CHAPTER 5

SOFTWARE DESCRIPTION

5.1 VERILOG

5.1.1 Introduction

In electronics, a hardware description language (HDL) is a specialized computer language used to program the structure, design and operation of electronic circuits, and most commonly, digital logic circuits.

A hardware description language enables a precise, formal description of an electronic circuit that allows for the automated analysis, simulation, and simulated testing of an electronic circuit. It also allows for the compilation of an HDL program into a lower level specification of physical electronic components, such asthe set of masks used to create an integrated circuit.

A hardware description language looks much like a programming language such as C, it is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs explicitly include the notion of time.

HDLs form an integral part of electronic design automation (EDA) systems, especially for complex circuits, such as microprocessors.

5.1.2 Motivation

Due to the exploding complexity of digital electronic circuits since the 1970s (see Moore's law), circuit designers needed digital logic descriptions to be performed at a high level without being tied to a specific electronic technology,

such as CMOS or BJT. HDLs were created to implement register-transfer level abstraction, a model of the data flow and timing of a circuit.

There are two major hardware description languages: VHDL and Verilog. There are different types of description in them "dataflow, behavioral and structural".

5.1.3 Structure of HDL

HDLs are standard text-based expressions of the structure of electronic systems and their behaviour over time. Like concurrent programming languages, HDL syntax and semantics include explicit notations for expressing concurrency. However, in contrast to most software programming languages, HDLs also include an explicit notion of time, which is a primary attribute of hardware. Languages whose only characteristic is to express circuit connectivity between a hierarchy of blocks are properly classified as netlist languages used in electric computer-aided design (CAD). HDL can be used to express designs in structural, behavioral or register-transfer-level architectures for the same circuit functionality; in the latter two cases the synthesizer decides the architecture and logic gate layout.

HDLs are used to write executable specifications for hardware. A program designed to implement the underlying semantics of the language statements and simulate the progress of time provides the hardware designer with the ability to model a piece of hardware before it is created physically. It is this executability that gives HDLs the illusion of being programming languages, when they are more precisely classified as specification languages or modeling languages. Simulators capable of supporting discrete-event (digital) and continuous-time (analog) modeling exist, and HDLs targeted for each are available.

5.1.4 Comparison with Control-Flow Languages

It is certainly possible to represent hardware semantics using traditional programming languages such as C++, which operate on control flow semantics as opposed to data flow, although to function as such, programs must be augmented with extensive and unwieldy class libraries. Generally, however, software programming languages do not include any capability for explicitly expressing time, and thus cannot function as hardware description languages. Before the introduction of System Verilog in 2002, C++ integration with a logic simulator wasone of the few ways to use object-oriented programming in hardware verification. System Verilog is the first major HDL to offer object orientation and garbage collection.

Using the proper subset of hardware description language, a program called a synthesizer, or logic synthesis tool, can infer hardware logic operations from the language statements and produce an equivalent netlist of generic hardware primitives[jargon] to implement the specified behaviour.[citation needed] Synthesizers generally ignore the expression of any timing constructs in the text. Digital logic synthesizers, for example, generally use clock edges as the way to time the circuit, ignoring any timing constructs. The ability to have a synthesizable subset of the language does not itself make a hardware description language.

5.2 HISTORY

The first hardware description languages appeared in the late 1960s, looking like more traditional languages. The first that had a lasting effect was described in

1971 in C. Gordon Bell and Allen Newell's text Computer Structures. This text introduced the concept of register transfer level, first used in the ISP language to describe the behavior of the Digital Equipment Corporation (DEC) PDP-8.

The language became more widespread with the introduction of DEC's PDP-16 RT-Level Modules (RTMs) and a book describing their use. At least two implementations of the basic ISP language (ISPL and ISPS) followed. ISPS was well suited to describe relations between the inputs and the outputs of the design and was quickly adopted by commercial teams at DEC, as well as by a number of research teams both in the USAand among its NATO allies. The RTM products never took off commercially and DEC stopped marketing them in the mid-1980s,as new techniques and in particular very-large-scale integration (VLSI) became more popular. Separate work done about 1979 at the University of Kaiserslautern produced a language called KARL, which included design calculus language features supporting VLSI chip floor planning [jargon] and structured hardware design. This work was also the basis of KARL's interactive graphic sister language ABL. ABL was implemented in the early 1980s by the Centro Laboratory Tele-communication (CSELT) in Torino, Italy, producing the ABLED graphic VLSI design editor. In the mid-1980s, a VLSI design framework was implementedaround KARL and ABL by an international consortium funded by the Commissionof the European Union.

By the late 1970s, design using programmable logic devices (PLDs) became popular, although these designs were primarily limited to designing finite state machines. The work at Data General in 1980 used these same devices to design the Data General Eclipse MV/8000, and commercial need began to grow for a language that could map well to them. By 1983 Data I/O introduced ABEL to fill that need.

As design shifted to VLSI, the first modern HDL, Verilog, was introduced by Gateway Design Automation in 1985. Cadence Design Systems later acquired the rights to Verilog-XL, the HDL simulator that would become the de facto standard of Verilog simulators for the next decade. In 1987, a request from the U.S. Department of Defense led to the development of VHDL (VHSIC Hardware Description Language). VHDL was based on the Ada programming language, as well as on the experience gained with the earlier development of ISPS. Initially, Verilog and VHDL were used to document and simulate circuit designs already captured and described in another form (such as schematic files). HDL simulation enabled engineers to work at a higher level of abstraction than simulation at the schematic level, and thus increased design capacity from hundreds of transistors to thousands.

The introduction of logic synthesis for HDLs pushed HDLs from the background into the foreground of digital design. Synthesis tools compiled HDL source files (written in a constrained format called RTL) into a manufacturable netlist description in terms of gates and transistors. Writing synthesizable RTL files required practice and discipline on the part of the designer; compared to a traditional schematic layout, synthesized RTL netlists were almost always larger inarea and slower in performance[citation needed]. A circuit design from a skilled engineer, using labor-intensive schematic-capture/hand-layout, would almost always outperform its logically-synthesized equivalent, but the productivity advantage held by synthesis soon displaced digital schematic capture to exactly those areas that were problematic for RTL synthesis: extremely high-speed, low- power, or asynchronous circuitry.

Within a few years, VHDL and Verilog emerged as the dominant HDLs in the electronics industry, while older and less capable HDLs gradually disappeared from use. However, VHDL and Verilog share many of the same limitations: neitheris suitable for analog or mixed-signal circuit simulation; neither possesses language constructs to describe recursively-generated logic structures. Specialized HDLs (such as Confluence) were introduced with the explicit goal of fixing specific limitations of Verilog and VHDL, though none were ever intended to replace them. Over the years, much effort has been invested in improving HDLs. The latest iteration of Verilog, formally known as IEEE 1800-2005 SystemVerilog, introduces many new features (classes, random variables, and properties/assertions) to address the growing need for better test bench randomization, design hierarchy, and reuse. A future revision of VHDL is also in development, and is expected to match SystemVerilog's improvements.

5.3 DESIGN USING HDL

As a result of the efficiency gains realized using HDL, a majority of modern digital circuit design revolves around it. Most designs begin as a set of requirements or a high-level architectural diagram. Control and decision structures are often prototyped in flowchart applications, or entered in a state diagram editor. The process of writing the HDL description is highly dependent on the nature of the circuit and the designer's preference for coding style. The HDL is merely the 'capture language', often beginning with a high-level algorithmic description such as a C++ mathematical model. Designers often use scripting languages such as Perl to automatically generate repetitive circuit structures in the HDL language. Special text editors offer features for automatic indentation, syntax-dependent coloration, and macro-based expansion of entity/architecture/signal declaration.

The HDL code then undergoes a code review, or auditing. In preparation for synthesis, the HDL description is subject to an array of automated checkers. The checkers report deviations from standardized code guidelines, identify potential ambiguous code constructs before they can cause misinterpretation, and check for common logical coding errors, such as dangling[jargon] ports or shorted outputs. This process aids in resolving errors before the code is synthesized.

In industry parlance, HDL design generally ends at the synthesis stage. Once the synthesis tool has mapped the HDL description into a gate netlist, the netlist is passed off to the back-end stage. Depending on the physical technology (FPGA, ASIC gate array, ASIC standard cell), HDLs may or may not play a significant role in the back-end flow. In general, as the design flow progresses toward a physically realizable form, the design database becomes progressively more laden with technology-specific information, which cannot be stored in a generic HDL description. Finally, an integrated circuit is manufactured or programmed for use.

5.4 SPARTAN 3

The Spartan-3 family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 50,000 to five million system gates. The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions.

These Spartan-3 FPGA enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry.

Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications including broadband access, home networking, display/ projection and digital television equipment. The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

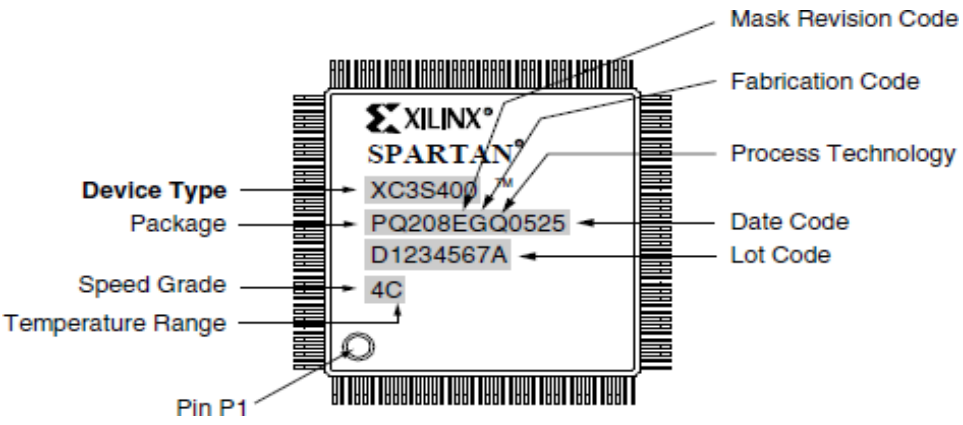

Figure 5.1 Spartan-3 Package XC3S400-4PQ208C

5.5 ARCHITECTURAL OVERVIEW

The Spartan-3 family architecture consists of five fundamental programmable functional elements:

1. Configurable Logic Blocks (CLBs) contain RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.

2. Input/ Output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Twenty-six different signal standards, including eight high-performance differential standards are available. Double Data-Rate (DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.

3.Block RAM provides data storage in the form of 18-K bit dual-port blocks.

4. Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.

5. Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing and phase shifting clocksignals.

A ring of IOBs surrounds a regular array of CLBs. The XC3S50 has a single column of block RAM embedded in the array. Those devices ranging from the XC3S200 to the XC3S2000have two columns of block RAM. The XC3S4000 and XC3S5000 devices have four RAM columns. Each column is made up of several 18-Kbit RAM blocks; each block is associated with a dedicated multiplier.

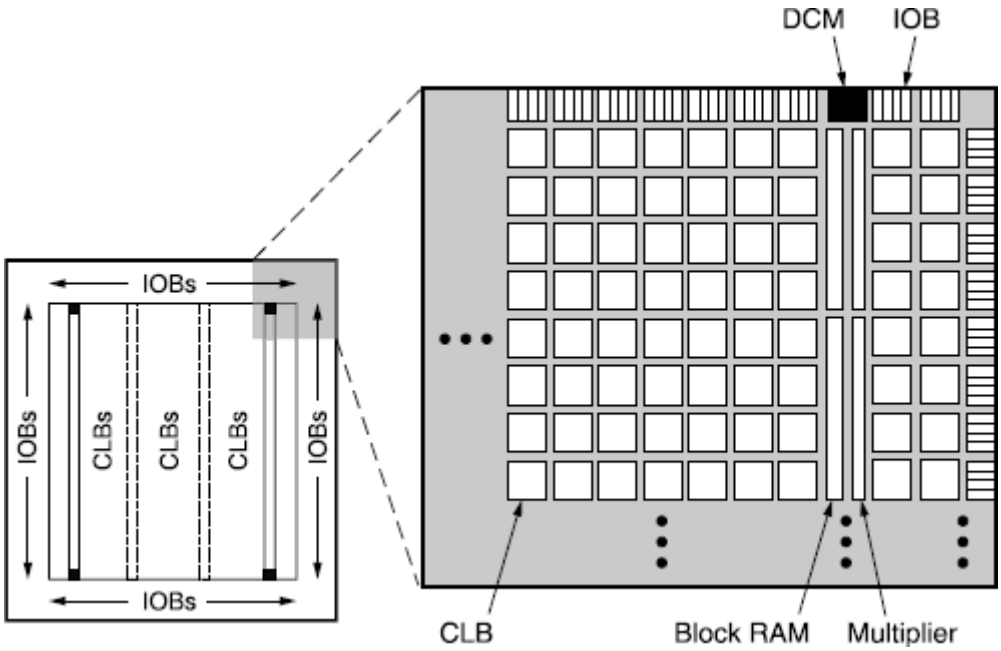

Figure 5.2 Spartan-3 Architecture

The DCMs are positioned at the ends of the outer block RAM columns. The Spartan-3 family features a rich network of traces and switches that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.

5.6 CONFIGURABLE LOGIC BLOCK

The Configurable Logic Blocks (CLBs) constitute the main logic resourcefor implementing synchronous as well as combinatorial circuits. Each CLB comprises four interconnected slices as shown in Fig.5.3. These slices are grouped in pairs. Each pair is organized as a column with an independent carry chain.

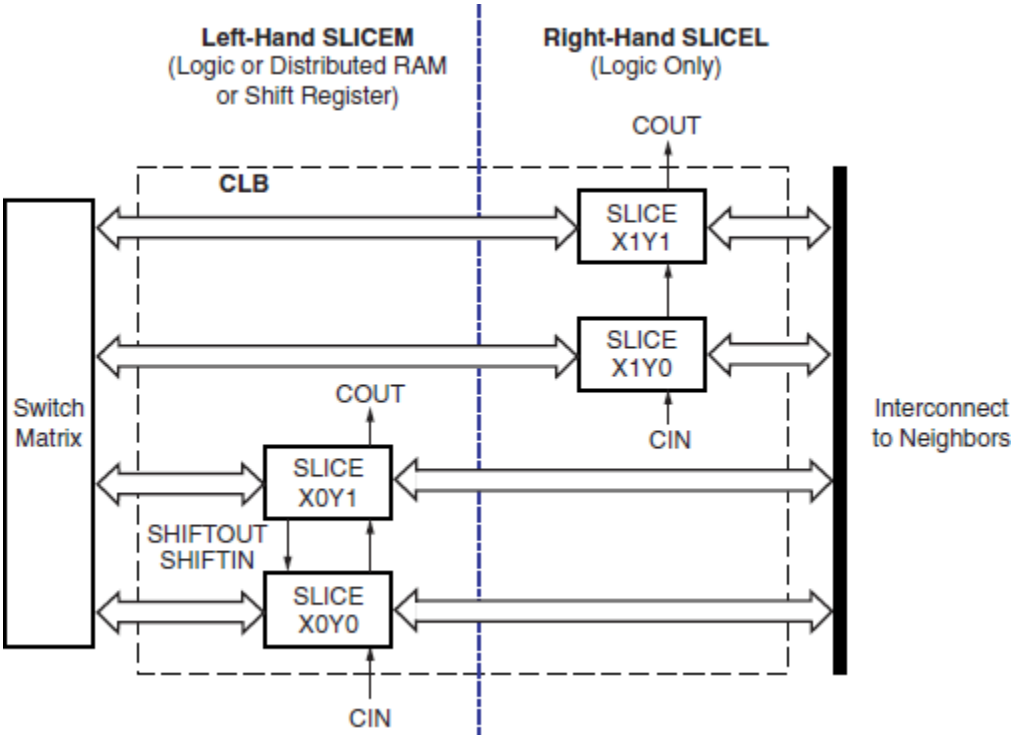

Figure 5.3 Arrangement of Slices within the CLB

The nomenclature that the FPGA Editor part of the Xilinx development software uses to designate slices is as follows: The letter 'X' followed by a number identifies columns of slices. The 'X' number counts up in sequence from the left side of the die to the right. The letter 'Y' followed by a number identifies the position of each slice in a pair as well as indicating the CLB row. The 'Y' number counts slices starting from the bottom of the die according to the sequence: 0, 1, 0, 1 (the first CLB row); 2, 3, 2, 3 (the second CLB row); etc. Fig. 5.3 shows the CLBlocated in the lower left-hand corner of the die. Slices X0Y0 and X0Y1 make up the column-pair on the left where as slices X1Y0 and X1Y1 make up the column- pair on the right. For each CLB, the term "left-hand" (or SLICEM) indicates the pair of slices labeled with an even 'X' number, such as X0, and the term "right- hand" (or SLICEL) designates the pair of slices with an odd 'X' number, e.g., X1.