

Design of Path Optimization Algorithm Using COTS Field Programmable Gate Array Hardware and Software Platforms

Neil Harold, John Ostapovich
Nallatech

E-mail: n.harold@nallatech.com
E-mail: j.ostapovich@nallatech.com

Rick Pancoast
Lockheed Martin MS2

E-mail: rick.pancoast@lmco.com

Jon Russo
Lockheed Martin ATL

E-mail: jrusso@atl.lmco.com

Introduction

It seems obvious to state that High Performance Embedded Computing (HPEC) technology has changed radically in the last 10 years, with significant upgrades in capability offering superior performance. One of the enablers of this is the commoditization of computer components helping to reduce the overall cost of HPEC technologies.

On closer analysis, however, one could argue that in fact very little has changed, in the sense that the challenges faced 10 years ago are the very same as those confronting developers today. Even more significantly, the techniques used to overcome these obstacles really have not changed either, in particular, the processing technologies that underpin embedded computing applications. As they start to reach performance limits imposed by the laws of physics, developers are faced with critical decisions over which direction to take to deliver future HPEC systems.

Revolution in Programmable Logic

Programmable logic has undergone a dramatic change over the last 10 years, with the advent of advanced multi-million gate devices, leading to the introduction of hugely powerful families of Field Programmable Gate Arrays (FPGAs). The increasing capability of these devices has seen them evolve from playing a supplementary role, often as “glue” logic in a system, to performing vital processing tasks that had previously been the domain of ASICs.

While the widespread use of FPGAs for embedded signal processing has taken longer to gather momentum, the insatiable appetite of military HPEC applications has meant that the DoD has shown more interest than most in using FPGAs for this purpose. The use of FPGAs has tended to be limited, however, to the role of a “hardware accelerator” supporting a more traditional processing architecture.

Vendors, developers and users are beginning to realize the potential of FPGAs for a greater share of embedded computing tasks, but even in this high-interest climate, questions remain over just how much extra performance can be gained when using FPGAs and, perhaps more importantly, at what cost?

FPGA Technology

The fixed architecture of general purpose processors (GPPs) and DSPs has constrained their main improvements

in capability over in the last 10 years to those gains attained through increasing clock frequencies. The flexible architecture of FPGAs, on the other hand, have been able to take full advantage of increasing transistor count to offer greater volumes of programmable logic combined with fixed function pieces of silicon such as embedded processors, multi-gigabit transceivers and embedded digital signal processing blocks.

The technical advances achieved in the last few years in particular have made FPGAs more powerful at the device level, easier to integrate at the board level and more applicable to signal processing applications at the system level. Unfortunately, these advances have not necessarily made developing applications for FPGAs any easier. Despite significant industry investment in design tools and compilers, the use of FPGAs remains mainly limited to user base with long-standing expertise in developing solutions using Hardware Description Languages (HDLs) such as VHDL and Verilog.

High-Level Languages for FPGAs

The last five years has seen unprecedented progress in the area of high-level tools for FPGAs, with varying degrees of success. There are a number of tools available offering a variety of approaches to solving this problem, mainly focused on offering translating high-level languages such as C/C++ and Matlab to a “middleware” level such as VHDL that can be mapped to the FPGA.

Many parameters influence the effectiveness of these tools, in particular the precise syntax of the high-level language used (they rarely operate on fully standard languages) and their ability to implement parallelism. Most of the available tools attack parallelism at the micro-level—they break down specific functions in the code that can be parallelized. Some tools look at the code at a macro-level, but this generally requires further variation from the standard language, i.e., more effort on the part of the programmer.

Nallatech’s DIME-C tool operates on a micro-level, translating discrete functions written in ANSI-C code into VHDL that can then be put through the standard synthesis and implementation process to generate a bitstream for the FPGA. It expresses the C-code as a data-flow and analyses that dataflow to determine which variables are related to each other in order to understand where inter-dependencies lie. This information is then used to implement parallel pipelines where appropriate.

