Capítulo 5

Diseño del circuito Serializador-Deserializador

5.1. Introducción

En el capítulo 2 de este proyecto se presentó el marco de aplicación de este circuito y el problema que venía a solucionar: serializar la comunicación AER entre matrices de neuronas que formaban parte de una red de procesamiento multicapa de alta velocidad. El capítulo 3 se dedicó a la tecnología que se va a usar para solventar este problema basada en un esquema diferencial y de baja amplitud, muy robusto a la hora de implementar sistemas de comunicación entre chips. El objetivo ahora es presentar el circuito que hay que añadir al generador AER (que da los datos en formato paralelo) para conseguir la serialización del enlace.

En el apartado 2.7 del capítulo 2 ya se esbozó la estructura del sistema a utilizar. Esta se basaba en utilizar un sistema de codificacación de los datos eficiente que permitiera enviar sobre la misma línea diferencial los datos y el reloj, teniéndose en el receptor circuitos de sincronismo capaces de recuperar este reloj y decodificar los datos correctamente. Para representar los datos en la línea se propuso una codificación Manchester que cumplía con los requerimientos de sincronización anteriormente expuestos y, además, permitía una implementación hardware sencilla y robusta. Si a todo esto añadimos un driver y un receptor LVDS (de los que se habló en el capítulo 3), completamos la arquitectura del nivel físico que se quiere implementar.

El diseño se realizará para una velocidad nominal de 1 Gbps, que, usando un código Manchester, se traduce en la necesidad de generar pulsos del orden de 500 ps. Para conseguir velocidades en el enlace del orden de Gbps se decidió utilizar una tecnología de 90 nm suministrada por *STMicroelectronics*. Se trata de una tecnología hasta cierto punto novedosa, por lo que otro de los retos del proyecto estaba en explorar las capacidades de esta tecnología e investigar acerca del flujo de diseño que es necesario seguir con ella. *STMicroelectronics* suministra su tecnología junto con una serie de librerías digitales, que han sido las que mayoritariamente se han usado para el diseño del circuito. Además, también nos proporcionaba unos drivers y receptores LVDS totalmente compatibles con el estándar y que se adaptaban a las necesidades de velocidad que se

habían propuesto. Por tanto, se ha usado estos circuitos para implementar el enlace, habiéndose realizado para ellos un estudio previo que se mostrará a lo largo de este documento.

La tecnología de 90 nm tiene como tensión de alimentación estándar 1 ó 1.2 V. Todo el deserializador-serializador, así como la parte digital de los drivers, funcionan a estas tensiones de alimentación (en concreto el diseño se ha realizado para 1V). Para permitir la integración de la tecnología con sus precedecesoras, el fabricante también proporciona transistores para ser utilizados en circuitos con tensiones de alimentación de 2.5 y 3.3 V. No obstante, las prestaciones de estos transistores son inferiores a las de los transistores típicos de la tecnología. Parte de los drivers y receptores LVDS de librerías están construidos con este tipo de transistores.

El objetivo de este capítulo es presentar la arquitectura del Serializador-Deserializador elegido y mostrar el proceso de diseño seguido para llegar hasta una versión funcional del mismo. Se intentó descomponer el diseño al máximo en bloques para que fuera lo más modular y lo menos sensible a errores de diseño posible. El esquema general del diseño se muestra en la figura 5.1.

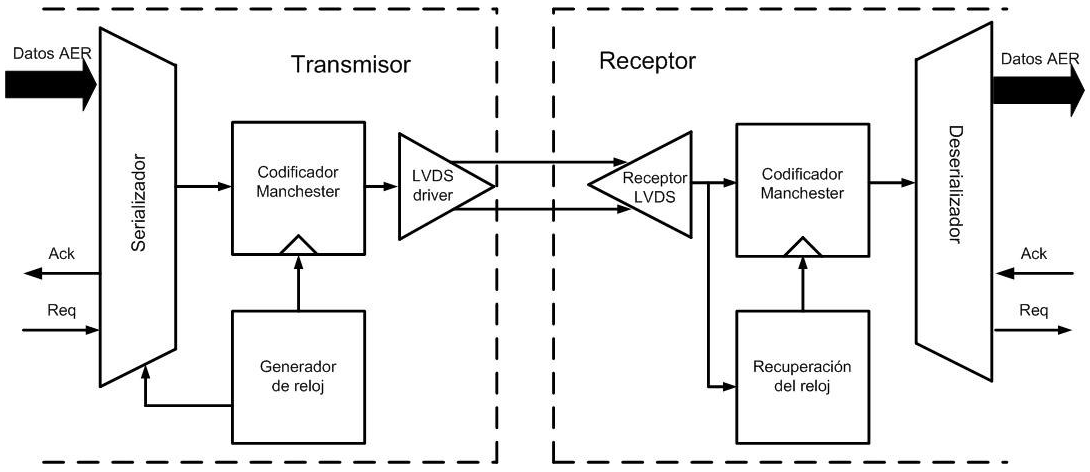


Figura 5.1: Esquema del enlace serie

5.2. El serializador

El objetivo de este bloque es, tomando como entradas la salida del generador AER que codifica una dirección en formato paralelo, generar una flujo de datos serie más un preámbulo.

Como ya se ha expuesto en capítulos anteriores, la transmisión en nuestro driver es a ráfagas, por lo que habrá períodos de inactividad en la línea. Por tanto, necesitamos un mecanismo que permita al receptor reconocer cuando empieza un nuevo dato. Para ello, se introducen dos 1's al inicio de cualquier dirección que nos sirven como inicio del dato y que, además, alimentan al bloque de recuperación de reloj que habrá en el receptor para comenzar a recuperar el reloj. Introducir dos bits al inicio de la dirección es una solución de compromiso: es una secuencia pequeña para no cargar demasiado el enlace con información de control y es suficiente para reconocer el inicio de un nuevo dato.

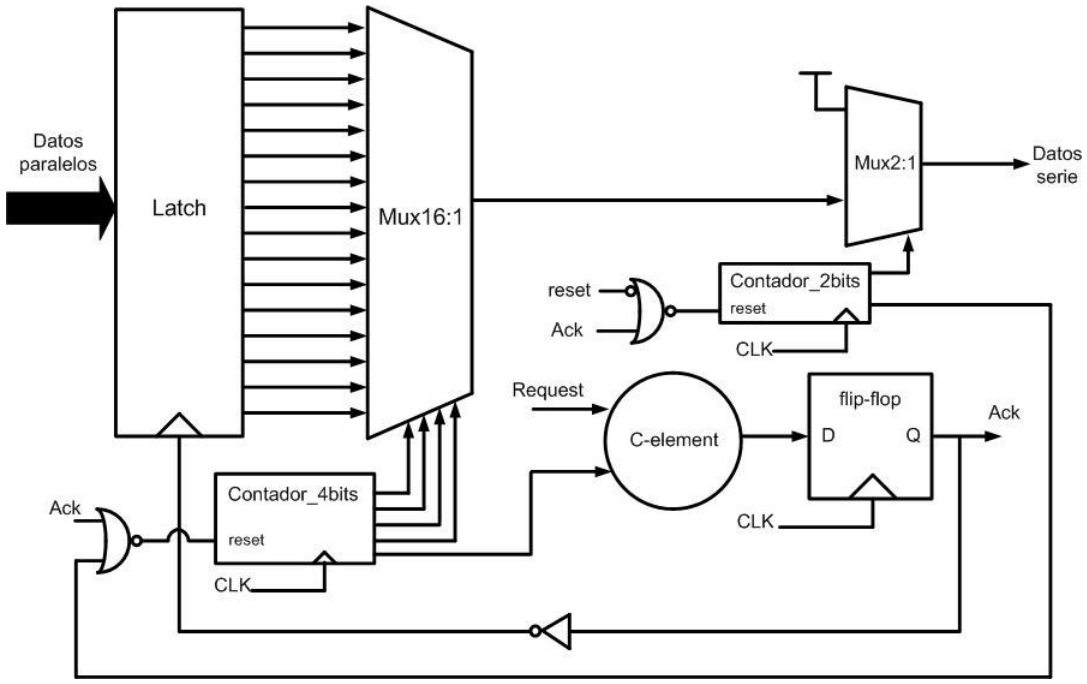


Figura 5.2: Esquema del serializador

Por otro lado, este bloque también se tiene que encargar de la gestión de las señales de petición y asentimiento para poder así interaccionar de manera transparente con los generadores AER que fueron descritos en el apartado 2.6 del capítulo 2. El bloque recibirá como entrada la línea de *Req* y generará como salida la de *Ack*. En el diseño se han tomado ambas líneas activas a nivel bajo, de forma que cuando *Req* baje y el serializador esté libre para transmitir un dato, se hará conmutar la señal de asentimiento hacia un valor bajo. De esta forma, se implementará la comunicación con el bloque paralelo. El esquema del circuito se muestra en la figura 5.2 [1].

El proceso de transmisión es muy sencillo. Comienza cuando el generador AER baja la línea de *Request* indicando que quiere transmitir un nuevo dato. Esto provoca que la señal *Ack* también pase a nivel bajo activando el proceso de transmisión y, por consiguiente, deja de resetear el contador de dos bits que se utiliza para generar el preámbulo. El contador evoluciona libremente seleccionándose así la salida del multiplexor 2 a 1 conectada a alimentación durante dos ciclos de reloj. Es así como se genera el preámbulo de los datos.

Una vez generado el preámbulo, el contador de dos bits genera una señal de fin de cuenta que provoca que el contador de 4 bits deje de estar reseteado (este es el estado mientras no se transmiten datos). Al evolucionar la cuenta de este contador, se van seleccionando las distintas entradas del multiplexor 16 a 1 a cuyas entradas están conectadas las salidas del latch que captura el dato procedente del generador AER. La señal de captura de dicho latch es directamente *Ack* (negada para tener en cuenta que el latch es activo por flanco de subida), puesto que dicha señal sólo se activa cuando es seguro que a la salida hay un dato estable procedente del generador y que, por tanto, debe ser almacenado.

Cuando el proceso de cuenta termina, se genera una señal de fin de cuenta en el contador de 4 bits. Esto provoca que la señal *Ack* pase a nivel alto indicando que ha acabado la transmisión de un dato y que se está dispuesto para enviar uno nuevo. La misión del elemento C es mantener el valor bajo en la señal *Ack* durante todo el proceso de transmisión (condición necesaria para que éste se produzca): el flanco de bajada lo provoca *Request* y el de subida la señal de fin de cuenta del contador de 4 bits.

El esquema de los contadores de 2 y 4 bits se muestra en la figura 5.3. En el primer caso, la señal de fin de cuenta es simplemente el negado del primer bit, generándose a partir de ella la activación del segundo contador. El segundo contador genera su señal de fin de cuenta cuando detecta que todos los bits del contador están a nivel alto (el valor de la cuenta es 16). De esta forma, en el siguiente ciclo de reloj se activará el reset del contador y serán transmitidos exclusivamente los 16 bits que se capturan del latch.

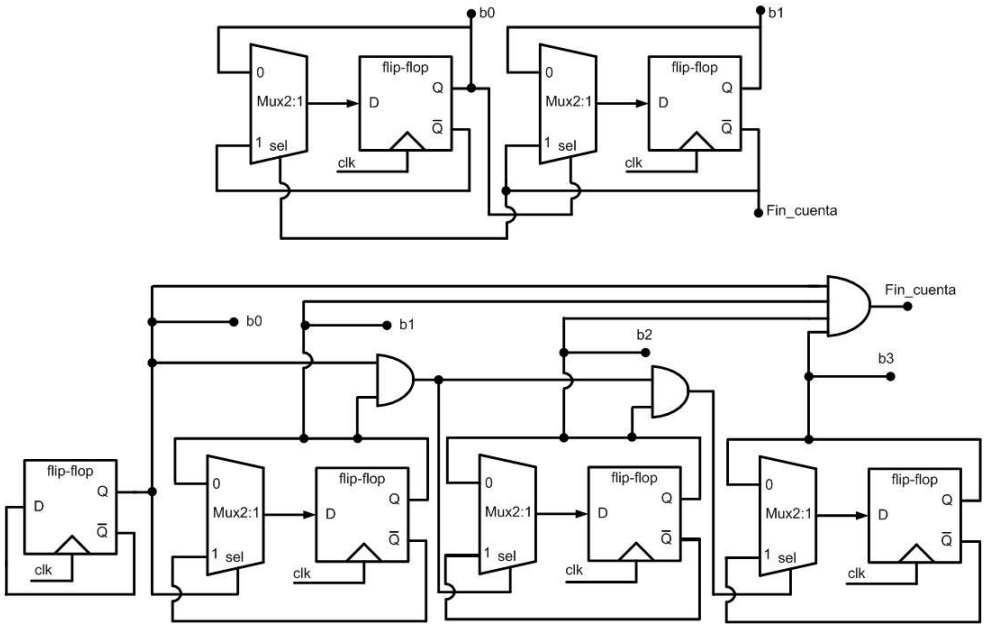
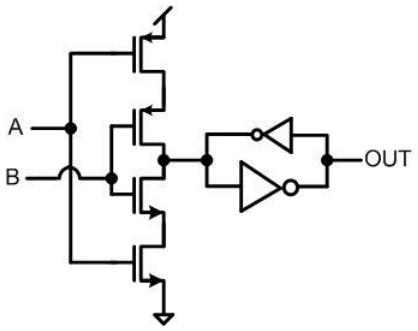


Figura 5.3: Contadores de 2 y 4 bits

El funcionamiento del elemento C fue comentado en apartados anteriores y en la figura 5.4 se muestra su estructura interna a base de transistores. El núcleo de funcionamiento es un celda de memoria formada por dos inversores realimentados. El diseño de estos inversores debe ser asimétrico: uno debe ser más grande que otro para asegurar la metaestabilidad del bucle y conseguir que el bit se almacene indefinidamente. La etapa de entrada simplemente se encarga de gestionar la lectura del elemento de memoria. Cuando las entradas conciden (a nivel alto o a nivel bajo), los transistores conducen y conectan la entrada de la celda de memoria con el raíl de V_{dd} o con el de tierra en función de si la coincidencia es a nivel bajo o alto respectivamente.



A	B	OUT
0	0	0
0	1	Mantiene valor
1	0	Mantiene valor
1	1	1

Figura 5.4: Esquemático y tabla de verdad del elemento C

La figura 5.5 ilustra gráficamente el proceso de generación de la señal de asentimiento a partir de las peticiones externas que se reciben.

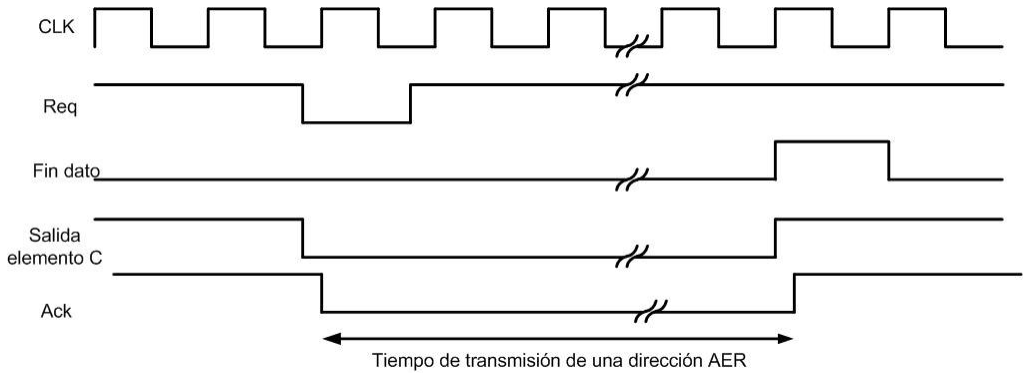


Figura 5.5: Ciclo de generación de Ack

Como ya se citó en la introducción de este capítulo, todos los bloques se implementaron con elementos de biblioteca, consituidos por puertas lógicas, multiplexores, flip-flops,... No obstante, los multiplexores eran 2:1, 4:1 u 8:1, lo que no es suficiente para nuestra aplicación. Para solventar esta limitación se construyó el multiplexor 16:1 a partir de multiplexores 8:1 y 2:1 como muestra la figura 5.6.

El latch de 16 bits se construyó a partir de flip-flops de biblioteca, uniendo la entrada de reloj y reset de todos ellos a una señal global. Las direcciones AER atacan directamente a las entradas D de los biestables y la salida se recoge directamente por la salida Q de los mismos.

5.3. El codificador Manchester

El bloque serializador nos proporciona a su salida un flujo de datos serie que contiene la información de la dirección AER paralela que se ha capturado, así como los dos bits que forman el preámbulo. Sin embargo, la codificación de estos datos es NRZ (*Non Return to Zero*) y no es apta para ser transmitida directamente sobre el canal, pues no permite la recuperación del reloj en el receptor. Por lo tanto, se necesita un bloque adicional que convierta estos datos NRZ a un

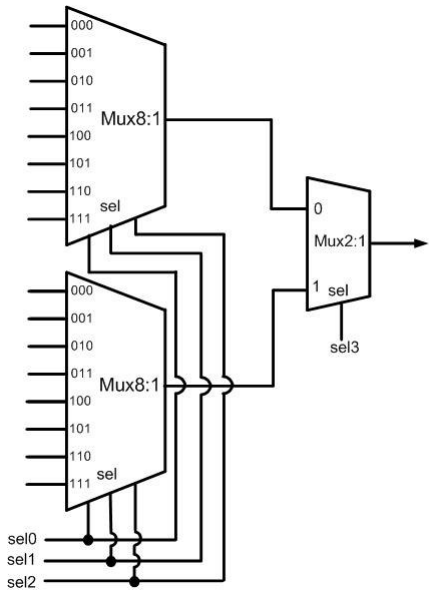


Figura 5.6: Estructura del multiplexor 16:1

formato Manchester que, como ya se ha razonado, sí permite la sincronización.

La figura 5.7 muestra el esquema clásico que permite pasar un tren de datos digitales generados por un cierto reloj a formato Manchester [53].

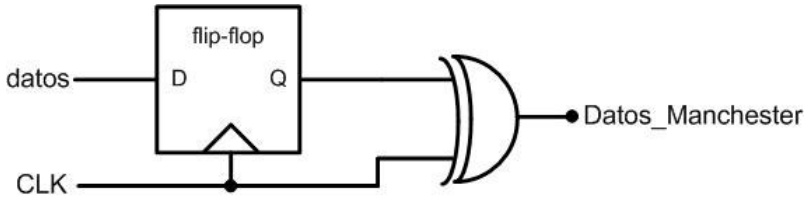


Figura 5.7: Esquema clásico codificador Manchester

Se puede comprobar cómo con una simple función lógica XOR entre el reloj y los datos es capaz de generar los datos en formato Machester. El biestable adicional simplemente sirve para sincronizar el reloj y los datos, alineando estos con el flanco de subida del reloj. Esta arquitectura realiza la codificación de forma eficiente al menos en el caso ideal. Sin embargo, aparecen problemas cuando se consideran los retrasos de la lógica: el tiempo de *setup* del biestable, el retraso del mismo, el retraso de la puerta XOR,... La figura 5.8 muestra un cronograma donde se pone de manifiesto este problema.

Se observa cómo si no se mantienen igualados los retrasos de los dos caminos de señal, se pueden producir *glitches*. Por tanto, es necesario compensar la diferencia de retraso del biestable y la puerta lógica para evitar posibles problemas. Para ello, se ha incluido una cadena de inversores a la entrada de la puerta XOR para poner en fase dicho reloj con los datos temporizados por el biestable. De esta forma, los dos caminos de señal podrán tener retrasos del mismo orden y a la entrada de la puerta lógica el reloj y los datos estarán en fase.

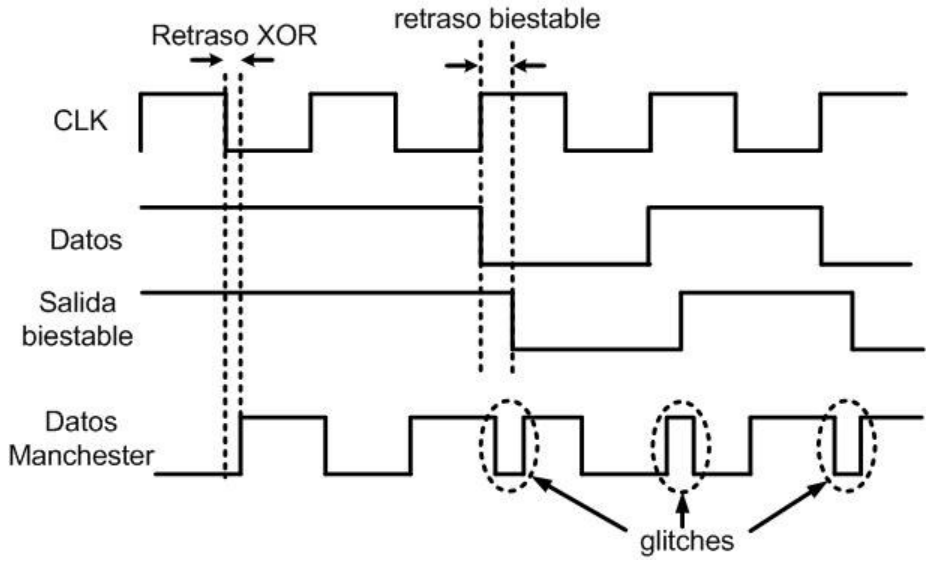


Figura 5.8: Cronograma codificador Manchester real

Otra modificación necesaria viene impuesta por la intermitencia de los datos en el enlace de salida. Es decir, hay que conseguir que, en los periodos de reposo, la línea de datos que ataca al driver LVDS permanezca a un valor constante. El codificador Manchester de por sí seguiría produciendo flancos a pesar de que su entrada no cambiara, por lo que, en cierta forma, es necesario desactivar la señal del codificador. Para ello, se recurre a la señal *Ack* que está a nivel bajo durante el tiempo que dura la serialización de un dato y a nivel alto en los periodos de silencio. La figura 5.9 muestra la lógica que, a partir de los datos serializados y el reloj, genera los datos Manchester, así como el mecanismo de desactivación del codificador.

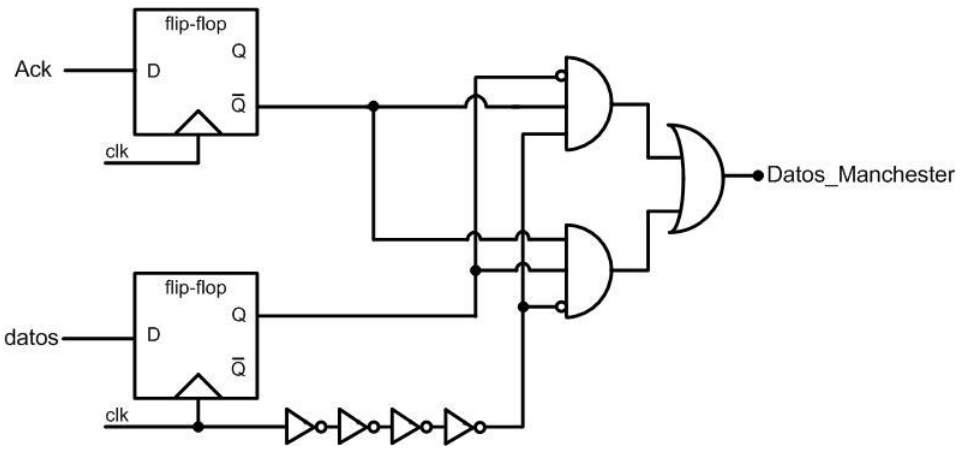


Figura 5.9: Codificador Manchester utilizado

5.4. La unidad de recuperación de reloj

La unidad de recuperación de reloj (CDR por sus siglas en inglés *Clock and Data Recovery*) es el bloque encargado de extraer la señal de reloj de los datos de entrada que entrega el driver. Aprovecha los flancos que la señal Manchester tiene, al menos, cada $T_b/2$ para extraer una señal de sincronismo que permite capturar los datos en el momento adecuado para interpretarlos correctamente. La realización de esta tarea se basa en la arquitectura presentada en la figura 5.10 [1].

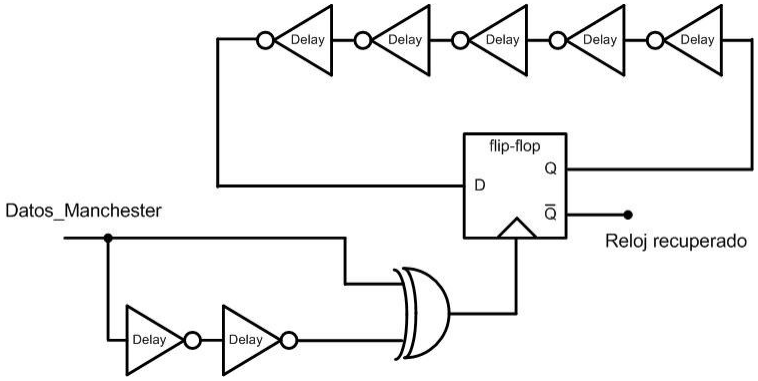


Figura 5.10: Esquema del CDR

El bloque central para la recuperación de reloj es un biestable configurado para comportarse como un oscilador de onda cuadrada. El valor de la señal generada cambiará cada vez que llegue un flanco de subida por la entrada de reloj. Dicha señal de reloj se obtiene mediante un generador de pulsos: la salida de la puerta XOR genera un pulso de corta duración cada vez que hay una transición en la señal de información. Para ello, se aplica a las entradas dos versiones de la señal de entrada desfasadas entre sí $T_b/4$, teniéndose el cronograma mostrado en la figura 5.11 para la señal de captura del biestable D.

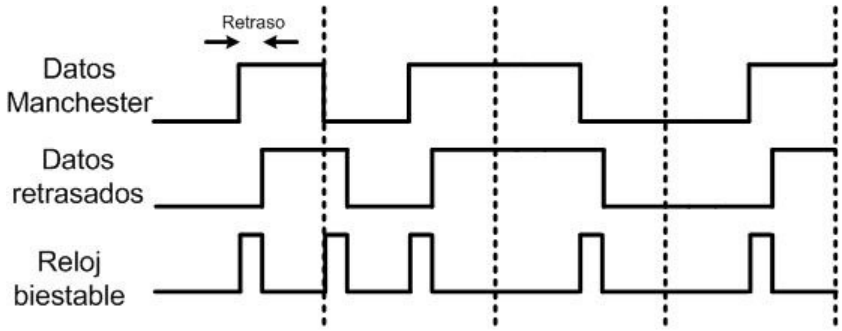


Figura 5.11: Cronograma de generación de pulsos

Para la generación de los retrasos se utiliza un elemento unitario de retraso diseñado para esta función y ajustado a un valor de $T_b/8$, que será igual a 125 ps para una tasa de 1 Gbps. Para que la generación de los flancos de captura sea correcta es necesario que el retraso sea lo

suficientemente grande como para que la puerta XOR sea capaz de responder en el tiempo de pulso que se fije y lo suficientemente pequeño para que los pulsos de unos flancos no afecten a los flancos vecinos. Una elección razonable para la tecnología utilizada es de 250 ps (por eso se utilizan dos elementos de retraso), valor que se comprueba que cumple con las condiciones anteriores.

Una señal codificada en Manchester tiene flancos en dos situaciones distintas: a mitad del intervalo de bit en todos los bits transmitidos y a principio del intervalo cuando se transmiten dos datos consecutivos al mismo valor. Esto quiere decir que el patrón de pulsos que se generará no tendrá un periodo fijo, sino que dependerá de la secuencia de datos. Sin embargo, el objetivo es generar una señal de reloj cuyos flancos coincidan en cuanto a su posición temporal con las transiciones que la señal Manchester tiene a mitad del intervalo de bit. Es por esto que se incluye la segunda línea de retraso entre la salida y la entrada del biestable.

Si este retraso no existiera, cada flanco en el reloj se traduciría en un cambio en la señal de reloj generada. Para evitar que los flancos que tienen lugar al principio del intervalo de bit produzcan un cambio en la salida, se introduce un retraso de $5T_b/8$, es decir, algo más que la mitad del tiempo de bit. El cronograma del proceso descrito se muestra en la figura 5.12. En él se observa cómo las transiciones a mitad del intervalo de bit hacen conmutar al oscilador, mientras que las que se deben a la repetición de un bit no lo hacen. Esto se debe a que la línea de retraso mantiene a la entrada del biestable el valor que capturó en el flanco de $T_b/2$ cuando llega el otro flanco, obligando así a mantener el valor que se tenía en el ciclo anterior a la salida.

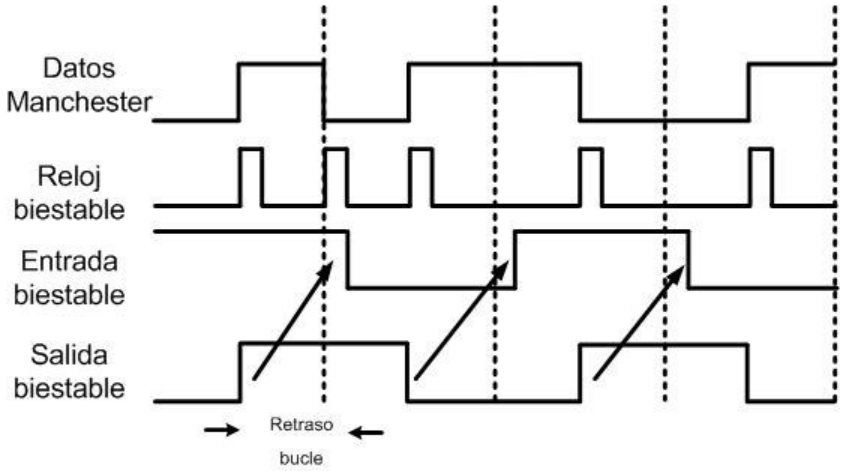


Figura 5.12: Cronograma del retraso en el bucle de realimentación

El valor del retraso introducido en el bucle de realimentación no necesita ser muy preciso: su retraso debe ser mayor que la mitad del tiempo de bit y menor que el tiempo de bit. Así se evitan los pulsos espúreos que se producen cuando se repite algún bit y, por otro lado, tampoco se pierden ninguno de los pulsos que se quieren capturar por un retraso excesivo. No obstante, debe controlarse el valor del retraso para que las variaciones del retardo se mantengan acotadas y, además, que el sistema sea robusto ante el jitter de la señal de reloj o la variabilidad, en cuanto

al valor nominal de retraso que se tenga, por las derivas en los parámetros de los transistores propias del proceso de fabricación.

Una de las grandes ventajas del esquema desarrollado es su inmunidad frente al jitter del reloj del transmisor. Esto se debe a que el reloj recuperado no hace ninguna asunción acerca del reloj que generó los datos y que los flancos de la señal de reloj generada se basan en los de la señal recibida. Esto hace que el jitter que pueda contener la propia señal sea corregido mediante el muestreo de la misma en los instantes de tiempo correctos. De esta forma, se minimiza la probabilidad de error y se hace el sistema más robusto frente a uno de los problemas más acuciantes de las interfaces LVDS.

5.5. La unidad de retraso

En el apartado anterior se pudo comprobar cómo uno de los bloques básicos a la hora de implementar la unidad de recuperación de reloj es la unidad de retraso: el correcto funcionamiento de los bloques digitales depende de que los retrasos mantengan valores razonables entorno al valor nominal de 125 ps. Implementar un retraso es sencillo: mediante una cadena de inversores se puede fijar casi cualquier valor de retraso jugando con el tamaño de los transistores. Sin embargo, esta forma de generar los retrasos tiene dos grandes inconvenientes:

- Fijar el valor del retraso a un valor deseado con exactitud es prácticamente imposible en las tecnologías actuales afectadas por el *mismatching*, incertidumbres en los parámetros del transistor por el propio proceso de fabricación,... Así, si el retraso se programara fijo y por cualquier circunstancia no cumpliera con las condiciones expuestas, no se tendría forma de corregirlo y el circuito sería inservible.

- Fijar un cierto valor de retraso introduce también una gran limitación en cuanto al rango de frecuencias de funcionamiento del circuito. Hay que recordar que los valores recomendados para los retrasos ($T_b/4$ y $5T_b/8$) dependen directamente del tiempo de bit que se elija (y, por tanto, de la tasa). Esto obligaría a especializar el deserializador para una cierta frecuencia, lo cual no sería eficiente a la hora de contemplar futuras ampliaciones del circuito y limitaría sus aplicaciones.

Parece pues que, de alguna forma, es necesario introducir un cierto grado de programabilidad en el valor del retraso unitario que se vaya a usar para que sea capaz de corregir los errores de fabricación que pueda tener el circuito y que permita su adaptación a distintas frecuencias. Además, también será necesario una circuitería adicional que sense las diferencias de retraso entre el valor nominal y el real y actúen sobre los retrasos adpatando su valor. No obstante, en este apartado nos vamos a centrar en cómo introducir la programabilidad en el retardo, dejando el diseño del bucle de realimentación que controla este retardo para apartados posteriores.

La forma má sencilla de implementar un retardo es usar un inversor. La idea es variar alguna de las características eléctricas de dicho inversor en función de una señal de control que permite su

programación. Normalmente, lo que se hace es variar la corriente con la que el inversor conmuta introduciendo fuentes de corriente adicionales para polarizarlo, o bien variar la impedancia de entrada del mismo, controlando así los tiempos de subida y de bajada. Además, exiten dos formas distintas de introducir esa señal de realimentación: mediante una señal analógica aplicada directamente a un transistor o mediante una señal digital (por ejemplo, el valor de un cierto registro).

La figura 5.13 muestra alguna de las arquitecturas típicas que explotan alguna de las ideas que se han comentado anteriormente [51].

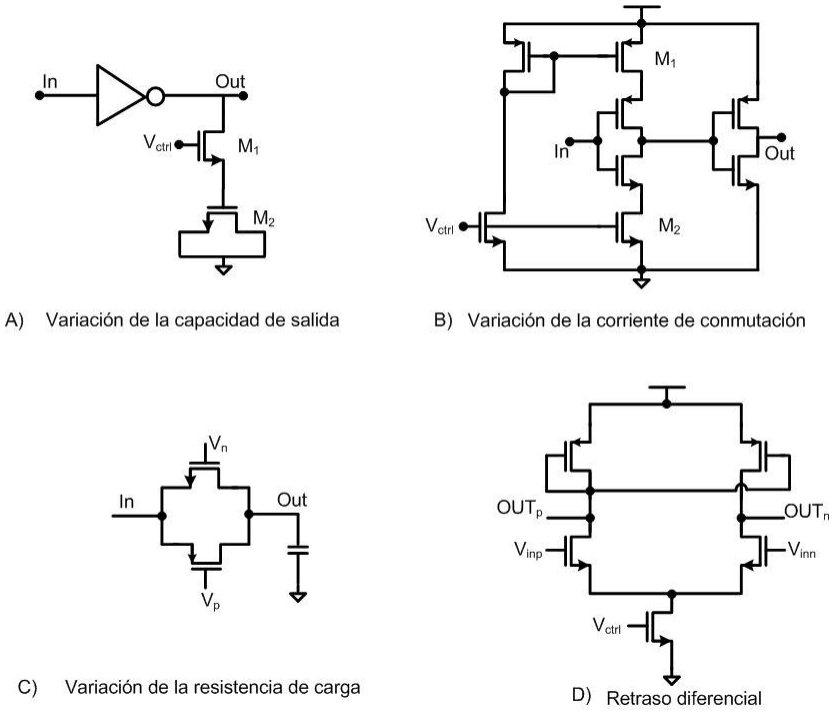


Figura 5.13: Ejemplos de retrasos controlados mediantre una señal analógica

La primera idea es variar la capacidad de salida de un inversor a través de un transistor que actúa como un condensador(M_2). El transistor que recibe la señal de control por su puerta (M_1) controla la corriente de carga y descarga sobre la capacidad de salida del inversor, variando a través de ella los retrasos de la lógica. Otra forma de hacer esto es variar la corriente con la que el inversor conmuta entre los valores lógicos introduciendo una fuente de corriente que aporte la corriente adicional. Para ello se utilizan los transistores M_1 y M_2 del circuito B) cuya corriente se puede controlar mediante una señal de control que configure el valor de la fuente de corriente. Los espejos de corriente se puede diseñar con una relación de aspecto adecuada para equilibrar la corriente de M_1 y M_2 y conseguir así que los tiempos de subida y de bajada estén igualados.

La figura D) es un ejemplo de un retraso diferencial construido a través de un par diferencial con una carga activa. Variando la corriente de polarización del par, es posible controlar los tiempos de subida y bajada de la salida diferencial. Otro parámetro sobre el que se puede tener control es la resistencia de carga del condensador de salida a través de una llave de paso como la

mostrada en la figura C). Controlando la tensión que se aplica sobre la puerta de los transistores se puede controlar la conductancia de los transistores y, por tanto, variar la resistencia a través de la que se carga el condensador, pudiéndose así controlar el retraso.

Otra opción es controlar el retraso digitalmente. Las ideas que hay detrás son las mismas que las comentadas anteriormente, introduciendo las modificaciones necesarias para tener en cuenta que la señal de control es ahora digital. La figura 5.14 muestra algunos ejemplos de arquitecturas típicas para los retrasos [52].

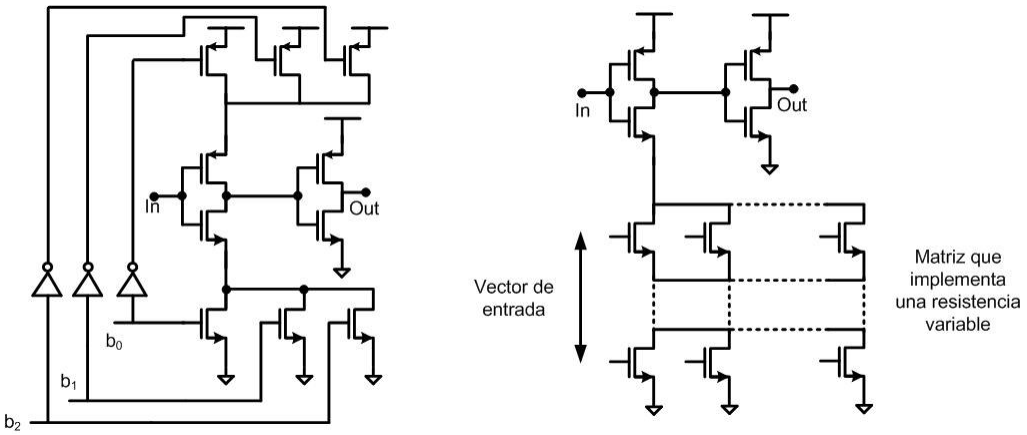


Figura 5.14: Control digital del retraso

En el diseño que aquí se presenta nos centramos en las soluciones que utilizaban una señal analógica para la sintonización. La principal razón fue que esto simplificaba mucho el bucle de realimentación (como se verá en apartados posteriores) y a la vez disminuía el área necesaria para su implementación. Por otro lado, tampoco se necesita una precisión excesiva en la fijación de los retrasos, por lo que no está justificado usar complejos sistemas de calibración digital que aumentarían el área y el consumo del circuito. En concreto, se optó por la arquitectura mostrada en la figura 5.15 [1] para implementar las unidades de retraso: es una de las más compactas, nos permitía un rango de programabilidad suficiente y la variación del retardo, con respecto a la tensión de control, era suficientemente gradual para la precisión que se maneja.

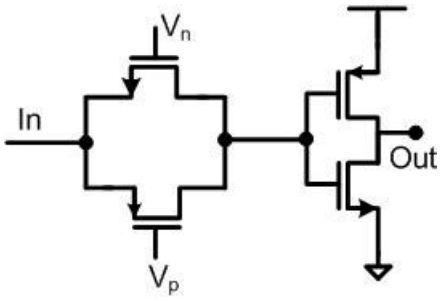


Figura 5.15: Unidad de retraso utilizada

La unidad de retraso es básicamente un inversor y una llave de paso CMOS que varía la resistencia de entrada y nos permite controlar el retraso mediante las señales de control V_n y

V_p. Estas señales de entrada varían la conductividad de la llave y provocan un retraso desde el nudo de entrada hasta la puerta de los transistores del inversor. Por su parte, el inversor se diseña con una relación de aspecto grande con el objeto de que aporte un retraso nominal. Es decir, el inversor aporta un valor de retraso lo suficientemente grande como para conseguir la especificación de que el retraso debe ser de $T_b/4$ y se hace un ajuste más fino variando la resistencia de la llave de entrada.

Así pues, el retraso total se puede dividir en dos partes. La primera de ellas es debida a la llave de paso que se comporta, en cierta forma, como una resistencia equivalente: el paralelo de la resistencia fuente-drenador del transistor p y del n. Así, el modelo equivalente es el de un circuito RC y el retraso (hasta $V_{dd}/2$) se puede calcular como sigue:

$$V_{out} = V_{dd}(1 - e^{-t/R_{eq}C_L}) \quad \Rightarrow \quad t_p = ln(2)R_{eq}C_L$$

La resistencia equivalente dependerá del estado de polarización del transistor que vendrá determinado por el valor de las tensiones de control. Por ejemplo, en el caso en que los transistores estén en zona óhmica, se tiene que:

$$t_p \approx ln(2)\frac{2V_{dd}}{K_n(V_n - V_{tn})^2 + K_n(V_p - V_{tp})^2}C_L$$

La otra componente es la debida al inversor de salida del elemento. Este valor depende de la capacidad de carga y resulta compleja de calcular debido a las no linealidades del transistor. Se puede estimar si suponemos que los transistores conmutan con una corriente promedio igual a la corriente de saturación:

$$I_{av} = \frac{K_p}{2}(V_{gs} - V_{tp})^2 = \frac{K_p}{2}(V_{dd} - V_{tp})^2$$

El retraso se puede estimar como la media de los retrasos de descarga y carga de la capacidad de salida, resultando:

$$t_p = \frac{C_L}{2}\left(\frac{1}{K_p(V_{dd} - V_{tp})^2} + \frac{1}{K_n(V_{dd} - V_{tn})^2}\right)$$

5.6. El circuito de control del retraso

En el apartado anterior hemos visto como la unidad de retraso programable necesita dos señales de entrada para controlar la resistencia de la llave CMOS. El DLL (*Delay Locked Loop*) [49], que es como se conoce a este circuito en la literatura, se encarga de generar estas señales para conseguir ajustar al máximo posible los retrasos a su valor nominal. El diagrama de bloques de este circuito se muestra en la figura 5.16.

En la misma se distinguen tres bloques fundamentales: un circuito que se encarga de generar una señal de referencia (*shifter360*), otro que procesa la diferencia de fases entre el reloj recuperado y el desfasado (*PFD, Phase and Frequency Detector*) y, por último, una bomba de carga

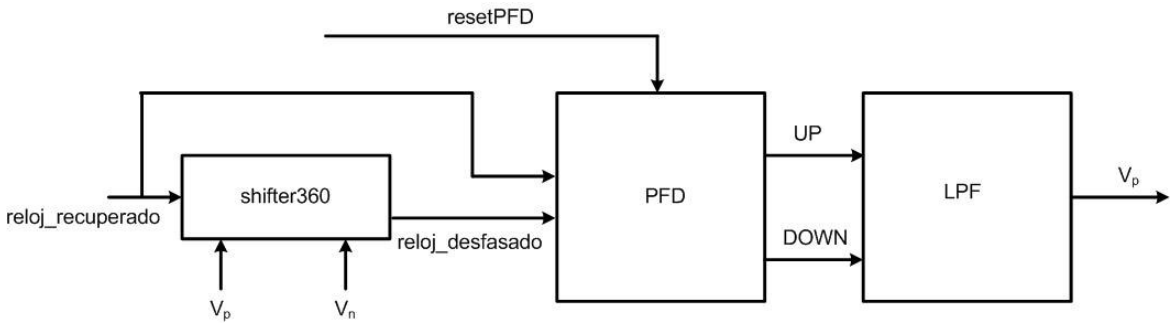


Figura 5.16: Diagrama de bloques del DLL

que a partir de la salida digital del PFD genera la tensión de control (*LPF*). Como entradas, el bloque recibe la señal y una señal de control externa, generando a su salida el valor de las tensiones de control.

El primer objetivo de cualquier DLL es sensar la diferencia entre el retraso nominal y el retraso real que se tiene en el sistema. Para ello, haya que establecer una referencia de fase adecuada que permita hacer la comparación de fase de forma sencilla. En nuestra arquitectura se ha decidido desfasar la señal de reloj generada mediante el CDR 360º, de forma que cuando haya que hacer la comparación de fase se pueda hacer directamente con los mismos flancos. Cada elemento de retraso nominalmente debería tener $T_b/8$, mientras que el periodo del reloj recuperado es de $2T_b$ (se produce un flanco cada T_b). Por tanto, serán necesarios 16 retrasos para generar el desfase requerido. Pues bien, el bloque *shifter360* no es más que eso: una cascada de 16 unidades de retraso como las que se estudiaron en el apartado 5.5. Por supuesto, estos retrasos vienen controlados por las mismas señales V_n y V_p que controlan el resto de retrasos y su implementación debe ser idéntica para que la señal de control generada se corresponda con las necesidades de la unidad de retraso.

Una vez que se tiene la señal para la comparación, hay que generar dos señales, *UP* y *DOWN*, que de alguna forma codifiquen la diferencia de retardos entre el reloj recuperado y el desfasado. Esta codificación debe ser compatible con la generación analógica de la señal de control. Además, debe tenerse en cuenta que la comparación es entre dos señales desfasadas un ciclo completo, por lo que durante el primer ciclo de la señal recuperada no debe realizarse la comparación, puesto que la señal desfasada aún no se ha generado convenientemente. El cronograma de la figura 5.17 ilustra este problema y muestra los flancos de reloj donde se produce la comparación de fase.

Una forma simple de generar unas señales que codifiquen de forma temporal esta información se muestra en la figura 5.18. Los dos biestables con las líneas de reset unidas forman el núcleo del comparador de fase, mientras que el primer biestable se encarga de evitar la comparación en el primer ciclo del reloj recuperado. También hay que hacer notar que el biestable que genera *DOWN* captura su entrada según los flancos de subida del reloj recuperado y el biestable que genera *UP* viene activado por el desfasado. La señal *resetPFD* se encarga de activar el proceso de comparación: como la recepción de datos es intermitente es necesario que se habilite la evolución

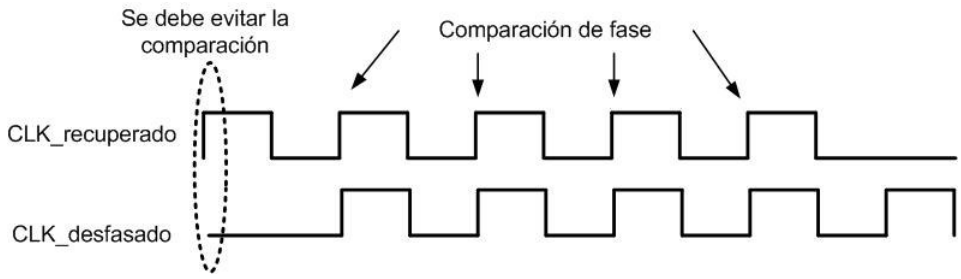


Figura 5.17: Cronograma para la comparación de fase

de los biestables. Por último, tenemos una lógica para la generación del reset de los biestables. Hay que recordar que la activación del reset es por nivel bajo (de ahí que sea necesario el inversor para *resetPFD*, que es activa a nivel alto como ya se verá).

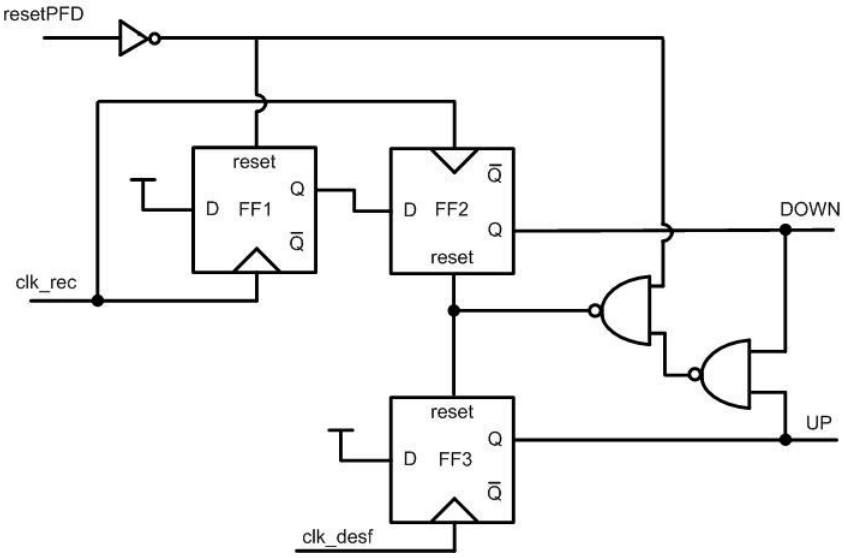


Figura 5.18: Esquema del PFD

Mientras el reset global del bloque esté activo, los biestables permanecen en reposo manteniendo las señales *UP* y *DOWN* a nivel bajo. Una vez que se detecta el inicio de un dato, el biestable FF1 captura un 1 a su entrada, mientras que FF3 no captura nada pues es activado por la señal retrasada un ciclo de reloj y ésta permanece en reposo al inicio de la recepción de una dato. Al siguiente flanco del reloj recuperado, el 1 inicial que capturó FF1 se transfiere al segundo biestable y FF3, que ahora si es activado, también computa otro 1. La lógica que controla el reset de FF2 y FF3, está a cero mientras *UP* y *DOWN* permanecen a nivel bajo. Sólo cuando ambas coinciden a nivel alto, genera un nivel bajo a su salida proporcionando el reset de los biestables.

La detección de la fase se produce por la diferencia de tiempos que *UP* y *DOWN* permanecen a nivel alto. Cuando existe una diferencia de retraso entre el reloj recuperado y el desfasado, los flancos de activación de los biestables también lo estarán y, alguna de las señales de control

pasará a nivel alto, mientras la otra mantiene su valor de reposo. Sólo cuando coincidan las dos a nivel alto, el sistema será reseteado quedando a la espera de nuevos flancos. El sistema diferencia retrasos y adelantos de la señal desfasada en función de qué señal se active antes: bien *UP* (indicando que hay que elevar la tensión de control) o bien *DOWN* (indicando que hay que disminuirla). El cronograma de la figura 5.19 clarifica el funcionamiento del bloque.

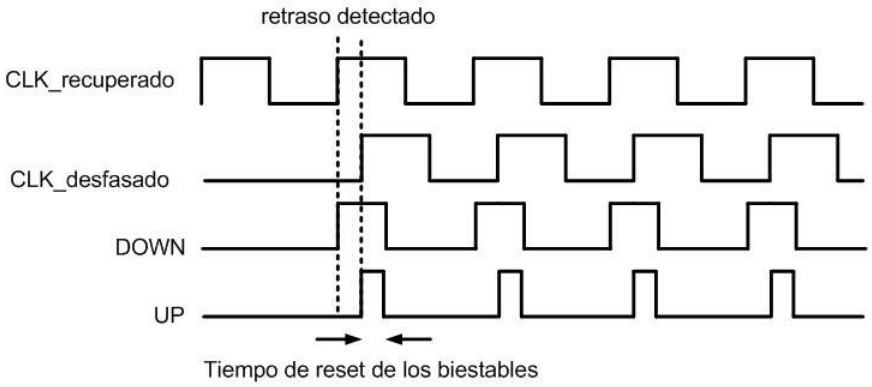


Figura 5.19: Cronograma del funcionamiento del PFD

En la figura propuesta se observa cómo existe un cierto tiempo en que las señales *UP* y *DOWN* permanecen conjuntamente a nivel alto. Esto es debido a los retrasos inherentes a la lógica: desde que la salida de los biestables se estabiliza a valor alto, hasta que las puertas lógicas procesan esta información dando un valor bajo a la salida y el biestable se resetea, pasa un cierto tiempo en el que la salida permanece constante a valor alto. Esto parece contradictorio pues se le está diciendo al siguiente bloque que debe a la vez elevar y disminuir la tensión que controla los retrasos, cuando en realidad no hay ninguna información sobre la comparación de fase en este hecho. Por ello, el *LPF* deberá ignorar esta situación manteniendo esta tensión constante.

La figura 5.20 presenta el esquema del *LPF*, encargado de traducir la información codificada en las señales *UP* y *DOWN* en una variación analógica de la tensión de control. El circuito es muy simple: dos fuentes de corriente del mismo valor cargan un condensador de salida que integra esta corriente cuando alguna de las llaves se cierran. Las llaves analógicas vienen controladas por las señales *UP* y *DOWN*, de forma que cuando existe un retraso indican a la bomba de carga qué camino de señal debe elegir: si desea elevar la tensión activará la llave del lado p, mientras que si desea disminuir la activará la llave del lado n.

El diseño de la constante RC que acompaña al condensador de carga se ha hecho lo suficientemente grande como para que en los tiempos de integración típicos (algunos cientos de picosegundos) no influya de manera significativa en la dinámica y que sea el condensador C_2 el que se encargue de variar la tensión de control. Dos son los parámetros que controlan el proceso de ajuste de la tensión de control: el valor de la corriente de carga y descarga y el valor del condensador donde se integra. La restricción sobre estos valores lo impone el grado de programabilidad de los retrasos. Si la integración es demasiado rápida, la tensión de control podría saturar, impidiendo que el bucle de control del retraso pudiera converger a un valor adecuado.

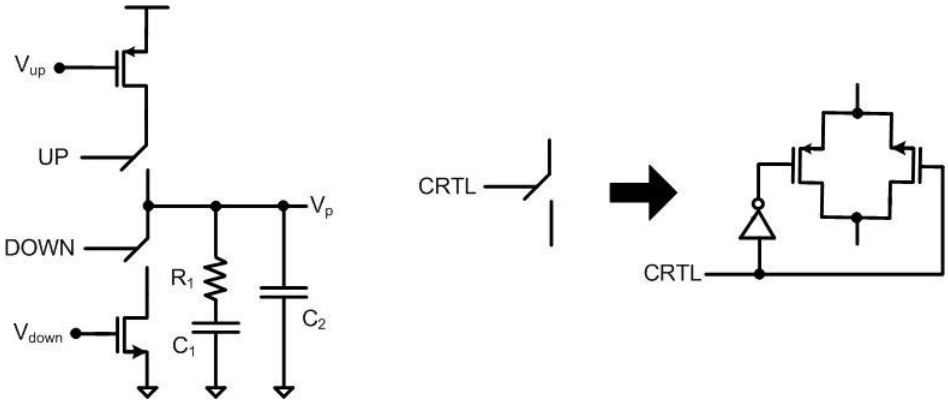


Figura 5.20: Esquema y componentes del LPF

En cambio, si es demasiado lenta, los tiempos de convergencia del bucle serían largos en tanto en cuante necesitaría una gran cantidad de datos recibidos para poder ajustar el valor de los retrasos.

La tensión en el nudo de salida se puede estimar como:

$$V_p = \frac{I_b}{C_2} t + V_p(0)$$

donde I_b es la corriente a la que se hayan programado para las fuentes de corriente p y n. Cuando los relojes recuperado y desfasado están desfasados una cantidad distinta de 360º, *UP* o *DOWN* se activa durante un tiempo igual al retraso entre los dos relojes. Si llamamos Δt a este tiempo, el incremento que esto provoca en la tensión de control se puede calcular como:

$$\Delta V_p = \frac{I_b}{C_2} \Delta t$$

El valor nominal de diseño para la corriente I_b es de 50 μA y el de la capacidad C_2 de 0.2 pF. Si tomamos como referencia una diferencia de retraso de 250 ps (cantidad que podemos considerar un valor típico), resulta en un incremento de 62.5 mV. Este valor está claro que está dentro del rango de programabilidad de la unidad de retraso, el cual viene limitado por la conducción de los transistores de una llave de paso CMOS. Otra consideración adicional es que habrá momentos tras cada comparación en que las señales *UP* y *DOWN* coincidan a nivel alto (lo que se llamó tiempo de reset en la figura 5.19) y la tensión de control deba permanecer constante. Esto se consigue igualando bien las corrientes de las fuente p y n que generan la carga y descarga del codensador. De ese modo, cuando ambas llaves se cierren, no podrá derivarse corriente hacia la capacidad, puesto que la corriente en ambas ramas de la bomba de carga coincidirá.

Como se ha visto, el *LPF* genera sólo la tensión para los transistores p de los elementos de retraso: la tensión para los n se genera a través de un amplificador que adapta el valor de la señal para conseguir igualar los retrasos. El esquemático de este circuito se muestra en la figura 5.21.

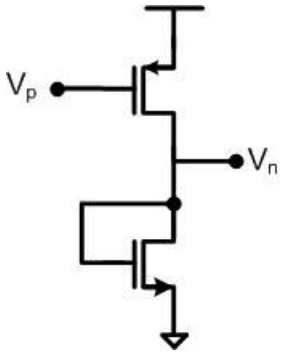


Figura 5.21: Esquema del amplificador-adaptador

El objetivo principal de este amplificador será el de desplazar el nivel de continua de la entrada, por lo que su ganancia deberá ser cercana a 1 y tener una característica DC desplazada. Si suponemos que los dos transistores operan en saturación y despreciamos los efectos de modulación de canal, las intensidades de los tansistores se pueden expresar:

$$I_p = \frac{1}{2} K_p \left(\frac{W_p}{L_p} \right) (V_{dd} - V_p - V_{tp})^2$$

$$I_n = \frac{1}{2} K_n \left(\frac{W_n}{L_n} \right) (V_n - V_{tn})^2$$

Igualando las dos corrientes se obtiene la siguiente característica de entrada salida para el amplificador:

$$V_n = V_{tn} + A_o (V_{dd} - V_p - V_{tp}) \quad donde \quad A_o = \sqrt{\frac{K_p (W/L)_p}{K_n (W/L)_n}}$$

Este comportamiento será así mientras los transistores estén en saturación. El transistor n, por su configuración, si conduce, siempre estará en saturación, por lo que la única condición que impone es que la tensión de salida supere la umbral de conducción del canal. El transistor p impone, en primer lugar, una restricción sobre la tensión de entrada: para que el transistor p conduzca, V_p debe estar por debajo de la cantidad $V_{dd} - V_{tp}$. Por otro lado, para asegurar la saturación de ese mismo transistor se debe cumplir que la tensión de entrada supere la de entrada desplazada en la tensión umbral del transistor p. Es decir, se debe cumplir la condición: $V_p > V_n - V_{tp}$.

5.7. El deserializador

La recepción de los datos Manchester se hace extremadamente sencilla con la unidad de recuperación de reloj presentada. Dicho bloque proporciona una señal de reloj completamente sincronizada con los datos: los flancos de subida y bajada de este reloj se corresponden con las transiciones a mitad de bit de los datos. Además, con la codificación Manchester usada, para

decodificar los datos basta con muestrear en el intervalo $[T_b/2, T_b]$ la línea serie para obtener el valor transmitido como muestra la figura 5.22.

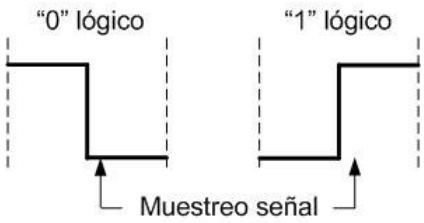


Figura 5.22: Muestreo de la señal Manchester

El deserializador, por tanto, consistirá en un registro de desplazamiento que almacena los bits serie a medida que se van recibiendo por la línea. Como hay que muestrear la línea de datos tanto en los flancos de subida como en los de bajada del reloj recuperado, los bits pares serán sensados por biestables activos por flanco de subida y los bits impares por biestables activos por flanco de bajada. Los valores leídos se irán desplazando por el registro de desplazamiento hasta llegar al último biestable. En ese caso se generará una señal de captura para un latch que capturará los datos y los presentará a la salida en formato paralelo. El esquema del deserializador se muestra en la figura 5.23 (el latch no se ha representado por claridad en la figura).

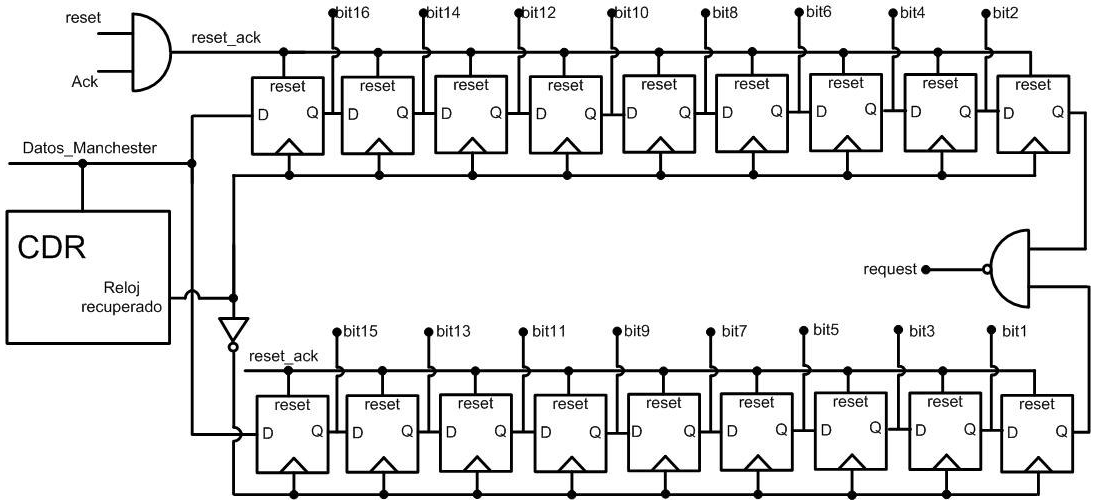


Figura 5.23: Esquema del deserializador

Además de proporcionar los datos en formato paralelo, el deserializador también debe cumplir con el protocolo AER, para lo que tiene que generar la señal de *Request* y sincronizarla con los cambios en *Ack*. Para ello, se usará el preámbulo de dos 1's que se antepone a los datos por razones de alineamiento del enlace. Cuando se complete la recepción de un dato, estos dos 1's llegarán a los últimos biestables de los registros de desplazamiento que reciben los bits pares e impares. En ese momento, la puerta NAND provocará un nivel bajo en la señal de *Request* indicando que se ha completado la recepción de un dato. A su vez esto activará la captura del latch que presentará los datos en formato paralelo. El receptor AER que utilice este deserializador

bajará la señal *Ack* cuando haya procesado los datos que se le presentan. En ese momento, una lógica reseteará los biestables de los dos registros de desplazamiento, llevando sus salidas a nivel bajo. Esto hará que la señal de *Request* vuelva automáticamente a nivel alto, pues desaparecen los 1's del preámbulo que se tenían almacenados. De esta forma, cumplimos con el protocolo AER.

El deserializador sólo tiene una limitación: si mientras el proceso de petición y asentimiento del protocolo AER llega un nuevo dato, éste puede sobrescribir la información que se tenía del dato anterior, provocando la recepción de un dato no válido. Este problema se podría subsanar construyendo un buffer de recepción que almacenara temporalmente los datos hasta que llegara el asentimiento a los mismos. Sin embargo, la alta velocidad de transmisión que se está utilizando, hace que esta circunstancia sea difícil que se produzca, pues requeriría un modo de transmisión cuasi-continuo y, como ya se ha razonado, se prevee que la transmisión sea a impulsos. No obstante, habrá que considerar este problema para ampliaciones futuras.

5.8. Problemas de la transmisión a ráfagas

Una de las particularidades de nuestro deserializador es que no recibe un flujo continuo de datos, sino que la recepción se produce a ráfagas con largos períodos de silencio entre un dato y el siguiente. Sin embargo, el bucle de realimentación que controla los retardos necesita continuamente flancos en la señal de entrada para poder extraer el reloj. De lo contrario, cada vez que se quisiera recibir un dato habría que esperar a que el bucle se estabilizara para asegurar que los datos son correctos. Esto obligaría a la transmisión de cabeceras largas (con la consiguiente pérdida de tasa) ó bien a la transmisión continua insertando relleno (con el incremento de consumo de potencia que esto conlleva). Por tanto, es necesario de alguna forma mantener el estado del bucle de control del retardo en los períodos de silencio de la línea.

La idea es introducir un elemento de memoria que se encargue de almacenar el valor de la tensión de control que se tiene al finalizar la recepción de un dato e imponerla durante los periodos de silencio de la línea. El formato más eficiente para almacenar esta información es en formato digital, pero la señal de control, como ya se ha razonado, es analógica. Por tanto, se necesitarán convertidores de datos (analógico-digital y digital-analógico) para congelar el estado del bucle, así como señales de control que se encarguen de gestionar el estado del nudo que fija la tensión de control. Así, ésta podrá ser fijada bien por el *LPF* o bien por un DAC (*Digital to Analog Converter*) que memorice la tensión de control. El esquema de la solución aportada se muestra en la figura 5.24.

El ADC (*Analog to Digital Converter*) sensa la salida del *LPF* convirtiendo la señal a formato digital, proceso que se realiza de forma continua. Este valor se almacena en el latch con los flancos de subida de la señal *latch*. Dichos eventos se producen cada vez que hay una transición en la señal de entrada. Mientras haya flancos en esta señal quiere decir que se está recibiendo un dato y, por tanto, es relevante almacenar el valor de la tensión de control. De la misma forma, cada vez que se latchea un dato, también se cierra una llave que lleva la señal *resetPD* a un valor

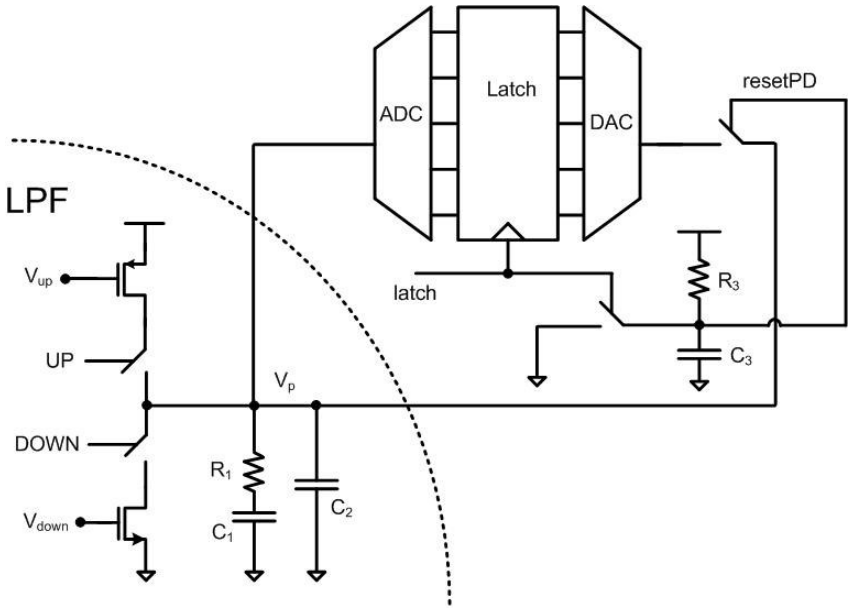


Figura 5.24: Esquema para memorizar estado del bucle de control del retardo

bajo, impidiendo que el CDA pueda imponer un valor en el nudo de la señal de control, V_p. Este estado se mantiene mientras que se esté recibiendo un dato.

Una vez que deja de recibirse un dato, no llegan más flancos por *latch* y en el registro se almacena el último valor analógico que se leyó. En ese momento, se deja evolucionar libremente el nudo que controla la tensión *resetPD*, a través de un circuito RC, hasta que la tensión alcanza la tensión de alimentación. En esa circunstancia, se cierra la llave que controla la salida del ADC y la tensión en el nudo V_p vendrá fijada por la conversión del valor digital almacenado en el latch, es decir, el último valor de la tensión de control que se leyó durante la recepción de un dato válido.

Esta arquitectura obliga a una solución de compromiso para los valores de R_3 y C_3, pues la constante de tiempo del circuito RC no puede ser cualquiera. Por un lado debe ser lo suficientemente grande como para que entre dos flancos consecutivos de la señal de entrada la señal *resetPD* apenas varíe. En el peor de los casos y para una tasa de 1 Gpbs se tiene un tiempo entre flancos de 1ns, por lo que $R_3C_3 >> 1$ ns. De no ser así, *resetPD* podría llegar a un valor alto a mitad del proceso de recepción de un dato indicando el final del mismo, lo cual supone un funcionamiente incorrecto de la interfaz. Por otro lado, la constante de tiempo no debe ser demasiado grande pues limitaría la velocidad de recepción de los datos: tras finalizar la recepción de un dato, hasta que no llega *resetPD* no se fija el valor correcto de la tensión de control para el retardo necesario. Esto impone una cota superior algo difusa, porque dependerá de la aplicación y de la tasa que queramos soportar. No obstante, $R_3C_3 \approx 10$ ns parece un valor razonable que no provoca activaciones espúreas de *resetPD* y que tampoco limita excesivamente el enlace.

Otro parámetro importante a determinar es el número de bits de los convertidores que digitalizan la señal de control. La variación de la señal de control es mucho más lenta que la

velocidad de transmisión de datos, por lo que no es necesario tener convertidores excesivamente potentes y rápidos para realizar esta tarea (podría bastar con algún convertidor de rampa o de aproximaciones sucesivas sencillo). Además, los errores de conversión tampoco suponen un excesivo problema, pues al operar dentro de un bucle de realimentación, dichos errores se ven atenuados por la operación del resto de componentes. Sin embargo, tampoco es conveniente usar un número grande de bits porque esto limitaría enormemente los valores de retrasos que se podrían conseguir y podría provocar que el bucle no llegara nunca a converger por los efectos de cuantización del retraso. Las simulaciones mostraron que con 5 bits hay más que suficiente para llevar a cabo esta labor.

Debido a la gran cantidad de parámetros a determinar y a que algunos influyen en la tasa máxima que soporta el enlace, se decidió que esta parte del circuito no se iba a integrar, sino que se iba a implementar externamente mediante elementos comerciales. Por tanto, la descripción de estos bloques no fue a nivel de transistor, sino a nivel funcional. Para ello se utilizó el lenguaje AHDL para la descripción funcional de bloques analógicos o mixtos que proporcionaba el entorno de diseño. Se modelaron en este lenguaje tanto los convertidores, como las llaves analógicas que controlan la activación de las señales.