Verilog crash course

Steffen Reith, Thorsten Knoll

Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Section 1

Introduction

Introduction

Verilog wurde 1983/1984 zunächst als Simulationssprache entwickelt, von Cadence aufgekauft und 1990 frei gegeben.

Erste Standardisierung 1995 durch die IEEE (Verilog 95). Neuere Version IEEE Standard 1364–2001 (Verilog 2001).

- Syntax vergleichbar mit C (VHDL ist an ADA / Pascal angeleht) mit kompakten Code
- Verbreitet in Nordamerika und Japan (weniger in Europa)
- Kann auch als Sprache von Netzlisten verwendet werden
- Unterstützung durch Open-Source-Tools
- Die Mehrheit der ASICs wird in Verilog entwickelt.
- Weniger ausdruckstark als VHDL (Fluch und Segen)

Die Nähe zu C und Java führt evtl. zu Verwechselungen! Auch in Verilog können z.B. Zeilen, die eine kombinatorische Schaltung beschreiben vertauscht werden.

Verilog ist eine Hardwarebeschreibungssprache!

In diesem Abschnitt legen wir auf auf einen Subset von synthetisierbaren Sprachkonstrukten fest.

Ziel unserer Auswahl sind nicht kommerzielle Tools, sondern offene Entwicklungswerkzeuge wie OpenRoad[1] oder Toolchains für bekannte FPGAs, d.h. wir verwenden auch einige Sprachkonstrukte von SystemVerilog, die durch das Synthesewerkzeug yosys unterstützt werden.

[1] https://theopenroadproject.org/

Literature

- Donald E. Thomas, Philip R. Moorby, Hardware Description Language, Kluwer Academic Publishers, 2002
- Blaine Readler, Verilog by example, Full Arc Press, 2011

Contributions, mentions and license

- This course is a translated, modified and ‘markdownized’ version of a Verilog crash course from Steffen Reith, original in german language.

  https://github.com/SteffenReith

- The initial rework (translate, modify and markdownize) was done by:

  https://github.com/ThorKn

- The build of the PDF slides is done with pandoc:

  https://pandoc.org/

- Pandoc is wrapped within this project:

  https://github.com/alexeygumirov/pandoc-beamer-how-to

- License:

  GPLv3

Section 2

Synthesis tool: Yosys

Synthesis tool: Yosys

Man sollte sich auch mit den Eigenheiten des Synthesetools beschäftigen! Das bekannte Open-Source-Synthesetool yosys schreibt dazu

> Yosys is a framework for VerilogRTLsynthesis. It currently has extensive Verilog-2005 support and provides a ba- sic set of synthesis algorithms for various application do mains. Selected features and typical applications:

- Process almost any synthesizable Verilog-2005 design
- Converting Verilog to BLIF / EDIF/ BTOR / SMT-LIB /simple RTL Verilog / etc.
- …

Section 3

Verilog elements

Structure of a verilog module

```
1  module module_name (port_list);
2  // Definition der Schnittstelle
3  Port-Deklaration
4  Parameter-Deklaration
5
6  // Beschreibung des Schaltkreises
7  Variablen-Deklaration
8  Zuweisungen
9  Modul-Instanzierungen
10
11 always-Bloecke
12
13 endmodule
```

In modernem Verilog können Portliste und Portdeklaration zusammengezogen werden. // leitet einen Kommentar ein.

Example: A linear shiftregister

```
1  module LSFR (
2
3  input  wire   load,
4  input  wire   loadIt,
5  input  wire   enable,
6  output wire   newBit,
7  input  wire   clk,
8  input  wire   reset);
9
10 wire       [17:0]   fsRegN;
11 reg        [17:0]   fsReg;
12 wire                taps_0, taps_1;
13 reg                 genBit;
14
15 assign taps_0 = fsReg[0];
16 assign taps_1 = fsReg[11];
17
18 always @(*) begin
19   genBit = (taps_0 ^ taps_1);
20   if(loadIt) begin
21     genBit = load;
22   end
23 end
```

```
1  assign newBit = fsReg[0];
2  assign fsRegN = {genBit,fsReg[17 : 1]};
3
4  always @(posedge clk) begin
5    if(reset) begin
6      fsReg <= 18'h0;
7    end else begin
8      if(enable) begin
9        fsReg <= fsRegN;
10     end
11   end
12 end
13
14 endmodule
```

Dabei geben **input** und **output** die **Richtung des Ports** an.

Constants and Operators

Es gibt vier Werte für eine Konstante / Signal:

- 0 oder 1
- X bzw. x (unbekannt)
- Z bzw. z (hochohmig)

Man kann die Breite von Konstanten angeben:

- Hexadezimalkonstante mit 32 Bit: 32'hDEADBEEF
- Binärkonstante mit 4 Bit: 4'b1011
- Zur besseren Lesbarkeit kann man auch den Underscore verwenden: 12'b1010_1111_0001

Zur Spezifikation der Zahlenbasis sind

- b (binär)
- h (hexadezimal),
- o (oktal)
- d (dezimal) zulässig.

Der Default ist dezimal (d) und die Bitbreite ist optional, d.h. 4711 ist eine zulässige (dezimal) Konstante.

Passend zu den Konstanten existiert eine Array-Schreibweise:

- wire [7:0] serDat;
- reg [0:32] shiftReg;
- Einzelne Bits können gesliced werden:
  - serDat[3 : 0] (low-nibble)
  - serDat[7] (MSB).
- {serDat[7:6], serDat[1:0]} notiert die Konkatenation.
- Bits können repliziert und in ein Array umgewandelt werden, d.h. {8{serData[7 : 4]}} enthält acht Kopien des high-nibble von serDat und hat eine Breite von 32.

Weiterhin existieren die üblichen arithmetischen Operationen, Ordnungsrelationen, Äquivalenzen und Negation:

- a + b, a - b, a * b, a / b und a % b
- a > b, a <= b, und a >= b
- a == b und a != b,
- !(a = b)

Achtung: Kommen x oder z vor, so ermittelt der Simulator bei einem Vergleich false. Will man dies vermeiden, so existieren die Operatoren === und !==. Also gilt:

```
1 if (4'b110z === 4'b110z)
2 // not taken
3 then_statement;
```

```
1 if (4'b110z == 4'b110z)
2 // not taken
3 then_statement;
```

Es existieren auch boolesche Operationen wie gewohnt:

bitweise Operatoren: & (UND), | (ODER, ~ (NICHT), ^ (XOR) und auch ~^ (XNOR)

logische Operatoren: && (UND), || (ODER) und ! (NICHT)

Schiebeoperation: a « b (schiebe a um b Stellen nach links) und a » b (verschiebe a um b Positionen nach rechts). Eine negative Anzahl b ist nicht zulässig, leere Stellen werden mit 0 aufgefüllt.

Parameters (old style)

Um Designs leichter anpassen zu können, bietet Verilog die Verwendung von Parametern an.

```
1  module mux (
2    in1, in2,
3    sel,
4    out);
5
6    parameter WIDTH = 8;  // Anzahl der Bits
7
8    input  [WIDTH - 1 : 0] in1, in2;
9    input sel;
10   output [WIDTH - 1 : 0] out;
11
12   assign out = sel ? in1 : in2;
13
14 endmodule
```

Instances and structural descriptions

Beschreibt man einen Schaltkreis durch seine (interne) Struktur oder soll ein Teilschaltkreis wiederverwendet werden, dann wird eine Instanz erzeugt und verdrahtet.

```
1 module xor2 (
2   input  wire a,
3   input  wire b,
4   output wire e);
5   assign e = a ^ b;
6 endmodule
```

```
1  module xor3 (
2    input  wire a,
3    input  wire b,
4    input  wire c,
5    output wire e);
6
7    wire tmp;
8    xor2 xor2_1 // Instanz 1
9      (
10       .a(a),
11       .b(b),
12       .e(tmp)
13     );
14   xor2 xor2_2 // Instanz 2
15     (
16       .a(c),
17       .b(tmp),
18       .e(e)
19     );
20
21 endmodule
```

Code for sequential circuits

Wie besprochen übernimmt ein Flipflop die Eingabe an den steigenden oder fallenden Flanken des Taktes. Dafür wird die Ereignissteuerung mit dem @-Symbol und always-Blöcken verwendet:

```
1  module FF (input  clk,
2             input  rst,
3             input  d,
4             output q);
5    reg q;
6
7    always @ ( posedge clk or
8               posedge reset)
9      begin
10       if ( rst )
11         q <= 1'b0;
12       else
13         q <= d;
14     end
15 endmodule
```

Die Signalliste hinter dem @ heißt Sensitivity-List. Der reset wird synchron, wenn man or posedge reset entfernt.

Section 4

A few basic circuits in verilog

Combinational circuits

Sequential circuits

Section 5

Selected features in verilog

Parameterized Hardware

The preprocessor

Yosys and Systemverilog