XAPP583 (v1.0) May 31, 2012

Using a Microprocessor to Configure 7 Series FPGAs via Slave Serial or Slave SelectMAP Mode

Author: Matt Nielson

Summary

With embedded systems becoming more popular, many designers want to reduce their component count and increase flexibility. To accomplish both of these goals, a microprocessor already available in the system can be used to configure an FPGA. This application note provides a thorough discussion of FPGA configuration via a microprocessor, covering the Xilinx 7 series FPGAs. C code is included to illustrate an example application using either Slave Serial or Slave SelectMAP mode. The included example configures a Kintex™-7 XC7K325T device from a MicroBlaze™ processor.

System Overview

Today's systems demand greater functionality in less space and at reduced cost. In addition, each generation of Xilinx FPGAs delivers higher performance and increased capabilities. Although Xilinx FPGAs support direct configuration from third-party flash, an embedded processor-based configuration solution can allow for advanced FPGA configuration applications and reduce board real estate requirements, assuming that an external, embedded processor with sufficient memory is already a prerequisite for the system. This technique requires that the processor be operational before FPGA functionality is required.

This application note describes a technique for configuring an FPGA from an embedded processor. The Xilinx design tools generate the FPGA configuration data, also known as a configuration bitstream. To configure the FPGA, the processor uses the described technique to load the generated bitstream into the FPGA. A system diagram is shown in Figure 1.

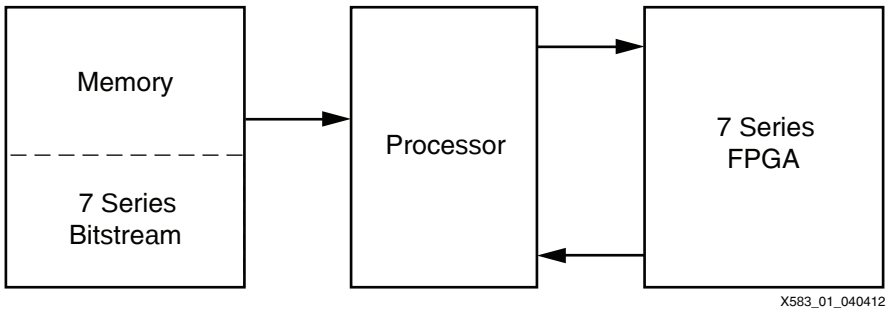


Figure 1: **System Diagram**

A microprocessor whose primary purpose is to perform other tasks can also be used to coordinate the loading of configuration data into a Xilinx FPGA. A processor provides greater flexibility, for example, in choosing which of multiple configuration files to program into the FPGA.

Configuration Background

Microprocessor configuration of 7 series FPGAs can be accomplished in either Slave Serial or Slave SelectMAP mode. There are several similarities between these two modes. Most importantly, the basic configuration sequence is identical for both modes.

The JTAG configuration interface can also be controlled via a microprocessor, but that interface is not included in this application note.

Basic Configuration Sequence

The basic configuration sequence is shown in Figure 2.

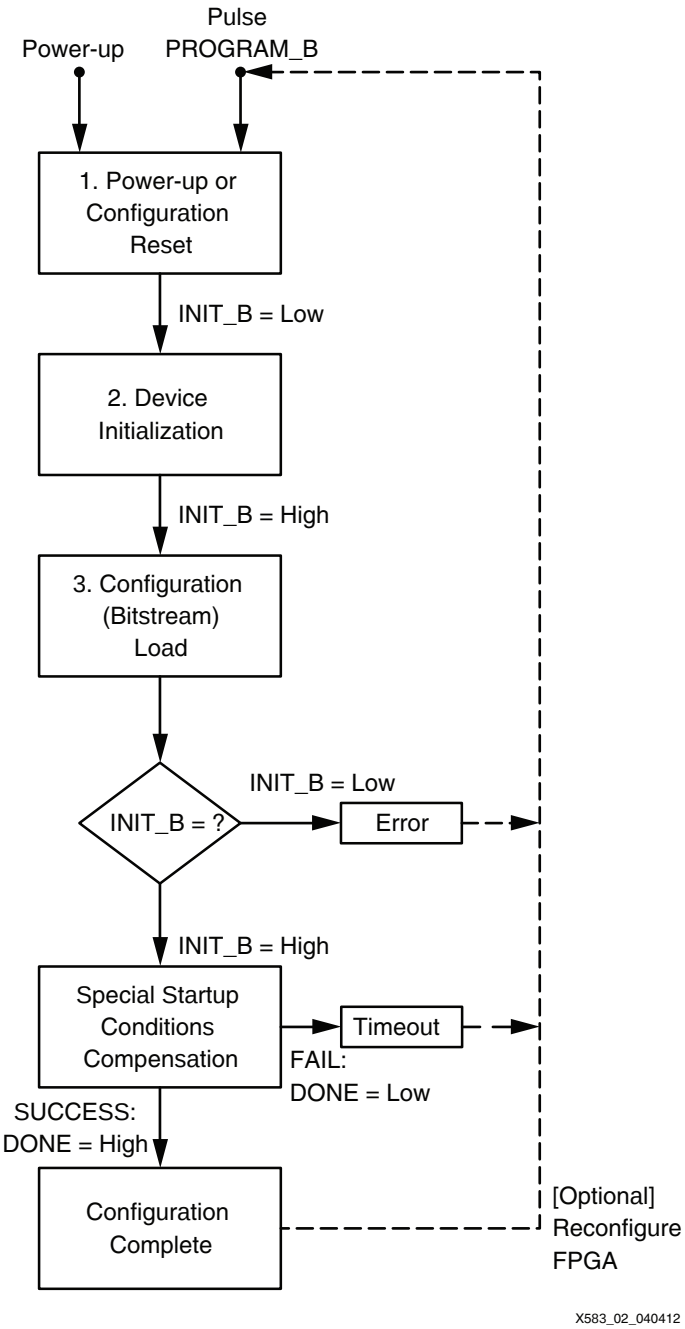


Figure 2: **Basic Configuration Sequence**

1. Power-up or configuration reset: The configuration sequence begins with power-up or configuration reset. Power-up is when power is first applied to the FPGA. Configuration reset occurs when the PROGRAM_B pin is asserted.
2. Device initialization: Power-up or configuration reset triggers a configuration memory initialization process. During configuration initialization, the FPGA drives the INIT_B pin Low, resets the internal configuration state machines, and clears the configuration memory. At the completion of the initialization process, the FPGA releases the INIT_B pin to a high-impedance state and waits in this state until the INIT_B pin goes High.

   When the INIT_B pin is released to a high-impedance state, an external resistor is required to pull the INIT_B signal High. When INIT_B goes High, the FPGA samples its configuration mode M[2:0] pins. The mode pins determine the configuration mode for the remaining steps of the FPGA configuration flow. When M[2:0] = `111`, the FPGA is set to the Slave Serial configuration mode. When M[2:0] = `110`, the FPGA is set to the Slave SelectMAP configuration mode. After sampling the mode pins, the FPGA is ready to receive configuration data (also known as a configuration bitstream).
3. Configuration load: When the FPGA is in a slave configuration mode, an external microprocessor can load the bitstream into the device during this step of the configuration sequence. For Slave Serial mode, the bitstream is loaded through the FPGA D01_DIN pin, one bit per each rising edge of CCLK. For Slave SelectMAP mode, data is loaded through the FPGA D[31:00] pins on each rising edge of CCLK when the FPGA CSI_B and RDWR_B pins are Low. See UG470, *7 Series FPGAs Configuration User Guide* for the bitstream bus width auto-detection pattern that determines whether 8, 16, or 32 bits are loaded through the SelectMAP D[07:00], D[15:00], or D[31:00] pins, respectively, on each rising edge of CCLK.

After the bitstream is loaded, the configuration sequence is complete.

Configuration Bitstream Details

The Xilinx design tools generate the FPGA configuration bitstream with all commands and data that are necessary to check for a matching target device identifier, load the configuration memory data, and start the FPGA design. See *7 Series FPGAs Configuration User Guide* for the details of the bitstream composition.

Generally, the processor code does not need to understand the composition of the bitstream. However, a few key commands within the bitstream should be understood for correctly starting the FPGA design, validating the processor configuration code, and debugging the processor configuration code.

Sync Word

After a few pad bits and the bus width auto-detection pattern, at the beginning of the bitstream is a 32-bit sync word (`0xAA995566`). The sync word bit order (or byte order) from the processor to the FPGA is the most important signature for correct delivery of the bitstream to the FPGA.

Device Identification

A device ID check follows the sync word in the bitstream. This checks that the appropriate device is receiving the bitstream. If the device ID check fails, the FPGA drives its INIT_B pin Low to indicate a configuration error.

CRC Check

Near the end of the bitstream is a command that checks the internal configuration cyclic redundancy check (CRC) against an expected value from the bitstream. If the device recognizes the CRC check command and if the CRC does not match the expected value, the FPGA drives its INIT_B pin Low to indicate a configuration error.

Start-Up

After the device receives all the configuration data from the bitstream and passes the CRC check, a start-up command near the end of the bitstream initiates a start-up sequence for the loaded design. See UG470, *7 Series FPGAs Configuration User Guide* for the details of the start-up sequence. During the start-up sequence, the FPGA releases its DONE pin to a high-impedance state. The FPGA waits at that point in the start-up sequence until the DONE pin goes High. A strong external pull-up resistor (or the BitGen DriveDONE option) is required to bring the DONE pin High. The start-up sequence completes when it reaches the end of start-up (EOS) state.

Note: The DONE signal is released before EOS. Thus, the processor code must not stop delivering the bitstream or stop delivering CCLK pulses when DONE transitions to High.

The processor code must deliver all bits/words of the bitstream to the FPGA. Typically, the start-up sequence completes to EOS before the last bit of the bitstream is delivered to the FPGA.

Special Start-Up Conditions

A few BitGen options affect FPGA start-up by potentially extending the start-up sequence beyond the end of the delivered bitstream. For a few of these options, additional CCLK pulses must be issued to the FPGA after the delivery of a configuration bitstream. Example BitGen options that affect start-up include LCK_CYCLE or MATCH_CYCLE.

To cover all possible bitstream start-up options, the processor code must be written to:

1. Load all the bitstream data.
2. Continue to apply CCLK cycles (while the data bits on D01_DIN or D[31:00] are all ones) until DONE is asserted High.
3. Apply eight additional CCLK cycles after DONE is asserted High to ensure completion of the FPGA start-up sequence.

If the LCK_CYCLE or MATCH_CYCLE are selected, the above sequence works when the LCK_CYCLE or MATCH_CYCLE are set to a value less than the DONE_CYCLE. This assures that DONE toggles after the start-up extending events occur. See *7 Series FPGAs Configuration User Guide* and the BitGen section of UG628, *Command Line Tools User Guide* within the ISE® Design Suite manuals for additional BitGen options and details.

Slave Serial Configuration

After INIT_B goes High, one bit of Slave Serial configuration data (presented on the D01_DIN pin) is loaded into the configuration logic on each rising CCLK edge (refer to the appropriate data sheet for setup and hold time specifications). Table 1 describes the pins used during Slave Serial configuration.

Table 1: **Slave Serial Pin Descriptions**

Signal Name	Direction	Description
CCLK	Input	Configuration clock.
PROGRAM_B	Input	Active-Low reset to configuration logic.
INIT_B	Input/Output	Active-Low FPGA initialization pin. Indicates when the device is ready to receive configuration data. Also indicates any configuration errors. Can be held Low externally to delay configuration.
DONE	Input/Output	Indicates configuration is complete. Can be held Low externally to delay start-up.
M[2:0]	Input	Configuration mode selection.

Table 1: **Slave Serial Pin Descriptions *(Cont'd)***

Signal Name	Direction	Description
D01_DIN	Input	Serial configuration data input.
DOUT	Output	Data output for serial daisy chains.

Slave SelectMAP Configuration

Slave SelectMAP x8 data bus configuration data is loaded one byte at a time when presented on the D[07:00] bus on each rising CCLK edge (see Data Formatting and Bit-Swapping, page 5 for more details). Table 2 lists the SelectMAP pins.

Table 2: **Slave SelectMAP Pin Descriptions**

Signal Name	Direction	Description
CCLK	Input	Configuration clock.
PROGRAM_B	Input	Active-Low reset to configuration logic.
INIT_B	Input/Output	Active-Low FPGA initialization pin. Indicates when the device is ready to receive configuration data. Also indicates any configuration errors. Can be held Low externally to delay configuration.
DONE	Input/Output	Indicates configuration is complete. Can be held Low externally to delay start-up.
M[2:0]	Input	Configuration mode selection.
D[31:00]	Input	Parallel configuration data input.
CSI_B	Input	Active-Low chip select input.
RDWR_B	Input	Active-Low write select/read select.

Two extra control signals are present for SelectMAP: CSI_B and RDWR_B. These signals must both be asserted Low for a configuration byte to be transferred to the FPGA.

Note: The appropriate 7 series FPGA family data sheet must be referred to for setup and hold specifications for all signals. The Slave SelectMAP configuration reference design for this application note demonstrates configuration via only the 8 bit SelectMAP bus. The 7 series FPGA families also support 16- or 32-bit-wide SelectMAP buses. With appropriate modifications and attention to bit ordering, the principles of this reference design can be extended for the 16- or 32-bit-wide SelectMAP bus. See UG470, *7 Series FPGAs Configuration User Guide* for more details.

Data Formatting and Bit-Swapping

Because the configuration bitstream is loaded into memory connected to the processor, it must be formatted in a way that the processor (or another device that programs the memory) can use. To support various solutions, Xilinx tools can produce a number of different formats (see Table 3). PROMGen, the PROM file generator, converts one or more bitstream files into a PROM file. PROM files do not need to be used with a PROM. They can be stored anywhere and delivered by any means.

Table 3: **Xilinx Tool Formats**

File Extension	Description
`.bit`	Binary file containing header information that should not be downloaded to the FPGA
`.rbt`	ASCII file containing a text header and ASCII 1s and 0s
`.bin`	Binary file containing no header information

Table 3: **Xilinx Tool Formats *(Cont'd)***

File Extension	Description
.mcs	ASCII PROM formats containing address as well as checksum information
.hex	ASCII PROM format only containing data

Data ordering in Slave Serial configuration is very simple. Loading begins with the first bit in the bitstream and continues one bit at a time until the end of the file is reached.

In contrast, data ordering for Slave SelectMAP configuration is slightly more complex. A description of bit order for the 8-bit SelectMAP parallel bus is presented here. (See UG470, *7 Series FPGAs Configuration User Guide* for the parallel bus bit order when using the SelectMAP 16-bit and 32-bit bus widths.) Configuration data is loaded one byte at each rising CCLK edge, and the MSB of each byte is presented on the D[00] pin, not the D[07] pin. Because of this non-conventional ordering, presenting the data as is from the .bin file is generally incorrect. The reason is that most processors interpret D[07] (not D[00]) as the most significant bit in each byte. Connecting D[07] on the processor to the D[07] on the FPGA SelectMAP data bus effectively loads the data backwards, resulting in unsuccessful configuration. For this reason, the source data stream might need to be bit-swapped, with bits in each byte in the data stream reversed. Figure 3 shows two bytes (0xABCD) being reversed.

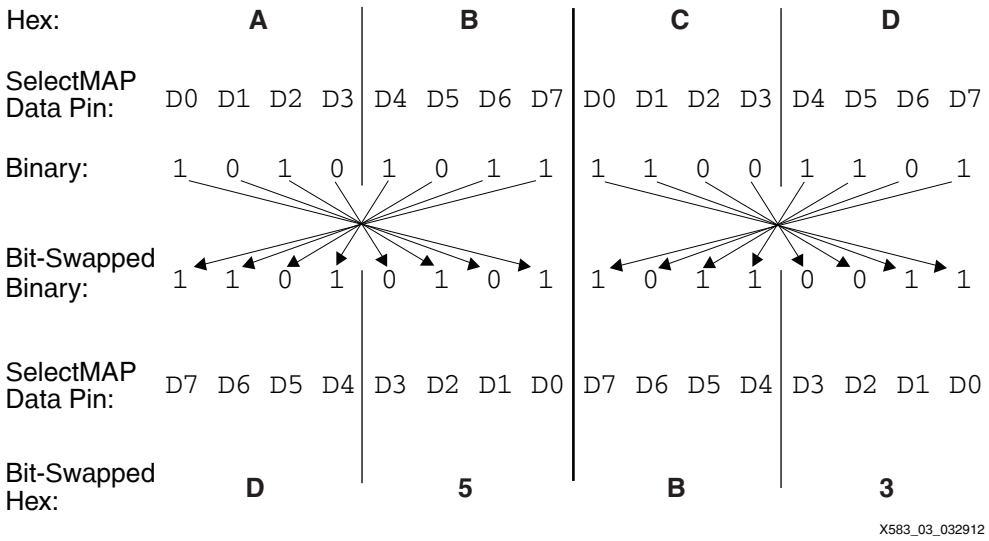


Figure 3: **Bit-Swapping Example for x8**

Regardless of the orientation of the data, the MSB of the first byte of the data must be transmitted to D[00]. However, in the bit-swapped version of the data, the bit that must be transmitted to D[00] is the rightmost bit and, in the non-bit-swapped data, the leftmost bit. With Xilinx tools, the .mcs files are always bit-swapped, and the .bit, .rbt, and .bin files are never bit-swapped. Hexadecimal files can be produced bit-swapped according to the PROMGen tool command line options.

Note: Whether or not data is bit-swapped is entirely processor- or application-dependent and is generally only applicable for Slave SelectMAP applications. Non-bit-swapped data should be used for Slave Serial downloads.

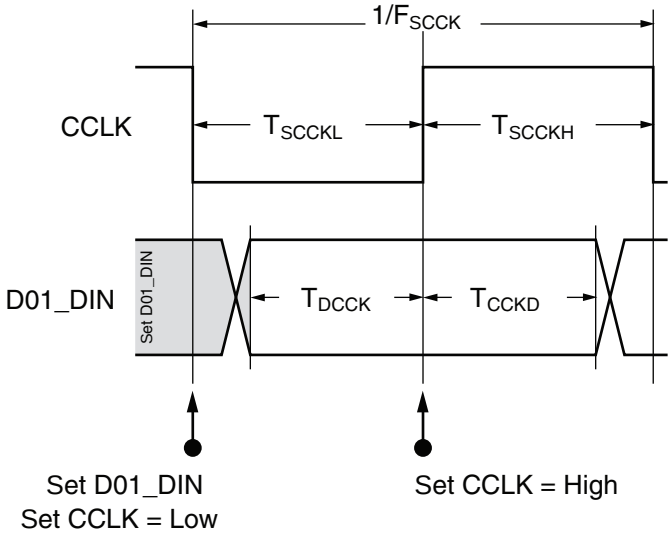
Errors and Troubleshooting

If configuration is not successful, the DONE pin does not go High after all the data is loaded. There are many different reasons why this situation can occur. See Configuration Bitstream Details, page 3 for possible sources of configuration error conditions.

Common debugging methods include:

- Use a Xilinx configuration cable and JTAG configuration mode to verify the ability to configure the FPGA with the bitstream and to read back device status information.
- Verify bit ordering and that bit-swapping is implemented, if necessary. See Sync Word, page 3 for the primary bit order reference value.
- Verify that data sheet configuration timing requirements are met (see Figure 4).
- If DONE does not go High after delivery of the bitstream, check the INIT_B pin.
  - If INIT_B is High:
    - The FPGA might not have recognized the sync word at the beginning of the bitstream. Check for proper delivery of the sync word.
    - The FPGA might not have received the complete bitstream.
  - If INIT_B is Low:
    - The FPGA might have detected an error during configuration. Check that the bitstream is for the correct target device.

Figure 4 shows the configuration timing requirements.



Symbols:

F_{SCCK} = Maximum CCLK frequency
$1/F_{SCCK}$ = Minimum CCLK period
T_{SCCKL} = Minimum CCLK low time
T_{SCCKH} = Minimum CCLK high time
T_{DCCK} = Minimum D01_DIN to CCLK rising-edge setup time
T_{CCKD} = Minimum CCLK rising-edge to D01_DIN hold time

Sample Sequence for a CCLK Cycle:

1. Set CCLK = Low and set D01_DIN = new_value.
2. Delay for Minimum[Maximum[T_{SCCKL}, T_{DCCK}]].
3. Set CCLK = High.
4. Delay for Minimum[Maximum[T_{SCCKH}, T_{CCKD}]].
5. Repeat from step 1.

X583_04_050412

Figure 4: **Configuration Timing Requirements**

Figure 5 shows the serial configuration sequence.

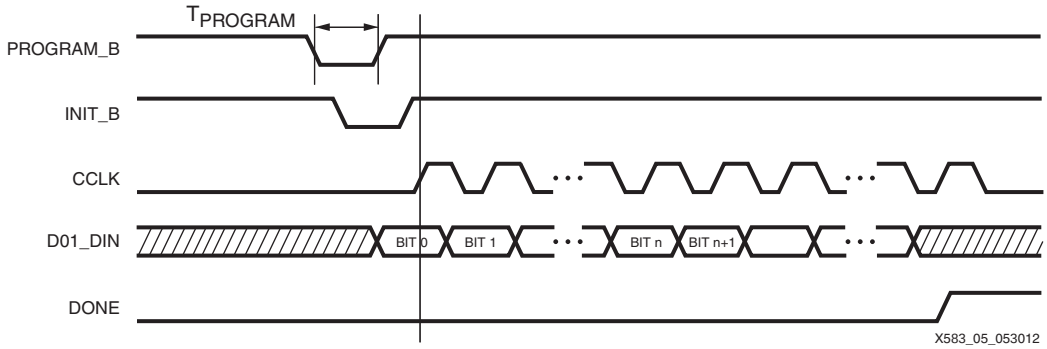


Figure 5: **Serial Configuration Clocking Sequence**

In Figure 5, data is ignored before the sync word and after configuration is complete.

Figure 6 shows the SelectMAP configuration sequence.

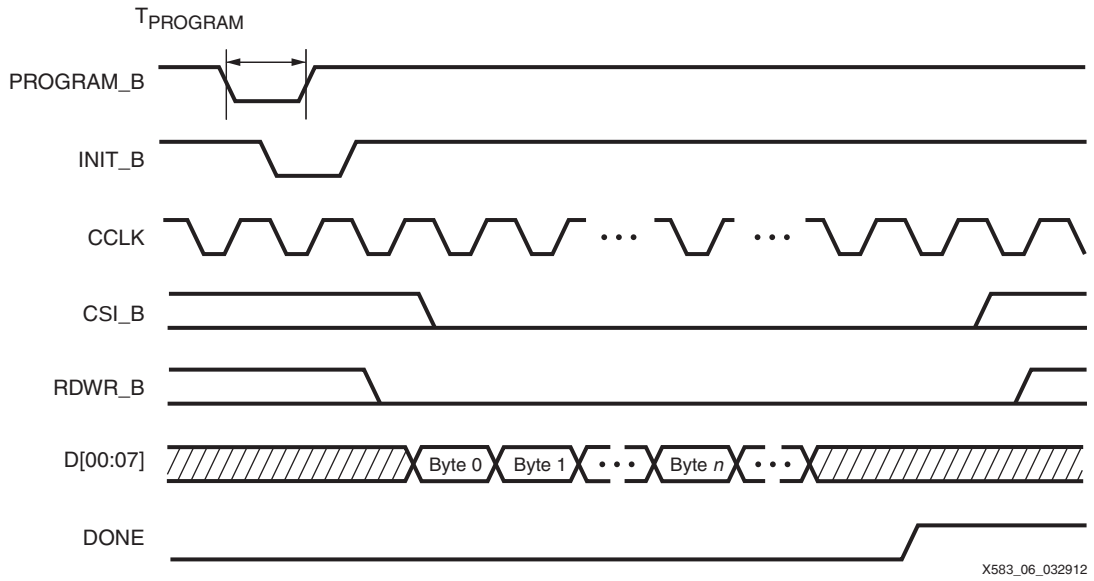


Figure 6: **SelectMAP Configuration Clocking Sequence**

Hardware Implementation

Microprocessor

The reference design is based on the Xilinx MicroBlaze processor. For more information on the MicroBlaze processor, see www.xilinx.com/microblaze.

Voltage Compatibility

The processor I/O needs to support a voltage that is compatible with the connected FPGA pins. The configuration interfaces include the JTAG and dedicated configuration pins in bank 0, and the dual-purpose pins in bank 14. To support the appropriate configuration interface voltage on bank 0 and bank 14, the configuration bank voltage select pin (CFGBVS) must be set to High or Low to set the configuration-related I/O for 3.3V/2.5V or 1.8V operation, respectively. Typically, both banks receive the same V_{CCO} voltage supply to ensure a consistent I/O voltage interface for all of the configuration interface pins. In the Virtex-7 FPGA, bank 14 is a high-performance bank limited to 1.8V or lower I/O standards. The design must meet the voltage restrictions shown in Table 4.

Table 4: **Voltage Compatibility Requirements**

Processor GPIO	FPGA Bank V_{CCO} Requirement		CFGBVS	FPGA Family Support			JTAG[1]
	Bank 0	Bank 14		Artix™-7	Kintex-7	Virtex™-7	
1.8V	1.8V	1.8V	GND	*	*	*	1.8V
2.5V	2.5V	2.5V	V_{CCO_0} (2.5V)	*	*	N/A	2.5V
3.3V	3.3V	3.3V	V_{CCO_0} (3.3V)	*	*	N/A	3.3V

Notes:

1. The JTAG pins are in bank 0. Therefore, the JTAG signal voltage follows the voltage required for bank 0.

Slave SelectMAP Hardware

This section discusses a reference design allowing a microprocessor to configure an FPGA device via the Slave SelectMAP mode. SelectMAP configuration mode is the fastest

configuration option. As shown in Figure 7, this design retrieves the FPGA configuration data stored in external memory and directly configures the target FPGA.

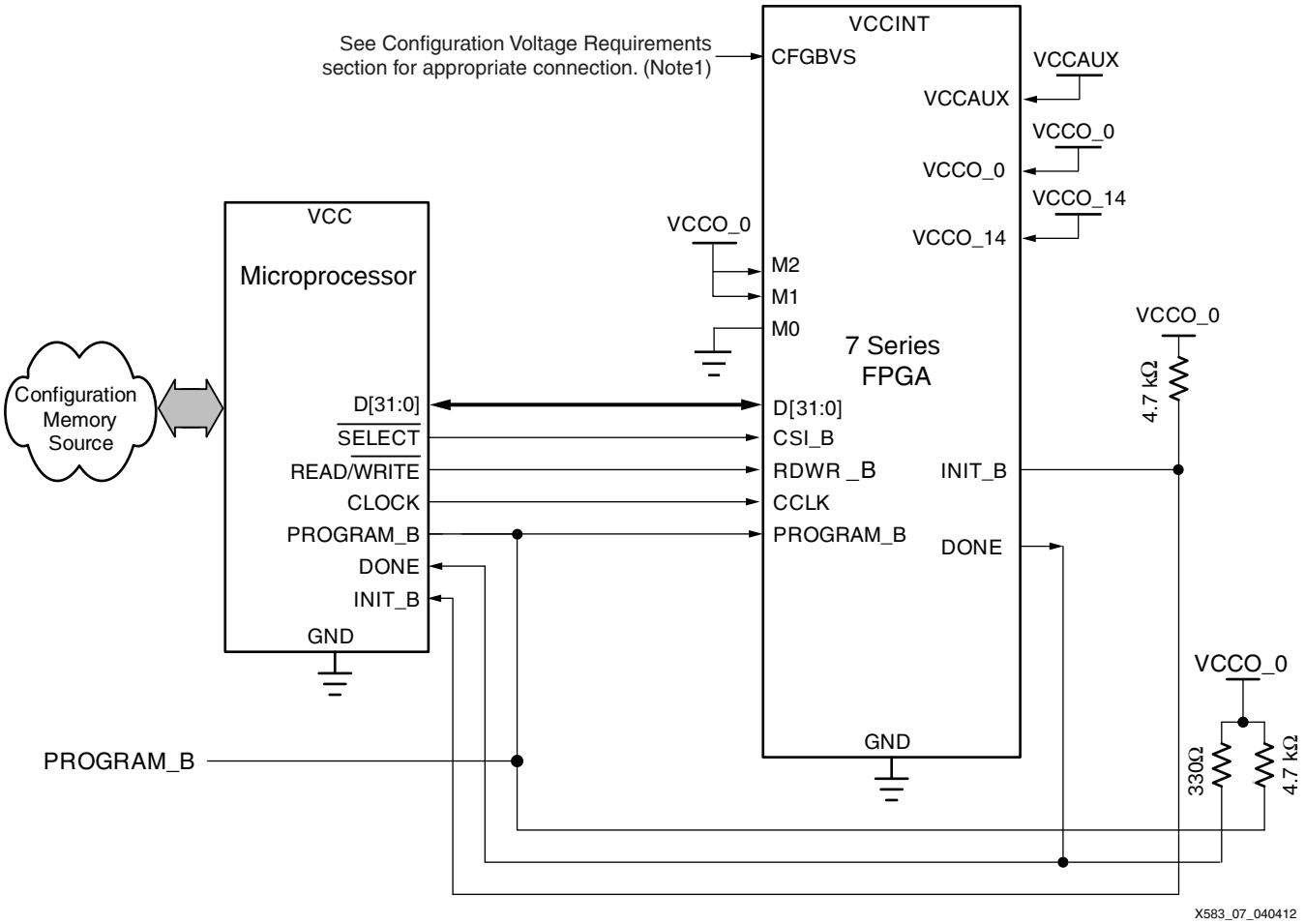


Figure 7: **Slave SelectMAP Configuration**

Notes relevant to Figure 7:

1. Refer to Table 4, page 8 for voltage requirements.

The microcontroller in this reference design reads the FPGA configuration data from memory one 32-bit word at a time. This design uses an 8-bit SelectMAP interface. A simple software program handles sequencing the correct byte onto D[7:0] and asserting CCLK for each byte. It also starts the configuration sequence by asserting PROGRAM_B and checks the DONE and INIT_B signals for status or errors.

Slave SelectMAP Pseudocode

The pseudocode discussed in this section allows a microprocessor to:

- Read FPGA configuration data from memory
- Generate a CCLK
- Deliver byte-wide data to the FPGA

Within the Slave SelectMAP pseudocode, configuration happens in the main() and the shift_word_out() functions. To begin configuration, the CSI_B, RDWR_B and PROGRAM_B pins are set Low. Then the PROGRAM_B pin is asserted and then deasserted after waiting for at least the required PROGRAM_B pulse width. The code then waits for INIT_B to be deasserted if necessary.

Note: See $T_{PROGRAM}$ in the target FPGA data sheet for specific requirements on the PROGRAM_B pulse width. Asserting PROGRAM_B is not required if the FPGA has just completed power-up or is reset through other means besides asserting PROGRAM_B.

After PROGRAM_B and INIT_B are deasserted, shift_word_out() is called for each 32-bit word in the target FPGA bitstream stored in memory. This function deasserts CCLK, sequences the next byte from the current word onto D[7:0], and then asserts CCLK.

After sending the target bitstream, main() waits for the target to assert DONE. To make sure that any special start-up conditions have been met, another eight CCLK assertion and deassertions are sent to the target.

Note: The following pseudocode represents a memory-mapped I/O structure such as in the MicroBlaze processor. While the `slave_selectmap.c` source file in the reference design should be easily portable, the user might need to modify the read and write commands and addresses to match the new system. Also, many systems might have one or two GPIO instances and need to use bit masking to drive the pins independently. The C code in this reference design accesses the I/O pins as individually memory-mapped peripherals and reads and writes from these I/O using pointers. This makes the code simple and portable to a wide range of microprocessors.

```
/* Global defines
 * Define the addresses for the I/O peripherals used to control and
 * monitor the target FPGA. Also define the location in memory the
 * bitstream is stored and its size. These are system dependent and
 * should be adjusted as needed
 */

/* Output GPIO addresses */
CCLK_GPIO_BASEADDR = 0x40020000
PROGRAM_B_GPIO_BASEADDR = 0x40030000
DATA_OUT_GPIO_BASEADDR = 0x40040000
RDRW_B_GPIO_BASEADDR = 0x40050000
CSI_B_GPIO_BASEADDR = 0x40060000

/* Input GPIO addresses */
INIT_B_GPIO_BASEADDR = 0x40070000
DONE_GPIO_BASEADDR = 0x40080000

/* Location in memory and size of the target bitstream */
MEMORY_BASEADDR = 0xC0000000
BITSTREAM_START_ADDR = MEMORY_BASEADDR + 0x2000000
BITSTREAM_SIZE_BYTES = 0xAEA68C

/* PROGRAM_B pulse width. Check the target FPGA data sheet for the
 * TPROGRAM pulse width. One microsecond is safe for any 7 series FPGA
 */
TPROGRAM = 1 /* Assumes sleep() is microseconds */

/* Serialize word and clock each bit on target's DIN and CCLK pins */
shift_word_out(data32)
{
   *cclk = CCLK_GPIO_BASEADDR
   *data_out = DATA_OUT_GPIO_BASEADDR

   /* Sequence the 32-bit word into bytes. The endianess can be either
    * little or big. The following assumes the sync word is read from
    * external memory as 0x665599AA (instead of 0xAA995566).
    */
   byte[0] = data32 >> 24
   byte[1] = data32 >> 16
   byte[2] = data32 >> 8
```

```
   byte[3] = data32

   *cclk = 0

   for (i = 0; i < 4; ++i) {
      *data_out = byte[i]
      shift_cclk(1)
   }
}

/* Assert and Deassert CCLK */
shift_cclk(count)
{
   *cclk = CCLK_GPIO_BASEADDR

   *cclk = 0
   for (i = 0; i < count; --i) {
      *cclk = 1
      *cclk = 0
   }
}

int main()
{
   bits_start = BITSTREAM_START_ADDR
   bits_size = BITSTREAM_SIZE_BYTES

   *program_b = PROGRAM_B_GPIO_BASEADDR
   *rdrw_b = RDRW_B_GPIO_BASEADDR
   *csi_b = CSI_B_GPIO_BASEADDR
   *init_b = INIT_B_GPIO_BASEADDR
   *done = DONE_GPIO_BASEADDR

   /* Bring csi_b, rdwr_b Low and program_b High */
   *program_b = 1
   *rdrw_b = 0
   *csi_b = 0
   /* Configuration Reset */
   *program_b = 0
   sleep(TPROGRAM)
   *program_b = 1

   /* Wait for Device Initialization */
   while(*init_b == 0)
      ;

   /* Configuration (Bitstream) Load */
   for (i = 0; i < bits_size ; i+=4) {
      shift_word_out(bits_start + i)
   }

   /* Check INIT_B */
   if (*init_b_pointer == 0) {
      return 1
   }

   /* Check INIT_B */
   if (*init_b == 0) {
      return 1
   }

   /* Wait for DONE to assert */
```

```
        while(*done == 0)
            ;

        /* Compensate for Special Startup Conditions */
        shift_cclk(8)
        return 0
    }
```

Slave Serial Hardware

This section discusses a reference design allowing a 7 series FPGA to be configured in the Slave Serial mode through a microprocessor. Slave Serial configuration is accomplished by providing a 7 series FPGA with a serial clock and delivering a single data bit at every rising edge of the clock until the final configuration bit is sent. The microprocessor reads the configuration bitstream from external memory. Figure 8 shows the Slave Serial system layout.

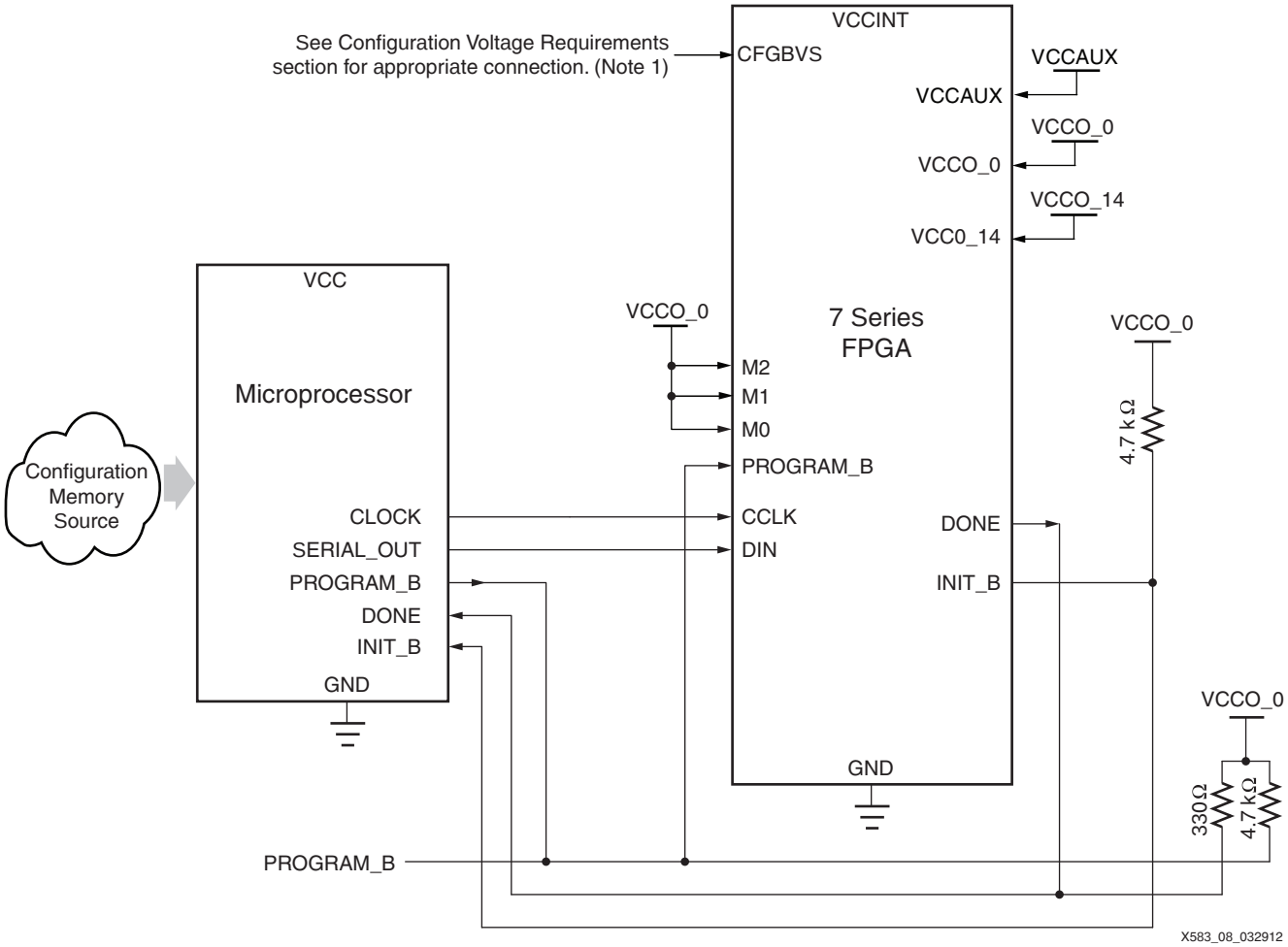


Figure 8: **Slave Serial Mode Configuration Example**

Notes relevant to Figure 8:

1. Refer to Table 4, page 8 for voltage requirements.

Slave Serial Pseudocode

The C code discussed in this section allows a microprocessor to:

- Read FPGA configuration data from memory
- Provide a CCLK
- Serialize the configuration bitstream

Within the `slave_serial.c` source file, configuration happens in the main() and the shift_word_out() functions. To begin configuration, the PROGRAM_B pin is asserted and then deasserted after waiting for at least the required PROGRAM_B pulse width. The code then waits for INIT_B to be deasserted if necessary.

Note: See $T_{PROGRAM}$ in the target FPGA data sheet for specific requirements on the program pulse width. Asserting PROGRAM_B is not required if the FPGA has just completed power-up or is reset through other means besides asserting PROGRAM_B.

After PROGRAM_B and INIT_B are deasserted, shift_word_out() is called for each 32-bit word in the target FPGA bitstream stored in memory. This function deasserts CCLK, serializes the next bits from the current word onto the DIN pin of the target, and then asserts CCLK.

After sending the target bitstream, main() waits for the target to assert DONE. To make sure that any special start-up conditions have been met, another eight CCLK assertions and deassertions are sent to the target.

Note: The following pseudocode represents a memory-mapped I/O structure such as in the MicroBlaze processor. While the `slave_serial.c` source file in the reference design should be easily portable, the user might need to modify the read and write commands and addresses to match the new system. Also, many systems might have one or two GPIO instances and need to use bit masking to drive the pins independently. The C code in this reference design accesses the I/O pins as individually memory-mapped peripherals and reads and writes from these I/O using pointers. This makes the code simple and portable to a wide range of microprocessors.

```
/* Global defines
 * Define the addresses for the I/O peripherals used to control and
 * monitor the target FPGA. Also define the location in memory the
 * bitstream is stored and its size. These are system dependent and
 * should be adjusted as needed
 */

/* Output GPIO addresses */
CCLK_GPIO_BASEADDR = 0x40020000
PROGRAM_B_GPIO_BASEADDR = 0x40030000
SERIAL_OUT_GPIO_BASEADDR = 0x40040000

/* Input GPIO addresses */
INIT_B_GPIO_BASEADDR = 0x40050000
DONE_GPIO_BASEADDR = 0x40060000

/* Location in memory and size of the target bitstream */
MEMORY_BASEADDR = 0xC0000000
BITSTREAM_START_ADDR = MEMORY_BASEADDR + 0x2000000
BITSTREAM_SIZE_BYTES = 0xAEA68C

/* PROGRAM_B pulse width. Check the target FPGA data sheet for the
 * TPROGRAM pulse width. One microsecond is safe for any 7 series FPGA
 */
TPROGRAM = 1 /* Assumes sleep() is microseconds */
/* Serialize a 32-bit word and clock each bit on the target's DIN and
 * CCLK pins */
shift_word_out(data32)
```

```
{
   *cclk = CCLK_GPIO_BASEADDR
   *serial_out = SERIAL_OUT_GPIO_BASEADDR
   *cclk = 0
   *serial_out = 0
   for (i = 31; i >= 0; --i){
      *serial_out = (data32 & 1 << i) ? 1 : 0
      shift_cclk(1)
   }
}

/* Assert and Deassert CCLK */
shift_cclk(count)
{
   *cclk = CCLK_GPIO_BASEADDR

   *cclk = 0
   for (i = 0; i < count; --i) {
      *cclk = 1
      *cclk = 0
   }
}

int main()
{
   bits_start = BITSTREAM_START_ADDR
   bits_size = BITSTREAM_SIZE_BYTES

   *program_b = PROGRAM_B_GPIO_BASEADDR
   *init_b = INIT_B_GPIO_BASEADDR
   *done = DONE_GPIO_BASEADDR

   /* Configuration Reset */
   *program_b = 0
   sleep(TPROGRAM)
   *program_b = 1

   /* Wait for Device Initialization */
   while(*init_b == 0)
      ;

   /* Configuration (Bitstream) Load */
   for (i = 0; i < bits_size; i+=4) {
      shift_word_out(bits_start + i)
   }

   /* Check INIT_B */
   if (*init_b_pointer == 0) {
      return 1
   }

   /* Wait for DONE to assert */
   while(*done == 0)
      ;

   /* Compensate for Special Startup Conditions */
   shift_cclk(8)
   return 0
}
```

Reference Design Files

The reference design files for this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=188189

Table 5 shows the reference design checklist.

Table 5: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Matt Nielson
Target devices (stepping level, ES, production, speed grades)	7 series FPGAs
Source code provided	Yes
Source code format	C
Design uses code and IP from existing Xilinx application note and reference designs, CORE Generator software, or third party	No
Simulation	
Functional simulation performed	No
Timing simulation performed	No
Test bench used for functional and timing simulations	No
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	No
Implementation	
Synthesis software tools/version used	N/A
Implementation software tools/versions used	mb-gcc (GCC) 4.1.2 20070214 (Xilinx 13.4)
Static timing analysis performed	No
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	SP605 board, XM105 debug card, Xilinx internal Kintex-7 FPGA testing platform

Conclusion

This application note provides background on configuration as well as a description of two complete sets of reference designs allowing a Xilinx FPGA to be configured through Slave SelectMAP or Slave Serial mode. Although the microprocessor C code targets a Xilinx MicroBlaze processor, it was written with portability in mind. Porting the code to another processor requires some effort, but all the design files are documented extensively.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
05/31/2012	1.0	Initial Xilinx release.

Notice of Disclaimer

Automotive Applications Disclaimer