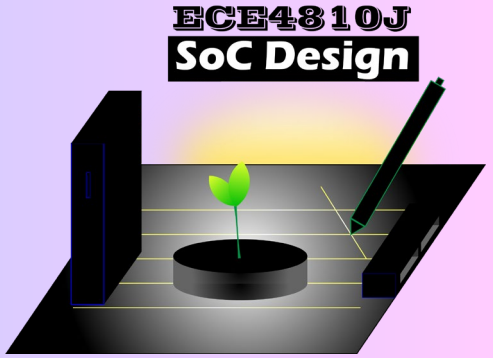
Lab #2
Introduction to Vitis HLS & PYNQ

ECE4810J System-on-Chip Design



Yihua Liu
UM-SJTU Joint Institute
ayka_tsuzuki@sjtu.edu.cn
Sept 26, 2022

Overview

Verilog 2005 Revisited

JOINT INSTITUTE 交大密西根学院

Structural Model

```
module VrInhibit(in, invin, out);
    input in, invin;
    output out;
    wire notinvin;
    not U1(notinvin, invin);
    and U2(out, in, notinvin);
endmodule
```

Dataflow Model/Description

```
module Vrprimed(N, F);
    input [3:0] N;
    output F;
    assign F = N[3] ?
        (N[0] & (N[1]^N[2])) :
        (N[0] | (~N[2]&N[1]));
endmodule
```

Behavioral Model/Description

```
module Vrbytecase(A, B, C,
        sel, Z);
    input [7:0] A, B, C;
    input [1:0] sel;
    output reg [7:0] Z;
    always @ (*)
      case
        2'd0: Z = A;
        2'd1: Z = B;
        2'd2: Z = C;
        2'd3: Z = 8'b0;
        default: Z = 8'bx;
      endcase
endmodule
```

Verilog built-in gates:

and	xor	bufif0
nand	xnor	bufif1
or	buf	notif0
nor	not	notif1

Continuous-assignment statements:

assign *net-name* = *expression;*
assign *net-name*[*bit-index*] = *expression*;
assign *net-name*[*msb*:*lsb*] = *expression*;
assign *net-concatenation* = *expression*;

Instance statements:

component-name instance-identifier(*expr*,...);
component-name instance-identifier(
.*port-name*(*expr*),...);

Synthesis

Procedural statements: assignment, `begin-end` blocks, `if`, `case`, `while`, and `repeat`

Verilog 2005 Revisited

Verilog signals: nets and variables

Nets: a wire in a physical circuit and provides connectivity between modules and other elements in a Verilog structural model

```
wire, tri, triand, trior, tri0, tr1, trireg, wand, wor, supply0, supply1, uwire
```

Variables: store values during a Verilog model's execution and need not have physical significance in a circuit.

```
reg, integer, real, realtime, time
```

A reg is NOT A FLIP-FLOP

Nets vs. variables: a variable's value can be changed only by procedural code within a module; it cannot be changed from outside. Procedural code can assign values only to variables, not nets.

Signed arithmetic: `signed`

```
reg signed [15:0] A, output reg signed [15:0] T
```

Signed operations are used only if all of an expression's operands are signed

```
4'sb1101 + 1'b1 = 14, (4'sb1101 << 1) + 1 = -3
```

Type-casting functions: `$signed(), $unsigned()`

Verilog 2005 Revisited

JOINT INSTITUTE
交大密西根学院

An **array** is an ordered set of variables of the same type indexed by array index

reg *identifier* [*start*:*end*];
reg [*msb*:*lsb*] *identifier* [*start*:*end*];
integer *identifier* [*start*:*end*];
wire *identifier* [*start*:*end*];
wire [*msb*:*lsb*] *identifier* [*start*:*end*];

Multi-dimensional arrays

```
reg [7:0] mem3 [1:10] [0:255]
```

The lower nibble of the byte in row 5, column 7: `mem3[5][7][3:0]`

One-dimensional array of vectors vs. two-dimensional array of bits:

```
reg [7:0] mem1 [0:255]          vs. reg mem2 [0:255] [7:0]
mem1[1] = 0;          // Assigns 0 to the second element of mem1
mem2[1][0] = 0;       // Assigns 0 to the bit referenced by indices [1][0]
mem1 = 0;             // Illegal - Attempt to write to entire array
mem2[1] = 0;         // Illegal - Attempt to write to elements [1:0]-[1:255]
mem2[1][1:3] = 0; // Illegal - Attempt to write to elements [1:1]-[1:3]
```

Verilog 2005 Revisited

Function	Task
execute in one simulation time unit	can contain time-controlling statements
cannot enable a task	can enable other tasks and functions
have at least one input type argument not have an output or inout type argument	can have zero or more arguments of any type
shall return a single value	shall not return a value

A function/task can be defined only within a module, i.e., its definition is local to the module.
To use in multiple modules, define it by itself in a file and use `` `include `` to include it in each module.
A useful example:

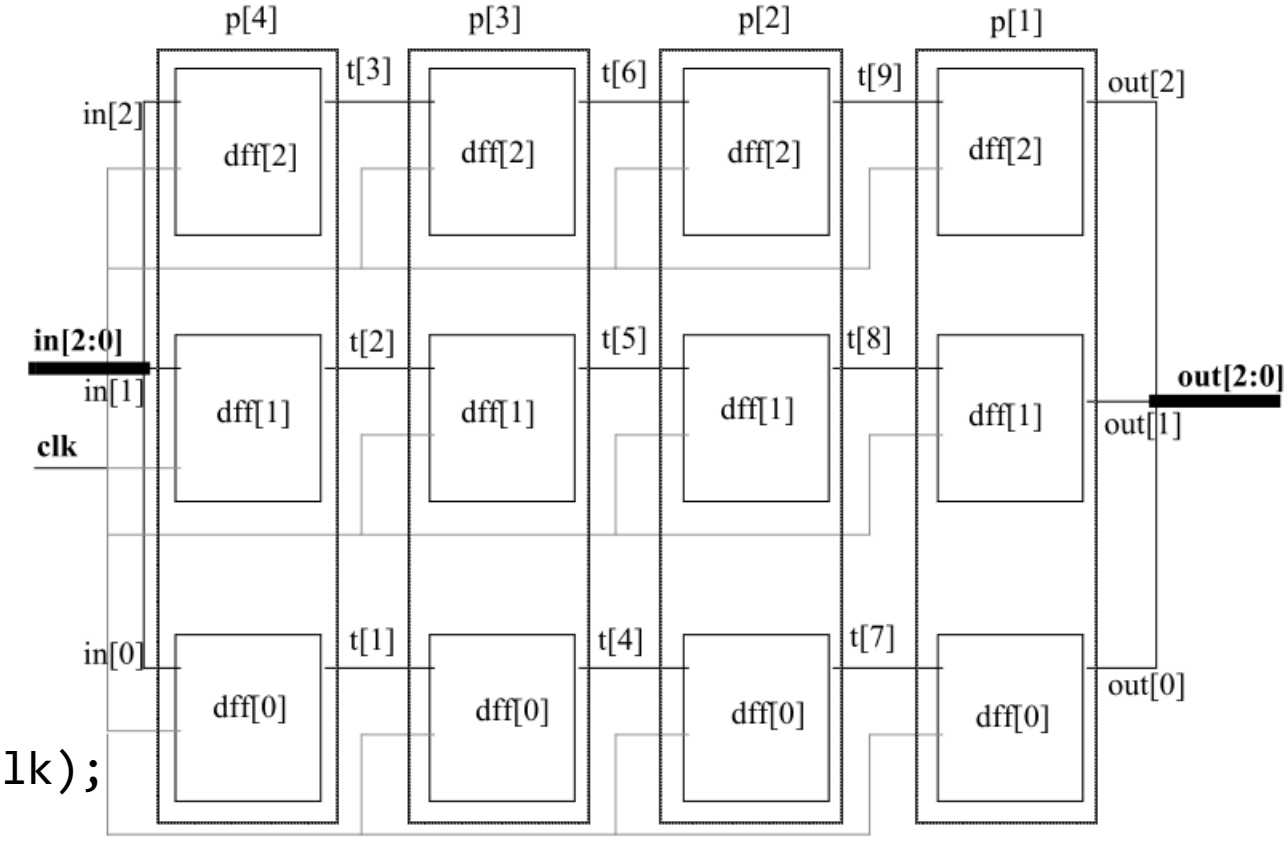
```
function integer clogb2;
    input [31:0] value;
    begin
        value = value - 1;
        for (clogb2 = 0; value > 0; clogb2 = clogb2 + 1)
        value = value >> 1;
    end
endfunction
```

Verilog 2005 Revisited

Array of Instances

To specify an array of instances, the instance name shall be followed by the range specification `[lhi:rhi]`, which shall represent an array of `abs(lhi-rhi)+1` instances. This example demonstrates how a series of modules can be chained together.

```
module dffn(q, d, clk);
    parameter bits = 1;
    input [bits - 1:0] d;
    output [bits - 1:0] q;
    input clk;
    DFF dff[bits - 1:0] (q, d, clk);
endmodule
module MxN_pipeline(in, out, clk);
    parameter M = 3, N = 4;  // N = depth
    input [M - 1:0] in;  // M = width
    output [M - 1:0] out;
    input clk;
    wire [M * (N - 1):1] t;
    dffn #(M) p[1:N] ({out, t}, {t, in}, clk);
endmodule
```

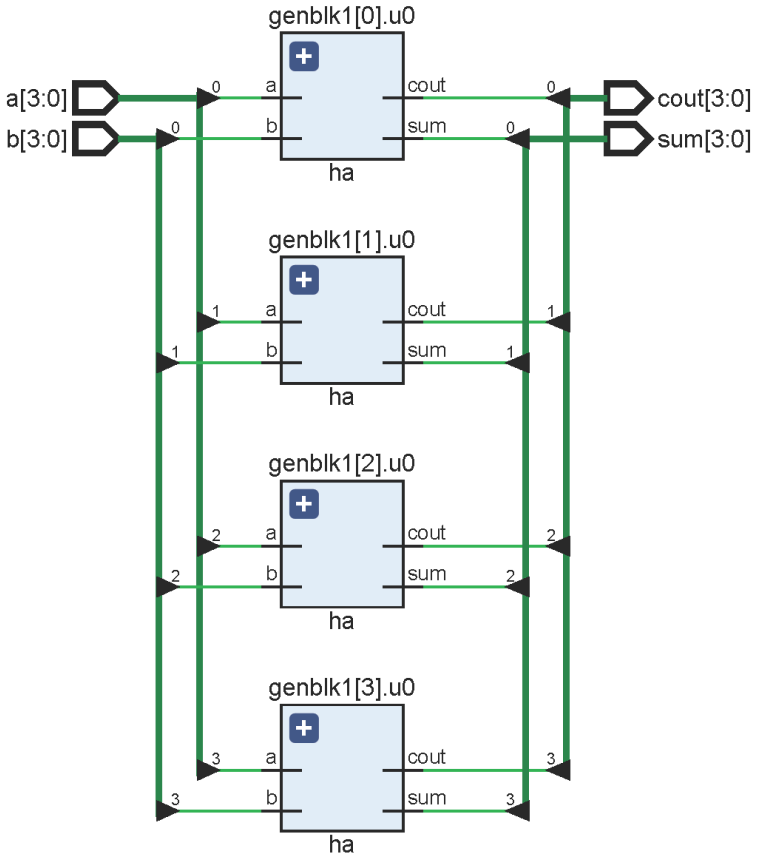


Verilog 2005 Revisited

Generate Blocks

Useful if you want to create multiple copies of a particular structure within a model
Within the block, `if`, `case`, and `for` statements may be used
If instances generated in iterative for loops, the loop is controlled by integer `genvar`
Example:

```
module ha(input a, b, output sum, cout);
    assign sum = a ^ b;
    assign cout = a & b;
endmodule  // Design for a half-adder
module my_design #(parameter N = 4)
       (input [N - 1:0] a, b, output [N - 1:0] sum, cout);
    genvar i;  // Generate a temporary loop variable
    generate  // Generate for loop to instantiate N times
        for (i = 0; i < N; i = i + 1) begin
            ha u0(a[i], b[i], sum[i], cout[i]);
        end
    endgenerate
endmodule  // Top level design that contains N inst of ha
```



Case Statement

Case statement with do-not-cares:

casez

```
reg [7:0] ir;
casez (ir)
    8'b1???????: instruction1(ir);
    8'b01??????: instruction2(ir);
    8'b00010???: instruction3(ir);
    8'b000001??: instruction4(ir);
endcase
```

casex

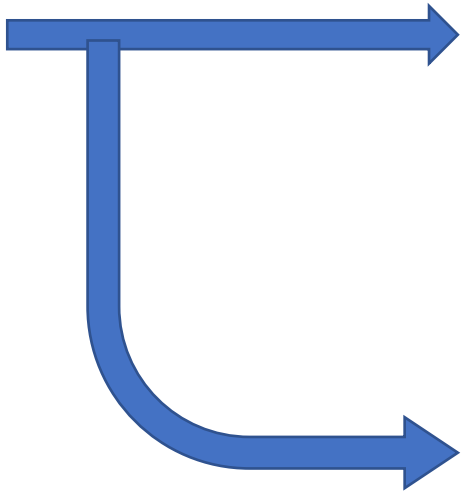
```
reg [7:0] r, mask;
mask = 8'bx0x0x0x0;
casex (r ^ mask)
    8'b001100xx: stat1;
    8'b1100xx00: stat2;
    8'b00xx0011: stat3;
    8'bxx010100: stat4;
endcase
```

Verilog 2005 Revisited

JOINT INSTITUTE 交大密西根学院

Latch: Memory device without a clock

```
always_comb begin
    if (cond)
        next_x = y;
end
```



```
always_comb begin
    next_x = x;
    if (cond)
        next_x = y;
end
```

```
always_comb begin
    next_x = x;
    if (cond)
        next_x = y;
    else
        next_x = x;
end
```

SystemVerilog 2017

Verilog	SystemVerilog
`wire`	`wire / logic`
`reg`	`logic`
`always (*)`	`always_comb`
`always (posedge clock)`	`always_ff @(posedge clock)`

An enumerated type declares a set of integral named constants.

```
enum bit [3:0] {bronze='h3, silver, gold='h5} medal2;
```

A type name can be given so that the same type can be used in many places.

```
typedef enum { red, green, blue, yellow, white, black } Colors;
Colors C;
```

SystemVerilog 2017

How to prevent unintentional latches?

unique, unique0, priority

`unique-if` and `unique0-if` assert that there is no overlap in a series of if–else–if conditions, i.e., they are mutually exclusive and hence it is safe for the conditions to be evaluated in parallel.

```
module lab2_1;
    int a, b, c;
    initial begin
        a = 50;
        b = 20;
        c = 40;
        unique0 if (a < b)
            $display("a<b");
        else if (a < c)
            $display("a<c")
    end
endmodule
```

Doesn't issue a *violation report*

g-[RT-NCMUIF]
dition matches in
ent

A `unique-if` or `unique0-if` is violated if more than one condition is found true.

```
always_comb begin
    {int2, int1, int0} = 3'b000;
    unique casez (irq)
        3'b1??: int2 = 1'b1;
        3'b?1?: int1 = 1'b1;
        3'b??1: int0 = 1'b1;
    endcase
end
```

If the keywords `unique` or `priority` are used, a violation report shall be issued if no condition matches unless there is an explicit `else`.

unique, unique0, priority

A `priority-if` indicates that a series of if–else–if conditions shall be evaluated in the order listed.

```
module lab2_2;
    bit [7:0] a;
    initial begin
        a = 0;
        priority case (a)
            0: $display("Fount to be 0");
            0: $display("Again fount to be 0");
            2: $display("Fount to be 2");
        endcase
    end
endmodule
```

```
ncsim > run
Found to be 0
ncsim: *W,RNQUIE: Simulation is complete.
```

Synthesis Isn't for Granted

Vivado Design Suite User Guid 2022.1)

Verilog Language Support: log Support:

From ISE to Vivado 2021.2
Division: supported only if 2nd operand is a power of 2 or both operands are constant
Modulus: supported only if 2nd operand is a power of 2

But, at what cost?

Expression	Symbol	Status
Arithmetic	+, -, *, **	Supported
Division	/	Supported
Modulus	%	Supported
Power	**	Supported: No real or X or Z 2 const, 2nd non-negative 1st is 2, 2nd can be a var

Verilog Constants	Support Status
tri1, tr0, trireg	Unsupported
delay (#)	Ignored
`timescale	Ignored
$fdisplay, $finish, $fopen	Ignored
$fclose, $fgets	Not Supported

truct	Status
e data types	Supported
4-state data types	Supported
Operat ors & ions $error	Supported only within initial blocks, only be used to evaluate constant expressions like parameters
$warning	
$info	
Increment operators	Supported
Decrement operators	Supported
Functio ns Void functions	Supported
Virtual functions	Not Supported
Compiler directives	Supported
Generate constructs	Supported
Class Instance, constructor	Supported

Pipelined Multiplication

Partial Products

- Multiply the first *n* bits of the two components
- Multiply the next *n* bits, etc.
- Sum the partial products to get the answer

Example: 4-stage Pipelined Multiplication

```
    multiplicand: 00001011   << 2
      multiplier: 00000111   >> 2
partial product: 00100001
    multiplicand: 00101100   << 2
      multiplier: 00000001   >> 2
partial product: 00101100
                + 00100001
                = 01001101
```

Pipelined Multiplication

Various choices:

Generate blocks	vs.	Array instantiation
Number of stages	vs.	Number of bits
Combinational logic	vs.	Sequential logic
Shift operators	vs.	Concatenation

Reference

① IEEE. “1364-2005 IEEE Standard for Verilog® Hardware Description Language.”
② IEEE. “1800-2017 IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language.”
③ John F. Wakerly. *Digital Design Principles and Practices Fifth Edition*.
④ Haoyang Zhang. “ECE4700J Summer 2022 Computer Architecture Lab 1 Verilog: Hardware Description Language”. University of Michigan – Shanghai Jiao Tong University Joint Institute. May 18, 2022.
⑤ https://support.xilinx.com/s/question/0D52E00006hpiTjSAI/does-vivado-ise-support-division-operator-for-synthesize. July 27, 2018.
⑥ Xilinx. Vivado Design Suite User Guide Synthesis UG901 (v2022.1). June 6, 2022.
⑦ Runxi Wang and Xinfei Guo. “[Lab 1] Timing Issue of divider IP.” Sept. 21, 2022. URL: https://piazza.com/class/l7nfec1tage34/post/7.
⑧ https://support.xilinx.com/s/question/0D52E00006lLgwMSAS/vhdl-divider-inference-in-vivado-synthesis-20191. Sept. 9, 2019.

Reference

⑨ https://support.xilinx.com/s/question/0D52E00006hpXuGSAU/multiplication-sign-and-division-sign-can-pass-synthesis-and-implementation-in-vivado.

⑩ ChipVerify. “SystemVerilog Unique Priority Case.”

⑪ Xilinx. “PG151 Divider Generator v5.1 LogiCORE IP Product Guide.” Feb. 4, 2021.