PROFINET Reference Design Bootstrap and Flash Access

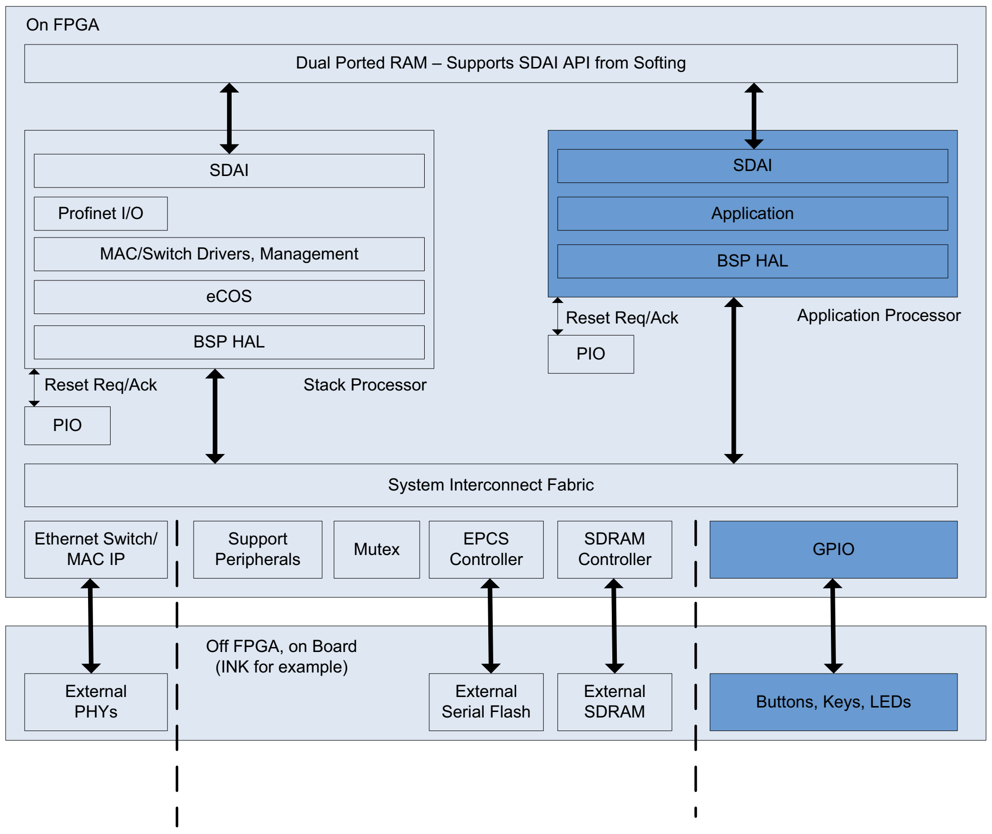
This document describes the Altera® PROFINET reference design bootstrap process. Use this process when you require custom modifications to the Altera PROFINET reference design.

A complete bootstrap loader is included with the Altera PROFINET reference design.

Overview

Figure 1 shows the control blocks and interfaces in the PROFINET reference design, including the components that are instantiated as soft logic on the FPGA and the components that are located off the FPGA.

Figure 1. PROFINET Reference Design Control Blocks and Interfaces



The Industrial Networking Kit (INK) from Altera provides onboard components that are referenced in the lowest block, labeled *Off FPGA, on Board* as shown at the bottom of Figure 1.

For more information on the INK, refer to the Industrial Networking Kit page on the Altera website.

Since you can retarget various components of the reference design to your specific board implementations, the Softing portions of the design, such as the Ethernet switch and MAC IP, must be adapted by Softing when creating a custom design. A user board with the same design as the INK may be completed on the application side of the design (everything shown as dark blue in Figure 1). If you require a custom design, certain guidelines must be followed for the user-provided portions to function as expected.

Asymmetric Multiprocessing

The Softing PROFINET reference design uses the asymmetric multiprocessing (AMP) architecture, as opposed to the symmetric multiprocessing (SMP) architecture. A well known example of SMP architecture is the modern personal computer, or server, that has more than one processor. An SMP system enables all the processors to have access to all the memory and peripherals in the system; the operating system grants access to all those resources appropriately. The benefit of SMP is that it is a simpler programming model from an application point of view, but this benefit is at the expense of some added complexity in hardware design and operating system. AMP architecture usually implements fixed access to particular peripherals, makes use of fixed functions per processor, and may share state or data across processors. Modern SMP systems require cache coherency mechanisms in software or hardware to maintain data consistency across cores (with the required added complexity). AMP systems may or may not implement cache consistency mechanisms, and if not, require inter-processor communication (IPC) mechanisms to maintain data consistency when messaging between the processors. In the Softing reference design case, DPRAM is used to pass messages between the processors, and a hardware mutex is used to claim ownership of the DPRAM before accessing.

The Stack and Application Processors

The PROFINET reference design contains a stack processor and an application processor. The stack processor runs the proprietary industrial Ethernet protocols provided by Softing. In the PROFINET reference design example, the stack processor runs the following:

- The eCos operating system
- The device drivers required to access the industrial Ethernet switch
- The MACs
- The PROFINET I/O software required to handle packet processing for PROFINET

A simple device application interface (SDAI) layer is present on this processor to communicate with the application software running on the application processor. All IP and software associated with the stack processor should be considered as a *gray* box (partially opaque) and not subject to changes. This requirement impacts modifications on the application processor side.

The application processor runs the application and communicates to the stack processor through SDAI as required. As provided by Softing, the application processor is running an application on top of a Hardware Abstraction Layer (HAL) that is created for the system design. The HAL is a set of defines, macros, and service routines that access the hardware components provided by the system design. If the system design is modified, the board support package (BSP) and HAL must be recreated and recompiled so the macros, defines, and service routines access the hardware correctly. These modifications usually involve changes to the I/O map or require the addition of resources. In the Softing reference design case, most of the hardware components should not need to be changed. With custom designs, you can remove the buttons, keys, and LEDs implemented by the reference design; you can replace these with user-specific design components.

The PROFINET reference design supports the multiple levels of abstraction required to enable you to create your own custom designs. The stack processor handles the industrial Ethernet I/O portion of a design and communicates with the application through the SDAI interface provided by Softing. Messages are received from the stack for I/Os to application specific I/O locations accessed by a control program running on a programmable logic controller (PLC). The industrial Ethernet protocol (Ethernet/IP, PROFINET, or EtherCAT) provides the communications substrate for messages to and from the PLC.

For more information on Softing's reference design and SDAI, see the Softing documentation provided with their PROFINET Development Kit.

NIOS II Considerations for Multiprocessor System Designs

This section addresses some of these issues with the multiprocessor solution implemented by the PROFINET reference design.

- By default, cache coherency is not provided by the NIOS II or by external logic. Not having cache coherency forces you to create software designs that use static partitioning of functions to particular processors. You can implement external cache coherency. Data is shared through explicit sharing mechanisms, such as DPRAM and the use of a mutex. Peripheral control is usually statically allocated to a particular NIOS processor, but could be shared through the use of a mutex.
- Altera provides a mutual exclusion peripheral primitive, a mutex. This mutex may be used to create mailbox or inter-processor mechanisms between multiple processors in a systems design.
- Interprocessor interrupts can be created using GPIO between processors.
- Each processor can share external RAM and flash memory or use its own internal dedicated RAM or flash memory. There are trade-offs involved for each choice.
- Queuing spin locks can be implemented on top of the Altera mutex primitive. A queuing spin lock addresses the *thundering herd* problem of multiple processors attempting to acquire a mutual exclusion primitive and a possible fairness of access problem.

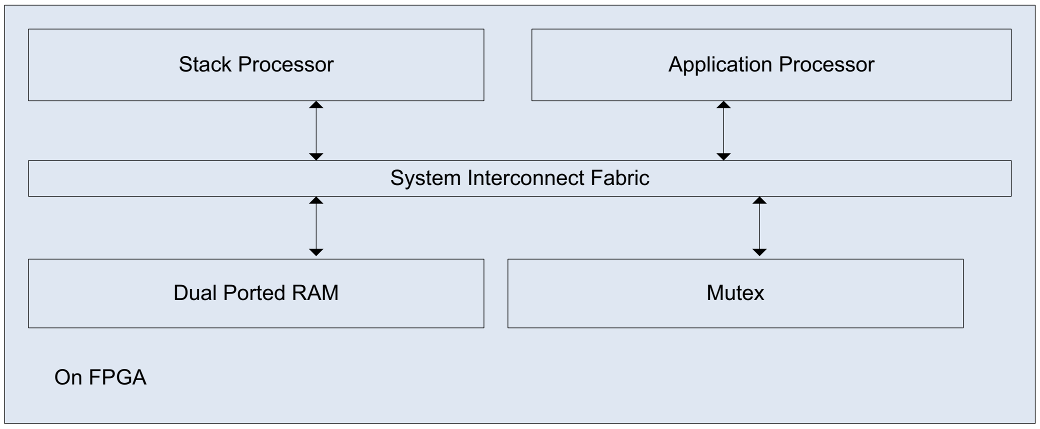
Using Dual-port Memory for Inter-Processor Communication

The Altera NIOS processor does not support multiprocessor cache coherency, and there are no logic blocks to provide coherency across multiple NIOS instances. However, the logic could be developed by a third party.

One common solution for using multiple NIOS processors is to allocate static tasks to specific processors and share data through a commonly accessible memory. A mutex is used to control access to this memory area such that only one processor accesses the memory at a time. When a processor needs to access the memory, it must first acquire the ownership mutex and, using cache management instructions, invalidate all cached memory from the shared resource before reading or writing to the memory. This operation ensures no false sharing of data occurs.

Figure 2 shows the two NIOS processors, the dual-ported shared RAM, and the mutex used to marshal access to the DPRAM.

Figure 2. Dual-ported Shared RAM



For the stack processor to write a message to the application processor, the following operations must occur:

- The stack processor acquires the access mutex.
- The stack processor invalidates any cache contents associated with memory locations in the DPRAM.
- The stack processor writes a message to the DPRAM, and making sure the write cycles are *written through* the NIOS cache to the DPRAM.
- The stack processor causes the application processor to get an interrupt, signaling that a message is ready in the DPRAM for consumption. This is done by writing to a PIO register, causing the application processor to receive an interrupt.

The application processor processes receipt of the message from NIOS as follows:

- The application processor receives an interrupt indicating a message is ready to be read from the DPRAM.
- The application processor acquires the access mutex.
- The application processor invalidates any cache contents associated with memory locations in the DPRAM.

- The application processor reads the message from the DPRAM and acts upon it.

This description does not involve describing any potential optimizations, such as keeping track of producer and consumer indices, posting multiple messages simultaneously, or direct and indirect types of messages. Those optimizations are left to the implementer.

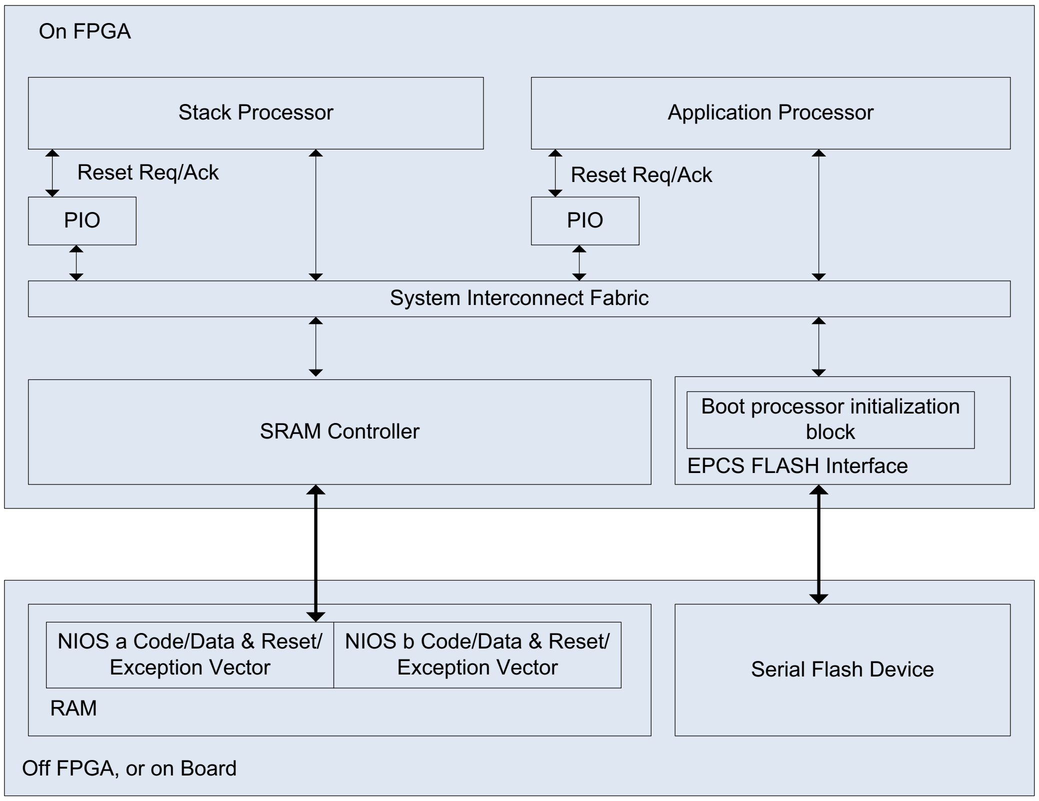
Bootstrapping the PROFINET Reference Design

This section describes the system design considerations, including the design choices made for the PROFINET reference design to support an orderly system bootstrap.

A system bootstrap process involves bringing up the processors and system peripherals to a known working state in an orderly fashion. The intent is to eliminate the possibility of race conditions in hardware or software that could show up as an occasional system initialization failure.

The PROFINET system design uses the application processor, shown in Figure 3 below, to bootstrap the system.

Figure 3. PROFINET System Design



The stack processor is designed to come out of a board-reset held in a reset state. The stack processor reset and exception vector point to a known location in the external RAM. The reset vector of the application processor points to the bootstrap location located in the EPCS flash controller, and the exception vector of the application processor points to a location in external RAM.

When the system design resets, the stack processor is held in reset and the application processor immediately starts executing code from its reset vector, pointing to the EPCS flash controller. The EPCS flash controller contains bootstrap code to copy the images of the stack and application processor execution to external RAM. After copying the executable images from the flash memory to external RAM, the bootstrap loader (executing on the application processor) takes the stack processor out of reset, causing the stack processor to start execution from its reset vector. The application processor then branches directly to the starting location of the image copied to external RAM.

Note that the stack and application processors have controls to restart the opposite processor. The application processor is used to bootstrap the system since the stack processor must be running the stack code first before the application can start execution.

Accessing Non-Volatile (Flash) Memory on the PROFINET Reference Design

When updating persistent configuration data in the flash memory, only one processor should be updating the flash memory at one time. Since the bootstrap process copies all executable code from flash memory to external RAM, and because the flash memory is a serial device with a EPCS controller, updates to the flash memory will not cause potential read errors that can occur with CFI flash memory and other processors reading code or data from the flash memory simultaneously.

Softing implements a flash memory update service running on the stack processor, allowing the application processor to update the flash memory through an SDAI interface such that flash memory updates only occur while running on the stack processor. This implementation eliminates possible conflicts or errors from both processors attempting to access (read or write) the flash memory simultaneously; since after the bootstrap process, only the stack processor may update the flash memory.

References

AN 485: Alternative Nios II Boot Methods

NIOS II Multiprocessor Design Example, Altera website

Softing Industrial Automation Gmbh provides custom engineering services for custom modifications.

Document Revision History

Table 1 lists the revision history for this document.

Table 1. Document Revision History

Date	Version	Changes
February 2013	1.0	Initial release.