This chapter will give the reader a first overview of the hardware description language VERILOG by several small examples. Together with the extensive introduction in Chapter 11, which can be used whenever needed in parallel to the remainder of the book, and with the training simulator VeriWell on the enclosed disk, all foundations and concepts for understanding the VERILOG models of the processor TOOBSE are presented. We assume and recommend that the reader knows at least one structured programming language like Pascal, Modula-2, or C; in particular, VERILOG is very similar to C.

VERILOG allows both, structural descriptions as netlists of gates, components, and modules, and algorithmic behavioral models with variables, case selections (if, case), loops (for, while), and procedures (function, task) as well as arbitrary combinations of structure and behavior (Section 2.2).

```
module count;         // this is a comment    // 00
    integer I;        // 01
    initial          // 02
        begin       // 03
            $display ("Starting simulation..."); // 04
            for (I=1; I <= 3; I=I+1)        // 05
                $display ("run %d", I);      // 06
            $display ("Finished");          // 07
        end                   // 08
    endmodule             // 09

Figure 4.1 A simple module
```

Modules are the basic units in VERILOG for gates, counters, CPUs, or complete computers. They can be nested hierarchically. Module count in Figure 4.1, enclosed by `module` and `endmodule`, outputs the numbers 1 to 3 on a display. In line 1, count variable I is defined. The compound statement following `initial` will be executed exactly once. The instruction `display` outputs a character string. As in C, `%d` outputs I as an integer. // starts a
comment until the end of line. Like all reserved words, begin is written with
small letters, and it is different from Begin and BEGIN. The instructions
between begin and end are executed in sequential order. The simulator
VERILOG-XL produces Figure 4.2 as result.

<table>
<thead>
<tr>
<th>VERILOG-XL 2.0.1  Jun 7, 1995 13:50:32</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>Compiling source file &quot;bspl.v&quot;</td>
</tr>
<tr>
<td>Highest level modules:</td>
</tr>
<tr>
<td>count</td>
</tr>
<tr>
<td>Starting simulation...</td>
</tr>
<tr>
<td>run 1</td>
</tr>
<tr>
<td>run 2</td>
</tr>
<tr>
<td>run 3</td>
</tr>
<tr>
<td>Finished</td>
</tr>
<tr>
<td>8 simulation events</td>
</tr>
<tr>
<td>CPU time: 0.1 secs to compile + 0.1 secs to link + 0.0 secs in simulation</td>
</tr>
<tr>
<td>End of VERILOG-XL 2.0.1  Jun 7, 1995 13:50:32</td>
</tr>
</tbody>
</table>

**Figure 4.2** Output of Figure 4.1

Module maximum in Figure 4.3 computes the maximum of two numbers A and
B and stores the result in MAX. The parameter list after the module name
contains the variables coming from the outside of the module (A and B) and go
to other modules (MAX). In addition, for every variable its direction input or
output has to be declared; inout means both directions. In lines 4 to 6, the
variable types are declared. Aside from integer, there are wire, reg, real,
event, and time.

We will only explain the difference between wire and reg, the others are
treated in Chapter 11. A wire of type wire represents a connection between
several nodes of a circuit. In a register of type reg, binary values can be written
and read. This is not possible for a wire only connecting two or more points in
all directions.

wire and reg are hardware oriented data types; both represent words of
several bits, where each bit may not only be 0 and 1, but also z and x: z means
the high impedance state, and x stands for unknown; when several sources
drive a wire differently, the result is x. For both types, the bit width may be
indicated by an interval in brackets before the variable definition. A[31]
accesses bit 31 of A.

The statement always executes the following (compound) statement always
again. When reaching condition

@ (A or B),