Abstract
This paper presents a methodology by which, irrespective of the tester used, test programs for testing integrated circuits can be created with the aid of a high-level language.

Introduction
Verification of the circuit design and testing of the actual device are very closely related activities. Both require from time to time a large number of test vectors, in order to be able to verify their specific functionality and freedom from faults. Conventional simulation-stimuli languages are, however, not in a position to guarantee the transfer of the simulation patterns to a test system. Therefore, it may happen that tests are defined which, due to tester restrictions, cannot, or only with great difficulty be realised on the target test system. This suggests the use of a pattern-design environment satisfying the following requirements:

- test vectors can be defined without detailed knowledge of the test system being used.
- timing is checked for compatibility with the target tester before simulation.
- the test vectors can be simulated at a logic workstation.
- the device response, which is determined through logic simulation, can be used for the generation of the test vectors.
- automatic comparison of the simulated and expected device response is possible.
- modular pattern development must be supported; i.e. subtests can be developed independently, they can make use of standardized, device specific routines.
- the language should support the documentation of patterns and timing.

Hierarchical test program design
The development of test vectors for complex VLSI devices can be divided into several subtests. A preferable approach is a hierarchical organization of pattern development, such as shown in Fig. 1.

The lowest level, the basic language, referred to as TOPS ("Testerorientierte Pattern Sprache" - tester oriented pattern language) in the rest of this paper,
represents the elementary language elements for the definition of patterns and timing conditions.

The next level, the device interface program, referred to as DIP ("Device interface program"), defines device specific symbols (e.g. register names and timing conditions) and access routines. An additional module in DIP determines tester specific data.

On the highest level, tests formulated in the high level language, referred to as HL ("High level") tests in this paper, are developed by the logic designer for the individual circuit blocks. HL tests make use of the access routines defined in the DIP, thus guaranteeing the consistency of all the subtests with respect to interface control and timing conditions.

**Figure 1: Hierarchical test program definition**

**Pattern generation process**

Pattern generation takes place in 2 stages (Fig.2). In the first stage, an input stimuli file for the logic simulator is generated with the aid of the pattern description language. Simulation generates an output file which consists of the device response. This file is used in the second stage for generating a test program for the target tester.

**Figure 2: Definition and generation of test programs**

**The basic language (TOPS)**

TOPS provides the concept of module, variable and constant of Pascal (1/1). TOPS statements are realized as Pascal procedure and function calls.

The predefined procedures can be divided into 4 groups:

- **Initialization**
  This is where the input files, output files and the documentation level are determined.

- **Signal definition**
  Definition of signals, the direction of signals, signal groups and the associated test strobes.

- **Timing**
  The timing of the device is, as it is usual in the case of test computers, defined in a cycle-
oriented manner using the concept of time sets according to the specification. Signal changes are determined relative to the beginning of the cycle.

- Pattern definition
  Signal sequences are defined by the activation of time sets and the determination of the signal changes in the time set. Test strobes can be activated in this group.

- Tester interface
  Timing and pattern definitions are tested as to whether the signal sequences generated are compatible with the target tester. In addition, it is possible to define how the tester hardware is used in the test program, e.g., which timing generator is connected to which device pin.

Device interface program
The DIP makes the routines available that the logic designer can use to generate patterns. Depending on the device it is possible, by means of procedures defined here, to access the various signals or signal groups. The timing of signals or test strobes is defined in the DIP by an experienced test engineer in accordance with the specification. The most important tester compatibility checks can be conducted while creating the DIP. Then, the generation of patterns using these procedures always leads to executable test programs. This can also be carried out by engineers who do not possess detailed test know-how. In order to unify the pattern development of the design engineers, a standardized definition of timing and pattern applying routines exist in DIP.

Furthermore, device-specific symbols, such as the names of device registers, pins, test strobes and buses, are agreed on.

Documentation support procedures can be made available, e.g., for the disassembling of registers and device register bits. Device programming can be monitored, and if necessary, forbidden programming of the device or inconsistencies can be detected as early as in the definition phase and reported by means of error messages.

Timing can be calculated automatically through analysis of device programming. For example with a serial controller where the baud rate is being programmed it is then possible to calculate the resulting data frequency.

Diagnostic functions can be implemented. They show which timing has been selected and which test strobes have been activated.

Device initialization sequences are also defined in DIP.

In general, the DIP provides a comfortable environment for pattern programming. This environment relieves the HL tests of unnecessary overhead. Therefore the resulting HL test modules are extremely compact and readable, and the degree of self-documentation is increased.

One part of the DIP is the tester dependent part. Through specification of the target tester and the associated tester specification file, the tester compatibility checks in TOPS are activated and the desired generation of the test programs made possible. In this part, tester periods and tester timing generators are assigned explicitly. The aim of this is to be able to determine in the High level language the structure of the test programs and assignment of the tester hardware. Together with the comments, included in the source code of the
test program, it is possible to generate readable test programs which significantly simplify the debugging at the test system.

Changing the tester family thus involves merely the exchange of the tester dependent module. After checking whether the timing definitions are still compatible one can generate easily the new test programs. Additionally changing the tester might make a slight modification of the timing definition in DIP necessary.

**HL tests**

HL tests make use of the routines made available in DIP. HL tests form the major part in formulating a device test, and due to the hierarchical programming, they are completely independent of the target tester.

A test program for an ISDN terminal adapter device was designed using the above methods. This device can be categorized as a CPU Peripheral Device. The device has 3 interfaces: a parallel microprocessor interface with an interrupt line and 2 serial interfaces which are operated with completely different rates and data formats. The device can receive and send serial data. Built-in are a USART controller and an HDLC controller with FIFOs. All 3 interfaces can be operated in parallel.

HL tests of the highest level have the following structure.

1. Determine the diagnostic level.
2. Reset the device.
3. Program the device register.
4. Define which data shall be sent to the serial interfaces.
5. Determine which of the predefined test strobes shall be activated.
6. Start command for sending of the serial data (which data rates have been programmed and whether the device has been switched to certain test modes is previously determined by device register programming. Timing and data are automatically calculated from this information.)
7. If necessary, repetition of points 3 to 6 with different device register programming and different data can be done.

The following program (Fig. 3) serves as a sample of a short subtest:

```pascal
PROCEDURE sample_test;
BEGIN
  set_message_level(2);
  define_subtitle('Sample Testprogram v1.0');
  msg('Start of DUT initialisation');
  reset; {reset DUT}
  wr_reg(brs,%xaa); {write DUT Register named brs}
  wr_reg(gcr,%xff); {write DUT Register named gcr}
  wr_reg(te2,%x01); {put DUT into Testmode by writing Register te 2}
  msg('End of DUT initialisation');
  write_dte(%x55); {provide data for serial-Interface 1}
  write_sni(%x33); {provide data for serial-Interface 2}
  show_variables; {show internal variables for Dok}
  run_cycles(fsc-period); {apply serial data for fsc-period}
  rd_reg(ist,%x00); {read and test DUT Register named IST the expected value is %x00}
  rd_dma(%xff); {read DUT fifos via DMA Access}
  wr_dma(%x55); {write DUT fifos via DMA Access}
  msg('End of Sample Test');
END;
```

**Figure 3: Sample program**

The above sample program generates the following log file (Fig. 4). One recognizes the automatic documentation-support feature for disassembling register access; e.g. it has been noted at cycle 3 in the DIP that the BRS register is a rate register. This explains why the rates programmed are stated in the next line.

After cycle 7, a diagnostic function is called up which indicates parameters that have been
calculated automatically by DIP from the programming of device register. In cycle 9, the serial interfaces of the device are activated. One sees that serial sending lasts 16 cycles. In cycles 26 and 27, device registers are read and tested. Reading is carried out twice once with DMA and once without DMA.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Time</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 nsec</td>
<td>RESET until 4000 nsec</td>
</tr>
<tr>
<td>3</td>
<td>4000 nsec</td>
<td>Write BRS (SE) ... AA hex 10101010 bin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-&gt; User rate: 64000 bits Net_rate: 64000 bits</td>
</tr>
<tr>
<td>5</td>
<td>4150 nsec</td>
<td>Write GCR (50) ... FF hex 11111111 bin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-&gt; PU:1 DOE:1 V24:1 V110:1 ASY:1 ENFR:1 DMA:1</td>
</tr>
<tr>
<td>7</td>
<td>4300 nsec</td>
<td>Write TE2 (2C) ... 01 hex 00000001 bin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-&gt; TMSP:0 PGCN:0 _1_CMP:0 TDIR:0 M:0</td>
</tr>
</tbody>
</table>

End of DUT initialisation

Program Version: 1
Oscillator_period: 3000
CLK_period: 6000
FSC_period: 16
FSC_offset: 0
FSC_high_time: 8
SDR_default: 0
TXD_default: 1
SN1_timeslot: 0
SN1_used_bits: 8
DTE_period: 8
IOM_default: 0

End of Sample Test

Figure 4: Log file of the sample program

Experience from application

The program is loadable in VAX/VMS. 18 subtests have been generated for Sentry 21 with a total of approx. 54,000 test vectors. The computing times are within the range of a few minutes, not only for the generation of simulator stimuli, but also for the generation of the test patterns (without logic simulation).

Figure 5 presents the various parts of the program in terms of lines of code. The number of target-tester-dependent lines of code is very small. Altogether, it accounts for only 6% of the total program. This target-tester-specific part is not particularly complex, this means that it should be possible, with respect to development effort, to cope with a change of the development environment for a different target tester in considerably less than a day.

HL tests: 11410 lines of code
DIP: 2847 lines of code
DIP tester-dependent: 87 lines of code
Total: 14340 lines of code

Figure 5: Test program portability

Conclusion

A method has been illustrated to generate complex and large pattern quantities by means of a structured methodology. As an example of this, we examined the development of a test program for an ISDN terminal adapter device. We think it would not be possible to develop a comparably complex test program for VLSI circuits within the bounds of acceptable outlay without a tool like TOPS.

References

1/ Herbert Eichinger, "TOPS a high level language for linking design and test", Grassau 2-4.5.89