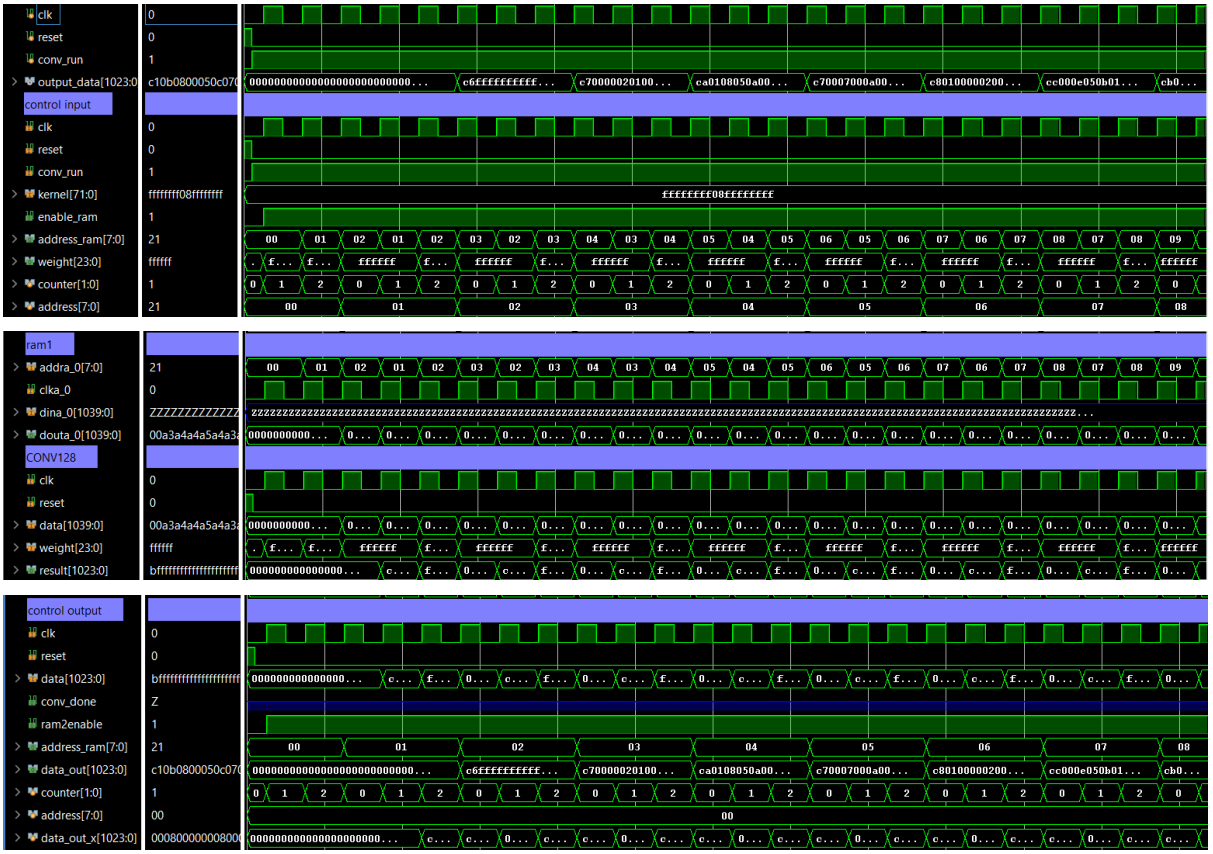
Top Module:

```
`timescale 1ns / 1ps

module TOP(
    input clk,
    input reset,
    input conv_run,
    output [1024:0] output_data
    );

    wire ram1_enable;
    wire notram2_enable;
    wire ram2enablew;
    wire [7:0] ram1_address;
    wire [7:0] ram2_address;
    wire [23:0] weight;

    wire [1039:0] input_data;
    wire [1023:0] to_output_c;
//    wire [1023:0] output_data;

    control_input ci (.clk(clk), .reset(reset), .conv_run(conv_run),

.kernel({8'b11111111,8'b11111111,8'b11111111,8'b11111111,
8'b00001000,8'b11111111,8'b11111111,8'b11111111,8'b11111111}),
                  .enable_ram(ram1_enable), .address_ram(ram1_address),
                  .weight(weight));

    bram1 r1 (.addra_0(ram1_address), .clka_0(clk),
.ena_0(ram1_enable), .douta_0(input_data));

    CONV128 conv128 (.clk(clk), .reset(reset), .data(input_data),
.weight(weight), .result(to_output_c));

    output_control outp (.clk(clk), .reset(reset), .data(to_output_c),
.conv_done(notram2_enable),
                         .address_ram(ram2_address),
.data_out(output_data), .ram2enable(ram2enablew));

    bram2 r2 (.clka_0(clk), .addra_0(ram2_address[6:0]),
.dina_0(output_data), .ena_0(~notram2_enable), .wea_0(ram2enablew));
```

```
endmodule
```

I used this module to wire my units.



Control Input

```
`timescale 1ns / 1ps

module control_input(
    input clk,
    input reset,
    input conv_run,
    input [71:0] kernel,
    output reg  enable_ram,
    output reg [7:0] address_ram,
    output reg [23:0] weight
    );

    reg [1:0] counter;
    reg [7:0] address;

    always @(posedge clk or posedge reset)begin
```

```
        if(reset)begin
            address <= 0;
            address_ram <= 0;
            enable_ram <= 0;
            counter <= 0;
            weight <= 0;
        end
        else if (conv_run) begin
            enable_ram <= 1;
                if(counter == 0)begin
                    address_ram <= address + counter;
                    weight <= kernel[71:48];
                    counter <= counter + 1;
                end else if  (counter == 1)begin
                    address_ram <= address + counter;
                    weight <= kernel[47:24];
                    counter <= counter + 1;
                end else if(counter == 2)begin
                    address_ram <= address + counter;
                    weight <= kernel [23:0];
                    address <= address + 1;
                    counter <= 0;
                end

        end
    end
endmodule
```

This module is used to change the address from which data is read from RAM and to update the kernel going to `conv128`. The weight is modified every cycle, and the address is incremented every three cycles.

CONV128

```
`timescale 1ns / 1ps


module CONV128(
    input clk,reset,
    input  [1039:0] data,
    input [23:0] weight,
    output  [1023:0] result
```

```
    );

    genvar i;
    generate
        for(i=0; i<128; i=i+1)begin
            CONV conv(.data(24'b0 + data[i*8+23 : i*8]),
.weight(weight), .clk(clk), .reset(reset), .result( result[i*8+7:i*8]
));
        end
    endgenerate
endmodule
```

This module is employed to compute the convolution of an entire row. It achieves this by utilizing the data retrieved from RAM and the weight obtained from the control input, updating the convolution result accordingly.

CONV

```
`timescale 1ns / 1ps


module CONV(
    input [23:0] data, weight,
    input clk, reset,
    output [7:0] result
    );

    wire[19:0] resultmac;
    MAC mac(.clk(clk), .reset(reset), .data(data), .weight(weight),
.result(resultmac));
    MAC_Normalize nmac(.data(resultmac), .result(result));

endmodule
```

This module takes the MAC (Multiply-Accumulate) of three 8-bit data inputs and subsequently normalizes the result.

Control Output

```
`timescale 1ns / 1ps
```

```
module output_control(
    input clk,
    input reset,
    input [1023:0] data,
    output conv_done,
    output reg ram2enable,
    output reg [7:0] address_ram,
    output reg [1023:0] data_out
    );


    reg [1:0] counter;
    reg [7:0] address;
    reg [1023:0] data_out_x;

    always @(posedge clk or posedge reset)begin
        if(reset)begin
            ram2enable <= 0;
            address <= 0;
            address_ram <= 0;
            counter <= 0;
            data_out_x <= 0;
            data_out <= 0;
        end else begin
            if(counter == 0) begin
                ram2enable <= 0;
                data_out_x <= data;
                counter <= counter +1;
            end else if(counter == 1) begin
                ram2enable <= 0;
                data_out_x <= data_out_x + data;
                counter <= counter +1;
            end else if(counter == 2) begin
                ram2enable <= 1;
                data_out_x <= data_out_x + data;
                data_out <= data_out_x;
                counter <= 0;
                address_ram <= address_ram +1;
            end

        end
    end
```

```
endmodule
```

This module accumulates the three data inputs received from `conv128`, sending them for writing to RAM. On the third clock cycle, the RAM is enabled, aiming to write the accumulated data to `ram2`. Additionally, the address is incremented every three clock cycles.

Testbench top:

```
`timescale 1ns / 1ps

module tb_TOP;

    reg clk;
    reg reset;
    reg conv_run;
    wire [1023:0] output_data;
    wire  conv_done;
        integer file;
    TOP TOP_inst (
        .clk(clk),
        .reset(reset),
        .conv_run(conv_run),
        .output_data(output_data),
        .conv_done(conv_done)
    );

    initial begin
        file = $fopen ("isim_3.txt", "w");
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        reset = 1;
        #2 reset = 0;
    end

    always@(output_data)begin
    if (conv_done == 0)$fwrite (file, "%h\n", output_data);
    end
    initial begin
```

```
        conv_run = 0;
        #2 conv_run = 1;


        #100;

        #3000;

        $finish;
    end

endmodule
```

Result: