Lecture 01. 1D Filter

filter_float.c

```
#include <stdio.h>

float filter(double in) {
    static float    x[21];
    float   h[21] = {-0.000306021779757585,      0.00189666282674638,
                      0.00257299891941012,      -0.0090277441097318,
                     -0.0075198095107301,        0.0291019349476909,
                      0.0145629522024416,       -0.0807110457423653,
                     -0.0210214194708373,        0.308909103630076,
                      0.523660422383779,         0.308909103630076,
                     -0.0210214194708373,       -0.0807110457423653,
                      0.0145629522024416,        0.0291019349476909,
                     -0.0075198095107301,       -0.0090277441097318,
                      0.00257299891941012,       0.00189666282674638,
                     -0.000306021779757585};

    for(int i=20;i>=1;i--) {
        x[i] = x[i-1];
    }
    x[0] = in;

    float   out = 0;
    for(int i=0;i<21;i++) {
        out += x[i] * h[i];
    }
    return out;
}

int main(void) {
    FILE        *inf, *outf;
    inf = fopen("filter_in.txt", "r");
    outf = fopen("filter_out.txt", "w");

    float   in;
    while(fscanf(inf, "%f", &in) > 0) {
        float out;
        out = filter(in);
        fprintf(outf, "%f\n", out);
    }
    fclose(inf);
    fclose(outf);
}
```

filter_fixed.c

```
#include <stdio.h>
#include <math.h>

int filter(int in) {
    static int  x[21];
    int     h[21] = {  -10,    62,    84,  -296,  -246,
                       954,   477, -2645,  -689, 10122,
                     17159, 10122,  -689, -2645,   477,
                       954,  -246,  -296,    84,    62,
                       -10};
    for(int i=20;i>=1;i--) {
        x[i] = x[i-1];
    }
    x[0] = in;
    int     out = 0;
    for(int i=0;i<21;i++) {
        out += x[i] * h[i];
    }
    out = (out+(1<<15)) >> 16;
    if(out > 32767) out = 32767;
    else if(out < -32767) out = -32767;
    return out;
}

int main(void) {
    FILE        *inf, *outf;
    FILE        *fixed_in, *fixed_out;
    inf = fopen("filter_in.txt", "r");
    outf = fopen("filter_out.txt", "w");
    fixed_in = fopen("filter_in_fixed.txt", "w");
    fixed_out = fopen("filter_out_fixed.txt", "w");

    float   in_f;
    while(fscanf(inf, "%f", &in_f) > 0) {
        int in_i = (int)floor(in_f*8192+0.5);
        if(in_i > 32767) in_i = 32767;
        else if(in_i < -32767) in_i = -32767;
        fprintf(fixed_in, "%04X\n", in_i & 0x0FFFF);
        int out_i;
        float   out_f;
        out_i = filter(in_i);
        //fprintf(fixed_out, "%11d\n", out_i);
        fprintf(fixed_out, "%04X\n", out_i & 0x0FFFF);
        out_f = out_i / 4096.;
        fprintf(outf, "%f\n", out_f);
    }
    fclose(inf);
    fclose(outf);
}
```

filter.v

```
module filter (
                input   clk,
                input   n_reset,
                input       [15:0]  x_in,
                output  reg [15:0]  y_out,

                input           h_write,
                input   [4:0]   h_idx,
                input   [15:0]  h_data
);

reg signed  [15:0]  x[20:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        x[0] <= 'b0;
        x[1] <= 'b0;
        x[2] <= 'b0;
        x[3] <= 'b0;
        x[4] <= 'b0;
        x[5] <= 'b0;
        x[6] <= 'b0;
        x[7] <= 'b0;
        x[8] <= 'b0;
        x[9] <= 'b0;
        x[10] <= 'b0;
        x[11] <= 'b0;
        x[12] <= 'b0;
        x[13] <= 'b0;
        x[14] <= 'b0;
        x[15] <= 'b0;
        x[16] <= 'b0;
        x[17] <= 'b0;
        x[18] <= 'b0;
        x[19] <= 'b0;
        x[20] <= 'b0;
    end else begin
        x[0] <= x_in;
        x[1] <= x[0];
        x[2] <= x[1];
        x[3] <= x[2];
        x[4] <= x[3];
        x[5] <= x[4];
        x[6] <= x[5];
        x[7] <= x[6];
        x[8] <= x[7];
        x[9] <= x[8];
        x[10] <= x[9];
        x[11] <= x[10];
        x[12] <= x[11];
        x[13] <= x[12];
        x[14] <= x[13];
        x[15] <= x[14];
        x[16] <= x[15];
        x[17] <= x[16];
        x[18] <= x[17];
        x[19] <= x[18];
        x[20] <= x[19];
    end
end

/*
reg signed  [15:0]  h[20:0] =
        {-16'd   10,  16'd    62,  16'd    84, -16'd   296, -16'd   246,
          16'd  954,  16'd   477, -16'd  2645, -16'd   689,  16'd 10122,
          16'd17159,  16'd 10122, -16'd   689, -16'd  2645,  16'd   477,
          16'd  954, -16'd   246, -16'd   296,  16'd    84,  16'd    62,
         -16'd   10};
*/
reg signed  [15:0]  h[20:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        h[ 0] <= -16'd   10;
        h[ 1] <=  16'd   62;
        h[ 2] <=  16'd   84;
        h[ 3] <= -16'd  296;
        h[ 4] <= -16'd  246;
        h[ 5] <=  16'd  954;
        h[ 6] <=  16'd  477;
        h[ 7] <= -16'd 2645;
        h[ 8] <= -16'd  689;
        h[ 9] <=  16'd10122;
        h[10] <=  16'd17159;
        h[11] <=  16'd10122;
        h[12] <= -16'd  689;
        h[13] <= -16'd 2645;
        h[14] <=  16'd  477;
        h[15] <=  16'd  954;
        h[16] <= -16'd  246;
        h[17] <= -16'd  296;
        h[18] <=  16'd   84;
        h[19] <=  16'd   62;
        h[20] <= -16'd   10;
    end else begin
        if(h_write == 1'b1) begin
            h[h_idx] <= h_data;
        end
    end
end

wire signed [31:0]  mul[20:0];  //(16,13) * (16,15) -> (31,28) if sym. sat.
                                  //(32, 28) sum
assign  mul[0] = x[0] * h[0];
assign  mul[1] = x[1] * h[1];
assign  mul[2] = x[2] * h[2];
assign  mul[3] = x[3] * h[3];
assign  mul[4] = x[4] * h[4];
assign  mul[5] = x[5] * h[5];
assign  mul[6] = x[6] * h[6];
assign  mul[7] = x[7] * h[7];
assign  mul[8] = x[8] * h[8];
assign  mul[9] = x[9] * h[9];
assign  mul[10] = x[10] * h[10];
assign  mul[11] = x[11] * h[11];
assign  mul[12] = x[12] * h[12];
assign  mul[13] = x[13] * h[13];
assign  mul[14] = x[14] * h[14];
assign  mul[15] = x[15] * h[15];
assign  mul[16] = x[16] * h[16];
assign  mul[17] = x[17] * h[17];
```

```
assign  mul[18] = x[18] * h[18];
assign  mul[19] = x[19] * h[19];
assign  mul[20] = x[20] * h[20];

wire signed [31:0]  sum = mul[0] + mul[1] + mul[2] + mul[3] + mul[4] +
                          mul[5] + mul[6] + mul[7] + mul[8] + mul[9] +
                          mul[10] + mul[11] + mul[12] + mul[13] + mul[14] +
                          mul[15] + mul[16] + mul[17] + mul[18] + mul[19] +
                          mul[20];

wire signed [31:0]  y_rnd = sum + (1<<15);  // for rounding
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        y_out <= 'b0;
    end else begin
        y_out <= y_rnd[31:16];
    end
end

endmodule
```

top_filter.v

```
module top_filter;

reg clk, n_reset;

initial clk = 1'b0;
always #5 clk = ~clk;

reg [15:0]  in_data[0:95999];

reg         [15:0]  x_in;
wire signed [15:0]  y_out;
integer idx;
initial begin
    n_reset = 1'b1;
    idx = 0;
    x_in = 0;
    $readmemh("../c/filter_in_fixed.txt", in_data);
    #3;
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    for(idx=0;idx<96000;idx=idx+1) begin
        x_in = in_data[idx];
        @(posedge clk);
    end
    x_in = 0;
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    $finish;
end

filter  i_filter (.clk(clk), .n_reset(n_reset), .x_in(x_in), .y_out(y_out));

always@(posedge clk) begin
    $display("%d", y_out);
end

endmodule
```

filter.v - prameterized

```
// parameter
module filter #(
                NUM_TAP = 21,
                W_ID = 16,
                F_ID = 13,
                W_C = 16,
                F_C = 15,
                W_OD = 16,
                F_OD = 12
) (
                input   clk,
                input   n_reset,
                input       [W_ID-1:0]  x_in,
                output  reg [W_OD-1:0]  y_out,

                input                               h_write,
                input   [$clog2(NUM_TAP)-1:0]   h_idx,
                input   [W_C-1:0]               h_data
);

reg signed  [W_ID-1:0]  x[NUM_TAP-1:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        for(int i=0;i<NUM_TAP;i++) begin
            x[i] <= 'b0;
        end
    end else begin
        x[0] <= x_in;
        for(int i=1;i<NUM_TAP;i++) begin
            x[i] <= x[i-1];
        end
    end
end

/*
reg signed  [15:0]  h[NUM_TAP-1:0] =
        {-16'd   10,  16'd    62,  16'd    84, -16'd   296, -16'd   246,
          16'd  954,  16'd   477, -16'd  2645, -16'd   689,  16'd 10122,
          16'd17159,  16'd 10122, -16'd   689, -16'd  2645,  16'd   477,
          16'd  954, -16'd   246, -16'd   296,  16'd    84,  16'd    62,
         -16'd   10};
*/
reg signed  [W_C-1:0]   h[NUM_TAP-1:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        h[ 0] <= -16'd   10;
        h[ 1] <=  16'd   62;
        h[ 2] <=  16'd   84;
        h[ 3] <= -16'd  296;
        h[ 4] <= -16'd  246;
        h[ 5] <=  16'd  954;
        h[ 6] <=  16'd  477;
        h[ 7] <= -16'd 2645;
        h[ 8] <= -16'd  689;
        h[ 9] <=  16'd10122;
        h[10] <=  16'd17159;
        h[11] <=  16'd10122;
        h[12] <= -16'd  689;
        h[13] <= -16'd 2645;
        h[14] <=  16'd  477;
        h[15] <=  16'd  954;
        h[16] <= -16'd  246;
        h[17] <= -16'd  296;
        h[18] <=  16'd   84;
        h[19] <=  16'd   62;
        h[20] <= -16'd   10;
    end else begin
        if(h_write == 1'b1) begin
            h[h_idx] <= h_data;
        end
    end
end

localparam  W_MUL = W_ID + W_C - 1;
localparam  F_MUL = F_ID + F_C;
localparam  W_SUM = W_MUL + $clog2(NUM_TAP);

wire signed [W_MUL-1:0] mul[NUM_TAP-1:0];

genvar  i;
for(i=0;i<NUM_TAP;i++) begin
    assign  mul[i] = x[i] * h[i];
end

reg signed  [W_SUM-1:0] sum;
always@(*) begin
    sum = 0;
    for(int i=0;i<NUM_TAP;i++) begin
        sum = sum + mul[i];
    end
end

wire signed [W_SUM-1:0] y_rnd = sum + (1<<(F_MUL-F_OD-1));  // for rounding
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        y_out <= 'b0;
    end else begin
        y_out <= y_rnd[F_MUL-F_OD+W_OD:F_MUL-F_OD];
    end
end

endmodule
```

Lecture 02. 2D Filter

filter_2d.c

```
#include <stdio.h>
#include <math.h>

void filter2d(unsigned char in_img[], unsigned char out_img[],
              int height, int width) {
    int     h[3][3] = {0x08, 0x10, 0x08, 0x10, 0x20, 0x10, 0x08, 0x10, 0x08};
    for(int i=0;i<height;i++) {
        for(int j=0;j<width;j++) {
            int sum = 0;
            if(i>0 && j>0)              sum += in_img[(i-1)*width+j-1]*h[0][0];
            if(i>0)                     sum += in_img[(i-1)*width+j  ]*h[0][1];
            if(i>0 && j<width-1)        sum += in_img[(i-1)*width+j+1]*h[0][2];
            if(j>0)                     sum += in_img[(i  )*width+j-1]*h[1][0];
                                        sum += in_img[(i  )*width+j  ]*h[1][1];
            if(j<width-1)               sum += in_img[(i  )*width+j+1]*h[1][2];
            if(i<height-1 && j>0)       sum += in_img[(i+1)*width+j-1]*h[2][0];
            if(i<height-1)              sum += in_img[(i+1)*width+j  ]*h[2][1];
            if(i<height-1 && j<width-1) sum += in_img[(i+1)*width+j+1]*h[2][2];
            sum = (sum + (1<<6)) >> 7;
            if(sum < 0) out_img[i*width+j] = 0;
            else if(sum > 255) out_img[i*width+j] = 255;
            else out_img[i*width+j] = sum;
        }
    }
}

int main(void) {
    int         i, a;
    FILE        *inf, *outf, *memf;
    unsigned char   in_img[256*256];
    unsigned char   out_img[256*256];
    inf = fopen("img_in.txt", "r");
    outf = fopen("img_out.txt", "w");
    memf = fopen("img_in.dat", "w");

    for(i=0;i<256*256;i++) {
        fscanf(inf, "%d,", &a);
        in_img[i] = a;
        fprintf(memf, "%02X\n", in_img[i]);
    }

    filter2d(in_img, out_img, 256, 256);

    for(i=0;i<256*256;i++) {
        fprintf(outf, "%3d ", out_img[i]);
        if(i%256 == 255) fprintf(outf, "\n");
    }

    fclose(inf);
    fclose(outf);
    fclose(memf);
}
```

filter2d.v - without buffer

```
module filter2d (
                input           clk,
                input           n_reset,
                input           start,
                output  reg     finish,

                output          cs,
                output          we,
                output  [16:0]  addr,
                output  [7:0]   din,
                input   [7:0]   dout,

                input           h_write,
                input   [3:0]   h_idx,
                input   [7:0]   h_data
);

reg         on_proc;
reg [3:0]   cnt;
reg [7:0]   cnt_x;
reg [7:0]   cnt_y;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        on_proc <= 1'b0;
        cnt <= 0;
        cnt_x <= 0;
        cnt_y <= 0;
        finish <= 1'b0;
    end else begin
        if(start == 1'b1) on_proc <= 1'b1;
        else if((cnt == 11) && (cnt_x == 255) && (cnt_y == 255)) on_proc <= 1'b0;

        if(on_proc == 1'b1) begin
            cnt <= (cnt == 11) ? 0 : cnt+1;
            if(cnt == 11) begin
                cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
                if(cnt_x == 255) begin
                    cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1;
                end
            end
        end
        finish <= ((cnt == 11) && (cnt_x == 255) && (cnt_y == 255));
    end
end

wire            mem_rd = (cnt >= 0) && (cnt <= 8) && (on_proc == 1'b1);
reg     [16:0]  rd_addr;
always@(*) begin
    case(cnt)
        4'd0:   rd_addr = (cnt_y-1)*256 + cnt_x-1;
        4'd1:   rd_addr = (cnt_y-1)*256 + cnt_x;
        4'd2:   rd_addr = (cnt_y-1)*256 + cnt_x+1;
        4'd3:   rd_addr = (cnt_y  )*256 + cnt_x-1;
        4'd4:   rd_addr = (cnt_y  )*256 + cnt_x;
        4'd5:   rd_addr = (cnt_y  )*256 + cnt_x+1;
        4'd6:   rd_addr = (cnt_y+1)*256 + cnt_x-1;
        4'd7:   rd_addr = (cnt_y+1)*256 + cnt_x;
        4'd8:   rd_addr = (cnt_y+1)*256 + cnt_x+1;
```

```
        default:    rd_addr = 'bx;
    endcase
end

reg     [7:0]   pd;
wire            pd_en = (cnt >= 1) && (cnt <= 9);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        pd <= 0;
    end else begin
        if(pd_en == 1'b1) pd <= dout;
    end
end

reg signed  [7:0]   h[0:8];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        h[0] <= 8'h08;
        h[1] <= 8'h10;
        h[2] <= 8'h08;
        h[3] <= 8'h10;
        h[4] <= 8'h20;
        h[5] <= 8'h10;
        h[6] <= 8'h08;
        h[7] <= 8'h10;
        h[8] <= 8'h08;
    end else begin
        if(h_write == 1'b1) begin
            h[h_idx] <= h_data;
        end
    end
end

wire signed [7:0]   coeff = h[cnt-2];
wire signed [15:0]  mul = pd * coeff;
reg  signed [19:0]  acc;
wire signed [19:0]  acc_in = (cnt == 1) ? 0 : mul + acc;
reg                 acc_en;

always@(*) begin
    acc_en = 1'b0;
    case(cnt)
        4'd 1: acc_en = 1'b1;
        4'd 2: if((cnt_y > 0) && (cnt_x >   0)) acc_en = 1'b1;
        4'd 3: if((cnt_y > 0)                 ) acc_en = 1'b1;
        4'd 4: if((cnt_y > 0) && (cnt_x < 255)) acc_en = 1'b1;
        4'd 5: if(cnt_x >   0)  acc_en = 1'b1;
        4'd 6:                  acc_en = 1'b1;
        4'd 7: if(cnt_x < 255)  acc_en = 1'b1;
        4'd 8: if((cnt_y < 255) && (cnt_x >   0)) acc_en = 1'b1;
        4'd 9: if((cnt_y < 255)                 ) acc_en = 1'b1;
        4'd10: if((cnt_y < 255) && (cnt_x < 255)) acc_en = 1'b1;
        default: acc_en = 1'b0;
    endcase
end

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        acc <= 'b0;
    end else begin
        if(acc_en == 1'b1) acc <= acc_in;
    end
end

wire    [19:0]  pd_rnd_1 = acc + (1<<6);
wire    [12:0]  pd_rnd = pd_rnd_1[19:7];
wire    [7:0]   pd_out = (pd_rnd < 0) ? 0 :
                         (pd_rnd > 255) ? 255 :
                         pd_rnd[7:0];
assign          din = pd_out;

wire            mem_wr = (cnt == 11);
wire    [16:0]  wr_addr = cnt_y * 256 + cnt_x + 256*256;

assign  cs = mem_rd | mem_wr;
assign  we = mem_wr;
assign  addr = (mem_rd == 1'b1) ? rd_addr : wr_addr;

endmodule
```

top1.v - stored in mem

```
module top_filter_2d;

reg     clk, n_reset;
reg     start;
wire    finish;

initial clk = 1'b0;
always #5 clk = ~clk;

initial begin
    n_reset = 1'b1;
    $readmemh("../c/img_in.dat", i_buf.data);

    #3;
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    start = 1'b1;
    @(posedge clk);
    start = 1'b0;
end

wire    cs, we;
wire    [16:0]  addr;
wire    [7:0]   din;
wire    [7:0]   dout;

filter2d    i_filter (.clk(clk), .n_reset(n_reset), .start(start), .finish(finish),
                .cs(cs), .we(we), .addr(addr), .din(din), .dout(dout),
                .h_write(1'b0), .h_idx(4'b0), .h_data(8'b0));

mem_single #(
                .WD(8),
                .DEPTH(256*256*2)
) i_buf (
                .clk(clk),
                .cs(cs),
                .we(we),
                .addr(addr),
                .din(din),
                .dout(dout)
);

always@(posedge clk) begin
    if(finish == 1'b1) begin
        for(int i=0;i<256;i++) begin
            for(int j=0;j<256;j++) begin
                $write("%3d ", i_buf.data[i*256+j+256*256]);
            end
            $write("\n");
        end
        $finish;
    end
end

endmodule
```

filter2d.v - including buffer

```
module filter2d (
                input   clk,
                input   n_reset,

                input           i_strb,
                input   [7:0]   i_data,

                output          o_strb,
                output  [7:0]   o_data
);

wire            start;
wire            mem_rd;
wire    [15:0]  rd_addr;
wire    [7:0]   rd_data;

filter2d_buf i_buf(
                .clk(clk),
                .n_reset(n_reset),
                .i_strb(i_strb),
                .i_data(i_data),

                .start(start),

                .mem_rd(mem_rd),
                .rd_addr(rd_addr),
                .rd_data(rd_data)
);

filter2d_op i_op(
                .clk(clk),
                .n_reset(n_reset),
                .start(start),

                .mem_rd(mem_rd),
                .rd_addr(rd_addr),
                .rd_data(rd_data),

                .o_strb(o_strb),
                .o_data(o_data)
);

endmodule
```

filter2d_buf.v

```
module filter2d_buf (
                input           clk,
                input           n_reset,
                input           i_strb,
                input   [7:0]   i_data,

                output reg      start,

                input           mem_rd,
                input   [15:0]  rd_addr,
                output  [7:0]   rd_data
);

reg [7:0]   cnt_x;
reg [7:0]   cnt_y;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        cnt_x <= 255;
        cnt_y <= 255;
    end else begin
        if(i_strb == 1'b1) begin
            cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
            if(cnt_x == 255) begin
                cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1;
            end
        end
    end
end

reg         mode;
wire        mode_change;
reg         mem_wr;
reg [7:0]   wr_data;

assign mode_change = (mem_wr == 1'b1) && (cnt_x == 255) && (cnt_y == 255);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        mode <= 1'b0;
        start <= 1'b0;
    end else begin
        if(mode_change == 1'b1) begin
            mode <= ~mode;
        end
        start <= mode_change;
    end
end

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        mem_wr <= 1'b0;
        wr_data <= 8'b0;
    end else begin
        mem_wr <= i_strb;
        wr_data <= i_data;
    end
end
wire    [15:0]  wr_addr = cnt_y*256 + cnt_x;

wire            cs0 = (mode == 1'b0) ? mem_wr : mem_rd;
wire            we0 = (mode == 1'b0) ? mem_wr : 1'b0;
wire    [15:0]  addr0 = (mode == 1'b0) ? wr_addr : rd_addr;
wire    [7:0]   din0 = (mode == 1'b0) ? wr_data : 'b0;
wire    [7:0]   dout0;

wire            cs1 = (mode == 1'b1) ? mem_wr : mem_rd;
wire            we1 = (mode == 1'b1) ? mem_wr : 1'b0;
wire    [15:0]  addr1 = (mode == 1'b1) ? wr_addr : rd_addr;
wire    [7:0]   din1 = (mode == 1'b1) ? wr_data : 'b0;
wire    [7:0]   dout1;

assign  rd_data = (mode == 1'b0) ? dout1 : dout0;


mem_single #(
                .WD(8),
                .DEPTH(256*256)
) i_buf0 (
                .clk(clk),
                .cs(cs0),
                .we(we0),
                .addr(addr0),
                .din(din0),
                .dout(dout0)
);

mem_single #(
                .WD(8),
                .DEPTH(256*256)
) i_buf1 (
                .clk(clk),
                .cs(cs1),
                .we(we1),
                .addr(addr1),
                .din(din1),
                .dout(dout1)
);

endmodule
```

filter2d_op.v

```
module filter2d_op (
                input           clk,
                input           n_reset,
                input           start,

                output              mem_rd,
                output reg  [15:0]  rd_addr,
                input       [7:0]   rd_data,

                output reg          o_strb,
                output reg  [7:0]   o_data,

                input           h_write,
                input   [3:0]   h_idx,
                input   [7:0]   h_data
);

reg         on_proc;
reg [3:0]   cnt;
reg [7:0]   cnt_x;
reg [7:0]   cnt_y;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        on_proc <= 1'b0;
        cnt <= 0;
        cnt_x <= 0;
        cnt_y <= 0;
    end else begin
        if(start == 1'b1) on_proc <= 1'b1;
        else if((cnt == 11) && (cnt_x == 255) && (cnt_y == 255)) on_proc <= 1'b0;

        if(on_proc == 1'b1) begin
            cnt <= (cnt == 11) ? 0 : cnt+1;
            if(cnt == 11) begin
                cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
                if(cnt_x == 255) begin
                    cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1;
                end
            end
        end
    end
end

assign mem_rd = (cnt >= 0) && (cnt <= 8) && (on_proc == 1'b1);
always@(*) begin
    case(cnt)
        4'd0:   rd_addr = (cnt_y-1)*256 + cnt_x-1;
        4'd1:   rd_addr = (cnt_y-1)*256 + cnt_x;
        4'd2:   rd_addr = (cnt_y-1)*256 + cnt_x+1;
        4'd3:   rd_addr = (cnt_y  )*256 + cnt_x-1;
        4'd4:   rd_addr = (cnt_y  )*256 + cnt_x;
        4'd5:   rd_addr = (cnt_y  )*256 + cnt_x+1;
        4'd6:   rd_addr = (cnt_y+1)*256 + cnt_x-1;
        4'd7:   rd_addr = (cnt_y+1)*256 + cnt_x;
        4'd8:   rd_addr = (cnt_y+1)*256 + cnt_x+1;
        default:    rd_addr = 'bx;
    endcase
end

reg     [7:0]   pd;
wire            pd_en = (cnt >= 1) && (cnt <= 9);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        pd <= 0;
    end else begin
        if(pd_en == 1'b1) pd <= rd_data;
    end
end

reg signed  [7:0]   h[0:8];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        h[0] <= 8'h08;
        h[1] <= 8'h10;
        h[2] <= 8'h08;
        h[3] <= 8'h10;
        h[4] <= 8'h20;
        h[5] <= 8'h10;
        h[6] <= 8'h08;
        h[7] <= 8'h10;
        h[8] <= 8'h08;
    end else begin
        if(h_write == 1'b1) begin
            h[h_idx] <= h_data;
        end
    end
end

wire signed [7:0]   coeff = h[cnt-2];
wire signed [15:0]  mul = pd * coeff;
reg  signed [19:0]  acc;
wire signed [19:0]  acc_in = (cnt == 1) ? 0 : mul + acc;
reg                 acc_en;

always@(*) begin
    acc_en = 1'b0;
    case(cnt)
        4'd 1: acc_en = 1'b1;
        4'd 2: if((cnt_y > 0) && (cnt_x >   0)) acc_en = 1'b1;
        4'd 3: if((cnt_y > 0)                 ) acc_en = 1'b1;
        4'd 4: if((cnt_y > 0) && (cnt_x < 255)) acc_en = 1'b1;
        4'd 5: if(cnt_x >   0)  acc_en = 1'b1;
        4'd 6:                  acc_en = 1'b1;
        4'd 7: if(cnt_x < 255)  acc_en = 1'b1;
        4'd 8: if((cnt_y < 255) && (cnt_x >   0)) acc_en = 1'b1;
        4'd 9: if((cnt_y < 255)                 ) acc_en = 1'b1;
        4'd10: if((cnt_y < 255) && (cnt_x < 255)) acc_en = 1'b1;
        default: acc_en = 1'b0;
    endcase
end

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        acc <= 'b0;
    end else begin
        if(acc_en == 1'b1) acc <= acc_in;
    end
end

wire    [19:0]  pd_rnd_1 = acc + (1<<6);
```

```
wire    [12:0]  pd_rnd = pd_rnd_1[19:7];
wire    [7:0]   pd_out = (pd_rnd < 0) ? 0 :
                         (pd_rnd > 255) ? 255 :
                         pd_rnd[7:0];

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        o_strb <= 1'b0;
        o_data <= 'b0;
    end else begin
        o_strb <= (cnt == 11);
        if(cnt == 11) begin
            o_data <= pd_out;
        end
    end
end

endmodule
```

top2.v - strobed input

```
module top_filter_2d;

reg     clk, n_reset;
reg     start;

initial clk = 1'b0;
always #5 clk = ~clk;

reg [7:0]   img_data[0:65535];
reg         i_strb;
reg [7:0]   i_data;
integer idx, cnt;
initial begin
    cnt = 0;
    n_reset = 1'b1;
    $readmemh("../c/img_in.dat", img_data);
    i_strb = 1'b0;
    i_data = 'bx;
    #3;
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    repeat(3) begin
        for(idx=0;idx<65536;idx=idx+1) begin
            i_strb = 1'b1;
            i_data = img_data[idx];
            @(posedge clk);
            repeat(16) begin
                i_strb = 1'b0;
                i_data = 'bx;
                @(posedge clk);
            end
        end
    end
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    $finish;
end

wire            o_strb;
wire    [7:0]   o_data;
filter2d    i_filter (
                    .clk(clk),
                    .n_reset(n_reset),
                    .i_strb(i_strb),
                    .i_data(i_data),
                    .o_strb(o_strb),
                    .o_data(o_data),
                    .h_write(1'b0),
                    .h_idx(4'b0),
                    .h_data(8'b0)
);

always@(posedge clk) begin
    if(o_strb == 1'b1) begin
        $write("%3d ", o_data);
```

```
        cnt = cnt + 1;
        if(cnt[7:0] == 0) begin
            $write("\n");
        end
    end
end

endmodule
```

filter2d.v - line buffer

```
module filter2d (
                input           clk,
                input           n_reset,

                input           i_strb,
                input   [7:0]   i_data,

                output reg          o_strb,
                output reg  [7:0]   o_data,

                input           h_write,
                input   [3:0]   h_idx,
                input   [7:0]   h_data
);

reg         garbage;
reg [3:0]   cnt;
reg [7:0]   cnt_x;
reg [7:0]   cnt_y;
reg [7:0]   i_data_d;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        garbage <= 1'b1;
        cnt <= 7;
        cnt_x <= 254;
        cnt_y <= 254;
        i_data_d <= 'b0;
    end else begin
        if(i_strb == 1'b1) begin
            cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
            if(cnt_x == 255) begin
                cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1;
                if(cnt_y == 255) garbage <= 1'b0;
            end
        end
        if(i_strb == 1'b1) cnt <= 0;
        else if(cnt < 7) cnt <= cnt+1;
        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end

reg     [7:0]   ibuf[2:0][2:0];
wire    [7:0]   dout;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        for(int i=0;i<3;i++) begin
            for(int j=0;j<3;j++) begin
                ibuf[i][j] <= 'b0;
            end
        end
    end else begin
        if(cnt == 0) begin
            for(int i=0;i<3;i++) begin
                for(int j=0;j<2;j++) begin
                    ibuf[i][j] <= ibuf[i][j+1];
                end
            end
            ibuf[2][2] <= i_data_d;
        end
```

```
        if(cnt == 1) ibuf[0][2] <= dout;
        if(cnt == 2) ibuf[1][2] <= dout;
    end
end

wire        mem_rd = (cnt == 0) || (cnt == 1);
wire        mem_wr = (cnt == 2);

reg     [8:0]   wr_addr;
wire    [8:0]   rd_addr0 = wr_addr;
wire    [8:0]   rd_addr1 = (wr_addr<256) ? wr_addr+256 : wr_addr-256;
wire    [8:0]   rd_addr = (cnt == 0) ? rd_addr0 : rd_addr1;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        wr_addr <= 0;
    end else begin
        if(mem_wr == 1'b1) begin
            wr_addr <= (wr_addr == 2*256-1) ? 0 : wr_addr + 1;
        end
    end
end

wire            cs = mem_rd | mem_wr;
wire            we = mem_wr;
wire    [8:0]   addr = (mem_wr == 1'b1) ? wr_addr : rd_addr;
wire    [7:0]   din = i_data_d;

mem_single #(
                .WD(8),
                .DEPTH(2*256)
) i_buf0 (
                .clk(clk),
                .cs(cs),
                .we(we),
                .addr(addr),
                .din(din),
                .dout(dout)
);

reg signed  [7:0]   h[0:8];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        h[0] <= 8'h08;
        h[1] <= 8'h10;
        h[2] <= 8'h08;
        h[3] <= 8'h10;
        h[4] <= 8'h20;
        h[5] <= 8'h10;
        h[6] <= 8'h08;
        h[7] <= 8'h10;
        h[8] <= 8'h08;
    end else begin
        if(h_write == 1'b1) begin
            h[h_idx] <= h_data;
        end
    end
end

reg signed  [15:0]  mul[2:0][2:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        for(int i=0;i<3;i++) begin
            for(int j=0;j<3;j++) begin
                mul[i][j] <= 'b0;
            end
        end
    end else begin
        if((cnt == 3) && (garbage == 1'b0)) begin
            mul[0][0] <= ((cnt_y > 0) && (cnt_x >   0)) ? ibuf[0][0] * h[0] : 'b0;
            mul[0][1] <= ((cnt_y > 0)                 ) ? ibuf[0][1] * h[1] : 'b0;
            mul[0][2] <= ((cnt_y > 0) && (cnt_x < 255)) ? ibuf[0][2] * h[2] : 'b0;
            mul[1][0] <= (cnt_x >   0) ? ibuf[1][0] * h[3] : 'b0;
            mul[1][1] <=                 ibuf[1][1] * h[4];
            mul[1][2] <= (cnt_x < 255) ? ibuf[1][2] * h[5] : 'b0;
            mul[2][0] <= ((cnt_y < 255) && (cnt_x >   0)) ? ibuf[2][0] * h[6] : 'b0;
            mul[2][1] <= ((cnt_y < 255)                 ) ? ibuf[2][1] * h[7] : 'b0;
            mul[2][2] <= ((cnt_y < 255) && (cnt_x < 255)) ? ibuf[2][2] * h[8] : 'b0;
        end
    end
end

reg  signed [19:0]  sum_in;
reg  signed [19:0]  sum;
always@(*) begin
    sum_in = 0;
    for(int i=0;i<3;i++) begin
        for(int j=0;j<3;j++) begin
            sum_in = sum_in + mul[i][j];
        end
    end
end
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        sum <= 'b0;
    end else begin
        if((cnt == 4) && (garbage == 1'b0)) begin
            sum <= sum_in;
        end
    end
end

wire    [19:0]  pd_rnd_1 = sum + (1<<6);
wire    [12:0]  pd_rnd = pd_rnd_1[19:7];
wire    [7:0]   pd_out = (pd_rnd < 0) ? 0 :
                         (pd_rnd > 255) ? 255 :
                         pd_rnd[7:0];

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        o_strb <= 1'b0;
        o_data <= 'b0;
    end else begin
        o_strb <= ((cnt == 5) && (garbage == 1'b0));
        if((cnt == 5) && (garbage == 1'b0)) begin
            o_data <= pd_out;
        end
    end
end

endmodule
```

filter_2d_dpi.c

```
#include <stdio.h>
#include <stdlib.h>

unsigned char   *in_img;
unsigned char   *out_img;
int             height, width;

void filter2d(void) {
    int     h[3][3] = {0x08, 0x10, 0x08, 0x10, 0x20, 0x10, 0x08, 0x10, 0x08};
    for(int i=0;i<height;i++) {
        for(int j=0;j<width;j++) {
            int sum = 0;
            if(i>0 && j>0)              sum += in_img[(i-1)*width+j-1]*h[0][0];
            if(i>0)                     sum += in_img[(i-1)*width+j  ]*h[0][1];
            if(i>0 && j<width-1)        sum += in_img[(i-1)*width+j+1]*h[0][2];
            if(j>0)                     sum += in_img[(i  )*width+j-1]*h[1][0];
                                        sum += in_img[(i  )*width+j  ]*h[1][1];
            if(j<width-1)               sum += in_img[(i  )*width+j+1]*h[1][2];
            if(i<height-1 && j>0)       sum += in_img[(i+1)*width+j-1]*h[2][0];
            if(i<height-1)              sum += in_img[(i+1)*width+j  ]*h[2][1];
            if(i<height-1 && j<width-1) sum += in_img[(i+1)*width+j+1]*h[2][2];
            sum = (sum + (1<<6)) >> 7;
            if(sum < 0) out_img[i*width+j] = 0;
            else if(sum > 255) out_img[i*width+j] = 255;
            else out_img[i*width+j] = sum;
        }
    }
}

void init_filter2d(int h, int w) {
    int         i, a;
    FILE        *inf;
    inf = fopen("../c/img_in.txt", "r");

    height = h;
    width = w;
    in_img = malloc(height*width*sizeof(unsigned char));
    out_img = malloc(height*width*sizeof(unsigned char));
    for(i=0;i<height*width;i++) {
        fscanf(inf, "%d,", &a);
        in_img[i] = a;
    }

    filter2d();

    fclose(inf);
}

unsigned char get_input(void) {
    static  int i;
    unsigned char res = in_img[i];
    i++;
    if(i==height*width) i = 0;
    return res;
}

unsigned char get_output(void) {
    static  int i;
    unsigned char res = out_img[i];
    i++;
    if(i==height*width) i = 0;
    return res;
}
```

top3.v - DPI

```
module top_filter_2d;

reg     clk, n_reset;
reg     start;

initial clk = 1'b0;
always #5 clk = ~clk;

import "DPI" function void init_filter2d(input int h, input int w);
import "DPI" function byte get_input();
import "DPI" function byte get_output();

reg         i_strb;
reg [7:0]   i_data;
initial begin
    n_reset = 1'b1;
    init_filter2d(256, 256);
    i_strb = 1'b0;
    i_data = 'bx;
    #3;
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    repeat(3) begin
        repeat(256*256) begin
            i_strb = 1'b1;
            i_data = get_input();
            @(posedge clk);
            repeat(16) begin
                i_strb = 1'b0;
                i_data = 'bx;
                @(posedge clk);
            end
        end
    end
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    $finish;
end

wire            o_strb;
wire    [7:0]   o_data;
filter2d    i_filter (
                    .clk(clk),
                    .n_reset(n_reset),
                    .i_strb(i_strb),
                    .i_data(i_data),
                    .o_strb(o_strb),
                    .o_data(o_data),
                    .h_write(1'b0),
                    .h_idx(4'b0),
                    .h_data(8'b0)
);

reg     [7:0]   out_ref;
always@(posedge clk) begin
    if(o_strb == 1'b1) begin
        out_ref = get_output();
        if(o_data != out_ref) begin
            $display("Error!! o_data = %3d, out_ref = %3d", o_data, out_ref);
            #10;
            $finish;
        end
    end
end

endmodule
```

Lecture 04. FFT

fft_float.c

```
#include <stdio.h>
#include <math.h>

typedef struct {
    float   r, i;
} t_complex;

void complex_add(t_complex *dst, t_complex src0, t_complex src1) {
    dst->r = src0.r + src1.r;
    dst->i = src0.i + src1.i;
}

void complex_sub(t_complex *dst, t_complex src0, t_complex src1) {
    dst->r = src0.r - src1.r;
    dst->i = src0.i - src1.i;
}

void complex_mul(t_complex *dst, t_complex src0, t_complex src1) {
    dst->r = src0.r * src1.r - src0.i * src1.i;
    dst->i = src0.r * src1.i + src0.i * src1.r;
}

float complex_mag(t_complex in) {
    return sqrt(in.r*in.r + in.i*in.i);
}

void twiddle_mul(t_complex *dst, int k, int N) {
    t_complex   twiddle_factor;
    const float pi = acos(-1.0);

    twiddle_factor.r = cos(-2*pi*k/N);
    twiddle_factor.i = sin(-2*pi*k/N);
    complex_mul(dst, *dst, twiddle_factor);
}

void fft(float in[1024], t_complex out[1024]) {
    int i, j, k, p;

    for(i=0;i<1024;i++) {
        out[i].r = in[i];
        out[i].i = 0;
    }
    for(i=0,p=1024;i<10;i++,p/=2) {
        for(j=0;j<1024/p;j++) {
            for(k=0;k<p/2;k++) {
                t_complex   bf0, bf1;
                complex_add(&bf0, out[j*p+k], out[j*p+k+p/2]);
                complex_sub(&bf1, out[j*p+k], out[j*p+k+p/2]);
                twiddle_mul(&bf1, k, p);
                out[j*p+k] = bf0;
                out[j*p+k+p/2] = bf1;
            }
        }
    }
}

int bit_reverse(int in) {
    int i, out = 0;
    for(i=0;i<10;i++) {
        out <<= 1;
        out |= in & 0x01;
        in >>= 1;
    }
    return out;
}

int main(void) {
    float       fft_in[1024];
    t_complex   fft_out[1024];
    int     i;
    const float pi = acos(-1.0);

    for(i=0;i<1024;i++) {
//      fft_in[i] = sin(2*pi*i*100/1024);
        fft_in[i] = (i<5) ? 1 : (i>=1019) ? 1 : 0;
    }
    fft(fft_in, fft_out);
/*
    for(i=0;i<1024;i++) {
        printf("%f %f\n", fft_out[bit_reverse(i)].r, fft_out[bit_reverse(i)].i);
    }
*/
    for(i=0;i<1024;i++) {
        printf("%f\n", complex_mag(fft_out[bit_reverse(i)]));
    }
}
```

fft_fixed.c

```
#include <stdio.h>
#include <math.h>

typedef struct {
    int     r, i;
} t_complex;

void complex_add(t_complex *dst, t_complex src0, t_complex src1) {
    dst->r = (src0.r + src1.r) >> 1;
    dst->i = (src0.i + src1.i) >> 1;
}

void complex_sub(t_complex *dst, t_complex src0, t_complex src1) {
    dst->r = (src0.r - src1.r) >> 1;
    dst->i = (src0.i - src1.i) >> 1;
}

void complex_mul(t_complex *dst, t_complex src0, t_complex src1) {
    dst->r = (src0.r * src1.r - src0.i * src1.i) >> 9;
    dst->i = (src0.r * src1.i + src0.i * src1.r) >> 9;
}

float complex_mag(t_complex in) {
    return sqrt(in.r*in.r/32./32. + in.i*in.i/32./32.);
}

void twiddle_mul(t_complex *dst, int k, int N) {
    t_complex   twiddle_factor;
    const float pi = acos(-1.0);

    twiddle_factor.r = floor(cos(-2*pi*k/N)*511 + 0.5);
    twiddle_factor.i = floor(sin(-2*pi*k/N)*511 + 0.5);
    complex_mul(dst, *dst, twiddle_factor);
}

void fft(int in[1024], t_complex out[1024]) {
    int i, j, k, p;

    for(i=0;i<1024;i++) {
        out[i].r = in[i];
        out[i].i = 0;
    }
    for(i=0,p=1024;i<10;i++,p/=2) {
        for(j=0;j<1024/p;j++) {
            for(k=0;k<p/2;k++) {
                t_complex   bf0, bf1;
                complex_add(&bf0, out[j*p+k], out[j*p+k+p/2]);
                complex_sub(&bf1, out[j*p+k], out[j*p+k+p/2]);
                twiddle_mul(&bf1, k, p);
                out[j*p+k] = bf0;
                out[j*p+k+p/2] = bf1;
            }
        }
    }
}

int bit_reverse(int in) {
    int i, out = 0;
    for(i=0;i<10;i++) {
        out <<= 1;
        out |= in & 0x01;
        in >>= 1;
    }
    return out;
}

int main(void) {
    int         fft_in[1024];
    t_complex   fft_out[1024];
    int     i;
    const float pi = acos(-1.0);

    for(i=0;i<1024;i++) {
        //fft_in[i] = floor(sin(2*pi*i*100/1024) * 32767 + 0.5);
        fft_in[i] = (i<5) ? 32767 : (i>=1019) ? 32767 : 0;
    }
    fft(fft_in, fft_out);
/*
    for(i=0;i<1024;i++) {
        printf("%d %d\n", fft_out[bit_reverse(i)].r, fft_out[bit_reverse(i)].i);
    }
*/
    for(i=0;i<1024;i++) {
        printf("%f\n", complex_mag(fft_out[bit_reverse(i)]));
    }

}
```

fft_frame.v

```
module fft (
              input     clk
            , input     n_reset
            , input     start
            , output    ready

            , output            cs
            , output            we
            , output    [9:0]   addr
            , output    [31:0]  w_data
            , input     [31:0]  r_data
);

reg             on_proc;
reg     [9:0]   cnt_p;  // counting stage, one hot, 512 to 1
                        // in C, 1024 to 2
wire    [9:0]   cnt_pi; // reverse of cnt_p
reg     [9:0]   cnt_j;  // counting block
reg     [9:0]   cnt_k;  // counting butterfly
reg     [9:0]   cnt_t;  // k*pi
reg     [2:0]   cnt_o;  // counting butterfly operation

wire            last_o = (cnt_o == 6);
wire            last_k = (cnt_k == cnt_p-1);
wire            last_j = (cnt_j == cnt_pi-1);
wire            last_p = (cnt_p == 10'h001);

assign ready = ~on_proc;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        on_proc <= 1'b0;
        cnt_p <= 10'h200;
        cnt_j <= 'b0;
        cnt_k <= 'b0;
        cnt_t <= 'b0;
        cnt_o <= 'b0;
    end else begin
        if((on_proc == 1'b0) && (start == 1'b1)) begin
            on_proc <= 1'b1;
            cnt_p <= 10'h200;
            cnt_j <= 'b0;
            cnt_k <= 'b0;
            cnt_t <= 'b0;
            cnt_o <= 'b0;
        end
        if(on_proc == 1'b1) begin
            cnt_o <= (last_o == 1'b1) ? 'b0 : cnt_o+1;
            if(last_o == 1'b1) begin
                cnt_k <= (last_k == 1'b1) ? 'b0 : cnt_k+1;
                cnt_t <= (last_k == 1'b1) ? 'b0 : cnt_t+cnt_pi;
                if(last_k == 1'b1) begin
                    cnt_j <= (last_j == 1'b1) ? 'b0 : cnt_j+1;
                    if(last_j == 1'b1) begin
                        cnt_p <= cnt_p >> 1;
                        if(last_p == 1'b1) begin
                            on_proc <= 1'b0;
                        end
                    end
                end
            end
        end
    end
end
genvar i;
for(i=0;i<10;i++) begin
    assign cnt_pi[i] = cnt_p[9-i];
end

assign  cs = (cnt_o == 0) || (cnt_o == 1) || (cnt_o == 5) || (cnt_o == 6);
assign  we = (cnt_o == 5) || (cnt_o == 6);
reg     [9:0]   addr0;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        addr0 <= 'b0;
    end else begin
        if(on_proc == 1'b1) begin
            if(last_o == 1'b1) begin
                if((last_j == 1'b1) && (last_k == 1'b1)) addr0 <= 'b0;
                else if(last_k == 1'b1) addr0 <= addr0 + cnt_p+1;
                else addr0 <= addr0 + 1;
            end
        end
    end
end
wire    [9:0]   addr1 = addr0 + cnt_p;
assign addr = (cnt_o == 0) || (cnt_o == 5) ? addr0 : addr1;

reg signed  [16:0]  in0_r, in0_i;
reg signed  [16:0]  in1_r, in1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        in0_r <= 'b0;
        in0_i <= 'b0;
        in1_r <= 'b0;
        in1_i <= 'b0;
    end else begin
        if(cnt_o == 1) begin
            in0_r <= {r_data[15], r_data[15:0]};
            in0_i <= {r_data[31], r_data[31:16]};
        end
        if(cnt_o == 2) begin
            in1_r <= {r_data[15], r_data[15:0]};
            in1_i <= {r_data[31], r_data[31:16]};
        end
    end
end

reg signed  [15:0]  bf0_r, bf0_i;
reg signed  [15:0]  bf1_r, bf1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bf0_r <= 'b0;
        bf0_i <= 'b0;
        bf1_r <= 'b0;
        bf1_i <= 'b0;
    end else begin
        if(cnt_o == 3) begin
            bf0_r <= (in0_r + in1_r) >> 1;
            bf0_i <= (in0_i + in1_i) >> 1;
            bf1_r <= (in0_r - in1_r) >> 1;
            bf1_i <= (in0_i - in1_i) >> 1;
```

```
        end
    end
end

reg             [19:0]  twid_lut;
wire    signed  [9:0]   cos = (cnt_t < 256) ? twid_lut[9:0] : twid_lut[19:10];
wire    signed  [9:0]   sin = (cnt_t < 256) ? twid_lut[19:10] : -twid_lut[9:0];
reg     signed  [9:0]   twid_r, twid_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        twid_r <= 'b0;
        twid_i <= 'b0;
    end else begin
        if(cnt_o == 3) begin
            twid_r <= cos;
            twid_i <= sin;
        end
    end
end

always@(*) begin
    case(cnt_t[7:0])
        0: twid_lut = {-10'd0,10'd511};
        1: twid_lut = {-10'd3,10'd511};
        2: twid_lut = {-10'd6,10'd511};
        3: twid_lut = {-10'd9,10'd511};
        ...
        252: twid_lut = {-10'd511,10'd13};
        253: twid_lut = {-10'd511,10'd9};
        254: twid_lut = {-10'd511,10'd6};
        255: twid_lut = {-10'd511,10'd3};
    endcase
end

wire signed [24:0]  mul_rr = bf1_r * twid_r;
wire signed [24:0]  mul_ri = bf1_r * twid_i;
wire signed [24:0]  mul_ir = bf1_i * twid_r;
wire signed [24:0]  mul_ii = bf1_i * twid_i;
reg signed [15:0]   bfmul_r;
reg signed [15:0]   bfmul_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bfmul_r <= 'b0;
        bfmul_i <= 'b0;
    end else begin
        if(cnt_o == 4) begin
            bfmul_r <= (mul_rr - mul_ii) >> 9;
            bfmul_i <= (mul_ri + mul_ir) >> 9;
        end
    end
end

assign w_data = (cnt_o == 5) ? {bf0_i, bf0_r} : {bfmul_i, bfmul_r};

endmodule
```

Twiddle Factor Generation

```
#include <stdio.h>
#include <math.h>

#define N   1024

int main(void) {
    const float pi = acos(-1.0);
    int k, c, s;
    for(k=0;k<N/4;k++) {
        c = floor(cos(-2*pi*k/N)*511 + 0.5);
        s = floor(sin(-2*pi*k/N)*511 + 0.5);
        printf("\t\t%d: twid_lut = {-10'd%d,10'd%d};\n", k, -s, c);
    }
}
```

top1.v

```
module top_fft;

reg     clk, n_reset;
reg     start;
wire    ready;

initial begin
    $vcdplusfile("top_fft.vpd");
    $vcdpluson(0, top_fft);
end
initial clk = 1'b0;
always #5 clk = ~clk;

import "DPI" function void init_fft();
import "DPI" function int unsigned get_input();
import "DPI" function int unsigned get_output();

int     i;
reg [31:0]  mem_data[0:1023];
reg [31:0]  c_data;
initial begin
    n_reset = 1'b1;
    start = 1'b0;
    init_fft();
    for(i=0;i<1024;i++) begin
        mem_data[i] = get_input();
    end
    #3;
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
    @(posedge clk);
    @(posedge clk);
    start = 1'b1;
    @(posedge clk);
    start = 1'b0;
    @(posedge ready);
    @(posedge clk);
    for(i=0;i<1024;i++) begin
        c_data = get_output();
        if(mem_data[i] !== c_data) begin
            $display("Error: mem_data[%d] = %8X, c_data = %8X"
                    , i, mem_data[i], c_data);
        end
    end
    @(posedge clk);
    @(posedge clk);
    $finish;
end

wire            cs, we;
wire    [9:0]   addr;
wire    [31:0]  w_data;
reg     [31:0]  r_data;

fft i_fft (
              .clk(clk)
            , .n_reset(n_reset)
            , .start(start)
            , .ready(ready)
            , .cs(cs)
            , .we(we)
            , .addr(addr)
            , .w_data(w_data)
            , .r_data(r_data)
);

always@(posedge clk) begin
    if(cs == 1'b1) begin
        if(we == 1'b1) mem_data[addr] <= w_data;
        else r_data <= mem_data[addr];
    end
end

endmodule
```

fft_pipeline.v

```
module fft (
              input     clk
            , input     n_reset

            , input             i_strb
            , input     [31:0]  i_data
            , output            o_strb
            , output    [31:0]  o_data

);

wire            o_strb0;
wire    [31:0]  o_data0;
wire            o_strb1;
wire    [31:0]  o_data1;
wire            o_strb2;
wire    [31:0]  o_data2;
wire            o_strb3;
wire    [31:0]  o_data3;
wire            o_strb4;
wire    [31:0]  o_data4;
wire            o_strb5;
wire    [31:0]  o_data5;
wire            o_strb6;
wire    [31:0]  o_data6;
wire            o_strb7;
wire    [31:0]  o_data7;
wire            o_strb8;
wire    [31:0]  o_data8;
wire            o_strb9;
wire    [31:0]  o_data9;

fft_pipe_stg0 i_stg0 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(i_strb)
            , .i_data(i_data)
            , .o_strb(o_strb0)
            , .o_data(o_data0)
);

fft_pipe_stg1 i_stg1 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(o_strb0)
            , .i_data(o_data0)
            , .o_strb(o_strb1)
            , .o_data(o_data1)
);

fft_pipe_stg2 i_stg2 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(o_strb1)
            , .i_data(o_data1)
            , .o_strb(o_strb2)
            , .o_data(o_data2)
);

fft_pipe_stg3 i_stg3 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(o_strb2)
            , .i_data(o_data2)
            , .o_strb(o_strb3)
            , .o_data(o_data3)
);

fft_pipe_stg4 i_stg4 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(o_strb3)
            , .i_data(o_data3)
            , .o_strb(o_strb4)
            , .o_data(o_data4)
);

fft_pipe_stg5 i_stg5 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(o_strb4)
            , .i_data(o_data4)
            , .o_strb(o_strb5)
            , .o_data(o_data5)
);

fft_pipe_stg6 i_stg6 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(o_strb5)
            , .i_data(o_data5)
            , .o_strb(o_strb6)
            , .o_data(o_data6)
);

fft_pipe_stg7 i_stg7 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(o_strb6)
            , .i_data(o_data6)
            , .o_strb(o_strb7)
            , .o_data(o_data7)
);

fft_pipe_stg8 i_stg8 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(o_strb7)
            , .i_data(o_data7)
            , .o_strb(o_strb8)
            , .o_data(o_data8)
```

```
);

fft_pipe_stg9 i_stg9 (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(o_strb8)
            , .i_data(o_data8)
            , .o_strb(o_strb9)
            , .o_data(o_data9)
);

assign o_strb = o_strb9;
assign o_data = o_data9;

endmodule
```

fft_pipe_stg0.v

```
module fft_pipe_stg0 (
              input     clk
            , input     n_reset

            , input             i_strb
            , input     [31:0]  i_data
            , output            o_strb
            , output    [31:0]  o_data
);


reg     [31:0]  i_data_d;
reg     [9:0]   cnt_k;  // counting input data
reg     [2:0]   cnt_o;  // counting butterfly operation
reg             first_garbage;

wire            last_o = (cnt_o == 4);
wire            last_k = (cnt_k == 1023);

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        cnt_k <= 'b0;
        cnt_o <= 7;
        i_data_d <= 'b0;
        first_garbage <= 1'b1;
    end else begin
        if(i_strb == 1'b1) cnt_o <= 0;
        if(cnt_o < 5) cnt_o <= cnt_o+1;
        if(last_o == 1'b1) begin
            cnt_k <= (last_k == 1'b1) ? 'b0 : cnt_k+1;
            if(cnt_k == 511) first_garbage <= 1'b0;
        end
        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end

wire            cs;
wire            we;
reg     [8:0]   addr;
wire    [31:0]  r_data;
wire    [31:0]  w_data;

assign  cs = (cnt_o == 0) || (cnt_o == 4);
assign  we = (cnt_o == 4);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        addr <= 'b0;
    end else begin
        if(last_o == 1'b1) addr <= addr + 1;
    end
end

reg     signed  [16:0]  in0_r, in0_i;
wire    signed  [16:0]  in1_r, in1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        in0_r <= 'b0;
        in0_i <= 'b0;
    end else begin
        if(cnt_o == 1) begin
```

```
            in0_r <= {r_data[15], r_data[15:0]};
            in0_i <= {r_data[31], r_data[31:16]};
        end
    end
end
assign in1_r = {i_data_d[15], i_data_d[15:0]};
assign in1_i = {i_data_d[31], i_data_d[31:16]};

reg signed  [15:0]  bf0_r, bf0_i;
reg signed  [15:0]  bf1_r, bf1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bf0_r <= 'b0;
        bf0_i <= 'b0;
        bf1_r <= 'b0;
        bf1_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            bf0_r <= (in0_r + in1_r) >> 1;
            bf0_i <= (in0_i + in1_i) >> 1;
            bf1_r <= (in0_r - in1_r) >> 1;
            bf1_i <= (in0_i - in1_i) >> 1;
        end
    end
end

reg             [19:0]  twid_lut;
wire    signed  [9:0]   cos = (cnt_k[8:0] < 256) ? twid_lut[9:0]
                                                 : twid_lut[19:10];
wire    signed  [9:0]   sin = (cnt_k[8:0] < 256) ? twid_lut[19:10]
                                                 : -twid_lut[9:0];
reg     signed  [9:0]   twid_r, twid_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        twid_r <= 'b0;
        twid_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            twid_r <= cos;
            twid_i <= sin;
        end
    end
end

always@(*) begin
    case(cnt_k[7:0])
        0: twid_lut = {-10'd0,10'd511};
        1: twid_lut = {-10'd3,10'd511};
        2: twid_lut = {-10'd6,10'd511};
        3: twid_lut = {-10'd9,10'd511};
        ...
        252: twid_lut = {-10'd511,10'd13};
        253: twid_lut = {-10'd511,10'd9};
        254: twid_lut = {-10'd511,10'd6};
        255: twid_lut = {-10'd511,10'd3};
    endcase
end

wire signed [24:0]  mul_rr = bf1_r * twid_r;
wire signed [24:0]  mul_ri = bf1_r * twid_i;
wire signed [24:0]  mul_ir = bf1_i * twid_r;
wire signed [24:0]  mul_ii = bf1_i * twid_i;
reg  signed [15:0]  bfmul_r;
reg  signed [15:0]  bfmul_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bfmul_r <= 'b0;
        bfmul_i <= 'b0;
    end else begin
        if(cnt_o == 3) begin
            bfmul_r <= (mul_rr - mul_ii) >> 9;
            bfmul_i <= (mul_ri + mul_ir) >> 9;
        end
    end
end

assign w_data = (cnt_k < 512) ? i_data_d : {bfmul_i, bfmul_r};

assign o_strb = (cnt_o == 4) && (first_garbage == 1'b0);
assign o_data = (cnt_k < 512) ? {in0_i[15:0], in0_r[15:0]}
                              : {bf0_i, bf0_r};

mem_single #(
              .WD(32)
            , .DEPTH(512)
) i_mem (
              .clk(clk)
            , .cs(cs)
            , .we(we)
            , .addr(addr)
            , .din(w_data)
            , .dout(r_data)
);

endmodule
```

fft_pipe_stg1.v

```
module fft_pipe_stg1 (
              input     clk
            , input     n_reset

            , input             i_strb
            , input     [31:0]  i_data
            , output            o_strb
            , output    [31:0]  o_data
);


reg     [31:0]  i_data_d;
reg     [8:0]   cnt_k;  // counting input data
reg     [2:0]   cnt_o;  // counting butterfly operation
reg             first_garbage;

wire            last_o = (cnt_o == 4);
wire            last_k = (cnt_k == 511);

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        cnt_k <= 'b0;
```

```
        cnt_o <= 7;
        i_data_d <= 'b0;
        first_garbage <= 1'b1;
    end else begin
        if(i_strb == 1'b1) cnt_o <= 0;
        if(cnt_o < 5) cnt_o <= cnt_o+1;
        if(last_o == 1'b1) begin
            cnt_k <= (last_k == 1'b1) ? 'b0 : cnt_k+1;
            if(cnt_k == 255) first_garbage <= 1'b0;
        end
        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end

wire            cs;
wire            we;
reg     [7:0]   addr;
wire    [31:0]  r_data;
wire    [31:0]  w_data;

assign  cs = (cnt_o == 0) || (cnt_o == 4);
assign  we = (cnt_o == 4);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        addr <= 'b0;
    end else begin
        if(last_o == 1'b1) addr <= addr + 1;
    end
end

reg     signed  [16:0]  in0_r, in0_i;
wire    signed  [16:0]  in1_r, in1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        in0_r <= 'b0;
        in0_i <= 'b0;
    end else begin
        if(cnt_o == 1) begin
            in0_r <= {r_data[15], r_data[15:0]};
            in0_i <= {r_data[31], r_data[31:16]};
        end
    end
end
assign in1_r = {i_data_d[15], i_data_d[15:0]};
assign in1_i = {i_data_d[31], i_data_d[31:16]};

reg signed  [15:0]  bf0_r, bf0_i;
reg signed  [15:0]  bf1_r, bf1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bf0_r <= 'b0;
        bf0_i <= 'b0;
        bf1_r <= 'b0;
        bf1_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            bf0_r <= (in0_r + in1_r) >> 1;
            bf0_i <= (in0_i + in1_i) >> 1;
            bf1_r <= (in0_r - in1_r) >> 1;
            bf1_i <= (in0_i - in1_i) >> 1;
        end
    end
end

reg             [19:0]  twid_lut;
wire    signed  [9:0]   cos = (cnt_k[7:0] < 128) ? twid_lut[9:0]
                                                 : twid_lut[19:10];
wire    signed  [9:0]   sin = (cnt_k[7:0] < 128) ? twid_lut[19:10]
                                                 : -twid_lut[9:0];
reg     signed  [9:0]   twid_r, twid_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        twid_r <= 'b0;
        twid_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            twid_r <= cos;
            twid_i <= sin;
        end
    end
end

always@(*) begin
    case(cnt_k[6:0])
        0: twid_lut = {-10'd0,10'd511};
        1: twid_lut = {-10'd6,10'd511};
        2: twid_lut = {-10'd13,10'd511};
        3: twid_lut = {-10'd19,10'd511};
        ...
        124: twid_lut = {-10'd510,10'd25};
        125: twid_lut = {-10'd511,10'd19};
        126: twid_lut = {-10'd511,10'd13};
        127: twid_lut = {-10'd511,10'd6};
    endcase
end

wire signed [24:0]  mul_rr = bf1_r * twid_r;
wire signed [24:0]  mul_ri = bf1_r * twid_i;
wire signed [24:0]  mul_ir = bf1_i * twid_r;
wire signed [24:0]  mul_ii = bf1_i * twid_i;
reg  signed [15:0]  bfmul_r;
reg  signed [15:0]  bfmul_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bfmul_r <= 'b0;
        bfmul_i <= 'b0;
    end else begin
        if(cnt_o == 3) begin
            bfmul_r <= (mul_rr - mul_ii) >> 9;
            bfmul_i <= (mul_ri + mul_ir) >> 9;
        end
    end
end
assign w_data = (cnt_k < 256) ? i_data_d : {bfmul_i, bfmul_r};

assign o_strb = (cnt_o == 4) && (first_garbage == 1'b0);
assign o_data = (cnt_k < 256) ? {in0_i[15:0], in0_r[15:0]}
                              : {bf0_i, bf0_r};

mem_single #(
```

```
              .WD(32)
            , .DEPTH(256)
) i_mem (
              .clk(clk)
            , .cs(cs)
            , .we(we)
            , .addr(addr)
            , .din(w_data)
            , .dout(r_data)
);

endmodule
```

fft_pipe_stg2.v

```
module fft_pipe_stg2 (
              input     clk
            , input     n_reset

            , input             i_strb
            , input     [31:0]  i_data
            , output            o_strb
            , output    [31:0]  o_data
);


reg     [31:0]  i_data_d;
reg     [7:0]   cnt_k;  // counting input data
reg     [2:0]   cnt_o;  // counting butterfly operation
reg             first_garbage;

wire            last_o = (cnt_o == 4);
wire            last_k = (cnt_k == 255);

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        cnt_k <= 'b0;
        cnt_o <= 7;
        i_data_d <= 'b0;
        first_garbage <= 1'b1;
    end else begin
        if(i_strb == 1'b1) cnt_o <= 0;
        if(cnt_o < 5) cnt_o <= cnt_o+1;
        if(last_o == 1'b1) begin
            cnt_k <= (last_k == 1'b1) ? 'b0 : cnt_k+1;
            if(cnt_k == 127) first_garbage <= 1'b0;
        end
        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end

wire            cs;
wire            we;
reg     [6:0]   addr;
wire    [31:0]  r_data;
wire    [31:0]  w_data;

assign  cs = (cnt_o == 0) || (cnt_o == 4);
assign  we = (cnt_o == 4);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        addr <= 'b0;
    end else begin
        if(last_o == 1'b1) addr <= addr + 1;
    end
end

reg     signed  [16:0]  in0_r, in0_i;
wire    signed  [16:0]  in1_r, in1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        in0_r <= 'b0;
        in0_i <= 'b0;
    end else begin
        if(cnt_o == 1) begin
            in0_r <= {r_data[15], r_data[15:0]};
            in0_i <= {r_data[31], r_data[31:16]};
        end
    end
end
assign in1_r = {i_data_d[15], i_data_d[15:0]};
assign in1_i = {i_data_d[31], i_data_d[31:16]};

reg signed  [15:0]  bf0_r, bf0_i;
reg signed  [15:0]  bf1_r, bf1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bf0_r <= 'b0;
        bf0_i <= 'b0;
        bf1_r <= 'b0;
        bf1_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            bf0_r <= (in0_r + in1_r) >> 1;
            bf0_i <= (in0_i + in1_i) >> 1;
            bf1_r <= (in0_r - in1_r) >> 1;
            bf1_i <= (in0_i - in1_i) >> 1;
        end
    end
end

reg             [19:0]  twid_lut;
wire    signed  [9:0]   cos = (cnt_k[6:0] < 64) ? twid_lut[9:0]
                                                : twid_lut[19:10];
wire    signed  [9:0]   sin = (cnt_k[6:0] < 64) ? twid_lut[19:10]
                                                : -twid_lut[9:0];
reg     signed  [9:0]   twid_r, twid_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        twid_r <= 'b0;
        twid_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            twid_r <= cos;
            twid_i <= sin;
        end
    end
end

always@(*) begin
    case(cnt_k[5:0])
```

```
        0: twid_lut = {-10'd0,10'd511};
        1: twid_lut = {-10'd13,10'd511};
        2: twid_lut = {-10'd25,10'd510};
        3: twid_lut = {-10'd38,10'd510};
        ...
        60: twid_lut = {-10'd509,10'd50};
        61: twid_lut = {-10'd510,10'd38};
        62: twid_lut = {-10'd510,10'd25};
        63: twid_lut = {-10'd511,10'd13};
    endcase
end

wire signed [24:0]  mul_rr = bf1_r * twid_r;
wire signed [24:0]  mul_ri = bf1_r * twid_i;
wire signed [24:0]  mul_ir = bf1_i * twid_r;
wire signed [24:0]  mul_ii = bf1_i * twid_i;
reg  signed [15:0]  bfmul_r;
reg  signed [15:0]  bfmul_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bfmul_r <= 'b0;
        bfmul_i <= 'b0;
    end else begin
        if(cnt_o == 3) begin
            bfmul_r <= (mul_rr - mul_ii) >> 9;
            bfmul_i <= (mul_ri + mul_ir) >> 9;
        end
    end
end
assign w_data = (cnt_k < 128) ? i_data_d : {bfmul_i, bfmul_r};

assign o_strb = (cnt_o == 4) && (first_garbage == 1'b0);
assign o_data = (cnt_k < 128) ? {in0_i[15:0], in0_r[15:0]}
                              : {bf0_i, bf0_r};

mem_single #(
              .WD(32)
            , .DEPTH(128)
) i_mem (
              .clk(clk)
            , .cs(cs)
            , .we(we)
            , .addr(addr)
            , .din(w_data)
            , .dout(r_data)
);

endmodule
```

fft_pipe_stg7.v

```
module fft_pipe_stg7 (
              input     clk
            , input     n_reset

            , input             i_strb
            , input     [31:0]  i_data
            , output            o_strb
            , output    [31:0]  o_data
);


reg     [31:0]  i_data_d;
reg     [2:0]   cnt_k;  // counting input data
reg     [2:0]   cnt_o;  // counting butterfly operation
reg             first_garbage;

wire            last_o = (cnt_o == 4);
wire            last_k = (cnt_k == 7);

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        cnt_k <= 'b0;
        cnt_o <= 7;
        i_data_d <= 'b0;
        first_garbage <= 1'b1;
    end else begin
        if(i_strb == 1'b1) cnt_o <= 0;
        if(cnt_o < 5) cnt_o <= cnt_o+1;
        if(last_o == 1'b1) begin
            cnt_k <= (last_k == 1'b1) ? 'b0 : cnt_k+1;
            if(cnt_k == 3) first_garbage <= 1'b0;
        end
        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end

wire            cs;
wire            we;
reg     [1:0]   addr;
wire    [31:0]  r_data;
wire    [31:0]  w_data;

assign  cs = (cnt_o == 0) || (cnt_o == 4);
assign  we = (cnt_o == 4);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        addr <= 'b0;
    end else begin
        if(last_o == 1'b1) addr <= addr + 1;
    end
end

reg     signed  [16:0]  in0_r, in0_i;
wire    signed  [16:0]  in1_r, in1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        in0_r <= 'b0;
        in0_i <= 'b0;
    end else begin
        if(cnt_o == 1) begin
            in0_r <= {r_data[15], r_data[15:0]};
            in0_i <= {r_data[31], r_data[31:16]};
        end
    end
end
assign in1_r = {i_data_d[15], i_data_d[15:0]};
assign in1_i = {i_data_d[31], i_data_d[31:16]};

reg signed  [15:0]  bf0_r, bf0_i;
```

```
reg signed  [15:0]  bf1_r, bf1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bf0_r <= 'b0;
        bf0_i <= 'b0;
        bf1_r <= 'b0;
        bf1_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            bf0_r <= (in0_r + in1_r) >> 1;
            bf0_i <= (in0_i + in1_i) >> 1;
            bf1_r <= (in0_r - in1_r) >> 1;
            bf1_i <= (in0_i - in1_i) >> 1;
        end
    end
end

reg             [19:0]  twid_lut;
wire    signed  [9:0]   cos = (cnt_k[1:0] < 2) ? twid_lut[9:0]
                                               : twid_lut[19:10];
wire    signed  [9:0]   sin = (cnt_k[1:0] < 2) ? twid_lut[19:10]
                                               : -twid_lut[9:0];
reg     signed  [9:0]   twid_r, twid_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        twid_r <= 'b0;
        twid_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            twid_r <= cos;
            twid_i <= sin;
        end
    end
end

always@(*) begin
    case(cnt_k[0])
        0: twid_lut = {-10'd0,10'd511};
        1: twid_lut = {-10'd361,10'd361};
    endcase
end

wire signed [24:0]  mul_rr = bf1_r * twid_r;
wire signed [24:0]  mul_ri = bf1_r * twid_i;
wire signed [24:0]  mul_ir = bf1_i * twid_r;
wire signed [24:0]  mul_ii = bf1_i * twid_i;
reg  signed [15:0]  bfmul_r;
reg  signed [15:0]  bfmul_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bfmul_r <= 'b0;
        bfmul_i <= 'b0;
    end else begin
        if(cnt_o == 3) begin
            bfmul_r <= (mul_rr - mul_ii) >> 9;
            bfmul_i <= (mul_ri + mul_ir) >> 9;
        end
    end
end
```

```
assign w_data = (cnt_k < 4) ? i_data_d : {bfmul_i, bfmul_r};

assign o_strb = (cnt_o == 4) && (first_garbage == 1'b0);
assign o_data = (cnt_k < 4) ? {in0_i[15:0], in0_r[15:0]}
                            : {bf0_i, bf0_r};

mem_single #(
              .WD(32)
            , .DEPTH(4)
) i_mem (
              .clk(clk)
            , .cs(cs)
            , .we(we)
            , .addr(addr)
            , .din(w_data)
            , .dout(r_data)
);

endmodule
```

fft_pipe_stg8.v

```
module fft_pipe_stg8 (
              input     clk
            , input     n_reset

            , input             i_strb
            , input     [31:0]  i_data
            , output            o_strb
            , output    [31:0]  o_data
);


reg     [31:0]  i_data_d;
reg     [1:0]   cnt_k;  // counting input data
reg     [2:0]   cnt_o;  // counting butterfly operation
reg             first_garbage;

wire            last_o = (cnt_o == 4);
wire            last_k = (cnt_k == 3);

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        cnt_k <= 'b0;
        cnt_o <= 7;
        i_data_d <= 'b0;
        first_garbage <= 1'b1;
    end else begin
        if(i_strb == 1'b1) cnt_o <= 0;
        if(cnt_o < 5) cnt_o <= cnt_o+1;
        if(last_o == 1'b1) begin
            cnt_k <= (last_k == 1'b1) ? 'b0 : cnt_k+1;
            if(cnt_k == 1) first_garbage <= 1'b0;
        end
        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end

wire            cs;
wire            we;
reg             addr;
```

```
wire    [31:0]  r_data;
wire    [31:0]  w_data;

assign  cs = (cnt_o == 0) || (cnt_o == 4);
assign  we = (cnt_o == 4);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        addr <= 'b0;
    end else begin
        if(last_o == 1'b1) addr <= addr + 1;
    end
end

reg     signed  [16:0]  in0_r, in0_i;
wire    signed  [16:0]  in1_r, in1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        in0_r <= 'b0;
        in0_i <= 'b0;
    end else begin
        if(cnt_o == 1) begin
            in0_r <= {r_data[15], r_data[15:0]};
            in0_i <= {r_data[31], r_data[31:16]};
        end
    end
end
assign in1_r = {i_data_d[15], i_data_d[15:0]};
assign in1_i = {i_data_d[31], i_data_d[31:16]};

reg signed  [15:0]  bf0_r, bf0_i;
reg signed  [15:0]  bf1_r, bf1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bf0_r <= 'b0;
        bf0_i <= 'b0;
        bf1_r <= 'b0;
        bf1_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            bf0_r <= (in0_r + in1_r) >> 1;
            bf0_i <= (in0_i + in1_i) >> 1;
            bf1_r <= (in0_r - in1_r) >> 1;
            bf1_i <= (in0_i - in1_i) >> 1;
        end
    end
end

reg             [19:0]  twid_lut;
wire    signed  [9:0]   cos = (cnt_k[0] == 1'b0) ? 511 : 0;
wire    signed  [9:0]   sin = (cnt_k[0] == 1'b0) ? 0 : -511;
reg     signed  [9:0]   twid_r, twid_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        twid_r <= 'b0;
        twid_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            twid_r <= cos;
            twid_i <= sin;
        end
    end
end

wire signed [24:0]  mul_rr = bf1_r * twid_r;
wire signed [24:0]  mul_ri = bf1_r * twid_i;
wire signed [24:0]  mul_ir = bf1_i * twid_r;
wire signed [24:0]  mul_ii = bf1_i * twid_i;
reg  signed [15:0]  bfmul_r;
reg  signed [15:0]  bfmul_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bfmul_r <= 'b0;
        bfmul_i <= 'b0;
    end else begin
        if(cnt_o == 3) begin
            bfmul_r <= (mul_rr - mul_ii) >> 9;
            bfmul_i <= (mul_ri + mul_ir) >> 9;
        end
    end
end
assign w_data = (cnt_k < 2) ? i_data_d : {bfmul_i, bfmul_r};

assign o_strb = (cnt_o == 4) && (first_garbage == 1'b0);
assign o_data = (cnt_k < 2) ? {in0_i[15:0], in0_r[15:0]}
                            : {bf0_i, bf0_r};

mem_single #(
              .WD(32)
            , .DEPTH(2)
) i_mem (
              .clk(clk)
            , .cs(cs)
            , .we(we)
            , .addr(addr)
            , .din(w_data)
            , .dout(r_data)
);

endmodule
```

fft_pipe_stg9.v

```
module fft_pipe_stg9 (
              input     clk
            , input     n_reset

            , input             i_strb
            , input     [31:0]  i_data
            , output            o_strb
            , output    [31:0]  o_data
);


reg     [31:0]  i_data_d;
reg     [0:0]   cnt_k;  // counting input data
reg     [2:0]   cnt_o;  // counting butterfly operation
reg             first_garbage;

wire            last_o = (cnt_o == 4);
```

```
wire            last_k = (cnt_k == 1);

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        cnt_k <= 'b0;
        cnt_o <= 7;
        i_data_d <= 'b0;
        first_garbage <= 1'b1;
    end else begin
        if(i_strb == 1'b1) cnt_o <= 0;
        if(cnt_o < 5) cnt_o <= cnt_o+1;
        if(last_o == 1'b1) begin
            cnt_k <= (last_k == 1'b1) ? 'b0 : cnt_k+1;
            first_garbage <= 1'b0;
        end
        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end

wire            cs;
wire            we;
reg             addr;
wire    [31:0]  r_data;
wire    [31:0]  w_data;

assign  cs = (cnt_o == 0) || (cnt_o == 4);
assign  we = (cnt_o == 4);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        addr <= 'b0;
    end else begin
        if(last_o == 1'b1) addr <= addr + 1;
    end
end

reg     signed  [16:0]  in0_r, in0_i;
wire    signed  [16:0]  in1_r, in1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        in0_r <= 'b0;
        in0_i <= 'b0;
    end else begin
        if(cnt_o == 1) begin
            in0_r <= {r_data[15], r_data[15:0]};
            in0_i <= {r_data[31], r_data[31:16]};
        end
    end
end
assign in1_r = {i_data_d[15], i_data_d[15:0]};
assign in1_i = {i_data_d[31], i_data_d[31:16]};

reg signed  [15:0]  bf0_r, bf0_i;
reg signed  [15:0]  bf1_r, bf1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bf0_r <= 'b0;
        bf0_i <= 'b0;
        bf1_r <= 'b0;
        bf1_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            bf0_r <= (in0_r + in1_r) >> 1;
            bf0_i <= (in0_i + in1_i) >> 1;
            bf1_r <= (in0_r - in1_r) >> 1;
            bf1_i <= (in0_i - in1_i) >> 1;
        end
    end
end

reg             [19:0]  twid_lut;
wire    signed  [9:0]   cos = 511;
wire    signed  [9:0]   sin = 0;
reg     signed  [9:0]   twid_r, twid_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        twid_r <= 'b0;
        twid_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            twid_r <= cos;
            twid_i <= sin;
        end
    end
end

wire signed [24:0]  mul_rr = bf1_r * twid_r;
wire signed [24:0]  mul_ri = bf1_r * twid_i;
wire signed [24:0]  mul_ir = bf1_i * twid_r;
wire signed [24:0]  mul_ii = bf1_i * twid_i;
reg  signed [15:0]  bfmul_r;
reg  signed [15:0]  bfmul_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bfmul_r <= 'b0;
        bfmul_i <= 'b0;
    end else begin
        if(cnt_o == 3) begin
            bfmul_r <= (mul_rr - mul_ii) >> 9;
            bfmul_i <= (mul_ri + mul_ir) >> 9;
        end
    end
end
assign w_data = (cnt_k < 1) ? i_data_d : {bfmul_i, bfmul_r};

assign o_strb = (cnt_o == 4) && (first_garbage == 1'b0);
assign o_data = (cnt_k < 1) ? {in0_i[15:0], in0_r[15:0]}
                            : {bf0_i, bf0_r};

reg [31:0]  mem_data;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        mem_data <= 'b0;
    end else begin
        if((cs ==  1'b1) && (we == 1'b1)) begin
            mem_data <= w_data;
        end
    end
end
assign r_data = mem_data;
```

```
endmodule
```

top3.v - Strobed Input

```
module top_fft;

reg             clk, n_reset;
reg             i_strb;
reg     [31:0]  i_data;
wire            o_strb;
wire    [31:0]  o_data;

initial begin
    $vcdplusfile("top_fft.vpd");
    $vcdpluson(0, top_fft);
end
initial clk = 1'b0;
always #5 clk = ~clk;

import "DPI" function void init_fft();
import "DPI" function int unsigned get_input();
import "DPI" function int unsigned get_output();

int     i;
initial begin
    n_reset = 1'b1;
    i_strb = 1'b0;
    i_data = 'bx;
    init_fft();
    #3;
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
    @(posedge clk);
    @(posedge clk);
    repeat(2) begin
        for(i=0;i<1024;i++) begin
            #1;
            i_strb = 1'b1;
            i_data = get_input();
            @(posedge clk);
            #1;
            i_strb = 1'b0;
            i_data = 'bx;
            repeat(5) @(posedge clk);
        end
    end
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    $finish;
end

fft i_fft (
              .clk(clk)
            , .n_reset(n_reset)

            , .i_strb(i_strb)
            , .i_data(i_data)
            , .o_strb(o_strb)
            , .o_data(o_data)
);

int j;
```

```
reg [31:0]  c_data;
initial begin
    for(j=0;j<1024;j++) begin
        @(posedge o_strb);
        @(posedge clk);
        c_data = get_output();
        if(o_data !== c_data) begin
            $display("Error: o_data[%d] = %8X, c_data = %8X"
                    , j, o_data, c_data);
        end
    end
end

endmodule
```

fft_pipeline.v - Generate

```
module fft (
              input     clk
            , input     n_reset

            , input             i_strb
            , input     [31:0]  i_data
            , output            o_strb
            , output    [31:0]  o_data

);

wire    [9:0]       i_strbs;
wire    [9:0][31:0] i_datas;
wire    [9:0]       o_strbs;
wire    [9:0][31:0] o_datas;

genvar  i;
for(i=0;i<10;i++) begin
    if(i==0) begin
        assign i_strbs[i] = i_strb;
        assign i_datas[i] = i_data;
    end else begin
        assign i_strbs[i] = o_strbs[i-1];
        assign i_datas[i] = o_datas[i-1];
    end
    fft_pipe_stg_param #(
                .N(1024/(2**i))
        ) i_stg (
                .clk(clk)
              , .n_reset(n_reset)

              , .i_strb(i_strbs[i])
              , .i_data(i_datas[i])
              , .o_strb(o_strbs[i])
              , .o_data(o_datas[i])
        );
end

assign o_strb = o_strbs[9];
assign o_data = o_datas[9];

endmodule
```

fft_pipe_stg.v - Parameterized

```
module fft_pipe_stg_param #(
              N = 1024
) (
               input     clk
             , input     n_reset

             , input             i_strb
             , input     [31:0]  i_data
             , output            o_strb
             , output    [31:0]  o_data
);

localparam  W_K = $clog2(N);
reg     [31:0]      i_data_d;
reg     [W_K-1:0]   cnt_k;  // counting input data
reg     [2:0]       cnt_o;  // counting butterfly operation
reg                 first_garbage;

wire            last_o = (cnt_o == 4);
wire            last_k = (cnt_k == N-1);

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        cnt_k <= 'b0;
        cnt_o <= 7;
        i_data_d <= 'b0;
        first_garbage <= 1'b1;
    end else begin
        if(i_strb == 1'b1) cnt_o <= 0;
        if(cnt_o < 5) cnt_o <= cnt_o+1;
        if(last_o == 1'b1) begin
            cnt_k <= (last_k == 1'b1) ? 'b0 : cnt_k+1;
            if((N==1) || (cnt_k == N/2-1)) first_garbage <= 1'b0;
        end
        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end

wire                cs;
wire                we;
reg     [W_K-1:0]   addr;   // W_K-2 is right.
wire    [31:0]      r_data;
wire    [31:0]      w_data;

assign  cs = (cnt_o == 0) || (cnt_o == 4);
assign  we = (cnt_o == 4);
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        addr <= 'b0;
    end else begin
        if(last_o == 1'b1) addr <= addr + 1;
    end
end

reg     signed  [16:0]  in0_r, in0_i;
wire    signed  [16:0]  in1_r, in1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        in0_r <= 'b0;
        in0_i <= 'b0;
    end else begin
        if(cnt_o == 1) begin
            in0_r <= {r_data[15], r_data[15:0]};
            in0_i <= {r_data[31], r_data[31:16]};
        end
    end
end
assign in1_r = {i_data_d[15], i_data_d[15:0]};
assign in1_i = {i_data_d[31], i_data_d[31:16]};

reg signed  [15:0]  bf0_r, bf0_i;
reg signed  [15:0]  bf1_r, bf1_i;
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bf0_r <= 'b0;
        bf0_i <= 'b0;
        bf1_r <= 'b0;
        bf1_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            bf0_r <= (in0_r + in1_r) >> 1;
            bf0_i <= (in0_i + in1_i) >> 1;
            bf1_r <= (in0_r - in1_r) >> 1;
            bf1_i <= (in0_i - in1_i) >> 1;
        end
    end
end

reg             [19:0]  twid_lut;
wire    signed  [9:0]   cos;
wire    signed  [9:0]   sin;
wire            [7:0]   lut_idx;
if(N>4) begin
    assign cos = (cnt_k[W_K-2:0] < N/4) ? twid_lut[9:0] : twid_lut[19:10];
    assign sin = (cnt_k[W_K-2:0] < N/4) ? twid_lut[19:10] : -twid_lut[9:0];
    assign lut_idx = cnt_k[W_K-3:0] << (10-W_K);
end else if(N==4) begin
    assign cos = (cnt_k[0] == 1'b0) ? 511 : 0;
    assign sin = (cnt_k[0] == 1'b0) ? 0 : -511;
    assign lut_idx = 0;
end else if(N==2) begin
    assign cos = 511;
    assign sin = 0;
    assign lut_idx = 0;
end
reg     signed  [9:0]   twid_r, twid_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        twid_r <= 'b0;
        twid_i <= 'b0;
    end else begin
        if(cnt_o == 2) begin
            twid_r <= cos;
            twid_i <= sin;
        end
    end
end

always@(*) begin
    case(lut_idx)
```

```
        0: twid_lut = {-10'd0,10'd511};
        1: twid_lut = {-10'd3,10'd511};
        2: twid_lut = {-10'd6,10'd511};
        3: twid_lut = {-10'd9,10'd511};
        ...
        252: twid_lut = {-10'd511,10'd13};
        253: twid_lut = {-10'd511,10'd9};
        254: twid_lut = {-10'd511,10'd6};
        255: twid_lut = {-10'd511,10'd3};
    endcase
end

wire signed [24:0]  mul_rr = bf1_r * twid_r;
wire signed [24:0]  mul_ri = bf1_r * twid_i;
wire signed [24:0]  mul_ir = bf1_i * twid_r;
wire signed [24:0]  mul_ii = bf1_i * twid_i;
reg  signed [15:0]  bfmul_r;
reg  signed [15:0]  bfmul_i;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        bfmul_r <= 'b0;
        bfmul_i <= 'b0;
    end else begin
        if(cnt_o == 3) begin
            bfmul_r <= (mul_rr - mul_ii) >> 9;
            bfmul_i <= (mul_ri + mul_ir) >> 9;
        end
    end
end
assign w_data = (cnt_k < N/2) ? i_data_d : {bfmul_i, bfmul_r};

assign o_strb = (cnt_o == 4) && (first_garbage == 1'b0);
assign o_data = (cnt_k < N/2) ? {in0_i[15:0], in0_r[15:0]}
                              : {bf0_i, bf0_r};

if(N>16) begin
    mem_single #(
                  .WD(32)
                , .DEPTH(N/2)
    ) i_mem (
                  .clk(clk)
                , .cs(cs)
                , .we(we)
                , .addr(addr[W_K-2:0])
                , .din(w_data)
                , .dout(r_data)
    );
end else if(N>2) begin
    reg [N/2-1:0][31:0]     mem_data;
    int i;
    always@(posedge clk or negedge n_reset) begin
        if(n_reset == 1'b0) begin
            for(i=0;i<N/2;i++) mem_data[i] <= 'b0;
        end else begin
            if((cs ==  1'b1) && (we == 1'b1)) begin
                mem_data[addr[W_K-2:0]] <= w_data;
            end
        end
    end
    assign r_data = mem_data[addr[W_K-2:0]];
end else begin
    reg [31:0]      mem_data;
    always@(posedge clk or negedge n_reset) begin
        if(n_reset == 1'b0) begin
            mem_data <= 'b0;
        end else begin
            if((cs ==  1'b1) && (we == 1'b1)) begin
                mem_data <= w_data;
            end
        end
    end
    assign r_data = mem_data;
end

endmodule
```