Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

Designing for Xilinx XC6200 FPGAs

Reiner W. Hartenstein, Michael Herz, Frank Gilbert

University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de
www: http://xputers.informatik.uni-kl.de

Abstract. With the XC6200 FPGA Xilinx introduced the first commercially available FPGA designed for reconfigurable computing. It has a completely new internal architecture, so new design algorithms and software is needed. Due to the fact that most applications are in the research area, the number of sold units seems to be small. Because of this progress of design tools for this architecture is rather low. This paper discusses the problems, which appear during designing for the XC6200 FPGAs. A dedicated design flow is presented and demonstrated on an example application.

1 Introduction

The XC6200 is an FPGA that has been designed to be used in two broad classes of applications. The first class is the conventional role of a general-purpose ASIC device for logic integration. The other role is that of an intelligent peripheral that can operate as a memory-mapped coprocessor in microprocessor systems. This is largely due to the advanced FPGA to CPU interface. The design philosophy appears to have been driven by the desire to produce a FPGA optimized for reconfigurable computing.
Designing for the XC6200 is similar to other FPGA families, but some limitations apply. The design software is still in a beta state and progress is slow, as the commercial use is low. Not all features are implemented yet and some are still buggy. Further routing of irregular structures is very difficult because of few routing resources of the target architecture. A lot of problems during application implementation have to be solved manually. Therefore most design flows directly start at gate level where directly the logic blocks of the FPGA are programmed (e.g. Lola [Lo98]). This is like a step back into the stone age of hardware design. To alleviate this drawback in this paper a dedicated design flow for the XC6200 FPGA family is proposed. A few steps appear similar to other FPGA technologies. But due to XC6200 specific problems each single step shows differences to other technologies. Designing consists of five steps: Design Creation, Logic Synthesis, Netlist Generation, Place and Route and Simulate Design with Timing Information. As stated before this looks familiar, although Netlist Generation is mostly a part of the Logic Synthesis step.
The paper is structured as follows. First in the next section the structure of the hardware is briefly introduced. After sketching the limitations of available design software the dedicated design flow is presented, which is mainly based on predefined macro cells. Its benefits are demonstrated with an generic 3x3 linear filter with configurable weights for image processing. This application benefits from the processor interface, which allows to change filter coefficients without reconfiguration of the complete device. To simplify the design process the control part of the design is synthesized. Performance results will justify the introduced method.

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

2 Overview on the Xilinx XC6k and the VCC HOT Works Board

Architectural Overview on the XC6200 Series [GL97]

The XC6200 is based on a fine-grained, sea-of-gates architecture. The XC6216 consists of an array of 64 x 64 core cells surrounded by 256 input-output ports. Every logic cell can implement any combinatorial logic function of two inputs. Each cell can also implement a D-type flip-flop which can be used to register the cell's combinatorial function.
Cells have nearest neighbor connections to their North, South, East and West neighbors. The device has a hierarchical busing scheme. Cells are organized into blocks of 4 x 4, 16 x 16, and so on, increasing by a factor of four each time. A set of fast buses is associated with each size of block. Access to these fast buses is via routing switches that are adjacent to the cells on periphery of the respective blocks.
The processor interface is a 32-bit wide data bus that may also be configured for 16 or 8 bit operation. The XC6200 has been designed to appear in system as random access memory. All data registers on the array are accessible, making it possible to interface with user logic via the processor interface alone. Registers are addressed in columns via a map register. Up to 32 of the 64 registers in column may be read from or written to by nominating their appropriate row position in the map register. This can be extended so that all of the 64 registers in a column may be written by a single 8-bit operation. This is particularly useful in reconfigurable computing applications.
For further information, please refer to the Xilinx XC6200 datasheet at [Xi98a] and some application notes at [Xi98b].

The HOT Works PCI-XC6200 Development System

The proposed design-flow and the implemented examples are tested on the HOT Works PCI Board by Virtual Computer Corporation [Vc98]. The board architecture allows the XC6200 to be accessed by a host CPU via the PCI-bus. The board consists of:

- a XC6216 for the user-designs
- a XC4013 implementing the PCI bus interface
- up to 2 MBytes of fast SRAM for user-data
- a programmable clock-generator.

The XC6216, the SRAM and a set of configuration registers are mapped to the memory space of the host CPU. This allows the host CPU to read or write the SRAM memory of the board and configure or read or write the user FPGA and the user design. For an introduction to the XC6200 Development System refer to [NG97], further information can be found at [Xi97a].

3 Design-Flow for the Xilinx XC6200

This chapter will describe a VHDL-based design flow for the XC6200. First, all parts of the development process (design libraries and software) are described. Then the general design flow, the use of the software, problems and limitations concerning the single design steps are discussed.
The design environment consists of five parts (figure 1). First, a VHDL-simulator is needed to validate the VHDL design description. Then behavioral VHDL must be synthesized in primitive gates of the target technology, therefore a synthesis tool with a corresponding technology library is required. The output format of the synthesis tool will be VHDL. To transform the VHDL netlist in the EDIF-input-format of the place-and-route tool a small program will be used. The place-and-route software is the last tool to mention in the design environment.

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

Technology Library

Basic part of the development process is a technology library. The primitives of this library are any two-input gate functions, any 2:1 multiplexer, constant 0 or 1, buffer, inverter and D-type register. Technology libraries for the XC6200 family exist for the Viewlogic schematic entry tool and the Synopsys Design Compiler. In our design environment Synopsys is utilized.

Using a textural description of the target technology, the Synopsys Library Compiler generates, in addition to a primitive library for the Synopsys Design Compiler, also a VITAL [Vi95] compliant technology library for VHDL-simulation and back-annotation. The VITAL library provides behavioral models of all primitive gates with default timing. The default timing can be overridden by exact timing values calculated by the Place-and-Route-software using SDF data (Standard Delay Format). The Synopsys library is included in the SunOS version of XACT Step 6000 only.

Simulating the design

As the synthesis primitive library can be used only with the Synopsys Design Compiler, the VITAL library is vendor independent. The VITAL library can be simulated with any VHDL simulator such as Synopsys' VSS, Model Technology's V-System or Mentor Graphics' QuickHDL. V-System and QuickHDL have two advantages compared to VSS. First both provide a VHDL foreign language interface to their simulators. This feature can be used for co-simulation of hardware and software [Xi97a]. Then, both support VHDL'93 standard, which is necessary for the coding technique shown in the following. Because of this QuickHDL it is chosen for simulation. Setting up the VITAL library for QuickHDL simulation is similar to VSS, described at [Xi97a]. For information about QuickHDL design libraries refer to [Me97].

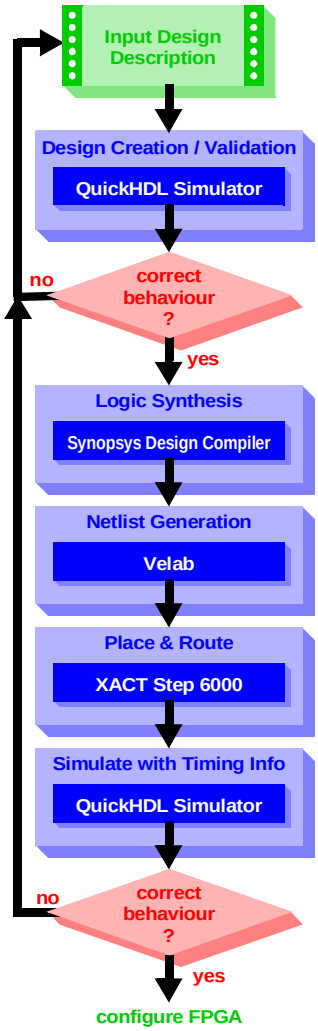

Fig. 1. XC6200 Design Flow

Logic Synthesis

As a technology library of the XC6200 family is provided for Synopsys Design Compiler, logic synthesis of behavioral VHDL can be done with some limitations. First the Synopsys Design Compiler can not synthesize pad-cells, which are needed for user IOBs (Synopsys creates only input/output-buffers). Therefore a top-level structural description of the I/Os has to be provided manually by the designer. Second, some design information such as the pinout or placement can only be attached by user defined attributes, which are not supported by Synopsys Design Compiler.

Another big disadvantage of logic synthesis is a problem of the Xilinx Place-and-Route software XACT Step 6000. Placement and routing of larger non-hierarchical designs without placement information results in bad designs. Hierarchical designs with preplaced macro-cells for adders, subtractors and other regular structures are necessary to allow manual floorplanning. As a consequence, only pure state-machines without

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

datapath (<100 primitive gates) should be synthesized with Synopsys. Synthesis of RTL-VHDL of larger designs (>1000 primitive gates) as for other FPGA technologies is not practicable at the moment.

Netlist Generation

For the datapath a well structured hierarchical design is necessary. Cells of regular structure, such as adders and multipliers, should be preplaced using user-defined attributes and instantiated in larger cells such as pipelines. The result of this design style is hierarchical VHDL-netlist of instantiated components and technology primitives. The input netlist format of XACTStep 6000 is EDIF. To transform VHDL-netlists to EDIF a small tool called VELAB is used [Xi97b]. VELAB is a free VHDL analyser and EDIF netlist generator for the XC6200 family provided by Xilinx. One of its major features is the ability to generate parametrized attributes.

Placement and Routing

XACTStep Series 6000 [Xi96] is a graphical tool for XC6200 family designs. This system is a back-end tool with EDIF as its primary input. The XACTStep Series 6000 editor preserves the hierarchy of the input design. This hierarchy information is used to support both top down design through floorplanning and bottom up design through either manual or automatic techniques. In addition, fully automatic place-and-route is supported. The graphical editor gives full access to all resources of the XC6200 family architecture [NG97].

In practice, the automatic techniques need a lot of manual assistance. Even when using preplaced structures floorplanning and manual placement is unavoidable. The automatic router often fails in routing the top-level design. Therefore placement and routing is not an automated design step as for most commercial FPGA technologies. A lot of manual work and design experience is essential for acceptable results. Timing constraints can not be set, timing analysis of the post-layouted design has to be done interactively by choosing source and destination cells in the graphical editor.

Design Step	Design Tool	Version
Design Validation (1), Simulation (5)	**Mentor Graphics QuickHDL**	**v8.5_4.6c**
Logic Synthesis (2)	**Synopsys Design Compiler**	**1997.08**
Netlist Generation (3)	**Xilinx Velab (freeware [Xi97b])**	**0.52**
Place & Route (4)	**Xilinx XACT Step 6000**	**1.1 beta build 4**

Table 1. Summary of the used design software

3.1 The XC6200 Specific Design Flow

As mentioned in the introduction the design flow for the XC6200 family consists of five steps (figure 1): Design Creation, Logic Synthesis, Netlist Generation, Place and Route and Simulate Design with Timing Information. In the following the architecture specific characteristics of this design flow will be explained by treating each design step in detail.

Design Creation / Validation

First, the design is described and a testbench is written in VHDL. Then the design is simulated and its specification is validated until the behavior of the design is satisfactory. The design has to be specified in a synthesizable subset of VHDL (RTL). The testbench may also include VHDL-statements which are not synthesizable.

Unfortunately the Xilinx place and route software XACT Step 6000 is unable to place and route designs of logic-cell usage larger than 15% to 25%. To handle larger designs,

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

floorplanning and manual placement must be done. To support floorplanning regularity and hierarchical information is necessary. In addition, very regular substructures such as adders, multipliers and other operators should be preplaced. In the applied design methodology for the XC6200 family, a design is partitioned in three entities: a top-level entity, a control unit (finite state machine) and a datapath unit. The datapath is restricted to primitive gates and macro-cells of primitive gates. The macro-cells [XC98] used in the datapath unit are included in a predefined design library. This library may be extended by the designer if a specific macro is not available.

Figure 2 illustrates this design step. Design Creation is an iterative process of writing/modifying VHDL code and simulation. All parts of the design apart from the technology library may be edited. Even the macro-library may be extended, if new regular substructures are needed. Writing a testbench forcing all input ports and verifying automatically the results simplifies this iterative process. Validating the design needs no interaction if no errors occur. Unfortunately two versions of the top-level description are necessary. The reason is that it is not possible to simulate the behavior of a special type of register (RPFDs) and there is no functional equivalent for pad-cells. RPFDs (Register Protected D type Flip-Flop) can only change value by reconfiguration. To simulate the behaviour of a RPFD it must be replaced by a UP_RPFD, which is a RPFD with a simplified processor interface. In future releases of XACT Step 6000 the mapping software will automatically replace a UP_RPFD by an ordinary RPFD. All these cells are included in the VITAL simulation library delivered with the Xilinx software.

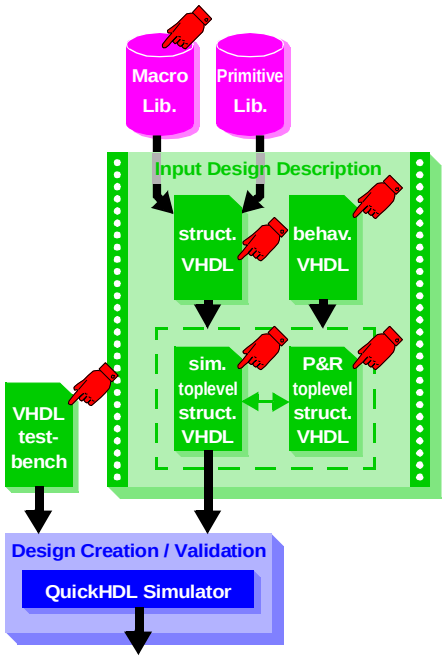

Fig. 2. Step1, Design Creation / Validation

If simulation results meet the design requirements, this step is finished. Further iterations may be necessary if the following design steps demand a redesign, e.g. timing requirements are not met or placement information has to be added for future enhancements.

Logic Synthesis of behavioral VHDL

This design step transforms the part of the design given in behavioral VHDL (e.g. finite state machines) into a structural description of technology gates. Using the Synopsys Design Compiler for the synthesis of XC6200 family technology gates is described in [Xi97c]. There a detailed introduction in setting up the Synopsys environment and a step by step explanation of the synthesis is given. However minor differences appear to our design flow:

1. First, no pad cells are needed, as they are instantiated in the top-level description.
2. The output format should be VHDL as the complete EDIF-netlist will be generated in the next design step with VELAB.

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

As indicated in figure 3 some modifications must be done. The netlist generated by the Synopsys Design Compiler will report errors when processed directly by VELAB. Synopsys writes a package declaration in the beginning of the netlist, which is not necessary and must be deleted, because it is not allowed in VELAB. Further, if ports of type std_logic_vector are used, Synopsys writes signal-assignments to a vector, which are also not accepted by VELAB. They need to be converted into bitwise assignments. To automate this modifications an awk-script [XC98] has been written. After this design step all design information consists only of structural VHDL description files.

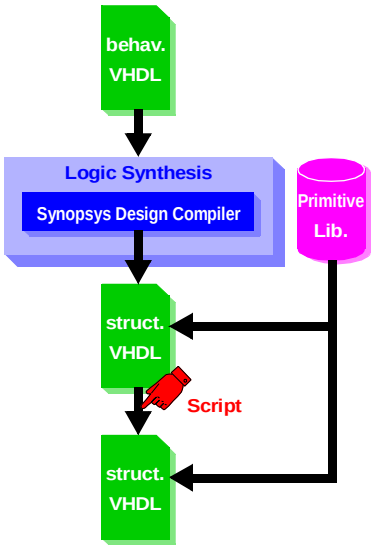

Fig. 3. Step 2, Logic Synthesis of behavioral VHDL

Netlist Generation with VELAB

VELAB is a free VHDL analyser and EDIF netlist generator for the XC6200 family. It can be downloaded from the Xilinx homepage at [Xi97b]. In this design flow, VELAB is used to generate a EDIF netlist for placement and routing and a second VHDL netlist for simulation (see figure 4).

The reason not to use Synopsys Design Compiler for netlist generation is, that it ignores all user defined attributes. But this is the only way to add placement information to the design. Macro library elements (e.g. adders, multipliers) need to be preplaced, because of the bad placement and routing results of the Xilinx XACT Step 6000. Preplacing is further useful if datapath registers have to be accessed via the processor interface. Adding placement information to user attributes is only supported by VELAB. VELAB accepts only a subset of VHDL, especially all structural parts. For a detailed description refer to [Xi97b]. One of the most useful features for preplacement of regular devices is its ability to handle parametrized attributes. This feature allows preplacement of devices with generic parameters such as size or layout. The macro-library used in the proposed design flow is based on this feature.

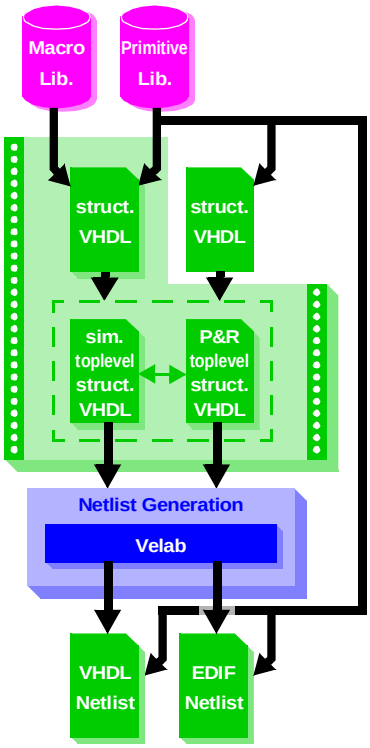

Fig. 4. Step 3, Netlist generation

As illustrated in figure 4 VELAB is used to generate two separate files. Therefore the top-level structural description for place-and-route (and of course all other design-files) are fed into VELAB to generate the EDIF netlist as an input to XACT-Step 6000. The top-level description for simulation is processed to a VHDL-netlist. This netlist is used for simulation of the placed and routed design with the exact timing parameters calculated by XACT Step 6000.

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

Place-and-Route

Placement and routing is an important step in the design flow. In contrast to other FGPA-technologies where floorplanning is optional, it is an essential part for most designs. XACT Step 6000 is a graphical tool allowing both manual floorplanning and automatic place and route. Unfortunately fully automatic place and route mostly fails or leads to bad results.

Figure 5 gives an overview to the place and route design step. The EDIF-netlist generated by VELAB is placed on the FPGA and all nets are routed. The configuration information of the FPGA is written to a CAL-file. A complete list of all nets and their corresponding delays are calculated for timing analysis. Setting up timing constraints for placement and routing and an automatic timing report is not supported. For a detailed analysis of critical paths source and destination nets are chosen from a list of all nets. The analysis information may be exported to a text file or used as an input to a standard spreadsheet. For timing simulation of the layouted design two other forms of timing extraction are supported, a delay table for the Viewlogic Simulator, and SDF (Standard Delay Format), which is used in most VHDL simulators. In the proposed design flow the SDF output is used as an input to the QuickHDL simulator. For the reason of bugs in both the VITAL technology library of the XC6200 family and the SDF-writer of XACT Step 6000 some modifications of the SDF-file are necessary. These bugs may be fixed in a future release of XACT Step 6000, meanwhile an awk-script [XC98] does the modifications.

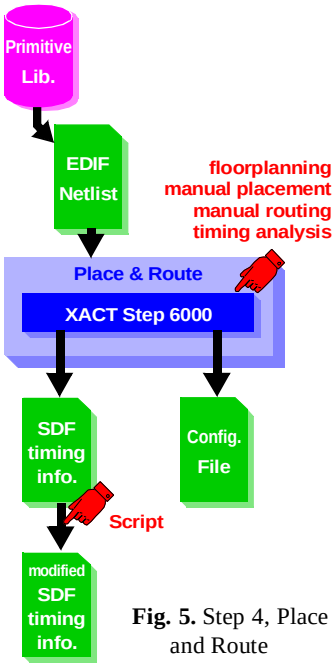

Fig. 5. Step 4, Place and Route

XACT Step 6000 is based on a bottom-up strategy. That means that one level of hierarchy is first placed and then routed, when all units of the lower design level are already placed and routed. In most cases manual floorplanning is necessary to achieve adequate results. The basis of the proposed method is the library of preplaced macro-cells for design units of high regularity such as adders or multipliers. Using this library elements avoids to floorplan at gate level. Because unstructured elements such as the synthesized FSM are more difficult to place, it is recommended to simplify the control logic as much as possible. Straight and short connections between the design units is the major goal of manual floorplanning.

Straight forward bottom-up placement is not recommended because of a placement must be found which simplifies not only the connections of the actual design level but also of the higher design hierarchy levels. That may be one of the major problems of a automatic place-and-route procedure. If the nearest-neighbor connection is not sufficient, higher routing levels such as length-4 or length-16 must be used restricting the possible placement on the next design hierarchy level. Therefore it is recommended to route the design on the top hierarchy level (except for the preplaced macro-cells).

Manual routing of the design is not well supported by Xilinx software. To completely route a design the sequence of the routed nets is an important issue. The order chosen by the software often leads to bad results. A good order would be to start with the obvious connections such as the neighbor-connections. Next, all critical nets should be routed. Critical nets include the timing-critical nets and the nets which are difficult to

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

route or even failed during automatic routing. Then all nets connecting I/O-cells should be routed. Last, all other nets are routed.
Note that the placement & routing guidelines reflect our experience with XACT Step 6000. They may not be best for all designs. As a consequence of using the above method only a limited design utilization can be achieved because the elements of the implemented macro-library are optimized for timing rather than high device utilization.

Simulating Design with Timing Information

After placement and routing of a design the exact timing-parameters are known. The VHDL-netlist generated by VELAB and the SDF-timing-information calculated by XACT Step 6000 can be simulated again using the QuickHDL-simulator (see figure 6). If a testbench has been created in step 1 it may be used here again simplifying this design step. If the testbench validates all design results and the simulator does not report any timing conflicts the design is correct under the tested conditions and the FPGA may be configured. Timing violations may be corrected by a different placement and routing leading back to step 4, but if the design does not show the correct behavior the whole design process starting with step 1 must be iterated.

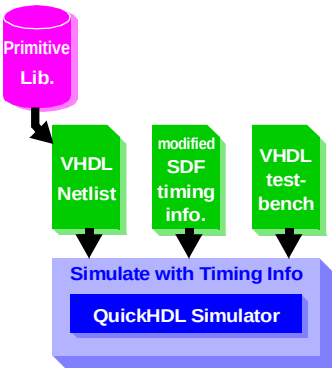

Fig. 6. Step 5, Simulate Design with Timing Information

4 Generic 3x3 Linear Filter for Image-Processing Example

The generic 3x3 linear filter processes an image by moving a 3x3 window over it (figure 7) and applying the following formula:

$$p_0^{new} = \frac{1}{j} \cdot \sum_{i=0}^{8} p_i \cdot k_i$$

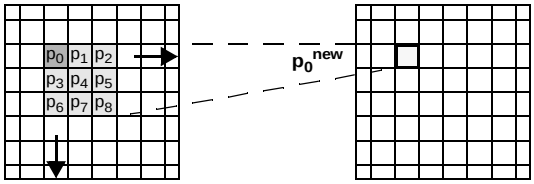

Fig. 7. Operation of a 3x3 Image-Transformation

This operation on an m*n-pixel image results in an image of size (m-2)*(n-2). All pixel-values are assumed to be an 8-bit grayscale value (0 to 255). Fig. 8 shows some filter coefficient examples.

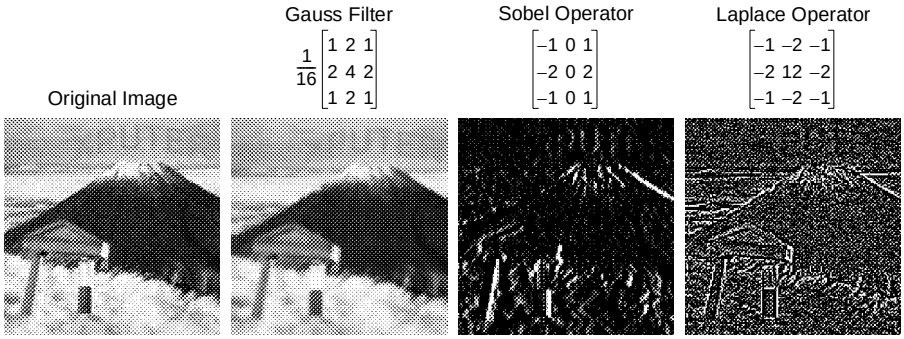

Fig. 8. Example Effects of Different 3x3 Linear Filter Operators

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

In this implementation of the filter, the coefficients k_i are signed integer numbers in the range from -16 to 15 (5 bit) and $j=2^n$ with n element of [0:8].
This leads to the dataflow-graph shown in figure 9. For multiplication of the coefficients k_i constant coefficient multipliers are used, which are included into the macro library. For macro implementation the description in [Xi97d] is modified for negative constants and higher device-utilization.

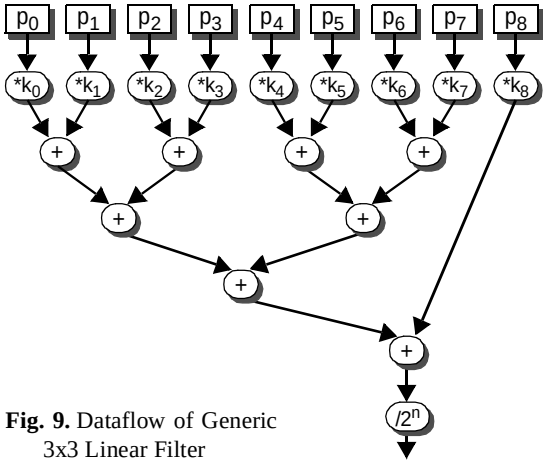

Fig. 9. Dataflow of Generic 3x3 Linear Filter

Fig. 10 shows the layout of the placed and routed generic 3x3 linear filter design. Because of the used preplaced macro cell elements, the constant-multipliers are always placed at the same location. Therefore the weights can be exchanged by only reconfiguring the related cells.
The design is working at a clock frequency of 25 MHz, where the computational pipeline is processing one pixel in two clock cycles. The design utilizes 2373 logic cells of the XC6216 (58%). Implementation details and more application examples can be found in [Gi98].

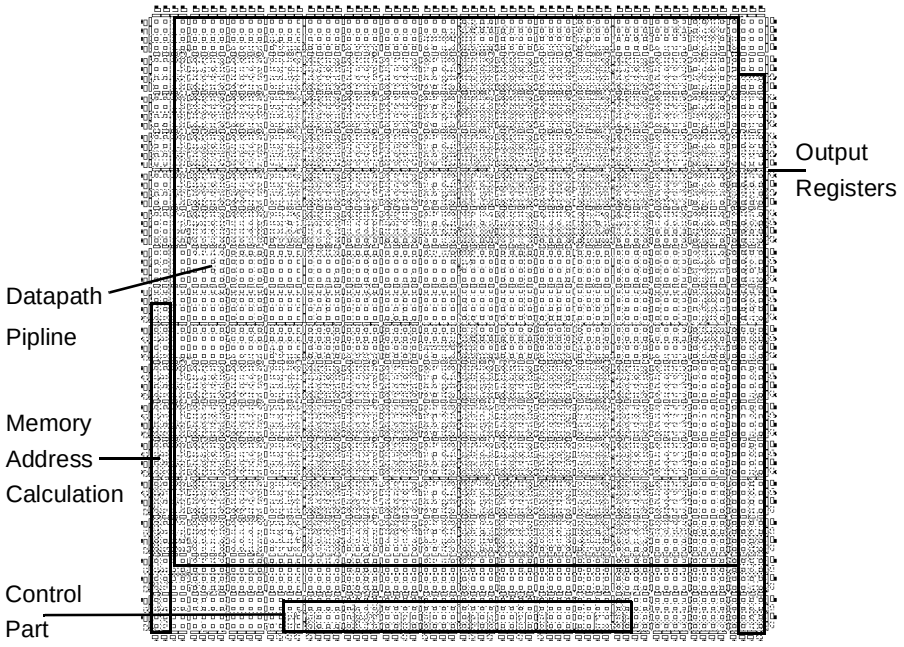

Fig. 10. Layout of the Generic 3x3 Linear Filter

5 Conclusions

A dedicated design flow for the Xilinx XC6200 FPGA series has been introduced to overcome problems of the available design software. The proposed method recommends partitioning of designs in a control part and a datapath. All regular structures of the datapath should be implemented with a macro library. This library has been implemented and is extendable for future designs. Its use has been proven on several examples [XC98], whereas an image processing example was presented.
Future work will be software support for dynamic reconfiguration and data access via the fast map interface.

6 References

[Gi98] Frank Gilbert: Development of a Design Flow and Implementation of Example Designs for the Xilinx XC6200 FPGA Series; Diploma Thesis, University of Kaiserslautern, Kaiserslautern, May 29, 1998. Download from [XC98].

[GL97] Gordon McGregor, Patrick Lysat: Extending Dynamic Circuit Switching to Meet the Challenges of New FPGA Architectures; Proceedings of 7th International Workshop, FPL'97 London, UK, Sept. 1997. LNCS 1304. Springer 1997

[Lo98] http://www.lola.ethz.ch/lola/

[Me97] Mentor Graphics Inc., QuickHDL User and Reference Manual, 1997

[NG97] Stuart Nisbet; Steven A. Guccione: The XC6200DS Development System; Proceedings of 7th International Workshop, FPL'97 London, UK, Sep. 1997. Lecture Notes in Computer Science 1304. Springer 1997

[Vc98] http://www.vcc.com

[Vi95] VITAL ASIC Modelling Specification; Draft IEEE 1076.4; New York October 1995; http://www.vhdl.org/vital

[Xi98a] http://www.xilinx.com/partinfo/6200.pdf

[Xi98b] http://www.xilinx.com/apps/6200.htm

[Xi97a] Xilinx Inc., Application Note XAPP 087: Co-Simulation of Hardware and Software, http://www.xilinx.com/xapp/xapp087.pdf, San Jose, CA, USA, 1997

[Xi97b] Xilinx Inc., Velab: VHDL Elaborator for XC6200, http://www.xilinx.com/apps/velabrel.htm, San Jose, CA, USA, 1997

[Xi97c] Xilinx Inc., Synthesis and simulation of a circuit in VHDL, using the Synopsys toolset, Synopsys_flow.pdf, XACT 6000 Synopsys Interface Documentation

[Xi97d] Xilinx Inc., Application Note XAPP 082: A Fast Constant Coefficient Multiplier for the XC6200, http://www.xilinx.com/xapp/xapp082.pdf, San Jose, CA, USA, 1997

[Xi96] Xilinx Inc., XACT Step Series 6000 User Guide, Scotland, UK 1996

[XC98] Xputer Lab's XC6k Pages: http://xputers.informatik.uni-kl.de/reconfigurable_computing/XC6k/index_xc6k.html

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

Designing for Xilinx XC6200 FPGAs

Reiner W. Hartenstein, Michael Herz, Frank Gilbert

University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de
www: http://xputers.informatik.uni-kl.de

Abstract. With the XC6200 FPGA Xilinx introduced the first commercially available FPGA designed for reconfigurable computing. It has a completely new internal architecture, so new design algorithms and software is needed. Due to the fact that most applications are in the research area, the number of sold units seems to be small. Because of this progress of design tools for this architecture is rather low. This paper discusses the problems, which appear during designing for the XC6200 FPGAs. A dedicated design flow is presented and demonstrated on an example application.

1 Introduction

The XC6200 is an FPGA that has been designed to be used in two broad classes of applications. The first class is the conventional role of a general-purpose ASIC device for logic integration. The other role is that of an intelligent peripheral that can operate as a memory-mapped coprocessor in microprocessor systems. This is largely due to the advanced FPGA to CPU interface. The design philosophy appears to have been driven by the desire to produce a FPGA optimized for reconfigurable computing.
Designing for the XC6200 is similar to other FPGA families, but some limitations apply. The design software is still in a beta state and progress is slow, as the commercial use is low. Not all features are implemented yet and some are still buggy. Further routing of irregular structures is very difficult because of few routing resources of the target architecture. A lot of problems during application implementation have to be solved manually. Therefore most design flows directly start at gate level where directly the logic blocks of the FPGA are programmed (e.g. Lola [Lo98]). This is like a step back into the stone age of hardware design. To alleviate this drawback in this paper a dedicated design flow for the XC6200 FPGA family is proposed. A few steps appear similar to other FPGA technologies. But due to XC6200 specific problems each single step shows differences to other technologies. Designing consists of five steps: Design Creation, Logic Synthesis, Netlist Generation, Place and Route and Simulate Design with Timing Information. As stated before this looks familiar, although Netlist Generation is mostly a part of the Logic Synthesis step.
The paper is structured as follows. First in the next section the structure of the hardware is briefly introduced. After sketching the limitations of available design software the dedicated design flow is presented, which is mainly based on predefined macro cells. Its benefits are demonstrated with an generic 3x3 linear filter with configurable weights for image processing. This application benefits from the processor interface, which allows to change filter coefficients without reconfiguration of the complete device. To simplify the design process the control part of the design is synthesized. Performance results will justify the introduced method.

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

2 Overview on the Xilinx XC6k and the VCC HOT Works Board

Architectural Overview on the XC6200 Series [GL97]

The XC6200 is based on a fine-grained, sea-of-gates architecture. The XC6216 consists of an array of 64 x 64 core cells surrounded by 256 input-output ports. Every logic cell can implement any combinatorial logic function of two inputs. Each cell can also implement a D-type flip-flop which can be used to register the cell's combinatorial function.
Cells have nearest neighbor connections to their North, South, East and West neighbors. The device has a hierarchical busing scheme. Cells are organized into blocks of 4 x 4, 16 x 16, and so on, increasing by a factor of four each time. A set of fast buses is associated with each size of block. Access to these fast buses is via routing switches that are adjacent to the cells on periphery of the respective blocks.
The processor interface is a 32-bit wide data bus that may also be configured for 16 or 8 bit operation. The XC6200 has been designed to appear in system as random access memory. All data registers on the array are accessible, making it possible to interface with user logic via the processor interface alone. Registers are addressed in columns via a map register. Up to 32 of the 64 registers in column may be read from or written to by nominating their appropriate row position in the map register. This can be extended so that all of the 64 registers in a column may be written by a single 8-bit operation. This is particularly useful in reconfigurable computing applications.
For further information, please refer to the Xilinx XC6200 datasheet at [Xi98a] and some application notes at [Xi98b].

The HOT Works PCI-XC6200 Development System

The proposed design-flow and the implemented examples are tested on the HOT Works PCI Board by Virtual Computer Corporation [Vc98]. The board architecture allows the XC6200 to be accessed by a host CPU via the PCI-bus. The board consists of:

- a XC6216 for the user-designs
- a XC4013 implementing the PCI bus interface
- up to 2 MBytes of fast SRAM for user-data
- a programmable clock-generator.

The XC6216, the SRAM and a set of configuration registers are mapped to the memory space of the host CPU. This allows the host CPU to read or write the SRAM memory of the board and configure or read or write the user FPGA and the user design. For an introduction to the XC6200 Development System refer to [NG97], further information can be found at [Xi97a].

3 Design-Flow for the Xilinx XC6200

This chapter will describe a VHDL-based design flow for the XC6200. First, all parts of the development process (design libraries and software) are described. Then the general design flow, the use of the software, problems and limitations concerning the single design steps are discussed.
The design environment consists of five parts (figure 1). First, a VHDL-simulator is needed to validate the VHDL design description. Then behavioral VHDL must be synthesized in primitive gates of the target technology, therefore a synthesis tool with a corresponding technology library is required. The output format of the synthesis tool will be VHDL. To transform the VHDL netlist in the EDIF-input-format of the place-and-route tool a small program will be used. The place-and-route software is the last tool to mention in the design environment.

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

Technology Library

Basic part of the development process is a technology library. The primitives of this library are any two-input gate functions, any 2:1 multiplexer, constant 0 or 1, buffer, inverter and D-type register. Technology libraries for the XC6200 family exist for the Viewlogic schematic entry tool and the Synopsys Design Compiler. In our design environment Synopsys is utilized.

Using a textural description of the target technology, the Synopsys Library Compiler generates, in addition to a primitive library for the Synopsys Design Compiler, also a VITAL [Vi95] compliant technology library for VHDL-simulation and back-annotation. The VITAL library provides behavioral models of all primitive gates with default timing. The default timing can be overridden by exact timing values calculated by the Place-and-Route-software using SDF data (Standard Delay Format). The Synopsys library is included in the SunOS version of XACT Step 6000 only.

Simulating the design

As the synthesis primitive library can be used only with the Synopsys Design Compiler, the VITAL library is vendor independent. The VITAL library can be simulated with any VHDL simulator such as Synopsys' VSS, Model Technology's V-System or Mentor Graphics' QuickHDL. V-System and QuickHDL have two advantages compared to VSS. First both provide a VHDL foreign language interface to their simulators. This feature can be used for co-simulation of hardware and software [Xi97a]. Then, both support VHDL'93 standard, which is necessary for the coding technique shown in the following. Because of this QuickHDL it is chosen for simulation. Setting up the VITAL library for QuickHDL simulation is similar to VSS, described at [Xi97a]. For information about QuickHDL design libraries refer to [Me97].

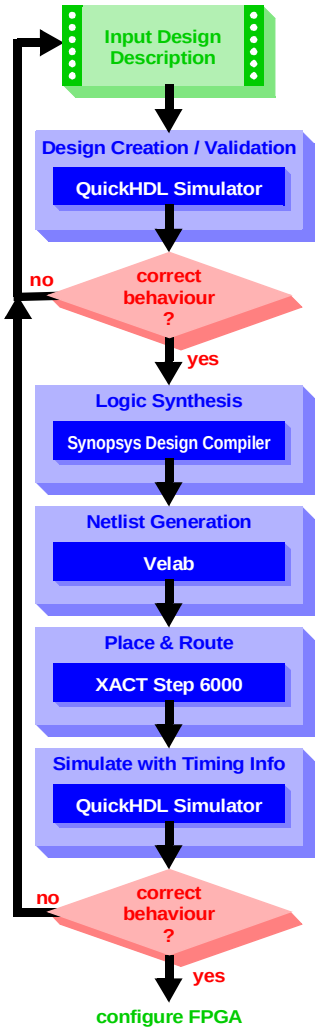

Fig. 1. XC6200 Design Flow

Logic Synthesis

As a technology library of the XC6200 family is provided for Synopsys Design Compiler, logic synthesis of behavioral VHDL can be done with some limitations. First the Synopsys Design Compiler can not synthesize pad-cells, which are needed for user IOBs (Synopsys creates only input/output-buffers). Therefore a top-level structural description of the I/Os has to be provided manually by the designer. Second, some design information such as the pinout or placement can only be attached by user defined attributes, which are not supported by Synopsys Design Compiler.

Another big disadvantage of logic synthesis is a problem of the Xilinx Place-and-Route software XACT Step 6000. Placement and routing of larger non-hierarchical designs without placement information results in bad designs. Hierarchical designs with preplaced macro-cells for adders, subtractors and other regular structures are necessary to allow manual floorplanning. As a consequence, only pure state-machines without

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

datapath (<100 primitive gates) should be synthesized with Synopsys. Synthesis of RTL-VHDL of larger designs (>1000 primitive gates) as for other FPGA technologies is not practicable at the moment.

Netlist Generation

For the datapath a well structured hierarchical design is necessary. Cells of regular structure, such as adders and multipliers, should be preplaced using user-defined attributes and instantiated in larger cells such as pipelines. The result of this design style is hierarchical VHDL-netlist of instantiated components and technology primitives. The input netlist format of XACTStep 6000 is EDIF. To transform VHDL-netlists to EDIF a small tool called VELAB is used [Xi97b]. VELAB is a free VHDL analyser and EDIF netlist generator for the XC6200 family provided by Xilinx. One of its major features is the ability to generate parametrized attributes.

Placement and Routing

XACTStep Series 6000 [Xi96] is a graphical tool for XC6200 family designs. This system is a back-end tool with EDIF as its primary input. The XACTStep Series 6000 editor preserves the hierarchy of the input design. This hierarchy information is used to support both top down design through floorplanning and bottom up design through either manual or automatic techniques. In addition, fully automatic place-and-route is supported. The graphical editor gives full access to all resources of the XC6200 family architecture [NG97].

In practice, the automatic techniques need a lot of manual assistance. Even when using preplaced structures floorplanning and manual placement is unavoidable. The automatic router often fails in routing the top-level design. Therefore placement and routing is not an automated design step as for most commercial FPGA technologies. A lot of manual work and design experience is essential for acceptable results. Timing constraints can not be set, timing analysis of the post-layouted design has to be done interactively by choosing source and destination cells in the graphical editor.

Design Step	Design Tool	Version
Design Validation (1), Simulation (5)	**Mentor Graphics QuickHDL**	**v8.5_4.6c**
Logic Synthesis (2)	**Synopsys Design Compiler**	**1997.08**
Netlist Generation (3)	**Xilinx Velab (freeware [Xi97b])**	**0.52**
Place & Route (4)	**Xilinx XACT Step 6000**	**1.1 beta build 4**

Table 1. Summary of the used design software

3.1 The XC6200 Specific Design Flow

As mentioned in the introduction the design flow for the XC6200 family consists of five steps (figure 1): Design Creation, Logic Synthesis, Netlist Generation, Place and Route and Simulate Design with Timing Information. In the following the architecture specific characteristics of this design flow will be explained by treating each design step in detail.

Design Creation / Validation

First, the design is described and a testbench is written in VHDL. Then the design is simulated and its specification is validated until the behavior of the design is satisfactory. The design has to be specified in a synthesizable subset of VHDL (RTL). The testbench may also include VHDL-statements which are not synthesizable.

Unfortunately the Xilinx place and route software XACT Step 6000 is unable to place and route designs of logic-cell usage larger than 15% to 25%. To handle larger designs,

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

floorplanning and manual placement must be done. To support floorplanning regularity and hierarchical information is necessary. In addition, very regular substructures such as adders, multipliers and other operators should be preplaced. In the applied design methodology for the XC6200 family, a design is partitioned in three entities: a top-level entity, a control unit (finite state machine) and a datapath unit. The datapath is restricted to primitive gates and macro-cells of primitive gates. The macro-cells [XC98] used in the datapath unit are included in a predefined design library. This library may be extended by the designer if a specific macro is not available.

Figure 2 illustrates this design step. Design Creation is an iterative process of writing/modifying VHDL code and simulation. All parts of the design apart from the technology library may be edited. Even the macro-library may be extended, if new regular substructures are needed. Writing a testbench forcing all input ports and verifying automatically the results simplifies this iterative process. Validating the design needs no interaction if no errors occur. Unfortunately two versions of the top-level description are necessary. The reason is that it is not possible to simulate the behavior of a special type of register (RPFDs) and there is no functional equivalent for pad-cells. RPFDs (Register Protected D type Flip-Flop) can only change value by reconfiguration. To simulate the behaviour of a RPFD it must be replaced by a UP_RPFD, which is a RPFD with a simplified processor interface. In future releases of XACT Step 6000 the mapping software will automatically replace a UP_RPFD by an ordinary RPFD. All these cells are included in the VITAL simulation library delivered with the Xilinx software.

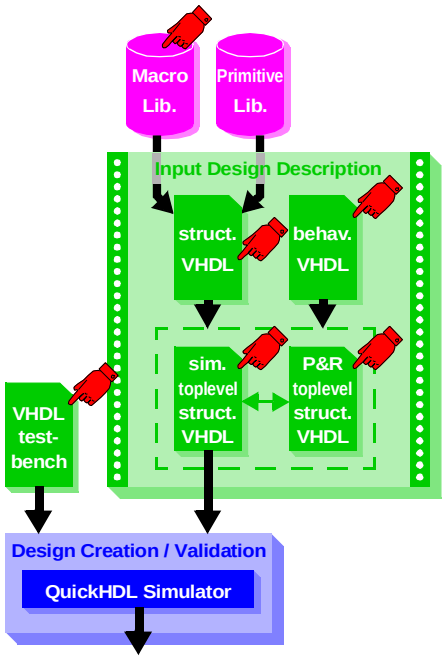

Fig. 2. Step1, Design Creation / Validation

If simulation results meet the design requirements, this step is finished. Further iterations may be necessary if the following design steps demand a redesign, e.g. timing requirements are not met or placement information has to be added for future enhancements.

Logic Synthesis of behavioral VHDL

This design step transforms the part of the design given in behavioral VHDL (e.g. finite state machines) into a structural description of technology gates. Using the Synopsys Design Compiler for the synthesis of XC6200 family technology gates is described in [Xi97c]. There a detailed introduction in setting up the Synopsys environment and a step by step explanation of the synthesis is given. However minor differences appear to our design flow:

1. First, no pad cells are needed, as they are instantiated in the top-level description.
2. The output format should be VHDL as the complete EDIF-netlist will be generated in the next design step with VELAB.

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

As indicated in figure 3 some modifications must be done. The netlist generated by the Synopsys Design Compiler will report errors when processed directly by VELAB. Synopsys writes a package declaration in the beginning of the netlist, which is not necessary and must be deleted, because it is not allowed in VELAB. Further, if ports of type std_logic_vector are used, Synopsys writes signal-assignments to a vector, which are also not accepted by VELAB. They need to be converted into bitwise assignments. To automate this modifications an awk-script [XC98] has been written. After this design step all design information consists only of structural VHDL description files.

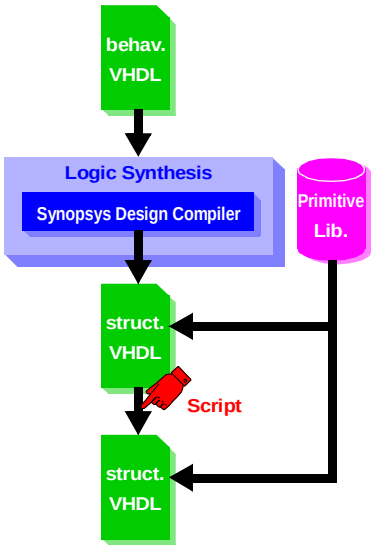

Fig. 3. Step 2, Logic Synthesis of behavioral VHDL

Netlist Generation with VELAB

VELAB is a free VHDL analyser and EDIF netlist generator for the XC6200 family. It can be downloaded from the Xilinx homepage at [Xi97b]. In this design flow, VELAB is used to generate a EDIF netlist for placement and routing and a second VHDL netlist for simulation (see figure 4).

The reason not to use Synopsys Design Compiler for netlist generation is, that it ignores all user defined attributes. But this is the only way to add placement information to the design. Macro library elements (e.g. adders, multipliers) need to be preplaced, because of the bad placement and routing results of the Xilinx XACT Step 6000. Preplacing is further useful if datapath registers have to be accessed via the processor interface. Adding placement information to user attributes is only supported by VELAB. VELAB accepts only a subset of VHDL, especially all structural parts. For a detailed description refer to [Xi97b]. One of the most useful features for preplacement of regular devices is its ability to handle parametrized attributes. This feature allows preplacement of devices with generic parameters such as size or layout. The macro-library used in the proposed design flow is based on this feature.

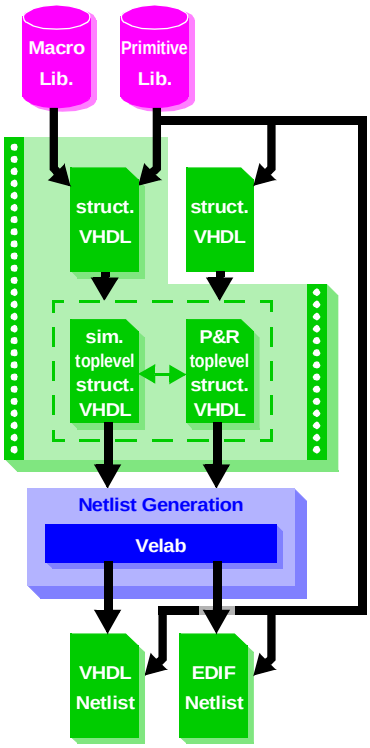

Fig. 4. Step 3, Netlist generation

As illustrated in figure 4 VELAB is used to generate two separate files. Therefore the top-level structural description for place-and-route (and of course all other design-files) are fed into VELAB to generate the EDIF netlist as an input to XACT-Step 6000. The top-level description for simulation is processed to a VHDL-netlist. This netlist is used for simulation of the placed and routed design with the exact timing parameters calculated by XACT Step 6000.

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

Place-and-Route

Placement and routing is an important step in the design flow. In contrast to other FGPA-technologies where floorplanning is optional, it is an essential part for most designs. XACT Step 6000 is a graphical tool allowing both manual floorplanning and automatic place and route. Unfortunately fully automatic place and route mostly fails or leads to bad results.

Figure 5 gives an overview to the place and route design step. The EDIF-netlist generated by VELAB is placed on the FPGA and all nets are routed. The configuration information of the FPGA is written to a CAL-file. A complete list of all nets and their corresponding delays are calculated for timing analysis. Setting up timing constraints for placement and routing and an automatic timing report is not supported. For a detailed analysis of critical paths source and destination nets are chosen from a list of all nets. The analysis information may be exported to a text file or used as an input to a standard spreadsheet. For timing simulation of the layouted design two other forms of timing extraction are supported, a delay table for the Viewlogic Simulator, and SDF (Standard Delay Format), which is used in most VHDL simulators. In the proposed design flow the SDF output is used as an input to the QuickHDL simulator. For the reason of bugs in both the VITAL technology library of the XC6200 family and the SDF-writer of XACT Step 6000 some modifications of the SDF-file are necessary. These bugs may be fixed in a future release of XACT Step 6000, meanwhile an awk-script [XC98] does the modifications.

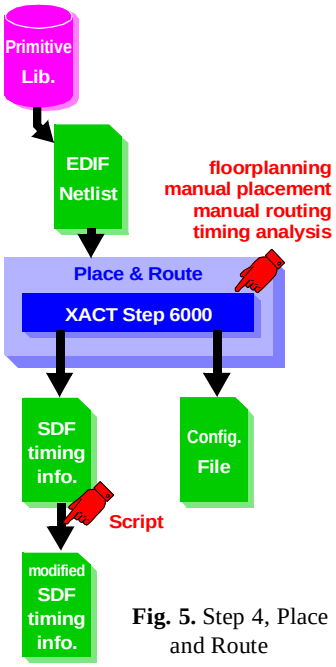

Fig. 5. Step 4, Place and Route

XACT Step 6000 is based on a bottom-up strategy. That means that one level of hierarchy is first placed and then routed, when all units of the lower design level are already placed and routed. In most cases manual floorplanning is necessary to achieve adequate results. The basis of the proposed method is the library of preplaced macro-cells for design units of high regularity such as adders or multipliers. Using this library elements avoids to floorplan at gate level. Because unstructured elements such as the synthesized FSM are more difficult to place, it is recommended to simplify the control logic as much as possible. Straight and short connections between the design units is the major goal of manual floorplanning.

Straight forward bottom-up placement is not recommended because of a placement must be found which simplifies not only the connections of the actual design level but also of the higher design hierarchy levels. That may be one of the major problems of a automatic place-and-route procedure. If the nearest-neighbor connection is not sufficient, higher routing levels such as length-4 or length-16 must be used restricting the possible placement on the next design hierarchy level. Therefore it is recommended to route the design on the top hierarchy level (except for the preplaced macro-cells).

Manual routing of the design is not well supported by Xilinx software. To completely route a design the sequence of the routed nets is an important issue. The order chosen by the software often leads to bad results. A good order would be to start with the obvious connections such as the neighbor-connections. Next, all critical nets should be routed. Critical nets include the timing-critical nets and the nets which are difficult to

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

route or even failed during automatic routing. Then all nets connecting I/O-cells should be routed. Last, all other nets are routed.
Note that the placement & routing guidelines reflect our experience with XACT Step 6000. They may not be best for all designs. As a consequence of using the above method only a limited design utilization can be achieved because the elements of the implemented macro-library are optimized for timing rather than high device utilization.

Simulating Design with Timing Information

After placement and routing of a design the exact timing-parameters are known. The VHDL-netlist generated by VELAB and the SDF-timing-information calculated by XACT Step 6000 can be simulated again using the QuickHDL-simulator (see figure 6). If a testbench has been created in step 1 it may be used here again simplifying this design step. If the testbench validates all design results and the simulator does not report any timing conflicts the design is correct under the tested conditions and the FPGA may be configured. Timing violations may be corrected by a different placement and routing leading back to step 4, but if the design does not show the correct behavior the whole design process starting with step 1 must be iterated.

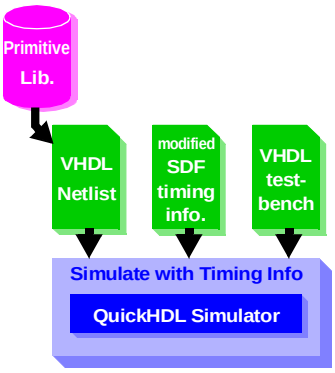

Fig. 6. Step 5, Simulate Design with Timing Information

4 Generic 3x3 Linear Filter for Image-Processing Example

The generic 3x3 linear filter processes an image by moving a 3x3 window over it (figure 7) and applying the following formula:

$$p_0^{new} = \frac{1}{j} \cdot \sum_{i=0}^{8} p_i \cdot k_i$$

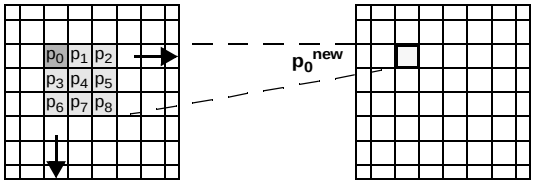

Fig. 7. Operation of a 3x3 Image-Transformation

This operation on an m*n-pixel image results in an image of size (m-2)*(n-2). All pixel-values are assumed to be an 8-bit grayscale value (0 to 255). Fig. 8 shows some filter coefficient examples.

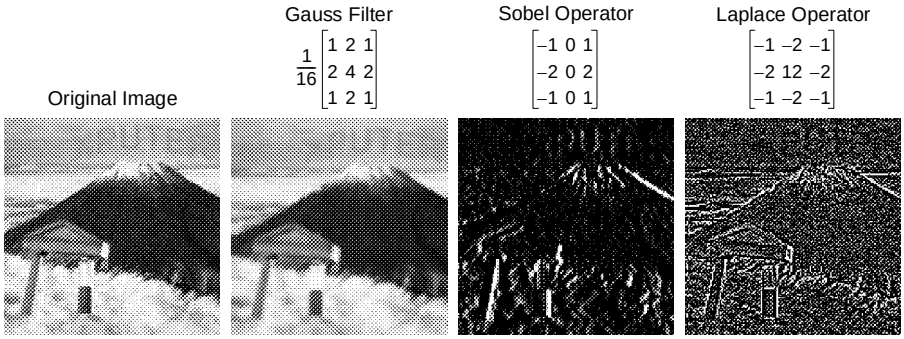

Fig. 8. Example Effects of Different 3x3 Linear Filter Operators

Reiner W. Hartenstein, Michael Herz, Frank Gilbert: Designing for Xilinx XC6200 FPGAs; Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998

In this implementation of the filter, the coefficients k_i are signed integer numbers in the range from -16 to 15 (5 bit) and $j=2^n$ with n element of [0:8].
This leads to the dataflow-graph shown in figure 9. For multiplication of the coefficients k_i constant coefficient multipliers are used, which are included into the macro library. For macro implementation the description in [Xi97d] is modified for negative constants and higher device-utilization.

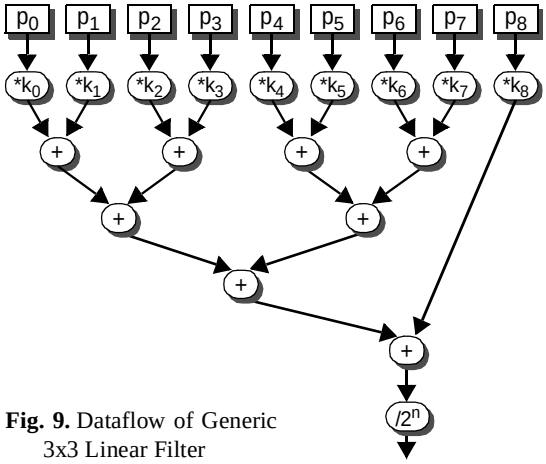

Fig. 9. Dataflow of Generic 3x3 Linear Filter

Fig. 10 shows the layout of the placed and routed generic 3x3 linear filter design. Because of the used preplaced macro cell elements, the constant-multipliers are always placed at the same location. Therefore the weights can be exchanged by only reconfiguring the related cells.
The design is working at a clock frequency of 25 MHz, where the computational pipeline is processing one pixel in two clock cycles. The design utilizes 2373 logic cells of the XC6216 (58%). Implementation details and more application examples can be found in [Gi98].

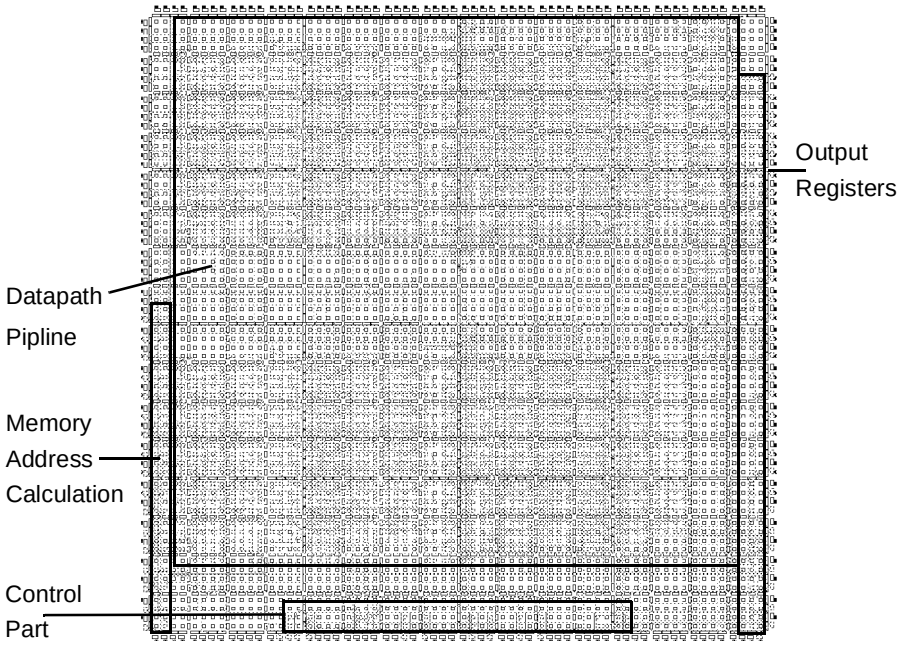

Fig. 10. Layout of the Generic 3x3 Linear Filter

5 Conclusions

A dedicated design flow for the Xilinx XC6200 FPGA series has been introduced to overcome problems of the available design software. The proposed method recommends partitioning of designs in a control part and a datapath. All regular structures of the datapath should be implemented with a macro library. This library has been implemented and is extendable for future designs. Its use has been proven on several examples [XC98], whereas an image processing example was presented.
Future work will be software support for dynamic reconfiguration and data access via the fast map interface.

6 References

[Gi98] Frank Gilbert: Development of a Design Flow and Implementation of Example Designs for the Xilinx XC6200 FPGA Series; Diploma Thesis, University of Kaiserslautern, Kaiserslautern, May 29, 1998. Download from [XC98].

[GL97] Gordon McGregor, Patrick Lysat: Extending Dynamic Circuit Switching to Meet the Challenges of New FPGA Architectures; Proceedings of 7th International Workshop, FPL'97 London, UK, Sept. 1997. LNCS 1304. Springer 1997

[Lo98] http://www.lola.ethz.ch/lola/

[Me97] Mentor Graphics Inc., QuickHDL User and Reference Manual, 1997

[NG97] Stuart Nisbet; Steven A. Guccione: The XC6200DS Development System; Proceedings of 7th International Workshop, FPL'97 London, UK, Sep. 1997. Lecture Notes in Computer Science 1304. Springer 1997

[Vc98] http://www.vcc.com

[Vi95] VITAL ASIC Modelling Specification; Draft IEEE 1076.4; New York October 1995; http://www.vhdl.org/vital

[Xi98a] http://www.xilinx.com/partinfo/6200.pdf

[Xi98b] http://www.xilinx.com/apps/6200.htm

[Xi97a] Xilinx Inc., Application Note XAPP 087: Co-Simulation of Hardware and Software, http://www.xilinx.com/xapp/xapp087.pdf, San Jose, CA, USA, 1997

[Xi97b] Xilinx Inc., Velab: VHDL Elaborator for XC6200, http://www.xilinx.com/apps/velabrel.htm, San Jose, CA, USA, 1997

[Xi97c] Xilinx Inc., Synthesis and simulation of a circuit in VHDL, using the Synopsys toolset, Synopsys_flow.pdf, XACT 6000 Synopsys Interface Documentation

[Xi97d] Xilinx Inc., Application Note XAPP 082: A Fast Constant Coefficient Multiplier for the XC6200, http://www.xilinx.com/xapp/xapp082.pdf, San Jose, CA, USA, 1997

[Xi96] Xilinx Inc., XACT Step Series 6000 User Guide, Scotland, UK 1996

[XC98] Xputer Lab's XC6k Pages: http://xputers.informatik.uni-kl.de/reconfigurable_computing/XC6k/index_xc6k.html