calculations, where it is not possible for the developer to manually intervene. In some cases language features may create many duplicate expressions. For instance, Cmacros, where macro expansions may result in common subexpressions not apparent in the original source code.

Compilers need to be judicious about the number of temporaries created to hold values. An excessive number of temporary values creates register pressure possibly resulting in spilling registers to memory, which may take longer than simply recomputing an arithmetic result when it is needed.

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.