

プロンプト

“verilogで低周波FIRフィルターの簡単なコード例を示して。

5kHzカットオフ

タップ数は21

入力パラメータとしてリセット必要。

入力パラメータとして、音声データを想定した符号付き整数12bit,

出力パラメータとして、符号付き整数12bit

出力パラメータとして、オーバーフローのある1/なし0

FIRの計算はオーバーフローを考慮した符号付き整数で行う

FIRの計算はスケールファクター128の固定小数点

係数はhamming窓、低域通過で計算する。

クロックは300kHz

verilogの文法は最新のベストプラクティス。

なるべくwireは使わずにlogicを使うこと。

verilogのコメントは英語で。

verilogソースの先頭に今日の日付を書く（米式）

verilogソースの先頭にGrokをベースにした旨のコメントを書く。

verilogソースの先頭に"prompt by tommie.jp (GitHub:open-tommie)"と書く。

回答は日本語で。

回答の冒頭に「プロンプト」と題して、このプロンプト全体を引用する。

次に仕様のまとめ。

次にコード例”

仕様のまとめ

この例は、低域通過FIRフィルタを実装したVerilogコードです。主な仕様は以下の通りです：

- ・ カットオフ周波数: 5kHz

- ・タップ数: 21
- ・サンプリング周波数（クロック周波数に基づく）: 300kHz
- ・入力: 符号付き12ビット整数（音声データを想定）
- ・出力: 符号付き12ビット整数、およびオーバーフローフラグ（1: オーバーフロー有り、0: 無し）
- ・計算: 符号付き整数を使用し、オーバーフローを検出。固定小数点演算（スケールファクター128）
- ・係数: Hamming窓を適用した低域通過フィルタ係数を使用（事前計算済み: [1,1,2,3,5,6,8,10,11,12,12,12,11,10,8,6,5,3,2,1,1]）
- ・リセット: 同期リセット入力
- ・その他: 計算結果が12ビット範囲を超えた場合、オーバーフローを検出して出力値を飽和させます。クロックは300kHzを想定した設計ですが、コード自体は周波数に依存しません。

コード例

以下は、指定に基づいたVerilogコードの例です。SystemVerilogのベストプラクティスを採用し、logic型を主に使用しています。

verilog

✕ 折りたたむ ≡ 広げる 📄 コピー

```
// 10/19/2025
// Based on Grok by xAI
// prompt by tommie.jp (GitHub:open-tommie)

// Module for a simple low-pass FIR filter with 21 taps
// Cutoff: 5kHz, Sampling rate: 300kHz, Hamming window
// Fixed-point with scale factor 128
// Input: signed 12-bit, Output: signed 12-bit with overflow flag

module fir_filter (
    input logic clk,                // Clock input (300kHz)
    input logic rst,                // Synchronous reset

    input
    logic signed [11:0] in_data,    // Signed 12-bit input data
```

```

        output logic signed [11:0] out_data,          // Signed 12-bit output data
        output logic overflow                        // Overflow flag (1:
overflow, 0: no)
    );

    // Shift register for input samples (21 taps)
    logic signed [11:0] shift_reg [0:20];

    // Accumulator for FIR computation (wide bit width to prevent internal
overflow)
    logic signed [31:0] accum;

    // Pre-computed coefficients (scaled by 128, Hamming window LPF)
    const logic signed [7:0] coeffs [0:20] = '{
        8'sd1, 8'sd1, 8'sd2, 8'sd3, 8'sd5, 8'sd6, 8'sd8, 8'sd10, 8'sd11,
8'sd12,
        8'sd12, 8'sd12, 8'sd11, 8'sd10, 8'sd8, 8'sd6, 8'sd5, 8'sd3, 8'sd2,
8'sd1, 8'sd1
    };

    // Sequential logic for shift and computation
    always_ff @(posedge clk) begin
        if (rst) begin
            // Reset shift register to zero
            for (int i = 0; i < 21; i++) begin
                shift_reg[i] <= 12'sd0;
            end
            out_data <= 12'sd0;
            overflow <= 1'b0;
        end else begin
            // Shift the register (new input at position 0)
            for (int i = 20; i >= 1; i--) begin
                shift_reg[i] <= shift_reg[i-1];
            end
            shift_reg[0] <= in_data;

            // Compute the FIR sum (MAC operation)
            accum <= 32'sd0;
            for (int i = 0; i < 21; i++) begin
                accum <= accum + (shift_reg[i] * coeffs[i]);
            end
        end
    end

```

```
// Scale down by 128 (arithmetic right shift)
logic signed [31:0] scaled = accum >>> 7;

// Check for overflow and saturate output
if (scaled > 2047) begin
    out_data <= 12'sd2047;
    overflow <= 1'b1;
end else if (scaled < -2048) begin
    out_data <= -12'sd2048;
    overflow <= 1'b1;
end else begin
    out_data <= scaled[11:0];
    overflow <= 1'b0;
end
end
end
end

endmodule
```

↳ FIRフィルタのテストベンチ例を作成

↳ IIRフィルタのVerilog実装