

Testen komplexer digitaler Schaltungen mit Python

S. Reichör¹, G. Hueber¹, R. Hagelauer¹, and M. Lindorfer²

¹Institut für integrierte Schaltungen, Universität Linz, Austria

²Linz Center of Mechatronics GmbH, Austria

Zusammenfassung. Die Verifikation von digitalen Schaltungen nimmt heutzutage einen bedeutenden Stellenwert ein. In diesem Paper wird ein Weg beschrieben, der die Erstellung und Wartung von funktionalen Testbenches für digitale Designs unterstützt. Für viele Projekte übersteigt der zeitliche Aufwand für das Testen den Aufwand für die Implementierung der Schaltung. In vielen Fällen beträgt der Aufwand für das Testen bereits 70% des Entwicklungsaufwands (Bergeon, 2000).

Typischerweise wird die Testbench auch in der gewählten Hardwarebeschreibungssprache (VHDL oder Verilog) implementiert. Diese Sprachen stellen jedoch nicht die beste Wahl für Verifikationsbelange dar. Gründe dafür sind darin zu suchen, dass diese Sprachen wichtige Konzepte aus den Softwaresprachen (wie z.B. Objektorientierung) nicht kennen. Weiters stehen komfortable Softwarebibliotheken (Zufallszahlengenerierung, Stringverarbeitung, etc.) den Hardwarebeschreibungen nicht zur Verfügung. In diesem Paper wird der Einsatz der Programmiersprache Python (PythonHomepage, 2003; Beazley, 2001) für die Verifikation vorgeschlagen, um die benötigte Zeit für die Funktionalen Tests zu reduzieren.

1 Einführung

Für den Entwurf von digitalen Schaltungen werden die Hardwarebeschreibungssprachen VHDL und Verilog verwendet. Die am Markt verfügbaren EDA (Electronic Design and Automation) Tools können Designs, die mit diesen Sprachen modelliert wurden, verarbeiten. Für die Beschreibung von digitaler Hardware stellen diese Sprachen eine ausgezeichnete Wahl dar. Im weiteren wird von der Beschreibungssprache VHDL gesprochen – im wesentlichen gilt das für VHDL gesagte auch für Verilog. Beim Entwurf der für die Verifikation benötigten Testbenches sollte man den Einsatz von reinen Hardwarebeschreibungssprachen jedoch überdenken, weil es

Correspondence to: S. Reichör
(reichoer@riic.at)

bei den Programmiersprachen im Softwarebereich Alternativen gibt, die es dem Designer ermöglichen, mit weniger Programmieraufwand bessere Testbenches zu implementieren.

Die Abb. 1 zeigt die Struktur einer Testbench in VHDL. Das VHDL Design wird als “Device under Test” (DUT) in eine Testbench eingebunden. Um die Funktionalität des DUT zu überprüfen, muss die Testbench Stimuli generieren. Diese Stimuli werden vom DUT verarbeitet und resultieren im Ergebnis der Schaltung.

Folgende Komponenten der Gesamtsimulation mit der Testbench verdienen eine genauere Betrachtung:

Stimuli. Müssen so modelliert werden, dass sie den realen Umgebungsbedingungen des DUT entsprechen, damit eine brauchbare und realitätsnahe Simulation durchgeführt werden kann.

DUT. Sollte womöglich keine Spezialbehandlung für die Simulation beinhalten. Durch die Synthese fällt diese Spezialbehandlung aus der Beschreibung heraus. Dadurch ist es nicht mehr möglich, diese Testbench auch für Postsynthesesimulationen zu verwenden. Die Testbench sollte daher so aufgebaut sein, dass sie mit der zugänglichen Portschnittstelle des DUT das Auslangen findet.

Response. Stellt die Antwort des DUT auf die Stimuli dar. Dieser Response muss auf seine Korrektheit überprüft werden. Dies kann durch manuelle Inspektion bzw. vorzugsweise durch automatisiertes Vergleichen mit erwarteten Werten durchgeführt werden.

2 Rückblick: Waves

In Hanna et al. (1997) wird vorgeschlagen, die benötigten Stimuli für den Test nicht in VHDL, sondern mit einer neu geschaffenen Stimulibeschreibungssprache (Waves) zu implementieren. Die Testbenchstruktur für die neue Herangehensweise ist in Abb. 2 zu sehen. Die Stimuli werden nicht mehr als VHDL-Programm implementiert, stattdessen werden die Stimuli in eigene Dateien abgelegt. Diese Dateien enthalten die Stimuli in der Stimulibeschreibungssprache. Eine VHDL Testbench liest diese Stimuli mittels eines Parsers (dieser ist auch in VHDL implementiert) ein und wandelt sie in Signalzuweisungen für das DUT um.

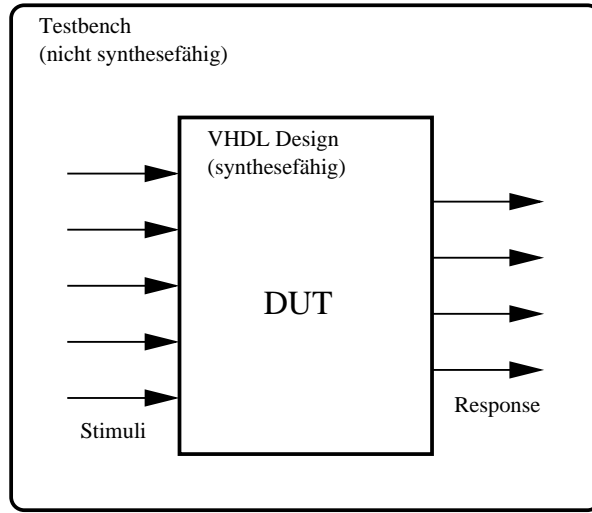


Abbildung 1. Prinzipieller Aufbau einer Testbench.

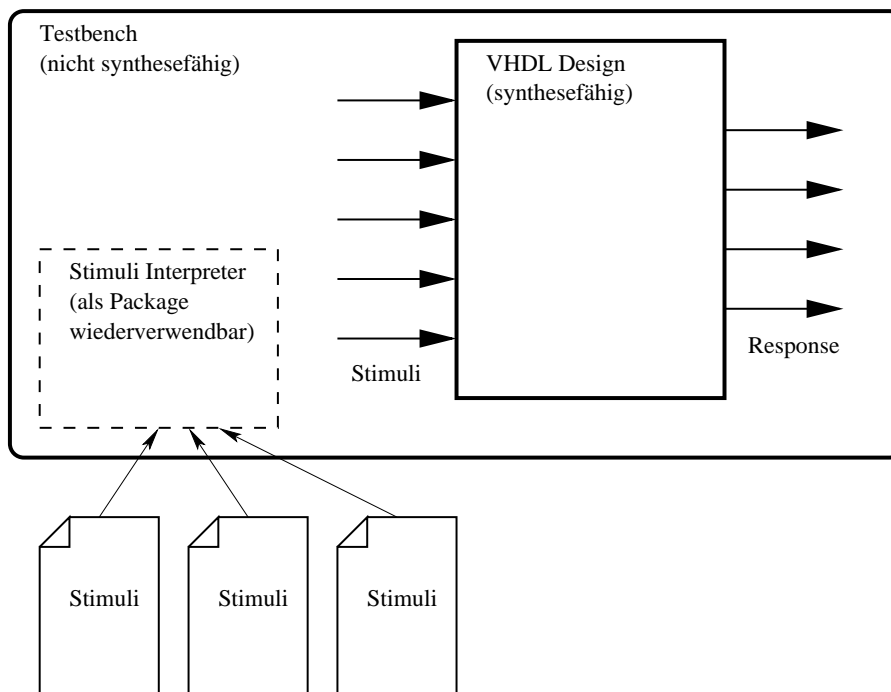


Abbildung 2. Aufbau einer Testbench mit Waves.

Dieser Zugang bietet folgende Vorteile:

- Die Stimulidaten sind von der VHDL Testbench entkoppelt. Dies ermöglicht es, mehrere unabhängige Stimulidateien bereitzustellen. Eine Änderung (z.B. ein Bugfix) in der Testbench erfordert keine Änderung in den Stimulidaten.
- Einfache Beschreibung für die Abfolge der Stimuli (siehe Abb. 3).
 - Kommentare beginnen mit dem “%” Symbol.
 - Die Stimuli für die einzelnen Eingänge werden als Binärvektoren kodiert
 - Die Zeit nach dem “:” gibt an, welcher zeitlicher Abstand zwischen diesem und dem nächsten Stimulipattern liegt. Diese Zeit wird in VHDL als wait for Statement interpretiert. Dadurch kann das zeitliche Verhalten dieser Waveforms beschrieben werden.
- Die Stimulifiles können auch den erwarteten Response enthalten. Dieses Feature kann für automatisiertes Vergleichen mit den Resultaten verwendet werden.
- Die VHDL Testbench muss nicht neu kompiliert werden, um neue/andere Stimuli zu simulieren

3 Testen mit Python

Die Abb. 4 zeigt die Testbenchstruktur beim Einsatz der Skriptsprache Python.

Ein Nachteil von Waves (Abschnitt 2) besteht darin, dass die Stimuli immer nach dem sehr einfachen Muster gestrickt werden müssen. Dadurch wird die Erstellung dieser Pattern von Hand sehr mühsam und fehleranfällig. In Waves wird daher noch die Möglichkeit geboten, alte Werte unverändert zu lassen. Es sind dann nur die Stimuli anzugeben, die neue Werte erhalten.

Der neue Ansatz beim Testen mit Python besteht darin, dass die Stimuli vom Designer in der Highlevel Sprache Python beschrieben werden. Danach wird dieses Stimuli Skript ausgeführt. Dadurch wird ein einfaches Stimulifile generiert, das in etwa mit dem von Waves (Abb. 3) vergleichbar ist.

Der Unterschied besteht darin, dass für jeden Eingangsport eine eigene Zeile erzeugt wird. Zusätzlich wird auch das benötigte Wait Statement in einer eigenen Zeile abgelegt (Abb. 5). Das Format dieser Beschreibung ist sehr einfach. Zuerst steht ein Integer, der ein auszuführendes Kommando für den VHDL Simulator kodiert. Danach folgt noch eine Parameterliste. In diesem Beispiel besteht diese Liste aus jeweils nur einem Parameter.

Durch das gewählte simple Datenformat ist der Parser in VHDL sehr einfach realisierbar (Das VHDL Standard Package textio bietet eine read Funktion an mit der Integer Werte direkt eingelesen werden können). Dieser einfache Parser ist viel einfacher zu implementieren als der Parser des Waves

Systems, dennoch bietet dieser Ansatz größere Flexibilität, weil die Pattern mittels der Highlevel Sprache Python vorbereitet werden.

3.1 Python Stimuli Skript

Die Beschreibung der Stimuli in Python macht dem Designer das Leben nicht schwerer (aufgrund der neuen vielleicht unbekannteren Programmiersprache) sondern einfacher, weil diese Sprache in sehr kurzer Zeit erlernt werden kann (siehe auch PythonTutHomepage, 2003).

Die Vorgehensweise sieht folgendermaßen aus:

- Erstellung der Stimuli mittels eines Python Skripts. Das Skript hat dabei die Aufgabe, die Stimuli Datei mittels Print Statements zu generieren.
- Ausführen des Python Skripts → Einfache Stimuli Datei
- Starten der VHDL Simulation, Einlesen und Interpretieren der Stimuli Datei

Die Abb. 6 zeigt Ausschnitte eines Python Skripts zur Generierung eines Stimulifiles wie in Abb. 5.

- Die Zeilen 1 bis 10 zeigen, wie man die einfachen Kommandos für den VHDL Simulator erzeugen kann. Die Funktion In.Port generiert beispielsweise die Zeile 6 in der Abb. 5.
- Die Funktion WriteDataBlock (ab Zeile 13) zeigt eine Möglichkeit auf, wie man einfache Funktionsaufrufe zur Generierung komplexerer Ansteuersignale einsetzen kann. Der Aufruf dieser Funktion in Zeile 26 generiert die Eingangsstimuli für vier Datenwörter.

3.2 Vorteile beim Einsatz von Python

Python ist eine highlevel objektorientierte interpretierte Skriptsprache. Skriptsprachen (wie Python, Perl, Tcl, Ruby) sind in der Regel dynamisch typisiert und erreichen dadurch einen höheren Abstraktionsgrad als statisch typisierte Sprachen wie Pascal, C, C++, und Java (Ousterhout, 1998). Ausserdem unterstützen die Skriptsprachen einen “Rapid Application Development” Ansatz.

Im Vergleich zu reinen VHDL Testbenches bietet Python unter anderem folgende Vorteile:

- Python ist Objektorientiert
- Gute Unterstützung beim Parsen von Files
- Highlevel Datentypen stehen zur Verfügung (Listen, Files, Strings, Dictionaries etc.)
- Eine große (ständig wachsende) Standardbibliothek (mit guter Dokumentation) kann eingesetzt werden. Unter anderem gibt es frei verfügbare Packages zu folgenden Anwendungsgebieten:
 - Stringverarbeitung

```

1 % CLK DAK IN_PORT R_W AS ADDR_BUS(9) READ_FINISH
2 0 0 00001111 1 0 01010101 1 : 60 ns;
3 0 0 11110000 1 0 01010101 1 : 60 ns;
4 0 0 00110011 1 0 01010101 0 : 60 ns;

```

Abbildung 3. Einfache Stimuli Datei für Waves.

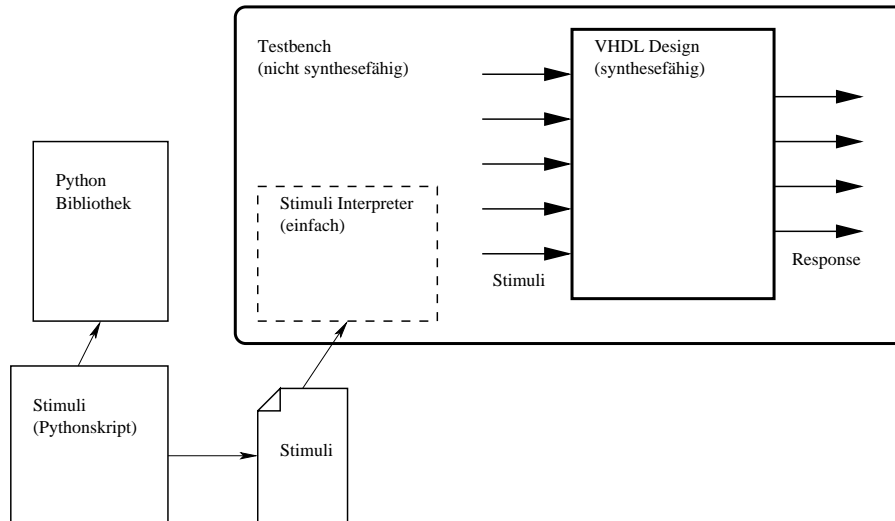


Abbildung 4. Aufbau einer Testbench mit Python.

```

1 % CLK DAK IN_PORT R_W AS ADDR_BUS(9) READ_FINISH
2 % 0 0 00001111 1 0 01010101 1 : 60 ns;
3 % CMD Parameter
4 0 0
5 1 0
6 2 15
7 3 1
8 4 0
9 5 85
10 6 1
11 18 60

```

Abbildung 5. Mittels Python generierte Stimuli Datei (die Stimuli entsprechen der zweiten Zeile der Waves Beschreibung).

```

1 # Declaration of subprograms
2 def Clk(param):
3     print "0_%" % param
4 def Dak(param):
5     print "1_%" % param
6 def In_Port(value):
7     print "2_%" % param
8 # ...
9 def Wait(time):
10    print "18_%" % time
11
12 # Higher level subprograms
13 def WriteDataBlock(block):
14     for data in block:
15         Clk(0)
16         Wait(5)
17         In_Port(data)
18         Wait(15)
19         Clk(1)
20         Wait(20)
21
22 # Stimuli generation
23 Clk(0)
24 In_Port(0xf)
25 #...
26 WriteDataBlock([3,0x20,8,67])

```

Abbildung 6. Python Stimuli Skript.

- Bildverarbeitung
- Numerische Berechnungen
- Zufallszahlen mit verschiedensten Verteilungsfunktionen
- Schnelles Interaktive Austesten von Kommandos möglich (z.B. in einer Python Shell oder im Texteditor Emacs).
- Die eingesetzte VHDL Testbench kann für verschiedene Testfälle (Stimuli Dateien) verwendet werden. Dadurch wird das Problem verhindert, dass man viele ähnliche VHDL Testbenches hat, die man gleichzeitig warten muss.
- Kein Kompilieren notwendig

3.3 Andere Lösungsmöglichkeiten

Für die Generierung der Stimulidaten (wie in Abb. 5) können natürlich auch andere Sprachen wie Perl, Tcl, Ruby, Scheme, etc. eingesetzt werden. Grundsätzlich ist es sinnvoll, eine Sprache zu verwenden, in der man bereits positive Erfahrungen gesammelt hat. Python hat jedoch den Vorteil, dass Skripte in dieser Sprache einfacher lesbar und somit auch einfacher wartbar sind als Skripte in Tcl oder Perl.

Einen anderen Weg zur gekoppelten Simulation von Hardware und Software geht SystemC (SystemCHomepage, 2003). Hier wird nicht versucht, eine Hardwarebeschreibungssprache so zu erweitern, dass auch Softwareelemente eingesetzt werden können. Es wird vielmehr der umgekehrte Weg gegangen, dass eine Softwaresprache so erweitert wird, dass auch Hardware damit simuliert werden kann. Dieser Ansatz bietet folgende Vorteile:

- Bestehende Softwarebibliotheken können eingebunden werden
- Hohe Simulationsperformance

Der Nachteil von SystemC ist jedoch der, dass man sich mit der komplexen Sprache C++ herumschlagen muss. Weiters können vergleichbare Applikationen mit Skriptsprachen in signifikant kürzerer Zeit (mindestens Faktor 2 schneller) als mit statisch typisierten kompilierten Sprachen wie C++ erstellt werden (siehe auch Table 1 in Ousterhout, 1998).

4 Zusammenfassung und Ausblick

Der in diesem Paper gezeigte Ansatz wurde erfolgreich zum Testen eines Asic Designs mit einer Komplexität von 500 000 kGates eingesetzt. Dabei wurden einige Testfiles mit Längen von über 10 000 Zeilen Programmcode erstellt. Der große Vorteil bei der vorgestellten Lösung bestand darin, dass eine VHDL Testbench zur Simulation von mehreren Testfällen (durch einzelne Stimulidateien repräsentiert) verwendet werden konnte.

In weiterer Folge werden wir versuchen, den VHDL Simulator mit einem Python Interpreter zu kopplen, damit die Generierung des Stimulidateien (wie in Abb. 4) nicht mehr notwendig ist. Statt dessen werden die Stimuli während der VHDL Simulation direkt vom in die VHDL Simulation eingebetteten Python Interpreter berechnet.

Literatur

- Beazley, D. M.: Python Essential Reference, New Riders, Indianapolis, Indiana, 2. Auflage, 2001.
- Bergeron, J.: Writing Testbenches, Kluwer Academic Publishers, Boston, Dordrecht, London, 2000.
- Hanna, J. P., Hillman, R. G., Hirsch, G. L., Noh, T. H., und Vemuri: R. R.: Using WAVES and VHDL for Effective Design and Testing. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.
- Ousterhout, J. K.: Scripting: Higher Level Programming for the 21st Century. IEEE Computer magazine, March 1998, <http://home.pacbell.net/ouster/scripting.html>.
- Python Homepage: <http://www.python.org>.
- Python Tutorial: <http://www.python.org/doc/tut>.
- SystemC Community: <http://www.systemc.org>.