained, it is not necessary to make a PROM file. See the *Hardware User Guide* for more information about daisy-chained designs

PROMGen Syntax

Use the following syntax to start PROMGen from the operating system prompt:

```
promgen [options]
```

Options can be any number of the options listed in"PROMGen Options" on page 239. Separate multiple options with spaces.

PROMGen Files

This section describes the PROMGen input and output files.

Input Files

The input to PROMGEN consists of BIT files— one or more bitstream files. BIT files contain configuration data for an FPGA design.

Output Files

Output from PROMGEN consists of the following files.

- PROM files—The file or files containing the PROM configuration information. Depending on the PROM file format your PROM programmer uses, you can output a TEK, MCS, or EXO file. If you are using a microprocessor to configure your devices, you can output a HEX file, which contains a hexadecimal representation of the bitstream.
- PRM file—The PRM file is a PROM image file. It contains a memory map of the output PROM file. The file has a .prm extension.

Bit Swapping in PROM Files

PROMGen produces a PROM file in which the bits within a byte are swapped compared to the bits in the input BIT file. Bit swapping (also called “bit mirroring”) reverses the bits within each byte, as shown in the following figure.

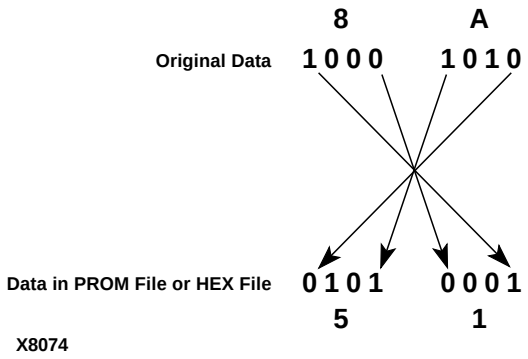

Figure 93: **Bit Swapping**

In a bitstream contained in a BIT file, the Least Significant Bit (LSB) is always on the left side of a byte. But when a PROM programmer or a microprocessor reads a data byte, it identifies the LSB on the right side of the byte. In order for the PROM programmer or microprocessor to read the bitstream correctly, the bits in each byte must first be swapped so they are read in the correct order.

In this release of the Xilinx Development System, the bits are swapped for all of the PROM formats: MCS, EXO, and TEK. For a HEX file output, bit swapping is on by default, but it can be turned off by entering a -b PROMGen option that is available only for HEX file format.

PROMGen Options

This section describes the options that are available for the PROMGen command.

-b (Disable Bit Swapping—HEX Format Only)

This option only applies if the -p option specifies a HEX file for the output of PROMGen. By default (no -b option), bits in the HEX file are swapped compared to bits in the input BIT files. If you enter a -b option, the bits are not swapped. Bit swapping is described in "Bit Swapping in PROM Files" on page 239.

-c (Checksum)

```
promgen -c
```

The -c option generates a checksum value appearing in the .prm file. This value should match the checksum in the prom programmer. Use this option to verify that correct data was programmed into the prom.

-d (Load Downward)

```
promgen -d hexaddress0 filename filename...
```

This option loads one or more BIT files from the starting address in a downward direction. Specifying several files after this option causes the files to be concatenated in a daisy chain. You can specify multiple -d options to load files at different addresses. You must specify this option immediately before the input bitstream file.

Here is the multiple file syntax.

```
promgen -d hexaddress0 filename filename...
```

Here is the multiple -d options syntax.

```
promgen -d hexaddress1 filename -d hexaddress2 filename...
```

-f (Execute Commands File)

```
-f command_file
```

The -f option executes the command line arguments in the specified *command_file*.

-help (Command Help)

This option displays help that describes the PROMGen options.

-l option (Disable Length Count)

```
promgen -l
```

The -l option disables the length counter in the FPGA bitstream. It is valid only for 4000EX, 4000XL, 4000XLA, 4000XV, and SpartanXL Devices. Use this option when chaining together bitstreams exceeding the 24 bit limit imposed by the length counter.

-n (Add BIT FIles)

```
-n file1[.bit] file2[.bit]...
```

This option loads one or more BIT files up or down from the next available address following the previous load. The first -n option *must* follow a -u or -d option because -n does not establish a direction. Files specified with this option are not daisy-chained to previous files. Files are loaded in the direction established by the nearest prior -u, -d, or -n option.

The following syntax shows how to specify multiple files. When you specify multiple files, PROMGen daisy-chains the files.

```
promgen -d hexaddress file0 -n file1 file2...
```

The following syntax when using multiple -n options prevents the files from being daisy-chained:

```
promgen -d hexaddress file0 -n file1 -n file2...
```

-o (Output File Name)

```
-o file1[.ext] file2[.ext]...
```

This option specifies the output file name of a PROM if it is different from the default. If you do not specify an output file name, the PROM file has the same name as the first BIT file loaded.

ext is the extension for the applicable PROM format.

Multiple file names may be specified to split the information into multiple files. If only one name is supplied for split PROM files (by you or by default), the output PROM files are named *file_#*.ext, where *file* is the base name, # is 0, 1, etc., and *ext* is the extension for the applicable PROM format.

```
promgen -d hexaddress file0 -o filename
```

-p (PROM Format)

```
-p {mcs | exo | tek | hex}
```

This option sets the PROM format to one of the following: MCS (Intel MCS86), EXO (Motorola EXORMAX), TEK (Tektronix TEKHEX). The option may also produce a HEX file, which is a hexadecimal representation of the configuration bitstream used for microprocessor downloads. If specified, the -p option must precede any -u, -d, or -n options. The default format is MCS.

-r (Load PROM File)

```
-r promfile
```

This option reads an existing PROM file as input instead of a BIT file. All of the PROMGen output options may be used, so the -r option can be used for splitting an existing PROM file into multiple PROM files or for converting an existing PROM file to another format.

-s (PROM Size)

```
-s promsize1 promsize2...
```

This option sets the PROM size in kilobytes. The PROM size must be a power of 2. The default value is 64 kilobytes. The -s option must precede any -u, -d, or -n options.

Multiple *promsize* entries for the -s option indicates the PROM will be split into multiple PROM files.

Notes:

PROMGen PROM sizes are specified in bytes. *The Programmable Logic Data Book* specifies PROM sizes in bits for Xilinx serial PROMs (see -x option).

-u (Load Upward)

```
-u hexaddress0 filename1 filename2...
```

This option loads one or more BIT files from the starting address in an upward direction. When you specify several files after this option, PROMGen concatenates the files in a daisy chain. You can load files at different addresses by specifying multiple -u options.

This option must be specified immediately before the input bitstream file.

-x (Specify Xilinx PROM)

```
-x xilinx_prom1 xilinx_prom2...
```

The -x option specifies one or more Xilinx serial PROMs for which the PROM files are targeted. Use this option instead of the -s option if you know the Xilinx PROMs to use.

Multiple *xilinx_prom* entries for the -x option indicates the PROM will be split into multiple PROM files.

Examples

To load the file test.bit up from address 0x0000 in MCS format, enter the following information at the command line.

```
promgen -u 0 test
```

To daisy-chain the files test1.bit and test2.bit up from address 0x0000 and the files test3.bit and test4.bit from address 0x4000 while using a 32K PROM and the Motorola EXORmax format, enter the following information at the command line.

```
promgen -s 32 -p exo -u 00 test1 test2 -u 4000 test3 test4
```

To load the file test.bit into the PROM programmer in a downward direction starting at address 0x400, using a Xilinx XC1718D PROM, enter the following information at the command line.

```
promgen -x xc1718d -d 0x400 test
```

To specify a PROM file name that is different from the default file name enter the following information at the command line.

```
promgen options filename -o newfilename
```

PROM Usage

Table 121: **Prom X-Table**

Device	Configuration Bits	XC18XX Family	XC17XX Family	XC17VXX Family
XCV50	559,200	XC18V01	XC1701L	
XCV100	781,216	XC18V01	XC1701L	
XCV150	1,040,096	XC18V01	XC1701L	
XCV200	1,335,840	XC18V02	XC1702L	
XCV300	1,751,808	XC18V02	XC1702L	
XCV400	2,546,048	XC18V04	XC1704L	
XCV600	3,607,968	XC18V04	XC1704L	
XCV800	4,715,616	XC18V04 + XC18V512	XC1704L + XC1701L	XC17V08
XCV1000	6,127,744	XC18V04 + XC18V02	XC1704L + XC1701L	XC17V08
XCV50E	630,048	XC18V01	XC1701L	

Table 121: **Prom X-Table**

Device	Configuration Bits	XC18XX Family	XC17XX Family	XC17VXX Family
XCV100E	863,840	XC18V01	XC1701L	
XCV200E	1,442,106	XC18V02	XC1702L	
XCV300E	1,875,648	XC18V02	XC1702L	
XCV400E	2,693,440	XC18V04	XC1704L	
XCV405E	3,340,400	XC18V04	XC1704L	
XCV600E	3,961,632	XC18V04	XC1704L	
XCV812E	6,519,648	2 of XC18V04	2 of XC1704L	XC17V08
XCV1000E	6,587,520	2 of XC18V04	2 of XC1704L	XC17V08
XCV1600E	8,308,992	2 of XC18V04	2 of XC1704L	XC17V16
XCV2000E	10,159,648	3 of XC18V04	3 of XC1704L	XC17V16
XCV2600E	12,922,336	4 of XC18V04	4 of XC1704L	XC17V16
XCV3200E	16,283,712	4 of XC18V04	4 of XC1704L	XC17V16

Table 122: **Number of Virtex-E FPGAs Configurable with one XC17VXX PROM**

	XC17V16	XC17V08
XCV50E	26	13
XCV100E	18	9
XCV200E	11	5
XCV300E	8	4
XCV400E	6	3
XCV405E	4	2
XCV600E	4	2
XCV812E	2	1
XCV1000E	2	1
XCV1600E	2	
XCV2600E	1	
XCV3200E	1	

Glossary

Bitstream

The bitstream is a binary representation of an implemented FPGA design. The bitstream is generated by the Xilinx bit generation tools (BitGen and Makebits) and is denoted with the extension <.bit>. For information on creating BIT files, refer to the Hardware Debugger Reference / User Guide.

Block SelectRAM Resource

One of several large, fully-synchronous, dual-port memories in the Virtex FPGAs. Each of these memories contain 4096 bits. The organization of each memory is configurable. The block SelectRAM resource complements the smaller, distributed, LUT SelectRAMs.

Boundary Scan Interface

One of the configuration interfaces on the Virtex device. This is a bit-serial interface. The Boundary Scan interface is also known as the JTAG port. Also see the SelectMAP interface.

Capture Data

The flip-flop and pad data saved from the logic cells and I/O blocks into the bitstream for Readback. Use the CAPTURE_VIRTEX primitive in your HDL code to specify the trigger and clock for the capture operation.

Command Register (CMD)

Configurable Logic Block (CLB)

The functional elements for constructing logic circuits. The Virtex CLB is made up of Slices, which contain Logic Cells.

Configuration

The process of programming Xilinx SRAM-based FPGAs with a bitstream.

Configuration Bitstream

Configuration commands, optionally with configuration data.

Configuration Commands

Instructions for the Virtex device. There are two classes of Configuration Command — Major and Minor. The Major Commands read and write data to configuration registers in

the Virtex device. The Minor commands instruct the Virtex configuration logic to perform specific functions. See "Command Register (CMD)"

Configuration Data

Bits that directly define the state of programmable logic. These are written to a Virtex device in a configuration bitstream, and read as Readback Data from a Virtex device.

Configuration Frame

The configuration bits in a Virtex device are organized in columns. A column of CLBs with the I/O blocks above and below the CLBs contain 48 frames of configuration bits. The smallest number of bits that can be read or written through the configuration interfaces is one frame.

Configuration Interface

A logical interface on the Virtex device through which configuration commands and data can be read and written. A interface consists of one or more physical Device Pins.

Configuration Readback

The operation of reading Configuration Data (also known as Readback Data) from a Virtex device.

CCLK

During configuration, the Configuration Clock (CCLK) is an output in Master modes or in the Asynchronous Peripheral mode but is an input in Slave, Synchronous Peripheral, Express, and SelectMAP/Slave Serial modes. After configuration, CCLK has a weak pull-up and can be selected as the Readback Clock.

$\overline{CS}$ Pin

The $\overline{CS}$ Pin is the Chip Enable Pin for Virtex/Spartan-II devices. It is only used in the SelectMAP mode. When $\overline{CS}$ is asserted (Low) the device examines the data on the Data bus. When $\overline{CS}$ is de-asserted (High), all CCLK transitions are ignored.

DataFrame

A DataFrame is a block of configuration data. A configuration bit-stream contains many such frames, each with a start bit and stop bits.

Device Pin

One of the electrical connections on the package containing the Virtex device.

DIN

During serial configuration, the DIN Pin is the serial configuration data input receiving data on the rising edge of CCLK. During parallel and peripheral configuration, DIN is the D0 input. After configuration, DIN is a user-programmable I/O Pin.

DONE Pin

The DONE Pin on a Xilinx FPGA is a bidirectional signal with an optional internal pull-up resistor. As an output, it indicates the completion of the configuration process. As an input, a low level on DONE can be configured to delay the global logic initialization and the enabling of outputs.

DOUT Pin

During configuration in any mode except Express and SelectMAP, the DOUT Pin is the serial configuration data output that can drive the DIN Pin of daisy-chained slave FPGAs. DOUT data changes on the falling edge of CCLK, one-and-a-half CCLK periods after it is received at the DIN Pin (in Master Serial Mode only).

DOUT/BUSY Pin

For Virtex/Spartan-II devices, the DOUT/BUSY pin has a dual purpose, depending on device mode. When the device is in Serial mode, this pin functions as DOUT. When the device is in SelectMAP/Slave Parallel mode, this pin functions as a handshaking signal. If BUSY is asserted (High) on a rising edge of CCLK, the data is not seen on the data bus, and should be held until the data is accepted.

Frame

See Configuration Frame.

FPGA

Xilinx Field Programmable Gate Arrays (FPGAs) are SRAM- based Programmable Logic Devices (PLDs). FPGAs contain an array of programmable logic elements. The configuration data for the array is stored in an internal SRAM bank called the configuration memory. The configuration memory can be cleared on-the-fly with the $\overline{\text{PROGRAM}}$ pin or every time power is cycled.

Header

The Header is the first section of the Bitstream. This 40-bit section contains the 4-bit Preamble, a 24-bit LengthCount, and 12 dummy bits.

HDC Pin

High During Configuration (HDC) is driven High until the I/Os become active in the Startup sequence. It is available as a control output indicating that configuration is not yet complete. After configuration, HDC is a user-programmable I/O Pin.

$\overline{\text{INIT}}$ Pin

The $\overline{\text{INIT}}$ Pin is a quadruple function signal. Before and during configuration, $\overline{\text{INIT}}$ is a bidirectional signal. A 1 - 10 kΩ external pull-up resistor is recommended. As an active-Low open-drain output, $\overline{\text{INIT}}$ is held Low during power stabilization and internal clearing of the configuration memory. As an active-Low input, it can be used to hold the FPGA in the internal WAIT state before the start of configuration. During configuration, a Low on this output indicates that a configuration data error has occurred. After the I/O become active in the Startup sequence, $\overline{\text{INIT}}$ becomes a user-programmable I/O.

JTAG Cable

See Parallel Cable III.

LengthCount

The LengthCount is a 24-bit binary number embedded in the Header of the bitstream. During configuration, the LengthCount is captured and stored in the Lengthcount Register. Every configuration data bit (one per CCLK period) is counted by the LengthCount Counter. When the LengthCount Register and the LengthCount Counter are equal, LengthCount Match is achieved. LengthCount Match is a requirement for entering the Startup sequence.

Logic Cell (LC)

The basic building block of the Virtex CLB. An LC includes a 4-input function generator, carry logic, and a storage element

LDC Pin

Low During Configuration (LDC) is driven Low until the I/Os become active in the Startup sequence. It is available as a control output indicating that configuration isn't complete. After configuration, LDC is a user-programmable I/O Pin.

LUT SelectRAM

Shallow RAM structure implemented in CLB Lookup Tables (LUTs). See also Block SelectRAM Resource.

MultiLINX Cable

The MultiLINX Cable provides many complex functions and can be loaded with new firmware as it becomes available. It can be connected to the host computer in two ways: via a Serial port or a USB port. The MultiLINX cable is supported by the Hardware Debugger for Slave Serial and SelectMAP/Slave Parallel programming (as appropriate) as well as Readback/Verify. It is also supported by the JTAG programmer for JTAG programming of both CPLDs and FPGAs. For more information on using the MultiLINX Cable, refer to "" chapter in this guide and the *JTAG Programmer Guide.*

Pad

Pad bits are extra bits used to make the total number of bits in a frame an integral multiple of 32, the number of bits in a configuration word. A Pad Word is an extra word used at the end of a Configuration Frame for pipelining. A Pad Frame is an extra Configuration Frame used at the beginning of a Configuration Readback and at the end of a Configuration Write for pipelining.

Parallel Cable III

The Xilinx Parallel Cable III (model DLC5) is a serial download cable. The Parallel cable uses a serial 25-pin interface to the parallel port of a host computer and two 6-pin headers for flying-wire connectors to a target board. The Parallel cable is supported by the Hardware Debugger software for performing Slave Serial configuration of FPGAs only. The Parallel cable is also supported by the JTAG Programmer software for performing Slave Serial and Boundary Scan configuration of FPGAs, and Boundary Scan programming of CPLDs. For more information on using the Parallel cable, refer to Chapter 8 or this guide, the Hardware Debugger Reference/Users Guide, and the JTAG Programmer Guide.

Preamble

The Preamble is a 4-bit binary sentinel ("0010"b) used to indicate the beginning of the LengthCount in the Header portion of the bitstream. At the beginning of configuration, FPGAs ignore all data prior to the preamble but counts the number of data bits preceding the preamble, and the LengthCount counter increments for every rising CCLK edge, even the ones proceeding the preamble.

$\overline{\text{PROGRAM}}$ Pin

The $\overline{\text{PROGRAM}}$ pin is an active-Low input that forces clearing of the FPGA configuration memory and is used to initiate a configuration cycle. While $\overline{\text{PROGRAM}}$ is held Low, the FPGA drives $\overline{\text{INIT}}$ Low and continues to clear the configuration memory. When

$\overline{\text{PROGRAM}}$ goes High, the FPGA finishes the current clear cycle, executes another complete clear cycle, goes into a WAIT state, and releases $\overline{\text{INIT}}$.

Readback

Initiating a Readback causes the configuration memory to become accessible to be serially clocked out and read from the device, or (byte-wide in SelectMAP / Slave Parallel modes). The configuration memory contains the configuration data, facilitating a Read-Verification of the data. The configuration memory can also contain the CLB output logic states facilitating a Read-Capture of the internal logic states.Read-Verification and Read-Capture are used by the Hardware Debugger for Hardware Verification. For information on the Readback specification and timing, refer to The Programmable Logic Data Book. For information on using the Readback component in a design, refer to the Libraries Guide. For information on enabling the Readback function in the Implementation Software refer to the Development System Reference Guide. For information on using the Hardware Debugger refer to the Hardware Debugger Reference / User Guide. For information on connecting the XChecker Cable for Readback, refer to the Hardware Users Guide.

Readback Data

Configuration data read from a Virtex device. The data is organized as Configuration Frames.

SelectMAP Interface

One of the configuration interfaces on the Virtex device. This is a byte-serial interface. The pins in the SelectMAP interface may be used as user I / O after configuration has been completed or remain configured as a configuration interface.

Slice

A subdivision of the Virtex CLB. There are two vertical slices in each Virtex CLB. Each slice contains two Logic Cells.

Start-Up

The Start-up sequence completes a configuration cycle by releasing the DONE Pin, activating the I / Os, releasing Global Write Enable (GWE), and de-asserting the Global Set / Reset (GSR), which globally initializes all internal flip-flops. The sequence of these steps is user-programmable during bitstream generation. For the FPGA configuration cycle to enter the Startup sequence, two conditions must be satisfied; the configuration memory must be FULL and LengthCount Match must be achieved, in that order. For more information on the Startup sequence and FPGA configuration, refer to The Programmable Logic Data Book.

Sync Word

A 32-bit word with a value that is used to synchronize the configuration logic.

$\overline{\text{WRITE}}$ Pin

The $\overline{\text{WRITE}}$ Pin is an input to Virtex / Spartan-II devices in the SelectMAP / Slave Parallel mode, indicating to the device which direction data is flowing on the Data bus. When $\overline{\text{WRITE}}$ is asserted (Low), data is entering the device (configuration). When $\overline{\text{WRITE}}$ is de-asserted (High), data is leaving the device (readback). If $\overline{\text{WRITE}}$ changes state when the device isn't expecting it, an abort occurs. For more information on the $\overline{\text{WRITE}}$ pin, refer to The Programmable Logic Data Book and Chapter 2 of this guide.

XChecker Cable

The Xilinx XChecker Cable (model DLC4) is a serial download cable. The XChecker uses a serial 9-pin interface to the communication port of a host computer and two 8-pin headers for flying-wire connectors to a target board. The XChecker cable is supported by the Hardware Debugger software for performing Slave Serial configuration and Readback of FPGAs. The XChecker cable is also supported by the JTAG Programmer software for performing Slave Serial and Boundary Scan configuration of FPGAs, and Boundary Scan programming of CPLDs. For more information on using the XChecker cable refer to the Hardware Users Guide and the Hardware Debugger Reference / Users Guide.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X