Digital Systems Design ELEC373/473

Tasks and Functions

Design Documentation

- Verilog models' usefulness to other people depends on the *correctness* and *clarity* of the description.
- **Tasks** and **Functions** are two types of subprograms that can improve the clarity of a description by encapsulating code.
  - **Functions** substitute for an expression
  - **Tasks** create a hierarchical organization of the procedural statements within a Verilog behaviour.
- Encapsulation of Verilog code hides the details of an implementation from the outside world.
- Overall, tasks and functions improve the readability, portability and maintainability of a model.

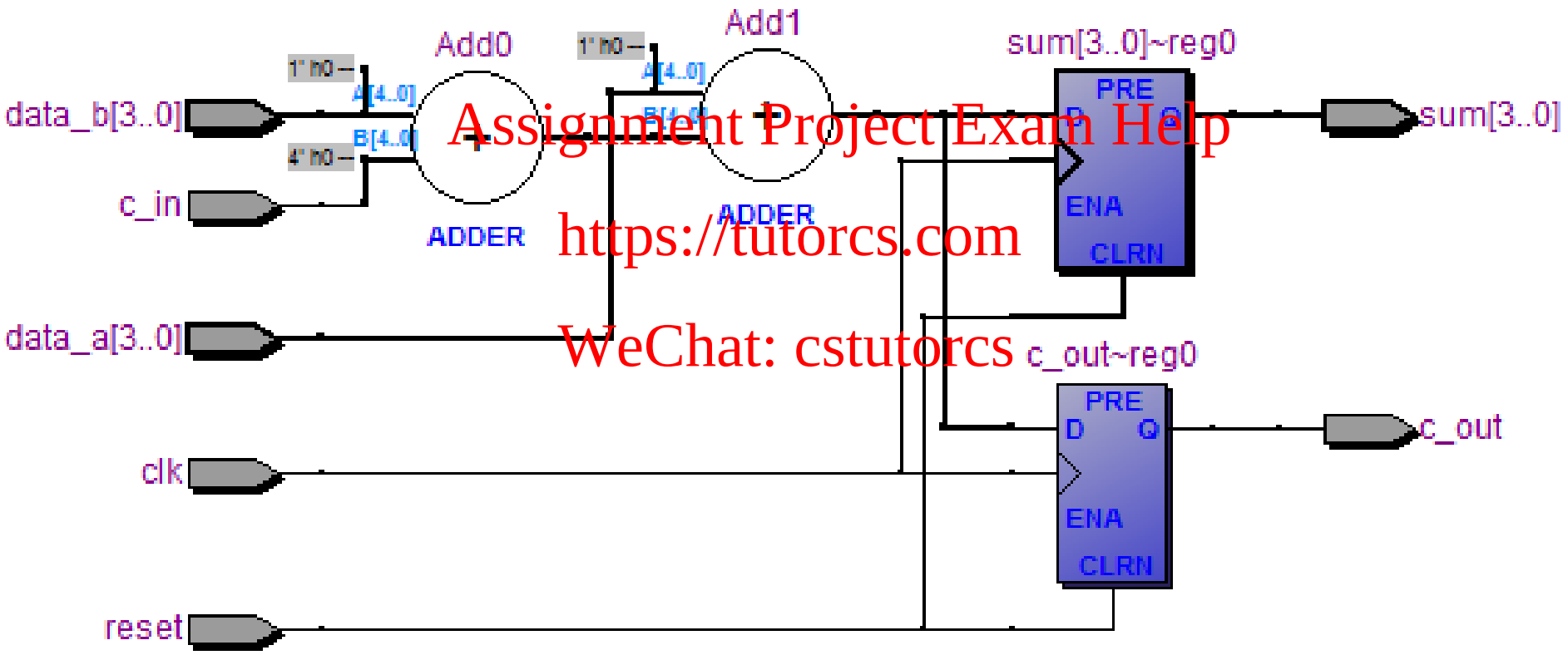
Tasks

- Tasks are declared within a module, and they may be referenced only from within a cyclic or single-pass behaviour.
- When a task is called, copies of the parameters in the environment are associated with the inputs, outputs, and inouts within the task ***according to their order***.
  - A task can call itself.
  - A task must be named.
  - All of the declarations of variables are local to the task.
  - The arguments of the task retain the type they hold in the environment that invokes the task.

Example 1- Adder_task

```
module adder_task (c_out, sum, c_in, data_a, data_b, clk, reset);
    output          [3:0]       sum;
    output                      c_out;
    input           [3:0]       data_a, data_b;
    input                       clk, reset, c_in;
    reg             [3:0]       sum;
    reg                         c_out;
    always @ (posedge clk or posedge reset)
        if (reset)  {c_out, sum} <= 0; else
                    add_values (c_out, sum, data_a, data_b, c_in);
    task add_values;
        output                  c_out;
        output      [3:0]       sum;
        input       [3:0]       data_a, data_b;
        input                   c_in;
        begin       {c_out, sum} <=  data_a + (data_b + c_in);     end
    endtask
endmodule
```

Synthesised hardware



Functions

- Verilog **functions** are declared within a parent module and can be referenced in any valid expression-for example, in the RHS of a continuous assignment statement.
- Functions may implement only combinational behaviour.
- Functions may not contain timing controls (no delay control [#], event control [@] , or wait statements).
- A function may contain a declaration of inputs and local variables (no output or inout port)
- The value of a function is returned by its name when the expression calling the function is executed.
- The definition of a function implicitly defines an internal register variable with the same name, range, and type as the function itself. This variable must be assigned a value within the function body.

Example 2 – Arithmetic_unit

```
module arithmetic_unit  (result_1, result_2, operand_1, operand_2);
        output          [4:0]       result_1;
        output          [3:0]       result_2;
        input           [3:0]       operand_1, operand_2;
        assign result_1 =  sum_of_operands (operand_1, operand_2);
        assign result_2 =  largest_operand (operand_1, operand_2);

        function        [4:0]       sum_of_operands;
                input   [3:0]       operand_1, operand_2;
                sum_of_operands  = operand_1 + operand_2;
        endfunction

        function        [3:0]       largest_operand ;
                input   [3:0]       operand_1, operand_2;
                largest_operand = (operand_1 >= operand_2) ? operand_1 : operand_2;
        endfunction
 endmodule
```

Example 2 – Arithmetic_unit