Guía Práctica 1

Diseño Digital Avanzado

Mathias Sebastian Garcia

September 18, 2024

Contents

1 Ejercicio 1

Código:

```
module sum_accumulator
#(
    // * ------------------------------------------------
    // * Parameters
    // * ------------------------------------------------
    parameter                           NB_DATA_IN  = 3 ,
    parameter                           NB_SEL      = 2 ,
    parameter                           NB_DATA_OUT = 6
)
(
    // * ------------------------------------------------
    // * Outputs
    // * ------------------------------------------------
    output logic [ NB_DATA_OUT - 1 : 0 ] o_data         ,
    output logic                         o_overflow     ,

    // * ------------------------------------------------
    // * Inputs
    // * ------------------------------------------------
    input  logic [ NB_DATA_IN  - 1 : 0 ] i_data1        ,
    input  logic [ NB_DATA_IN  - 1 : 0 ] i_data2        ,
    input  logic [ NB_SEL      - 1 : 0 ] i_sel          ,

    // * ------------------------------------------------
    // * Clock and reset
    // * ------------------------------------------------
    input  logic                         i_rst_n        ,
    input  logic                         i_clk
) ;

    // * ------------------------------------------------
    // * Internal logics
    // * ------------------------------------------------
    logic        [ NB_DATA_IN      : 0 ] expanded_data1  ;
    logic        [ NB_DATA_IN      : 0 ] expanded_data2  ;
    logic        [ NB_DATA_IN      : 0 ] in_data_sum     ;
    logic        [ NB_DATA_IN      : 0 ] data_sel        ;
    logic        [ NB_DATA_OUT     : 0 ] sum             ;
    logic        [ NB_DATA_OUT - 1 : 0 ] sum_d           ;
    logic                                overflow        ;

    // * ------------------------------------------------
    // * Expansion and sum
    // * ------------------------------------------------
    assign expanded_data1 = {1'b0, i_data1}                ;
    assign expanded_data2 = {1'b0, i_data2}                ;
    assign in_data_sum    = expanded_data1 + expanded_data2 ;

    // * ------------------------------------------------
    // * Data selection
    // * ------------------------------------------------
    typedef enum logic [NB_SEL-1:0] {
        SELECT_DATA_1   = 2'b10 ,
        SELECT_DATA_2   = 2'b00 ,
        SELECT_DATA_SUM = 2'b01
    } selector_e;

    always_comb
    begin : proc_muxing
        unique case (i_sel)
            SELECT_DATA_1   : data_sel = expanded_data1 ;
            SELECT_DATA_2   : data_sel = expanded_data2 ;
            SELECT_DATA_SUM : data_sel = in_data_sum    ;
        endcase
    end

    // * ------------------------------------------------
    // * Accumulation
    // * ------------------------------------------------
    assign sum = {1'b0, sum_d} + {{NB_DATA_OUT-NB_DATA_IN{1'b0}}, data_sel};

    always_ff @(posedge i_clk or negedge i_rst_n)
    begin : proc_accum
        if (~i_rst_n) begin
            sum_d    <= '0              ;
            overflow <= '0              ;
```

```
        end else begin
            sum_d    <= sum              ;
            overflow <= sum[NB_DATA_OUT] ;
        end
    end

    // * -----------------------------------------------
    // * Output assignment
    // * -----------------------------------------------
    assign o_data     = sum_d [ NB_DATA_OUT - 1 : 0 ] ;
    assign o_overflow = overflow                      ;

    // * -----------------------------------------------
    // * Simulation define
    // * -----------------------------------------------
    `ifdef COCOTB_SIM
        initial begin
            $dumpfile ("dump.vcd");
            $dumpvars ();
            #1;
        end
    `endif
endmodule
```

Se realizaron 7 TCs utilizando cocotb, para no sobrecargar el archivo, se encuentran en github. Los mismos consisten en:

- TC001: Entrada constante con el selector en 2
- TC002: Entrada constante con el selector en 0
- TC003: Entrada constante con el selector en 1
- TC004: Ambas entradas en 1 con el selector en 1
- TC005: Cambio en las entradas de dato con selector constante
- TC006: Cambio en todas las entradas
- TC007: Reset y recuperación

El archivo test_module.py contiene los TCs y en model.py se encuentra el modelado del RTL. Asimismo, se subió el .vcd donde se dumpearon las señales.
Teniendo en cuenta que el acumulador suma de a 2 y la salida es de 6 bits, cuyo valor de cuenta máximo es 63, se necesitan 32 clocks para que produzca overflow.

2 Ejercicio 2

Código:

```
module arith_operator
#(
    // * -----------------------------------------------------------------
    // * Parameters
    // * -----------------------------------------------------------------
    parameter                               NB_DATA        = 16 ,
    parameter                               NB_SEL         =  2
)
(
    // * -----------------------------------------------------------------
    // * Output
    // * -----------------------------------------------------------------
    output logic signed [ NB_DATA - 1 : 0 ] o_data_c            ,

    // * -----------------------------------------------------------------
    // * Inputs
    // * -----------------------------------------------------------------
    input  logic signed [ NB_DATA - 1 : 0 ] i_data_a            ,
    input  logic signed [ NB_DATA - 1 : 0 ] i_data_b            ,
    input  logic        [ NB_SEL  - 1 : 0 ] i_sel
);
    // * -----------------------------------------------------------------
    // * Vars
    // * -----------------------------------------------------------------
    logic        signed [ NB_DATA - 1 : 0 ] result              ;

    // * -----------------------------------------------------------------
    // * Muxing
    // * -----------------------------------------------------------------
    typedef enum logic [NB_SEL-1:0] {
        OP_ADDITION     = 2'b00 ,
        OP_SUBSTRACTION = 2'b01 ,
        OP_AND          = 2'b10 ,
        OP_OR           = 2'b11
    } op_options_e;

    always_comb
    begin : proc_op_selection
        case (i_sel)
            OP_ADDITION     : result = i_data_a + i_data_b ;
            OP_SUBSTRACTION : result = i_data_a - i_data_b ;
            OP_AND          : result = i_data_a & i_data_b ;
            OP_OR           : result = i_data_a | i_data_b ;
        endcase
    end

    // * -----------------------------------------------------------------
    // * Output assignment
    // * -----------------------------------------------------------------
    assign o_data_c = result;

    // * -----------------------------------------------
    // * Simulation define
    // * -----------------------------------------------
    `ifdef COCOTB_SIM
        initial begin
            $dumpfile ("dump.vcd");
            $dumpvars ();
            #1;
        end
    `endif
 endmodule
```

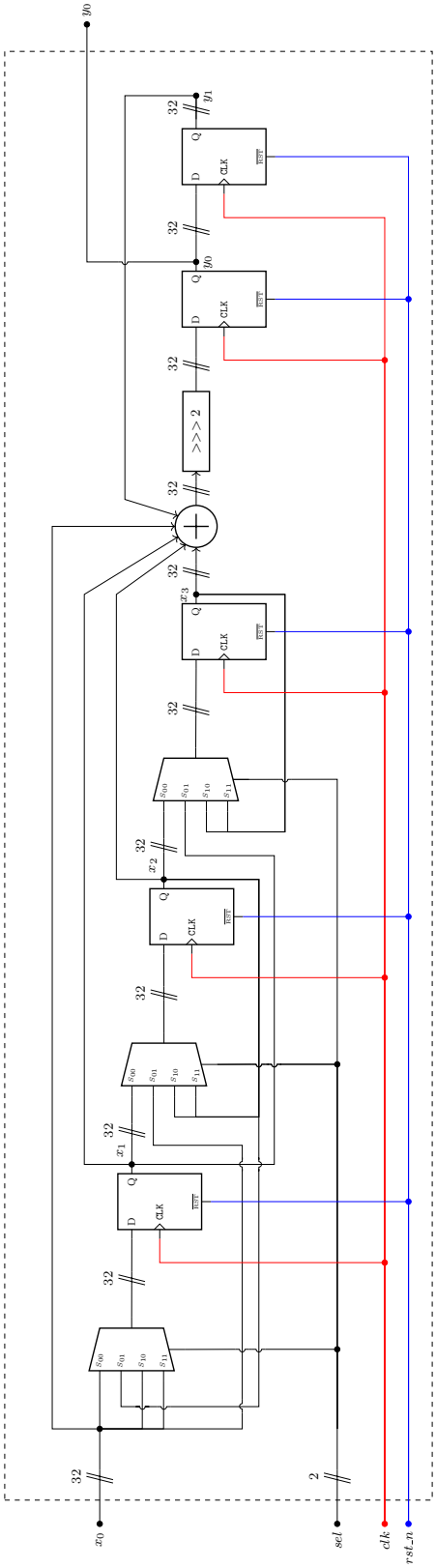
Se realizaron 5 TCs, que se pueden encontrar en este link de github, utilizando cocotb al igual que en el inciso anterior:

- TC001: Suma con datos de entrada aleatorios
- TC002: Resta con datos de entrada aleatorios
- TC003: Operación And con datos de entrada aleatorios
- TC004: Operación Or con datos de entrada aleatorios

- TC005: Randomización de operación y datos de entrada

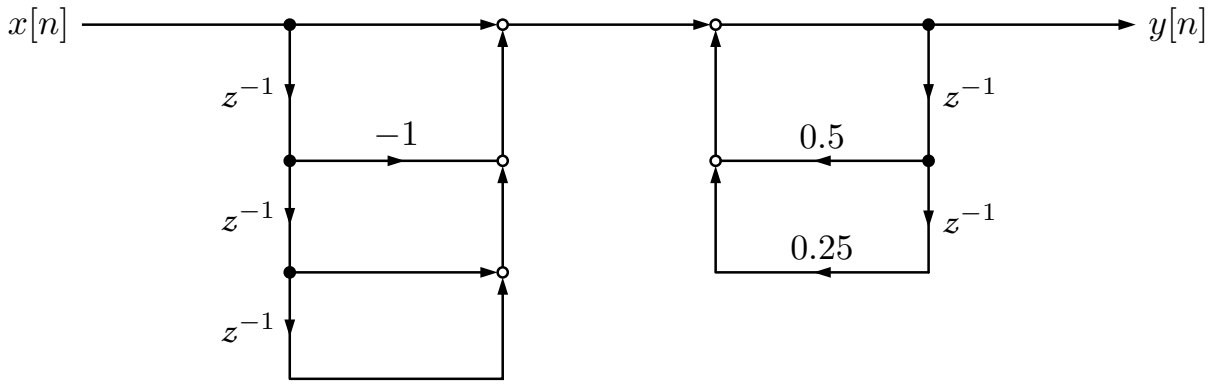
En este caso, tanto el modelo del RTL como los TCs se colocaron en el mismo archivo por simplicidad. También se subió el .vcd donde se dumpearon las señales.

3 Ejercicio 3



4 Ejercicio 4

4.1 Arquitectura



4.2 Cálculo de la cantidad de bits de salida

Se consideró que las 4 muestras en X podían tener los valores límites, es decir, -128 y 127. De donde se obtiene un mínimo de $4 \cdot (-128) = -512$ y un máximo de $4 \cdot 127 = 508$.
Para el caso de las muestras en Y se tuvieron en cuenta los extremos considerados previamente, de donde se obtuvo: $0.75 \cdot (-512) = -384$ y $0.75 \cdot 508 = 381$.
A partir de estos extremos, se calculó el rango:

$$rango = \begin{cases} Min = -512 - 384 = -896 \\ Max = 508 + 381 = 889 \\ Total = 889 - (-896) + 1 = 1786 \end{cases}$$

Para representar dicho rango se requieren 11 bits.

4.3 Testcases

Se plantearon 6 TCs, los cuales son:

- TC001: Entrada constante
- TC002: Cambio aleatorio en la entrada cada cinco clocks
- TC003: Onda senoidal de 50kHz
- TC004: Onda senoidal de 1MHz
- TC005: Onda senoidal de 10MHz
- TC006: Reset y recuperación con una onda cuadrada

Los mismos se pueden encontrar en github, estando los TCs en el archivo test_module.py y el modelo del RTL en model.py. Al igual que en los ejercicios previos, se cargó el .vcd con el dump de las señales.

4.4 Código

```
module iir_filter
#(
    // * -------------------------------------------------------------------------------------------
    // * Parameters
    // * -------------------------------------------------------------------------------------------
    parameter                                                   NB_DATA_IN       =  8 ,
    parameter                                                   NB_DATA_OUT      = 11 ,
    parameter                                                   N_INPUT_SAMPLES  =  3 ,
    parameter                                                   N_OUTPUT_SAMPLES =  2 ,
    parameter                                                   ADD_OUTPUT_PIPE  =  0
)
(
    // * -------------------------------------------------------------------------------------------
    // * Outputs
    // * -------------------------------------------------------------------------------------------
    output logic signed [ NB_DATA_OUT                           - 1 : 0 ] o_data               ,

    // * -------------------------------------------------------------------------------------------
    // * Inputs
    // * -------------------------------------------------------------------------------------------
    input  logic signed [ NB_DATA_IN                            - 1 : 0 ] i_data               ,

    // * -------------------------------------------------------------------------------------------
    // * Clock and reset
    // * -------------------------------------------------------------------------------------------
    input  logic                                                          i_reset              ,
    input  logic                                                          i_clock
) ;
    // * -------------------------------------------------------------------------------------------
    // * Internal logics
    // * -------------------------------------------------------------------------------------------
    logic  signed       [ N_INPUT_SAMPLES  - 1 : 0 ] [ NB_DATA_IN  - 1 : 0 ] x_samples             ;
    logic  signed       [ N_INPUT_SAMPLES  - 1 : 0 ] [ NB_DATA_OUT - 1 : 0 ] partial_sums_x        ;
    logic  signed       [ N_OUTPUT_SAMPLES - 1 : 0 ] [ NB_DATA_OUT - 1 : 0 ] y_samples             ;
    logic  signed       [ N_OUTPUT_SAMPLES - 1 : 0 ] [ NB_DATA_OUT - 1 : 0 ] partial_sums_y        ;
    logic  signed       [ N_OUTPUT_SAMPLES - 1 : 0 ] [ NB_DATA_OUT - 1 : 0 ] y_samples_divided     ;
    logic  signed       [ NB_DATA_OUT                              - 1 : 0 ] ed_result             ;
    logic  signed       [ NB_DATA_OUT                              - 1 : 0 ] pipe_result           ;

    // * -------------------------------------------------------------------------------------------
    // * Output divisions
    // * -------------------------------------------------------------------------------------------
    generate
        for (genvar g_sample = 0 ; g_sample < N_OUTPUT_SAMPLES ; g_sample++)
        begin : gen_assigns_divisions
            assign y_samples_divided[g_sample] = y_samples[g_sample] >>> 1;
        end
    endgenerate

    // * -------------------------------------------------------------------------------------------
    // * Result calculation
    // * -------------------------------------------------------------------------------------------
    assign partial_sums_x[0] = i_data - x_samples[0];
    generate
        for (genvar g_sum_xi = 1 ; g_sum_xi < N_INPUT_SAMPLES ; g_sum_xi++)
        begin : gen_xi_partial_sums
            assign partial_sums_x[g_sum_xi] = partial_sums_x[g_sum_xi-1] + x_samples[g_sum_xi];
        end
    endgenerate

    assign partial_sums_y[0] = y_samples_divided[0];
    generate
        for (genvar g_sum_yi = 1 ; g_sum_yi < N_OUTPUT_SAMPLES ; g_sum_yi++)
        begin : gen_yi_partial_sums
            assign partial_sums_y[g_sum_yi] = partial_sums_y[g_sum_yi-1] + y_samples_divided[g_sum_yi];
        end
    endgenerate

    assign ed_result = partial_sums_x[N_INPUT_SAMPLES-1] + partial_sums_y[N_OUTPUT_SAMPLES-1];

    // * -------------------------------------------------------------------------------------------
    // * Sampling
    // * -------------------------------------------------------------------------------------------
    always_ff @(posedge i_clock)
    begin : proc_filtering
        if (i_reset) begin
            x_samples <= '0;
            y_samples <= '0;
        end else begin
```

```
            x_samples <= {x_samples        [N_INPUT_SAMPLES -2:0], i_data   };
            y_samples <= {y_samples_divided[N_OUTPUT_SAMPLES-2:0], ed_result};
        end
    end

    // * ---------------------------------------------------------------------------------------------
    // * Optional pipe out
    // * ---------------------------------------------------------------------------------------------
    if (ADD_OUTPUT_PIPE)
    begin : gen_pipe_out
        always_ff @(i_clock)
        begin : proc_pipe_out
            pipe_result <= ed_result;
        end
    end else
    begin : gen_direct_out
        assign pipe_result = ed_result;
    end

    // * ---------------------------------------------------------------------------------------------
    // * Output assignment
    // * ---------------------------------------------------------------------------------------------
    assign o_data = pipe_result;

    // * ---------------------------------------------------------------------------------------------
    // * Simulation define
    // * ---------------------------------------------------------------------------------------------
    `ifdef COCOTB_SIM
        initial begin
            $dumpfile ("dump.vcd");
            $dumpvars ();
            #1;
        end
    `endif
endmodule
```