ECE4700J Lab 2 - A Pipelined Integer Square Root Module

Mingjian Li

UM-SJTU JI

lmjhshxhc@sjtu.edu.cn

May 27, 2024

Overview

Lab Logistics

Lab2 Logistics

- Due: 23:59 at June 4th (Beijing time)
- Contents:
    - Pipeline multiplier
    - Integer square root module

Tips and Advice

- Iverilog is an good tool for systemVerilog simulation if you are a little tired of vivado
- If error happens in post-synthesis simulation and you think your code is perfect, try to enlargen the clock period to at least 100ns
- Start early, this lab could be much more time consuming than Lab 1

Advanced SystemVerilog Usage

SystemVerilog Macros - Definition

- Good practice to define constants, e.g. data width, address width...
- Initialize: `` `define <name> <value> ``
  - `` `define HIGH 32'hffff ``
- Usage: `` `<name> ``
  - assign count = `` `HIGH; ``
- Can also `` `undef <name> ``

SystemVerilog Macros - Flow Control

- Conditionally define constants
- `` `ifdef ``: Checks if something is defined
  `` `else ``: Normal else behavior
  `` `endif ``: End the if
  `` `ifndef ``: Checks if something is not defined

SystemVerilog Macros - Header Files

- Include the header files like in C/C++
- Syntax: `` `include <filename> ``
- Commonly used combination:

```
`ifndef __FILENAME_H__
`define __FILENAME_H__
.
.
`endif
```

Parameters

- Functions:
  - Define constants inside a module
  - Used to set module properties
  - Can be overwritten on instantiation
- Example:

  module xx {input a, output logic b};

  `parameter PARAMETERA = 8`

  xxxxxxxxxxxxxxxxxxxxxxx

  endmodule
- Overwritten:
  - `module_name #(.PARAMETERA(10)) instance1(...)`
  - `defparam instance1.PARAMETERA = 10`

Defining multiple instances like the following?

```
module_name inst0 (...)
module_name inst1 (...)
module_name inst2 (...)
module_name inst3 (...)
module_name inst4 (...)
module_name inst5 (...)
```

Bad idea:(

Better way of Multiple Instances

```
module eight_bit_adder (
    input en,
    input cin,
    input [7:0] a, b,
    output [7:0] sum,
    output [6:0] cout
);
    wire [6:0] carries;
    one_bit_adder adder [7:0] (
        .en(en), .a(a), .b(b), .cin({carries, cin}),
        .sum(sum), .cout({cout, carries})
    );
endmodule
```

Loops

- Loops in SystemVerilog will be unrolled during run time
- Loops are valid and synthesizable if loop number is constant
- It can replace the repetitive part in your code, but only for combinational logic not sequential logic.
- Blocking assignment in loops:

```
always_comb begin
    for (int i=0; i<32; i++) a = i;
end
```

We will have `a == 31` finally as `i = 31` will be the last loop after unrolling.

Generate Blocks

- Use a generate block to build the hardware

```
generate
    genvar i;
    for (i=0; i<N; i++) begin
        one_bit_adder (
            .a(a[i]), .b(b[i]),
            .cin(carries[i]),
            .sum(sum[i]),
            .cout(carries[i+1])
        );
    end
endgenerate
```

- The tool will “elaborate” the design
- It will evaluate “if” statement and unroll “for” loops

Thoughts on final project?

Final project grouping is coming this week, get ready:).

- out of order execution?
- superscalar pipeline?
- advanced pipeline?

Thank you!