Chapter 1

Introduction

1.1 What VLSI is all about

Engineering is about the construction of physical implementations from higher-level specifications, and VLSI design is no different. For digital VLSI design, the topic of these notes, the specifications tend to be algorithmic in nature, and may be represented by computer programs. Most of the creative work in digital VLSI design, and this may surprise you,consists of manipulating specifications of other kinds into algorithms, and algorithms of one form into algorithms of another. What drives the transformations is notions of efficiency: we want the transformations to yield a program from which it is easier to obtain a faster, smaller, or more energy efficient circuit. We continue such transformations until the remaining steps are simple enough so that they can be carried out by a computer program (CAD tool). Where we stop depends on how much time we have available and how good a job

the tools can do from the point where we stop.

In programming, measures of efficiency that drive the transformations, often referred to as *refinement steps*, are relatively well understood. Complexity theory gives us a good understanding of the speed of the (compiled) program, and the length of the code, plus the size of the declared variables, gives us a measure of how much memory is required to run the program. It is therefore relatively easy to give direction to the transformations.

Designing VLSI circuits amounts to much the same thing, except that the resources for the final program tend to be very limited, and the rules for correspondence between the algorithms and the efficiency of their realizations in silicon much less understood.

Becoming a good VLSI designer, therefore, requires that you become an excellent programmer, and that you develop a good sense of how choices at the algorithmic level are reflected in the final implementation. The latter is not possible without a firm understanding of the implementation medium: in our case CMOS integrated circuits.

One of the main goals of this course is to raise the level of abstraction (you'll hear this term a lot) as high as we can without sacrificing too much performance. Just as programmers worry only occasionally about how a certain programming construct is translated into machine code, we hope to let you worry about the physical properties of the transistors as little as possible.

There is, however, a category of programmers which worries almost exclusively about the correspondence between programs and code; we call them "compiler writers'." Some of you may want to learn to write "silicon com-

pilers," so this is another reason not to ignore the transistors underneath it all.

1.2 Discrete circuits

One of the very first problems we face is that whereas programs are defined in terms of discrete (non-continuous) transitions and (often) discrete values, physical systems, at least at a macroscopic scale, are described in terms of continuous parameters. We have to choose a mapping from the continuous domain to the discrete domain in order to define what it means for a circuit to correspond to a program. Our chosen technology is CMOS, so let us examine the characteristics of a fundamental CMOS circuit, the inverter, to help us choose judiciously.

The plot in Figure 1.1 is the "dc-characteristic" of a CMOS inverter (dc means that the input voltage is changed slowly enough so that changing it any slower would make no difference). We see that there are three distinctly different regions in the plot. In the region in the middle the inverter acts as an amplifier with a gain much bigger than one. In the two other regions the inverter acts as an amplifier with gain less than one, that is, the slope of the curve is between 0 and -1. As a consequence, if the input voltage is in one of the two outer regions, the output voltage will be driven even closer to one of the extremes, therefore the extremes make a good choice for our discrete values. We will use the name V_l for a region around GND and V_h for a region around the high voltage, commonly referred to as Vdd. The size of these regions is chosen so that they fall comfortably within the region where

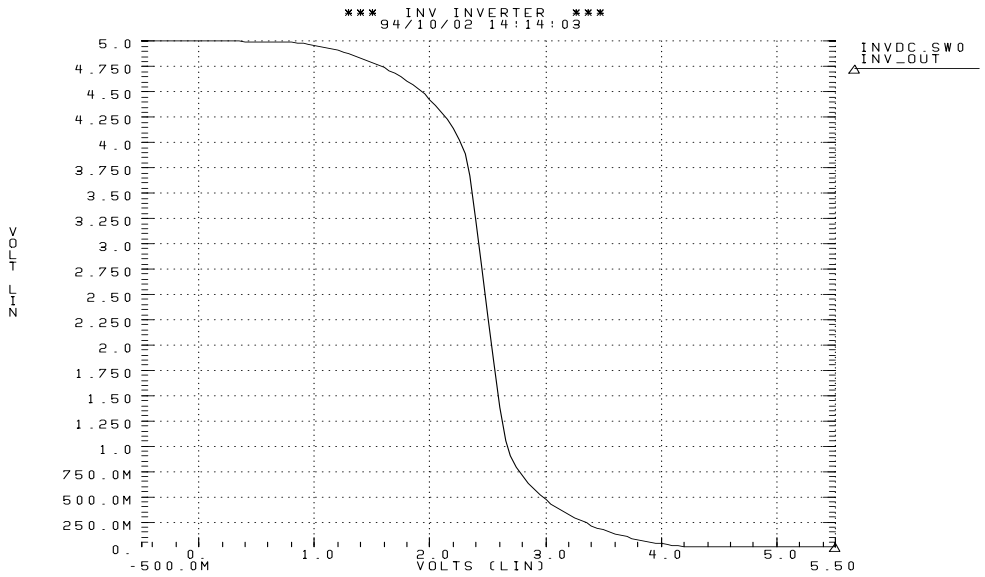

Figure 1.1: dc characterisctic of a CMOS inverter

the inverter operates with gain less than one. Because of this, if the input is within V_h or V_l, the ouput is even closer to the middle of the (other) range; this is commonly referred to as "noise immunity."

We have now solved our problem of discrete values, but not the problem of discrete transitions. If in a program a boolean variable x is set from *true* to *false*, we do not expect to observe undefined values inbetween. As you can see in Figure 1.2 an inverter does not respond instantaneously to a (nearly) instantaneous transition on its input, and intermediate values can be observed.

One way to solve this problem is to agree to only examine the outputs at certain prespecified times, and to only change the inputs at certain (other) prespecified times. A clock defines these times for us, and helps us to examine

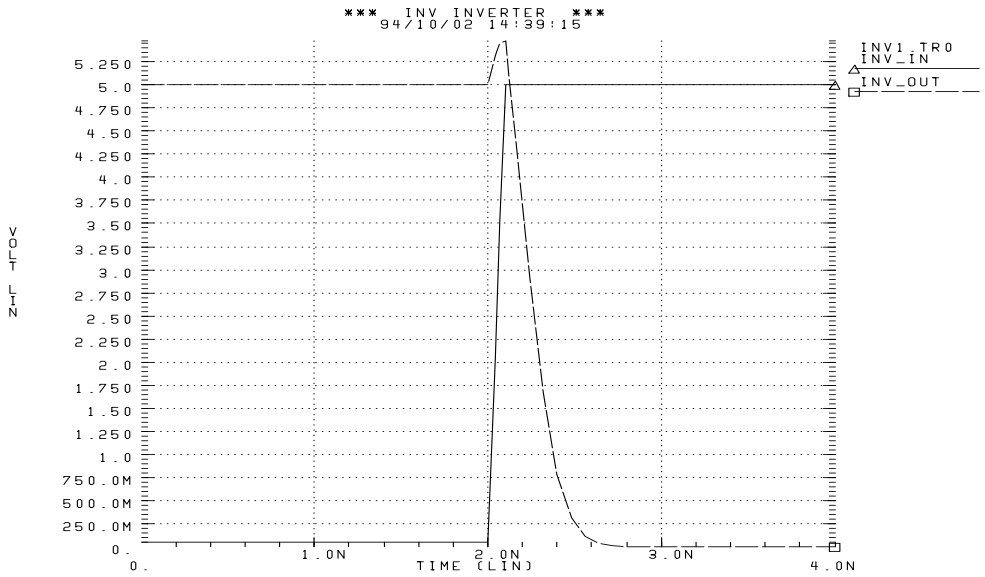

Figure 1.2: response to a step function

our analog circuits "discretely." We will see much later in this course that this is not the only possible solution; contrary to what many people believe, clocks are not necessary to build discrete circuits.