Spartan-II FPGA Family Configuration and Readback

XAPP176 (v0.9) December 4, 1999 Advance Application Note

Summary

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

Introduction

Configuration is the process of loading a design bit-stream into FPGA internal configuration memory. Readback is the process of reading that data out.

While the Spartan-II configuration logic is significantly different from that of the Spartan/XL family; nevertheless, it maintains a great deal of compatibility to all Xilinx FPGA products.

Spartan-II vs. Spartan/XL Configuration

This section first presents the major differences in Spartan-II configuration from that of previous families. Whether you are an experienced FPGA user, or a first time user, the information in this section is important to review.

Configuration Modes and Daisy-Chains

Spartan-II FPGAs may be configured in eight different modes, shown in Table 1. There are four primary modes (Master Serial, Slave Serial, Slave Parallel, and Boundary-scan), each with the option to have the user I/Os pulled-up 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 9.

Table 1: Spartan-II Configuration Modes

Configuration Mode	M2	M1	M0	Pull-ups
Master Serial	0	0	0	No
Slave Serial	1	1	1	No
Slave Parallel	1	1	0	No
Boundary-scan	1	0	1	No
Master Serial (with pull-ups)	1	0	0	Yes
Slave Serial (with pull-ups)	0	1	1	Yes
Slave Parallel (with pull-ups)	0	1	0	Yes
Boundary-scan (with pull-ups)	0	0	1	Yes

Parallel Modes

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

Spartan-II devices do not have a Master Parallel mode. Users who prefer to store configuration data on parallel EPROMs should read the Xilinx Application Note XAPP178 "Configuring Spartan-II FPGAs from Parallel EPROMs".

Daisy-Chaining

Spartan-II FPGAs can be serially daisy-chained for configuration just as all previous Xilinx FPGAs, see "Master/Slave Serial Modes" on page 9. All devices in the chain must be in one of the serial modes. The Slave Parallel mode does not support any serial daisy-chaining. Multiple Spartan-II FPGAs can, however, be configured through the Slave Parallel interface in a parallel fashion, see "Slave Parallel Mode" on page 11. An example of this is also demonstrated in Application Note XAPP178 "Configuring Spartan-II FPGAs from Parallel EPROMs".

Boundary-Scan

The boundary-scan interface is always active from the moment of power-up, before, during, and after configuration. Boundary-scan modes select the optional pull-ups and prevent configuration in any other modes.

Configuring Spartan-II FPGAs through the boundary-scan interface is not described in this note. For more information on Spartan-II Boundary Scan, refer to Application Note XAPP139 "Virtex Configuration and Readback Through Boundary Scan", which also applies to the Spartan-II family.

Initialization and Timing

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

Mixed Voltage Environments

Spartan-II FPGAs have separate voltage sources for the internal core circuitry (V_{CCINT} = 2.5V) and the I/O circuitry (SelectI/O). SelectI/O is separated into eight banks of I/O groups. Each bank may be configuSelectI/Ored with one of multiple I/O standards. Refer to the Spartan-II data sheet 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 3.3V 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 3.3V supply for output configuration pins to operate normally. This is a requirement for Master Serial configuration and readback through the Slave Parallel ports.

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

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

For Slave Serial and Slave Parallel configuration modes, V_{CCO} may be driven at any voltage, or not at all, as long as 3.3V pull-up resistors are added on the DONE, INIT, and DOUT/BUSY pins. Additionally, V_{CCO_2} must be pulled to a value above 1.0V during power-up of the FPGA. If V_{CCO_2} is left floating, add a pull-up resistor between 1.5V and 3.3V to any V_{CCO_2} pin.

BitGen Switches and Options

This section describes new optional settings for bit-stream generation that pertain only to Spartan-II devices. The new BitGen options found in the configuration options template of the Xilinx development software are listed in Table 2.

Table 2: Spartan-II BitGen Options

Switch	Default Setting	Optional Setting
Readback	N/A	N/A
ConfigRate (MHz) (nominal)	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, KEEP
GSR_cycle	6	1, 2, 3, 4, 5, DONE, KEEP
GWE_cycle	6	1, 2, 3, 4, 5, DONE, KEEP
LCK_cycle	NoWait	0, 1, 2, 3, 4, 5, 6
Persist	No	Yes
DriveDONE	No	Yes
DonePipe	No	Yes
Security	None	Level1, Level2
UserID	0xFFFFFFFF	<hex string> (32-bit)
Gclkdel0	11111	<binary string>
Gclkdel1	11111	<binary string>
Gclkdel2	11111	<binary string>
Gclkdel3	11111	<binary string>

Readback

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

ConfigRate

The ConfigRate is the internally generated frequency of CCLK in Master Serial mode. The default frequency is4 MHz. At the start of configuration, CCLK toggles at 2.5 MHz, then changes to the selected (or default) frequency after the first 60 bytes of the bit-stream have been loaded. It should also be noted that the CCLK periods have a variance of +45%/–30% 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 "Virtex Configuration and Readback through Boundary-Scan", which also applies to the Spartan-II family.

DONE_cycle

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

GSR_cycle

The GSR_cycle specifies which cycle 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 releases the GSR on the first clock cycle after the DONE pin is externally released.

GWE_cycle

The GWE_cycle specifies which cycle in the Start-up Sequence releases the internal GlobalWriteEnable signal. This signal is not accessible to the user. It keeps all flip-flops, SelectRAM, and BlockRAM from changing state. However, the DLL is unaffected by the GWE. The DONE setting releases the GWE on the first clock cycle after the DONE pin is externally released.

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 releases the GTS on the first clock cycle after the DONE pin is externally released.

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* should be used whenever a DLL is not used in a design, or if a DLL is used, but 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, refer to Application Note XAPP174 "Using Delay-Locked Loops in the Spartan-II FPGAs".

Persist

If the Persist option is unspecified, or specified for its default setting of *No*, then all configuration pins other than CCLK, PROGRAM, and DONE, become user I/O after configuration. If set to "Yes", the Persist switch causes the configuration pins to retain their configuration function even after configuration. "Yes" must be selected if readback is to be performed through Slave Parallel. 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 go externally High before continuing. The rise time of DONE depends on its external capacitive loading.

The DonePipe option adds a pipeline register stage between the DONE pin and the start-up circuitry. This is needed at 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 Slave Parallel 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 deconfigure it by asserting PROGRAM or recycling power.

UserID

The UserID is a 32-bit data word accessible through the boundary-scan IDCODE command. The data word can be any arbitrary 32-bit value. To include a UserID in a bit-stream, 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 and should not be used unless instructed to do so.

CCLK and LengthCount

Previous Xilinx FPGA families used a LengthCount number that was embedded in their bit-stream. This lengthcount number indicates to the FPGA how many CCLK cycles should be observed before activating the Start-up Sequence that activates the FPGA. This method also required that the FPGA not receive any CCLK transitions prior to the loading of the bit-stream. Otherwise, this would throw off the lengthcount and the Start-up Sequence would not be activated at the appropriate time. Thus, free running oscillators could not be used to generate the CCLK for configuration.

Spartan-II FPGAs do not use any such lengthcount number in their configuration streams. The Start-up Sequence for these devices is controlled by a set of configuration commands that are embedded near the end of the configuration stream. See "Bit-stream Format" on page 15. Therefore, Spartan-II FPGAs may have a free running oscillator driving the CCLK pin.

Start-up Sequence

Start-up is the transition from the configuration state to the operational state. The Start-up Sequence activates an FPGA upon a 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. This activates all the I/Os.
3. Asserts GWE. This allows all RAMs and flip-flops to change state, although flip-flops cannot change state while GSR is asserted.
4. Negates GSR. This allows all flip-flops to change state.
5. Asserts EOS. The End-Of-Start-up flag is always set in phase 7. This is an internal flag that is not user accessible.

The order of the Start-up Sequence is controlled by BitGen options. The default Start-up Sequence is the bold line shown in Figure 1. The Start-up Sequence may also be stalled at any

phase until either the DONE has externally transitioned High, or a specified DLL has established LOCK. For details, see "BitGen Switches and Options" on page 3.

At the cycle selected for the DONE to be released, the sequencer always waits in that state until the DONE is externally released. This is similar to the "SyncToDONE" behavior in the Spartan/XL 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 one clock cycle after DONE externally transitions High.

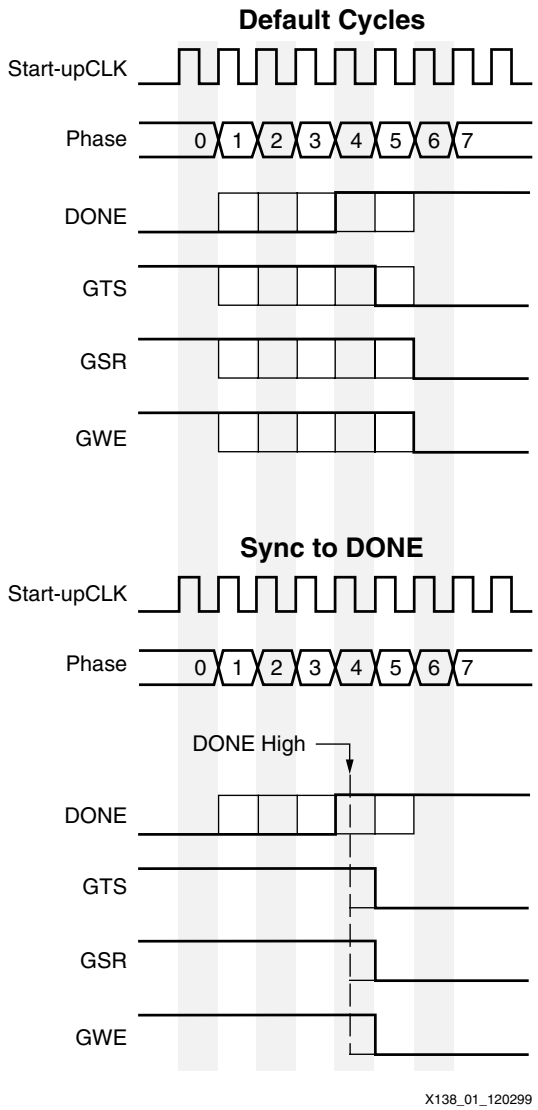

Figure 1: Default Start-up Sequence

Configuration Process and Flow

The external configuration process is simply a matter of loading the configuration stream 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 should be supplied with a 2.5V source. The rise time for the core voltage should be a maximum of 50 ms to rise from 1V to 2.4V. The IOB output voltage input for Bank 2 (V_{CCO_2}) is also used as a logic input to the Power-On-Reset (POR) circuitry. This value must be greater than 1.0V for Power-up to continue. If this bank is not being used, a pull-up should be added to V_{CCO_2}.

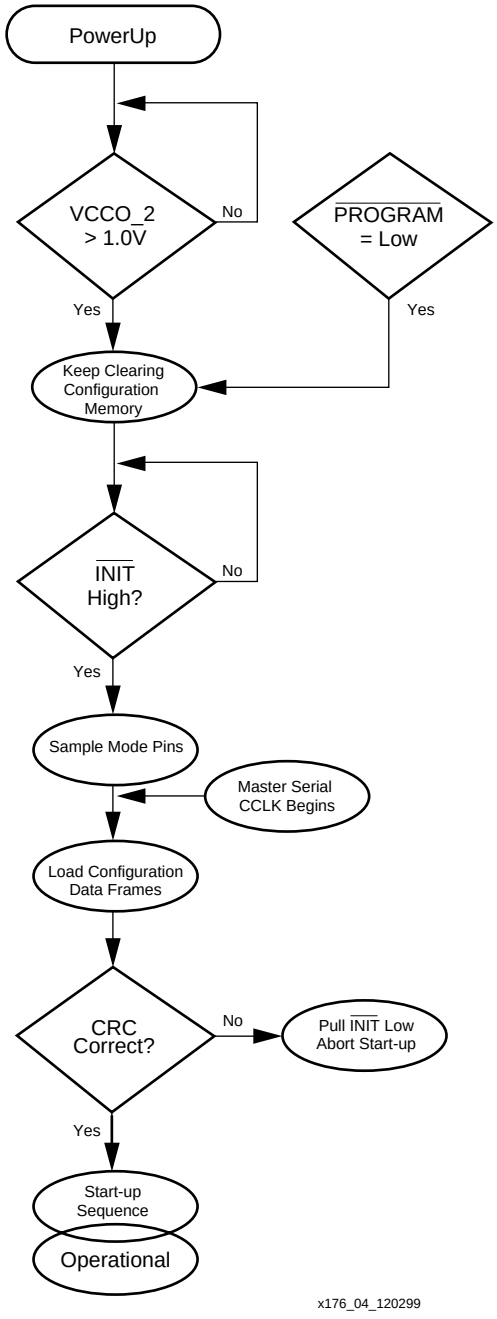

Figure 2: Configuration Flow Diagram

Clearing Configuration Memory

After power-up, the configuration memory is automatically cleared. The 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 PROGRAM pin is held Low, the FPGA continues to clear its configuration memory while holding INIT Low to indicate the configuration memory is being cleared. When PROGRAM is released, the FPGA continues to hold INIT Low until it has completed clearing all the configuration memory.

Delaying Configuration

The INIT pin may also be held Low externally to delay configuration of the FPGA. The FPGA samples its mode pins on the rising edge of INIT. After INIT has transitioned High, configuration may begin. No additional time-out or waiting periods are required, but configuration does not need to commence immediately after the transition of INIT. The configuration logic does not begin processing data until the synchronization word from the bit-stream 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 9 and "Slave Parallel Mode" on page 11.

CRC Error Checking

Twice during the loading of configuration data, an embedded CRC value is checked against an internally calculated CRC value, once just before the last configuration frame is loaded, and then again at the end of configuration. If the CRC values do not match, INIT is asserted Low to indicate that a CRC error has occurred. The Start-up Sequence is aborted and the FPGA does not become active.

To reconfigure the device, the 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 22.

Start-up and Operational States

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

Configuration Pins

Certain pins in the FPGA have been designated to be used in this process. The configuration pins are listed in Table 3. Some pins are dedicated to their configuration function while others are dual-function pins which may 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.
$\overline{\text{PROGRAM}}$	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 tap controller.
TCK	Input		Boundary-scan clock.
TDI	Input		Boundary-scan data input.
TDO	Output	Active	Boundary-scan data output.
Dual Function Pins			
DIN (D0)	Input/Output	Active Bidirectional	Serial configuration data input.
D[0:7]	Input/Output	Active Bidirectional	Slave Parallel configuration data input, readback data output.
$\overline{\text{CS}}$	Input		Chip Select (Slave Parallel only).
$\overline{\text{WRITE}}$	Input		Active Low write select, read select (Slave Parallel only).
BUSY/ DOUT	Output	Open-Drain/ Active	Busy/$\overline{\text{Ready}}$ status for Slave Parallel (open-drain). Serial configuration data output for serial daisy-chains (active).
$\overline{\text{INIT}}$	Input/Output	Open-Drain	Delay configuration, indicate configuration clearing or 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 FPGA's CCLK pin is driven by an external source. In both serial configuation modes, the MSB of each data byte is always written to the DIN pin first.

The Master Serial mode is designed so that the FPGA may configure itself from a Serial PROM. An example of this is shown in Figure 3. The speed of the CCLK is selectable by BitGen options, see "BitGen Switches and Options" on page 3. Care must be observed not to select a CCLK speed that the SPROM cannot support.

The Slave Serial configuration mode allows for FPGAs to be configured from other logic devices, such as microprocessors, or in a daisy-chain configuration. 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

Spartan-II FPGAs may only be daisy-chained with Virtex, XC4000X, Spartan-XL or other Spartan-II FPGAs for configuration. There are no restrictions to the order of the chain. However, if a Spartan-II FPGA is placed as the Master and a non-Spartan-II FPGA is placed as a slave, care must be taken not to select a configuration CCLK speed that is not supported by all devices in the chain.

The separate bit-streams for the FPGAs in a daisy-chain are combined into a single PROM file by using either the PROM File Formatter or the PROMgen utility. This is a requirement for daisy-chain configuration. Separate PROM files may NOT be simply concatenated together to

form a daisy-chain bit-stream. The bit-stream for each device in the daisy-chain must be encapsulated within the bit-stream for the previous device in the chain.

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. At the end of the first stream, the data for the next device is loaded. The data for the down-stream device shows up on the DOUT typically about 64 CCLK cycles after being loaded into the DIN. This is due to internal packet processing. Each daisy-chained bit-stream carries its own synchronization word. Nothing of the first bit-stream is passed to the next device in the chain other than the daisy-chained configuration data.

For the Start-up Sequence of each Spartan-II part to not commence until all of the DONE pins have been released, either the DONE_cycle must be set before the GTS and GSR, or the GTS_cycle and GSR_cycle must be set to the value DONE. When daisy-chaining multiple Spartan-II FPGAs together, it is recommended to 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-Spartan-II FPGAs are included in the daisy-chain, it is important to set their bit-streams for SyncToDONE in the BitGen options. For more information on Spartan-II BitGen options, see "BitGen Switches and Options" on page 3.

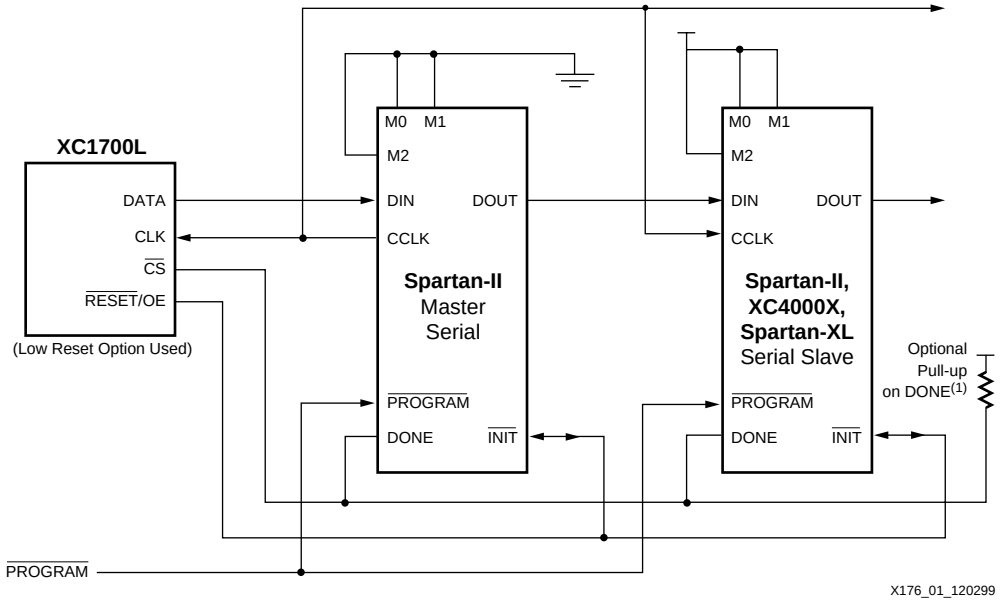

Figure 3: Master/Slave Serial Mode Circuit Diagram

Note:

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

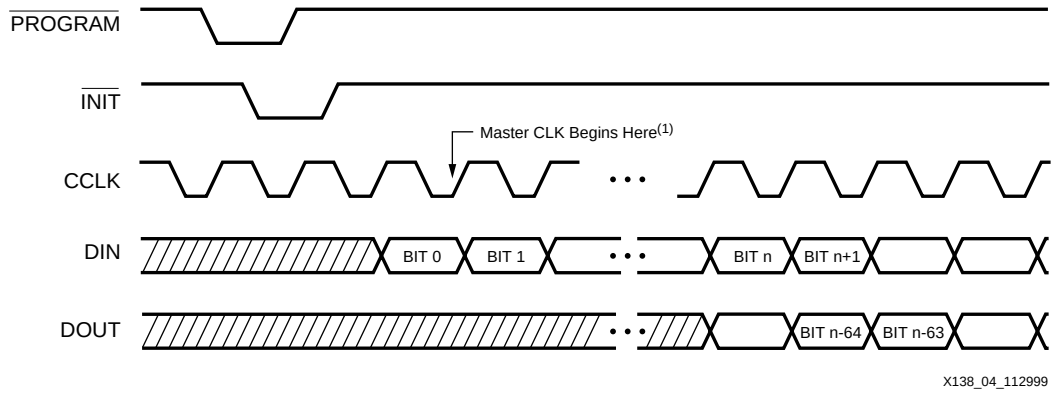

Figure 4: Serial Configuration Clocking Sequence

Note:

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

Slave Parallel Mode

Slave Parallel provides an 8-bit bidirectional data bus interface to the Spartan-II configuration logic that may be used for both configuration and readback. Spartan-II FPGAs may not be serially daisy-chained when the Slave Parallel interface is used. However, they may be connected in a parallel-chain as shown in Figure 5. The DATA pins (D0:D7), CCLK, WRITE, BUSY, PROGRAM, DONE and INIT may be connected in common between all the FPGAs. The CS inputs should be kept separate so that each FPGA may be accessed individually. If all FPGAs are to be configured with the same bit-stream, and readback is not to be used, and the CCLK is less than 50 MHz, the CS pins may be connected to a common line so that the FPGAs are configured simultaneously.

Figure 5 does not show a control module for the Slave Parallel interface. Typically, the Slave Parallel interface is driven by a processor, or microcontroller, or some other logical device such as an FPGA or CPLD.

DATA Pins (D0:D7)

The D0 through D7 pins are a bidirectional data bus in the Slave Parallel mode. Configuration data is written to the bus, while readback data is read from the bus. The bus direction is controlled by the WRITE signal. See "Bit-stream Format" on page 15.

The D0 pin is considered the MSB bit of each byte.

WRITE

When asserted Low, the WRITE signal indicates that data is being written to the data bus. When asserted High, the WRITE signal indicates that data is to be read from the bus.

CS

The Chip Select input (CS) enables the Slave Parallel databus. To write or read data onto or from the bus, the CS signal must be asserted Low. When CS is High, the FPGA ignores all CCLK transitions and does not drive or read from the bus.

BUSY

When 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 that both CS and 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 CS is NOT asserted, BUSY is 3-stated and asserted High.

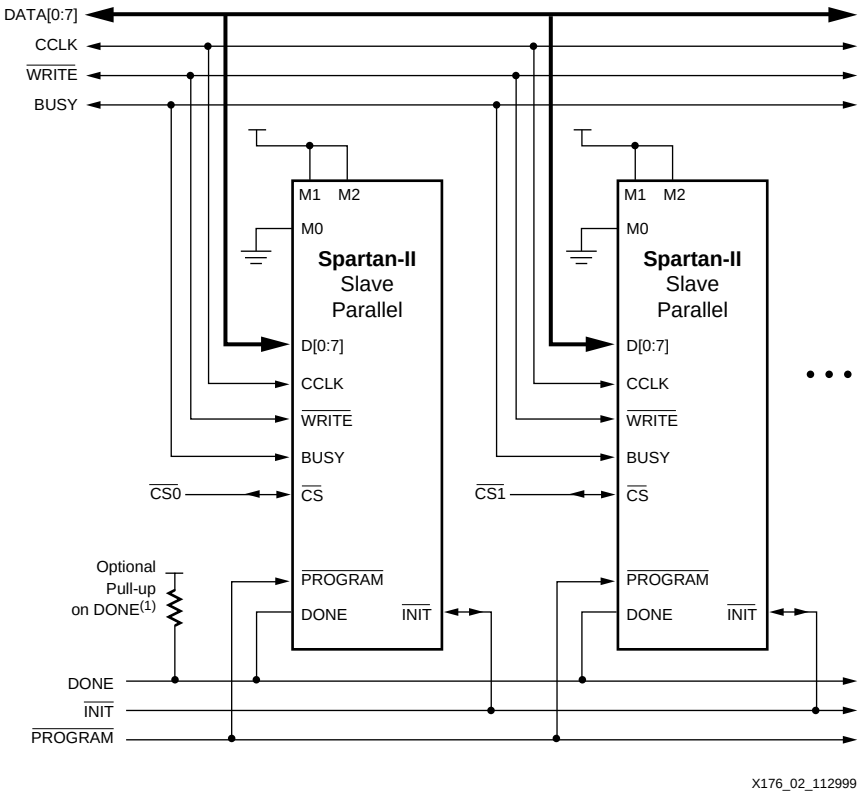

Figure 5: Slave Parallel Mode Circuit Diagram

Note:

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

BUSY need only be used for CCLK frequencies above 50 MHz. For frequencies at or below 50 MHz, BUSY may be completely ignored, see "Express Style Loading" on page 13. For parallel chains such as shown in Figure 5, where the same bit-stream is to be loaded into multiple FPGAs 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 Slave Parallel interface that synchronizes all loading and reading of the data bus for configuration and readback. Additionally, the CCLK drives internal configuration circuitry. The CCLK may be driven either by a free running oscillator or an externally generated signal.

Free Running CCLK

A free running oscillator may be used to drive Spartan-II 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 13. For applications that cannot provide a continuous stream, missing clock edges, refer to the timing diagram discussed in "Non-Contiguous Data Strobing" on page 14. An alternative to a free running CCLK is discussed in "Controlled CCLK" on page 14.

Express Style Loading

In express style loading, a data byte is loaded on every rising edge of the CCLK 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 must be monitored. If BUSY is High, the current byte must be reloaded when BUSY is Low.

The first byte may be loaded on the first rising CCLK edge that INIT is High and both CS and WRITE are asserted Low. CS and WRITE may be asserted anytime before or after INIT has gone High; However, the Slave Parallel interface is not active until after INIT has gone High. The order of CS and WRITE does not matter, but WRITE must be held asserted throughout the configuration. If WRITE is de-asserted before all the data has been loaded, the FPGA aborts the operation. To complete configuration the FPGA must be reset by PROGRAM and reconfigured with the entire stream. For applications that need to de-assert WRITE between bytes, see "Controlled CCLK" on page 14.

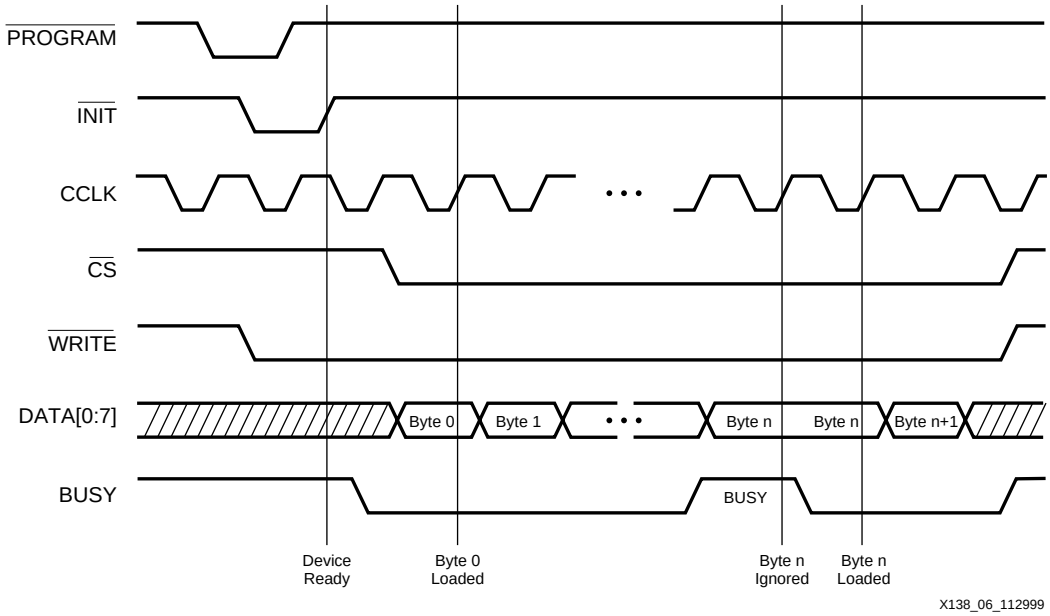

Figure 6: "Express Style" Continuous Data Loading in Slave Parallel

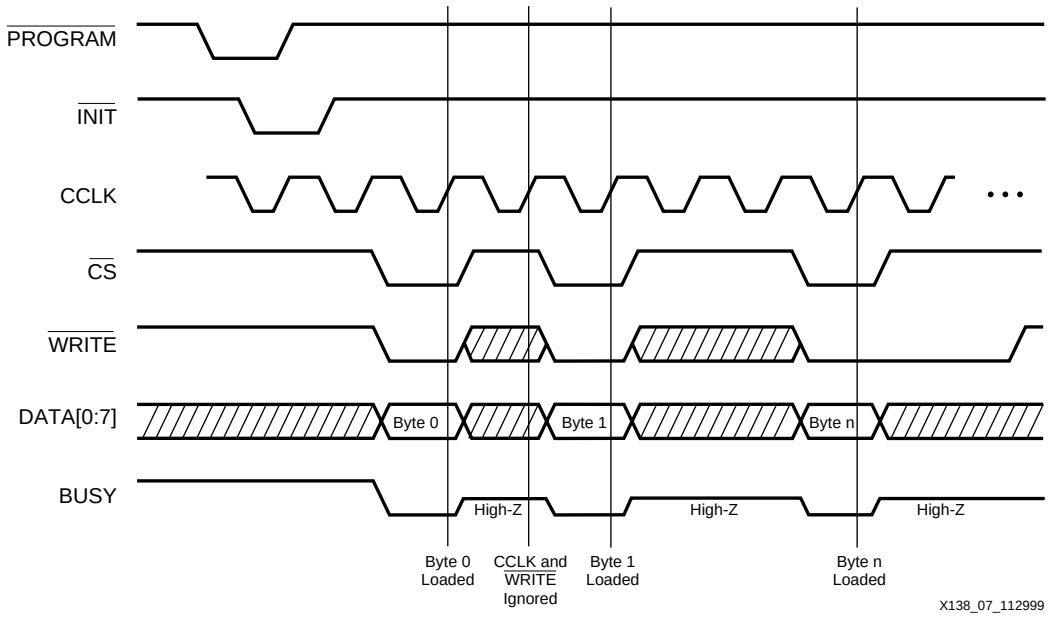

Figure 7: Separating Data Loads by Multiple CCLK Cycles Using $\overline{CS}$

Non-Contiguous Data Strobing

In applications where multiple clock cycles may be required to access the configuration data before each byte can be loaded into the Slave Parallel interface, data may not be ready for each consecutive CCLK edge. In such a case the $\overline{CS}$ signal may be de-asserted until the next data byte is valid on the DATA[0:7] pins. This is demonstrated in Figure 7. While $\overline{CS}$ is High, the Slave Parallel 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 14.

Controlled CCLK

Some applications require that $\overline{WRITE}$ be de-asserted between the loading of configuration data bytes asynchronously from the $\overline{CS}$. Typically, this would be due to the $\overline{WRITE}$ 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, $\overline{CS}$ and $\overline{WRITE}$ are asserted Low while a data byte becomes active. Once the CCLK has transitioned High, the data is loaded. $\overline{WRITE}$ may be de-asserted and re-asserted as many times as necessary, just as long as it is Low before the next rising CCLK edge.

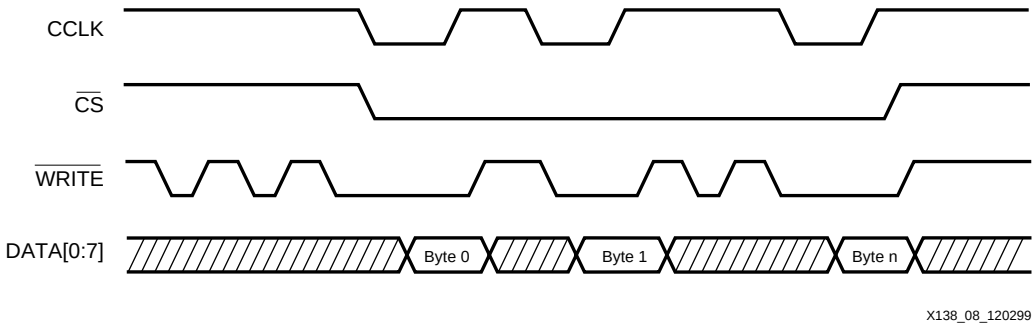

Figure 8: Controlling CCLK for WRITE De-assertion

Bit-stream Format

The Spartan-II bit-stream 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 bit-stream format may be necessary. Therefore, this section describes the Spartan-II bit-stream, 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 bit-stream 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: Spartan-II Configuration Data Frames

Device	Frames	Bits per Frame	Configuration Bits
XC2S15	877	224	197,728
XC2S30	1165	288	336,800
XC2S50	1453	384	559,232
XC2S100	1741	448	781,248
XC2S150	2029	512	1,040,128

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

The Command Register (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

The Frame Length Register (FLR) is used to indicate the frame size to the internal configuration logic. This allows the internal configuration logic to be identical for all Spartan-II 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 bit-stream, 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.

The Configuration Option Register (COR) is loaded with the user selected options from bit-stream generation. See "BitGen Switches and Options" on page 3.

The Control Register (CTL) controls internal functions such as *Security* and *Port Persistence*.

The Mask Register (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.

The Frame Address Register (FAR) sets the starting frame address for the next configuration data input write cycle.

The Frame Data Register Input (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.

The CRC Register (CRC) is loaded with a CRC value that is embedded in the bit-stream and compared against an internally calculated CRC value. Resetting the CRC register and circuitry is controlled by the CMD register.

The Frame Data Register Output (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.

The Legacy Data Output Register (LOUT) pipelines 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 Spartan-II device internally follows the flow chart shown in Figure 9. All the associated commands to perform configuration are listed in Table 7.

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 words, 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 any internal control data to the CTL. 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.

Table 7: Configuration Register Writes

Type	32-bit Words
Command Set 1	
Dummy words	2(1)
Synchronization word	1
Write CMD (RCRC)	2
Write FLR	2
Write COR	2
Write MASK	2
Write CTL	2
Write CMD (SWITCH)	2
Command Set 2	
Write CMD (WCFG)	2
Write FAR	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 CRC	2
Dummy words	4
TOTAL	40

Note:

1. Different versions of BitGen may insert less dummy words at the beginning of the stream.

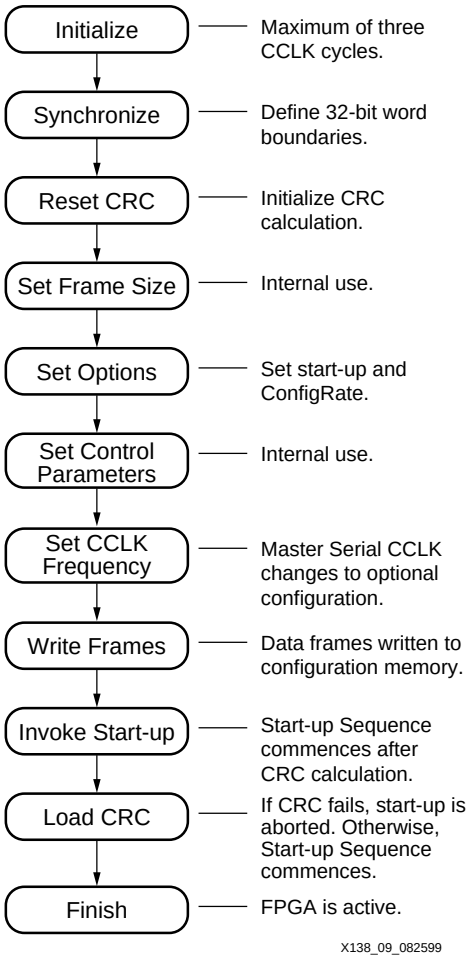

Figure 9: Internal Configuration Processing Flow

The second command set loads the configuration data frames. First, a WCFG (Write Configuration) command is loaded into the CMD register. This, among other things, activates 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 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 kicks off 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 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 Bit-stream

Spartan-II FPGAs have the ability to be only partially re-configure or readback; however, this topic is beyond the scope of this note. For more information on partial configuration, refer to Application Note XAPP151 "Virtex Configuration Architecture, Advanced Users' Guide", which also applies to the Spartan-II family. The standard bit-stream, 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 has been broken into three tables to follow the three command sets described in the previous subsection.

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

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

Table 8: Bit-stream Header and Configuration Options

Data Type	Data Field
Dummy word	FFFF FFFFh
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 CTL	3000 A001h
Packet Data: Control commands	0000 0000h
Packet Header: Write to CMD register	3000 8001h
Packet Data: SWITCH	0000 0009h
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 12, is always a single 32-bit word that describes the packet type, whether it is a read/write function or a specific configuration register address (see Table 5) as the destination, and how many 32-bit words are in the following Packet Data portion. A Type 1 Packet Data portion may contain anywhere from 0 to 2047 32-bit data words.

The first packet in Table 8 is a Type 1 packet header that specifies to write one data word to the CMD register. The following packet data is [that] data word which specifies 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 will be the frame size from Table 4, divided by 32 and minus 1, and converted to Hex (e.g., the FLR for an XC2S150 is 0Fh).

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: Bit-stream Data Frames and CRC

Data Type	Data Field
Packet Header: Write to FAR register	3000 2001h
Packet Data: Starting frame address	0000 0000h
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: Configuration Data Frames in 32-bit words. Total number of words specified in packet header	---- ----h ....

The third packet loads the configuration options into the COR register. The binary description of this register is not documented. Following are similar writes to the MASK and CTL, followed by the SWITCH command to the CMD register which selects the CCLK frequency specified in the COR. Finally, the WCFG command is loaded into the CMD register so that the loading of frame data may commence.

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

The loading of data frames uses 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 138,000,000 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 neither Register Address nor 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, a Type 1 packet header issues a write command to the FDRI, followed by a Type 2 packet header which specifies the number of data words to be loaded, and then followed by the actual frame data as the Type 2 packet data. Writing data frames may require a Type 1/Type 2 packet combination, or a Type 1 only. This depends on the amount of data being written.

This series of FAR and FDRI writes are 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 loading of the final CRC check. The FPGA does not go active until after the final CRC has been loaded. The number of clock cycles required to complete the start-up depends on the BitGen options selected; however, completion of the configuration process requires eight to 16 clock cycles after the final CRC has been loaded. Typically, DONE is released within the first seven CCLK cycles after the final CRC value is loaded; however, 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 would require only 16 bits (two bytes), but in Slave Parallel mode this would require 16 bytes of dummy words at the end of the bit-stream. Since the intended configuration mode to be used is unknown by BitgGen, four 32-bit dummy words (16 bytes) are always placed at the end of the bit-stream.

Table 10: Bit-stream 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 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

Spartan-II configuration utilizes a standard 16-bit CRC checksum algorithm to verify bit-stream integrity during configuration. The 16-bit CRC polynomial is:

$$CRC\text{–}16 = X^{16} + X^{15} + X^{2} + 1$$

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 in Table 11, the address code for the register, preceded by the 32-bit data word, are 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 will not effect the CRC checksum.

Table 11: CRC Effecting Registers

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

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

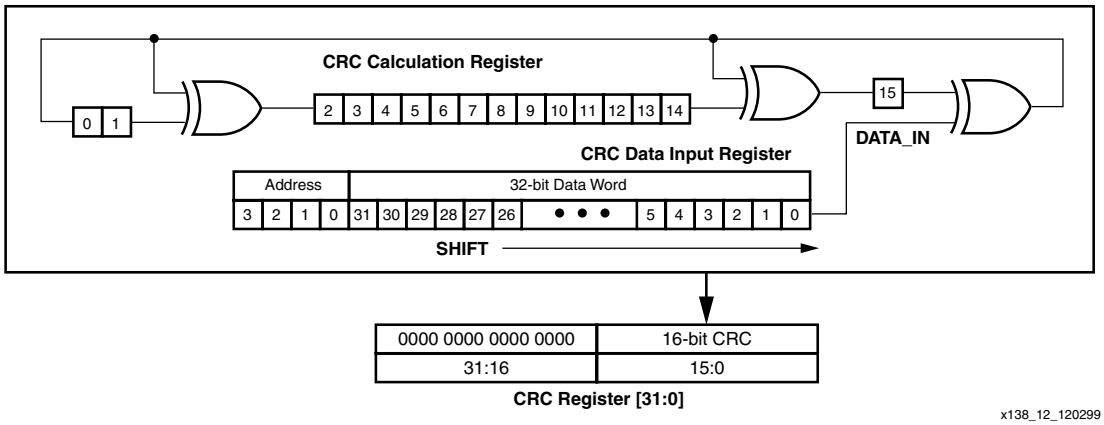

Figure 12: Serial 16-bit CRC Circuitry

Readback

Readback is the process of reading out 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 current LUT RAM and BlockRAM values.

Readback is only available through the Slave Parallel and boundary-scan interfaces. This application note demonstrates using the Slave Parallel interface for performing readback. For information on using Boundary-Scan for readback refer to Application Note XAPP139 "Virtex Configuration and Readback through Boundary-Scan", which also applies to the Spartan-II family.

Readback Verification and Capture

Readback verification is used to verify the validity of the stored configuration data. 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 and may be extracted after a readback of the configuration memory.

While both verify and capture can be performed in one readback, each requires slightly different preparation and post processing. This section continues with an explanation of the needed preparation with regards to design entry and needed software options for both verification and capture. A description of processing the readback stream is given for verification and capture separately (see "Verifying Configuration Data" on page 28 and "Capturing Register States" on page 32, respectively).

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, then the Spartan-II library primitive CAPTURE_SPARTAN2 must be instantiated in the user design as shown in Figure 13.

The CAPTURE_SPARTAN2 component is used in the FPGA design to control when the logic states of all the registers are captured into configuration memory. The CLK pin may be driven by any clock source that would synchronize CAPTURE to the changing logic states of the registers. The CAP pin is an enable control. When CAP is asserted, the register states will be captured into memory on the next rising CLK transition.

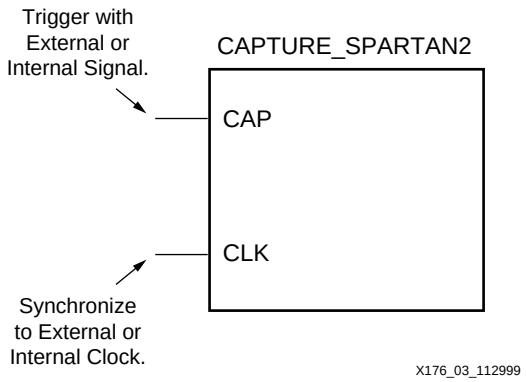

Figure 13: Readback Capture Library Primitive

Enabling Readback in the Software

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

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

The **-w** overwrites existing output. The **-l** generates a *Logic Allocation* file, which is discussed in "Capturing Register States" on page 32. The **-m** generates a *Mask* file, which is discussed in "Verifying Configuration Data" on page 28. The **-g readback** generates the *readback bit* file, which is discussed in "Readback Operations" on page 25, and the **-g persist:X8** keeps the Slave Parallel interface active after configuration. A listing of all the associated BitGen files used for readback is shown in Table 12.

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

Table 12: BitGen files used in Readback

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

Readback Operations

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

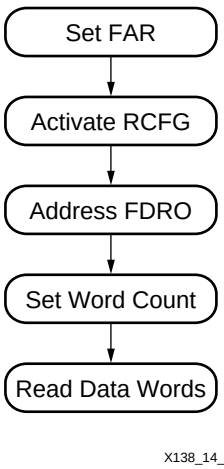

Figure 14: CLB Readback Operation Flow

The entire configuration memory map cannot be read out in one readback sequence. Three sequences are required: one for the CLB frames and two for the BlockRAM 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 two frame sections contain the configuration data for the two columns of BlockRAMs. The BlockRAM configuration data need not be used for verification purposes, but may be used to extract the current internal states of the BlockRAMs 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 only requires one (the CLB frames). This section describes the process for readback of the CLB frames. For readback of the BlockRAM frames first review this section then refer to "Readback of BlockRAM Frames" on page 33.

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 on page 31, `<design>.rba` and the `<design>.msk` file shown in Figure 17 on page 29.

To perform the first readback sequence after configuration, it is not necessary to resynchronize the Slave Parallel interface. However, if resynchronization is required, then an ABORT process should be executed followed by loading in the synchronization word. See Table 13. If a resynchronization is not needed, the synchronization word may be omitted from the readback command set. If the synchronization word is reloaded, it is merely interpreted as a "No Operation" command and is ignored. The total readback command set, not including the synchronization word, is 24 bytes.

Table 13: Readback Command Set for CLB Frames

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

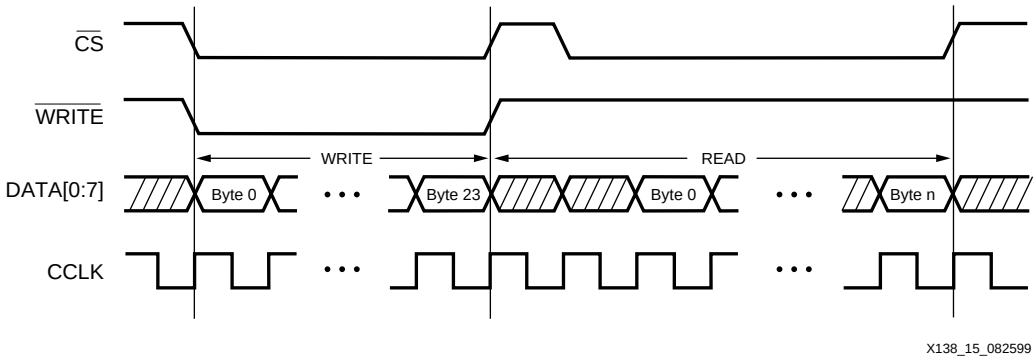

Figure 15: Readback Clocking Sequence

Since all data loaded through the Slave Parallel 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 the FAR, shown in Table 13. For a complete readback of the CLB frames this is always 0000 0000h. However, this value is different for the BlockRAM frames. See "Readback of BlockRAM Frames" on page 33.

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 read back. The number of data words to be read back depends on which Spartan-II device is being read back, shown in Table 14. Type 1 or Type 2 headers may be used depending on the amount of data that is to be read back. See "Bit-stream Format" on page 15.

Table 14: CLB Frame Word Counts per Device

Device	Type 2 Packet Header for CLB Frames	CLB Frames Word Count
XC2S15	4800 146Dh	5229
XC2S30	4800 2463h	9315
XC2S50	4800 3E04h	15876
XC2S100	4800 581Ah	22554
XC2S150	4800 76B0h	30384

Now the readback data is ready to be clocked out. The readback sequence is shown in waveform format in Figure 16. First assert the CS and 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 28 for a detailed description of these files. Then, de-assert WRITE and CS, and de-activate any external drivers on the D0 through D7 pins. To begin reading back the data, assert CS leaving 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 CS. The process may be repeated for additional readbacks.

Readback Data Format

The readback data stream contains only 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. The readback stream also does not contain any CRC values since this information is stored in internal configuration registers, not the configuration memory, and no CRC calculation is performed during readback.

The readback stream consists of data frames each preceded by one 32-bit word of pad data, shown in Figure 17. 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 bit-stream. 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.

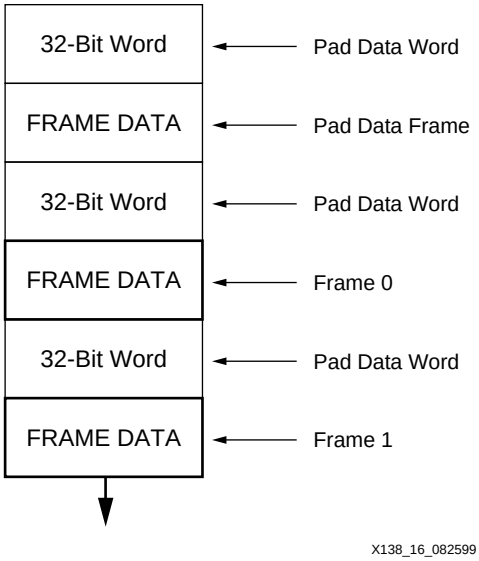

Figure 16: Readback Data Stream

Table 15: Readback System Bytes for CLB Frames

Device	CLB Frames	Bytes per Frame	Frame Bytes	Pad Bytes	Readback Bytes
XC2S15	746	24	17904	3012	20916
XC2S30	1034	32	33088	4172	37260
XC2S50	1322	44	58168	5336	63504
XC2S100	1610	52	83720	6496	90216
XC2S150	1898	60	113880	7656	121536

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

The declarations portion is throw-away data. The first command set is for the CLB frames and includes the synchronization word which may be omitted if an Abort has not been executed. The second and third command sets are for BlockRAM 0 and BlockRAM 1.

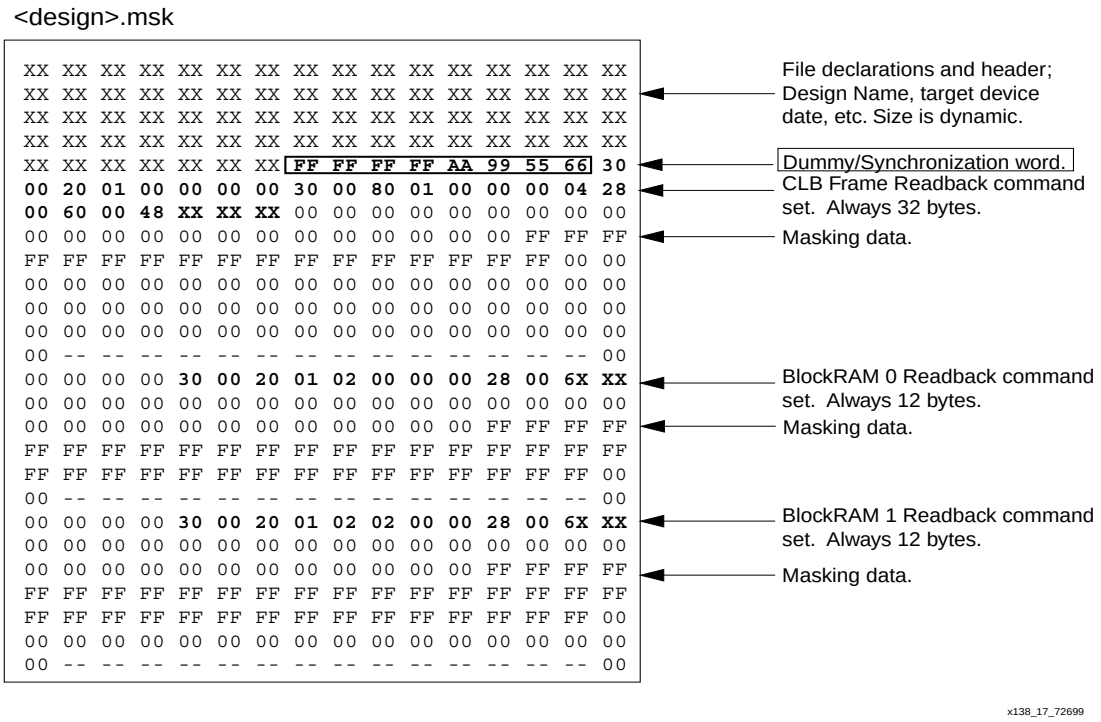

Figure 17: Readback Mask File

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 is:

$$RBB[i] = \overline{MSK[i]} * DATA[i].$$

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

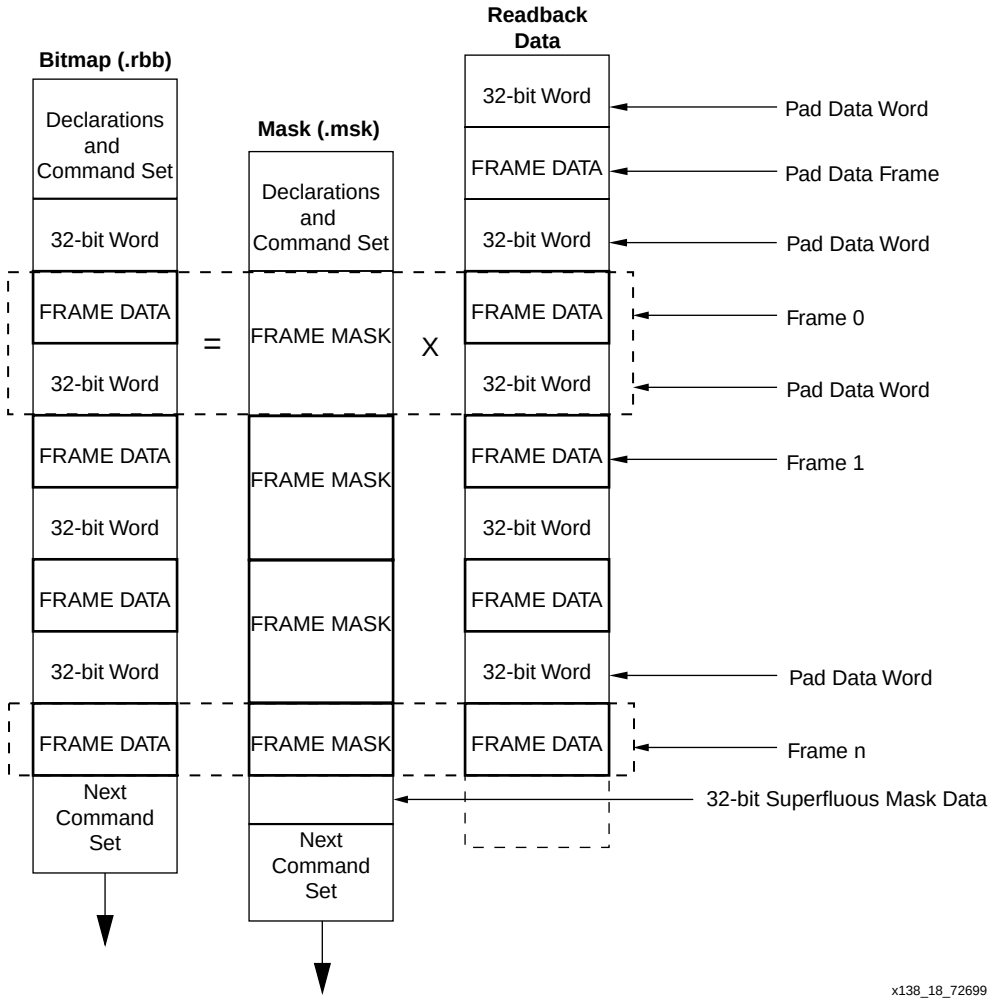

Figure 18: Readback Data Stream Alignment

The readback bit file `<design>.rbb` provides the configuration stream bit-map (data frames) for verifying the readback data stream. This must be used instead of the bit-stream `<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.

Unlike the mask file, the bitmap does take into account that the pad data in between the data frames in the readback data stream precede, 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 21.

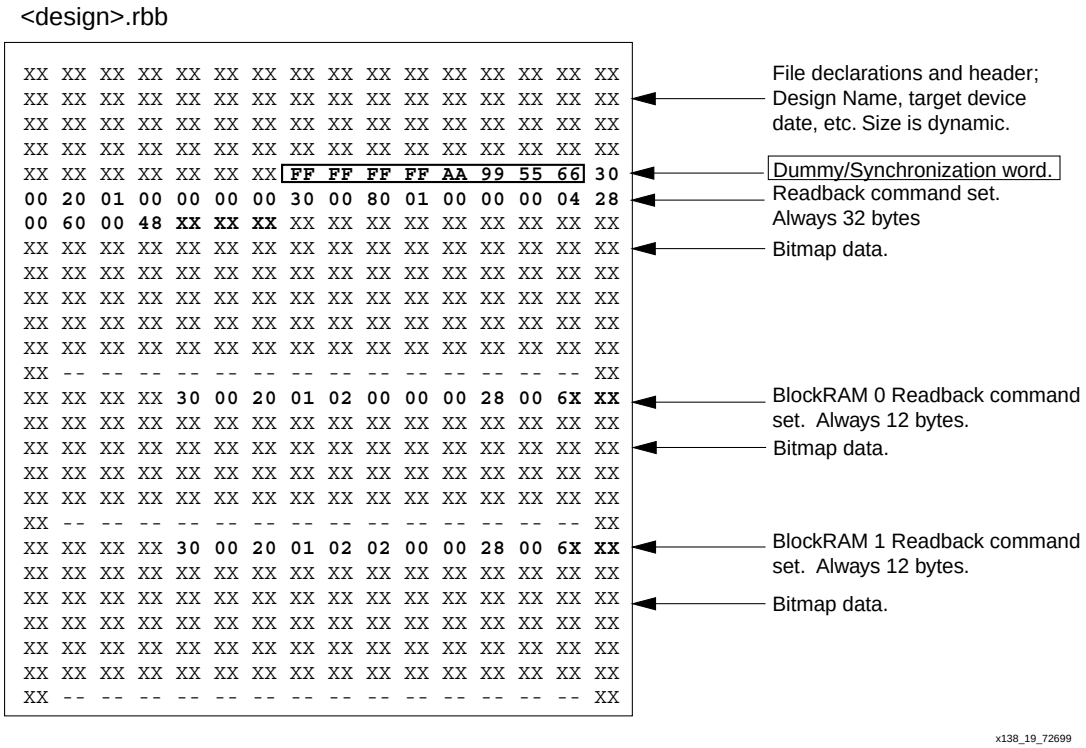

Figure 19: Readback Bit File

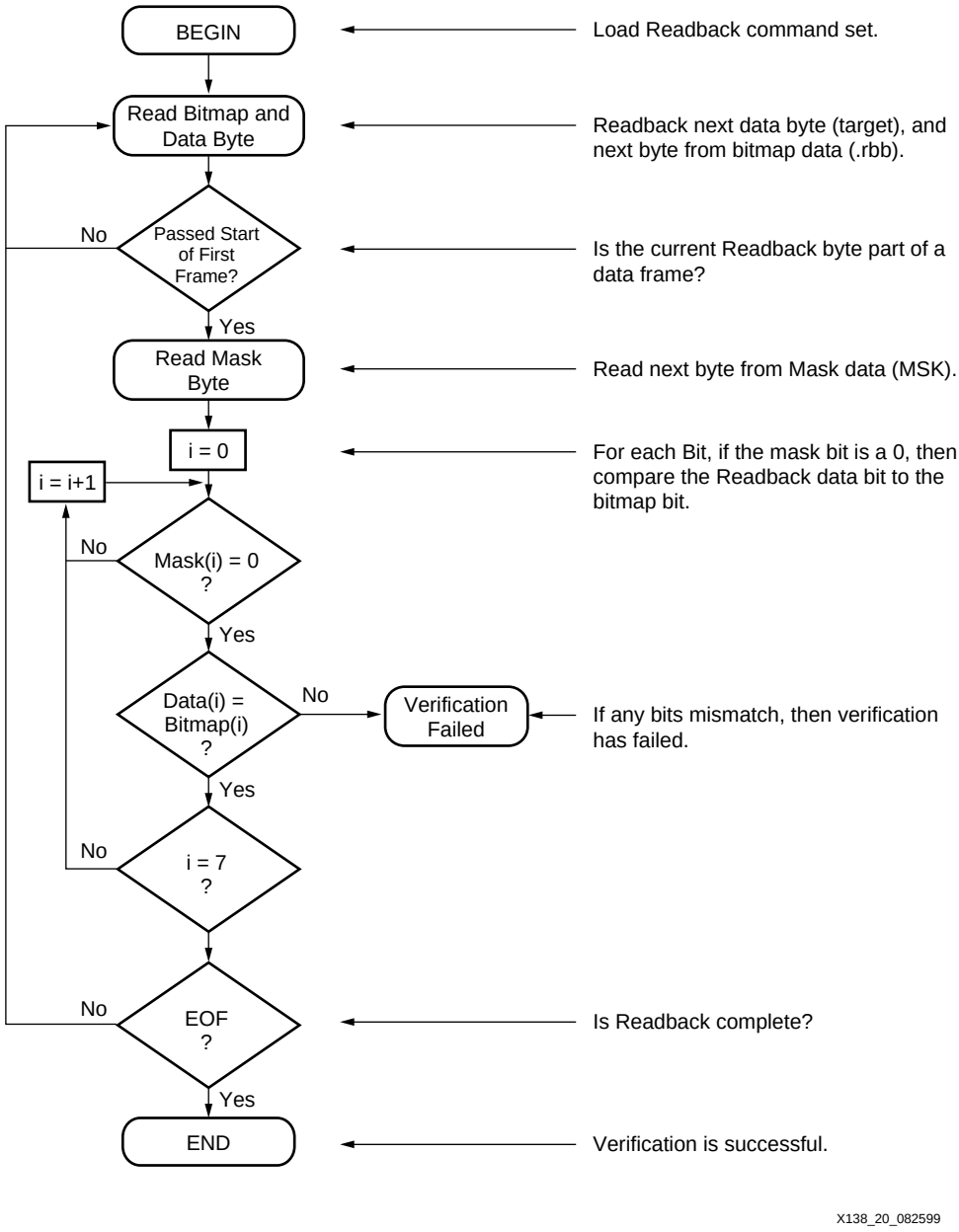

Figure 20: Readback Verification Flow Diagram

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 BlockRAM 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 + 1) x FrameLength + FrameOffset + 32

FrameLength = Bits per frame (Table 4 on page 15)

The readback bit number, calculated above, is relevant to the entire readback stream. That is, all readback data, including the pad frame and pad data words, should be counted to find the state bit represented by the readback bit number.

The frame order for the BRAMs assume that all the bits from a complete readback of all the CLB frames have been counted, and that count continues on with the readback of BlockRAM_0 frames and then the BlockRAM_1 frames. Therefore, when readback of BlockRAMs is used, the data frame bit count must be offset by the number of bits in all the CLB frames. This offset may be obtained from the Frame Bytes number in Table 15 and multiplying by 8.

Readback of BlockRAM Frames

A readback of the BlockRAM frames may follow the same procedure as that for the CLB frames. However, when a readback of the BlockRAM is initiated, control of the BlockRAM is

taken from the user logic so that the BlockRAM memory elements may be accessed by the configuration circuitry. In order to make this hand off smooth and glitch free it is recommended that a SHUT-DOWN be performed prior to readback, and then start-up again after readback. After a Shut-down Sequence has been performed, all user logic and I/O will be disabled until a Start-up Sequence is performed. This is the same Start-up Sequence that is used in configuration. See "Start-up Sequence" on page 5. The Start-up Sequencer is used for both start-up and shut-down.

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

The flow for a BlockRAM readback is shown in Figure 21. The readback follows the same process as the CLB frames, but with a different FAR value. See "Readback Operations" on page 25. However, the synchronization step may be omitted if a previous readback sequence has already synchronized the Slave Parallel 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 the 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 17. 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 initiate the readback sequence of the BlockRAM.

To readback both columns of BlockRAMs, the readback sequence must be repeated for each column. The sequence is the same except for different FAR values which are shown in Table 16.

Table 16: Starting Frame Address

Frame Type	FAR
BlockRAM_0	0200 0000h
BlockRAM_1	0202 0000h

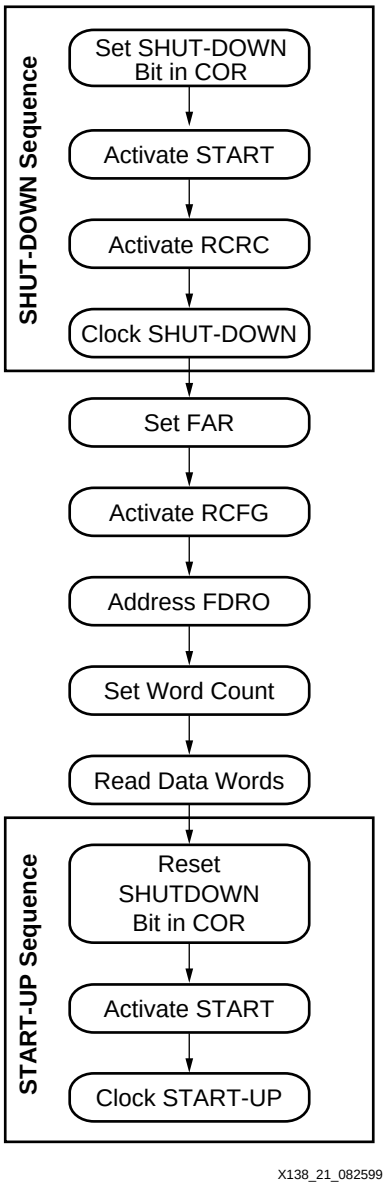

Figure 21: BlockRAM Readback Operation Flow

Table 17: Readback Command Set for BlockRAM 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
BlockRAM_0 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
BlockRAM_1 Readback Sequence	
Packet Header: Write to FAR register	3000 2001h
Packet Data: Starting frame address	0202 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 BlockRAM Frames in every device size consist of 65 frames in each column; however, the frame sizes vary per device. The word count for BlockRAM readback and associated opcode for each device are shown in Table 18.

Table 18: Word Counts per Frame Section

Device	Type 2 Packet Header for BlockRAM Frames	BlockRAM Frames Word Count (1 of 2)
XC2S15	TBD	TBD
XC2S30	TBD	TBD
XC2S50	4800 030Ch	780
XC2S100	4800 038Eh	910
XC2S150	4800 0410h	1040

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

Acknowledgements

Original material by Carl Carmichael.

Revised for the Spartan-II family by Kim Goldblatt

Revision History

Date	Revision #	Activity
12/04/99	0.9	Advance Information Release.