

PRÁCTICA 1: COMUNICACIÓN SERIE CON UN PC. DISEÑO DEL RECEPTOR



Alumnos: Pablo Menéndez Ruiz de Azúa y Carles Olucha Royo

Fecha de realización: 27/01/2020

Fecha de entrega: 10/02/2021

Asignatura: Sistemas digitales I

Profesor: Pedro Olmos González

Índice

| | |
|--|----|
| Objetivos | 3 |
| Introducción | 3 |
| Diagrama de bloques | 3 |
| Componentes | 4 |
| Registro serie-paralelo | 4 |
| Detector de errores de paridad | 6 |
| Contador de 1 bit | 8 |
| Unidad de Control | 9 |
| Circuito completo | 13 |
| Implantación física | 17 |
| Conclusiones y análisis de los resultados..... | 19 |

Objetivos

El objetivo de esta práctica es el diseño de un receptor serie RS-232 para recibir datos de un PC y mostrarlos en binario en los LEDs de la tarjeta DE1.

Introducción

Los datos que se reciben desde el PC constan de 1 bit de arranque, 8 bits de datos, un bit de paridad (paridad impar) y un bit de stop. La velocidad de transmisión será de 19200 baudios. Como en la transmisión serie RS-232 no se transmite el reloj, es necesario sincronizar el reloj del receptor en cada byte, para lo que se utiliza el bit de arranque. En la figura 1 se muestra un diagrama de tiempos de la transmisión, en el que se muestran los instantes en los que es deseable muestrear la línea de datos. Como se puede apreciar, para disminuir los efectos de la asincronía entre los relojes del emisor y el receptor, los datos han de muestrearse en la mitad del tiempo de transmisión de cada bit, tal como se muestra en la figura.

El funcionamiento del circuito será el siguiente: cuando se detecte el flanco de bajada de la señal de datos, lo cual indica el comienzo de la trama, se esperará el tiempo de medio bit y se verificará que la señal de datos sigue a cero. A continuación, se esperará el tiempo de un bit y se muestrearán el estado de la señal de entrada, introduciendo su valor en un registro de desplazamiento. Este proceso se repetirá ocho veces para obtener en la salida paralelo del registro de desplazamiento el dato enviado en la trama. A continuación, se leerá el bit de paridad (después de esperar el tiempo de un bit) y por último el bit de stop. Si se produce un error de paridad se iluminará un LED para indicarlo. Por otro lado, si el bit de start no es cero o el bit de stop no es uno, se iluminará un LED para indicar que se ha producido un error de trama (framing error). Ambos LED permanecerán encendidos hasta que se reciba el siguiente bit de start.



Figura 1 – Diagrama de tiempos de la señal transmitida

Diagrama de bloques

Antes de empezar a programar cada uno de los componentes necesarios para diseñar nuestro receptor, vamos a hacer un diagrama de bloques con cada componente del circuito.

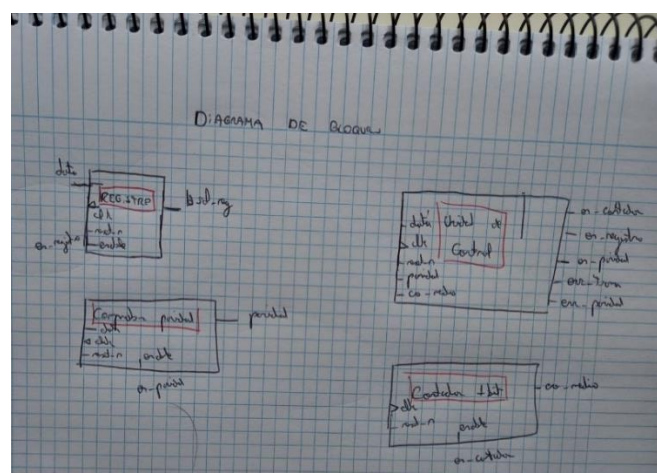


Figura 2 – Diagrama de bloques del circuito

En esta imagen, podemos ver los 4 bloques principales de nuestro receptor: el registro serie-paralelo, el comprobador de paridad, la unidad de control y el contador de un bit que nos va a permitir controlar los tiempos.

Componentes

Los componentes que hemos diseñado para esta práctica han sido los siguientes: Un registro serie-paralelo de 8 bits que utilizaremos para ir registrando el dato que nos pasa el transmisor, un comprobador de paridad para ver si hay errores de paridad, un contador de un bit para generar un pulso cuando pase el tiempo correspondiente a medio bit, un contador de módulo 9 para que se repita la acción de registrar y esperar un bit, la cual se tiene que repetir 8 veces, un detector de flanco de bajada para saber que se empiezan a transmitir los bits, un registro paralelo que nos permite copiar los bits que van saliendo y la unidad de control que va a ser la que controle en qué momento ocurre cada cosa.

De todos estos componentes, únicamente vamos a explicar y a enseñar en el informe el código del registro serie-paralelo, el comprobador de paridad, el contador de un bit y la unidad de control. Hemos elegido estos porque nos parecen los más importantes y necesarios para explicar. El resto de los componentes consideramos que no son muy complejos por lo que no vemos necesario enseñar y explicar su código. De todas formas, si quieren ver de todos los componentes utilizados para esta práctica y los testbenches utilizados para las simulaciones, se adjuntan junto a este documento.

Registro serie-paralelo

En primer lugar, es necesario realizar un registro de desplazamiento de 8 bits para almacenar el dato recibido. Ha de tenerse en cuenta que dicho registro ha de ser síncrono, es decir, su reloj ha de ser el mismo que el de los demás componentes de la FPGA (50 MHz). Por tanto, ha de diseñarse el registro de manera que solo realice un desplazamiento mientras este activa una señal de control (enable).

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Registro is
5      port(
6          clk      : in std_logic;
7          enable    : in std_logic;
8          reset_n   : in std_logic;
9          data      : in std_logic;
10         bits      : out std_logic_vector(7 downto 0)
11     );
12 end Registro;
13
14 architecture behavioral of Registro is
15     signal registro : std_logic_vector(7 downto 0);
16 begin
17     process(clk, reset_n)
18     begin
19         if reset_n = '0' then
20             registro <= (others => '0');
21         else
22             if rising_edge(clk) then
23                 if enable = '1' then
24                     registro(7) <= data;
25                     registro(6 downto 0) <= registro(7 downto 1);
26                 end if;
27             end if;
28         end if;
29     end process;
30
31     bits <= registro(7 downto 0);
32 end behavioral;
```

Figura 3 – Código VHDL del registro serie-paralelo

Una vez que hemos diseñado en VHDL el registro, vamos a simularlo para comprobar que funciona correctamente. Para ello, nos creamos primero un testbench y lo simulamos obteniendo el siguiente resultado. Todo el código VHDL y los testbenches utilizados para realizar esta práctica vienen adjuntos junto a este documento.

```

63 p_clk: process
64 begin
65     clk <= '0';
66     wait for clk_per / 2;
67     clk <= '1';
68     wait for clk_per / 2;
69 end process;
70
71 p_rstn : process
72 begin
73     reset_n <= '0';
74     wait for 100 ns;
75     reset_n <= '1';
76     wait;
77 end process;
78
79 p_stim : PROCESS
80 -- optional sensitivity list
81 -- (
82 -- variable declarations
83 BEGIN
84     data <= '0';
85     enable <= '0';
86
87     wait until reset_n = '1';
88
89     wait for 5 ns;
90
91     for n in 0 to 3 loop
92         data <= '1';
93         wait for clk_per;
94         enable <= '1';
95
96         wait for clk_per;
97         enable <= '0';
98         wait for 80 ns;
99     end loop;
100
101     enable <= '0';
102     wait for 80 ns;
103
104     data <= '0';
105     wait for clk_per;
106
107     enable <= '1';
108     wait for clk_per;
109
110     enable <= '0';
111     wait for 80 ns;
112
113     assert bits = "01010101"
114     report "Error: salida del registro errónea"
115     severity failure;
116
117     assert false
118     report "Fin de la simulación"
119     severity failure;
120
121     -- code executes for every event on sensitivity list
122 END PROCESS p_stim;
123 END Registro_arch;
124

```

Figura 4 – Testebench del registro

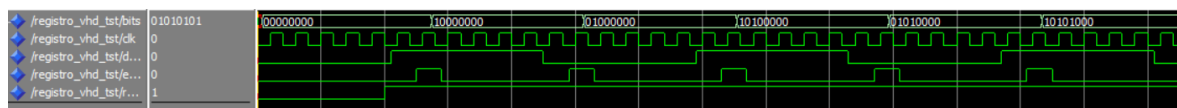


Figura 5 – Simulación del registro serie-paralelo

Detector de errores de paridad

En segundo lugar, es preciso diseñar un circuito secuencial para verificar que la transmisión ha sido correcta. En el caso de paridad impar, la suma de todos los bits, incluido el de paridad, ha de ser impar.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Comprobar_paridad is
5      port(
6          data : in std_logic;
7          paridad : out std_logic;
8          clk : in std_logic;
9          reset_n : in std_logic;
10         back_to_0 : in std_logic;
11         enable : in std_logic
12     );
13 end Comprobar_paridad;
14
15 architecture behavioral of Comprobar_paridad is
16     signal b : std_logic;
17 begin
18     process(clk,reset_n)
19     begin
20         if reset_n = '0' then
21             b <= '0';
22         elsif back_to_0 = '1' then
23             b <= '0';
24         elsif rising_edge(clk) then
25             if enable = '1' then
26                 b <= data xor b;
27             end if;
28         end if;
29     end process;
30
31     paridad <= b;
32 end behavioral;
```

Figura 6 – Código VHDL del detector de paridad

Para detectar la paridad lo único que vamos a hacer es pasar el bit de entrada por una xor que lo compara con un acumulado de xor de los bits anteriores. Por tanto, si al final después de haber pasado por todos los bits, incluido el de paridad, la función devuelve 1 es impar y si devuelve 0 es par. Además, en el código hemos añadido una señal que actúa como reset pero cuando lo necesitemos, ya que al simular el componente final de la práctica tuvimos un pequeño problema con esto y era que la variable paridad no volvía a 0, por lo que ya el siguiente número nos daba un error de paridad

Este componente también lo vamos a simular y al simularlo, obtenemos lo siguiente.

```

46 BEGIN
47     data <= '0';
48     enable <= '0';
49
50     wait until reset_n = '1';
51
52     wait for 10 ns;
53
54     for n in 0 to 3 loop
55
56         data <= '1';
57         enable <= '1';
58         wait for clk_per;
59
60         enable <= '0';
61
62         wait for 100 ns;
63
64         data <= '0';
65         enable <= '1';
66
67         wait for clk_per;
68
69         enable <= '0';
70
71         wait for 100 ns;
72     end loop;
73
74     data <= '1';
75     enable <= '1';
76     wait for clk_per;
77
78     enable <= '0';
79
80     wait for 100 ns;
81
82     assert paridad = '0'
83     report "Error: salida de paridad errónea"
84     severity failure;
85
86     assert false
87     report "Fin de la simulación"
88     severity failure;

```

Figura 7 – Testbench del comprobador de paridad

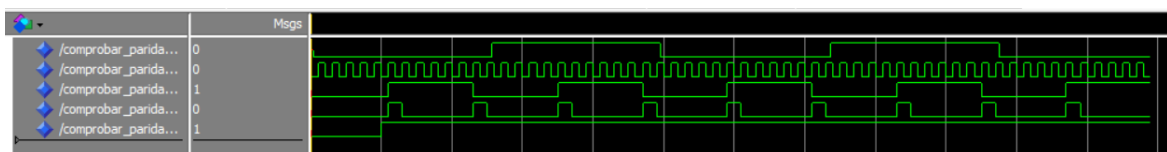


Figura 8 – Simulación detector de paridad

Como vemos, en este caso, le hemos pasado 9 bits de los cuales 5 eran unos. Por lo tanto, nos tenía que devolver al final un uno el bit de paridad como ha ocurrido.

Contador de 1 bit

Como la señal nos llega a una velocidad de 19200 baudios debemos sincronizar esta señal con nuestro reloj de 50 MHz. Para ello, debemos crearnos un contador que devuelva un pulso cuando se tenga que registrar un bit, que es en el centro de cada pulso que veíamos en la figura uno. Después de hacer los cálculos, llegamos a que un bit se manda cada 2604 pulsos de nuestro reloj de 50 Mhz. Por lo tanto, como siempre queremos registrar cuando esté en medio, vamos a hacer un contador que llegue como máximo hasta 2604 pero que saque una señal cada 1302 pulsos.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Contador1bit is
6  port(
7      reset_n : in std_logic;
8      back_to_0 : in std_logic;
9      clk      : in std_logic;
10     en       : in std_logic;
11     co_medio  : out std_logic
12 );
13 end Contador1bit;
14
15 architecture behavioral of Contador1bit is
16     signal contador : unsigned(11 downto 0);
17 begin
18     process(reset_n,clk)
19     begin
20         if reset_n = '0' then
21             contador <= (others => '0');
22         elsif back_to_0 = '1' then
23             contador <= (others => '0');
24         else
25             if rising_edge(clk) then
26                 if en='1' then
27                     if contador= 2604 then
28                         contador <= (others => '0');
29                     else
30                         contador <= contador + 1;
31                     end if;
32                 end if;
33             end if;
34         end if;
35     end process;
36     co_medio <= '1' when contador = 1302 and en = '1' else '0';
37 end behavioral;
```

Figura 9 – Código VHDL del contador de 1 bit (2604 pulsos de reloj)

Como vemos, en este código también le hemos añadido la señal back_to_0, porque sino cuando había registrado el primer número se quedaba en 1302, y el siguiente bit de arranque ya no se miraba cuando pasara medio bit sino cuando pasaba un bit y ya se nos descuadraba todo el circuito. Al ser un elemento tan sencillo como un contador, este componente decidimos no simularlo.

Unidad de Control

La unidad de control nos sirve para controlar tanto el registro como el detector de paridad. Antes de empezar a diseñar la unidad de control en vhdl tenemos que hacernos el diagrama de estados, que es el siguiente.

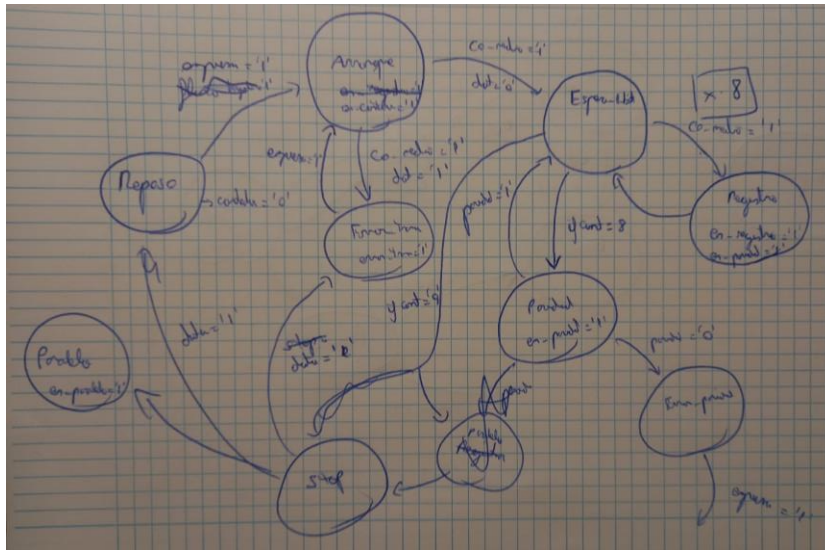


Figura 10 – Diagrama de estados de la unidad de control

Sabemos que en la imagen no se ve muy claro el diagrama, pero una vez que veamos en el código cómo cambian de un estado a otro y cómo van cambiando las salidas, creemos que quedará más claro. Lo único que hemos añadido respecto al apartado de introducción es que, si al final no hay error de trama ni de paridad, se pase a un estado que llamamos Paralelo, que nos permite copiar los datos del registro en paralelo a la salida.

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity Unidad_control is
6     port(
7         clk : in std_logic;
8         reset_n : in std_logic;
9         data : in std_logic;
10        paridad : in std_logic;
11        co_medio : in std_logic;
12        en_contador : out std_logic;
13        en_registro : out std_logic;
14        en_paridad : out std_logic;
15        err_trama : out std_logic;
16        err_paridad : out std_logic;
17        en_paralelo : out std_logic;
18        contar : out std_logic_vector(3 downto 0);
19        back_to_0 : out std_logic;
20    );
21    -- esta variable la utilizo solo para la simulación
22 end Unidad_control;
23
24 architecture behavioral of Unidad_control is
25     signal empieza, en, aux : std_logic;
26     signal cuenta : std_logic_vector(3 downto 0);
27
28     type t_estados is (Reposo, Arranque, EsperaBit, Registrar, Par, ErrorParidad, ErrorTrama, Parar, Paralelo); --
29     signal estado_act, estado_sig : t_estados;
30
31     component DetectorFlancoBajada
32     port(
33         e : in std_logic;
34         reset_n : in std_logic;
35         clk : in std_logic;
36         s : out std_logic;
37     );
38 end component;
39
40 component Contador
41     port(
42         reset_n : in std_logic;
43         aux : in std_logic;
44         clk : in std_logic;
45         en : in std_logic;
46         salida : out std_logic_vector(3 downto 0)
47     );
48 end component;
49
50 begin
51     i1_Flanco: DetectorFlancoBajada
52     port map(
53         e => data,
54         reset_n => reset_n,
55         clk => clk,
56         s => empieza
57     );
58
59     i1_Contador: Contador
60     port map(
61         reset_n => reset_n,
62         aux => aux,
63         clk => clk,
64         en => en,
65         salida => cuenta
66     );
67
68     VarEstado: process(clk, reset_n)
69     begin
70         if reset_n = '0' then
71             estado_act <= Reposo;
72         elsif rising_edge(clk) then
73             estado_act <= estado_sig;
74         end if;
75     end process VarEstado;
76
77     process(clk, reset_n)
78     begin
79         if reset_n = '0' then
80             estado_act <= Reposo;
81         elsif rising_edge(clk) then
82             estado_act <= estado_sig;
83         end if;
84     end process;
85
86     process(clk, reset_n)
87     begin
88         if reset_n = '0' then
89             estado_act <= Reposo;
90         elsif rising_edge(clk) then
91             estado_act <= estado_sig;
92         end if;
93     end process;
94
95     process(clk, reset_n)
96     begin
97         if reset_n = '0' then
98             estado_act <= Reposo;
99         elsif rising_edge(clk) then
100            estado_act <= estado_sig;
101        end if;
102    end process;
103
104     process(clk, reset_n)
105     begin
106         if reset_n = '0' then
107             estado_act <= Reposo;
108         elsif rising_edge(clk) then
109             estado_act <= estado_sig;
110         end if;
111     end process;
112
113     process(clk, reset_n)
114     begin
115         if reset_n = '0' then
116             estado_act <= Reposo;
117         elsif rising_edge(clk) then
118             estado_act <= estado_sig;
119         end if;
120     end process;
121
122     process(clk, reset_n)
123     begin
124         if reset_n = '0' then
125             estado_act <= Reposo;
126         elsif rising_edge(clk) then
127             estado_act <= estado_sig;
128         end if;
129     end process;
130
131     process(clk, reset_n)
132     begin
133         if reset_n = '0' then
134             estado_act <= Reposo;
135         elsif rising_edge(clk) then
136             estado_act <= estado_sig;
137         end if;
138     end process;
139
140     process(clk, reset_n)
141     begin
142         if reset_n = '0' then
143             estado_act <= Reposo;
144         elsif rising_edge(clk) then
145             estado_act <= estado_sig;
146         end if;
147     end process;
148
149     process(clk, reset_n)
150     begin
151         if reset_n = '0' then
152             estado_act <= Reposo;
153         elsif rising_edge(clk) then
154             estado_act <= estado_sig;
155         end if;
156     end process;
157
158     process(clk, reset_n)
159     begin
160         if reset_n = '0' then
161             estado_act <= Reposo;
162         elsif rising_edge(clk) then
163             estado_act <= estado_sig;
164         end if;
165     end process;
166
167     process(clk, reset_n)
168     begin
169         if reset_n = '0' then
170             estado_act <= Reposo;
171         elsif rising_edge(clk) then
172             estado_act <= estado_sig;
173         end if;
174     end process;
175
176     process(clk, reset_n)
177     begin
178         if reset_n = '0' then
179             estado_act <= Reposo;
180         elsif rising_edge(clk) then
181             estado_act <= estado_sig;
182         end if;
183     end process;
184
185     process(clk, reset_n)
186     begin
187         if reset_n = '0' then
188             estado_act <= Reposo;
189         elsif rising_edge(clk) then
190             estado_act <= estado_sig;
191         end if;
192     end process;
193
194     process(clk, reset_n)
195     begin
196         if reset_n = '0' then
197             estado_act <= Reposo;
198         elsif rising_edge(clk) then
199             estado_act <= estado_sig;
200         end if;
201     end process;
202
203     process(clk, reset_n)
204     begin
205         if reset_n = '0' then
206             estado_act <= Reposo;
207         elsif rising_edge(clk) then
208             estado_act <= estado_sig;
209         end if;
210     end process;
211
212     process(clk, reset_n)
213     begin
214         if reset_n = '0' then
215             estado_act <= Reposo;
216         elsif rising_edge(clk) then
217             estado_act <= estado_sig;
218         end if;
219     end process;
220
221     process(clk, reset_n)
222     begin
223         if reset_n = '0' then
224             estado_act <= Reposo;
225         elsif rising_edge(clk) then
226             estado_act <= estado_sig;
227         end if;
228     end process;
229
230     process(clk, reset_n)
231     begin
232         if reset_n = '0' then
233             estado_act <= Reposo;
234         elsif rising_edge(clk) then
235             estado_act <= estado_sig;
236         end if;
237     end process;
238
239     process(clk, reset_n)
240     begin
241         if reset_n = '0' then
242             estado_act <= Reposo;
243         elsif rising_edge(clk) then
244             estado_act <= estado_sig;
245         end if;
246     end process;
247
248     process(clk, reset_n)
249     begin
250         if reset_n = '0' then
251             estado_act <= Reposo;
252         elsif rising_edge(clk) then
253             estado_act <= estado_sig;
254         end if;
255     end process;
256
257     process(clk, reset_n)
258     begin
259         if reset_n = '0' then
260             estado_act <= Reposo;
261         elsif rising_edge(clk) then
262             estado_act <= estado_sig;
263         end if;
264     end process;
265
266     process(clk, reset_n)
267     begin
268         if reset_n = '0' then
269             estado_act <= Reposo;
270         elsif rising_edge(clk) then
271             estado_act <= estado_sig;
272         end if;
273     end process;
274
275     process(clk, reset_n)
276     begin
277         if reset_n = '0' then
278             estado_act <= Reposo;
279         elsif rising_edge(clk) then
280             estado_act <= estado_sig;
281         end if;
282     end process;
283
284     process(clk, reset_n)
285     begin
286         if reset_n = '0' then
287             estado_act <= Reposo;
288         elsif rising_edge(clk) then
289             estado_act <= estado_sig;
290         end if;
291     end process;
292
293     process(clk, reset_n)
294     begin
295         if reset_n = '0' then
296             estado_act <= Reposo;
297         elsif rising_edge(clk) then
298             estado_act <= estado_sig;
299         end if;
300     end process;
301
302     process(clk, reset_n)
303     begin
304         if reset_n = '0' then
305             estado_act <= Reposo;
306         elsif rising_edge(clk) then
307             estado_act <= estado_sig;
308         end if;
309     end process;
310
311     process(clk, reset_n)
312     begin
313         if reset_n = '0' then
314             estado_act <= Reposo;
315         elsif rising_edge(clk) then
316             estado_act <= estado_sig;
317         end if;
318     end process;
319
320     process(clk, reset_n)
321     begin
322         if reset_n = '0' then
323             estado_act <= Reposo;
324         elsif rising_edge(clk) then
325             estado_act <= estado_sig;
326         end if;
327     end process;
328
329     process(clk, reset_n)
330     begin
331         if reset_n = '0' then
332             estado_act <= Reposo;
333         elsif rising_edge(clk) then
334             estado_act <= estado_sig;
335         end if;
336     end process;
337
338     process(clk, reset_n)
339     begin
340         if reset_n = '0' then
341             estado_act <= Reposo;
342         elsif rising_edge(clk) then
343             estado_act <= estado_sig;
344         end if;
345     end process;
346
347     process(clk, reset_n)
348     begin
349         if reset_n = '0' then
350             estado_act <= Reposo;
351         elsif rising_edge(clk) then
352             estado_act <= estado_sig;
353         end if;
354     end process;
355
356     process(clk, reset_n)
357     begin
358         if reset_n = '0' then
359             estado_act <= Reposo;
360         elsif rising_edge(clk) then
361             estado_act <= estado_sig;
362         end if;
363     end process;
364
365     process(clk, reset_n)
366     begin
367         if reset_n = '0' then
368             estado_act <= Reposo;
369         elsif rising_edge(clk) then
370             estado_act <= estado_sig;
371         end if;
372     end process;
373
374     process(clk, reset_n)
375     begin
376         if reset_n = '0' then
377             estado_act <= Reposo;
378         elsif rising_edge(clk) then
379             estado_act <= estado_sig;
380         end if;
381     end process;
382
383     process(clk, reset_n)
384     begin
385         if reset_n = '0' then
386             estado_act <= Reposo;
387         elsif rising_edge(clk) then
388             estado_act <= estado_sig;
389         end if;
390     end process;
391
392     process(clk, reset_n)
393     begin
394         if reset_n = '0' then
395             estado_act <= Reposo;
396         elsif rising_edge(clk) then
397             estado_act <= estado_sig;
398         end if;
399     end process;
400
401     process(clk, reset_n)
402     begin
403         if reset_n = '0' then
404             estado_act <= Reposo;
405         elsif rising_edge(clk) then
406             estado_act <= estado_sig;
407         end if;
408     end process;
409
410     process(clk, reset_n)
411     begin
412         if reset_n = '0' then
413             estado_act <= Reposo;
414         elsif rising_edge(clk) then
415             estado_act <= estado_sig;
416         end if;
417     end process;
418
419     process(clk, reset_n)
420     begin
421         if reset_n = '0' then
422             estado_act <= Reposo;
423         elsif rising_edge(clk) then
424             estado_act <= estado_sig;
425         end if;
426     end process;
427
428     process(clk, reset_n)
429     begin
430         if reset_n = '0' then
431             estado_act <= Reposo;
432         elsif rising_edge(clk) then
433             estado_act <= estado_sig;
434         end if;
435     end process;
436
437     process(clk, reset_n)
438     begin
439         if reset_n = '0' then
440             estado_act <= Reposo;
441         elsif rising_edge(clk) then
442             estado_act <= estado_sig;
443         end if;
444     end process;
445
446     process(clk, reset_n)
447     begin
448         if reset_n = '0' then
449             estado_act <= Reposo;
450         elsif rising_edge(clk) then
451             estado_act <= estado_sig;
452         end if;
453     end process;
454
455     process(clk, reset_n)
456     begin
457         if reset_n = '0' then
458             estado_act <= Reposo;
459         elsif rising_edge(clk) then
460             estado_act <= estado_sig;
461         end if;
462     end process;
463
464     process(clk, reset_n)
465     begin
466         if reset_n = '0' then
467             estado_act <= Reposo;
468         elsif rising_edge(clk) then
469             estado_act <= estado_sig;
470         end if;
471     end process;
472
473     process(clk, reset_n)
474     begin
475         if reset_n = '0' then
476             estado_act <= Reposo;
477         elsif rising_edge(clk) then
478             estado_act <= estado_sig;
479         end if;
480     end process;
481
482     process(clk, reset_n)
483     begin
484         if reset_n = '0' then
485             estado_act <= Reposo;
486         elsif rising_edge(clk) then
487             estado_act <= estado_sig;
488         end if;
489     end process;
490
491     process(clk, reset_n)
492     begin
493         if reset_n = '0' then
494             estado_act <= Reposo;
495         elsif rising_edge(clk) then
496             estado_act <= estado_sig;
497         end if;
498     end process;
499
500     process(clk, reset_n)
501     begin
502         if reset_n = '0' then
503             estado_act <= Reposo;
504         elsif rising_edge(clk) then
505             estado_act <= estado_sig;
506         end if;
507     end process;
508
509     process(clk, reset_n)
510     begin
511         if reset_n = '0' then
512             estado_act <= Reposo;
513         elsif rising_edge(clk) then
514             estado_act <= estado_sig;
515         end if;
516     end process;
517
518     process(clk, reset_n)
519     begin
520         if reset_n = '0' then
521             estado_act <= Reposo;
522         elsif rising_edge(clk) then
523             estado_act <= estado_sig;
524         end if;
525     end process;
526
527     process(clk, reset_n)
528     begin
529         if reset_n = '0' then
530             estado_act <= Reposo;
531         elsif rising_edge(clk) then
532             estado_act <= estado_sig;
533         end if;
534     end process;
535
536     process(clk, reset_n)
537     begin
538         if reset_n = '0' then
539             estado_act <= Reposo;
540         elsif rising_edge(clk) then
541             estado_act <= estado_sig;
542         end if;
543     end process;
544
545     process(clk, reset_n)
546     begin
547         if reset_n = '0' then
548             estado_act <= Reposo;
549         elsif rising_edge(clk) then
550             estado_act <= estado_sig;
551         end if;
552     end process;
553
554     process(clk, reset_n)
555     begin
556         if reset_n = '0' then
557             estado_act <= Reposo;
558         elsif rising_edge(clk) then
559             estado_act <= estado_sig;
560         end if;
561     end process;
562
563     process(clk, reset_n)
564     begin
565         if reset_n = '0' then
566             estado_act <= Reposo;
567         elsif rising_edge(clk) then
568             estado_act <= estado_sig;
569         end if;
570     end process;
571
572     process(clk, reset_n)
573     begin
574         if reset_n = '0' then
575             estado_act <= Reposo;
576         elsif rising_edge(clk) then
577             estado_act <= estado_sig;
578         end if;
579     end process;
580
581     process(clk, reset_n)
582     begin
583         if reset_n = '0' then
584             estado_act <= Reposo;
585         elsif rising_edge(clk) then
586             estado_act <= estado_sig;
587         end if;
588     end process;
589
590     process(clk, reset_n)
591     begin
592         if reset_n = '0' then
593             estado_act <= Reposo;
594         elsif rising_edge(clk) then
595             estado_act <= estado_sig;
596         end if;
597     end process;
598
599     process(clk, reset_n)
600     begin
601         if reset_n = '0' then
602             estado_act <= Reposo;
603         elsif rising_edge(clk) then
604             estado_act <= estado_sig;
605         end if;
606     end process;
607
608     process(clk, reset_n)
609     begin
610         if reset_n = '0' then
611             estado_act <= Reposo;
612         elsif rising_edge(clk) then
613             estado_act <= estado_sig;
614         end if;
615     end process;
616
617     process(clk, reset_n)
618     begin
619         if reset_n = '0' then
620             estado_act <= Reposo;
621         elsif rising_edge(clk) then
622             estado_act <= estado_sig;
623         end if;
624     end process;
625
626     process(clk, reset_n)
627     begin
628         if reset_n = '0' then
629             estado_act <= Reposo;
630         elsif rising_edge(clk) then
631             estado_act <= estado_sig;
632         end if;
633     end process;
634
635     process(clk, reset_n)
636     begin
637         if reset_n = '0' then
638             estado_act <= Reposo;
639         elsif rising_edge(clk) then
640             estado_act <= estado_sig;
641         end if;
642     end process;
643
644     process(clk, reset_n)
645     begin
646         if reset_n = '0' then
647             estado_act <= Reposo;
648         elsif rising_edge(clk) then
649             estado_act <= estado_sig;
650         end if;
651     end process;
652
653     process(clk, reset_n)
654     begin
655         if reset_n = '0' then
656             estado_act <= Reposo;
657         elsif rising_edge(clk) then
658             estado_act <= estado_sig;
659         end if;
660     end process;
661
662     process(clk, reset_n)
663     begin
664         if reset_n = '0' then
665             estado_act <= Reposo;
666         elsif rising_edge(clk) then
667             estado_act <= estado_sig;
668         end if;
669     end process;
670
671     process(clk, reset_n)
672     begin
673         if reset_n = '0' then
674             estado_act <= Reposo;
675         elsif rising_edge(clk) then
676             estado_act <= estado_sig;
677         end if;
678     end process;
679
680     process(clk, reset_n)
681     begin
682         if reset_n = '0' then
683             estado_act <= Reposo;
684         elsif rising_edge(clk) then
685             estado_act <= estado_sig;
686         end if;
687     end process;
688
689     process(clk, reset_n)
690     begin
691         if reset_n = '0' then
692             estado_act <= Reposo;
693         elsif rising_edge(clk) then
694             estado_act <= estado_sig;
695         end if;
696     end process;
697
698     process(clk, reset_n)
699     begin
700         if reset_n = '0' then
701             estado_act <= Reposo;
702         elsif rising_edge(clk) then
703             estado_act <= estado_sig;
704         end if;
705     end process;
706
707     process(clk, reset_n)
708     begin
709         if reset_n = '0' then
710             estado_act <= Reposo;
711         elsif rising_edge(clk) then
712             estado_act <= estado_sig;
713         end if;
714     end process;
715
716     process(clk, reset_n)
717     begin
718         if reset_n = '0' then
719             estado_act <= Reposo;
720         elsif rising_edge(clk) then
721             estado_act <= estado_sig;
722         end if;
723     end process;
724
725     process(clk, reset_n)
726     begin
727         if reset_n = '0' then
728             estado_act <= Reposo;
729         elsif rising_edge(clk) then
730             estado_act <= estado_sig;
731         end if;
732     end process;
733
734     process(clk, reset_n)
735     begin
736         if reset_n = '0' then
737             estado_act <= Reposo;
738         elsif rising_edge(clk) then
739             estado_act <= estado_sig;
740         end if;
741     end process;
742
743     process(clk, reset_n)
744     begin
745         if reset_n = '0' then
746             estado_act <= Reposo;
747         elsif rising_edge(clk) then
748             estado_act <= estado_sig;
749         end if;
750     end process;
751
752     process(clk, reset_n)
753     begin
754         if reset_n = '0' then
755             estado_act <= Reposo;
756         elsif rising_edge(clk) then
757             estado_act <= estado_sig;
758         end if;
759     end process;
760
761     process(clk, reset_n)
762     begin
763         if reset_n = '0' then
764             estado_act <= Reposo;
765         elsif rising_edge(clk) then
766             estado_act <= estado_sig;
767         end if;
768     end process;
769
770     process(clk, reset_n)
771     begin
772         if reset_n = '0' then
773             estado_act <= Reposo;
774         elsif rising_edge(clk) then
775             estado_act <= estado_sig;
776         end if;
777     end process;
778
779     process(clk, reset_n)
780     begin
781         if reset_n = '0' then
782             estado_act <= Reposo;
783         elsif rising_edge(clk) then
784             estado_act <= estado_sig;
785         end if;
786     end process;
787
788     process(clk, reset_n)
789     begin
790         if reset_n = '0' then
791             estado_act <= Reposo;
792         elsif rising_edge(clk) then
793             estado_act <= estado_sig;
794         end if;
795     end process;
796
797     process(clk, reset_n)
798     begin
799         if reset_n = '0' then
800             estado_act <= Reposo;
801         elsif rising_edge(clk) then
802             estado_act <= estado_sig;
803         end if;
804     end process;
805
806     process(clk, reset_n)
807     begin
808         if reset_n = '0' then
809             estado_act <= Reposo;
810         elsif rising_edge(clk) then
811             estado_act <= estado_sig;
812         end if;
813     end process;
814
815     process(clk, reset_n)
816     begin
817         if reset_n = '0' then
818             estado_act <= Reposo;
819         elsif rising_edge(clk) then
820             estado_act <= estado_sig;
821         end if;
822     end process;
823
824     process(clk, reset_n)
825     begin
826         if reset_n = '0' then
827             estado_act <= Reposo;
828         elsif rising_edge(clk) then
829             estado_act <= estado_sig;
830         end if;
831     end process;
832
833     process(clk, reset_n)
834     begin
835         if reset_n = '0' then
836             estado_act <= Reposo;
837         elsif rising_edge(clk) then
838             estado_act <= estado_sig;
839         end if;
840     end process;
841
842     process(clk, reset_n)
843     begin
844         if reset_n = '0' then
845             estado_act <= Reposo;
846         elsif rising_edge(clk) then
847             estado_act <= estado_sig;
848         end if;
849     end process;
850
851     process(clk, reset_n)
852     begin
853         if reset_n = '0' then
854             estado_act <= Reposo;
855         elsif rising_edge(clk) then
856             estado_act <= estado_sig;
857         end if;
858     end process;
859
860     process(clk, reset_n)
861     begin
862         if reset_n = '0' then
863             estado_act <= Reposo;
864         elsif rising_edge(clk) then
865             estado_act <= estado_sig;
866         end if;
867     end process;
868
869     process(clk, reset_n)
870     begin
871         if reset_n = '0' then
872             estado_act <= Reposo;
873         elsif rising_edge(clk) then
874             estado_act <= estado_sig;
875         end if;
876     end process;
877
878     process(clk, reset_n)
879     begin
880         if reset_n = '0' then
881             estado_act <= Reposo;
882         elsif rising_edge(clk) then
883             estado_act <= estado_sig;
884         end if;
885     end process;
886
887     process(clk, reset_n)
888     begin
889         if reset_n = '0' then
890             estado_act <= Reposo;
891         elsif rising_edge(clk) then
892             estado_act <= estado_sig;
893         end if;
894     end process;
895
896     process(clk, reset_n)
897     begin
898         if reset_n = '0' then
899             estado_act <= Reposo;
900         elsif rising_edge(clk) then
901             estado_act <= estado_sig;
902         end if;
903     end process;
904
905     process(clk, reset_n)
906     begin
907         if reset_n = '0' then
908             estado_act <= Reposo;
909         elsif rising_edge(clk) then
910             estado_act <= estado_sig;
911         end if;
912     end process;
913
914     process(clk, reset_n)
915     begin
916         if reset_n = '0' then
917             estado_act <= Reposo;
918         elsif rising_edge(clk) then
919             estado_act <= estado_sig;
920         end if;
921     end process;
922
923     process(clk, reset_n)
924     begin
925         if reset_n = '0' then
926             estado_act <= Reposo;
927         elsif rising_edge(clk) then
928             estado_act <= estado_sig;
929         end if;
930     end process;
931
932     process(clk, reset_n)
933     begin
934         if reset_n = '0' then
935             estado_act <= Reposo;
936         elsif rising_edge(clk) then
937             estado_act <= estado_sig;
938         end if;
939     end process;
940
941     process(clk, reset_n)
942     begin
943         if reset_n = '0' then
944             estado_act <= Reposo;
945         elsif rising_edge(clk) then
946             estado_act <= estado_sig;
947         end if;
948     end process;
949
950     process(clk, reset_n)
951     begin
952         if reset_n = '0' then
953             estado_act <= Reposo;
954         elsif rising_edge(clk) then
955             estado_act <= estado_sig;
956         end if;
957     end process;
958
959     process(clk, reset_n)
960     begin
961         if reset_n = '0' then
962             estado_act <= Reposo;
963         elsif rising_edge(clk) then
964             estado_act <= estado_sig;
965         end if;
966     end process;
967
968     process(clk, reset_n)
969     begin
970         if reset_n = '0' then
971             estado_act <= Reposo;
972         elsif rising_edge(clk) then
973             estado_act <= estado_sig;
974         end if;
975     end process;
976
977     process(clk, reset_n)
978     begin
979         if reset_n = '0' then
980             estado_act <= Reposo;
981         elsif rising_edge(clk) then
982             estado_act <= estado_sig;
983         end if;
984     end process;
985
986     process(clk, reset_n)
987     begin
988         if reset_n = '0' then
989             estado_act <= Reposo;
990         elsif rising_edge(clk) then
991             estado_act <= estado_sig;
992         end if;
993     end process;
994
995     process(clk, reset_n)
996     begin
997         if reset_n = '0' then
998             estado_act <= Reposo;
999         elsif rising_edge(clk) then
1000            estado_act <= estado_sig;
1001        end if;
1002    end process;
1003
1004     process(clk, reset_n)
1005     begin
1006         if reset_n = '0' then
1007             estado_act <= Reposo;
1008         elsif rising_edge(clk) then
1009             estado_act <= estado_sig;
1010         end if;
1011     end process;
1012
1013     process(clk, reset_n)
1014     begin
1015         if reset_n = '0' then
1016             estado_act <= Reposo;
1017         elsif rising_edge(clk) then
1018             estado_act <= estado_sig;
1019         end if;
1020     end process;
1021
1022     process(clk, reset_n)
1023     begin
1024         if reset_n = '0' then
1025             estado_act <= Reposo;
1026         elsif rising_edge(clk) then
1027             estado_act <= estado_sig;
1028         end if;
1029     end process;
1030
1031     process(clk, reset_n)
1032     begin
1033         if reset_n = '0' then
1034             estado_act <= Reposo;
1035         elsif rising_edge(clk) then
1036             estado_act <= estado_sig;
1037         end if;
1038     end process;
1039
1040     process(clk, reset_n)
1041     begin
1042         if reset_n = '0' then
1043             estado_act <= Reposo;
1044         elsif rising_edge(clk) then
1045             estado_act <= estado_sig;
1046         end if;
1047     end process;
1048
1049     process(clk, reset_n)
1050     begin
1051         if reset_n = '0' then
1052             estado_act <= Reposo;
1053         elsif rising_edge(clk) then
1054             estado_act <= estado_sig;
1055         end if;
1056     end process;
1057
1058     process(clk, reset_n)
1059     begin
1060         if reset_n = '0' then
1061             estado_act <= Reposo;
1062         elsif rising_edge(clk) then
1063             estado_act <= estado_sig;
1064         end if;
1065     end process;
1066
1067     process(clk, reset_n)
1068     begin
1069         if reset_n = '0' then
1070             estado_act <= Reposo;
1071         elsif rising_edge(clk) then
1072             estado_act <= estado_sig;
1073         end if;
1074     end process;
1075
1076     process(clk, reset_n)
1077     begin
1078         if reset_n = '0' then
1079             estado_act <= Reposo;
1080         elsif rising_edge(clk) then
1081             estado_act <= estado_sig;
1082         end if;
1083     end process;
1084
1085     process(clk, reset_n)
1086     begin
1087         if reset_n = '0' then
1088             estado_act <= Reposo;
1089         elsif rising_edge(clk) then
1090             estado_act <= estado_sig;
1091         end if;
1092     end process;
1093
1094     process(clk, reset_n)
1095     begin
1096         if reset_n = '0' then
1097             estado_act <= Reposo;
1098         elsif rising_edge(clk) then
1099             estado_act <= estado_sig;
1100         end if;
1101     end process;
1102
1103     process(clk, reset_n)
1104     begin
1105         if reset_n = '0' then
1106             estado_act <= Reposo;
1107         elsif rising_edge(clk) then
1108             estado_act <= estado_sig;
1109         end if;
1110     end process;
1111
1112     process(clk, reset_n)
1113     begin
1114         if reset_n = '0' then
1115             estado_act <= Reposo;
1116         elsif rising_edge(clk) then
1117             estado_act <= estado_sig;
1118         end if;
1119     end process;
1120
1121     process(clk, reset_n)
1122     begin
1123         if reset_n = '0' then
1124             estado_act <= Reposo;
1125         elsif rising_edge(clk) then
1126             estado_act <= estado_sig;
1127         end if;
1128     end process;
1129
1130     process(clk, reset_n)
1131     begin
1132         if reset_n = '0' then
1133             estado_act <= Reposo;
1134         elsif rising_edge(clk) then
1135             estado_act <= estado_sig;
1136         end if;
1137     end process;
1138
1139     process(clk, reset_n)
1140     begin
1141         if reset_n = '0' then
1142             estado_act <= Reposo;
1143         elsif rising_edge(clk) then
1144             estado_act <= estado_sig;
1145         end if;
1146     end process;
1147
1148     process(clk, reset_n)
1149     begin
1150         if reset_n = '0' then
1151             estado_act <= Reposo;
1152         elsif rising_edge(clk) then
1153             estado_act <= estado_sig;
1154         end if;
1155     end process;
1156
1157     process(clk, reset_n)
1158     begin
1159         if reset_n = '0' then
1160             estado_act <= Reposo;
1161         elsif rising_edge(clk) then
1162             estado_act <= estado_sig;
1163         end if;
1164     end process;
1165
1166     process(cl
```

```

TransicionEstados : process (estado_act, estado_sig, empieza, co_medio, data, paridad)
begin
    estado_sig <= estado_act;
    case estado_act is
        when Reposo =>
            if empieza = '1' then
                estado_sig <= Arranque;
            end if;
        when Arranque =>
            if co_medio = '1' then
                if data = '0' then
                    estado_sig <= Espera1bit;
                elsif data = '1' then
                    estado_sig <= ErrorTrama;
                end if;
            end if;
        when Espera1bit =>
            if co_medio = '1' then
                if cuenta = "1000" then
                    estado_sig <= Par;
                elsif cuenta = "1001" then
                    estado_sig <= Parar;
                else
                    estado_sig <= Registrar;
                end if;
            end if;
        when Registrar =>
            estado_sig <= Espera1bit;
        when Par =>
            estado_sig <= Espera1bit;
        when Parar =>
            if data = '0' then
                estado_sig <= ErrorTrama;
            elsif paridad = '0' then
                estado_sig <= ErrorParidad;
            else
                estado_sig <= Paralelo;
            end if;
        when Paralelo =>
            estado_sig <= Reposo;
        when ErrorTrama =>
            if empieza = '1' then
                estado_sig <= Arranque;
            end if;
        when ErrorParidad =>
            if empieza = '1' then
                estado_sig <= Arranque;
            end if;
        when others =>
            estado_sig <= Reposo;
    end case;
end process TransicionEstados;

```

Figura 11 – Transición de estados de la máquina de estados

En la imagen de arriba vemos como cambian los estados de uno a otro y vamos a explicar esta transición. En primer lugar, si la señal de empieza, que es la que sale del flanco de bajada, se pone a uno, pasamos al estado de Arranque. Una vez en este estado, esperamos medio bit y si al terminar ese tiempo data no es 0 pasamos a ErrorTrama y si, por el contrario, es 0, pasamos a Espera1bit. Después de esperar un bit, vamos a registrar el siguiente número que entre y volvemos a esperar un bit. Este proceso se repite 8 veces. Cuando termine este proceso, comprobamos la paridad: si es 1 pasamos a comprobar el bit de stop y si es 0, vamos a ErrorParidad. Si cuando comprobamos el bit de stop, data es 0 vamos a ErrorTrama y si es 1, vamos a registrar la salida en paralelo. Después de registrar la salida, volvemos al reposo.

```

134 Salidas : process (estado_act)
135 begin
136
137     aux      <= '0';
138     back_to_0 <= '0';
139     en       <= '0';
140     en_contador <= '0';
141     err_trama <= '0';
142     en_registro <= '0';
143     en_paridad <= '0';
144     err_paridad <= '0';
145     en_paralelo <= '0';
146
147     case estado_act is
148     when Reposo =>
149         back_to_0 <= '1';
150     when Arranque =>
151         aux <= '1';
152         en_contador <= '1';
153     when Espera1bit =>
154         en_contador <= '1';
155     when Registrar =>
156         en <= '1';
157         en_registro <= '1';
158         en_paridad <= '1';
159         en_contador <= '1';
160     when Par =>
161         en <= '1';
162         en_paridad <= '1';
163         en_contador <= '1';
164     when Parar =>
165         null;
166     when Paralelo =>
167         en_paralelo <= '1';
168     when ErrorTrama =>
169         err_trama <= '1';
170     when ErrorParidad =>
171         err_paridad <= '1';
172     when others =>
173         null;
174     end case;
175 end process Salidas;
176
177 contar <= cuenta;
178
179 end behavioral;

```

Figura 12 – Salidas en de la Máquina de estados

En lo que se refiere a las salidas, ocurre lo siguiente. Inicialmente, colocamos todos nuestros componentes en 0. Cuando estamos en Arranque, habilitamos la cuenta del contador de un bit. Esta cuenta va a seguir habilitada cuando espera un bit, cuando está registrando y cuando comprueba la paridad. Cuando estamos en el estado de Registro habilitamos el registro para que se vayan registrando los bits. En Registro y cuando comprobamos la paridad, también tendremos activas las señales del contador de 9 bits y la que habilita el componente que comprueba la paridad. Por último, si estamos en cualquiera de los dos estados de error, se activará la correspondiente señal de error y cuando estemos en paralelo, se habilitará el registro paralelo.

Una vez que hemos terminado nuestro código y hemos comprobado que no teníamos errores compilando, vamos a simular la unidad de control para ver si nuestro código es funcional. A la hora de simular, le he metido una variable al circuito para que fuera llevando la cuenta del contador de módulo 9 para que pudiera ver si había algún error. Como se puede observar la simulación funciona correctamente, ya que realiza el trabajo para el que ha sido diseñada.

```

73  p_stim: process
74  begin
75      data <= '1';
76      paridad <= '0';
77      co_medio <= '0';
78
79      wait until reset_n = '1';
80
81      wait for 5 ns;
82
83      -- debería empezar el programa
84
85      data <= '0';
86
87      wait for 60 ns;
88
89      assert en_contador = '1'
90          report "El contador no se enciende"
91          severity failure;
92      wait for 40 ns;
93      co_medio <= '1';
94
95
96      wait for clk_per;
97      co_medio <= '0';
98      assert en_contador = '1'
99          report "El contador se para después al pasar a Espera1bit"
100         severity failure;
101
102      wait for 80 ns;
103
104      for n in 0 to 3 loop
105          data <= '1';
106
107          wait for 100 ns;
108
109          co_medio <= '1';
110
111          wait for clk_per;
112
113          co_medio <= '0';
114
115          wait for 80 ns;
116
117          data <= '0';
118
119          wait for 100 ns;
120
121          co_medio <= '1';
122
123          wait for clk_per;
124
125          co_medio <= '0';
126
127          wait for 80 ns;
128      end loop;
129
130      -- paridad
131      data <= '1';
132      wait for 100 ns;
133      co_medio <= '1';
134      wait for clk_per;
135      co_medio <= '0';
136      wait for 180 ns;
137      co_medio <= '1';
138      wait for clk_per;
139      co_medio <= '0';
140
141      wait for 100 ns;
142
143      assert false
144          report "Fin de la simulación"
145          severity failure;
146
147  end process p_stim;
148  end Unidad_control_arch;
149

```

Figura 13 – Testbench de la unidad lógica

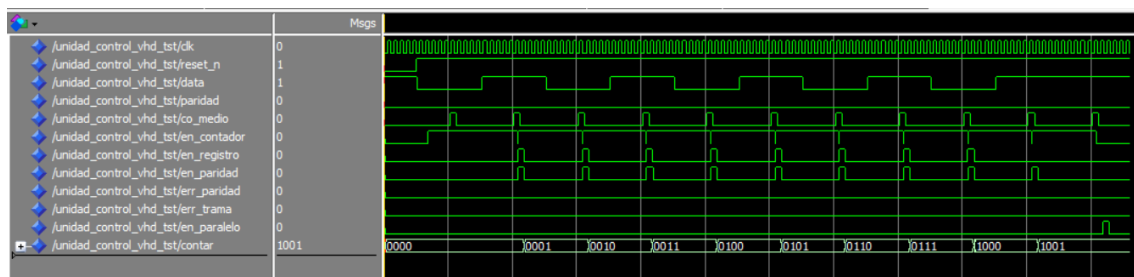


Figura 14 – Simulación de la Unidad Lógica

Circuito completo

Por último, debemos juntar todos los componentes en un componente más grande al que llamaremos práctica1. Lo único que vamos a hacer en este componente es instanciar el resto de los componentes que hemos nombrado en el apartado anterior.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Practical is
5      port(
6          clk : in std_logic;
7          reset_n : in std_logic;
8          data : in std_logic;
9          err_paridad : out std_logic;
10         err_trama : out std_logic;
11         --sal_registro : out std_logic_vector(7 downto 0);
12         --contar : out std_logic_vector(3 downto 0);
13         bits : out std_logic_vector(7 downto 0)
14         --para : out std_logic
15     );
16 end Practical;
17
18 architecture structural of Practical is
19
20     signal paridad, en_paridad, en_registro, co_medio, en_contador, back_to_0: std_logic;
21     signal en_paralelo: std_logic;
22     signal sal_regi : std_logic_vector(7 downto 0);
23
24     component Comprobar_paridad
25     port(
26         data : in std_logic;
27         paridad : out std_logic;
28         clk : in std_logic;
29         reset_n : in std_logic;
30         back_to_0 : in std_logic;
31         enable : in std_logic
32     );
33 end component;
34
35     component Registro
36     port(
37         clk : in std_logic;
38         enable : in std_logic;
39         reset_n : in std_logic;
40         data : in std_logic;
41         bits : out std_logic_vector(7 downto 0)
42     );
43 end component;
44
45     component Contador1bit
46     port(
47         reset_n : in std_logic;
48         back_to_0 : in std_logic;
49         clk : in std_logic;
50         en : in std_logic;
51         co_medio : out std_logic
52     );
53 end component;
54
55     component Unidad_Control
56     port(
57         clk : in std_logic;
58         reset_n : in std_logic;
59         data : in std_logic;
60         paridad : in std_logic;
61         co_medio : in std_logic;
62         en_contador : out std_logic;
63         en_registro : out std_logic;
64         en_paridad : out std_logic;
65         err_trama : out std_logic;
66         err_paridad : out std_logic;
67         en_paralelo : out std_logic;
68         contar : out std_logic_vector(3 downto 0);
69         back_to_0 : out std_logic
70     );
71 end component;
72
73     component Registro_paralelo
74     port(
75         clk : in std_logic;
76         enable : in std_logic;
77         reset_n : in std_logic;
78         entrada : in std_logic_vector(7 downto 0);
79         salida : out std_logic_vector(7 downto 0)
80     );
81 end component;
```

```

83 begin
84
85     i1_Comprobarparidad: Comprobar_paridad
86     port map(
87         data => data,
88         paridad => paridad,
89         clk => clk,
90         reset_n => reset_n,
91         enable => en_paridad,
92         back_to_0 => back_to_0
93     );
94
95     i1_Registro : Registro
96     port map(
97         clk => clk,
98         enable => en_registro,
99         reset_n => reset_n,
100        data => data,
101        bits => sal_regi
102    );
103
104     i1_Contador : Contador1bit
105     port map(
106         clk => clk,
107         back_to_0 => back_to_0,
108         reset_n => reset_n,
109         en => en_contador,
110         co_medio => co_medio
111    );
112
113     i1_UnidadControl: Unidad_Control
114     port map(
115         clk => clk,
116         reset_n => reset_n,
117         data => data,
118         paridad => paridad,
119         co_medio => co_medio,
120         en_contador => en_contador,
121         en_registro => en_registro,
122         en_paridad => en_paridad,
123         err_trama => err_trama,
124         err_paridad => err_paridad,
125         en_paralelo => en_paralelo,
126         contar => open,
127         back_to_0 => back_to_0
128    );

```

```

130     i1_RegistroPara: Registro_paralelo
131     port map(
132         clk => clk,
133         enable => en_paralelo,
134         reset_n => reset_n,
135         entrada => sal_regi,
136         salida => bits
137    );
138
139     --sal_registro <= sal_regi;
140     --para <= en_paralelo;
141 end structural;

```

Figura 15 – Código VHDL de la práctica1

Una vez que hemos juntado todos los componentes en el elemento práctica 1, vamos a simular todo el componente para ver si realmente funciona nuestro código.

```

62     p_stim : process
63
64     begin
65         data <= '1';
66
67         wait until reset_n = '1';
68
69         wait for 5 ns;
70
71         --bits de start
72
73         data <= '0';
74
75         wait for 26040 ns;
76
77         --bits de entrada
78
79         for n in 0 to 1 loop
80             data <= '0';
81
82             wait for 26040 ns;
83
84             data <= '0';
85
86             wait for 26040 ns;
87         end loop;
88
89         data <= '1';
90
91         wait for 26040 ns;
92
93         data <= '1';
94
95         wait for 26040 ns;
96
97         data <= '0';
98
99         wait for 26040 ns;
100
101         data <= '0';
102
103         wait for 26040 ns;
104
105         -- bit de paridad
106
107         data <= '1';
108
109         wait for 26040 ns;
110
111         -- bit de stop
112         data <= '1';
113
114         wait for 50000 ns;
115
116         assert bits="00110000"
117             report "El número obtenido no es el deseado"
118             severity failure;
119
120         data <= '0';
121
122         wait for 26040 ns;
123
124         --bits de entrada
125
126         data <= '1';
127
128         wait for 26040 ns;
129
130         data <= '0';
131
132         wait for 26040 ns;
133
134         data <= '0';
135
136         wait for 26040 ns;
137
138         data <= '0';
139
140         wait for 26040 ns;
141
142         data <= '1';
143
144         wait for 26040 ns;
145
146         data <= '1';
147
148         wait for 26040 ns;
149
150         data <= '0';
151
152         wait for 26040 ns;
153
154         data <= '0';
155
156         wait for 26040 ns;
157

```

```

155     data <= '0';
156
157     wait for 26040 ns;
158
159     -- bit de paridad
160
161     data <= '0';
162
163     wait for 26040 ns;
164
165     -- bit de stop
166     data <= '1';
167
168     wait for 50000 ns;
169
170     assert bits="00110001"
171     report "El número obtenido no es el deseado"
172     severity failure;
173
174     data <= '0';
175
176     wait for 26040 ns;
177
178     --bits de entrada
179
180     for n in 0 to 3 loop
181         data <= '1';
182
183         wait for 26040 ns;
184
185         data <= '0';
186
187         wait for 26040 ns;
188     end loop;
189
190     -- bit de paridad
191
192     data <= '1';
193
194     wait for 26040 ns;
195
196     -- bit de stop
197     data <= '1';
198
199     wait for 50000 ns;
200
201     assert bits="01010101"
202     report "El número obtenido no es el deseado"
203     severity failure;
204
205     data <= '0';
206
207     wait for 26040 ns;
208
209     --bits de entrada
210
211     for n in 0 to 2 loop
212         data <= '1';
213
214         wait for 26040 ns;
215
216         data <= '0';
217
218         wait for 26040 ns;
219     end loop;
220     data <= '1';
221
222     wait for 26040 ns;
223
224     data <= '1';
225
226     wait for 26040 ns;
227
228     -- bit de paridad
229
230     data <= '0';
231
232     wait for 26040 ns;
233
234     -- bit de stop
235     data <= '1';
236
237     wait for 50000 ns;
238
239     assert bits="11010101"
240     report "El número obtenido no es el deseado"
241     severity failure;
242
243     assert false
244     report "Fin de la simulación"
245     severity failure;
246
247     end process p_stim;
248 end Practical_arch;

```

Figura 16 – Testbench de la práctica

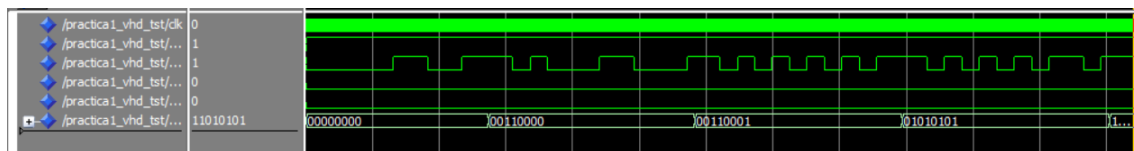


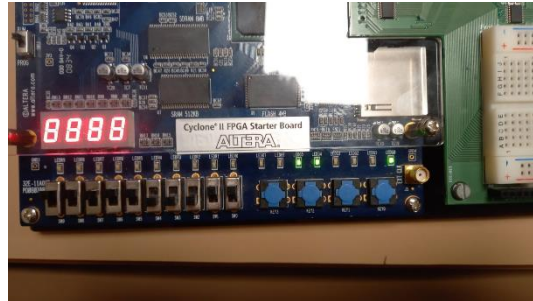
Figura 17 – Simulación de la práctica entera

Como vemos le hemos pasado 4 números seguidos con paridad impar para ver si funciona correctamente nuestro código y efectivamente ha devuelto lo que esperábamos.

Implantación física

Para realizar la implantación física y la comprobación mediante el código [ASCII](#), hemos necesitado conectar la placa mediante el cable USB y mediante el COM1 usando el programa hyperterm.

Hemos obtenido los siguientes resultados:



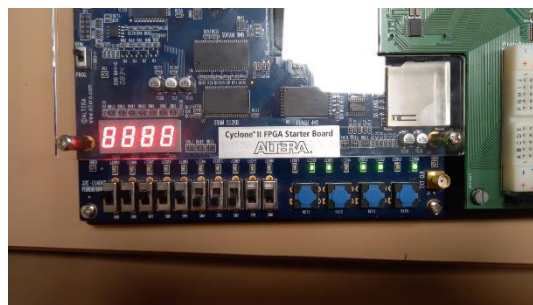
1--> 49 -> 00110001 (binario) -> 31 (hexadecimal)



0--> 48 -> 00110000 (binario) -> 30 (hexadecimal)



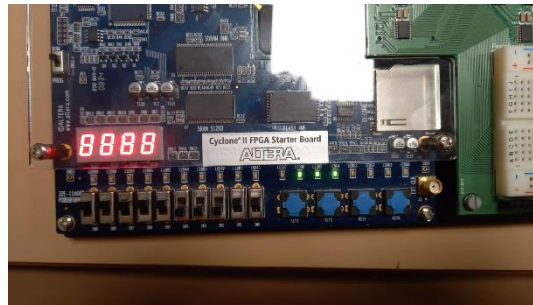
c-->99 -> 01100011 (binario) -> 63 (hexadecimal)



m--> 109 -> 01101101 (binario) -> 6D (hexadecimal)



o--> 111 -> 01101111 (binario) -> 6F (hexadecimal)



p--> 112 -> 01110000 -> 70 (hexadecimal)

Conclusiones y análisis de los resultados

Al inicio de esta práctica teníamos tres objetivos principales. El primero era comprender el funcionamiento del transmisor y su unidad de control. Al principio, nos fue muy difícil comprender la práctica, qué era lo que teníamos que hacer, pero tras la explicación del profesor y mientras íbamos haciendo el código, fuimos comprendiendo mucho mejor cómo funcionaba un transmisor RS-232. Después de haber realizado la práctica y, principalmente, haber diseñado en VHDL el transmisor RS-232, hemos llegado a comprender cómo funciona completamente.

Nuestro segundo objetivo era diseñar, simular e implantar el transmisor RS-232 en FPGA. Este objetivo también lo hemos cumplido ya que como hemos visto hemos diseñado en VHDL un transmisor desde 0, componente a componente. Después de haber escrito el código, hemos conseguido simularla e implantarla en la placa FPGA con éxito.

Una vez realizada la práctica, debemos decir que realmente nos ha costado llegar a simular e implantar con éxito el componente. Hemos cometido bastantes errores, pero lo bueno es que gracias a la simulación hemos podido identificarlos para después corregirlos. Por ejemplo, la primera vez que lo implantamos en la placa, íbamos confiados porque la simulación nos había salido bien. Sin embargo, únicamente habíamos realizado la simulación del componente pasándole un número y no varios. Por lo tanto, al meterlo en la placa, el primer número se transmitía bien, pero si seguías transmitiendo, aparecían errores de paridad, de trama y se iluminaban bits que no se tenían que iluminar. Después de realizar una simulación más completa descubrimos que había que reinicializar nuestro comprobador de paridad y también nuestro contador de un bit. Una vez que lo corregimos, lo volvimos a implantar y en este caso ya funcionaba correctamente en la placa.