2006-1777: DEVELOPMENT AND INTEGRATION OF A DIGITAL CONTROL LABORATORY WITH A DIGITAL SYSTEM LABORATORY AT YOUNGSTOWN STATE UNIVERSITY

Ben Shaw, Youngstown State University

Faramarz Mossayebi, Youngstown State University

FlexARM1: An ARM Based IP Core for the UP3 Education Kit

Introduction

Today's embedded solutions require a rapid product development time to meet strict market demands[1]. It is essential for system design engineers to verify complex designs in hardware before final implementation. In order for upper level undergraduate students to gain exposure to this verification process, a system level prototyping environment is a necessary tool to provide hands on experience for realizing complex digital systems. System Level Solutions[2] and Altera[3] offer the UP3 Education Kit as a low cost prototyping platform for system level co hardware/ software development. The UP3 features a powerful Altera Cyclone FPGA[4] and an abundance of I/O, peripheral, and memory components allowing for intellectual property (IP) design, prototyping and testing using a hardware descriptive language (HDL).

Modern FPGAs are equipped with features that were not previously available. Today's FPGAs usually come with phase-locked loops, low-voltage differential signaling, hardware multipliers for DSP, memory, programmable I/O, IP cores, and microprocessor cores[5]. Because of these features, FPGAs are now a viable choice for the implementation of entire system on a chip, the so-called system-on-a-chip (SoC) concept. HDL design flows using Verilog, VHDL, or SystemC along with today's advanced logic synthesis tools support the rapid development of these high density programmable SoCs. It is important for students to develop the necessary skills and experience for this emerging technology with the use of available IP cores[6], EDA vendor's logic synthesis tools, and FPGA development boards.

At the heart of every SoC lies a CPU responsible for coordinating the tasks of various components of the system. This paper presents the FlexARM1 processor and its IP design methodology, which could be used as the central core for teaching processor-based systems on FPGAs. This also provides a library of synthesizable VHDL modules for this RISC based CPU, which are currently used in the senior level Computer Architecture course at our institution.

FlexARM1 Design Methodology

The FlexARM1 architecture based on the ARM9 family of commercially available processors developed by ARM[7], utilizes the same Harvard architecture and memory mapped I/O concept as the ARM9. The FlexARM1 is a load/store architecture, which implements the following addressing modes from the ARM architecture: data processing immediate shift, data processing register shift, data processing immediate, load/store immediate offset, load/store register offset, and branch and branch with link. All load/store addresses are determined from the register contents and instruction fields only. The FlexARM1 also employs a conditional execution option for each instruction in order to maximize execution throughput. These include fourteen available conditions allowing for the equality and inequality testing of the condition code flags zero, carry, overflow, and negative. The FlexARM1 instruction set is fully compatible with the ARM instruction set. However, time constraints were an issue in implementing every instruction. Some of the more notable instructions not implemented as of

now are coprocessor instructions, multiple register loads, status register access instructions, and exception-generating instructions[8].

The proposed IP methodology allows the software and hardware paths to be developed concurrently as shown in Figure 1. This approach introduces the students to current SoC design issues, such as bridging the design gap between the software and hardware engineer[9]. This design flow may be partitioned within a project group. Thus, allowing the software and hardware paths to be developed simultaneously. During the time the hardware components of the CPU are being designed, test vectors are created using the FlexARM1 instruction set with the uVison3 ARM assembler developed by Keil[10]. The assembler translates the test instructions into machine code as an output file (.hex) in Intel Hex Format.

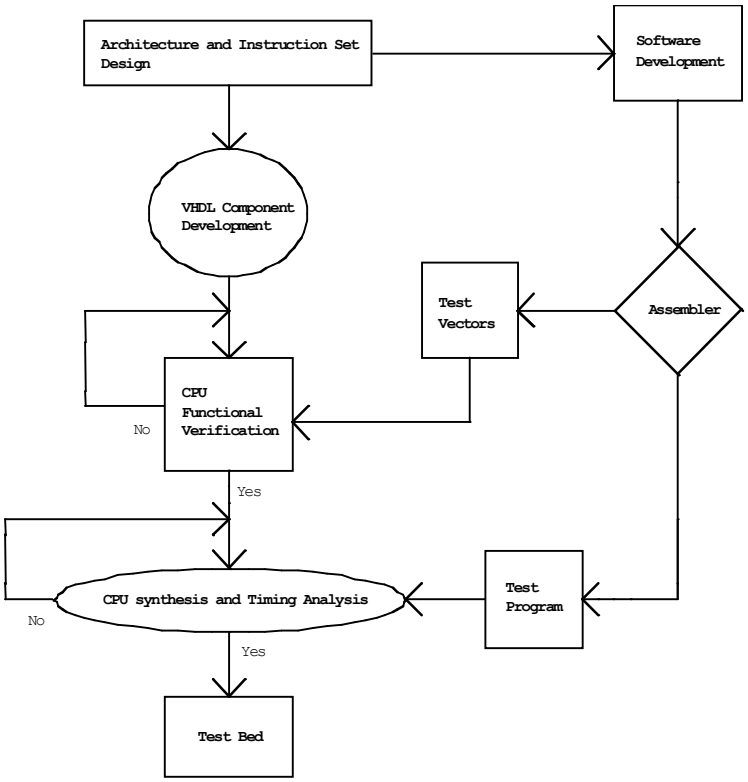

Figure 1: FlexARM1 Design Flow

Different test vector sequences simulate certain CPU operations. For instance, the file Forward.hex runs a series of FlexARM1's single clock cycle data-processing instructions to test the forwarding of the 5-stage pipeline and verify there are no data hazards found in the instruction stream. The software development also includes the writing of test (application) programs for the synthesizable FlexARM1 core. These application programs ensure overall functionality and provide a demonstration of the FlexARM1 operating in hardware. We are presently developing several application programs (which we hope to finalize and demonstrate at

the time of presentation of this work). The design flowchart in Figure 1 illustrates where the software path interacts with the hardware path.

The first stage of the design process consisted of analyzing the ARM9 architecture[8] along with its instruction set. The focus here was to understand how the ARM processor worked internally, i.e. programmer's model, addressing modes, execution unit, etc; and how they affect the operation of the processor. From this analysis, one can compare the architecture of the processor with how well its components would map into the Cyclone FPGA architecture. For example, the ARM datapath incorporates a barrel shifter and a hardware multiplier. Due to the large amount of interconnect associated with these components, they utilize large amounts of valuable area on the Cyclone FPGA. We decided to use a scaled down version of the barrel shifter in order to implement multiplication in software to conserve area. However, depending on the features of the target FPGA this might not be an issue due to embedded resources such as hardware multipliers. After weighing several design issues, a preliminary block diagram of the FlexARM1's datapath was prepared for the VHDL component development phase.

Most of the components of the FlexARM1 are found in other popular commercial CPUs. These components make up our library of synthesizable VHDL models. The list of synthesizable models includes an ALU, Register File, Barrel Shifter, I/O controller, ROM, and RAM. The other components found in our library are specific to the ARM architecture but can be used as a basis for other CPU architectures. They are comprised of several units, which control the pipelined dataflow of the FlexARM1 and handle each instruction. The Control Unit directs the operation of the pipeline, whereas, the Data Hazard and Forwarding Units detect hazardous sequences and forward the correct data to the appropriate pipeline stage. A data hazard may occur if a register is read before the result is written back to the register from a previous instruction. The Immediate Unit and Instruction Decoder are the last two components of the FlexARM1 and are responsible for generating the immediate constant and the decoding of the FlexARM1 instructions.

The VHDL Component Development phase consists of the following three steps: Functional, Logic Usage, and Timing as shown in Figure 2. Each component of the FlexARM1 followed this procedure, which allowed us to experimentally show how certain modules synthesized in the Cyclone FPGA according to their VHDL abstraction level. Examples of VHDL abstraction levels are structural, behavioral, and dataflow (also known as register transfer language, RTL). Lower levels of abstraction, i.e. dataflow and structural, usually lead to more efficient logic usage on the FPGA. As this is the case with the more complex components, basic building blocks synthesize equally as efficient at the behavioral level, i.e. multiplexors, adders, registers, etc.

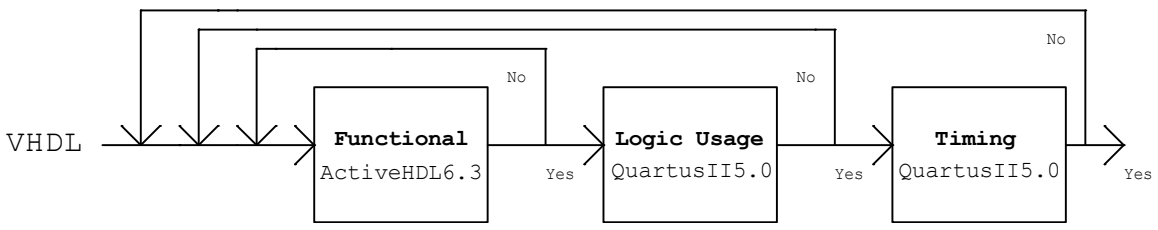

Figure 2: VHDL Component Development

The Functional block serves to verify that the component under development is working properly at the behavioral level with testbenches in a simulation environment. The functional verification was confirmed with Aldec's VHDL simulator, ActiveHDL 6.3[11]. For the Logic Usage block phase Altera's QuartusII 5.0 logic synthesis software is utilized to determine the logic resources used on the Cyclone FPGA for each synthesizable component. The logic resources in the Cyclone FPGA are known as logic elements (LE). At this step, we compare the behavioral description to its lower level abstraction to determine which provides a better synthesizable result, as measured by LE usage. Lastly, the Timing block's input is also a synthesizable VHDL description, which QuartusII compiles to determine the maximum timing delay. If the inputs and outputs of the component are registered, QuartusII will calculate a maximum frequency and a propagation delay of the component. Table 1 shows the experimental results from the VHDL Component Development phase for the FlexARM1's ALU.

Table 1: ALU synthesis results

Logic Element Usage		Propagational Delay (ns)
Behavioral Level	**Dataflow Level**	
459 / 5980 (7%)	175 / 5980 (2%)	6.61

After each component has been tested, the CPU functional verification phase combines all the components into a VHDL hierarchical design to form the datapath and control unit. This stage performs a functional verification on the VHDL hierarchy model shown in Figure 3 to confirm the CPU is working properly. ActiveHDL's verification software features automated testbench capabilities, which allow different test vectors to be used with the same testbench. Once the designer has created the desired testbench along with its specific test vectors, the testbench can be used numerous times to perform automatic verification of multiple revisions of a HDL design. If the design does not pass this stage, the hardware debug issues must be resolved before logic synthesis. The fully functional CPU core was synthesized by QuartusII. The results from FlexARM1's logic synthesis are shown in Table 2.

The test bed in the last stage of the design flow is the UP3 development board and a monitor. After the functionality and timing of the CPU core is verified, the CPU can be downloaded to the UP3 board for hardware testing. Due to the rich features of the UP3 board, there are many options for the CPU to be interfaced with in order to verify its operation in hardware. The test application we set up involves the VGA port. A VGA IP core was added which interfaced the FlexARM1 to the VGA port allowing us to output internal registers to a monitor for display in real time. The VGA core is responsible for generating the vertical and horizontal sync signals along with the three RGB signals required for video output.

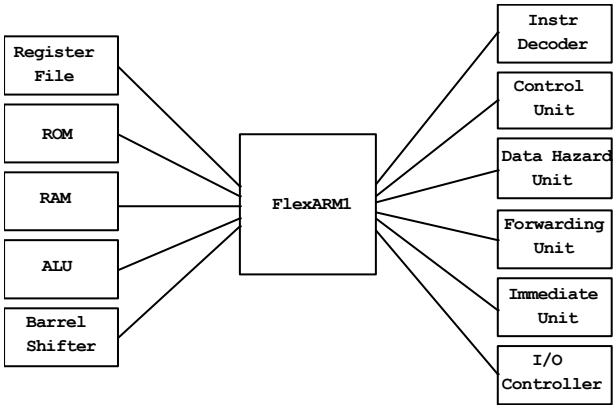

Figure 3: Hierarchy VHDL Model

Table 2: FlexARM1 synthesis results

Logic Elements Used	Memory Bits Used	Fmax (MHz)
2104 / 5980 (36%)	13200 / 92160 (14%)	55

Conclusion

The FlexARM1 IP methodology presented provides a design flow geared toward helping students become familiar with current FPGA design trends with an emphasis on verification. The FlexARM1 processor's VHDL code has been successfully implemented in the Cyclone FPGA and is available for real time demonstration. A set of application programs for this test bed is currently under development. This versatile hardware/software platform can be used to enable students to learn how to "quickly" implement processors in hardware and interface them to various I/O, memory, and communication protocols, as we plan to do so for the spring of 2007 offering of the computer architecture course at our institution.

References

1. M. Thompson, "FPGAs Accelerate Time to Market for Industrial Designs", EE Times Online News, July 2, 2004
2. System Level Solutions, Inc., 14100 Murphy Ave., San Martin, CA 95046, www.slscorp.com
3. Altera Corporation, 101 Innovation Drive, San Jose, CA 95134, www.altera.com
4. Cyclone Device Handbook, Volume 1, www.altera.com/literature/lit-cyc.jsp
5. J. Kriegbaum, "FPGA's vs. ASIC's", EE Times Online News, Sept. 13, 2004
6. www.opencores.org
7. ARM, Inc., 141 Caspian Court Sunnyvale, CA 94089, www.arm.com
8. ARM Architecture Reference Manual, www.arm.com/documentation/books/1183.html
9. J. Bruister, "Bridging the Hardware/Software Design Gap", TechOnLine, March 1, 2001

10. Keil Software, Inc., 1501 10th Street Suite 110, Plano TX 75074, www.keil.com
11. Aldec, Inc., 2260 Corporate Circle, Henderson, NV 89074, www.aldec.com.