

FPGA vs. DSP Design Reliability and Maintenance

Introduction

Digital signal processing (DSP) underpins modern wireless and wireline communications, medical diagnostic equipment, military systems, audio and video equipment, and countless other products, becoming increasingly common in consumers' lives. Due to advances in semiconductor technology, ever more complex DSP algorithms, protocols, and applications are now feasible, which, in turn, increase the complexity of the systems and products. As the complexity increases, the system reliability is no longer solely defined by the hardware platform reliability, typically quantified in mean time between failure (MTBF) calculations. System reliability is increasingly determined by hardware and software architecture, development and verification processes, and the level of design maintainability.

One fundamental architecture issue is the type of hardware platform. DSP functions are commonly implemented on two types of programmable platforms: digital signal processors and field programmable gate arrays (FPGAs). Digital signal processors are a specialized form of microprocessor, while FPGAs are a form of highly configurable hardware. In the past, the use of digital signal processors was nearly ubiquitous, but with the needs of many applications outstripping the processing capabilities of digital signal processors (measured in millions of instructions per second (MIPS)), the use of FPGAs is growing rapidly. Currently, the primary reason most engineers choose use a FPGA over a digital signal processors is driven by the application's MIPS requirements. Thus, the comparison between digital signal processors and FPGAs focuses on MIPS comparison, which, while certainly important, is not the only advantage of an FPGA. Equally important, and often overlooked, is the FPGA's inherent advantage in product reliability and maintainability. This second advantage is the focus of this white paper.

FPGA and Digital Signal Processor Development Process

Nearly all engineering project managers can readily quote the date of their next product software update, or release. At most technology companies, there is usually a long internal list of software bugs or problem reports along with the software releases that will contain the associated patches, or fixes. It has come to be expected that all software, including DSP code, will contain some level of bugs and that the best one can do is to minimize this. By comparison, FPGA designs tend to be updated much less frequently, and it is generally an unusual event for a manufacturer to issue a field upgrade of a FPGA configuration file.

The reasons behind this are due to differences in the digital signal processor and FPGA engineering development process. There is a fundamental challenge in developing complex software for any type of processor. In essence, the digital signal processor is a specialized processing engine, which is constantly reconfigured for many different tasks, some DSP related, others more control or protocol oriented. Resources such as processor core registers, internal and external memory, DMA engines, and I/O peripherals are shared by all tasks, often referred to as "threads". These create many opportunities for the tasks to interact, often in unexpected or non-obvious ways. In addition, most DSP algorithms are required to run in "real time," so even unanticipated delays or latencies can cause system failures. A few of the more common causes of DSP software bugs are due to:

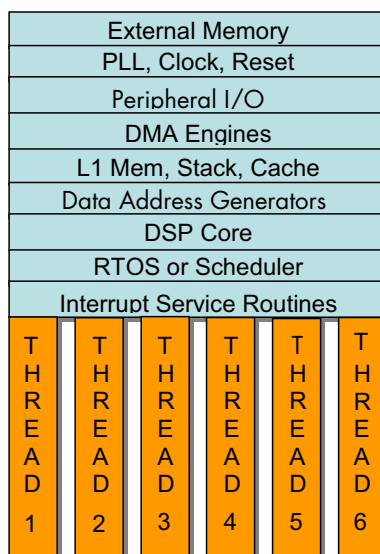
- Non-uniform assumptions regarding processor resources by multiple engineers simultaneously developing and integrating disparate functions
- Failure of interrupts to completely restore processor state upon completion
- Blocking of critical interrupt by another interrupt or by an uninterruptible process
- Undetected corruption or non-initialization of pointers
- Failing to properly initialize or disable circular buffering addressing modes
- Memory leaks, the gradual consumption of available volatile memory due to failure of a thread to release all memory when finished
- Dependency of DSP routines on specific memory arrangements of variables
- Unexpected memory rearrangement by optimizing memory linkers and compilers

- Use of special DSP “core mode” instruction options in core
- Conflict or excessive latency between peripheral accesses, such as DMA, serial ports, L1, L2, and external SDRAM memories
- Corrupted stack or semaphores
- Subroutine execution times dependent on input data or configuration
- Mixture of “C” or high-level language subroutines with assembly language subroutines
- Pipeline restrictions of some assembly instructions

Managing Resources

Microprocessor, digital signal processor, and operating system (OS) vendors have attempted to address these problems by creating different levels of protection or isolation between tasks or threads. The operating system, or kernel, is used to manage access to the processor resources, such as allowable execution time, memory, or common peripheral resources. However, there is an inherent conflict between processing efficiency and the level of protection offered by the OS. In digital signal processors (shown in [Figure 1](#)), where processing efficiency and deterministic latency are often critical, the result is usually minimal or zero OS isolation between tasks. Each task often requires unrestricted access to many processor resources in order to run efficiently.

Figure 1. Digital Signal Processor Block Diagram

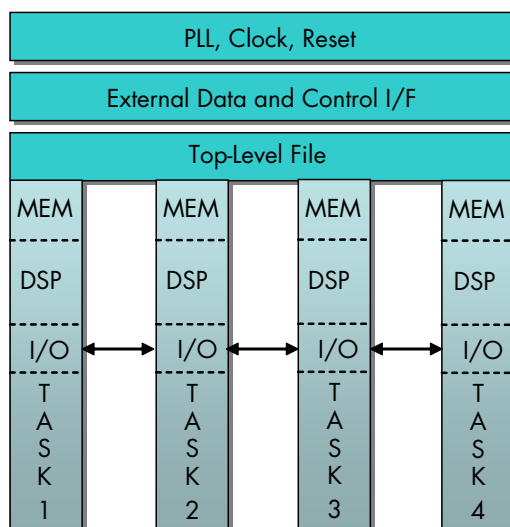


Compounding these development difficulties is incomplete verification coverage, during both initial development and regression testing for subsequent code releases. The near impossibility of testing all the possible permutations (often referred to as “corner cases”) and interactions between different tasks or threads that may occur during field operation makes it arguably the most challenging part of the software development process. Even with automated test scripts, it is not possible to test all possible scenarios. This process must be repeated after every update or modification of the software to correct known bugs or to add new features. Occasionally, a new software release also inadvertently introduces new bugs, which forces yet another release to correct the new bug. As products grow in complexity, the lines of code increase, as do the number of processor cores, and an ever-greater percentage of the development effort is devoted to software testing.

So how exactly does the FPGA development process improve on this unhappy state of affairs? The complexity of each task is more or less equivalent, no matter whether the design uses digital signal processor or FPGA implementation. Both routes offer the option to use third-party implementations of common signal processing algorithms, interfaces, and protocols. Each also offers the ability to reuse existing intellectual property (IP) on future designs, but that is where the similarity ends.

FPGAs offer a more native implementation for most DSP algorithms. Each task is allocated its own resources, and runs independently. It intuitively makes more sense to process each step of a continuously streaming signal processing chain in an assembly line-like process, with dedicated resources for each step. As Henry Ford discovered nearly 100 years ago, this process style dramatically increases the throughput.

Figure 2. FPGA Block Diagram



The FPGA resources assigned can be tailored to the task requirements, which can be broken up along logical partitions. This makes for a well-defined interface between tasks, and largely eliminates unexpected interaction between tasks. Because each task runs continuously, much less memory is required than in the digital signal processor, which must buffer the data and process it in batches. As FPGAs distribute memory throughout the device, each task is permanently allocated the dedicated memory it needs. This provides a high degree of isolation between tasks and results in modification of one task being unlikely to cause unexpected behavior in another task. This, in turn, allows developers to easily isolate and fix bugs in a logical and predictable fashion.

FPGA Design Verification

At the fundamental level, FPGA design and verification tools are closely related to ASIC development tools, and in practice, most ASIC designs are prototyped on FPGAs. This is critical, because bugs are not tolerated in ASICs. Since the possibility of field upgrades to remedy design bugs in an ASIC is remote, and because the time and development cost is very high, ASIC developers go to extreme lengths to verify designs. This has led to test methodologies that provide near-complete coverage of every gate in all scenarios with all possible inputs, accurate modeling of routing delays within the devices, and comprehensive timing analysis.

FPGA verification tools are close cousins of their ASIC counterparts, and benefit enormously from many years of investment in the ASIC verification process. The use of FPGA partitioning, test benches, and simulation models makes both integration and on-going regression testing very effective in quickly isolating problems, speeding development processes, and simplifying product maintenance and feature additions. These crucial advantages in the FPGA versus digital signal processor development process comparison will increase as the complexity of designs and the size of development teams increase.

For example, Altera® FPGAs are supported by a comprehensive set of in-house and third-party tools to provide a unified tool flow for architecture partitioning, floor planning, facilitation of design intent, simulation, timing closure, optimization, and maintainability. In particular, partitioning is integral to the design entry process. This partitioning, which normally includes all of the chip resources required within the partition, is extended during timing closure and

ongoing maintenance phases of the development, which guarantees a high degree of isolation. Each logical partition, as well as the overall design, can have independent test benches and simulation models.

Conclusion

The large electronic design automation (EDA) industry continually drives the development of FPGA and ASIC test and verification tools. It does not have a comparable counterpart in the software development world. This may change, as the industry realizes the enormous costs and challenges in software verification, but for now, the practical software solution is to keep downloading the latest patch.

Most engineering managers understand that the rate of software updates to remedy bugs far exceeds the rate of comparable FPGA updates in nearly all cases. It is expected and normal to roll out bug fixes on embedded software on a regular basis. With the availability of both low-cost and high-end DSP-optimized FPGA devices, extensive IP cores, availability of high-level design entry methods, and the inherent robustness of the design and verification process, FPGAs will increasingly be the preferred choice for implementing DSP.

Acknowledgements

- Michael Parker, Sr. Technical Marketing Manager, IP and Technology Product Marketing, Altera Corporation



101 Innovation Drive
San Jose, CA 95134
www.altera.com

Copyright © 2007 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.