6×6 Pipelined Array Multiplier in Verilog

Concept Explanation: 6×6 Array Multiplier and Pipelining

6×6 Array Multiplier:

A 6×6 array multiplier multiplies two 6-bit binary numbers using an array of AND gates and adders.

- Each bit of one operand is ANDed with all bits of the other to form partial products.
- These are arranged in a matrix and added with carry-save adders.
- The final product is 12-bit wide.

Pipelining:

Pipelining divides the multiplication process into stages using flip-flops.

- Each stage processes a part of the computation and passes partial results forward.
- Increases throughput and clock frequency at the cost of higher area.

```
module pipelined_multiplier (
    input clk,
    input rst,
    input [5:0] A, B,
    output reg [11:0] P
);

    reg [5:0] A_reg, B_reg;
    reg [11:0] partial_sum1, partial_sum2;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            A_reg <= 0;
            B_reg <= 0;
        end else begin
            A_reg <= A;
            B_reg <= B;
        end
    end

    wire [11:0] pp [5:0];
    assign pp[0] = A_reg[0] ? {6'b0, B_reg} : 12'b0;
    assign pp[1] = A_reg[1] ? {5'b0, B_reg, 1'b0} : 12'b0;
    assign pp[2] = A_reg[2] ? {4'b0, B_reg, 2'b0} : 12'b0;
    assign pp[3] = A_reg[3] ? {3'b0, B_reg, 3'b0} : 12'b0;
    assign pp[4] = A_reg[4] ? {2'b0, B_reg, 4'b0} : 12'b0;
    assign pp[5] = A_reg[5] ? {1'b0, B_reg, 5'b0} : 12'b0;

    always @(posedge clk or posedge rst) begin
        if (rst)
            partial_sum1 <= 0;
        else
            partial_sum1 <= pp[0] + pp[1] + pp[2];
    end

    always @(posedge clk or posedge rst) begin
        if (rst)
            partial_sum2 <= 0;
        else
            partial_sum2 <= pp[3] + pp[4] + pp[5];
    end

    always @(posedge clk or posedge rst) begin
        if (rst)
            P <= 0;
        else
            P <= partial_sum1 + partial_sum2;
    end

endmodule
```

```
`timescale 1ns/1ps

module tb_pipelined_multiplier;

    reg clk, rst;
    reg [5:0] A, B;
    wire [11:0] P;

    pipelined_multiplier uut (
        .clk(clk),
        .rst(rst),
        .A(A),
        .B(B),
        .P(P)
    );

    always #5 clk = ~clk;

    initial begin
        $monitor("Time = %0t | A = %d, B = %d => P = %d", $time, A, B, P);
        clk = 0;
        rst = 1;
        A = 0;
        B = 0;

        #10 rst = 0;
        #10 A = 6'd12; B = 6'd15;
        #10 A = 6'd25; B = 6'd7;
        #10 A = 6'd63; B = 6'd63;
        #10 A = 6'd0;  B = 6'd45;
        #50 $stop;
    end

endmodule
```