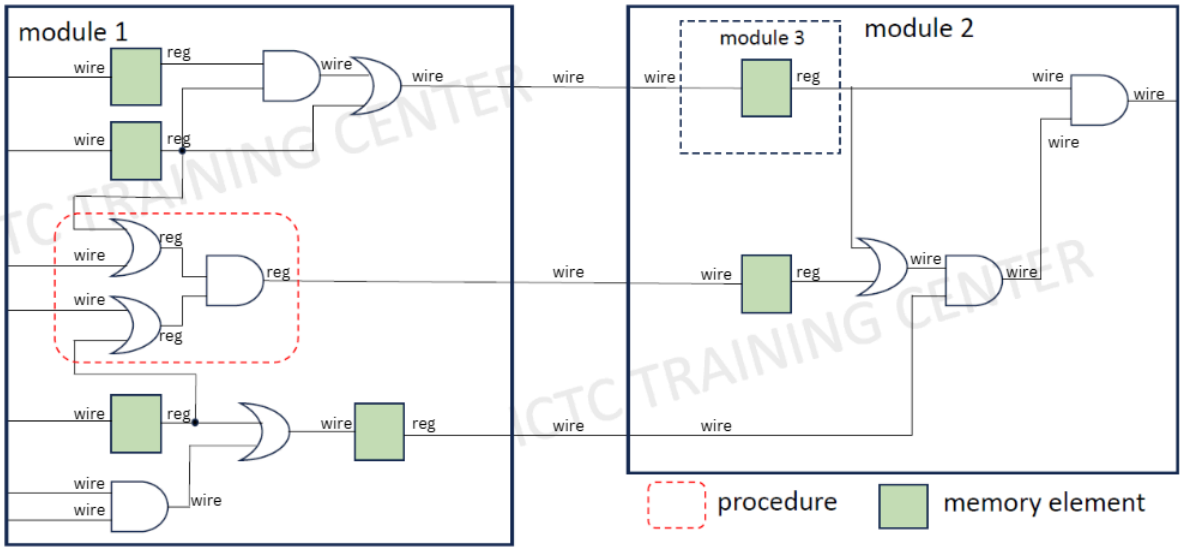
Homework1: identify data type wire/reg for below diagram.
Note:
+ input of module is always wire type
+ output of module can be wire/reg type

Bài làm



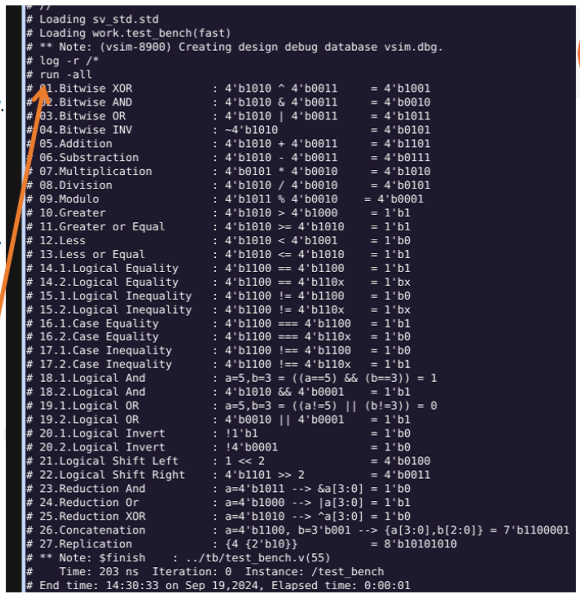
Homework 2:

- Copy */ictc/student_data/share/ico/05_ss5/* to your home directory.
- Modify the *tb/test_bench.v* based on the requirement.
- Take a snapshot of the homework result after running and submit it through your form. The mentor will check your code in your directory.
- *Your can replace the Makefile by your Makefile in ss4/homewor2*

Homework2(*): Perform all the Verilog operator examples from today's session. Check the result of the testbench and ensure they match the results in our slides.
The output of the testbench should be similar to the picture on the right.

Note: this is the advanced homework, you need to investigate how to write the code to perform those operations.
Detail of how to write testbench will be showed in later sessions.

Operator	Name	Example
^	Bitwise XOR	4'b1010 ^ 4'b0011 → 4'b1001
&	Bitwise AND	4'b1010 & 4'b0011 → 4'b0010
|	Bitwise OR	4'b1010 | 4'b0011 → 4'b1011
~	Bitwise invert	~4'b1010 → 4'b0101



Bài làm

```
huugiap@ictc-eda-ldap:~/05_ss5/tb$ cat test_bench.v
module test_bench;

  reg [3:0] a, b;
  reg [7:0] c;
  reg [1:0] d;
  reg a_s,b_s;
  wire [3:0] result_xor, result_and, result_or, result_inv;
  wire [3:0] result_add, result_sub, result_mul, result_div, result_mod;
  wire result_greater, result_ge, result_less, result_le;
  wire result_log_eq1, result_log_eq2, result_log_neq1, result_log_neq2;
  wire result_case_eq1, result_case_eq2, result_case_neq1, result_case_neq2;
  wire result_log_and1, result_log_and2, result_log_or1, result_log_or2;
  wire result_log_inv1, result_log_inv2;
  wire [3:0] result_shift_left, result_shift_right;
  wire result_red_and, result_red_or, result_red_xor;
  wire [7:0] result_concat;
  wire [7:0] result_replic;


  initial begin
    // Bitwise XOR
    a = 4'b1010;
    b = 4'b0011;
    #5;
    $display("01.Bitwise XOR         : 4'b1010 ^ 4'b0011 = %b", a ^ b);

    // Bitwise AND
    #5;
    $display("02.Bitwise AND         : 4'b1010 & 4'b0011 = %b", a & b);

    // Bitwise OR
    #5;
    $display("03.Bitwise OR          : 4'b1010 | 4'b0011 = %b", a | b);

    // Bitwise INV
    #5;
    $display("04.Bitwise INV         : ~4'b1010 = %b", ~a);

    // Addition
    #5;
    $display("05.Addition            : 4'b1010 + 4'b0011 = %b", a + b);

    // Substraction
    #5;
```

```
    // Substraction
    #5;
    $display("06.Substraction        : 4'b1010 - 4'b0011 = %b", a - b);

    // Multiplication
      a = 4'b0101;
      b = 4'b0010;
    #5;
    $display("07.Multiplication      : 4'b0101 * 4'b0010 = %b", a * b);


    // Division
      a = 4'b1010;
      b = 4'b0010;
    #5;
    $display("08.Division            : 4'b1010 / 4'b0010 = %b", a / b);

    // Modulo
    a = 4'b1011;
    b = 4'b0010;
    #5;
    $display("09.Modulo              : 4'b1011 %% 4'b0010 = %b", a % b);

    // Greater
    a = 4'b1010;
    b = 4'b1000;
    #5;
    $display("10.Greater             : 4'b1010 > 4'b1000 = %b", a > b);

    // Greater or Equal
    a = 4'b1010;
    b = 4'b1010;
    #5;
    $display("11.Greater or Equal    : 4'b1010 >= 4'b1010 = %b", a >= b);

    // Less
    a = 4'b1010;
    b = 4'b1001;
    #5;
    $display("12.Less                : 4'b1010 < 4'b1001 = %b", a < b);
```

```
 // Less or Equal
 #5;
$display("13.Less or Equal        : 4'b1010 <= 4'b1010 = %b", a<=b);

// Logical Equality (1)
a = 4'b1100;
b = 4'b1100;
#5;
$display("14.1.Logical Equality  : 4'b1100 == 4'b1100 = %b", a == b);

// Logical Equality (2)
a = 4'b1100;
b = 4'b110x;
#5;
$display("14.2.Logical Equality  : 4'b1100 == 4'b110x = %b", a == b);

// Logical Inequality (1)
a = 4'b1100;
b = 4'b1100;
#5;
$display("15.1.Logical Inequality: 4'b1100 != 4'b1100 = %b", a != b);

// Logical Inequality (2)
a = 4'b1100;
b = 4'b110x;
#5;
$display("15.2.Logical Inequality: 4'b1100 != 4'b110x = %b", a != b);

// Case Equality (1)
a = 4'b1100;
b = 4'b1100;
#5;
$display("16.1.Case Equality     : 4'b1100 === 4'b1100 = %b", a === b);

// Case Equality (2)
a = 4'b1100;
b = 4'b110x;
#5;
$display("16.2.Case Equality     : 4'b1100 === 4'b110x = %b", a === b);
```

```
// Case Inequality (1)
a = 4'b1100;
b = 4'b1100;
#5;
$display("17.1.Case Inequality   : 4'b1100 !== 4'b1100 = %b", a !== b);

// Case Inequality (2)
a = 4'b1100;
b = 4'b110x;
#5;
$display("17.2.Case Inequality   : 4'b1100 !== 4'b110x = %b", a !== b);

// Logical And (1)
  a_s = 5;
  b_s = 3;
#5;
$display("18.1.Logical And       : a=5,b=3 = ((a==5) && (b==3)) = %b", (a_s==5) && (b_s==3) );

// Logical And (2)
  a = 4'b1010;
  b = 4'b0001;
#5;
$display("18.2.Logical And       : 4'b1010 && 4'b0001 = %b", a && b);

// Logical Or (1)
   a_s = 5;
   b_s = 3;
#5;
  $display("19.1.Logical OR : a=5,b=3 = ((a!=5) || (b!=3)) = %b",(a_s != 5) || (b_s !=3));
// Logical Or (2)
  a = 4'b0010;
  b = 4'b0001;
#5;
$display("19.2.Logical OR        : 4'b0010 || 4'b0001 = %b", a || b);

// Logical Invert (1)
#5;
$display("20.1.Logical Invert    : !1'b1 = %b", !1'b1);

// Logical Invert (2)
  a = 4'b0001;
#5;
$display("20.2.Logical Invert    : !4'b0001 = %b", !a);
```

```
    // Logical Shift Left
      a = 4'b0001;
    #5;
    $display("21.Logical Shift Left  : 1 << 2 = %b", a << 2);

    // Logical Shift Right
      a = 4'b1101;
    #5;
    $display("22.Logical Shift Right : 4'b1101 >> 2 = %b", a >> 2);

    // Reduction And
      a = 4'b1011;
    #5;
    $display("23.Reduction And       : a = 4'b1011 -> &a[3:0] = %b", &a);

    // Reduction Or
    a = 4'b1000;
    #5;
    $display("24.Reduction Or        : a = 4'b1000 -> |a[3:0] = %b", |a);

    // Reduction Xor
    a = 4'b1010;
    #5;
    $display("25.Reduction XOR       : a = 4'b1010 -> ^a[3:0] = %b", ^a);

    // Concatenation
    a = 4'b1100;
    b = 4'b001;
    #5;
    $display("26.Concatenation      : a = 4'b1100, b = 3'b001 --> {a[3:0],b[2:0]} = %b", {a[3:0], b[2:0]});

    // Replication
    d = 2'b10;
    #5;
    $display("27.Replication         : {4{2'b10}} = %b", {4{d}});

    #100;
    $finish;
  end

endmodule
```

```
huugiap@ictc-eda-ldap:~/05_ss5/tb$ vi test_bench.v
huugiap@ictc-eda-ldap:~/05_ss5/tb$ cd ../sim/
huugiap@ictc-eda-ldap:~/05_ss5/sim$ make build
vlib work
** Warning: (vlib-34) Library already exists at "work".
Errors: 0, Warnings: 1
vmap work work
Questa Intel Starter FPGA Edition-64 vmap 2023.3 Lib Mapping Utility 2023.07 Jul 17 2023
vmap work work
Modifying modelsim.ini
vlog -f compile.f | tee compile.log
Questa Intel Starter FPGA Edition-64 vlog 2023.3 Compiler 2023.07 Jul 17 2023
Start time: 22:36:21 on Mar 22,2025
vlog -f compile.f
-- Compiling module test_bench

Top level modules:
        test_bench
End time: 22:36:21 on Mar 22,2025, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
```

```
huugiap@ictc-eda-ldap:~/05_ss5/sim$ make run
vsim -debugDB -l test_adder.log -voptargs=+acc -assertdebug -c test_bench -do "log -r /*;run -all;"
Reading pref.tcl

# 2023.3

# vsim -debugDB -l test_adder.log -voptargs="+acc" -assertdebug -c test_bench -do "log -r /*;run -all;"
# Start time: 22:36:28 on Mar 22,2025
# ** Note: (vsim-3812) Design is being optimized...
# ** Note: (vsim-8611) Generating debug db.
# ** Warning: (vopt-10587) Some optimizations are turned off because the +acc switch is in effect. This will cause your simulation to run slowly. Please use -access/-debug to maintain needed visibility.
# ** Note: (vsim-12126) Error and warning message counts have been restored: Errors=0, Warnings=1.
# //  Questa Intel Starter FPGA Edition-64
# //  Version 2023.3 linux_x86_64 Jul 17 2023
# //
# //  Copyright 1991-2023 Mentor Graphics Corporation
# //  All Rights Reserved.
# //
# //  QuestaSim and its associated documentation contain trade
# //  secrets and commercial or financial information that are the property of
# //  Mentor Graphics Corporation and are privileged, confidential,
# //  and exempt from disclosure under the Freedom of Information Act,
# //  5 U.S.C. Section 552. Furthermore, this information
# //  is prohibited from disclosure under the Trade Secrets Act,
# //  18 U.S.C. Section 1905.
# //
```

```
# Loading work.test_bench(fast)
# ** Note: (vsim-8900) Creating design debug database vsim.dbg.
# log -r /*
# run -all
# 01.Bitwise XOR         : 4'b1010 ^ 4'b0011 = 1001
# 02.Bitwise AND         : 4'b1010 & 4'b0011 = 0010
# 03.Bitwise OR          : 4'b1010 | 4'b0011 = 1011
# 04.Bitwise INV         : ~4'b1010 = 0101
# 05.Addition            : 4'b1010 + 4'b0011 = 1101
# 06.Substraction        : 4'b1010 - 4'b0011 = 0111
# 07.Multiplication      : 4'b0101 * 4'b0010 = 1010
# 08.Division            : 4'b1010 / 4'b0010 = 0101
# 09.Modulo              : 4'b1011 % 4'b0010 = 0001
# 10.Greater             : 4'b1010 > 4'b1000 = 1
# 11.Greater or Equal    : 4'b1010 >= 4'b1010 = 1
# 12.Less                : 4'b1010 < 4'b1001 = 0
# 13.Less or Equal        : 4'b1010 <= 4'b1010 = 0
# 14.1.Logical Equality  : 4'b1100 == 4'b1100 = 1
# 14.2.Logical Equality  : 4'b1100 == 4'b110x = x
# 15.1.Logical Inequality: 4'b1100 != 4'b1100 = 0
# 15.2.Logical Inequality: 4'b1100 != 4'b110x = x
# 16.1.Case Equality     : 4'b1100 === 4'b1100 = 1
# 16.2.Case Equality     : 4'b1100 === 4'b110x = 0
# 17.1.Case Inequality   : 4'b1100 !== 4'b1100 = 0
# 17.2.Case Inequality   : 4'b1100 !== 4'b110x = 1
# 18.1.Logical And       : a=5,b=3 = ((a==5) && (b==3)) = 0
# 18.2.Logical And       : 4'b1010 && 4'b0001 = 1
# 19.1.Logical OR : a=5,b=3 = ((a!=5) || (b!=3)) = 1
# 19.2.Logical OR        : 4'b0010 || 4'b0001 = 1
# 20.1.Logical Invert    : !1'b1 = 0
# 20.2.Logical Invert    : !4'b0001 = 0
# 21.Logical Shift Left  : 1 << 2 = 0100
# 22.Logical Shift Right : 4'b1101 >> 2 = 0011
# 23.Reduction And       : a = 4'b1011 -> &a[3:0] = 0
# 24.Reduction Or        : a = 4'b1000 -> |a[3:0] = 1
# 25.Reduction XOR       : a = 4'b1010 -> ^a[3:0] = 0
# 26.Concatenation      : a = 4'b1100, b = 3'b001 --> {a[3:0],b[2:0]} = 1100001
# 27.Replication         : {4{2'b10}} = 10101010
# ** Note: $finish    : ../tb/test_bench.v(205)
#    Time: 270 ns  Iteration: 0  Instance: /test_bench
# End time: 22:36:29 on Mar 22,2025, Elapsed time: 0:00:01
# Errors: 0, Warnings: 1
```