System-on-Chip Design and Implementation

Linda E. M. Brackenbury, Luis A. Plana, *Senior Member, IEEE*, and Jeffrey Pepper

***Abstract*—The system-on-chip module described here builds on a grounding in digital hardware and system architecture. It is thus appropriate for third-year undergraduate computer science and computer engineering students, for post-graduate students, and as a training opportunity for post-graduate research students. The course incorporates significant practical work to illustrate the material taught and is centered around a single design example of a drawing machine. The exercises are composed so that students can regard themselves as part of a design team where they undertake the complete design of their own particular section of the system. These design tasks range from algorithmic specification and transaction-level modeling (TLM) of the architecture down to describing the design at the register transfer level (RTL) with subsequent verification of their prototype on a field-programmable gate array (FPGA). With this approach, students are able to explore and gain experience of the different techniques used at each level of the design hierarchy and the problems in translating to the next level down. Throughout the module, there is emphasis on using industry standard tools for the modeling and simulation, leading to the use of the SystemC and Verilog hardware description languages and Cadence for the simulation environment.**

***Index Terms*—Integrated circuit design, large-scale systems modeling, system-level design, system-on-chip, systems engineering education, transaction-level modeling (TLM).**

I. Introduction

THE design of a modern system-on-chip (SoC) is a complex task involving a range of skills and a deep understanding of a hierarchy of perspectives on design, from processor architecture down to signal integrity. At a time when many organizations are walking away from the difficult challenge of teaching a SoC module that incorporates significant practical work on the system level design required to implement SoC, a course in this area is set to be given to third-year Computer Science, Computer Engineering, and Computer Systems Engineering students at the University of Manchester in the U.K. These students will have undertaken digital design in their first year and an introductory course in VLSI design in their second year.

The module aims to show how correctly working chips can be obtained by presenting and demonstrating the techniques and stages used in the design and implementation of chips. Since leading-edge, industry-standard software tools and languages are taught and practiced, the skills gained with this course are directly transferable into companies and post-graduate research in this area. Both the taught and practical components emphasize the methodology of the design process. This is accomplished through the use of a top-down design hierarchy, modeling and simulation of hardware using hardware description languages, partitioning of systems appropriately to deal with design complexity, and the use of computer-aided design (CAD) environments for providing a design flow from the system specification down to implementation, together with software tools for design verification.

This course differs from earlier courses in its consistent treatment of all design levels from specification to implementation. Due to the complexity of the topic, many courses emphasize one level of design. For example, in [1], register transfer-level (RTL) design and implementation is emphasized, and lecture topics focus on on-board peripherals and on-chip components. In [2], lecture topics cover a wider range of levels, but the practicals only cover the RTL implementations of a memory display and a video compressor. In [3], on the other hand, the emphasis is on system-level design using SystemC.

The guiding principles in the design of the course were the following:

- Consistency across levels of design: The architecture is constant although progressively refined. The same verification environment can be used in all practicals including the processor code developed by the students.
- A sample system that includes a commercial processor, multiple communications channels with routing and arbitration, and multiple clock domains is used in the practicals.
- Students are seen as part of a team: They work on a specific section of the system, but are aware of the complete system design. They have adequate documentation and clear interface specifications in order to take their design down from initial concept to implementation.
- External intellectual property (IP) is extensively used, and the integration of the IP and their custom design is carefully verified.
- Commercial tools and standard languages are used across all levels of design.

II. System-on-Chip Design Hierarchy

Both the lectures and the practical work follow the design methodology for top-down SoC design [4], [5]. This methodology partitions the design into a number of stages where one level is designed, tested, and modified until correct. The process then repeats at the next level down, beginning with the translation of the design from the upper to the lower level; unfortunately, this translation is not always a direct translation. The generic design hierarchy used is summarized in Fig. 1.

The design starts with the user requirements, which are then translated into a formal system specification written in a high-

Manuscript received August 19, 2008; revised January 21, 2009. First published August 07, 2009; current version published May 05, 2010.

The authors are with the Advanced Processor Technologies Group, School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K. (e-mail: lbrackenbury@manchester.ac.uk; plana@cs.man.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TE.2009.2014858

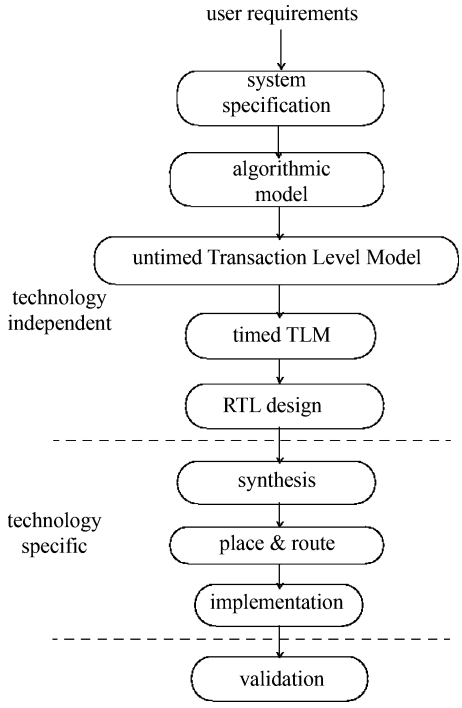

Fig. 1. Design hierarchy for SoC.

level programming language such as C or C++. This specification is the user's view of the system, and this model is modified until under testing it performs all the functions required by the user. This level is suited to the exploration of different algorithms for implementing functions, and while the external view provided has no details of any internal hardware, these may be implied from the algorithm. Once completed, the model then represents a "gold standard" by which all other models at other levels are compared.

Architectural design follows, identifying the functional blocks and their interconnections. Genuine SoC design demands the use of abstract modeling in order to give the design the flexibility to explore the hardware/software divide and different architectural arrangements without needing to specify low-level detail such as the communication protocol. For these reasons, a transaction-level model (TLM) [6] having computational blocks and separately modeled interconnecting channels is adopted at this level [7], [8]. Such a model allows the external behavior of the computational and channel components to be expressed as a set of information transfers between them. Such a high-level view of the system requires a high-level hardware description language for efficient modeling, with SystemVerilog [9], [10] or SystemC [11], [12] being the main choices.

Different TLMs are possible depending on whether the computational blocks and the channels are timed or untimed. Totally untimed models enable hardware/software and architectural exploration, while totally timed models give a performance indication. Both model types are used in the course to ease the transition between the system specification and RTL stages.

Hardware implementation at the RTL follows. Here, the internal functionality is described in terms of the registers and combinatorial logic functions required [13]; the former require storage elements and so have outputs that depend on inputs at a previous time, while the latter has outputs totally defined by the inputs at that time. The model is usually a behavioral description that describes the internal data flow and the operations required. The RTL description is usually written in the lower-level hardware description language of VHDL [14], [15] or Verilog [16], [17] since these are compatible with the CAD tools used at the lower levels of the hierarchy.

From this point onward, much of the process is automated by the CAD tools. Thus, an important feature of the RTL model is to obtain a description that can be synthesized by the tools. Synthesis takes the RTL design and translates this to a digital logic gate implementation on the targeted technology. Although practical considerations have dictated the use of a field-programmable gate array (FPGA) as the implementation vehicle in the practical work so that the translation and mapping onto the FPGA is a single process, the more usual target for SoC design is semicustom silicon. In this case, at the small transistor geometries used in current SoC of 130 nm and below, interconnection delays dominate the timing, and these are not accurately known until layout on silicon. Thus, the logic synthesis tool for the targeted technology needs to be timing-aware, and this is incorporated via a set of timing constraints specified by the user. The tool uses models to estimate worst-case interconnection timing, and so determines whether the user's timing requirements can be met.

The synthesized logic design is then placed and routed onto silicon. This CAD tool performs physically aware place and routing synthesis to ensure the integrity of all signals on the chip. It checks that pickup between adjacent signals, current flows, the supply voltage to logic elements, surface unevenness arising from multiple layers on the chip, and antennae effects are within limits that ensure the correct transmission of binary information between any two points. After checking the outcome of the placed and routed design for correctness against the top-level specification, and checking that the design is sufficiently toleranced with respect to operating and transistor variations, the design is ready for manufacture. Following fabrication, the resulting chip needs to be validated, usually using the same functional tests as at the system specification stage.

III. Course Objectives

A. Course Aims

Usually, in their first two years, students study the fundamentals of hardware and software design, ranging from computer technology to databases backed by appropriate practical exercises, which tend to be relatively short in scope and length. Third- and fourth-year courses are advanced modules on particular themes and/or courses based around a particular research area; the SoC module fits into both these categories. The primary objective of the module is to provide in-depth theoretical and practical insights into the design methodology, focusing primarily on the high-level issues of system modeling, IP core reuse, architecture modeling and testing, on-chip interconnect, and RTL synthesis.

The course aims to give students experience through practicing the methodology and the techniques required at each level of the design hierarchy. To this end, a single design problem

runs throughout the course. Although relatively simple and constrained, it is significant enough to illustrate all the problems likely to be encountered in more complex multiprocessor, multibus SoC such as arbitration, routing, and the crossing of time domains. Furthermore, the use of a single problem and the incremental nature of the practical work gives students a coherent view of the design flow with the steps that need to be taken in translating from one level to the next.

An important feature of the practical work is the understanding gained of industry-standard languages and CAD environment. Cadence is used for the simulation environment, while SystemC is used for the high-level architectural modeling. Verilog is used for behavioral modeling at the RTL because of its ability to be synthesized.

B. Learning Outcomes

The specific student learning outcomes resulting from completing this course unit are:

1) a knowledge and understanding of the principle industry-standard tools used in system-level design;
2) an understanding of the issues relating to on-chip interconnect, architecture, modeling, testing and design verification;
3) an understanding of the role of RTL synthesis, technology mapping, cell libraries, and timing closure in the SoC design process;
4) an ability to apply this understanding to the design of prototype systems;
5) an insight into future developments in SoC technology.

C. Contribution to Program Learning Outcomes

This module contributes to an overall program of study undertaken by Computer Science and Computer Engineering students in the following four areas:

- Knowledge and Understanding
  1) acquire knowledge in an advanced topic in Computer Science at the forefront of research;
  2) understand, apply and develop leading-edge technologies in computer engineering.
- Intellectual Skills
  1) use methodologies for the development of computational systems at an advanced level;
  2) solve problems in an academic environment that are also applicable to an industrial context.
- Practical Skills
  1) develop applications to satisfy given requirements.
- Transferable Skills
  1) write reports to a professional standard;
  2) give talks to a high level of proficiency.

IV. Course Overview

Lectures and the practicals run in parallel and are synchronized throughout the course, with each following the top-down design methodology. In this way, lectures and practical work support each other. This enables students to directly see the relevance of taught topics that they then have to apply in their exercise work. Similarly, the design example provides illustrations of good practice and techniques for use in the lectures to which the students can readily relate. Lectures not only include the technical aspects required for SoC design, but also the languages required to support the exercises undertaken.

TABLE I
Lecture Topics

Lecture Topic	Practical
setting the scene	Algorithmic Level Model (ALM)
system specification	
C++ system modeling	
TLMs (2)	
SystemC HDL (2)	untimed Transaction Level Model (uTLM)
assembly language	
timed TLM (2)	
translating to RTL (2)	
behavioral Verilog (3)	timed TLM (tTLM)
design for test	Register Transfer Level (RTL) and implementation
debugging	
power issues	
tools & verification	
design flow	
timing closure	
future of SoC	

Table I lists the lectures and practicals in time sequence. The lectures occupy 22 lecture slots at two per week, as shown in the left column of the table with the numbers in brackets indicating the number of lectures on a topic if more than one. The lectures follow the themes of: 1) What do you want? 2) How do you design it? 3) How do you implement it and get it to work? and 4) Where is it all going? In the first few weeks, the lecture sequence is informed by the need to provide students with the knowledge and understanding to undertake the practical exercises.

The practical work also occupies 22 h and concentrates on four areas of the design hierarchy: algorithmic, untimed TLM, timed TLM, and RTL followed by implementation. The most novel feature of the module is the scope of the practical work, where students are not only able to gain real experience of design at different levels of the hierarchy, but also able to get a view of the processes required in moving from one level to another through the use of a single example that contains all the essential elements of SoC design.

V. Design Exercises

The design example chosen is a drawing machine capable of accepting commands to draw a particular shape, computing the points to be drawn, and writing these into a memory called the frame store. The contents of this memory can then be displayed on a screen giving an immediate visual check on system activity and likely correctness.

This type of system partitions naturally into two sections, comprising an environment testbench and drawing machine modules. The latter perform the actual drawing of the desired shapes, while the former initiates the drawing operations and

can receive the results of the requests to draw shapes. Since it is difficult for students with no prior knowledge to get a system up and running, students are given a complete working basic system, at each level of the design hierarchy, which is capable of drawing and displaying points and lines.

The student task is to add another shape and to integrate their design into the system, making any changes required to the existing system they are given. The integration of their shape into the system is the major and most challenging part of the exercises. This task exposes students to the major problems in SoC design, namely arbitration to access shared resources, the crossing of time domains, and routing to a specific destination. While the drawing machine is a relatively simple design with these features, the techniques used to solve these are illustrative of those required in locally synchronous multiprocessor multibus systems.

So far, the shape chosen by students is circle drawing, as this is a computation-friendly algorithm requiring no multiplication hardware. Testing that the expanded system functions at each level as expected plays an important part in their learning process. Equally important is their ability to analyze the results if incorrect, diagnose the problem, and rectify the design. Thus, diagnostic skills as well as design techniques are enhanced.

Algorithmic Exploration

Since students are given the system specification, the design starts at the algorithmic level. Here, the system is a top-level view of what the user wants the system to do without any implication of how this might be implemented. Any high-level programming language could be chosen to describe the system, but C++ [18] is adopted because, at the architectural design levels which follow, the system description language used is implemented as a set of C++ library routines.

Fig. 2 shows the logical partitioning of the system into software blocks performing distinct behavioral functions, with the environment to the left of the dotted line and the drawing machine to the right. The command interpreter reads commands from the input stream. Point and line drawing requests are checked for the correct number of parameters before being passed to the drawing engine code for execution; in the case of a line, this engine computes the coordinates of the next pixel to be lit on the line and then writes the specified color to that coordinate in the frame store (implemented as a two dimensional array). The frame store contents are displayed on a CPU monitor in order to give the students direct feedback of the memory contents.

Other requests to the command interpreter cause actions within the environment designed to assist the students. Thus, the "clear" instruction sets the frame store to all-black, the "sleep" instruction inserts a delay enabling screen effects that change to be observed, while the "dump" order writes nonblack entries in the frame store to a text file enabling students to have the opportunity to verify formally the correctness of their computational code.

In the first exercise, students create a separate code block to draw their particular shape. Thus, they need to determine the format of the order and also amend the command interpreter to handle the request. A key part of this exercise is to explore

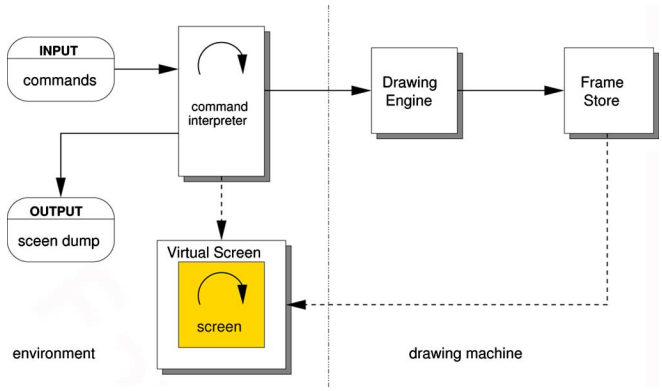

Fig. 2. Algorithmic model of the drawing machine.

different algorithms for the chosen shape, with the emphasis on it being computation-friendly since the computations performed by the algorithm will be reflected in the final hardware implementation. Thus, operations such as multiplication and division are to be avoided since they involve significant hardware, and such operations can usually be performed using the simpler, smaller hardware of addition or subtraction combined with shifting.

A. Drawing Machine Transaction-Level Model

Architectural design follows, and here the system is partitioned into computational blocks with interconnecting communication channels; these channels can be simple connections such as those used in buses or can be modules of any complexity, such as buffering data and performing data manipulation or transformations. Formally separating out the computation from communication allows each to be modeled separately, enabling extremely large and complex designs to be partitioned between different teams, enhancing the chances of correct operation when the different sections are merged. A key step in this process is the formal specification of the interfaces between modules. Another significant advantage of this approach is that software development can commence in parallel with hardware development; previously when the interconnections weren't formally modeled, software development was unable to proceed significantly until the hardware was defined at the RTL.

The basic TLM for the testbench and drawing machine given to students is shown by the solid lines in Fig. 3. The computational blocks for the drawing machine, to the right of the vertical dotted line, comprise distinct functions with the drawing engine and frame store code evolving directly from the algorithmic-level description. Located also on the drawing machine side, the inquisitor, in response to a request from the testbench, reads a byte from the frame store at a requested address and returns the pixel color to the testbench. The cathode ray tube (CRT) controller is an autonomous unit that reads the frame store sequentially and displays the colors read on a (virtual) screen. The inquisitor provides a verification tool for use on implemented hardware, which of course has no software dump facility.

As well as the computational blocks, the drawing machine has two interconnection channels. The command channel routes requests from the testbench to the correct computational block.

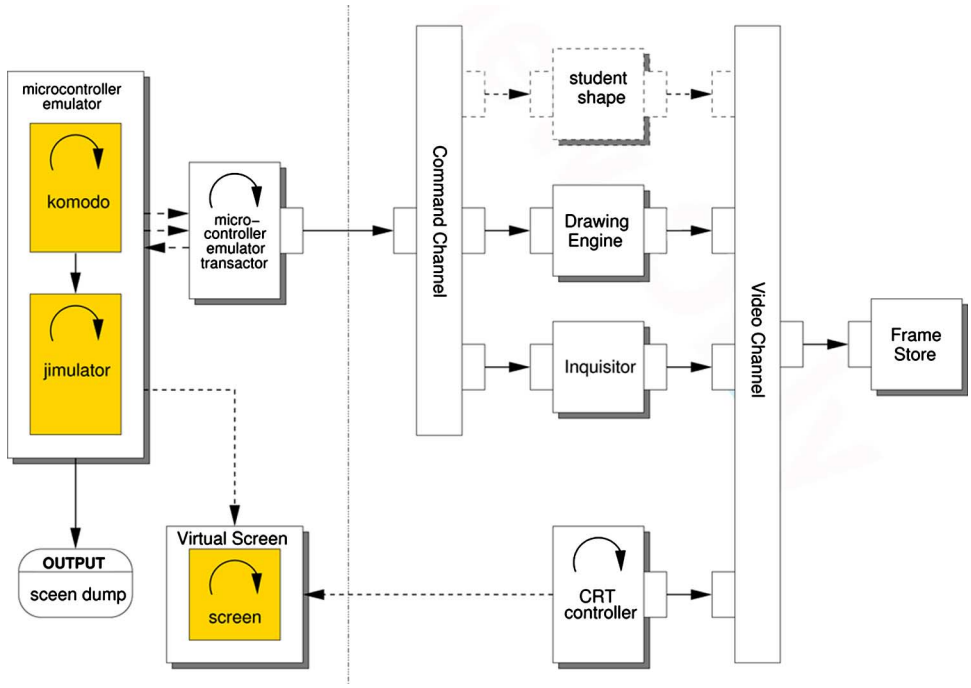

Fig. 3. Transaction-level model.

The drawing engine, inquisitor, and CRT controller can all compete to make read or write requests to the frame store, and the video channel arbitrates to determine which one of these master requests is forwarded to the frame store. The frame store is a slave unit as it only ever responds to requests from a master.

An important consideration in developing the testbench from the TLM downward was to provide a uniform environment for the students. This uniform environment would mean that any software written to test their computational block need only be developed once and could then be reused at each level. The software environment provided is similar to that used on the provided hardware, both of which were previously developed for use in a Microcontrollers course [19]. The Jimulator, an in-house emulator written in C, mirrors the operation of an advanced RISC machine (ARM) microprocessor, taking executable ARM instructions from an input program and performing them. The emulator operation is (automatically) linked to, and displayed on, an in-house graphical debugger named Komodo.

ARM instructions that store data to, or load data from, particular memory addresses communicate with the microcontroller emulator transactor. A transactor converts from TLM transactions to RTL signals or vice versa. In the drawing machine, the microcontroller emulator transactor translates the RTL signals from the microcontroller emulator into the transactions required by the TLM of the command channel. The microcontroller emulator also has access to the shared memory used by the screen and is also able to dump the screen nonblack content to a text file.

SystemC [11], [20], [21] consists of a set of library functions and a simulation kernel. All of the drawing machine modules are written in this system description language. Information transfers between modules as a set of transactions, and the code for the model conforms to the three-layer standard comprising application, protocol, and transport layers. This is shown in Fig. 4. At the top application layer, the code corresponds to the functional operations performed by the master and slave computational blocks. Masters produce read and write requests. Their initiator code transforms this into a get, put, or transport (which is get followed by put) transaction that passes to the transport layer.

The transport layer implements the channels. If several masters that can simultaneously request transfers are connected to this channel, then arbitration is required to select just one for transmission. If the channel can pass transactions to many computational blocks, then the channel has to route the get, put, or transport transaction to the specified recipient. The target code in the slave's protocol layer transforms the transactions into appropriate read or write requests, and the functional slave code at the application layer contains the methods to perform these requests.

B. Untimed Transaction-Level Model

Many of the computational blocks used in SoC design are acquired from firms as IP, and therefore, it is usual to only need to design the channels and any custom computational blocks required. Acquired computational blocks are usually supplied with encrypted descriptions in the system design language SystemC and in the (lower-level) hardware description language Verilog. These blocks cannot be modified in any way, and to reflect this real life situation, students are told that they are only allowed to modify the channels of the drawing engine and that all supplied computational blocks at whatever level must be left as is. However, to aid with teaching and learning, the code for all modules is made visible and available to the students.

SystemC models can be either timed or untimed. Simulation of a model comprising untimed computational blocks and channels enables an exploration of different architectures of the hard-

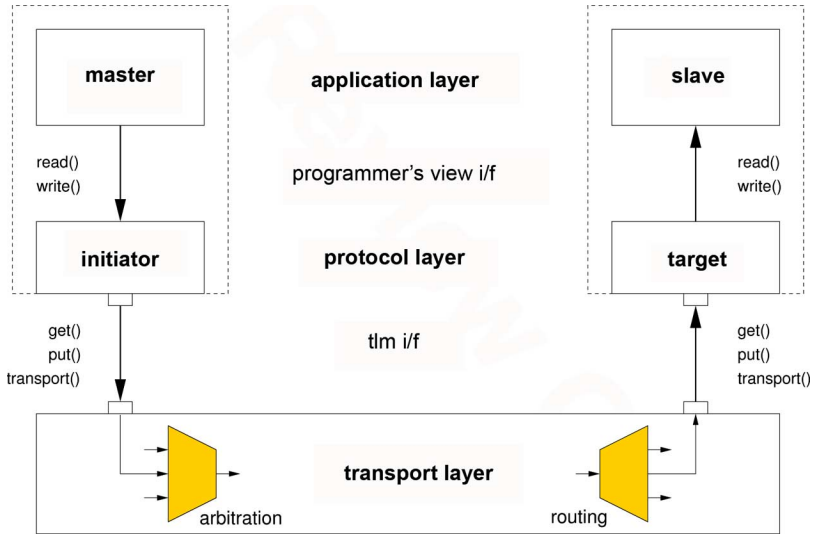

Fig. 4. Three-layer model for transaction level.

ware–software divide, while simulation with timed modules enables a rough estimate of the system performance to be gained. The former model yields a system close to the algorithmic description, while the latter makes it easier to translate to the hardware description of the RTL level. Usually, just one SystemC model is supplied. However, the use of a single model leads to a conceptually large gap for students to bridge either from the algorithmic level above or to the RTL below. For this reason, the design exercises start with an untimed TLM and then move to a timed TLM.

In both cases, students are given a complete, working basic system and are asked to design and add a custom block in SystemC based on their selected algorithm for their chosen shape. This master computational block is placed in parallel with the drawing engine and inquisitor blocks and is indicated by the dotted blocks in Fig. 3. In both TLM models, the insertion of this new computational block leads to students having to make significant modifications to the command and video channels in order to integrate it into the system.

In the untimed TLM, no timing is associated with any of the modules, and operations are considered to occur in zero time. Thus, the command channel only needs to detect and route draw requests to the new shape block. The video channel services requests from the master blocks connected to it. The servicing of these requests occurs in the order in which they arrive since the simulation runs on a single processor imposing an ordering of arrival in practice. Thus, the video channel in the untimed model need only be modified to accommodate write requests from the new block to the frame store.

Having integrated their block into the basic untimed TLM model, students are expected to expand on the testbench code running in the emulator so as to test their shape thoroughly and confirm that modifications to the system have been successful. Again, a screen allows for a rapid informal assessment as to whether the model and test code are correct, while the dump facility is also included to allow for formal verification of correctness.

C. Timed Transaction-Level Model

In moving to a timed TLM, the students need to convert their new drawing shape model to a timed block, make appropriate changes to the command and video channels, test the complete system and verify correct operation. The test programs developed for the untimed TLM in ARM assembly language code can again be used so students can concentrate on their drawing shape and integration code.

Timing their drawing block involves students in working through the logical time progression of operations in order to compute which locations in the frame store to draw and allocating states to this sequence. States are updated if required on the positive clock edge, and although operations are allocated to a state, a state may occupy several clock periods—for example, when reading from or writing to the frame store. All these considerations give students experience of the real problems encountered in design while being contained within the framework of a relatively confined and manageable design example. In addition, students do have the exemplar of the line drawing code in the drawing engine block to assist them in their block design and channel modifications.

The timed TLM also illustrates important problems in obtaining correct SoC designs, namely crossing clock domains and the need to arbitrate when timed masters simultaneously compete for the same resource. In the drawing system, the environment to the left of the dotted line in Fig. 3 operates from one clock, and the modules to the right all operate from a different clock. These clocks are independent of one another, and thus, data can change at any time on one side with respect to the clock on the other side. In the model, the synchronization of such data to the new clock domain is achieved by checking for such data on the positive clock edge in the new domain.

If no request is currently in progress, the video channel inspects for incoming master requests on the positive (drawing machine) clock edge and selects one for forwarding to the frame store. Arbitration between masters in the video channel allows

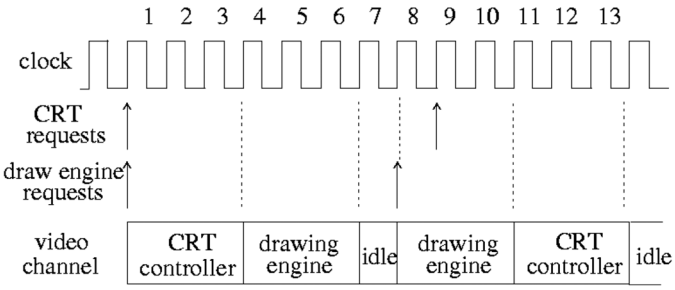

Fig. 5. Timing in the video channel.

students to consider different methods of achieving this with realistic timings, since the clock rate on the targeted hardware is known $(20\times10^{-9}\ \mathrm{s} = 20\ \mathrm{ns})$ as is the time to read from or write to the frame store (55 ns). Thus, the store access time means that once the video channel grants a frame store request, then each request granted occupies the video channel for three clock slots.

A further complication is that the autonomous CRT controller sends the video channel a request every eight clock cycles to read four sequential pixels from the frame store. The controller needs the data at this rate in order to maintain a continuous image on the screen. Since the CRT controller is unable to wait for its data, it is given top priority in the video channel for the servicing of its requests. Because the CRT controller occupies three out of every eight time slots, this leaves an average of five out of every eight of the available time slots for the drawing engine, new draw shape, and inquisitor requests to the frame store.

Fig. 5 illustrates the video channel timing when the drawing engine is also making requests. When the CRT controller and drawing engine make their first request simultaneously, the CRT controller request is granted first and when complete the drawing engine request is granted. This request completes at the end of the sixth clock cycle, and the drawing engine is able to enter its next request to the video channel for the start of the eighth clock cycle. This new request is granted since the CRT controller won't present its next request until the start of the next clock cycle. The CRT is now queued waiting for the release of the video channel, and so, will not be granted access until clock eleven.

In the basic system, the code for the video channel arbitration assigns a fixed priority to the masters connected to it, with the highest priority request present selected. The CRT controller is assigned top priority with the drawing engine next, and the inquisitor has bottom priority. In adding their new drawing shape, students can assign it a fixed priority or are free to consider other allocation strategies such as a dynamic priority scheme allowing masters other than the CRT controller a fair share of the remaining clock slots.

D. RTL Design

The SystemC description is a user's view of the operations of each module and the transactions that take place between them. While the modules may imply much of the internal hardware required, the description is not sufficiently detailed to generate hardware automatically using CAD software tools. Thus, the next stage in the design process is to specify the internal hardware of each module, as a behavioral model, using a hardware description language that can be synthesized by software tools to provide a logic implementation. The Verilog hardware description language [5], [16], [17] is chosen since this is the most commonly used in industry, and hence, the software CAD tools are designed to work with this language, including the processing tools mapping a logic design onto a hardware implementation. The Verilog simulation of the RTL design runs under the Cadence CAD environment.

Again, students are given a complete basic system as shown in Fig. 6. As can be seen, the environment is exactly the same as before, except that the transactor to translate from signals to transactions is no longer needed and has been replaced by an emulator external timing unit to add timing information to the emulator output; this unit is not required on the actual implementation since this output uses the timing on the provided hardware.

An RTL design requires the precise definition of signals between modules, and a protocol is needed to indicate to a module when its input lines are valid. Often, a source is required to hold the request on its output constant until the receiver signals that the request has been accepted, indicating that the source can remove the request. This simple handshake protocol has been adopted in the RTL design and replaces the request–response protocol used in the TLMs.

Handshaking is illustrated in Fig. 7. The request line from the source going high indicates a valid request on the output lines, while the acknowledge going high from the receiver frees the source and causes it to lower its request line. The receiver then lowers its acknowledge line in response to the lowering of the request signal. This protocol is essentially asynchronous, although actions in blocks following the receipt of a request or acknowledge signal are normally not initiated until the positive edge of the next clock cycle.

Again, the task for students is to convert their draw shape to an RTL model, to integrate this into the system making appropriate changes to the RTL channel descriptions, and to simulate the system with the previously generated test programs so as to observe correct operation on the screen.

Translation to RTL is the most difficult step in the whole process, requiring the movement from an abstract model to a real description of the hardware required. This stage is thus specifically supported in lectures that examine in detail the translation of the timed TLM of the basic system into its RTL representation. For the computational blocks, much of the functional behavior in the TLM can be directly ported into Verilog.

Allocating states to the RTL model is far more complex. This allocation requires consideration of the hardware and timing required since allocating too many parallel or sequential operations to a state is usually not viable in practice. For example, allocating two additions to a state will result in an implementation of two adders. If all other states only require a block to have a single adder, then using two adders for just one state is an inefficient use of resources. However, the use of just a single adder would result in an extra state, thus affecting throughput. Similarly, a sequence of actions in a TLM state may extend over a clock cycle in reality requiring partitioning into more than one

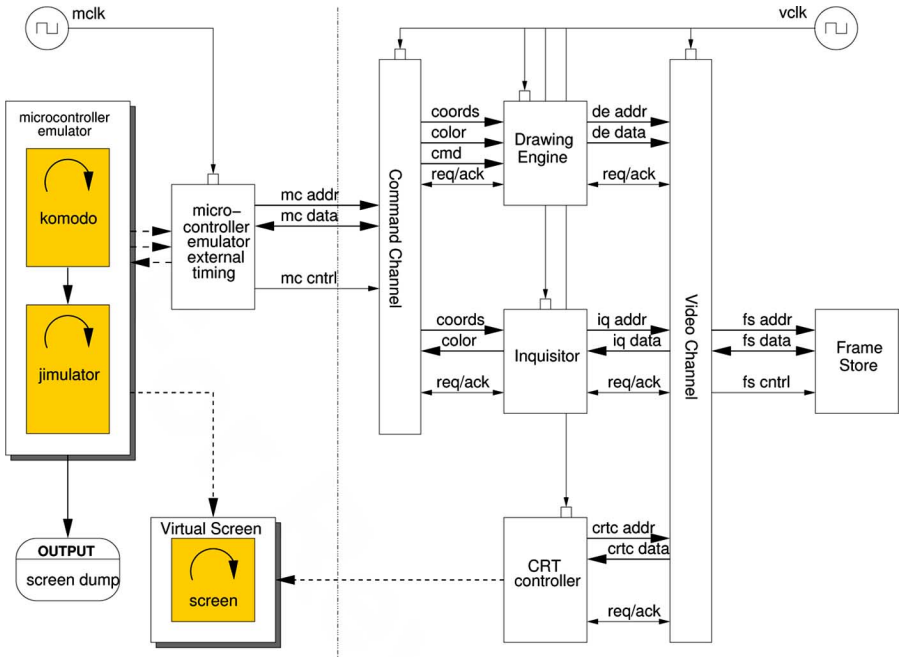

Fig. 6. Register transfer-level model.

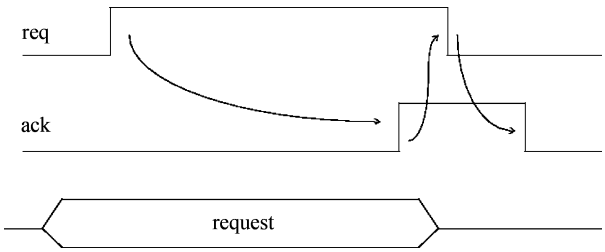

Fig. 7. Communication using handshaking.

state at RTL. For these reasons, the number of TLM states used normally expands at RTL and in the drawing engine; for example, four TLM states expand to nine at RTL.

Students are encouraged to take the states developed for their drawing shape in the timed TLM as a starting point for their RTL design. A major part of the task at this level is the need for students to consider resource implications for the first time in the design process. This exploration of the speed and area tradeoffs is again typical of the design decisions that SoC designers face in implementing their designs.

In integrating their drawing shape into the system, again modifications are needed to the command and video channels. In the former, code must be included to deal properly with the synchronization required for moving from one clock regime to another. As previously explained, data changes from one regime occur independently of the timing of the clock in the other regime. While simulators can synchronize by inspecting the data state on the new regime clock edge, real hardware may go into "metastability"— a halfway state between "0" and "1"—if the data change and clock edge are near simultaneous, and this state can last an indeterminate time and have a nondeterministic outcome (i.e., either "0" or "1"). Normally, hardware in this halfway state will settle to a valid binary "0"or "1" level if given an additional time to settle. For this reason, the logic is given an additional clock period before using the data from the sending regime, and this is coded into the command channel.

Transfers across the video channel are performed on a priority basis as before, with different states of the channel used to effect transfers across it. States are updated on each clock, and because the frame store access is so much greater than the clock period, many states do little more than effectively provide a delay. However, extra delays can easily be inserted by just adding extra states. An extra delay proved to be necessary because the frame store access was greater than 60 ns on the provided hardware due to the additional delay introduced by the wiring between the chips, which required allowing four clock cycles for the frame store access.

E. Hardware Prototyping

Following a successful simulation of the behavioral RTL model, the design is ready for hardware prototyping, which can be viewed as another stage in the verification process. As time constraints dictate that the full application-specific integrated circuit (ASIC) route is not feasible, the student design is mapped onto an FPGA. This route is also used nowadays by a number of SoC designers to verify their designs prior to commitment onto silicon.

The in-house designed board of hardware available to students [19] is shown in Fig. 8. The microcontroller comprises an ARM processor (with its own memory) and an interface to a host workstation. The microcontroller emulator for the drawing machine system is implemented on the board's processor. The frame store is implemented on the board's random access memory (RAM). The RAM and the microcontroller connect to an FPGA. This general purpose hardware block is configured as specified by the user to implement specific hardware functions; the FPGA used is able to provide around 200 000 logic gates. In

TABLE II
STUDENT ASSESSMENT

Question	2006-2007	2007-2008
The teaching I received was excellent	82%	100%
The material studied was intellectually stimulating	86%	92%
The skills I developed are valuable	90%	92%
The feedback I received was helpful	75%	83%
Teaching and support staff were approachable	93%	100%
Facilities needed for my work were available	82%	83%
The unit content was very relevant to my program	72%	100%
The unit was well integrated with other units on my program	50%	75%
The unit was not unnecessarily difficult	36%	92%
The unit overall was very interesting	90%	92%

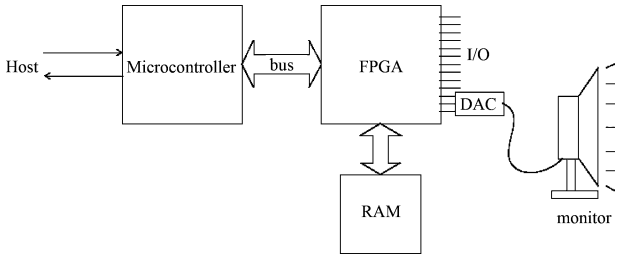

Fig. 8. Experimental board.

the design example, it is used to implement the modules of the drawing machine, i.e., the command channel, drawing engine, inquisitor, video channel, CRT controller, and the student's draw shape. The monitor shown is driven by the FPGA via a digital-to-analogue converter (DAC).

When the download of the design from the host onto the FPGA is complete, the test programs are downloaded into the ARM processor's memory and then run on the emulator. Having been successfully simulated, the student's design should run successfully and demonstrate the benefits of modeling hardware prior to implementing it. However, if there are errors and on-board debugging is desired, then the inquisitor has to be activated to return frame store values to the ARM emulator. Since this is likely to yield only limited information, a better approach is to return to the RTL simulation endeavoring to recreate the fault conditions. This process would be followed by resynthesis, download, and retesting on the board, and it demonstrates to students the advantages of implementing prototype hardware on a reconfigurable FPGA.

VI. ASSESSMENT

A. Learning Outcomes

Assessment of students is based on continual assessment during the time the design runs and on a 2-h examination at the end. Since the emphasis in the course is to learn by "doing," the practical work forms the larger part of the assessment. Students are expected to demonstrate their practical work and understanding in the laboratory sessions as well as to submit short professional reports. The evaluation is more heavily weighted toward the demonstration of the practical work.

The examinations are designed to demonstrate those aspects of the learning outcomes covering knowledge, understanding, and intellectual skills. The practical work demonstrates these as well, but also tests the development of a student's practical skills. The transferable skills involving written and spoken communication are mainly practiced through the written reports and laboratory demonstrations.

In meeting the learning outcomes, all students successfully completed the algorithmic and TLM stages, while most went on to complete the RTL design and subsequent implementation. The exam marks and written student feedback confirmed that a large majority of the students had knowledge and understanding of the concepts covered in the lectures and in the practical work.

B. Student Assessment

Official student feedback is measured via a course experience questionnaire. These are distributed toward the end of the course, and students are asked a variety of questions to which they give an integer score ranging from $+2$ indicating strong agreement to -2 indicating total disagreement. Scores are then averaged to give mean figures. In Table II, which gives the assessment over the academic years 2006–2007 and 2007–2008, average scores have been linearly converted to percentages.

The low score for integration reflects the fact that most students undertaking the SoC course are straight Computer Science students, and therefore, this module does not integrate well with their other software courses. Clearly, the students in 2006–2007 found the course difficult, and further informal feedback has enabled some problem areas in moving from the TLM to RTL levels to be successfully addressed in 2007–2008.

VII. CONCLUSION

SoC is a highly challenging topic for including in a degree course. It includes many difficult concepts and hardware languages that are unfamiliar to students and, thus, represents a large, steep learning curve. Nevertheless, the course described does illustrate that given a strictly bounded design, the major problems in SoC design can be demonstrated, with students gaining the techniques necessary to solve these. These design skills, together with a practical knowledge of industry-standard tools, are directly transferable to firms working in this area.

However, probably the best advert for the course is the sense of achievement and enthusiasm that students get from taking

their design from conception down to implementation. As well as a great sense of satisfaction from obtaining working hardware, there is also the perception that designing hardware is fun.

Finally, the authors are also very enthusiastic about the practical work. Thus, the authors would be happy to enable other academic institutions to have access to the code, and requests for this should be directed to the lead author.

ACKNOWLEDGMENT

A project of this size is a large undertaking. Thus, the authors are particularly grateful for the assistance of Dr. J. Garside, Dr. C. Brej, and Mr. D. Clark and for discussions and suggestions from other colleagues.

REFERENCES

[1] A. Bindal, S. Mann, B. N. Ahmed, and L. A. Raimundo, "An undergraduate system-on-chip (SoC) course for computer engineering students," *IEEE Trans. Edu.*, vol. 48, no. 2, pp. 279–289, May 2005.

[2] *"ECE 527 System-on-Chip Design"* Dept. Elect. and Comput. Eng., Univ. Illinois at Urbana-Champaign, Jan. 27, 2009 [Online]. Available: http://courses.ece.uiuc.edu/ece527/

[3] R. D. Walstrom, J. Schneider, and D. T. Rover, "Teaching system-level design using SpecC and SystemC," in *Proc. 2005 IEEE Int. Conf. MSE*, 2005, pp. 95–96.

[4] *"ASIC Design Flow"* Jan. 27, 2009 [Online]. Available: http://vlsi-homepage.com/category/asic-design-flow

[5] T. R. Padmanabhan and B. B. T. Sundari, *Design Through Verilog HDL.* New York: Wiley-IEEE Press, 2003.

[6] L. Cai and D. Gajski, "Transaction level modelling: An overview," in *Proc. 1st IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign and Syst. Synthesis*, Oct. 2003, pp. 19–24.

[7] A. Donlin, "Transaction level modeling: Flows and use models," in *Proc. 2nd IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign and Syst. Synthesis*, Sep. 2004, pp. 75–80.

[8] M. Radetzki, "Object-oriented transaction level modelling," in *Advances in Design and Specification Languages for Embedded Systems*, S. Huss, Ed. New York: Springer, 2007, ch. 10.

[9] S. Sutherland, S. Davidman, and P. Flake, *SystemVerilog for Design*. New York: Springer, 2006.

[10] B. Cohen, S. Venkataramanan, and A. Kumari, *SystemVerilog Assertions Handbook*. Palos Verdes Peninsula, CA: VhdlCohen, 2005.

[11] *"SystemC Golden Reference Guide,"* 2nd ed. Doulos, 2006.

[12] SystemC, Jan. 27, 2009 [Online]. Available: http://www.systemc.org

[13] F. Vahid, *Digital Design*. Hoboken, NJ: Wiley, 2006.

[14] P. J. Ashenden, *The Designer's Guide to VHDL.* San Mateo, CA: Morgan Kaufmann, 1995.

[15] V. A. Pedroni, *Circuit Design With VHDL.* Cambridge, MA: MIT Press, 2004.

[16] M. D. Ciletti, *Advanced Digital Design With Verilog HDL.* Upper Saddle River, NJ: Pearson, 2002.

[17] J. Bhasker, *A Verilog HDL Primer*, 2nd ed. Allentown, PA: Star Galaxy Press, 1999.

[18] S. Oualline, *Practical C++ Programming*, 2nd ed. Sebastopol, CA: O'Reilly, 2006.

[19] J. D. Garside, "A microcomputer interfacing laboratory," *Int. J. Elect. Eng. Educ.*, vol. 40, no. 1, pp. 13–26, 2003.

[20] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design With SystemC.* New York: Springer, 2002.

[21] *Transaction-Level Modeling With SystemC: TLM Concepts and Applications for Embedded Systems*, F. Ghenassia, Ed. New York: Springer, 2005.

Linda E. M. Brackenbury received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Manchester, Manchester, U.K.

She has been on the Computer Science Lecturing Staff at the University of Manchester since the late 1960s. She got her first experience of hardware design working on the large asynchronous MU5 machine designed to run high-level programming languages efficiently. Her involvement and interest in chip design began in the mid-1980s. As well as writing a student text in this area, she has been involved with many chip designs that have resulted in silicon implementations using both CMOS and bipolar technology. Her current research interests are in low-power design, particularly for battery-powered applications, and shortest path routing.

Luis A. Plana (M'97–SM'07) received the Ingeniero Electrónico degree from Universidad Simón Bolívar, Caracas, Venezuela, in 1978; the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1984; and the Ph.D. degree in computer science from Columbia University, New York, NY, in 1998.

He is a Research Fellow in the School of Computer Science at the University of Manchester. Before coming to Manchester, he worked at Universidad Politécnica, Venezuela, for over 20 years, where he was Professor and Head of the Department of Electronic Engineering. His research interests include the design and synthesis of asynchronous, embedded, and globally asynchronous, locally synchronous (GALS) systems.

Jeffrey Pepper received the diploma in Electronic Engineering and Computer Technology and the B.Sc. (Hons.) degree in engineering from Open University, Milton Keynes, U.K.

He joined the Technical Staff of the School of Computer Science at the University of Manchester, Manchester, U.K. in the mid-1980s. He is now a Senior Experimental Officer in the School of Computer Science specializing in CAD tools, chip design methodologies, and hardware description languages.