A System-On-Chip Course Using Altera's Excalibur Device and Quartus II Software

Authors:

Ahmet Bindal, Computer Eng. Dept., San Jose State University, San Jose, CA 95192, ahmet.bindal@sjsu.edu
Sandeep Mann, Computer Eng. Dept., San Jose State University, San Jose, CA 95192, ssm11b@yahoo.com

Abstract *—The authors have developed a senior -level undergraduate System -on-Chip (SoC) course at San Jose State University that emphasizes SoC design methods and hardware -software co -design techniques. The course uses a "real world" design project as the teaching vehicle, and it implements a SoC platform to control a five -axis robotic arm using Altera's state -of-the-art Excalibur chip. The Excalibur chip contains both an embedded ARM processor and a Programmable L ogic Device (PLD) array. The course goes through a complete hardware -software co -design flow from implementing custom hardware devices on a PLD to developing an embedded algorithm in a state -of-the-art design environment for a complete SoC solution. Stud ents are introduced to "real world" design methods by acquiring their design data from the Internet in the form of current data sheets and application notes and using them in conjunction with the Altera EPXA1 board and Quartus II design environment to crea te their design. Students learn to use the design environment by examining the sample design files in the development kit and following the step -by-step instructions towards creating a simple embedded application. After this familiarization process, stud ents define the architectural specifications of a memory -mapped servo controller, implement it in the Excalibur's PLD array and interface this device with the ARM processor's internal bus in order to control each robotic arm servo. Functional regression t ests and post -synthesis timing verification steps are applied to the servo controller following the implementation phase. Subsequently, students integrate the servo controller with the rest of the system and perform board -level functional verification tes ts to observe whether or not the robotic arm can move an object from a source to a destination point accurately. Students also develop an embedded algorithm, which translates user inputs in Cartesian coordinates into robotic arm movements in spherical coo rdinates during laboratory sessions.*

Index Terms *— System -On-Chip, Hardware/Software Co -Design, Excalibur Device, Quartus II.*

INTRODUCTION

As the popularity of System-on-Chip (SoC) design approach grows in prototyping a product, students must be introduced to the necessary tools and knowledge that will make them immediately productive upon graduation. To achieve this goal, two important issues must be dealt with. The first issue is students' reluctance to learn "advanced" digital design topics from library-based technical articles or web-based, state-of-the-art company resources. Students still prefer the traditional, theory-based textbooks, which often lack practical application and up-to-date information on rapidly evolving technologies. The second issue is fact that commercial CAD tools used for the development of SoC platforms can actually aid students' learning of complex digital design concepts such as system architecture. This manuscript addresses possible solutions to both of these issues.

This SoC course differs from the earlier embedded system design courses published in the literature [1-4] since it requires students to obtain current design information from the Internet. Using web-based resources and commercial CAD tools, students can perform a complete hardware-software co-design consisting of hardware designs in Verilog and software designs in C and Assembler. The earlier courses taught software development on a system board and hardware development on a Field Programmable Gate Array (FPGA). In some courses, the board containing a configurable processor was connected to a second board to complete the system [5]. The Altera's Excalibur chip used in this course contains both an embedded ARM processor and a Programmable Logic Device (PLD) array, and can implement both hardware and software designs in a single chip [6-7].

COURSE OBJECTIVES

Throughout their junior and senior years, Computer Engineering students at San Jose State University learn fundamental, text book-based hardware and software topics from digital logic design to operating system design. However, as soon as they

face senior graduation projects where they have to apply their course background to an actual project, they often encounter difficulties in carrying out the design. Therefore, to eliminate these transitional problems and increase students' ability to link the previously taught, core hardware and software subject matters, we created a course in the form of a real-life design project, which requires detailed understanding of hardware and software co-design techniques. Even though students were exposed to only one chip set throughout the course, the design methods taught were not specific to one chip set or the manufacturer and could be extended to design other SoC platforms composed of different chip sets.

At the end of this course, three main objectives are accomplished:

Learning Hardware/Software Co-Design Principles and Methodology:

Hardware Design:

- Understanding the architecture of a typical development board and the functionality of on-board devices [6, 8, 9, 10].
- Understanding the Excalibur's chip architecture and the functionality of its components [7, 11].
- Understanding the Excalibur's internal Advanced Microprocessor Bus Architecture (AMBA) [12].
- Defining a system architecture using the Excalibur chip and the on-board modules.
- Designing and implementing a custom hardware and its interface with AMBA on the Excalibur's PLD array.
- Integrating the custom hardware on the Excalibur's PLD array with the rest of the system and verifying the system functionality using a logic analyzer.

Software Design:

- Understanding the details of the boot code and how it relates to initializing and enabling the on-board modules and the on-chip devices.
- Developing a mathematical algorithm for the robotic arm movements and integrating it with Quartus II as the application software.
- Executing a verification plan for robotic arm movements between selected source and destination coordinates.

Electromechanical Device Control:

- Understanding servomotor operation and control as it is the choice of the electromechanical device for this project.
- Building a five-axis robotic arm using servomotors for each axis.

Learning To Use State-of-the-Art Design Environments:

- Understanding the capabilities of Cadence Design Environment for Verilog design entry, functional verification and waveform browsing for device(s) implemented in the Excalibur's PLD.
- Understanding the capabilities of Quartus II Design Environment [13] for design synthesis, place-route, integration and post synthesis timing verification for device(s) implemented in the Excalibur's PLD array.

Promoting Web-Based Learning and Library Research:

- Downloading and understanding the state-of-the-art information from ARM, Altera and Toshiba sites for the actual design work.
- Encouraging students to conduct research from library-based resources and technical papers.

COURSE OVERVIEW

The course is divided into two sections. The first section summarizes the experiments leading to a complete SoC design and the methodologies employed during these experiments. The second section consists of teaching material related to the experiments during lecture hours.

Experiments

Prior to experiments students conduct extensive research in Altera's EPXA1 board architecture [6], Excalibur's chip architecture [7], AMBA protocol [12] and understand the ARM Programmer's Model [14, 15]. They gather information from data sheets and sample designs from the web to understand how a system architecture and data-path to control an electro-mechanical device can be defined.

Experiment 1: Learning Quartus II Design Environment

The first experiment allows students to become familiar with the EXPA1 board and the Quartus II design environment. This experiment has no design involvement; students simply follow Altera's user guide, "EXPA1 Development Kit, Getting Started" [16]. Using the steps and sample files in this guide, students create a simple embedded application. They learn how to configure the devices in Excalibur's Stripe and develop the software settings for it. At the end of this experiment, students are able to download their first "Hello Application" to the EXPA1 board.

Experiment 2: Learning How to Actuate a Servomotor and the Five-Axis Robotic Arm

The second assignment is to understand the control mechanism to rotate a servomotor at a desired angle between –45° and +45°. Each servo is pulse-driven and the angular position of its arm depends on the width of the pulse it receives during its operation as shown in Figure 1.

Students also build the Lynxmotion robotic arm used in these experiments. This robotic arm has five axes of motion [17] and each motion is controlled by an individual servo whose sweep range is between -45° and +45°. The complete set up is shown in Figure 2.

Experiment 3: Developing the Embedded Algorithm in Quartus II Design Environment

In this assignment, students develop an embedded algorithm for operating the robotic arm between user-defined source and destination coordinates. First, the algorithm aligns the robotic arm with the object. Then it extends the arm to hover over the object and lowers towards the object. The amount lowered is determined by the object height supplied by the user. Subsequently, the robotic arm grips the object and raises it until the portion of the robotic arm between the wrist and the elbow becomes parallel to the ground. Next, the arm moves in an arc to align with the destination point. The last task of the algorithm is to lower the object at the destination point and release it. The arm returns to its idle position once it completes its task.

This application follows a set of equations to translate the user inputs in Cartesian coordinates to servo arm angular sweeps of the robotic arm.

Experiment 4: Actuating the Five-Axis Robotic Arm Using the On-Chip Servo Controller

(a) Defining the System Architecture and Data -Path Using the On -Chip Servo Controller:

This assignment constitutes the most challenging part of this course, because at this point, students have to formulate a possible data-path to transfer the user data from the Graphical User Interface (GUI) to the servo controller using only the existing modules on EXPA1 board and the devices on Excalibur chip. To complete this task, students refer back to the sample designs they have already examined at the beginning of the course, consult with the instructor, and modify a sample architecture until a possible prototype data-path is obtained. Needless to say, this method is recursive and time-consuming; it requires detailed understanding of how data format changes throughout the selected data-path based on the functionality of the selected devices and how this data propagation relates to the boot code.

One such data-path is shown in Figure 3. In this figure, the user data packet carrying the new servo ID and position first arrives to the UART FIFO in Excalibur's Stripe, then to the UART Receive (Rx) Buffer in the SDRAM and finally to the user-defined servo address space in the SDRAM as illustrated with solid lines in Figure 3. This address space consists of the old servo position, the new servo position and a valid bit to indicate whether the old servo position is updated with the new servo position for each servo. Each servo data residing in this address space is directed through the Stripe-PLD Bridge to the servo controller's dedicated register file. The servo controller subsequently uses these new register values to move the designated servos in the robotic arm.

(b) Implementing the On -Chip Servo Controller in Excalibur's PLD:

Throughout this experiment, students are encouraged to conduct brainstorming sessions with other groups and consult possible solutions with the instructor to acquire the most important element of a design process: the correct thought pattern of how to conceptualize and synthesize a digital system from specifications.

Once the data-path is identified as in Figure 3, students start defining the micro-architecture of a memory-mapped servo controller, which is responsible of generating variable-length pulses to move a servo arm between –45° and +45° for each servomotor in the robotic arm. Students also interface this servo controller with five servo registers where each register serves a servomotor in the robotic arm and each is programmed with user data values supplied by the GUI.

As a design constraint, students are allowed to use only three functional blocks to build their version of the servo controller micro-architecture: a counter, a control unit and a data-path unit, all operating at 25 MHz [16]. First, they have to understand the purpose of each block in the design to generate servomotor pulses in Figure 1. Second, they need to produce

detailed schematics of each block, define their I/O ports, interconnect these ports and sketch a timing diagram composed of data-path and control signals to show the complete servo controller functionality.

In this design, the purpose of the counter is to count the number of clock cycles up to 4.165 million since this value represents the end of the 60 Hz cycle of the servomotor in a 25 MHz clock domain as shown in Figure 1. The purpose of the control unit is simply to monitor the counter output and reset the counter by generating a flag when the counter output reaches 4.165 million. The purpose of the data-path unit is to compare the counter output with a particular servo register value, ServoOut0 through ServoOut4, and generate a pulse for the corresponding servomotor as shown in Figure 1.

When students configure the I/O of each of these three functional blocks and interconnect them as shown in Figure 4, they obtain the first version of the servo controller micro-architecture. To obtain a more detailed, second version, students choose to subdivide the data-path into smaller functional blocks.

One such design example is shown in the inset of Figure 4, in which the data-path is composed of a comparator, five memory-mapped servo registers and an address decoder. According to this design, as long as the counter output is less than the value in the servo register, the comparator generates one at PulseOut; otherwise, the PulseOut value becomes zero. This is achieved by using a subtractor as a comparator and assigning the sign bit of the subtractor to be PulseOut of the comparator. To rotate the servo arm to –45° from neutral, one needs a value of 250,000 in the servo register; therefore, as long as the counter output is below this value, the comparator generates one at PulseOut for that particular servo. A servo arm rotation to 0° requires a register value of 375,000 and +45° a value of 500,000.

Students are also informed to be aware of synchronizing all five servomotors to maintain good robotic arm coordination when moving objects. One good solution to achieve servo synchronicity is to employ two sets of servo registers as seen in Figure 4. The first register set, RegOut0 through RegOut4, records user updates happening sporadically within a 60 Hz clock period. These updates are simultaneously transferred to the second set, ServoOut0 through ServoOut4, when the reset flag, Restart, becomes high at the end of a 60 Hz period.

Students perform Verilog design entry, initial logic verification and waveform browsing tasks using Cadence Design System's design environment.

(c) Implementing the AMBA Interface for the On -Chip Servo Controller:

After designing the servo controller and defining its I/O ports, students start designing the AMBA slave interface that connects the servo controller to the embedded Stripe. The servo controller interface is responsible for translating the AMBA address and control signals to the address and controls of the servo controller in the PLD array. In addition to this translation, the interface also resolves timing differences between the bus and the servo controller. AMBA protocol requires that the address and control signals be sent one bus cycle before the data. This timing separation between address/control and data signals allows AMBA to pipeline its operation. On any given cycle, an AMBA master releases the "next" address and control signals to the bus while the "present" data is being processed [12]. The servo controller, on the other hand, expects the address, control and data in same cycle.

(d) Integrating the On -Chip Servo Controller with the System and Verification Tasks:

Integrating the servo controller with the rest of the system requires instantiating the controller as one of the system devices in Quartus II and connecting its slave interface to the Stripe. Quartus II provides a schematic entry for this portion of the design. The steps describing this process, which consists of placing and routing the custom modules in the PLD array, their I/O pin mapping to the Excalibur device, post-synthesis timing verification are described elsewhere [13, 16].

Students also prepare a "formal" verification plan, which contains functional verification tests for the servo controller and system verification procedures for the EXPA1 board. Each test in this plan describes a specific input to be applied to the unit and the expected output generated as a result of this input. Students create a test bench in Cadence design environment to test the functionality of the servo controller before starting the implementation phase. The on-board system verification is performed using a logic analyzer after the servo controller is successfully implemented in Excalibur's PLD array. Students map each output of the servo controller, PulseOut [4:0], to the expansion header pins of the EPXA1 board and measure the pulse widths to check the accuracy of servo arm rotations using a logic analyzer. The final phase of system verification is done using GUI. In this verification step, students provide random source and destination coordinates for the object to the GUI in order to actuate the robotic arm. The algorithm mentioned above translates these coordinates into servo sweep angles, which become the contents of the appropriate servo controller registers. All the boundary coordinates at maximum allowable reach of the robotic arm are tested using this method.

Lecture Topics

The lecture part of this course consists of teaching the functionality of commonly used devices in various SoC platforms and design methodologies employed to carry out the experiments.

The part related to SoC devices is composed of two parts. The first part examines the functionality of on-chip components and the system bus while the second part deals with the functionality of on-board system peripherals.

The design techniques and methodology are taught in a generalized manner such that students are able to tackle different designs using either the EXPA1 board or a different chip set as mentioned earlier. The lectures are organized into topic sets as follows.

Topic Sets on On-Chip Components:

- Topic 1: The AMBA as the on-chip system bus is explained in detail. The functionality of the ARM 9 and 10 cores, the modes of operation, its stack pointer registers and interrupt controller are also explained in the same chapter. Students refer to the documents from ARM site [11].
- Topic 2: The architecture of a simple Direct Memory Access (DMA) unit is explained.

Topic Sets on On-Board Peripherals:

- Topic 3: The architecture of SRAM, its functionality and interface with the AMBA are explained. Students refer to the documents from Toshiba site [8].
- Topic 4: The architecture of SDRAM, its functionality, modes of operation and interface with the AMBA are explained. Students refer to the documents from Toshiba site [9].
- Topic 5: The architecture of E^2PROM, its functionality and interface with the AMBA are explained. Students refer to the documents from Toshiba site [10].
- Topic 6: The EXPA-1 boot code is examined to understand how to control on-board devices. Students refer to ARM assembly manual [14, 15] to understand various constructs in the boot code.

Following these lectures, student groups give thirty-minute technical presentations on various SoC topics. These topics range from component level such as a DSP core to system level such as a wireless LAN protocol.

STUDENT FEEDBACK

Once students accepted the course philosophy that required library and web-based learning, became experienced in Altera design environment and understood that expertise in such a design platform was an industry-desired skill set from a new engineer, their initial reluctance disappeared. Furthermore, "successful" student groups expressed a desire to develop different applications as graduation projects using the same design environment. The results of the student surveys are shown in Figure 5.

There are two parts in this bar chart: the solid bars correspond to "How important is this course to your educational objectives?", and the dashed bars correspond to "What percentage of the material covered in this course do you feel you have learned?".

The course is rated against the following 6 criteria from a scale 0 to 5:

- Being able to design a SoC platform, which consists of a CPU, memory sub-systems and system peripherals on a high-speed CPU bus such as AMBA.
- Being able to construct a verification test bench to test the functionality of a single SoC component and the entire SoC platform.
- Being able to design the appropriate AMBA interface for a custom peripheral such as a servo controller.
- Being able to know the details of contemporary high-speed bus protocols such as AMBA, which is commonly used in conjunction with ARM processor cores.
- Being able to use hardware and implementation skills to design SoC platforms with state-of-the-art commercial design environments such as Cadence Design Systems' LDV and Altera's Quartus II.
- Being able to communicate with lab partners and make technical presentations to fellow students.

One important observation from Figure 5 is that almost all students acknowledged the importance of such a course for their career path. Since this course is a "multi-disciplinary" in nature some students initially suffered from deficiencies in logic design while some others in operating system, and they had to build up their background as the course progressed. However, this fact partially prevented them to absorb the course contents fully.

CONCLUSIONS

This senior-level, undergraduate SoC course follows atypical approach to teach students to obtain their design information

from the Internet-based company resources and guides them through a complete hardware-software co-design and functional verification process.

The course combines a multitude of design elements, from embedded software development to hardware design and implementation. A small-scale expeditionary undergraduate research project that started from data sheets and sample designs quickly turned into an industry-standard SoC course containing hardware/software co-design techniques and methodology.

The laboratory projects require that students learn Altera's EPXA1 board architecture, the ARM Programmer's Model, sample designs and data sheets provided in the web and development kit prior to the design phase. Students examine these prior designs along with the EXPA-1 boot code before attempting to define a possible data-path composed of system devices and a memory-mapped servo controller in Excalibur's PLD. Upon understanding the data format at the input and output of each device on the proposed data-path, students generate functional specifications for a memory-mapped servo controller to be implemented in the Excalibur's PLD array. Designing the servo controller interface and the servo controller, performing functional regression and timing verification tests on the module are the routine implementation tasks before starting board-level integration and verification. The laboratory projects also help students refresh various skill sets such as C language and ARM Assembly language for embedded software implementation, and Verilog for hardware implementation. In addition, students are introduced to new design environments such as Cadence's Verilog design entry and functional verification, Quartus II for hardware/software portioning, synthesis and system timing.

The seminar is organized and taught out of the latest ARM and Toshiba related documentation on the Internet, and closely compounded with the laboratory assignments. Students are encouraged to find their own engineering solutions to the problems they encounter during the design phase of this course. Therefore, when compared to theoretical courses solely based on textbooks, this laboratory-based course gives extensive experience to undergraduate students and prepares them well for the industry.

REFERENCES

[1] K. Newman, J. O. Hamblen, T. S. Hall, "An Introductory Digital Design Course Using a Low Cost Autonomous Robot", *IEEE Trans. Educ* ., Vol. 45, August 2002, pp. 289.

[2] J. O. Hamblen, "Rapid Prototyping Using Field-Programmable Logic Devices", *IEEE Micro* , Vol. 20, May/June 2000, pp. 29.

[3] J. O. Hamblen, H. L. Owen, S. Yalamanchili, B. Dao, "An Undergraduate Computer Engineering Rapid Systems Prototyping Design Laboratory", *IEEE Trans. Educ* ., Vol. 42, Feb. 1999, pp. 8.

[4] A. Striegel, D. Rover, "Enhancing Student Learning in an Introductory Embedded Systems Laboratory", *IEEE Frontiers in Education* , Nov. 2002, pp. T1D7.

[5] J. Kairus, J. Forsten, M. Tommiska, J. Skytta, "Bridging the Gap Between Future Software and Hardware Engineers: A Case Study Using the Nios Softcore Processor", *IEEE Frontiers in Education* , Nov. 2003, pp. F2F1.

[6] EXPA1 Development Board, Hardware Reference Manual, September 2002, Version 1.1, http://altera.com/literature/manual.

[7] Excalibur Devices, Hardware Reference Manual, November 2002, Version 3.1, http://altera.com/literature/manual.

[8] Toshiba 8-Mbit SRAM (part no: TC554001AF), http://www.toshiba.com.

[9] Toshiba 64-Mbit SDRAM (part no: TC59S6432CFT), http://www.toshiba.com.

[10] Toshiba 64-bit (8Mx8 bits) NAND-type E^2PROM (part no: TC58V64BFT), http://www.toshiba.com.

[11] ARM922T Technical Reference Manual, Revision 0.0, http://www.arm.com.

[12] AMBA Specification, Revision 2.0, http://www.arm.com.

[13] Quartus II Development Software, Application Notes, http://altera.com/literature.

[14] Steve Furber, "ARM System-On-Chip Architecture", Second Edition, Addison Wesley.

[15] David Seal, "ARM Architecture Reference Manual", Second Edition, Addison Wesley.

[16] EPXA1 Development Kit, Getting Started, User Guide, January 2003 (see Boot-from-Flash).

[17] Lynxmotion 5-axis Arm Kit, http://lynxmotion.com.

ACKNOWLEDGEMENT

Authors thank Mike Phipps of Altera Corporation for the donation of 20 EXPA1 boards.

FIGURES AND TABLES

FIGURE 1
PULSES GENERATED TO DEFINE THE ANGULAR POSITION OF THE SERVO ARM

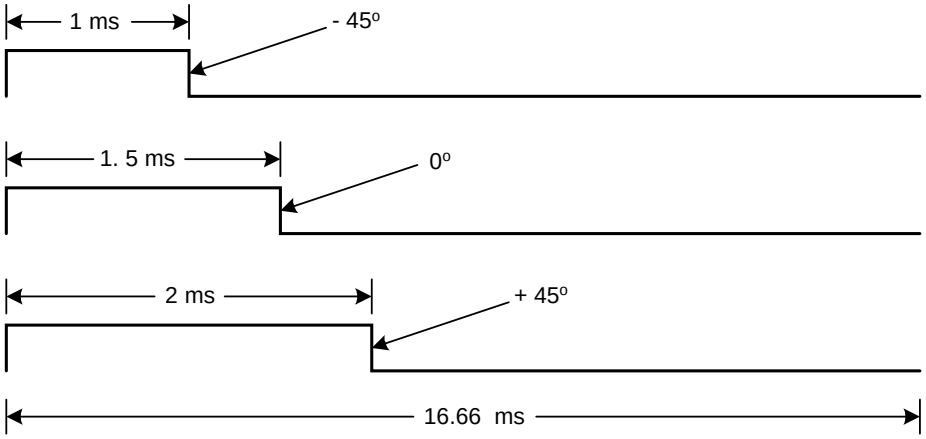

FIGURE 2
THE SETUP USED IN THE EXPERIMENTS

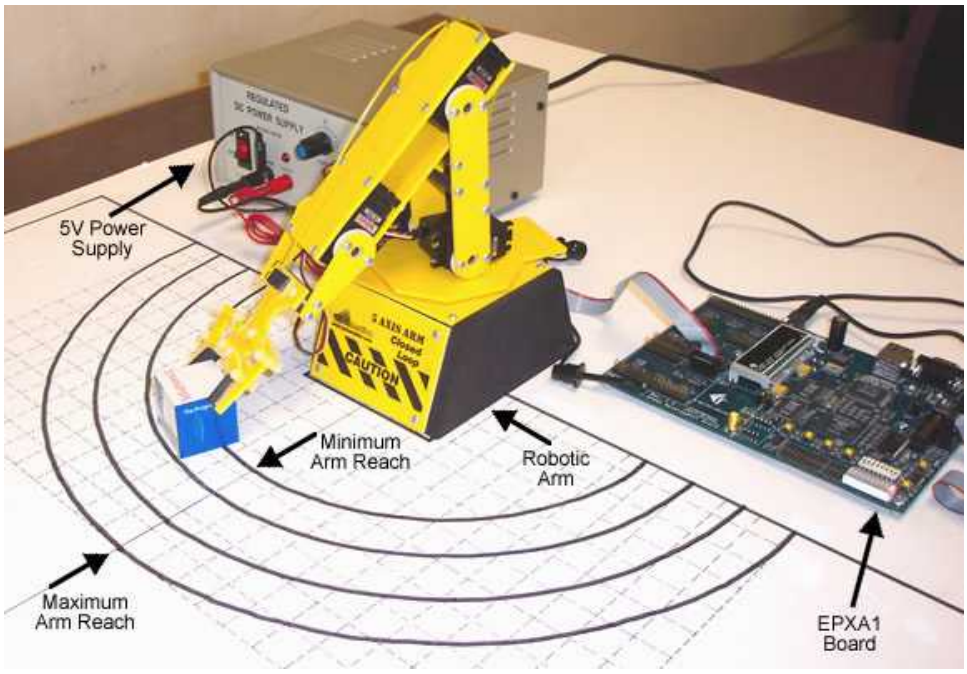

FIGURE 3
OVERALL SYSTEM ARCHITECTURE WITH THE SERVO CONTROLLER (THE NUMBERS ATTACHED TO EACH DATA PATTERN INDICATE THE SEQUENCE IN DATA FLOW)

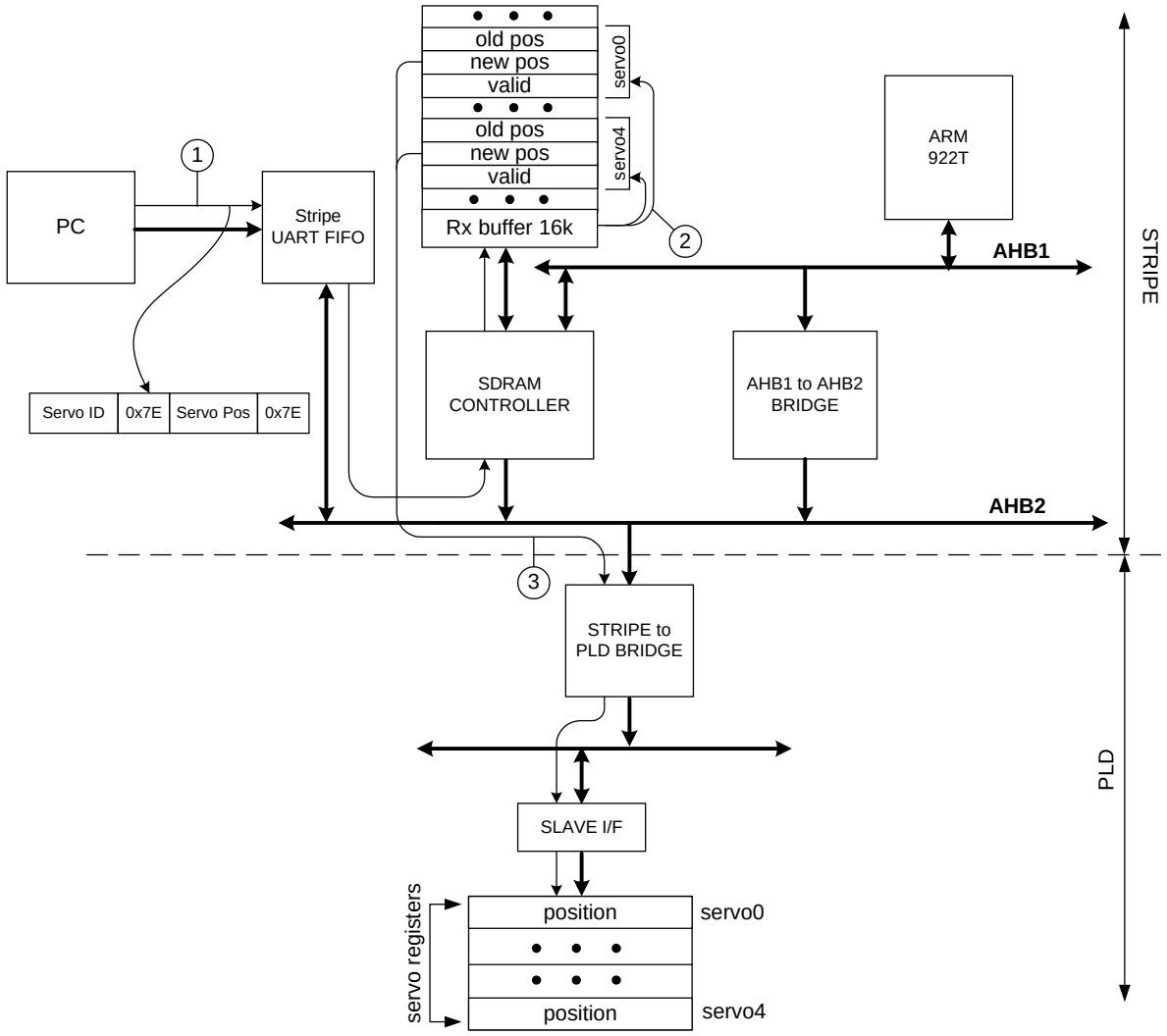

FIGURE 4
THE SERVO CONTROLLER ARCHITECTURE AND ITS INTERFACE WITH AMBA (THE INSETS ARE THE SIMPLIFIED CONTROL AND DATA-PATH UNITS FOR THE SERVO CONTROLLER)

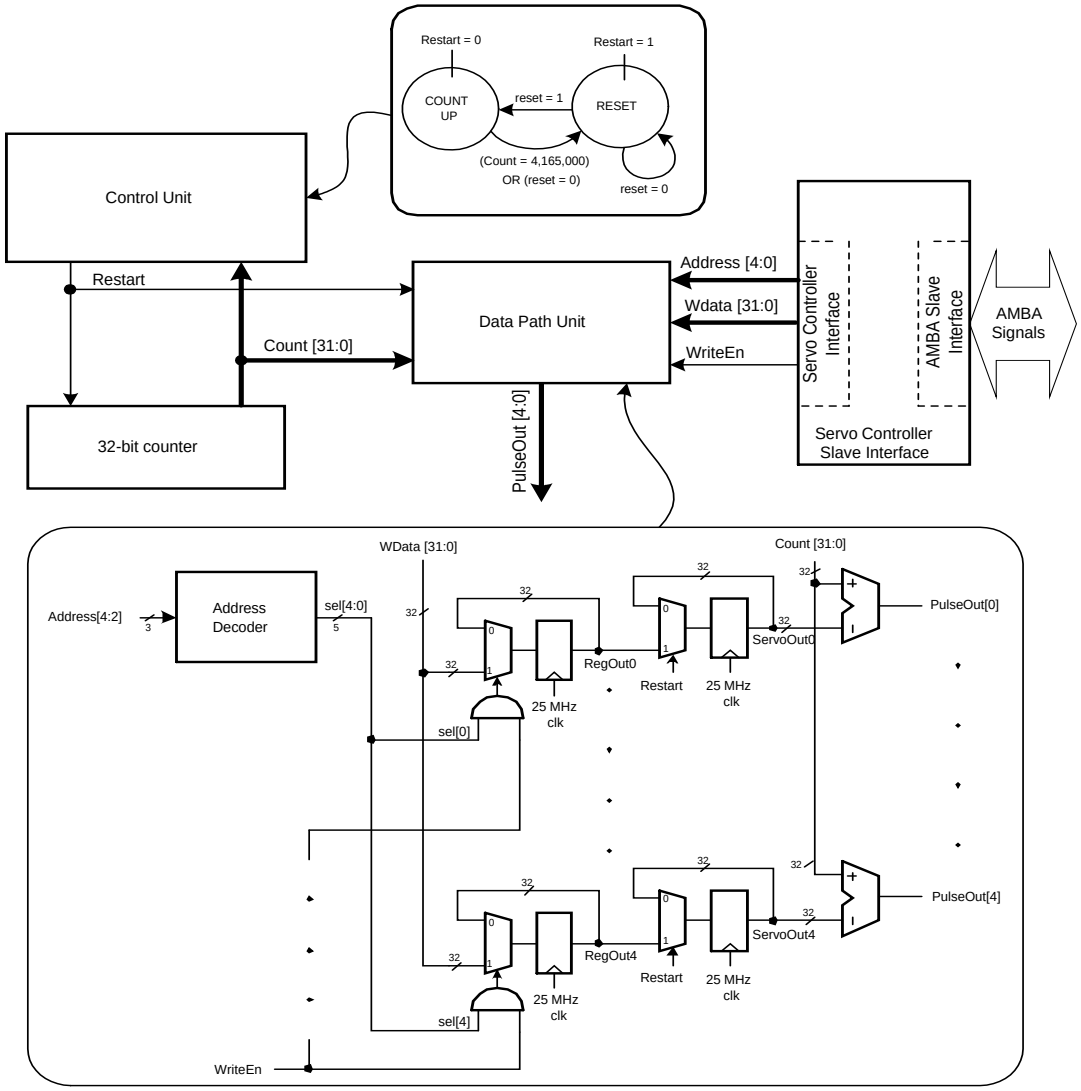

FIGURE 5
SPRING 2004 STUDENT SURVEY RESULTS