Development System for FPGA-Based Digital Circuits

V.Sklyarov, J.Fonseca, R.Monteiro, A.Oliveira, A.Melo, N.Lau, I.Skliarova, P.Neves, A.Ferrari

Department of Electronics and Telecommunications – Aveiro University

DESIGN PROBLEMS

The paper discusses some new hardware and software tools that can be used for the design of virtual circuits based on dynamically reconfigurable FPGAs. With the aid of these tools we can implement a system that requires some hardware resources R_c, on available hardware that has resources R_h, where $R_c > R_h$. The main idea of the approach supported by these tools is the rational combination of FPGA capabilities with some proposed methods for producing a modifiable specification, together with a novel technique for architectural and logic synthesis, which has been incorporated into the new design environment.

The considered digital circuit is composed of two traditional components, a control unit modeled as a FSM, and a datapath containing a set of functional blocks that provide the required processing of the data. Note that for virtual circuits, just a part of operations will be physically implemented in reconfigurable hardware. This part includes a subset $\pi^i \subset \pi$ of the control algorithms and an associated subset of datapath $D^i \subset D$, which will be required in order to perform the operations described by π^i. In general, when we modify the control algorithms π^i we have implemented, we also have to modify the corresponding datapath D^i. In order to change the sets π^i and D^i we can load new configuration data into the predefined areas of the FPGA SRAM from external memory, keeping all the required configurations.

Thus the purpose of our design is to build the circuit with the properties of modifiability, extensibility and, finally, virtualability. We want also to solve some supplementary problems, such as incorporate incomplete design for purposes of preliminary tests, etc.

Note that field programmable technology only provides support for some of the properties we have considered above. The remainder have to be implemented through new architectures for digital circuits that is suited to meet the target requirements. There are some commercial boards that are used mainly as coprocessors for PCs, such as the FireFly™ (Annapolis), the Riley-2 (Imperial College), etc. These allow the critical parts of software (usually time-consuming) to be implemented in hardware, and consequently improve many aspects of the related programs, such as performance for instance. However, it is difficult, and in many cases unrealistic, to use such boards to construct digital systems that are not dependent on a host computer (such as embedded systems), and for which the properties considered above are satisfied. For this reason, we have designed a stand-alone board that can be programmed from either the parallel or serial ports of a host computer. The board is based on the dynamically reconfigurable XC6200 FPGA.

The next stage is the architectural, logic and topological design of digital circuits based on either a stand-alone or a built in PC board. Our approach differs from known methods and tools in two ways. It presents a set of formal methods that on the one hand allows modifiable circuits to be synthesized for both the control unit and the datapath, and on the other hand provides systematic support for dynamic reconfiguration, starting from the behavioral level. The approach is based on a proposed decomposition of the resulting scheme into permanent, parameterized and modifiable parts. The permanent and parameterized parts can be implemented within an FPGA based on pre-designed templates. The modifiable part can be designed using the software tools that have been developed for this purpose. Finally, the proposed technique allows the complete digital circuit to be designed, simulated, implemented in hardware and debugged. Note that there are some CAD systems such as ViewLogic, as well as other software packages that can be used to construct circuits based on dynamically reconfigurable hardware. However, they do not allow dynamic reconfiguration of routing resources, i.e. they can only be used to make run-time modifications to functions of individual cells. That is why we have developed new software that provides an interface with commercially available packages, such as XACT6000 and Velab, and allows us to perform design steps that are common to a number of digital circuits (to embedded applications in particular). The new software also enables us to construct circuits that satisfy all the requirements that we considered above. Finally, we can combine existing software with our newly developed modules to make use of both stand-alone and built-in boards based on dynamically reconfigurable FPGAs.

DESIGN TOOLS

The developed design tools provide facilities for synthesis, simulating, testing, and debugging of digital circuits based on dynamically reconfigurable hardware, such as XC6200 family FPGAs. These tools have been integrated into the design environment for logic synthesis (IDELS)[1]. The software has been developed using Visual C++ and allows access to both stand-alone and built-in PC boards with the aid of the RALLib library (some components of this library have been changed, and all components have been included in the DLL). Fig. 1 shows the primary blocks of IDELS, and illustrates a feasible design flow with the aid of IDELS (see the rectangles with shadow) and commercially available CAD tools such as ViewLogic, Velab and XACT6000.

The digital system is considered as a composite of a control unit and a datapath. To describe the behavior of the system we have used the algorithmic state machine notation (that is a kind of flow-chart), or graph-schemes (GS) and their varieties, such as hierarchical GS (HGS) and parallel HGS (PHGS). Any GS Γ_k describes the corresponding sub-algorithm π_k from the set π, i.e. we have to map the set $\pi=\{\pi_1,...,\pi_K\}$ onto the set $\Gamma=\{\Gamma_1,...,\Gamma_K\}$: $\pi \Rightarrow \Gamma$. Each sub-algorithm π_k from the set π describes a desired sequence of operations O_k from the set $O=\{O_1,...,O_K\}$. Since we want to provide extensibility and modifiability of the digital system, we have to be able to alter the sequence in which algorithms from the set π are to be executed in hardware, and alter each separate algorithm $\pi_k \in \pi$ itself. For many practical applications it is important to provide for the reuse of algorithms from the set π in future products.

Let us consider π as a set of relatively independent modules, and assume that some of them might be reused. To provide for dynamic replacement of various modules, we can link them flexibly, in such a way that the required binding between modules will be established during run time. This will enable us to realize just a part of π in hardware, and swap it with another part when

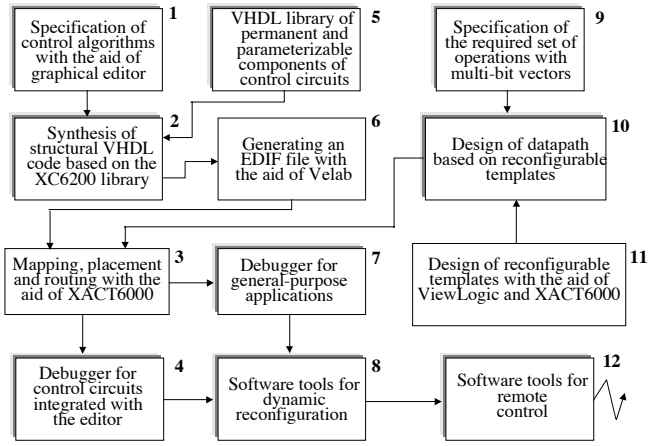

Figure 1 - Design flow for embedded system with the aid of IDELS and commercially available CAD tools.

required. As a result, we will construct a virtual digital system, i.e. a system for which $R_c > R_h$.

For many practical applications we want to change not only the sequence of modules, but also the modules themselves. This facility has been implemented with the aid of the following technique:

- for each component $\pi_k \in \pi$ we have introduced some constraints, such as the maximum number of inputs/outputs, etc. As a result we can implement any algorithm $\pi_k \in \pi$ in a pre-allocated area of an FPGA;
- modification of the required algorithm has been achieved with the aid of field programmable technology, i.e. we assume a partial reconfiguration of the FPGA within the area where the algorithm $\pi_k \in \pi$ has to be implemented.

The request for dynamic reconfiguration is generated either in accordance with some predefined sequence of changes, or unpredictably when an FPGA generates a hardware interrupt that forces the execution of the swapping procedure. The latter has been used in particular for virtual embedded systems. Thus we have combined architectural models directly supporting dynamic modifications, and the topological capabilities of field programmable devices, such as rerouting and the potential for altering the functions for primary logic elements, such as FPGA cells.

Note that, in the general case, when we change any algorithm $\pi_k \in \pi$ we have to change the set of operations $O_k \in O$ associated with this algorithm. This has been achieved using a reconfigurable ALU and stack-based sequence of operations presented in the form of reverse Polish notation.

Let us consider the primary blocks depicted in fig. 1 in more detail. The desired behavioral specification can be prepared and entered with the aid of a graphical editor (block 1), which can also accept input in a textual format.

After a GS (a HGS) has been prepared, it can be tested for correctness by examining some formal rules. Then we can invoke the synthesizer (block 2), i.e. the program, which will perform all synthesis steps (see the next section for details) for the control unit modeled as a hierarchical FSM (HFSM). The output of the synthesizer is a structural VHDL code for the designed control circuit based on primitives from the PRIMS library. The latter describes possible configurations of cells for FPGA of XC6200 family of Xilinx. This code can be further processed by the Velab elaborator, which creates an EDIF file containing the initial data for XACT6000 (block 3). The latter carries out mapping, placement and routing procedures and builds *.CAL, *.SYM and *.RAL files that will be further utilized for initial configuration (*.CAL), keeping information about the accommodation of various elements of the circuit in FPGA (*.SYM), and for dynamic reconfiguration (*.RAL).

The debugger for control circuits (block 4) has been integrated with the editor (block 1) and enables the following operations to be carried out:

- configuring an FPGA in accordance with the files considered above. For preliminary test purposes we have used the Annapolis FireFly™ PC board with FPGAs XC6216/6264;
- examining (during run-time) any individual algorithm from the set π in different modes, such as step by step, continuously, with interrupts at specified break points, etc;
- testing hierarchical calls;

After the control unit has been designed and tested, we can construct the attached datapath. The first step is to specify the desired set of operations $O_k \in O$ with multi-bit vectors (block 9). The number of operations is limited and depends on the capabilities of the reconfigurable ALU. Then we build the basic structure of the datapath (block 10), which is based on pre-designed templates that were created with the aid of ViewLogic. The process of synthesis assumes the generation of a proper modifiable core for the reconfigurable ALU, which is based on components taken from the ViewLogic XC6000 library. Finally we use ViewLogic to create an EDIF file for the complete datapath that is composed of templates (block 11) and the reconfigurable circuits (block 10). Then the EDIF file is processed by XACT6000 and all the required files, such as *.CAL, *.SYM and *.RAL, are generated.

Finally we invoke the debugger for general-purpose applications (block 7), which can be used for any circuit implemented in an XC6200 family FPGA. All debugging facilities have been provided for both built-in and stand-alone boards. We assume that initially the designed circuits will be tested with the aid of a built-in board, such as the FireFly™. Then we can configure a stand-alone board and provide its dynamic reconfiguration when required. In order to do this the following technique has been applied:

Initially all permanent and parameterizable components of the circuit have to be loaded;

Next we load an initial configuration for the modifiable components. All these components are located in preallocated areas of the FPGA, which have been associated with related windows in the configuration SRAM. When any modification is required, just these windows (or parts of these windows) will be updated. Partial dynamic reconfiguration of the circuit is assumed; when a new configuration is required, the board generates a hardware interrupt. The functionality of the FPGA hardware that has to be reconfigured will be suspended, and the other FPGA hardware will operate in normal working mode;

The host PC computer handles this interrupt. Firstly it reads information from the board, which identifies the area of the FPGA that has to be reconfigured (the window in configuration SRAM that has to be swapped). Then the desired configuration will be retrieved from predefined segments stored in the host PC computer and loaded into the selected areas of the FPGA SRAM.

The results of experiments allow to conclude that hardware and software tools we have presented appears to be very fast, even when synthesizing some rather complex designs.

1. V.Sklyarov, R.Monteiro, N.Lau, A.Oliveira, A.Melo, K.Kondratjuk. “Integrated Development Environment for Logic Synthesis Based on Dynamically Reconfigurable FPGAs” Proceeding of FPL'98, Tallinn, 1998.