Universidad de Costa Rica

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

Avance de Proyecto
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

17 de noviembre 2024

Índice

1. Descripción arquitectónica

La estructura de este proyecto se enfocará 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 será implementado para un diseño 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. A continuación se analiza la estructura de cada uno de los módulos con sus respectivas diagramas de estados, entradas y salidas.

1.1. 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 2.

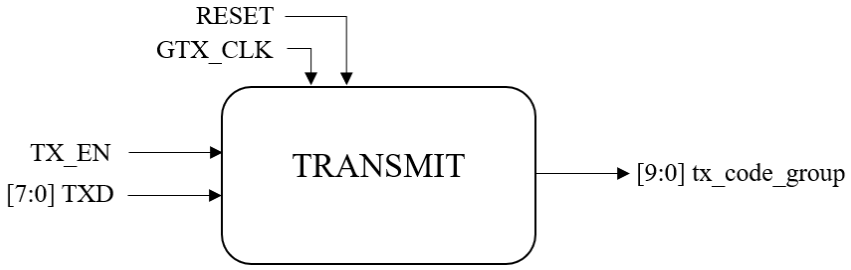


Figura 1: Diagrama de bloques del transmisor.

Este módulo cuenta con dos máquinas de estados que se interconectan dentro de un mismo archivo. 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, a continuación se puede observar el diagrama de estados del primer módulo **transmit-ordered-set**.

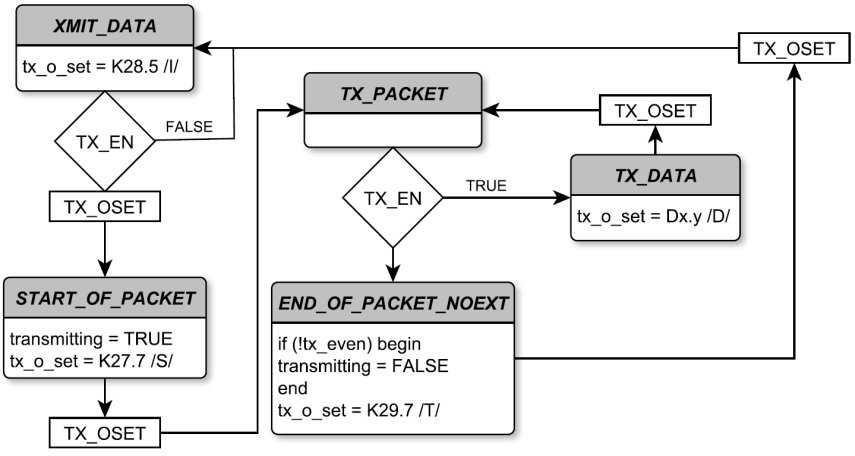


Figura 2: Diagrama ASM del módulo *transmit-ordered-set*

Para este módulo se tiene 5 estados tal como se observa en la Tabla 1:

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

Estado	Código
XMIT_DATA	0 0001
START_OF_PACKET	0 0010
TX_PACKET	0 0100
TX_DATA	0 1000
END_OF_PACKET_NOEXT	1 0000

Para la primer máquina de estados la función es poder recibir 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. Para 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.

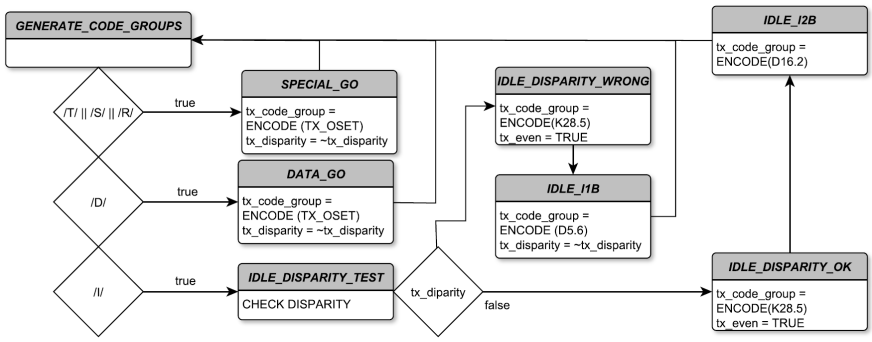


Figura 3: Diagrama ASM del módulo *transmit-code-group*

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

1.2. Sincronizador

El módulo del sincronizador posee 9 estados, el cual ha sido codificado según la tabla 3.
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:

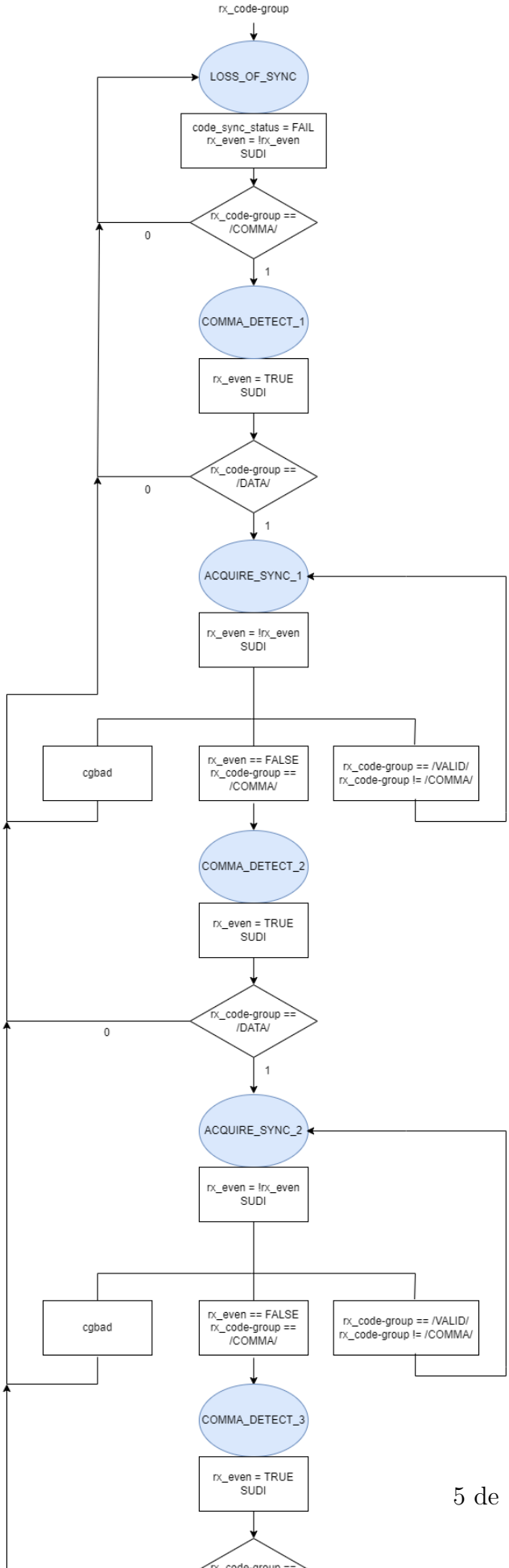
Tabla 2: Variables del módulo transmit

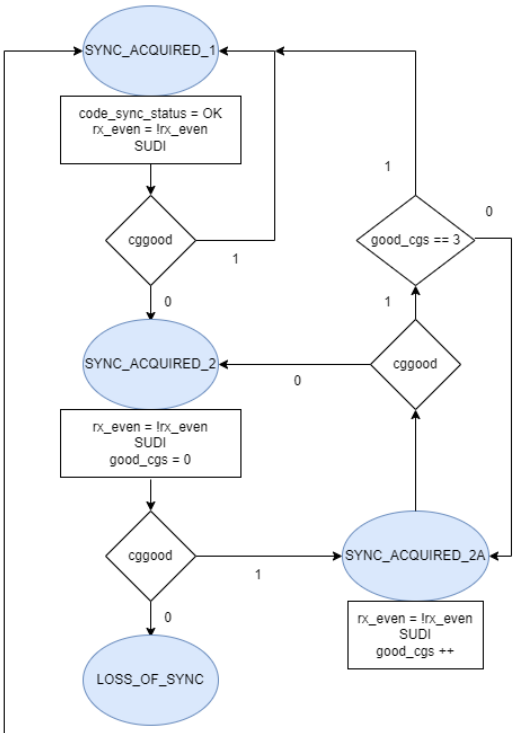
Tipo	Nombre	Descripción
Input	TX_EN	Inicio a la trasnmisión de un dato
Input	TXD	8 bits que se desean codificar
Input	GTX_CLK	Es el reloj del transmit
Input	RESET	Reset de todos los módulos
Interna	tx_o_set	Dato que se recibe para enviar
Interna	tx_even	Indica la paridad del byte
Output/Input	TX_OSET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

Tabla 3: Estados del módulo del sincronizador.

Estado	Código
LOSS OF SYNC	0 0000 0001
COMMA DETECT 1	0 0000 0010
ACQUIRE SYNC 1	0 0000 0100
COMMA DETECT 2	0 0000 1000
ACQUIRE SYNC 2	0 0001 0000
COMMA DETECT 3	0 0010 0000
SYNC ACQUIRED 1	0 0100 0000
SYNC ACQUIRED 2	0 1000 0000
SYNC ACQUIRED 2A	1 0000 0000

- 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 12 code groups especiales cuyas etiquetas inician con K. VALID reprenta entonces los 22 code groups (44 en total considerando RD).
- 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.


rx_code-group
LOSS_OF_SYNC
code_sync_status = FAIL
rx_even = !rx_even
SUDI
rx_code-group == /COMMA/
0
1
COMMA_DETECT_1
rx_even = TRUE
SUDI
rx_code-group == /DATA/
0
1
ACQUIRE_SYNC_1
rx_even = !rx_even
SUDI
cgbad
rx_even == FALSE
rx_code-group == /COMMA/
rx_code-group == /VALID/
rx_code-group != /COMMA/
COMMA_DETECT_2
rx_even = TRUE
SUDI
rx_code-group == /DATA/
0
1
ACQUIRE_SYNC_2
rx_even = !rx_even
SUDI
cgbad
rx_even == FALSE
rx_code-group == /COMMA/
rx_code-group == /VALID/
rx_code-group != /COMMA/
COMMA_DETECT_3
rx_even = TRUE
SUDI


SYNC_ACQUIRED_1
code_sync_status = OK
rx_even = !rx_even
SUDI
cggood
1
0
good_cgs == 3
1
0
1
SYNC_ACQUIRED_2
0
cggood
rx_even = !rx_even
SUDI
good_cgs = 0
cggood
1
SYNC_ACQUIRED_2A
0
rx_even = !rx_even
SUDI
good_cgs ++
LOSS_OF_SYNC

1.3. Receptor

Finalmente, el módulo del receptor trabajará inicialmente con 8 estados. Sin embargo, estos estados podrían ser simplificados durante el proceso, lo que podría reducir su número en la implementación final.

Estado	Código
LINK_FAILED	0 0000 0001
WAIT_FOR_K	0 0000 0010
RX_K	0 0000 0100
IDLE_D	0 0000 1000
START_OF_PACKET	0 0001 0000
RECEIVE	0 0010 0000
RX_DATA	0 0100 0000
TRI_RRI	1 0000 0000

Tabla 4: Declaración de Estados del Receptor

El módulo cuenta con las siguientes entradas y salidas:

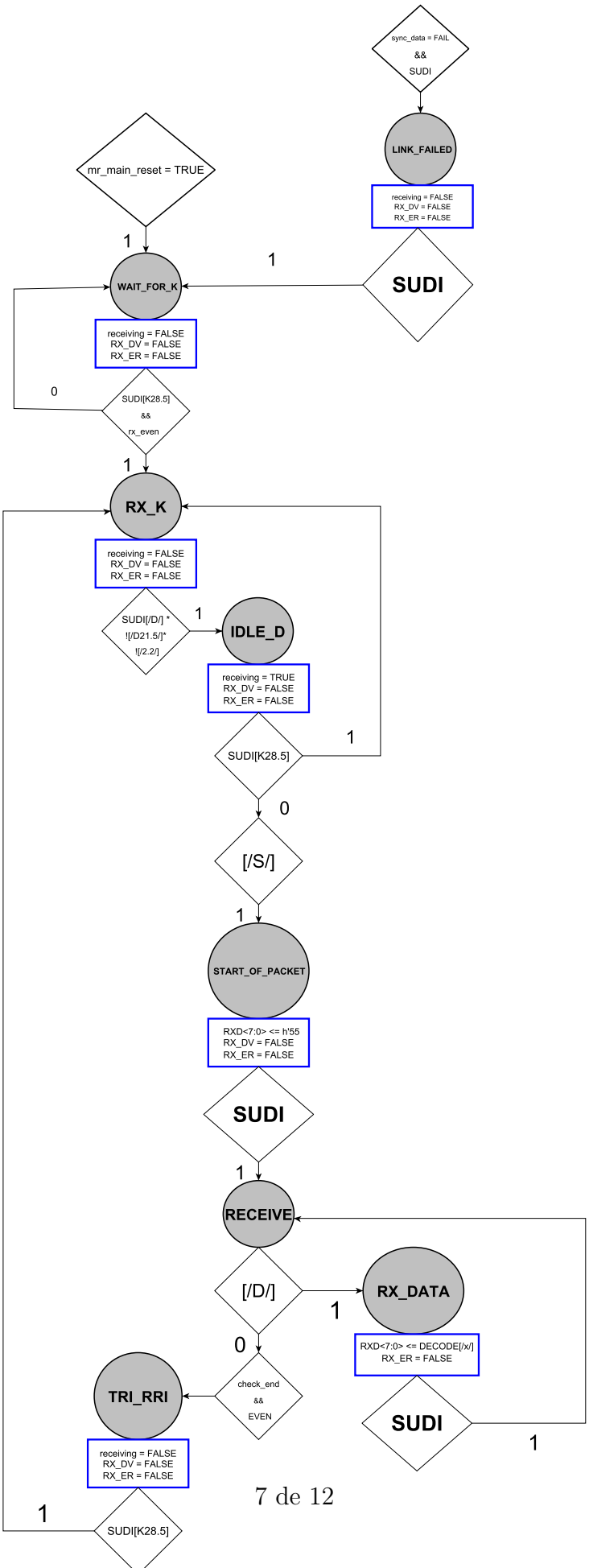
Tipo	Nombre	Descripción
Input	`RX_CLK`	Reloj de sincronización del receptor.
Input	`mr_main_reset`	Señal de reinicio principal.
Input	`sync_status`	Estado de sincronización del receptor.
Input	`SUDI`	Grupo de códigos recibidos (`10 bits`).
Input	`rx_even`	Señal o variable que indica si el receptor está 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 5: Entradas y salidas del módulo receptor

Siguiendo el estándar Ethernet, específicamente la Cláusula 36 y enfocándose en la parte del receptor discutida en clase, se realizaron diversas simplificaciones:

Proceso de Simplificación:

- **Eliminación de Secciones:** Se eliminaron las partes relacionadas con la `Auto-Negotiation` y `Carrier Sense` las variables de esos módulos que aparezcan en el receptor se van a ignorar.
- **Exclusión de Casos de Errores:** No se incluyeron las situaciones de error en esta implementación, unicamente se desea observar el comportamiento general en el manejo de datos.
- **Integración de Diagramas del Receptor:** Se eliminó el tercer diagrama del receptor, y a partir de los dos diagramas restantes se realizó una unificación. Esto permitió trabajar con un único diagrama ASM, que servirá como guía para implementar su funcionamiento en Verilog.


sync_data = FAIL
&&
SUDI
LINK_FAILED
receiving = FALSE
RX_DV = FALSE
RX_ER = FALSE
mr_main_reset = TRUE
1
1
WAIT_FOR_K
SUDI
receiving = FALSE
RX_DV = FALSE
RX_ER = FALSE
0
SUDI[K28.5]
&&
rx_even
1
RX_K
receiving = FALSE
RX_DV = FALSE
RX_ER = FALSE
SUDI[/D/] *
![/D21.5/]*
![/2.2/]
1
IDLE_D
receiving = TRUE
RX_DV = FALSE
RX_ER = FALSE
SUDI[K28.5]
1
0
[/S/]
1
START_OF_PACKET
RXD<7:0> <= h'55
RX_DV = FALSE
RX_ER = FALSE
SUDI
1
RECEIVE
[/D/]
1
RX_DATA
RXD<7:0> <= DECODE[x/]
RX_ER = FALSE
SUDI
1
0
check_end
&&
EVEN
TRI_RRI
receiving = FALSE
RX_DV = FALSE
RX_ER = FALSE
1
SUDI[K28.5]

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 `RECEIVE`, si se detecta un dato válido (`/D/`), el sistema avanza al estado `RX_DATA`, donde los datos se decodifican. En este estado, `RXD` toma valores decodificados de `SUDI`, y `RX_ER` se mantiene en `FALSE`. Si se detecta el indicador de `check_end` en un ciclo par, el sistema transita 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. Tareas Realizadas

2.1. Transmisor

En el transmisor se implementaron las máquinas de estados que se describen en la sección anterior, se implementa al código Verilog la estructura con los estados para posteriormente implementar la parte del diseño conductual de cada paso. Para poder realizar la función del transmisor 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.

[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 6: 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 7: Valores de Special code-group.

Figura 8: Tabla de code-goroups para la codificación.

2.2. Sincronizador

En el archivo *lookup.v*, se ha implementado los métodos necesarios para implementar los dos criterios que gobiernan la transición de estados. Se muestra un ejemplo en la figura 9. Para la implementación, se utilizó macros. Presentó una dificultad inicial por sus reglas de sintaxis, y para resolver el problema fue necesario utilizar diferentes fuentes de información. [1] [2]

Con la implementación de estos métodos, se espera que la implementación de estados sea claro y conciso.

```
`define IsData(rx_code-group) \
    ( \
    (rx_code-group == `D5_6_10N) || (rx_code-group == `D5_6_10P) || \
    (rx_code-group == `D16_2_10N) || (rx_code-group == `D16_2_10P) || \
    (rx_code-group == `D0_0_10N) || (rx_code-group == `D0_0_10P) || \
    (rx_code-group == `D16_0_10N) || (rx_code-group == `D16_0_10P) || \
    (rx_code-group == `D0_1_10N) || (rx_code-group == `D0_1_10P) || \
    (rx_code-group == `D16_1_10N) || (rx_code-group == `D16_1_10P) || \
    (rx_code-group == `D0_2_10N) || (rx_code-group == `KD0_2_10P) || \
    (rx_code-group == `D0_3_10N) || (rx_code-group == `D0_3_10P) || \
    (rx_code-group == `D16_3_10N) || (rx_code-group == `D16_3_10P) || \
    (rx_code-group == `D0_4_10N) || (rx_code-group == `D0_4_10P) ||
    )
```

Figura 9: Método para identificar DATA.

2.3. Receptor

Hasta el momento, se han realizado los siguientes avances en la implementación del receptor:

- **Definición de parámetros y estados:**
  - Se han declarado los estados principales necesarios para el receptor: `LINK_FAILED`, `WAIT_FOR_K`, `RX_K`, `IDLE_D`, `START_OF_PACKET`, `RECEIVE`, `RX_DATA`, y `TRI_RRI`.
  - Se definieron los parámetros asociados a los code-groups relevantes, incluyendo:
    - `/K28.5/`, `/D16.2/`, `/S/`, `/D21.5/`, y `/D2.2/`.
- **Implementación de la lógica secuencial:**
  - Se implementó la lógica secuencial para controlar la transición entre estados en el flanco negativo del reloj `RX_CLK`.
  - Se consideraron las siguientes condiciones:
    - Transición al estado `WAIT_FOR_K` cuando se detecta un reinicio (`mr_main_reset`).
    - Transición al estado `LINK_FAILED` cuando se pierde la sincronización (`sync_status = FAIL`).
- **Implementación parcial de la lógica combinacional:**
  - Se desarrolló la lógica combinacional para los estados:
    - `LINK_FAILED`: Verificación de sincronización y transición a `WAIT_FOR_K` al detectar un símbolo válido (`SUDI`).

- `WAIT_FOR_K`: Transición a `RX_K` al detectar `/K28.5/` con `rx_even = TRUE`.
- `RX_K`: Transición a `IDLE_D` si los símbolos recibidos no coinciden con `/D21.5/` o `/D2.2/`.
- `IDLE_D`: Transición a `RX_K` si se detecta `/K28.5/`, o a `START_OF_PACKET` si se detecta `/S/`.
- `START_OF_PACKET`: Inicialización de las señales `RXD`, `RX_DV`, y transición al estado `RECEIVE`.

- **Inicialización de salidas del receptor:**
  - La variable interna `rx_lpi_active` ha sido inicializada correctamente en los estados implementados.
  - Se han configurado las señales de salida `RX_DV`, `RX_ER` y `RXD` en los cinco estados implementados hasta el momento.

3. Tareas pendientes

Para finalización de la implementación e interconexión de los módulos del proyecto se tienen varios elementos por abordar. Se debe desarrollar la funcionalidad de cada módulo, garantizar la correcta transición de estados y validar el diseño completo conectado. La integración de cada tarea pendiente se debe realizar en las próximas semanas por lo que en la Figura 10 se visualiza el cronograma de como se trabajará la última parte del proyecto.

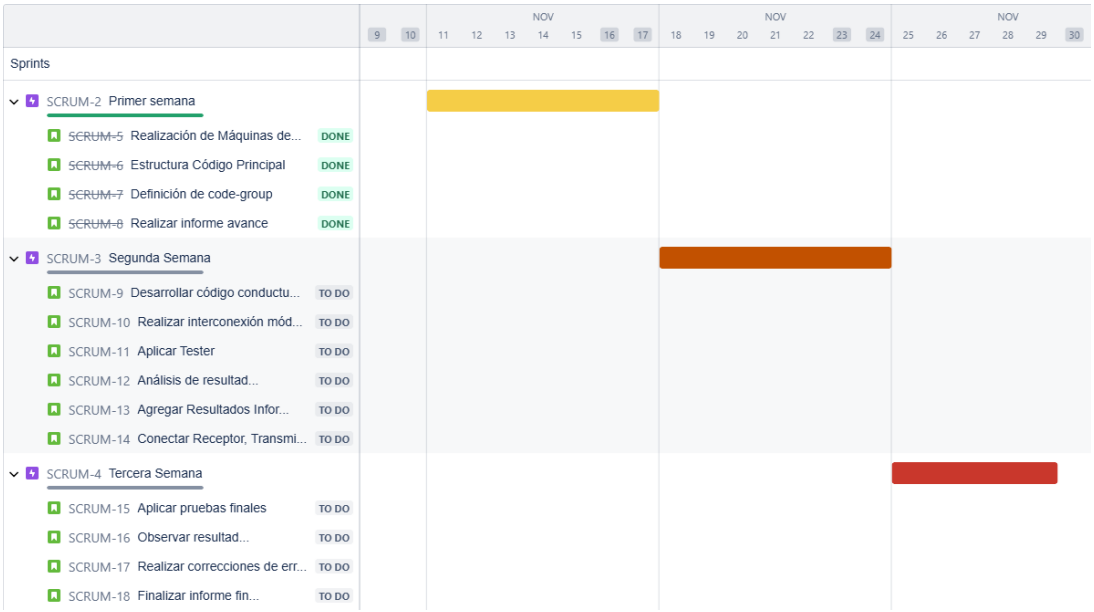


Figura 10: Cronograma de trabajo SCRUM

3.1. Transmisor

En el caso del transmisor, las tareas pendientes se centran en finalizar su implementación y validar su correcto funcionamiento. Estas incluyen:

1. Implementar el código con la lógica combinacional.
2. Interconectar ambos módulos por medio del testbench.
3. Comprobar el funcionamiento correcto mediante simulaciones y pruebas.

3.2. Sincronizador

Se han implementado los métodos necesarios para transicionar entre estados. Entre los pendientes en el módulo de sincronización están la implementación de los estados, la implementación del módulo tester y la implementación del módulo testbench.

3.3. Receptor

A pesar de los avances realizados, aún quedan tareas pendientes en la implementación:

- **Implementación de la lógica en los estados finales:**
  - Completar la lógica de los estados:
    - `RECEIVE`: Implementar las condiciones para evaluar los de datos recibidos y decidir la transición a `RX_DATA` o `TRI_RRI`.
    - `RX_DATA`: Decodificar los datos recibidos de `SUDI`, asignar valores a `RXD`, y gestionar la transición al siguiente estado según sea necesario.
    - `TRI_RRI`: Finalizar la recepción y reiniciar las variables internas antes de regresar a `RX_K`.
- **Implementación de la función de decodificación:**
  - Crear una función o módulo para decodificar los símbolos de 10 bits (`SUDI`) en valores de 8 bits (`RXD`).
  - Integrar esta función en el estado de `RX_DATA`
- **Manejo del fin de paquete de datos:**
  - Implementar la señal `check_end` para verificar la finalización de paquete de datos en los estados `RECEIVE` y `RX_DATA`.
- **Pruebas y simulación:**
  - Diseñar casos de prueba para verificar la funcionalidad de cada estado.
  - Simular el diseño en un `tester` individual para validar el comportamiento del receptor bajo diferentes escenarios.

Referencias

1. ChipVerify. Verilog define macro. https://www.chipverify.com/verilog/verilog-define-macros.
2. ——. Systemverilog define macro. https://www.chipverify.com/systemverilog/systemverilog-define-macro.