Task Unit 3: Validation of a Binary to BCD Converter

José Manuel Caballero Sánchez — Master in Microelectronics, University of Seville

1. Objective

The goal of this assignment is to validate the correct operation of a parameterized Binary to BCD converter written in Verilog (`N=9`) through simulation, fulfilling the following specifications:

1. Correct conversion for `N=9`.
2. The conversion continues even if the `convierte` pulse goes low before finishing.
3. `rst` is a synchronous high reset active signal that interrupts the ongoing conversion.
4. The input data `IN` can be modified without affecting the current conversion.

2. Validation through Simulation

The description provided by avedillo@us.es was the following:

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 06.11.2018 13:22:39
// Design Name:
// Module Name: unit3_1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////

module unit3_1  (clk, rst, convierte, IN, Rcentenas, Rdecenas,
   Runidades, fin, listo);

  parameter N= 9;
  input clk, rst, convierte;
```

```
input [N-1:0] IN;
output reg [3:0] Rcentenas, Rdecenas, Runidades;
output fin, listo;


function integer F1;
  input [31:0] num;
  integer i;

  begin
    i = num;
    for(F1 = 0; i > 0; F1 = F1 + 1)
      i = i >> 1;
  end

endfunction

reg[2:0] ;
reg [N-1:0] RIN;
parameter espera=3'b000, prepara=3'b001, opera1=3'b010, opera2=3'
   b011, ultimo=3'b100;
reg [2:0] state;
assign listo = (state==espera);
assign fin =(state==ultimo);

always @(posedge clk)

if (rst)
  state<= espera;

else
  case (state)

    espera:

      begin if (convierte == 1)
              state <= prepara;
            else
              state <= espera;
      end

    prepara:

      begin
        Rcentenas <= 0;
        Rdecenas  <= 0;
        Runidades <= 0;
        Contador  <= N-1;
        RIN       <= IN;
        state     <= opera1;
      end
```

```
      opera1:

        begin
          if (Rdecenas >= 5)
            Rdecenas <= Rdecenas +3;

          if (Runidades >=5)
            Runidades <= Runidades + 3;

          if (Rcentenas >=5)
            Rcentenas <= Rcentenas + 3;
            state <= opera2;
        end

      opera2:

        begin
          Contador <= Contador -1;
          Rcentenas <= {Rcentenas[2:0], Rdecenas[3]};
          Rdecenas <= {Rdecenas[2:0], Runidades[3]};
          Runidades <= {Runidades[2:0], RIN[N-1]};
          RIN <= RIN << 1;

          if (Contador == 0)
            state <= ultimo;
          else
            state <= opera1;
        end

      ultimo:

        state <= espera;

      default: state<= espera;

    endcase

endmodule
```

Although by inspection we can already identify some issues in the code regarding the required specifications, a dedicated testbench has been created to validate the correct functionality of the design.

```verilog
// ====================== TESTBENCH ======================
`timescale 1ns/1ps
module tb;
  parameter N = 9;

  reg                clk = 0;
  reg                rst = 0;
  reg                convierte = 0;
  reg   [N-1:0]      IN = 0;
  wire  [3:0]        Rcentenas, Rdecenas, Runidades;
  wire               fin, listo;

  unit3_1_fixed #(.N(N)) dut (
    .clk(clk), .rst(rst), .convierte(convierte), .IN(IN),
    .Rcentenas(Rcentenas), .Rdecenas(Rdecenas), .Runidades(Runidades)
      ,
    .fin(fin), .listo(listo)
  );

  // Reloj (100 MHz)
  always #5 clk = ~clk;

  initial begin
    $dumpfile("wave.vcd");
    $dumpvars(0, tb);
  end

  initial begin
    // Reset sincrono
    rst = 1;
    repeat (3) @(posedge clk);
    rst = 0;

    //Requisito 1 y 2 => Prueba 1 y 2
    // PRUEBA 1: IN = 19
    IN = 19;
    convierte = 1;
    @(posedge clk);
    convierte = 0;

    wait (fin == 1);
    @(posedge clk);
    $display("IN=19  -> C:%0d D:%0d U:%0d", Rcentenas, Rdecenas,
       Runidades);

    // PRUEBA 2: IN = 511
    IN = 511;
    convierte = 1;
    @(posedge clk);
    convierte = 0;

```

```
    wait (fin == 1);
    @(posedge clk);
    $display("IN=511 -> C:%0d D:%0d U:%0d", Rcentenas, Rdecenas,
       Runidades);

    // Requisito 3
    // PRUEBA 3: 511 Con rst interrumpiendo
    IN = 511;
    convierte = 1;
    @(posedge clk);
    convierte = 0;
    repeat (5) @(posedge clk);
    rst = 1;
    @(posedge clk);
    rst = 0;
    @(posedge clk);
    $display("IN =511 tras reset en medio: C=%0d D=%0d U=%0d",
       Rcentenas, Rdecenas, Runidades);

    // Resuisito 4
    // PRUEBA 4: IN = 19 y luego IN =511
    IN = 19;
    convierte = 1;
    @(posedge clk);
    convierte = 0;
    repeat (2) @(posedge clk);

    IN = 511;
    convierte = 1;
    @(posedge clk);
    convierte = 0;
    wait (fin==1);
    $display("IN = 19 y cambio IN=511 -> C:%0d D:%0d U:%0d",
       Rcentenas, Rdecenas, Runidades);

    $finish;
  end
endmodule
```

The obtained result in https://www.edaplayground.com/ was the following:

```
VCD info: dumpfile wave.vcd opened for output.
IN=19 -> C:0 D:0 U:0
IN=511 -> C:0 D:0 U:1
IN =511 after mid-reset: C=0 D=0 U:1
IN = 19 and change IN=511 -> C:0 D:0 U:0
testbench.sv:82: $finish called at 235000 (1ps)
```

After running the tests, several observations can be made:

1. The converter does not correctly convert either 19 or 511. Both yield incorrect

results (000 and 001), though different — allowing further validation. → Does **not** meet requirement 1.

2. Although the conversion result for 511 is wrong, we can see that when the `convierte` signal is deactivated, the conversion continues and outputs 001. → Meets requirement 2.

3. In Test 3, activating `rst` during an ongoing conversion does not reset the process. It should be noted that, with the current design, a "reset" could seem to occur if the reset occurs too early in the sequence. For this reason, 5 clock cycles were repeated before activating `rst`, ensuring that the system was in an advanced conversion state (`opera1` or `opera2`). Activating `rst` here shows that it does not interrupt the process. If fewer cycles were used (e.g., 2), the output could appear reset but only because the FSM would return to `espera` before starting conversion — inheriting zeros from `prepara`, not due to an actual reset. → Does **not** meet requirement 3.

4. Changing `IN` from 19 to 511 during Test 4 does not affect the ongoing conversion, which still prints the original value of 19 (000). → Meets requirement 4.

3. Correct Description with All Specifications

The corrected design is as follows:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 06.11.2018 13:22:39
// Design Name:
// Module Name: unit3_1_fixed
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////

module unit3_1_fixed (clk, rst, convierte, IN, Rcentenas, Rdecenas,
   Runidades, fin, listo);

  parameter N = 9;

  // Ports
```

```verilog
input             clk;
input             rst;
input             convierte;
input      [N-1:0] IN;

output reg [3:0]  Rcentenas;
output reg [3:0]  Rdecenas;
output reg [3:0]  Runidades;
output            fin;
output            listo;

// Auxiliar function
function integer F1;
  input [31:0] num;
  integer i;
  begin
    i = num;
    for(F1 = 0; i > 0; F1 = F1 + 1)
      i = i >> 1;
  end
endfunction

// Inner registers
reg [F1(N)-1:0] Contador;
reg [N-1:0]     RIN;
reg [2:0]       state;

initial begin
state      = 3'b000;
Contador   = 0;
RIN        = 0;
Rcentenas  = 0;
Rdecenas   = 0;
Runidades  = 0;
end

// States
parameter espera  = 3'b000,
          prepara = 3'b001,
          opera1  = 3'b010,
          opera2  = 3'b011,
          ultimo  = 3'b100;

// Status signals
assign listo = (state == espera);
assign fin   = (state == ultimo);

// Main
always @(posedge clk) begin

  if (rst) begin
```

```
      state     <= espera;
      Rcentenas <= 0;
      Rdecenas  <= 0;
      Runidades <= 0;
      Contador  <= 0;
      RIN       <= 0;


    end else begin
      case (state)

        espera: begin

          if (convierte == 1)
            state <= prepara;
          else
            state <= espera;
        end

        prepara: begin
          Rcentenas <= 0;
          Rdecenas  <= 0;
          Runidades <= 0;
          RIN       <= IN;
          Contador  <= N-1;
          state     <= opera1;
        end

        opera1: begin
          if (Rdecenas  >= 5) Rdecenas  <= Rdecenas  + 3;
          if (Runidades >= 5) Runidades <= Runidades + 3;
          if (Rcentenas >= 5) Rcentenas <= Rcentenas + 3;
          state <= opera2;
        end

        opera2: begin
          Contador  <= Contador - 1;
          Rcentenas <= {Rcentenas[2:0], Rdecenas[3]};
          Rdecenas  <= {Rdecenas [2:0], Runidades[3]};
          Runidades <= {Runidades[2:0], RIN[N-1]};
          RIN       <= RIN << 1;

          if (Contador == 0)
            state <= ultimo;
          else
            state <= opera1;
        end

        ultimo: begin
          state <= espera;
        end
```

```
      default: begin
        state <= espera;
      end

    endcase
  end

end

endmodule
```

- First, an **adaptation and standardization of the original code** was performed to improve readability and adhere to Verilog HDL best coding practices.
- The module structure was reorganized, clearly separating *port declarations*, *parameters*, *internal signals*, *auxiliary functions*, and the *main sequential block*.
- A consistent format was applied for indentation, spacing, and comments, improving clarity and maintainability.
- Explanatory comments were added, and assignments were rewritten in an ordered and uniform way.
- An `initial` block was added to ensure all signals start from a defined value at the start of the simulation, avoiding undefined (`X`) states.

With all these improvements, a more coherent and readable version of the original design — named **`unit3_1_fixed`** — was obtained. In addition, several functional corrections were introduced:

1. **Dynamic counter width:** The original version used a fixed 3-bit counter, which caused overflow errors. The new version uses the F1 function to automatically calculate the required bit width (`[F1(N)-1:0]`), ensuring the correct behavior for any `N`.
2. **Controlled register initialization:** An `initial` block sets all registers to zero at startup, preventing undefined states during the first cycles.
3. **Signal cleanup during the waiting state:** In the original design, after returning to `espera` (the waiting state) following a reset or the end of a conversion, some output registers retained residual values from the previous operation. In the updated version, these signals are cleared **when `rst` is activated** and reinitialized in `prepara` at the start of each new conversion, ensuring consistent outputs before the next cycle begins.
4. **Improved synchronous reset:** The `rst` signal now performs a complete synchronous reset of the FSM and internal registers, **forcing a return to `espera` upon activation** with outputs and counters set to zero. This guarantees that every new conversion starts from a fully defined and stable condition.

5. **Main logical behavior preserved:** The *double dabble* conversion algorithm in the `opera1` and `opera2` states was kept identical to maintain functional equivalence.

6. **Minor style and consistency corrections:** Conditions and assignments were compacted when convenient for better readability while preserving semantics.

With these modifications, the new design **successfully meets all the required specifications**. In particular, improvements 1, 3, and 4 fix the main issues — the first solves incorrect conversion (requirement 1), while 3 and 4 address the lack of proper reset handling (requirement 3).

4. Results

For the improved design, the same **testbench** was used, and the result obtained in https://www.edaplayground.com/ was:

```
VCD info: dumpfile wave.vcd opened for output.
IN=19 -> C:0 D:1 U:9
IN=511 -> C:5 D:1 U:1
IN =511 after mid-reset: C=0 D=0 U=0
IN = 19 and change IN=511 -> C:0 D:1 U:9
testbench.sv:82: $finish called at 665000 (1ps)
```

We can observe that the conversion is now correct for both 19 and 511, even when `convierte` is deactivated; the reset correctly clears all registers, and changing `IN` during an active conversion does not affect the result, as expected.

Finally, the generated waveform file confirms the correct evolution of the states and output signals:

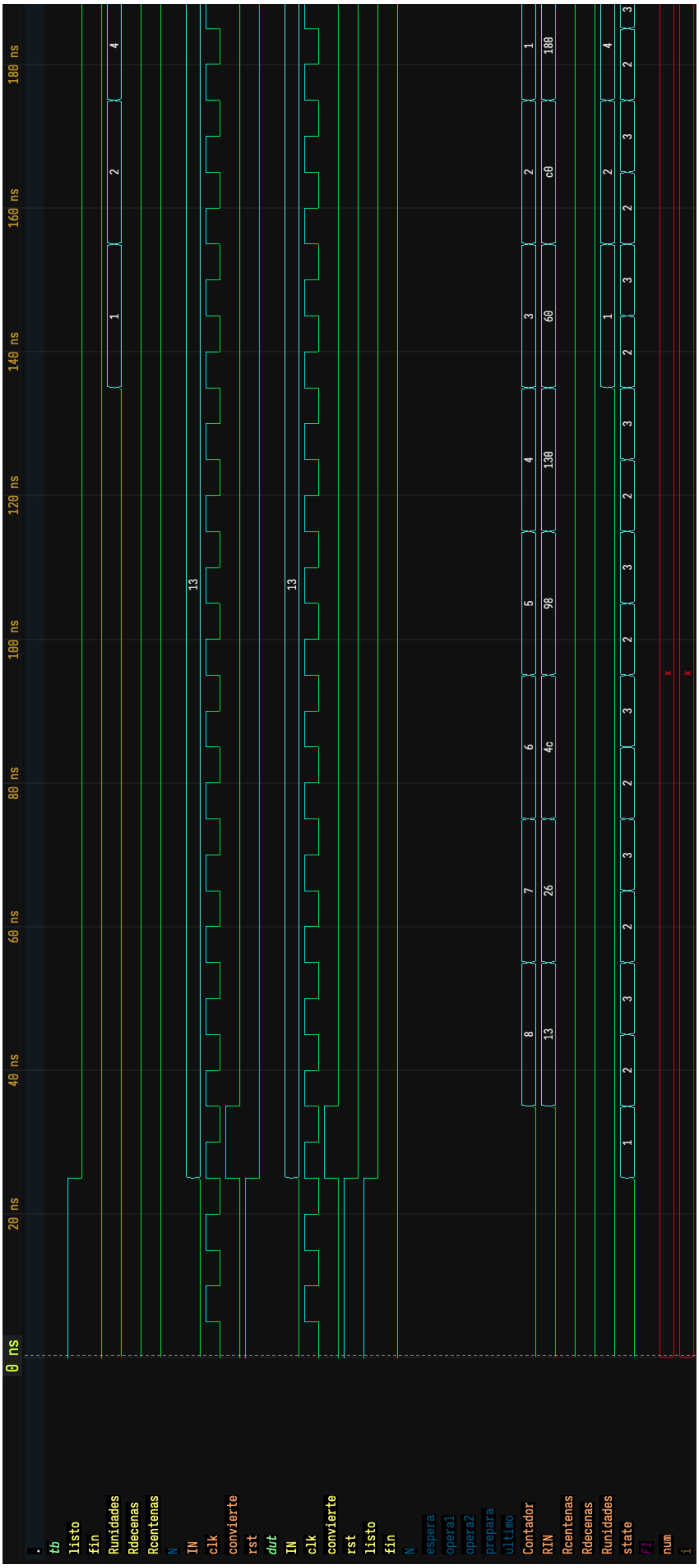

Figure 1: Waveform file wave.vcd generated after simulation