VLSI 1 - Notes Week 3

Ruben Schenk, ruben.schenk@inf.ethz.ch

January 10, 2022

The FPGA fabric includes embedded memory elements that can be used as random-access memory (RAM), read-only memory (ROM), or shift registers. These elements are block RAMs (BRAMs), LUTs, and shift registers.

- Using LUTs as SRAM is called *distributed RAM.*
- Included dedicated RAM components in the FPGA fabric are called *block RAM.*

The **I/O PADs** connect the signals from the PCB to the internal logic. The **IOBs** are organized in banks. All the PADs in the same bank share a common supply voltage. The I/O Blocks (IOB) support a wide range of commercial standards, both single ended and differential.

Chapter 2: SystemVerilog Design – Introduction and Basics

2.1 Overview

During the next few lectures we will:

- Learn to describe digital hardware efficiently using HDL:
  - Combinational circuits
  - Sequential building blocks and Finite State Machines
  - Rely on regular and simple structures
- Be able to program FPGAs to realize digital circuits
  - This will be valuable in many fields
  - You will be able to use what you learn also to design ASIC circuits
- Learn how to verify the correct functionality of these circuits
  - Writing testbenches in HDL

Hardware Description Languages (HDL) is not *really* a programming language, however it looks a lot like one. HDL was developed to simplify circuit schematics.

We quickly compare VHDL and SystemVerilog:

VHDL	SystemVerilog
Case-insensitive	Case-sensitive
9 different logic values	4 different logic values
More verbose	Compact code
Strong type checking	Not much checking
More popular in Europe	More popular in US

For this lecture we will be using SystemVerilog. Both of them are very similar, i.e. key concepts are identical.

2.2 Drawing Circuit Schematics

2.2.1 `module`

A **module** is defined by the *name* of the block, some optional *parameters*, and some *signal names* with a direction and a type.

```
module top #(parameter int Width = 16) (
    input   logic               clk_i,
    input   logic               rst_ni,
    input   logic               mode_i,
    input   logic [Width-1:0]   data_in_i,
    output  logic [Width-1:0]   result_o
);
```

Remark: Some notes on good practices using HDL:

- Always use one file per SystemVerilog module
- Make sure that the file and the module are named the same
- SystemVerilog is case-sensitive, try to stick to one way of describing inputs/outputs/logic etc.

2.2.2 body

The **body** describes the function. We declare **local wires** to use for internal connections.

```
// Declare signals to be used in the module
logic [Width-1:0] first, second;
logic [Width-1:0] combine_and, combine_or;
```

2.2.3 instantiation

One can include other modules in his own module. This allows for some level of hierarchy. Below is a code example where two instances of the component `ffs`, called `i_reg_1` and `frank`, are instantiated. These are described in some other module.

```
// Instantiate two blocks with different names i_reg_1 and frank
ffs #(.Width(Width)) i_reg_1 (
    .clk_i(clk_i), .rst_ni(rst_ni), .in_i(data_in_i), .out_o(first));
ffs #(.Width(Width)) frank (
    .clk_i(clk_i), .rst_ni(rst_ni), .in_i(first), .out_o(second));
)
```

The module we instantiated (`ffs`) will be described separately. One must make sure that the module description and the instantiation match.

2.2.4 Comments

SystemVerilog uses C++ style comments, i.e. everything after `//` is a comment. One could also use `/*...*/`, but we do not recommend that.

2.3 More on HDL

What we have look at so far is called **structural HDL.** The circuit includes other circuits which are interconnected. There is no other information than interconnections and instantiations.

A **netlist** is a structural HDL that instantiates only library components. The library components are simple pre-designed circuits with physical properties. Once we have a circuit netlist, it has a corresponding physical form, and it can be manufactured.

Additional benefits of HDL came much later:

- *Simulation:* It was soon discovered that we could also see how a digital circuit described in HDL is working.
- *Synthesis:* Logic synthesis algorithms can map the HDL description into a library of common logic gates such as AND, OR gates using boolean algebra resulting in a *gate-level netlist.*

2.4 Data Type of HDL

2.4.1 Logic Values

In SystemVerilog with have 4 different **logic values:**

- *0:* logic low value
- *1:* logic high value
- *X:* used for a) a drive conflict, i.e. when a wire is driven by two sources at the same time, b) when we do not really know the value of a wire, and c) if we do not care what the input is
- *Z:* used as the inactive logic state, i.e. when we do not drive the output.

2.4.2 `logic`

The main data type we use in SystemVerilog is **logic.** If we are defining wires, inputs, outputs, signals and constants we use `logic`. If a constant is only used in one module, we can declare them as `localparam`. We can use the `assign` statement to connect these signals/constants.

```
module top (
    input   logic data_in_i,
    output  logic results_ready_o
);

    logic   first, second;
    logic   third;
    localparam ZERO         = 1'b0;         // constant definition

    assign first            = data_in_i;    // this connects first to data_in_i
    assign results_ready_o  = third;        // connections represent electrical wires
    assign third            = first;        // the order you define them is not important
    assign second           = ZERO;         // assign a constant value to a wire
```

2.4.3 `wire/reg`

One annoying exception are **tristate wires.** These need to be `wire` or `reg`.

Tristate wires are wires that can be both written to and ream from, they cannot be `logic`.

2.4.4 `logic` Arrays

If we need more than one bit, `logic` can also be an array. Arrays can be as large as one wants, the range can be specified as one likes to. WE can define the array from MSB to LSB, wehich we call the *standard packed format.* An array can also have multiple dimensions: `logic [15:0][7:0] varname`

```
logic   [7:0] eight_bit_bus;
logic   [15:0] big bus;
logic   [0:0] tiny_bus;
logic   fist, second, third;

assign eight_bit_bus = 8'b0000_0000;    // assigning a constant
assign big_bus[15:8] = eight_bit_bus;   // partial assignment
assign first         = big_bus[3];      // picking out single bits
```

2.4.5 Packed vs. Unpacked Arrays

In *packed* arrays, dimensions are declared after the type and before the data identifier name: `logic [15:0] some_wire_name` They are ordered from MSB to LSB.

In *unpacked* arrays, dimensions are declared after the data identifier: `int some_name [0:15]` They can be ordered in both ways, but ETH *insists* on ordering from LSB to MSB. The shorthand `int some_name [8]` is equivalent to `int some_name [0:7]`. They can be used with any type, but ETH *insists* that they are not used for logic types.

2.4.6 Accessing and Assigning Vectors

We use `{ }` to combine vectors together:

```
assign y = {a[2], a[1], a[0], a[3]}
assign x = {a[0], a[0], a[0], a[0]}
assign x = {4{a[0]}}                // It is possible to define multiple copies
```

2.4.7 `typedef`

SystemVerilog gives us great flexibility by allowing us to define new subtypes using `typedef`.

- This causes a lot of problems.
- Consistent use of coding styles is very important in debugging.

```
typedef logic [7:0] u8_t;
logic         [31:0] u32_word;
u8_t          [1:0] u16_word;
u8_t          byte3, byte2, byte1, byte0;

assign u16_word = {byte1, byte0};
assign u32_word = {byte3, byte2, u16_word};
```

2.4.8 Numbers

Numbers in SystemVerilog are expressed in the form `N'Bxx`, e.g. `8'b0000_0001`. In detail:

- `N` is the number of bits.
- `B` is the base and can be `b` (binary), `h` (hexadecimal), `d` (decimal), or `o` (octal)
- `xx` is the number, i.e. the value expressed in the chosen base. Apart from numbers this can also include `X` and `Z` as values.

2.5 Naming Conventions

The basic idea of the **naming conventions** used at IIS is to add a `_` and a regular suffix to identify special signals:

Type	Suffix	Description
inputs	`_i`	Signals that are declared input in the module
outputs	`_o`	Signals that are declared output in the module
types	`_t`	Type definitions
reset	`_r`	Asynchronous reset used for flip-flops and latches
clock	`_c`	Clock signals for flip-flops and latches
active low	`_n`	Signals that are active when they have the value 0

2.6 Simple Implementations

2.6.1 Basic Logic Functions

```
assign y0 =  ~a;        // NOT
assign y1 =   a & b;    // AND
assign y2 =   a | b;    // OR
assign y3 =   a ^ b;    // XOR
assign y4 = ~(a & b);   // NAND
assign y5 = ~(a | b);   // NOR
```

2.6.2 Simple Multiplexers

Multiplexers are very common building blocks in digital design.

```
module mux2(
    input  logic [3:0] data0_i, data1_i,
    input  logic       select_i,
    output loigc [3:0] result_o
);

assign result_o = select_i ? data1_i : data0_i;
```

Remark: In SystemVerilog we can also use the **ternary operator:** `assign value = condition ? true : false;`

Multi-level multiplexers can also be defined:

```
module mux4(
    input  logic [3:0] data0_i, data1_i, data2_i, data3_i,
    input  logic [1:0] select_i,
    output logic [3:0] result_0
);

assign result_o = (select_i == 2'b11) ? data3_i :
                  (select_i == 2'b10) ? data2_i :
                  (select_i == 2'b01) ? data1_i :
                  data0_i;
```

2.6.3 Bitwise vs. Logical Operators

Bitwise operators are of the form `~`, `&`, `|`, `^`, `=`. They will work on all bits of a vector and result in a vector of the same size. They are used to implement simple boolean functions.

Logical operators are of the form `!`, `&&`, `||`, `==`, `!=`. They will generate a true or false results and are used as a condition in, for example, ternary operators.