DOWNLOAD PDF ARM SYSTEM-ON-CHIP ARCHITECTURE

Chapter 1 : ARM System-on-Chip Architecture - Wikipedia

ARM System-on-Chip Architecture is an essential handbook for system-on-chip designers using ARM processor cores and engineers working with the ARM. It can also be used as a course text for undergraduate and masters students of computer science, computer engineering and electrical engineering.

Preface Aims This book introduces the concepts and methodologies employed in designing a system-on-chip SoC based around a microprocessor core and in designing the microprocessor core itself. The principles of microprocessor design are made concrete by extensive illustrations based upon the ARM. The aim of the book is to assist the reader in understanding how SoCs and microprocessors are designed and used, and why a modern processor is designed the way that it is. The reader who wishes to know only the general principles should find that the ARM illustrations add substance to issues which can otherwise appear somewhat ethereal; the reader who wishes to understand the design of the ARM should find that the general principles illuminate the rationale for the ARM being as it is. Other microprocessor architectures are not described in this book. The reader who wishes to make a comparative study of architectures will find the required information on the ARM here but must look elsewhere for information on other designs. Audience The book is intended to be of use to two distinct groups of readers: Professional hardware and software engineers who are tasked with designing an SoC product which incorporates an ARM processor, or who are evaluating the ARM for a product, should find the book helpful in their duties. Although there is considerable overlap with ARM technical publications, this book provides a broader context with more background. Students of computer science, computer engineering and electrical engineering should find the material of value at several stages in their courses. Some chapters are closely based on course material previously used in undergraduate teaching; some other material is drawn from a postgraduate course. Prerequisite knowledge This book is not intended to be an introductory text on computer architecture or computer logic design. Readers are assumed to have a level of familiarity with these subjects equivalent to that of a second year undergraduate student in computer science or computer engineering. Some first year material is presented, but this is more by way of a refresher than as a first introduction to this material. No prior familiarity with the ARM processor is assumed. The highlights of the last decade of ARM development include: Most of the principles of modern SoC and processor design are illustrated somewhere in the ARM family, and ARM has led the way in the introduction of some concepts such as dynamically decompressing the instruction stream. The inherent simplicity of the basic 3-stage pipeline ARM core makes it a good pedagogical introductory example to real processor design, whereas the debugging of a system based around an ARM core deeply embedded into a complex system chip represents the cutting-edge of technological development today. Book structure Chapter 1 starts with a refresher on first year undergraduate processor design material. It illustrates the principle of abstraction in hardware design by reviewing the roles of logic and gate-level representations. It then introduces the important concept of the Reduced Instruction Set Computer RISC as background for what follows, and closes with some comments on design for low power. Chapter 2 describes the ARM processor architecture in terms of the concepts introduced in the previous chapter, and Chapter 3 is a gentle introduction to user-level assembly language programming and could be used in first year undergraduate teaching for this purpose. Chapter 4 describes the organization and implementation of the 3- and 5-stage pipeline ARM processor cores at a level suitable for second year undergraduate teaching, and covers some implementation issues. Chapters 5 and 6 go into the ARM instruction set architecture in increasing depth. Chapter 5 goes back over the instruction set in more detail than was presented in Chapter 3, including the binary representation of each instruction, and it penetrates more deeply into the corners of the instruction set. It is probably best read once and then used for reference. Chapter 6 backs off a bit to consider what a high-level language in this case, C really needs and how those needs are met by the ARM instruction set. This chapter is based on second year undergraduate material. It is of peripheral interest to a generic study of computer science, but adds an interesting lateral perspective to a postgraduate course. Chapter 8 raises the issues involved in debugging systems which use embedded processor cores and in the production testing of board-level systems. Chapter 10 introduces the concept of memory hierarchy, discussing the principles of memory management and caches. Chapter 11 reviews the requirements of a modern operating system at a second year undergraduate level and describes the approach adopted by the ARM to address these requirements. Chapter 13 covers the issues of designing SoCs with embedded processor cores. Here, the ARM is at the leading edge of technology. Several examples are presented of production embedded system chips to show the solutions that have been developed to the many problems inherent in committing a complex application-specific system to silicon. Chapter 14 moves away from mainstream ARM developments to describe the asynchronous ARM-compatible processors and systems developed at the University of Manchester, England, during the s. After a decade of research the AMULET technology is, at the time of writing, about to take its first step into the commercial domain. Chapter 14 concludes with a description of the DRACO SoC design, the first commercial application of a bit asynchronous microprocessor. A short appendix presents the fundamentals of computer logic design and the terminology which is used in Chapter 1. A glossary of the terms used in the book and a bibliography for further reading are appended at the end of the book, followed by a detailed index. Course relevance The chapters are at an appropriate level for use on undergraduate courses as follows: Chapter 1 basic processor design ; Chapter 3 assembly language programming ; Chapter 5 instruction binaries and reference for assembly language programming. Chapter 4 simple pipeline processor design ; Chapter 6 architectural support for high-level languages ; Chapters 10 and 11 memory hierarchy and architectural support for operating systems. Chapter 8 embedded system debug and test ; Chapter 9 advanced pipelined processor design ; Chapter 12 advanced CPUs ; Chapter 13 example embedded systems. A postgraduate course could follow a theme across several chapters, such as processor design Chapters 1, 2, 4, 9, 10 and 12 , instruction set design Chapters 2, 3, 5, 6, 7 and 11 or embedded systems Chapters 2, 4, 5, 8, 9 and Chapter 14 contains material relevant to a third year undergraduate or advanced postgraduate course on asynchronous design, but a great deal of additional background material not presented in this book is also necessary. Support material Many of the figures and tables will be made freely available over the Internet for non-commercial use. The only constraint on such use is that this book should be a recommended text for any course which makes use of such material. Information about this and other support material may be found on the World Wide Web at: The assertion of the copyright for this book outlined on page iv remains unaffected. Feedback The author welcomes feedback on the style and content of this book, and details of any errors that are found. Please email any such information to: As a policy decision I have not named in the text the individuals with principal responsibilities for the developments described therein since the lists would be long and attempts to abridge them invidious. History has a habit of focusing credit on one or two high-profile individuals, often at the expense of those who keep their heads down to get the job done on time. However, it is not possible to write a book on the ARM without mentioning Sophie Wilson whose original instruction set architecture survives, extended but otherwise largely unscathed, to this day. The book has been considerably enhanced by helpful comments from reviewers of draft versions. I am grateful for the sympathetic reception the drafts received and the direct suggestions for improvement that were returned. The publishers, Addison Wesley Longman Limited, have been very helpful in guiding my responses to these suggestions and in other aspects of authorship. Lastly I would like to thank my wife, Valerie, and my daughters, Alison and Catherine, who allowed me time off from family duties to write this book.

Chapter 2 : System on a chip - Wikipedia

System-on-chip technology is changing the way we use computers, The ARM is at the heart of this trend, leading the way in system-on-chip (SoC) development and becoming the processor core of choice for many embedded applications.

The state in a stored-program digital computer. The Storedprogram Computer The stored-program digital computer keeps its instructions and data in the same memory system, allowing the instructions to be treated as data when necessary. This enables the processor itself to generate instructions which it can subsequently execute. Although programs that do this at a fine granularity self-modifying code are generally considered bad form these days since they are very difficult to debug, use at a coarser granularity is fundamental to the way most computers operate. Whenever a computer loads in a new program from disk overwriting an old program and then executes it the computer is employing this ability to change its own program. Computer applications Because of its programmability a stored-program digital computer is universal, which means that it can undertake any task that can be described by a suitable algorithm. Sometimes this is reflected by its configuration as a desktop machine where the user runs different programs at different times, but sometimes it is reflected by the same processor being used in a range of different applications, each with a fixed program. Such applications are characteristically embedded into products such as mobile telephones, automotive engine-management systems, and so on. A modern microprocessor may be built from several million transistors each of which can switch a hundred million times a second. Watch a document scroll up the screen An Introduction to Processor Design 4 on a desktop PC or workstation and try to imagine how a hundred million million transistor switching actions are used in each second of that movement. Now consider that every one of those switching actions is, in some sense, the consequence of a deliberate design decision. None of them is random or uncontrolled; indeed, a single error amongst those transitions is likely to cause the machine to collapse into a useless state. How can such complex systems be designed to operate so reliably? Transistors A clue to the answer may be found in the question itself. We have described the operation of the computer in terms of transistors, but what is a transistor? It is a curious structure composed from carefully chosen chemical substances with complex electrical properties that can only be understood by reference to the theory of quantum mechanics, where strange subatomic particles sometimes behave like waves and can only be described in terms of probabilities. Yet the gross behaviour of a transistor can be described, without reference to quantum mechanics, as a set of equations that relate the voltages on its terminals to the current that flows though it. These equations abstract the essential behaviour of the device from its underlying physics. Logic gates The equations that describe the behaviour of a transistor are still fairly complex. If each of the input wires A and B is held at a voltage which is either near to Vdd or near to Vss, the output will will also be near to Vdd or Vss according to the following rules: The concepts that the logic designer works with are illustrated in Figure 1. This describes the logic function of the gate, and encompasses everything that the logic designer needs to know about the gate for most purposes. The significance here is that it is a lot simpler than four sets of transistor equations. The gate abstraction The point about the gate abstraction is that not only does it greatly simplify the process of designing circuits with great numbers of transistors, but it actually Figure 1. A logic circuit should have the same logical behaviour whether the gates are implemented using field-effect transistors the transistors that are available on a CMOS process , bipolar transistors, electrical relays, fluid logic or any other form of logic. The implementation technology will affect the performance of the circuit, but it should have no effect on its function. It is the duty of the transistor-level circuit designer to support the gate abstraction as near perfectly as is possible in order to isolate the logic circuit designer from the need to understand the transistor equations. Levels of abstraction It may appear that this point is being somewhat laboured, particularly to those readers who have worked with logic gates for many years. However, the principle that is illustrated in the gate level abstraction is repeated many times at different levels in computer science and is absolutely fundamental to the process which we began considering at the start of this section, which is the management of complexity. The process of gathering together a few components at one level to extract their essential joint behaviour and hide all the unnecessary detail at the next level enables us to scale orders of complexity in a few steps. For instance, if each level encompasses four components of the next lower level as our gate model does, we can get from a transistor to a microprocessor comprising a million transistors in just ten steps. In many cases we work with more than four components, so the number of steps is greatly reduced. A typical hierarchy of abstraction at the hardware level might be: The process of understanding a design in terms of levels of abstraction is reasonably concrete when the design is expressed in hardware. Gate-level design The next step up from the logic gate is to assemble a library of useful functions each composed of several gates. Typical functions are, as listed above, adders, multiplexers, decoders and flip-flops, each 1-bit wide. This book is not intended to be a gen- MU0 - a simple processor 7 eral introduction to logic design since its principal subject material relates to the design and use of processor cores and any reader who is considering applying this information should already be familiar with conventional logic design. It includes brief details on: If any of these terms is unfamiliar, a brief look at the appendix may yield sufficient information for what follows. Note that although the appendix describes these circuit functions in terms of simple logic gates, there are often more efficient CMOS implementations based on alternative transistor circuits. There are many ways to satisfy the basic requirements of logic design using the complementary transistors available on a CMOS chip, and new transistor circuits are published regularly. Such a design has been employed at the University of Manchester for many years to illustrate the principles of processor design. It is a design developed only for teaching and was not one of the large-scale machines built at the university as research vehicles, though it is similar to the very first Manchester machine and has been implemented in various forms by undergraduate students. The MU0 instruction set MU0 is a bit machine with a bit address space, so it can address up to 8 Kbytes of memory arranged as 4, individually addressable bit locations. Instructions are 16 bits long, with a 4-bit operation code or opcode and a bit address field S as shown in Figure 1. The simplest instruction set uses only eight of the 16 available opcodes and is summarized in Table 1. To understand how this instruction set might be implemented we will go through the design process in a logical order. The approach taken here will be to separate the design into two components: All the components carrying, storing or processing many bits in parallel will be considered part of the datapath, including the accumulator, program counter, ALU and instruction register. For these components we will use a register transfer level RTL design style based on registers, multiplexers, and so on. Everything that does not fit comfortably into the datapath will be considered part of the control logic and will be designed using a finite state machine FSM approach. Datapath design There are many ways to connect the basic components needed to implement the MU0 instruction set. Where there are choices to be made we need a guiding principle to help us make the right choices. Here we will follow the principle that the memory will be the limiting factor in our design, and a memory access will always take a clock cycle. Hence we will aim for an implementation where: Referring back to Table 1. In practice we would probably not worry about the efficiency of the STP instruction since it halts the processor for ever. Therefore we need a datapath design which has sufficient resource to allow these instructions to complete in two or one clock cycles. A suitable datapath is shown in Figure 1. Datapath operation The design we will develop assumes that each instruction starts when it has arrived in the instruction register. After all, until it is in the instruction register we cannot know which instruction we are dealing with. Therefore an instruction executes in two stages, possibly omitting the first of these: Access the memory operand and perform the desired operation. The address in the instruction register is issued and either an operand is read from memory, combined with the accumulator in the ALU and written back into the accumulator, or the accumulator is stored out to memory. Fetch the next instruction to be executed. Either the PC or the address in the instruction register is issued to fetch the next instruction, and in either case the address is incremented in the ALU and the incremented value saved into the PC. Initialization The processor must start in a known state. Usually this requires a reset input to cause it to start executing instructions from a known address. We will design MU0 to start executing from address There are several ways to achieve this, one of which is to use the reset signal to zero the ALU output and then clock this into the PC register. Register transfer level design The next step is to determine exactly the control signals that are required to cause the datapath to carry out the full set of operations. We assume that all the registers change state on the falling edge of the input clock, and where necessary have control signals that may be used to prevent them from changing

DOWNLOAD PDF ARM SYSTEM-ON-CHIP ARCHITECTURE

on a particular clock edge. A suitable register organization is shown in Figure 1. The other signals shown are outputs from the datapath to the control logic, including the opcode bits and signals indicating whether ACC is zero or negative which control the respective conditional jump instructions. Control logic The control logic simply has to decode the current instruction and generate the appropriate levels on the datapath control signals, using the control inputs from the datapath where necessary. Although the control logic is a finite state machine, and therefore in principle the design should start from a state transition diagram, in this case the FSM is trivial and the diagram not worth drawing. MU0 - a simple processor 11 The control logic can be presented in tabular form as shown in Table 1. Once the ALU function select codes have been assigned the table may be implemented directly as a PLA programmable logic array or translated into combinatorial logic and implemented using standard gates. A quick scrutiny of Table 1. The program counter and instruction register clock enables PCce and IRce are always the same. This makes sense, since whenever a new instruction is being fetched the ALU is computing the next program counter value, and this should be latched too. Therefore these control signals may be merged into one. Similarly, whenever the accumulator is driving the data bus ACCoe is high the memory should perform a write operation Rn W is low , so one of these signals can be generated from the other using an inverter. After these simplifications the control logic design is almost complete. It only remains to determine the encodings of the ALU functions. Similarly, whenever the accumulator is driving the data bus ACCoe is high the memory should perform a write operation RnW is low , so one of these signals can be generated from the other using an inverter. The ALU functions that are required are listed in Table 1. Therefore the reset signal can control this function directly and the control logic need only generate a 2-bit function select code to choose between the other four. If the principal ALU inputs are the A and B operands, all the functions may be produced by augmenting a conventional binary adder: Aen enables the A operand or forces it to zero; Binv controls whether or not the B operand is inverted. The carry-out Cout from one bit is connected to the carry-in Cin of the next; the carry-in to the first bit is controlled by the ALU function selects as are Aen and Binv , and the carry-out from the last bit is unused Together with the multiplexers, registers, control logic and a bus buffer which is used to put the accumulator value onto the data bus , the processor is complete. Add a standard memory and you have a workable computer. MU0 extensions Although MU0 is a very simple processor and would not make a good target for a high-level language compiler, it serves to illustrate the basic principles of processor design. The design process used to develop the first ARM processors differed mainly in complexity and not in principle. MU0 designs based on microcoded control logic have also been developed, as have extensions to incorporate indexed addressing. Like any good processor, MU0 has spaces left in the instruction space which allow future expansion of the instruction set. To turn MU0 into a useful processor takes quite a lot of work. The following extensions seem most important: An Introduction to Processor Design 14 1.

Chapter 3 : arm system on chip architecture | Download eBook pdf, epub, tuebl, mobi

ARM System-on-Chip Architecture is a book detailing the system-on-chip ARM architecture, as a specific implementation of reduced instruction set computing. It was written by Steve Furber, who co-designed the ARM processor with Sophie Wilson.

The design flow for an SoC aims to develop this hardware and software at the same time, also known as architectural co-design. Most SoCs are developed from pre-qualified hardware component IP core specifications for the hardware elements and execution units , collectively "blocks", described above, together with software device drivers that may control their operation. Of particular importance are the protocol stacks that drive industry-standard interfaces like USB. The hardware blocks are put together using computer-aided design tools, specifically electronic design automation tools; the software modules are integrated using a software integrated development environment. Once the architecture of the SoC has been defined, any new hardware elements are written in an abstract hardware description language termed register transfer level RTL which defines the circuit behavior, or synthesized into RTL from a high level language through high-level synthesis. These elements are connected together in a hardware description language to create the full SoC design. The logic specified to connect these components and convert between possibly different interfaces provided by different vendors is called glue logic. Functional verification and Signoff electronic design automation Chips are verified for logical correctness before being sent to a semiconductor foundry. Bugs found in the verification stage are reported to the designer. Traditionally, engineers have employed simulation acceleration, emulation or prototyping on reprogrammable hardware to verify and debug hardware and software for SoC designs prior to the finalization of the design, known as tape-out. This is used to debug hardware, firmware and software interactions across multiple FPGAs with capabilities similar to a logic analyzer. In parallel, the hardware elements are grouped and passed through a process of logic synthesis , during which performance constraints, such as operational frequency and expected signal delays, are applied. This generates an output known as a netlist describing the design as a physical circuit and its interconnections. These netlists are combined with the glue logic connecting the components to produce the schematic description of the SoC as a circuit which can be printed onto a chip. This process is known as place and route and precedes tape-out in the event that the SoCs are produced as application-specific integrated circuits ASIC. Optimization goals[edit] Systems-on-chip must optimize power use , area on die , communication, positioning for locality between modular units and other factors. Optimization is necessarily a design goal of systems-on-chip. If optimization was not necessary, the engineers would use a multi-chip module architecture without accounting for the area utilization, power consumption or performance of the system to the same extent. Common optimization targets for system-on-chip designs follow, with explanations of each. In general, optimizing any of these quantities may be a hard combinatorial optimization problem, and can indeed be NP-hard fairly easily. Therefore, sophisticated optimization algorithms are often required and it may be practical to use approximation algorithms or heuristics in some cases. Additionally, most SoC designs contain multiple variables to optimize simultaneously , so Pareto efficient solutions are sought after in SoC design. Oftentimes the goals of optimizing some of these quantities are directly at odds, further adding complexity to design optimization of systems-on-chip and introducing trade-offs in system design.

Chapter 4 : Pearson - ARM System-on-Chip Architecture, 2/E - Steve Furber

Steve Furber has a long association with the ARM, having helped create the first ARM chips during the s. Now an academic, but still actively involved in ARM development, he presents an authoritative perspective on the many complex factors that influence the design of a modern system-on-chip and the microprocessor core that is at its heart.

The highlights of the last decade of ARM development include: Most of the principles of modern SoC and processor design are illustrated somewhere in the ARM family, and ARM has led the way in the introduction of some concepts such as dynamically decompressing the instruction stream. The inherent simplicity of the basic 3-stage pipeline ARM core makes it a good pedagogical introductory example to real processor design, whereas the debugging of a system based around an ARM core deeply embedded into a complex system chip represents the cutting-edge of technological development today. Book Structure Chapter 1 starts with a refresher on first year undergraduate processor design material. It illustrates the principle of abstraction in hardware design by reviewing the roles of logic and gate-level representations. It then introduces the important concept of theReduced Instruction Set Computer RISC as background for what follows, and closes with some comments on design for low power. Chapter 2 describes the ARM processor architecture in terms of the concepts introduced in the previous chapter, and Chapter 3 is a gentle introduction to user-level assembly language programming and could be used in first year undergraduate teaching for this purpose. Chapter 4 describes the organization and implementation of the 3- and 5-stage pipeline ARM processor cores at a level suitable for second year undergraduate teaching, and covers some implementation issues. Chapters 5 and 6 go into the ARM instruction set architecture in increasing depth. Chapter 5 goes back over the instruction set in more detail than was presented in Chapter 3, including the binary representation of each instruction, and it penetrates more deeply into the comers of the instruction set. It is probably best read once and then used for reference. Chapter 6 backs off a bit to consider what a high-level language in this case, C really needs and how those needs are met by the ARM instruction set. This chapter is based on second year undergraduate material. It is of peripheral interest to a generic study of computer science, but adds an interesting lateral perspective to a postgraduate course. Chapter 8 raises the issues involved in debugging systems which use embedded processor cores and in the production testing of board-level systems. Chapter 10 introduces the concept of memory hierarchy, discussing the principles of memory management and caches. Chapter 11 reviews the requirements of a modern operating system at a second year undergraduate level and describes the approach adopted by the ARM to address these requirements. Chapter 13 covers the issues of designing SoCs with embedded processor cores. Here, the ARM is at the leading edge of technology. Several examples are presented of production embedded system chips to show the solutions that have been developed to the many problems inherent in committing a complex application-specific Chapter 14 moves away from mainstream ARM developments to describe the asynchronous ARM-compatible processors and systems developed at the University of Manchester, England, during the s. After a decade of research the AMULET technology is, at the time of writing, about to take its first step into the commercial domain.

Chapter 5 : ARM System-on-Chip Architecture (2nd Edition) - PDF Free Download

This book introduces the concepts and methodologies employed in designing a system-on-chip (SoC) based around a microprocessor core and in designing the microprocessor core itself. The principles of microprocessor design are made concrete by extensive illustrations based upon the ARM. The aim of the.

Chapter 6 : Support | Arm System-on-chip Architecture â€“ Arm Developer

ARM system-on-chip architecture - Free ebook download as PDF File .pdf), Text File .txt) or read book online for free. Scribd is the world's largest social reading and publishing site. Search Search.

DOWNLOAD PDF ARM SYSTEM-ON-CHIP ARCHITECTURE

Chapter 7 : ARM System-on-Chip Architecture - PDF Free Download

System-on-chip (SoC) technology is revolutionizing the way computers are designed and used, driving down their cost and making them more pervasive than ever before. However, it's extremely challenging for designers to get their SoC designs right the first time. ARM System Architecture, Second.

Chapter 8 : ARM System-on-chip Architecture - Stephen Bo Furber - Google Books

We use cookies to make interactions with our website easy and meaningful, to better understand the use of our services, and to tailor advertising.

Chapter 9 : Furber, ARM System-on-Chip Architecture, 2nd Edition | Pearson

ARM's developer website includes documentation, tutorials, support resources and more. Over the next few months we will be adding more developer resources and documentation for all the products and technologies that ARM provides.