Modern System-on-Chip Design on Arm

TEXTBOOK

David J. Greaves
Modern System-on-Chip Design on Arm

DAVID J. GREAVES
Contents

Preface xvii
Acknowledgments xix
Author Biography xxi
List of Figures xxiii
List of Tables xxxvii

1. Introduction to System-on-Chip
   1.1 What is a System-on-Chip? 2
     1.1.1 Historical Review 2
     1.1.2 Simple Microprocessor Net-level Connections 4
     1.1.3 Full Netlist and Memory Map for a Microcomputer 5
     1.1.4 Separate Read and Write Data Busses 7
   1.2 Microcontrollers 8
   1.3 Later Chapters 10
   1.4 SoC Design Flows 11
     1.4.1 Functional Model 11
     1.4.2 Architectural Partition 14
     1.4.3 Architectural Partition and Co-design 15
     1.4.4 IP Blocks 16
     1.4.5 Synthesis 17
     1.4.6 Simulation 17
     1.4.7 Back-end Flow 18
     1.4.8 Example: A Cell Phone 20
     1.4.9 SoC Example: Helium 210 21
   1.5 SoC Technology 24
   1.6 Summary 26
     1.6.1 Exercises 26
# 2. Processors, Memory and IP Blocks

## 2.1 Processor Cores
- 2.1.1 ISAs  
- 2.1.2 Vector Instructions  
- 2.1.3 Custom Instructions  
- 2.1.4 The Classic Five-stage Pipeline  

## 2.2 Super-scalar Processors
- 2.2.1 Virtual Memory Management Units: MMU and IOMMU  

## 2.3 Multi-core Processing
- 2.3.1 Simultaneous Multithreading  

## 2.4 Cache Design
- 2.4.1 Snooping and Other Coherency Protocols  

## 2.5 Interrupts and the Interrupt Controller
- 2.5.1 Interrupt Structure Within a Device  

## 2.6 Memory Technology
- 2.6.1 Logical and Physical Layouts  
- 2.6.2 Mask-programmed ROM  
- 2.6.3 Static Random Access Memory  
- 2.6.4 Synchronous Static RAM  
- 2.6.5 Dual-ported Static RAM  
- 2.6.6 Dynamic RAM  
- 2.6.7 Electrically Alterable ROMs  
- 2.6.8 Floating-gate EA-ROMs and Flash  
- 2.6.9 Emerging Memory Technologies  
- 2.6.10 Processor Speed versus Memory Speed  

## 2.7 SoC I/O Blocks
- 2.7.1 Universal Asynchronous Receiver-Transmitter (UART)  
- 2.7.2 Parallel Ports Using General-purpose I/O  
- 2.7.3 General-purpose Input/Output Pins  
- 2.7.4 Counter/Timer Blocks
3. **SoC Interconnect**

3.1 **Interconnect Requirements**
- 3.1.1 Protocol Adaptors
- 3.1.2 On-chip Protocol Classes
- 3.1.3 Simple Bus Structures
- 3.1.4 Ordered and Unordered Interconnects
- 3.1.5 AMBA AXI Interconnect
- 3.1.6 Directory-based Coherence
- 3.1.7 Further Bus Operations

3.2 **Basic Interconnect Topologies**
- 3.2.1 Simple Bus with One Initiator
- 3.2.2 Shared Bus with Multiple Initiators
- 3.2.3 Bridged Bus Structures

3.3 **Simple Packet-Switched Interconnect**
- 3.3.1 Multi-access SoC and Inter-chip Interconnect
- 3.3.2 Multi-access Folded Bus

3.4 **Network-on-Chip**
- 3.4.1 NoC Routing and Switching
- 3.4.2 Virtual Channels
- 3.4.3 NoC Deadlocks
- 3.4.4 Credit-based Flow Control
- 3.4.5 AMBA 5 CHI
3.5 Advanced Interconnect Topologies 126
  3.5.1 Traffic Flow Matrix 127
3.6 Interconnect Building Blocks 133
3.7 Long-distance Interconnects 136
  3.7.1 Domain Crossing 136
  3.7.2 Metastability Theory 137
  3.7.3 CD-crossing Bridge 138
  3.7.4 Harmonic Clocks 139
  3.7.5 PD Crossing 141
3.8 Serialiser and Deserialiser: SERDES 142
  3.8.1 PCIe and SATA 144
  3.8.2 CCIX, CXL and NVLink 144
3.9 Automatic Topology Synthesis 145
  3.9.1 Domain Assignment 145
  3.9.2 FIFO Buffer Sizing 146
  3.9.3 Link Sizing 147
3.10 Summary 147
  3.10.1 Exercises 148

4. System Design Considerations 154
  4.1 Design Objectives and Metrics 154
  4.2 Parallel Speedup Theory 156
    4.2.1 Contention and Arbitration 157
  4.3 FIFO Queuing Theory and QoS 159
    4.3.1 Classical Single-server and Open Queue Models 160
    4.3.2 Expedited Service Queuing 163
    4.3.3 Statistical Multiplexing Gain 164
    4.3.4 QoS Policing 166
  4.4 Design Trade-offs 168
    4.4.1 Thermal Design 169
    4.4.2 Folding, Re-timing and Recoding 171
## Contents

4.5  **Design Trade-offs in Memory Systems**  
4.6  **SoC Energy Minimisation**  
   4.6.1  Power, Resistance and Capacitance  
   4.6.2  Dynamic Energy and Dynamic Power  
   4.6.3  Static Power Use  
   4.6.4  Wiring and Capacitance Modelling  
   4.6.5  Landauer Limit and Reversible Computation  
   4.6.6  Gate Delay as a Function of Supply Voltage  
   4.6.7  SPICE Simulation of an Inverter  
   4.6.8  Dynamic Voltage and Frequency Scaling  
   4.6.9  Dynamic Clock Gating  
   4.6.10  Dynamic Supply Gating  
   4.6.11  Future Trends for Energy Use  
4.7  **Designing for Testability and Debug Integration**  
   4.7.1  Application Debugging  
   4.7.2  Multi-core Debug Integration  
   4.7.3  Debug Navigation and JTAG  
   4.7.4  Additional DAP Facilities  
   4.7.5  Boundary and General Path Scans  
   4.7.6  BIST for SRAM Memories (MBIST)  
4.8  **Reliability and Security**  
   4.8.1  Physical Faults, Performance Degradation, Error Detection and Correction, and Pre- and Post-silicon Mitigation Techniques  
4.9  **Hardware-based Security**  
   4.9.1  Trusted Platform and Computer Modules  
   4.9.2  Trusted Execution Mode  
   4.9.3  Capability-based Protection  
   4.9.4  Clock Sources  
   4.9.5  PLL and Clock Trees  
   4.9.6  Clock Skewing and Multi-cycle Paths  
4.10  **Summary**  
   4.10.1  Exercises
# 5. Electronic System-Level Modelling

## 5.1 Modelling Abstractions
- 5.1.1 ESL Flow Diagram

## 5.2 Interconnect Modelling
- 5.2.1 Stochastic Interconnect Modelling
- 5.2.2 Cycle-accurate Interconnect Modelling

## 5.3 SystemC Modelling Library
- 5.3.1 SystemC Structural Netlist
- 5.3.2 SystemC Threads and Methods
- 5.3.3 SystemC Plotting and its GUI
- 5.3.4 Towards Greater Modelling Efficiency

## 5.4 Transaction-level Modelling
- 5.4.1 OSCI TLM 1.0 Standard
- 5.4.2 OSCI TLM 2.0 Standard
- 5.4.3 TLM Models with Timing (TLM+T)
- 5.4.4 TLM with Loosely Timed Modelling
- 5.4.5 Modelling Contention Under Loosely Timed TLM
- 5.4.6 Non-blocking TLM coding
- 5.4.7 Typical ISS Setup with Loose Timing and Temporal Decoupling
- 5.4.8 TLM Transactors for Bridging Modelling Styles
- 5.4.9 ESL Model of the LocalLink Protocol

## 5.5 Processor Modelling with Different Levels of Abstraction
- 5.5.1 Forms of ISS and Their Variants
- 5.5.2 Using the C Preprocessor to Adapt Firmware
- 5.5.3 ESL Cache Modelling and DMI

## 5.6 ESL Modelling of Power, Performance and Area
- 5.6.1 Estimating the Operating Frequency and Power with RTL
- 5.6.2 Typical Macroscopic Performance Equations: SRAM Example
- 5.6.3 Typical Macroscopic Performance Equations: DRAM Example
- 5.6.4 Macroscopic Phase and Mode Power Estimation Formula
### Contents

6. **Architectural Design Exploration**

6.1 **Hardware and Software Design Partition**

6.1.1 Design Partitioning Example: A Bluetooth Module

6.2 **Design Space Exploration**

6.2.1 DSE Workflows

6.2.2 C Functional Models

6.2.3 Functional Model Refactoring to ESL

6.2.4 Microarchitecture of a Subsystem

6.2.5 Interconnect Optimisation

6.3 **Hazards**

6.3.1 Hazards From Array Memories

6.3.2 Overcoming Structural Hazards using Holding Registers

6.3.3 Name Alias Hazards

6.3.4 FIFO Buffers and Dual-port IP Blocks

6.4 **Custom Accelerators**

6.4.1 Accelerator Communication

6.4.2 Multiple Sub-tasks

6.4.3 Custom Accelerator Example I

6.4.4 FPGA Acceleration in the Cloud

6.5 **Super FPGAs**

6.6 **Asymptotic Analysis**

6.6.1 Illustration 1: Hierarchical Cache

6.6.2 Illustration 2: big.LITTLE
## Contents

- **6.6.3 Illustration 3: NoC Topology Trade-off** 311
- **6.6.4 Illustration 4: Static and Dynamic Power Trade-off** 314

### 6.7 Virtual Platform Examples

- **6.7.1 The Prazor/Zynq Virtual Platform** 315
- **6.7.2 Co-design Worked Example: MPEG Video Compression** 317

### 6.8 Design-entry Languages

- **6.8.1 Functional Units** 323
- **6.8.2 Accellera IP-XACT** 326
- **6.8.3 Hardware Construction Languages** 331
- **6.8.4 Handel-C** 334
- **6.8.5 Bluespec System Verilog** 335

### 6.9 High-level Synthesis

- **6.9.1 Discovering Parallelism and Shared Variables in Iterations** 347

### 6.10 Summary

- **6.10.1 Exercises** 357

---

### 7. Formal Methods and Assertion-based Design

#### 7.1 Formal Languages and Tools

- **7.1.1 Verification Coverage** 367
- **7.1.2 Property Completeness** 369
- **7.1.3 When is a Formal Specification Complete?** 369

#### 7.2 Assertions

- **7.2.1 Predicate and Property Forms** 373
- **7.2.2 Assertion-based Design** 374
- **7.2.3 Regression Testing** 375

#### 7.3 Simulation with Assertions

- **7.3.1 Simulations and Dynamic Validation** 376
- **7.3.2 Automated Stimulus Generation: Directed and Constrained Random Verification** 376
- **7.3.3 Simulation versus Formal Checking** 378
<table>
<thead>
<tr>
<th>7.4</th>
<th>Property Specification Language</th>
<th>380</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.1</td>
<td>PSL Four-level Syntax Structure</td>
<td>382</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Extended Regular Expressions and SERES</td>
<td>383</td>
</tr>
<tr>
<td>7.4.3</td>
<td>System Verilog Assertions</td>
<td>384</td>
</tr>
<tr>
<td>7.5</td>
<td>Formal Interface Protocol Checkers</td>
<td>385</td>
</tr>
<tr>
<td>7.6</td>
<td>Equivalence Checking</td>
<td>389</td>
</tr>
<tr>
<td>7.6.1</td>
<td>Boolean Equivalence Checking</td>
<td>389</td>
</tr>
<tr>
<td>7.6.2</td>
<td>Sequential Equivalence Checking</td>
<td>390</td>
</tr>
<tr>
<td>7.6.3</td>
<td>X-propagation Checking</td>
<td>392</td>
</tr>
<tr>
<td>7.6.4</td>
<td>Model Checking the Items in a Data Path</td>
<td>396</td>
</tr>
<tr>
<td>7.7</td>
<td>Connectivity Checking</td>
<td>398</td>
</tr>
<tr>
<td>7.8</td>
<td>Checking the Security Policy</td>
<td>399</td>
</tr>
<tr>
<td>7.9</td>
<td>Summary</td>
<td>399</td>
</tr>
<tr>
<td>7.9.1</td>
<td>Exercises</td>
<td>401</td>
</tr>
</tbody>
</table>

| 8. | Fabrication and Production | 405 |
| 8.1 | Evolution of Design Closure | 407 |
| 8.1.1 | Physically Aware Design Flows | 407 |
| 8.2 | VLSI Geometry | 410 |
| 8.2.1 | VLSI Evolution | 412 |
| 8.2.2 | Typical and Future Values | 415 |
| 8.3 | Register Transfer Languages | 418 |
| 8.3.1 | RTL Structural Elaboration | 420 |
| 8.3.2 | Unsynthesisable RTL | 427 |
| 8.3.3 | RTL Simulation Algorithms | 429 |
| 8.3.4 | Event-driven Simulation | 431 |
| 8.3.5 | Inertial and Transport Delays | 431 |
| 8.3.6 | Compute/Commit Modelling and the Delta Cycle | 432 |
| 8.3.7 | Mixed Analogue and Digital Simulation | 433 |
Contents

8.3.8 Logic Synthesis 439
8.3.9 Arrays and RAM Inference in RTL 442
8.3.10 Memory Macrocell Compiler 443
8.3.11 Conventional RTL Compared with Software 444
8.3.12 Synthesis Intent and Goals 445

8.4 Chip Types and Classifications 448
  8.4.1 Semi-custom (Cell-based) Design 452
  8.4.2 Standard Cell Data 454
  8.4.3 SPICE Characterisation 455
  8.4.4 PVT Variations 456
  8.4.5 Electromigration 456
  8.4.6 Waveform-based Cell Characterisation 458
  8.4.7 Noise Characterisation 460

8.5 Gate Arrays 461
  8.5.1 Pass-transistor Multiplexers 462
  8.5.2 Field-programmable Gate Arrays 464
  8.5.3 Structured ASIC 467
  8.5.4 FPGA SoC Emulators 467

8.6 Floor and Power Planning 468
  8.6.1 Power Planning 468

8.7 Flow Steps 470
  8.7.1 Placement 470
  8.7.2 Clock Tree Insertion 473
  8.7.3 Routing 473
  8.7.4 Timing and Power Verification 474
  8.7.5 Post-route Optimisation 475
  8.7.6 Layout versus Schematic Check 475
  8.7.7 Sign-off and Tapeout 475
9. **Putting Everything Together**

9.1 **Firmware**

9.1.1 Secure Bootstrapping

9.2 **Powering up**

9.2.1 Engineering Sample Testing

9.3 **Success or Failure?**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Firmware</td>
<td>512</td>
</tr>
<tr>
<td>9.1.1 Secure Bootstrapping</td>
<td>513</td>
</tr>
<tr>
<td>9.2 Powering up</td>
<td>514</td>
</tr>
<tr>
<td>9.2.1 Engineering Sample Testing</td>
<td>514</td>
</tr>
<tr>
<td>9.3 Success or Failure?</td>
<td>517</td>
</tr>
</tbody>
</table>

Glossary of Abbreviations

Index

Index
Preface

Silicon technology has seen relentless advances in the past 50 years, driven by constant innovation and miniaturisation. As a result, more and more functionality has been placed into a single chip. Today, entire systems, including processors, memory, sensors and analogue circuitry, are integrated into one single chip (hence, a system-on-chip or SoC), delivering increased performance despite tight area, power and energy budgets. The aim of this textbook is to expose aspiring and practising SoC designers to the fundamentals and latest developments in SoC design and technologies. The processors within a SoC run a huge body of software. Much of this code is portable over many platforms, but low-level components, such as device drivers, are hardware-dependent and may be CPU-intensive. Power use can be reduced using custom accelerator hardware. Although this book emphasises the hardware design elements, it also addresses co-design, in which the hardware and software are designed hand in hand. It is assumed that the reader already understands the basics of processor architecture, computer technology, and software and hardware design.

Is This Book Suitable For You?
We assume that you have some experience with hardware design using an RTL such as Verilog or VHDL, and that you understand assembly language programming and basic principles of operating systems. In other words, you have completed the first two years of a degree in Computer Science or Electronic Engineering.

Many of the principles taught in this book are relevant for all forms of system architect, including those who are designing cloud-scale applications, custom accelerators or IoT devices in general, or those making FPGA designs. But the details of design verification in Chapter 8 are likely to be just of interest to those designing semi-custom silicon using standard cells.

A Git repository of online additional material is available at http://bitbucket.org/djg11/modern-soc-design-djg

This contains data used for generating tables and graphs in the book, as well as further source code, lab materials, examples and answers to selected exercises.

The repo contains a SystemC model of the Zynq super FPGA device family, coded in blocking TLM style. It is sufficient to run an Arm A9 Linux kernel using an identical boot image as the real silicon.

Book Structure
This book contains nine chapters, each devoted to a different aspect of SoC design.

Chapter 1 reviews basic computer architecture, defining terms that are used in later chapters. Readers are expected to be largely familiar with most of this material, although the transactional-level
modelling (TLM) view of the hardware is likely to be new. A SoC is an assembly of intellectual property (IP) blocks.

Chapter 2 describes many of the standard IP blocks that make up a typical SoC, including processors, memories, input/output devices and interrupts.

Chapter 3 considers the interconnect between the IP blocks, covering the evolution of processor busses and networks-on-chip (NoCs).

Chapter 4 teaches basic principles of system architecture, including dimensioning of busses and queuing theory and arbitration policies. It also discusses debug support.

Chapter 5 presents Electronic System Level (ESL) modelling, where a simulation model for a whole SoC, also known as a virtual platform, is put together. The ESL model is sufficient to test and develop software, as well as to perform architectural exploration, where the throughput, energy use and silicon area of a proposed system implementation can be examined at a high level.

Chapter 6 presents further architectural exploration considerations, including the design of custom accelerators for a specific application. The languages Bluespec and Chisel are described as alternatives to RTL for design entry and the basic principles of high-level synthesis (HLS) are covered.

Chapter 7 is a primer for formal verification of SoCs, comparing the usefulness of formal compared with simulation for bug hunting and right-first-time solutions. A number of useful formal tricks are covered.

Chapter 8 presents semi-custom fabrication flows for making the physical silicon and covers advanced verification and variability mitigation techniques for today’s deep sub-micron devices using FinFETs.

Chapter 9 covers what to do when the first SoC samples arrive back from the wafer processing plant, including booting an operating system and checking environmental compatibility (operating temperature and unwanted radio emissions).
Acknowledgements

I am very grateful to Professor Sir Andy Hopper, who was my PhD supervisor, who has been a constant source of inspiration and direction, and who has often been my boss both in industry and at the Computer Laboratory. He introduced me to the field of chip design. I am also very grateful to the late M. G. Scroggie, the principal author of ‘Foundations of Wireless’, which was a book I read and re-read all through my childhood. I can only hope some people find this current book as valuable as I found his. Certainly I have tried to mix breadth and depth in the same accessible way that he managed. I would like to thank those working in the Computer Laboratory who helped with this book, including David Chisnall, Robert Mullins, Omer Sella and Milos Puzovic. I would also like to thank my wife, Aldyth, for putting up with me for this last year. I’ve often read such comments in the acknowledgement sections of other books, but now I understand what causes it.

Most importantly, I’d like to thank the many Arm staff who have helped with this book, either by contributing text to large chunks of it, or with additional information and suggestions:

Khaled Benkrid, who made this book possible.

Liz Warman, who kept me on track and assisted me with the process.

Shuojin Hang and Francisca Tan who helped create the scope and reviewed early drafts.

This book would not have been possible without the collaboration of the following Arm engineers who have co-written, reviewed and commented on the book:

- Chapter 2: Processors, Memory and IP Blocks
  Rahul Mathur, Staff Engineer

- Chapter 3: SoC Interconnects
  Anup Gangwar, Distinguished Engineer
  Antony Harris, Senior Principal AMBA Architect

- Chapter 6: Architectural Design Exploration
  Edwin Dankert, Director, Technology Management

- Chapter 7: Formal Methods and Assertion-Based Design
  Daryl Stewart, Distinguished Engineer

- Chapter 8: Fabrication and Production
  Jim Dodrill, Fellow
  Christophe Lopez, Senior Principal Engineer
  Aurelien Merour, Principal Engineer
  Jean-Luc Pelloie, Fellow
Author Biography

Dr. David J. Greaves, PhD CEng. is Senior Lecturer in Computing Science at the University of Cambridge, UK and a Fellow of Corpus Christi College. Born in London, he grew up in a house full of engineering textbooks, circuit diagrams and pieces of telecommunications equipment. His grandfather had built his own television set as soon as television broadcasts started. His father worked at EMI and IBM, developing modems and computer interfaces. With the shift of head office of IBM UK to Portsmouth, the family moved to Romsey in Hampshire.

Plessey Roke Manor was also situated in Romsey, along with IBM's UK research laboratories at Hursley Park. These were, and remain, world-leading research centres in the field of radio communications and computing. The young David J. Greaves was a regular visitor and intern at both sites, and by the age of 17 had designed and built his first computer. The chips had been mostly removed from old circuit boards using a blow lamp. The software, including the disk operating system and a Pascal compiler, had all been written from scratch.

During his A-level studies, Greaves designed a local area network for Commodore PET computers. The design was published in Wireless World magazine and commercially replicated.

As an undergraduate at St John's College Cambridge, he offered consultancy services to various small electronics companies in the field of professional audio, developing MIDI interfaces and low-noise pre-amplifiers. His final-year degree project was a fully digital keyboard instrument that was serialised in Wireless World and copied by many enthusiasts worldwide. A main interest became the design and implementation of compilers, as encouraged by Dr. Martin Richards of St Johns, who had developed the BCPL language, the direct precursor of C.

Greaves designed his first silicon chips during his PhD studies, which were in the field of metropolitan area networks. He designed fibre optic transceivers that sent the first mono-mode signals over the newly installed fibres that criss-crossed Cambridge. In collaboration with Acorn Computer, in 1995 Greaves was the chief network architect for the Cambridge ITV trial, which put ATM switches in the kerbside cabinets belonging to Cambridge Cable Ltd and delivered video on demand to 50 or so homes. It was 20 years later that the last movie rental shop in Cambridge closed.

Also in 1995, he implemented CSYN, one of the first Verilog compilers for synthesising hardware specifically for field programmable gate arrays. This compiler was distributed widely among local companies on the Cambridge Science Park and also used for undergraduate teaching. It was licensed to a multinational to bundle with its own family of FPGAs.

Greaves had visited Arm when it first spun out of Acorn and consisted of six engineers in a barn. At the university, Greaves used a donation of Arm circuit boards for a new practical course in which the
students wrote assembly language and Verilog to learn about low-level hardware and software interfacing. These courses still run today and the lecture notes have evolved into this textbook.

Greaves has been on the board or technical advisory board of at least ten start-up companies. He has supervised or examined more than 60 PhD students. He holds at least five international patents in the field of communications and electronics. His company Tenison EDA was, before acquisition, directly providing tools to all major chip makers. His current research interests remain in the field of compilation tools for design automation and scientific acceleration.
Modern System-on-Chip Design on Arm

SoC design has seen significant advances in the decade and Arm-based silicon has often been at the heart of this revolution. Today, entire systems including processors, memories, sensors and analogue circuitry are all integrated into one single chip (hence “System-on-Chip” or SoC). The aim of this textbook is to expose aspiring and practising SoC designers to the fundamentals and latest developments in SoC design and technologies using examples of Arm® Cortex®-A technology and related IP blocks and interfaces. The entire SoC design process is discussed in detail, from memory and interconnects through to validation, fabrication and production. A particular highlight of this textbook is the focus on energy efficient SoC design, and the extensive supplementary materials which include a SystemC model of a Zynq chip.

This textbook is aimed at final year undergraduate students, master students or engineers in the field looking to update their knowledge. It is assumed that readers will have a pre-existing understanding of RTL, Assembly Language and Operating Systems. For those readers looking for a entry-level introduction to SoC design, we recommend our Fundamentals of System-on-Chip Design on Arm Cortex-M Microcontrollers textbook.

Table of contents
1 Introduction to System-on-Chip
2 Processors, Memory and IP Blocks
3 SoC Interconnect
4 System Design Considerations
5 Electronic System-level Modelling
6 Architectural Design Exploration
7 Formal Methods and Assertion-based Design
8 Fabrication and Production
9 Putting Everything Together

Dr. David J. Greaves, PhD CEng. is a Senior Lecturer in Computing Science at the University of Cambridge, UK and a Fellow of Corpus Christi College.

In 1995, Dr Greaves implemented CSYN, one of the first Verilog compilers for synthesising hardware specifically for field programmable gate arrays. This compiler was distributed widely among local companies on the Cambridge Science Park and also used for undergraduate teaching. It was licensed to a multinational to bundle with its own family of FPGAs.

Dr Greaves holds at least five international patents in the field of communications and electronics. His current research interests remain in the field of compilation tools for design automation and scientific acceleration.