A Methodology for System-on-a-Programmable-Chip Resources Utilization

Sin Ming Loo[1] and Simon Y. Foo[2]

[1]Department of Electrical & Computer Engineering
Boise State University
Boise, Idaho 83725
smloo@boisestate.edu

[2]Department of Electrical & Computer Engineering
Florida A&M University – Florida State University
Tallahassee, Florida 32310
foo@eng.fsu.edu

Abstract

From the inception of an idea to the actual system implementation, the development time has been fast shrinking as the technology advances. This is partially due to system development costs. The other factor is to release the system/product to the market before other competitors. One interesting development during the last decade that has been perfectly suitable to be utilized in reducing the development time and costs is reconfigurable/programmable hardware. Reconfigurable hardware has been progressively replacing the application specific hardware in small volume designs. The use of reconfigurable hardware allows the system to be implemented and upgraded without non-recurring engineering costs and the system can be reconfigured in the field after deployment. This flexibility allows the hardware structures for a given portions of an application to be specialized and optimized to achieve a performance that can be orders of magnitude greater than that which can be achieved within most traditional processing systems that employ a Von Neumann style architecture [1].

However, the amount logic resources available within commercially off-the-shelf reconfigurable hardware are limited (number of pins for input/output, flip-flops, look-up-tables, etc.). For most applications, this limitation means that it is impossible to configure the reconfigurable hardware in a manner where all portions of the design are implemented for optimal performance. This is because such performance optimal implementations would consume enormous amounts of reconfigurable hardware resources -- probably orders of magnitude larger than can be made available.

To clearly understand this resource constraint problem, consider this scenario. A project is to be built with the in-house available FPGAs. An enormous number of these devices have been acquired after the preliminary design inquiry determines that the design will fit into this device. However, the design changes (including new features) have caused the design not to fit. The management has re-emphasized that the on-hand devices will be utilized, upgrading to a larger device within the same device family which has more on-chip resources is not an option. Many different synthesis options and synthesis tools have been tried and failed to place & route the design. At this point, the question for the design team is how to best utilize the available resources to fit the design and meet the performance constraints.

The desired compromise is to carefully use the reconfigurable medium in a manner that is cognizant of the space/time trade-off that is associated with computations performed in digital hardware [3]. In general, high performance can be gained through increased concurrency by employing more of the hardware to solve the problem. Since, reconfigurable resources of the targeted reconfigurable hardware environment are limited; the compromise translates into determining how much concurrency should be employed in order to meet the performance requirements of the application without exceeding the resource limits of the reconfigurable hardware medium. The key to finding this effective balance is to develop techniques that can determine, within the confines of the resource limitations, which portions of the problem must have increased levels of concurrency to meet the overall performance constraints and which portions of the problem can be implemented more sequentially to save room for the higher performing portions of the design. One might think that this is another resource constraint problem. In fact, it is a resource constraint problem with an added twist

to it. Because of reconfigurable hardware, the technique is able to determine which functional unit should be employed in order to meet the overall design and resource constraints.

From the resource utilization/requirement point of view, one of the most efficient organizations for hardware resource usage is a traditional Von Neumann style processing configuration. The sequential execution of individual instructions allows the functional units (fetch, decode, and execute) that are implemented in hardware to be reused many times in a time shared manner. They are very flexible and support very irregularly structured computation, but suffer from the performance constraints associated with highly sequential operation. Fortunately, the current state of commercially off-the-shelf reconfigurable devices allows multiple sizable Von Neumann style processing core units to fit within commercial off-the-shelf reconfigurable hardware. Some of the notable examples are Altera's Nios soft processor and Xilinx's Microblaze soft processor [4,5]. This means that incorporating one or more such mainstream units within the reconfigurable hardware, along with other function specific hardware modules, may be a viable option to solve the space/time trade-off. Unfortunately, the task of finding the combination of functional units that meets the performance and resource constraints is really complicated. In fact, it is an NP-Complete problem.

In this paper, reconfigurable hardware is used to implement both the high-speed logic that has been designed to execute the most time intensive portions of the application problem and traditional Von Neumann style processing cores to save valuable hardware resources. In this arrangement, the processor cores, the application specific modules, the input/output logic, and the routing between each of these hardware entities are contained within the finite resources of the reconfigurable logic. The trade-off is to determine the number and types of each of these entities that will best meet the needs of the application and fit within the available reconfigurable hardware resources. Simply placing all the functionality in application specific modules will probably never represent a valid solution because of the finite resource constraints. At the same time, placing all the functionality in a single large processing core unit that will be implemented within the reconfigurable logic is also not desirable, since it is subject to poor performance. The goal of this paper is to show solutions to this space/time trade-off in the cases where the application can be decomposed into a well behaved system of tasks that can be implemented directly in hardware or executed sequentially on one or more processing cores. The solution presented here employs developing parallel processing style static scheduling techniques that can be applied to this finite resource reconfigurable hardware environment. It is believed that insight gained through this well defined problem domain can be extrapolated and extended to allow formalisms to be developed for less deterministic problems such as those associated with dynamically reconfigurable/schedulable systems.

[1] Kai Hwang, Advanced Computer Architecture Parallelism Scalability Programmability, McGraw-Hill Inc., New York, 1993.

[2] M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," San Francisco: W. H. Freeman, 1979.

[3] S. M. Loo, B. Earl Wells, "Applying Stochastic Static Task Scheduling to a Reconfigurable Hardware Environment," submitted to International Journal of Computer and Their Applications, July 2003.

[4] Altera Nios Processor, http://www.altera.com/nios

[5] Xilinx MicroBlaze Processor, http://www.xilinx.com/microblaze