Verilog Review

Instructor: Siting Liu

School of Information Science and Technology (SIST), ShanghaiTech University

Shanghai Institute of Semiconductor Innovation (SISI)

https://faculty.sist.shanghaitech.edu.cn/liust/EE219/EE219.html



Computer Architecture - EE219

faculty.sist.shanghaitech.edu.cn

立志成才报国裕民

Outline

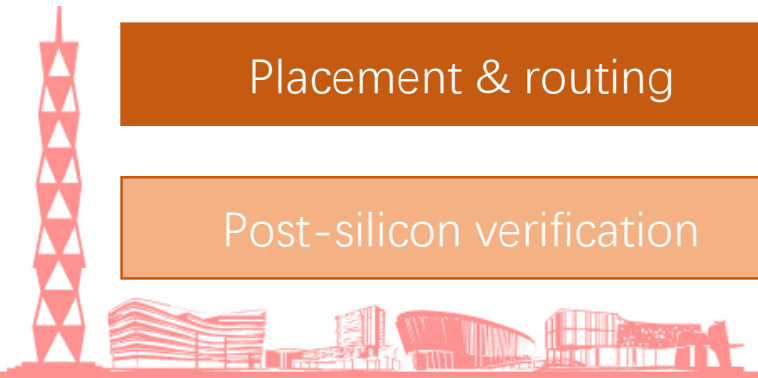
- Basics on Verilog HDL
- A concrete example: systolic array



立志成才报国裕民

What is Verilog HDL for?

Design Specification	• Functionality/algorithm implemented, communication interface etc.
Architecture Specification	• Dataflow, functional blocks (IPs), software/hardware co-optimize, ISA
Design/RTL Pre-silicon verification	• **Finite state machine (controller), datapath, arithmetic, memory**, mixed-signal circuits, DFT, clock tree, etc. • Apply different stimulus to verify the design (more time-consuming)
Logic synthesis/netlist	• Generate netlist (components and how they are connected) that meets PPA specifications
Equivalence check	• Check the equivalence between netlist & RTL
Placement & routing	• Generate layout (Cadence Encounter/Synopsys IC compiler) & verify
Post-silicon verification	• After fabrication, post-silicon validation and test



立志成才报国裕民

What is Verilog HDL for?

- **Digital Design Documentation:** Describe and document digital circuits and system architectures.
- **Digital Hardware Design:** Design and implement digital circuits and systems.
- **Hardware Simulation and Verification:** Simulate circuit behavior and verify functional correctness.
- Enable high-level (RTL and above) digital system development using C-like syntax.

```
module max2(a,b,c);
input wire a, b;
output reg c;
always @(a,b)
begin
   if (a>b)
      c <= 1;
   else
      c <= 0;
end
endmodule
```

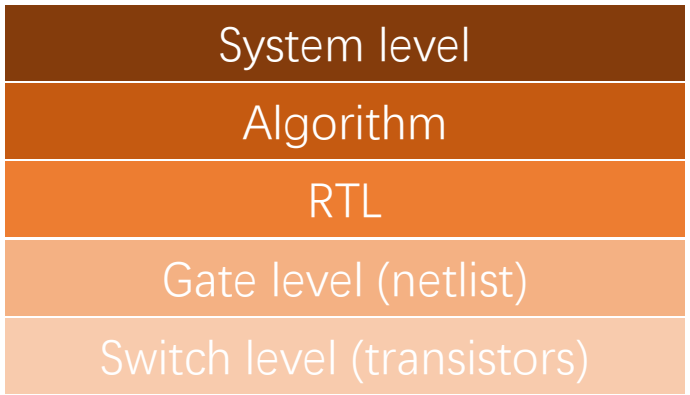
Verilog

```
int main()
{
   int a, b, c;
   printf("Input two numbers");
   scanf("%d %d", &a, &b);
   if (a>b) c = 1;
   else c = 0;
   printf("c=%d\n", c);
   return 0;
}
```

立志成才报国裕民

Terminology

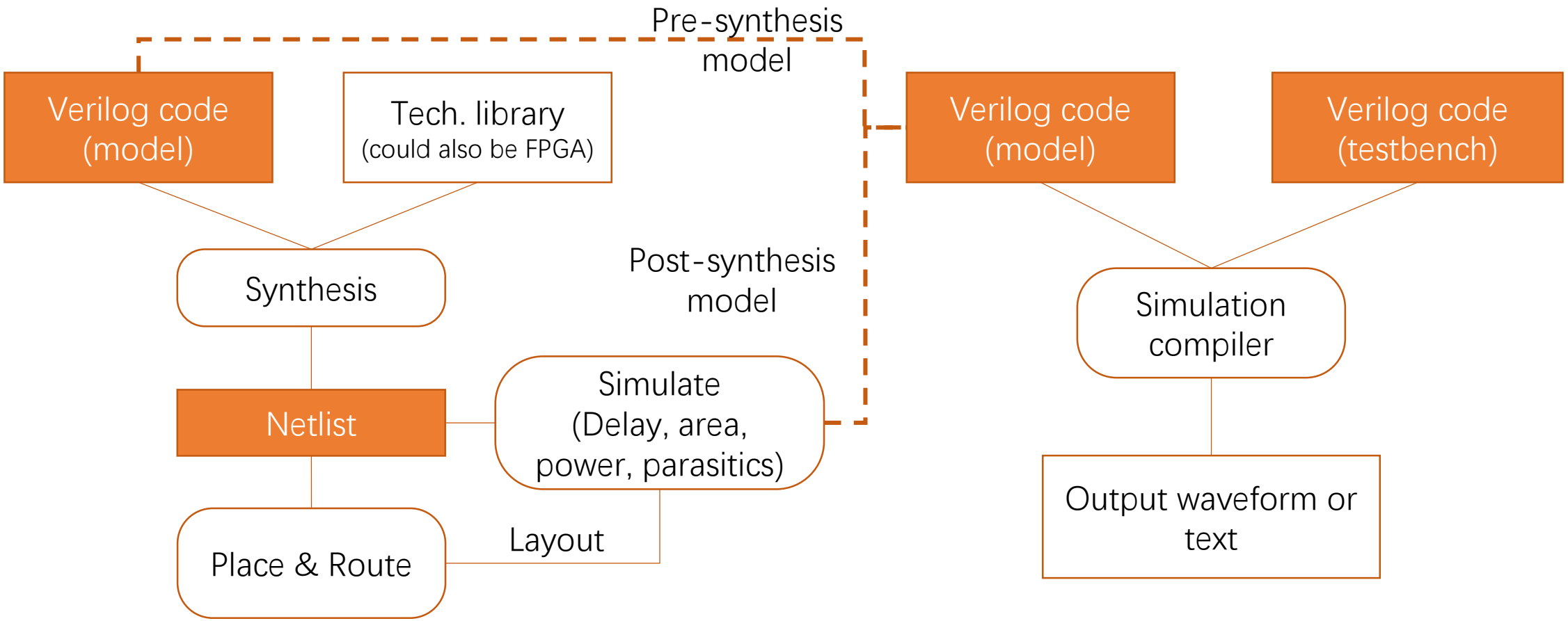
- Register-Transfer Level (RTL)
  - Usually models a synchronous digital circuit in terms of the flow of digital signals between registers and the logical operations performed on those signals.
- Synthesis
  - Translate HDL model to a circuit and optimize
- RTL synthesis
  - Translate RTL model to an optimized netlist in **certain technology node/target standard-cell library**





立志成才报国裕民

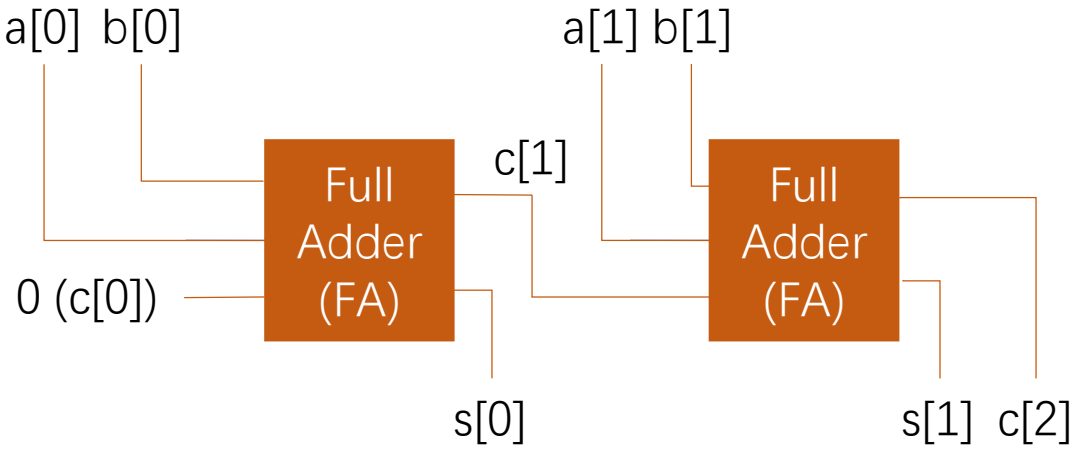
RTL Synthesis & Simulation Flow



Two Ways to Model a Circuit

- Structural Modeling






```
module fa(cout,sum,a,b,cin);
input a,b,cin;
output cout,sum;
wire axorb, aandb, bandcin, aandcin, abcin;
xor xor1(axorb,a,b);
xor xor2(sum,cin,axorb);
and and1(aandb,a,b);
and and2(bandcin,b,cin);
and and3(aandcin,a,cin);
or  or1(abcin,aandb,bandcin);
or  or2(cout,abcin,aandcin);
endmodule
```

```
module fa2(input [1:0] a, input [1:0] b, output [2:0] sum);
wire [2:0] c;
wire [1:0] s;
assign c[0]=1'b0;
fa fa1(.cout(c[1]), .sum(s[0]), .a(a[0]), .b(b[0]), .cin(c[0]));
fa fa2(.cout(c[2]), .sum(s[1]), .a(a[1]), .b(b[1]), .cin(c[1]));
assign sum={c[2], s};
endmodule
```

Two Ways to Model a Circuit – Behavior Model

- Behavior Modeling
  - Only the functionality of the circuit is described
  - Synthesis tool produces correct logic automatically

```
module fa8_behavior(input [7:0] a, input [7:0] b, output [8:0] sum);
assign sum={a[7], a} + {b[7], b};  //signed adder
endmodule

module multi4_beh(input [3:0] a, input [3:0] b, output [7:0] prod);
assign prod=a*b; //unsigned multiplier
endmodule
```

- Easier to describe but less control to the detailed circuit implementation
- Usually use behavioral model to describe each component, and use structural model to connect them

立志成才报国裕民

Basic Component – a Module

```
//Module name   //Port declaration/list of input & output
module fa2(input [1:0] a, input [1:0] b, output [2:0] sum);
                  2-bit input signals a and b   3-bit output signal sum
wire [2:0] c;
wire [1:0] s;          //Declaration of internal signals

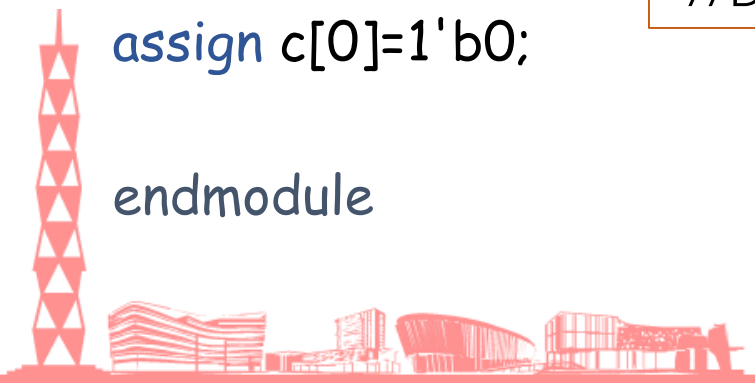
fa fa1(.cout(c[1]), .sum(s[0]), .a(a[0]), .b(b[0]), .cin(c[0]));
fa fa2(.cout(c[2]), .sum(s[1]), .a(a[1]), .b(b[1]), .cin(c[1]));
                                                    //Instantiation of modules

assign sum={c[2], s};   //Description of behavior
assign c[0]=1'b0;

endmodule
```

- A port list is not a must for a testbench
- A port list can also be declared within the module

- Start with module, end with endmodule
- Case-sensitive
- All keywords are lowercase
- Semicolon as statement terminators
- //, /* */ for comments



立志成才报国裕民

Data Types

- Constant
  - Number
  - Parameter
- Variable/signal/numerics
  - Wire
  - Reg
  - Memory
- … …

上海科技大学 ShanghaiTech University



立志成才报国裕民

Constant

Actual bit width — 8'hab

<bit width>'<base><digits>

base	radix
b	binary
d	decimal
h	hexadecimal
o	octal

8'b10000000

Can represent 128 in unsigned integer or -128 in signed integer or an 8-bit instruction, etc.

<base><digits> Default bit width, e.g., 32

<digits> Decimal number

10 ↔ 32'd10 ↔ 32'b1010 ↔ 32'b00...001010

-1 ↔ -32'd1 ↔ 32'hFFFFFFFF

"AB" ↔ 16'b01000001_01000010 ↔ 16'h4142

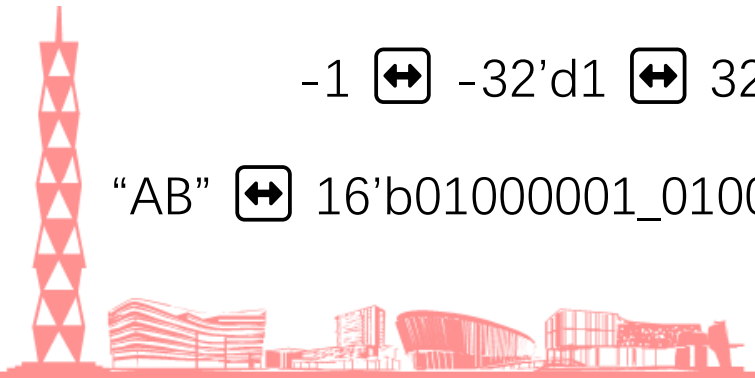
- High impedance/uncertainty: z(?)/x

  8'b1000000x

  8'hax ↔ 8'b1010xxxx

- Negative numbers -8'd5 ↔ 8'b11111011

  -5 in 2's complement



立志成才报国裕民

Constant (Cont'd)

- Parameter
  - parameter para1=expression1, para2=expression2...;

```
module faN(input [W-1:0] a, input [W-1:0] b, output [W:0] sum);
parameter W=8;
wire [W:0] c;
wire [W-1:0] s;
assign c[0]=1'b0;
genvar i;
generate for (i=0; i<W; i=i+1)
fa faN(.cout(c[i+1]), .sum(s[i]), .a(a[i]), .b(b[i]), .cin(c[i]));
endgenerate
assign sum={c[W], s};
endmodule
```

Instantiate a module with changeable parameters.

- Method 1

```
module somemodule(... ...);
... ...
faN #(6) fulladder(.a(in1),.b(in2),.sum(sum));
... ...
endmodule
```

- Method 2

```
module somemodule(... ...);
... ...
defparam fulladder.W=6;
faN fulladder (.a(in1),.b(in2),.sum(sum));
... ...
endmodule
```

上海科技大学 ShanghaiTech University
立志成才报国裕民

Wire Type

- Wire: physical connections between blocks, can be input/output of a block, but cannot store values.
- Input/output ports of a module are wires by default.

```
wire a;          //a 1-bit wire

wire [7:0] b;    //an 8-bit wire

wire [8:1] c;    //an 8-bit wire

wire [1:6] d;    //a 6-bit wire
```

b[i] to index one signal in b.

```
module fa2(input [1:0] a, input [1:0] b, output [2:0] sum);
wire [2:0] c;
wire [1:0] s;
assign c[0]=1'b0;
fa fa1(.cout(c[1]), .sum(s[0]), .a(a[0]), .b(b[0]), .cin(c[0]));
fa fa2(.cout(c[2]), .sum(s[1]), .a(a[1]), .b(b[1]), .cin(c[1]));
assign sum={c[2], s};
endmodule
```

Can use assign to (continuously) assign the value.
Equivalent to a physical wire/logic circuit.

立志成才报国裕民

Reg Type

- Reg: usually represent a register as the name suggests, mostly used in always block.
- Input cannot be reg

```
reg a;          //a 1-bit reg

reg [7:0] b;    //an 8-bit reg
```

b[i] to index one signal in b.

Default initiated with 'x' (uncertain).

Reg data can be assigned negative value, but will be treated as unsigned number performing arithmetic.

```
input d;
output reg q;
input clock;

always @(posedge clock)
begin
  q <= d;
end
```

上海科技大学 ShanghaiTech University

Wire vs. reg

- wire
  - wire can be used to connect instances/modules
  - wire can be used for module inputs & outputs declaration
  - wire are the only legal type on the left-hand side of an assign statement
  - wire cannot be used as the left-hand side of an = or <= sign in an always@ block
- reg
  - reg can be connected to the input of an instance
  - reg cannot be connected to the output port of a module instantiation
  - reg can be used for module output declaration but not input
  - reg is the only legal type on the left-hand side of an always block = or <= sign and an initial block = sign (used in test benches)
  - reg can be used to create registers when used in conjunction with always@ (*edge clock) blocks
- When two types are interchangeable
  - Both can appear on the right-hand side of assign statements and always block = or <= signs.
  - Both can be connected to the input ports of module instantiations

上海科技大学 ShanghaiTech University
立志成才报国裕民

Reg Array (Memory)

- Memory: its logic behavior can be modeled by regs.
- In reality, the model can be generated by memory compiler for fabrication.
- The type of an array can also be wire (to represent a vector)

```
reg [n-1:0] memory_name[m-1:0];
```

Bit width of each number

The number of elements

```
reg [n-1:0] reg1;
reg reg2[n-1:0];
```

n-bit register

A memory has n 1-bit register

```
reg1 <= 0;
```

√

```
reg2 <= 0;
```

×

- Cambricon has 64 32-bit general registers

```
parameter reg_size=32, num_reg=64;
reg [reg_size-1:0] reg_file [num_reg-1:0];
```

```
reg2[i] <= 0;
```

√

上海科技大学 ShanghaiTech University

Numeric

- signed/unsigned
  - Can be combined with wire/reg, e.g., wire signed a; (sign-extended when used for arithmetic)
  - Unsigned by default
  - $signed()/$unsigned() for type conversion
- Integer
  - Used in for loop as loop variable or constants
  - Example: integer a = 255;
- Real
  - real r1, r2;
  - $bitstoreal()/$realtobits() for type conversion
  - r1 = $bitstoreal(vec_val)

上海科技大学 ShanghaiTech University

Operators

- +, -, *, /, % (add, sub., multi., div., modulo)
- =, <= (assign, in statement)
- >, <, >=, <=(compare, in expression)
- &&, ||, ! (logical)
- &, |, ~, ^, ^~ (bitwise logic, can be considered as logic gates, or used as reduction operators)
- ?: (conditional)
- <<, >> (logical shift) <<<, >>> (arithmetic shift)
- {} (concatenation)
- ==, !=, ===, !== (equal or not)

Table 5-12—Bitwise binary and operator

&	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

Table 5-13—Bitwise binary or operator

|	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

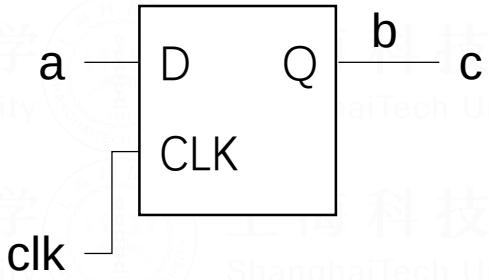
IEEE Standard for Verilog® Hardware Description Language

立志成才报国裕民

Blocking/Non-blocking Assignments

- =, <= (blocking/non-blocking assignment)

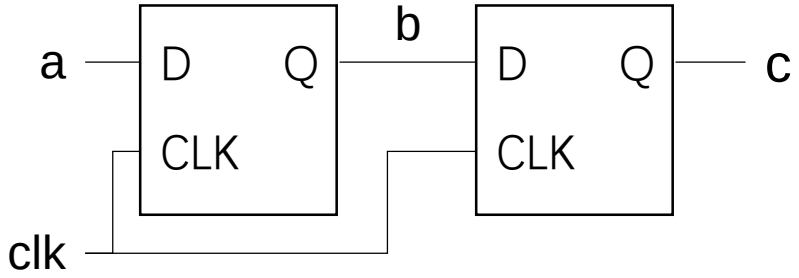
```
module dffb(a,clk,c);
input a,clk;
output reg c;
reg b;
always @(posedge clk)
  begin
    b=a;
    c=b;
  end
endmodule
```



Sequential

Evaluate the RHS and complete assignment without interruption from any other statement; Take effect/assign immediately.

```
module dff(a,clk,c);
input a,clk;
output reg c;
reg b;
always @(posedge clk)
  begin
    b<=a;
    c<=b;
  end
endmodule
```



Systolic/pipeline

Evaluate the RHS together; Take effect/assign after evaluation of RHS

立志成才报国裕民

Blocking/Non-blocking Guideline

- Guideline
  - Use non-blocking assignments in always blocks that describe sequential logic.
  - Use blocking assignments in always blocks that describe combinational logic.
  - Do not make assignments to the same variable from more than one always block.

上海科技大学 ShanghaiTech University



立志成才报国裕民

Code Block

- A module can have multiple blocks (sub-circuits), and they works in parallel
- always block
  - always @([sensitive list]) begin … … end
  - Event-driven and can repeat
- initial block (mostly used in Verilog testbench)
  - initial begin … … end
  - Execute only once
  - Generate waveform or initialize memory elements
  - Un-synthesizable



立志成才报国裕民

always Block

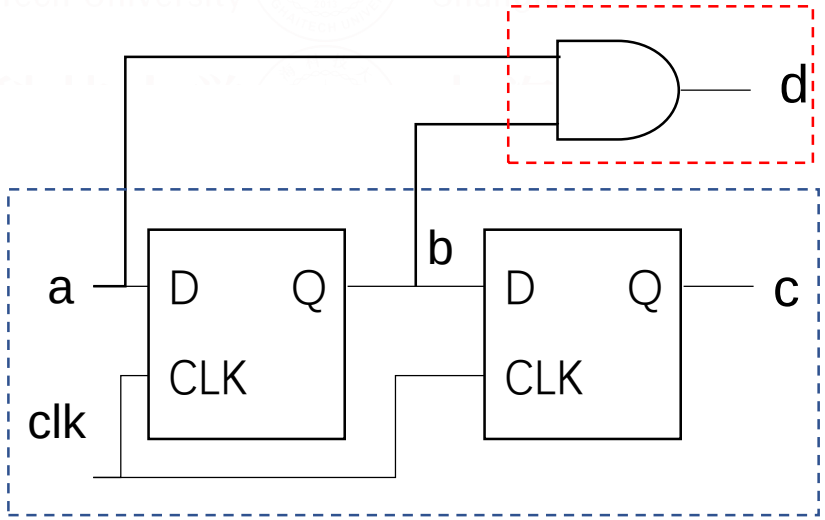
- always @([sensitive list]) begin … … end
- Event-driven and can repeat
- Use always @(*) to automatically generate the sensitive list

Use cases

```
`timescale 100ns/10ns
//time unit/time precision
reg clk;
always #half_clk_period clk = ~clk


input clk, reset;
// typical sequential logic
always @ (posedge clk or posedge reset)
   begin
      … …
   end
```

negedge




```
module dff(a,clk,c,d);
input a,clk;
output reg c, d;
reg b;
always @(posedge clk)
   begin
      b<=a;
      c<=b;
   end
always @(a, b)
   begin
      d=a & b;
   end
endmodule
```

立志成才报国裕民

Inside the Code Blocks

- if … else
- It must be within a code block (always, initial)

```
always @(reset, posedge clk)
begin
   if (!reset)
      q <= 1'b0;
   else
      q <= d;
end
```

DFF with asyn. reset

```
always @(a,b)
begin
   if (a>b)
      c = 1;
   else
      c = 0;
end
```

Combinational comparator

```
initial begin
if (TEST_MODE == 0) begin
        $display("Running Basic
Functionality Test");
        // ... More basic tests
    end
    else if (TEST_MODE == 1) begin
        $display("Running Boundary Test");
         // ... More boundary tests
    end
    else if (TEST_MODE == 2) begin
        $display("Running Random Stress
Test");
        // ... More random tests
    end
    else begin
        $display("Error: Unknown test
mode!");
        $finish; // finish the simulation
    end
end
```

- case
- It must be within a code block (always, initial)
- Also widely used in finite state machine model

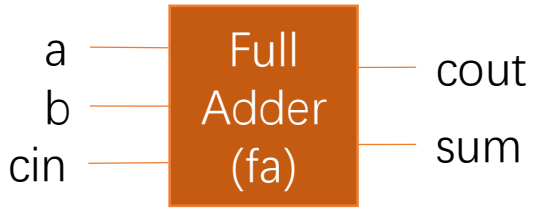
```
module mux(out, i0, i1, i2, i3, sel);
input i0, i1, i2, i3;
input wire [1:0] sel;
output reg out;
always @(*)
   begin
      case (sel)
         2'b00: out=i0;
         2'b01: out=i1;
         2'b10: out=i2;
         2'b11: out=i3;
         default: out=1'b0;
      endcase
   end
endmodule
```

立志成才报国裕民

Instantiation

- [component name] [instance name] (port list)

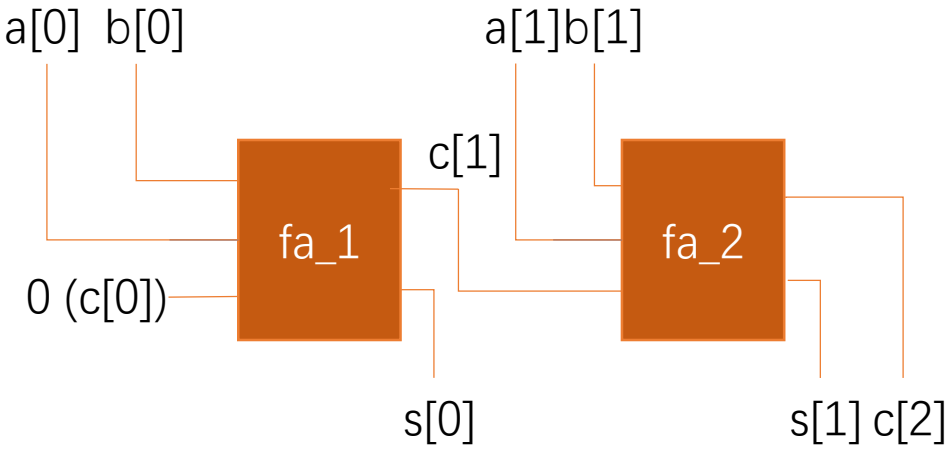
Module name of the lower-level component	Unique name of the component	List of signals to connect to the component




```
module fa(cout,sum,a,b,cin);
```

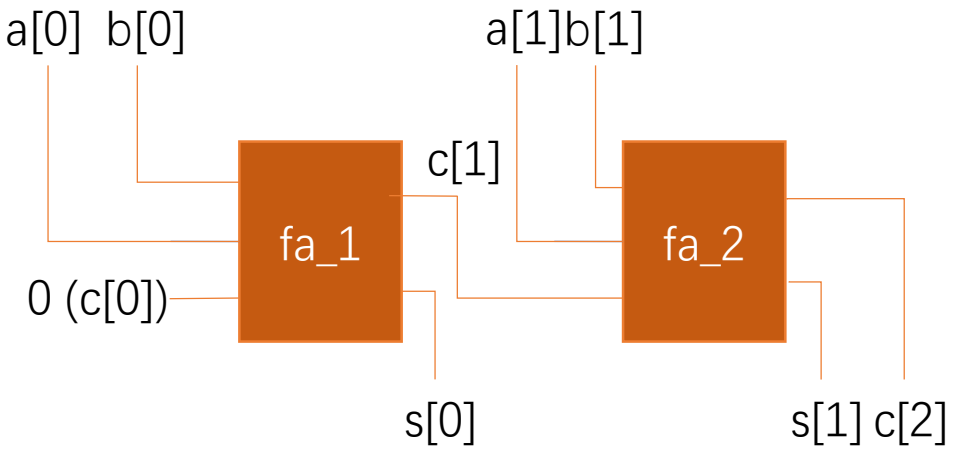
- Connecting the input/output ports
  - Method 1: by ordered list

```
module fa2(input [1:0] a, input [1:0] b,
output [2:0] sum);
wire [2:0] c;
wire [1:0] s;
assign c[0]=1'b0;
fa fa_1(c[1], s[0], a[0], b[0], c[0]);
fa fa_2(c[2], s[1], a[1], b[1], c[1]);
assign sum={c[2], s};
endmodule
```



立志成才报国裕民

Instantiation by Name (Recommended)




```
module fa2(input [1:0] a, input [1:0] b, output [2:0] sum);
wire [2:0] c;
wire [1:0] s;
assign c[0]=1'b0;
fa fa_1(.cout(c[1]), .sum(s[0]), .a(a[0]), .b(b[0]), .cin(c[0]));
fa fa_2(.cout(c[2]), .sum(s[1]), .a(a[1]), .b(b[1]), .cin(c[1]));
assign sum={c[2], s};
endmodule
```

Can have open ports that are not connected

立志成才报国裕民

Instantiation with Parameters

- Parameter
- parameter para1=expression1, para2=expression2...;

```
module faN(input [W-1:0] a, input [W-1:0] b, output [W:0] sum);
parameter W=8;
... ...
endmodule
```

IEEE verilog-2001 style

```
module faN #(parameter W = 8)
(input [W-1:0] a, input [W-1:0] b, output [W:0] sum);
... ...
endmodule
```

Instantiate a module with changeable parameters.

- Method 1

```
module somemodule(... ...);
... ...
faN #(6) fulladder
(.a(in1),.b(in2),.sum(sum));
... ...
endmodule
```

- Method 2

```
module somemodule(... ...);
... ...
defparam fulladder.W=6;
faN fulladder (.a(in1),.b(in2),.sum(sum));
... ...
endmodule
```

上海科技大学 ShanghaiTech University

Loop

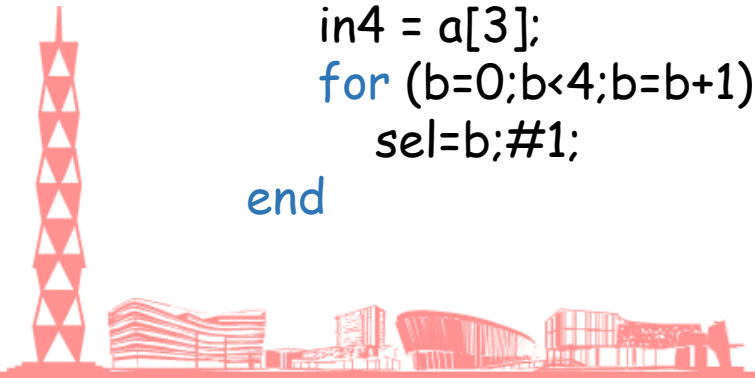
- “forever”, generates periodic waveform used in simulation (testbench file) and different from “always”, it must be in “initial” block;
- repeat ([constant expression]) statements;
- while ([expression]) statements;
- for ([initialization]; [end condition]; [change loop parameter]) statements;

```
for (a=0;a<16;a=a+1)
   begin
      in1 = a[0];
      in2 = a[1];
      in3 = a[2];
      in4 = a[3];
      for (b=0;b<4;b=b+1)
         sel=b;#1;
   end
```

Generate test vectors or certain waveform

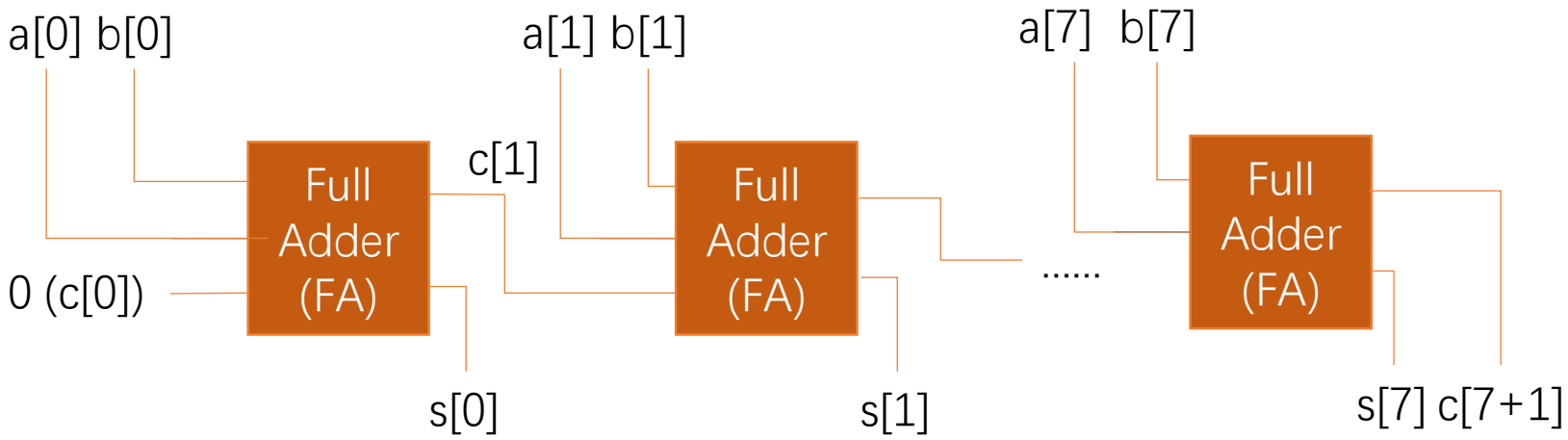
```
module lshifter(in,result);
input [1:0] in;
output reg [3:0] result;
integer i;
always @(in) begin
   for (i=2;i<=3;i=i+1) begin
         result[i]=in[i-2]; end
   result[1:0]=0;
end
endmodule
```

Unroll the for loop, repeat the circuit structure



立志成才报国裕民

Generate … for Loop






```
module fa(cout,sum,a,b,cin);
```

```
module fa8(input [7:0] a, input [7:0] b, output [8:0] sum);
wire [8:0] c;
wire [7:0] s;
assign c[0]=1'b0;
genvar i;
generate for (i=0; i<8; i=i+1)
fa fa8(.cout(c[i+1]), .sum(s[i]), .a(a[i]), .b(b[i]), .cin(c[i]));
endgenerate
assign sum={c[8], s};
endmodule
```

立志成才报国裕民

Conditional generate

- Assume two multiplicand a0 and a1 with bit width of a0_width and a1_width, cal_multiplier & tree_multiplier are two submodules

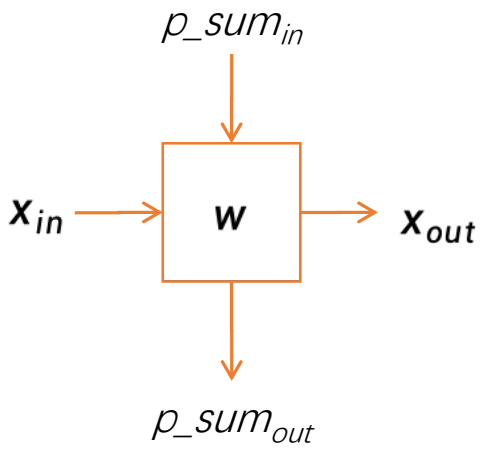
```
generate
if (a0_width<8) || (a1_width<8)
    cal_multiplier  #(a0_width, a1_width) m0(product, a0, a1);
else
    tree_multiplier #(a0_width, a1_width) m0(product, a0, a1);
endgenerate
```



立志成才报国裕民

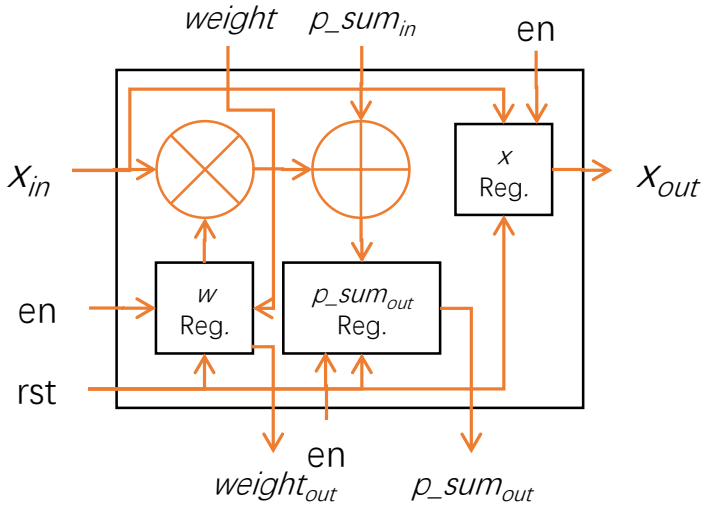
Build Systolic Array with Verilog HDL

- Single PE unit for matrix multiplication
  - The datapath implements the multiply-and-accumulate (MAC)
  - Requires extra circuitry to load the weight
  - Can reset/idle/compute/load (4 instructions)



$$x_{out} = x_{in}$$

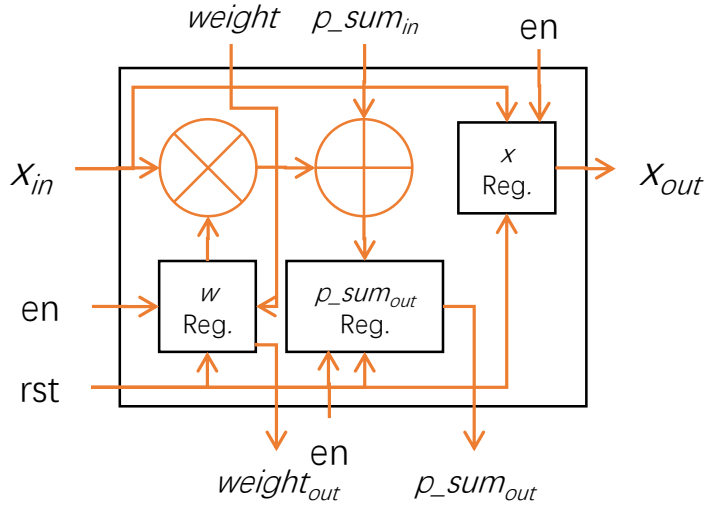
$$p_sum_{out} = p_sum_{in} + x_{in} * w$$



Build Systolic Array with Verilog HDL

```
module tpu_pe(
  input signed [7:0] IA_w,   input signed [7:0] weight_n,
  input signed [31:0] p_sum_n,  input clk, input [1:0] ctrl_in,
  //00 reset, 01 load, 10 compute, 11 idle
  output reg signed [7:0] IA_e,
  output reg signed [7:0] weight_s,
  output reg signed [31:0] p_sum_s);

always @(posedge clk)
begin
  case (ctrl_in)
  2'b00: begin IA_e <= 0; weight_s <= 0; p_sum_s <= 0; end
  2'b11: begin ; end
  2'b01: begin weight_s <= weight_n; p_sum_s <= 0; end
  2'b10: begin IA_e <= IA_w; p_sum_s <= $signed(p_sum_n) +
$signed(weight_s)*$signed(IA_w); end
  default: begin IA_e <= 0; weight_s <= 0; p_sum_s <= 0; end
  endcase
end
endmodule
```



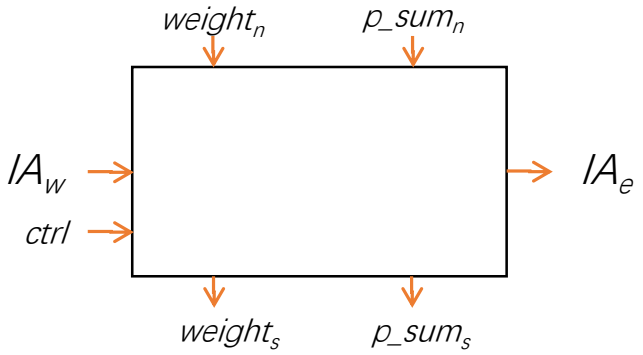
立志成才报国裕民

Testbench

```
module tpu_pe_tb;

reg signed [7:0] IA_w, weight_n;
reg signed [31:0] p_sum_n;
reg clk;
reg [1:0] ctrl_in;
wire [7:0] IA_e, weight_s;
wire [31:0] p_sum_s;

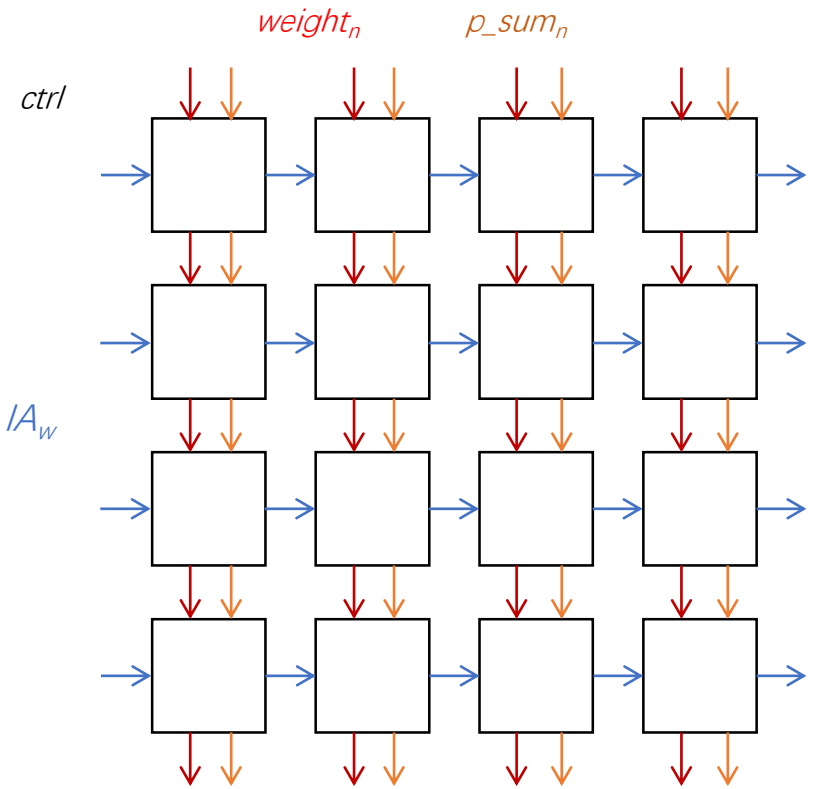
always #1 clk=~clk;
initial
begin
    clk=0;
    IA_w = 1;
    weight_n = 2;
    p_sum_n = 3;
    ctrl_in=0;   //test reset
    #2 IA_w = 1; //test load
    weight_n = 2;
    p_sum_n = 3;
    ctrl_in=1;
    #2 IA_w = 1;
    weight_n = 0;
    p_sum_n = 3;
    ctrl_in=2;   //test compute
    #2 IA_w = 1;
    weight_n = 2;
    p_sum_n = 3;
    ctrl_in=3;   //test idle
    #2 $finish;
end
tpu_pe DUT(
  IA_w,
  weight_n,
  p_sum_n,
  clk,
  ctrl_in,
  //00 reset, 01 load, 10 compute, 11 idle
  IA_e,
  weight_s,
  p_sum_s);
initial
begin
    $dumpfile("pe.vcd");     //生成的vcd文件名称
    $dumpvars(0, tpu_pe_tb); //tb模块名称
end
endmodule
```



立志成才报国裕民

PE Array – Structural Model

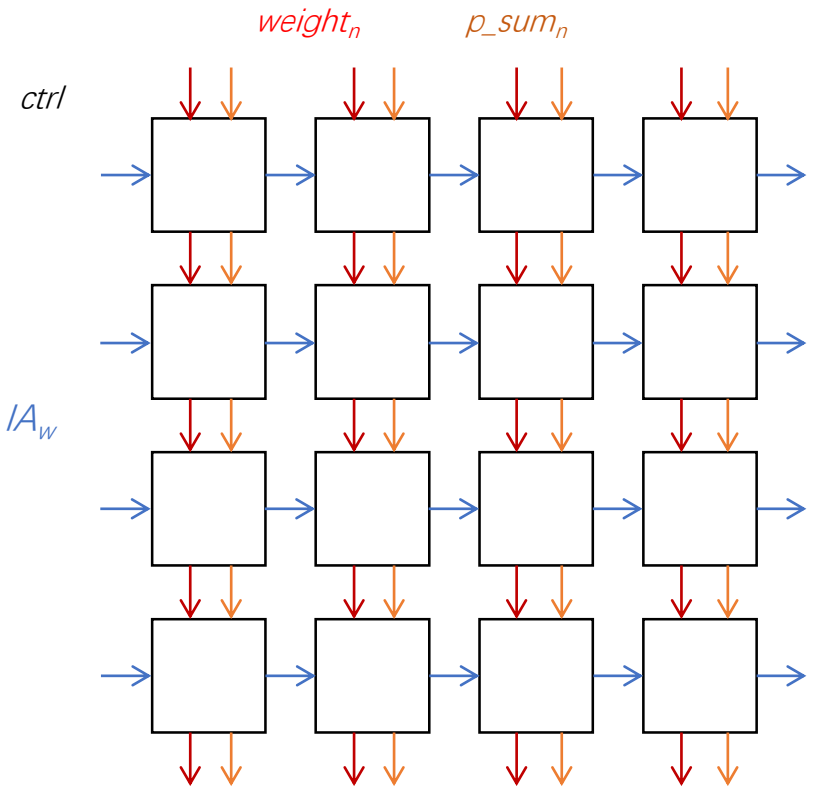
```
module tpu_datapath(
    input clk, input [1:0] ctrl,
    input [31:0] IA,  input [31:0] weight,
    output [127:0] p_sum);
//internal signals
wire [127:0] p_sum_temp [4:0];
wire [31:0] IA_temp [4:0];
wire [31:0] weight_temp [4:0];
assign IA_temp[4] = IA;
assign weight_temp[4] = weight;
assign p_sum_temp[4] = 0;
assign p_sum = p_sum_temp[0];
```



立志成才报国裕民

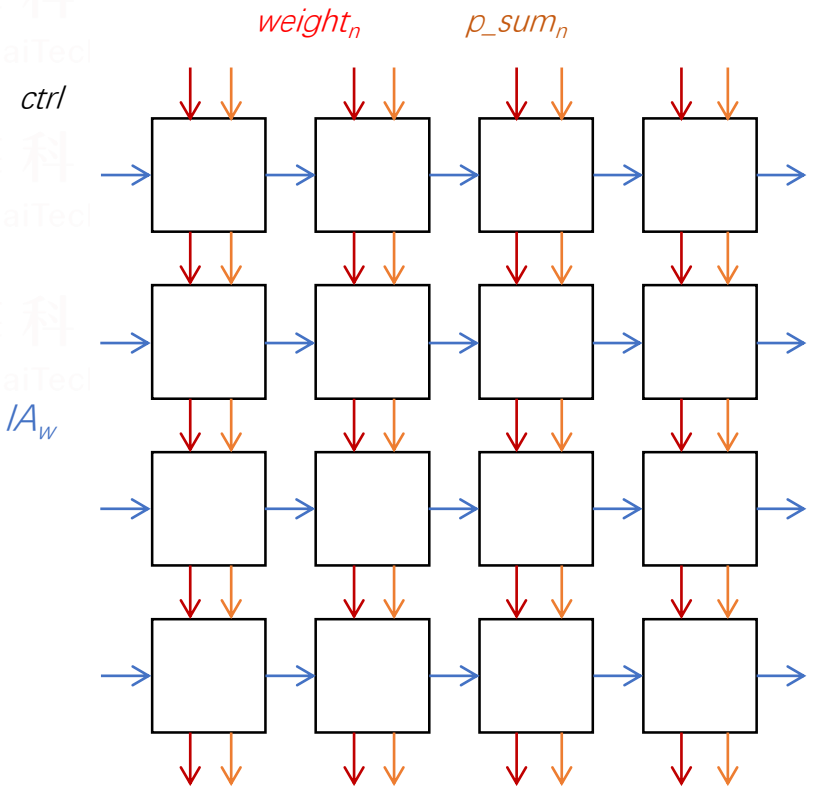
PE Array – Structural Model (Cont'd)

```
genvar i;
generate
  for (i=0; i<4; i=i+1) begin
    genvar j;
    for (j=0; j<4; j=j+1) begin
      tpu_pe tpu_pe_ij(
        .IA_w(IA_temp[4-j][(31-i*8):(24-i*8)]),
        .weight_n(weight_temp[4-i][(31-j*8):(24-j*8)]),
        .p_sum_n(p_sum_temp[4-i][(127-j*32):(96-j*32)]),
        .clk(clk), .ctrl_in(ctrl),
        //00 reset, 01 load, 10 compute, 11 idle
        .IA_e(IA_temp[3-j][(31-i*8):(24-i*8)]),
        .weight_s(weight_temp[3-i][(31-j*8):(24-j*8)]),
        .p_sum_s(p_sum_temp[3-i][(127-j*32):(96-j*32)]));
    end
  end
endgenerate
endmodule
```



PE Array Controller – FSM

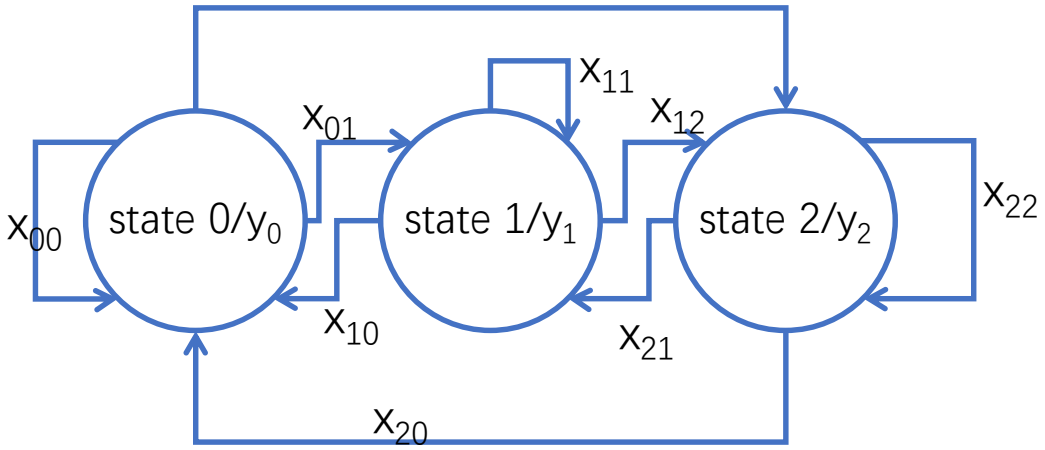
- Assume
  - Data/input are pre-arranged
  - All the PEs execute the same instruction (control sig.)
  - Return to “idle” state when no valid instruction received
  - No “double-buffering”



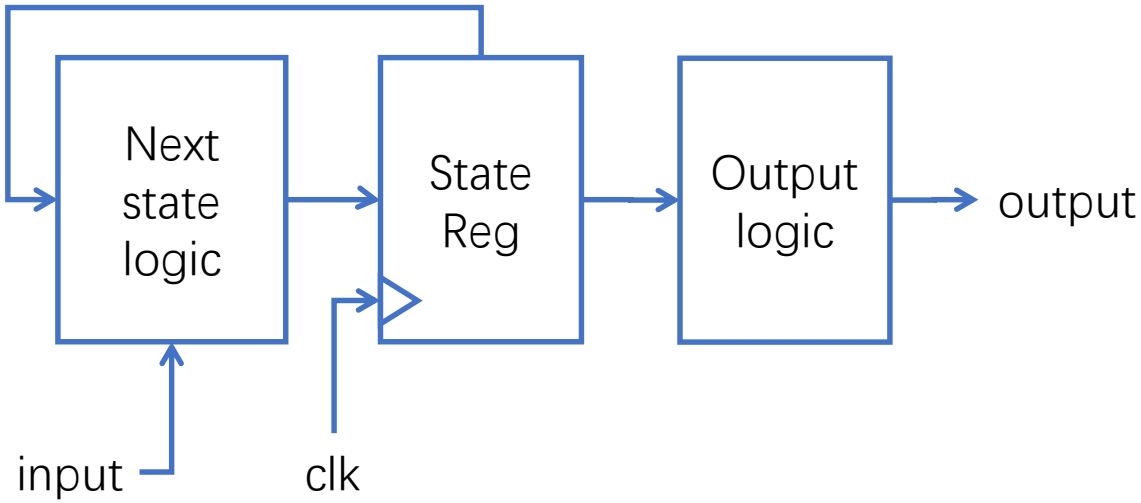
立志成才报国裕民

Finite State Machine (FSM)

- **Moore Machine**
- Input associated with each transition line;
- Output associated with each state

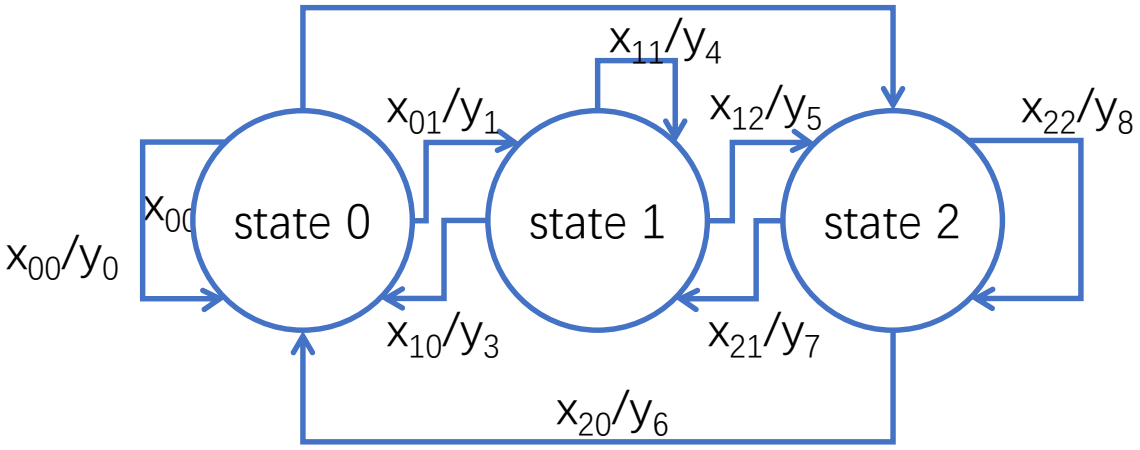


A 3-state FSM transition diagram

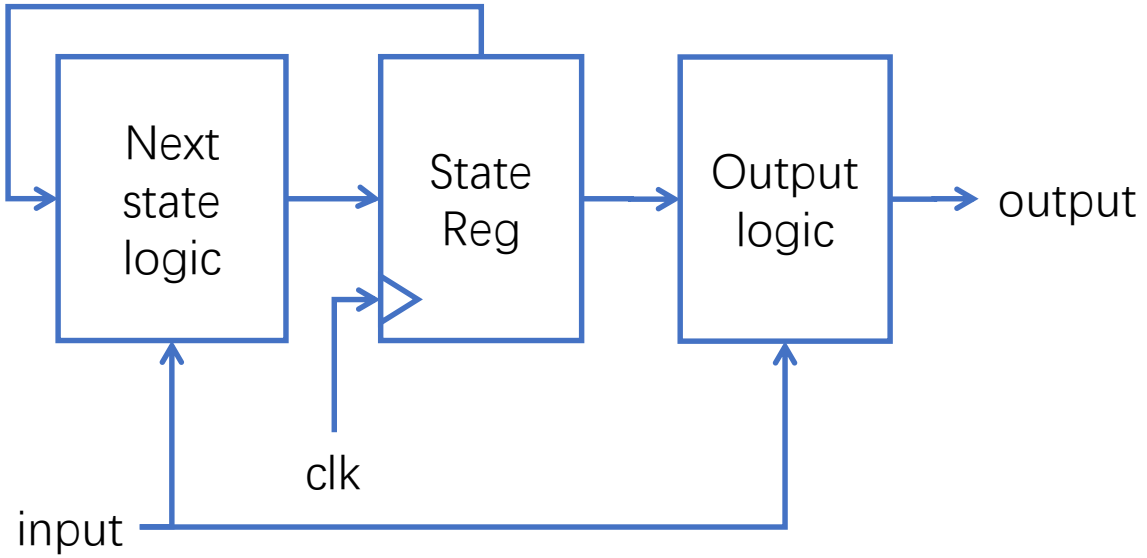


Finite State Machine (FSM) – Mealy Machine

- Input and output associated with each state transition line;



A 3-state FSM
transition diagram

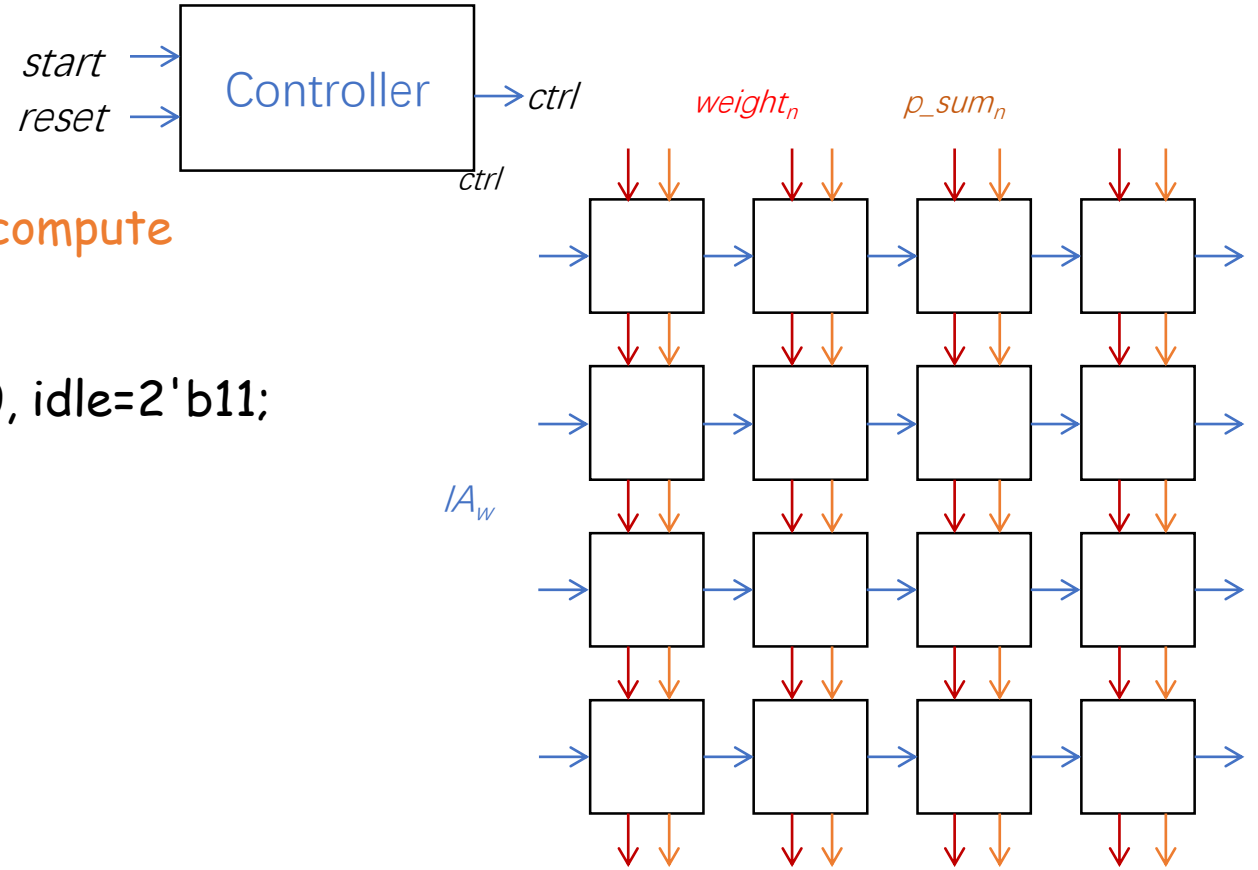


PE Array Controller – FSM (Cont'd)

```
module tpu_ctrl(
  input clk,
  input reset,
  input [1:0] start, //01 to load weight, 10 to compute
  output reg [1:0] ctrl);

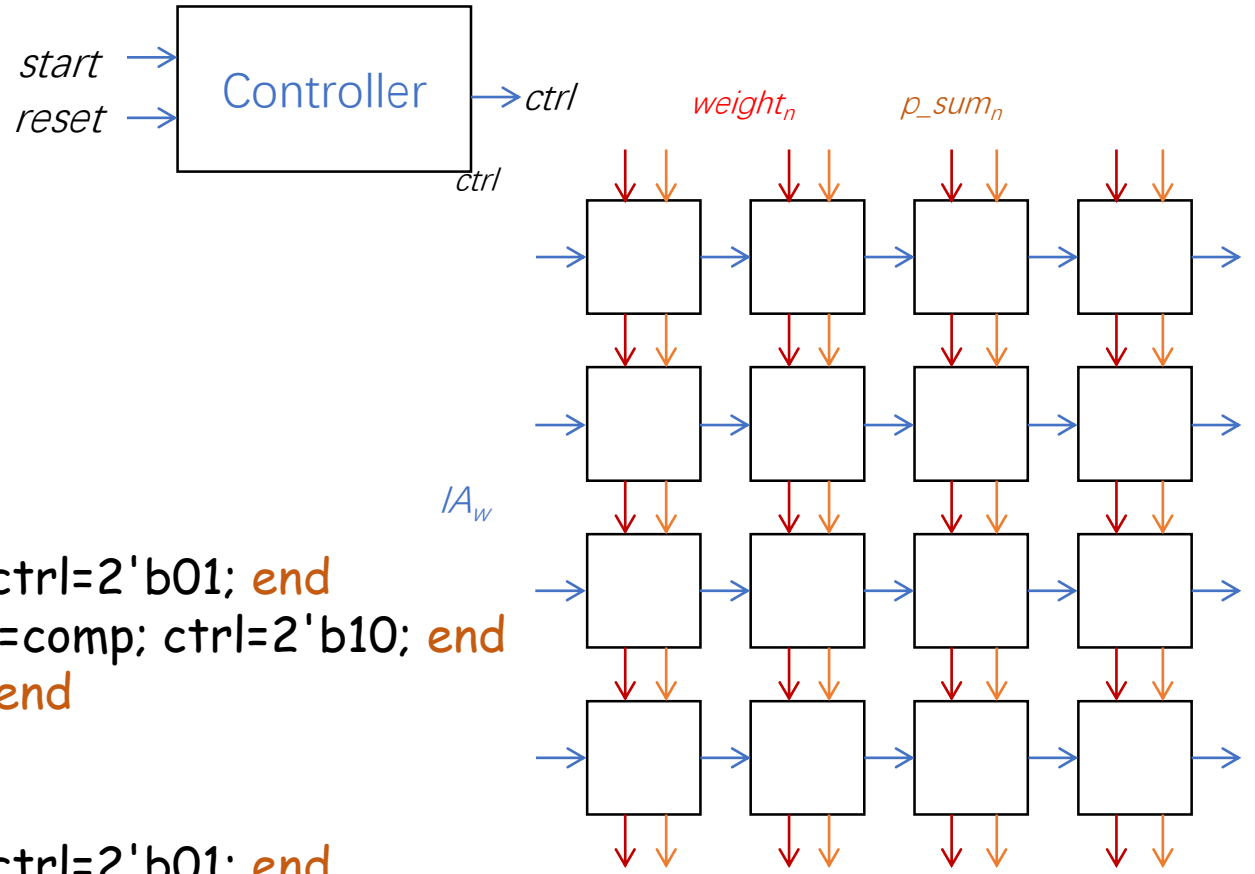
parameter rst=2'b00, ld = 2'b01, comp=2'b10, idle=2'b11;
reg [1:0] current_state, next_state;

//State update
always @(posedge clk)
begin
  if (reset) current_state <= rst;
  else current_state <= next_state;
end
```



PE Array Controller – FSM (Cont'd)

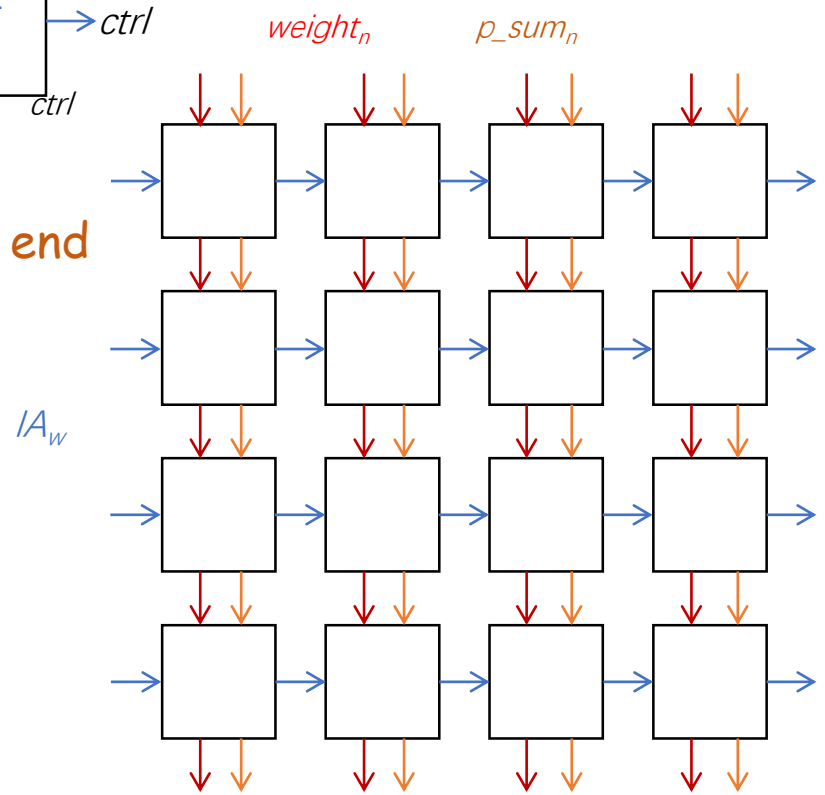
```
//Next_state logic & Output logic
always @(current_state, start)
begin
  case(current_state)
  rst: begin
         ctrl = 0;
         next_state=idle;
       end
  idle: begin
          if (start==2'b01) begin next_state=ld; ctrl=2'b01; end
          else if (start==2'b10) begin next_state=comp; ctrl=2'b10; end
          else begin next_state=idle; ctrl=2'b11; end
        end
  ld: begin
          if (start==2'b01) begin next_state=ld; ctrl=2'b01; end
          else if (start==2'b10) begin next_state=comp; ctrl=2'b10; end
          else begin next_state=idle; ctrl=2'b11; end
      end
```



PE Array Controller – FSM (Cont'd)



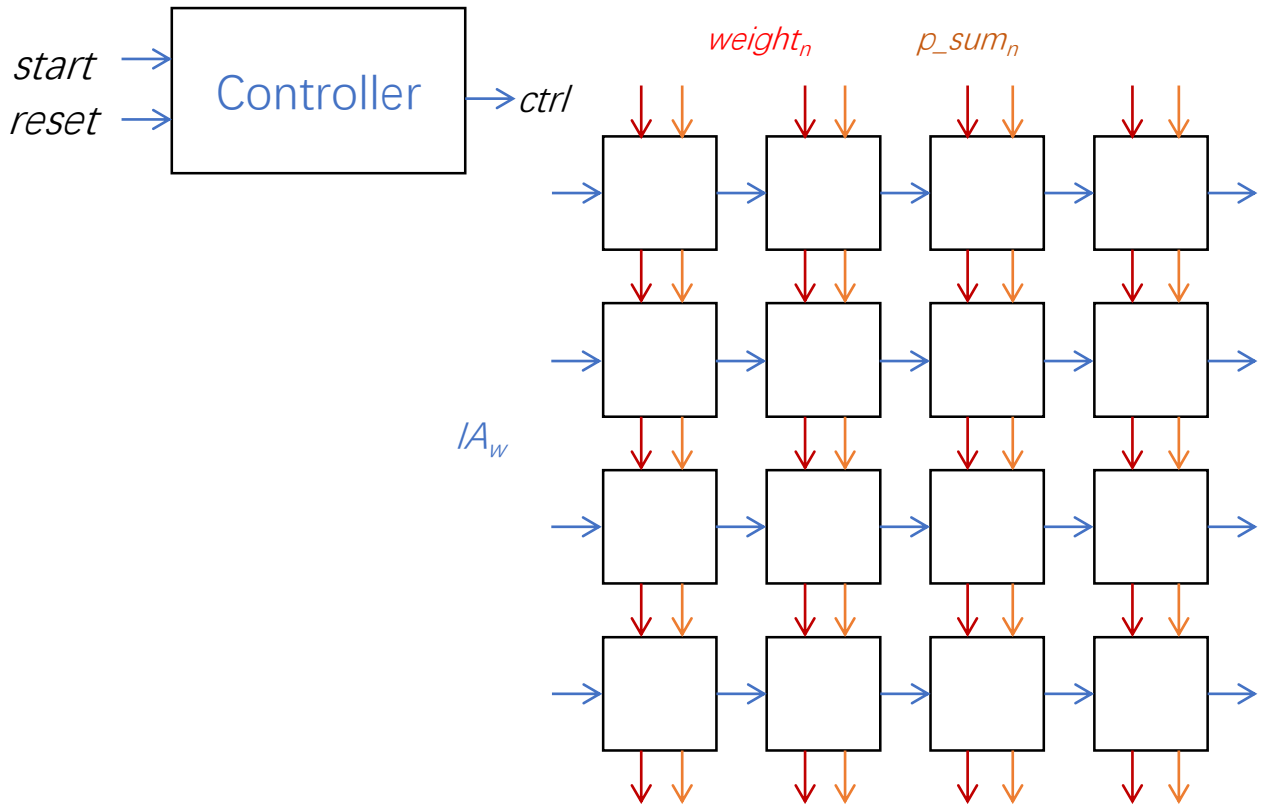

```
comp: begin
        if (start==2'b01) begin next_state=ld; ctrl=2'b01; end
        else if (start==2'b10) begin next_state=comp; ctrl=2'b10; end
        else begin next_state=idle; ctrl=2'b11; end
      end
default: begin
          ctrl = 2'b11;
          next_state=idle;
         end
endcase
end
endmodule
```



立志成才报国裕民

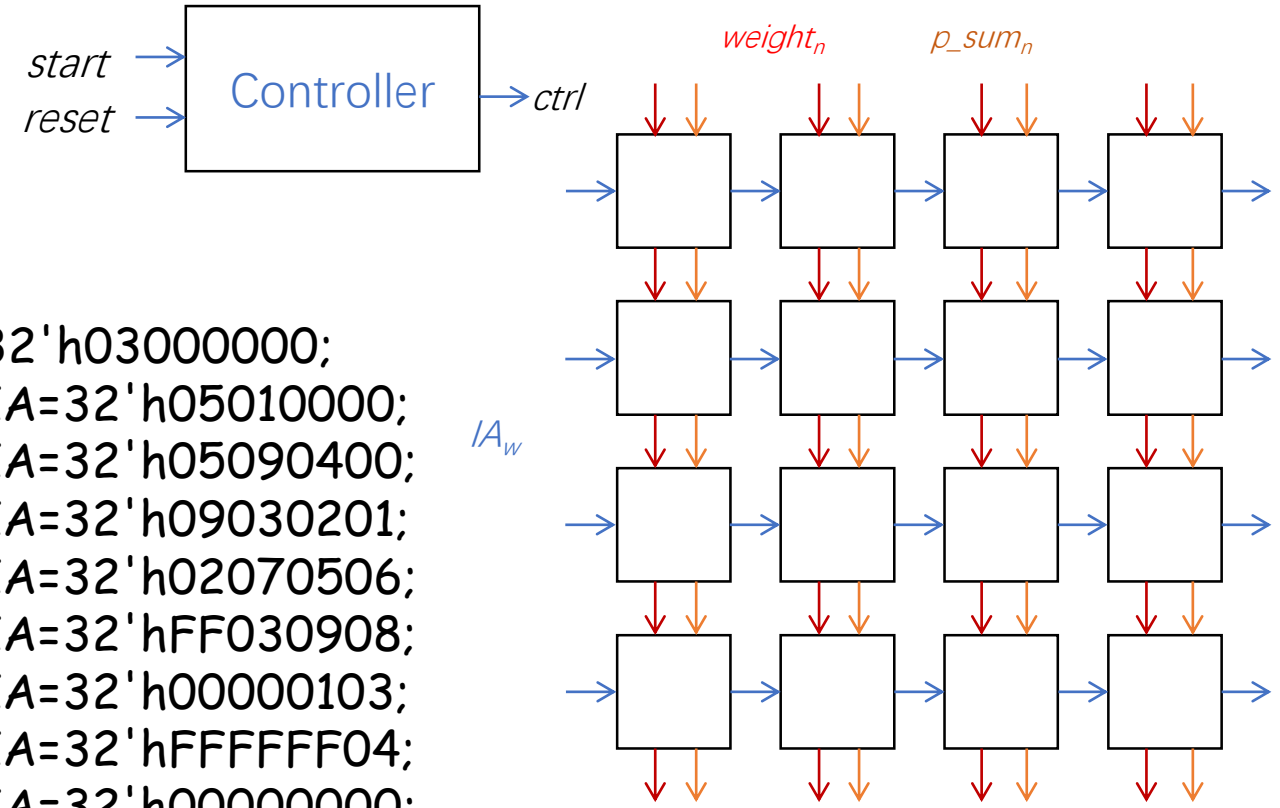
GEMM Array (Top module)

```
module tpu_top(
    input clk,
    input reset,
    input [1:0] start,
    input [31:0] IA,
    input [31:0] weight,
    output [127:0] p_sum);
wire [1:0] ctrl;
tpu_ctrl tpu_ctrl1(
  .clk(clk), .reset(reset),
  .start(start), .ctrl(ctrl));
tpu_datapath tpu_datapath1(
  .clk(clk), .ctrl(ctrl),
  .IA(IA), .weight(weight),
  .p_sum(p_sum));
endmodule
```



Testbench (Cont'd)

```
initial
begin
  reset=1;
  start=0;
  IA=0;
  weight=0;
  #6 reset=0;
  start=0;
  IA=0;
  weight=0;
  #2 reset=0;
  start=1;
  weight=32'h01020304;
  IA=32'h00000000;
  #2 weight=32'h04030201;
  #2 weight=32'h01020304;
  #2 weight=32'h04030201;
  #2 weight=32'h01020304;
  start=2;
  #2 weight=32'h04030201;
```

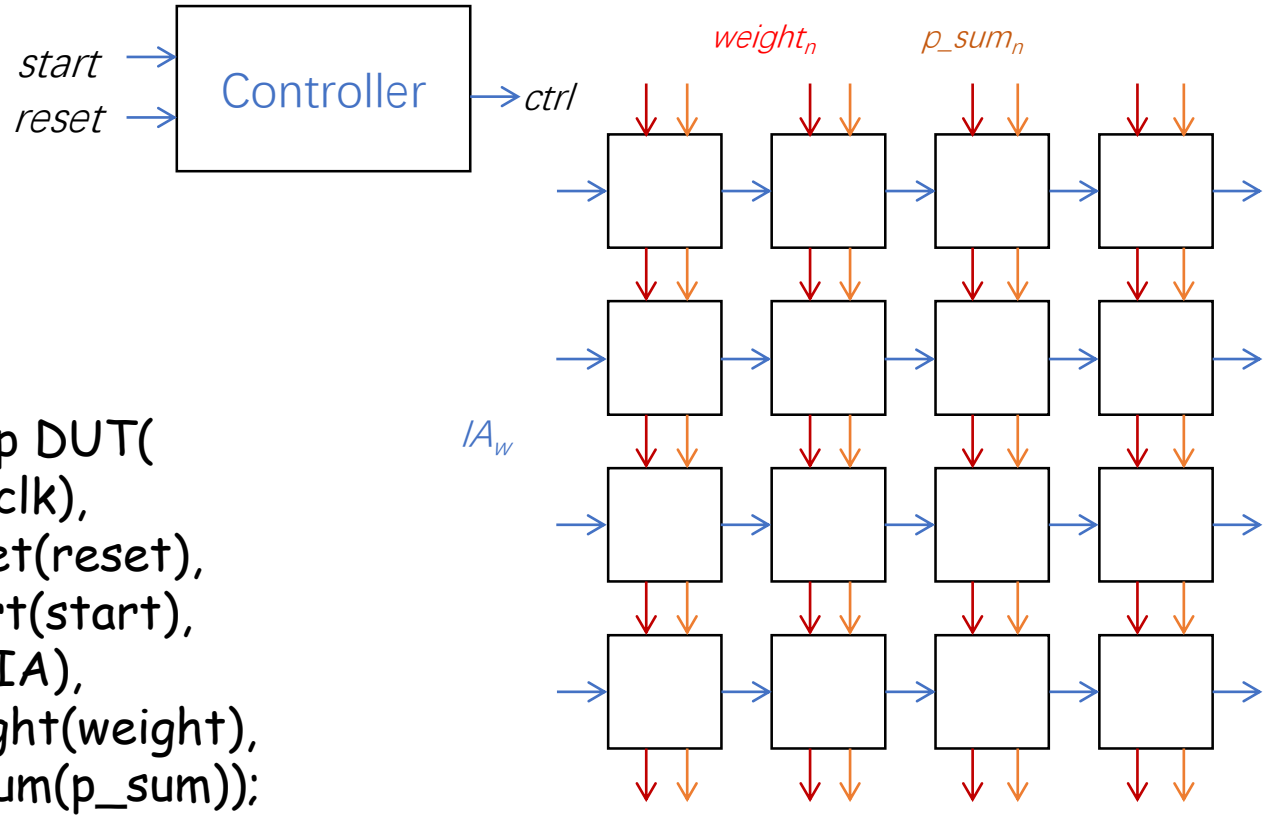



```
IA=32'h03000000;
#2 IA=32'h05010000;
#2 IA=32'h05090400;
#2 IA=32'h09030201;
#2 IA=32'h02070506;
#2 IA=32'hFF030908;
#2 IA=32'h00000103;
#2 IA=32'hFFFFFF04;
#2 IA=32'h00000000;
#2 IA=32'h0000FFFF;
```

立志成才报国裕民

Testbench (Cont'd)

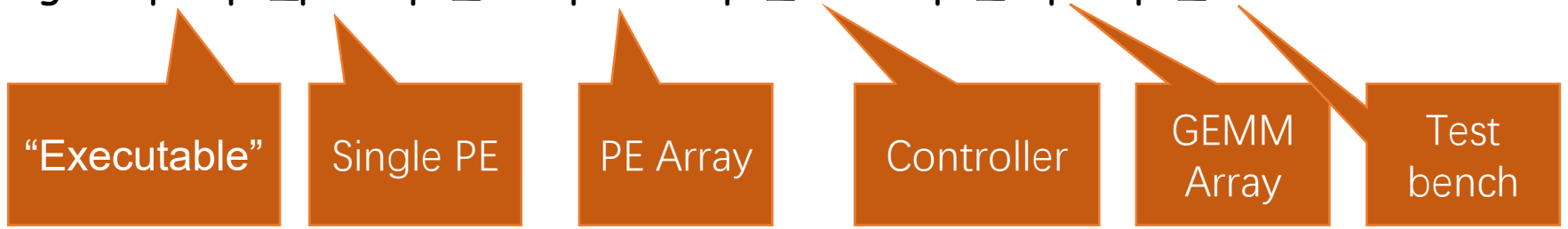
```
  #2 start =2;
  IA=32'h01000000;
  #2 IA=32'h04020000;
  #2 IA=32'h01030300;
  #2 IA=32'h04020204;
  #2 IA=32'h00030301;
  #2 IA=32'h00000204;
  #2 IA=32'h00000001;
  #2 IA=0;
end
initial
begin
    $dumpfile("tpu.vcd");
    $dumpvars(0, tpu_tb);
end
```




```
tpu_top DUT(
    .clk(clk),
    .reset(reset),
    .start(start),
    .IA(IA),
    .weight(weight),
    .p_sum(p_sum));
endmodule
```

iverilog & gtkwave

iverilog -o tpu tpu_pe.v tpu_datapath.v tpu_ctrl.v tpu_top.v tpu_tb.v in terminal



vvp tpu in terminal

C/C++-based tools for efficient simulation: verilator, Synopsys VCS, etc.



立志成才报国裕民