TOOLS AND TECHNOLOGIES FOR DESIGNING CONTROL SYSTEMS USING PROGRAMMABLE LOGIC DEVICES

Adam MILIK, Mariusz DYKIEREK

Aldec-ADT, Sp. z o.o.,
Widok 23, 40-118 Katowice, Poland
adamm@aldec.com.pl, mariuszd@aldec.com.pl

Abstract: The paper presents modern programmable logic devices and design methodologies with appropriate tools reference. It briefly shows enhancements introduced in the architecture of FPGA and its influence on circuit capabilities. The paper presents also the design process from idea to functional implementation. It discusses algorithmic and functional approaches to design. Abstraction level and transaction mechanisms are shown as a method of hiding details without compromising design functionality. Finally tools for different design approaches and design environment requirements are presented. *Copyright © 2005 IFAC*

Keywords: Programmable Logic Device; PLD; Field Programmable Gate Array; FPGA; Complex Programmable Logic Device; CPLD; VHDL; Verilog®;

1. INTRODUCTION

Today's Programmable Logic Devices are no longer similar to their ancestors from early '80s. Changes in the manufacturing technology and the feature sets offered by the leading CPLD and FPGA vendors have been more revolutionary than evolutionary, especially over the past few years. Deep submicron processes, reaching 90nm (and the cutting-edge 65nm announced), ensure enormous capacity of the high speed logic fabric available to users. Besides the growing logic density, contemporary FPGAs offer specialized Digital Signal Processing units like Multipliers and Multiply-Accumulate circuits (MACs). With the introduction of industry standard RISC processor hard macros like PowerPC™ 405 in Xilinx's Virtex II Pro family and ARM™ 922T in Altera's Excalibur devices, FPGAs enter the world of System-On-Chip (SOC) designs where software co-exists, coordinates and cooperates with the hardware on a single silicon die. Both Altera® and Xilinx® also offer soft-core processors (mapped into programmable resources of FPGAs) that can be custom-tailored to the needs, requirements and capacity limitations. Configurable, high speed, built-in serial transceivers complying with several protocols simplify process of data transmission between devices and subsystems.

The flexibility of programmable devices is yet another advantage that should not be underestimated. The hardware implemented in a PLD can be upgraded while it is already in the end-user product. Static reconfiguration is a great advantage that allows for functionality update and bug removal without physical modification of the circuit structure. Moreover, the process of structure modification can be performed dynamically, while the system is operating. A dynamic reconfiguration allows for a rational hardware utilization for the appropriate task. Only required processing components are used. Dynamic reconfiguration seems to be the most challenging area in hardware-software co-design (Hauck, 1999, Hazel et.al. 2002).

The features mentioned above alongside continuously dropping prices of programmable chips open the doors for FPGAs and CPLDs in applications that were reserved for ASICs in the past.

To empower the designers with the ability to efficiently use the capabilities of modern devices, new techniques and tools for design and verification

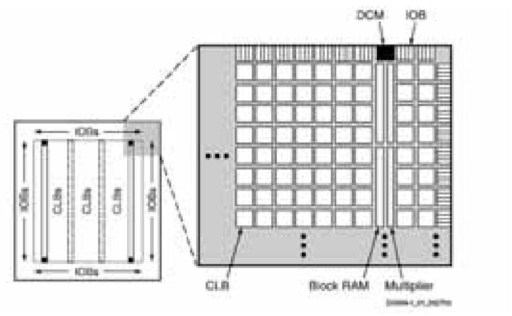

Fig. 1. Modern FPGA Architecture (Spartan 3)

are necessary. Some of these will be presented in the following sections of this paper.

2. FPGA Hardware Platform

FPGAs available commercially differ depending on the manufacturer and the family of devices (Xilinx, 2003, Altera). There are common programmable resources typical for a general architecture of FPGAs like look-up tables, flip-flops and programmable routing resources (Fig. 1). In this section examples of resources that provide advantages to designers are shown. Those additional functional blocks allow for better logic utilization, increased performance and improved board level interconnection system.

2.1 Functional extensions and peripheral blocks

Among typical logic resources appear additional specific blocks required for a typical digital design. The need of implementing such functionality was mainly caused by their inefficient implementation by general purpose logic resources.

Memories

Internal memories offer new approach to designs based on of FPGA devices. Implementing small memories with typical logic resources inside FPGAs was very expensive. On the other hand adding external memory requires additional I/O pins and reduces maximal operating frequency. Maximal capacity of an on-chip memory is 9 Mbits for Altera Startix FPGA family (Altera).
Implemented memory can be arranged in different functional configurations like single port, dual-port and bidirectional dual port. Placing RAM cells in the configuration stream allows to pre-initialize memory contents during the configuration process. In this case memory can be used as ROM or initialized RAM (reduces hardware overhead for memory initialization). All those benefits allow FIFOs, CAMs, addressable register blocks or functional blocks like multipliers to be integrated in FPGA.

Arithmetic extension circuits

Arithmetic operation are common for many designs especially for DSP. The most problematic and resource consuming operation is multiplication. There are several different algorithms of sequential multiplication but the fastest multipliers are combinatorial. Virtex-II family FPGAs offers up to 168 combinatorial multipliers in the largest chip. Multipliers operate on 18 bit words in 2's complement notation (Xilinx, 2003). Much more complex block can be found in the Stratix family from Altera, where typical MAC circuits (register multiplier adder/subtractor) are implemented. Argument length can have up to 36 bit. Several multiplier configurations are possible – from one 36-bit to four 18-bit. The configuration can be changed dynamically. Altera calls those blocks DSP as they are particularly useful for signal processing.

Hardware processor core

Among dedicated logic resources some families of FPGAs incorporate embedded microprocessors on the same silicon die. Such a solution promotes a typical FPGA device to a fully programmable system level solution. Altera's Excalibur is equipped with ARM 32-bit RISC microprocessor from series 9 (Altera). Additional peripherals devices for processor are also available. Power PC is a microprocessor that powers a competitive solution from Xilinx. Up to four Power PC cores are integrated in the Virtex-II Pro family (Xilinx, 2003).

Clock deskew circuitry

Clock distribution in a large digital synchronous circuit encounters the problem of the skew effect. In the delay compensation circuit PLL (Phase Locked Loop) or DLL (Delay Locked Loop) units are used. Those circuits can also be used for frequency synthesis purposes and for clock correction (duty factor and phase correction).

Universal IO Blocks

FPGAs offer a large number of user IO pins that reaches to about 1000. Presented features show that complex systems can be implemented in one or several FPGA chips. There is a need for communication with different peripheral devices and memories. This requirements forces implementing flexible, multi-standard input-output blocks. Those blocks allow for transferring signals in both directions in different logic standards. Several logic standards (LVTTL, PCI, GTL, LVCMOS,…) are supported by the programmable reference voltage. Differential signal transmissions and double data rates (DDR) are also supported through appropriate receivers. For proper signal conditioning digitally configurable impedance (DCI) is used that allows impedance matching of transmission lines and reducing signal reflection. Impedance matching system allows impedance control of both output and input by applying serial or parallel resistance, respectively. IO flexibility reduces or almost removes needs for additional external passive or active signal matching and conditioning components.

Reconfiguration

In general, each FPGA device must be programmed before use. Depending on the configuration storage

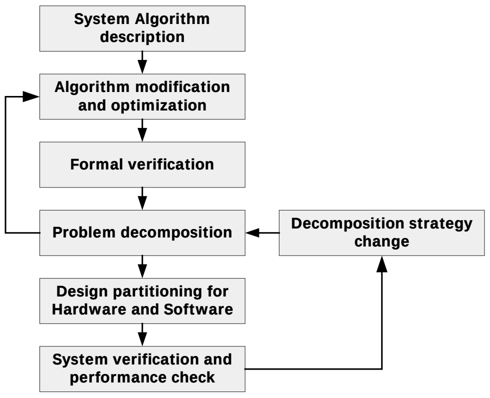
System Algorithm description
Algorithm modification and optimization
Formal verification
Problem decomposition
Decomposition strategy change
Design partitioning for Hardware and Software
System verification and performance check

Fig. 2. System level design flow

this process can be executed once in the life time (OTP) or the device can be reprogrammed. Devices based on SRAM configuration memory must be configured each time after power-on. Commands to reconfigure circuit and load new configuration data can also be issued during operation. The ability to change configuration data has an important implication – it allows to update the functionality of the circuit without changing its physical structure. When the configuration circuitry supports partial modification of configuration memory contents, it is possible to create a run-time re-configurable circuit. Circuits, similarly to computers with exchangeable programs, can be modified according to the current requirements. This allows to reduce the logic capacity of the circuit by providing only a subset of the functionality – such that is required to execute the current task (which can be a small subset of all executed tasks).

3. DESIGN PROCESS

Contemporary designs are based on the system level paradigm (Ferrari A. and A. Sangiovanni-Vincentelli, 1999, Müller W *et.al.* 2003). System level design techniques are based on ready system components with a custom design part. The unmodifiable part of the system usually consists of a microprocessor system with a basic set of peripheral units used to solve a given problem (automotive, industrial, telecommunication, household applications etc.). Specific features of the designed product are located in the custom part of hardware and of course in the dedicated software. The custom part of the hardware can be implemented as a mask programmed part of the chip or as an FPGA configured specifically for design needs. A very important factor in the system level design is performance. The performance problem is to a large extent dependant on placing a boundary between the hardware and the software part of design. This boundary is constrained by several factors like power consumption, FPGA part size, system performance, etc. In order to achieve best possible results, the partitioning process should be performed for different combinations of possible solutions (Fig. 2). The presented system solution requires a new design approach that allows a coherent design and verification of the product during each stage of the development.

In the past concept of the design was divided into several steps that were unlinked together. Refining the design from one abstraction level to another requires an extremely high effort from the design team. Models developed for a given stage of abstraction can hardly be used in other abstraction levels. The design starts from the concept modelled with the use of high level languages (C, Fortran, Pascal,…) or tools (Matlab, Mathcad,…). This level of abstraction allows to obtain algorithmic verification in early stages of the design. Verification in the domain of algorithm only assures the designer that a proper algorithm was chosen. In fact at this stage it is difficult to estimate the performance of the system while the performance can be impacted by several factors that are design dependant. An extremely important step after algorithmic verification is the design partitioning. An improper design partitioning can lead to inefficient implementation that hardly meets design constraints. Usually, during partitioning there are several factors that are taken into account. In general algorithms implemented in hardware are better than software competitors. Software implementation and debug process is much easier and less time consuming than for hardware. A properly partitioned design should have flexible hardware structures that allow to assure required performance with the help of the software platform. When this is combined with a system level approach, it is obvious that a specific approach is required for design development.

As already mentioned, high level programming languages are used for algorithmic verification purposes. C language is extremely popular in the world of programmers, embedded system designers and hardware engineers. C language can be treated as a universal platform for program implementation and design verification. Universal also means unconstrained. In general, C language has not implemented any mechanisms to model system behaviour contrary to HDL languages (concurrency, signal resolution, events). On the other hand, a lot of high level programming language structures are inherited by HDL languages. The gap between general programming languages and system design tool requirements can bridged by specific libraries that introduce system and hardware behaviour implemented directly in C++ with specific classes

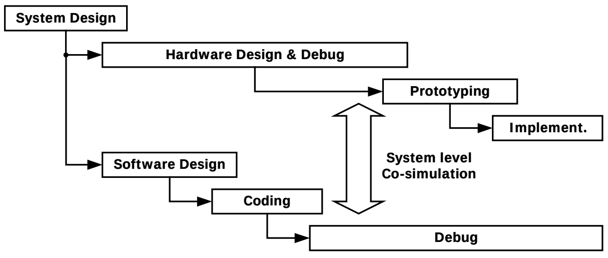

Fig. 3. Integrated design flow

and templates (Gajski 2000). System design flow from idea to implementation with hardware and software paths is shown in (Fig. 3).

4. Modern design methodologies

As it was already mentioned, design starts from an algorithmic description or general requirements provided as spoken rules. Before an algorithm and a set of requirements can be transformed into a fully functional device, the designing process must be completed. This assumes a hierarchical approach to problem solving. There are two basic approaches to hierarchical design: top-down and bottom-up. In real the design process is usually started from the top-down methodology. When the design description is detailed enough, the methodology is reversed and goes from the bottom up by assembling primitive components into functionally complete unit.
At the early beginning designers focus on general requirements imposed on the product. Each feature or constraint is expanded and reflects part or the whole design. The number of details on each level of abstraction must be reduced to the perception level of a human being. Reducing the number of details allows for a better design development management and maintenance. The approach to the design can be either algorithmic or functional.

4.1 Algorithmic high level approach

In the area of automatic control and DSP a typical design approach is based on mathematical algorithms. In the domain of mathematical algorithms Matlab and Simulink environments become very popular. This mathematical environment is suitable to solve different problems from different technical branches from signal processing to process control (Sigmon K and T.A.Davis 2001). This tool allows to solve algorithmic problems in discrete and continuous time domain. It is also equipped with a comprehensive analysis tool set. Simulink block diagram allows to construct hierarchical models of the explored problem. Nowadays it is not enough to have an algorithm. The complexity of processing is so high that some support during algorithm implementation and final verification is also required.

4.2 Functional description - Transaction level modelling

The traditional modelling method, based on RTL, is inadequate to designs with the complexity of system level (Ferrari A. and A. Sangiovanni-Vincentelli, 1999). The ideal method must be very general and suitable for modelling whole systems with hardware and software. At an early stage of the design, the method should enable general algorithmic verification with the possibility of step-by-step design refinement. This method can easily be derived from the RTL technique (De Micheli 1994 Devadas *et.al.* 1994). In order to reduce the number of details in the design, it is necessary to introduce abstract objects called transactions. Reducing and encapsulating details of design into transactions allows the designer to get a quick perception of the whole design in terms of its functionality. Transaction-level modelling is a general approach to a hierarchical design methodology. This method has an unconstrained transaction form that can describe general device behaviour, software activity and hardware behaviour. Simplicity of the transaction also reduces computation power required for system verification.
Transaction, in the world of system level design, can be defined as interaction of two components. A transaction acts as a general container used for passing or exchanging information or data sets. It is independent from data protocols, bus size or exchange protocols, which allows the designer to reduce unneeded details by hiding them behind a transaction (Fig. 4). This approach allows engineers to concentrate on real design problems for a given abstraction level while other problems are hidden out of sight. It also allows the reuse of test patterns prepared during the early stages of the design to verify system functionality.
Transaction-level modelling starts with describing general system requirements with high-level transactions. Functional models are evaluated in timeless space or time can be represented by transaction passing. A network interface that processes data can be considered as an example. At an early stage of the design process, a set of commands and appropriate activities are described. At this level the only important thing to implement is command execution. Command passing is generally represented by transactions. The internal structure of a command, data frame or transmission protocol, are beyond the scope of this level. When a satisfactory level of behaviour is obtained, refinement can be started. At this stage, a greater number of details can appear. Transactions that represent whole data containers can be represented by appropriate data fields, or data frames, that allow for design refinement by splitting functionality of top-level modules into data processing units and execution units. During design refinement, the functional part that is responsible for delivering external transactions

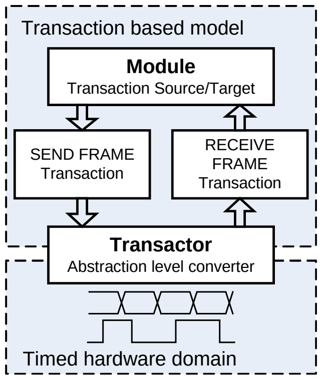

Fig. 4. Transaction level concept

operates as a testbench. This testbench can be used during the entire lifetime of the design to verify different abstraction levels with the same transaction.

5. Tools for modern designs

Presented design methodologies require appropriate tools integration. Design process requires a skilled design team and a good toolset for design development. Design process operates on different levels of abstraction for completing design tasks. When the system level approach is used, different components described with all possible methods can be encountered.

5.1 Algorithmic approach tools

In the area of algorithms Matlab and Simulink tools are extremely popular. This package is an open environment that allows different developers to deliver their own solutions. The Matlab is typically used as a standardized mathematical and algorithmic verification environment (Fig. 5). Simulink model or Matlab equation or algorithm is only an intermediate state of design. This stage assures only about correctness of the selected algorithm or implemented data processing. From the implementation point of view additional tools are required for automatic code or hardware generation. This often requires transformation of floating point to fixed point arithmetic and generating the appropriate data flow controller. AccelChip delivers a tool that is able to convert mathematical description into functional hardware. This tool allows for calculating the number length for proper calculation resolution and converts mathematic formulas into a hardware structure. Xilinx System generator is another solution embedded into the Matlab environment that allows designing signal processing systems directly in Simulink with the use of delivered blocks. Those blocks are finally synthesized into a circuit description. Often there is a need for verifying the existing solution in a complex mathematic environment. Co-simulation with mathematical environment is possible in Aldec simulators Active-HDL and Riviera.

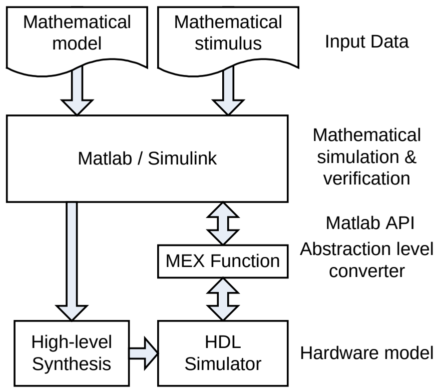

Fig. 5. Matlab open environment hardware synthesis and simulation example

5.2 SystemC programming language extension

The C/C++ language is very often used at the early stages of the design as a modelling environment. In its concept it is a programming language and it is not suitable for modelling concurrent process typical for hardware and hardware-software systems. Adding to a standard language a specific library of classes that introduces concurrency mechanisms and data flow with the use of transactions allows to extend possible areas of usage of the C++ programming language. Nowadays C++ can be treated as a universal language. SystemC has introduced a cycle based simulation process and several benefits that were available only in HDL simulators (Synopsys, 2002). Combining a typical programming language with a cycle based simulation capable environment gives an opportunity for integrating the description of different levels of abstraction and belonging to different domains (hardware / software). The transaction based methodology presented in the paper allows for design development and refinement according to simple rules from idea to implementation-ready prototype. Design process is carried out with a C++ compiler. The main disadvantage of the C++ environment is the lack of proper visualization and diagnostic tools. It should be pointed out that the development of C code is not the primary task in SystemC environment but in system level design. One of simulation environments proposed for SystemC system level design is CoCentric from Synopsys (Synopsys, 2003). A synthesizable subset of classes can be automatically synthesized by Celoxica tools. Finally can be obtained functional equivalent of description in hardware.

5.3 Standard Hardware Description Languages

For several years VHDL and Verilog have been used in the domain of hardware design. Those languages occupy an established place in the electronics system design. They are well described by IEEE standards. An important fact is a constant development of HDL synthesis tools that allow to obtain fully functional circuits from the HDL description. Nowadays those languages can be used not only as the design mainstream but also as an intermediate form between high level synthesis or other graphical tools that aid high-level design.

5.4 Graphical tools for high level design

A graphical representation is much easier for the human than a textual one. Determining changes or relations between components on a drawing is much easier than analysing textual descriptions in HDL. In order to simplify the high level design, the engineer can use block diagrams and graphical FSMs. Both are available in Aldec tools. The Block Diagram Editor allows to easily assemble HDL modules into a functional circuit (Fig. 6). Instantiation of different HDL components is much easier in the graphical

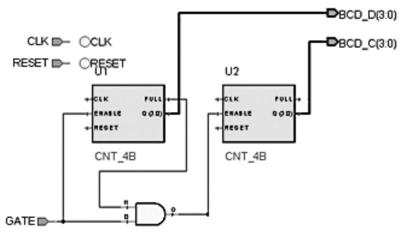

Fig. 6. Schematic of a circuit in Block Diagram Editor

environment than writing it as text. Designer also can determine block dependencies and connections on the schematic.

Finite state machines are basic components in the design of control circuits. Designers were used to drawing FSM's on paper. After that they were translated to HDL. Delivering a tool that allows for drawing FSMs and automatic HDL code generation reduces overhead connected with transforming an abstract drawing into a synthesizable description (Fig. 7). Again, a graphical representation allows for efficient designing.

5.5 *Environment for mixed design*

In the real world, the design is made from different components. Each part uses its own design methodology. There is an increasing need for integration of design environments in order to enable using different techniques and components in one design. Design environment should combine simulation of HDLs as well as SystemC. Not only simulation is important but also design management and visualization tools. Design management tools allow to keep track of all components that belong to the design. Visualization tools can graphically present the relationship between components and simulation results. They make it possible to quickly analyse design changes or dependencies in the graphical form that is more natural for the human perception then text. Active-HDL from Aldec combines all described simulation technologies and tools for the graphical result representation and design.

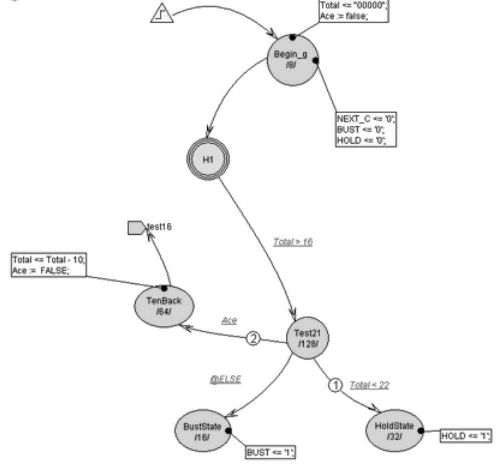

Fig. 7. FSM diagram view in the graphical editor

6. Conclusion

Increasing complexity of problems that are solved require new approaches to the design process. Two basic approaches were presented based on algorithmic-mathematical concept and on transaction system description. Both methods require appropriate tools for abstract model development. After the verification of virtual description automatic synthesis tools allow to obtain functional circuit. Graphical representation and analysis tools make the design and verification process easier and more efficient.

References

Altera "Stratix II Device Handbook" available at www.altera.com

De Micheli G. "Synthesis and Optimization of Digital Circuits", McGraw-Hill, Inc. 1994

Devadas S., A. Ghosh, K. Keutzer „Logic Synthesis", McGraw-Hill, Inc. 1994

Ferrari A. and A. Sangiovanni-Vincentelli, System Design. "Traditional Concepts and New Paradigms". Proceedings of the 1999 Int. Conf. On Comp. Des, Oct 1999, Austin

Gajski, D., SpecC: Specification Language and Methodology. Kluwer Academic Publishers, Norwell MA, 2000

Müller W., W. Rstenstiel, J. Ruf, „SystemC Methodologies and Applications" Kluwer Academic Publisher, 2003

Sigmon K. T.A. Davis "Matlab Primer Sixth Edition" CRC Press, 2001

Synopsys "SystemC Version 2.0 User's Guide", Synopsys Inc., CoWare Inc., Frontier Design Inc., 2002

Synopsys "System Studio" available at http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html (2003)

Baker M. "Creative Uses for Spartan-3 dedicated resources", Xilinx 2003

Goddard I. M.Trepanier "The Role of FPGA-based Processing in Medical Imaging" VMEbus Systems Apr. 2003 available at http://www.vmebus-systems.com

Hauck S. "The future of Reconfigurable Systems" Keynote Address, 5th Canadian Conf. on Field Programmable Devices

Hezel S. A. Kugel R. Mner D.M. Gavrila „FPGA – based template matching using distance transforms" Proc 10th Annual IEEE Symp on Field Programmable Custom Computing 2002

Xilinx. "The Programmable Logic Data Book", San Jose 2003 (CD-ROM edition)

Xilinx App.Note XAPP132 "Using the Virtex Delay Locked Loop", available at http://www.xilinx.com/bvdocs/appnotes/xapp132.pdf

Xilinx App .Note XAPP290 "Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations" available at http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf