

Programmable Logic IP Cores in SoC Design: Opportunities and Challenges

Steven J.E. Wilton and Resve Saleh
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, B.C., Canada
{steve, res}@ece.ubc.ca

Abstract

As SoC design enters into mainstream usage, the ability to make post-fabrication changes will become more and more attractive. This ability can be realized using programmable logic cores. These cores are like any other IP in the SoC design methodology, except that their function can be changed after fabrication. This paper outlines ways in which programmable logic cores can simplify SoC design, and describes some of the challenges that must be overcome if the use of programmable logic cores is to become a mainstream design technique.

Introduction

Recent years have seen impressive improvements in the achievable density of integrated circuits. In order to maintain this rate of improvement, designers need new techniques to handle the increased complexity inherent in these large chips. One such emerging technique is the System-on-a-Chip (SoC) design methodology. In this methodology, pre-designed and pre-verified blocks, often called *cores* or *intellectual property (IP)*, are obtained from internal sources or third-parties, and combined onto a single chip. These cores may include embedded processors, memory blocks, or circuits that handle specific processing functions. The SoC designer, who would have only limited knowledge of the structure of these cores, could then combine them onto a chip to implement complex functions.

No matter how seamless the SoC design flow is made, and no matter how careful an SoC designer is, there will always be some chips that are designed, manufactured, and then deemed unsuitable. This may be due to design errors not detected by simulation or it may be due to a change in requirements. This problem is not unique to chips designed using the SoC methodology. However, the SoC methodology provides an elegant solution to the problem: one or more programmable logic cores can be incorporated into the SoC. The programmable logic core is a flexible logic fabric that can be customized to implement *any* digital circuit *after fabrication*. Before fabrication, the designer embeds a programmable fabric (consisting of many uncommitted gates and

programmable interconnects between the gates). After the fabrication, the designer can then program these gates and the connections between them.

Several companies already provide programmable logic cores [1,2,3,4,5,6,7]. Yet, the use of these cores is still far from mainstream. In this paper, we first describe what a programmable logic core could look like. We then show how these cores can be used to improve both the design time and overall quality of SoC designs. Finally, we outline some of the challenges that need to be overcome before the use of these cores becomes mainstream.

Reconfigurable Cores

Figure 1 shows a hypothetical SoC-style design containing a programmable logic core. The chip consists primarily of fixed function IP (Intellectual Property) cores, likely obtained from a third-party. In this particular example, the fixed cores include an embedded processor, some on-chip memory, and two other fixed-function cores. In addition, the chip contains a programmable logic core. Unlike the other cores, the function implemented by the programmable logic core need not be defined until after the chip is fabricated. Instead, the programmable logic core contains a set of uncommitted gates, surrounded by a set of programmable interconnects.

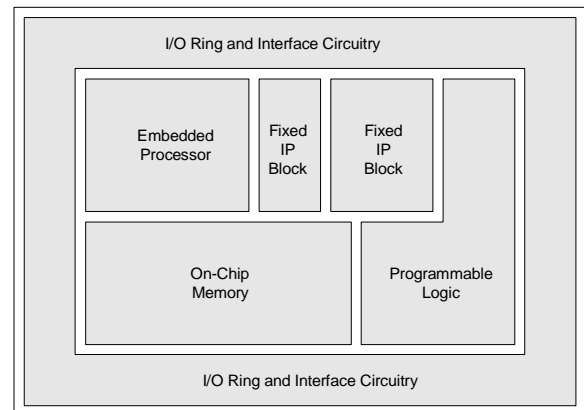


Figure 1: Example SOC-style design

The structure of the programmable logic fabric itself may look very much like the structure of a Field-Programmable Gate Array (FPGA) [8]. Typically, logic is implemented in an FPGA in small 16-bit memories, called *lookup-tables*. Several lookup tables and flip-flops are grouped into *clusters* (called Configurable Logic Blocks by Xilinx and Logic Array Blocks by Altera). The clusters are interconnected using fixed metal tracks; these tracks can be connected to each other and to clusters using configurable switches (usually implemented as pass transistors or repowering buffers in conjunction with pass transistors). Thus the configurability provided by a fabric is two-fold: the function implemented by each look-up table can be configured, and the interconnect between the fixed routing tracks and the look-up tables can be configured.

The configuration information is typically stored in a set of SRAM bits (in some cases, anti-fuses are used rather than SRAM bits). The values in these SRAM bits determine what function each lookup table implements and how they are connected. The SRAM bits can be configured into one or more shift registers, and, once the chip is powered up, can be loaded by the user. Besides providing the ability to customize the circuitry after fabrication, the use of SRAM bits allows the designer/user to re-configure the core as many times as desired. For a more detailed description of programmable logic circuitry, see [8].

Promises of Reconfigurability

This section describes how a reconfigurable fabric can be used within an SOC-style chip, and outlines four scenarios in which the use of a reconfigurable core is compelling.

Scenario 1: Changing Requirements

Consider the task of designing a network interface chip, for which the network interface protocol has yet to be finalized. In today's competitive market it is common that a chip is developed before standards have been finalized. A changing standard late in the design cycle may have a very negative impact on the development schedule of the product. If, however, the logic that implements the network interface protocol is implemented using programmable logic, then these last-minute changes can be accommodated without changing the chip design.

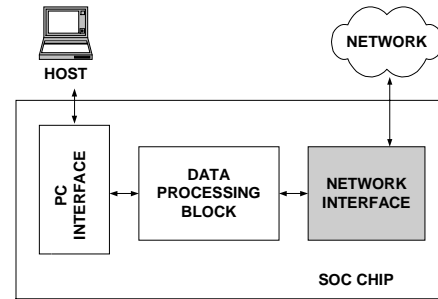


Figure 2: A hypothetical SOC which can be modified after fabrication

Consider Figure 2. This shows an implementation of a hypothetical PC-to-network bridge. Most of the chip (the PC interface and the data processing block) is designed using either fixed ASIC logic or fixed-function cores. The network protocol, however, is left to programmable logic.

Scenario 2: Customization and New Features

In many cases, a company designs a chip that will be used by several customers. These customers may have unique requirements. One option is to build all the features required by all the customers in the chip, and then provide a mechanism for selectively enabling certain features for certain customers. Another approach, enabled by using a programmable logic core, is to implement the customer-specific portion of the chip using programmable logic.

Scenario 3: Platform-Based Design

Due to the difficulties inherent with system-on-a-chip design, some CAD tool vendors are now offering an alternative: platform-based design. In this design style, a CAD tool vendor will work with the customer to produce a "platform", a set of pre-placed, pre-wired, and pre-verified cores for a specific application. The customer can then customize the platform to suit a particular application. For example, a CAD tool vendor might work with a customer to produce a network processor platform. The CAD tool vendor would obtain cores, place them on a chip, verify them, and verify their interconnect. The result would be a generic network processor which could then be customized by the customer. In addition, the CAD tool vendor could re-use non-proprietary sections of the platform for future customers who also wish to build a network processor.

There are two ways in which programmable logic can fit into the platform-based design approach. First, the

CAD tool developer might find that the platform requirements are slightly different between customers. These differences could be encapsulated in programmable logic. This would greatly simplify the task of modifying a platform for a second customer, if all that needs to be changed is the circuitry implemented in programmable logic. A second, and possibly more compelling possibility is that the CAD tool developer might implement the entire platform using fixed cores, and fixed ASIC logic. The remaining customization by the customer can then be done using programmable logic. In this way, the customer has to do no chip design at all. The CAD tool developer is now doing the entire chip design, and selling a generic chip which can be customized.

Scenario 4: On-Chip Testing

One of the greatest challenges in building SoC design is making them testable. Although, standard design-for-testability techniques can be used on each core individually, it is unclear how to combine the various test structures so that the entire chip is testable. Figure 3 shows one solution, based around a programmable logic core. In this hypothetical chip, there are three cores, two with a scan chain and one with external buses (which can be driven or monitored to enhance observability or controllability). In addition, the chip contains an on-chip jitter-measurement circuit and some circuits used to measure precise timing delays to help with circuit calibration. This chip also contains a programmable logic core to tie these various controllability and observability functions together. The test engineer can programmably, after fabrication, implement a circuit in this core which combines various test results into a signature, which can then be analyzed by high-speed test equipment. At the same time, the circuit implemented in the programmable logic core can be used to stimulate the circuit. The fact that the test circuitry does not need to be designed before the chip is fabricated is key: the same programmable core can be used to implement many different test analysis and stimulus circuits, each testing a part of the chip. As testing proceeds, if errors are found, new tests can be devised, and the on-chip test circuitry implemented in the programmable core.

Challenges and Issues

In order for programmable logic cores to become mainstream, there are several challenges and issues that must be addressed.

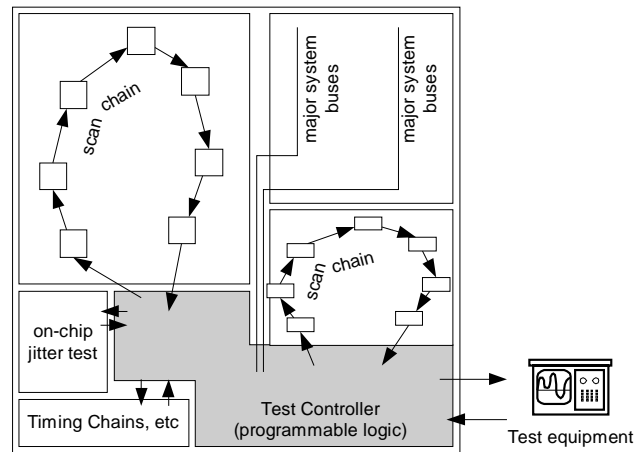


Figure 3: SOC-style chip with a test controller implemented using programmable logic

Architecture Issues

There are two ways a vendor could provide programmable logic cores to an SoC designer. One approach is to offer a “family” of cores, each member of the family having a different capacity, chip area, shape, and other characteristics. Because there are many possibilities for the shape and size of these cores, a large library would be required. A second, more flexible approach, is to provide a “programmable logic generator”, similar to a memory generator. Using this generator, the SoC designer can request a core with certain properties (shape, size, number and type of interface pins etc.), and the generator would automatically create a programmable logic core that meets the user’s specifications. This second approach has the advantage that many more different shapes and sizes can be created; it is more likely that a core that meshes well with the rest of the fixed cores can be used.

Regardless of whether the cores come from a library or are automatically generated, the development of non-square programmable logic cores is challenging. Although there has been much previous work on programmable logic optimization, most of this work has targeted FPGAs, which are typically square. It is known that highly rectangular arrays prefer a routing architecture with more tracks in the long direction than in the short direction [9], however it is not known how to create routing architectures for L-shaped cores (as in Figure 1) or for irregularly shaped cores (as in Figure 3). If the routing architecture is not carefully designed,

it is likely that “choke points”, or highly congested regions may be created, leading to a core that is unroutable for many applications.

Pre-Fabrication Verification

Perhaps the most challenging issue will be that of pre-fabrication verification. At the time an SoC with a programmable logic core is designed, the circuitry that is to be implemented in the programmable portion is unknown. To perform a system-level simulation of the entire SoC, at least a behavioural description of at least one possible configuration of the programmable logic core is required. For example, when simulating the interface chip of Figure 2, it is necessary to have a behavioural description of at least one implementation of the network interface in order to simulate the entire chip. A more thorough verification job would verify the circuit with a number of different network interface circuits. Thus, there is a tradeoff between the amount of verification and the time needed to perform the verification. A study of this tradeoff is essential.

For some verification tasks, a behavioural description will not be sufficient. Consider the case of power grid sizing. This task involves the analysis of the power requirements of each core, an IR-drop analysis, and the creation of a power grid that can meet these power requirements. In a programmable logic core, however, the amount of power required depends on the digital logic implemented in the core. To ensure that sufficient power is supplied to the core, it is likely necessary to take into account the *worst-case* power requirements of the core. The development of worst-case power models for these cores needs to be created.

Timing analysis is another important issue. The load of wires entering and exiting a programmable logic core depends on the digital logic implemented in the core. One option is to create a fully buffered (and likely registered) interface between the programmable logic and the fixed logic. The development of such an interface, and its interaction with a timing analysis tool is an important research area.

Post-Fabrication Verification

Once the chip has been fabricated, the designer must create logic to be implemented in the programmable logic core. Although the core can be reprogrammed many times, development of this logic will be simplified if simulation models of the rest of the SoC are available. These simulation models may be already be provided by the core vendors, but their integration

into a system which can simulate both fixed and programmable logic is an open research area. In the end, it may be necessary to have programmable IP with programmable on-chip test structures to fully exploit this approach.

Applications

A fourth issue that must be studied is the identification of which sorts of applications can make use of a programmable logic core, and how they can use it.

Summary

It is clear that the ability to make post-fabrication changes will become essential as time-to-market pressures become increasingly tight, and as the complexities of integrated circuits increase. Programmable logic cores is a natural way to support these post-fabrication changes. One of the primary ways in which these cores will be used is to allow designers to leave certain aspects of the design unspecified until after fabrication. The use of these cores in platform-based design and as a programmable test controller may also be very valuable.

Although these programmable cores have been available for approximately a year, their use is not yet mainstream. The verification issue is an important concern that must be addressed. However, possibly the most important issue will be how the cores can be used within specific applications. Programmable logic adds another dimension that designers must come to grips with before the full potential of these cores can be realized.

References

- [1] C. Matsumoto, “LSI Logic ASICs to add Programmable Logic Cores”, *E.E. Times*, August 29, 1999.
- [2] S. Ohr, “ADI Taps Systolix Processor Array,” *E.E. Times*, April 21, 2000.
- [3] “Lucent Introduces ORCA Series 4 FPGA,” *Programmable Logic News and Views*, pages 7-11.
- [4] R. Merritt, “QuickLogic Steps up Merger of FPGA with IP Cores – DSP First Target,” *E.E. Times*, August 9, 2000.
- [5] “Embedded FPGA Cores Enable Programmable ASICs, ASSPs,” *E.E. Times*, September 20, 1999.
- [6] C. Matsumoto, “Actel Plans to Produce FPGAs as ASIC Cores,” *E.E. Times*, June 5, 2000.
- [7] C. Matsumoto, “Startup Puts a Fresh Spin on Programmable Cores,” *E.E. Times*, September 15, 2000.
- [8] V. Betz, J. Rose, A. Marquardt, “*Architecture and CAD for Deep-Submicron FPGAs*,” Kluwer Academic Pub., 1999.
- [9] V. Betz and J. Rose, “Directional Bias and Non-Uniformity in FPGA Global Routing Architectures,” *ICCAD*, 1996, pages 652 - 659.

