Universidad de Costa Rica

Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0523 - Circuitos Integrados Digitales
II ciclo de 2024

Informe de Proyecto Final
Capa PCS del protocolo Ethernet

Profesor:
Enrique Coen Alfaro

Estudiante:
Gabriel Siles Chaves - C17530
Jun Hyun Yeom Song - B17326
Jorge Loría Chaves - C04406

Grupo 01 - Subgrupo 3

30 de noviembre 2024

Índice

1. Resumen

El propósito de este proyecto consiste en la implementación de la capa PCS (Physical Coding Sublayer) del protocolo Ethernet, basada en el estándar IEEE 802.3, Cláusula 36, diseñada para la operación de sistemas 1000BASE-X. Para fines de este proyecto esta capa esta compuesta por tres módulos principales: **Transmisor**, **Sincronizador** y **Receptor**. Cada uno de estos módulos fue implementado de forma conductual, utilizando el lenguaje de descripción de hardware Verilog (HDL). Estos módulos se estructuran de manera que sigan las máquinas de estados estipuladas en el estándar para poder tener una comunicación fiable con el GMII.

2. Descripción arquitectónica

2.1. Transmisor

El transmisor se encarga de codificar y enviar los datos de manera eficiente, este recibe del GMII 1 byte (8 bits) por medio de la entrada *TXD* y una entrada que inicia la transmisión *TX_EN*, teniendo una salida de 10 bits, *tx-ordered-set*. El bloque del transmisor se puede observar en la Figura **??**.

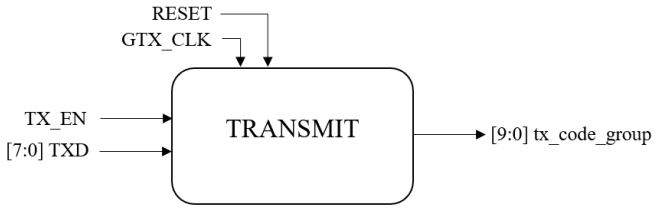

Figura 1: Diagrama de bloques del transmisor.

Este módulo cuenta con dos máquinas de estados que están dentro del mismo archivo de *transmit.v*. Para la primer máquina de estado se manejan los 8 bits y el inicio de transmisión que define el GMII por medio de la entrada TX_EN, el módulo *transmit-ordered-set* es el encargado de clasificar que tipo de dato se debe codificar al recibir el enable, realizando la transmisión efectiva. Además es importante tener en consideración que permanece en Idle en espera del inicio de transmisión, a continuación se puede observar el diagrama de estados del primer módulo **transmit-ordered-set**.

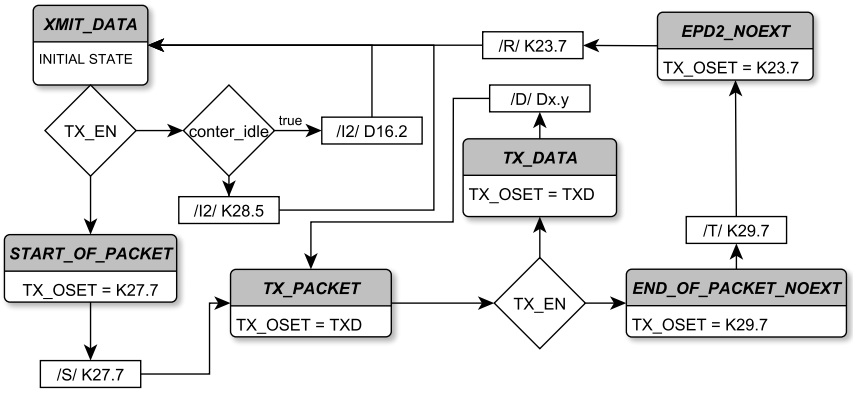

Figura 2: Diagrama ASM del módulo *transmit-ordered-set*

Para este módulo se tiene 6 estados tal como se observa en la Tabla 2:

Tabla 1: Estados del módulo del transmit-ordered-set.

Estado	Código	Descripción
XMIT_DATA	00 0001	Generar Idles hasta recibir TX_EN
START_OF_PACKET	00 0010	Iniciar transmisión con/S/
TX_PACKET	00 0100	Mantener estado de envío de /D/
TX_DATA	00 1000	Envía TXD recibido del GMII
END_OF_PACKET_NOEXT	01 0000	Finaliza transmisión con /T/
EPD2_NOEXT	10 0000	Envía Carrier /R/

Como se ha mencionado previamente la primer máquina de estados recibe los 8 bits y reconocer los diferentes procesos de transmisión este puede recibir tanto /I/, /S/, /D/ y /T/. Este módulo tiene una salida ***TX_ OSET*** la cual será la entrada para la segunda máquina. La segunda máquina llamada ***transmit-code-group*** recibirá el TX_OSET y se asegura de calcular la disparidad con repecto al estándar IEEE 802.3 al igual de codificar los 8 bits para la salida llamada **tx_code_group**. En la Figura 3 se puede visualizar la segunda máquina de estados.

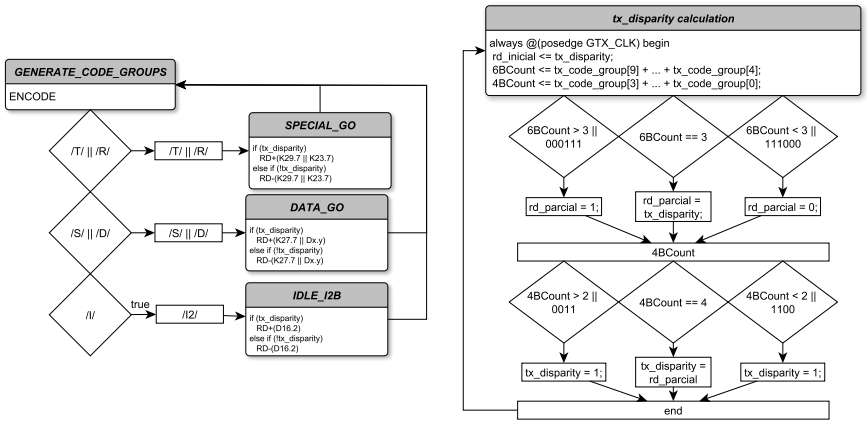

Figura 3: Diagrama ASM del módulo *transmit-code-group*

Esta maquina cuenta con un total de 4 estados para realizar la codificación de 8 bits a 10, donde el *tx_ disparity* es el de importancia al ser el encargado de indicar cual es la disparidad que se aplicará al siguiente code group, se debe tener en consideración que se calcula cada flanco del reloj, en la Figura 3 se puede observar diagrama de *tx_ disparity_ calculation*.

Seguidamente se definen las funcionalidades de todas las varibales que se utilizarán en ambas máquinas para el funcionamiento óptimo del transmit

Seguidamente se definen las funcionalidades de todas las varibales que se utilizarán en ambas máquinas para el funcionamiento óptimo del transmit

Para poder realizar la codificación de 8 bits a 10 bits se definen la tabla de la codificación en un archivo llamado *code_ group.v* donde se define el byte como su codificación para disparidad negativa y postitiva, a continuación muestra la definición de los principales code-groups que se utilizarán para comprobar la correcta función del transmisor.

Tabla 2: Estados del módulo del transmit-ordered-set.

Estado	Código	Descripción
GENERATE_CODE_GROUPS	0001	Recibe los 8 Bits e identifica el tipo de code group
IDLE_I2B	0020	Se encarga de completar el Idle enviando D16.2
SPECIAL_GO	0100	Codifica /T/ y /R/
DATA_GO	1000	Realiar codificacion de /D/

Tabla 3: Variables del módulo transmit

Tipo	Nombre	Descripción
Input	TX_EN	Inicio a la trasnmisión de un dato
Input	TXD	8 bits que se desean codificar
Input	GTX_CLK	Es el reloj del transmit
Input	RESET	Reset de todos los módulos
Output/Input	TX_OSET	Es el byte a codificar
Interna	cg_timer_done	Indica que se ha conclusión de ENCODE
Output	tx_code_group	Son los 10 bits codificados

2.2. Sincronizador

La única entrada del módulo es rx_code_group, el cual para efectos de este proyecto es igual que tx_code_group.

Las tres salidas del módulo son:

- code_sync_status
- tx_even
- SUDI

Las tres salidas son salidas de tipo Moore porque se activan ante cambios de estado.

Si se observa el diagrama ASM con detenimiento, la mayoría de transiciones de estado dependen de dos criterios:

- La categoría a la cual pertenece el dato
  - COMMA. Equivale al dato D28.5. Existen otros datos con COMMA (D28.1 y D28.7), pero se han omitido para efectos de este proyecto.
  - DATA. Su etiqueta inicia con D. COMMA es un ejemplo de DATA. Para efectos de este proyecto, se han definido 10 datos como válidos (20 en total considerando RD). Fue necesario incluir D5.6 y D28.5 para formar los IDLE.
  - VALID. Además de los 10 datos definidos como válidos, hay otros code groups especiales cuyas etiquetas inician con K. VALID reprenta entonces los DATA y los SPECIAL.

[b]0.43

```
// Data group code
// D16.2
    `define D16_2_octet     8'b101_11100
    `define D16_2_rd_neg    10'b001111_1010
    `define D16_2_rd_pos    10'b110000_0101

// D5.6
    `define D5_6_octet     8'b101_11100
    `define D5_6_rd_neg    10'b001111_1010
    `define D5_6_rd_pos    10'b110000_0101
```

Figura 4: Valores de Data code-groups.

[b]0.45

```
// Special group code

// K28.5 IDLE /I/
    `define K28_5_octet     8'b101_11100
    `define K28_5_rd_neg    10'b001111_1010
    `define K28_5_rd_pos    10'b110000_0101

// K23.7 EXTEND /R/
    `define K23_7_octet     8'b111_10111
    `define K23_7_neg       10'b111010_1000
    `define K23_7_pos       10'b000101_0111
```

Figura 5: Valores de Special code-group.

Figura 6: Tabla de code-goroups para la codificación.

- cggood o cgbad: el estándar proporciona una definición lógica formal para estas variables internas. Estas variables internas dependen de dos entradas: rx_code-group y rx_even.

El módulo del sincronizador posee 9 estados, el cual ha sido codificado según la tabla 4.

Tabla 4: Estados del módulo del sincronizador.

Estado	Código	Representación decimal
LOSS OF SYNC	0 0000 0001	1
COMMA DETECT 1	0 0000 0010	2
ACQUIRE SYNC 1	0 0000 0100	4
COMMA DETECT 2	0 0000 1000	8
ACQUIRE SYNC 2	0 0001 0000	16
COMMA DETECT 3	0 0010 0000	32
SYNC ACQUIRED 1	0 0100 0000	64
SYNC ACQUIRED 2	0 1000 0000	128
SYNC ACQUIRED 2A	1 0000 0000	256

El estado LOSS OF SYNC representa el estado inicial del sincronizador, en donde no existe sincronización en la subcapa PCS. Para salir de este estado, el sincronizador debe recibir un coma. Al recibirlo, pasa al estado COMMA DETECT 1, donde pone rx_even en 1. Para pasar del estado COMMA DETECT 1 al estado ACQUIRE SYNC 1, debe recibir un dato (específicamente, D16.2). En el estado ACQUIRE SYNC 1, invierte rx_even, y se queda esperando el segundo coma. Al recibir el segundo coma estando rx_even en 0, pasa al estado COMMA DETECT 2. En este estado, vuelve a poner rx_even en 1. Para pasar del estado COMMA DETECT 2 a ACQUIRE SYNC 2, debe recibir otro D16.2. En el estado ACQUIRE SYNC 2, invierte rx_even, y se queda esperando el tercer coma. Tras recibir el tercer coma estando rx_even en 0, pasa al estado COMMA DETECT 3. En este estado, vuelve a poner rx_even en 1. Si en este estado recibe otro D16.2, adquire la sincronización y pasa al estado SYNC ACQUIRED 1, donde coloca la señal code_sync_status en 1 y alterna rx_even.

Si en el estado SYNC ACQUIRED 1, recibe un cgbad, el sincronizador no pierde la sincronización, pero pasa al estado SYNC ACQUIRED 2. En este sentido, SYNC ACQUIRED 2 representa un estado de tolerancia al error. Si en SYNC ACQUIRED 2 vuelve a recibir otro cgbad, el sincronizador pierde la sincronización, pero si recibe un cggood, pasa al estado SYNC ACQUIRED 2A. Si en este estado, recibe 3 cggood seguidos, regresa al estado SYNC ACQUIRED 1. Si el estado SYNC ACQUIRED 2 representa un estado de tolerancia al error, entonces SYNC ACQUIRED 2A representa un estado de recuperación del error.

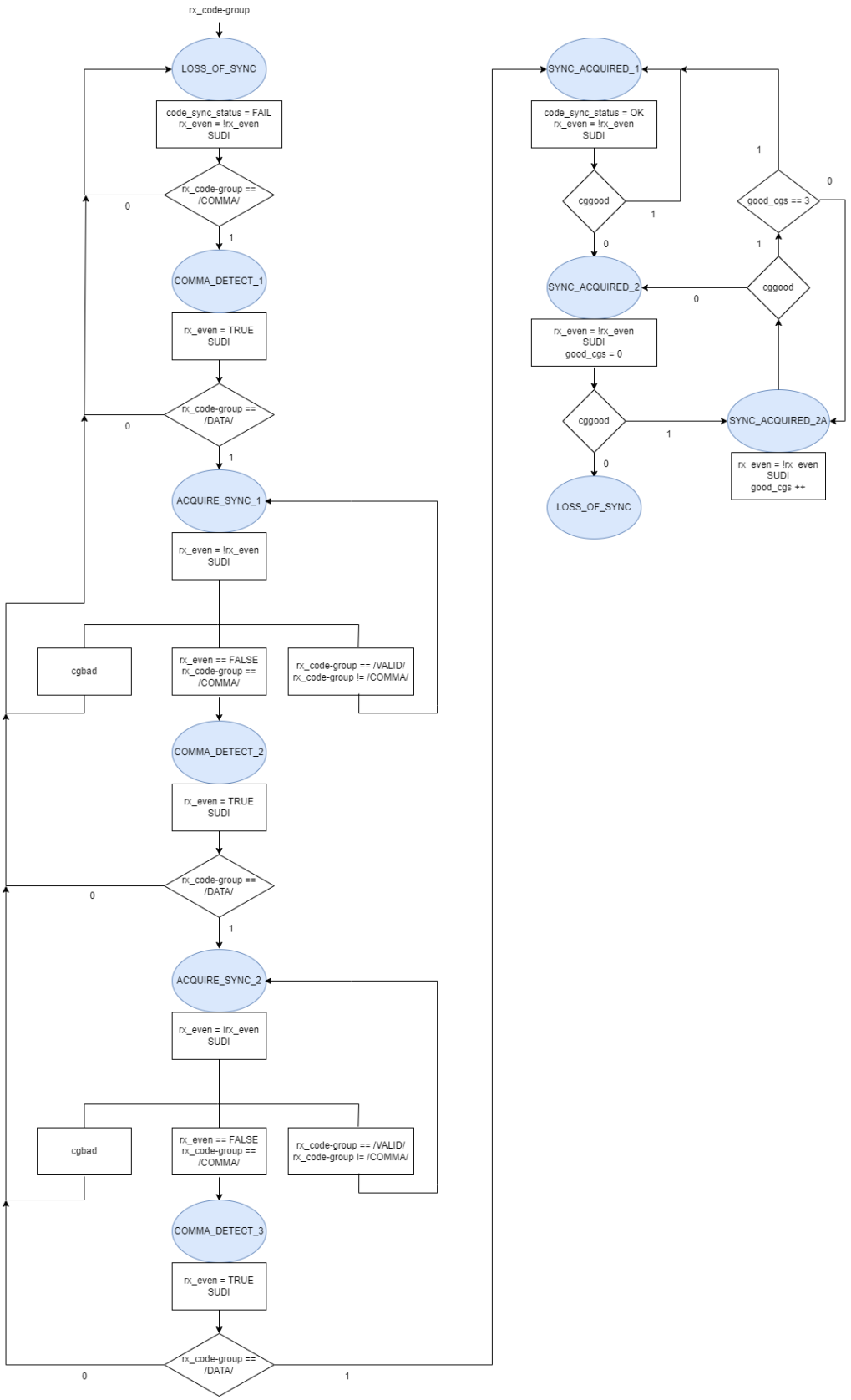

Figura 7: Diagrama ASM del módulo del sincronizador.

2.3. Receptor

Finalmente, el módulo del receptor se trabajó con 7 estados.

Estado	Código (Binario)	Código (Hexadecimal)
LINK_FAILED	0000001	01
WAIT_FOR_K	0000010	02
RX_K	0000100	04
IDLE_D	0001000	08
START_OF_PACKET	0010000	10
RECEIVE	0100000	20
TRI_RRI	1000000	40

Tabla 5: Estados y códigos del módulo receptor en binario y hexadecimal

El módulo cuenta con las siguientes entradas y salidas:

Tipo	Nombre	Descripción
Input	RX_CLK	Reloj de sincronización del receptor.
Input	mr_main_reset	Señal de reinicio principal.
Input	sync_status	Estado de sincronización del receptor.
Input	SUDI	Grupo de códigos recibidos (10 bits).
Input	rx_even	Señal o variable que indica si el receptor está en un ciclo par
Output	RXD	Datos decodificados de 8 bits.
Output	RX_DV	Señal que indica datos válidos.
Output	RX_ER	Señal que indica error en la recepción.

Tabla 6: Entradas y salidas del módulo receptor

El principal cambio realizado fue la unión de los estados RECEIVE y RX_DATA. Esta decisión se tomó debido a que, durante las pruebas, se observó que la transición entre estos estados tomaba más tiempo del esperado, lo que impedía la decodificación correcta del primer dato recibido. La integración de ambos estados permitió el correcto funcionamiento del receptor en el proceso de decodificación.

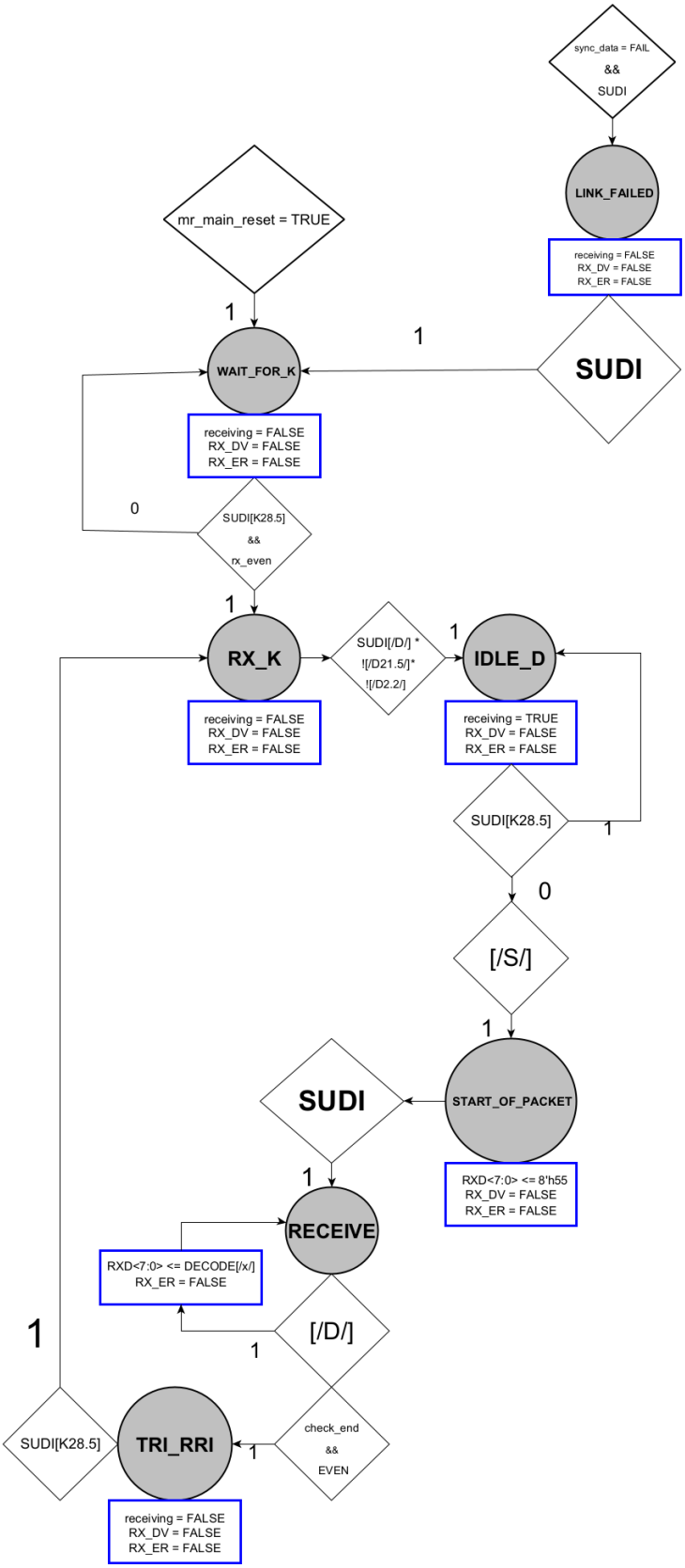

Figura 8: Diagrama Final del Receptor

Descripción de la transición de Estados:

El receptor comienza dependiendo de dos posibles condiciones iniciales: si se realiza un `mr_main_reset`, el sistema pasa directamente al estado `WAIT_FOR_K`, y si se detecta que la sincronización de datos se ha perdido (`sync_status = FAIL`) y si se detecta el `SUDI`, el receptor transita al estado `LINK_FAILED`.

En el estado `LINK_FAILED`, las salidas `receiving`, `RX_DV` y `RX_ER` se inicializan en `FALSE`. La transición desde `LINK_FAILED` ocurre al detectar el símbolo `SUDI` únicamente, que indica que los datos se han sincronizado correctamente. Esto permite al sistema pasar al estado `WAIT_FOR_K`, donde se espera la recepción del símbolo `K28.5` para confirmar que la sincronización inicial es válida.

En `WAIT_FOR_K`, el sistema permanece atento al símbolo de coma `K28.5` representado en `SUDI`. Si este símbolo se detecta y `rx_even` es válido, el receptor transita al estado `RX_K`. En este estado, se verifica la recepción de símbolos que aseguren un enlace estable. Si se reciben datos de `/D/`, y mientras no se detecten valores como `D21.5 o D2.2`, el sistema avanza al estado `IDLE_D`.

En el estado `IDLE_D`, `receiving` se establece en `TRUE`, mientras que `RX_DV` y `RX_ER` permanecen en `FALSE`. La transición desde este estado ocurre si se detecta un nuevo símbolo `K28.5`, devolviéndose a `RX_K`, o si se detecta el paquete de `/S/` , en cuyo caso se pasa al estado `START_OF_PACKET`.

En `START_OF_PACKET`, el sistema procesa los datos iniciales de un paquete. `RXD` se establece en un valor predeterminado `0101 0101`, y las señales `RX_DV` y `RX_ER` permanecen en `FALSE`. Si se reciben los datos mediante `SUDI`, el receptor transita al estado `RECEIVE`, donde se inicia la recepción activa de datos del paquete.

En el estado `RECEIVE`, si se recibe un carácter `/D/` perteneciente al conjunto de datos definidos en el `lookup table`, el sistema procede a realizar la decodificación. En este proceso, la salida `RXD` toma los valores decodificados correspondientes a `SUDI`, mientras que `RX_ER` se mantiene en `FALSE`, indicando una recepción exitosa y `RX_DV` en `TRUE` si el dato es válido. Adicionalmente, si se detecta el indicador `check_end` durante un ciclo par, el sistema realiza una transición al estado `TRI_RRI`.

Finalmente, en `TRI_RRI`, el sistema completa el ciclo de recepción. En este estado, las salidas principales se reinician: `receiving = FALSE`, `RX_DV = FALSE` y `RX_ER = FALSE`. Si se detecta un símbolo `K28.5` en `SUDI`, el sistema regresa al estado `RX_K` para reiniciar el proceso de recepción con sincronización.

2.4. Subcapa PCS

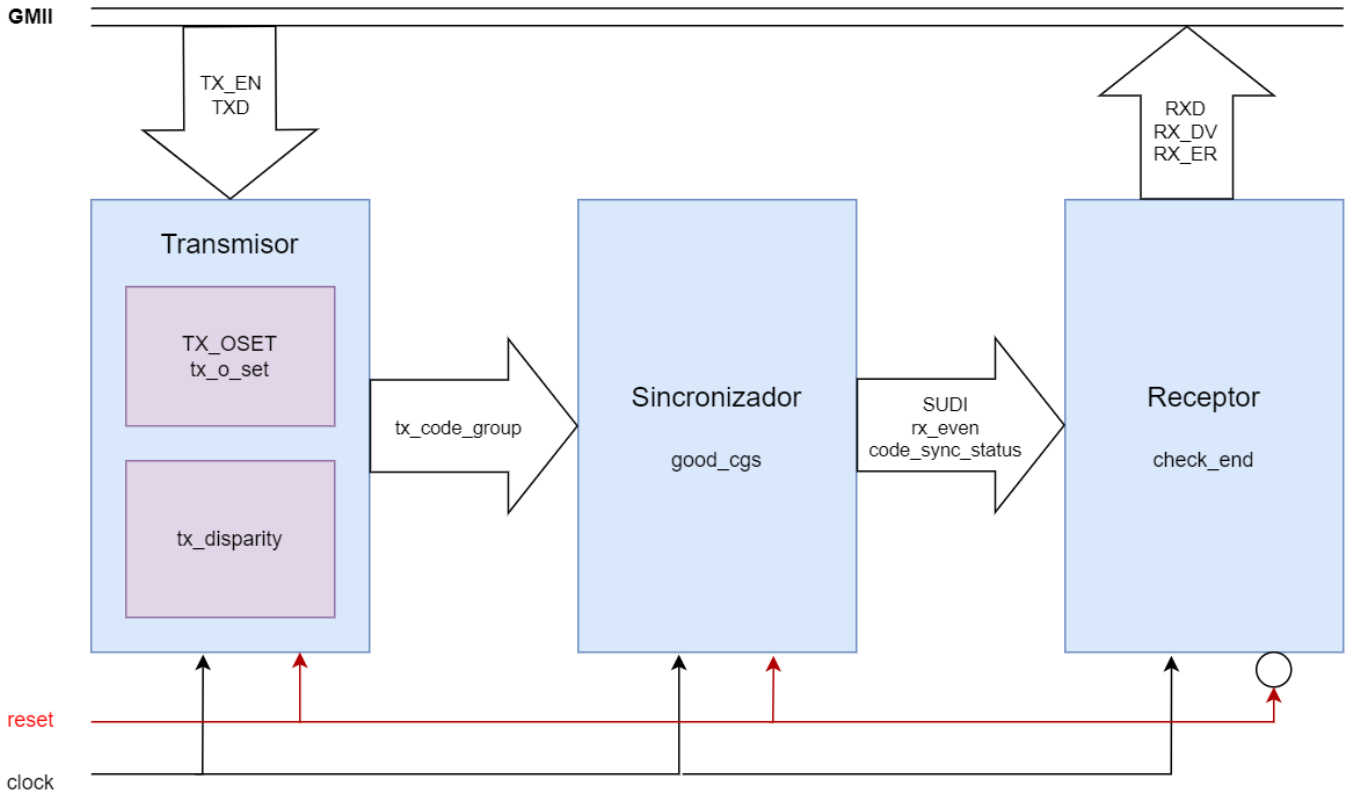

Figura 9: Diagrama de bloques de la subcapa PCS.

3. Plan de pruebas

3.1. Transmisor

- `/I/I/I/S/D/D/D/D/D/T/R/I/I/I/`
- Prueba 1: Estado de Idle. El transmisor no recibe ningún dato y automáticamente general Idle2 de manera que envía 3 K28.5 y 3 D16.2.
- Prueba 2: Inicio de transmisión. El GMII habilita la entrada TX_EN y da inicio de la transmión donde el transmisor envía /S/.
- Prueba 3: Envío de Datos. Se envían en total 5 datos /D/, D0.0, D0.2, D16.1, D0.4 Y D0.3.
- Prueba 4: Fin de transmisión. Se da por terminado el envío de datos por lo que deshabilitada la entrada TX_EN.

3.2. Sincronizador

- Prueba 1: Funcionamiento normal. El sincronizador recibe tres IDLE seguidos y se sincroniza. El sincronizador ha pasado esta prueba.

- Prueba 2: Tolerancia a un cgbad. El sincronizador se sincroniza, recibe un cgbad, y luego tres cggood (condición de recuperación). El sincronizador se recupera del error. El sincronizador ha pasado esta prueba.
- Prueba 3: Pérdida de sincronización. El sincronizador se sincroniza, recibe un cgbad, dos cggood, y otro cgbad. El sincronizador pierde la sincronización. El sincronizador ha pasado esta prueba.

3.3. Receptor

- **Detalles de la secuencias:**
  - `/I/`: Caracteres IDLE enviados antes y después de la sincronización.
  - `/S/`: Inicio del paquete (carácter especial `K27.7`).
  - `/D/`: Datos válidos enviados después de la sincronización.
  - `/T/`: Carácter de fin de paquete.
  - `/R/`: Carácter de especial de carrier extend
  - `/K28.5/`: Carácter de coma (IDLE).
- **Prueba 1: Inicio en el estado `WAIT_FOR_K`**
  - **Descripción:** El sistema receptor comienza sincronizado (`sync_status = 1'b1`). Se activa la señal de reinicio principal (`mr_main_reset`) de manera temporal para asegurar el inicio en el estado `WAIT_FOR_K`.
  - **Secuencia de datos:**
    - `/I/I/I/I/S/D/D/D/D/D/T/R/K28.5/I/I/`
  - **Objetivo:** Verificar que el receptor transita correctamente entre los estados esperados y que las salidas `RXD`, `RX_DV`, y `RX_ER` responden según lo especificado.
- **Prueba 2: Inicio en el estado `LINK_FAILED`**
  - **Descripción:** El receptor comienza en estado de fallo de enlace (`sync_status = 1'b0`). Posteriormente, la señal `sync_status` se activa (`1'b1`) para permitir la sincronización con los datos entrantes.
  - **Secuencia de datos:**
    - `/I/I/I/I/S/D/D/D/T/R/K28.5/I/I/`
  - **Objetivo:** Validar la recuperación del receptor desde un estado no sincronizado y verificar la transición a estados operativos normales. También se evalúan las señales de salida en función de los datos recibidos.

3.4. Subcapa PCS

- Prueba única: envío de una trama completa de Ethernet. Los tres módulos se sincronizan correctamente, el transmisor envía una trama completa y el receptor recibe la misma trama. La estructura de la trama utilizada fue: /S/D/D/D/D/D/T/R/K28.5/, donde los cinco datos fueron:

Tabla 7: Datos utilizados en la prueba de la subcapa PCS

Dato	Código	Octeto
1	D0.0	00000000
2	D0.2	01000000
3	D16.1	00110000
4	D0.4	10000000
5	D0.3	01100000

4. Instrucciones de simulación

En el repositorio 2S2024_G3, encontrará varios branches. En el main branch, encontrará los archivos necesarios para simular la subcapa PCS en el folder *TransmitSyncRec*.

Si desea simular cada módulo individualmente, colóquese en el branch correspondiente (*Transmisor*, *Sincronizador* o *Receptor*) y descargue el folder correspondiente.

Los branches *SyncRec* y *TransmitSyncRec* son branches donde se probó la conexión de los diferentes módulos.

1. Asegúrese de que tiene acceso a:
   - iverilog
   - gtkwave
   - make
2. Descargue el folder que desee simular.
3. En la terminal, localícese en la dirección del folder.
4. En la terminal, ejecute el comando *make*.

5. Resultados

5.1. Transmisor

Se realizaron las 4 pruebas que se mencionaron anteriormente, por lo que a continuación se puede observar los resultados del transmisor:

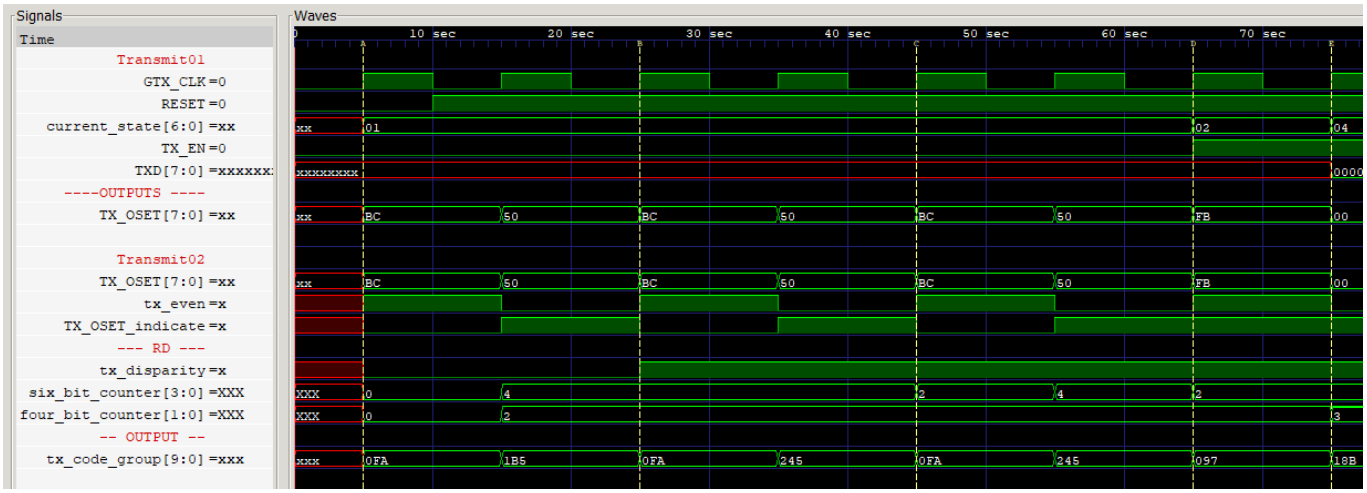

Figura 10: Prueba 1. Estado de Idle.

En la Figura 10 se visualiza donde se tienen los dos módulos de transmisor, y se divide en 3 bloques principales, en el primer bloque se puede observar que el TX_OSET es *BC* cuyo valor es el K28.5 representado en 7 bits, por lo que en el mismo flanco se generan los 10 bits codificado en *0FA*, posteriormente se obtiene el *50* el cual representa el valor de D16.2 en Hexadecimal y se codifica a 1B5, este se repite por el tiempo que el valor de TX_EN está en bajo.

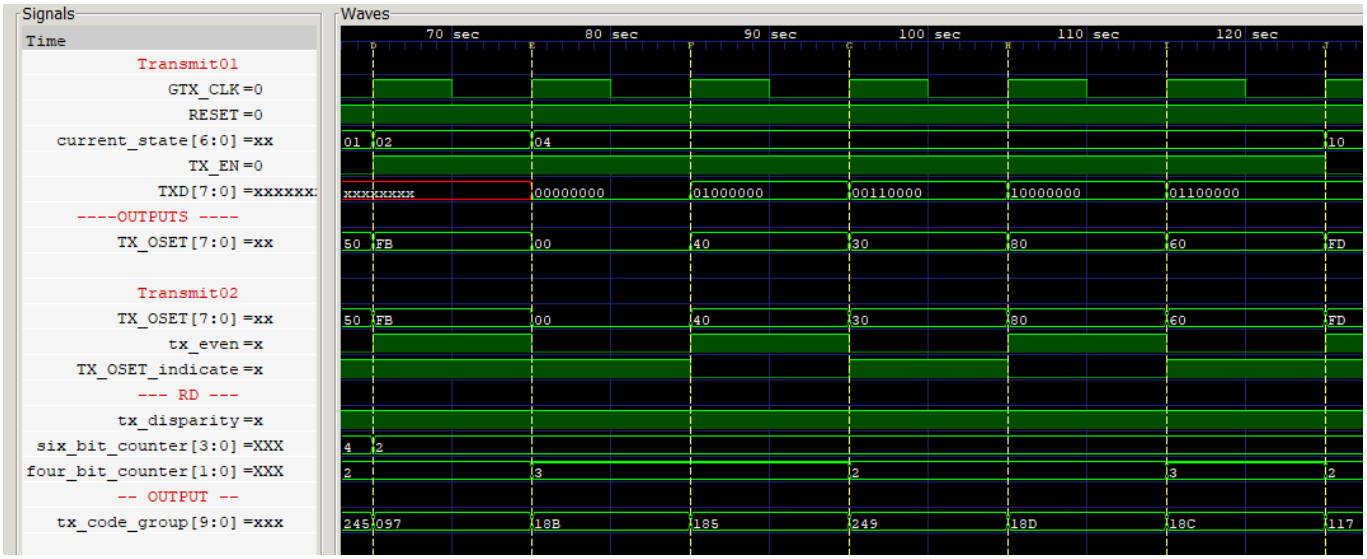

Figura 11: Prueba 2 y 3. Inicio de transmisión y Envío de Datos.

En la Figura 11 se observa el comportamiento de las ondas del gtkwave de 2 pruebas, en el momento que la entrada Tx_EN se pone en alto inicia el envío de transmisión de los datos empezando con el valor de 8 bits de *FB* y en su manera codificada 097 indicando el inicio mandando el valor /S/ K27.7 evidenciando la manera correcta de funcionalidad, por lo que continúan la serie de 5 datos tipo /D/ los cuales en 8 bits son *00*, *40*, *30*, *80*, *60* que una vez codificado obtenemos los valores *18B*, *185*, *249*, *18D* y *18C* respectivamente, por lo que los datos son codificados de manera correcta, donde en el momento que la entrada de TX_EN se pone en bajo termina la prueba.

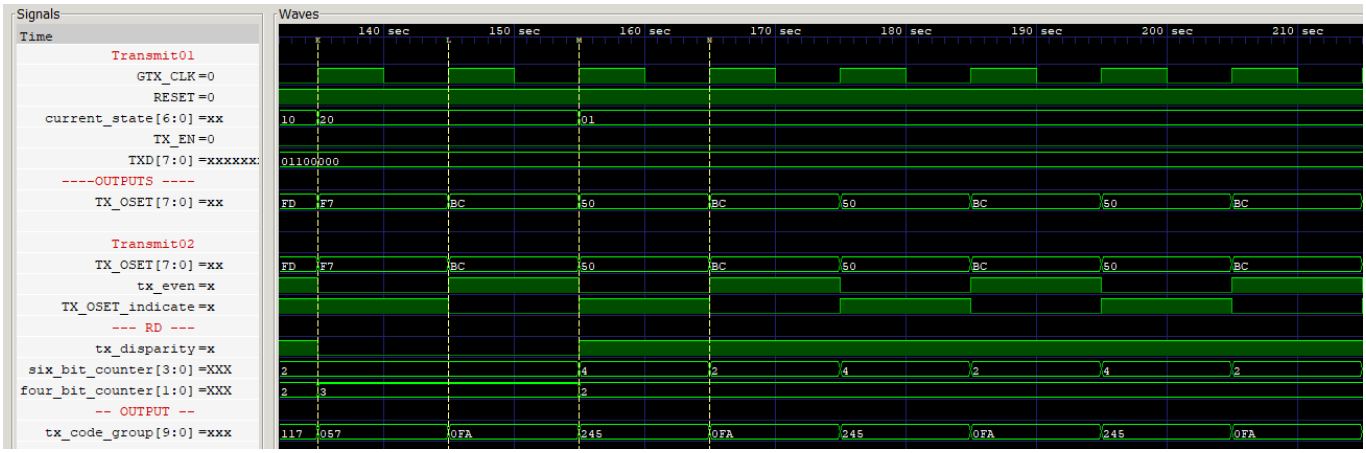

Figura 12: Prueba 4. Fin de transmisión.

Por último, tenemos la prueba final que se realiza para terminar la transmisión de los datos del PCS por lo que en Figura 12 se puede observar que al desactivarse la entrada TX_EN se envía el valor de *FD* a TX_OSET codificado a *117* indicando el fin con el valor /T/ K29.7, para que posteriormente se envíe el valor *F7* codificado a *057* enviando el valor del carrier por lo que una vez finalizada vuelve a enviar los idles observando que en tx_code_group se vuelve a *0FA* y *245* que en este caso como la disparidad calculada fue positiva se envía un dato D16.2 positivo a diferencia al principio que se envió un *1B5* mostrando que el cálculo de la disparidad funciona correctamente.

5.2. Sincronizador

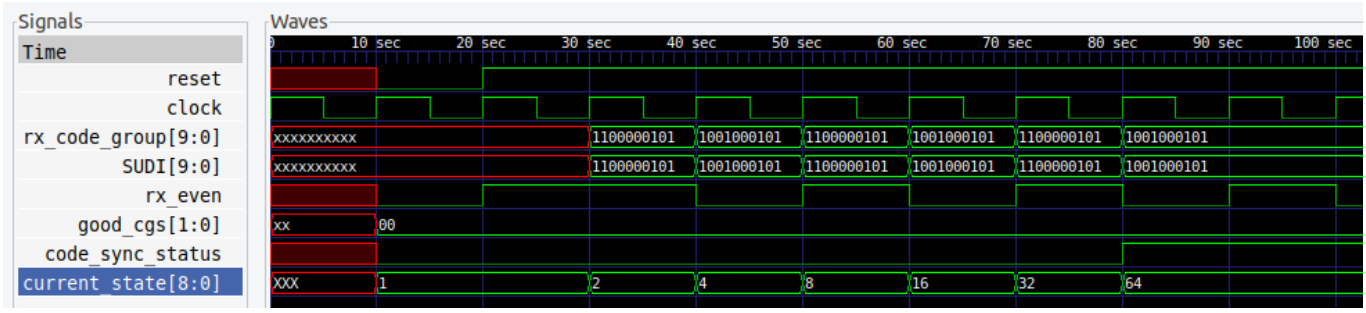

Figura 13: Prueba 1. Funcionamiento normal.

Como se observa en la figura 13, el sincronizador inicia en LOSS OF SYNC (estado 1). Al recibir el primer coma (en este caso, K28.5 RD+), cambia al estado COMMA DETECT 1 (estado 2) y pone rx_even en 1. Luego, al recibir un dato (en este caso, D16.2 RD+), cambia al estado ACQUIRE SYNC 1 (estado 4) e invierte rx_even. Para pasar del estado 4 (ACQUIRE SYNC 1) al estado 8 (COMMA DETECT 2), el sincronizador debe recibir un coma estando rx_even en 0. Al recibir el coma, pone rx_even en 1. Para pasar del estado 8 (COMMA DETECT 2) al estado 16 (ACQUIRE SYNC 2), recibe nuevamente un D16.2 RD+ e invierte rx_even. Al recibir un tercer coma, el sincronizador pasa al estado COMMA DETECT 3 (estado 32) y pone rx_even en 1. Finalmente, al recibir un nuevo D16.2 RD+, pasa al estado SYNC ACQUIRED 1 (estado 64) y se pone la señal code_sync_status en 1.

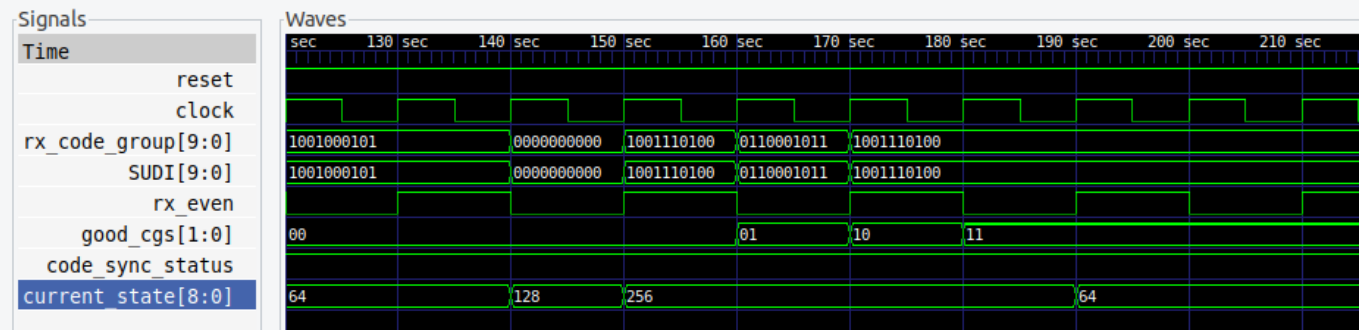

Figura 14: Prueba 2. Tolerancia a un cgbad.

Como se observa en la figura 14, el sincronizador inicia en el estado 64 (sincronizado). Recibe 10 bits de 0, el cual representa una condición cgbad. El sincronizador pasa al estado 128 (SYNC ACQUIRED 2). Posteriormente, recibe un cggood (D0.0 RD-). El sincronizador pasa al estado 256 (SYNC ACQUIRED 2A). En total, se envían tres datos válidos (D0.0 RD-, D0.0 RD+, D0.0 RD-). La variable interna good_cgs cuenta correctamente los tres cggood, y el sincronizador recupera el estado sincronizado.

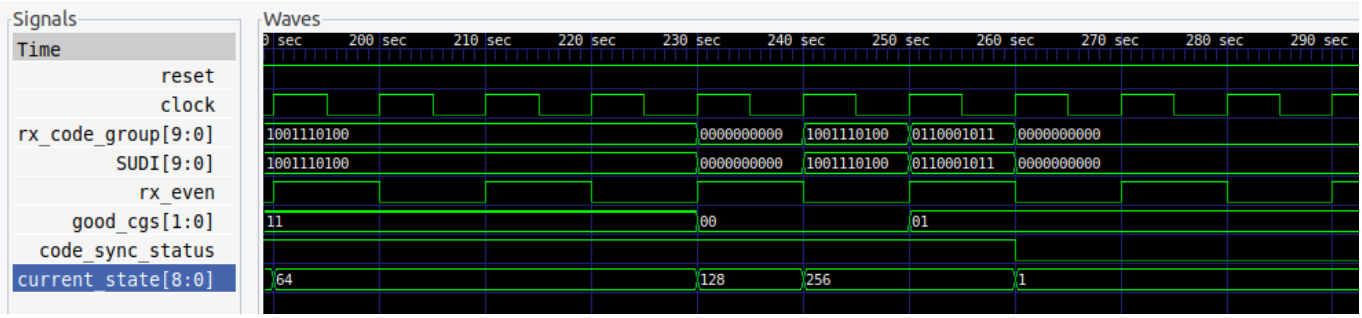

Figura 15: Prueba 3. Pérdida de sincronización

Como se observa en la figura 15, el sincronizador inicia en el estado 64 (sincronizado). Recibe 10 bits de 0, el cual representa una condición cgbad. El sincronizador pasa al estado 128 (SYNC ACQUIRED 2). Posteriormente, recibe un cggood (D0.0 RD-). El sincronizador pasa al estado 256 (SYNC ACQUIRED 2A). A diferencia de la prueba 2, en esta prueba sólo recibe dos datos válidos en total (D0.0 RD-, D0.0 RD+). Tras recibir otro cgbad, el sincronizador pierde la sincronización (estado 1).

5.3. Receptor

- **Prueba 1:**

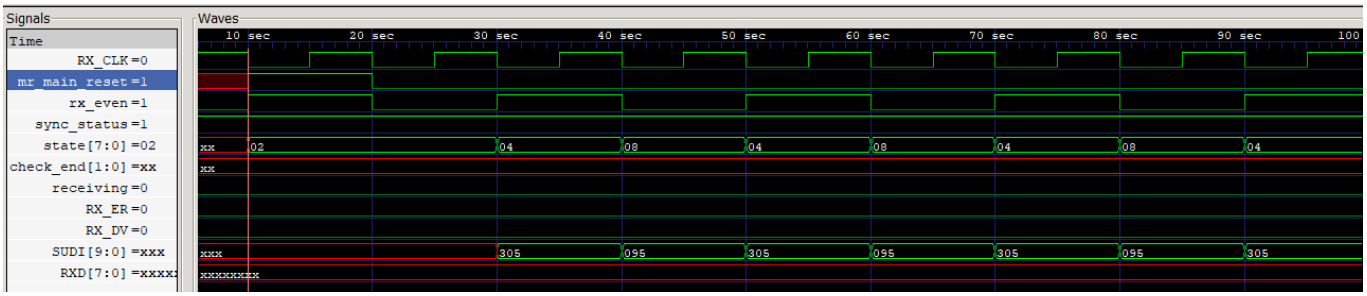

Figura 16: IDLES

- **Descripción:** Se observa que se inicia en el estado de `WAIT_FOR_K` debido a que se inicia con un `mr_main_reset`, además se están recibiendo los IDLES por medio de SUDI, pero como todavía no nos encontramos en el estado de `RECEIVE` no se realiza la decodificación de los mismos.

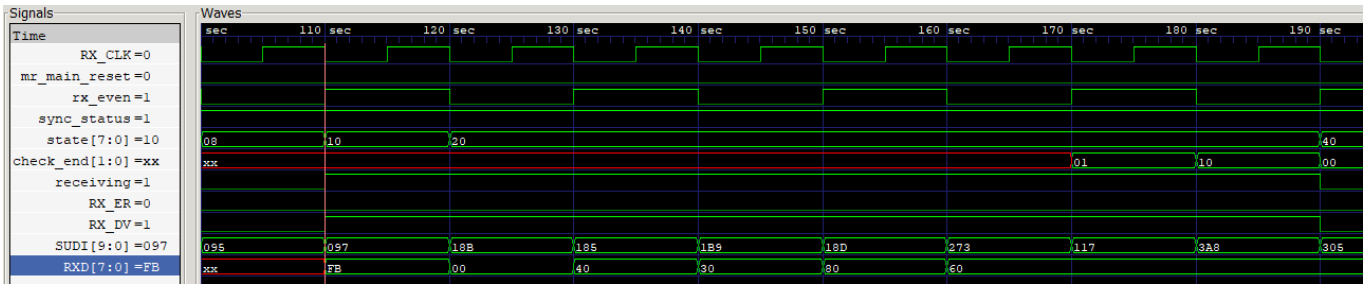

Figura 17: Decodificación de Datos

- **Descripción:** Una vez se recibe el carácter de `/S/` y recibe datos por medio de `SUDI` se pasa al estado de `RECEIVE` donde se realiza la decodificación de 10 bits a 8 bits de todos los datos recibidos en la señal de `RXD`.

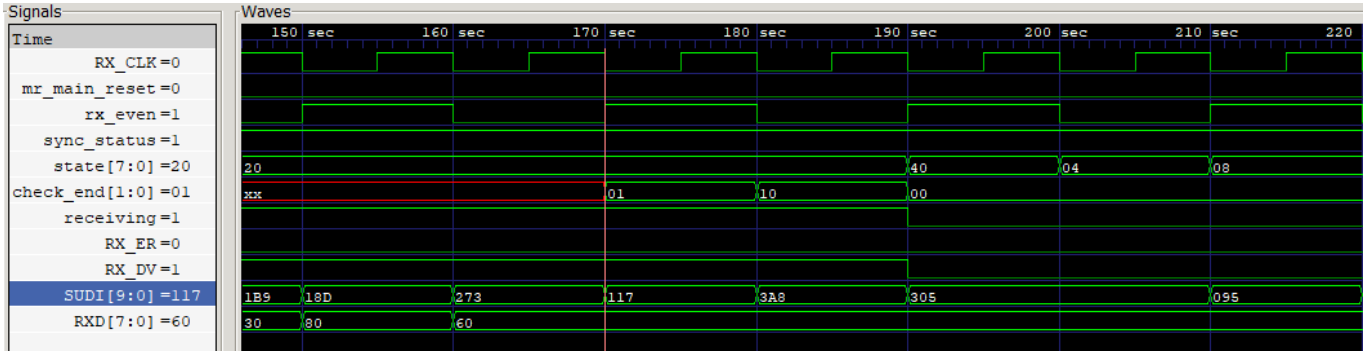

Figura 18: Secuencia de finalización(`check_end`)

- **Descripción:** Cuando en el estado de `RECEIVE` detecta el dato `/T/` se inicializa el contador interno de la función de `check_end`, y aumenta al recibir `/R/` y finalmente `/K28.5/`. Al recibir está secuencia se realiza la transición de estado a `TRI_RRI`, siendo este el estado final. Seguidamente se aprecia que regresa al `RX_K` y luego pasa alternando con el estado `IDLE_D`.
- **Prueba 2:**

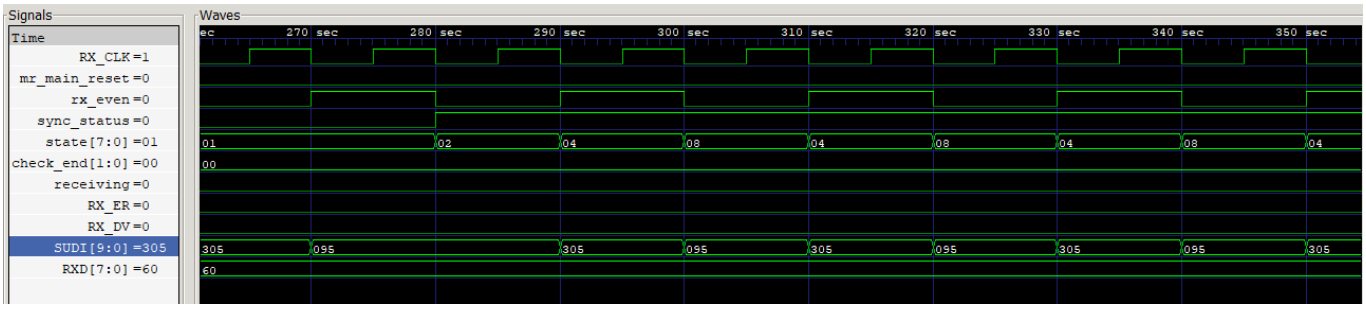

Figura 19: Receptor no sincronizado

- **Descripción:** Se observa que se inicia en el estado de `LINK_FAILED`, debido a que se inicia con un `sync_status` en bajo, es decir no se encuentra sincronizado, además se están recibiendo los IDLES por medio de `SUDI`, pero como todavía no está sincronizado no se realiza ninguna tarea. Hasta que la señal de `sync_status` se pone en alto procede a funcionar de formal normal.

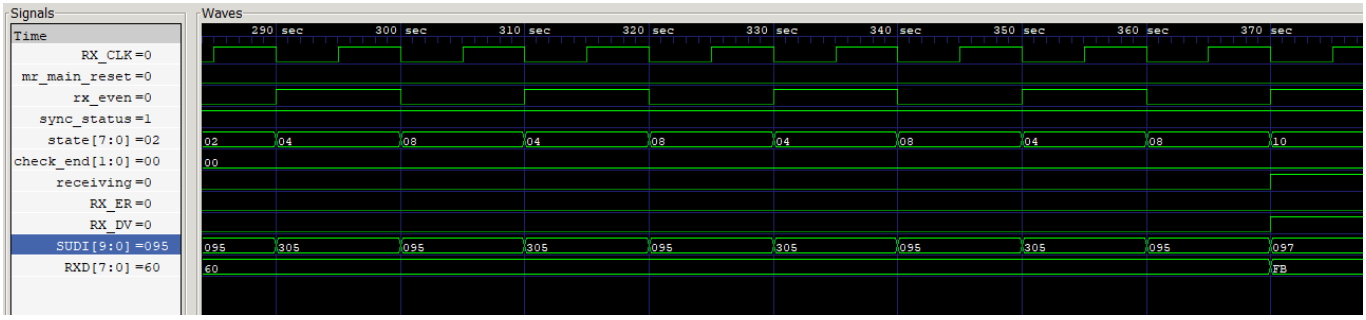

Figura 20: Sincronización adquirida

- **Descripción:** Una vez sincronizado el receptor, empieza a alternar entre los estados `RX_K` y `IDLE_D`. Hasta que se recibe el carácter de `/S/` se realiza la transición al estado de `START_OF_PACKET` y decodifica este valor en `RXD`.

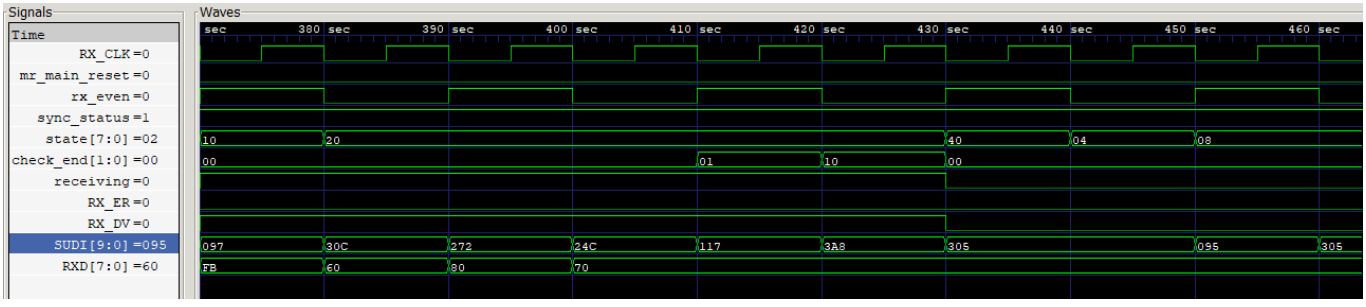

Figura 21: Fin de la secuencia

- **Descripción:** En el estado de `RECEIVE` se realiza la decodificación de los datos recibos en la señal de `RXD`. Cuando en se detecta el dato `/T/` se inicializa el contador interno de la función de `check_end`, y aumenta al recibir `/R/` y finalmente `/K28.5/`. Al recibir está secuencia se realiza la transición de estado a `TRI_RRI`, siendo este el estado final. Seguidamente se aprecia que regresa al `RX_K` y luego pasa alternando con el estado `IDLE_D`.

5.4. Subcapa PCS

- **IDLES**

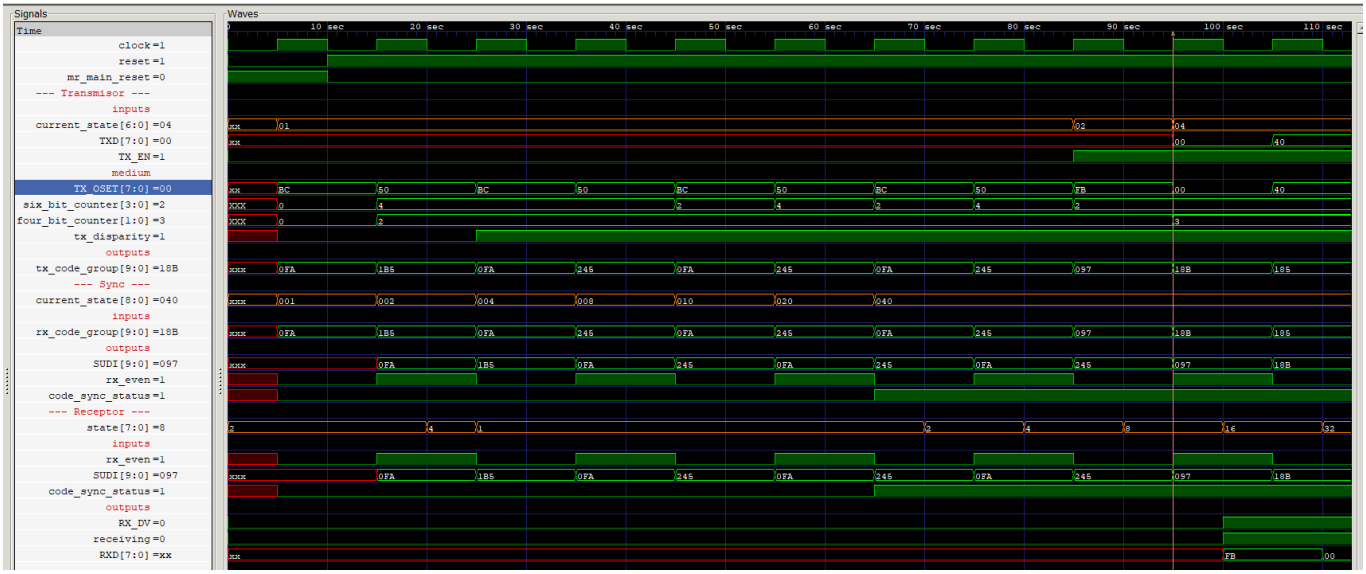

Figura 22: Etapa de IDLES y sincronización

Se puede observar que el transmisor trabaja de manera natural codificando los idles que el mismo módulo genera cuando la entrada TX_EN se encuentra en bajo. Se puede visualizar que se sincroniza a las #10 unidades de tiempo de cuando se realizó la codificación.

Se observa que la detección de las 3 commas es efectivo y realiza efectivamente el Acquire Sync para recibir los D16.2 invirtiendo la variable rx_even y por último entra al estad 40 en HEX poniendo la señal en alto code_sync_status.

En esta parte el receptor no se encuentra en operación. Inicialmente, debido a que se realiza un `mr_main_reset` comienza en el estado `WAIT_FOR_K` y transiciona al estado de `RX_K`, pero el receptor nota que la señal de `code_sync_status` se encuentra en bajo, indicando que el dispositivo no se ha sincronizado entonces automáticamente reacciona y pasa al estado de `LINK_FAILED`. Hasta el momento en que la señal `code_sync_status` se activa indicando significa que se ha sincronizado el dispositivo y puede seguir con las transiciones de estado.

- **Operación**

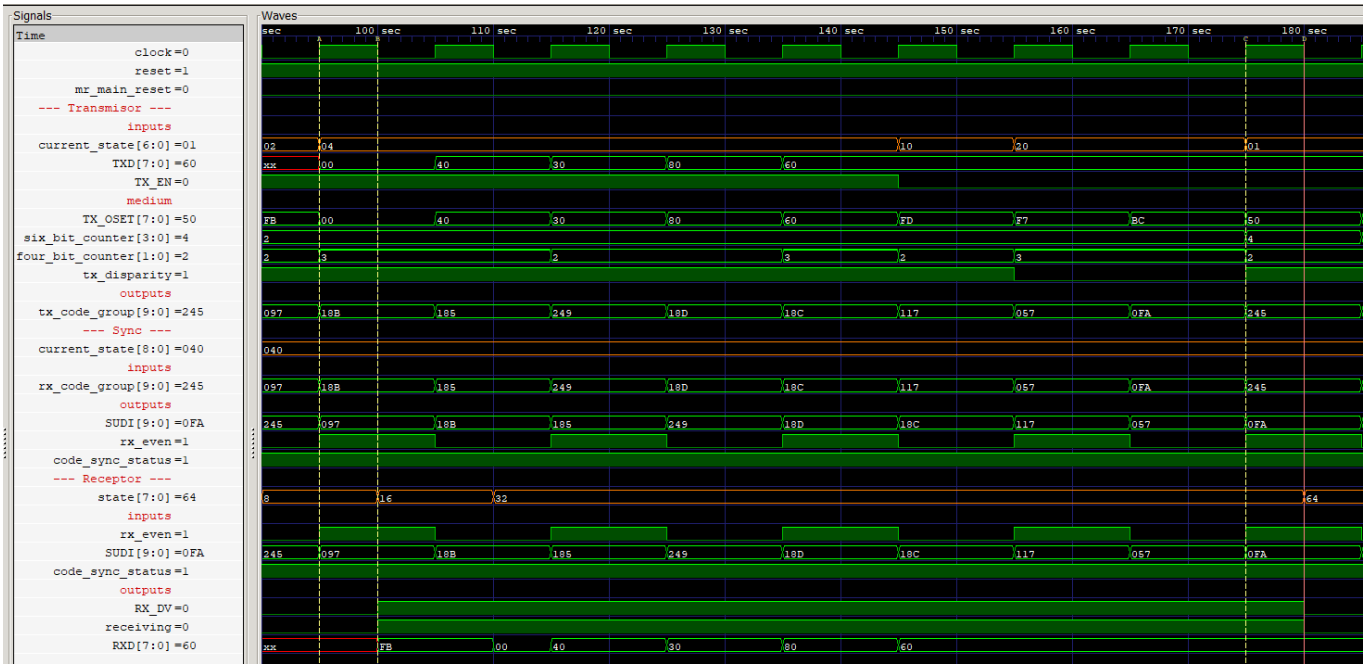

Figura 23: Etapa de Inicio de transmisión de datos

Una vez el transmisor envía /S/ inicia la transmisión de datos se puede visualizar que se decodifica los 10 bits #15 unidades de tiempo del momento que se envió por lo que se tiene una sincronización y una recepción de datos eficiente y correcta.

Finalmente, cuando el receptor detecta el carácter `/S/` y comienza a recibir datos a través de `SUDI`, transita al estado `RECEIVE`. En este estado, se lleva a cabo la decodificación de los datos, convirtiendo los 10 bits en su equivalente de 8 bits y se envían a través de `RXD`.

- **Fin de la operación**

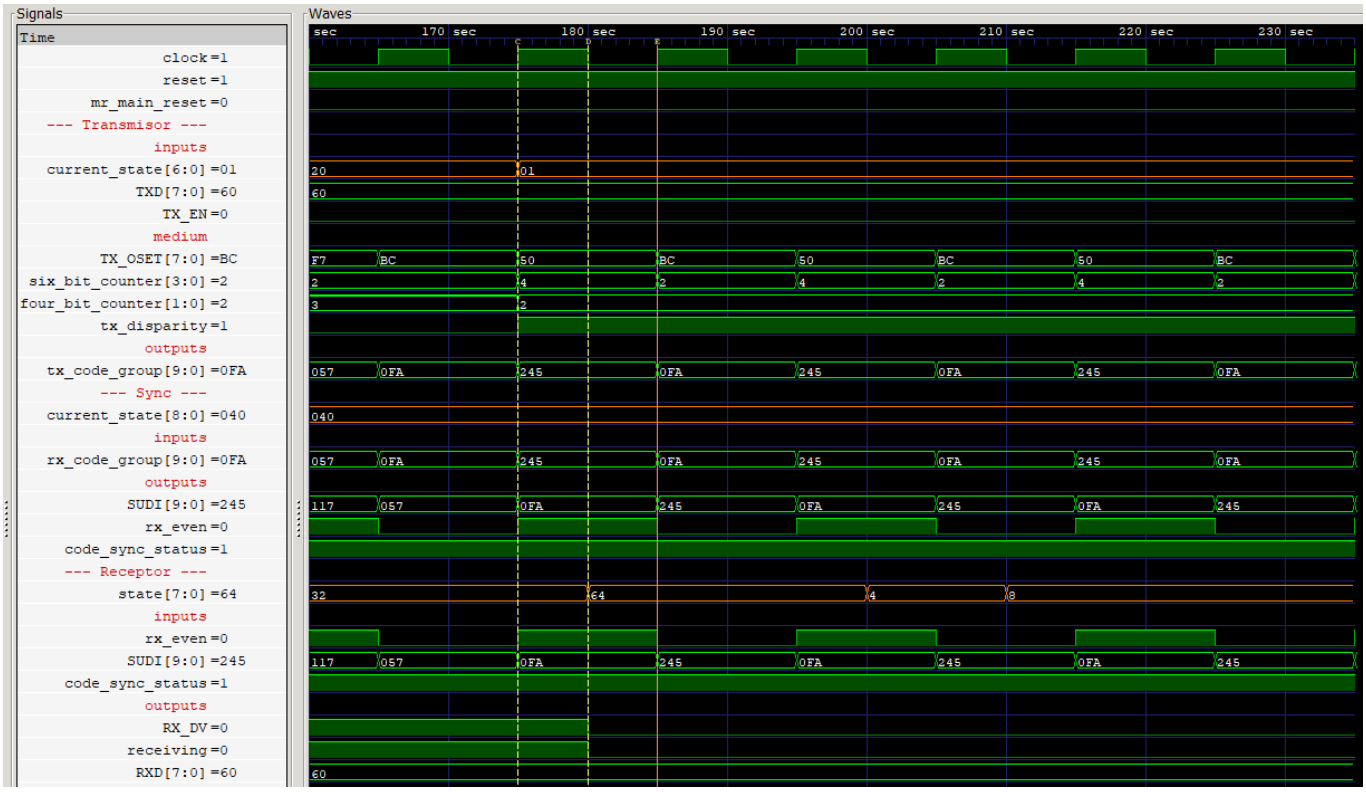

Figura 24: Fin de la transmisión de los datos

Como fin de la operación el transmisor termina de enviar los datos, enviar /T/ y el carrier /R/, el sincronizador mantiene todo el rato la sincronización debido a que ya se recibieron las commas y se mantendrá de la misma manera hasta que se ponga en bajo el RESET. De la misma manera los datos tienen un diferencia de #10 unidades de tiempo.

En el caso del receptor, en el estado de `RECEIVE` detecta el dato `/T/` se inicializa el contador interno de la función de `check_end`, y aumenta al recibir `/R/` y finalmente `/K28.5/`. Al recibir está secuencia se realiza la transición de estado a `TRI_RRI`, siendo este el estado final. Seguidamente se aprecia que regresa al `RX_K` y luego pasa alternando con el estado `IDLE_D`.

6. Conclusiones

Por lo tanto, se logró implementar exitosamente los tres módulos de la capa PCS del protocolo Ethernet. La integración de estos módulos se realizó de manera fluida, gracias a una adecuada distribución de tareas y a la comunicación efectiva entre los miembros del equipo, lo que permitió optimizar el desarrollo y garantizar la funcionalidad esperada.

Se puede concluir que la implementación del *lookup table* fue fundamental para el diseño del transmisor, sincronizador y receptor en la capa PCS del protocolo Ethernet. Este componente permitió la validación eficiente de los grupos de códigos especiales y de datos, garantizando una decodificación precisa de las señales recibidas.

Además, el establecimiento de un flujo de trabajo claro y el consenso sobre las metodologías empleadas ayudaron a evitar confusiones y garantizar que los resultados fueran consistentes con los

objetivos planteados. Esto permitió no solo cumplir con los plazos establecidos, sino también abordar problemas de manera efectiva.

En conclusión, el éxito del proyecto no solo radicó en el diseño técnico, sino también en la planificación y comunicación efectiva dentro del equipo, aspectos que son fundamentales para cualquier desarrollo colaborativo en ingeniería. Este enfoque integral asegura que el proyecto fuera confiable y cumpliera con los estándares esperados de funcionalidad y precisión.