Configuring Spartan-II FPGAs from Parallel EPROMs

XAPP178 (v0.9) December 3, 1999 Advance Application Note

Summary

This application note describes a simple CPLD-based interface design to configure a Spartan™-II device from a parallel EPROM using the Slave Parallel configuration mode.

Introduction

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

Those users interested in finding ways to configure Spartan Series devices without the use of a dedicated PROM should refer to Application Note XAPP098 "The Low Cost Efficient Configuration of Spartan FPGAs".

Configuring Spartan-II FPGAs through the Slave Parallel mode is eight times faster than bit-serial mode. The Slave Parallel configuration mode, otherwise known as SelectMAP, utilizes an 8-bit configuration bus and seven control signals for synchronization and handshaking of data.

Combining larger memory capacities with an 8-bit interface results in faster configuration that requires fewer storage elements and speeds up system initialization. This is made possible by the addition of a small interface design to provide addresses for the parallel PROM and the necessary control signals for Slave Parallel configuration.

Interface Design

To configure the Spartan-II FPGA from a parallel PROM, a small interface is needed to generate the PROM addresses and load the data into the FPGA. Any one of several technologies might be used, and a Xilinx CPLD is a simple and cost-effective implementation, as shown in Figure 1.

Voltage Supplies and Pullups

The Spartan-II FPGA was designed for mixed-voltage environments. The internal voltage (V_{CCINT}) requires a 2.5V supply. During configuration, the I/O are automatically set to the LVTTL standard which requires a V_{CCO} of 3.3V. If this is incompatible with the I/O standard to be used after configuration, V_{CCO} may be driven for the desired standard during configuration with the configuration pins (DONE, INIT, and BUSY) up to V_{PU}. If the V_{CCO} is left floating, then V_{CCO_2} must have a pullup also. The V_{PU} refers to the operating voltage of the CPLD and PROM. Most likely this will either be 3.3V or 5.0V. Either value is acceptable to the FPGA. No translation is needed. The recommended pullup value is 4.7K. This value is not critical. Smaller pullup resistors will of course increase current draw when that node is driven to a logic Low.

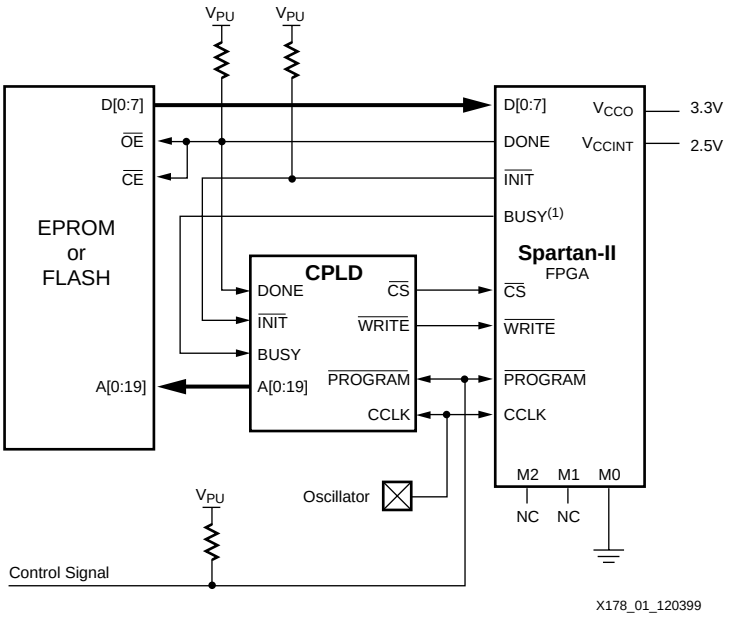

Figure 1: Slave Parallel and EPROM Interface Circuit Diagram

Note:

1. BUSY is only needed when the oscillator is running at a frequency greater than 50 MHz.

Slave Parallel Mode

Slave Parallel mode uses a byte-wide bidirectional access port for reading and writing the configuration data of a Spartan-II FPGA. It is very similar to the Express mode used by Spartan-XL FPGAs. The following signals are used in Slave Parallel mode.

D[7:0]

D[7:0] comprise the 8-bit bidirectional data bus.

CCLK

CCLK is the configuration clock input signal used by the internal configuration logic.

$\overline{\text{PROGRAM}}$

The $\overline{\text{PROGRAM}}$ input signal resets the internal configuration logic and re-initializes the internal configuration memory.

DONE

The DONE output indicates the completion of configuration and the beginning of the Startup sequence.

INIT

The bidirectional open-drain $\overline{\text{INIT}}$ pin is used to hold off configuration initialization, and it indicates when CRC errors occur in the configuration data.

$\overline{\text{WRITE}}$

The $\overline{\text{WRITE}}$ input is a write strobe that must be asserted and held throughout the loading of data.

BUSY

When $\overline{CS}$ is asserted, BUSY output indicates whether the current byte is being loaded or ignored. When $\overline{CS}$ is not asserted, BUSY is disabled and pulled High.

$\overline{CS}$

The $\overline{CS}$ input is the Chip Select signal used to enable the FPGA's configuration interface when the Slave Parallel mode is selected.

M2, M1, and M0

The MODE pins M[2:0] must be set to select the desired configuration mode. For Slave Parallel configuration these lines must be driven High, High, and Low, respectively.

Configuration Process Steps

The following are the steps for performing a Slave Parallel configuration.

Reset the Configuration Logic

If configuration follows power-up, or a recycling of the power source, then the configuration logic will automatically be initialized. However, if the FPGA is to be reconfigured without the recycling of power, then the $\overline{PROGRAM}$ input must be asserted Low for at least 300 ns to reset the configuration logic.

Time-out Start of Configuration

After the release of the $\overline{PROGRAM}$ input, or the powering up of the FPGA, the $\overline{INIT}$ output stays Low while the internal configuration memory is automatically cleared.

Unlike previous generations of FPGAs, the Spartan-II device does not require an additional time-out period following initialization. Configuration may begin as soon as $\overline{INIT}$ has gone High. The Spartan-II FPGA uses the first three clock cycles after $\overline{INIT}$ has gone High to initialize the configuration circuitry; however it is padded with eight dummy bytes at the beginning of the data stream to account for this.

Loading Configuration Data

To begin configuration, the $\overline{WRITE}$ and $\overline{CS}$ inputs must be asserted Low and held. Each byte is loaded on the rising edge of CCLK. If BUSY was Low during the CCLK transition then the byte was accepted. If it was High then the byte was ignored and must be reloaded on the next clock cycle. BUSY is a synchronous signal. Therefore, CCLK can not be suspended until BUSY is de-asserted.

For configuration clocks speeds below 50 MHz, BUSY is guaranteed to be inactive, and thus may be ignored entirely, and removed from the interface design.

End of Configuration

The DONE output transitions High to indicate the end of configuration and the beginning of the startup sequence. During the startup sequence the D[7:0], $\overline{WRITE}$, BUSY, $\overline{INIT}$, and $\overline{CS}$ pins all become user I/O. If any of these pins are used by the configured design, then any driving source used only for configuration purposes must be disabled so as not to contend.

Depending on the startup options selected in the BitGen configuration option template, part of the Xilinx development software, completing the startup phase will require as many as eight CCLK cycles after DONE has transitioned High. However, since the example shown below uses a free-running oscillator, this is not a concern. The Spartan-II FPGA will complete the startup phase even if the $\overline{CS}$ signal is de-asserted.

The PROMMAP Design

The CPLD design PROMMAP, shown in Figure 2, consists of an address counter, a write control register, some tristate buffers and random logic. The PROGRAM input acts as a global reset. DONE acts as a global tristate. INIT and BUSY hold off the address counter from incrementing while the FPGA initializes or processes data.

Address Counter

The address counter needs to be large enough to accommodate the address bus width of the PROM. In Figure 1 a 1-Mbyte (8-Mbit) PROM is shown which requires a 20-bit address bus. For connecting multiple PROMs see "Interfacing Multiple PROMs" on page 5.

Write Control Register

The CS and WRITE input signals to the FPGA must be asserted and held Low for configuration. If only one FPGA is being targeted for configuration, these two signals may be derived from the same source. For targeting multiple FPGAs see "Multiple FPGAs in a Parallel Chain" on page 6.

Configuration must be delayed until the INIT signal has transitioned High indicating that FPGA initialization has completed.

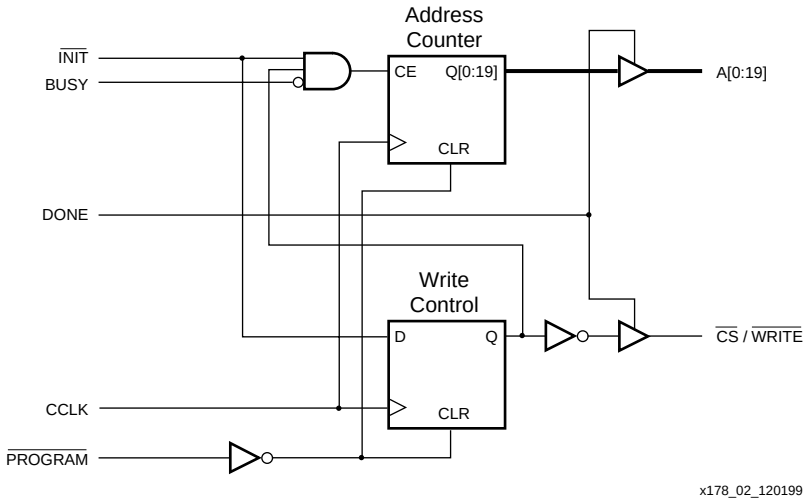

Figure 2: PROMMAP Interface Design

Oscillator

The configuration logic of the FPGA was designed to work with a free-running oscillator. In this design the maximum frequency for the oscillator is constrained by the access time of the PROM (T_{ACC}) plus the setup time for the Slave Parallel data inputs (T_{SMDCC}).

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

PROMMAP Design Files

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

Interfacing Multiple PROMs

If the needed memory capacity requires the use of multiple PROMs then the interface design can be altered to accommodate PROM daisy-chaining as shown in Figure 3.

The output enable ($\overline{OE}$) input pins of the PROMs can be driven by an address decode of the most significant bits of the address counter in the PROMMAP design as shown in Figure 4.

For the example shown in Figure 3, the 8-MBit PROM has been replaced with two 4-MBit PROMs. The common address bus is now only 19 bits wide and the MSB A(19) is used to select between the PROMs, see Table 1.

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

Table 1: Address Decode for Two Prom Selection

A(19)	$\overline{OE0}$	$\overline{OE1}$
0	0	1
1	1	0

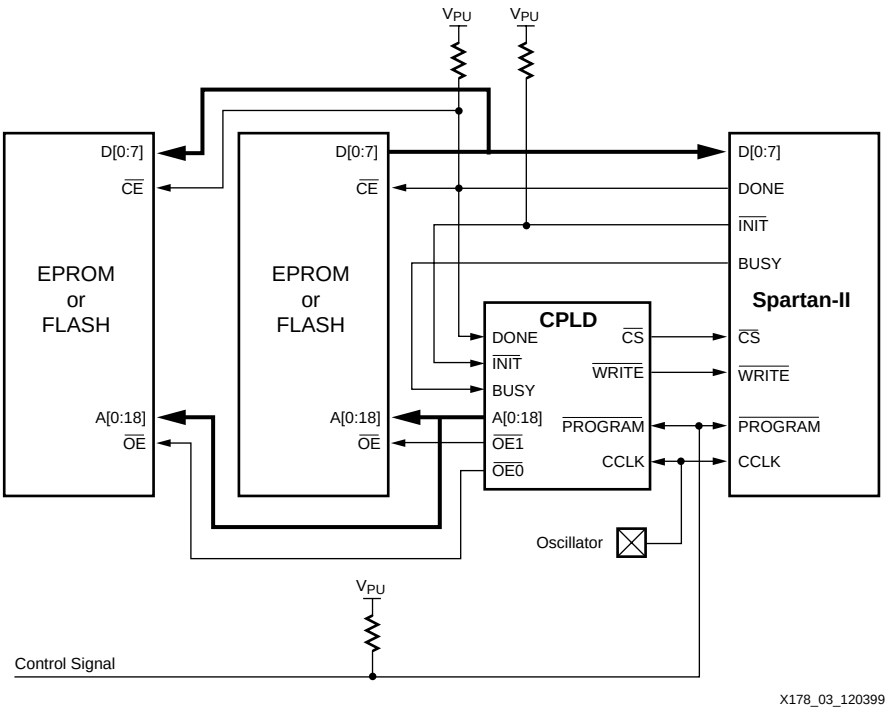

Figure 3: Connecting Mutiple PROMs in Daisy-chain

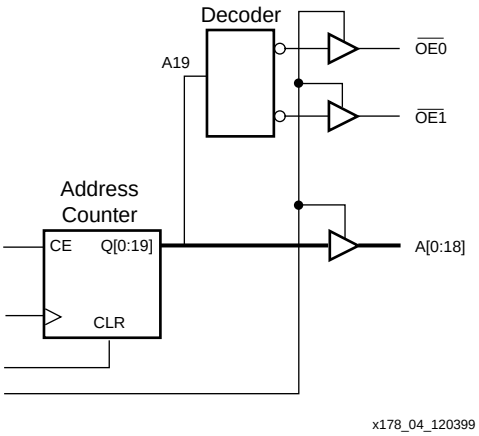

Figure 4: Selecting Multiple PROMs

Multiple FPGAs in a Parallel Chain

Serial Parallel configuration does not support a standard daisy-chain configuration like the bit-serial modes do. However, in an application such as this, where there are multiple FPGAs to be configured, the PromMAP design may be altered to accommodate a parallel chain so that multiple interface chips are not necessary. As shown in Figure 5, the parallel-chaining of multiple FPGAs is similar to cascading multiple PROMs.

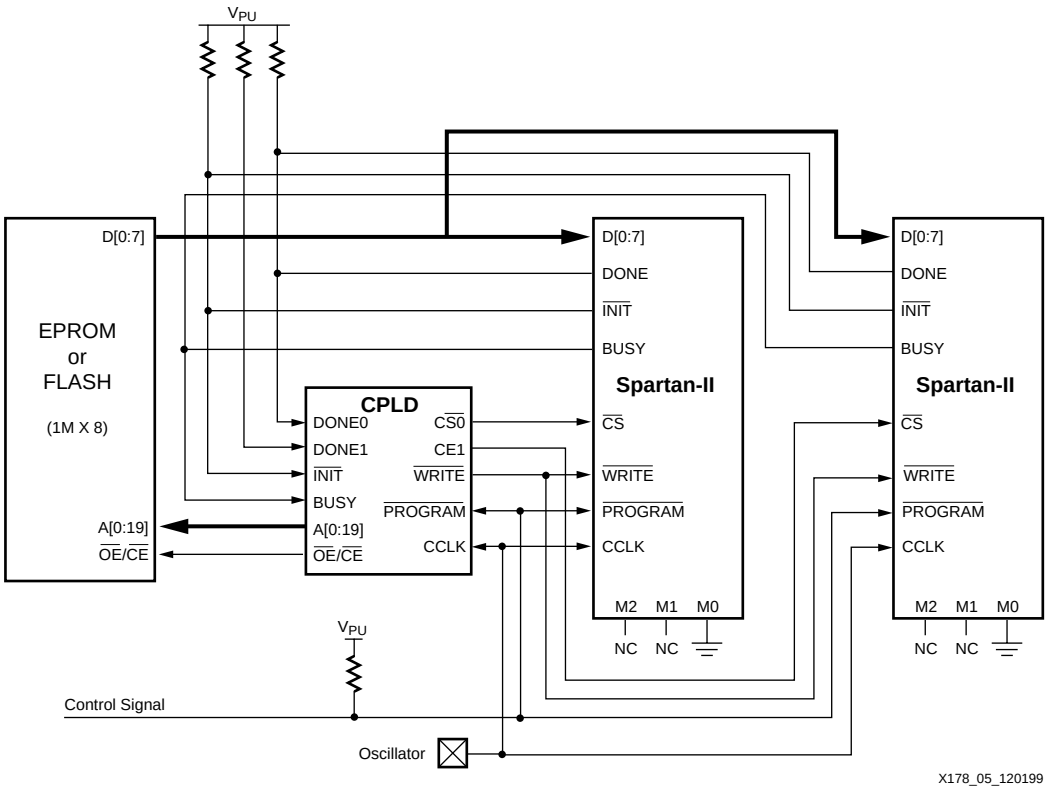

Figure 5: Connecting Multiple FPGAs for Parallel Configuration Chain

For each FPGA the DONE and $\overline{CS}$ signals must be connected to the interface design separately. The BUSY and $\overline{WRITE}$ signals can be a common line between the FPGAs. When the first FPGA raises its DONE signal the interface must disable that FPGA's $\overline{CS}$ input and

enable the next one in the chain, and so on until all DONEs are high. An example of this is shown in Figure 6.

Using this method will cause each FPGA to become active after its DONE transitions High. Since the FPGAs will not go active simultaneously, it is important that none of the configuration pins used for Slave Parallel configuration be used as I/O in the FPGA designs. Otherwise contention could result when one or more FPGAs are configured and active, but others are not yet configured.

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

Figure 6 shows how the $\overline{\text{CS}}$ outputs are cascaded accordingly with the DONE inputs. In the single FPGA example the DONE input is the global tristate control for all outputs. For multiple FPGAs, the global tristate signal (DONE_ALL) is generated by ANDing together all the DONE lines from the separate FPGAs.

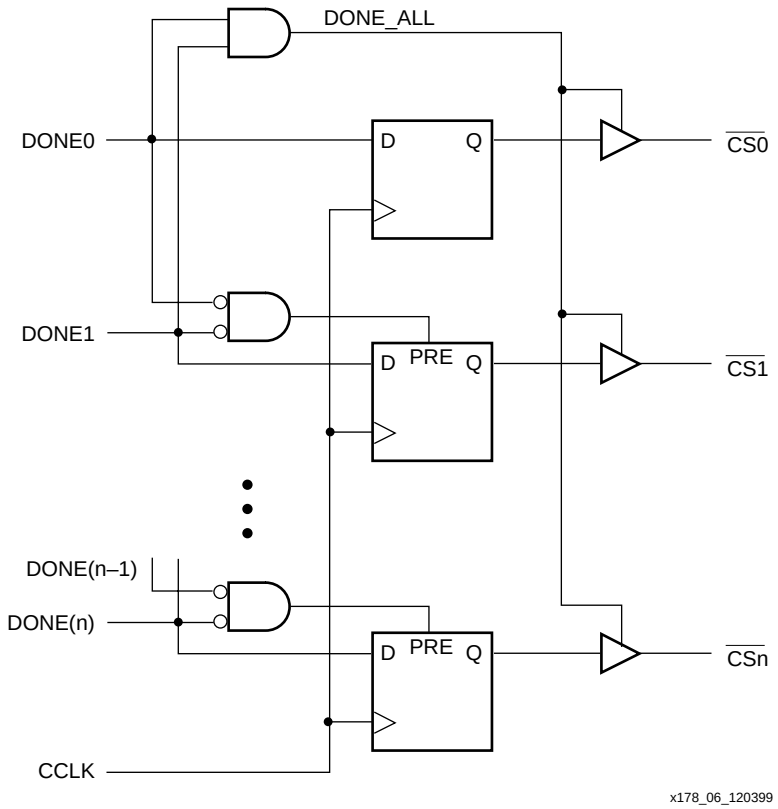

Figure 6: Sequencing FPGA Chip Selects

The dotted extension in Figure 6 demonstrates that more FPGAs may be added to the chain by adding chip select registers. For each segment, the register is held asynchronously PREset when the DONE (n-1) and the associated DONE (n) are both Low. When DONE (n-1) transitions High, the PREset is released and the register loads the value of the associated DONE (n) signal on each clock cycle. When the associated DONE transitions High, the corresponding $\overline{\text{CS}}$ output will transition High disabling the configuration for that FPGA in the chain. This removes the PREset of the next segment and allows configuration of the next FPGA in the chain. The first segment does not make use of a PREset. When all DONE signals are High all outputs will disable.

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

Synchronizing Startup

An alternative method to activating the FPGAs sequentially would be to synchronize the startup of all the FPGAs by combining all the DONE lines into one signal. Spartan-II FPGAs automatically synchronize the startup sequence to wait for DONE to be externally released. It is, however, important that the BitGen option to "DriveDone" is deselected. Unless otherwise specified, the DONE pin will be an open-drain output.

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

In Figure 7, decoding logic PREsets the $\overline{\text{CS}}$ control register at Address 0 (set all $\overline{\text{CS}}$ registers High in the beginning) and toggles the register at the starting address and ending address for the desired configuration data. Set the starting address of the first register to 1h. Otherwise, the register will not toggle. The decode of the starting address for one stage can also serve as the ending address decode of the previous stage.

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

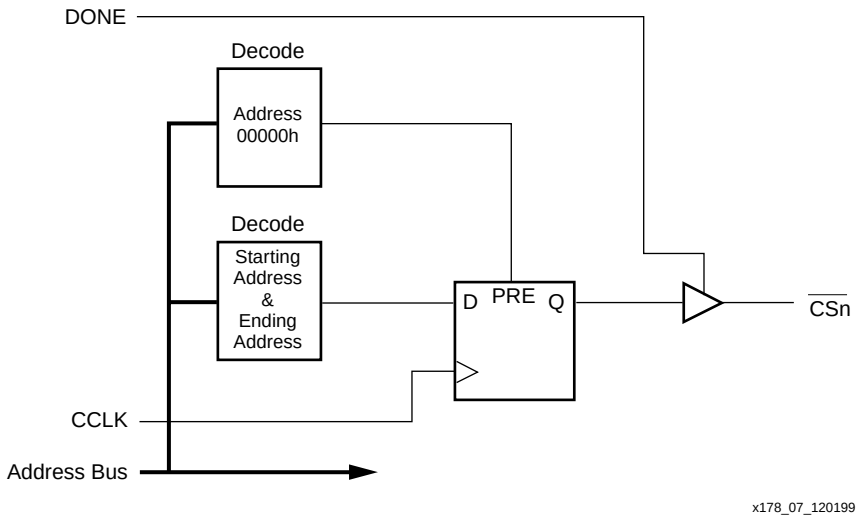

Figure 7: $\overline{\text{CS}}$ Address Decoding

Cascading PROM Files

The PROM file formatter (PROMGen), part of the Xilinx development software, must be run on each FPGA design separately, and the resulting PROM files must be concatenated . Care must be taken to assure that the PROM files are in the desired order. Do not use PROMGen to create a single PROM file as would be done for bit-serial daisy-chaining. This would result in an incorrect bit stream for this configuration scheme.

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

If your programmer can use HEX files, then a simple concatenation of hex files is all that is required. The PromGen utility has a switch option to create HEX files.

If your PROM Programmer's software has edit ability, then it may allow you to read in each of the PROM files individually and specify which address to load up from.

If the programmer requires a single PROM file in addressed format (mcs, tek, exo), then some manual editing may be required. It should be noted that a chain of Spartan-II PROM files will typically be too big for the MCS and TEK formats. EXO is usually the better choice, and easier to work with.

To concatenate EXO files, first generate the EXO for the first bit file in the chain with PromGen loading up from address 0h. View the EXO file. The very last line should be deleted. The second to last line should also be deleted, but first note the starting address for that line. For example, the last two lines might read:

```
S20803576000000000 3D
S804000000FB
```

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

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

Acknowledgements

Original material by Carl Carmichael.

Revised for the Spartan-II family by Kim Goldblatt

Revision Table

Date	Version #	Revision
12/03/99	0.9	Advance Information Release.