XAPP1328 (v1.0) March 5, 2018

Configuration or Boot with NOR Flash for Sourcing Flexibility and Cost Reduction

Author: Eric Crabill

Summary

Xilinx® devices are interoperable with commodity NOR flash for configuration or boot. This agility is useful for sourcing flexibility and cost reduction, but a system designer must approach the use of NOR flash with a design-for-substitution mindset. The recommendations in this application note for configuration or boot solutions provide solutions for common NOR flash sourcing challenges, as well as open opportunities for cost reduction.

Introduction

Xilinx devices are SRAM-based and require loading of programming information at least once after power-on. This programing information can be loaded through a variety of methods, including passive methods where the device awaits programming information to be served, and active methods where the device autonomously retrieves programming information. The most common active methods interface NOR flash to the Xilinx device for non-volatile storage of programming information that the Xilinx device uses to automatically configure or boot.

In systems designed with Xilinx devices where NOR flash is used for configuration or boot, there are numerous factors that can influence the NOR flash selection process. For example, a system design with size constraints can drive the selection of NOR flash in aggressive packaging to minimize board area. Similarly, a system design with power-on latency constraints can drive the selection of NOR flash with increased bandwidth to minimize configuration or boot time. In such cases, the resulting selections can meet system design requirements, but can also impact NOR flash sourcing flexibility and cost.

The majority of system designs using Xilinx devices with NOR flash do not have constraints that impact the configuration or boot solution. For those unconstrained system designs, for system designs anticipating an extended lifetime, and for highly cost-sensitive system designs, Xilinx recommends a design-for-substitution mindset in the use of NOR flash. The goals of this mindset are:

- Enable sourcing flexibility (primary)
- Enable cost reduction (secondary)

For a better understanding of how this mindset can help system designers deliver on these goals, a quick review of several relevant dynamics is helpful.

Relevant Dynamics

System Lifetime

How long is the system going to be in production? Is there a need for extended maintenance of deployed systems after production ends? Consider an avionic flight controller where production and extended maintenance can require enduring several NOR flash sourcing transitions.

System Cost

Is the system cost sensitive where the NOR flash pricing significantly impacts the project's profit margin? Are there planned or opportunistic cost reduction possibilities? A commercial entertainment device could require a cost reduction with the flexibility of form, fit, and function replacements of NOR flash, based on the lowest available price at the time of purchase.

Interface

Is there any interfacing trend that could influence the NOR flash selection? Historically this has not been the case, however, based on NOR flash market data, serial interfaces are preferred over parallel interfaces. As a result, serial NOR flash is likely to be available from more sources and for a longer time than parallel NOR flash.

Supply Constraint

Is there a current, or future risk of a supply constraint that could increase the need for sourcing flexibility over the system lifetime? Commodities such as NOR flash, NAND flash, SRAM, and DRAM experience swings in availability and pricing as a result of market forces. Sourcing flexibility helps manage this risk.

Discontinuance or Vendor Exit

Is there a current, or future risk that the chosen NOR flash could be discontinued, or the NOR flash vendor exits the market? NOR flash vendors can provide assured product lifetimes, yet these are presumably contingent on the vendor's continued independence and financial health.

Design-for-Substitution

Application of the design-for-substitution mindset in the use of NOR flash has a light touch on several areas of design and manufacturing of a system. The following sections discuss these areas and provide recommendations on how to design for substitution. With these recommendations, system designers can gauge their needs in the context of their risk tolerance and relevant dynamics to make informed choices while maximizing sourcing flexibility and cost reduction within the available degrees of freedom.

Provisioning – Board and NOR Flash Programming

For a configuration or boot solution based on NOR flash, provisioning involves providing a place on the printed circuit board to receive the NOR flash and supplying the necessary connectivity to program it. Before this can be planned in detail, a critical choice must be made, the selection of serial or parallel interface.

It is reasonable to anticipate a continuing reduction of parallel NOR flash options in the market. While there are systems with requirements that suggest the use of parallel NOR flash, Xilinx has worked to eliminate the need for parallel NOR flash in many cases by providing alternative solutions:

- Integrated PCI Express® (PCIe) peripherals enable use of serial NOR flash for boot of many Zynq® UltraScale+™ MPSoCs and Zynq-7000 All Programmable (AP) SoCs in applications with PCIe interfaces used in open systems.
- Tandem configuration enables use of serial NOR flash for configuration of many UltraScale™, UltraScale+, and 7 series FPGAs in applications with PCIe interfaces used in open systems.
- The dual Quad-SPI configuration method available in UltraScale and UltraScale+ FPGAs enables bandwidth higher than any parallel NOR flash available for use in new designs.

Given the interface dynamics in the NOR flash market and the alternative solutions from Xilinx, parallel NOR flash is best considered a single-source component and therefore, not appropriate to approach with a design-for-substitution mindset. A single-source component is typically a source of exceptional value or differentiation in a system design. It brings benefits to the system design outweighing other considerations but requires different planning than multiple-source commodity components. This planning should include collaboration with the vendor to understand the component lifetime and end-of-life options including last-time purchasing opportunities or inventory banking.

Based on these considerations, Xilinx recommends serial NOR flash for configuration or boot of Xilinx devices unless there is a clear and compelling reason to use parallel NOR flash. Selection of serial NOR flash is required for design-for-substitution and the recommendations made in this application note assume the use of serial NOR flash.

Serial NOR Flash Packaging and PCB Footprint Design

A wide range of packaging options can be found across multiple vendors of serial NOR flash. Use packaging options that are available across multiple vendors such as SO16W, SO8W, and BGA24. Fortunately, vendors have gravitated toward pin-compatibility in these common packages. Avoid using unique or vendor-specific packaging options such as WLCSP to reduce the creation of an artificial sole-source condition.

To substantially increase flexibility, it is possible to implement a serial NOR flash PCB footprint that accepts at least three different packages. Based on a survey of packaging options versus serial NOR flash density across multiple vendors, two trends in compatible packaging emerge:

- For 64 Mb and smaller: SO16W (300 mil), SO8W (208 mil), MLP8 (6×5 mm)
- For 128 Mb and larger: SO16W (300 mil), BGA24 (5×5), BGA24 (4×6)

Figure 1 shows the 64 Mb and smaller case, where composite MLP8 (6×5 mm) and SO8W (208 mil) landings fit inside the larger SO16W (300 mil) landings. The SO16W can be omitted, with corresponding reduction in flexibility, if space is at a premium.

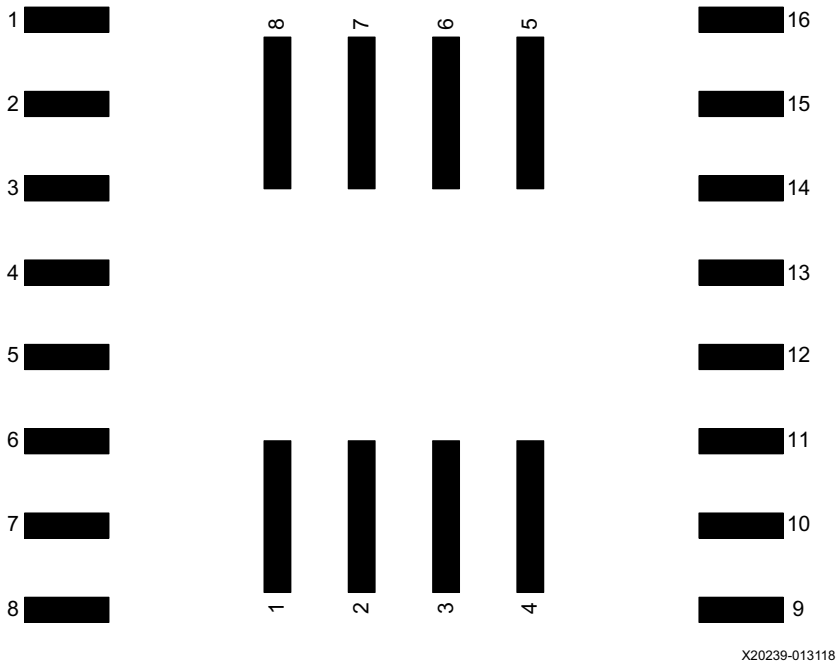

Figure 1: **Composite MLP8 (6x5 mm), SO8W (208 mil), and SO16W (300 mil)**

Figure 2 shows the 128 Mb and larger case, where composite BGA24 (4×6) and BGA24 (5×5) landings fit inside the larger SO16W (300 mil) landings. As with the smaller case, the SO16W can be omitted. Both BGA24 packages have the same relative pin assignments, making it possible to place either on a composite 5×6 array.

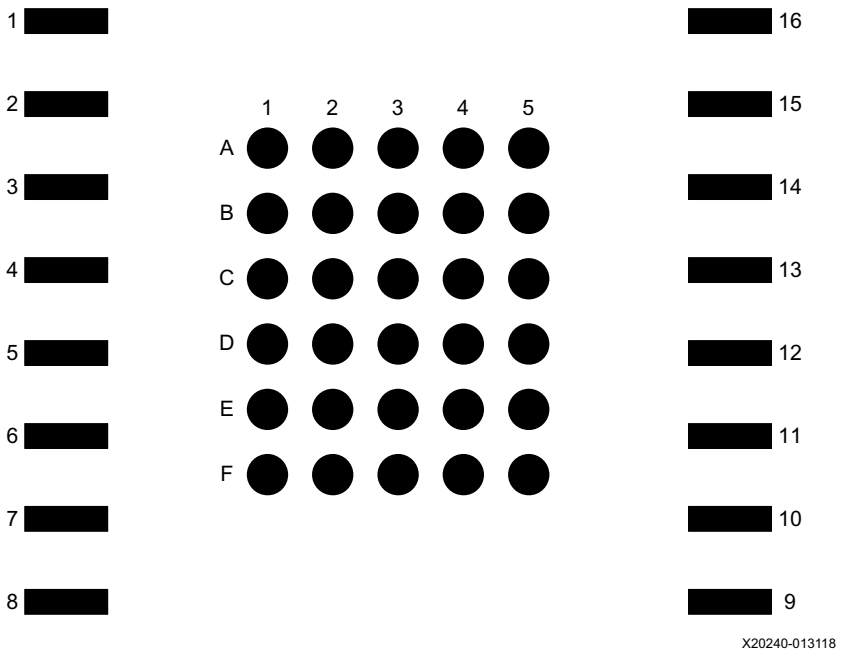

Figure 2: **Composite BGA24 (4x6), BGA24 (5x5), and SO16W (300 mil)**

Serial NOR flash are available with support for a wide range of voltages. Take particular note of power supplies that can be required for the I/O and core power of serial NOR flash. Most serial NOR flash use a single V_{CC} that is shared for I/O and core power, with the required voltage based on the I/O switching threshold—often 3.3V or 1.8V. However, a few serial NOR flash use separate V_{CC} and V_{IO} for core and I/O power, respectively. On devices requiring separate V_{IO}, it usually takes the place of what would be an N/C pin on other serial NOR flash.

Independent of single or composite footprint, to enable the flexible use of devices with separate V_{IO} such as certain members of the Cypress Semiconductor S25FL-S family, the board design must provide flexibility to make or break the V_{IO} connection. This is easily accomplished by connecting the V_{IO} landing(s) to the desired V_{IO} supply rail through a 0Ω resistor. For serial NOR flash that require V_{IO}, this resistor must be assembled onto the board, but for all other cases it is omitted.

Serial NOR flash are available with a variety of supplemental pin functions beyond fundamental clock, select, and data lines. These functions include hold, write protect, and reset. Many vendors provide ordering codes to allow customers control over the supplemental functions that exist on the serial NOR flash they purchase. Hold and write protect functions are generally active-Low and shared on the two highest order data pins, which is one reason board-level pull-up resistors are desirable on these signals.

Reset functions are generally active-Low. With Zynq UltraScale+ MPSoCs, UltraScale and UltraScale+ FPGAs, Zynq-7000 AP SoCs, and 7 series FPGAs, any dedicated reset function provided by the serial NOR flash should not be connected to the Xilinx device but instead to a board-level pull-up resistor. This resistor can be assembled onto the board as needed depending on the requirement of the serial NOR flash.

Programming Methods Overview

There are a variety of methods to program serial NOR flash. For development, debug, prototyping, and validation purposes, in-system programming has become a necessary capability to minimize the need for sockets or board re-work. In production, certain types of in-system programming can be used, but most production runs leverage pre-programmed serial NOR flash that are then assembled onto the board directly.

Figure 3 illustrates the concept and connections to support in-system programming of serial NOR flash. The figure shows indirect programming (through or using the Xilinx device) as well as direct programming (bypassing or around the Xilinx device).

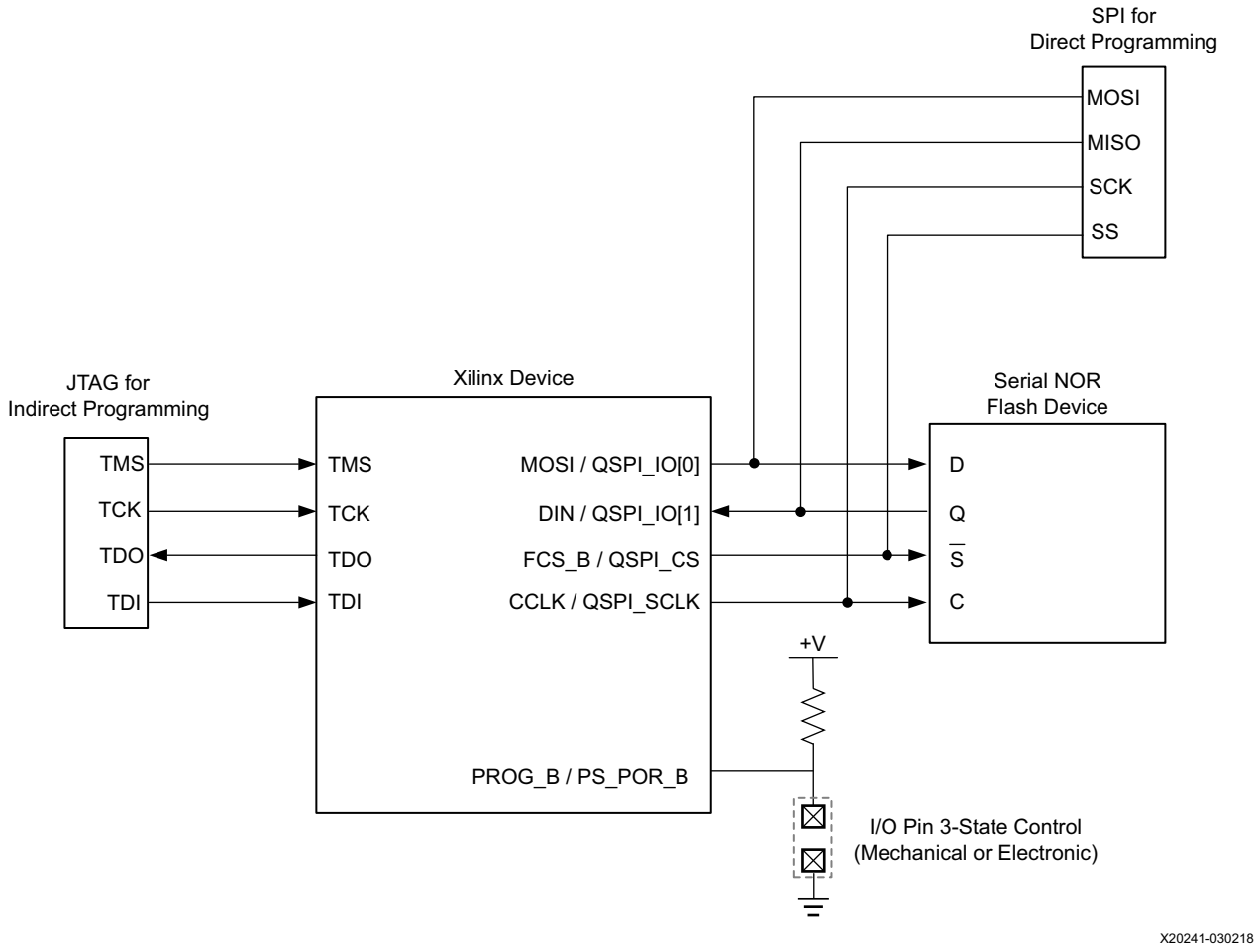

Figure 3: **Indirect and Direct In-System Programming of Xilinx Devices**

It is best to provide provisions for both indirect and direct programming because this greatly increases sourcing flexibility and cost reduction opportunities. In every case, board-level signal integrity is important. System designers and board designers must collaborate to route JTAG and SPI traces carefully, paying particular attention to clock (SCK, TCK) routing topology, stub minimization, and the different drivers on SPI lines. Board-level modeling with IBIS is recommended. Extracted signal delays are valuable in timing analysis of the JTAG and SPI interfaces to ensure they are not operated beyond their limits.

Indirect Programming – Through or Using a Xilinx Device

The prevalent method of indirect programming connects a JTAG cable as shown in Figure 3, with the Vivado® Design Suite controlling the JTAG interface. The Vivado Design Suite configures or boots the Xilinx device to implement a JTAG-to-SPI bridge. Afterwards, the Vivado Design Suite sends serial NOR flash programming data through the Xilinx device to the serial NOR flash.

Depending on the Xilinx device, the bridge can be realized in hardware, software, or a combination of both. For UltraScale, UltraScale+, and 7 series FPGAs, a proprietary bridge is downloaded to the programmable logic. Xilinx pre-compiles and verifies an implementation of the bridge for each member of these FPGA families and integrates the results into the Vivado Design Suite. For Zynq UltraScale+ MPSoCs and Zynq-7000 AP SoCs, a bridge based on U-Boot is downloaded to the processing system. U-Boot is an open source universal boot loader that is frequently used in embedded development.

This indirect programming method requires only a JTAG cable and the Vivado Design Suite. Most system designs implement JTAG for test and debug purposes, therefore, the incremental board cost is zero. A compelling benefit is the exceptional ease of use during development, debug, prototyping, and validation. Note that the Vivado Design Suite is not a universal device programmer. For the current list of devices the Vivado Design Suite is capable of programming, see the *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1]. Also refer to section 4(a) of the Xilinx End User License Agreement [Ref 2] for important information regarding use of the Vivado Design Suite in production programming.

Another method of indirect programming is to create designs capable of field update. Initially, the design is programmed using a JTAG cable as shown in Figure 3, but the design contains a second interface suitable for receiving programming information. When prompted to perform a field update, serial NOR flash programming data is received over this interface and subsequently written into the serial NOR flash by the design.

This indirect programming method requires a JTAG cable and the Vivado Design Suite, plus a system design with sufficient functionality implemented to support field update of the chosen serial NOR flash. Using system jumpstart can be suitable for production programming.

Direct Programming – Bypassing or Around a Xilinx Device

Direct programming uses a third party in-system programmer that attaches to the SPI bus as shown in Figure 3. A wide range of these tools are available from vendors including DediProg [Ref 3], TotalPhase [Ref 4], and Corelis [Ref 5]. The in-system programmer is connected by USB to a PC, with the PC running a software application driving the transfer of programming data directly to the serial NOR flash without involvement from the Xilinx device. This also means without interference from the Xilinx device.

The simplest way to prevent interference from the Xilinx device during direct programming is to hold the Xilinx device in reset and force its I/O connections to the SPI bus to 3-state. Figure 3 shows a provision for this capability by mechanical or electronic control. Details are provided in Table 1.

Table 1: **Forcing Xilinx Device I/O Connections to the SPI Bus to 3-State**

Xilinx Device Family	I/O Pin 3-State Control	Notes
Zynq UltraScale+ MPSoCs	PS_POR_B	Hold Low at power-on, or any time after, for duration of direct programming.
UltraScale and UltraScale+ FPGAs	PROGRAM_B	Hold Low at power-on, or any time after, for duration of direct programming.
Zynq-7000 AP SoCs	PS_POR_B	Hold Low at power-on, or any time after, for duration of direct programming.
7 Series FPGAs	PROGRAM_B	Hold Low after power-on, for duration of direct programming.

Although Figure 3 suggests a traditional connector for the third party in-system programmer, other methods of connection are possible. Chip-clips are available that can grab serial NOR flash in SO8W or SO16W packages that have been soldered to a board. Pomona Electronics [Ref 6] and 3M [Ref 7] are examples of vendors who manufacture such clips. When only a few units are to be programmed, flying wires can be soldered to a board, provided strategically placed vias exist on the signal traces for this purpose.

Direct programming requires a third party in-system programmer, board-level provisions to connect it, and the capability to hold the Xilinx device in reset. The incremental board cost is negligible. The third party in-system programmer represents a nominal investment in a highly useful tool. A compelling benefit is the extended selection of serial NOR flash substitution given the capabilities of a universal device programmer. Using direct programming can be suitable for production programming.

Recommendation Summary

- Use a composite footprint capable of accepting three packages.
- Provide both indirect and direct programming capability.
  - Indirect programming by Vivado Design Suite provides ease of use with serial NOR flash as listed in the *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1].
  - Direct programming by third party in-system programmer extends the selection of serial NOR flash for substitutions.
- Perform board-level modeling with IBIS.
  - Design for signal integrity on JTAG and SPI interfaces.
  - Obtain signal delays to inform timing analysis.

Selection – NOR Flash Technical Considerations

Xilinx devices are designed for compatibility with a wide range of serial NOR flash. *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1] lists the devices that can be programmed by the Vivado Design Suite. Many other serial NOR flash can be used. Check the following technical considerations for compatibility between a Xilinx device, a candidate serial NOR flash, and system design requirements.

Density and I/O Standard

A candidate serial NOR flash must have sufficient density to hold at least one full, uncompressed programming data payload for the Xilinx device. In support of reliable field update capability, the density requirement should be doubled to provide for storage of a fallback golden image to guard against risk of power failure or other interruption during a field update operation. In addition to programming data payload(s) for the Xilinx device, additional density can be required to support application needs such as code or data storage, an operating system image, or a file system.

A candidate serial NOR flash must have an I/O standard that is compatible with the configuration or boot interface of the Xilinx device that is often constrained by system design requirements or the Xilinx device itself. Common I/O standards for configuration or boot interfaces are 3.3V or 1.8V. Other voltages can be used as long as all devices are operated within their data sheet specifications.

Command Support

A candidate serial NOR flash must have command support for at least those commands that are issued by the Xilinx device as part of configuration or boot. In the interest of widest compatibility, Xilinx devices draw from a small command set of the most common commands.

The commands that can be issued during configuration by UltraScale, UltraScale+, and 7 series FPGAs are shown in Table 2. Many commands are optional.

Table 2: **UltraScale, UltraScale+, and 7 Series FPGAs Configuration Command Set**

Command Code	Command Name	Notes
0x0B	Fast read, 24-bit address, single output	This command is unconditionally issued at the start of configuration and therefore, absolutely required.
0x3B	Fast read, 24-bit address, dual output	These commands are appropriate for higher bandwidth in cases where the programming data does not exceed 128 Mb. These commands are optional and only issued if the header of the programming data indicates the FPGA is to read the remainder of the programming data using one of them.
0x6B	Fast read, 24-bit address, quad output	
0x0C	Fast read, 32-bit address, single output	These commands are optional in cases where the programming data does not exceed 128 Mb. For cases where the programming data exceeds 128 Mb, one (and only one) of these commands is required. These commands are only issued if the header of the programming data indicates the FPGA is to read the remainder of the programming data using one of them.
0x3C	Fast read, 32-bit address, dual output	
0x6C	Fast read, 32-bit address, quad output	

UltraScale, UltraScale+, and 7 series FPGAs do not negotiate during configuration. The bus widths and commands are set when the programming data is generated and are at the direction of the system designer. Based on Table 2, the minimum command set a candidate serial NOR flash must support is 0x0B for programming data that does not exceed 128 Mb. Otherwise, the minimum command set is 0x0B and 0x0C for programming data that exceeds 128 Mb.

The FPGA does not issue commands to enable 32-bit addresses or to enable quad mode. A candidate serial NOR flash must be ready at power-on to respond to the few commands that are issued by the FPGA during configuration. This can require programming of control bits in its non-volatile control register to permanently enable features.

Additionally, it is important to understand that the FPGA expects to read the entire payload of programming data with at most two read commands and possibly only one. For this reason, a candidate serial NOR flash must support a full array read with a single read command. In other words, it must be possible to read all data from it by issuing just one read command. While

most serial NOR flash support full array read, certain products built with die-stacking technology or with multiple chip selects, might not—making it necessary to confirm.

Zynq UltraScale+ MPSoCs have two boot modes for serial NOR flash. The selection is made by bootstrapping the mode pins to select between the use of 24-bit or 32-bit addresses. When 24-bit addresses are selected, the BootROM can only issue the commands shown in Table 3. Similarly, when 32-bit addresses are selected, the BootROM can only issue the commands shown in Table 4. Use of 32-bit addresses is optional in cases where the programming data does not exceed 128 Mb. For cases where the programming data exceeds 128 Mb, 32-bit addresses are required. Several commands listed in the following tables are optional.

Table 3: **Zynq UltraScale+ MPSOCs and Zynq-7000 SoCs Boot Command Set, 24-bit Addresses**

Command Code	Command Name	Notes
0x03	Read, 24-bit address, single output	This command is unconditionally issued by the BootROM at the start of boot to read the boot header, and therefore, absolutely required.
0x6B	Fast read, 24-bit address, quad output	This command is issued by the BootROM after initial read of the boot header in single mode to test if the boot header can be read in quad mode. If the read is successful, boot continues in quad mode. If the read fails, dual mode is attempted.
0x3B	Fast read, 24-bit address, dual output	This command is issued by the BootROM after failed read of the boot header in quad mode to test if the boot header can be read in dual mode. If the read is successful, boot continues in dual mode. If the read fails, single mode is used.

Table 4: **Zynq UltraScale+ MPSOCs Boot Command Set, 32-bit Addresses**

Command Code	Command Name	Notes
0x13	Read, 32-bit address, single output	This command is unconditionally issued by the BootROM at the start of boot to read the boot header, and therefore, absolutely required.
0x3C	Fast read, 32-bit address, quad output	This command is issued by the BootROM after initial read of the boot header in single mode to test if the boot header can be read in quad mode. If the read is successful, boot continues in quad mode. If the read fails, dual mode is attempted.
0x6C	Fast read, 32-bit address, dual output	This command is issued by the BootROM after failed read of the boot header in quad mode to test if the boot header can be read in dual mode. If the read is successful, boot continues in dual mode. If the read fails, single mode is used.

Zynq UltraScale+ MPSoCs do not negotiate address size during boot, but they do negotiate bus width. Based on Table 3 and Table 4, the minimum command set a candidate serial NOR flash must support is 0x03 for programming data that does not exceed 128 Mb. Otherwise, the minimum command set is 0x13 for programming data that exceeds 128 Mb. When two serial NOR flash are used in a dual parallel topology, the BootROM understands this as a performance optimization. Therefore, the BootROM assumes the two serial NOR flash are both capable of operating in quad mode and does not negotiate for a smaller or mixed bus width.

Zynq-7000 AP SoCs behave similarly to Zynq UltraScale+ MPSoCs, but do not support 32-bit addresses for boot. This means that only Table 3 is relevant for Zynq-7000 AP SoCs with 0x03 as the minimum command set.

Zynq UltraScale+ MPSoCs and Zynq-7000 AP SoCs are similar to UltraScale, UltraScale+, and 7 series FPGAs because they do not issue commands to enable 32-bit addresses or to enable quad mode. They expect to read the entire payload of programming data with at most two read commands and possibly only one.

Additionally, it is important to understand the distinction between BootROM behavior, which cannot be modified by users, and the subsequent behavior of firmware or other software that will be loaded. In most designs, the Xilinx-provided first stage boot loader (FSBL) is an attractive solution and is loaded by the BootROM. The FSBL for Zynq-7000 AP SoCs can be used with quad output serial NOR flash and issues quad output commands, thereby increasing the minimum command set from 0x03 only, to 0x03 and 0x6B. The FSBL for Zynq UltraScale+ MPSoCs provides increased flexibility because it can be used with a broader range of serial NOR flash and issues quad output commands by default. It however, provides a compile-time option to reduce the command set to dual output or single output. Using this option to reduce the command set to single output commands only preserves the 0x03/0x13 minimum command set.

Performance

System designs with power-on latency constraints can require increased bandwidth to minimize configuration or boot time. This requirement propagates into bus width, clock frequency, Xilinx device settings necessary to achieve these, candidate serial NOR flash selection, and aspects of board design. It is recommended that you design for optimum performance from the solution, taking as much as you need but not more. This is not to suggest the design of a low performance solution, but rather, an informed decision to take less performance than is available from the solution. By taking more performance, perhaps leaving no margin, degrees of freedom in sourcing flexibility and cost reduction are constrained.

When choosing the bus width, a solution that uses fewer bus commands is supportive of sourcing flexibility. Clock frequency is an additional but separate consideration where making a choice to operate at a frequency lower than the maximum possible for the solution also supports sourcing flexibility.

Timing analysis is necessary to determine the maximum frequency of operation, and requires I/O timing for the components involved (typically data sheet parameters) as well as signal delays on the board design (typically modeled with an IBIS simulation tool using information extracted from the board layout). Xilinx recommends board-level modeling, not only for confirming signal integrity of the SPI interface clock (SCK), but to arrive at a confident appraisal of its maximum frequency. Knowledge of the maximum frequency of operation then provides opportunity to design slack in the timing budget to allow interoperability with slower, lower performance serial NOR flash.

At the start of configuration or boot, Xilinx devices use a very low clock frequency to ensure the header of the programming data can be read reliably. An optional increase in clock frequency can be communicated through the header and is set when the programming data is generated.

UltraScale, UltraScale+, and 7 series FPGAs provide a range of frequency options. The most common option is CCLK, sourced from an internal oscillator, with support for a variety of division ratios. Another option is EMCCLK, which is sourced from an external clock signal with support for division by powers of two. CCLK provides a broader range of division options than EMCCLK making it more flexible. However, CCLK can be less precise depending on the precision of the EMCCLK source. Generally, use of EMCCLK is a performance optimization for increased precision in configuration time, or for dialing-in a specific frequency with the intent to approach the maximum frequency of operation. Where sourcing flexibility is concerned, CCLK is preferable over EMCCLK due to the increased flexibility it provides, assuming it can meet system design requirements.

Zynq UltraScale+ MPSoCs and Zynq-7000 AP SoCs also provide a range of frequency options. The processing systems in these devices are supported by a rich peripheral set that includes an SPI controller used for booting from serial NOR flash. The available frequencies are high precision and derived by multiple stages of division from a PLL source. Frequency modifications, through divider programming changes, are accomplished by register initialization parameters contained in the boot header.

The best opportunities for performance choices in favor of sourcing flexibility exist during initial design development and before the transition to production. While it is never too late for design changes given the programmability of Xilinx devices, executing a proposed change

requires appropriately trained staff with knowledge of the design and Vivado Design Suite. Design changes can potentially trigger re-validation if required by company policy or certification regulations. The engineering team could have disbanded and in extreme cases the time needed for re-validation could result in revenue delay.

Power–on Sequence

During the power-on sequence, both the Xilinx device and the candidate serial NOR flash experience their respective power supplies ramping. When the power supplies for each device become valid, each device can then require additional time to initialize itself. If the Xilinx device completes its initialization first and begins configuration or boot before the candidate serial NOR flash is ready to receive read commands, then configuration or boot can fail.

In general, system designers must consider the timing relationship effects of the power supply sequence, the power supply ramp rates, Xilinx device power-on timing, and serial NOR flash power-on timing. Most recent serial NOR flash that are likely to be considered for use in new designs exhibit sub-millisecond initialization after receiving valid power. However, timing relationship effects from characteristics of the board-level power supplies can significantly skew the readiness of the serial NOR flash with respect to the Xilinx device.

There is no specific requirement for power-on timing of the serial NOR flash. However, it is a requirement to verify that the power-on sequence ensures the serial NOR flash is ready before the Xilinx device. Allocating at least one millisecond to the serial NOR flash for initialization after receiving valid power supports sourcing flexibility.

Recommendation Summary

- To increase sourcing flexibility, use only the needed configuration or boot performance. Prefer lower frequency and narrower data width, where possible.
- For development, debug, and prototyping select a serial NOR flash listed in the *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1].
  - Compatibility of these has already been confirmed.
  - Always check board-specific power-on sequencing.
- If serial NOR flash listed in the *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1] meet production criteria, continue to validation.
- Otherwise, check compatibility of alternatives and validate on a known good system.

System – Application Storage and Field Update

After configuration or boot, many designs continue to access the serial NOR flash at runtime for application storage, such as code or data. Application storage might need to support writes and reads to enable file systems or field update. In support of reliable field update capability, two programming data payloads are stored, one being the fallback golden image that is rarely, if ever, rewritten and the other being the update image that can often be entirely rewritten during a field update operation.

UltraScale, UltraScale+, and 7 series FPGAs support direct access to the SPI interface after configuration has completed. This allows connection of soft IP cores implementing SPI controller functionality, or connection of custom logic for unique applications. In Zynq UltraScale+ MPSoCs and Zynq-7000 SoCs, the integrated SPI controller allows continued use of the SPI interface after boot has completed.

The system design is responsible for managing flash read and write access at runtime. This broad statement not only covers the SPI controller functionality, but also firmware, device drivers, applications, and even operating systems. The operating systems include those executing on the Xilinx device or executing remotely and controlling operations on the SPI interface of the Xilinx device. Commands, data widths, and clock frequencies, up to the full capability of the board-level solution, can be used at the discretion of the system design. Where sourcing flexibility is concerned, the guiding principal is the same for runtime use of the serial NOR flash as it is for configuration or boot—take as much as you need but not more.

Given the complexity of many system designs, and the common reuse of large blocks of third party or legacy design sources (in hardware as well as software forms), it can be challenging to dictate use of specific commands, data widths, and clock frequencies. A more pragmatic approach that can have several benefits is to develop a complete understanding of the commands, data widths, and clock frequencies being used to:

- Ensure neither the SPI interface nor the serial NOR flash are operated beyond their limits.
- Identify use of uncommon or vendor-specific commands with a goal of eliminating them.
- Identify unnecessary use of vendor specific status, command, and identification registers.
- Identify unnecessary performance take, reducing frequency and bus width if possible.

This approach is multi-disciplinary because it is likely to involve both hardware and software developers, especially in team-based design of complex systems. It is unreasonable to expect every developer on a project to have a design-for-substitution mindset but not unreasonable to expect adherence to a list of development guidelines.

Other areas of system design can intersect the application of serial NOR flash for runtime use as well as for configuration or boot:

- Design security is increasingly required and should be anchored by encryption, authentication, and secure boot features of Xilinx devices rather than ad hoc solutions leveraging serial NOR flash vendor-specific features. Xilinx devices support a range of security features, many of which are integrated into the devices and can use non-volatile

eFUSE for keys and settings to provide robust protection without increasing development or bill of material costs.

- Open-source software is increasingly being used, and due diligence in the selection process should include an assessment of the health of upstream development. This is of particular importance for driver, file system, and second stage boot loader sources because these can enable (or restrict) the ability to make substitutions with serial NOR flash that become available in the future. An example of how the development community moves on can be seen through file systems that are transitioning away from JFS2 towards UBIFS.

Form, fit, and function substitutes for a given density serial NOR flash can also include higher density serial NOR flash. While this type of substitution is usually not desirable because it is likely to increase costs, it is useful in scenarios where the original density is transiently unavailable or has been discontinued. A temporary or even permanent cost increase can be preferable over halting production. For this reason, avoid implementation of any system behaviors or checks that would prevent use of serial NOR flash devices of higher density than required.

Recommendation Summary

- To increase sourcing flexibility, use only the needed runtime capability and performance.
  - Identify, reduce, and document the commands that are being used.
  - Prefer lower frequency and narrower data width, where possible.
- Architect for supporting a variety of serial NOR flash during the lifetime of the system.
  - Eliminate unnecessary software checks of vendor identification registers.
  - Move as much functionality as possible from hardware into software.
  - Prefer software sources with healthy upstream development for longevity.

Programming – Memory Configuration Files

The Vivado Design Suite supports a range of design flows. In the case where serial NOR flash is used for configuration or boot, these flows converge after the generation of programming data. The point of convergence is a memory configuration file to be used for programming a serial NOR flash. A memory configuration file is simply a file containing a representation of the data to be programmed into the serial NOR flash. It can contain one or more programmable logic bitstreams, processing system code or data partitions, as well as additional partitions for application storage, file systems, and operating system images.

There are several industry standard file formats for memory configuration files including MCS, HEX, and SREC. These formats are based on ASCII text records, a representation that increases file size on disk by a factor of two or more but provides support for non-contiguous data blocks, checksums, and easy readability and editing. Binary data files, often with BIN or DAT filename extensions, are also suitable as a no-frills alternative. The Vivado Design Suite supports MCS and BIN formats.

Using Vivado Design Suite for Programming

Use of the Vivado Design Suite for indirect programming of serial NOR flash listed in the *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1] is recommended for development, debug, and prototyping with possible continued use during validation if the same serial NOR flash is used in production. Wizards in the Vivado Design Suite guide users through the process of generating a memory configuration file and subsequently programming the serial NOR flash.

For UltraScale, UltraScale+, and 7 series FPGAs, the Generate Memory Configuration File wizard in the Hardware Manager GUI, guides users to select a serial NOR flash listed in the *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1]. In the wizard, select the data bus width and identify programmable logic bitstream(s) and any additional data to be included in the memory configuration file. The output from the Generate Memory Configuration File wizard is an MCS file that is created by the `write_cfgmem` utility. For details on using the Vivado Design Suite Hardware Manager to perform indirect programming of the serial NOR flash with the contents of the MCS file, refer to the *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1].

For Zynq UltraScale+ MPSoCs and Zynq-7000 AP SoCs, the Create Boot Image wizard in the Software Development Kit (SDK) GUI guides users to identify boot image partition(s) and any additional data to be included in the memory configuration file. The output from the Create Boot Image wizard is an MCS file that is created by the bootgen utility. Refer to the *Using Xilinx SDK* (UG782) [Ref 8] for details on using the SDK to perform indirect programming of the serial NOR flash with the contents of the MCS file. The *Xilinx Software Command-Line Tool (XSCT) Reference Guide* (UG1208) [Ref 9] contains a use case of applying the Xilinx software command line tools to script a similar operation. Also see the *Zynq UltraScale+ MPSoC Software Developers Guide* (UG1137) [Ref 10] and *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821) [Ref 11] for information on bootgen. Xilinx also provides embedded design tutorials *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) [Ref 12] and *Zynq-7000 All Programmable SoC: Embedded Design Tutorial* (UG1165) [Ref 13].

Although not obvious, both the Hardware Manager and SDK must often program more than just the data array of the serial NOR flash. In addition to programming the data array with the contents of the MCS file, when needed the programming tools also program certain bits in non-volatile control registers. These non-volatile control register bits can be vendor or even device-specific, and can require programming to permanently enable features of the serial NOR flash, such as 32-bit addresses or quad mode. As a reminder, the Xilinx device does not issue commands to enable capabilities during configuration or boot. The serial NOR flash must power on with these capabilities enabled.

Using Third-Party Solutions for Programming

After using the Vivado Design Suite for indirect programming of serial NOR flash listed in the *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1] for development, debug, and prototyping, it can also be used for validation if the same serial NOR flash is used in production. In cases where use of alternative serial NOR flash is desired to meet production criteria, a transition to a third-party direct in-system programming solution should be made for validation. In either case, after production begins, use of a third-party direct bulk pre-programming solution is recommended to enable low-cost, high-volume assembly of programmed serial NOR flash directly onto the board.

MCS being a standard file format, most third-party programming solutions directly read MCS files generated by the Vivado Design Suite. Vivado Design Suite can also generate BIN files. Most third-party programming solutions have a file format conversion tool, and there are also excellent open-source file format conversion tools such as the SRecord project on SourceForge [Ref 14].

With a third-party programming solution, care must be taken to continue the programming of non-volatile control register bits in the serial NOR flash, when needed, to permanently enable features such as 32-bit addresses or quad mode in support of Xilinx device configuration or boot. In itself, the programming of non-volatile control register bits is simple, however:

- It becomes an explicit step in the programming process that can be missed due to oversight.
    - The remedy during validation work using direct in-system programming is simply to apply the missed step in the programming process.
    - The remedy during direct bulk pre-programming is similar, but has a direct cost for a second programming pass.
- Programming could be required after each erase performed with a third-party programming solution. Some vendors interpret erase device to mean erasure of the data array as well as the non-volatile control registers where other vendors could erase the data array only.

As a result, when transitioning from programming with the Vivado Design Suite to a third-party programming solution, always validate a first article from the new programmer. This is especially important when bulk pre-programming to avoid incurring additional programming passes.

Recommendation Summary

- Use the Vivado Design Suite for indirect programming of serial NOR flash listed in the *Vivado Design Suite User Guide Programming and Debugging* (UG908) [Ref 1] during the early stages of a project for shorter design cycles, lower risk, and broader engineering access.
- Transition to direct programming with third-party programming solutions during validation or at production, depending on preference to continue use of serial NOR flash listed in the *Vivado Design Suite User Guide Programming and Debugging* (UG908) versus substitution of a compatible alternative that better meets production goals.
  - Continue to use existing MCS or BIN files with third-party programming solutions.
  - Permanently enable serial NOR flash features such as 32-bit addresses or quad mode using the third-party programming solution if the Xilinx device is using these capabilities during configuration or boot.

Production – Component Substitution

In most companies, business is a team effort with staff organized functionally. It is unusual to find one individual responsible or knowledgeable about every aspect of a business. Over time, organizations and staff can change. Without product continuity planning, a company can gradually lose the capability to maintain and support a system for a long lifetime. This can include the loss of agility to make serial NOR flash substitutions.

Often, a component sourcing or production team is responsible for responding to changes in availability of components in the system bill of materials. Reasonably, these teams could know nothing about the design of the system compared to the original design team. Yet, they need to:

- Identify form, fit, and function substitute(s) for the serial NOR flash
- Validate substitute(s) in lab until at least one passes
- Issue an engineering change order

This can be time consuming and risky without access to the original design team. However, the original design team could have disbanded. It is not difficult to imagine a scenario where a company must reverse engineer its own system to regain lost knowledge.

Adopt the best practice of having the design team create sustaining documentation for the production team during development. Capture that information in document control to ensure it can be preserved and found by any team with need-to-know. Where serial NOR flash is concerned, the outline of considerations presented in this application note serve as a guide to what ought to be recorded. Further, develop and capture the procedure for programming and validation of substitutes. Test the documentation for completeness by having someone unfamiliar with the process follow the procedure unassisted as a clean room rehearsal.

Recommendation Summary

- Have development teams create sustaining documentation for systems.
  - Capture serial NOR flash functional and performance requirements.
  - Establish procedure for programming and validation of substitutes.
- Preserve sustaining documentation in document control.

Conclusion

With the growth in volatility of the NOR flash market over the past decade and the relentless increase in cost reduction pressure, adoption of a design-for-substitution mindset is becoming essential for many projects. This mindset provides protection against component market transients for products with moderate to long lifetimes, and exploitation of planned and opportunistic cost reduction opportunities for products with aggressive cost targets.

The recommendations in this application note provide resilience to common NOR flash sourcing challenges, as well as open opportunities for cost reduction. More importantly, the recommendations illustrate how choices in the application of NOR flash in a product can significantly change sourcing flexibility, making it clear that informed decisions and planning ahead can materially reduce risk and increase profitability.

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

1. *Vivado Design Suite User Guide Programming and Debugging* (UG908)
2. Xilinx End User License Agreement
3. DediProg (www.dediprog.com)
4. TotalPhase (www.totalphase.com)
5. Corelis (www.corelis.com)
6. Pomona Electronics (www.pomonaelectronics.com)
7. 3M (www.3m.com)
8. *Using Xilinx SDK* (UG782)
9. *Xilinx Software Command-Line Tool (XSCT) Reference Guide* (UG1208)
10. *Zynq UltraScale+ MPSoC Software Developers Guide* (UG1137)
11. *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821)
12. *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209)
13. *Zynq-7000 All Programmable SoC: Embedded Design Tutorial* (UG1165)
14. SRecord Package(www.srecord.sourceforge.net)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/05/2018	1.0	Initial Xilinx release.

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.