Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0523 – Circuitos Digitales II
I ciclo 2023

Bloque de PCS 1000BASE-X

Proyecto Final

Grupo 3

Marvin Castro Castro C01884
Ronny Granados Perez C03505
Lorena Solís Extenny B97657
Gabriel Briceño Cambronero C11262

Profesor: Enrique Coen Alfaro

8 de julio, 2023

1. Resumen

Por medio del estudio detallado e individual del estándar 802.3 sección 3 de la IEEE, se creo un bloque de la subcapa de codificación física (PCS, por sus siglas en inglés), de acuerdo con las especificaciones del estándar, el cual esta compuesto por cuatro submódulos de los cuales dos submódulos forman el bloque de TRANSMIT y los otros dos submódulos corresponden a los bloques de SYNCHRONIZATION y RECEIVE respectivamente. Se desarrollaron las descripciones de cada submódulo en Verilog y luego se unieron en un wrapper llamado PCS.v el cual al ser probado fue capaz de recibir una secuencia de datos por una entrada TXD y mandarlos a través de todos los submódulos para crear una salida RXD por la cual sale la misma secuencia de datos.

2. Descripción Arquitectónica

El módulo principal `PCS.v` consta de los siguientes tres módulos: `TRANSMIT, SYNCHRONIZE` y `RECEIVE`.

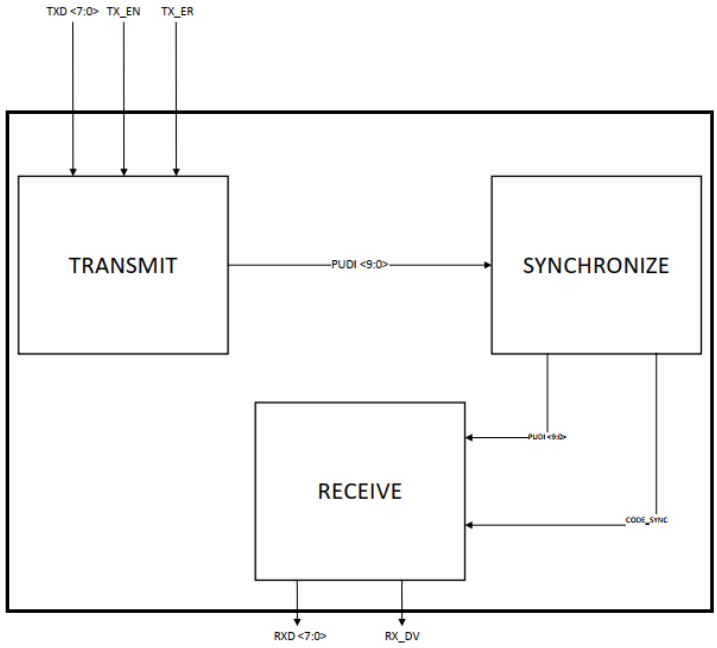


Figura 1: Módulo PCS

Se tiene una conexión directa de `TRANSMIT` a `SYNCHRONIZE` ya que está en una configuración de loopback, donde los datos enviados por `TRANSMIT` son los recibidos en `SYNCHRONIZE`. Se realiza la sincronización, y finalmente la máquina de `RECEIVE` se encarga de poner los caracteres de datos en la salida del modulo, decodificados a 8 bits. A continuación se dará una descripción de cada módulo

2.1. TRANSMIT

La principal función de la máquina de `TRANSMIT` es realizar la codificación a 10 bits, de los datos de 8 bits que entran por medio de `TXD`. El correcto funcionamiento de esta máquina dependen de los estados de `TX_EN` y `TX_ER`. Para la implementación de este módulo en Verilog, se dividió el módulo en dos: `TRANSMIT_ORDERED_SET` y `TRANSMIT_CODE_GROUPS`

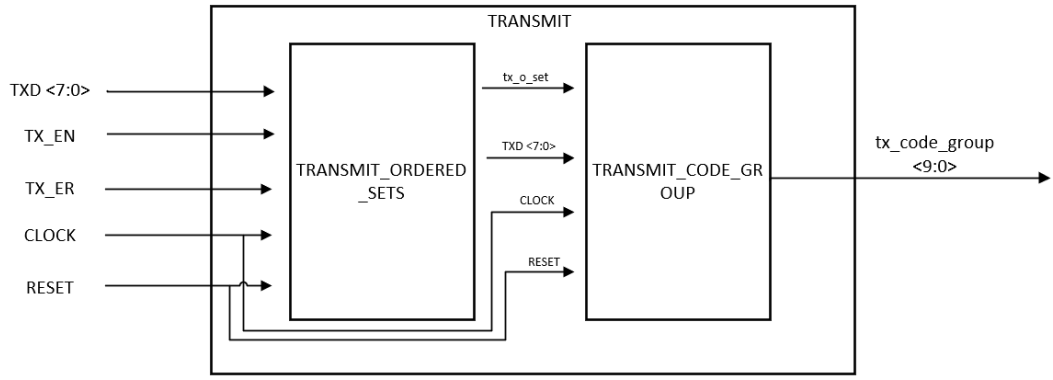


Figura 2: Módulo TRANSMIT

2.1.1. TRANSMIT_ORDERED_SET

Este submódulo se encarga de definir el orden de la trama de datos mencionada en la figura 8, le dice a `TRANSMIT_CODE_GROUPS` cuales codegroups debe poner en su salida, siguiendo el orden de un idle, start of packet, datos, y los codegroups de final /T/R/. Su salida `tx_o_set` le indica al módulo de code groups que clase de carácter debe codificar, para ponerlo en su salida de 10 bits. A continuación se muestra el diagrama de estados para esta máquina

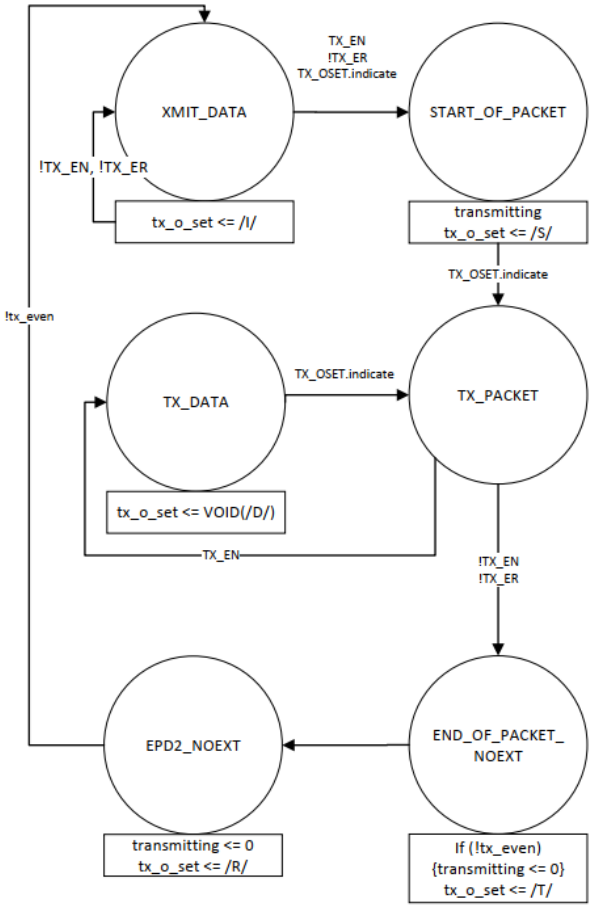


Figura 3: Diagrama de estados TRANSMIT_ORDERED_SET

El módulo funciona de la siguiente manera:

- Si `TX_EN = 0`, o `TX_ER = 1` se mantiene en el estado `XMIT_DATA` y se envían carácteres de idle /I/ por `tx_o_set`.

- Si `TX_EN = 1` y `TX_ER = 0`, se pasa al estado de `START_OF_PACKET`, que por medio de `tx_o_set`, indica que se debe enviar un carácter /S/, start of packet.
- Por la señal `TX_OSET.indicate` proveniente de `TRANSMIT_CODE_GROUPS`, se pasa al `TX_PACKET`. En `TX_DATA` se envía los carácteres de datos, y después se regresa a TX_PACKET. Estos dos estados se repiten entre sí hasta completar la trama de datos
- Una vez completada la trama de datos, se pasa por `END_OF_PACKET_NOEXT` y `EPD2_NOEXT` que se encargan de enviar los caracteres /T/, /R/, respectivamente.

2.2. TRANSMIT_CODE_GROUP

Se encarga de realizar la codificación de 8 a 10 bits, siguiendo el orden de la trama de datos establecida por la salida `tx_o_set` de `TRANSMIT_ORDERED_SET`. En su salida `tx_code_group` se tiene el carácter codificado a 10 bits.

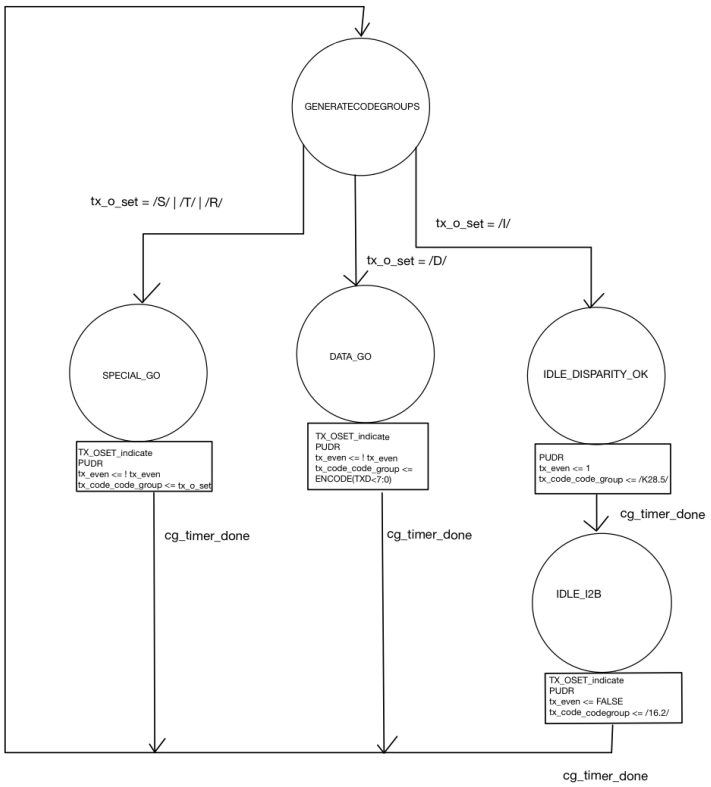
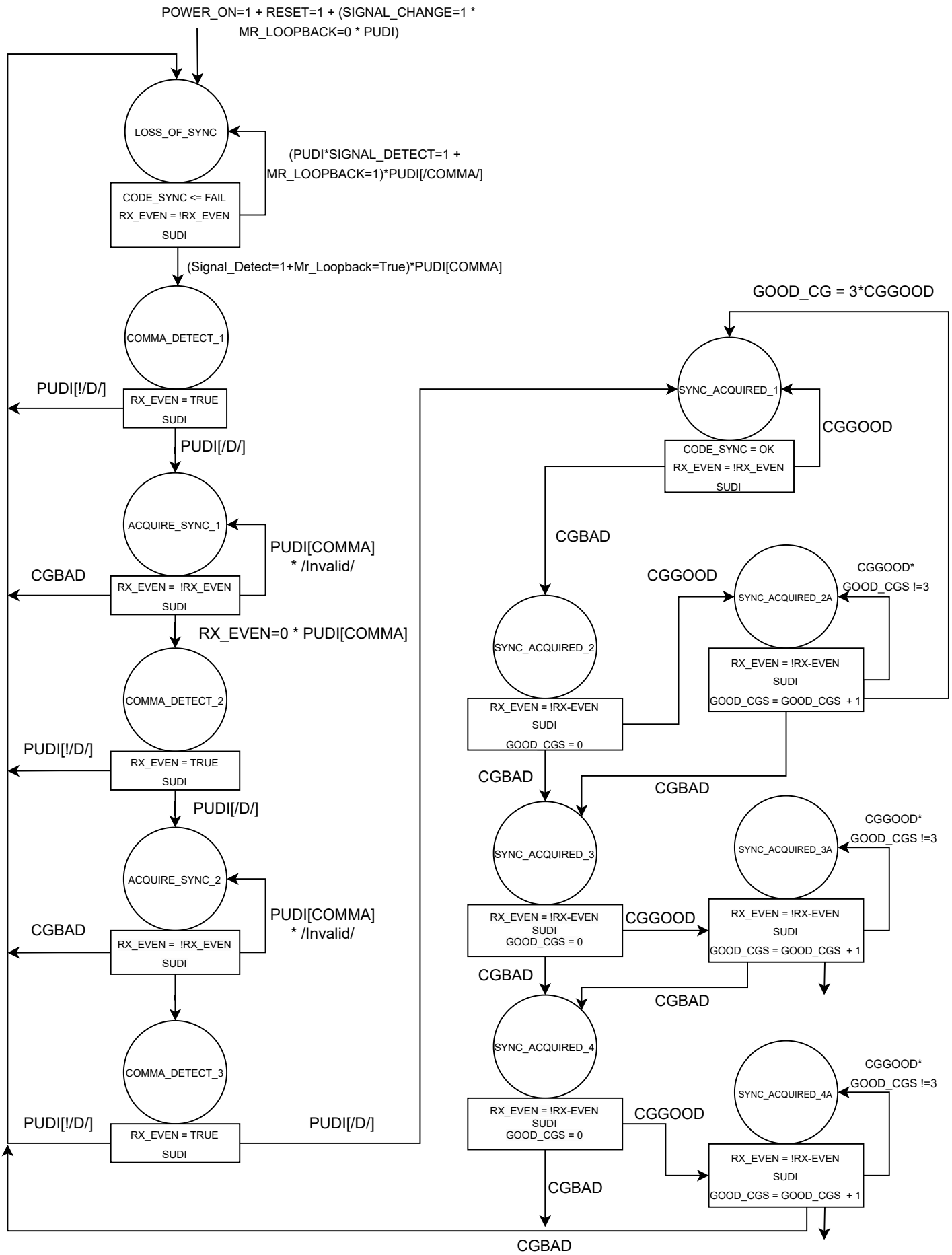


Figura 4: Diagrama de estados TRANSMIT_CODE_GROUP

- En el estado incial `GENERATE_CODE_GROUPS`, si se recibe un carácter especial se va a `SPECIAL_GO`, y se realiza la codificación de ese carácter.
- Si es un carácter de data, se va a `DATA_GO` y se codifica en ese estado.
- Si es un carácter de IDLE, primero se envía la comma `K28.5` en el estado `IDLE_DISPARITY_OK` y después el carácter `D16.2` en `IDLE_I2B`

2.3. SYNCHRONIZE


POWER_ON=1 + RESET=1 + (SIGNAL_CHANGE=1 *
MR_LOOPBACK=0 * PUDI)
LOSS_OF_SYNC
(PUDI*SIGNAL_DETECT=1 +
MR_LOOPBACK=1)*PUDI[/COMMA/]
CODE_SYNC <= FAIL
RX_EVEN = !RX_EVEN
SUDI
(Signal_Detect=1+Mr_Loopback=True)*PUDI[COMMA]
COMMA_DETECT_1
PUDI[!/D/]
RX_EVEN = TRUE
SUDI
PUDI[/D/]
ACQUIRE_SYNC_1
PUDI[COMMA]
* /Invalid/
CGBAD
RX_EVEN = !RX_EVEN
SUDI
RX_EVEN=0 * PUDI[COMMA]
COMMA_DETECT_2
PUDI[!/D/]
RX_EVEN = TRUE
SUDI
PUDI[/D/]
ACQUIRE_SYNC_2
PUDI[COMMA]
* /Invalid/
CGBAD
RX_EVEN = !RX_EVEN
SUDI
COMMA_DETECT_3
PUDI[!/D/]
RX_EVEN = TRUE
SUDI
PUDI[/D/]
GOOD_CG = 3*CGGOOD
SYNC_ACQUIRED_1
CGGOOD
CODE_SYNC = OK
RX_EVEN = !RX_EVEN
SUDI
CGBAD
SYNC_ACQUIRED_2
CGGOOD
SYNC_ACQUIRED_2A
CGGOOD*
GOOD_CGS !=3
RX_EVEN = !RX-EVEN
SUDI
GOOD_CGS = 0
RX_EVEN = !RX-EVEN
SUDI
GOOD_CGS = GOOD_CGS + 1
CGBAD
CGBAD
SYNC_ACQUIRED_3
SYNC_ACQUIRED_3A
CGGOOD*
GOOD_CGS !=3
RX_EVEN = !RX-EVEN
SUDI
GOOD_CGS = 0
CGGOOD
RX_EVEN = !RX-EVEN
SUDI
GOOD_CGS = GOOD_CGS + 1
CGBAD
CGBAD
SYNC_ACQUIRED_4
SYNC_ACQUIRED_4A
CGGOOD*
GOOD_CGS !=3
RX_EVEN = !RX-EVEN
SUDI
GOOD_CGS = 0
CGGOOD
RX_EVEN = !RX-EVEN
SUDI
GOOD_CGS = GOOD_CGS + 1
CGBAD
CGBAD

El módulo de sincornización se encarga de unir los módulos de recepción y de transmisión determinando mediante los grupos de 10 bits que recibe por su entrada (provenientes del módulo de transmisión) si el módulo de recepción está listo para entrar en estado de operación.

2.4. RECEIVE

Este módulo se encarga de recibir los datos ya sincronizados gracias a la máquina de `SYNCHRONIZE`, y decodificarlos a 8 bits.

Figura 5: Módulo RECEIVE

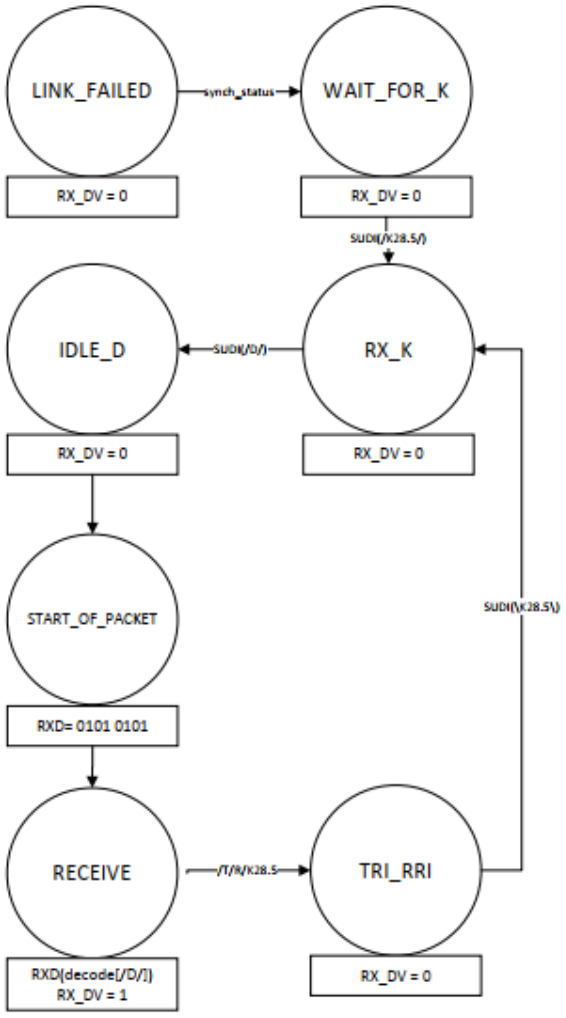


Figura 6: Diagrama de estados para RECEIVE

RECEIVE funciona de la siguiente manera:

- Se espera la comma K28.5 y un carácter de datos. Una vez recibido, se envía el preámbulo `8'b0101_0101` por medio de `START_OF_PACKET`.
- Una vez enviado, se pasa a `RECEIVE`, este estado se encarga de traducir los carácteres de data de 10 bits a 8 bits.
- En todo momento se esperan los carácteres /T/R/K28.5. Si se reciben estos tres code-groups seguidos se pasa al estado `TRI_RRI` que indica el final de la trama. Si se recibe una comma, se regresa a `RX_K`

2.5. PARAMETERS

`PARAMETERS.v` es un archivo de Verilog que contiene la definición de los parámetros para las codificaciones en 8 bits y 10 bits. Se hizo con el propósito de instanciarlo en los módulos `RECEIVE`, `SYNCHRONIZE` y hacer el acceso a los parámetros más simple.

Para efectos de este trabajo, se implementaron los siguientes code groups de data:

Tabla 1: Valores DATA codificados

Code Group Name	Octet BITS	RD +	RD -
D0.0	0000 0000	10 0111 0100	01 1000 1011
D1.0	0000 0001	01 1101 0100	10 0010 1011
D2.1	0010 0010	10 1101 1001	01 0010 1001
D3.2	0100 0011	10 0010 1011	11 0001 0101
D4.3	0110 0100	11 0101 0011	00 1010 1100
D5.4	1000 0101	10 1001 1101	10 1001 0010
D6.5	1010 0110	01 1001 1010	01 1001 1010
D7.6	1100 0111	11 1000 0110	00 0111 0110
D8.7	1110 1000	11 1001 0001	00 0110 1110
D9.7	1110 1001	10 0101 0001	10 0101 1110

Para los code groups especiales, se codificó toda la tabla establecida por el estándar 802.3

Code Group Name	Octet Value	Octet Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj	Notes
K28.0	1C	000 11100	001111 0100	110000 1011	1
K28.1	3C	001 11100	001111 1001	110000 0110	1,2
K28.2	5C	010 11100	001111 0101	110000 1010	1
K28.3	7C	011 11100	001111 0011	110000 1100	1
K28.4	9C	100 11100	001111 0010	110000 1101	1
K28.5	BC	101 11100	001111 1010	110000 0101	2
K28.6	DC	110 11100	001111 0110	110000 1001	1
K28.7	FC	111 11100	001111 1000	110000 0111	1,2
K23.7	F7	111 10111	111010 1000	000101 0111	
K27.7	FB	111 11011	110110 1000	001001 0111	
K29.7	FD	111 11101	101110 1000	010001 0111	
K30.7	FE	111 11110	011110 1000	100001 0111	

NOTE 1—Reserved.

NOTE 2—Contains a comma.

Figura 7: Code groups especiales codificados

3. Plan de Pruebas

Como prueba mínima del funcionamiento correcto del bloque de PCS, se utilizo una prueba de loopback que envió una trama completa de Ethernet, según el diagrama de tiempos de la siguiente figura:

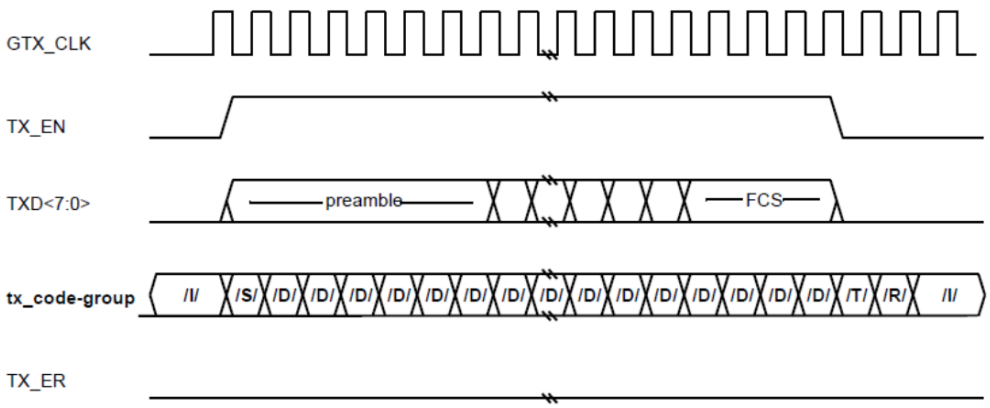


Figura 8: Prueba de loopback para la capa PCS

Ahora esta prueba es para todo el bloque de PCS por lo que en realidad fue subdividida en múltiples pruebas que luego fueron subdividas en incluso mas pruebas para cada submódulo para asegurarse de que cada submódulo funcionara de manera correcta ya que esto aseguraría el funcionamiento correcto del bloque.

En este caso solo se mencionara el plan de pruebas para todo el bloque ya que al verificar el correcto comportamiento de estas pruebas se puede verificar el correcto comportamiento de todas las pruebas individuales de cada submódulo.

- **Prueba 1:** Se realiza un RESET para limpiar las salidas y variables internas necesarias y para entrar en los estados iniciales de los cuatro submódulos al mismo tiempo, se pasó la prueba.
- **Prueba 2:** Se comienzan a mandar sets de /I/ desde $TRANSMIT\_ORDERED\_SET$ hacia $TRANSMIT\_CODE\_GROUP$ y este al recibirlos manda los codegroups correspondientes a la maquina de $SYNCHRONIZATION$ para que esta empiece la sincronización, se pasó la prueba.
- **Prueba 3:** Al recibir tres 6 sets de /I/ y mandar los codegroups correspondientes a /K28.5/ y / D16.2/ tres veces a la maquina de $SYNCHRONIZATION$ se completa la sincronización y se prepara la maquina de $RECEIVE$ para recibir datos, se pasó la prueba.
- **Prueba 4:** Al tener la entrada TX_EN en alto y la entrada TX_ER en bajo se inicia la transmisión de datos mandando mandando el set /S/ y teniendo en la salida RXD el preámbulo.
- **Prueba 5:** Se mandan 10 datos diferentes por la entrada TXD y estos salen por la salida RXD luego de ser traducidos de 8 a 10 bits en el bloque de $TRANSMIT$ y traducidos de vuelta a 8 bits en la maquina de $RECEIVE$, se pasó la prueba.

- **Prueba 6:** Al terminar el paquete de datos se pone la entrada TX_EN en bajo y se mandan sets de /T/ y de /R/ desde el bloque de $TRANSMIT$ para indicarle el fin del paquete a los demás submódulos, se pasó la prueba.
- **Prueba 7:** Luego de mandar los sets de /T/ y /R/ se vuelven a mandar sets de /I/, se pasó la prueba.

4. Instrucciones de utilización de la simulación

Para hacer funcionar la simulación en todos los casos que especifica el plan de pruebas solo se necesita utilizar el comando make en una terminal que se encuentre en una carpeta que contenga todos los archivos:

```
$ make
```

Figura 9: Comando make

Este comando va a tomar los comandos que se encuentran en el makefile y los va a correr en la terminal, los comandos que están en el makefile son los siguientes:

```
all:
        iverilog  PCS.v
        iverilog  TRANSMIT_ORDERED_SET.v
        iverilog  TRANSMIT_CODE_GROUP.v
        iverilog  synch.v
        iverilog  RECEIVE.v
        iverilog tester.v
        iverilog testbench.v
        iverilog -o salida.vvp testbench.v
        vvp salida.vvp
        gtkwave pcs.vcd
```

Figura 10: Comandos presentes en el makefile

Primero hay 7 comandos para compilar el wrapper, los cuatro submódulos, el tester y el testbench, luego un comando para generar la salida y luego se abre gtkwave para poder ver las formas de onda de los resultados.

5. Ejemplos de los resultados

El resultado principal del proyecto es la prueba de loopback que en la cual se envió una trama completa de Ethernet tal y como se propuso en la figura 8 y puede ser observado en la siguiente figura:

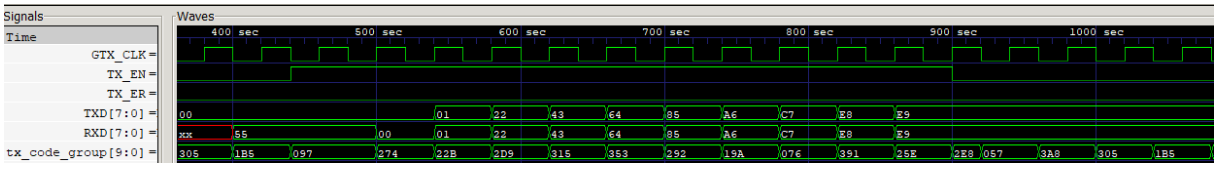


Figura 11: Trama completa de Ethernet en el bloque de PCS creado

Podemos afirmar que se logro la prueba propuesta debido a que se puede observar como se pasa de mandar codegroups de /I/, que corresponden a /K28.5/ que en tx_even positivo se traduce a 305 en hex y a /D16.2/ que en tx_even negativo se traduce a 1B5 en hex, a mandar primero un codegroup de /S/ que se traduce a 097 en hex y luego 10 codegroups de datos, que corresponden a las traducciones de 10 bits de los datos que llegan por TXD, y por ultimo se mandan un codegroup de /T/ que en tx_even negativo se traduce a 2E8 en hex y 2 codegroups de /R/ que en tx_even positivo se traduce a 057 en hex y en tx_even negativo se traduce a 3A8 en hex, luego de mandar el ultimo codegroup de /R/ se observa como se vuelven a mandar codegroups de /I/ indicando que se vuelve al estado de IDLE del bloque. La razón por la cual se mandan 2 codegroups de /R/ a diferencia de la figura 8 es debido a que para poder volver a mandar codegroups de /I/ se necesita que el primer codegroup que se mande sea en tx_even positivo ya que sera una comma (/K28.5/) y la maquina de sincronización necesita que estas se manden en tx_even positivo para poder verificar que el bloque esta sincronizado. Toda esta prueba de loopback verifica que se pasaron las pruebas 4,5,6 y 7 que estaban relacionadas con los sets mandados en el envió de una trama completa de Ethernet. A continuación se mostraran los resultados que verifican las pruebas 1,2 y 3 que están relacionadas con el inicio de las maquinas y el proceso de sincronización de las mismas.

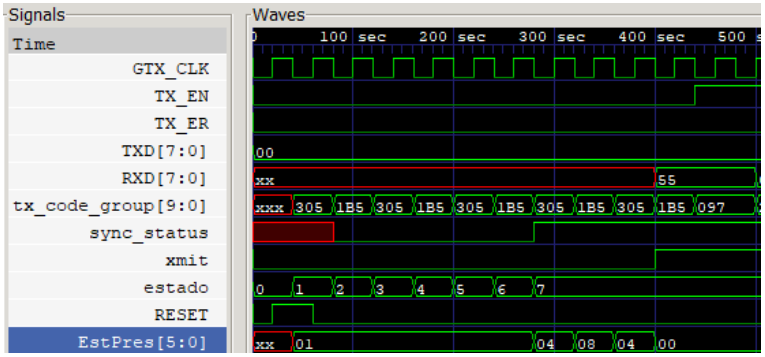


Figura 12: Proceso de Sincronización y RESET de las maquinas

En esta figura primero se muestra como al llegar el RESET se empiezan a mandar codegroups de /I/ indicando que se paso a los estados iniciales de ambos submódulos de TRANSMIT y se muestran las variables internas llamadas estado y EstPres que corresponden a los estados de las maquinas de SYNCHRONIZE y de RECEIVE respectivamente para mostrar como se pasan a los estados iniciales de ambas maquinas al llegar el RESET, este comportamiento verifica la prueba 1. Luego se observa como al llegar varios codegroups de /K28.5/ y /D16.2/ que corresponden a sets de /I/ la maquina de sincronización va cambiando de estado a medida que se va sincronizando, cumpliendo así la prueba 2. Por ultimo se observa que después de que llega el sexto set de /I/ correspondiente a 1B5 que es un codegroup de /D16.2/ en el siguiente flanco negativo de GTX_CLK se activa la variable sync_status indicando que se logro sincronizar la maquina y permitiendo así que cambie el estado de la maquina de RECEIVE, logrando así pasar la prueba 3, se nota que la maquina de RECEIVE pasa al estado de RECEIVE (00), en el cual primero recibirá un preámbulo mientras se reciban codegroups de /S/ y luego traducirá los datos que le lleguen por SUDI, cuando la variable xmit pasa a alto.

6. Conclusiones y recomendaciones

6.1. Conclusiones

La forma es que esta subcapa recibe y transmite datos es, realmente, mucho más compleja de lo que aquí se trabajó, sin embargo, da una idea de cómo es trabajar en la industria construyendo subcapas de protocolos complejos, como lo es el Ethernet. En lo como grupo trabajamos, fue muy importante la utilización estricta de los nombres que el standard indicaba, pues al momento de conectar los módulos, esto podía desencadenar problemas, pues Verilog no iba a comprender que se trataba del mismo puerto o la misma señal, por ejemplo. Las pruebas que se establecieron para demostrar el correcto funcionamiento de los sub módulos fue exitoso, lográndose una correcta sincronización mediante la recepción de datos por medio del **tester.v**; en una situación real, estos datos provendrían de alguna otra subcapa que debe de entre conectarse con la nuestra y así sucesivamente, he aquí la importancia de este proyecto.

En síntesis, se logró pasar todas las puebas necesarias para lograr las pruebas establecidas en el standard, tal como indica la figura 8 y en la sección de resultados se ofrece una muy amplia descripción de cada una de las señales y su interpretación con respecto a los code groups de la figura 7.

6.2. Recomendaciones

Posiblemente la recomendación más importante sea que para trabajar proyectos de este tipo, se utilicen herramientas de versionamiento de código, tal como lo es GitHub, pues al ser un trabajo que se realizó entre cuatro personas, manejar el código por aparte podía ser extremadamente complicado, así que esta herramienta facilitó y aceleró la forma en que trabajamos como equipo. También, al momento de comenzar a diseñar alguno de los módulos es necesario comprender el nombre de las señales y las variables que estos manejan y no existe forma de hacer esto que no sea leyendo el estándar de la IEEE correspondiente (o de cualquier otra institución) ya que será la única manera de que se logre hacer que las señales funcionen conectadas a los sub módulos internos. También, para visualizar las pruebas que se realizaron a la subcapa, fue necesario utilizar varias herramientas como Icarus Verilog y GTK Wave, esto significa correr múltiples comandos en terminal que pueden generar confusión; para evitar esto, todos los comandos pueden enviarse a un Makefile y con un solo comando se ahorra mucho tiempo y se evitan errores; así que otra recomendación sería utilizar un Makefile. Como cada uno de los módulos utilizaba los mismos parámetros de code groups, se enviaron estos parámetros a un archivo que luego pudo ser importado a cada uno de los módulos y así trabajar de una forma más ordenada, optimizando el espacio en el archivo.