

Chat : <https://chatgpt.com/share/281bb559-4f6b-4153-8c23-8c0c85ecdc2a>

This verilog code "module hanning\_window (

```
    input wire clk,  
    input wire rst,  
    input wire [15:0] sample_in,  
    input wire sample_valid,  
    output reg [15:0] sample_out_real,  
    output reg [15:0] sample_out_imag,  
    output reg sample_out_valid
```

```
);
```

```
localparam N = 256; // Frame size
```

```
localparam Q = 15; // Fixed-point precision
```

```
localparam NF = 512; // Power-of-two size for zero-padding
```

```
reg [15:0] sample_buffer [0:N-1];
```

```
reg [$clog2(NF)-1:0] sample_count;
```

```
reg [$clog2(N)-1:0] coeff_count;
```

```
reg [15:0] coeff;
```

```
reg [15:0] sample_out_real_reg;
```

```
reg [15:0] sample_out_imag_reg;
```

```
reg sample_out_valid_reg;
```

```
// Fixed-point constants
```

```
localparam [15:0] CONST_05 = 16'h4000; // 0.5 in Q15
```

```
localparam [15:0] CONST_2PI = 16'h6487; //  $2\pi$  in Q15
```

```
// CORDIC approximation of cosine
```

```
function [15:0] cordic_cos;
```

```

input [15:0] angle;
reg [15:0] x, y, z;
reg [3:0] i;
begin
    x = 16'h4DBA; // 0.607252935 in Q15
    y = 0;
    z = angle;

    for (i = 0; i < 12; i = i + 1) begin
        if (z[15] == 1) begin
            x = x - (y >>> i);
            y = y + (x >>> i);
            z = z + cordic_atan_table(i);
        end else begin
            x = x + (y >>> i);
            y = y - (x >>> i);
            z = z - cordic_atan_table(i);
        end
    end
end

cordic_cos = x;
end

endfunction

// CORDIC arctangent table (Q15)
function [15:0] cordic_atan_table;
    input [3:0] index;
begin
    case (index)
        4'd0: cordic_atan_table = 16'h3243;
        4'd1: cordic_atan_table = 16'h1DAC;
    endcase
end

```

```

4'd2: cordic_atan_table = 16'h0FAD;
4'd3: cordic_atan_table = 16'h07F5;
4'd4: cordic_atan_table = 16'h03FE;
4'd5: cordic_atan_table = 16'h01FF;
4'd6: cordic_atan_table = 16'h0100;
4'd7: cordic_atan_table = 16'h0080;
4'd8: cordic_atan_table = 16'h0040;
4'd9: cordic_atan_table = 16'h0020;
4'd10: cordic_atan_table = 16'h0010;
4'd11: cordic_atan_table = 16'h0008;
default: cordic_atan_table = 16'h0000;
endcase
end
endfunction

always @(posedge clk) begin
    if (rst) begin
        sample_count <= 0;
        coeff_count <= 0;
        sample_out_real_reg <= 0;
        sample_out_imag_reg <= 0;
        sample_out_valid_reg <= 0;
    end else begin
        if (sample_valid) begin
            sample_buffer[sample_count] <= sample_in;
            sample_count <= (sample_count == N-1) ? 0 : sample_count + 1;

            if (sample_count == N-1) begin
                coeff_count <= 0;
                sample_out_valid_reg <= 1;
            end else begin

```

```

        sample_out_valid_reg <= 0;
    end
end else begin
    if (coeff_count < N) begin
        coeff <= CONST_05 - (CONST_05 * cordic_cos((CONST_2PI * coeff_count) / N)) >>> Q;
        sample_out_real_reg <= (sample_buffer[coeff_count] * coeff) >>> Q;
        sample_out_imag_reg <= 0; // Hanning window is a real-valued function
        coeff_count <= coeff_count + 1;
    end else if (coeff_count < NF) begin
        sample_out_real_reg <= 0; // Zero-padding
        sample_out_imag_reg <= 0; // Zero-padding
        coeff_count <= coeff_count + 1;
    end else begin
        sample_out_valid_reg <= 0;
    end
end
end
end
end

```

```

always @(posedge clk) begin
    sample_out_real <= sample_out_real_reg;
    sample_out_imag <= sample_out_imag_reg;
    sample_out_valid <= sample_out_valid_reg;
end

```

endmodule" causes this error "Creating register for signal  
\hanning\_window.\$memwr\$\sample\_buffer\$/openlane/designs/mfcc\_accel/src/hanning\_window.v:  
84\$25592\_DATA' using process  
\hanning\_window.\$proc\$/openlane/designs/mfcc\_accel/src/hanning\_window.v:75\$25593'.  
created \$dff cellERROR: Multiple edge sensitive events found for this signal!". Refactor the code to fix  
the issue.

ChatGPT

The error you're encountering suggests that there are multiple edge-sensitive events for the signal. This typically occurs when there are conflicting assignments or multiple always blocks attempting to drive the same signal.

To address this issue, I'll refactor the code by ensuring that each signal is driven by only one always block. The sample buffer management will be handled separately from the coefficient calculation and output generation. Here is the refactored code:

verilog

Copy code

```
module hanning_window (  
    input wire clk,  
    input wire rst,  
    input wire [15:0] sample_in,  
    input wire sample_valid,  
    output reg [15:0] sample_out_real,  
    output reg [15:0] sample_out_imag,  
    output reg sample_out_valid  
);  
  
localparam N = 256; // Frame size  
localparam Q = 15; // Fixed-point precision  
localparam NF = 512; // Power-of-two size for zero-padding  
  
reg [15:0] sample_buffer [0:N-1];  
reg [$clog2(NF)-1:0] sample_count;  
reg [$clog2(N)-1:0] coeff_count;  
reg [15:0] coeff;  
reg [15:0] sample_out_real_reg;  
reg [15:0] sample_out_imag_reg;  
reg sample_out_valid_reg;
```

```
// Fixed-point constants

localparam [15:0] CONST_05 = 16'h4000; // 0.5 in Q15
localparam [15:0] CONST_2PI = 16'h6487; //  $2\pi$  in Q15
```

```
// CORDIC approximation of cosine
```

```
function [15:0] cordic_cos;
```

```
    input [15:0] angle;
```

```
    reg [15:0] x, y, z;
```

```
    reg [3:0] i;
```

```
begin
```

```
    x = 16'h4DBA; // 0.607252935 in Q15
```

```
    y = 0;
```

```
    z = angle;
```

```
    for (i = 0; i < 12; i = i + 1) begin
```

```
        if (z[15] == 1) begin
```

```
            x = x - (y >>> i);
```

```
            y = y + (x >>> i);
```

```
            z = z + cordic_atan_table(i);
```

```
        end else begin
```

```
            x = x + (y >>> i);
```

```
            y = y - (x >>> i);
```

```
            z = z - cordic_atan_table(i);
```

```
        end
```

```
    end
```

```
    cordic_cos = x;
```

```
end
```

```
endfunction
```

```
// CORDIC arctangent table (Q15)
```

```

function [15:0] cordic_atan_table;
    input [3:0] index;
begin
    case (index)
        4'd0: cordic_atan_table = 16'h3243;
        4'd1: cordic_atan_table = 16'h1DAC;
        4'd2: cordic_atan_table = 16'h0FAD;
        4'd3: cordic_atan_table = 16'h07F5;
        4'd4: cordic_atan_table = 16'h03FE;
        4'd5: cordic_atan_table = 16'h01FF;
        4'd6: cordic_atan_table = 16'h0100;
        4'd7: cordic_atan_table = 16'h0080;
        4'd8: cordic_atan_table = 16'h0040;
        4'd9: cordic_atan_table = 16'h0020;
        4'd10: cordic_atan_table = 16'h0010;
        4'd11: cordic_atan_table = 16'h0008;
        default: cordic_atan_table = 16'h0000;
    endcase
end
endfunction

```

```

// Sample buffer management

```

```

always @(posedge clk or posedge rst) begin
    if (rst) begin
        sample_count <= 0;
        sample_out_valid_reg <= 0;
    end else begin
        if (sample_valid) begin
            sample_buffer[sample_count] <= sample_in;
            sample_count <= (sample_count == N-1) ? 0 : sample_count + 1;
        end
    end
end

```

```

    if (sample_count == N-1) begin
        sample_out_valid_reg <= 1;
    end else begin
        sample_out_valid_reg <= 0;
    end
end else if (sample_out_valid_reg) begin
    sample_out_valid_reg <= 0;
end
end
end

// Coefficient calculation and output generation
always @(posedge clk or posedge rst) begin
    if (rst) begin
        coeff_count <= 0;
        sample_out_real_reg <= 0;
        sample_out_imag_reg <= 0;
    end else begin
        if (sample_out_valid_reg && (coeff_count < N)) begin
            coeff <= CONST_05 - (CONST_05 * cordic_cos((CONST_2PI * coeff_count) / N)) >>> Q;
            sample_out_real_reg <= (sample_buffer[coeff_count] * coeff) >>> Q;
            sample_out_imag_reg <= 0; // Hanning window is a real-valued function
            coeff_count <= coeff_count + 1;
        end else if (coeff_count < NF) begin
            sample_out_real_reg <= 0; // Zero-padding
            sample_out_imag_reg <= 0; // Zero-padding
            coeff_count <= coeff_count + 1;
        end
    end
end
end
end

```



```
// Output assignments

always @(posedge clk or posedge rst) begin

    if (rst) begin

        sample_out_real <= 0;

        sample_out_imag <= 0;

        sample_out_valid <= 0;

    end else begin

        sample_out_real <= sample_out_real_reg;

        sample_out_imag <= sample_out_imag_reg;

        sample_out_valid <= sample_out_valid_reg;

    end

end

endmodule
```

In this refactored code, the sample buffer management, coefficient calculation, and output generation are separated into different always blocks, each driven by the same clock and reset signals. This should resolve the "Multiple edge sensitive events" error. The sample buffer is updated in one always block, while the coefficient calculation and output generation are handled in another. Finally, the output assignments are managed in a separate always block to ensure there are no conflicting signal assignments.

Creating register for signal '\hanning\_window.\sample\_buffer[248]' using process  
'\hanning\_window.\$proc\$/openlane/designs/mfcc\_accel/src/hanning\_window.v:76\$25594'.

created \$dff ce!ERROR: Multiple edge sensitive events found for this signal!

ChatGPT

The error indicates that there are multiple edge-sensitive assignments for the same signal, which typically happens if a register is being driven by more than one always block or if there are conflicting assignments within the same always block.

Let's correct this by making sure the signal is assigned in only one always block. I'll also ensure that each block has a single purpose and that there is no overlap in the handling of signals.

Here's the revised code:

verilog

Copy code

```
module hanning_window (  
    input wire clk,  
    input wire rst,  
    input wire [15:0] sample_in,  
    input wire sample_valid,  
    output reg [15:0] sample_out_real,  
    output reg [15:0] sample_out_imag,  
    output reg sample_out_valid  
);  
  
localparam N = 256; // Frame size  
localparam Q = 15; // Fixed-point precision  
localparam NF = 512; // Power-of-two size for zero-padding  
  
reg [15:0] sample_buffer [0:N-1];  
reg [$clog2(NF)-1:0] sample_count;  
reg [$clog2(N)-1:0] coeff_count;  
reg [15:0] coeff;  
reg [15:0] sample_out_real_reg;  
reg [15:0] sample_out_imag_reg;  
reg sample_out_valid_reg;  
  
// Fixed-point constants  
localparam [15:0] CONST_05 = 16'h4000; // 0.5 in Q15  
localparam [15:0] CONST_2PI = 16'h6487; //  $2\pi$  in Q15  
  
// CORDIC approximation of cosine  
function [15:0] cordic_cos;  
    input [15:0] angle;
```

```

    reg [15:0] x, y, z;
    reg [3:0] i;
begin
    x = 16'h4DBA; // 0.607252935 in Q15
    y = 0;
    z = angle;

    for (i = 0; i < 12; i = i + 1) begin
        if (z[15] == 1) begin
            x = x - (y >>> i);
            y = y + (x >>> i);
            z = z + cordic_atan_table(i);
        end else begin
            x = x + (y >>> i);
            y = y - (x >>> i);
            z = z - cordic_atan_table(i);
        end
    end
end

    cordic_cos = x;
end
endfunction

// CORDIC arctangent table (Q15)
function [15:0] cordic_atan_table;
    input [3:0] index;
begin
    case (index)
        4'd0: cordic_atan_table = 16'h3243;
        4'd1: cordic_atan_table = 16'h1DAC;
        4'd2: cordic_atan_table = 16'h0FAD;
    endcase
end

```

```

    4'd3: cordic_atan_table = 16'h07F5;
    4'd4: cordic_atan_table = 16'h03FE;
    4'd5: cordic_atan_table = 16'h01FF;
    4'd6: cordic_atan_table = 16'h0100;
    4'd7: cordic_atan_table = 16'h0080;
    4'd8: cordic_atan_table = 16'h0040;
    4'd9: cordic_atan_table = 16'h0020;
    4'd10: cordic_atan_table = 16'h0010;
    4'd11: cordic_atan_table = 16'h0008;
    default: cordic_atan_table = 16'h0000;
endcase
end
endfunction

// Sample buffer management
always @(posedge clk or posedge rst) begin
    if (rst) begin
        sample_count <= 0;
    end else begin
        if (sample_valid) begin
            sample_buffer[sample_count] <= sample_in;
            sample_count <= (sample_count == N-1) ? 0 : sample_count + 1;
        end
    end
end

// Coefficient calculation and output generation
always @(posedge clk or posedge rst) begin
    if (rst) begin
        coeff_count <= 0;
        sample_out_real_reg <= 0;
    end
end

```

```

    sample_out_imag_reg <= 0;
    sample_out_valid_reg <= 0;
end else begin
    if (sample_valid && sample_count == N-1) begin
        coeff_count <= 0;
        sample_out_valid_reg <= 1;
    end else if (coeff_count < N) begin
        coeff <= CONST_05 - ((CONST_05 * cordic_cos((CONST_2PI * coeff_count) / N)) >>> Q);
        sample_out_real_reg <= (sample_buffer[coeff_count] * coeff) >>> Q;
        sample_out_imag_reg <= 0; // Hanning window is a real-valued function
        coeff_count <= coeff_count + 1;
    end else if (coeff_count < NF) begin
        sample_out_real_reg <= 0; // Zero-padding
        sample_out_imag_reg <= 0; // Zero-padding
        coeff_count <= coeff_count + 1;
    end else begin
        sample_out_valid_reg <= 0;
    end
end
end
end

```

```

// Output assignments

```

```

always @(posedge clk or posedge rst) begin

```

```

    if (rst) begin

```

```

        sample_out_real <= 0;

```

```

        sample_out_imag <= 0;

```

```

        sample_out_valid <= 0;

```

```

    end else begin

```

```

        sample_out_real <= sample_out_real_reg;

```

```

        sample_out_imag <= sample_out_imag_reg;

```

```

        sample_out_valid <= sample_out_valid_reg;

```

```
end  
end
```

```
endmodule
```

In this refactored version, the sample buffer is managed in a separate always block that only updates the sample buffer and sample count when sample\_valid is high. The coefficient calculation and output generation are handled in another always block that operates based on the sample\_count reaching the end of the buffer. This separation should prevent multiple edge-sensitive events on the same signal, resolving the error.

Creating register for signal '\hanning\_window.\sample\_buffer[248]' using process  
\hanning\_window.\$proc\$/openlane/designs/mfcc\_accel/src/hanning\_window.v:76\$25594'.

created \$dff cell ` \$prERROR: Multiple edge sensitive events found for this signal!

ChatGPT

The error you're encountering suggests that there are still conflicting assignments for sample\_buffer. The main issue is that Verilog requires that any given signal be driven by exactly one always block. Given that sample\_buffer is a multi-bit signal, it is necessary to ensure it is only assigned in one place and that other logic is separated properly.

To fix this, let's separate the process of updating sample\_buffer and generating the output signals into distinct always blocks. We'll make sure that the sample\_buffer is updated only within a single always block to avoid multiple edge-sensitive events for the same signal.

Here is the corrected Verilog code:

verilog

Copy code

```
module hanning_window (  
    input wire clk,  
    input wire rst,  
    input wire [15:0] sample_in,  
    input wire sample_valid,  
    output reg [15:0] sample_out_real,  
    output reg [15:0] sample_out_imag,
```

```

        output reg sample_out_valid
    );

    localparam N = 256; // Frame size
    localparam Q = 15; // Fixed-point precision
    localparam NF = 512; // Power-of-two size for zero-padding

    reg [15:0] sample_buffer [0:N-1];
    reg [$clog2(NF)-1:0] sample_count;
    reg [$clog2(N)-1:0] coeff_count;
    reg [15:0] coeff;
    reg [15:0] sample_out_real_reg;
    reg [15:0] sample_out_imag_reg;
    reg sample_out_valid_reg;

    // Fixed-point constants
    localparam [15:0] CONST_05 = 16'h4000; // 0.5 in Q15
    localparam [15:0] CONST_2PI = 16'h6487; // 2 $\pi$  in Q15

    // CORDIC approximation of cosine
    function [15:0] cordic_cos;
        input [15:0] angle;
        reg [15:0] x, y, z;
        reg [3:0] i;
    begin
        x = 16'h4DBA; // 0.607252935 in Q15
        y = 0;
        z = angle;

        for (i = 0; i < 12; i = i + 1) begin
            if (z[15] == 1) begin

```

```

        x = x - (y >>> i);
        y = y + (x >>> i);
        z = z + cordic_atan_table(i);
    end else begin
        x = x + (y >>> i);
        y = y - (x >>> i);
        z = z - cordic_atan_table(i);
    end
end

cordic_cos = x;
end
endfunction

// CORDIC arctangent table (Q15)
function [15:0] cordic_atan_table;
    input [3:0] index;
begin
    case (index)
        4'd0: cordic_atan_table = 16'h3243;
        4'd1: cordic_atan_table = 16'h1DAC;
        4'd2: cordic_atan_table = 16'h0FAD;
        4'd3: cordic_atan_table = 16'h07F5;
        4'd4: cordic_atan_table = 16'h03FE;
        4'd5: cordic_atan_table = 16'h01FF;
        4'd6: cordic_atan_table = 16'h0100;
        4'd7: cordic_atan_table = 16'h0080;
        4'd8: cordic_atan_table = 16'h0040;
        4'd9: cordic_atan_table = 16'h0020;
        4'd10: cordic_atan_table = 16'h0010;
        4'd11: cordic_atan_table = 16'h0008;
    endcase
end

```



```

        default: cordic_atan_table = 16'h0000;
    endcase
end
endfunction

// Sample buffer management
always @(posedge clk or posedge rst) begin
    if (rst) begin
        sample_count <= 0;
    end else begin
        if (sample_valid) begin
            sample_buffer[sample_count] <= sample_in;
            sample_count <= (sample_count == N-1) ? 0 : sample_count + 1;
        end
    end
end

// Coefficient calculation and output generation
always @(posedge clk or posedge rst) begin
    if (rst) begin
        coeff_count <= 0;
        sample_out_real_reg <= 0;
        sample_out_imag_reg <= 0;
        sample_out_valid_reg <= 0;
    end else begin
        if (sample_count == N && sample_valid) begin
            coeff_count <= 0;
            sample_out_valid_reg <= 1;
        end else if (sample_out_valid_reg) begin
            if (coeff_count < N) begin
                coeff <= CONST_05 - ((CONST_05 * cordic_cos((CONST_2PI * coeff_count) / N)) >>> Q);
            end
        end
    end
end

```

```

    sample_out_real_reg <= (sample_buffer[coeff_count] * coeff) >>> Q;
    sample_out_imag_reg <= 0; // Hanning window is a real-valued function
    coeff_count <= coeff_count + 1;
end else if (coeff_count < NF) begin
    sample_out_real_reg <= 0; // Zero-padding
    sample_out_imag_reg <= 0; // Zero-padding
    coeff_count <= coeff_count + 1;
end else begin
    sample_out_valid_reg <= 0;
end
end
end
end

// Output assignments
always @(posedge clk or posedge rst) begin
    if (rst) begin
        sample_out_real <= 0;
        sample_out_imag <= 0;
        sample_out_valid <= 0;
    end else begin
        sample_out_real <= sample_out_real_reg;
        sample_out_imag <= sample_out_imag_reg;
        sample_out_valid <= sample_out_valid_reg;
    end
end

endmodule

```

In this revised code:

Sample Buffer Management: The sample\_buffer is updated only within one always block. This avoids multiple assignments to the same register.

Coefficient Calculation and Output Generation: The coefficient calculation and output generation are handled in a separate always block. The coeff\_count is reset when sample\_count reaches N and sample\_valid is high, indicating that a full frame has been received.

Output Assignments: The output signals (sample\_out\_real, sample\_out\_imag, and sample\_out\_valid) are assigned in another always block driven by clk and rst.

This separation ensures that each signal is driven by a single always block, thus avoiding multiple edge-sensitive events for the same signal and resolving the error.

Also can you find a way to implement the real-valued Hanning window code?

ChatGPT

Certainly! The Hanning window is a real-valued window function used to smooth signals before performing a Fourier transform. The window function can be mathematically expressed as:

$$w(n) = 0.5 \left( 1 - \cos\left(\frac{2\pi n}{N} \right) \right)$$

```

)
)
w(n)=0.5(1-cos(
N-1
2πn

))

```

In fixed-point arithmetic, we can implement this by precomputing the cosine values using the CORDIC algorithm and then using these values to compute the window coefficients.

Here's the revised Verilog code to implement the real-valued Hanning window:

verilog

Copy code

```

module hanning_window (
    input wire clk,
    input wire rst,
    input wire [15:0] sample_in,
    input wire sample_valid,
    output reg [15:0] sample_out_real,
    output reg sample_out_valid
);

    localparam N = 256; // Frame size
    localparam Q = 15; // Fixed-point precision
    localparam NF = 512; // Power-of-two size for zero-padding

    reg [15:0] sample_buffer [0:N-1];
    reg [$clog2(NF)-1:0] sample_count;
    reg [$clog2(N)-1:0] coeff_count;

```

```

reg [15:0] coeff;

reg [15:0] sample_out_real_reg;

reg sample_out_valid_reg;


// Fixed-point constants
localparam [15:0] CONST_05 = 16'h4000; // 0.5 in Q15
localparam [15:0] CONST_2PI = 16'h6487; //  $2\pi$  in Q15


// CORDIC approximation of cosine
function [15:0] cordic_cos;
    input [15:0] angle;
    reg [15:0] x, y, z;
    reg [3:0] i;
begin
    x = 16'h4DBA; // 0.607252935 in Q15
    y = 0;
    z = angle;

    for (i = 0; i < 12; i = i + 1) begin
        if (z[15] == 1) begin
            x = x - (y >>> i);
            y = y + (x >>> i);
            z = z + cordic_atan_table(i);
        end else begin
            x = x + (y >>> i);
            y = y - (x >>> i);
            z = z - cordic_atan_table(i);
        end
    end
end

cordic_cos = x;

```

end

endfunction

// CORDIC arctangent table (Q15)

function [15:0] cordic\_atan\_table;

input [3:0] index;

begin

case (index)

4'd0: cordic\_atan\_table = 16'h3243;

4'd1: cordic\_atan\_table = 16'h1DAC;

4'd2: cordic\_atan\_table = 16'h0FAD;

4'd3: cordic\_atan\_table = 16'h07F5;

4'd4: cordic\_atan\_table = 16'h03FE;

4'd5: cordic\_atan\_table = 16'h01FF;

4'd6: cordic\_atan\_table = 16'h0100;

4'd7: cordic\_atan\_table = 16'h0080;

4'd8: cordic\_atan\_table = 16'h0040;

4'd9: cordic\_atan\_table = 16'h0020;

4'd10: cordic\_atan\_table = 16'h0010;

4'd11: cordic\_atan\_table = 16'h0008;

default: cordic\_atan\_table = 16'h0000;

endcase

end

endfunction

// Sample buffer management

always @(posedge clk or posedge rst) begin

if (rst) begin

sample\_count <= 0;

end else begin

if (sample\_valid) begin

```

        sample_buffer[sample_count] <= sample_in;

        sample_count <= (sample_count == N-1) ? 0 : sample_count + 1;

    end

end

end

// Coefficient calculation and output generation
always @(posedge clk or posedge rst) begin

    if (rst) begin

        coeff_count <= 0;

        sample_out_real_reg <= 0;

        sample_out_valid_reg <= 0;

    end else begin

        if (sample_count == N && sample_valid) begin

            coeff_count <= 0;

            sample_out_valid_reg <= 1;

        end else if (sample_out_valid_reg) begin

            if (coeff_count < N) begin

                //  $w(n) = 0.5 * (1 - \cos(2\pi * n / (N-1)))$ 

                //  $coeff = 0.5 * (1 - \cos(2\pi * coeff\_count / (N-1)))$ 

                coeff <= CONST_05 - ((CONST_05 * cordic_cos((CONST_2PI * coeff_count) / (N-1))) >>> Q);

                sample_out_real_reg <= (sample_buffer[coeff_count] * coeff) >>> Q;

                coeff_count <= coeff_count + 1;

            end else if (coeff_count < NF) begin

                sample_out_real_reg <= 0; // Zero-padding

                coeff_count <= coeff_count + 1;

            end else begin

                sample_out_valid_reg <= 0;

            end

        end

    end

end
end

```

```

end

// Output assignments
always @(posedge clk or posedge rst) begin
    if (rst) begin
        sample_out_real <= 0;
        sample_out_valid <= 0;
    end else begin
        sample_out_real <= sample_out_real_reg;
        sample_out_valid <= sample_out_valid_reg;
    end
end

endmodule

```

Explanation:

**CORDIC Cosine Function:** The `cordic_cos` function approximates the cosine of an angle using the CORDIC algorithm.

**Sample Buffer Management:** This always block updates the sample buffer and sample count whenever `sample_valid` is high.

**Coefficient Calculation and Output Generation:** This always block computes the Hanning window coefficients using the CORDIC-based cosine function and multiplies the input samples by these coefficients.

**Output Assignments:** The final always block assigns the computed real-valued Hanning window output to the output ports.

The key changes:

The Hanning window coefficient calculation is explicitly defined as  $0.5 * (1 - \cos(2\pi * \text{coeff\_count} / (N-1)))$ .

The sample buffer management and coefficient calculation are separated into different always blocks to ensure there's no conflict in signal assignments.

Zero-padding is handled appropriately for `coeff_count` values greater than `N`.

This ensures a proper implementation of the real-valued Hanning window, avoiding multiple edge-sensitive events on the same signal.



hanning\_out\_imag is missing. I want that part too.

ChatGPT

Sure, we can add the sample\_out\_imag output as well. Since the Hanning window is a real-valued function, the imaginary part will always be zero. However, for completeness and to meet your requirement, we'll include the imaginary part in the output.

Here is the updated Verilog code:

verilog

Copy code

```
module hanning_window (  
    input wire clk,  
    input wire rst,  
    input wire [15:0] sample_in,  
    input wire sample_valid,  
    output reg [15:0] sample_out_real,  
    output reg [15:0] sample_out_imag,  
    output reg sample_out_valid  
);  
  
localparam N = 256; // Frame size  
localparam Q = 15; // Fixed-point precision  
localparam NF = 512; // Power-of-two size for zero-padding  
  
reg [15:0] sample_buffer [0:N-1];  
reg [$clog2(NF)-1:0] sample_count;  
reg [$clog2(N)-1:0] coeff_count;  
reg [15:0] coeff;  
reg [15:0] sample_out_real_reg;  
reg [15:0] sample_out_imag_reg;  
reg sample_out_valid_reg;
```

```
// Fixed-point constants  
localparam [15:0] CONST_05 = 16'h4000; // 0.5 in Q15  
localparam [15:0] CONST_2PI = 16'h6487; //  $2\pi$  in Q15
```

```
// CORDIC approximation of cosine
```

```
function [15:0] cordic_cos;  
    input [15:0] angle;  
    reg [15:0] x, y, z;  
    reg [3:0] i;  
begin  
    x = 16'h4DBA; // 0.607252935 in Q15  
    y = 0;  
    z = angle;  
  
    for (i = 0; i < 12; i = i + 1) begin  
        if (z[15] == 1) begin  
            x = x - (y >>> i);  
            y = y + (x >>> i);  
            z = z + cordic_atan_table(i);  
        end else begin  
            x = x + (y >>> i);  
            y = y - (x >>> i);  
            z = z - cordic_atan_table(i);  
        end  
    end  
end  
  
    cordic_cos = x;  
end  
endfunction
```

```
// CORDIC arctangent table (Q15)
```

```

function [15:0] cordic_atan_table;
    input [3:0] index;
begin
    case (index)
        4'd0: cordic_atan_table = 16'h3243;
        4'd1: cordic_atan_table = 16'h1DAC;
        4'd2: cordic_atan_table = 16'h0FAD;
        4'd3: cordic_atan_table = 16'h07F5;
        4'd4: cordic_atan_table = 16'h03FE;
        4'd5: cordic_atan_table = 16'h01FF;
        4'd6: cordic_atan_table = 16'h0100;
        4'd7: cordic_atan_table = 16'h0080;
        4'd8: cordic_atan_table = 16'h0040;
        4'd9: cordic_atan_table = 16'h0020;
        4'd10: cordic_atan_table = 16'h0010;
        4'd11: cordic_atan_table = 16'h0008;
        default: cordic_atan_table = 16'h0000;
    endcase
end

endfunction

// Sample buffer management
always @(posedge clk or posedge rst) begin
    if (rst) begin
        sample_count <= 0;
    end else begin
        if (sample_valid) begin
            sample_buffer[sample_count] <= sample_in;
            sample_count <= (sample_count == N-1) ? 0 : sample_count + 1;
        end
    end
end

```

end

// Coefficient calculation and output generation

always @(posedge clk or posedge rst) begin

if (rst) begin

coeff\_count <= 0;

sample\_out\_real\_reg <= 0;

sample\_out\_imag\_reg <= 0;

sample\_out\_valid\_reg <= 0;

end else begin

if (sample\_count == N && sample\_valid) begin

coeff\_count <= 0;

sample\_out\_valid\_reg <= 1;

end else if (sample\_out\_valid\_reg) begin

if (coeff\_count < N) begin

//  $w(n) = 0.5 * (1 - \cos(2\pi * n / (N-1)))$

//  $coeff = 0.5 * (1 - \cos(2\pi * coeff\_count / (N-1)))$

coeff <= CONST\_05 - ((CONST\_05 \* cordic\_cos((CONST\_2PI \* coeff\_count) / (N-1))) >>> Q);

sample\_out\_real\_reg <= (sample\_buffer[coeff\_count] \* coeff) >>> Q;

sample\_out\_imag\_reg <= 0; // Hanning window is a real-valued function

coeff\_count <= coeff\_count + 1;

end else if (coeff\_count < NF) begin

sample\_out\_real\_reg <= 0; // Zero-padding

sample\_out\_imag\_reg <= 0; // Zero-padding

coeff\_count <= coeff\_count + 1;

end else begin

sample\_out\_valid\_reg <= 0;

end

end

end

end

```
// Output assignments
always @(posedge clk or posedge rst) begin
    if (rst) begin
        sample_out_real <= 0;
        sample_out_imag <= 0;
        sample_out_valid <= 0;
    end else begin
        sample_out_real <= sample_out_real_reg;
        sample_out_imag <= sample_out_imag_reg;
        sample_out_valid <= sample_out_valid_reg;
    end
end

endmodule
```

Explanation:

Added sample\_out\_imag Output: The imaginary part of the output, sample\_out\_imag, is included in the output assignment always block.

Hanning Window Calculation: The real-valued Hanning window calculation is performed and assigned to sample\_out\_real. Since the Hanning window is real-valued, sample\_out\_imag is always zero.

Zero-padding: Both sample\_out\_real and sample\_out\_imag are set to zero for zero-padding when coeff\_count exceeds N.

This ensures the Hanning window is applied to the input samples, with the real part (sample\_out\_real) computed using the window coefficients and the imaginary part (sample\_out\_imag) set to zero, as required.