Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
Curso IE-0523: Circuitos Digitales II
II ciclo 2023

Diseño de un Bloque de PCS Tipo 1000BASE-X

Leonardo Leiva Vasquez C14172
Juan Pablo Morales Vargas B95322
Fabricio Donato Hidalgo B62378
Grupo 08

Profesor: Enrique Coen Alfaro

9 de diciembre, 2023

1. Resumen

Este proyecto se centra en el desarrollo de tres componentes simplificados esenciales del estándar de ethernet, especificamente para conformar un bloque PCS tipo 1000BASE-X: los módulos receptor, transmisor y sincronizador.

El módulo receptor es responsable de la recepción precisa de datos transmitidos a través de la red, convirtiéndose en una pieza clave para asegurar la integridad y la confiabilidad de la información recibida. Por otro lado, el módulo transmisor se ocupa de la correcta codificación y envío de datos, asegurando que la información sea transmitida de manera efectiva y sin errores atreves de las pruebas teniendo en cuenta las reglas de paridad. Finalmente, el módulo sincronizador juega un rol vital en mantener la coherencia y la sincronización de los datos a lo largo de estos procesos, garantizando así una comunicación fluida y coherente entre los módulos.

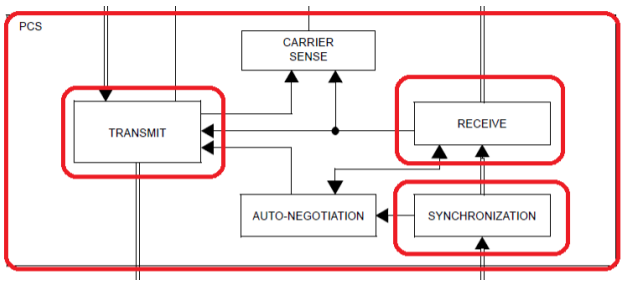

Figura 1: Diagrama General del PCS

Tras una serie de pruebas individuales en las que se prueba a los módulos uno por uno y asegurándose que funcionen correctamente, se realiza una prueba en general para ver cómo funcionan los módulos unidos en un wrapper y conectados sus salidas y entradas. Los resultados son bastante satisfactorios ya que los tres se comunicaron como se tenía planeado y transmiten y reciben los datos con una buena sincronía. Cuando se vayan a realizar trabajos de varios módulos siempre se recomienda probarlos por separado para comprobar que funcionen correctamente. Además en un futuro, se podría explorar secciones adicionales del estándar, como PMA o PMD, para comprender mejor la transmisión de datos. Estas secciones ofrecen una visión más detallada de la codificación y transmisión a través del medio físico, mejorando el conocimiento en el diseño e implementación, especialmente en aspectos físicos.

1.1. Arquitectura General

La arquitectura del sistema diseñado se fundamenta en la implementación de la subcapa de codificación física (PCS) según lo especificado en la cláusula 36 del estándar IEEE 802.3. Esta implementación se estructura en varios módulos de Verilog, cada uno correspondiendo a una parte fundamental de la PCS. La interconexión de estos módulos forma una representación simplificada y funcional del sistema de transmisión y recepción.

El módulo `Transmisor.v` se construye como el núcleo de la sección de transmisión, encapsulando los procesos de codificación y serialización de los datos. Dentro de este módulo, se instancian dos componentes clave: `Transmisor_code.v` y `Transmisor_oset.v`. El módulo `Transmisor_code.v` se encarga de la codificación 8B/10B de los datos, una técnica esencial para la transmisión de Ethernet que mejora la sincronización de la señal y permite una detección de errores más efectiva. Por su parte, el módulo `Transmisor_oset.v` gestiona el desplazamiento

de la señalización y la alineación de los bordes de señal, garantizando que la transmisión cumpla con las especificaciones de temporización del estándar.

```
15  Tx_oset oset(
16    .power(power),
17    .clock(clock),
18    .reset(reset),
19    .tx_o_set(tx_o_set),
20    .tx_even(tx_even),
21    .TX_OSET_indicate(TX_OSET_indicate),
22    .transmitting(transmitting),
23    .TX_EN(TX_EN),
24    .TXD(TXD)
25  );
26
27  Tx_code code(
28    .power(power),
29    .clock(clock),
30    .reset(reset),
31    .tx_o_set(tx_o_set),
32    .tx_even(tx_even),
33    .TX_OSET_indicate(TX_OSET_indicate),
34    .TX_code_group(TX_code_group),
35    .TXD(TXD)
36  );
37
```

Figura 2: Intancias en el modulo `Transmisor.v`

En el lado de la recepción, los módulos `Receptor.v` y `Sincronizador.v` son instanciados dentro del módulo más grande, `Wrapper.v`. El `Receptor.v` maneja la recepción y decodificación de las señales entrantes, mientras que el `Sincronizador.v` asegura que las señales estén correctamente alineadas en el tiempo para su procesamiento. El módulo `Wrapper.v` encapsula toda la funcionalidad de la PCS, proporcionando una interfaz clara y manejable para la transmisión y recepción de datos dentro de la red, por lo que además de instaciar al `Receptor.v` y al `Sincronizador.v` también instancia al `Transmisor.v`.

```
// Instancias de los módulos
Transmisor TRANSMISOR (/*AUTOINST*/
            // Outputs
            .TX_code_group   (TX_code_group),
            // Inputs
            .power       (power),
            .clock       (clock),
            .reset       (reset),
            .TX_EN       (TX_EN),
            .TXD         (TXD[7:0]));

Receptor RECEPTOR (/*AUTOINST*/
            // Outputs
            .RXD     (RXD[7:0]),
            .RX_DV       (RX_DV),
            // Inputs
            .CLK     (clock),
            .SUDI        (rx_code_group_out[9:0]),
            .sync_status (sync_status),
            .RESET       (reset));

Sincronizador SINCRONIZADOR (/*AUTOINST*/
             // Outputs
             .rx_code_group_out (rx_code_group_out[9:0]),
             .RX_EVEN       (RX_EVEN),
             .sync_status           (sync_status),
             // Inputs
             .clk           (clock),
             .reset         (reset),
             .rx_code_group_in  (TX_code_group));
```

Figura 3: Intancias en el modulo `Wrapper.v`

Para la verificación y prueba del sistema, se emplea el módulo `testbench.v`. Este entorno de prueba es crucial para validar el diseño del sistema PCS. En él se instancian `Wrapper.v` y `Tester.v`, donde `Tester.v` genera patrones de prueba y simula las condiciones operativas para el sistema. El `testbench.v` permite la simulación de casos de prueba que emulan el comportamiento del sistema en un ambiente de producción, asegurando que todas las partes del diseño trabajen coherentemente y según las especificaciones.

```
23  //Instancias
24  Tester TESTER(/*AUTOINST*/
25              // Outputs
26              .clock              (clock),
27              .reset              (reset),
28              .power              (power),
29              .TXD                (TXD),
30              .TX_EN              (TX_EN),
31              .RX_EVEN            (RX_EVEN),
32              .rx_code_group          (rx_code_group),
33              .TX_code_group          (TX_code_group),
34              .RX_DV              (RX_DV));
35
36  Wrapper WRAPPER (/*AUTOINST*/
37              // Outputs
38              .clock              (clock),
39              .reset              (reset),
40              .power              (power),
41              .TXD                (TXD),
42              .TX_EN              (TX_EN),
43              .RX_EVEN            (RX_EVEN),
44              .rx_code_group          (rx_code_group),
45              .TX_code_group          (TX_code_group),
46              .RX_DV              (RX_DV));
47
48  endmodule
49
```

Figura 4: Intancias en el modulo `testbench.v`

A través de este esquema de instanciación y pruebas, el diseño se verifica, garantizando que cada componente realice su función designada y que el sistema en conjunto opere de manera efectiva y eficiente, conforme a los estándares establecidos.

1.2. Módulo Receptor

El módulo receptor es el modulo encargado de verificar el correcto patrón de envio de datos (I/S/D/D/D/D/T/R/I) y en la decodificación de 10 a 8 bits.

1.2.1. Diagrama de Estados

El diagrama de estados para el módulo receptor se ha completado. Este diagrama proporciona una representación visual detallada del funcionamiento del módulo en distintas situaciones, sirviendo como una guía para la implementación del código.

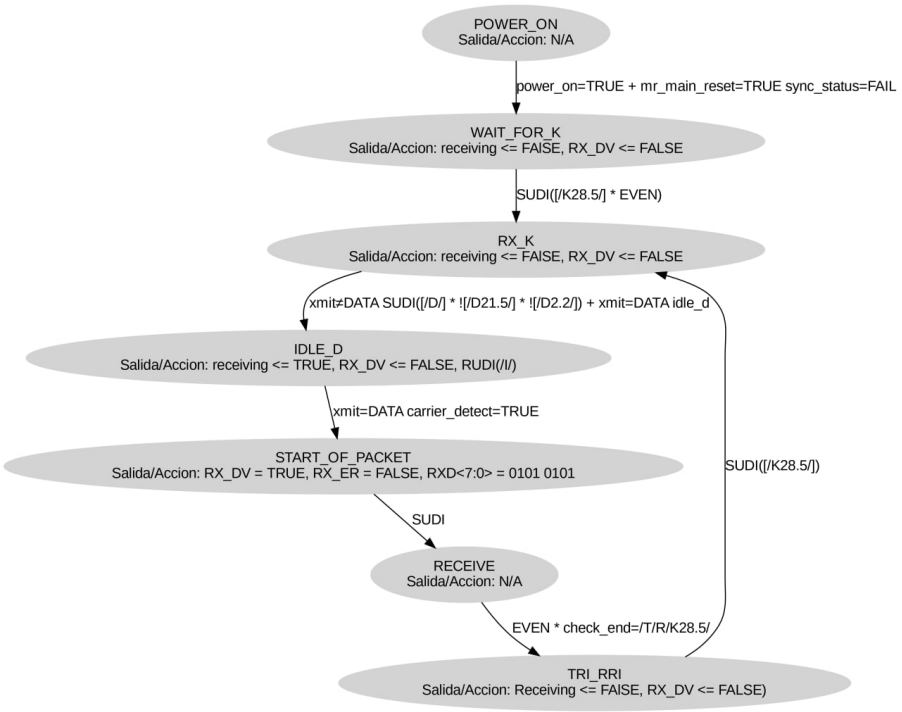

Figura 5: Diagrama General del PCS

- **WAIT_FOR_K:** El módulo espera la señal 'K'. Mientras se encuentra en este estado, no está recibiendo datos (*receiving = FALSE*), y las señales *RX_DV* y *RX_ER* están inactivas. La acción asociada es la emisión de la secuencia de sincronización *SUDI([/K28.5/] * EVEN)*.

- **RX_K:** Aquí el sistema ha detectado la señal 'K'. Continúa sin recibir datos (*receiving = FALSE*), y tanto *RX_DV* como *RX_ER* permanecen inactivos. La salida en este estado es la secuencia *SUDI([/D21.5/] + [/D2.2/])*.

- **IDLE_D:** Es un estado de espera activa. El sistema está listo pero no recibe datos activamente, emitiendo una señal de reposo *RUDI(/I/)*.

- **START_OF_PACKET:** Este estado indica el inicio de la recepción de un paquete de datos. *RX_DV* se activa y *RX_ER* se desactiva, con *RXD¡7:0¿* configurado a 0101 0101 para marcar el comienzo del paquete.

- **RECEIVE:** El sistema está en el proceso de recibir datos, sin acciones o salidas definidas, sugiriendo un enfoque en la recepción y procesamiento interno de los datos.

- **TRI_RRI:** Estado para concluir la recepción de datos y prepararse para recibir una nueva señal de inicio. Este estado señala el fin de un paquete y el sistema se prepara para volver al estado de *RX_K* en anticipación de nuevos datos.

1.2.2. Pruebas

Plan de Pruebas para el Receptor:

El plan de pruebas para el módulo receptor se diseñó para asegurar la correcta decodificación y sincronización de las señales recibidas. Se efectúan pruebas individuales para cada submódulo y pruebas integradas para evaluar el sistema en conjunto.

Pruebas Individuales: Cada componente del receptor, incluyendo la lógica de decodificación y sincronización, es probado de manera aislada para confirmar su funcionamiento adecuado:

1. Inicialización de la unidad de recepción con valores predeterminados y observación de la respuesta a la ausencia de señales entrantes.

2. Aplicación de señales de prueba conocidas y comparación de la salida decodificada con los valores esperados para confirmar la correcta operación de la decodificación 8B/10B.

3. Verificación de la sincronización de la señalización, asegurando que el sincronizador ajusta correctamente el tiempo de las señales entrantes.

Pruebas de Sistema Integradas: Una vez que se ha confirmado el funcionamiento individual de los componentes, se procede a realizar pruebas integradas:

1. Se configura el sistema completo, incluyendo el módulo `Wrapper.v`, y se inicializan todas las señales a sus valores necesarios.

2. Se activan las señales de reset y power para iniciar las máquinas de estado del receptor y se espera a que el sistema se estabilice.

3. Se simulan entradas representando IDLEs seguidos de un start of frame para comprobar la capacidad del receptor de reconocer el inicio de un frame.

4. Se introducen secuencias de code groups válidos y se verifica que el receptor los decodifique correctamente y los transmita al siguiente módulo del sistema.

5. Se simulan entradas con errores, como bits perdidos o code groups inválidos, para observar la respuesta del receptor y su capacidad de manejar errores.

6. Finalmente, se verifica la secuencia de terminación de frame para asegurarse de que el receptor maneja adecuadamente el fin de la transmisión.

1.3. Módulo Transmisor

Es el módulo que se encarga de realizar la conversión de 8 a 10 bits y darle envió al sitnonizador para que el proceso continué.

1.3.1. Diagrama de Estados

El módulo cuenta con dos maquinas de estados, una funciona para la transmisión y una para la conversión.

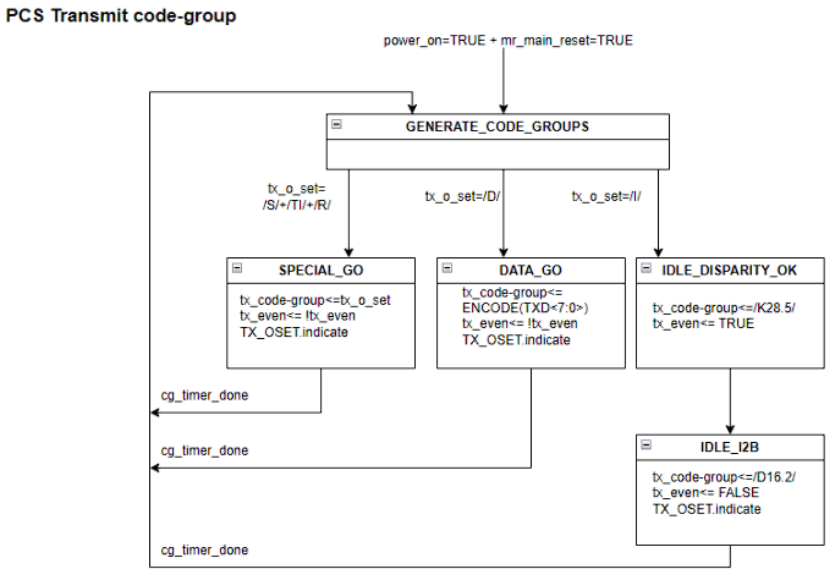

Figura 6: Diagrama del transmisor

- **GENERATE_CODE_GROUPS:** El primer estado iniciamos si se tiene reset. Y se inicia conb el tx_even en negativo.
- **SPECIAL_GO:** Si se tiene tx_o_set en S, R o T se pasa a este estado, en donde tx_code-group se establece como tx_o_set. Saliendo de este estado se pasa al estado inicial.
- **DATA_GO:** Si se tiene tx_o_set en D se pasa a este estado, en donde tx_codegroup se establece como mediante ENCODE(TXD7 0). Saliendo de este estado se pasa al estado inicial.
- **IDLE_DISPARITY_OK:** Si se tiene tx_o_set en IDEL se pasa a este estado en donde tx_codegroup se establece como K28.5.
- **IDLE_I2B:** Si se tiene tx_o_set en IDEL se pasa a este estado en donde tx_codegroup se establece como D16.2.

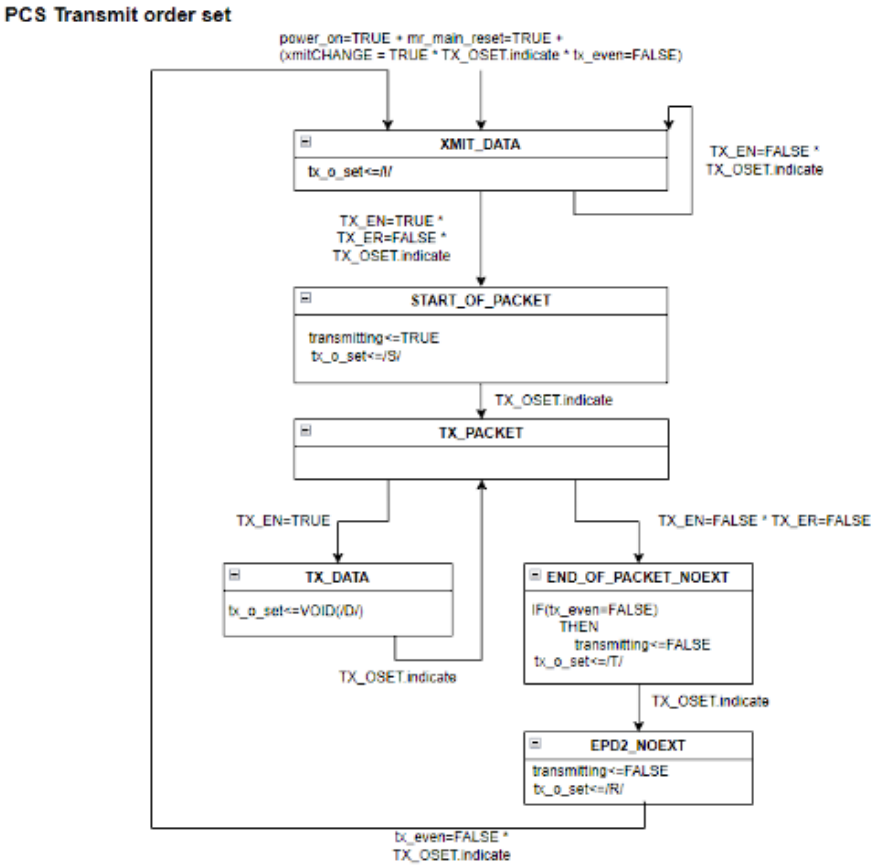

Figura 7: Diagrama del transmisor

- **XMIT_DATA_:** El primer estado iniciamos si se tiene reset. Y se mantiene en este estado siempre y cuando TX_EN sea FALSE. Una vez TX_EN sea TRUE se pasa al siguiente. Además en este estado se establece a tx_o_set como IDLE.
- **START_OF_PACKET:** Se activa transmitting en TRUE. Y el tx_o_set se pone en S.
- **TX_PACKET:** Este estado es de paso para decidir el camino. Si TX_EN es TRUE se pasa a TX_DATA. Pero siu TX_EN es FALSE se pasa a END_OF_PACKET_NOEXT.
- **TX_DATA:** Se establece tx_o_set como VOID(/D) y se regresa a TX_PACKET.
- **END_OF_PACKET_NOEXT:** Se establece tx_o_set como T, se apaga la señal transmitting y se pasa a EPD2_NOEXT.
- **EPD2_NOEXT:** Se establece tx_o_set como R, se apaga la señal transmitting y se pasa al estado incial.

1.3.2. Código

Para el caso del transmisor, se contaba con dos máquinas de estado. Para cada una de ellas se realizó un módulo en verilog y posteriormente se unieron en uno solo llamado Transmisor.v. Es importante destacar que el modo de trabajo consistió en probar cada uno de los módulos por separado. Posteriormente sé probo el módulo en el cual se realizó la unión y una vez todo funcionaba se integró con los módulos del receptor y sincronizador. Los resultados se pueden visualizar en el repositorio del Github.

1.4. Módulo Sincronizador

La tarea principal del sincronizador consiste en asegurar que el receptor esté sincronizado con la señal de transmisión, lo que permite extraer la información de manera precisa. La función esencial del sincronizador es esperar a recibir la coma y el dato de manera consecutiva 3 veces para poder realizar la sincronización de la transmisión y recepción.

1.4.1. Diagrama de Estados

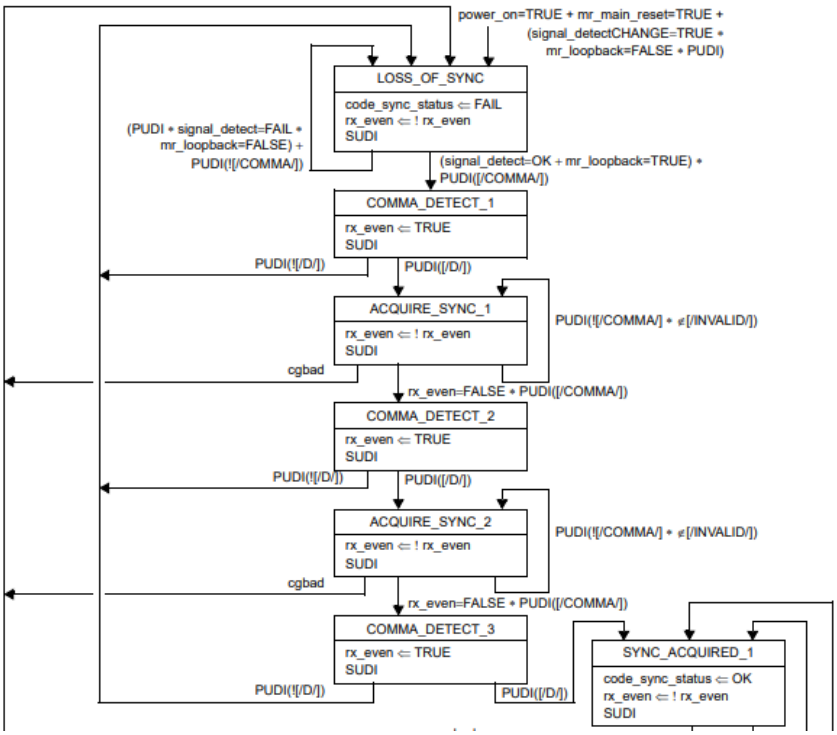

Figura 8: Diagrama del sincronizador

- **LOSS_OF_SYNC:** El primer estado iniciamos asumiendo que la sincronía está perdida, ya que se necesitan varios pasos para poder lograr la sincronía. Cuando recibe la primera coma correcta, es cuando su estado pasa al siguiente estado.
- **COMMA_DETECT_1:** Cuando recibe si primera coma se pone en este estado y se activa el rx_even, pero ahora necesitamos un D valido, que sea recibido como el siguiente, cuando se recibe se pasa al próximo estado, si no regresa a la pérdida de sincronía.
- **ACQUIRE_SYNC_1:** En este paso se confirma el recibimiento de la D por lo que confirmamos el primer traspaso del Idle completada. Esperamos una segunda coma sino reiniciamos el proceso de sincronía.
- **COMMA_DETECT_2:** Seguidamente recibimos la segunda coma y esperamos la segunda D, de lo contrario regresamos al primer estado.
- **ACQUIRE_SYNC_2:** En este paso se confirma el recibimiento de la segunda D por lo que confirmamos la segunda fase del Idle completada. Esperamos una tercera coma sino reiniciamos el proceso de sincronía.
- **COMMA_DETECT_3:** Cuando se recibe la tercera coma y se espera a la tercera D, cuando se recibe se pasa a el último estado donde ya se han recibido todas lo necesario.

- **SYNC_ACQUIRED:** En este estado es cuando se considera que la sincronía se ha logrado ya que ha pasado tres veces por esta.
- **ERROR_DETECT:** En este último estado, se llega cuando se detecta un error en la transmisión, es como el primer error que perdona pero de estos errores hará que se pierda la sincronía por completo.

1.4.2. Código

Para este código se realizaron y agregaron las entradas, salidas, variables internas, los parámetros de la COMMA y D, y por último los estados anteriormente descritos. Este programa se probó por separado y se comprobó su funcionamiento deseado. Las pruebas se realizaron para poder asegurarse que cambiara de estado cuando recibiera los IDLE y COMMAs deseados, además de que pasaría si recibiera uno no deseado.

1.5. Testbench

Este testbench instancia los módulos Tester y Wrapper para simular el comportamiento del sistema que involucra estos módulos. Las señales de entrada y salida del testbench se conectan a los puertos correspondientes de los módulos para realizar la simulación y verificar el diseño del sistema. El wrapper se encarga de encapsular y conectar el sincronizador, el receptor y el transmisor, en un sistema más grande, proporcionando una interfaz común.

```
`include "Tester.v"
`include "Wrapper.v"
module testbench;

wire [9:0]          TX_code_group;
wire                RX_DV;
wire [9:0]          rx_code_group;
wire                clock;
wire                TX_EN;
wire [7:0]          TXD;
wire                power;
wire                reset;
wire                RX_EVEN;

 // Inicializaci n para la generaci n del archivo VCD
  initial begin
        $dumpfile("resultados.vcd");
        $dumpvars(-1, WRAPPER);
  end



//Instancias
Tester TESTER(/*AUTOINST*/
              // Outputs
              .clock                    (clock),
              .reset                    (reset),
              .power                    (power),
```

```
                .TXD                    (TXD),
               .TX_EN                   (TX_EN),
               .RX_EVEN                 (RX_EVEN),
               .rx_code_group           (rx_code_group),
               .TX_code_group           (TX_code_group),
               .RX_DV                   (RX_DV));

Wrapper WRAPPER (/*AUTOINST*/
                 // Outputs
               .clock                   (clock),
               .reset                   (reset),
               .power                   (power),
                .TXD                    (TXD),
               .TX_EN                   (TX_EN),
               .RX_EVEN                 (RX_EVEN),
               .rx_code_group           (rx_code_group),
               .TX_code_group           (TX_code_group),
               .RX_DV                   (RX_DV));

endmodule
```

2. Plan de Pruebas

Primero se realizaron pruebas de cada uno de los módulos por separado para comprobar su funcionamiento. Posteriormente para las pruebas de todo el sistema se realizó primeramente la prueba general:

1. Se inicializa el sistema dándole a cada una de las señales importantes el valor necesario.

2. Luego de un tiempo se ponen las señales de reset y power en alto para dar comienzo a las máquinas de estado. Y posteriormente se apaga la señal de reset para que el transmisor comience a refeljar IDLEs.

3. Luego se pone la señal de tx enable en alto para que se comience el ciclo enviándole el code group del start al sincronizador (punto clave es que esto se hace una vez se han pasado suficientes IDLEs como para que el sincronizador comience indique que ya está sincronizado).

4. Luego se cambia el valor de TXD por code groups de la tabla de datos para que sean los datos que se quieren transmitir.

5. Se le pasan 10 datos para comprobar el correcto funcionamiento de la salida del receptor y también poder ver el funcionamiento de la paridad.

6. Por último se pone la señal de tx enable en bajo para que se finalice la transmisión y observar los code groups de T y R para observar que la trama completa de ethernet se ha completado.

A continuación se puede ver el tester que se utilizó para realizar la prueba anteriormente descrita.

2.1. Tester

```
module Tester(power, clock, reset,TXD, TX_EN, RX_EVEN, RX_DV,
rx_code_group, TX_code_group);


input wire [9:0]         TX_code_group;
input wire               RX_DV;
output reg [9:0] rx_code_group;
output reg                        clock;
output reg                        TX_EN;
output reg                        power;
output reg [7:0]       TXD;
output reg                        reset;
input wire                        RX_EVEN;


initial begin
    // Inicializaci n de las se ales
    clock = 0;
        reset = 0;
        power = 0;
        TX_EN = 0;
        TXD = 0;
    rx_code_group = 10'b0000000000;

        #20 reset = 1;
        power = 1;

    // Ciclo de reset
    #10 reset = 0;
    #65 TX_EN = 1; // Se prueba el paso a START_OF_PACKET

    #20 TXD = 8'h01; //Se prueba la tranmisi n del dato: D1_0_8

    #10 TXD = 8'h02; //Se prueba la tranmisi n del dato: D2_0_8

    #10 TXD = 8'h03; //Se prueba la tranmisi n del dato: D3_0_8

    #10 TXD = 8'h04; //Se prueba la tranmisi n del dato: D4_0_8

    #10 TXD = 8'h05; //Se prueba la tranmisi n del dato: D5_0_8

    #10 TXD = 8'h06; //Se prueba la tranmisi n del dato: D2_0_8

    #10 TXD = 8'h07; //Se prueba la tranmisi n del dato: D3_0_8

    #10 TXD = 8'h08; //Se prueba la tranmisi n del dato: D4_0_8

    #10 TXD = 8'h09; //Se prueba la tranmisi n del dato: D5_0_8
```

```
    #10 TXD = 8'h00; //Se prueba la tranmisi n del dato: D5_0_8


    #75 TX_EN = 0; // Se prueba la transici n a END_OF_PACKET_T
    //(envia (/T/),

    // despu s deber a pasar a END_OF_PACKET_R, env a /R/

    // por  ltimo  deber a volver a a enviar IDLES

    // Finalizar simulaci n
    #100 $finish;
end
always begin
    #5 clock = !clock;
end
endmodule
```

3. Instrucciones de utilización de la simulación

Para simular el programa se deben tener todos los programas y correr el makefile con make PCS, con este archivo solo se debe correr en la terminal para su simulación.

```
PCS:    Receptor.v Sincronizador.v Transmisor.v Transmisor_code.v
Transmisor_oset.v Wrapper.v Tester.v testbench.v
        iverilog -o salida testbench.v
        vvp salida
        gtkwave resultados.vcd
```

4. Ejemplos de los resultados

En esta sección se van a presentarlos principales resultados, es importante que si se quiere tener mayor detalle y analizar alguna señal que no es mencionada se puede simular el proyecto siguiendo las instrucciones de la sección anterior. Primeramente, para la parte de transmisión se tiene en las figuras 9 y 10 las principales señales.

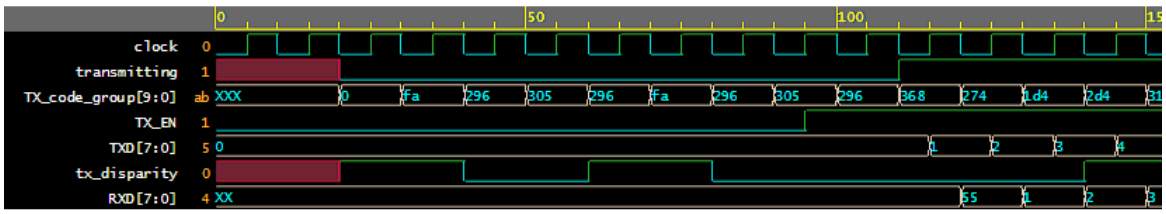

Figura 9: Resultados de Transmisión primera parte.

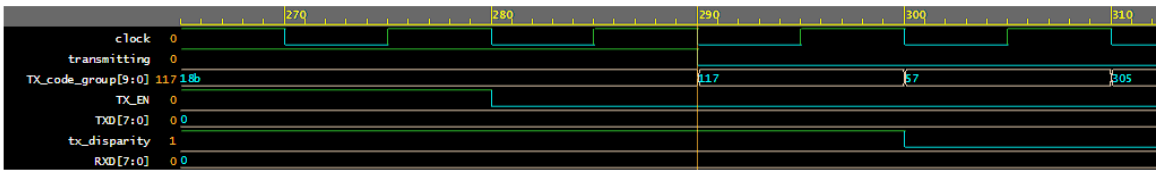

Figura 10: Resultados de Transmisión segunda parte.

Analizando un poco las señales que se presentan en la figura 9 primero se tiene el la señal de transsmitting, que se puede observar que se pone en alta cuando la senal TX_EN se pone en alto tal y como se espera. Luego se tiene la señal más relevante como resultado que es la señal de TX_code_group. Se puede observar que primeramente se ponen en 0, esto es simplemente inicialización. Luego se tiene fa que corresponde a K28.5 negativo dado que así lo establecen las reglas de paridad. Luego se tiene 296 que corresponde a D5.6(igual para los negativos como los positivos), luego viene 305 que es K28.5 positivo igual por reglas de paridad y esta secuencia se mantienen hasta que la senal de TX_EN se pone en alto, en ese momento se mantiene la secuencia por un ciclo más debido a como están establecidas las máquinas de estado. Luego se pone la señal en 368 que corresponde al code group /S/ y después se comienza a tener todos los datos. Es importante resaltar que tanto la secuencia de ethernet como las reglas de paridad se cumplen por lo que hasta este punto se puede afirmar que los resultados son positivos. Para continuar con la transmisión en la figura 10 se puede ver que la senal tx_code_group se pone en 117 que corresponde al code group /T/, luego en 57 que corresponde a /R/ y por último vuelve a la secuencia de IDLE. Todo esto pasa posteriormente de que la señal de TX_EN se ponga en 0. Se puede apreciar que todo se comporta como debería.

5. Análisis de Resultados del Receptor

Durante las pruebas del módulo receptor, se observan varias señales clave para evaluar la funcionalidad y el rendimiento del sistema. La Figura 11 ilustra las formas de onda correspondientes a una sesión de prueba estándar del receptor.

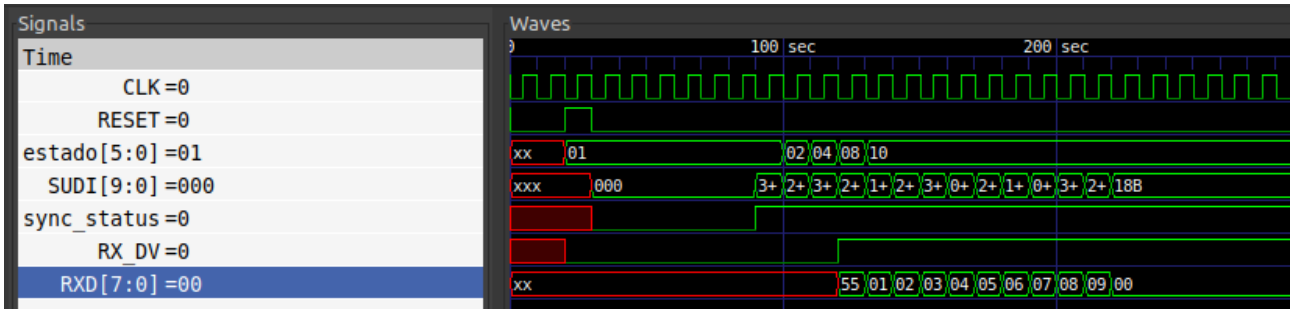

Figura 11: Formas de onda observadas en la salida del receptor

La señal `sync_status` es emitida por el sincronzador, indicando la correcta sincronización después de detectar tres IDLEs consecutivos. Esto muestra que el sincronizador está listo para transmitir datos al receptor. La activación precisa de `sync_status` es fundamental para el intercambio de datos sincronizados.

Por su parte, la señal `RX_DV` se activa tras la detección de la señal de START, lo que indica que el receptor ha comenzado a procesar los datos entrantes. La presencia y el timing correcto de `RX_DV` son cruciales para confirmar que el receptor está correctamente alineado con el flujo de datos.

Los datos codificados que llegan al receptor están representados por la señal `SUDI[9:0]`, que es procesada por el sincronizador antes de llegar al receptor. A su vez, el receptor decodifica estos datos y los presenta en la salida `RXD[7:0]`, confirmando que la decodificación de 8 bits se está llevando a cabo correctamente.

La máquina de estados del receptor es monitoreada a través de la señal `estado[5:0]`. La observación de los cambios en esta señal a lo largo de los estados esperados confirma que la lógica de control del receptor está funcionando según lo diseñado.

Las señales `CLK` y `RESET` son esenciales para la operación sincronizada del sistema. La señal `CLK` dicta el ritmo de todas las operaciones sincronizadas, y la señal `RESET` garantiza que el sistema se inicie desde un estado conocido y predecible. La inicialización y el comportamiento adecuados de estas señales son indispensables para la integridad del sistema.

El análisis de las formas de onda demuestra que el receptor está funcionando de acuerdo a lo esperado, gestionando eficazmente tanto las señales de control como los datos codificados y decodificados. Estos resultados corroboran la validez del diseño y la implementación del receptor en el sistema de comunicaciones propuesto.

6. Conclusiones y recomendaciones

- Primeramente se puede concluir que se lograron implementar 4 máquinas de estado utilizando los conocimientos obtenidos durante el curso y el estándar 803.3 para Ethernet. Fueron dos máquinas para la parte de transmisión, una para la parte de recepción y una para la sincronización. Las mismas fueron probadas individualmente y se comportan tal y como se deseaba. Y además que se logró realizar un acople entre ellas.

- En el futuro, para comprender mejor el proceso de transmisión de datos, sería beneficioso explorar más secciones del estándar, como PMA o PMD. Estas secciones abordan aspectos específicos de la capa física y proporcionarán una visión más completa y detallada de cómo los datos se codifican y se transmiten a través del medio físico. Estudiando y tomando en cuenta más partes del estándar, se podrá obtener un conocimiento más profundo del diseño y la implementación en la transmisión de datos, especialmente en el aspecto más físico.