Automated Method to Generate Bitstream Intellectual Property Cores for Virtex FPGAs

Edson L. Horta[1] and John W. Lockwood[2]

[1] Department of Electronic Engineering, Laboratory of Integrated Systems, EPUSP
edson-horta@ieee.org,
[2] Department of Computer Science, Applied Research Lab, Washington University,
lockwood@arl.wustl.edu,
http://www.arl.wustl.edu/arl/projects/fpx/parbit

Abstract. This paper presents an innovative way to deploy Bitstream Intellectual Property (BIP) cores. By using standard tools to generate bitstreams for Field Programmable Gate Arrays (FPGAs) and a tool called PARBIT, it is possible to extract a partial bitstream containing a modular component developed on one Virtex FPGA that can be placed or relocated inside another Virtex FPGAs. The methodology to obtain the BIP cores is explained, along with details about PARBIT and Virtex devices.

1 Introduction

Field Programmable Gate Arrays (FPGAs) available now contain millions of equivalent logic gates. Large systems are implemented on these FPGAs by integrating multiple intellectual property blocks. There are essentially three types of intellectual property blocks, also called IP cores [1]. The first, called soft cores, are blocks delivered as Verilog or VHDL code. The chip developer is responsible for synthesizing and implementing the logic into FPGA's logic. The second, called firm cores, are circuits described in a netlist format, mainly EDIF, and contain some placement information. These are used by the chip developer to implement the core in the FPGA. The third, called hard cores, are placed and routed blocks ready to be used inside the FPGA. They can not be altered by the chip developer.

This paper presents a new approach for the generation of IP cores: Bitstream Intellectual Property (BIP) cores. The BIP uses a partial bitstream configuration file to deliver the IP core. The full bitstream files can be generated with commercial tools already available from Xilinx. PARBIT, a tool capable of extracting a partial bitstream from full bitstream files, enables the chip developer to generate, develop and test an IP core in isolation on one FPGA then deploy it into a region of another.

The methodology presented in this paper can also be used to develop new reconfigurable modules, which can be loaded into a great variety of reconfigurable platforms, such as [2]. The Field Programmable Port Extender uses one FPGA

to reconfigure the bitstream of another [3] [4]. The self-reconfiguring platform presented in [5] implements the reconfiguration control inside the same FPGA with its own logic resources. All of these approaches introduce a new level of flexibility for extensible systems. The configuration bitstream may come from a network or be compressed and stored on a device attached to the FPGA. The time required to generate new modules for such platforms can be reduced significantly using BIP cores.

2   Design Flow

PARBIT [6] processes two full bitstream files called original (containing the reconfigurable module) and target (containing the static logic). The tool has three operation modes: Slice, Block, and BIP. The slice and block modes can be used to perform partial run-time reconfiguration of an FPGA-based platform [7]. The BIP (Bitstream Intellectual Property) mode generates IP cores directly from bitstreams obtained through Xilinx tools [8]. To add a BIP core developed from an original device into a system being implemented on a target device, the source bitstream (containing the BIP core) is reformatted to configure a target device containing a reserved area for the BIP. Logic configuration and routing is specified for an area inside the CLB columns of the chip that exclude the top and bottom frame control bits of the FPGA's Input/Output Block (IOB). The area within the FPGA that defines the BIP core is extracted from the original design an merged into a target bitstream for another device of same family of FPGA. The tool generates the partial bitstream file containing the BIP core area and this file is used to program the core into the target device.

The interface between the reconfigurable module and the static logic is well defined. Modules can be placed into specific locations inside the FPGA. Interfaces between modules, called Gasket in [9], were redefined in [10] to be called a Bus Macro.

Just a few extra steps are needed to generate an original bitstream with the BIP core and a target bitstream that contains static logic with an open area reserved for the BIP core. The original bitstream is generated by the BIP core developer, and the target bitstream is generated by the final user.

The BIP core is defined as the reconfigurable area inside the original device, delimited by the coordinates: Start Row, End Row, Start Column and End Column. This FPGA will not have any logic outside the BIP core area.

Each Bus Macro, used to interface the BIP area with the static logic area of target device, bridges four signals, and occupies 8 CLBs. Four of the CLBs are located inside the reconfigurable area, and four outside. As shown in Figure 1, the connection points outside the BIP area are connected to the routed signals inside of the target device. The location of each one of the bus macro is fixed by a constraint command inside the UCF (User Constraints File) for the project.

A core developed with the BIP methodology consists of two files: the full original bitstream file, which is used by PARBIT to extract the configuration bits for the BIP core; and the Top Level VHDL, which contains the instantiation
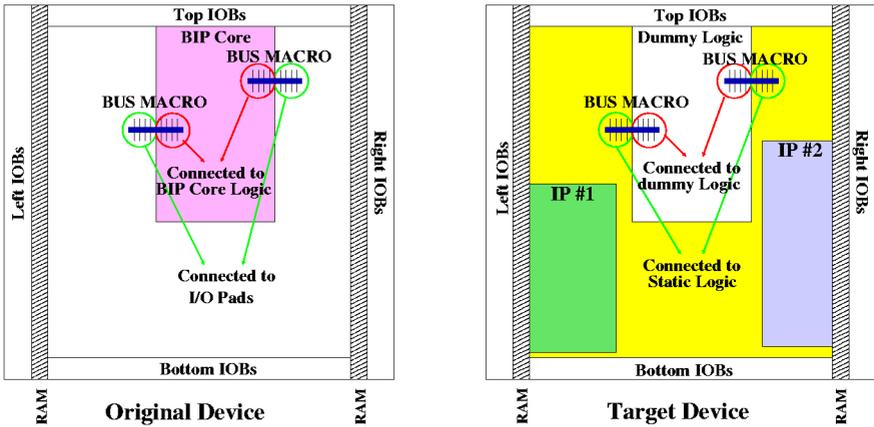
Fig. 1. BIP Original and Target

for the BIP core and the Bus Macro. The BIP width (CLB columns) and height (CLB rows) are added to VHDL in the form of comments. Likewise, the position of each interface signal, relative to the upper left corner of the BIP, is specified.

In order to perform run-time reconfiguration, the target bitstream would be reconfigured within the reserved area that receives the BIP core extracted from the original bitstream. The target bitstream contains static logic around the module that will not change during the downloading of the BIP core. This static logic may include other IPs, previously reconfigured into the FPGA, as shown in Figure 1.

It is assumed that an area was reserved that is the same size of the BIP core, and that the bus macros were placed in the same positions relative to the upper left corner of the BIP. It is not necessary to use the same device employed to generate the BIP bitstream file, provided that the BIP area fits inside the new device.

In order to keep the clock signals active to be used in the BIP core we insert Flip/Flops inside the reserved area to create the needed clock routing.

3 BIP Core Example

To demonstrate the methodology of using a BIP core, a reconfigurable calculator was implemented, based on an application note from Xilinx [10]. This calculator has three modules: adder, lcd_driver and pushbutton. In our example, the lcd_driver and the pushbutton modules are already placed in the target FPGA, and the adder module is used as a BIP core.

First, the size of the BIP core is determined and used to set the position of each bus macro. Six bus macros were used for the original 'calc' design: three of them are used between the lcd_driver and the adder; the other ones to interface

the adder within the pushbutton module. For this example, the BIP core (adder) is confined in the region delimited by: Start Column = 5; End Column = 18; Start Row = 1; End Row = 16. Bus macros are placed on rows 5, 10 and 15, horizontally centered on the left and right sides of the BIP.

To implement and test the BIP adder core, an FPGA circuit was implemented with the adder connected through external bus macro signals to the FPGA I/O pins. The FPGA with the adder module was then synthesized, implemented and simulated with the BIP core in isolation.

Figure 2 shows the floorplan for the BIP core. This core is confined in a region containing 14 columns with 16 rows. The figure shows that only the bus macro and clock routing signals cross the boundary region. Because clock routing do not depend on the CLB configuration frames, they are allowed to cross the boundary region.
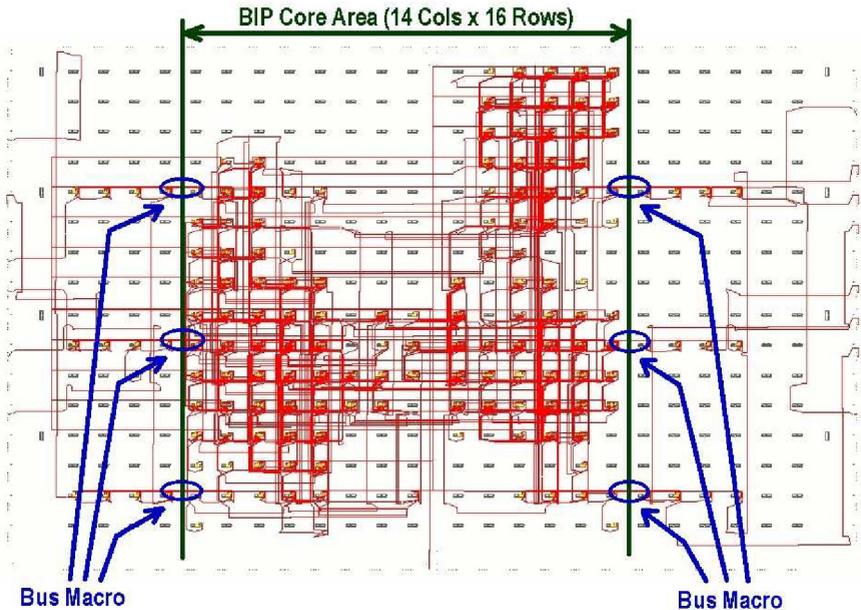

Fig. 2. BIP Core - XCV50

The target device was implemented with the lcd_driver and the pushbutton modules placed on the left and right sides of the chip, respectively. An empty space for the adder module was left by connecting a dummy circuit with Flip/Flops to internal bus macro signals. These flip/flops generate a clock routing template, which makes the clock signals available to any BIP core implemented in the target device. To ensure that all clock signals were routed across the 33 rows of the routed chip, a BIP core with 14 columns and 33 rows was inserted into this region.

4   Conclusions

A new way to generate an IP core was presented, utilizing existing Xilinx tools and PARBIT. The method uses bitstream files generated by commercial tools, and provides a secure method to deliver the IP.

When used with the Xilinx ISE V5.2i tools, the BIP core presented allows the mapper and place and route tools to run 8 times faster than without using a BIP core because the original circuit on the smaller XCV50 routed and placed much faster than when targeting directly the much larger XCV600 FPGA. Thus, the methodology presented in this paper can shorten the time necessary to design new reconfigurable modules for large FPGAs. With a BIP core, a small device can be used to generate and test a module before the implementation onto a larger target device.

References

1. VSIA: VSI Alliance Architecture Document. VSI Alliance, Online `http://www.-vsi.org/resources/techdocs/vsi-or.pdf` (1997)
2. Kalte, H., Porrmann, M., Rückert, U.: A Prototyping Platform for Dynamically Reconfigurable System on Chip Designs. In: IEEE Workshop Heterogeneous reconfigurable Systems on Chip (SoC), Hamburg, Germany (2002)
3. Lockwood, J.W.: An open platform for development of network processing modules in reprogrammable hardware. In: IEC DesignCon'01, Santa Clara, CA (2001) WB–19
4. Lockwood, J.W., Turner, J.S., Taylor, D.E.: Field programmable port extender (FPX) for distributed routing and queuing. In: ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000), Monterey, CA, USA (2000) 137–144
5. Blodget, B., James-Roxby, P., Kelle, E., McMillan, S., Sundararajan, P.: A self-reconfiguring platform. In: Field-Programmable Logic and Applications / The Roadmap to Reconfigurable Computing (FPL'2003), Lisbon, Portugal (2003) 565–574
6. Horta, E., Lockwood, J.W.: PARBIT: a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAs). Technical Report WUCS-01-13, Washington University in Saint Louis, Department of Computer Science (July 6, 2001)
7. Horta, E.L., Lockwood, J.W., Taylor, D.E., Parlour, D.: Using PARBIT to implement Partial Run-Time Reconfigurable Systems . In: 12th International Conference on Field Programmable Logic and Applications, Montpellier, France (2002)
8. Xilinx, I.: Foundation series user guide. http://toolbox.xilinx.com/docsan /xilinx6/pdf/docs/fsu/fsu.pdf (2004)
9. Horta, E.L., Lockwood, J.W., Taylor, D.E., Parlour, D.: Dynamic hardware plugins in an FPGA with partial run-time reconfiguration. In: Design Automation Conference (DAC), New Orleans, LA (2002)
10. Lim, D., Peattie, M.: Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations. Xilinx XAPP290 (2002)