Guía Práctica 1

Diseño Digital Avanzado

Mathias Sebastian Garcia

19 de septiembre de 2024

Índice

1. Ejercicio 1

1.1. 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
```

1.2. Overflow

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.

1.3. Verificación

Se realizaron 7 TCs utilizando cocotb, para no sobrecargar el archivo se colocaron solo las funciones principales, las cuales son:

- TC001: Entrada constante con el selector en 2

```
async def TC001(dut):
    await init_rst_and_clk(dut)
    dut.i_data1.value = 1
    dut.i_data2.value = 2
    dut.i_sel.value   = 2

    for _ in range(200):
        await RisingEdge(dut.i_clk)
```

- TC002: Entrada constante con el selector en 0

```
async def TC002(dut):
    await init_rst_and_clk(dut)
    dut.i_data1.value = 1
    dut.i_data2.value = 2
    dut.i_sel.value   = 0

    for _ in range(100):
        await RisingEdge(dut.i_clk)
```

- TC003: Entrada constante con el selector en 1

```
async def TC003(dut):
    await init_rst_and_clk(dut)
    dut.i_data1.value = 3
    dut.i_data2.value = 2
    dut.i_sel.value   = 1

    for _ in range(100):
        await RisingEdge(dut.i_clk)
```

- TC004: Ambas entradas en 1 con el selector en 1

```
async def TC004(dut):
    await init_rst_and_clk(dut)
    dut.i_data1.value =  1
    dut.i_data2.value =  1
    dut.i_sel.value   =  1
    count             = -1

    for _ in range(200):
        await RisingEdge(dut.i_clk)
        count += 1
        if (dut.o_overflow.value == 1):
```

```
            print(f"Clocks passed until overflow rises: {count}")
            count = 0
```

- TC005: Cambio en las entradas de dato con selector constante

```
async def TC005(dut):
    await init_rst_and_clk(dut)
    dut.i_sel.value   = 1

    for _ in range(10):
        rnd_wait = rnd.randint(10,50)
        dut.i_data1.value = rnd.randint(0,2**(int(dut.NB_DATA_IN.value))-1)
        dut.i_data2.value = rnd.randint(0,2**(int(dut.NB_DATA_IN.value))-1)
        for _ in range (rnd_wait):
            await RisingEdge(dut.i_clk)
```

- TC006: Cambio en todas las entradas

```
async def TC006(dut):
    await init_rst_and_clk(dut)

    for _ in range(10):
        rnd_wait = rnd.randint(10,50)
        dut.i_sel.value   = rnd.randint(0,2)
        dut.i_data1.value = rnd.randint(0,2**(int(dut.NB_DATA_IN.value))-1)
        dut.i_data2.value = rnd.randint(0,2**(int(dut.NB_DATA_IN.value))-1)
        for _ in range (rnd_wait):
            await RisingEdge(dut.i_clk)
```

- TC007: Reset y recuperación

```
async def TC007(dut):
    await init_rst_and_clk(dut)

    for _ in range(10):
        dut.i_data1.value = rnd.randint(0,2**(int(dut.NB_DATA_IN.value))-1)
        dut.i_data2.value = rnd.randint(0,2**(int(dut.NB_DATA_IN.value))-1)
        dut.i_sel.value   = rnd.randint(0,2)

        dut.i_rst_n.value = 0
        await Timer(rnd.randint(150,500), units='ns')

        dut.i_rst_n.value = 1
        await Timer(rnd.randint(150,500), units='ns')
```

La función común init_rst_and_clk(dut) es:

```
async def init_rst_and_clk(dut):
    cocotb.start_soon(Clock(dut.i_clk, 10, units="ns").start())
    tester = RtlModel(dut)

    dut.i_rst_n.value = 1
    dut.i_data1.value = 0
    dut.i_data2.value = 0
    dut.i_sel.value   = 0
    await Timer(10, units='ns')
    dut.i_rst_n.value = 0
    await Timer(35, units='ns')
    dut.i_rst_n.value = 1
    tester.start()
```

Y, por último, las funciones básicas para realizar los chequeos necesarios:

```
def beh_model(self, in1: int, in2: int, sel: int) -> List[LogicArray]:
    if (sel == 0):
        self.sum += int(in2)
    elif (sel == 1):
        self.sum += int(in1)+int(in2)
    elif (sel == 2):
        self.sum += int(in1)

    max_sum_value = 2**int(self.dut.NB_DATA_OUT.value)
    oflow = self.sum >= max_sum_value
    if (oflow):
        self.sum -= max_sum_value

    returning_sum   = LogicArray.from_signed(self.sum, Range(int(self.dut.NB_DATA_OUT.value)+1, "downto", 0))
    returning_oflow = LogicArray('1' if oflow else '0', Range(0, "downto", 0))

    return [returning_sum, returning_oflow]
```

```python
async def _check(self) -> None:
    self.delayed_inputs = []

    while True:
        if (self.dut.i_rst_n.value == 0):
            await Timer(1, units='ns') # NOTE: Improves run time
            assert self.dut.o_data.value     == 0
            assert self.dut.o_overflow.value == 0
            self.ca_reset()
            continue

        await RisingEdge(self.dut.i_clk)
        await RisingEdge(self.dut.i_clk)

        outputs = await self.output_monitor.values.get()
        inputs  = self.input_monitor.values.get_nowait()

        self.delayed_inputs.append(inputs)

        if (len(self.delayed_inputs) > 1):
            self.delayed_input = self.delayed_inputs.pop(0)
            expected = self.beh_model(in1 = self.delayed_input["in1"], in2 = self.delayed_input["in2"], sel = self.delayed_input["sel"])
            # print(f'Expected: {expected[0]} | Got: {outputs["data"]}')
            assert outputs["data"]     == expected[0][int(self.dut.NB_DATA_OUT.value)-1:0]
            assert outputs["overflow"] == expected[1]
```

Los pythons completos, asi como el makefile y un VCD con el dump de señales se puede encontrar en este link de github. Específicamente los archivos test_module.py contiene los TCs y en model.py se encuentra el modelado del RTL.

2. Ejercicio 2

2.1. 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
```

2.2. Verificación

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

```
async def TC001(dut):
    await init_test(dut)
    dut.i_sel.value    = 0
    for _ in range(100):
        dut.i_data_a.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        dut.i_data_b.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        await Timer(rnd.randint(10,50), 'ns')
```

- TC002: Resta con datos de entrada aleatorios

```
async def TC002(dut):
    await init_test(dut)
    dut.i_sel.value    = 1
    for _ in range(100):
        dut.i_data_a.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        dut.i_data_b.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        await Timer(rnd.randint(10,50), 'ns')
```

- TC003: Operación And con datos de entrada aleatorios

```
async def TC003(dut):
    await init_test(dut)
    dut.i_sel.value    = 2
    for _ in range(100):
        dut.i_data_a.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        dut.i_data_b.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        await Timer(rnd.randint(10,50), 'ns')
```

- TC004: Operación Or con datos de entrada aleatorios

```
async def TC004(dut):
    await init_test(dut)
    dut.i_sel.value    = 3
    for _ in range(100):
        dut.i_data_a.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        dut.i_data_b.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        await Timer(rnd.randint(10,50), 'ns')
```

- TC005: Randomización de operación y datos de entrada

```
async def TC005(dut):
    await init_test(dut)
    for _ in range(100):
        dut.i_sel.value    = rnd.randint(0, 2**int(dut.NB_SEL.value)-1)
        dut.i_data_a.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        dut.i_data_b.value = rnd.randint(-2**(int(dut.NB_DATA.value)-1),2**(int(dut.NB_DATA.value)-1)-1)
        await Timer(rnd.randint(10,50), 'ns')
```

La función common para inicializar el test es:

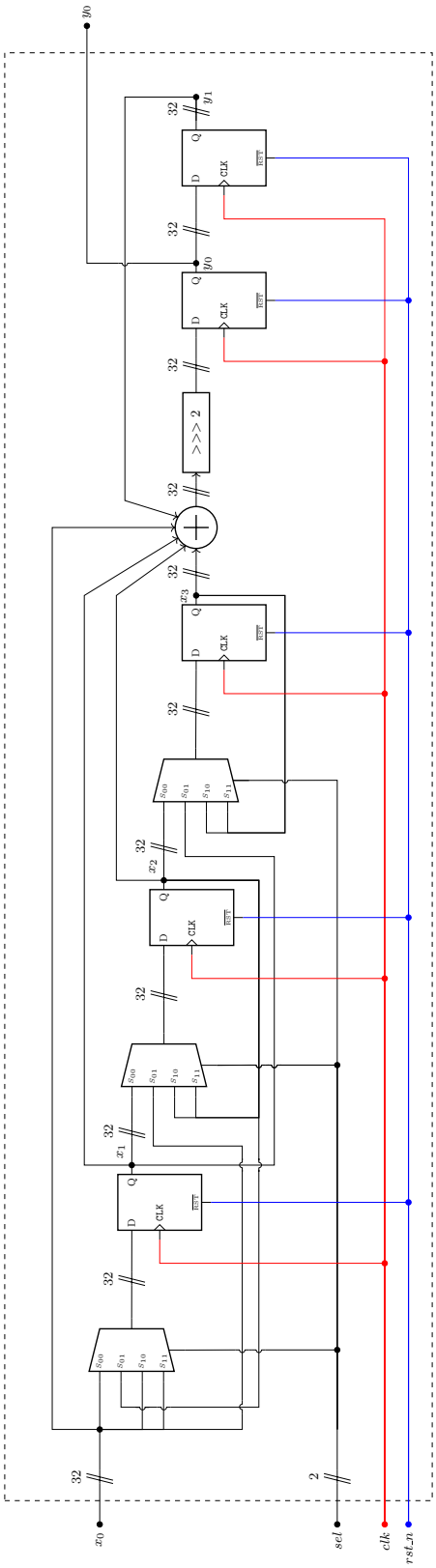
```
async def init_test(dut):
    tester = RtlModel(dut)

    dut.i_sel.value    = 0
    dut.i_data_a.value = 0
    dut.i_data_b.value = 0
    await Timer(rnd.randint(10,50), 'ns')
    tester.start()
```

La función para modelar el RTL es:

```
async def run_beh(self, dut) -> None:
    while (True):
        await Timer(1, 'ns')
        obtained_data = dut.o_data_c.value
        if   (self.dut.i_sel.value == LogicArray('00', Range(int(dut.NB_SEL.value)-1, 'downto', 0))):
            assert obtained_data == LogicArray.from_signed(int(dut.i_data_a.value) + int(dut.i_data_b.value), Range(int(self.dut.NB_DATA.value)+1, "downto", 0))[int(self.dut.NB_DATA.value)-1:0]
        elif (self.dut.i_sel.value == LogicArray('01', Range(int(dut.NB_SEL.value)-1, 'downto', 0))):
            assert obtained_data == LogicArray.from_signed(int(dut.i_data_a.value) - int(dut.i_data_b.value), Range(int(self.dut.NB_DATA.value)+1, "downto", 0))[int(self.dut.NB_DATA.value)-1:0]
        elif (self.dut.i_sel.value == LogicArray('10', Range(int(dut.NB_SEL.value)-1, 'downto', 0))):
            assert dut.o_data_c.value == dut.i_data_a.value & dut.i_data_b.value
        else:
            assert obtained_data == dut.i_data_a.value or dut.i_data_b.value
```

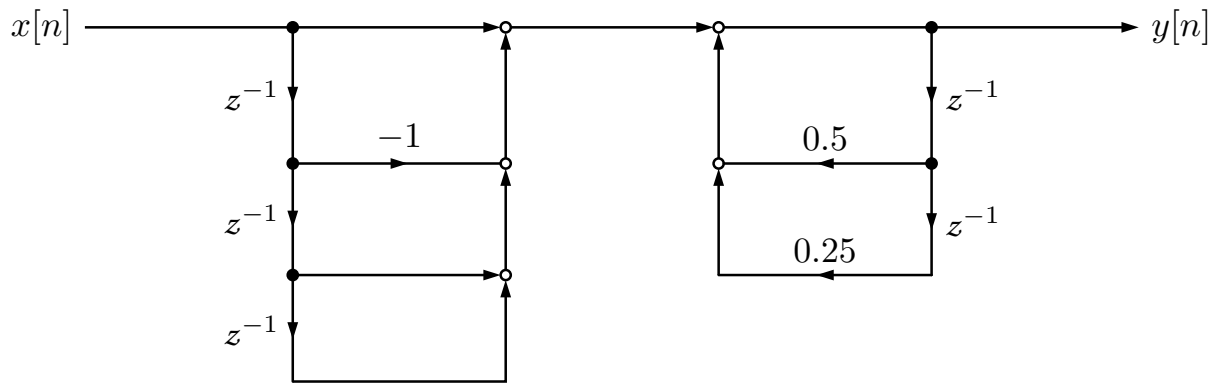
En este caso, tanto el modelo del RTL como los TCs se colocaron en el mismo archivo por simplicidad. También se subió el .vcd donde se dumpearon las señales.

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. 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
```

4.4. Verificación

Se plantearon 6 TCs, los cuales son:

- TC001: Entrada constante

```
async def TC001(dut):
    await init_rtl(dut)
    dut.i_data.value = -128

    for _ in range(200):
        await RisingEdge(dut.i_clock)
```

- TC002: Cambio aleatorio en la entrada cada cinco clocks

```
async def TC002(dut):
    await init_rtl(dut)

    for _ in range(rnd.randint(5,20)):
        await RisingEdge(dut.i_clock)

    for _ in range(500):
        dut.i_data.value = rnd.randint(-100,100)
        for _ in range(5):
            await RisingEdge(dut.i_clock)
```

- TC003: Onda senoidal de 50kHz

```
async def TC003(dut):
    await init_rtl(dut)
    await run_sinusoidal_wave(dut = dut, waiting_clocks = 10000, freq_in_hz = 50000, amplitude = 20)
```

- TC004: Onda senoidal de 1MHz

```
async def TC004(dut):
    await init_rtl(dut)
    await run_sinusoidal_wave(dut = dut, waiting_clocks = 1000, freq_in_hz = 1000000, amplitude = 20)
```

- TC005: Onda senoidal de 10MHz

```
async def TC005(dut):
    await init_rtl(dut)
    await run_sinusoidal_wave(dut = dut, waiting_clocks = 1000, freq_in_hz = 10000000, amplitude = 20)
```

- TC006: Reset y recuperación con una onda cuadrada

```
async def TC006(dut):
    await init_rtl(dut)

    timespace     = np.linspace(0, 10000*10e-9, 10000, endpoint=False)
    square_signal = signal.square(2*np.pi*1e6*timespace, duty = 0.5)
    reset_times   = 3

    for value in square_signal:
        await RisingEdge(dut.i_clock)
        dut.i_data.value = 20*int(value)
        if (rnd.randint(0,1200) == 0) and (reset_times > 0):
            await reset_rtl(dut)
            reset_times -= 1
```

Las funciones common para los tests fueron:

```
async def init_rtl(dut):
    cocotb.start_soon(Clock(dut.i_clock, 10, units="ns").start())
    tester = RtlModel(dut)

    dut.i_reset.value = 0
    dut.i_data.value  = 0
    await RisingEdge(dut.i_clock)
    await RisingEdge(dut.i_clock)
    dut.i_reset.value = 1
    tester.start()
    await RisingEdge(dut.i_clock)
    await RisingEdge(dut.i_clock)
    dut.i_reset.value = 0
    await RisingEdge(dut.i_clock)

async def reset_rtl(dut):
    await RisingEdge(dut.i_clock)
    dut.i_reset.value = 0
    await RisingEdge(dut.i_clock)
    dut.i_reset.value = 1
    await RisingEdge(dut.i_clock)
```

```
    dut.i_reset.value = 0

async def run_sinusoidal_wave(dut: SimHandleBase, waiting_clocks: int, freq_in_hz: int, amplitude: int):
    sampling_freq = 1/(10e-9)
    for i in range(waiting_clocks):
        await RisingEdge(dut.i_clock)
        dut.i_data.value = int(amplitude*np.sin(2*np.pi*i*(freq_in_hz/sampling_freq)))
```

La principal función de verificación, la cual se encuentra dentro del objeto RtlModel, fue:

```
    async def _check(self) -> None:
        while True:
            await RisingEdge(self.dut.i_clock)
            if (int(self.dut.i_reset.value) == 1):
                self.clear_monitors()
                continue

            result  = self.input_monitor.get_n_sample(0) .to_signed()
            result -= self.input_monitor.get_n_sample(1) .to_signed()
            result += self.input_monitor.get_n_sample(2) .to_signed()
            result += self.input_monitor.get_n_sample(3) .to_signed()
            result += self.output_monitor.get_n_sample(2).to_signed()//4
            result += self.output_monitor.get_n_sample(1).to_signed()//2

            # print(f'Result: {result} | Got: {self.output_monitor.get_n_sample(0).to_signed()} ')
            assert result == self.output_monitor.get_n_sample(0).to_signed()

            _ = self.output_monitor.values.get_nowait()
            _ = self.input_monitor.values.get_nowait()
```

El entorno completo se puede encontrar en este link de github, estando los TCs en el archivo test_module.py y el modelo del RTL en model.py.