Universidad de Costa Rica

Circuitos Digitales II

Proyecto final
Diseño de un bloque de PCS tipo 1000BASE-X

Prof. Enrique Coen Alfaro

Marco Vásquez Ovares B17032
Isaí Vargas Ovares B88263
Kevin Campos Castro B91519
Jose Pablo Eras Saborio B72704
Grupo: 04

I - 2023

Índice

1. Resumen **3**

2. Descripción arquitectónica **3**
2.1. Transmit ordered set . . . 4
2.1.1. Entradas . . . 4
2.1.2. Salidas . . . 4
2.2. Transmit code-group . . . 5
2.2.1. Entradas . . . 5
2.2.2. Salidas . . . 5
2.3. Receptor . . . 6
2.3.1. Entradas . . . 6
2.3.2. Salidas . . . 6
2.4. Sincronizador . . . 7
2.4.1. Entradas . . . 7
2.4.2. Salidas . . . 7
2.5. Diagramas de estado simplificados . . . 8
2.6. Grupos de código a usar . . . 11

3. Plan de Pruebas **11**

4. Instrucciones de utilización de la simulación **12**

5. Resultados **12**
5.1. Transmisor . . . 13
5.2. Sincronizador . . . 14
5.3. Receptor . . . 14
5.4. PCS . . . 15

6. Conclusiones y recomendaciones **16**

Referencias **17**

1. Resumen

La subcapa de Codificación Física (PCS) según la cláusula 36 del estándar IEEE 802.3 es responsable de procesar y convertir los datos de la capa superior en un formato adecuado para la transmisión a través del medio físico de la red. Esto implica operaciones como el mapeo de datos en símbolos de codificación, la inserción de bits de control y sincronización, y la sincronización del reloj en el receptor. Su objetivo es garantizar una transmisión confiable y eficiente de los datos en una red Ethernet. La subcapa cuenta con 3 bloques principales. El bloque transmisor en la subcapa de PCS se encarga de recibir los datos de entrada directamente de la capa de enlace de datos (Data Link Layer), específicamente del GMII (Gigabit Media Independent Interface), el cual es una interfaz estandarizada utilizada para la conexión entre el transmisor y receptor físico y la subcapa MAC (Media Access Control).

La sincronización es un aspecto crucial en la subcapa de PCS. Permite al receptor ajustar su reloj interno al ritmo de transmisión de los datos y asegura una recepción precisa. La sincronización se logra mediante la inserción de patrones de bits especiales en los datos transmitidos, que el receptor utiliza para sincronizar su reloj.

El bloque receptor en la subcapa de PCS se encarga de recibir los datos transmitidos a través del medio físico y recuperar la información original. Realiza operaciones como el desempaquetado de los símbolos de codificación, la detección y corrección de errores, y la extracción de los datos originales para ser entregados a la capa superior.

2. Descripción arquitectónica

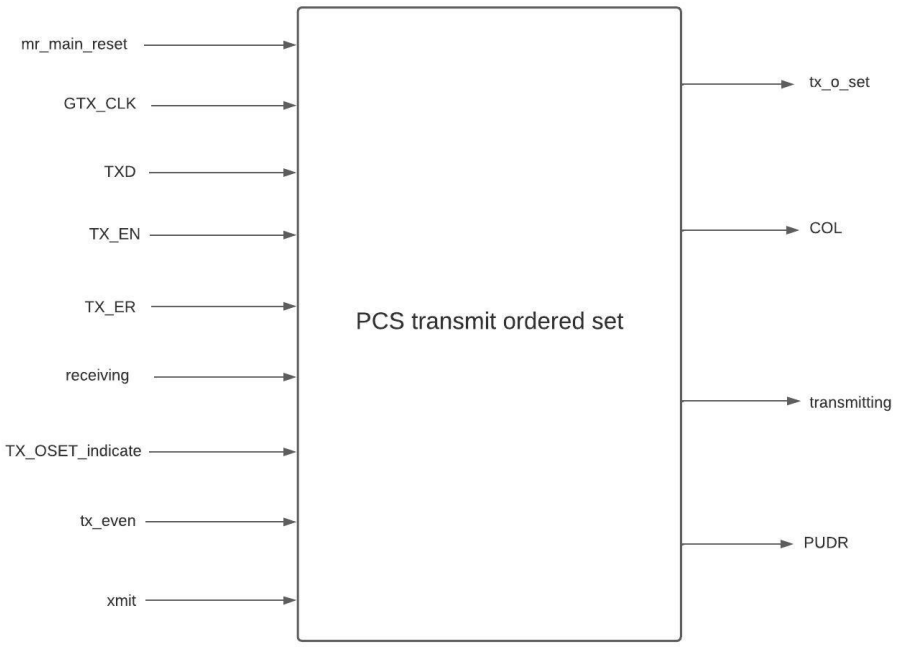

Figura 1: Transmit ordered set

2.1. Transmit ordered set

2.1.1. Entradas

- mr_main_reset: Esta señal representa el reset principal. Se utiliza para restablecer o reiniciar la lógica del módulo.
- GTX_CLK: Es la señal de reloj que sincroniza las operaciones en la máquina de estados.
- TXD: Los datos de transmisión se ingresan a través de este bus de 8 bits. Estos datos se codificarán y enviarán como parte de los conjuntos de órdenes de transmisión.
- TX_EN: La habilitación de transmisión indica si se deben transmitir los datos presentes en TXD.
- TX_ER: Esta señal representa un error de transmisión. Puede ser utilizada para indicar la detección de un error en la transmisión.
- receiving: Esta señal indica si el módulo está en modo de recepción.
- TX_OSET_indicate: Indica si se debe generar un conjunto de órdenes de transmisión.
- tx_even: Es la señal de paridad utilizada para los datos de transmisión. Puede utilizarse para realizar un control de paridad en los datos transmitidos.
- xmit: Esta señal de 3 bits representa el estado actual de transmisión en la máquina de estados.

2.1.2. Salidas

- tx_o_set: Esta salida de 7 bits representa el conjunto de órdenes de transmisión generado. Los conjuntos de órdenes de transmisión son secuencias de bits utilizadas para propósitos específicos, como el control de enlace y la sincronización.
- COL: Esta salida indica si se ha detectado una colisión durante la transmisión. Una colisión ocurre cuando dos o más dispositivos intentan transmitir datos al mismo tiempo en una red Ethernet.
- transmitting: Esta salida indica si hay una transmisión en progreso.
- tx_code_group: Es una salida de 10 bits que representa el grupo de código de transmisión. Este grupo de código puede ser utilizado para identificar y distinguir diferentes tipos de conjuntos de órdenes de transmisión.

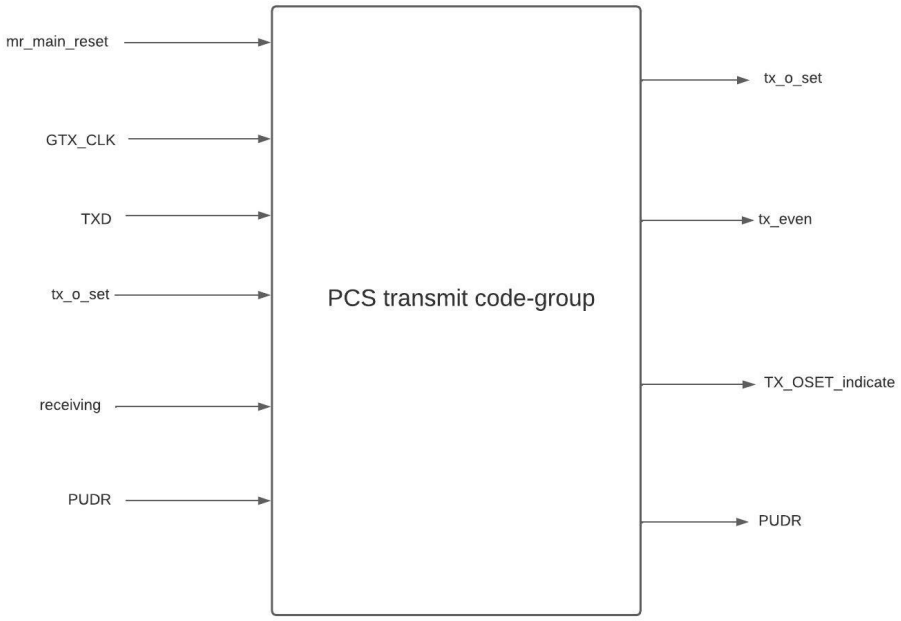

Figura 2: Transmit code-group

2.2. Transmit code-group

2.2.1. Entradas

- mr_main_reset: Señal de reinicio principal. Esta entrada se utiliza para reiniciar la máquina de estados y restablecer su estado interno.
- GTX_CLK: Reloj de transmisión. Esta entrada proporciona el pulso de reloj que sincroniza las operaciones dentro del módulo.
- tx_o_set: Conjunto de salida de transmisión. Esta entrada de 7 bits representa el conjunto de salida que se va a transmitir.
- TXD: Datos de transmisión. Esta entrada de 8 bits contiene los datos que se van a transmitir.

2.2.2. Salidas

- tx_even: Bit de paridad de transmisión. Esta salida representa el bit de paridad asociado con la transmisión actual. Su valor puede ser 0 o 1.
- TX_OSET_indicate: Indicador de conjunto de salida de transmisión. Esta salida indica si el conjunto de salida de transmisión ha sido reconocido y se ha generado el código de grupo correspondiente. Su valor puede ser 0 o 1.
- PUDR: Código de grupo de transmisión. Esta salida de 10 bits representa el código de grupo generado para la transmisión actual.

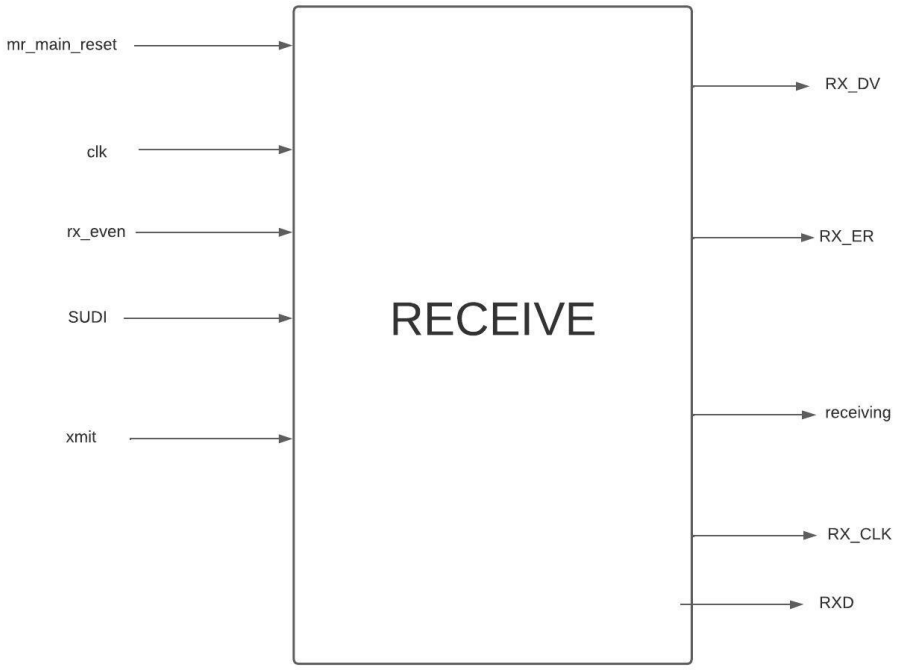

Figura 3: Receptor

2.3. Receptor

2.3.1. Entradas

- mr_main_reset: Señal de reinicio principal. Esta entrada se utiliza para reiniciar la máquina de estados y restablecer su estado interno.
- clk: Reloj. Esta entrada proporciona el pulso de reloj que sincroniza las operaciones dentro del módulo.
- rx_even: Bit de paridad de recepción. Esta entrada indica si el bit de paridad de los datos recibidos es par (1) o impar (0).
- SUDI: Datos recibidos. Esta entrada de 10 bits contiene los datos recibidos.
- xmit: Conjunto de salida de transmisión. Esta entrada de 3 bits indica el conjunto de salida de transmisión esperado.

2.3.2. Salidas

- RX_DV: Validación de recepción. Esta salida indica si se ha recibido un conjunto de datos válido. Su valor puede ser 0 o 1.
- RX_ER: Error de recepción. Esta salida indica si se ha producido un error en la recepción de datos. Su valor puede ser 0 o 1.
- receiving: Indicador de recepción. Esta salida indica si el receptor está en estado de recepción de datos. Su valor puede ser 0 o 1.

- RX_CLK: Reloj de recepción. Esta salida es una versión invertida del reloj de entrada clk.
- RXD: Datos recibidos. Esta salida de 8 bits representa los datos decodificados y recuperados de la transmisión recibida.

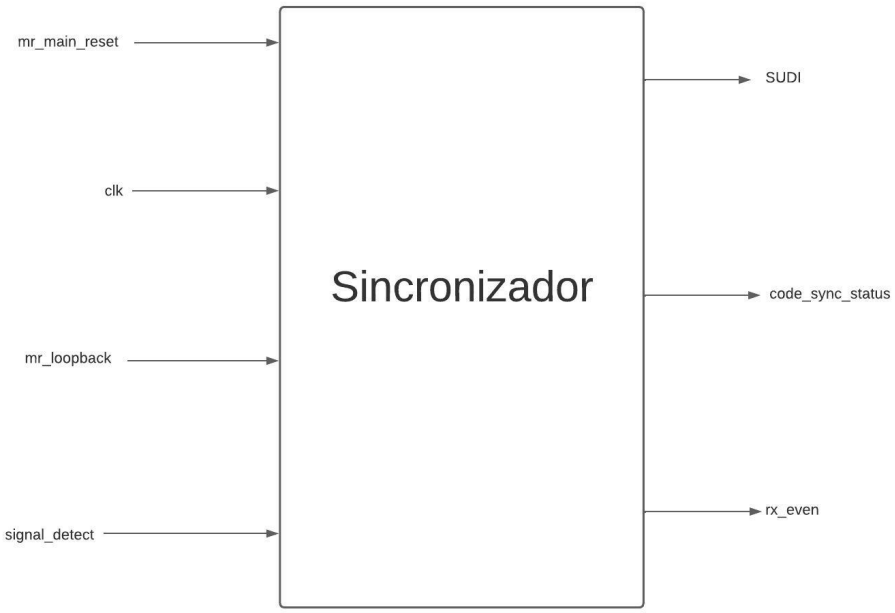

Figura 4: Sincronizador

2.4. Sincronizador

2.4.1. Entradas

- clk: Entrada de reloj. Esta entrada proporciona el pulso de reloj que sincroniza las operaciones dentro del módulo.
- mr_main_reset: Señal de reset principal. Esta entrada se utiliza para reiniciar la máquina de estados y restablecer su estado interno.
- mr_loopback: Señal de bucle de retroalimentación. Esta entrada indica si el bucle de retroalimentación está habilitado.
- signal_detect: Señal de detección de señal. Esta entrada indica si se ha detectado una señal.
- PUDI: Código de grupo de 10 bits proveniente del receptor. Esta entrada representa el código de grupo recibido.

2.4.2. Salidas

- SUDI: Código de grupo de 10 bits sincronizado. Esta salida representa el código de grupo sincronizado.

- code_sync_status: Estado de sincronización del código. Esta salida indica si se ha logrado la sincronización del código. Su valor puede ser 0 o 1.
- rx_even: Indicador de paridad del código recibido. Esta salida indica si el código de grupo recibido tiene una paridad par (1) o impar (0).

2.5. Diagramas de estado simplificados

Esta sección está dedicada a los diagramas de flujo simplificados en los que se va a basar el proyecto, los mismos son utilizado como referencia, además no incluye los estados opcionales mencionados en clase, sino solo la base, si se dispone de tiempo adicional se tiene pensado implementar algunos.

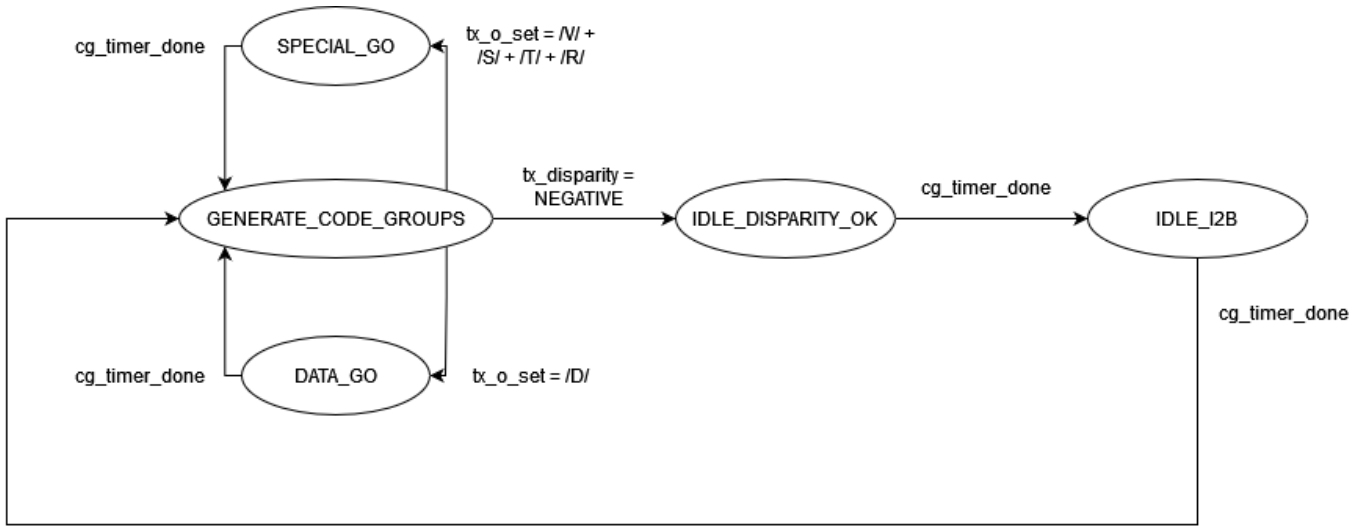

Figura 5: Diagrama de estados simplificado del transmisor, transmisión de grupo de código

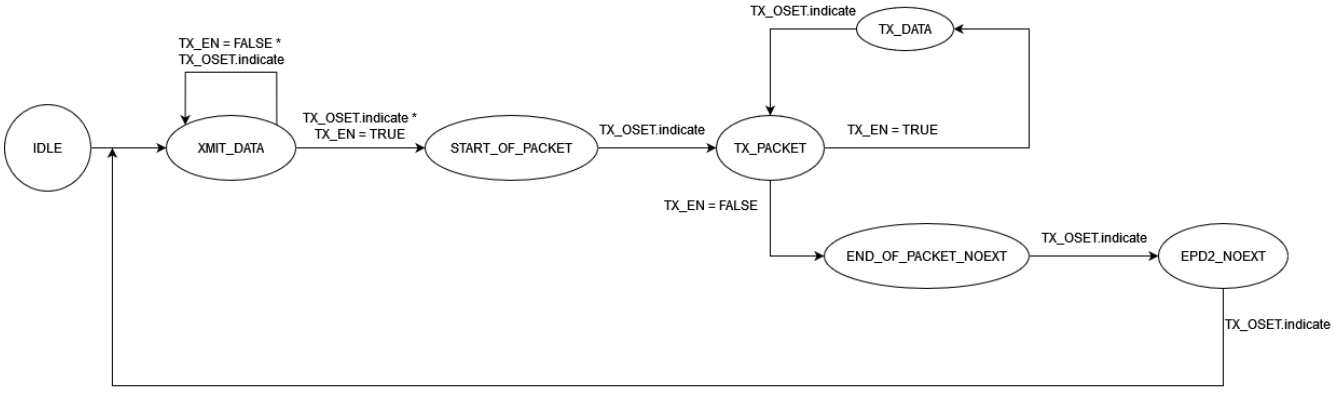

Figura 6: Diagrama de estados simplificado del transmisor, transmisión conjunto ordenado.

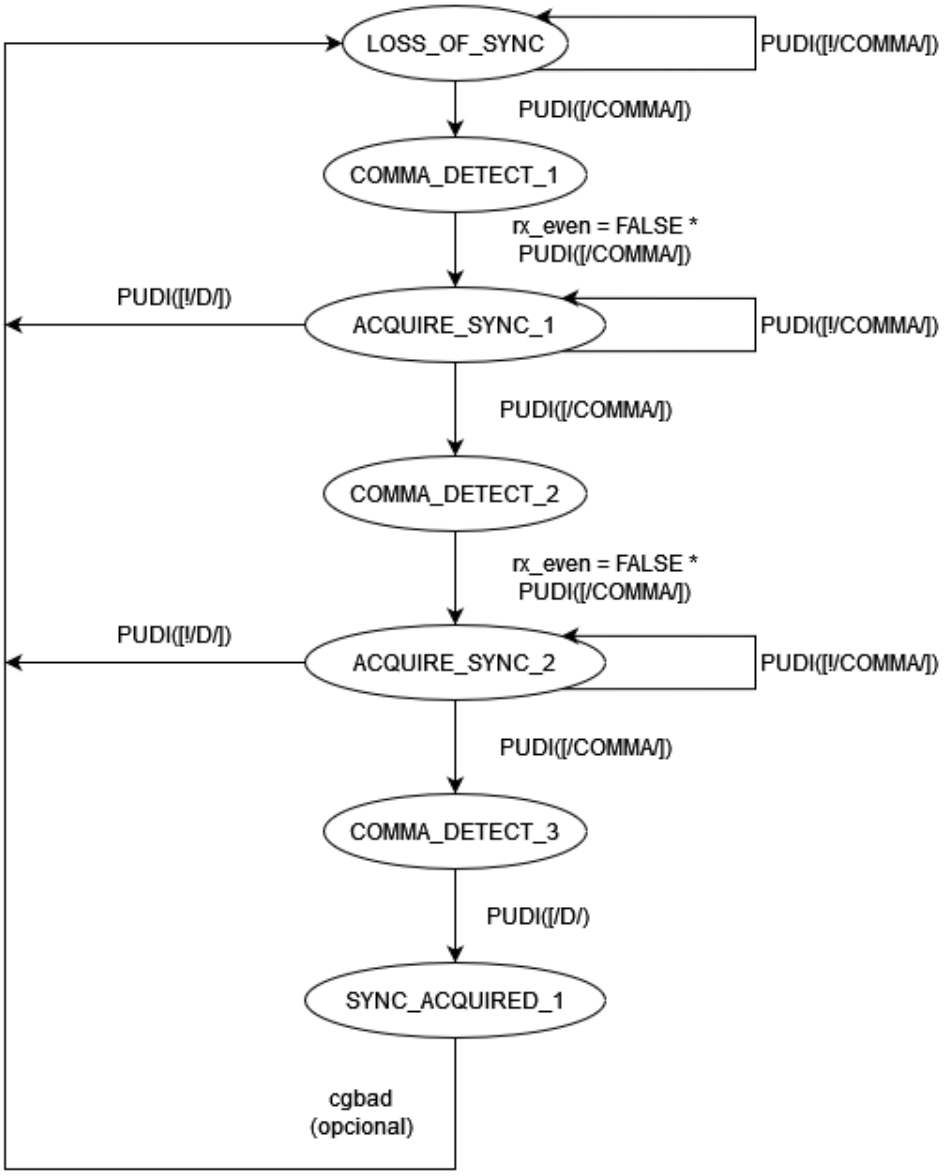

Figura 7: Diagrama de estados simplificado del bloque de sincronización.

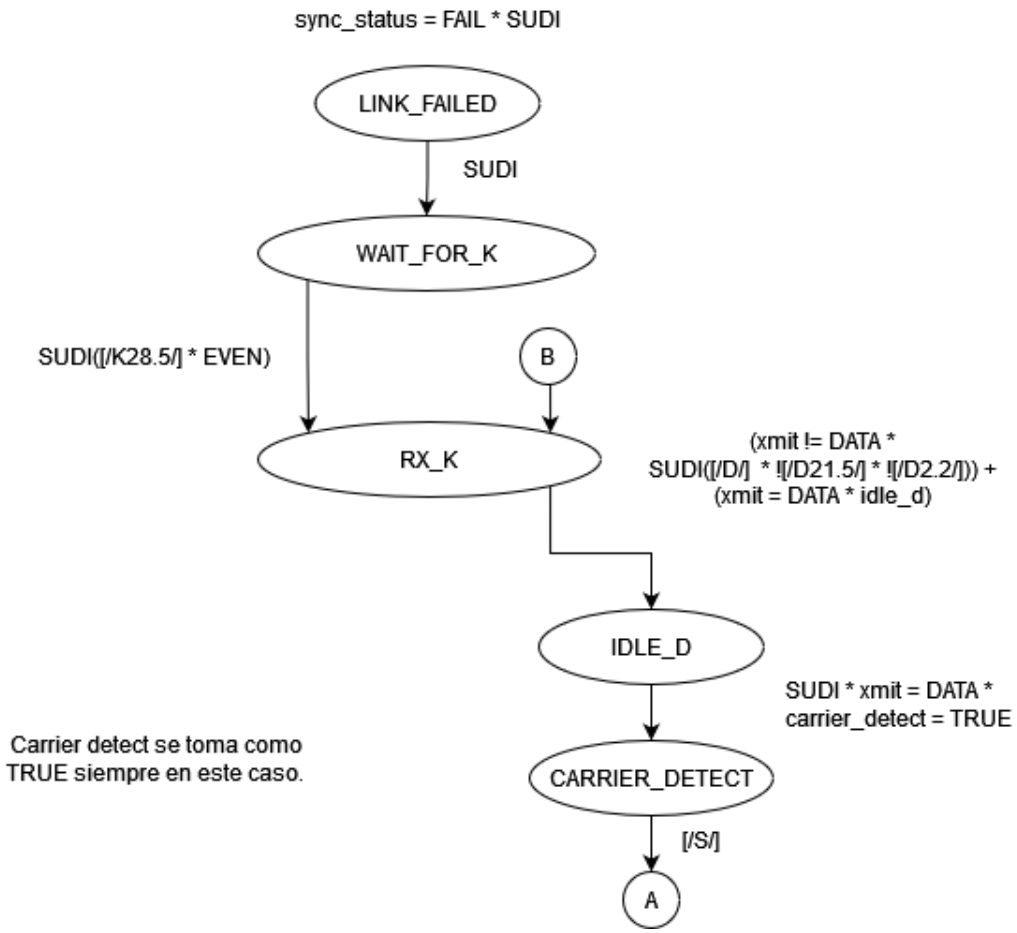

Figura 8: Diagrama de estados simplificado del receptor, parte a.

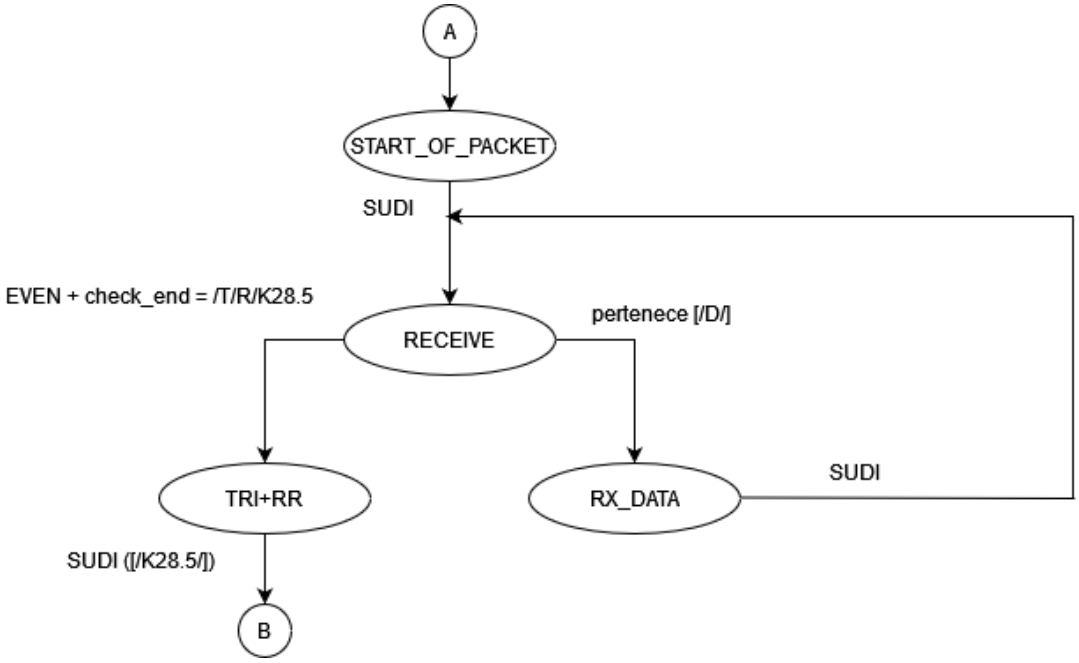

Figura 9: Diagrama de estados simplificado del receptor, parte b.

2.6. Grupos de código a usar

Para los grupos de código especiales, se incluyeron todos los de la clásula 36, sin embargo los más importantes para mostrar el comportamiento adecuado son lo siguientes:

Code Group Name	Valor en Hex	Current RD
/D16.2/	1B5	011011 0101
/K27.7/	97	001001 0111
/K23.7/	57	000101 0111
/K29.7/	117	010001 0111
/K28.5/	305	110000 0101

Los grupos de código a utilizar son los siguientes:

Code Group Name	Current RD -
D0.0	100111 0100
D1.0	011101 0100
D2.0	101101 0100
D3.0	110001 1011
D2.2	101101 0101
D16.2	011011 0101
D26.4	010110 1101
D6.5	011001 1010
D21.5	101010 1010
D5.6	101001 0110

3. Plan de Pruebas

En cuanto a la transmisión de datos, se envían una serie de bytes, representados por los valores hexadecimales 01, 02, 03, 50, 9A, 03, A6, 02, 01 y C5. Cada uno de estos bytes se asigna a la señal TXD con un retardo de 2 unidades de tiempo entre ellos. Estos bytes se transmiten secuencialmente como parte del proceso de transmisión de datos.

El bloque inicial del módulo tester_SYNC marca el inicio de la simulación y se encarga de la inicialización de varias variables.

En primer lugar, se establece clock en 1, lo que significa que se inicializa la señal de reloj en un estado alto.

A continuación, se desactiva el reset principal (mr_main_reset) estableciéndolo en 0. Esto indica que el sistema no está en estado de reset.

Después de un retardo de 2 unidades de tiempo, se activa el reset principal. Esto se logra asignando el valor binario 1 (1'b1) al registro mr_main_reset. El retardo de 2 unidades de tiempo se logra utilizando #2 antes de la asignación.

Luego, se activa la detección de señal (signal_detect) estableciendo su valor en 1. Esto indica que el sistema está detectando una señal.

A continuación, se activa el bucle de retroalimentación (mr_loopback) estableciéndolo en 1. Esto implica que los datos se están enviando de nuevo al mismo sistema.

Se asigna el valor 0011111010 al registro rx_code_group. Este valor representa el código K28.5, que se interpreta como el inicio de un paquete de datos. El sistema simula recibir este código en el momento de la inicialización.

Después de un retardo de 2 unidades de tiempo, se asigna el valor 0110110101 al registro rx_code_group. Este valor representa el código D16.2, que se interpreta como datos. Esto simula la recepción de datos después de un tiempo determinado.

Estos pasos se repiten varias veces con diferentes valores de rx_code_group, lo que simula la recepción de diferentes códigos de grupo en secuencia. Esto permite probar el comportamiento del sistema ante diferentes datos recibidos.

Se agrega un retardo de 50 unidades de tiempo utilizando #50. Este retardo proporciona un tiempo adicional antes de que finalice la simulación.

Finalmente, $finish se utiliza para finalizar la simulación. Esto indica que no se realizarán más operaciones y se terminará el proceso de simulación.

4. Instrucciones de utilización de la simulación

A continuación se presentan una lista de pasos para el correcto funcionamiento del código realizado:

1. Clonar el repositorio de git https://github.com/ecoenucr/Grupo4_IS23.git.

2. Dentro del repositorio moverse a la carpeta *src*.

3. Dentro del la carpeta *src* ejecutar el comando:

```
make
```

   El cual contiene lo siguiente:

```
make:
        iverilog -o out tb_PCS.v
                vvp out
              gtkwave tb_PCS.vcd
```

4. Al abrirse el programa *GTKWAVE*, se seleccionan las entradas y salidas de interés.

5. Resultados

La siguiente sección del reporte consiste en los resultados finales obtenidos una vez concluido el trabajo, para ello se basa en la siguiente figura 10 que era el objetivo final del trabajo:

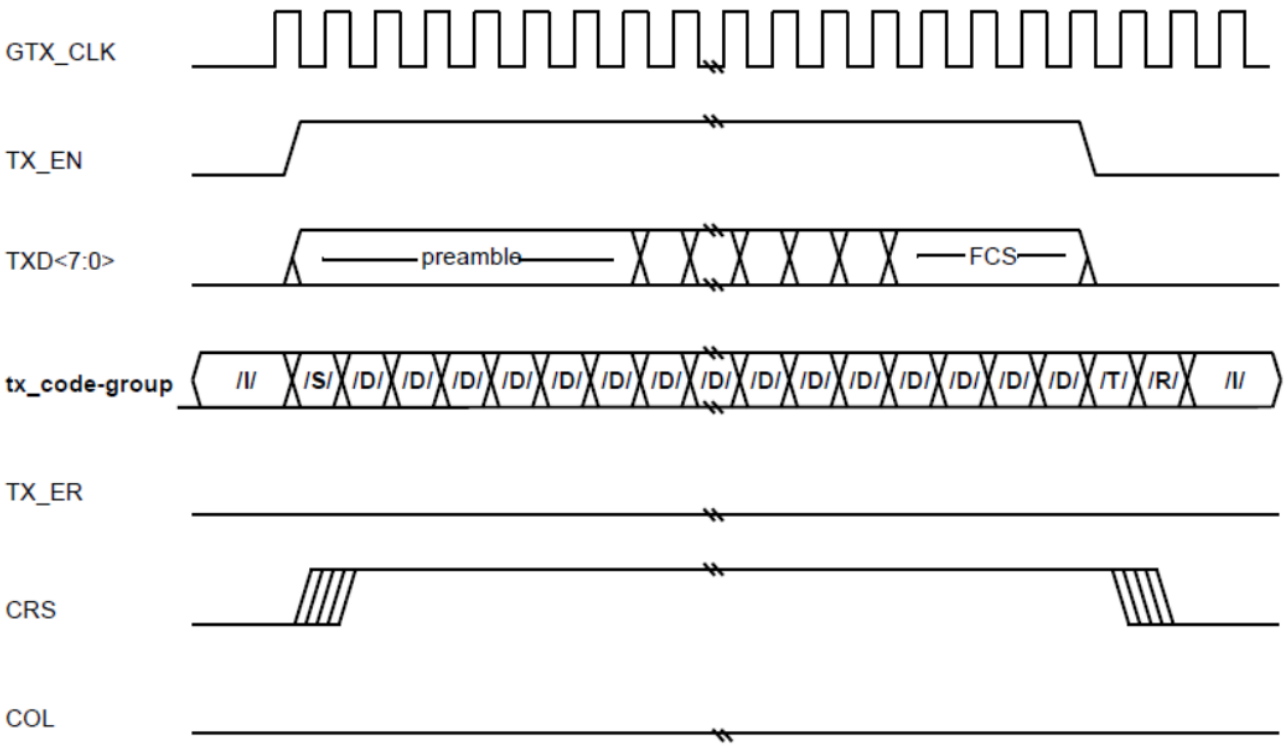

Figura 10: Trama completa de ethernet.

5.1. Transmisor

Se procede a explicar los resultados obtenidos de la simulación del transmisor. Lo primero que se espera es que en el transmisor inicie con una condición de IDLE, cuando se quiere empezar la comunicación se dé la condición de inicio (/S/) y una vez dado esto, se empiezan a mandar datos, una vez enviado los datos, se da una condición de terminación dado por la secuencia /T/R/K28.5/, con esto se da la conclusión de un tramo. En la figura 11, lo primero que se observa es la señal de PUDR, esto son los datos codificados que está enviando el transmisor, al inicio se tiene una secuencia de 305, 245 (esto es hexadecimal), esto en binario es 110000 0101 y 100100 0101 que son los caracteres de K28.5 (comma) y D16.2, esta combinación hace referencia el código de IDLE 2 especificado en la clausula, por lo que este estado de "stand by" funciona de forma correcta. El carácter de inicio de paquete está dado por /K27.7/, el valor obtenido en la simulación para iniciar es el 097, que en binario es 001001 0111, observando la tabla se tiene que esto es el valor de /K27.7/, por lo que se da la condición de inicio de paquete, es decir, que se va a transmitir datos. Un ciclo después de que se da el inicio se empieza a mandar datos, es por esto que la señal de TXD pasa a tener valores de 00 a tomar valores de datos de transmisión. Una vez finalizada la transmisión se tienen que dar la secuencia de finalización de paquete, esto está dado por el código de /T/ y /R/, que están dados por /K29.7/ seguido de /K23.7/, analizando las tablas esto son los valores de 010001 0111 y 000101 1110, esto en hexadecimal se representa como 117 y 57, por lo que la secuencia de finalización de tramo se dio de forma correcta. Una vez terminado la secuencia de finalización se observa que vuelve a aparecer la secuencia de 305 y 245, que es la condición de idle, por lo que el tramo ya se envió, con este análisis se verifica el funcionamiento correcto del transmisor.

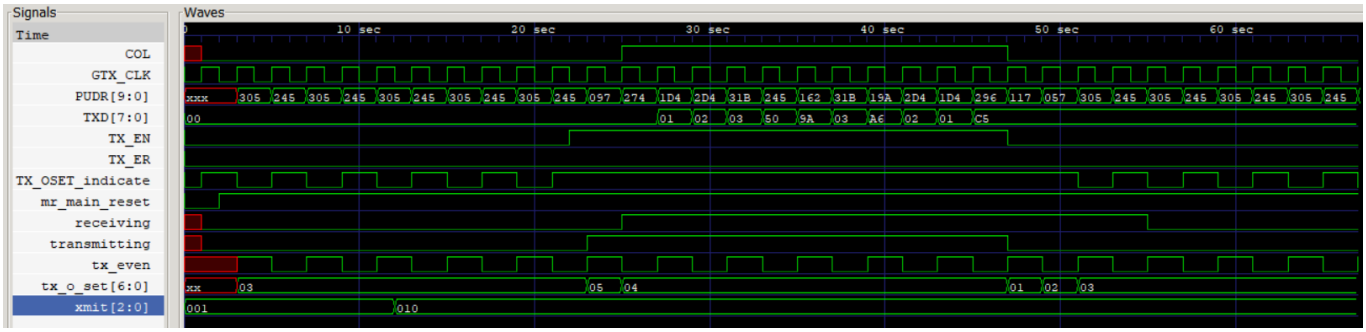

Figura 11: Trama completa de ethernet.

5.2. Sincronizador

Para esta sección lo que se espera es mandar la secuencia de /COMMA/D/, /COMMA/D/, /COMMA/D/ y alcanzar el estado de sincronizado, esto se debe a que se trabaja sin errores. Para corroborar este resultado se observa la figura 12, las señales más importantes son las de PUDI y estado, en este caso el estado de sincronización en hexadecimal es 0006. Recordando el análisis del transmisor, el código de comma (K28.5) está dado por el valor hexadecimal de 305, el valor de dato a transmitir es el D16.2 que en hex es 1B5, analizando las ondas, se tiene que la señal de code_sync_status se pone en alto un ciclo de reloj después de haber recibido 3 commas y que el estado alcanzado es el 0006 y que además, se mantiene ahí ya que la sincronización se alcanzó. Con esto se verificó el comportamiento correcto del sincronizador.

Figura 12: Trama completa de ethernet.

5.3. Receptor

En el caso del receptor, lo que sucede es que los datos recibidos no son necesariamente los que le van a llegar a la capa de PCS, sino que primero, se tiene que alcanzar la condición de inicio, para

que esto suceda las condiciones que se tienen que dar es que una vez llegado al estado de carrier detect y llega el código /S/ que es la condición de inicio (explicado en el transmisor como /S/, en hex como 097 y en la tabla es el código /K27.7/), la señal de "receiving" se pone en alto, esto es cuando el valor de SUDI es 097. Después de que se da la condición de inicio se empiezan a enviar los datos correspondientes codificados (de 8 bits, es decir, esto es igual a RXD que sale del PCS), si se analiza la figura 13 los datos recibidos son los valores después del valor de SUDI = 097. Una vez enviado el tramo deseado se tiene que dar la condición de que ya no se están recibiendo datos, esto se identifica por medio de la secuencia /T/R/K28.5/, como se había mencionado en la sección de transmisor, esto es representado por los valores hexadecimales de 117, 57 y 305 respectivamente, es por esto que una vez recibida esta secuencia la señal de receiving se pone en bajo. Con esto se concluye el funcionamiento correcto del receptor.

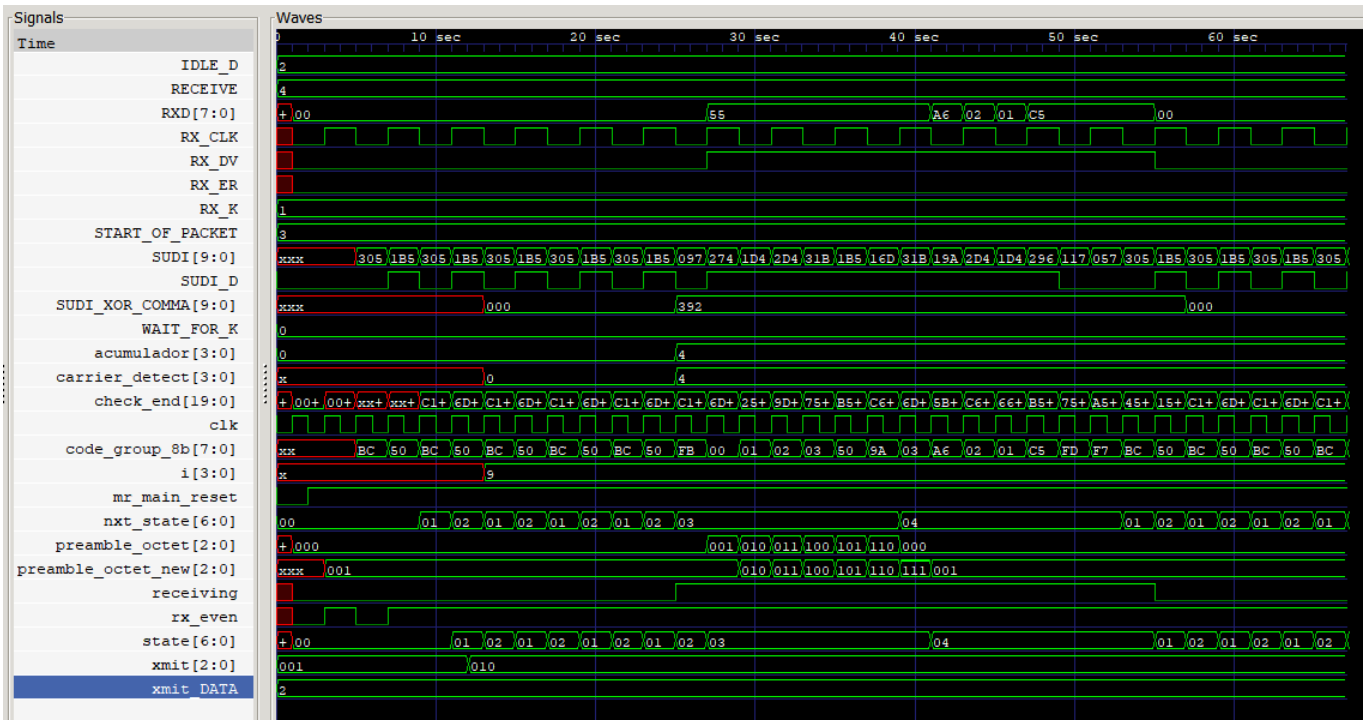

Figura 13: Trama completa de ethernet.

5.4. PCS

Por último, en la etapa de PCS, debiera de llegar el mensaje original enviado en el transmisor ya decodificado, si se observa la figura 11, el mensaje enviado es 01, 02, 03, 50, 9A, 03, A6, 02, 01, C5 y analizando la figura 14 se observa que esta es la secuencia recibida. Con esto se verifica que el funcionamiento de la implementación de la capa PCS de la cláusula 36 del estándar IEEE 802.3 se hizo de forma correcta.

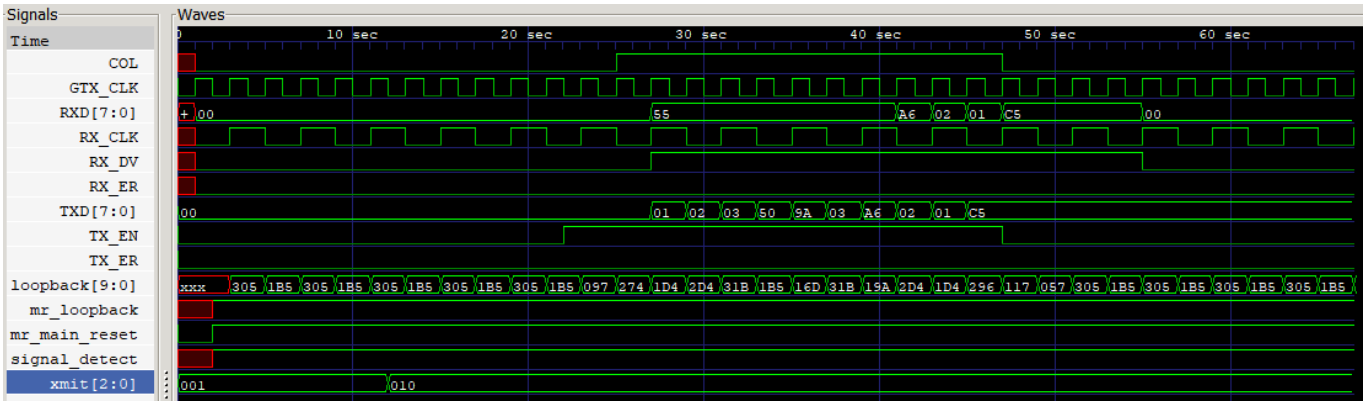

Figura 14: Trama completa de ethernet.

6. Conclusiones y recomendaciones

Siendo que se implemento la clausula 36 del estándar IEEE 802.3, que se refiere a la especificación de interfaces físicas para transmitir datos Ethernet, obteniendo resultados similares a lo descrito en la clausula en el envió de una trama de datos, en términos generales se tuvo una implementación exitosa en los módulos de transmisión y recepción, aunque se tuvieron varias complicaciones con el modulo de sincronización, sin embargo se logro entender el funcionamiento del estándar y cumplir con los requerimientos mínimos.

- **Oportunidades de mejora:** A pesar de los buenos resultados, se tiene mucho de donde mejorar, esto ya que no se implemento la clausula completa, por lo que aspectos como la detección de errores seria interesante implementarlos y analizar el comportamiento ante casos de envíos de datos mas reales y no idealizados.

- **Problemas de integración:** Se tuvieron muchas complicaciones a la hora de unir el trabajo realizado por cada integrante, ya que a pesar de seguir lo estipulado en la clausula se tuvieron tienen pequeñas variantes en los nombres de los módulos o variables, incluso se tuvieron complicaciones con el sistema operativo sobre el que se trabaja, algunos integrantes trabajan en Linux y otros en Windows, por lo que se tuvieron algunos errores en la compilación, pero fueron corregidos.

- **Importancia del estándar:** Utilizar en las variables la convención definida en el estándar de trabajo, esta es una de las recomendaciones más importante y que a largo plazo puede disminuir la cantidad de conflictos, y más cuando se trabaja entre varias personas.

- **Aplicación practica:** El proyecto ha dado la oportunidad de aplicar lo aprendido en el estudio de la cláusula 36 del estándar IEEE 802.3 y la programación en Verilog en un contexto práctico. Fortaleciendo la comprensión de los conceptos y principios fundamentales y adquiriendo conocimiento valioso para el desarrollo profesional en el campo de las redes de comunicación.

- **Trabajo efectivo:** La finalización del proyecto es muestra de una buena metodología de trabajo, lo que ha permitido avanzar en el desarrollo del proyecto en equipo, sirviendo esta como referencia para futuros trabajos en equipo. Remarcando la importancia de la comunicación y colaboración en el equipo.

- **Importancia de la documentación del código:** Siendo que nos dividimos el trabajo entre los integrantes, se debe también entender la lógica plateada por cada integrante, para esto es esencial añadir documentación en el código, esto principalmente para resolver bugs de compatibilidad.

Referencias

1. *IEEE Standard for Ethernet (IEEE Std 802.3™-2018)* . IEEE Computer Society, 2018.